Wavelet Toolbox™

Reference

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi

MATLAB®

R2022**a**

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.
1 Apple Hill Drive
Natick, MA 01760-2098

Wavelet Toolbox™ Reference

Trademarks

Patents

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)
March 2018	Online only	Revised for Version 5.0 (Release 2018a)
September 2018	Online only	Revised for Version 5.1 (Release 2018b)
March 2019	Online only	Revised for Version 5.2 (Release 2019a)
September 2019	Online only	Revised for Version 5.3 (Release 2019b)
March 2020	Online only	Revised for Version 5.4 (Release 2020a)
September 2020	Online only	Revised for Version 5.5 (Release 2020b)
March 2021	Online only	Revised for Version 5.6 (Release 2021a)
September 2021	Online only	Revised for Version 6.0 (Release 2021b)
March 2022	Online only	Revised for Version 6.1 (Release 2022a)

Contents

1

Functions

addLabelDefinitions

Add label definitions to labeled signal set

Syntax

```
addLabelDefinitions(lss,lbldefs)
addLabelDefinitions(lss,lbldefs,lblname)
```

Description

`addLabelDefinitions(lss,lbldefs)` adds the labels defined in the vector of signal label definitions `lbldefs` to the labeled signal set `lss`.

`addLabelDefinitions(lss,lbldefs,lblname)` adds the labels defined in `lbldefs` as sublabels of the label `lblname`.

Examples

Add Label Definition

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Create a label definition that specifies whether a signal corresponds to a calf or to an adult whale.

```
calf = signalLabelDefinition('Calf','LabeldataType','logical','DefaultValue',false, ...
    'Description','Is the specimen a calf, or an adult?')
```

```
calf = 
  signalLabelDefinition with properties:

                  Name: "Calf"
             LabelType: "attribute"
         LabelDataType: "logical"
    ValidationFunction: []
```

```
         DefaultValue: 0
            Sublabels: [0x0 signalLabelDefinition]
                  Tag: ""
          Description: "Is the specimen a calf, or an adult?"

 Use labeledSignalSet to create a labeled signal set.
```

Add the definition to the labeled signal set. Retrieve the names of the labels.

```
addLabelDefinitions(lss,calf)

getLabelNames(lss)
```

```
ans = 4x1 string
    "WhaleType"
    "MoanRegions"
    "TrillRegions"
    "Calf"
```

Create a label definition that specifies the sex of the whale. Add the label to the set as a sublabel of `'WhaleType'`.

```
sx = signalLabelDefinition('Sex','LabelDataType','categorical', ...
    'Categories',["male" "female"]);
addLabelDefinitions(lss,sx,'WhaleType')

labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: Sex
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: TrillPeaks
     Calf
       Sublabels: []
     '
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lbldefs` — Signal label definitions

`signalLabelDefinition` object | vector of `signalLabelDefinition` objects

Signal label definitions, specified as a `signalLabelDefinition` object or a vector of `signalLabelDefinition` objects.

Example:
`signalLabelDefinition("Asleep",'LabelType','roi','LabelDataType','logical')`
can label a region of a signal in which a patient is asleep.

`lblname` — Label name

character vector | string scalar

Label name, specified as a character vector or a string scalar.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

addlift

(To be removed) Add lifting steps to lifting scheme

Note This version of `addlift` will be removed in a future release. Use the new versions of `addlift`, `liftingStep`, and `liftingScheme`. For more information, see “Compatibility Considerations”.

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]                          []
};
```

Compatibility Considerations

addlift will be removed

Not recommended starting in R2021a

This version of addlift, that adds steps to a lifting scheme created using liftwave, will be removed in a future release.

For lifting, use the new version of addlift, liftingStep, and liftingScheme. To update your code, follow these steps:

1. Create a lifting scheme using liftingScheme.
2. Create a lifting step or an array of lifting steps using liftingStep.
3. Add the lifting step or lifting steps using addlift.

See Also

liftfilt | addlift | liftingScheme | liftingStep

Introduced before R2006a

addlift

Add elementary lifting steps

Syntax

```
lsn = addlift(lscheme,els)
lsn = addlift(lscheme,els,loc)
```

Description

`lsn = addlift(lscheme,els)` appends the array of elementary lifting steps `els` to the lifting scheme object `lscheme`.

`lsn = addlift(lscheme,els,loc)` inserts the array of elementary lifting steps `els` in the lifting scheme `lscheme` at the specified location `loc`.

Examples

Insert Elementary Lifting Steps

Create a lifting scheme associated with the `db2` wavelet.

```
lscheme = liftingScheme('Wavelet','db2')
```

```
lscheme = 
      Wavelet               : 'db2'
     LiftingSteps          : [3 × 1] liftingStep
     NormalizationFactors  : [1.9319 0.5176]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'predict'
    Coefficients: -1.7321
        MaxOrder: 0

            Type: 'update'
    Coefficients: [-0.0670 0.4330]
        MaxOrder: 1

            Type: 'predict'
    Coefficients: 1
        MaxOrder: -1
```

Create an array that consists of two elementary lifting steps.

```
elsA = liftingStep('Type','predict',...
    'Coefficients',[-sqrt(3) 1],'MaxOrder',0);

elsB = liftingStep('Type','update',...
    'Coefficients',[2 sqrt(2)],'MaxOrder',0);

els = [elsA;elsB];
```

Insert the array at the second position.

```
loc = 2;
lsn = addlift(lscheme,els,loc)
```

```
lsn = 
      Wavelet                : 'custom'
     LiftingSteps          : [5 × 1] liftingStep
     NormalizationFactors  : [1.9319 0.5176]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'predict'
    Coefficients: -1.7321
        MaxOrder: 0

            Type: 'predict'
    Coefficients: [-1.7321 1]
        MaxOrder: 0

            Type: 'update'
    Coefficients: [2 1.4142]
        MaxOrder: 0

            Type: 'update'
    Coefficients: [-0.0670 0.4330]
        MaxOrder: 1

            Type: 'predict'
    Coefficients: 1
        MaxOrder: -1
```

Input Arguments

lscheme — Lifting scheme

liftingScheme object

Lifting scheme, specified as a liftingScheme object.

els — Lifting steps

structure array

Lifting steps, specified as a structure.

loc — Location

length(lscheme.LiftingSteps) (default) | positive integer

Location to add the lifting steps in lscheme, specified as a positive integer between 1 and length(lscheme.LiftingSteps) inclusive.

- If `loc` is 1, the lifting steps are inserted at the beginning of the lifting scheme.
- If `loc` is `length(lscheme.LiftingSteps)`, the lifting steps are added at the end of the lifting scheme.
- If `loc` is greater than 1 and less than `length(lscheme.LiftingSteps)`, the lifting steps are inserted after the (`loc`-1)th step of `lsc`.

Data Types: `double`

Output Arguments

`lsn` — Lifting scheme
`liftingScheme` object

Lifting scheme, returned as a `liftingScheme` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`liftingStep` | `liftingScheme` | `deletelift`

Introduced in R2021a

addMembers

Add members to labeled signal set

Syntax

```
addMembers(lss,src)
addMembers(lss,src,tinfo)
addMembers(lss,src,tinfo,mnames)
```

Description

`addMembers(lss,src)` adds members to the labeled signal set `lss` from the input data source `src`.

`addMembers(lss,src,tinfo)` sets the time information for the new members to `tinfo`.

`addMembers(lss,src,tinfo,mnames)` sets the names of the new members to `mnames`. The length of `mnames` must be equal to the number of new members.

Examples

Add Member to Labeled Signal Set

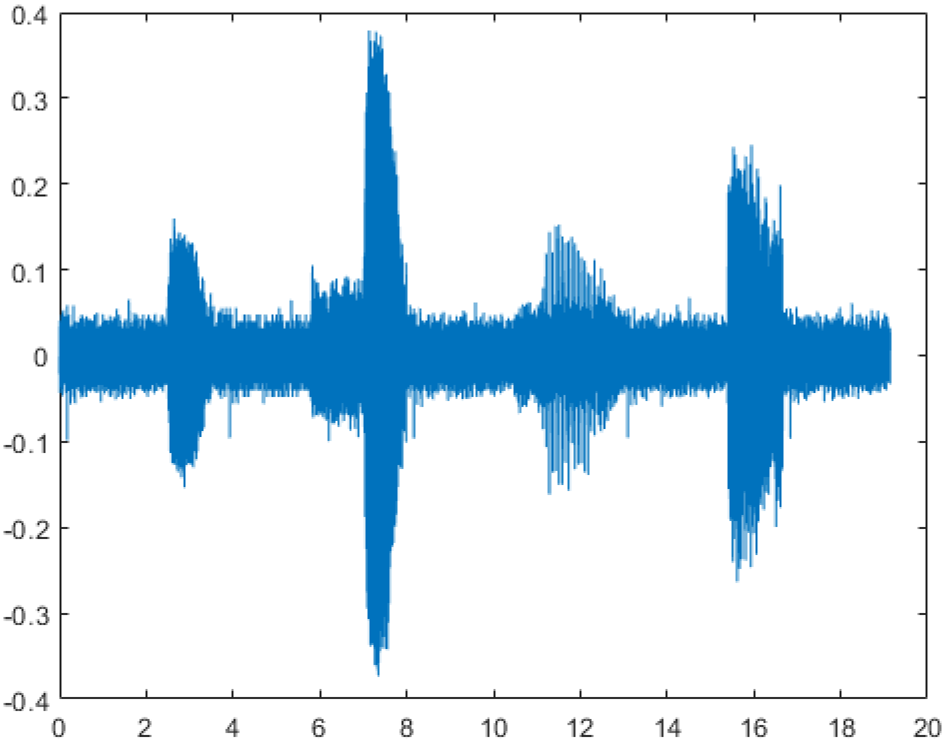
Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

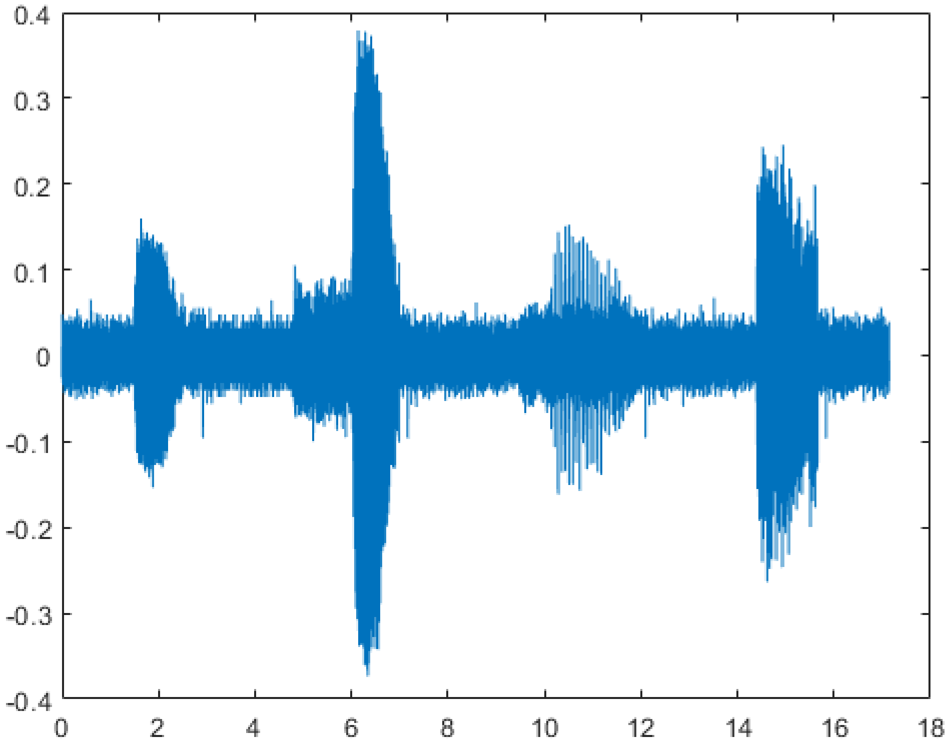
Retrieve the second member of the set and plot it.

```
[song,tinfo] = getSignal(lss,2);

t = (0:length(song)-1)/tinfo.SampleRate;

plot(t,song)
```



Remove the first and last seconds of the retrieved signal.

```
song2 = song(t>1 & t<t(end)-1);
t2 = (0:length(song2)-1)/tinfo.SampleRate;

plot(t2,song2)
```



Add the shorter signal as a new member of the labeled set.

```
addMembers(lss,song2)
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {3x1 cell}
         NumMembers: 3
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [3x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Flip the shorter signal upside-down and add it as a new member of the labeled set. Specify that the new member is sampled at 1 kHz.

```
addMembers(lss,flipud(song2),1000)
lss.SampleRate
```

```
ans = 4×1

        4000
```

```
    4000
    4000
    1000
```

Input Arguments

`lss` — Labeled signal set
`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`src` — Input data source
matrix | cell array | timetable | `signalDatastore` object | `audioDatastore` object

Input data source, specified as a matrix, a cell array, a timetable, a `signalDatastore` object, or an `audioDatastore` object. The particular form of `src` depends on the "Source" on page 1-0 property of `lss`.

- If "Source" on page 1-0 is a cell array of matrices:
  - Specify `src` as a matrix to add one member to the set.
  - Specify `src` as a cell array of matrices to add multiple members to the set.
- If "Source" on page 1-0 is a cell array containing cell arrays of vectors:
  - Specify `src` as a cell array of vectors to add one member to the set.
  - Specify `src` as a cell array containing cell arrays of vectors to add multiple members to the set.
- If "Source" on page 1-0 is a cell array of timetables:
  - Specify `src` as a timetable to add one member to the set.
  - Specify `src` as a cell array of timetables to add multiple members to the set.
- If "Source" on page 1-0 is a datastore, then add members by setting `src` as another datastore that points to new files.

Example: `{randn(10,3),randn(17,9)}` specifies two members. The first member contains three 10-sample signals. The second member contains nine 17-sample signals.

Example: `{{randn(10,1)},{randn(17,1),randn(27,1)}}` specifies two members. The first member contains one 10-sample signal. The second member contains a 17-sample signal and a 27-sample signal.

Example: `{{timetable(seconds(1:10)',randn(10,3)),timetable(seconds(1:7)',randn(7,2))},{timetable(seconds(1:3)',randn(3,1))}}` specifies two members. The first member contains three signals sampled at 1 Hz for 10 seconds and two signals sampled at 1 Hz for 7 seconds. The second member contains one signal sampled at 1 Hz for 3 seconds.

Example: signalDatastore Object Pointing to Files

Specify the path to a set of sample sound signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'
```

```
nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to `Fs`, which is common to all files. Generate a subset of the datastore that excludes the file `mtlb.mat`, which differs from the other files in that the signal variable is not called `y`.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sdss = subset(sds,~strcmp(nms,"mtlb.mat"));
```

Use the subset datastore as the source for a `labeledSignalSet` object.

```
lss = labeledSignalSet(sdss)
```

```
lss = 
  labeledSignalSet with properties:

             Source: [1x1 signalDatastore]
         NumMembers: 6
    TimeInformation: "inherent"
             Labels: [6x0 table]
        Description: ""

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

`tinfo` — Time information for new members

scalar | vector | matrix | `duration` scalar | `duration` vector

Time information for new members, specified as a scalar, a vector, a matrix, a duration scalar, or a duration vector. This argument is valid only if the “TimeInformation” on page 1-0 property of `lss` is `'sampleRate'`, `'sampleTime'`, or `'timeValues'`.

- If “TimeInformation” on page 1-0 is `'sampleRate'`, then `tinfo` specifies sample rate values.
- If “TimeInformation” on page 1-0 is `'sampleTime'`, then `tinfo` specifies sample time values.
- If “TimeInformation” on page 1-0 is `'timeValues'`, then `tinfo` specifies time values.

If you add multiple members to a set, then specifying only one value of `tinfo` sets the same value for all members. If you want to specify a different value for each new member, then set `tinfo` to have multiple values.

When no source has been specified, or when the labeled signal set source is empty, you can change the “TimeInformation” on page 1-0 property to `'sampleRate'`, `'sampleTime'`, or `'timeValues'` to make `lss` interpret `tinfo` correctly.

Example: `addMembers(ks,{randn(10,5),randn(10,3)},seconds([1 2]))` adds two new members with different time information to `ks = labeledSignalSet(randn(10,3),'SampleTime',seconds(1))`.

Example: `addMembers(ks,{randn(10,5),randn(10,3)},[1:10;2:2:20]')` adds two new members with different time information to `ks = labeledSignalSet(randn(10,3),'TimeValues',1:10)`.

`mnames` — Member names
character vector | string scalar | cell array of character vectors | string array

Member names, specified as a character vector, a string scalar, a cell array of character vectors, or a string array.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},'MemberNames',{'llama' 'alpaca'})` specifies a set of random signals with two members, `'llama'` and `'alpaca'`.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

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 correspondence in a table.

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

```
ans=31×2 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
⋮
```

Introduced before R2006a

appcoef

1-D approximation coefficients

Syntax

```
A = appcoef(C,L,wname)
A = appcoef(C,L,LoR,HiR)
A = appcoef( ___ ,N)
```

Description

`A = appcoef(C,L,wname)` returns the approximation coefficients at the coarsest scale using the wavelet decomposition structure `[C,L]` of a 1-D signal and the wavelet specified by `wname`. (See `wavedec` for more information.)

`A = appcoef(C,L,LoR,HiR)` uses the lowpass reconstruction filter `LoR` and highpass reconstruction filter `HiR`. (See `wfilters` for more information.)

`A = appcoef( ___ ,N)` returns the approximation coefficients at level `N`. If `[C,L]` is the `M`-level wavelet decomposition structure of a 1-D signal, then `0 ≤ N ≤ M`.

Examples

Level 3 Approximation Coefficients

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

Load the signal consisting of electricity usage data.

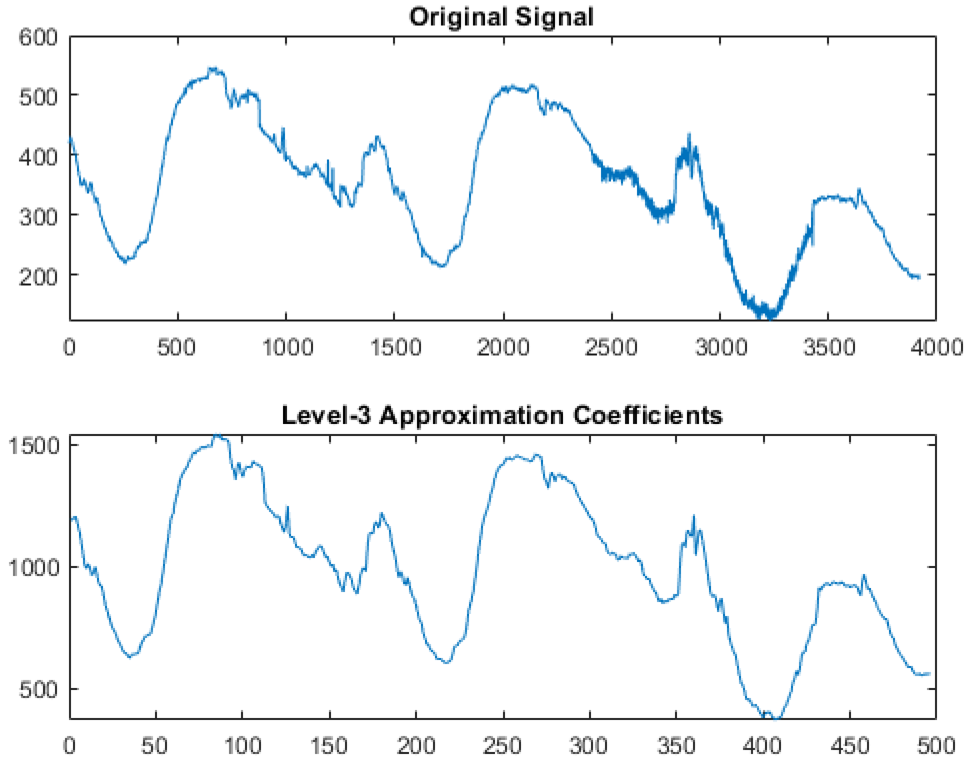
```
load leleccum;
sig = leleccum(1:3920);
```

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');
```


Original Signal

Level-3 Approximation Coefficients

You can substitute any value from 1 to 5 for `Lev` to obtain the approximation coefficients for the corresponding level.

Input Arguments

`C` — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector of a 1-D signal, specified as a real-valued vector. `C` is the output of `wavedec`. The bookkeeping vector `L` is used to parse the coefficients in the wavelet decomposition vector by level.

Example: `[C,L] = wavedec(randn(1,256),4,'coif1')` returns the 4-level wavelet decomposition of a vector.

Data Types: `single` | `double`

`L` — Bookkeeping vector

vector of positive integers

Bookkeeping vector of the wavelet decomposition of a 1-D signal, specified as a vector of positive integers. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition vector `C` by level.

Example: `[C,L] = wavedec(randn(1,256),4,'coif1')` returns the 4-level wavelet decomposition of a vector.

Data Types: single | double

wname — Wavelet
character vector | string scalar

Wavelet used to generate the wavelet decomposition of a 1-D signal, specified as a character vector or string scalar. The wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See wavemngr for the wavelets available in each family.

Example: 'db4'

LoR — Wavelet lowpass reconstruction filter
even-length real-valued vector

Wavelet lowpass reconstruction filter, specified as an even-length real-valued vector. LoR must be the same length as HiR. LoR must be the lowpass reconstruction filter associated with the wavelet used to create the wavelet decomposition structure [C,L]. (See wfilters for more information.)

HiR — Wavelet highpass reconstruction filter
even-length real-valued vector

Wavelet highpass reconstruction filter, specified as an even-length real-valued vector. HiR must be the same length as LoR. HiR must be the highpass reconstruction filter associated with the wavelet used to create the wavelet decomposition structure [C,L]. (See wfilters for more information.)

N — Approximation coefficients level
positive integer

Approximation coefficients level, specified as a positive integer. If [C,L] is the M-level wavelet decomposition structure of a 1-D signal, then 0 ≤ N ≤ M.

Output Arguments

A — Approximation coefficients
real-valued vector

Approximation coefficients at level N, returned as a real-valued vector.

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.
- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`detcoef` | `wavedec`

Introduced before R2006a

appcoef2

2-D approximation coefficients

Syntax

```
A = appcoef2(C,S,wname)
A = appcoef2(C,S,LoR,HiR)
A = appcoef2( ___ ,N)
```

Description

`A = appcoef2(C,S,wname)` returns the approximation coefficients at the coarsest scale using the wavelet decomposition structure [`C`,`S`] of a 2-D signal and the wavelet specified by `wname`. (See `wavedec2` for more information.)

`A = appcoef2(C,S,LoR,HiR)` uses the lowpass reconstruction filter `LoR` and highpass reconstruction filter `HiR`. (See `wfilters` for more information.)

`A = appcoef2( ___ ,N)` returns the approximation coefficients at level `N`. If [`C`,`S`] is the `M`-level wavelet decomposition structure of a 2-D signal, then `0 ≤ N ≤ M`.

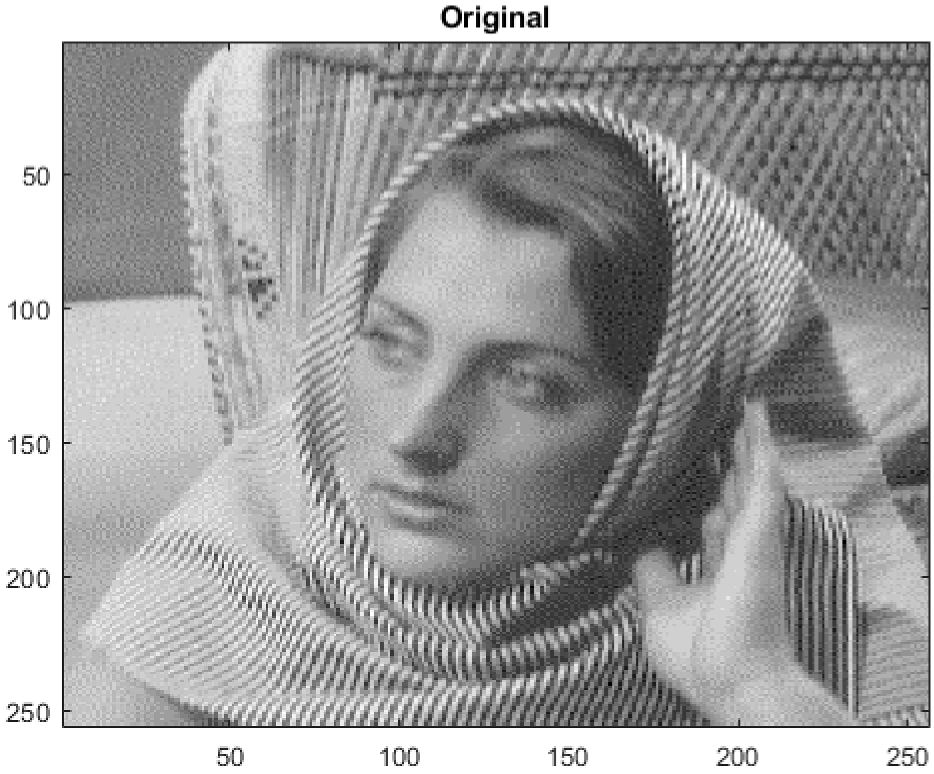
Examples

Reconstruct Approximation Coefficients of an Image

This example shows how to reconstruct approximation coefficients from a multilevel wavelet decomposition of an image.

Set the DWT extension mode to zero-padding. Load and display an image.

```
origmode = dwtmode('status','nodisplay');
dwtmode('zpd','nodisp')
load woman
image(X)
colormap(map)
title('Original')
```



```
size(X)
```

```
ans = 1×2

   256   256
```

Perform a three-level wavelet decomposition of the image using the `db1` wavelet. Display the number of elements in the coefficients array `cfs`, and the contents of the bookkeeping matrix `inds`. Note that `cfs` has the same number of elements as X.

```
wv = 'db1';
[cfs,inds] = wavedec2(X,3,wv);
numel(X)
```

```
ans = 65536
```

```
numel(cfs)
```

```
ans = 65536
```

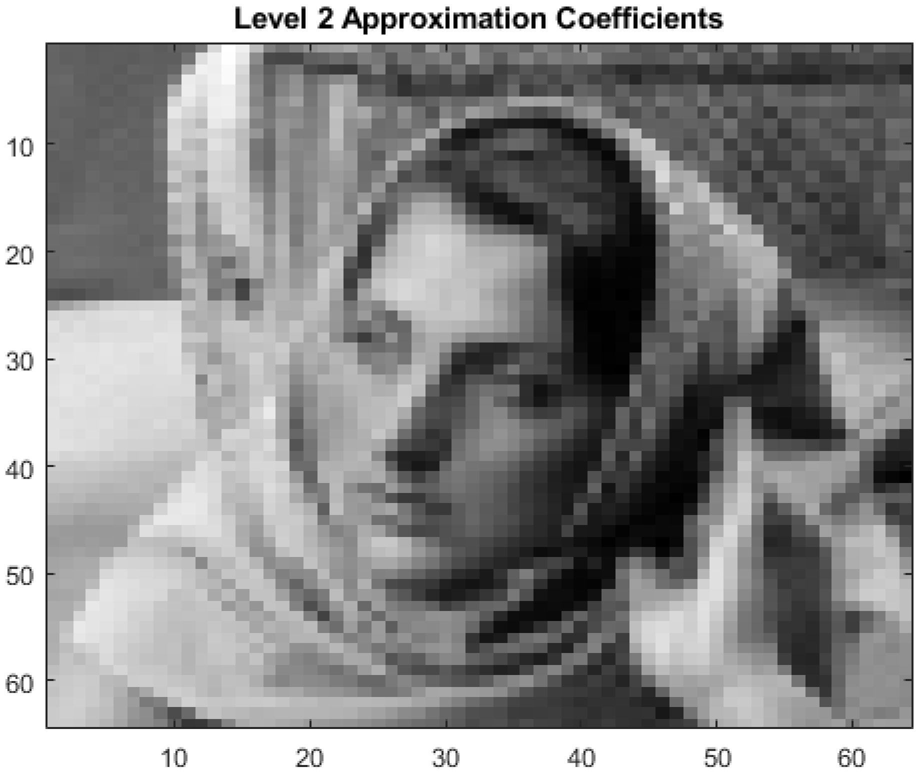
```
inds
```

```
inds = 5×2

    32    32
    32    32
    64    64
   128   128
```

```
    256    256
```

Extract and display the approximation coefficients at level 2.

```
cfs2 = appcoef2(cfs,inds,wv,2);
figure
imagesc(cfs2)
colormap('gray')
title('Level 2 Approximation Coefficients')
```

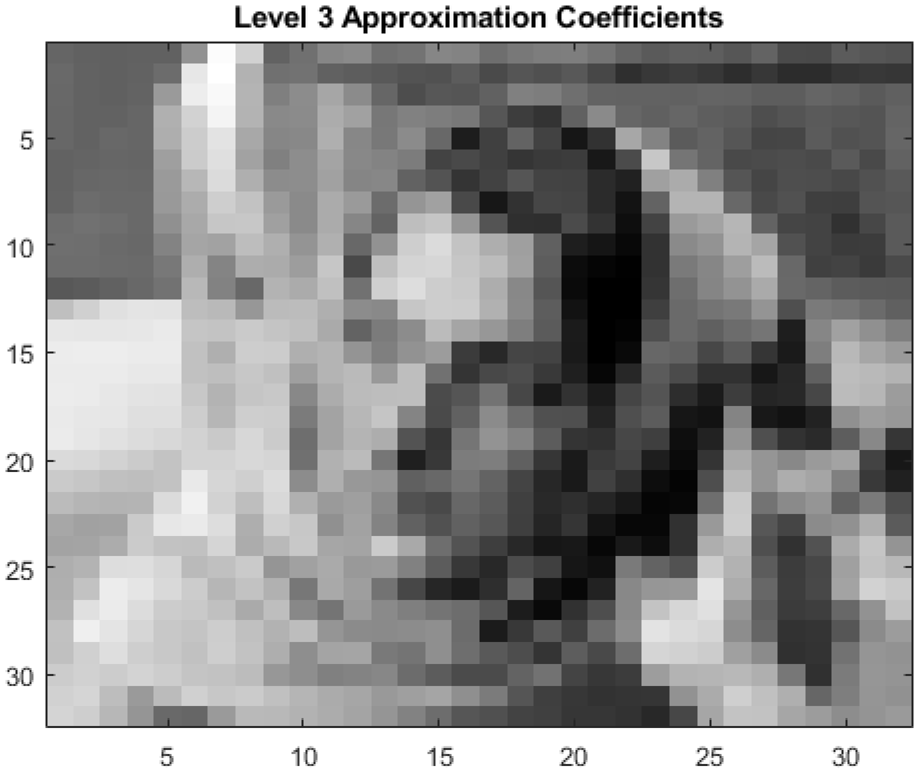


```
size(cfs2)
```

```
ans = 1×2

    64    64
```

Extract and display the approximation coefficients at level 3.

```
cfs3 = appcoef2(cfs,inds,wv,3);
figure
imagesc(cfs3)
colormap('gray')
title('Level 3 Approximation Coefficients')
```



```
size(cfs3)
```

```
ans = 1×2

    32    32
```

Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

Input Arguments

C — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector of a 2-D signal, specified as a real-valued vector. `C` is the output of `wavedec2`. The bookkeeping matrix `S` contains the dimensions of the coefficients by level.

Example: `[C,S] = wavedec2(randn(256,256),4,'db4')` returns the 4-level wavelet decomposition of a matrix.

Data Types: `double`

S — Bookkeeping matrix

matrix of positive integers

Bookkeeping matrix of the wavelet decomposition of a 2-D signal, specified as a matrix of positive integers. The bookkeeping matrix is used to parse the coefficients in the wavelet decomposition vector `C` by level.

Example: `[C,S] = wavedec2(randn(256,256),4,'db4')` returns the 4-level wavelet decomposition of a matrix.

Data Types: `double`

`wname` — Wavelet
character vector | string scalar

Wavelet used to generate the wavelet decomposition of a 2-D signal, specified as a character vector or string scalar. The wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wavemngr` for the wavelets available in each family.

Example: `'db4'`

`LoR` — Wavelet lowpass reconstruction filter
even-length real-valued vector

Wavelet lowpass reconstruction filter, specified as an even-length real-valued vector. `LoR` must be the same length as `HiR`. `LoR` must be the lowpass reconstruction filter associated with the wavelet used to create the wavelet decomposition structure [`C`,`S`]. (See `wfilters` for more information.)

Data Types: `double`

`HiR` — Wavelet highpass reconstruction filter
even-length real-valued vector

Wavelet highpass reconstruction filter, specified as an even-length real-valued vector. `HiR` must be the same length as `LoR`. `HiR` must be the highpass reconstruction filter associated with the wavelet used to create the wavelet decomposition structure [`C`,`S`]. (See `wfilters` for more information.)

Data Types: `double`

`N` — Approximation coefficients level
positive integer

Approximation coefficients level, specified as a positive integer. If [`C`,`S`] is the `M`-level wavelet decomposition structure of a 2-D signal, then `0 ≤ N ≤ M`.

Data Types: `double`

Output Arguments

`A` — Approximation coefficients
real-valued matrix | real-valued 3-D array

Approximation coefficients at level `N`, returned as a real-valued matrix or 3-D real-valued array. 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 `image` and `imfinfo`.

Data Types: `double`

Algorithms

The input vector `C` and bookkeeping matrix `S` contain all the information about the 2-D signal decomposition.

Let `NMAX = size(S,1)-2`; then `C = [A(NMAX) H(NMAX) V(NMAX) D(NMAX) … H(1) V(1) D(1)]` where `A`, `H`, `V`, and `D` are vectors. If `N = NMAX`, then a simple extraction is done; otherwise, `appcoef2` 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.
- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`detcoef2` | `wavedec2`

Introduced before R2006a

basisPursuit

Recover sparse signal using the basis pursuit algorithm

Syntax

```
[Xr,MSE,lambda] = basisPursuit(A,Y)
[Xr,MSE,lambda] = basisPursuit(___,Name=Value)
```

Description

`[Xr,MSE,lambda] = basisPursuit(A,Y)` recovers the sparse signal approximation `Xr` of `Y` by solving the “Basis Pursuit Denoising Problem” on page 1-31 using the `sensingDictionary` `A`. The `basisPursuit` function also returns the minimum mean squared error `MSE` and the corresponding Lagrangian parameter `lambda`.

`[Xr,MSE,lambda] = basisPursuit(___,Name=Value)` specifies options using one or more name-value arguments in addition to the input argument in the previous syntax. For example, `[Xr,MSE,lambda] = basisPursuit(A,Y,RelTol=5e-2)` sets a relative tolerance of `5e-2`.

Examples

Basis Pursuit Approximation of Signal

Load the ECG signal.

```
load wecg
```

Create a sensing dictionary that can be applied to the signal. Use the `dct` basis type.

```
D = sensingDictionary(Size=length(wecg),Type={'dct'});
```

Obtain the best fit for the signal using the dictionary and basis pursuit. Obtain the minimum mean squared error.

```
[XBP,MSE,lambda] = basisPursuit(D,wecg);
MSE
```

```
MSE = 1.2349e-04
```

Extract the sensing dictionary matrix. Use the matrix to construct the approximation.

```
A = subdict(D,1:D.Size(1),1:D.Size(2));
wecgR = A*XBP;
```

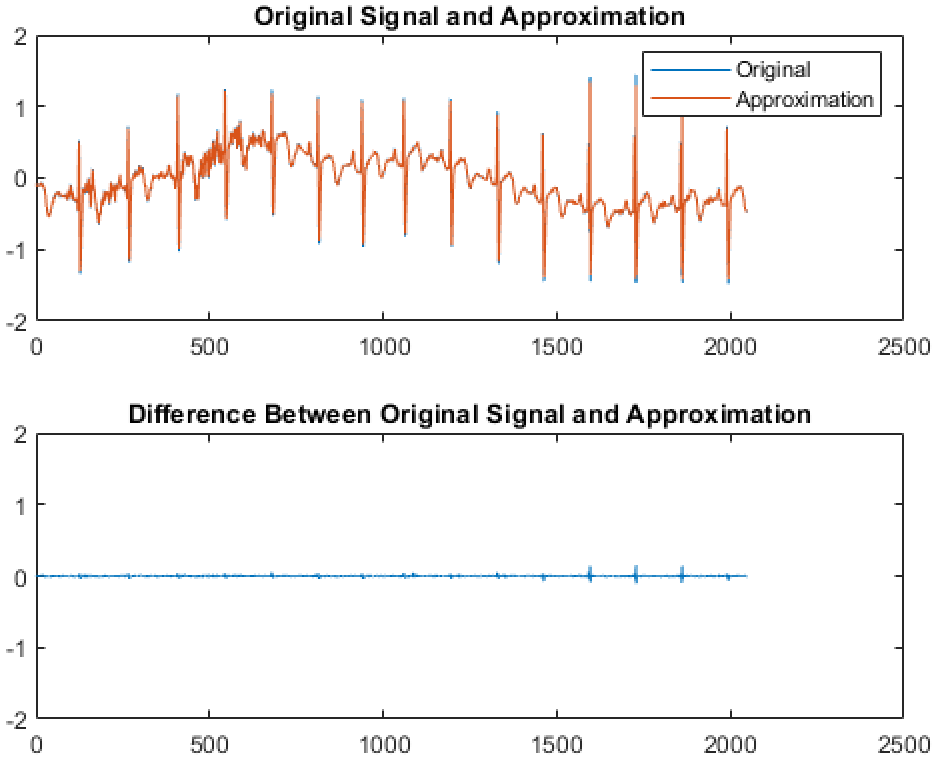
Obtain the norm of the difference between the original signal and its approximation.

```
norm(wecg-wecgR)
```

```
ans = 0.5029
```

Plot the signal and the approximation. Plot the difference at the same scale.

```
subplot(2,1,1)
plot(wecg)
hold on
plot(wecgR)
hold off
legend("Original","Approximation")
title("Original Signal and Approximation")
ylimits = get(gca,"YLim");
subplot(2,1,2)
plot(wecg-wecgR)
ylim(ylimits)
title("Difference Between Original Signal and Approximation")
```



Input Arguments

A — Sensing dictionary

`sensingDictionary` object

Sensing dictionary, specified as a `sensingDictionary` object.

Y — Sensor measurements

vector

Sensor measurements, specified as a vector `Y` such that `Y = AX`, where `X` is a sparse signal.

Data Types: `single` | `double`

Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Xr = basisPursuit(A,Y,RelTol=1e-3)` recovers `Xr` using stopping criteria based on a relative tolerance of `1e-3`.

`maxIterations` — Maximum number of iterations

`200` (default) | positive integer

Maximum number of iterations executed to recover the sparse signal, specified as a positive integer.

Example: `Xr = basisPursuit(A,Y,maxIterations=35)` recovers `Xr` using at most 35 iterations.

Data Types: `single` | `double`

`RelTol` — Relative tolerance

`1e-4` (default) | positive scalar

Relative tolerance used to recover the signal, specified as a positive scalar. The stopping criteria is based on the relative tolerance.

Example: `Xr = basisPursuit(A,Y,RelTol=1e-3)` recovers `Xr` using stopping criteria based on a relative tolerance of `1e-3`.

Data Types: `single` | `double`

`AbsTol` — Absolute tolerance

`1e-5` (default) | positive scalar

Absolute tolerance used to recover the signal, specified as a positive scalar. The stopping criteria is based on the absolute tolerance.

Example: `Xr = basisPursuit(A,Y,AbsTol=1e-4)` recovers `Xr` using stopping criteria based on a absolute tolerance of `1e-4`.

Data Types: `single` | `double`

`MaxErr` — Maximum error

positive scalar

Maximum error used to recover the signal, specified as a positive scalar. The `basisPursuit` function recovers the `Xr` that satisfies

$$\|Y - AX_r\|_2^2 \le \text{MaxErr}.$$

If unspecified, `Xr` is the solution of the “Basis Pursuit Denoising Problem” on page 1-31.

Example: `Xr = basisPursuit(A,Y,MaxErr=1e-1)` recovers `Xr` using stopping criteria based on a maximum error of `1e-1`.

Data Types: `single` | `double`

Output Arguments

Xr — Sparse signal
vector

Sparse signal recovered, returned as a vector.

Data Types: single | double
Complex Number Support: Yes

MSE — Minimum mean squared error
scalar

Minimum mean squared error, returned as a scalar.

Data Types: single | double

lambda — Lagrangian parameter
scalar

Lagrangian parameter, returned as a scalar.

Data Types: single | double

More About

Basis Pursuit Denoising Problem

Basis pursuit denoising recovers the sparse signal Xr by solving

$$\min_{X} \frac{1}{2} \left\| Y - AX \right\|_2^2 + \lambda \left\| X \right\|_1,$$

where

- A — Sensing dictionary
- Y — Measurement vector
- λ — Lagrangian parameter. Adjusting λ controls the balance between sparsity and accuracy of reconstruction.

Extended Capabilities

Tall Arrays
Calculate with arrays that have more rows than fit in memory.

This function supports tall arrays with the limitations:

- If the value of the CustomDictionary property of the sensingDictionary A is a tall array, then the sensor measurements Y must also be a tall array.

For more information, see “Tall Arrays”.

See Also
matchingPursuit | sensingDictionary

Topics
"Signal Deconvolution and Impulse Denoising Using Pursuit Methods"
"Matching Pursuit Algorithms"

Introduced in R2022a

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.

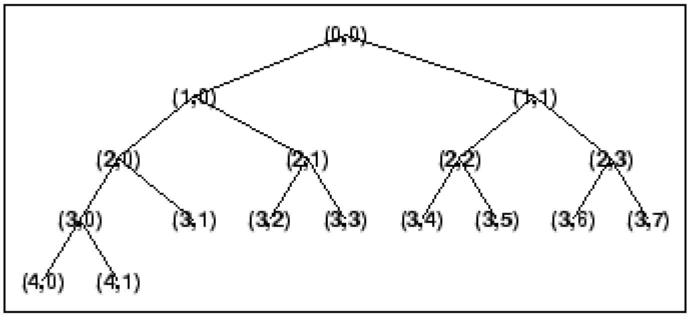
`T = bestlevt(T)` computes the modified wavelet packet tree *T* corresponding to the best level tree decomposition.

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

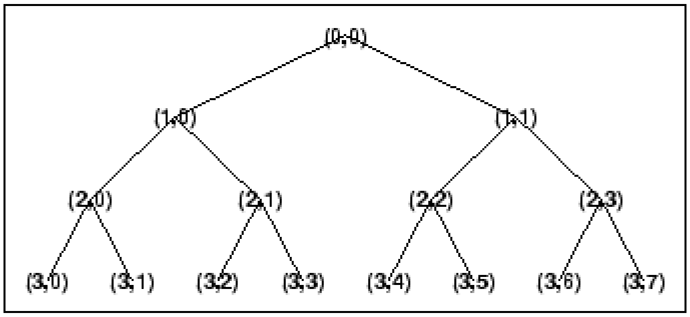
The optimal entropy of the node, whose index is `j-1`, is `E(j)`.

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');
```

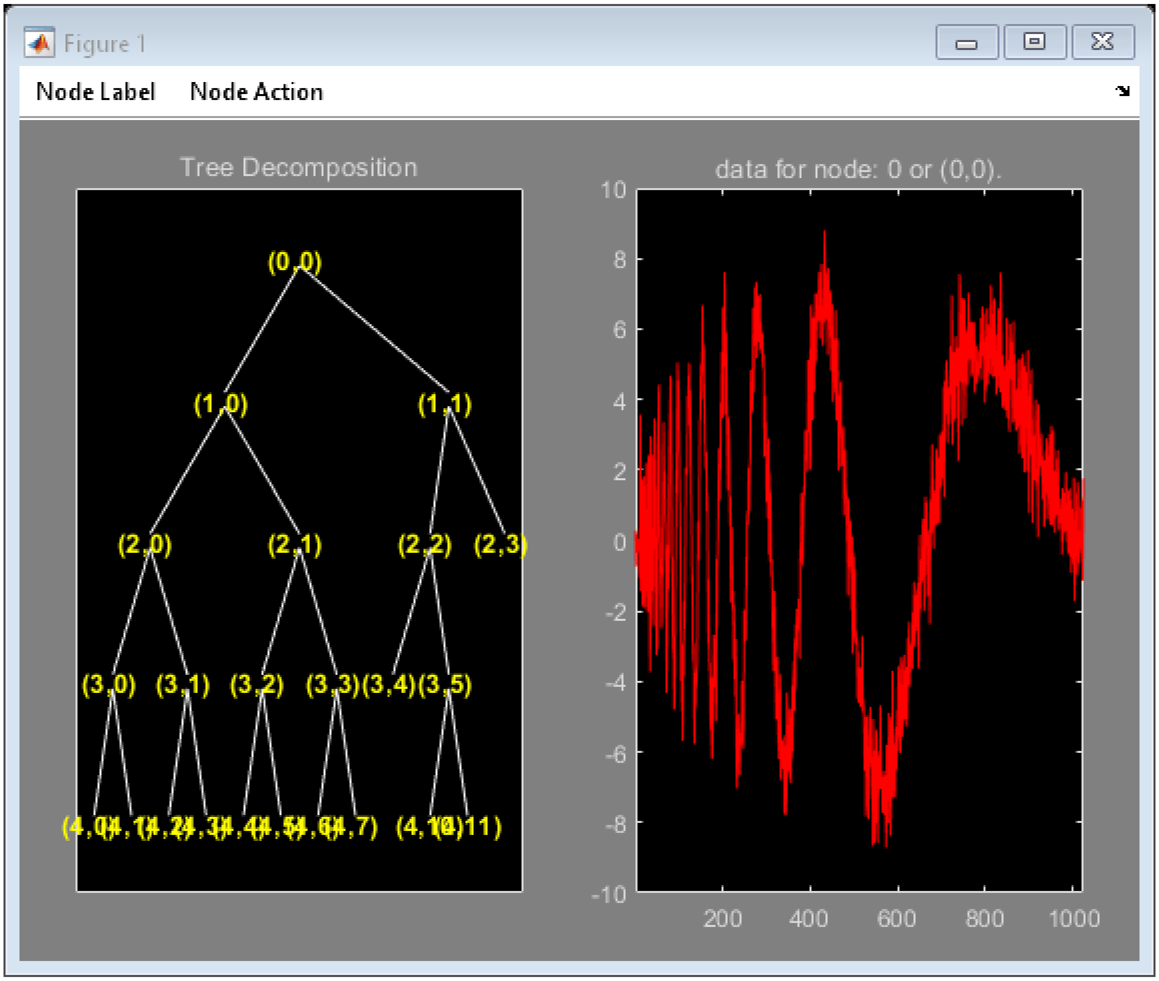
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

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

****************************************
**  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

```
[LoD,HiD,LoR,HiR] = biorfilt(DF,RF)
[LoD1,HiD1,LoR1,HiR1,LoD2,HiD2,LoR2,HiR2] = biorfilt(DF,RF,'8')
```

Description

`[LoD,HiD,LoR,HiR] = biorfilt(DF,RF)` returns four filters associated with the biorthogonal wavelet specified by decomposition filter `DF` and reconstruction filter `RF`. These filters are

- `LoD` — Decomposition lowpass filter
- `HiD` — Decomposition highpass filter
- `LoR` — Reconstruction lowpass filter
- `HiR` — Reconstruction highpass filter

`[LoD1,HiD1,LoR1,HiR1,LoD2,HiD2,LoR2,HiR2] = biorfilt(DF,RF,'8')` returns eight filters, the first four associated with the decomposition wavelet, and the last four associated with the reconstruction wavelet.

Examples

Biorthogonal Filters and Transfer Functions

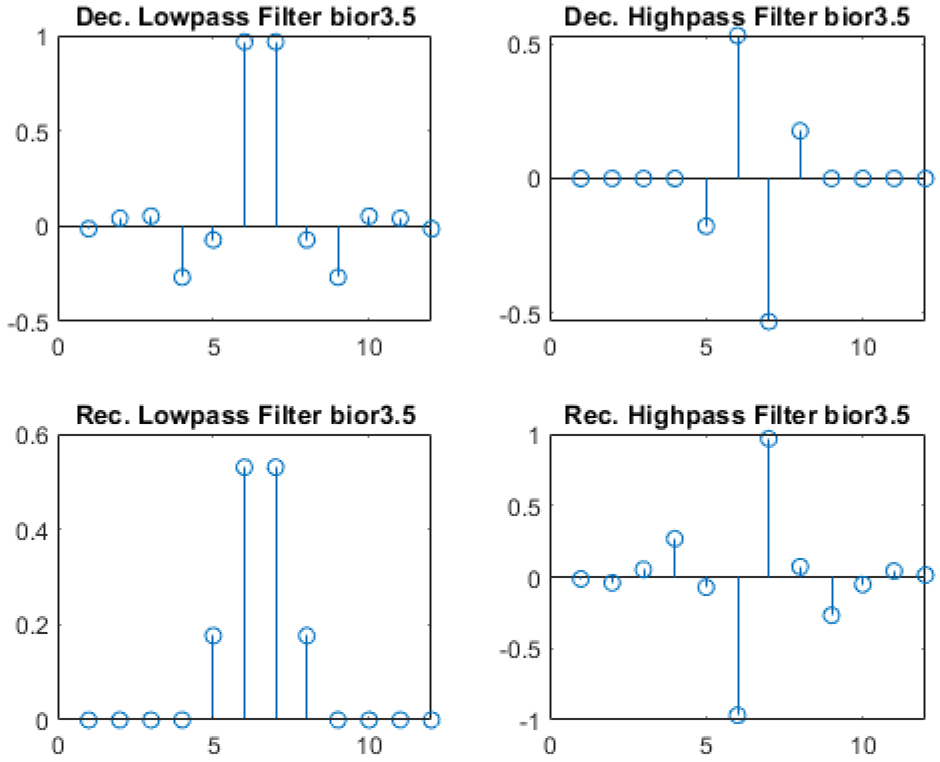
This example shows how to obtain the decomposition (analysis) and reconstruction (synthesis) filters for the `'bior3.5'` wavelet.

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

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

Plot the filter impulse responses.

```
subplot(2,2,1)
stem(LoD)
title(['Dec. Lowpass Filter ',wv])
subplot(2,2,2)
stem(HiD)
title(['Dec. Highpass Filter ',wv])
subplot(2,2,3)
stem(LoR)
title(['Rec. Lowpass Filter ',wv])
subplot(2,2,4)
stem(HiR)
title(['Rec. Highpass Filter ',wv])
```



Dec. Lowpass Filter bior3.5

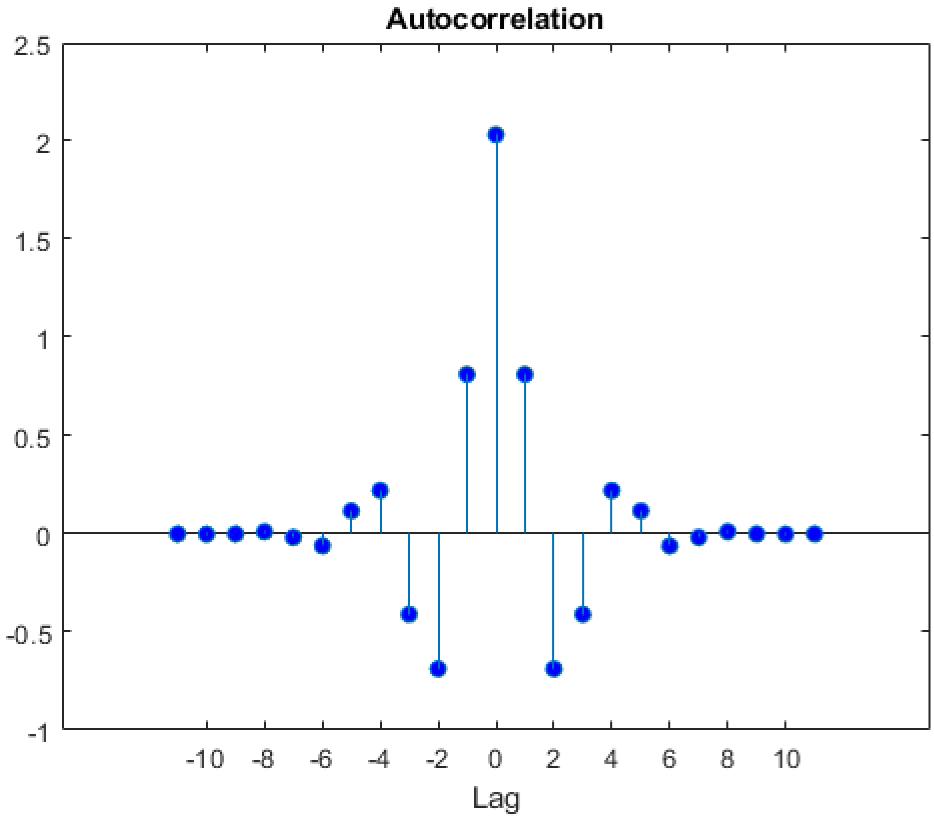
Dec. Highpass Filter bior3.5

Rec. Lowpass Filter bior3.5

Rec. Highpass Filter bior3.5

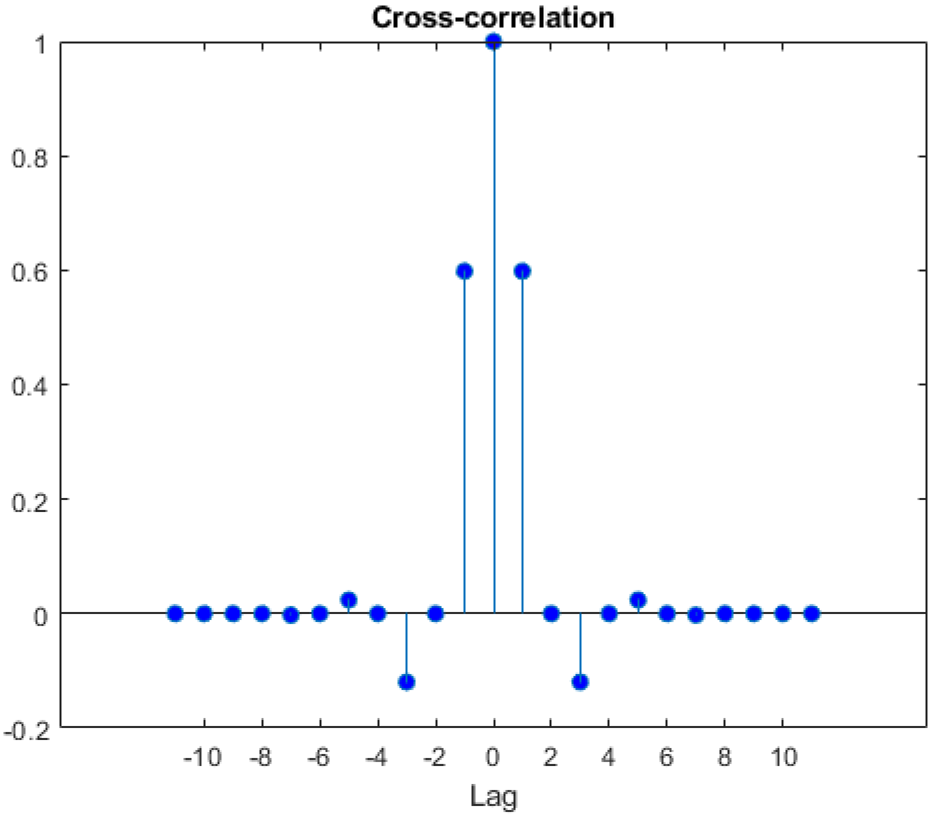
Demonstrate that autocorrelations at even lags are only zero for dual pairs of filters. Examine the autocorrelation sequence for the lowpass decomposition filter.

```
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)
title('Autocorrelation')
xlabel('Lag')
```



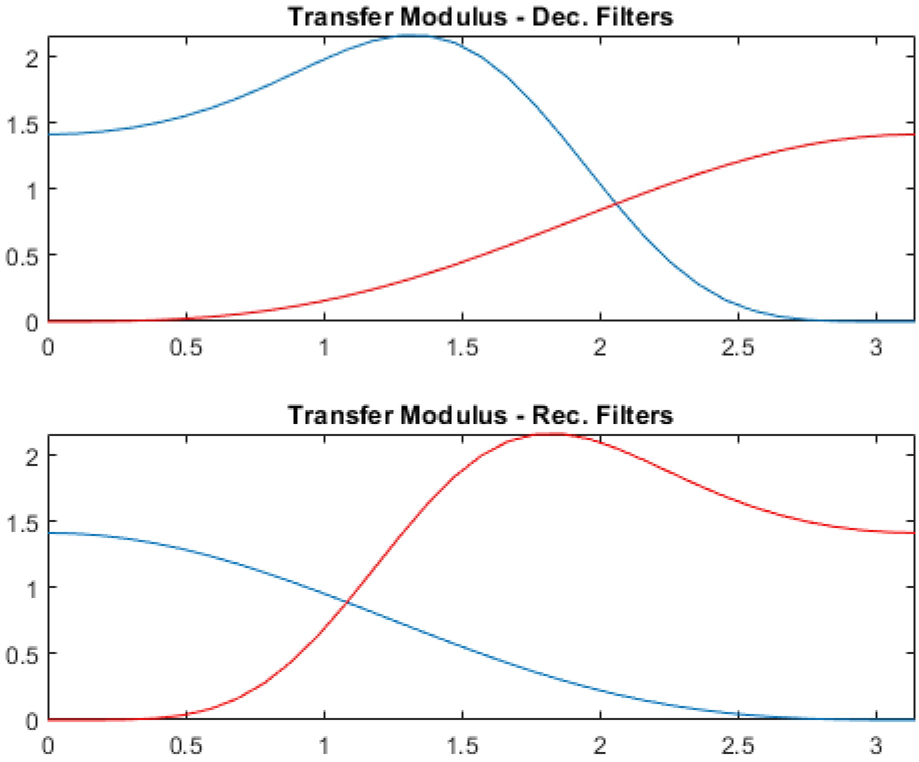
Examine the cross-correlation sequence for the lowpass decomposition and synthesis filters. Compare the result with the preceding figure. At even lags, the cross-correlation is zero.

```
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)
title('Cross-correlation')
xlabel('Lag')
```



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(2,1,1)
plot(freqvec,abs(dftLoD),freqvec,abs(dftHiD),'r')
axis tight
title('Transfer Modulus - Dec. Filters')
subplot(2,1,2)
plot(freqvec,abs(dftLoR),freqvec,abs(dftHiR),'r')
axis tight
title('Transfer Modulus - Rec. Filters')
```



Input Arguments

DF — Decomposition scaling filter

vector

Decomposition scaling filter associated with a biorthogonal wavelet, specified as a vector.

Data Types: `double`

RF — Reconstruction scaling filter

vector

Reconstruction scaling filter associated with a biorthogonal wavelet, specified as a vector.

Data Types: `double`

Output Arguments

`LoD,HiD` — Decomposition filters

even-length real-valued vectors

Wavelet decomposition filters, returned as a pair of even-length real-valued vectors. `LoD` is the lowpass decomposition filter, and `HiD` is the highpass decomposition filter.

LoR,HiR — Reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, returned as a pair of even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter.

LoD1,HiD1,LoR1,HiR1 — Filters
even-length real-valued vectors

Filters associated with the decomposition (analysis) wavelet, returned as even-length real-valued vectors.

- LoD1 — Decomposition lowpass filter
- HiD1 — Decomposition highpass filter
- LoR1 — Reconstruction lowpass filter
- HiR1 — Reconstruction highpass filter

LoD2,HiD2,LoR2,HiR2 — Filters
even-length real-valued vectors

Filters associated with the reconstruction (synthesis) wavelet, returned as even-length real-valued vectors.

- LoD2 — Decomposition lowpass filter
- HiD2 — Decomposition highpass filter
- LoR2 — Reconstruction lowpass filter
- HiR2 — Reconstruction highpass filter

More About

Biorthogonal Filters

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.

References

[1] Cohen, Albert. "Ondelettes, analyses multirésolution et traitement numérique du signal," *Ph. D. Thesis*, University of Paris IX, DAUPHINE. 1992.

[2] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

See Also

`biorwavf` | `orthfilt`

Introduced before R2006a

biorwavf

Biorthogonal spline wavelet filter

Syntax

```
[RF,DF] = biorwavf(wname)
```

Description

`[RF,DF] = biorwavf(wname)` returns the reconstruction (synthesis) and decomposition (analysis) scaling filters, `RF` and `DF`, respectively, associated with the biorthogonal wavelet specified by `wname`.

Examples

Biorthogonal Spline Wavelet Filter

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

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

```
RF = 1×3

    0.2500    0.5000    0.2500


DF = 1×5

   -0.1250    0.2500    0.7500    0.2500   -0.1250
```

Add Biorthogonal Wavelet Filters

This example shows how to take analysis and synthesis filters associated with a biorthogonal wavelet and make them compatible with Wavelet Toolbox™. Wavelet Toolbox requires that analysis and synthesis lowpass and highpass filters have equal even length. This example uses the nearly orthogonal biorthogonal wavelets based on the Laplacian pyramid scheme of Burt and Adelson (Table 8.4 on page 283 in [1]). The example also demonstrates how to examine properties of the biorthogonal wavelets.

Define the analysis and synthesis filter coefficients of the biorthogonal wavelet.

```
Hd = [-1 5 12 5 -1]/20*sqrt(2);
Gd = [3 -15 -73 170 -73 -15 3]/280*sqrt(2);
Hr = [-3 -15 73 170 73 -15 -3]/280*sqrt(2);
Gr = [-1 -5 12 -5 -1]/20*sqrt(2);
```

Hd and Gd are the lowpass and highpass analysis filters, respectively. Hr and Gr are the lowpass and highpass synthesis filters. They are all finite impulse response (FIR) filters. Confirm the lowpass filter coefficients sum to sqrt(2) and the highpass filter coefficients sum to 0.

```
sum(Hd)/sqrt(2)
```

```
ans = 1.0000
```

```
sum(Hr)/sqrt(2)
```

```
ans = 1.0000
```

```
sum(Gd)
```

```
ans = -1.0061e-16
```

```
sum(Gr)
```

```
ans = -9.7145e-17
```

The z-transform of an FIR filter h is a Laurent polynomial $h(z)$ given by $h(z) = \sum_{k = k_b}^{k_e} h_k z^{-k}$. The degree $|h|$ of a Laurent polynomial is defined as $|h| = k_e - k_b$. Therefore, the length of the filter h is $1 + |h|$. Examine the Laurent expansion of the scaling and wavelet filters.

```
PHd = laurentPolynomial(Coefficients=Hd,MaxOrder=2)
```

```
PHd = 
  laurentPolynomial with properties:

    Coefficients: [-0.0707 0.3536 0.8485 0.3536 -0.0707]
        MaxOrder: 2
```

```
PHr = laurentPolynomial(Coefficients=Hr,MaxOrder=3)
```

```
PHr = 
  laurentPolynomial with properties:

    Coefficients: [-0.0152 -0.0758 0.3687 0.8586 0.3687 -0.0758 -0.0152]
        MaxOrder: 3
```

```
PGd = laurentPolynomial(Coefficients=Gd,MaxOrder=3)
```

```
PGd = 
  laurentPolynomial with properties:

    Coefficients: [0.0152 -0.0758 -0.3687 0.8586 -0.3687 -0.0758 0.0152]
        MaxOrder: 3
```

```
PGr = laurentPolynomial(Coefficients=Gr,MaxOrder=2)
```

```
PGr = 
  laurentPolynomial with properties:

    Coefficients: [-0.0707 -0.3536 0.8485 -0.3536 -0.0707]
```

```
        MaxOrder: 2
```

Since the filters are associated with biorthogonal wavelet, confirm $PHd(z)\ PHr(z) + PG(z)\ PGr(z) = 2$.

```
PHd*PHr + PGd*PGr
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 2
        MaxOrder: 0
```

Wavelet Toolbox™ requires that filters associated with the wavelet have even equal length. To use the Laplacian wavelet filters in the toolbox, you must include the missing powers of the Laurent series as zeros.

The degrees of PHd and PHr are 4 and 6, respectively. The minimum even-length filter that can accommodate the four filters has length 8, which corresponds to a Laurent polynomial of degree 7. The strategy is to prepend and append 0s as evenly as possible so that all filters are of length 8. Prepend 0 to all the filters, and then append two 0s to Hd and Gr.

```
Hd = [0 Hd 0 0];
Gd = [0 Gd];
Hr = [0 Hr];
Gr = [0 Gr 0 0];
```

You can examine properties of the biorthogonal wavelets by creating DWT filter banks. Create two custom DWT filter banks using the filters, one for analysis and the other for synthesis. Confirm the filter banks are biorthogonal.

```
fb = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',[Hd' Hr'],...
    'CustomWaveletFilter',[Gd' Gr']);

fb2 = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',[Hd' Hr'],...
    'CustomWaveletFilter',[Gd' Gr'],...
    'FilterType','Synthesis');

fprintf('fb: isOrthogonal = %d\tisBiorthogonal = %d\n',...
    isOrthogonal(fb),isBiorthogonal(fb));
```

```
fb: isOrthogonal = 0    isBiorthogonal = 1
```

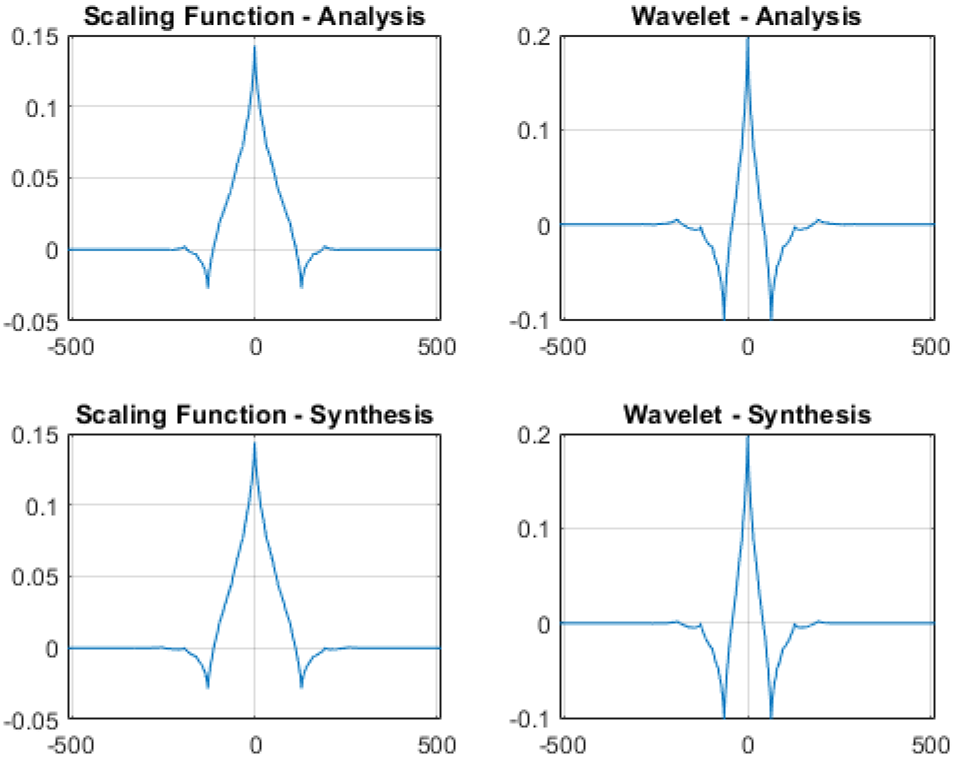
```
fprintf('fb2: isOrthogonal = %d\tisBiorthogonal = %d\n',...
    isOrthogonal(fb2),isBiorthogonal(fb2));
```

```
fb2: isOrthogonal = 0    isBiorthogonal = 1
```

Plot the scaling and wavelet functions associated with the filter banks at the coarsest scale.

```
[phi,t] = scalingfunctions(fb);
[psi,~] = wavelets(fb);
[phi2,~] = scalingfunctions(fb2);
[psi2,~] = wavelets(fb2);
```

```
subplot(2,2,1)
plot(t,phi(end,:))
grid on
title('Scaling Function - Analysis')
subplot(2,2,2)
plot(t,psi(end,:))
grid on
title('Wavelet - Analysis')
subplot(2,2,3)
plot(t,phi2(end,:))
grid on
title('Scaling Function - Synthesis')
subplot(2,2,4)
plot(t,psi2(end,:))
grid on
title('Wavelet - Synthesis')
```



Compute the filter bank framebounds.

```
[analysisLowerBound,analysisUpperBound] = framebounds(fb)
```

```
analysisLowerBound = 0.9505
```

```
analysisUpperBound = 1.0211
```

```
[synthesisLowerBound,synthesisUpperBound] = framebounds(fb2)
```

```
synthesisLowerBound = 0.9800
```

```
synthesisUpperBound = 1.0528
```

Input Arguments

wname — Name of biorthogonal wavelet

character vector | string scalar

Name of biorthogonal wavelet, specified as `'biorNr.Nd'` where possible values for `Nr` and `Nd` are as follows:

`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`

`Nr` and `Nd` are the numbers of vanishing moments for the reconstruction and decomposition filters, respectively.

Example: `'biorwavf3.7'`

Output Arguments

RF — Reconstruction filter

real-valued vector

Reconstruction filter associated with the biorthogonal wavelet `wname`, returned as a real-valued vector.

DF — Decomposition filter

real-valued vector

Decomposition filter associated with the biorthogonal wavelet `wname`, returned as a real-valued vector.

See Also

`biorfilt` | `waveinfo`

Introduced before R2006a

BPfrequencies

CWT filter bank bandpass center frequencies

Note `BPfrequencies` is not recommended and may be removed in a future release. Use `centerFrequencies` instead.

Syntax

```
bpcf = BPfrequencies(fb)
```

Description

`bpcf = BPfrequencies(fb)` returns the wavelet bandpass center frequencies `bpcf` for the CWT filter bank `fb`. Frequencies are ordered from high to low. Frequencies are in cycles/sample if a sampling frequency or sampling period is not specified. If a sampling frequency is specified, `bpcf` has units of hertz. If a sampling period is specified, `bpcf` has units cycles/unit time where the time unit is the same as the duration `SamplingPeriod`.

Examples

Wavelet Bandpass Center Frequencies

Create a CWT filter bank.

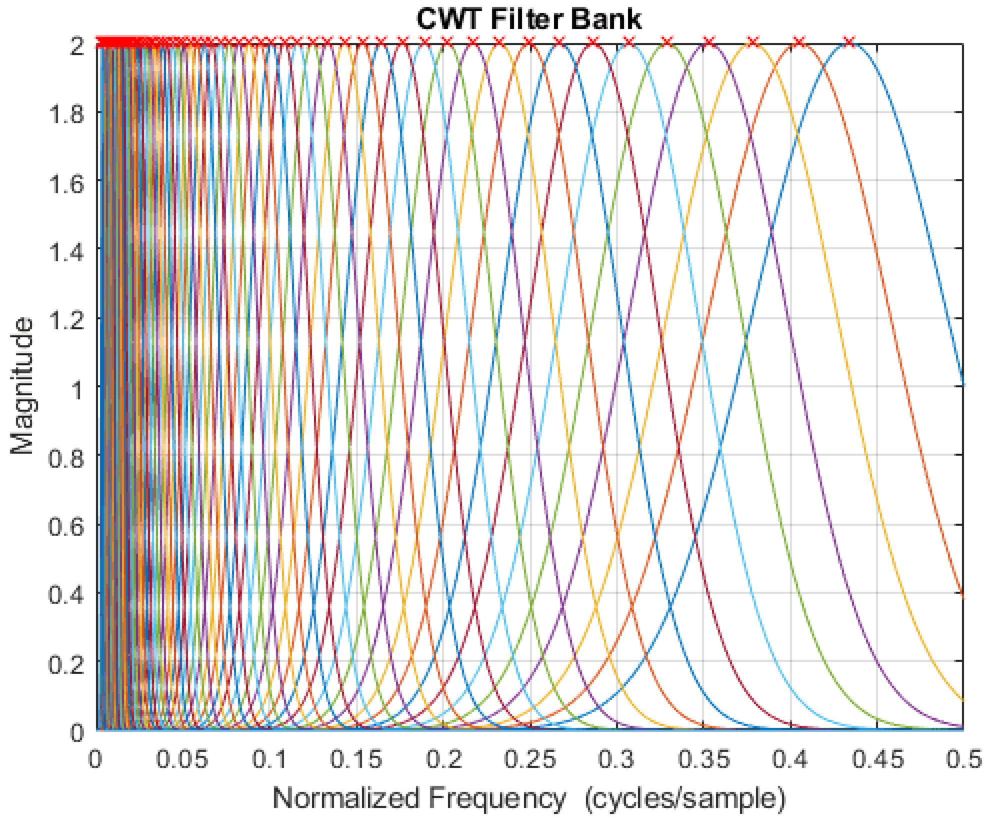
```
fb = cwtfilterbank;
```

Calculate the bandpass center frequencies.

```
bpcf = centerFrequencies(fb);
```

Plot the frequency responses of the filter bank and the bandpass center frequencies. The bandpass center frequencies correspond to the peaks of the frequency response of each wavelet in the filter bank.

```
freqz(fb)
hold on
plot(bpcf,2*ones(size(bpcf)),'rx')
```



Input Arguments

`fb` — Continuous wavelet transform filter bank

`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

`bpcf` — Wavelet bandpass center frequencies

real-valued vector

Wavelet bandpass center frequencies, returned as a real-valued vector of length `Ns` where `Ns` is the number of scales in the filter bank. Frequencies are ordered from high to low. Frequencies are in cycles/sample if a sampling frequency or sampling period is not specified. If a sampling frequency is specified, `bpcf` has units of hertz. If a sampling period is specified, `bpcf` has units cycles/unit time where the time unit is the same as the duration `SamplingPeriod`.

Compatibility Considerations

`BPfrequencies` will be removed

Not recommended starting in R2018b

The BPfrequencies object function of cwtfilterbank has been renamed centerFrequencies. The functionality remains unchanged. BPfrequencies will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
BPfrequencies	Still runs	Use centerFrequencies	Replace all instances of BPfrequencies with centerFrequencies.

See Also

cwtfilterbank | powerbw | freqz | centerPeriods

Introduced in R2018a

BPperiods

CWT filter bank bandpass periods

Note `BPperiods` is not recommended and may be removed in a future release. Use `centerPeriods` instead.

Syntax

```
p = BPperiods(fb)
```

Description

`p = BPperiods(fb)` returns the wavelet bandpass periods, `p`, for the continuous wavelet transform (CWT) filter bank, `fb`.

Examples

Wavelet Filter Bank Bandpass Periods

Create two CWT filter banks. Set the sampling period of the first filter bank to 0.5 seconds, and the sampling frequency of the second filter bank to 2 Hz.

```
fb = cwtfilterbank('SamplingPeriod',seconds(0.5));
fb2 = cwtfilterbank('SamplingFrequency',2);
```

Obtain the bandpass center periods of both filter banks. Confirm the center periods of both filter banks are equal.

```
bp = centerPeriods(fb);
bp2 = centerPeriods(fb2);
bp(1:5)
```

```
ans = 5x1 duration
   1.1517 sec
   1.2344 sec
    1.323 sec
    1.418 sec
   1.5197 sec
```

```
bp2(1:5)
```

```
ans = 5×1

    1.1517
    1.2344
    1.3230
    1.4180
    1.5197
```

Obtain the bandpass center frequencies of the second filter bank. Confirm the reciprocals of the center frequencies are equal to the center periods.

```
f2 = centerFrequencies(fb2);
1./f2(1:5)
```

```
ans = 5×1

    1.1517
    1.2344
    1.3230
    1.4180
    1.5197
```

Input Arguments

fb — Continuous wavelet transform filter bank

cwtfilterbank object

Continuous wavelet transform (CWT) filter bank, specified as a cwtfilterbank object.

Output Arguments

p — Wavelet bandpass filter periods

real-valued vector | duration array

Wavelet bandpass filter periods, returned as a real-valued vector of length Ns where Ns is the number of scales in the filter bank.

If SamplingPeriod is specified, p is a duration array with the same units and format as SamplingPeriod. If SamplingFrequency is specified, p is in seconds.

Compatibility Considerations

BPperiods will be removed

Not recommended starting in R2018b

The BPperiods object function of cwtfilterbank has been renamed centerPeriods. The functionality remains unchanged. BPperiods will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
BPperiods	Still runs	Use centerPeriods	Replace all instances of BPperiods with centerPeriods.

See Also

cwtfilterbank | powerbw | freqz | centerFrequencies

Introduced in R2018a

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 = liftingScheme(Wavelet="cdf3.1");
```

Display the lifting scheme.

```
disp(lscdf)
```

```
      Wavelet               : 'cdf3.1'
     LiftingSteps          : [3 × 1] liftingStep
     NormalizationFactors  : [2.1213 0.4714]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'update'
    Coefficients: -0.3333
        MaxOrder: -1

            Type: 'predict'
    Coefficients: [-0.3750 -1.1250]
```

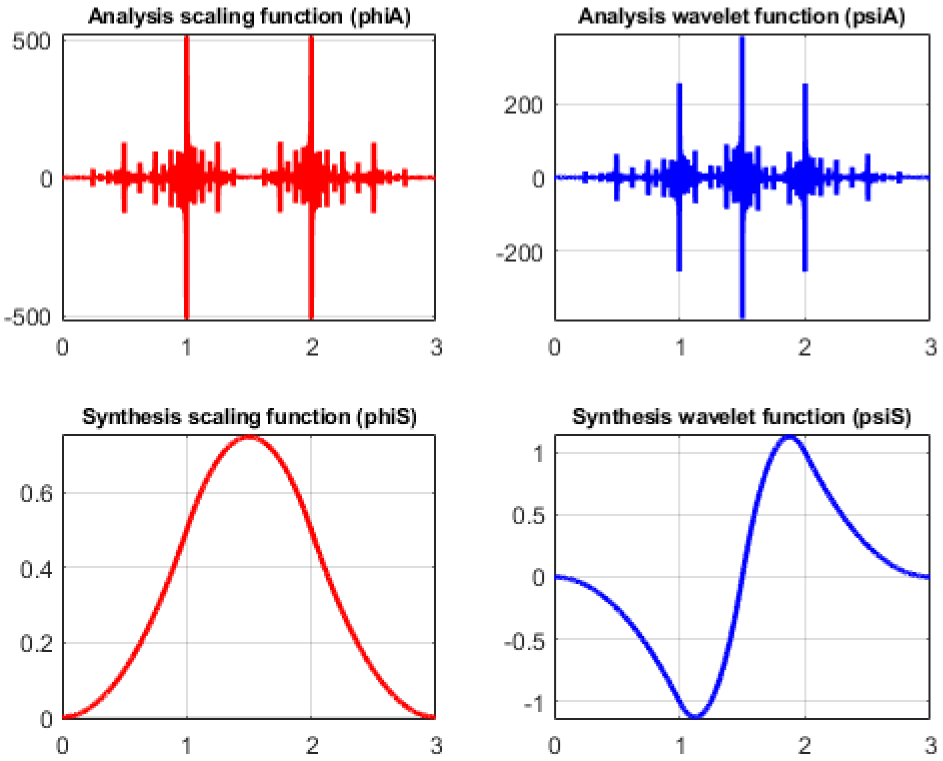
```
        MaxOrder: 1

            Type: 'update'
    Coefficients: 0.4444
        MaxOrder: 0
```

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

centerFrequencies

CWT filter bank bandpass center frequencies

Syntax

```
bpcf = centerFrequencies(fb)
```

Description

`bpcf = centerFrequencies(fb)` returns the wavelet bandpass center frequencies `bpcf` for the CWT filter bank `fb`. Frequencies are ordered from high to low. Frequencies are in cycles/sample if a sampling frequency or sampling period is not specified. If a sampling frequency is specified, `bpcf` has units of hertz. If a sampling period is specified, `bpcf` has units of cycles/unit time, where the time unit is the same as the duration `SamplingPeriod`.

Examples

Wavelet Bandpass Center Frequencies

Create a CWT filter bank.

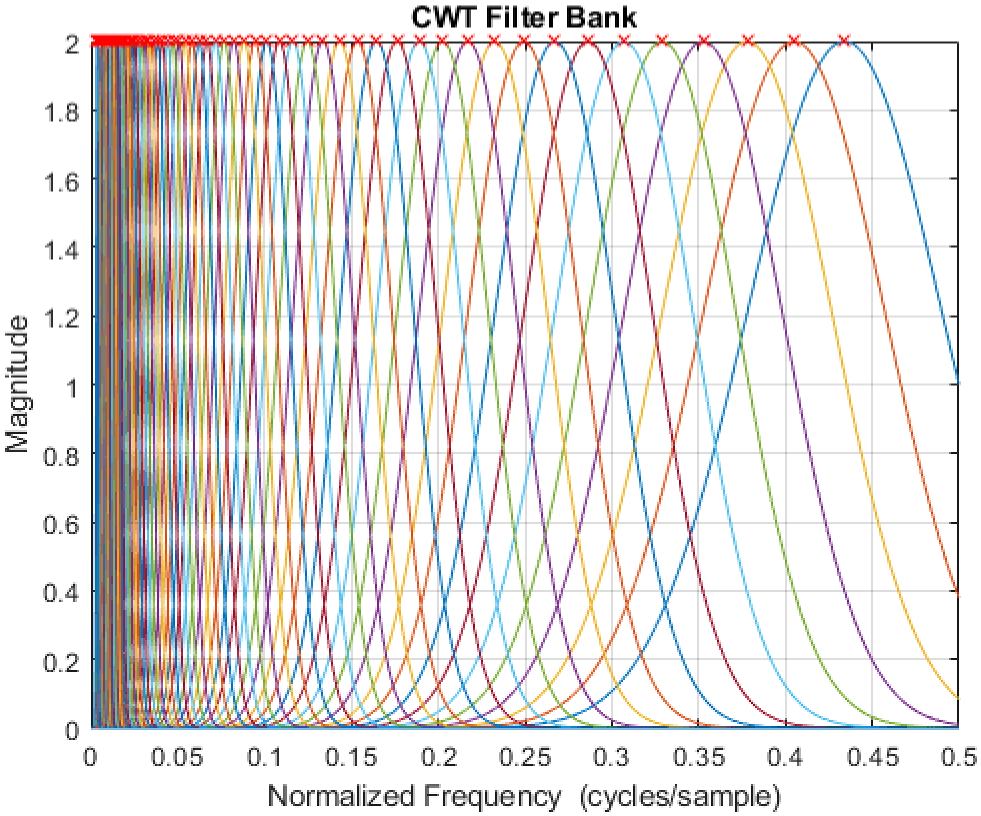
```
fb = cwtfilterbank;
```

Calculate the bandpass center frequencies.

```
bpcf = centerFrequencies(fb);
```

Plot the frequency responses of the filter bank and the bandpass center frequencies. The bandpass center frequencies correspond to the peaks of the frequency response of each wavelet in the filter bank.

```
freqz(fb)
hold on
plot(bpcf,2*ones(size(bpcf)),'rx')
```



Input Arguments

`fb` — Continuous wavelet transform filter bank
`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

`bpcf` — Wavelet bandpass center frequencies
real-valued vector

Wavelet bandpass center frequencies, returned as a real-valued vector of length *Ns*, where *Ns* is the number of scales in the filter bank. Frequencies are ordered from high to low. Frequencies are in cycles/sample if a sampling frequency or sampling period is not specified. If a sampling frequency is specified, `bpcf` has units of hertz. If a sampling period is specified, `bpcf` has units of cycles/unit time, where the time unit is the same as the duration `SamplingPeriod`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`cwtfilterbank` | `powerbw` | `freqz` | `centerPeriods`

Introduced in R2018b

centerFrequencies

Wavelet scattering bandpass center frequencies

Syntax

```
F = centerFrequencies(sf)
F = centerFrequencies(sf,filterbanks)
```

Description

`F = centerFrequencies(sf)` returns the wavelet bandpass center frequencies for all filter banks of the wavelet time scattering network, `sf`. The output `F` is a cell array with *Nfb* elements, where *Nfb* is the number of scattering filter banks. Each element of `F` is a real-valued vector. If you specify a sampling frequency in `sf`, `F` is in hertz. If you do not specify a sampling frequency, `F` is in cycles/sample.

If there is only one filter bank in the scattering network, `F` is a real-valued vector containing the wavelet bandpass center frequencies.

`F = centerFrequencies(sf,filterbanks)` returns the wavelet bandpass center frequencies for the specified `filterbanks`. The argument `filterbanks` is a scalar or vector with all the elements between 1 and *Nfb* inclusive, where *Nfb* is the number of scattering filter banks.

Examples

Wavelet Bandpass Center Frequencies

Create a wavelet time scattering network with a sampling frequency of 50 Hz.

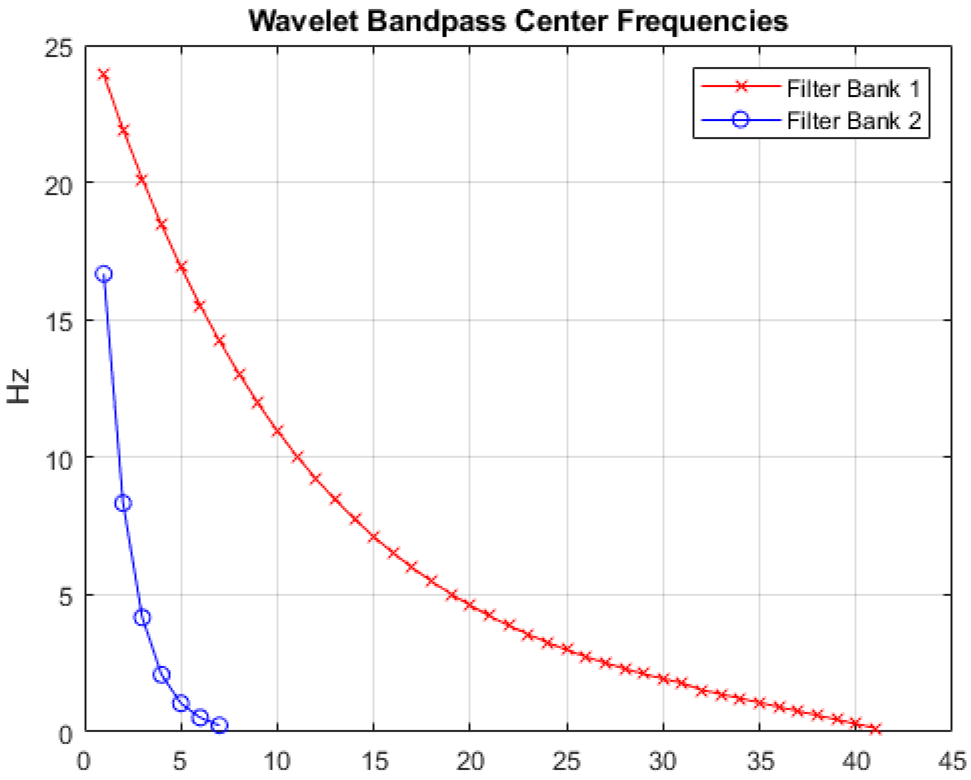
```
sf = waveletScattering('SamplingFrequency',50)
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 10.2400
        QualityFactors: [8 1]
              Boundary: 'periodic'
     SamplingFrequency: 50
             Precision: 'double'
    OversamplingFactor: 0
          OptimizePath: 0
```

Plot the wavelet bandpass center frequencies for all the filter banks.

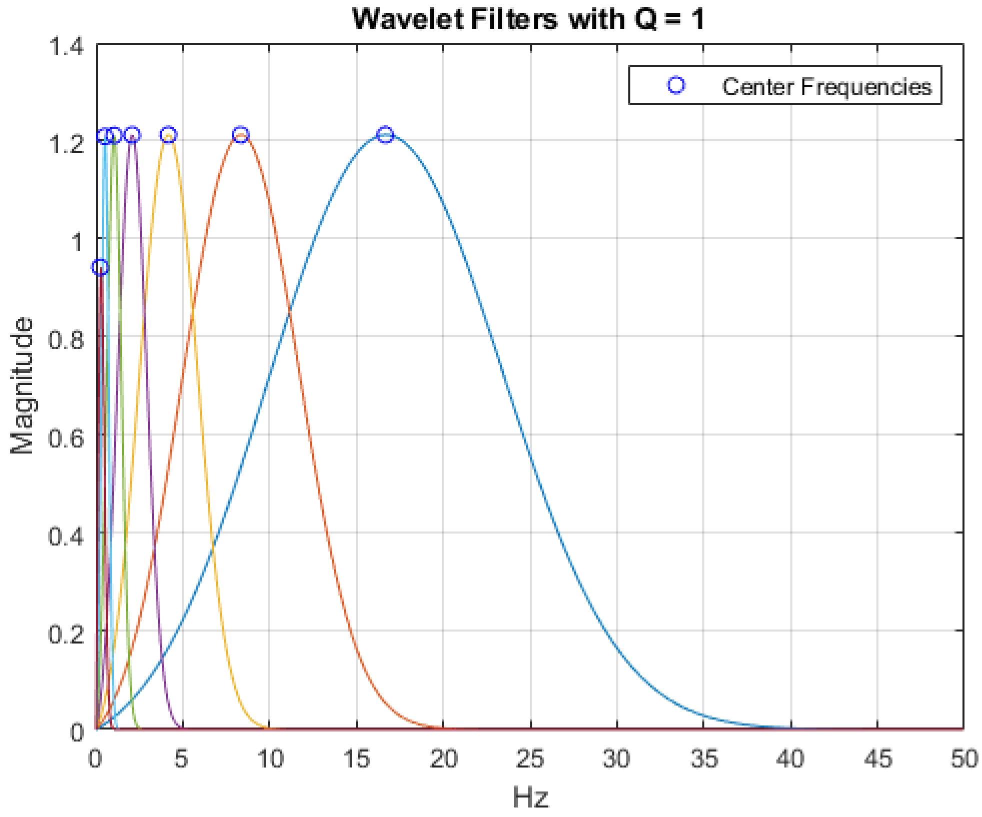
```
bpcf = centerFrequencies(sf);
plot(bpcf{1},'rx-')
hold on
plot(bpcf{2},'bo-')
```

```
grid on
title('Wavelet Bandpass Center Frequencies')
legend('Filter Bank 1','Filter Bank 2')
ylabel('Hz')
```



Plot the wavelets filters used in computing the second-order scattering coefficients.

```
orderCoef = 2;
[filters,f] = filterbank(sf);
figure
plot(f,filters{orderCoef+1}.psift)
grid on
title('Wavelet Filters with Q = 1')
xlabel('Hz')
ylabel('Magnitude')
hold on
pl = plot(bpcf{orderCoef},max(filters{orderCoef+1}.psift),'bo');
legend(pl,'Center Frequencies')
```



.

Input Arguments

`sf` — Wavelet time scattering network

`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

`filterbanks` — Filter bank indices

positive integer | vector of integers

Filter bank indices, specified as a positive integer or vector of integers. Elements of `filterbanks` are integers between 1 and *Nfb* inclusive, where *Nfb* is the number of scattering filter banks.

Example: `F = centerFrequencies(sf,[1 2])` returns the wavelet bandpass center frequencies for the first two filter banks in `sf`.

Data Types: `double`

Output Arguments

`F` — Wavelet bandpass center frequencies

vector | cell array

Wavelet bandpass center frequencies for filter banks of the scattering network `sf`, returned as a vector or cell array of vectors. If there is only one filter bank in `sf`, or if `filterbanks` is a scalar, then `F` is a real-valued vector. Otherwise, `F` is a cell array, where each element is a real-valued vector.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Generated code always returns a cell array, whereas MATLAB® returns a vector if `filterbanks` is a scalar, or `sf` has only one filter bank.

See Also

`waveletScattering`

Introduced in R2018b

centerPeriods

CWT filter bank bandpass center periods

Syntax

```
p = centerPeriods(fb)
```

Description

`p = centerPeriods(fb)` returns the wavelet bandpass center periods `p` for the continuous wavelet transform (CWT) filter bank `fb`.

Examples

Wavelet Filter Bank Bandpass Periods

Create two CWT filter banks. Set the sampling period of the first filter bank to 0.5 seconds, and the sampling frequency of the second filter bank to 2 Hz.

```
fb = cwtfilterbank('SamplingPeriod',seconds(0.5));
fb2 = cwtfilterbank('SamplingFrequency',2);
```

Obtain the bandpass center periods of both filter banks. Confirm the center periods of both filter banks are equal.

```
bp = centerPeriods(fb);
bp2 = centerPeriods(fb2);
bp(1:5)
```

```
ans = 5x1 duration
   1.1517 sec
   1.2344 sec
    1.323 sec
    1.418 sec
   1.5197 sec
```

```
bp2(1:5)
```

```
ans = 5×1

    1.1517
    1.2344
    1.3230
    1.4180
    1.5197
```

Obtain the bandpass center frequencies of the second filter bank. Confirm the reciprocals of the center frequencies are equal to the center periods.

```
f2 = centerFrequencies(fb2);
1./f2(1:5)

ans = 5×1

    1.1517
    1.2344
    1.3230
    1.4180
    1.5197
```

Input Arguments

fb — Continuous wavelet transform filter bank

cwtfilterbank object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

p — Wavelet bandpass center periods

real-valued vector | duration array

Wavelet bandpass center periods, returned as a real-valued vector of length *Ns*, where *Ns* is the number of scales in the filter bank.

If `SamplingPeriod` is specified, `p` is a duration array with the same units and format as `SamplingPeriod`. If `SamplingFrequency` is specified, `p` is in seconds.

See Also

`cwtfilterbank` | `powerbw` | `freqz` | `centerFrequencies`

Introduced in R2018b

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 specified by `wname` (see `wavefun` for more information).

`FREQ = centfrq(wname,ITER)` uses `ITER` many iterations to generate the wavelet.

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

Examples

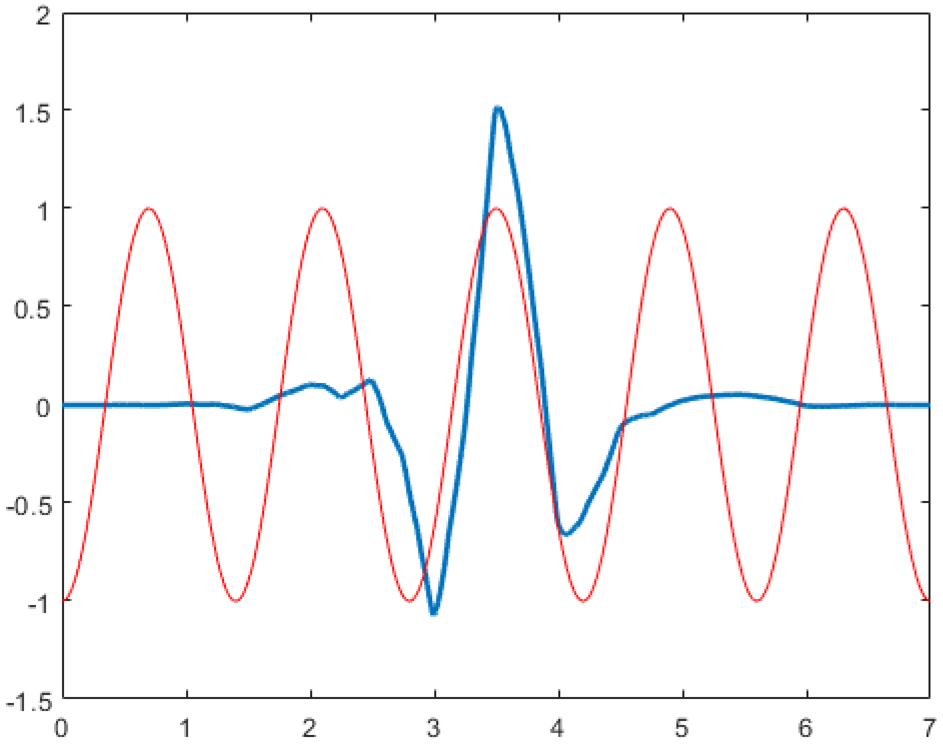
Determine Center Frequency

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.^(0:numvoices*numoctaves-1).*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.

Input Arguments

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. See `wavefun` for more information.

ITER — Number of iterations

8 (default) | positive integer

Number of iterations, specified by a positive integer, used to generate the wavelet `wname`. Internally, `centfrq` uses `wavefun` to generate the wavelet.

Output Arguments

FREQ — Wavelet center frequency

scalar

Wavelet center frequency in hertz, returned as a scalar.

XVAL — Grid points

real-valued vector

Grid points where the center frequency-based approximation to the wavelet is evaluated, returned as a real-valued vector.

RECFREQ — Center frequency-based approximation

vector

Center frequency-based approximation to the wavelet, returned as a vector. Depending on the wavelet, RECFREQ is either a real- or complex-valued vector.

See Also

`scal2frq`

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).

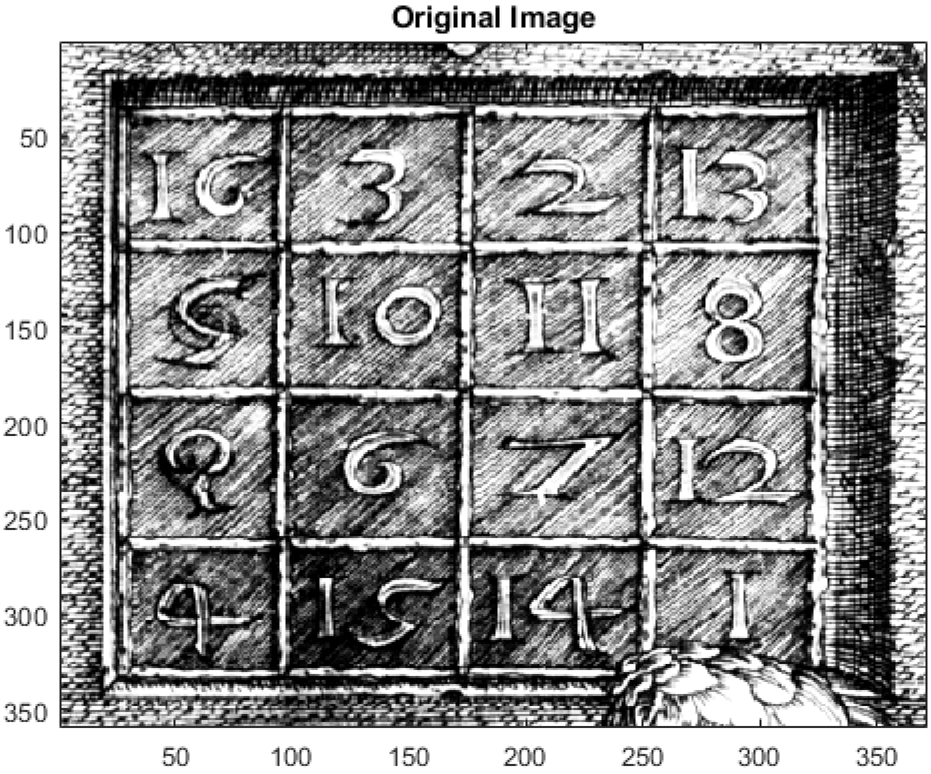
Examples

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.

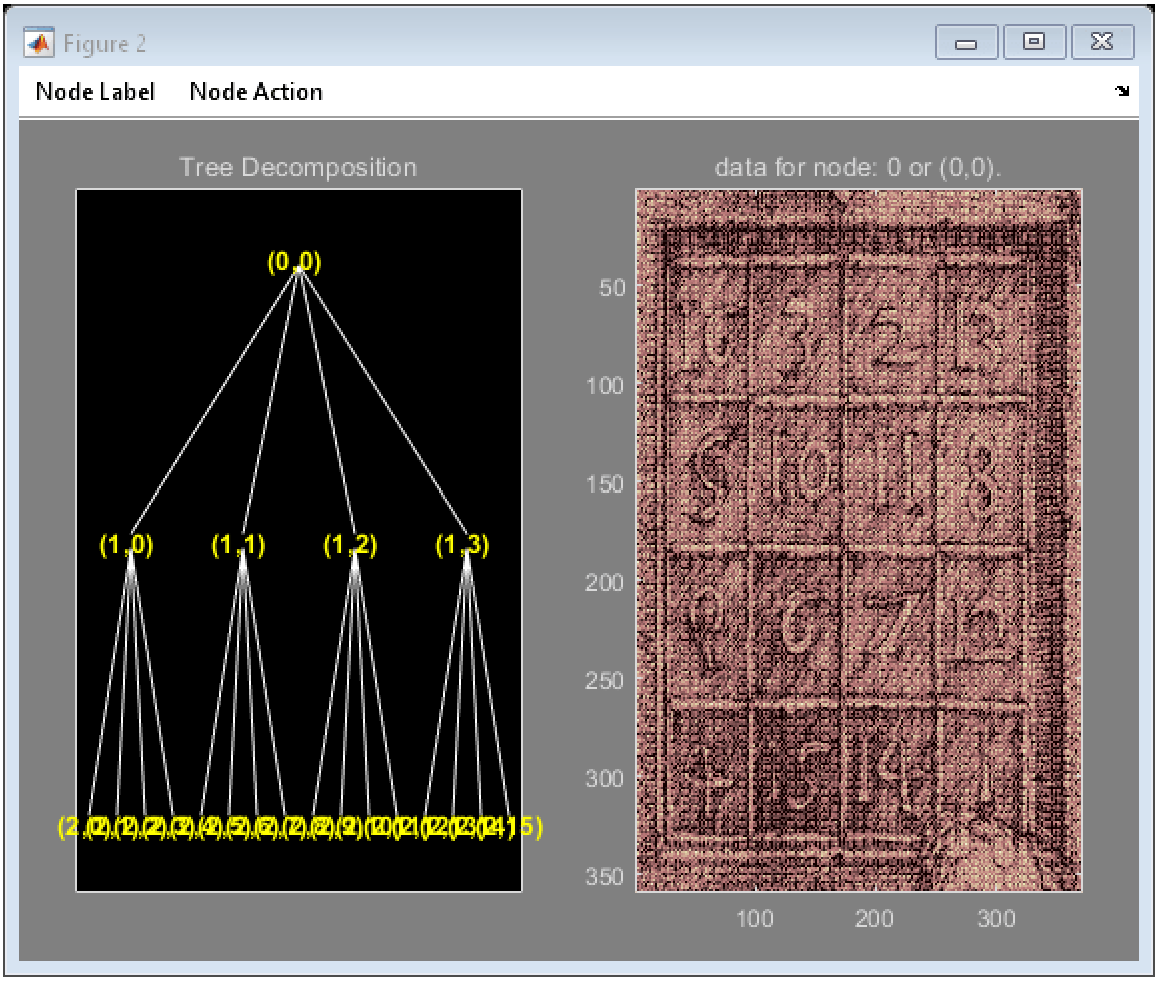
```
load detail;
imagesc(X); colormap gray; title('Original Image');
```



```
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.

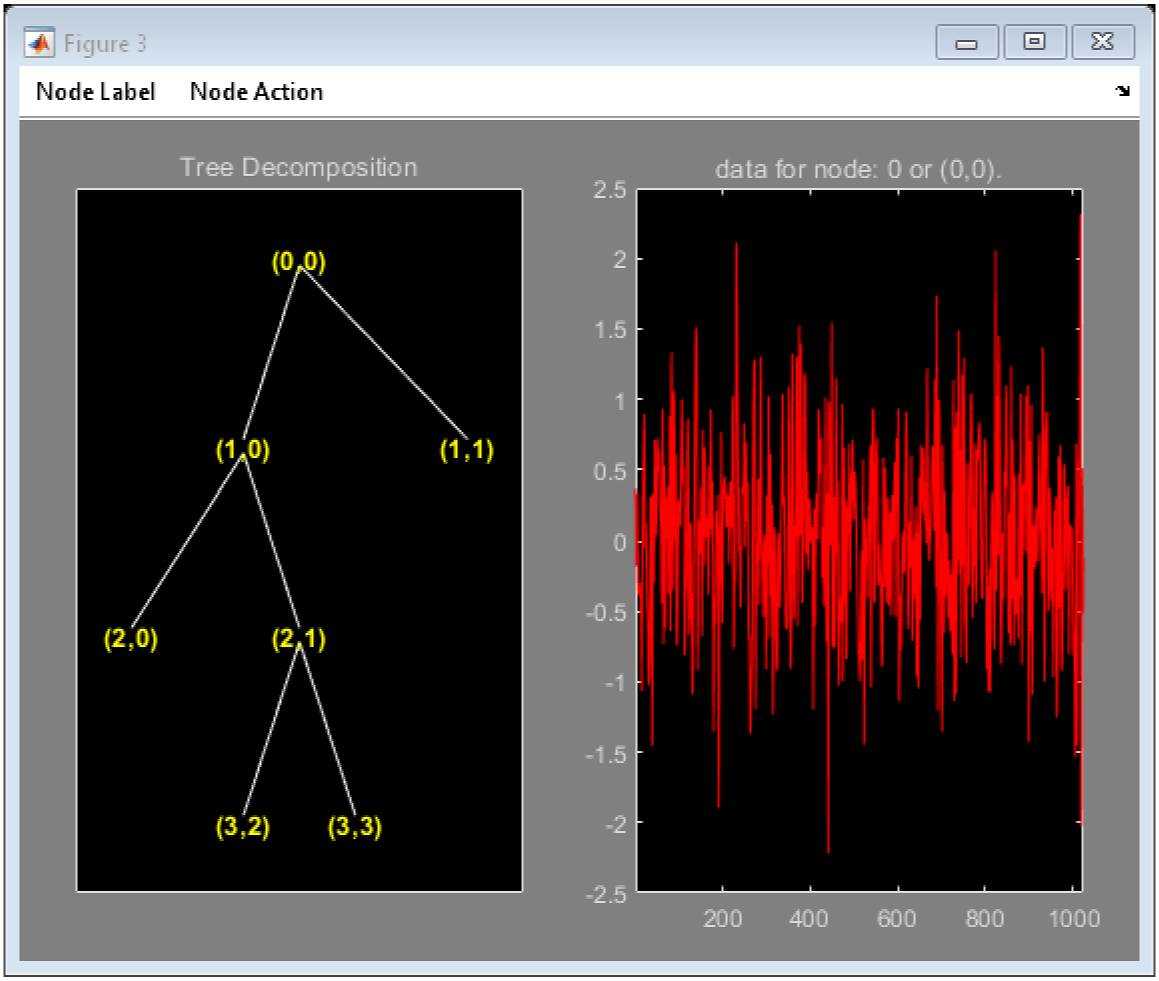
```
tr = cfs2wpt('sym4',[1 1024],[2 3 9 10]',2);
```

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] = cgauwavf(lb,ub,n,p)
[psi,x] = cgauwavf(lb,ub,n,wname)
```

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] = cgauwavf(lb,ub,n,p)` returns the `p`th derivative. `p` is an integer from 1 through 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.

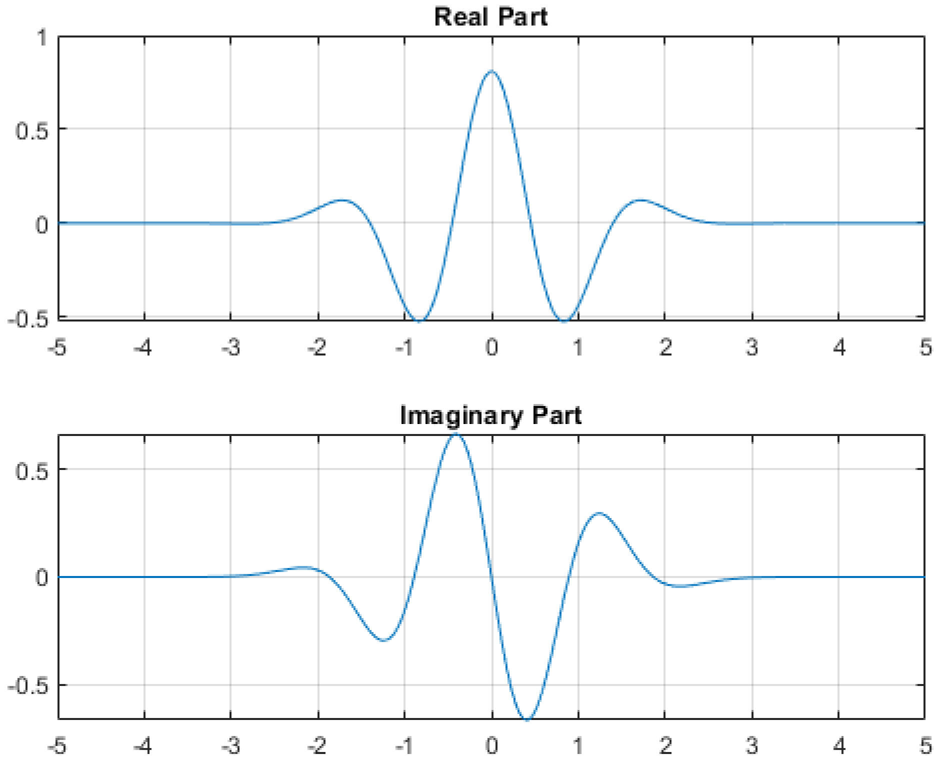
`[psi,x] = cgauwavf(lb,ub,n,wname)` used the valid wavelet family short name `wname` plus the order of the derivative in a character vector or string scalar, such as `'cgau4'`. To see valid character vectors for complex-valued Gaussian wavelets, use `waveinfo('cgau')` or use `wavemngr('read',1)` and 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(2,1,1)
plot(x,real(psi))
title('Real Part')
grid on
subplot(2,1,2)
plot(x,imag(psi))
title('Imaginary Part')
grid on
```



Input Arguments

lb — Left endpoint

real number

Left endpoint of the closed interval, specified as a real number. lb is strictly less than ub.

Data Types: double

ub — Right endpoint

real number

Right endpoint of the closed interval, specified as a real number. ub is strictly greater than lb.

Data Types: double

n — Number of regularly spaced points

positive integer

Number of regularly spaced points in the interval [lb,ub], specified as a positive integer. The derivative of the complex-valued Gaussian wavelet is evaluated at these points.

Data Types: double

p — Derivative

positive integer

Positive integer defining the order of the derivative of the complex-valued Gaussian, specified as a positive integer. `p` is an integer from 1 through 8.

`wname` — Gaussian wavelet
character vector | string scalar

Gaussian wavelet to evaluate, specified as a character vector or string scalar. `wname` is of the form `'cgauN'` where *N* is an integer that denotes the order of the derivative of the complex-valued Gaussian. *N* is an integer from 1 through 8.

Example: `'cgau4'` denotes the fourth derivative of the complex-valued Gaussian wavelet.

Output Arguments

`psi` — Derivative of complex-valued Gaussian wavelet
complex-valued vector

Derivative of the complex-valued Gaussian wavelet, returned as a complex-valued 1-by-`N` vector.

`x` — Sample points
real-valued vector

Sample points where the derivative of the complex-valued Gaussian wavelet is evaluated, returned as a real-valued 1-by-`N` vector. The sample points are evenly distributed between `lb` and `ub`.

See Also

`waveinfo` | `wavemngr`

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 ≤ LEV ≤ 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

Change Decomposition Coefficients

Load the 23 channel EEG data Espiga3 [1]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the db2 wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

Change the coefficients of details at level 1. Replace all the values by 0.

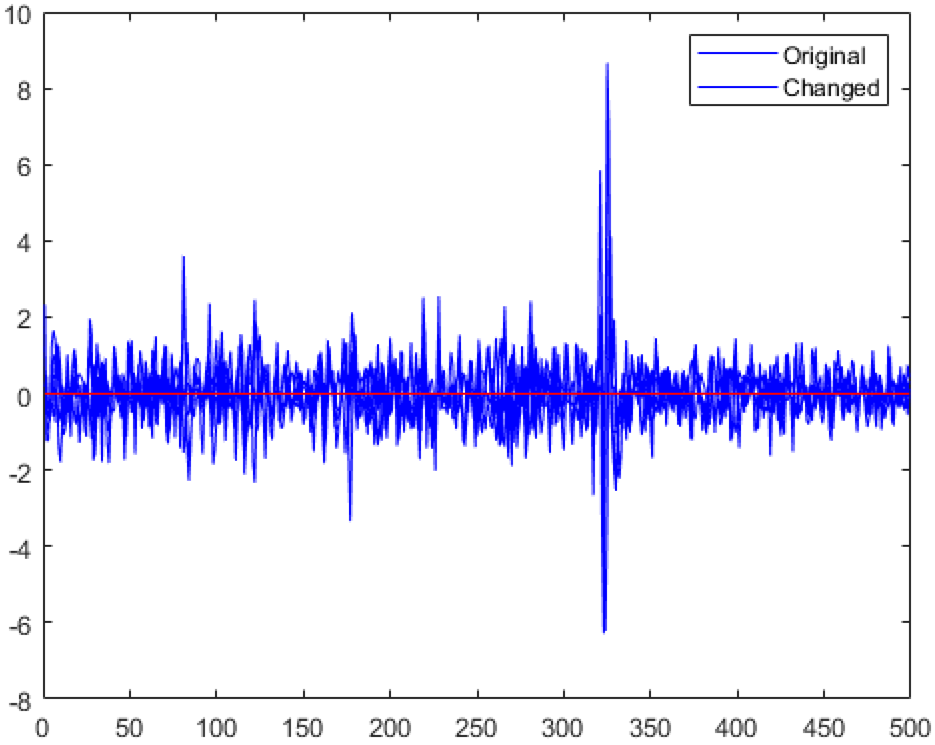
```
decBis = chgwdeccfs(dec,'cd',0,1);
```

Change the coefficients of details at level 1 and level 2 for signals 11 to 15. Replace all values by 0.

```
decTer = chgwdeccfs(dec,'cd',0,1:2,11:15);
```

Compare the original and new coefficients for details at level 1 for signals 11 to 15.

```
plot(dec.cd{1}(:,11:15),'b')
hold on
plot(decTer.cd{1}(:,11:15),'r')
legend('Original','Changed')
```



References

[1] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

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

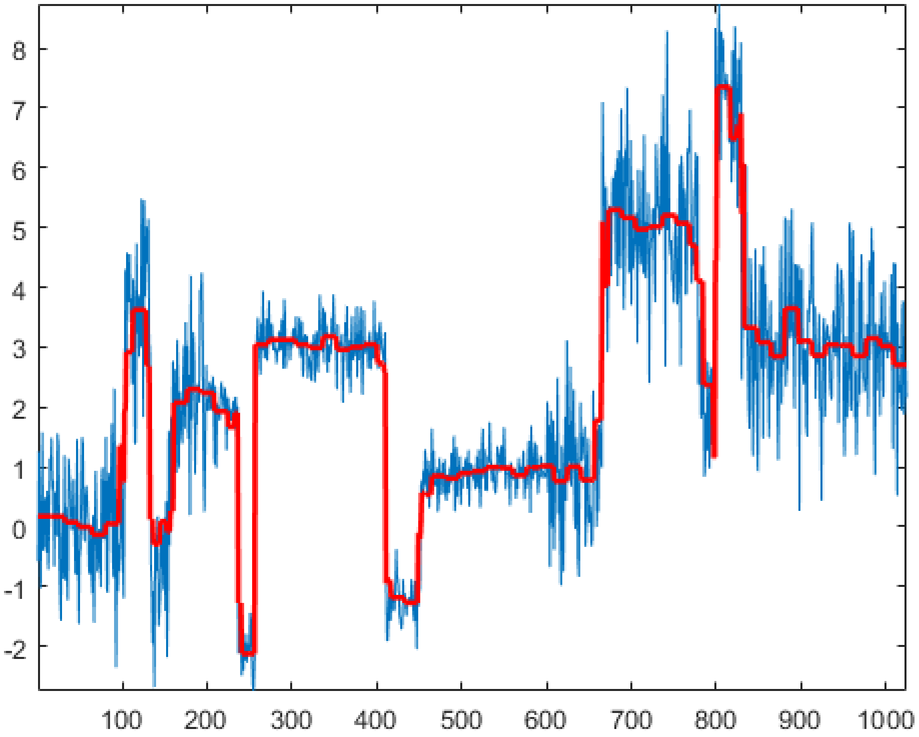
Denoising Blocks Signal with Haar Wavelet

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;
```



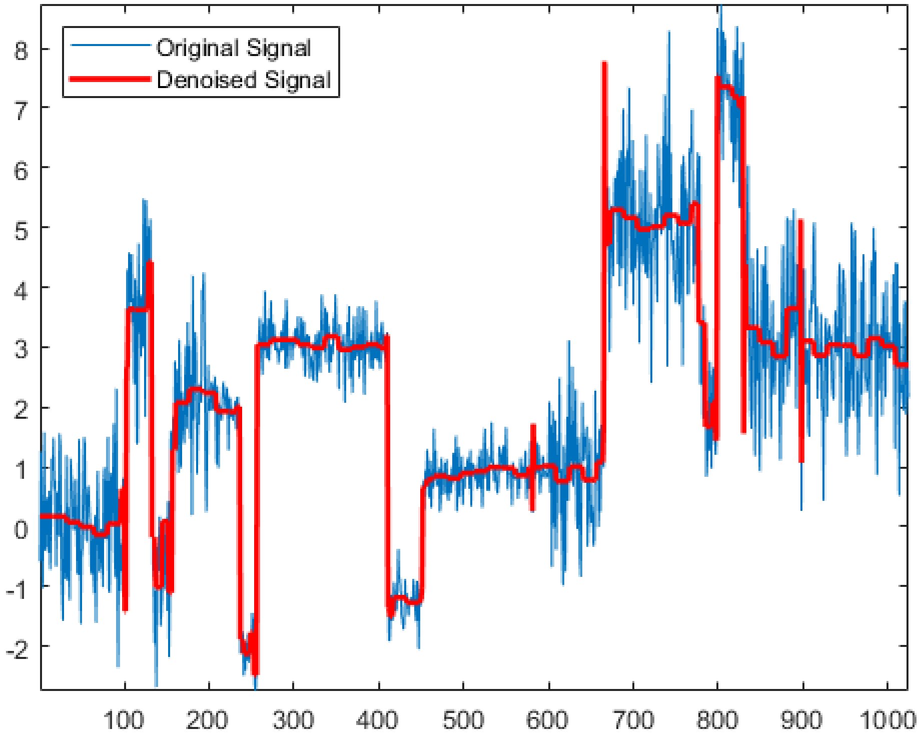
Denoising Blocks Signal with Hard Thresholding

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.

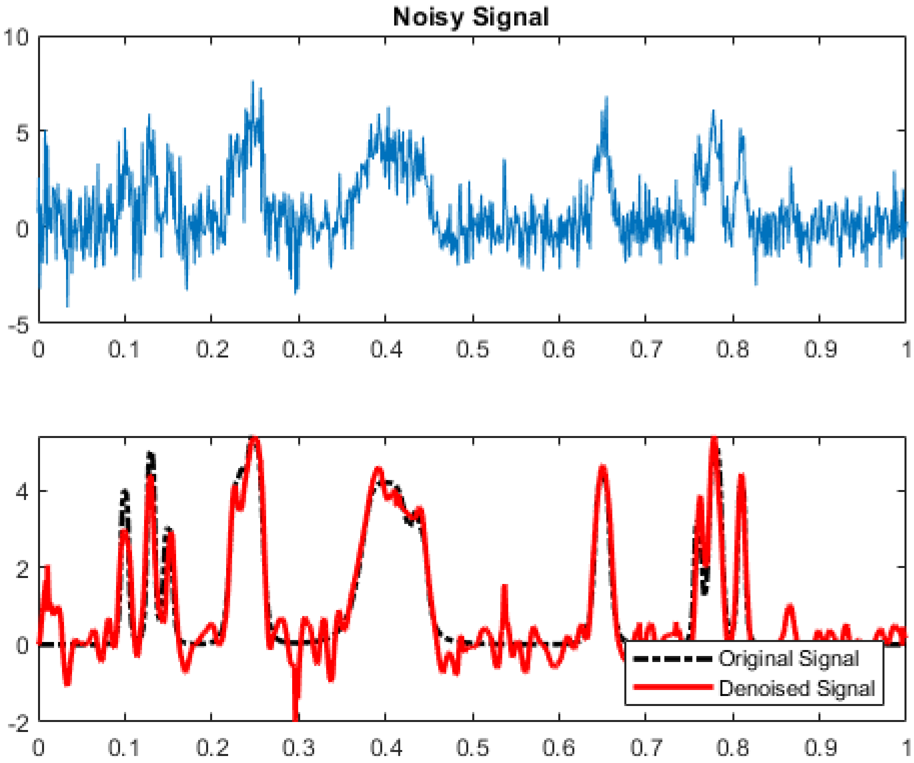
```
t = [0.1 0.13 0.15 0.23 0.25 0.40 0.44 0.65 0.76 0.78 0.81];
h = [4  -5 3 -4 5  -4.2   2.1   4.3  -3.1   5.1  -4.2];
h  = abs(h);
len = 1000;
w  = 0.01*[0.5 0.5 0.6 1 1 3 1 1 0.5 0.8 0.5];
tt = linspace(0,1,len);
x = zeros(1,len);
for j=1:11
  x = x + ( h(j) ./ (1+ ((tt-t(j))/w(j)).^4));
end
```

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.

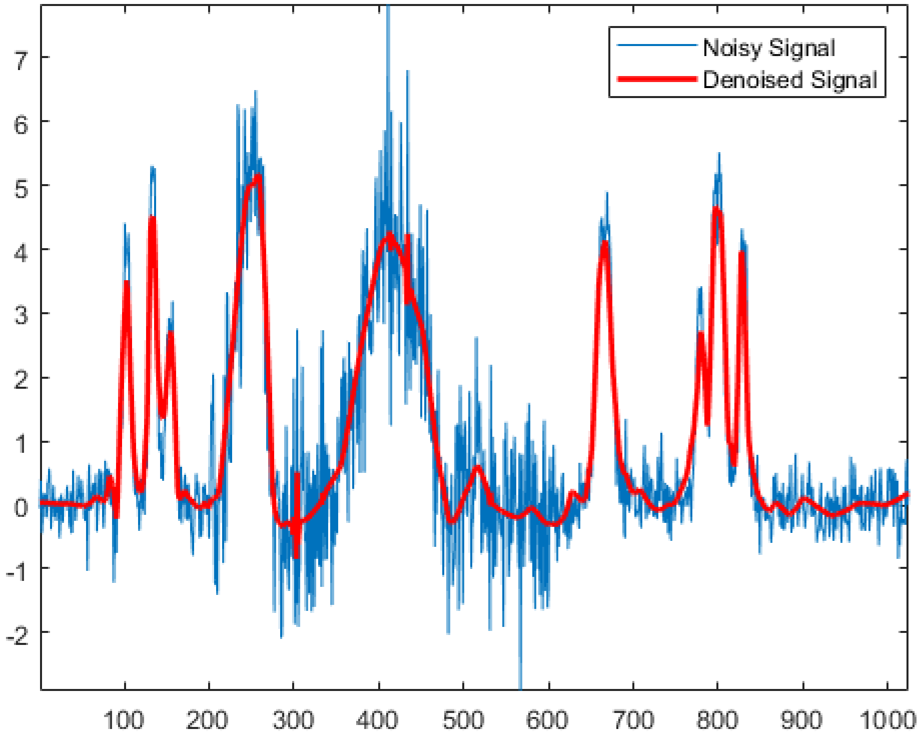
```
load nbumpr1.mat;
```

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');
```



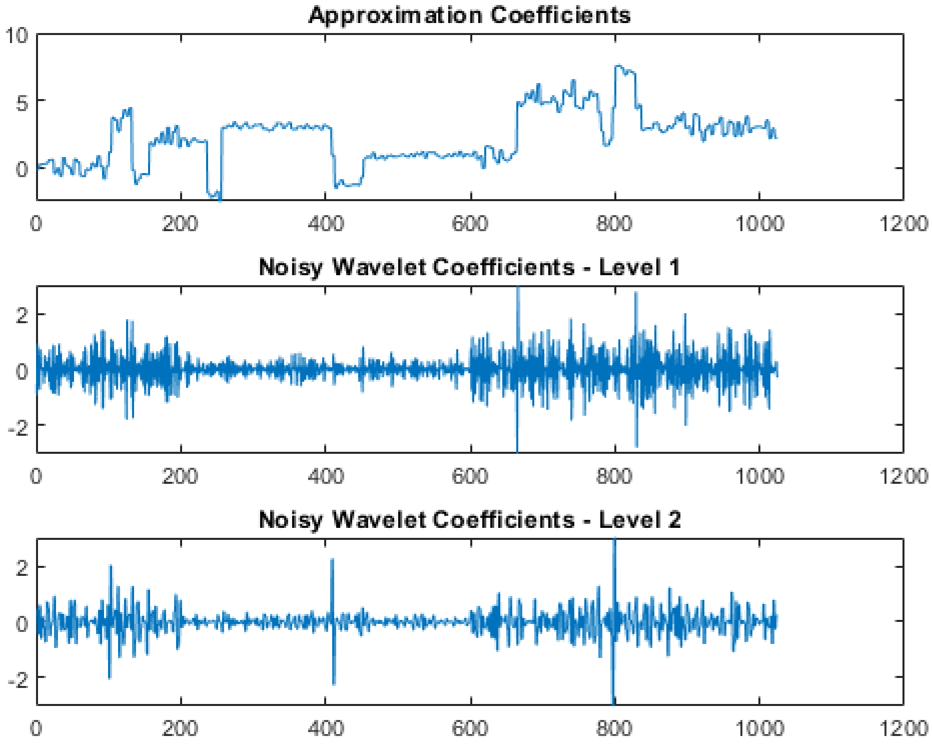
Return Denoised Wavelet Coefficients

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');
```

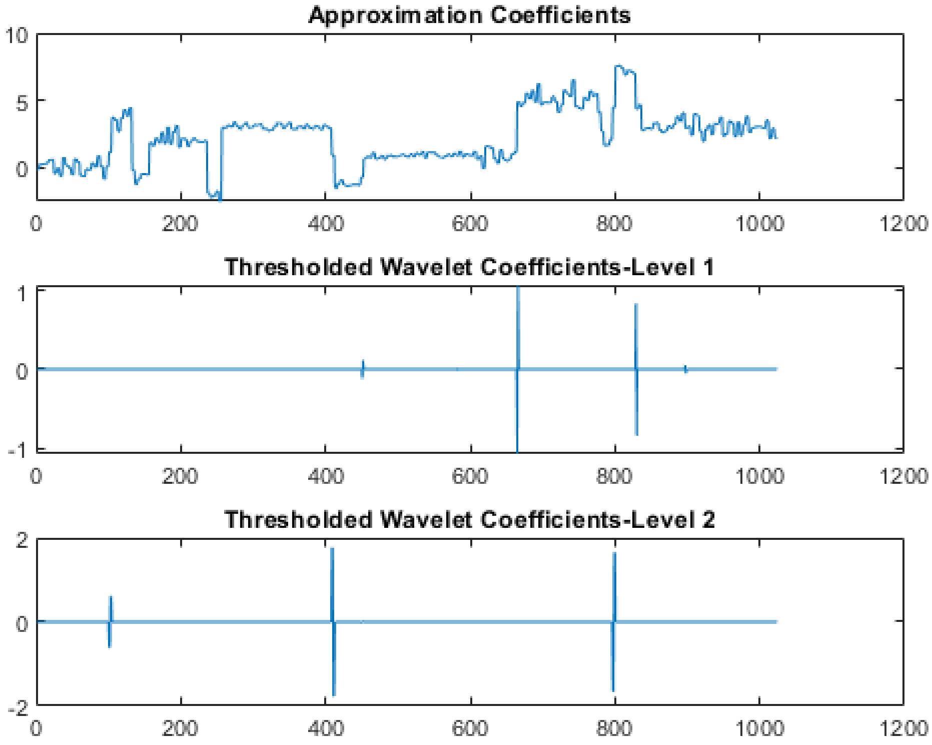
Plot reconstructions based on the level-2 approximation and level-2 and level-1 detail coefficients for the noisy signal.

```
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
```

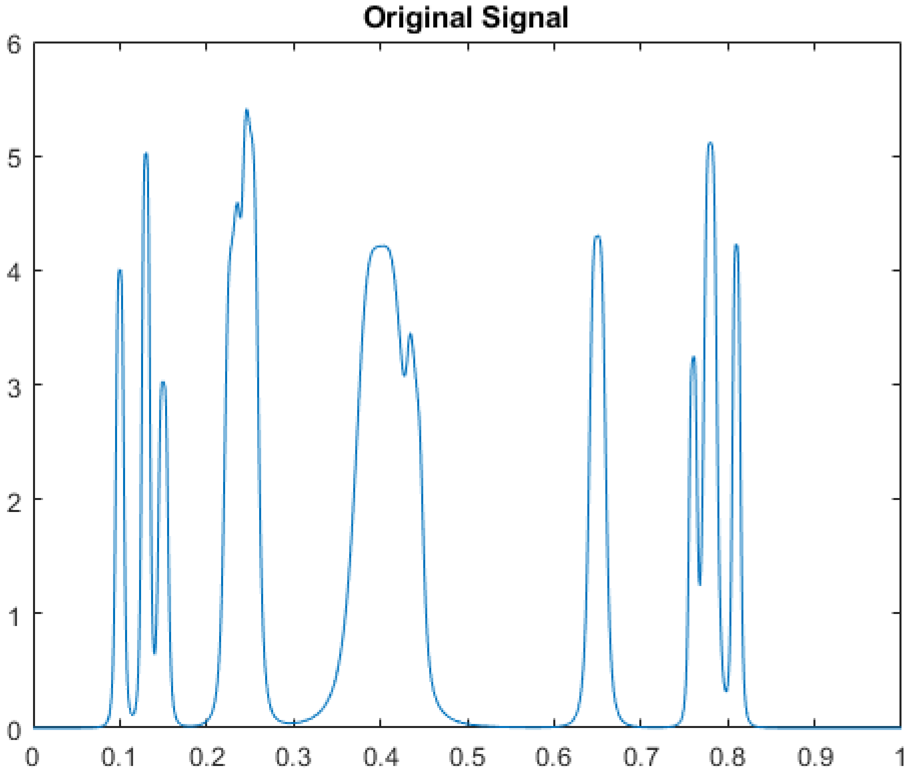


The approximation coefficients are identical in the noisy and denoised signal, but most of the detail coefficients in the denoised signal are close to zero.

Output Intervals and Thresholds

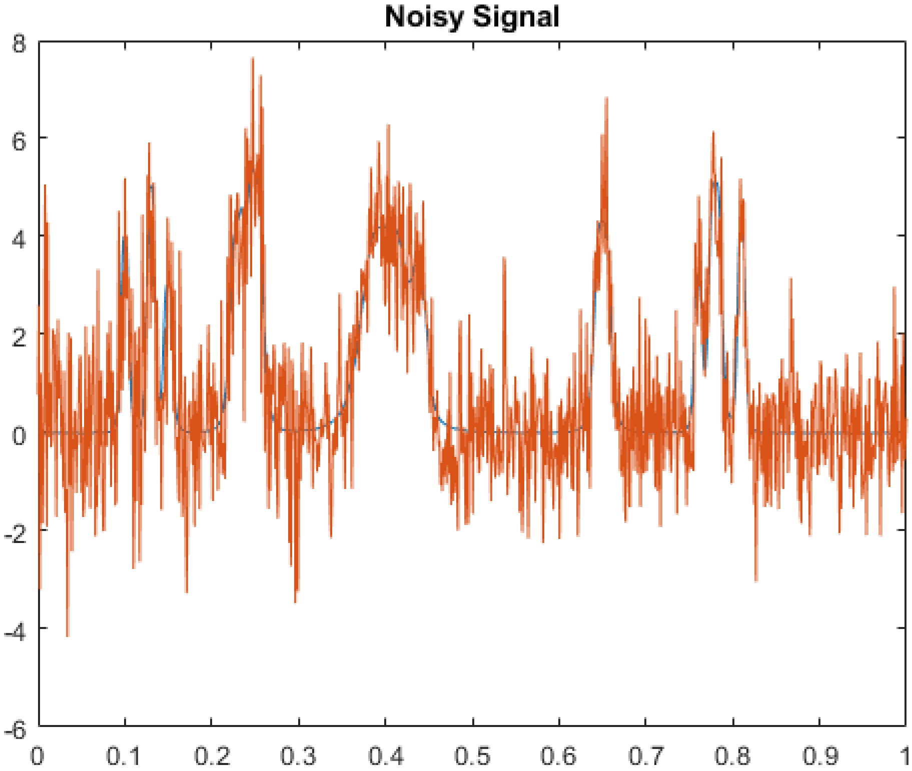
Create a signal sampled at 1 kHz. The signal consists of a series of bumps of various widths.

```
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 = 2×3
10^3 ×

    0.0010    0.2930    0.0036
    0.2930    1.0000    0.0028
```

`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 mlptdenoise for details.

Data Types: double

wname — Wavelet name

character vector | string scalar

Wavelet name, specified as a character vector or string scalar. 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

Functions
`thselect` | `wavedec` | `wthresh` | `wvarchg` | `wdenoise`

Apps
`Wavelet Signal Denoiser`

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 wavelet is evaluated on an `n`-point regular grid, `x`, for the interval [`lb`,`ub`]. The general expression for the complex Morlet wavelet is

$$\psi(x) = \frac{1}{\sqrt{\pi \cdot \mathrm{fb}}}\exp(2\pi i \cdot \mathrm{fc} \cdot x)\exp(-x^2/\mathrm{fb})\,.$$

`[psi,x] = cmorwavf(lb,ub,n,fb,fc)` returns the complex Morlet wavelet with time-decay parameter, `fb`, and center frequency, `fc`.

Examples

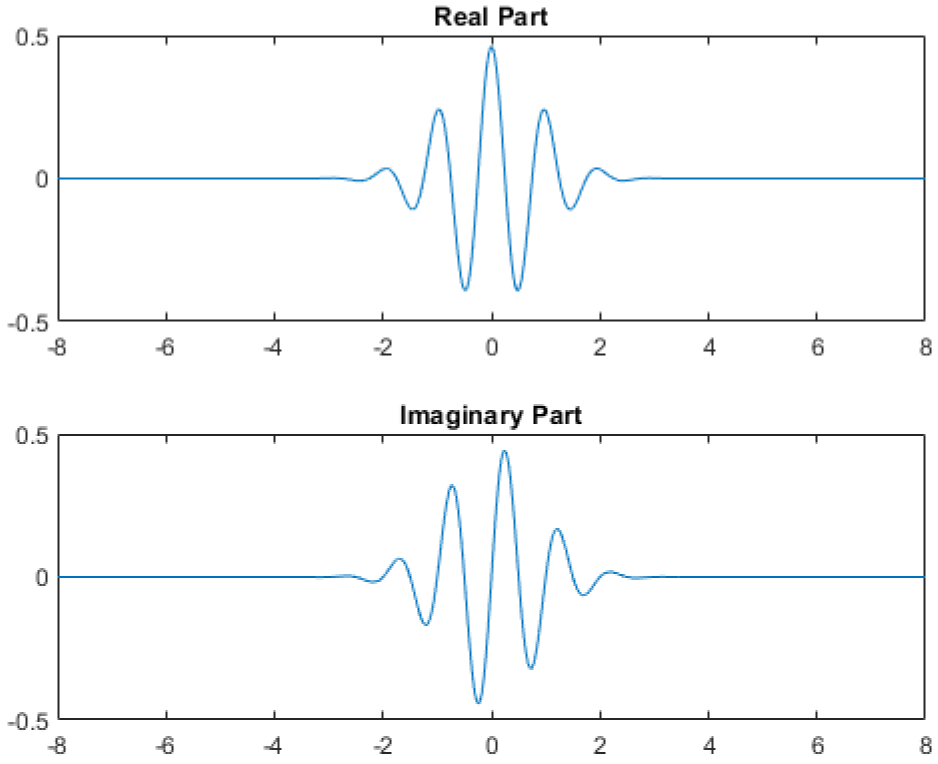
Complex Morlet Wavelet

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.

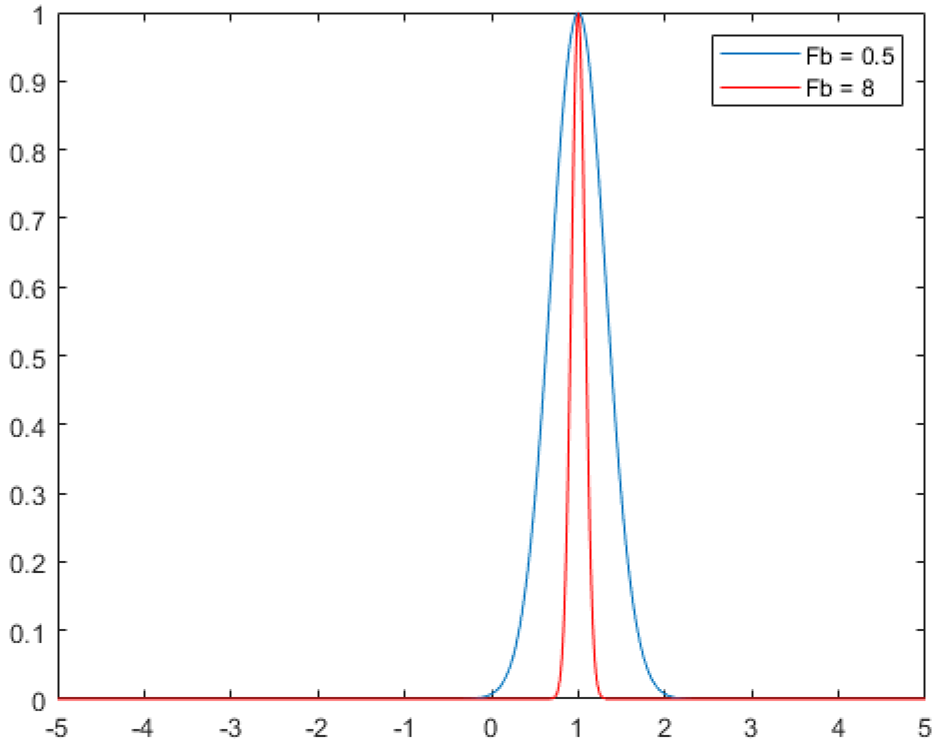
```
subplot(2,1,1)
plot(x,real(psi)); title('Real Part');
subplot(2,1,2)
plot(x,imag(psi)); title('Imaginary Part');
```



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.

Input Arguments

`lb` — Left endpoint
scalar

Left endpoint of the closed interval, specified as a scalar. `lb` is strictly less than `ub`.

`ub` — Right endpoint
scalar

Right endpoint of the closed interval, specified as a scalar. `ub` is strictly greater than `ub`.

`n` — Number of regularly spaced points
positive integer

Number of regularly spaced points in the interval `[lb,ub]`, specified as a positive integer.

`fb` — Time-decay parameter
positive scalar

Time-decay parameter, specified as a positive scalar. 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.

fc — Center frequency

positive scalar

Center frequency, specified as a positive scalar.

Output Arguments

psi — Complex Morlet wavelet

vector

Complex Morlet wavelet, returned as a complex-valued 1-by-n vector.

x — Sample points

vector

Sample points where the complex Morlet vector is evaluated, returned as a 1-by-n vector. The sample points are evenly distributed between lb and ub.

References

[1] Teolis, Anthony. *Computational Signal Processing with Wavelets*. Boston, MA: Birkhäuser Boston, 1998. https://doi.org/10.1007/978-1-4612-4142-3.

See Also

waveinfo

Introduced before R2006a

coefficientSize

Size of image scattering coefficients

Syntax

```
sz = coefficientSize(sf)
```

Description

`sz = coefficientSize(sf)` returns the scattering coefficient sizes for the wavelet image scattering network, `sf`. The output `sz` is a two-element row vector that gives the scattering coefficient output size in the row and column dimensions. For an RGB image, the actual output size is `[sz(1) sz(2) 3]`.

Examples

Scattering Coefficient Sizes for Image Scattering Network

This example shows how to determine the scattering coefficient sizes of an image scattering network.

Create a wavelet image scattering network with an image size of 128-by-64. Obtain the coefficient sizes of the network.

```
sf = waveletScattering2('ImageSize',[128 64]);
sz = coefficientSize(sf)
```

```
sz = 1×2

    16     8
```

Create a second wavelet image scattering network with an image size of 128-by-64 and an oversampling factor equal to 1. Obtain the coefficient sizes of the network. Since the oversampling factor is equal to 1, the scattering transform of the second network returns 2-by-2-by-*P* as many coefficients for each scattering path with respect to the critically sampled number.

```
sf2 = waveletScattering2('ImageSize',[128 64],'OversamplingFactor',1);
sz = coefficientSize(sf2)
```

```
sz = 1×2

    32    16
```

Input Arguments

`sf` — Wavelet image scattering network
`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

See Also

`waveletScattering2` | `paths`

Introduced in R2019a

coifwavf

Coiflet wavelet filter

Syntax

```
f = coifwavf(wname)
```

Description

`f = coifwavf(wname)` returns the scaling filter `f` associated with the Coiflet wavelet specified by `wname`. `f` is a real-valued vector.

Examples

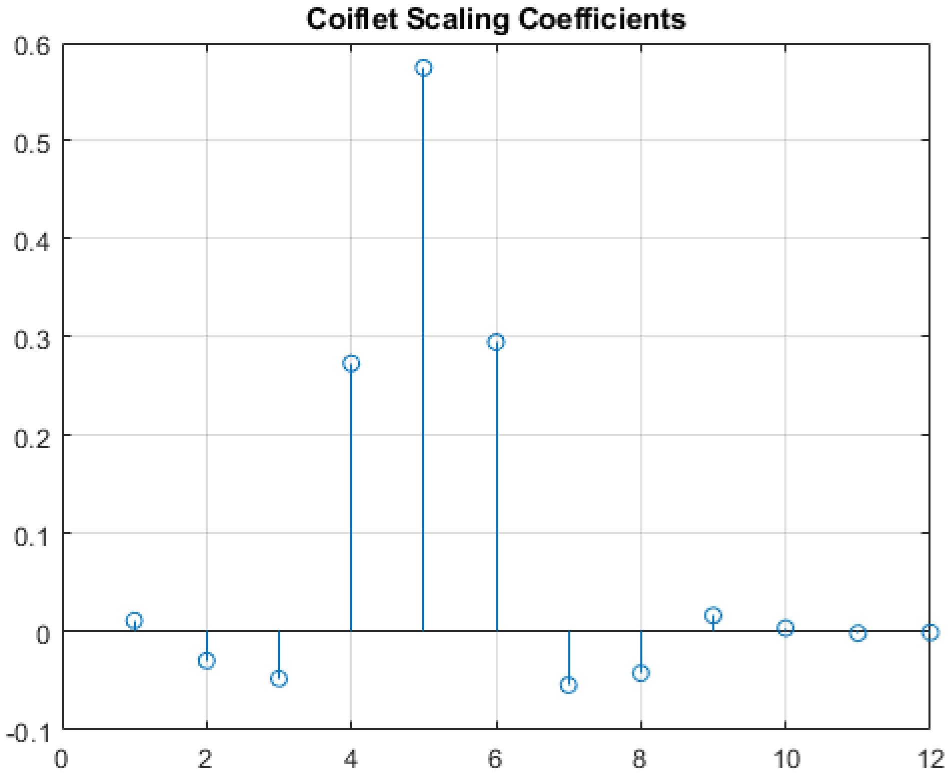
Coiflet Wavelet Filter

Set the Coiflet wavelet name.

```
wname = 'coif2';
```

Compute and plot the scaling filter coefficients associated with the Coiflet.

```
f = coifwavf(wname);
stem(f)
grid on
title('Coiflet Scaling Coefficients')
```



Input Arguments

`wname` — Name of Coiflet

character vector | string scalar

Name of Coiflet, specified as `'coifN'` where N is an integer between 1 and 5.

Example: `'coif3'`

See Also

`waveinfo`

Introduced before R2006a

concatenate

Concatenate two or more labeled signal sets

Syntax

```
lssnew = concatenate(lss1,...,lssN)
```

Description

`lssnew = concatenate(lss1,...,lssN)` concatenates *N* labeled signal set objects, `lss1,...,lssN`, and returns a labeled signal set `lssnew` containing all the members and label values of the input sets.

Examples

Concatenate Labeled Signal Sets

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Create a new signal set with the same data source, time information, and labels as `lss`.

```
newlss = copy(lss)
```

```
newlss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Concatenate the two signal sets.

```
lssconcat = concatenate(lss,newlss)
```

```
lssconcat = 
  labeledSignalSet with properties:

             Source: {4x1 cell}
         NumMembers: 4
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [4x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Input Arguments

`lss1,...,lssN` — Input labeled signal sets

`labeledSignalSet` objects

Input labeled signal sets, specified as `labeledSignalSet` objects. All input sets must have the same time information settings, label definitions, and data source type.

Output Arguments

`lssnew` — Concatenated labeled signal set

`labeledSignalSet` object

Concatenated labeled signal set, returned as a `labeledSignalSet` object. The set `lssnew` contains a signal source, label definitions, and label values that are independent of those in the input labeled signal sets. Changing any of the input labeled signal sets does not affect the concatenated labeled signal set. Changing the concatenated labeled signal set does not affect the input label signal sets.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

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 or string scalar 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.

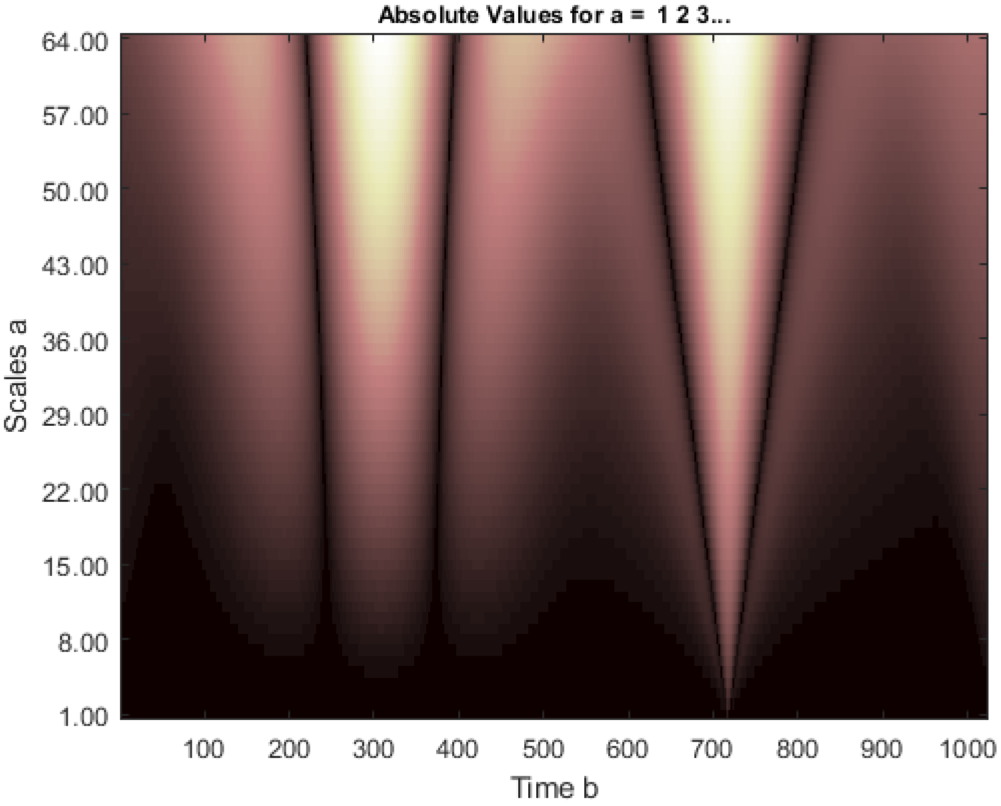
```
load cuspamax
signal = cuspamax;
```

Set up the wavelet.

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

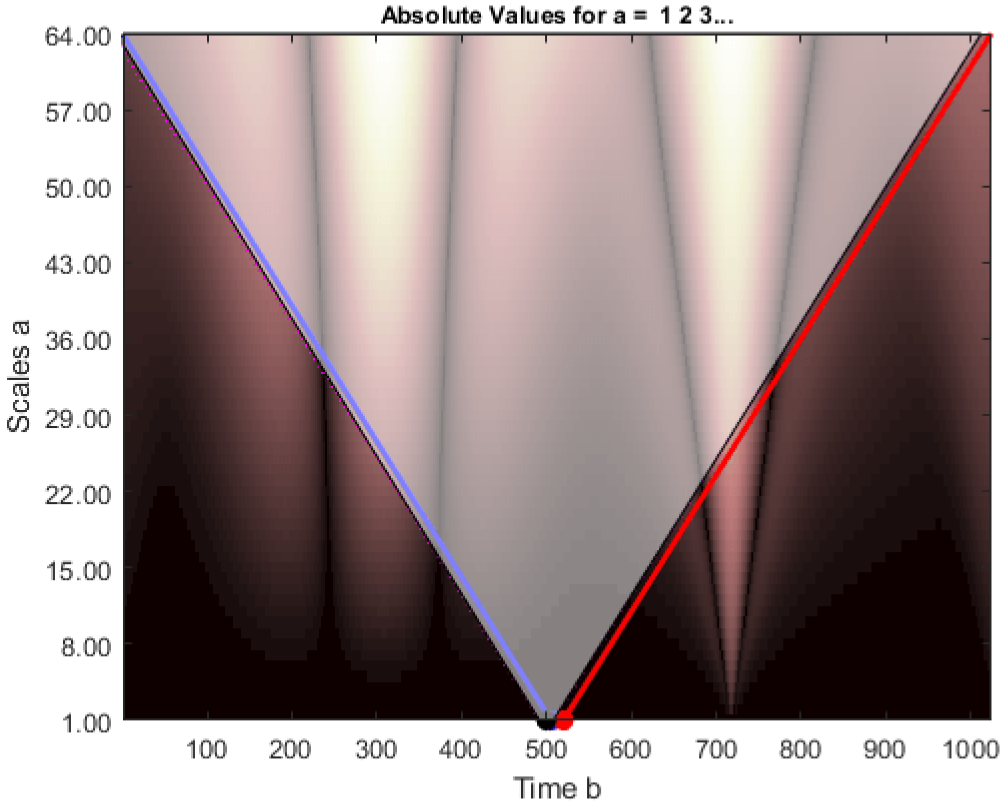
Plot the wavelet.

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



Plot the cone of influence.

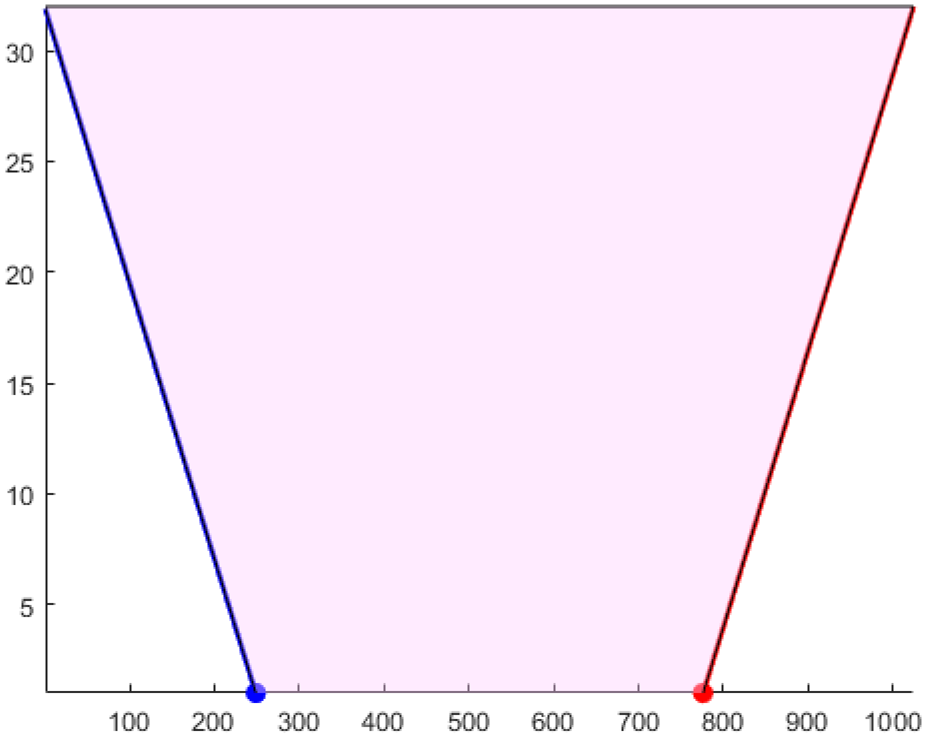
```
hold on
[cone,PL,PR,Pmin,Pmax] = conofinf(wname,scales,lenSIG,x,'plot');
set(gca,'Xlim',[1 lenSIG])
```



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 = 1×2

   -0.1245   32.0000
```

```
PRmax
```

```
PRmax = 1×2

    0.1250  -96.0000
```

More About

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

countLabelValues

Count label values

Syntax

```
cnt = countLabelValues(lss,lblname)
```

Description

`cnt = countLabelValues(lss,lblname)` counts the values of the label named `lblname` and returns results in table `cnt`. `cnt` contains label value counts and percentages. When `lblname` is an ROI or point label, `cnt` also contains the number of members with at least one value of a particular category. `countLabelValues` does not support:

- Sublabels
- Label definitions with the LabelDataType property set to `'table'` or `'timetable'`
- Labels with instance values that cannot be converted to a vector with a discrete set of categories. It must be possible to group label values using a set of unique discrete categories. Examples of labels that are not supported include:
  - Cell arrays of timetables
  - Cell arrays containing matrices of different sizes

Examples

Count Label Values

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get the names of the labels in the set.

```
getLabelNames(lss)
```

```
ans = 3x1 string
    "WhaleType"
    "MoanRegions"
    "TrillRegions"
```

Verify that the two members of the set are blue whales.

```
countLabelValues(lss,"WhaleType")
```

```
ans=3×3 table
    WhaleType    Count    Percent
    _________    _____    _______

    blue           2        100  
    humpback       0          0  
    white          0          0  
```

Verify that each member has three moan regions.

```
countLabelValues(lss,"MoanRegions")
```

```
ans=2×4 table
    MoanRegions    Count    Percent    MemberCount
    ___________    _____    _______    ___________

       false         0          0           0     
       true          6        100           2     
```

Verify that each member has one trill region.

```
countLabelValues(lss,"TrillRegions")
```

```
ans=2×4 table
    TrillRegions    Count    Percent    MemberCount
    ____________    _____    _______    ___________

       false          0          0           0     
       true           2        100           2     
```

Count Label Values and Create Datastores

Specify the path to a set of audio signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'
```

```
nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to `Fs`, which is common to all files. Generate a subset of the datastore that excludes the file `mtlb.mat`. Use the subset datastore as the source for a `labeledSignalSet` object.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sds = subset(sds,~strcmp(nms,"mtlb.mat"));
lss = labeledSignalSet(sds);
```

Create three label definitions to label the signals:

- Define a logical attribute label that is true for signals that contain human voices.
- Define a numeric point label that marks the location and amplitude of the maximum of each signal.
- Define a categorical region-of-interest (ROI) label to pick out nonoverlapping, uniform-length random regions of each signal.

Add the signal label definitions to the labeled signal set.

```
vc = signalLabelDefinition("Voice",'LabelType','attribute', ...
    'LabelDataType','logical','DefaultValue',false);
mx = signalLabelDefinition("Maximum",'LabelType','point', ...
    'LabelDataType','numeric');
rs = signalLabelDefinition("RanROI",'LabelType','ROI', ...
    'LabelDataType','categorical','Categories',["ROI" "other"]);
addLabelDefinitions(lss,[vc mx rs])
```

Label the signals:

- Label `'handel.mat'` and `'laughter.mat'` as having human voices.
- Use the `islocalmax` function to find the maximum of each signal. Label its location and value.
- Use the `randROI` on page 1-0 function to generate as many regions of length *N*/10 samples as can fit in a signal of length *N* given a minimum separation of *N*/6 samples between regions. Label their locations and assign them to the `ROI` category.

When labeling points and regions, convert sample values to time values. Subtract 1 to account for MATLAB® array indexing and divide by the sample rate.

```
kj = 1;
while hasdata(sds)

    [sig,info] = read(sds);
    fs = info.SampleRate;

    [~,fn] = fileparts(info.FileName);
    if fn=="handel" || fn=="laughter"
        setLabelValue(lss,kj,"Voice",true)
    end

    xm = find(islocalmax(sig,'MaxNumExtrema',1));
    setLabelValue(lss,kj,"Maximum",(xm-1)/fs,sig(xm))
    N = length(sig);
    rois = randROI(N,round(N/10),round(N/6));
    setLabelValue(lss,kj,"RanROI",(rois-1)/fs,repelem("ROI",size(rois,1)))

    kj = kj+1;

end
```

Verify that only two signals contain voices.

```
countLabelValues(lss,"Voice")
```

```
ans=2×3 table
    Voice    Count    Percent
    _____    _____    _______

    false      4      66.667 
    true       2      33.333 
```

Verify that two signals have a maximum amplitude of 1.

```
countLabelValues(lss,"Maximum")
```

```
ans=5×4 table
           Maximum             Count    Percent    MemberCount
    ______________________    _____    _______    ___________

    0.80000000000000004441      1      16.667          1     
    0.89113331915798421612      1      16.667          1     
    0.94730769230769229505      1      16.667          1     
    1                           2      33.333          2     
    1.0575668990330560071       1      16.667          1     
```

Verify that each signal has four nonoverlapping random regions of interest.

```
countLabelValues(lss,"RanROI")
```

```
ans=2×4 table
    RanROI    Count    Percent    MemberCount
    ______    _____    _______    ___________

    ROI        24        100           6     
    other       0          0           0     
```

Create two datastores with the data in the labeled signal set:

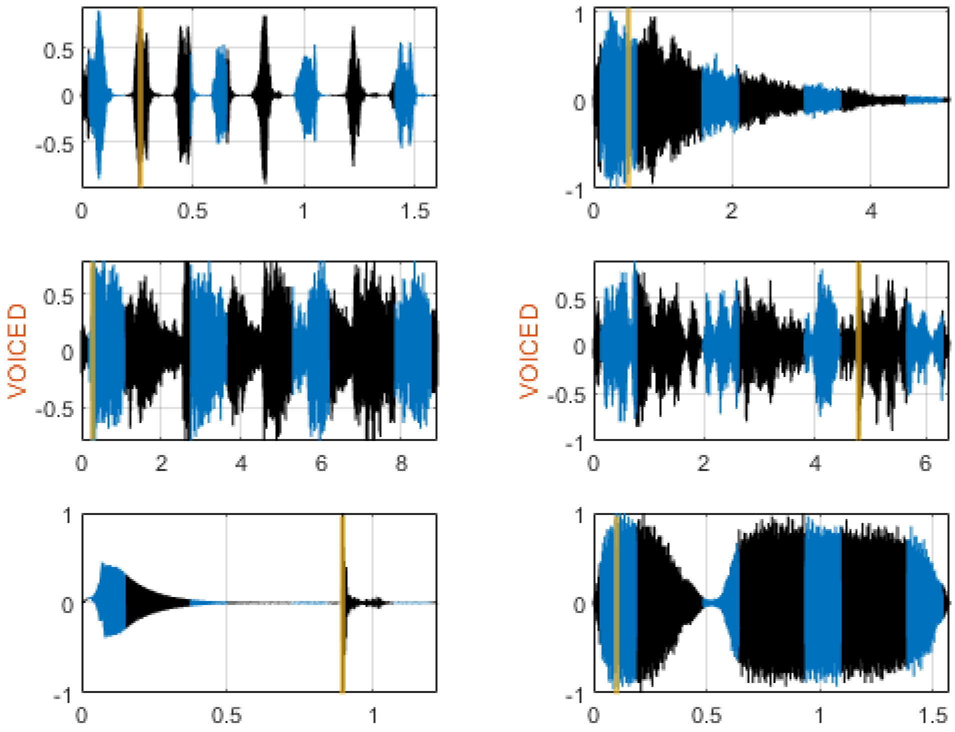
- The `signalDatastore` (Signal Processing Toolbox) object `sd` contains the signal data.
- The `arrayDatastore` object `ld` contains the labeling information. Specify that you want to include the information corresponding to all the labels you created.

```
[sd,ld] = createDatastores(lss,["Voice" "RanROI" "Maximum"]);
```

Use the information in the datastores to plot the signals and display their labels.

- Use a `signalMask` (Signal Processing Toolbox) object to highlight the regions of interest in blue.
- Plot yellow lines to mark the locations of the maxima.
- Add a red axis label to the signals that contain human voices.

```
tiledlayout flow

while hasdata(sd)

    [sg,nf] = read(sd);

    lbls = read(ld);

    nexttile

    msk = signalMask(lbls{:}.RanROI{:},'SampleRate',nf.SampleRate);
    plotsigroi(msk,sg)
    colorbar off
    xlabel('')

    xline(lbls{:}.Maximum{:}.Location, ...
        'LineWidth',2,'Color','#EDB120')

    if lbls{:}.Voice{:}
        ylabel('VOICED','Color','#D95319')
    end

end
```



```
function roilims = randROI(N,wid,sep)

num = floor((N+sep)/(wid+sep));
hq = histcounts(randi(num+1,1,N-num*wid-(num-1)*sep),(1:num+2)-1/2);
roilims = (1 + (0:num-1)*(wid+sep) + cumsum(hq(1:num)))' + [0 wid-1];

end
```

Input Arguments

lss — Labeled signal set

labeledSignalSet object

Labeled signal set, specified as a labeledSignalSet object.

Example: labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female')) specifies a two-member set of random signals containing the attribute 'female'.

lblname — Label name

character vector | string scalar

Label name, specified as a character vector or string scalar.

Data Types: char | string

Output Arguments

cnt — Results table

table

Results table, returned as a table with the following variables:

- Count — Number of label values for a particular category.
- Percent — Number of label values for a particular category as a percentage of all label values.
- MemberCount — Number of members with at least one value of a particular category. This variable is returned only for an ROI or a point label.

See Also

labeledSignalSet | signalLabelDefinition

Introduced in R2021a

cqt

Constant-Q nonstationary Gabor transform

Syntax

```
cfs = cqt(x)
[cfs,f] = cqt(x)
[cfs,f,g,fshifts] = cqt(x)
[cfs,f,g,fshifts,fintervals] = cqt(x)
[cfs,f,g,fshifts,fintervals,bw] = cqt(x)
[ ___ ] = cqt( ___ ,Name,Value)
cqt( ___ )
```

Description

`cfs = cqt(x)` returns the constant-Q transform (CQT), `cfs`, of the input signal `x`. The input signal must have at least four samples.

- If `x` is a vector, then `cqt` returns a matrix corresponding to the CQT.
- If `x` is a matrix, then `cqt` obtains the CQT for each column (independent channel) of `x`. The function returns a multidimensional array corresponding to the maximally redundant version of the CQT.

`[cfs,f] = cqt(x)` returns the approximate bandpass center frequencies, `f`, corresponding to the rows of `cfs`. The frequencies are ordered from 0 to 1 and are in cycles/sample.

`[cfs,f,g,fshifts] = cqt(x)` returns the Gabor frames, `g`, used in the analysis of `x` and the frequency shifts, `fshifts`, in discrete Fourier transform (DFT) bins between the passbands in the rows of `cfs`.

`cfs`, `g`, and `fshifts` are required inputs for the inversion of the CQT with `icqt`.

`[cfs,f,g,fshifts,fintervals] = cqt(x)` returns the frequency intervals, `fintervals`, corresponding the rows of `cfs`. The `k`th element of `fshifts` is the frequency shift in DFT bins between the `((k-1) mod N)` and `(k mod N)` element of `fintervals` with `k = 0,1,2,...,N-1` where `N` is the number of frequency shifts. Because MATLAB indexes from 1, `fshifts(1)` contains the frequency shift between `fintervals{end}` and `fintervals{1}`, `fshifts(2)` contains the frequency shift between `fintervals{1}` and `fintervals{2}`, and so on.

`[cfs,f,g,fshifts,fintervals,bw] = cqt(x)` returns the bandwidth, `bw`, in DFT bins of the frequency intervals, `fintervals`.

`[ ___ ] = cqt( ___ ,Name,Value)` returns the CQT with additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

`cqt( ___ )` with no output arguments plots the CQT in the current figure. Plotting is supported for vector inputs only. If the input signal is real and `Fs` is the sampling frequency, the CQT is plotted over the range `[0,Fs/2]`. If the signal is complex, the CQT is plotted over the range `[0,Fs)`.

Note In order to visualize a sparse CQT, coefficients have to be interpolated. When interpolation occurs, the plot can have significant smearing and be difficult to interpret. If you want to plot the CQT, we recommend using the default `TransformType` value `'full'`.

Examples

Constant-Q Transform Using Default Values

Load a signal and obtain the constant-Q transform.

```
load noisdopp
cfs = cqt(noisdopp);
```

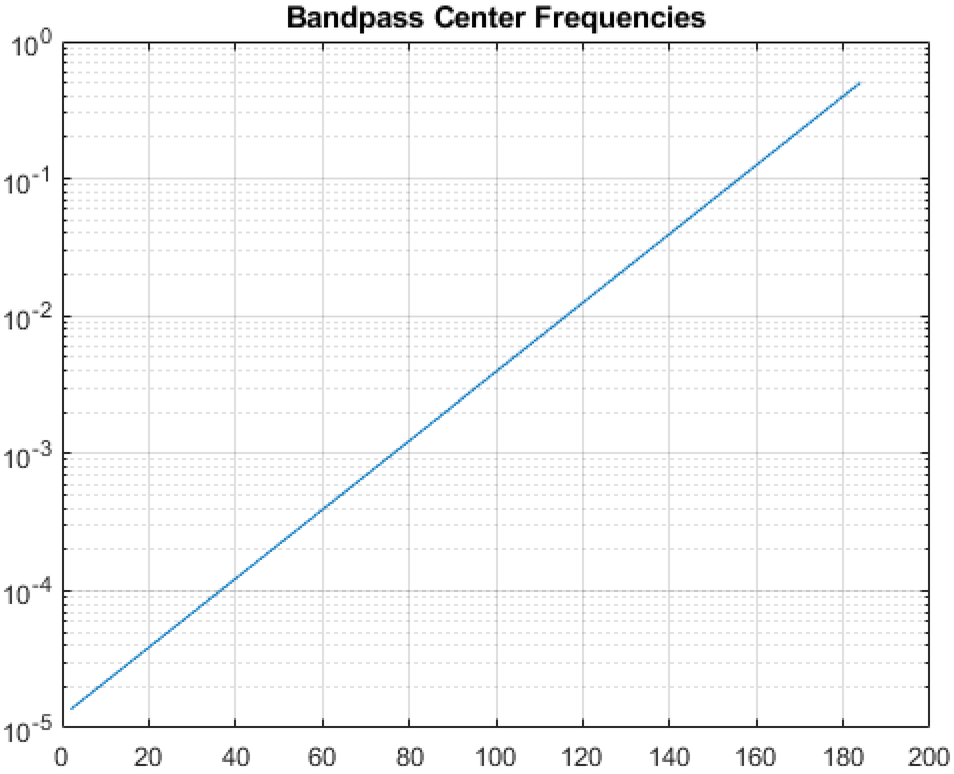
Center Frequencies of the Constant-Q Transform

Load a real-valued signal and obtain the constant-Q transform. Return the approximate bandpass center frequencies.

```
load handel
[cfs,f] = cqt(y);
```

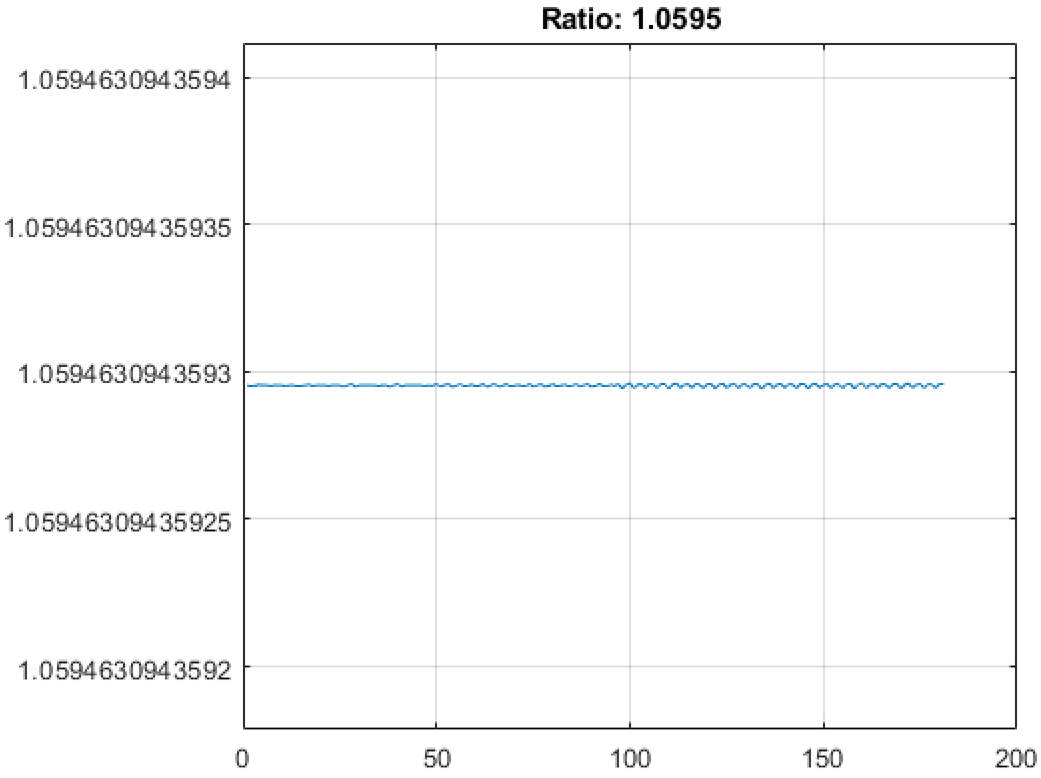
Plot on a logarithmic scale the bandpass center frequencies through the Nyquist frequency.

```
lfreq = length(f);
nyquistBin = floor(lfreq/2)+1;
plot(f(1:nyquistBin))
title('Bandpass Center Frequencies')
grid on
set(gca,'yscale','log')
```



To confirm the ratios of consecutive pairs of frequencies are constant, plot the ratios. Since cqt uses 12 bins per octave by default, the ratio should equal $2^{1/12}$. Since the DC and Nyquist frequencies are not members of the geometric sequence of center frequencies but are included in the frequency vector, exclude them from the plot.

```
figure
plot(f(3:nyquistBin-1)./f(2:nyquistBin-2))
grid on
title(['Ratio: ',num2str(2^(1/12))])
```



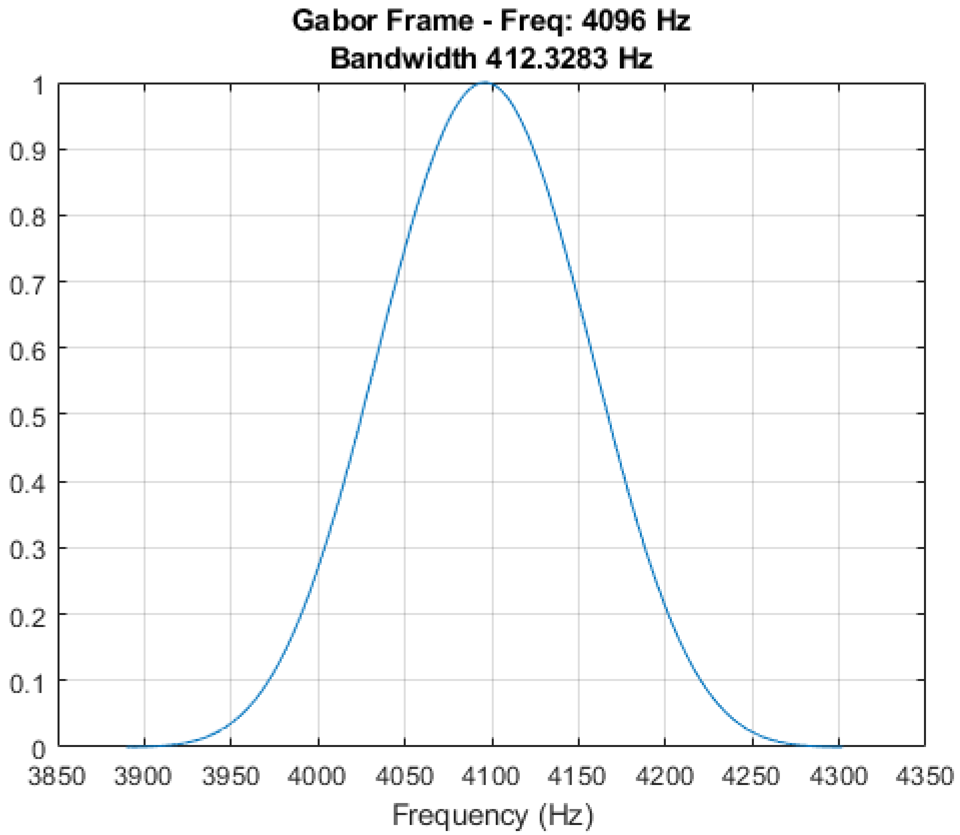
Visualize and Apply Constant-Q Transform Gabor Frames

Obtain the minimally redundant constant-Q transform of an audio signal. Use the Blackman-Harris window as the prototype function for the Gabor frames.

```
load handel
df = Fs/numel(y);
[cfs,f,g,fshifts,fintervals,bw] = cqt(y,'SamplingFrequency',Fs,'TransformType',"sparse",'Window'
```

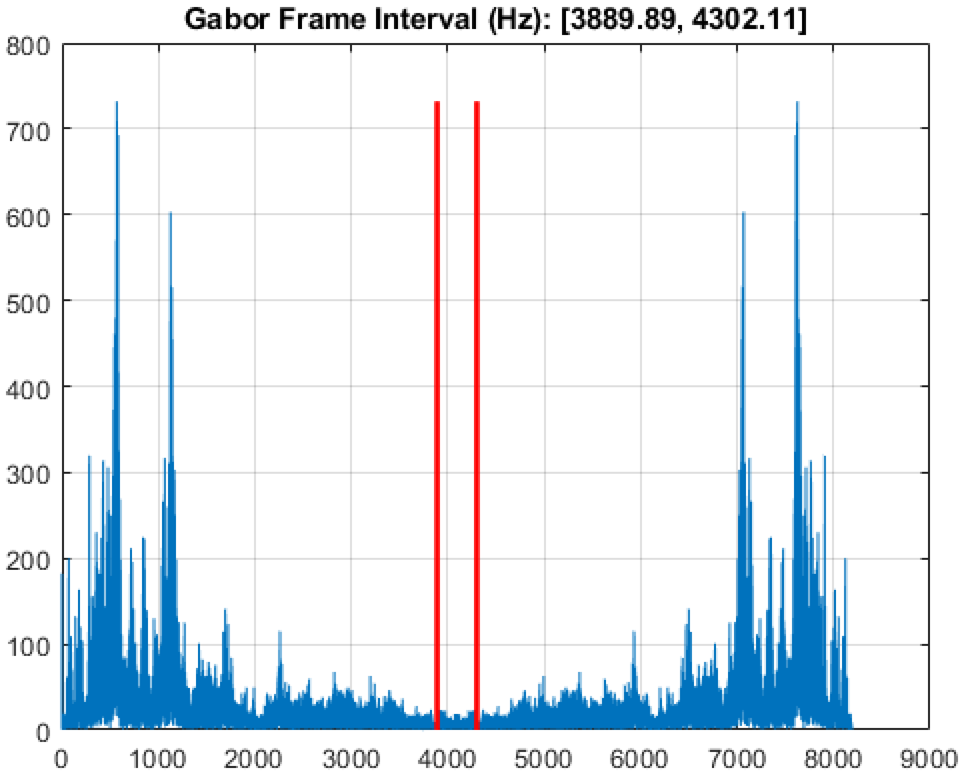
`cfs` is a cell array, where each element in the array corresponds to a bandpass center frequency and Gabor frame. Plot the Gabor frame associated with the Nyquist frequency.

```
lf = length(f);
ind = floor(lf/2)+1;
gFrame = fftshift(g{ind});
fvec = f(ind-1):df:f(ind+1)-df;
plot(fvec,gFrame)
xlabel('Frequency (Hz)')
grid on
title({['Gabor Frame - Freq: ',num2str(f(ind)),' Hz'];['Bandwidth ',num2str(bw(ind)*Fs/numel(y))
```



In the constant-Q transform, the Gabor frames are applied to the discrete Fourier transform of the input signal, and the inverse discrete Fourier transform is performed. The k-th Gabor frame is applied to the k-th frequency interval specified in `fintervals`. Take the discrete Fourier transform of the signal and plot its magnitude spectrum. Use `fintervals` to indicate over which Fourier coefficients are the Gabor frame associated with the Nyquist frequency are applied.

```
yDFT = fft(y);
lyDFT = length(yDFT);
plot(Fs*(0:lyDFT-1)/lyDFT,abs(yDFT))
grid on
fIntervalGabor = fintervals{ind};
mx = max(abs(yDFT));
hold on
plot([df*fIntervalGabor(1) df*fIntervalGabor(1)],[0 mx],'r-','LineWidth',2)
plot([df*fIntervalGabor(end) df*fIntervalGabor(end)],[0 mx],'r-','LineWidth',2)
str = sprintf('Gabor Frame Interval (Hz): [%3.2f, %3.2f]',df*fIntervalGabor(1),df*fIntervalGabor
title(str)
```



Window the Fourier coefficients in the interval with the Gabor frame, and take the inverse discrete Fourier transform. Normalize the result, and compare with computed constant-Q coefficients and confirm they are equal.

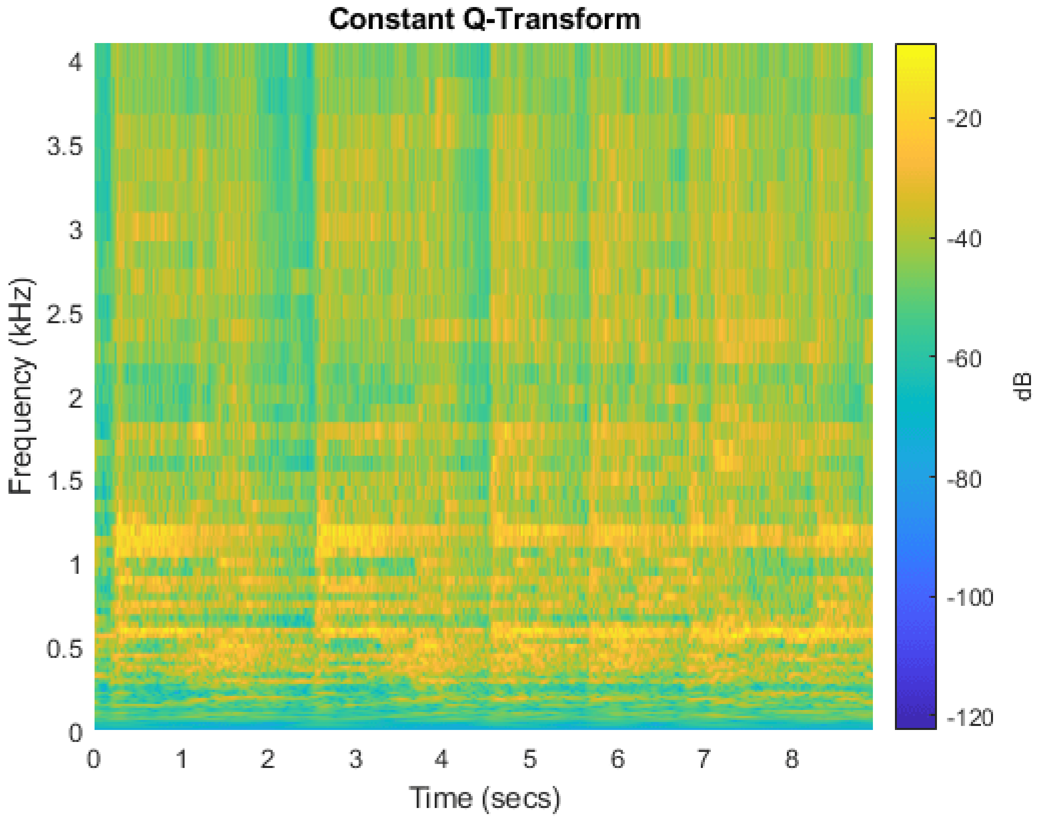
```
lGframe = length(gFrame);
indx = 1:lGframe;
indx = fftshift(indx);
winDFT(indx) = yDFT(fIntervalGabor).*fftshift(gFrame(indx));
cqCoefs = ifft(winDFT);
cqCoefs = (2*lGframe/length(y))*cqCoefs;
max(abs(cqCoefs(:)-cfs{ind}(:)))
```

```
ans = 0
```

Constant-Q Transform of Audio Signal

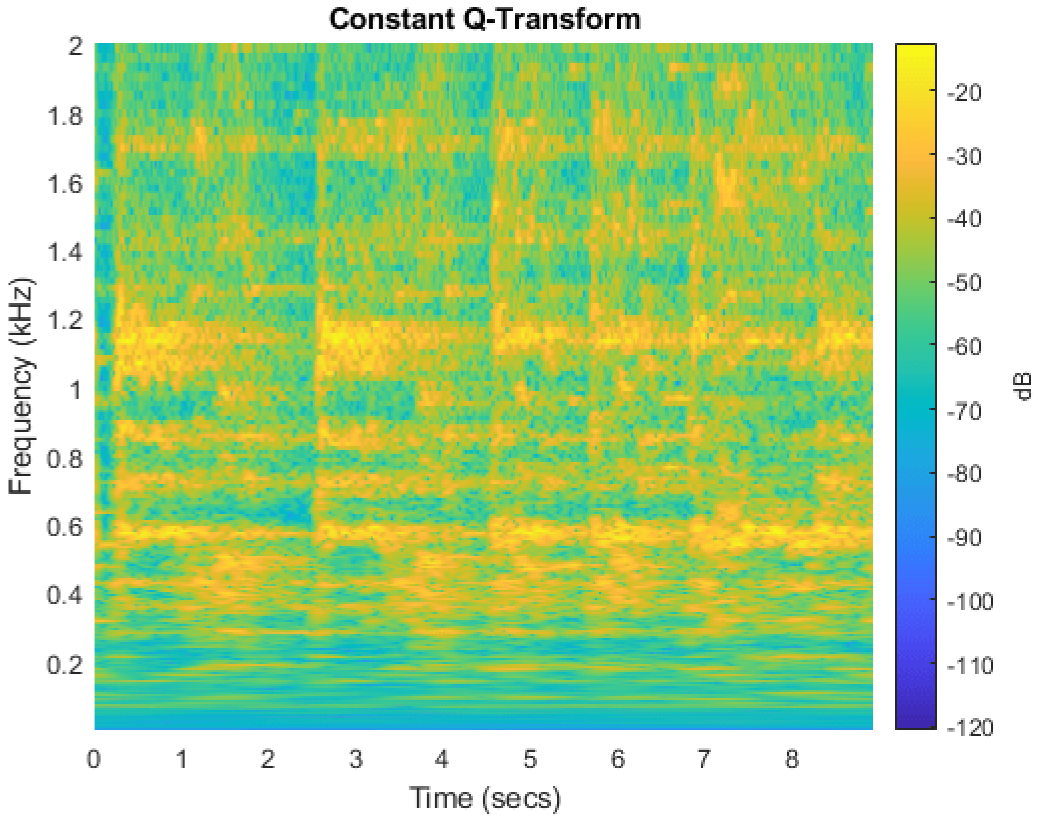
Load an audio signal. Plot the constant-Q transform (CQT) using the maximally redundant version of the transform and using 12 bins per octave.

```
load handel
cqt(y,'SamplingFrequency',Fs)
```



Perform the CQT of the same signal using 48 bins per octave. Set the frequency range over which the CQT has a logarithmic frequency response to be the minimum allowable frequency to 2 kHz.

```
minFreq = Fs/length(y);
maxFreq = 2000;
figure
cqt(y,'SamplingFrequency',Fs,'BinsPerOctave',48,'FrequencyLimits',[minFreq maxFreq])
```



Input Arguments

x — Input signal

vector | matrix

Input signal, specified as a real or complex vector or matrix. `x` must have at least four samples.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'SamplingFrequency',20,'BinsPerOctave',15`

`SamplingFrequency` — Sampling frequency

positive scalar

Sampling frequency, in Hz, specified as the comma-separated pair consisting of `'SamplingFrequency'` and a positive scalar.

`BinsPerOctave` — Number of bins per octave

12 (default) | positive integer from 1 to 96

Number of bins per octave to use in the CQT, specified as a positive integer from 1 to 96.

`TransformType` — Type of constant-Q transform

`'full'` (default) | `'sparse'`

Type of constant-Q transform to perform, specified as the comma-separated pair consisting of `'TransformType'` and `'full'` or `'sparse'`. The sparse transform is the minimally redundant version of the constant-Q transform.

`FrequencyLimits` — Frequency limits

two-element real vector

Frequency limits over which the CQT has a logarithmic frequency response with the specified number of frequency bins per octave, specified as the comma-separated pair `'FrequencyLimits'` and a two-element real vector.

- The first element must be greater than or equal to `Fs/N`, where `Fs` is the sampling frequency and `N` is the length of the signal.
- The second element must be strictly less than the Nyquist frequency.

`Window` — Window to use as prototype function

`'hann'` (default) | `'hamming'` | `'blackmanharris'` | `'itersine'` | `'bartlett'`

Window to use as the prototype function for the nonstationary Gabor frames, specified as `'hann'`, `'hamming'`, `'blackmanharris'`, `'itersine'`, or `'bartlett'`. These compactly support functions are defined in frequency. For normalized frequencies, they are defined on the interval `(-1/2,1/2)`. If you specify a sampling frequency, `Fs`, they are defined on the interval `(-Fs/2,Fs/2)`.

Output Arguments

`cfs` — Constant-Q transform

matrix | multidimensional array | cell array | structure array

Constant-Q transform, returned as a matrix, multidimensional array, cell array, or structure array.

- If `'TransformType'` is specified as `'full'` without `'FrequencyLimits'`, `cfs` is a matrix or multidimensional array.

  - If `x` is a vector, then `cqt` returns a matrix corresponding to the CQT.
  - If `x` is a matrix, then `cqt` obtains the CQT for each column (independent channel) of `x`. The function returns a multidimensional array corresponding to the maximally redundant version of the CQT.

  The array, `cfs`, corresponds to the maximally redundant version of the CQT. Each row of the pages of `cfs` corresponds to passbands with normalized center frequencies (cycles/sample) logarithmically spaced between 0 and 1. A normalized frequency of 1/2 corresponds to the Nyquist frequency. The number of columns, or hops, corresponds to the largest bandwidth center frequency, which usually occurs one frequency bin below or above the Nyquist bin.
- If `'TransformType'` is specified as `'full'` and you specify frequency limits, `cfs` is returned as a structure array with the following four fields.

  - c — Coefficient matrix of multidimensional array for the frequencies within the specified frequency limits. This includes both the positive and "negative" frequencies.
  - DCcfs — Coefficient vector or matrix for the passband from 0 to the lower frequency limit.
  - Nyquistcfs — Coefficient vector or matrix for the passband from the upper frequency limit to the Nyquist.
  - NyquistBin — DFT bin corresponding to the Nyquist frequency. This field is used when inverting the CQT.
- If 'TransformType' is specified as 'sparse', cfs is a cell array with the number of elements equal to the number of bandpass frequencies. Each element of the cell array, cfs, is a vector or matrix with the number of rows equal to the value of the bandwidth in DFT bins, bw.

cfs, g, and fshifts are required inputs for the inversion of the CQT with icqt.

f — Approximate bandpass center frequencies
real-valued vector

Approximate bandpass center frequencies corresponding to the rows of cfs, returned as a real-valued vector. The frequencies are ordered from 0 to 1 and are in cycles/sample. If you specified 'SamplingFrequency', then f is in Hertz.

g — Gabor frames
cell array of real-valued vectors

Gabor frames used in the analysis of x, returned as a cell array of real-valued vectors. Each vector in g corresponds to a row of cfs.

cfs, g, and fshifts are required inputs for the inversion of the CQT with icqt.

fshifts — Frequency shifts
real-valued vector

Frequency shifts in discrete Fourier transform bins, returned as a real-valued vector. The shifts are between the passbands in the rows of cfs.

cfs, g, and fshifts are required inputs for the inversion of the CQT with icqt.

fintervals — Frequency intervals
cell array of real-valued vectors

Frequency intervals corresponding to the rows of cfs, returned as a cell array. Each element in fintervals is a real-valued vector. The kth element of fshifts is the frequency shift in DFT bins between the ((k-1) mod N) and (k mod N) element of fintervals with k = 0,1,2,...,N-1 where N is the number of frequency shifts. Because MATLAB indexes from 1, fshifts(1) contains the frequency shift between fintervals{end} and fintervals{1}, fshifts(2) contains the frequency shift between fintervals{1} and fintervals{2}, and so on.

bw — Bandwidths
real-valued vector

Bandwidths in DFT bins of the frequency intervals, fintervals, returned as a real-valued vector.

Algorithms

Nonstationary Gabor Frames

The theory of nonstationary Gabor (NSG) frames for frequency-adaptive analysis and efficient algorithms for analysis and synthesis using NSG frames are due to Dörfler, Holighaus, Grill, and Velasco [1],[2]. The algorithms used in CQT and ICQT were developed by Dörfler, Holighaus, Grill, and Velasco and are described in [1],[2]. In [3], Schörkhuber, Klapuri, Holighaus, and Dörfler develop and provide algorithms for a phase-corrected CQT transform which matches the CQT coefficients that would be obtained by naïve convolution. The Large Time-Frequency Analysis Toolbox (https://github.com/ltfat) provides an extensive suite of algorithms for nonstationary Gabor frames [4].

Perfect Reconstruction

To achieve the perfect reconstruction property of the constant-Q analysis with nonstationary Gabor frames, `cqt` internally prepends the zero frequency (DC) and appends the Nyquist frequency to the frequency interval. The negative frequencies are mirrored versions of the positive center frequencies and bandwidths

References

[1] Holighaus, N., M. Dörfler, G. A. Velasco, and T. Grill. "A framework for invertible real-time constant-Q transforms." *IEEE Transactions on Audio, Speech, and Language Processing.* Vol. 21, No. 4, 2013, pp. 775–785.

[2] Velasco, G. A., N. Holighaus, M. Dörfler, and T. Grill. "Constructing an invertible constant-Q transform with nonstationary Gabor frames." In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*. Paris, France: 2011.

[3] Schörkhuber, C., A. Klapuri, N. Holighaus, and M. Dörfler. "A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution." Submitted to the *AES 53rd International Conference on Semantic Audio.* London, UK: 2014.

[4] Průša, Z., P. L. Søndergaard, N. Holighaus, C. Wiesmeyr, and P. Balazs. *The Large Time-Frequency Analysis Toolbox 2.0*. Sound, Music, and Motion, Lecture Notes in Computer Science 2014, pp 419-442.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The value of the `'TransformType'` name-value pair argument must be constant. Use `coder.Constant`.
- Plotting is not supported.

See Also
`icqt`

Topics
“Nonstationary Gabor Frames and the Constant-Q Transform”
“Time-Frequency Gallery”

Introduced in R2018a

createDatastores

Create datastores pointing to signal and label data

Syntax

```
[sigdata,lbldata] = createDatastores(lss,lblnames)
```

Description

`[sigdata,lbldata] = createDatastores(lss,lblnames)` creates a datastore, `sigdata`, containing signal member data, and a datastore, `lbldata`, containing label data from labels specified in the string array `lblnames`. `createDatastores` does not apply to sublabels. Set `lblnames` to one or more parent label names to get the parent labels and the corresponding sublabel values.

Examples

Create Datastores

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Display the labels for the first member of the set.

```
lss.Labels(1,:)
```

```
ans=1×3 table
                 WhaleType    MoanRegions    TrillRegions
                 _________    ___________    ____________

    Member{1}      blue       {3x2 table}    {1x3 table} 
```

Get the names of the labels in the set. Create a signal datastore with the signal information and an array datastore with the label information.

```
lbls = getLabelNames(lss);
[sgd,lbd] = createDatastores(lss,lbls)

sgd = 
  signalDatastore with properties:

    MemberNames:{
                 'Member{1}';
                 'Member{2}'
                 }
        Members: {2x1 cell}
       ReadSize: 1
     SampleRate: 4000


lbd = 
  ArrayDatastore with properties:

              ReadSize: 1
    IterationDimension: 1
            OutputType: "cell"
```

Display the labels for the first member of the set.

```
lbls = read(lbd);
lbls{:}

ans=1×3 table
    WhaleType    MoanRegions     TrillRegions
    _________    ___________     ____________

      blue       {3x2 table}     {1x3 table} 
```

Count Label Values and Create Datastores

Specify the path to a set of audio signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'

nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to Fs, which is common to all files. Generate a subset of the datastore that excludes the file mtlb.mat. Use the subset datastore as the source for a labeledSignalSet object.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sds = subset(sds,~strcmp(nms,"mtlb.mat"));
lss = labeledSignalSet(sds);
```

Create three label definitions to label the signals:

- Define a logical attribute label that is true for signals that contain human voices.
- Define a numeric point label that marks the location and amplitude of the maximum of each signal.
- Define a categorical region-of-interest (ROI) label to pick out nonoverlapping, uniform-length random regions of each signal.

Add the signal label definitions to the labeled signal set.

```
vc = signalLabelDefinition("Voice",'LabelType','attribute', ...
    'LabelDataType','logical','DefaultValue',false);
mx = signalLabelDefinition("Maximum",'LabelType','point', ...
    'LabelDataType','numeric');
rs = signalLabelDefinition("RanROI",'LabelType','ROI', ...
    'LabelDataType','categorical','Categories',["ROI" "other"]);
addLabelDefinitions(lss,[vc mx rs])
```

Label the signals:

- Label 'handel.mat' and 'laughter.mat' as having human voices.
- Use the islocalmax function to find the maximum of each signal. Label its location and value.
- Use the randROI on page 1-0 function to generate as many regions of length *N*/10 samples as can fit in a signal of length *N* given a minimum separation of *N*/6 samples between regions. Label their locations and assign them to the ROI category.

When labeling points and regions, convert sample values to time values. Subtract 1 to account for MATLAB® array indexing and divide by the sample rate.

```
kj = 1;
while hasdata(sds)

    [sig,info] = read(sds);
    fs = info.SampleRate;

    [~,fn] = fileparts(info.FileName);
    if fn=="handel" || fn=="laughter"
        setLabelValue(lss,kj,"Voice",true)
    end

    xm = find(islocalmax(sig,'MaxNumExtrema',1));
    setLabelValue(lss,kj,"Maximum",(xm-1)/fs,sig(xm))

    N = length(sig);
    rois = randROI(N,round(N/10),round(N/6));
    setLabelValue(lss,kj,"RanROI",(rois-1)/fs,repelem("ROI",size(rois,1)))

    kj = kj+1;
```

```
end
```

Verify that only two signals contain voices.

```
countLabelValues(lss,"Voice")
```

```
ans=2×3 table
    Voice    Count    Percent
    _____    _____    _______

    false      4      66.667 
    true       2      33.333 
```

Verify that two signals have a maximum amplitude of 1.

```
countLabelValues(lss,"Maximum")
```

```
ans=5×4 table
           Maximum             Count    Percent    MemberCount
    ______________________     _____    _______    ___________

    0.80000000000000004441       1      16.667          1     
    0.89113331915798421612       1      16.667          1     
    0.94730769230769229505       1      16.667          1     
    1                            2      33.333          2     
    1.0575668990330560071        1      16.667          1     
```

Verify that each signal has four nonoverlapping random regions of interest.

```
countLabelValues(lss,"RanROI")
```

```
ans=2×4 table
    RanROI    Count    Percent    MemberCount
    ______    _____    _______    ___________

    ROI        24        100           6     
    other       0          0           0     
```

Create two datastores with the data in the labeled signal set:

- The `signalDatastore` (Signal Processing Toolbox) object `sd` contains the signal data.
- The `arrayDatastore` object `ld` contains the labeling information. Specify that you want to include the information corresponding to all the labels you created.

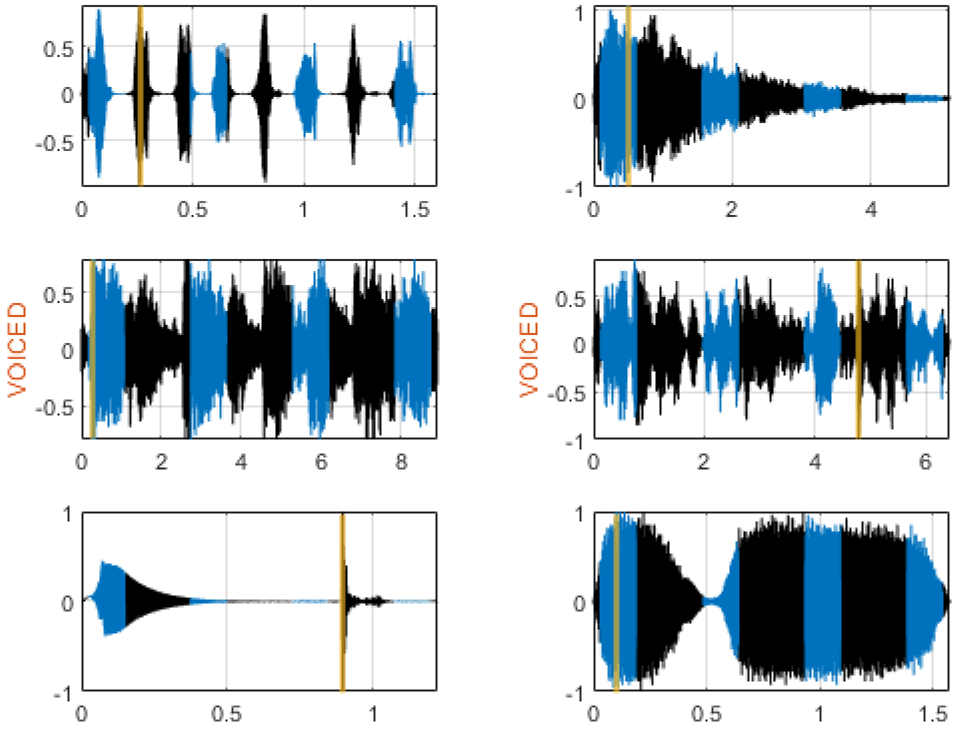
```
[sd,ld] = createDatastores(lss,["Voice" "RanROI" "Maximum"]);
```

Use the information in the datastores to plot the signals and display their labels.

- Use a `signalMask` (Signal Processing Toolbox) object to highlight the regions of interest in blue.
- Plot yellow lines to mark the locations of the maxima.
- Add a red axis label to the signals that contain human voices.

```
tiledlayout flow
```

```
while hasdata(sd)

    [sg,nf] = read(sd);

    lbls = read(ld);

    nexttile

    msk = signalMask(lbls{:}.RanROI{:},'SampleRate',nf.SampleRate);
    plotsigroi(msk,sg)
    colorbar off
    xlabel('')

    xline(lbls{:}.Maximum{:}.Location, ...
        'LineWidth',2,'Color','#EDB120')

    if lbls{:}.Voice{:}
        ylabel('VOICED','Color','#D95319')
    end

end
```



```
function roilims = randROI(N,wid,sep)

num = floor((N+sep)/(wid+sep));
hq = histcounts(randi(num+1,1,N-num*wid-(num-1)*sep),(1:num+2)-1/2);
roilims = (1 + (0:num-1)*(wid+sep) + cumsum(hq(1:num)))' + [0 wid-1];
```

```
end
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lblnames` — Label names

character vector | string scalar | cell array of character vectors | string array

Label names, specified as a character vector, a string scalar, a cell array of character vectors, or a string array.

Data Types: `char` | `string`

Output Arguments

`sigdata` — Signal data

`signalDatastore` object | `audioDatastore` object

Signal data, returned as a `signalDatastore` object or an `audioDatastore` object.

`lbldata` — Label data

`arrayDatastore` object

Label data, returned as an `arrayDatastore` object.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2021a

ctranspose

Laurent matrix transpose

Syntax

```
B = ctranspose(A)
```

Description

`B = ctranspose(A)` returns the transpose of the Laurent matrix `A`.

Examples

Laurent Matrix Transpose

Create two Laurent polynomials:

- $a(z) = 2 + 4z^{-1} + 6z^{-2}$
- $b(z) = z^2 + 3z + 5$

```
lpA = laurentPolynomial(Coefficients=[2 4 6]);
lpB = laurentPolynomial(Coefficients=[1 3 5],MaxOrder=2);
```

Create the Laurent matrix `lmat` $= \begin{bmatrix} -1 & a(z) \\ b(z) & 7 \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={-1 lpA; lpB 7});
```

Display the elements of the transpose of `lmat`.

```
lmatTrans = ctranspose(lmat);
for j=1:2
    for k=1:2
        elt = lmatTrans.Elements{j,k};
        fprintf("===================\nlmatTrans(%d,%d):\n",j,k);
        elt
    end
end
```

```
===================
lmatTrans(1,1):

elt = 
  laurentPolynomial with properties:

    Coefficients: -1
        MaxOrder: 0


===================
lmatTrans(1,2):
```

```
elt = 
  laurentPolynomial with properties:

    Coefficients: [1 3 5]
        MaxOrder: 2


===================
lmatTrans(2,1):

elt = 
  laurentPolynomial with properties:

    Coefficients: [2 4 6]
        MaxOrder: 0


===================
lmatTrans(2,2):

elt = 
  laurentPolynomial with properties:

    Coefficients: 7
        MaxOrder: 0
```

Input Arguments

A — Laurent matrix
`laurentMatrix` object

Laurent matrix, specified as a `laurentMatrix` object.

Output Arguments

B — Transpose
`laurentMatrix` object

Transpose of a Laurent matrix, returned as a `laurentMatrix` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

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

[___,coi,fb] = cwt(___)
[___,fb,scalingcfs] = cwt(___)
[___] = cwt(___,Name=Value)

cwt(___)
```

Description

wt = cwt(x) returns the continuous wavelet transform (CWT) of x. The CWT is obtained using the analytic Morse wavelet with the symmetry parameter, gamma (γ), 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 energy spread of the wavelet in frequency and time.

The cwt function uses L1 normalization. With L1 normalization, if you have equal amplitude oscillatory components in your data at different scales, they will have equal magnitude in the CWT. Using L1 normalization shows a more accurate representation of the signal. See “L1 Norm for CWT” on page 1-167 and “Continuous Wavelet Transform of Two Complex Exponentials” on page 1-143.

wt = cwt(x,wname) uses the analytic wavelet specified by wname to compute the CWT.

[wt,f] = cwt(___,fs) specifies the sampling frequency, fs, in hertz, and returns the scale-to-frequency conversions f in hertz. If you do not specify a sampling frequency, cwt returns f in cycles per sample.

[wt,period] = cwt(___,ts) specifies the sampling period, ts, as a positive duration scalar. cwt uses ts to compute the scale-to-period conversions, period. period is an array of durations with the same Format property as ts.

[wt,f,coi] = cwt(___) returns the cone of influence, coi, in cycles per sample. Specify a sampling frequency, fs, in hertz, to return the cone of influence in hertz.

[wt,period,coi] = cwt(___,ts) returns the cone of influence, coi, as an array of durations with the same Format property as ts.

`[ ___ ,coi,fb] = cwt( ___ )` returns the filter bank used in the CWT. See `cwtfilterbank`.

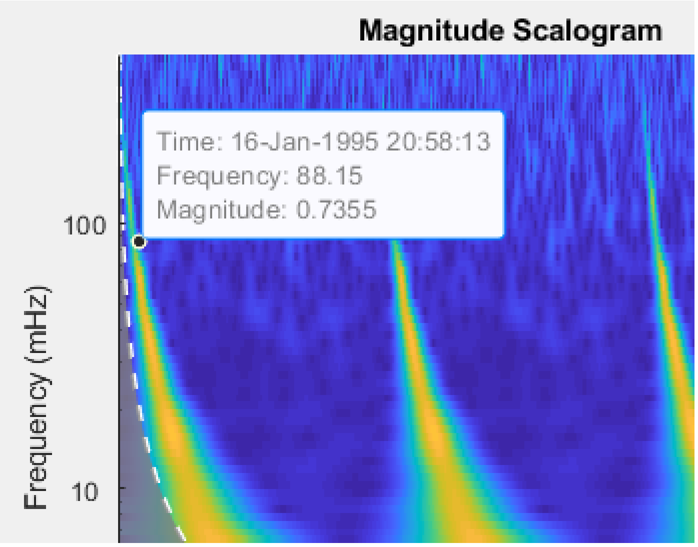
`[ ___ ,fb,scalingcfs] = cwt( ___ )` returns the scaling coefficients for the wavelet transform.

`[ ___ ] = cwt( ___ ,Name=Value)` specifies one or more additional name-value arguments. For example, `wt = cwt(x,TimeBandwidth=40,VoicesPerOctave=20)` specifies a time-bandwidth product of 40 and 20 voices per octave.

`cwt( ___ )` with no output arguments plots the CWT scalogram. The scalogram is the absolute value of the CWT plotted as a function of time and frequency. Frequency is plotted on a logarithmic scale. 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 or sampling period, the frequencies are plotted in cycles per sample. If you specify a sampling frequency, the frequencies are in hertz. If you specify a sampling period, the scalogram is plotted as a function of time and periods. If the input signal is a timetable, the scalogram is plotted as a function of time and frequency in hertz and uses the RowTimes as the basis for the time axis.

To see the time, frequency, and magnitude of a scalogram point, enable data tips in the figure axes toolbar and click the desired point in the scalogram.



Note Before plotting, `cwt` clears (`clf`) the current figure. To plot the scalogram in a subplot, use a plotting function. See "Plot CWT Scalogram in Subplot" on page 1-159.

Examples

Continuous Wavelet Transform Using Default Values

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

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

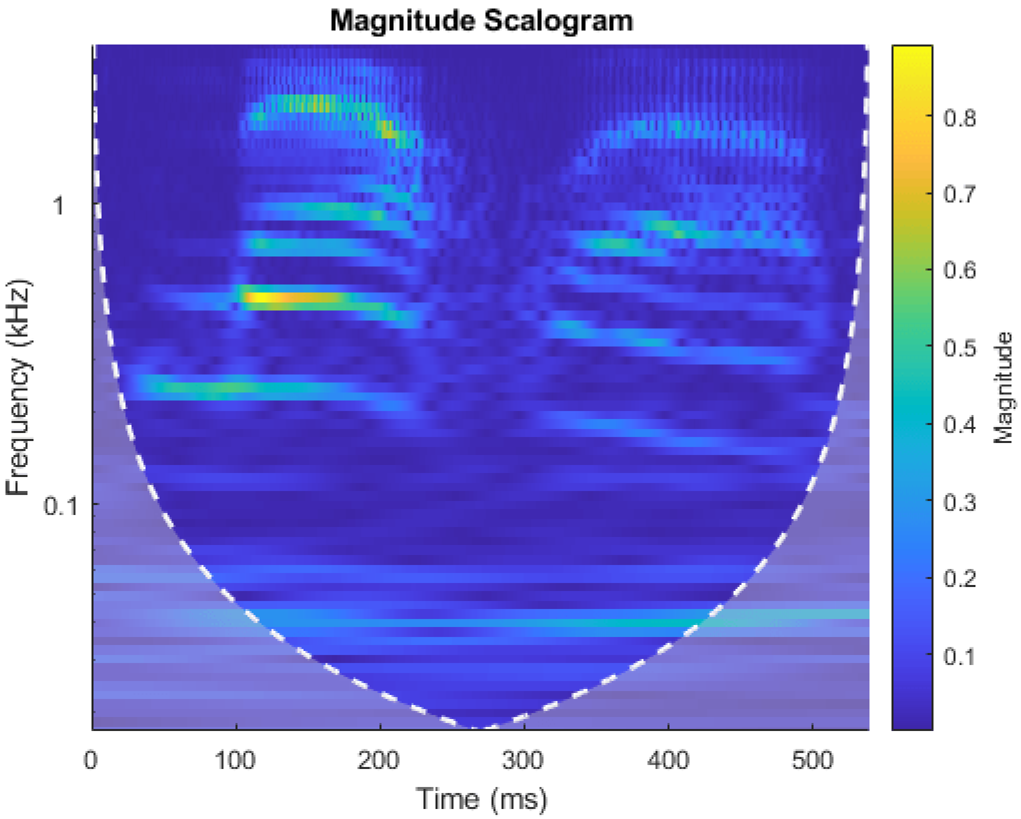
Continuous Wavelet Transform Using Specified Wavelet

Load a speech sample.

```
load mtlb
```

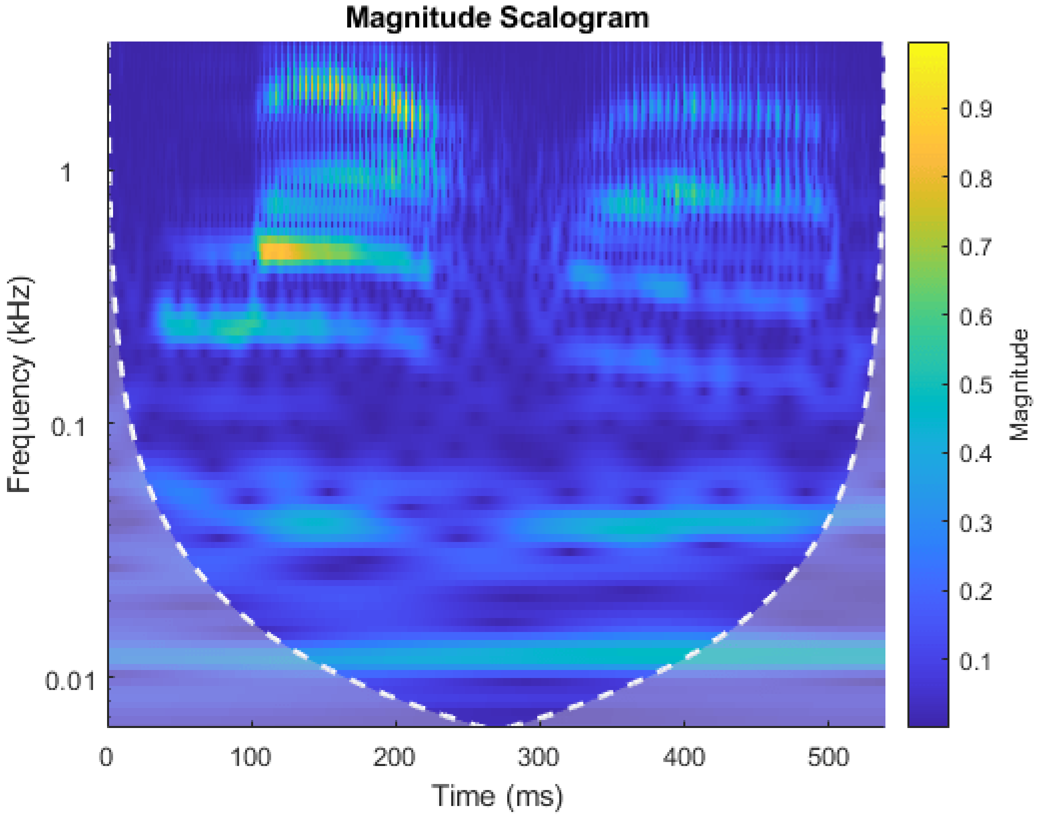
Loading the file `mtlb.mat` brings the speech signal, `mtlb`, and the sample rate, `Fs`, into the workspace. Display the scalogram of the speech sample obtained using the bump wavelet.

```
load mtlb
cwt(mtlb,"bump",Fs)
```



Compare with the scalogram obtained using the default Morse wavelet.

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



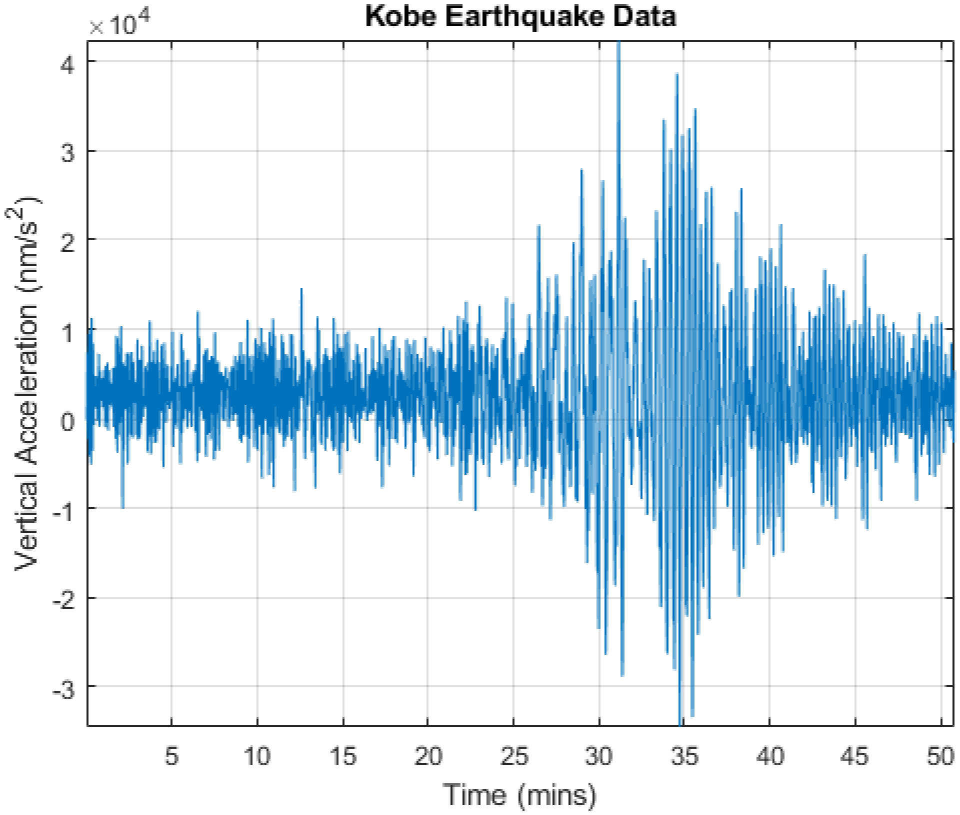
Continuous Wavelet Transform of Earthquake Data

Obtain the CWT of the Kobe earthquake data. The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes. The sampling frequency is 1 Hz.

```
load kobe
```

Plot the earthquake data.

```
plot((1:numel(kobe))./60,kobe)
xlabel("Time (mins)")
ylabel("Vertical Acceleration (nm/s^2)")
title("Kobe Earthquake Data")
grid on
axis tight
```

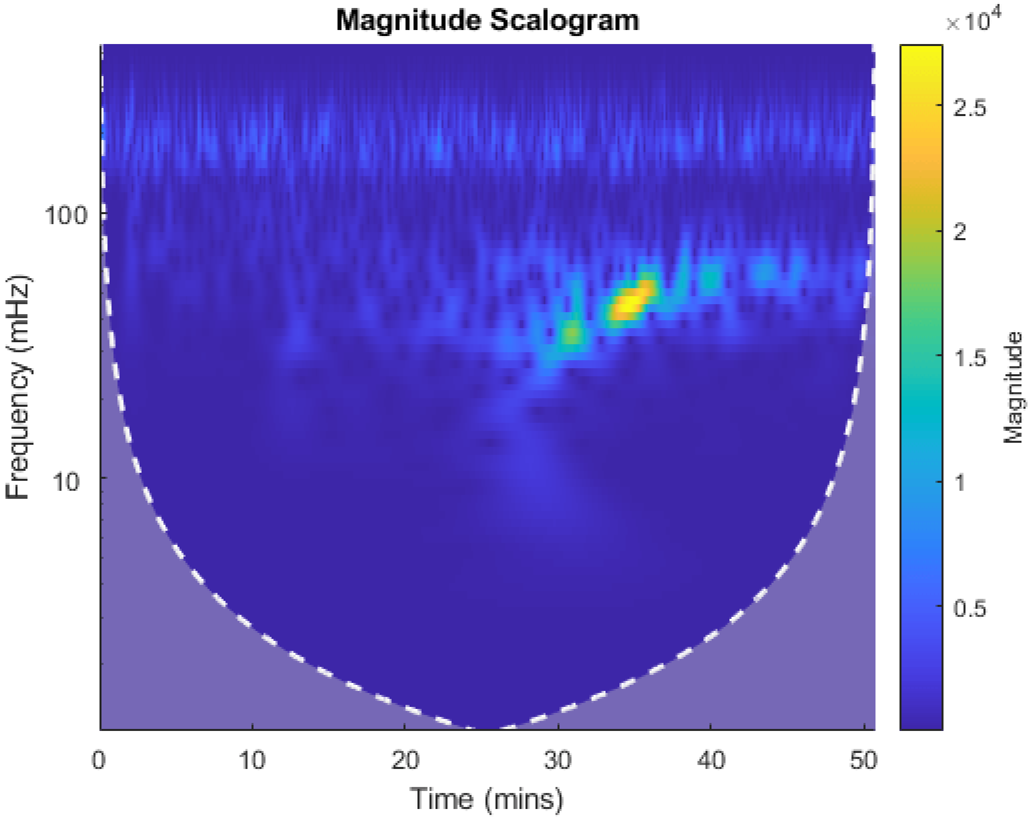


Obtain the CWT, frequencies, and cone of influence.

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

View the scalogram, including the cone of influence.

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

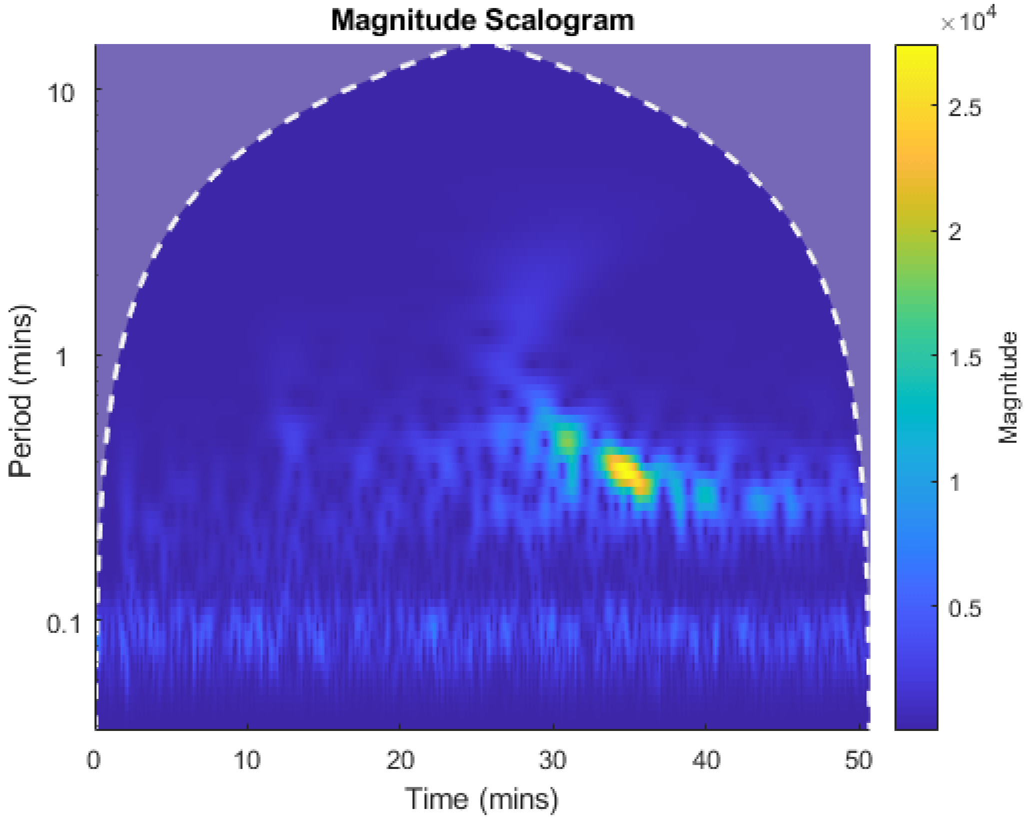


Obtain the CWT, time periods, and cone of influence by specifying a sampling period instead of a sampling frequency.

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

View the scalogram generated when specifying a sampling period.

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



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.

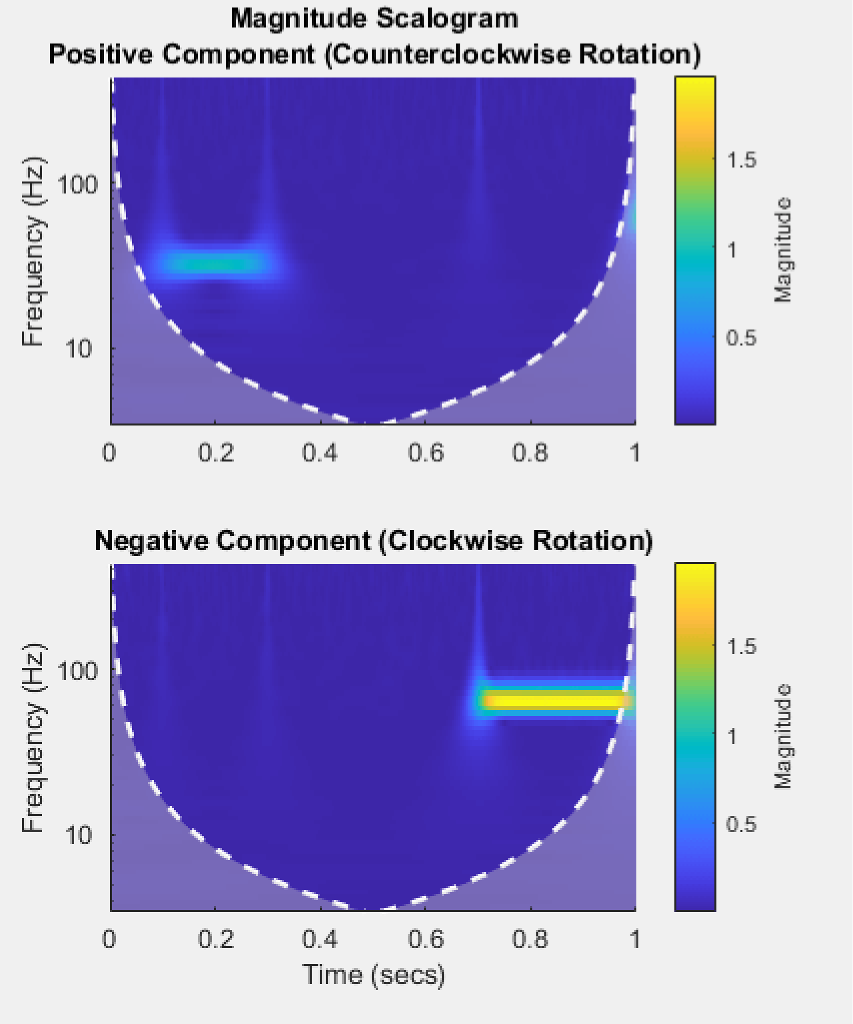
```
Fs = 1e3;
t = 0:1/Fs:1;
z = exp(1i*2*pi*32*t).*(t>=0.1 & t<0.3)+2*exp(-1i*2*pi*64*t).*(t>0.7);
```

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

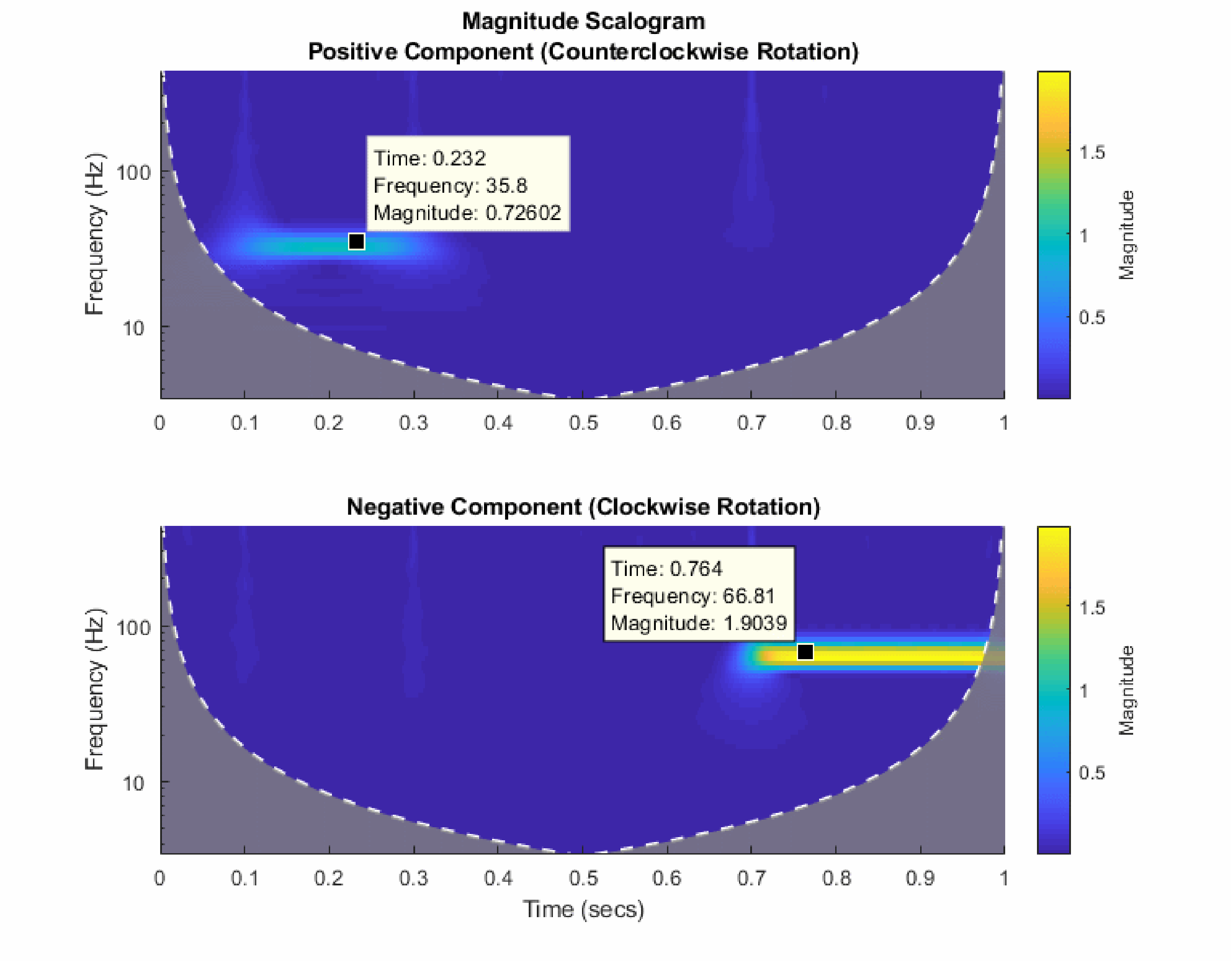
```
wgnNoise = 0.05/sqrt(2)*randn(size(t))+1i*0.05/sqrt(2)*randn(size(t));
z = z+wgnNoise;
```

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 data cursor.



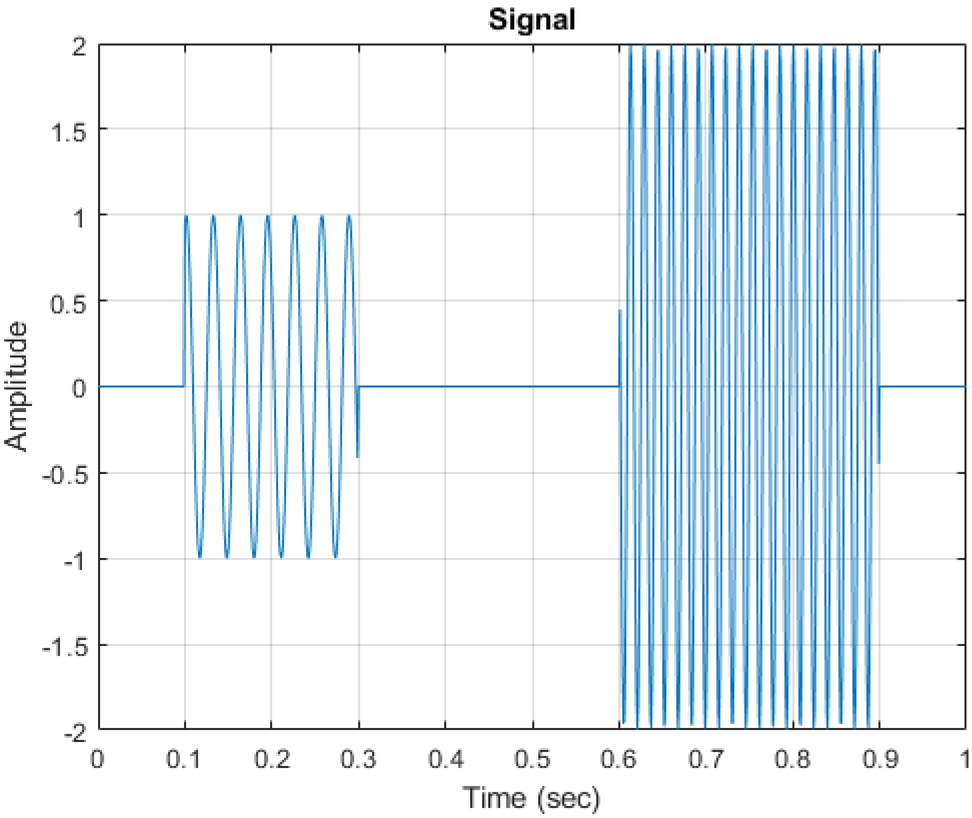
Sinusoid and Wavelet Coefficient Amplitudes

This example shows that the amplitudes of oscillatory components in a signal agree with the amplitudes of the corresponding wavelet coefficients.

Create a signal composed of two sinusoids with disjoint support in time. One sinusoid has a frequency of 32 Hz and amplitude equal to 1. The other sinusoid has a frequency of 64 Hz and amplitude equal to 2. The signal is sampled for one second at 1000 Hz. Plot the signal.

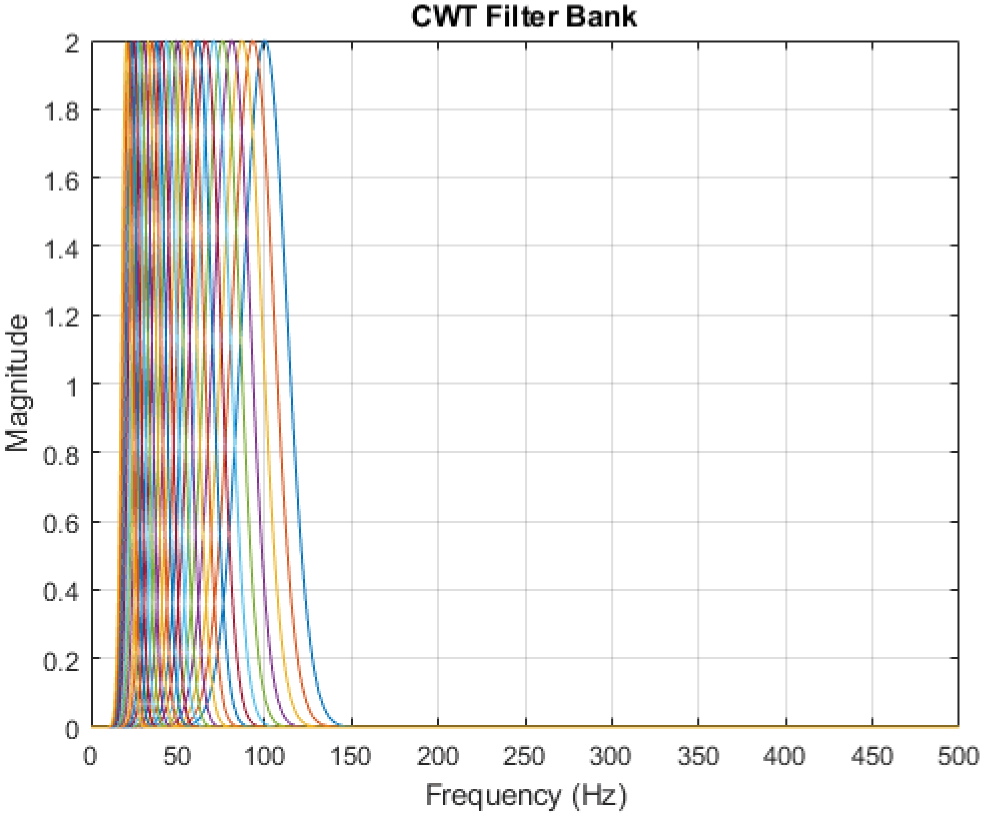
```
frq1 = 32;
amp1 = 1;
frq2 = 64;
amp2 = 2;

Fs = 1e3;
t = 0:1/Fs:1;
```

```
x = amp1*sin(2*pi*frq1*t).*(t>=0.1 & t<0.3)+...
    amp2*sin(2*pi*frq2*t).*(t>0.6 & t<0.9);

plot(t,x)
grid on
xlabel("Time (sec)")
ylabel("Amplitude")
title("Signal")
```



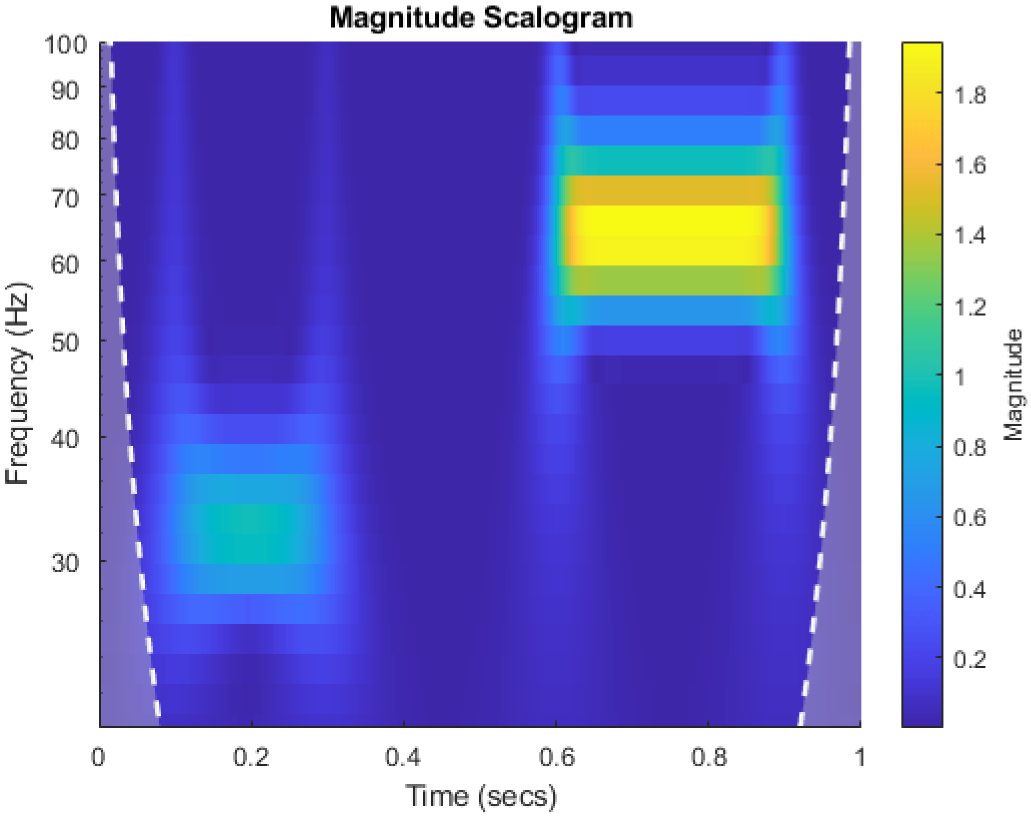
Create a CWT filter bank that can be applied to the signal. Since the signal component frequencies are known, set the frequency limits of the filter bank to a narrow range that includes the known frequencies. To confirm the range, plot the magnitude frequency responses for the filter bank.

```
fb = cwtfilterbank(SignalLength=numel(x),SamplingFrequency=Fs,...
    FrequencyLimits=[20 100]);
freqz(fb)
```

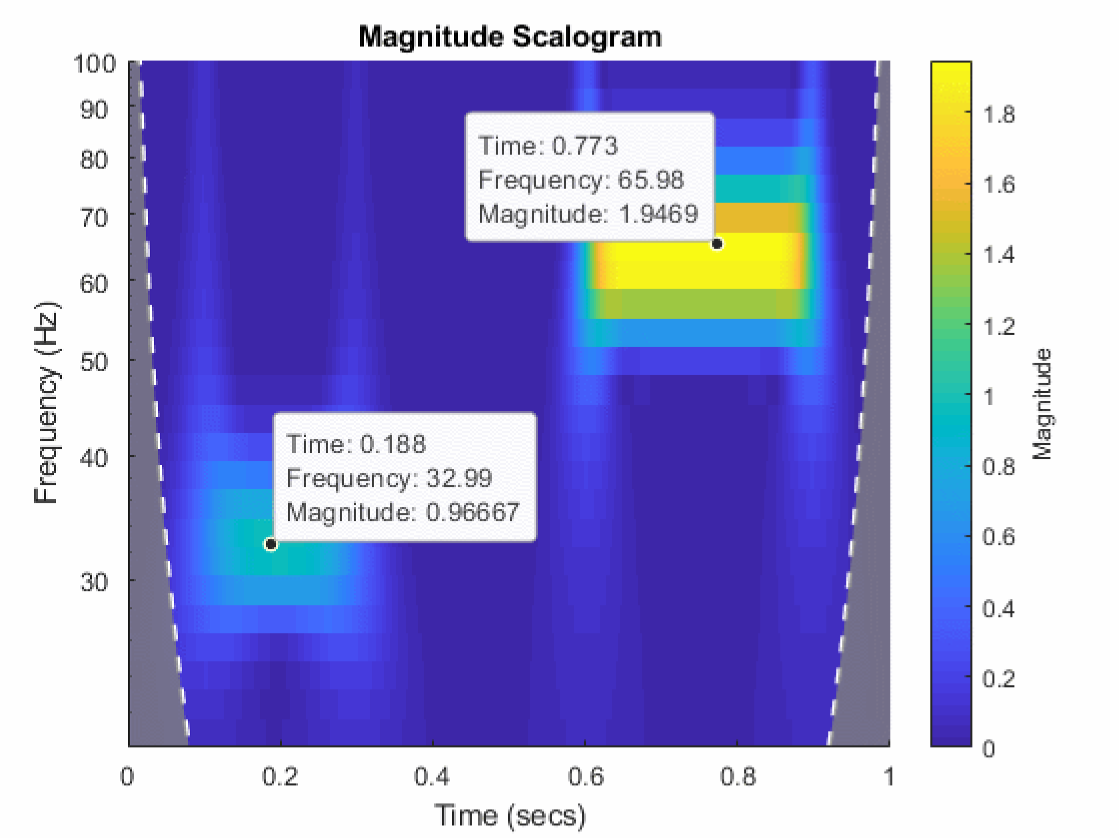


Use `cwt` and the filter bank to plot the scalogram of the signal.

```
cwt(x,FilterBank=fb)
```



Use a data cursor to confirm that the amplitudes of the wavelet coefficients are essentially equal to the amplitudes of the sinusoidal components. Your results should be similar to the ones in the following figure.



Using CWT Filter Bank on Multiple Time Series

This example shows how using a CWT filter bank can improve computational efficiency when taking the CWT of multiple time series.

Create a 100-by-1024 matrix x. Create a CWT filter bank appropriate for signals with 1024 samples.

```
x = randn(100,1024);
fb = cwtfilterbank;
```

Use cwt with default settings to obtain the CWT of a signal with 1024 samples. Create a 3-D array that can contain the CWT coefficients of 100 signals, each of which has 1024 samples.

```
cfs = cwt(x(1,:));
res = zeros(100,size(cfs,1),size(cfs,2));
```

Use the cwt function and take the CWT of each row of the matrix x. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = cwt(x(k,:));
end
toc
```

```
Elapsed time is 0.928160 seconds.
```

Now use the wt object function of the filter bank to take the CWT of each row of x. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = wt(fb,x(k,:));
end
toc
```

```
Elapsed time is 0.393524 seconds.
```

CUDA Code from CWT

This example shows how to generate a MEX file to perform the continuous wavelet transform (CWT) using generated CUDA code.

First, ensure that you have a CUDA-enabled GPU and the NVCC compiler. See “The GPU Environment Check and Setup App” (GPU Coder) to ensure you have the proper configuration.

Create a GPU coder configuration object.

```
cfg = coder.gpuConfig("mex");
```

Generate a signal of 100,000 samples at 1,000 Hz. The signal consists of two cosine waves with disjoint time supports.

```
t = 0:.001:(1e5*0.001)-0.001;
x = cos(2*pi*32*t).*(t > 10 & t<=50)+ ...
    cos(2*pi*64*t).*(t >= 60 & t < 90)+ ...
    0.2*randn(size(t));
```

Cast the signal to use single precision. GPU calculations are often more efficiently done in single precision. You can however also generate code for double precision if your NVIDIA GPU supports it.

```
x = single(x);
```

Generate the GPU MEX file and a code generation report. To allow generation of the MEX file, you must specify the properties (class, size, and complexity) of the three input parameters:

- coder.typeof(single(0),[1 1e5]) specifies a row vector of length 100,000 containing real single values.
- coder.typeof('c',[1 inf]) specifies a character array of arbitrary length.
- coder.typeof(0) specifies a real double value.

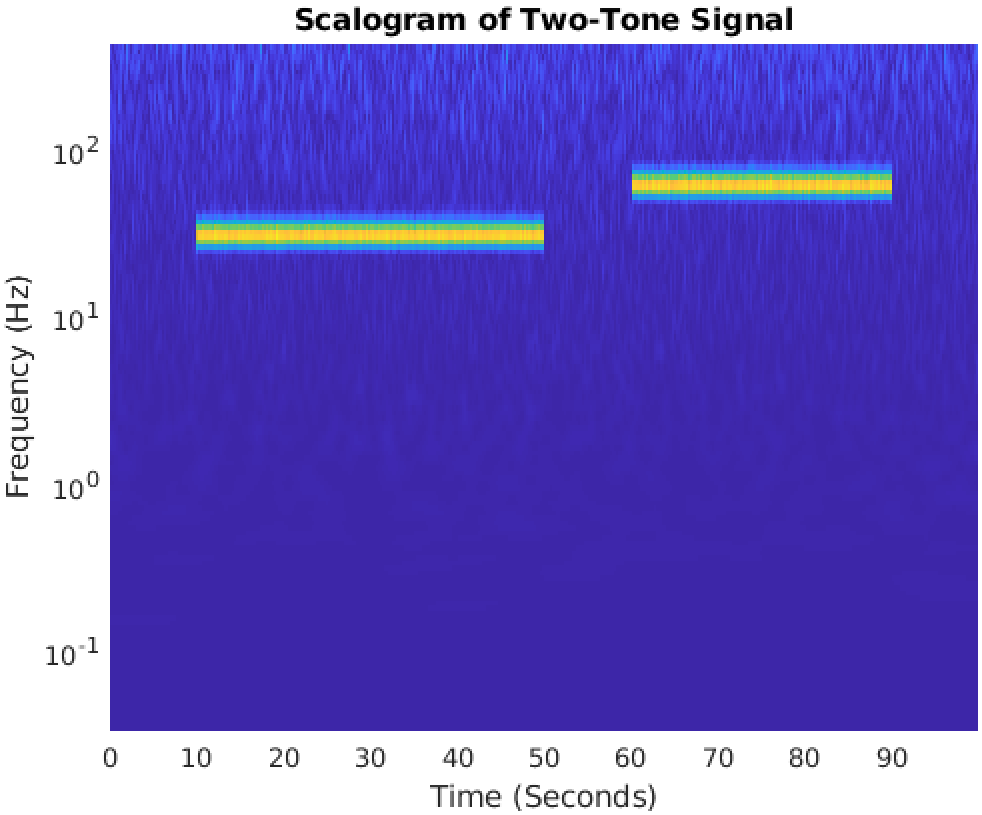
```
sig = coder.typeof(single(0),[1 1e5]);
wav = coder.typeof('c',[1 inf]);
sfrq = coder.typeof(0);
codegen cwt -config cfg -args {sig,wav,sfrq} -report
```

```
Code generation successful: View report
```

The -report flag is optional. Using -report generates a code generation report. In the **Summary** tab of the report, you can find a **GPU code metrics** link, which provides detailed information such as the number of CUDA kernels generated and how much memory was allocated.

Run the MEX file on the data and plot the scalogram. Confirm the plot is consistent with the two disjoint cosine waves.

```
[cfs,f] = cwt_mex(x,'morse',1e3);
image("XData",t,"YData",f,"CData",abs(cfs),"CDataMapping","scaled")
set(gca,"YScale","log")
axis tight
xlabel("Time (Seconds)")
ylabel("Frequency (Hz)")
title("Scalogram of Two-Tone Signal")
```



Run the CWT command above without appending the _mex. Confirm the MATLAB and the GPU MEX scalograms are identical.

```
[cfs2,f2] = cwt(x,'morse',1e3);
max(abs(cfs2(:)-cfs(:)))
```

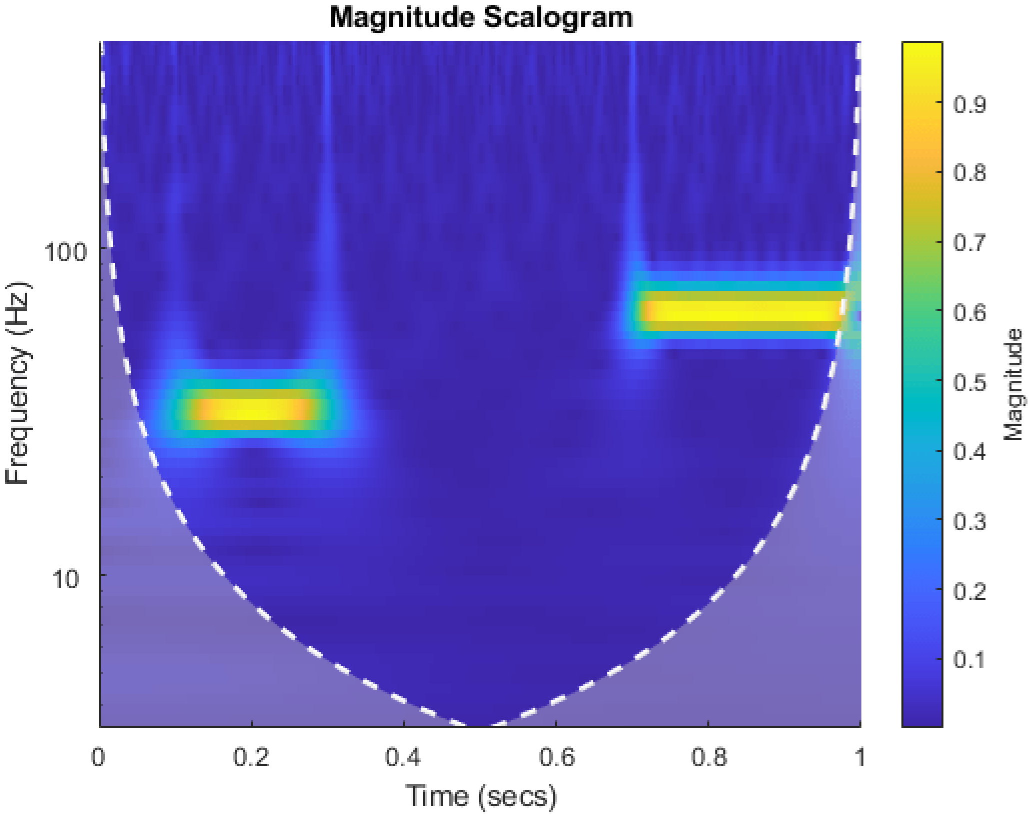
```
ans = single
    7.3380e-07
```

Change Default Frequency Axis Labels

This example shows how to change the default frequency axis labels for the CWT when you obtain a plot with no output arguments.

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. Add white Gaussian noise with a standard deviation of 0.05. Obtain and plot the CWT using the default Morse wavelet.

```
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);
wgnNoise = 0.05*randn(size(t));
x = x+wgnNoise;
cwt(x,1000)
```



The plot uses a logarithmic frequency axis because frequencies in the CWT are logarithmic. In MATLAB, logarithmic axes are in powers of 10 (decades). You can use `cwtfreqbounds` to determine what the minimum and maximum wavelet bandpass frequencies are for a given signal length, sampling frequency, and wavelet.

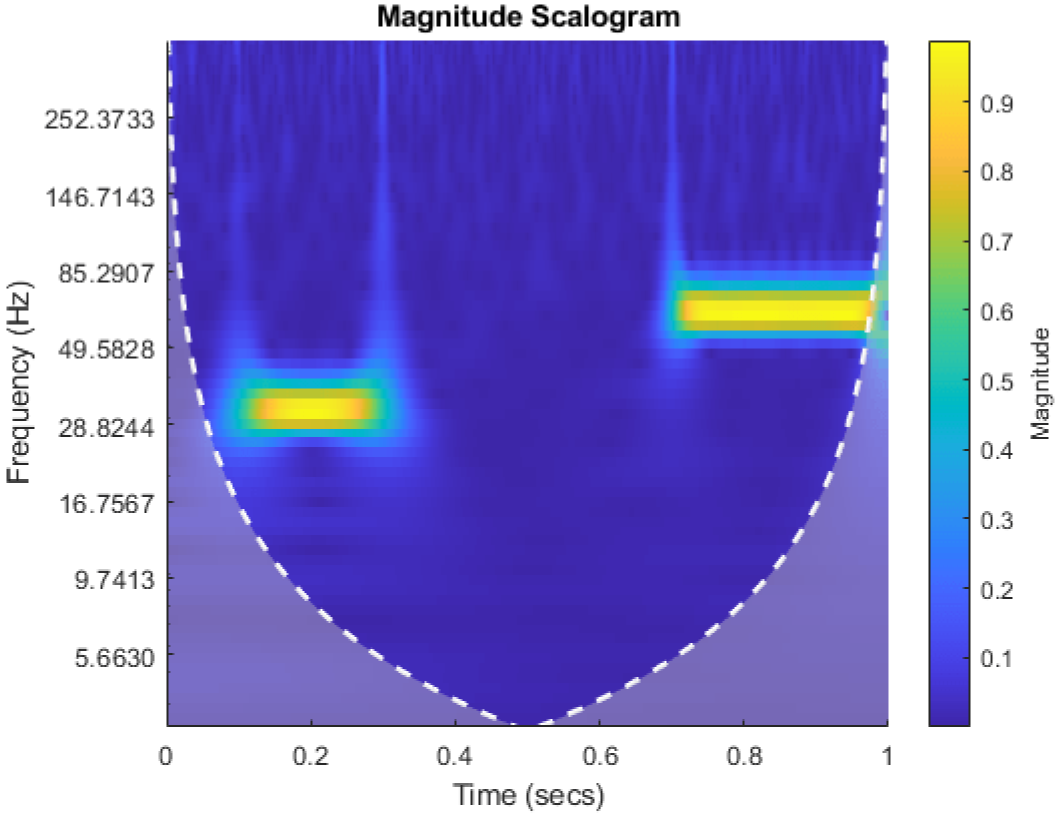
```
[minf,maxf] = cwtfreqbounds(numel(x),1000);
```

You see that by default MATLAB has placed frequency ticks at 10 and 100 because those are the powers of 10 between the minimum and maximum frequencies. If you wish to add more frequency axis ticks, you can obtain a logarithmically spaced set of frequencies between the minimum and maximum frequencies using the following.

```
numfreq = 10;
freq = logspace(log10(minf),log10(maxf),numfreq);
```

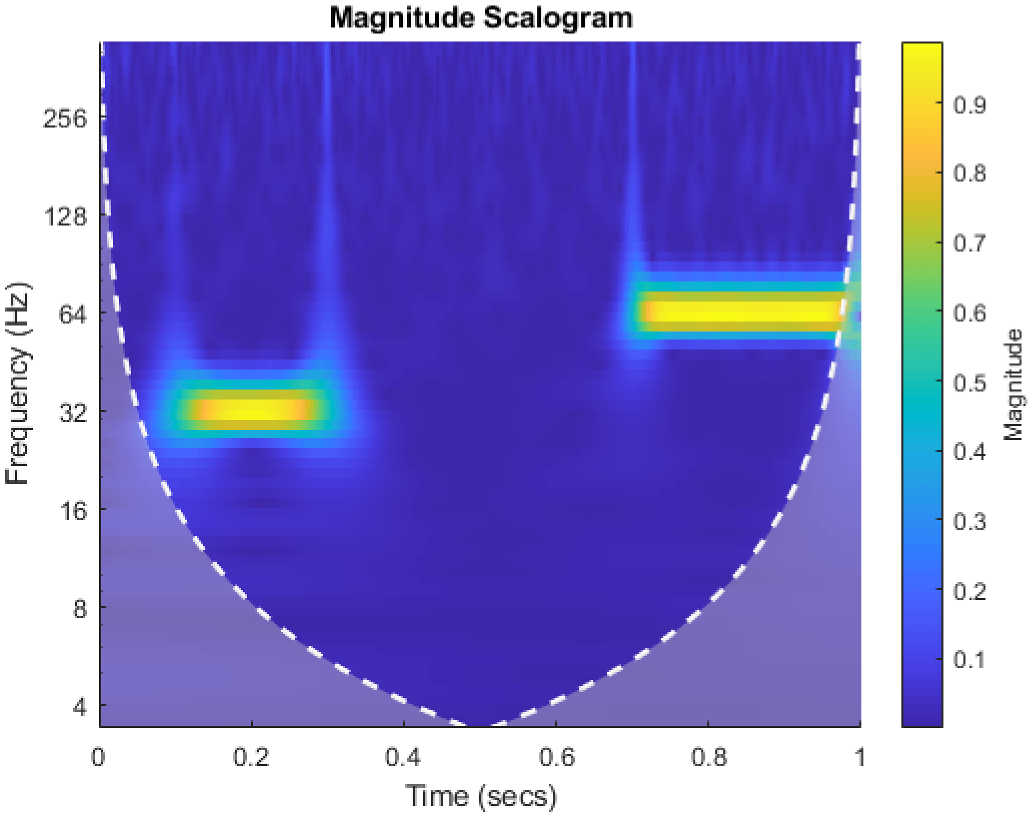
Next, get the handle to the current axes and replace the frequency axis ticks and labels with the following.

```
AX = gca;
AX.YTickLabelMode = "auto";
AX.YTick = freq;
```



In the CWT, frequencies are computed in powers of two. To create the frequency ticks and tick labels in powers of two, you can do the following.

```
newplot
cwt(x,1000)
AX = gca;
freq = 2.^(round(log2(minf)):round(log2(maxf)));
AX.YTickLabelMode = "auto";
AX.YTick = freq;
```

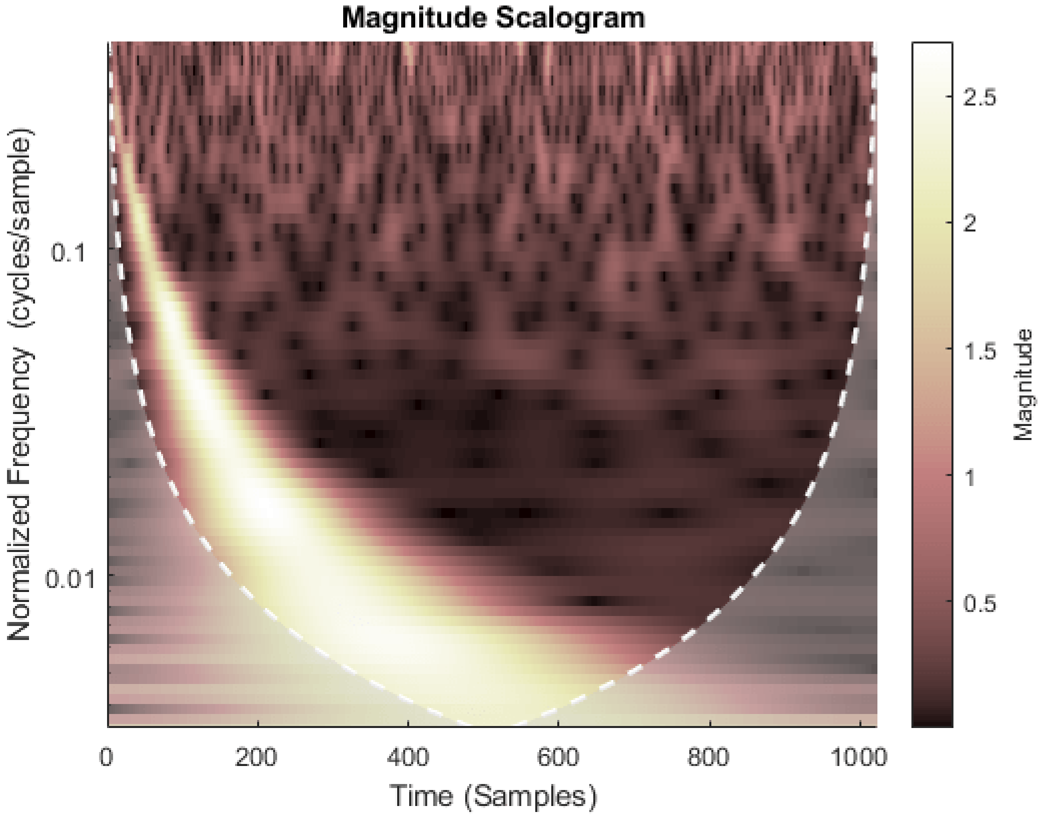


Change Scalogram Coloration

This example shows how to scale scalogram values by maximum absolute value at each level for plotting.

Load in a signal and display the default scalogram. Change the colormap to `pink(240)`.

```
load noisdopp
cwt(noisdopp)
colormap(pink(240))
```



Take the CWT of the signal and obtain the wavelet coefficients and frequencies.

```
[cfs,frq] = cwt(noisdopp);
```

To efficiently find the maximum value of the coefficients at each frequency (level), first transpose the absolute value of the coefficients. Find the minimum value at every level. At each level, subtract the level's minimum value.

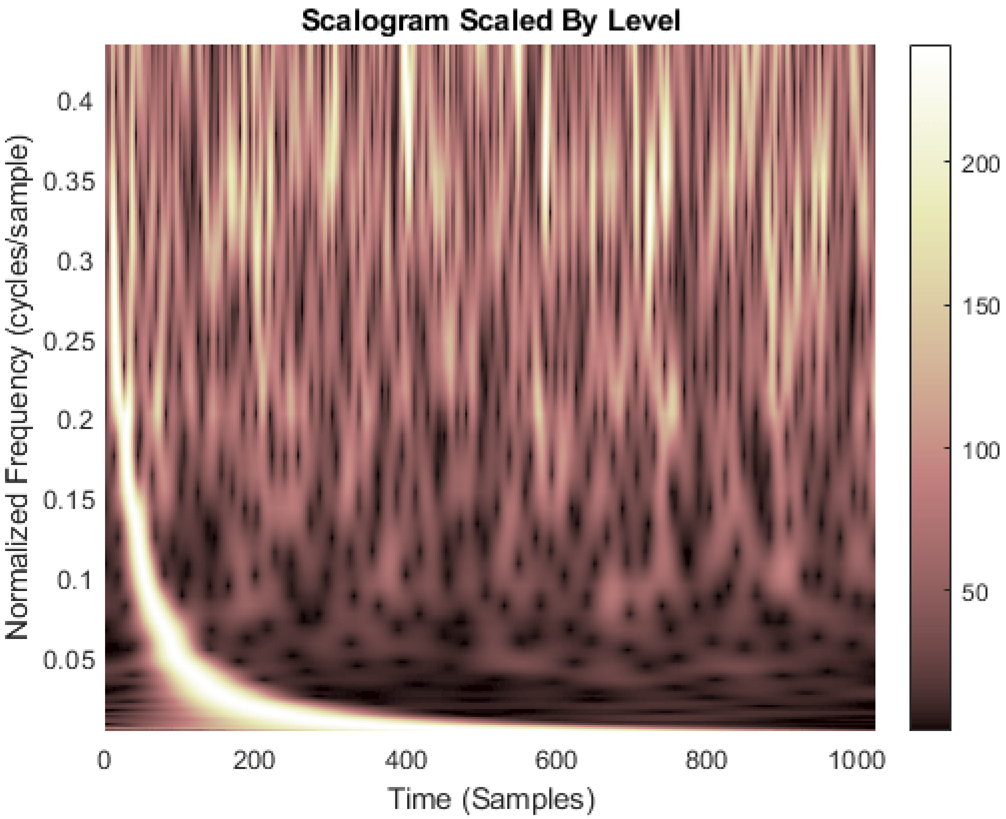
```
tmp1 = abs(cfs);
t1 = size(tmp1,2);
tmp1 = tmp1';
minv = min(tmp1);
tmp1 = (tmp1-minv(ones(1,t1),:));
```

Find the maximum value at every level of `tmp1`. For each level, divide every value by the maximum value at that level. Multiply the result by the number of colors in the colormap. Set equal to 1 all zero entries. Transpose the result.

```
maxv = max(tmp1);
maxvArray = maxv(ones(1,t1),:);
indx = maxvArray<eps;
tmp1 = 240*(tmp1./maxvArray);
tmp2 = 1+fix(tmp1);
tmp2(indx) = 1;
tmp2 = tmp2';
```

Display the result. The scalogram values are now scaled by the maximum absolute value at each level. Frequencies are displayed on a linear scale.

```
t = 0:length(noisdopp)-1;
pcolor(t,frq,tmp2)
shading interp
xlabel("Time (Samples)")
ylabel("Normalized Frequency (cycles/sample)")
title("Scalogram Scaled By Level")
colormap(pink(240))
colorbar
```



Changing the Time-bandwidth Product

This example shows that increasing the time-bandwidth product P^2 of the Morse wavelet creates a wavelet with more oscillations under its envelope. Increasing P^2 narrows the wavelet in frequency.

Create two filter banks. One filter bank has the default `TimeBandwidth` value of 60. The second filter bank has a `TimeBandwidth` value of 10. The `SignalLength` for both filter banks is 4096 samples.

```
sigLen = 4096;
fb60 = cwtfilterbank(SignalLength=sigLen);
fb10 = cwtfilterbank(SignalLength=sigLen,TimeBandwidth=10);
```

Obtain the time-domain wavelets for the filter banks.

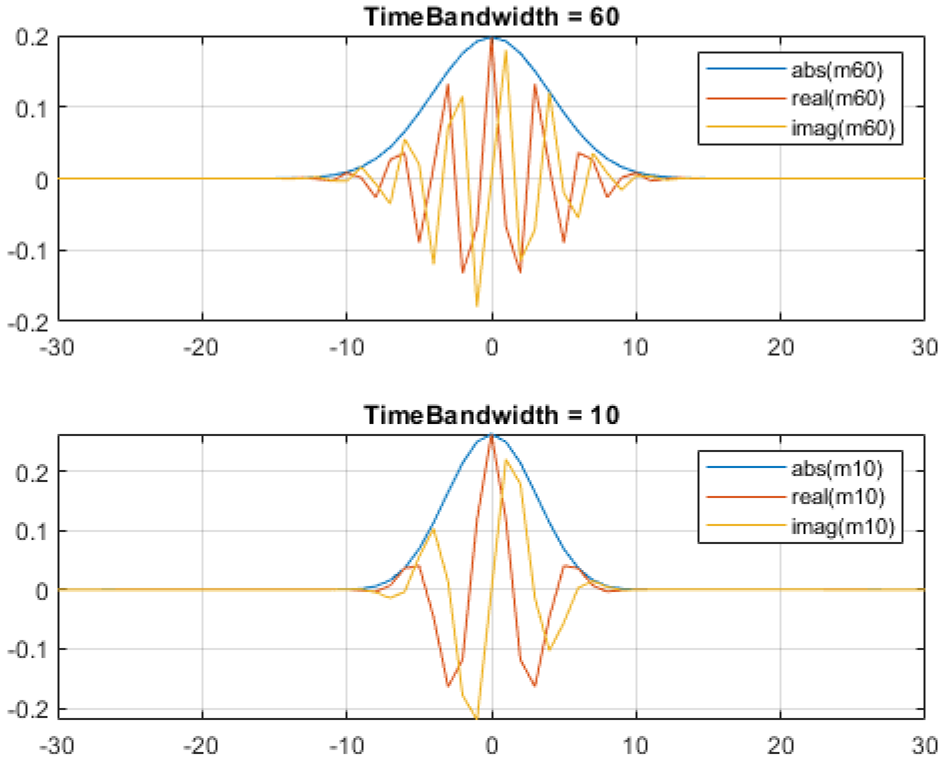
```
[psi60,t] = wavelets(fb60);
[psi10,~] = wavelets(fb10);
```

Use the `scales` function to find the mother wavelet for each filter bank.

```
sca60 = scales(fb60);
sca10 = scales(fb10);
[~,idx60] = min(abs(sca60-1));
[~,idx10] = min(abs(sca10-1));
m60 = psi60(idx60,:);
m10 = psi10(idx10,:);
```

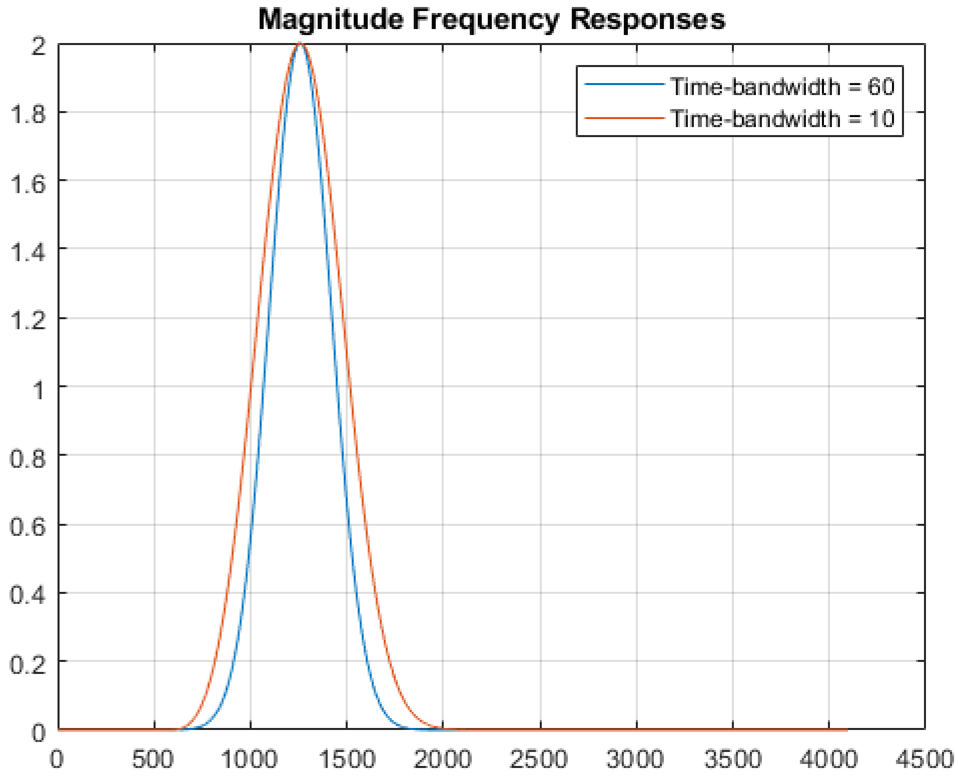
Since the time-bandwidth product is larger for the `fb60` filter bank, verify the `m60` wavelet has more oscillations under its envelope than the `m10` wavelet.

```
subplot(2,1,1)
plot(t,abs(m60))
grid on
hold on
plot(t,real(m60))
plot(t,imag(m60))
hold off
xlim([-30 30])
legend("abs(m60)","real(m60)","imag(m60)")
title("TimeBandwidth = 60")
subplot(2,1,2)
plot(t,abs(m10))
grid on
hold on
plot(t,real(m10))
plot(t,imag(m10))
hold off
xlim([-30 30])
legend("abs(m10)","real(m10)","imag(m10)")
title("TimeBandwidth = 10")
```



Align the peaks of the `m60` and `m10` magnitude frequency responses. Verify the frequency response of the `m60` wavelet is narrower than the frequency response for the `m10` wavelet.

```
cf60 = centerFrequencies(fb60);
cf10 = centerFrequencies(fb10);

m60cFreq = cf60(idx60);
m10cFreq = cf10(idx10);

freqShift = 2*pi*(m60cFreq-m10cFreq);
x10 = m10.*exp(1j*freqShift*(-sigLen/2:sigLen/2-1));

figure
plot([abs(fft(m60)).' abs(fft(x10)).'])
grid on
legend("Time-bandwidth = 60","Time-bandwidth = 10")
title("Magnitude Frequency Responses")
```

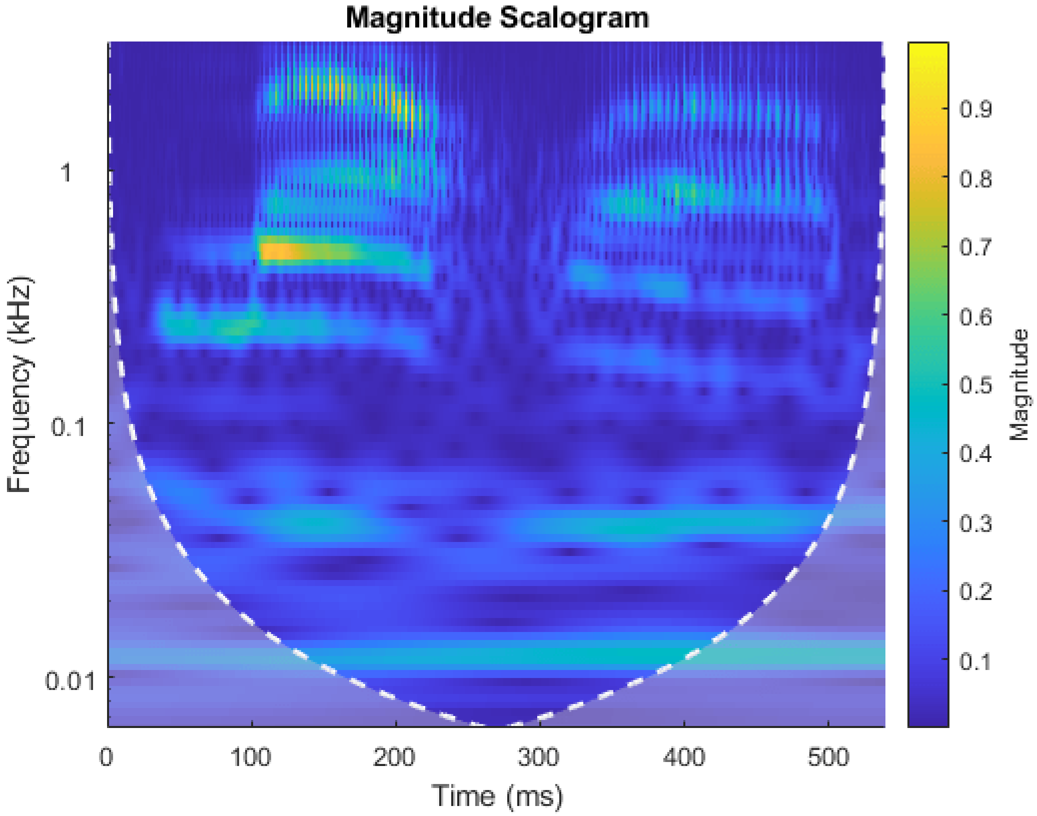


Plot CWT Scalogram in Subplot

This example shows how to plot the CWT scalogram in a figure subplot.

Load the speech sample. The data is sampled at 7418 Hz. Plot the default CWT scalogram.

```
load mtlb
cwt(mtlb,Fs)
```



Obtain the continuous wavelet transform of the signal, and the frequencies of the CWT.

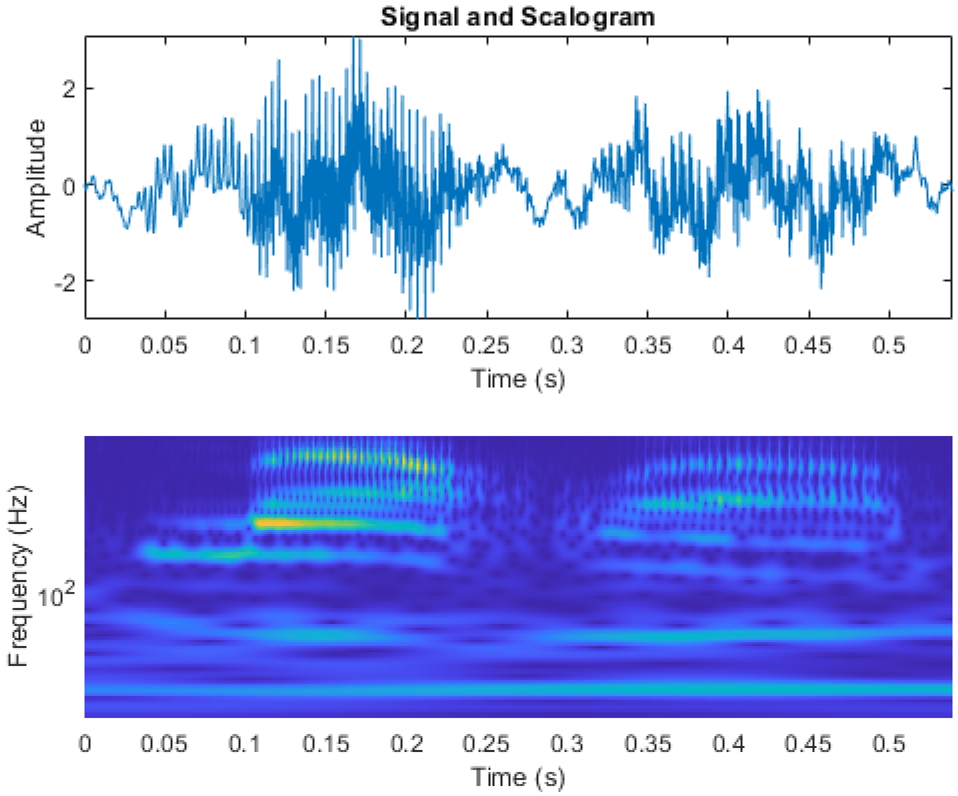
```
[cfs,frq] = cwt(mtlb,Fs);
```

The `cwt` function sets the time and frequency axes in the scalogram. Create a vector representing the sample times.

```
tms = (0:numel(mtlb)-1)/Fs;
```

In a new figure, plot the original signal in the upper subplot and the scalogram in the lower subplot. Plot the frequencies on a logarithmic scale.

```
figure
subplot(2,1,1)
plot(tms,mtlb)
axis tight
title("Signal and Scalogram")
xlabel("Time (s)")
ylabel("Amplitude")
subplot(2,1,2)
surface(tms,frq,abs(cfs))
axis tight
shading flat
xlabel("Time (s)")
ylabel("Frequency (Hz)")
set(gca,"yscale","log")
```



Input Arguments

x — Input signal

real- or complex-valued vector | timetable | gpuArray

Input signal, specified as a real- or complex-valued vector, or a single-variable regularly sampled timetable. The input x must have at least four samples.

The cwt function also accepts GPU array inputs. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

Data Types: single | double

wname — Analytic wavelet

"morse" (default) | "amor" | "bump"

Analytic wavelet used to compute the CWT. Valid options for wname are "morse", "amor", and "bump", which specify the Morse, Morlet (Gabor), and bump wavelet, respectively.

The default Morse wavelet has symmetry parameter gamma (γ) equal to 3 and time-bandwidth product equal to 60.

Data Types: char | string

fs — Sampling frequency

positive scalar

Sampling frequency in hertz, specified as a positive scalar. If you specify `fs`, then you cannot specify `ts`. If `x` is a timetable, you cannot specify `fs`. `fs` is determined from the RowTimes of the timetable.

Data Types: `single` | `double`

`ts` — Sampling period

scalar duration

Sampling period, also known as the time duration, specified as a scalar duration. Valid durations are `years`, `days`, `hours`, `minutes`, and `seconds`. You cannot use calendar durations. If you specify `ts`, then you cannot specify `fs`. If `x` is a timetable, you cannot specify `ts`. `ts` is determined from the RowTimes of the timetable when you set the `PeriodLimits` name-value argument.

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

Data Types: `duration`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `wt = cwt(x,"bump",VoicesPerOctave=10)` returns the CWT of `x` using the bump wavelet and 10 voices per octave.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `wt = cwt(x,"ExtendedSignal",true,"FrequencyLimits",[0.1 0.2])` extends the input signal symmetrically and specifies frequency limits of 0.1 to 0.2 samples per cycle.

`ExtendSignal` — Extend input signal symmetrically

`true` or `1` (default) | `false` or `0`

Option to extend the input signal symmetrically by reflection, specified as one of these:

- `1` (`true`) — Extend symmetrically
- `0` (`false`) — Do not extend symmetrically

If `ExtendSignal` is `false`, the signal is extended periodically. Extending the signal symmetrically can mitigate boundary effects.

Note If you want to invert the CWT using `icwt` with scaling coefficients and approximate synthesis filters, set `ExtendSignal` to `false`.

Data Types: `logical`

`FrequencyLimits` — Frequency limits

two-element scalar vector

Frequency limits to use in the CWT, specified as a two-element vector with positive strictly increasing entries. The first element specifies the lowest peak passband frequency and must be greater than or equal to the product of the wavelet peak frequency in hertz and two time standard deviations divided by the signal length. The second element specifies the highest peak passband frequency and must be less than or equal to the Nyquist frequency. The base-2 logarithm of the ratio of the upper frequency limit, freqMax, to the lower frequency limit, freqMin, must be greater than or equal to 1/NV, where NV is the number of voices per octave:

    log$_2$(freqMax/freqMin) ≥ 1/NV.

If you specify frequency limits outside the permissible range, cwt truncates the limits to the minimum and maximum valid values. Use cwtfreqbounds to determine frequency limits for different parameterizations of the CWT. For complex-valued signals, (-1) × flimits is used for the anti-analytic part, where flimits is the vector specified by FrequencyLimits.

Example: wt = cwt(x,1000,VoicesPerOctave=10,FrequencyLimits=[80 90])

Data Types: double

PeriodLimits — Period limits

two-element duration array

Period limits to use in the CWT, specified as a two-element duration array with strictly increasing positive entries. The first element must be greater than or equal to 2 × ts where ts is the sampling period. The maximum period cannot exceed the signal length divided by the product of two time standard deviations of the wavelet and the wavelet peak frequency. The base-2 logarithm of the ratio of the minimum period, minP, to the maximum period, maxP, must be less than or equal to -1/NV, where NV is the number of voices per octave:

    log$_2$(pMin/pMax) ≤ -1/NV.

If you specify period limits outside the permissible range, cwt truncates the limits to the minimum and maximum valid values. Use cwtfreqbounds to determine period limits for different parameterizations of the wavelet transform. For complex-valued signals, (-1) × plimits is used for the anti-analytic part, where plimits is the vector specified by PeriodLimits.

Example: wt = cwt(x,seconds(0.1),VoicesPerOctave=10,PeriodLimits=[seconds(0.2) seconds(3)])

Data Types: duration

VoicesPerOctave — Number of voices per octave

10 (default) | integer from 1 to 48

Number of voices per octave to use for the CWT, specified as an integer from 1 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.

TimeBandwidth — Time-bandwidth product of the Morse wavelet

60 (default) | scalar greater than or equal to 3 and less than or equal to 120

Time-bandwidth product of the Morse wavelet, specified as a scalar greater than or equal to 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 sqrt(TimeBandwidth/2). The standard deviation of the Morse wavelet in frequency is approximately 1/2 × sqrt(2/TimeBandwidth).

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

In the notation of “Morse Wavelets”, TimeBandwidth is P^2.

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

Symmetry and time-bandwidth product of the Morse wavelet, specified as 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 or equal to 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.

For more information, see “Morse Wavelets”.

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

`FilterBank` — CWT filter bank
`cwtfilterbank` object

CWT filter bank to use to compute the CWT, specified as a `cwtfilterbank` object. If you set `FilterBank`, you cannot specify any other options. All options for the computation of the CWT are defined as properties of the filter bank. For more information, see `cwtfilterbank`.

If `x` is a timetable, the sampling frequency or sampling period in `fb` must agree with the sampling frequency or sampling period determined by the `RowTimes` of the timetable.

Example: `wt = cwt(x,FilterBank=cfb)`

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 1-166 for information on how the scales are determined.

Data Types: `single` | `double`

`f` — Scale-to-frequency conversions
vector

Scale-to-frequency conversions of the CWT, returned as a vector. If you specify a sampling frequency, `fs`, then `f` is in hertz. If you do not specify `fs`, `cwt` returns `f` in cycles per sample. If the input `x` is complex, the scale-to-frequency conversions apply to both pages of `wt`.

`period` — Scale-to-period conversions
array

Scale-to-period conversions, returned as an array of durations with the same Format property as `ts`. Each row corresponds to a period. If the input `x` is complex, the scale-to-period conversions apply to both pages of `wt`.

`coi` — Cone of influence
array of real numbers | array of durations

Cone of influence for the CWT. If you specify a sampling frequency, `fs`, the cone of influence is in hertz. If you specify a scalar duration, `ts`, the cone of influence is an array of durations with the same Format property as `ts`. If the input `x` is complex, the cone of influence applies to both pages of `wt`.

The cone of influence indicates where edge effects occur in the CWT. Due to the edge effects, give less credence to areas that are outside or overlap the cone of influence. For additional information, see "Boundary Effects and the Cone of Influence".

`fb` — CWT filter bank
`cwtfilterbank` object

CWT filter bank used in the CWT, returned as a `cwtfilterbank` object. See `cwtfilterbank`.

`scalingcfs` — Scaling coefficients
real- or complex-valued vector

Scaling coefficients for the CWT, returned as a real- or complex-valued vector. The length of `scalingcfs` is equal to the length of the input `x`.

More About

Analytic Wavelets

Analytic wavelets are complex-valued wavelets whose Fourier transform vanish for negative frequencies. Analytic wavelets are a good choice when doing time-frequency analysis with the CWT. Because the wavelet coefficients are complex-valued, the coefficients provide phase and amplitude information of the signal being analyzed. Analytic wavelets are well suited for studying how the frequency content in real world nonstationary signals evolves as a function of time.

Analytic wavelets are almost exclusively based on rapidly decreasing functions. If $\psi(t)$ is an analytic rapidly decreasing function in time, then its Fourier transform $\widehat{\psi}(\omega)$ is a rapidly decreasing function in frequency and is small outside of some interval $\alpha < \omega < \beta$ where $0 < \alpha < \beta$. Orthogonal and biorthogonal wavelets are typically designed to have compact support in time. Wavelets with compact support in time have relatively poorer energy concentration in frequency than wavelets which rapidly decrease in time. Most orthogonal and biorthogonal wavelets are not symmetric in the Fourier domain.

If your goal is to obtain a joint time-frequency representation of your signal, we recommend you use `cwt` or `cwtfilterbank`. Both functions support the following analytic wavelets:

- Morse Wavelet Family (default)
- Analytic Morlet (Gabor) Wavelet
- Bump

If you want to do time-frequency analysis using orthogonal or biorthogonal wavelets, we recommend `modwpt`.

When using wavelets for time-frequency analysis, you usually convert scales to frequencies or periods to interpret results. `cwt` and `cwtfilterbank` do the conversion. You can obtain the corresponding scales associated by using `scales` on the optional `cwt` output argument `fb`.

For more information regarding Morse wavelets, see “Morse Wavelets”. For guidance on how to choose the wavelet that is right for your application, see “Choose a Wavelet”.

Tips

- When performing multiple CWTs, for example inside a for-loop, the recommended workflow is to first create a `cwtfilterbank` object and then use the `wt` object function. This workflow minimizes overhead and maximizes performance. See “Using CWT Filter Bank on Multiple Time Series” on page 1-149.

Algorithms

Minimum Scale

To determine the minimum scale, find the peak frequency ω_x of the base wavelet. For Morse wavelets, dilate the wavelet so that the Fourier transform of the wavelet at π radians is equal to 10% of the peak frequency. 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 $\sqrt{\frac{P^2}{2}}$, where P^2 is the time-bandwidth product. The standard deviation in frequency, σ_f, is approximately $\frac{1}{2}\sqrt{\frac{2}{P^2}}$. 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 ranges from 0 to the largest scale. For a specific small scale, s_0:

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

Converting to log2:

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

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

Therefore, the maximum scale is

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

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:

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

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 1-143.

Compatibility Considerations

`'NumOctaves'` name-value argument will be removed

Not recommended starting in R2018a

The `NumOctaves` name-value argument will be removed in a future release. Use either:

- Name-value argument `FrequencyLimits` to modify the frequency range of the CWT.
- Name-value argument `PeriodLimits` to modify the period range of the CWT.

See `cwtfreqbounds` for additional information.

References

[1] Lilly, J. M., and S. C. Olhede. "Generalized Morse Wavelets as a Superfamily of Analytic Wavelets." *IEEE Transactions on Signal Processing* 60, no. 11 (November 2012): 6036–6041. https://doi.org/10.1109/TSP.2012.2210890.

[2] Lilly, J.M., and S.C. Olhede. "Higher-Order Properties of Analytic Wavelets." *IEEE Transactions on Signal Processing* 57, no. 1 (January 2009): 146–160. https://doi.org/10.1109/TSP.2008.2007607.

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

[4] Lilly, Jonathan M. "Element Analysis: A Wavelet-Based Method for Analysing Time-Localized Events in Noisy Time Series." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473, no. 2200 (April 30, 2017): 20160776. https://doi.org/10.1098/rspa.2016.0776.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Single- and double-precision input signal are supported. The precision must be set at compile time.
- Timetable input signal is not supported.
- Only analytic Morse (`'morse'`) and Morlet (`'amor'`) wavelets are supported.
- The following input arguments are not supported: Sampling period (`ts`), `PeriodLimits` name-value pair, `NumOctave` name-value pair, and `FilterBank` name-value pair.
- Scaling coefficient output and filter bank output are not supported.
- Plotting is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Run MATLAB Functions on a GPU" (Parallel Computing Toolbox).

See Also

Apps

Wavelet Time-Frequency Analyzer

Functions

`cwtfilterbank` | `icwt` | `cwtfreqbounds`

Topics

"Practical Introduction to Continuous Wavelet Analysis"
"Using Wavelet Time-Frequency Analyzer App"

“Continuous and Discrete Wavelet Transforms”
“CWT-Based Time-Frequency Analysis”
“Boundary Effects and the Cone of Influence”
“Morse Wavelets”
“Time-Frequency Gallery”

Introduced in R2016b

cwtfilterbank

Continuous wavelet transform filter bank

Description

Use `cwtfilterbank` to create a continuous wavelet transform (CWT) filter bank. The default wavelet used in the filter bank is the analytic Morse (3,60) wavelet. You can vary the time-bandwidth and symmetry parameters for the Morse wavelets, to tune the Morse wavelet for your needs. You can also use the analytic Morlet (Gabor) wavelet or bump wavelet. When analyzing multiple signals in time-frequency, for improved computational efficiency, you can precompute the filters once and then pass the filter bank as input to `cwt`. With the filter bank, you can visualize wavelets in time and frequency. You can also create filter banks with specific frequency or period ranges, and measure 3-dB bandwidths. You can determine the quality factor for the wavelets in the filter bank.

Creation

Syntax

```
fb = cwtfilterbank
fb = cwtfilterbank(Name=Value)
```

Description

`fb = cwtfilterbank` creates a continuous wavelet transform (CWT) filter bank `fb`. The filters are normalized so that the peak magnitudes for all passbands are approximately equal to 2. The default filter bank is designed for a signal with 1024 samples. The default filter bank uses the analytic Morse (3,60) wavelet. The filter bank uses the default scales: approximately 10 wavelet bandpass filters per octave (10 voices per octave). The highest-frequency passband is designed so that the magnitude falls to half the peak value at the Nyquist frequency.

As implemented, the CWT uses L1 normalization. With L1 normalization, equal amplitude oscillatory components at different scales have equal magnitude in the CWT. L1 normalization provides a more accurate representation of the signal. The amplitudes of the oscillatory components agree with the amplitudes of the corresponding wavelet coefficients. See “Sinusoid and Wavelet Coefficient Amplitudes” on page 1-180.

`fb` can be used as input for `cwt`.

`fb = cwtfilterbank(Name=Value)` creates a CWT filter bank `fb` with “Properties” on page 1-171 using one or more name-value arguments. Properties can be specified in any order as `Name1=Value1,...,NameN=ValueN`.

Note You cannot change a property value of an existing filter bank. For example, if you have a filter bank `fb` with a `SignalLength` of 2000, you must create a second filter bank `fb2` to process a signal with 2001 samples. You cannot assign a different `SignalLength` to `fb`.

Properties

`SignalLength` — Length of the signal

1024 (default) | positive integer ≥ 4

Length of the signal, specified as a positive integer. The signal must have at least four samples.

Example: `fb = cwtfilterbank(SignalLength=1700)`

Data Types: `double`

`Wavelet` — Analysis wavelet

`"Morse"` (default) | `"amor"` | `"bump"`

Analysis wavelet used in the filter bank, specified as `"Morse"`, `"amor"`, or `"bump"`. These strings specify the analytic Morse, Morlet (Gabor), and bump wavelet, respectively. The default wavelet is the analytic Morse (3,60) wavelet.

By default, for Morse wavelets, the frequency response decays to 50% of the peak magnitude at the Nyquist. For the Morlet and bump wavelets, the frequency response decays to 10% of the peak magnitude. You can change the decay percentage by setting the filter bank `FrequencyLimits` property. See `cwtfreqbounds`.

For Morse wavelets, you can also parameterize the wavelet using the `TimeBandwidth` or `WaveletParameters` properties.

Example: `fb = cwtfilterbank(SignalLength=1700,wavelet="bump")`

`VoicesPerOctave` — Number of voices per octave

`10` (default) | integer between 1 and 48

Number of voices per octave to use for the CWT, specified as an integer between 1 and 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.

You can use `cwtfreqbounds` to determine the frequency limits of the wavelet filter bank. The frequency limits depend on parameters such as the energy spread of the wavelet, number of voices per octave, signal length, and sampling frequency.

Data Types: `single` | `double`

`SamplingFrequency` — Sampling frequency in hertz

`1` (default) | positive scalar

Sampling frequency in hertz, specified as a positive scalar. If unspecified, frequencies are in cycles/sample and the Nyquist frequency is ½. To specify scales in periods, use the `SamplingPeriod` and `PeriodLimits` name-value arguments.

You cannot specify both the `SamplingFrequency` and `SamplingPeriod` properties.

Example: `fb = cwtfilterbank(SamplingFrequency=5,wavelet="amor")`

Data Types: `single` | `double`

`FrequencyLimits` — Frequency limits

two-element scalar vector

Frequency limits of the wavelet filter bank, specified as a two-element vector with positive strictly increasing entries.

- The first element specifies the lowest peak passband frequency. The frequency must be greater than or equal to the product of the wavelet peak frequency in hertz and two time standard deviations divided by the signal length.
- The second element specifies the lowest peak passband frequency. The high frequency limit must be less than or equal to the Nyquist.
- The base-2 logarithm of the ratio of the high frequency limit, fMax, to the low frequency limit, fMin, must be greater than or equal to 1/NV, where NV is the number of voices per octave: $\log_2(\mathrm{fMax}/\mathrm{fMin}) \geq 1/\mathrm{NV}$.

If you specify frequency limits outside the permissible range, cwtfilterbank truncates the limits to the minimum and maximum values. Use cwtfreqbounds to determine frequency limits for different parametrizations of the wavelet transform.

If using a sampling period in the filter bank, you cannot specify the FrequencyLimits property.

Example: If fb = cwtfilterbank(SignalLength=1000,SamplingFrequency=1000,FrequencyLimits=[90 100]), then $\log_2(100/90) \geq 1/\mathrm{fb.VoicesPerOctave}$.

Data Types: double

SamplingPeriod — Sampling period
duration scalar

Sampling period, specified as a duration scalar. You cannot specify both the SamplingFrequency and SamplingPeriod properties.

Example: fb = cwtfilterbank(SamplingPeriod=seconds(0.5))

Data Types: duration

PeriodLimits — Period limits
two-element duration array

Period limits of the wavelet filter bank, specified as a two-element duration array with positive strictly increasing entries.

- The first element of PeriodLimits specifies the largest peak passband frequency and must be greater than or equal to twice the SamplingPeriod.
- The maximum period cannot exceed the signal length divided by the product of two time standard deviations of the wavelet and the wavelet peak frequency.
- The base-2 logarithm of the ratio of the minimum period, minP, to the maximum period, maxP, must be less than or equal to -1/NV, where NV is the number of voices per octave: $\log_2(\mathrm{minP}/\mathrm{maxP}) \leq -1/\mathrm{NV}$.

If you specify period limits outside the permissible range, cwtfilterbank truncates the limits to the minimum and maximum values. Use cwtfreqbounds to determine period limits for different parametrizations of the wavelet transform.

If using a sampling frequency in the filter bank, you cannot specify the PeriodLimits property.

Example: If `fb = cwtfilterbank(SignalLength=1000,SamplingPeriod=seconds(0.1),PeriodLimits=[seconds(0.2) seconds(3)])`, then $\log_2$`(0.2/3) ≤ -1/fb.VoicesPerOctave`.

Data Types: `duration`

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

Time-bandwidth product for the Morse wavelet, specified as a positive scalar. The symmetry (gamma) of the Morse wavelet is assumed to be 3. The larger the time-bandwidth parameter, the more spread out the wavelet is in time and narrower the wavelet is in frequency. The standard deviation of the Morse wavelet in time is approximately `sqrt(TimeBandwidth/2)`. The standard deviation in frequency is approximately `1/2*sqrt(2/TimeBandwidth)`.

This property specifies the time-bandwidth product of the Morse wavelet with the symmetry parameter (gamma) fixed at 3. `TimeBandwidth` is a positive number greater than or equal to 3 and less than or equal to 120.

The larger the time-bandwidth product, the more spread out the wavelet is in time and narrower the wavelet is in frequency. The standard deviation of the Morse wavelet in time is approximately `sqrt(TimeBandwidth/2)`. The standard deviation in frequency is approximately `1/2 × sqrt(2/TimeBandwidth)`. See “Generalized Morse and Analytic Morlet Wavelets” on page 1-184.

The `TimeBandwidth` and `WaveletParameters` properties cannot both be specified.

In the notation of “Morse Wavelets”, `TimeBandwidth` is P^2.

Example: `'TimeBandwidth',20`

Data Types: `double`

`WaveletParameters` — Morse wavelet parameters
`[3,60]` (default) | two-element vector of scalars

Morse wavelet parameters, specified as a two-element vector. The first element is the symmetry parameter (gamma), which must be greater than or equal to 1. The second element is the time-bandwidth product, which must be greater than or equal to gamma. The ratio of the time-bandwidth product to gamma cannot exceed 40.

When gamma is equal to 3, the Morse wavelet is perfectly symmetric in the frequency domain. The skewness is equal to 0. Values of gamma greater than 3 result in positive skewness, while values of gamma less than 3 result in negative skewness.

For more information, see “Morse Wavelets”.

The `WaveletParameters` and `TimeBandwidth` name-value arguments cannot both be specified.

Example: `fb = cwtfilterbank(WaveletParameters=[4,20])`

`Boundary` — Boundary extension
`"reflection"` (default) | `"periodic"`

Boundary extension of signal, specified as either `"reflection"` or `"periodic"`. Determines how the data is treated at the boundary.

Note If you intend to invert the CWT using the dual frame, or approximate synthesis filters, set `Boundary` to `"periodic"`.

Example: `fb = cwtfilterbank(Boundary="periodic")`

Object Functions

wt	Continuous wavelet transform with filter bank
freqz	CWT filter bank frequency responses
timeSpectrum	Time-averaged wavelet spectrum
scaleSpectrum	Scale-averaged wavelet spectrum
wavelets	CWT filter bank time-domain wavelets
scales	CWT filter bank scales
waveletsupport	CWT filter bank time supports
qfactor	CWT filter bank quality factor
powerbw	CWT filter bank 3 dB bandwidths
centerFrequencies	CWT filter bank bandpass center frequencies
centerPeriods	CWT filter bank bandpass center periods

Examples

Continuous Wavelet Transform Filter Bank

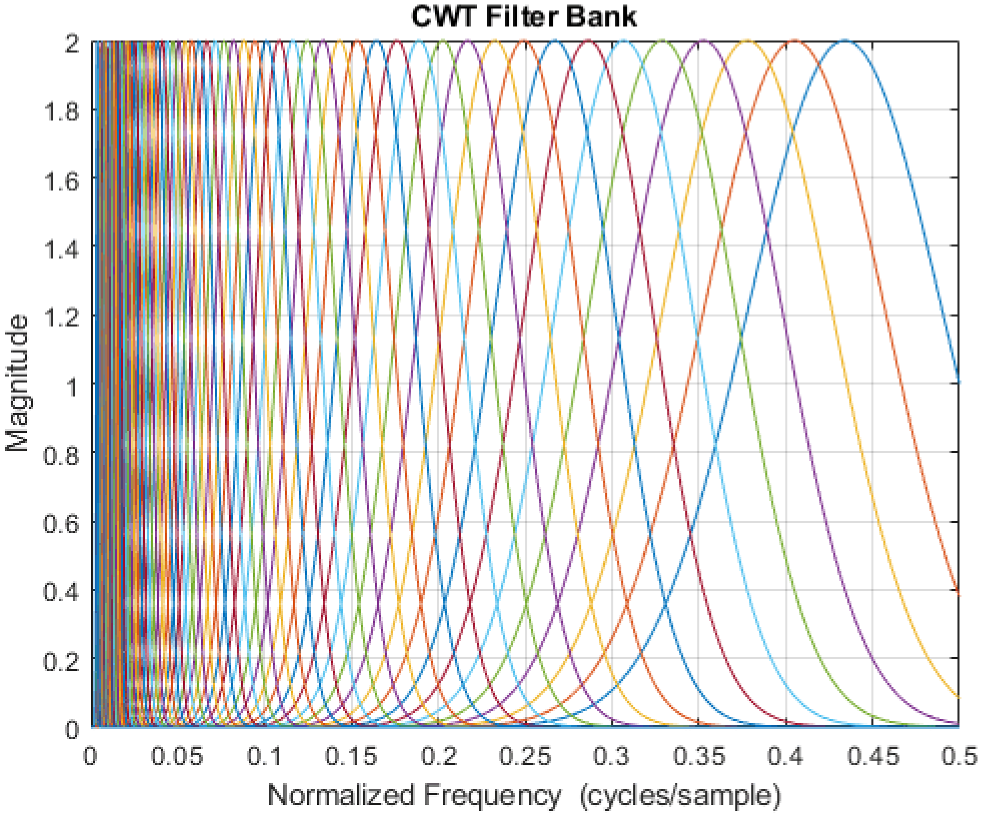
Create a continuous wavelet transform filter bank.

```
fb = cwtfilterbank
```

```
fb = 
  cwtfilterbank with properties:

      VoicesPerOctave: 10
              Wavelet: 'Morse'
    SamplingFrequency: 1
       SamplingPeriod: []
         PeriodLimits: []
         SignalLength: 1024
      FrequencyLimits: []
        TimeBandwidth: 60
    WaveletParameters: []
             Boundary: 'reflection'
```

Plot the magnitude frequency response.

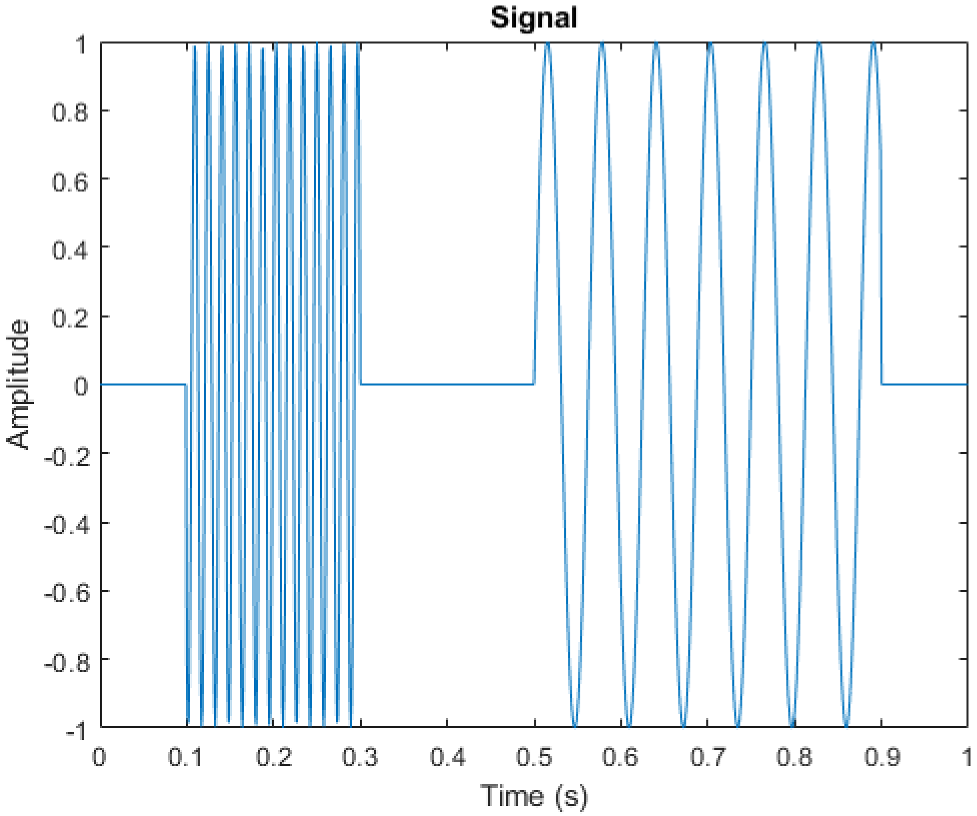
```
freqz(fb)
```



Frequency Resolution of Continuous Wavelet Transform Filter Banks

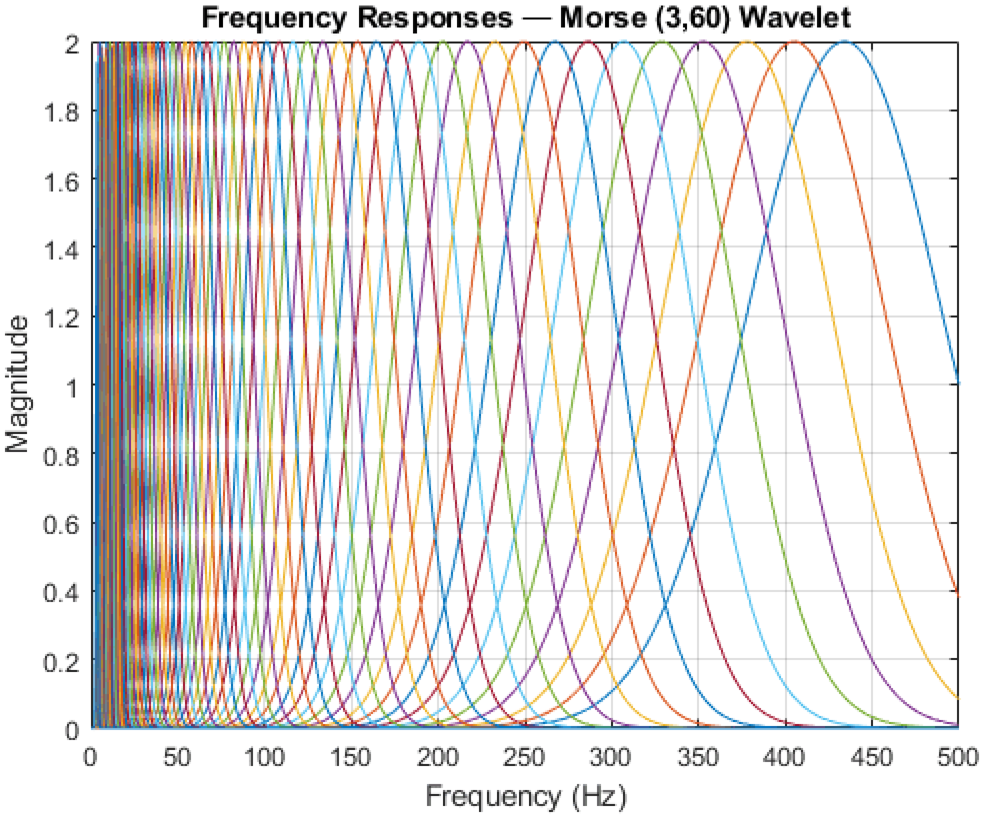
Create two sine waves with frequencies of 16 and 64 Hz. The data is sampled at 1000 Hz. Plot the signal.

```
Fs = 1e3;
t = 0:1/Fs:1-1/Fs;
x = cos(2*pi*64*t).*(t>=0.1 & t<0.3)+ ...
    sin(2*pi*16*t).*(t>=0.5 & t<0.9);
plot(t,x)
title("Signal")
xlabel("Time (s)")
ylabel("Amplitude")
```



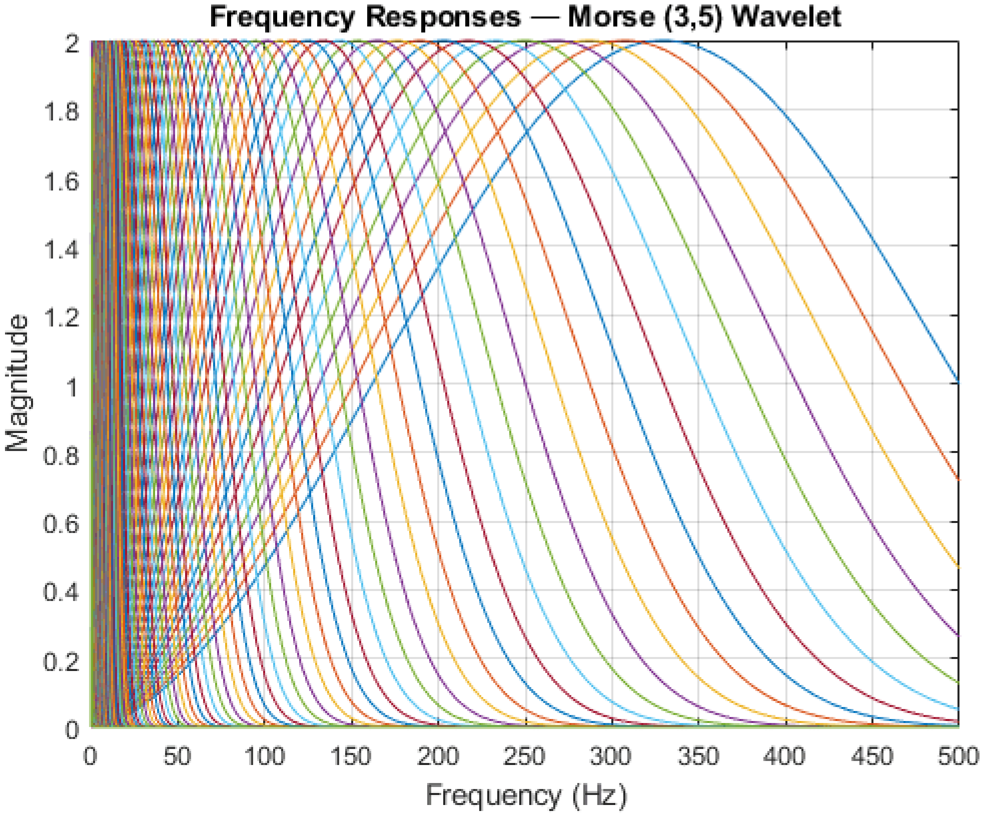
Create a CWT filter bank for the signal. Plot the frequency responses of the wavelets in the filter bank.

```
fb = cwtfilterbank(SignalLength=numel(t),SamplingFrequency=Fs);
figure
freqz(fb)
title("Frequency Responses — Morse (3,60) Wavelet")
```



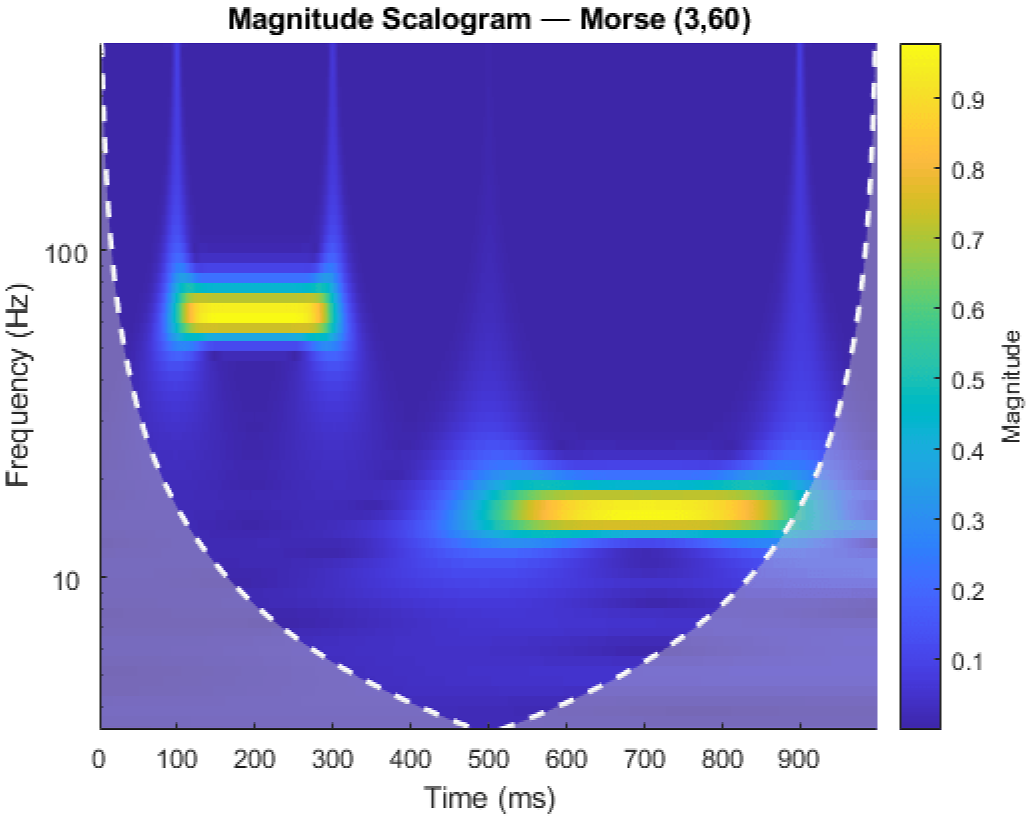
The analytic Morse (3,60) wavelet is the default wavelet in the filter bank. The wavelet has a time-bandwidth product equal to 60. Create a second filter bank identical to the first filter bank but instead uses the analytic Morse (3,5) wavelet. Plot the frequency responses of the wavelets in the second filter bank.

```
fb3x5 = cwtfilterbank(SignalLength=numel(t),SamplingFrequency=Fs,...
    TimeBandwidth=5);
figure
freqz(fb3x5)
title("Frequency Responses — Morse (3,5) Wavelet")
```

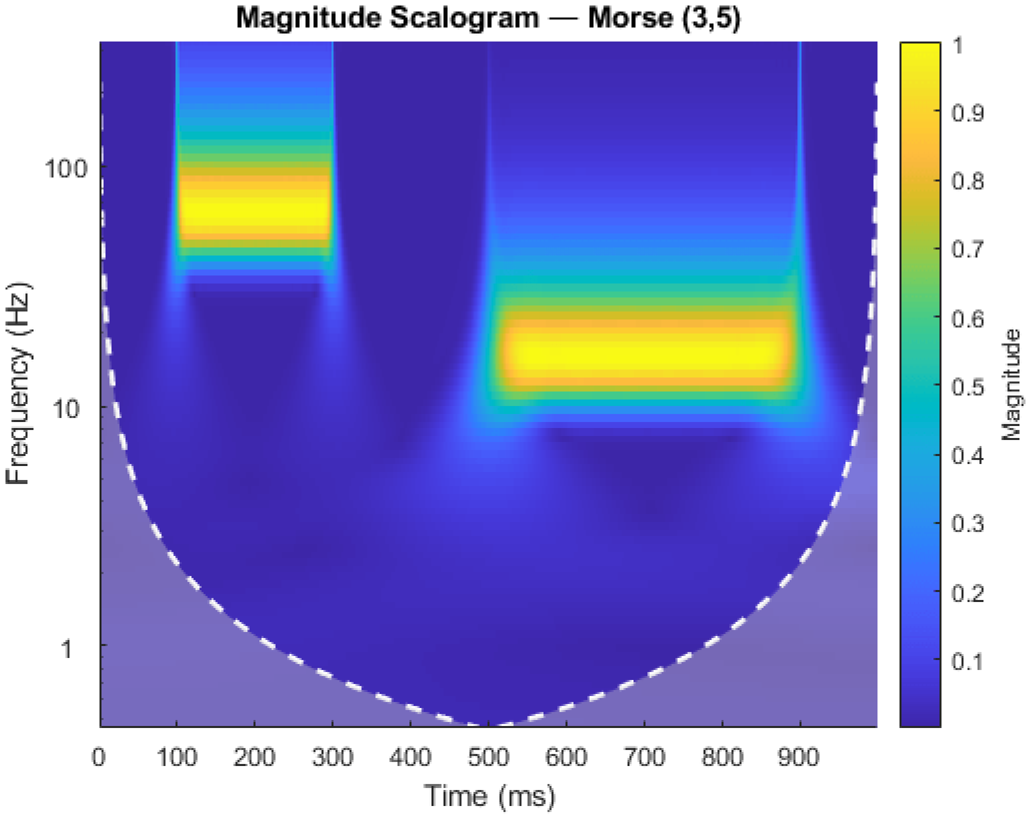


Observe that the frequency responses are wider than in the first filter bank. The Morse (3,60) wavelet is better localized in frequency than the Morse (3,5) wavelet. Apply each filter bank to the signal and plot the resulting scalograms. Observe that the Morse (3,60) wavelet has better frequency resolution than the Morse (3,5) wavelet.

```
figure
cwt(x,FilterBank=fb)
title("Magnitude Scalogram — Morse (3,60)")
```



```
figure
cwt(x,FilterBank=fb3x5)
title("Magnitude Scalogram — Morse (3,5)")
```

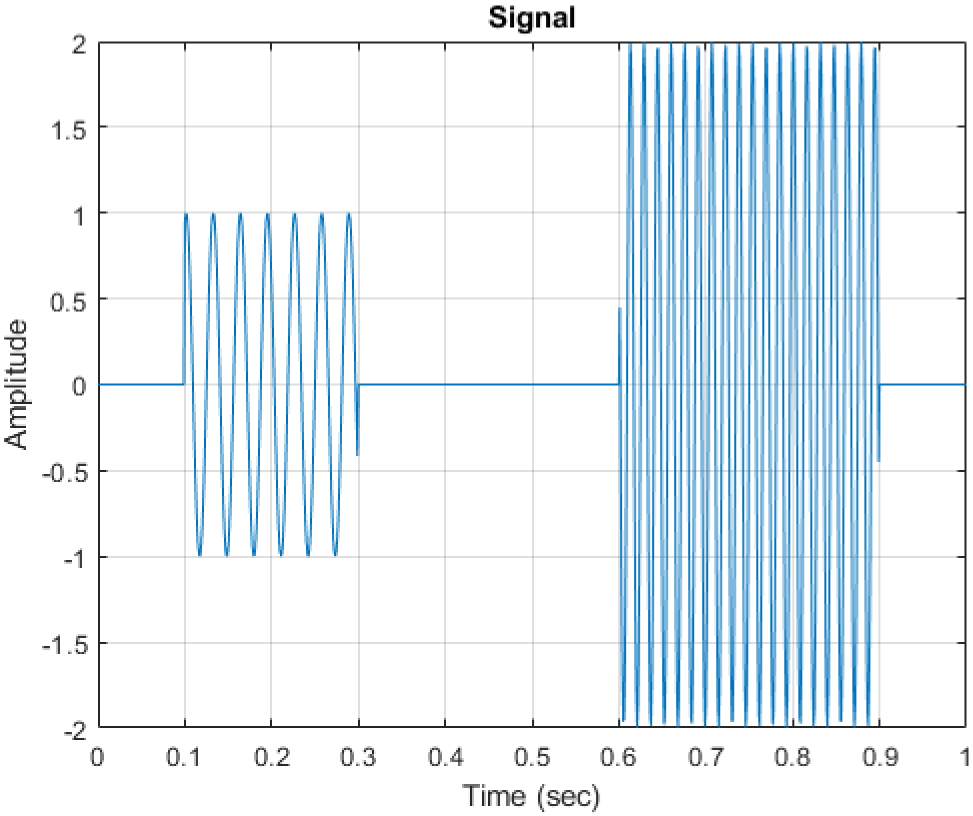


Sinusoid and Wavelet Coefficient Amplitudes

This example shows that the amplitudes of oscillatory components in a signal agree with the amplitudes of the corresponding wavelet coefficients.

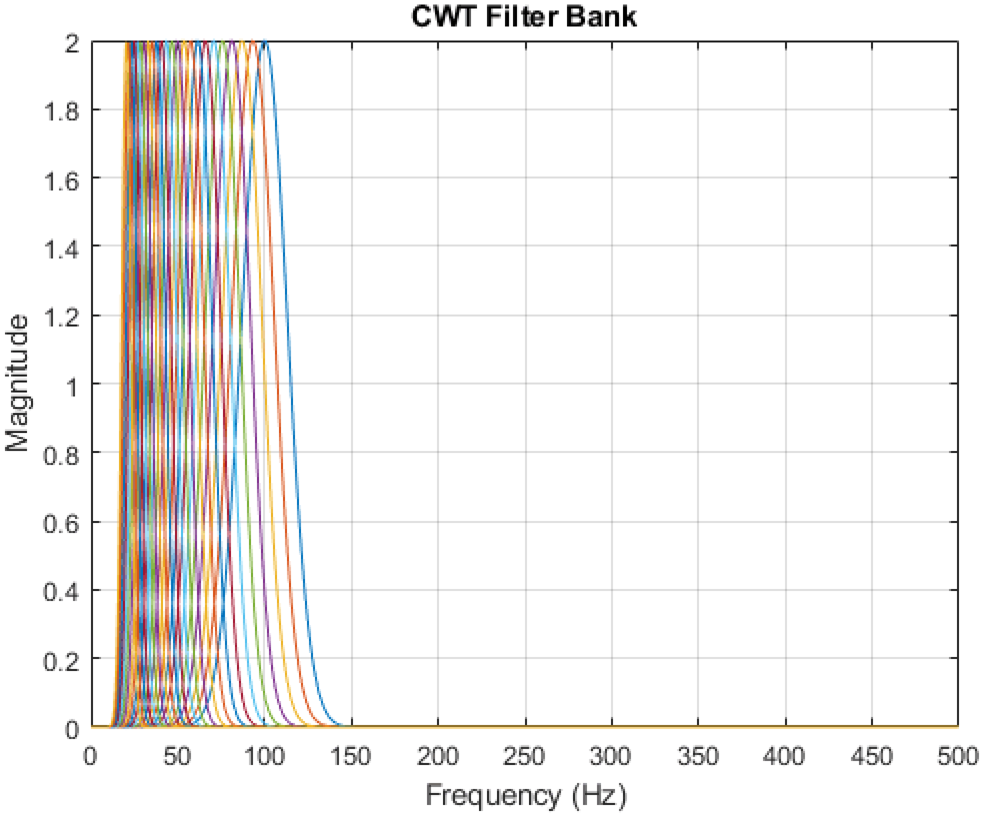
Create a signal composed of two sinusoids with disjoint support in time. One sinusoid has a frequency of 32 Hz and amplitude equal to 1. The other sinusoid has a frequency of 64 Hz and amplitude equal to 2. The signal is sampled for one second at 1000 Hz. Plot the signal.

```
frq1 = 32;
amp1 = 1;
frq2 = 64;
amp2 = 2;

Fs = 1e3;
t = 0:1/Fs:1;
x = amp1*sin(2*pi*frq1*t).*(t>=0.1 & t<0.3)+...
    amp2*sin(2*pi*frq2*t).*(t>0.6 & t<0.9);

plot(t,x)
grid on
xlabel("Time (sec)")
ylabel("Amplitude")
title("Signal")
```



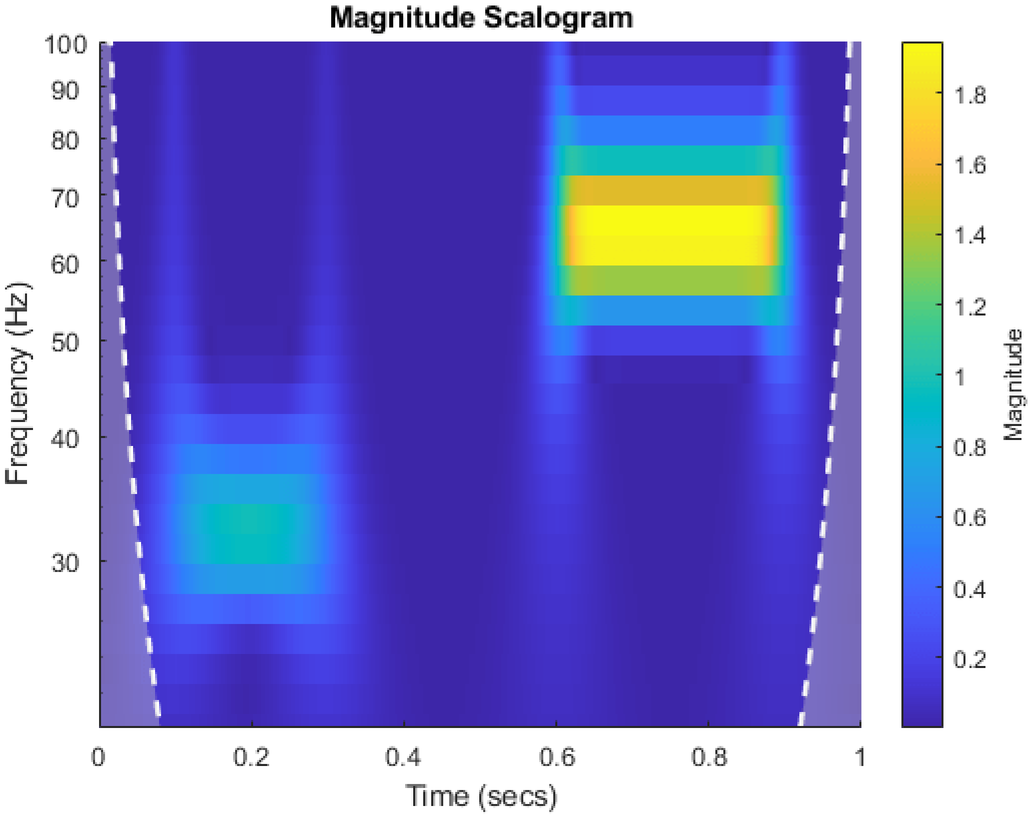
Create a CWT filter bank that can be applied to the signal. Since the signal component frequencies are known, set the frequency limits of the filter bank to a narrow range that includes the known frequencies. To confirm the range, plot the magnitude frequency responses for the filter bank.

```
fb = cwtfilterbank(SignalLength=numel(x),SamplingFrequency=Fs,...
    FrequencyLimits=[20 100]);
freqz(fb)
```

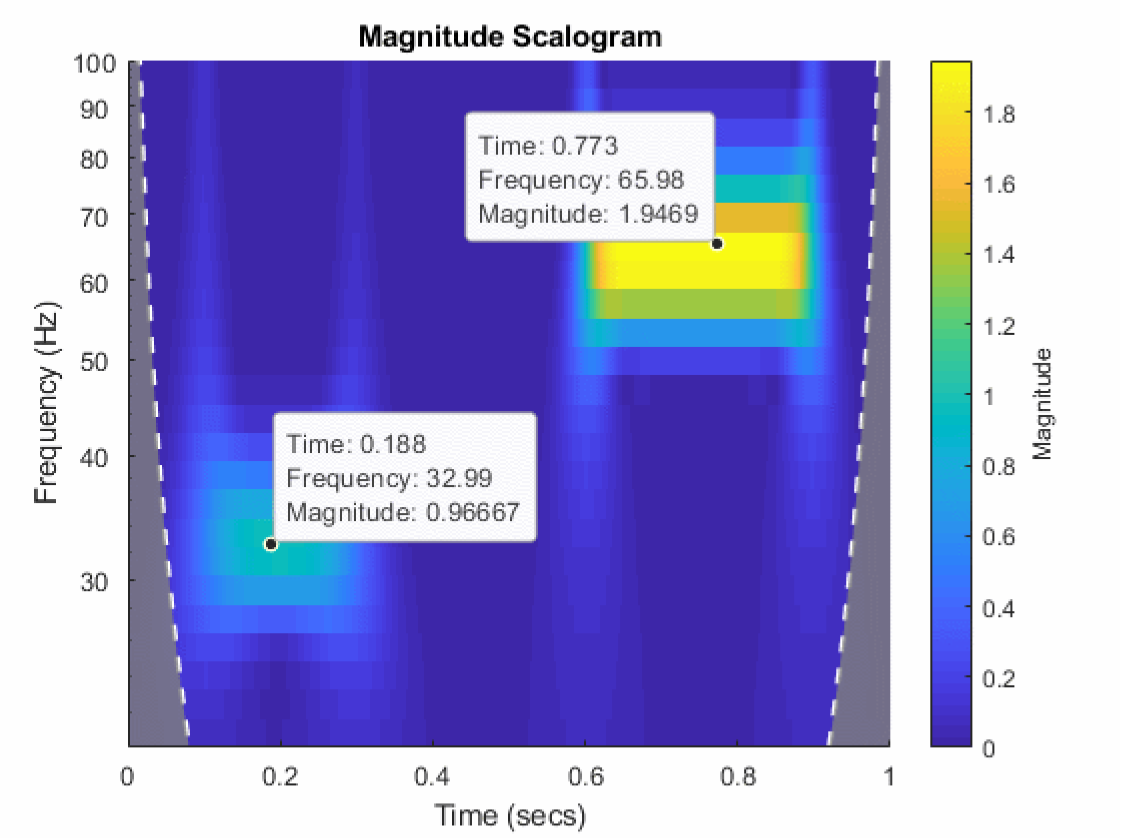


Use `cwt` and the filter bank to plot the scalogram of the signal.

```
cwt(x,FilterBank=fb)
```



Use a data cursor to confirm that the amplitudes of the wavelet coefficients are essentially equal to the amplitudes of the sinusoidal components. Your results should be similar to the ones in the following figure.



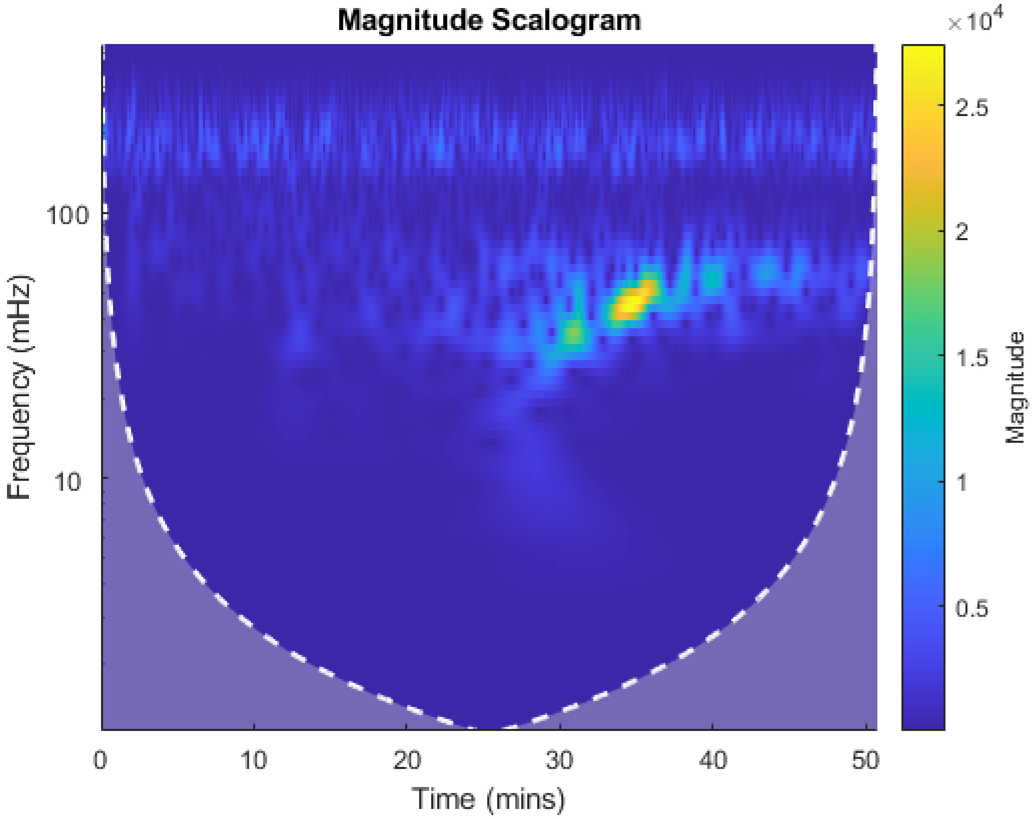
Generalized Morse and Analytic Morlet Wavelets

This example shows how to vary the time-bandwidth parameter of the generalized Morse wavelet to approximate the analytic Morlet wavelet.

Generalized Morse wavelets are a family of exactly analytic wavelets. Morse wavelets have two parameters, symmetry and time-bandwidth product. You can vary these parameters to obtain analytic wavelets with different properties and behaviors. For additional information, see “Morse Wavelets” and the references therein.

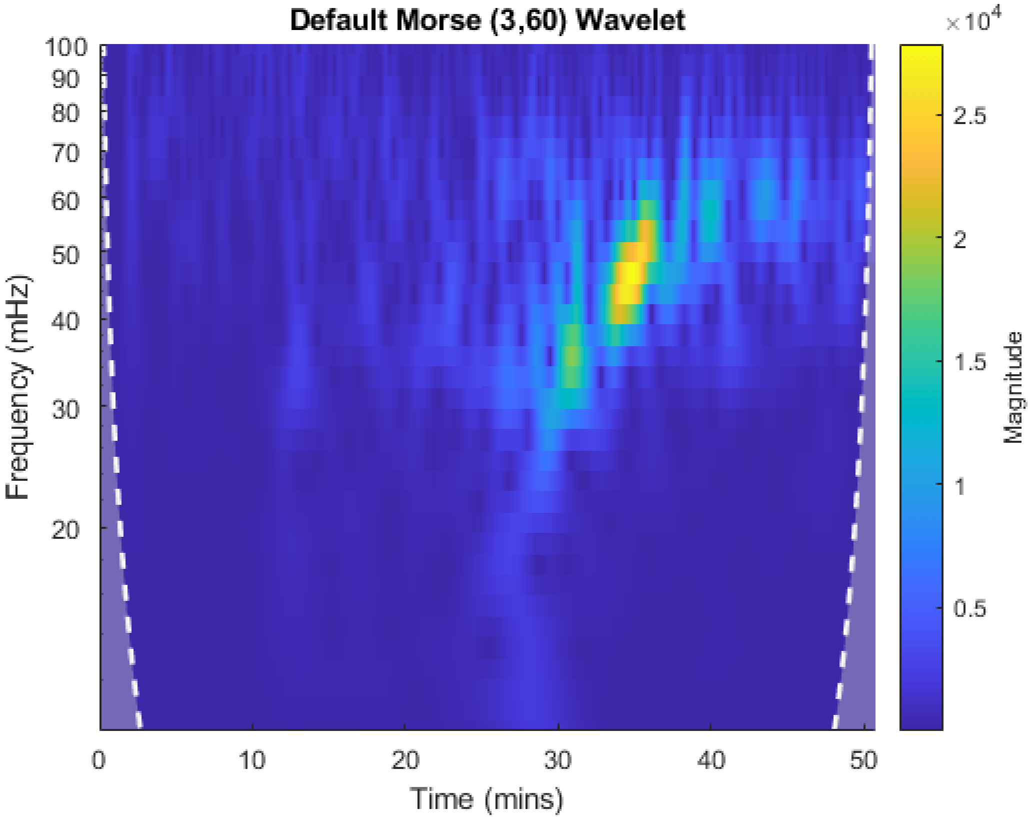
Load the seismograph data recorded during the 1995 Kobe earthquake. The data are seismograph (vertical acceleration, nm/sq.sec) measurements recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMT) and continuing for 51 minutes at 1 second intervals. Create a CWT filter bank with default settings that can be applied to the data. Use the filter bank to generate the scalogram.

```
load kobe
fb = cwtfilterbank(SignalLength=numel(kobe),SamplingFrequency=1);
cwt(kobe,FilterBank=fb)
```



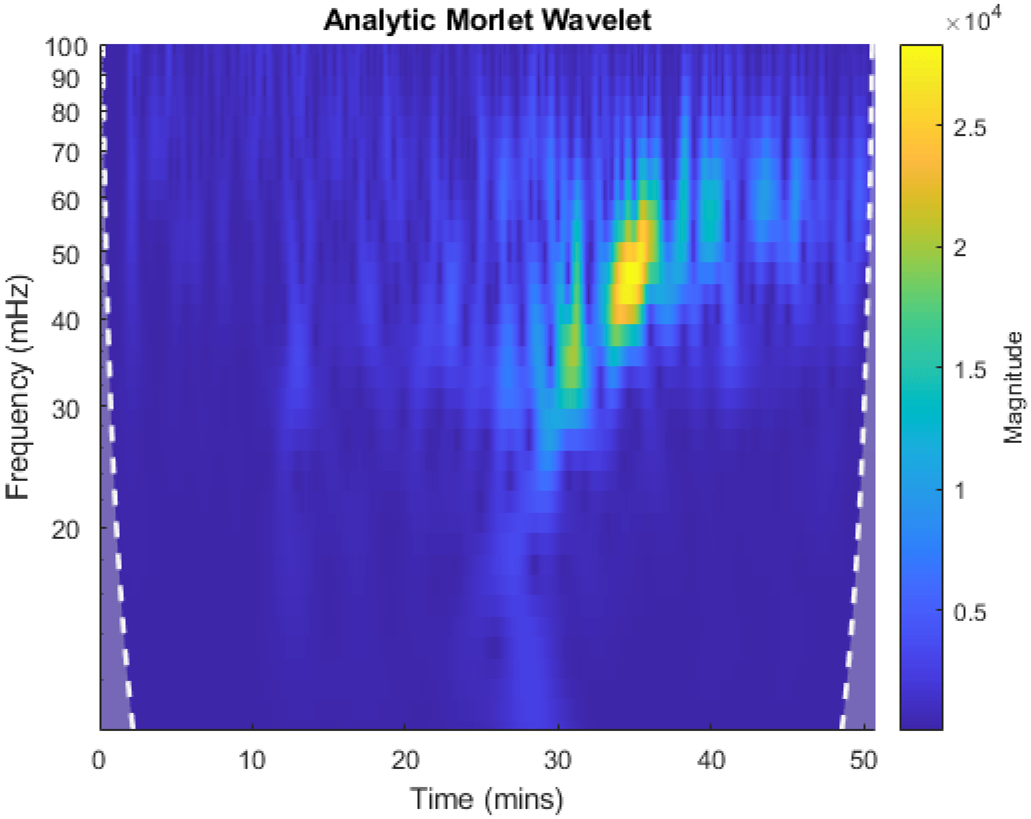
The magnitude of the wavelet coefficients is large in the frequency range from 10 mHz to 100 mHz. Create a new filter bank with frequency limits set to these values. Generate the scalogram.

```
fb2 = cwtfilterbank(SignalLength=numel(kobe),SamplingFrequency=1,...
    FrequencyLimits=[1e-2 1e-1]);
cwt(kobe,FilterBank=fb2)
title("Default Morse (3,60) Wavelet")
```



By default, `cwtfilterbank` uses the Morse (3,60) wavelet. Create a filter bank using the analytic Morlet wavelet with the same frequency limits. Generate a scalogram and compare with the scalogram generated by the Morse (3,60) wavelet.

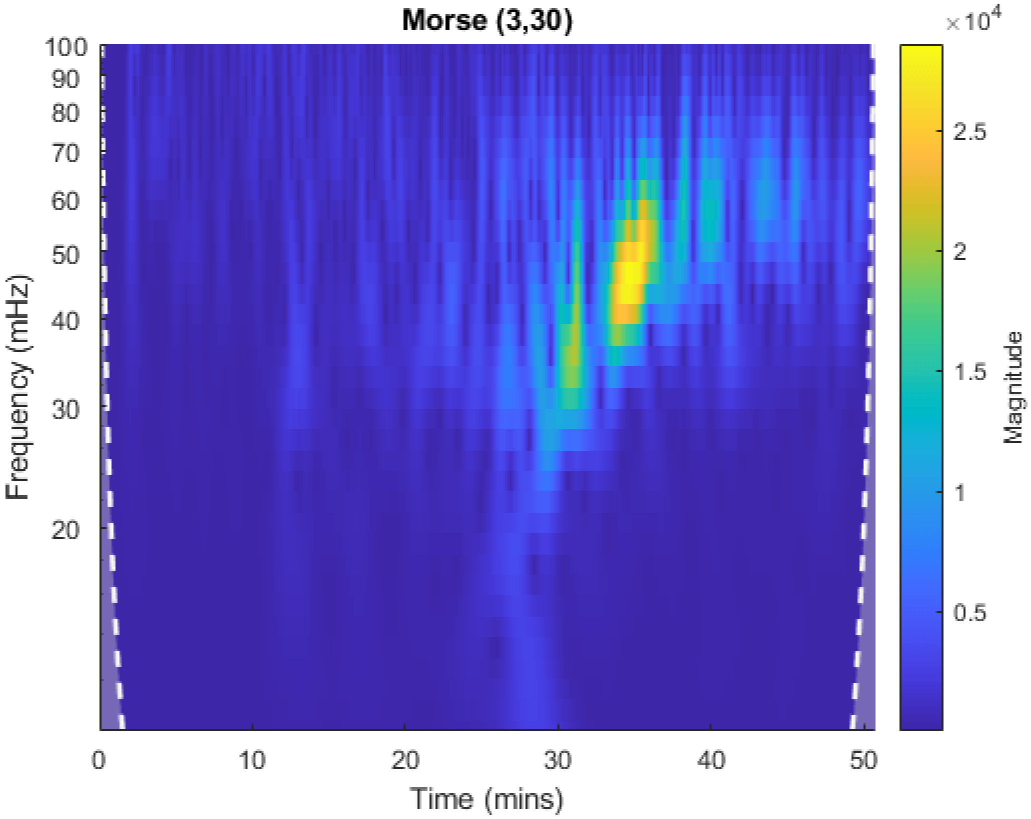
```
fbMorlet = cwtfilterbank(SignalLength=numel(kobe),SamplingFrequency=1,...
    FrequencyLimits=[1e-2 1e-1],...
    Wavelet="amor");
cwt(kobe,FilterBank=fbMorlet)
title("Analytic Morlet Wavelet")
```



The Morlet wavelet is not as well localized in frequency as the (3,60) Morse wavelet. However, by varying the time-bandwidth product, you can create a Morse wavelet with properties similar to the Morlet wavelet.

Create a filter bank using the Morse wavelet with a time-bandwidth value of 30 [2], with frequency limits as above. Generate the scalogram of the seismograph data. Note there is smearing in frequency nearly identical to the Morlet results.

```
fbMorse = cwtfilterbank(SignalLength=numel(kobe),SamplingFrequency=1,...
    FrequencyLimits=[1e-2 1e-1],...
    TimeBandwidth=30);
cwt(kobe,FilterBank=fbMorse)
title("Morse (3,30)")
```



Now examine the wavelets associated with the `fbMorlet` and `fbMorse` filter banks. From both filter banks, obtain the wavelet center frequencies, filter frequency responses, and time-domain wavelets. Confirm the center frequencies are nearly identical.

```
cfMorlet = centerFrequencies(fbMorlet);
[frMorlet,fMorlet] = freqz(fbMorlet);
[wvMorlet,tMorlet] = wavelets(fbMorlet);
cfMorse = centerFrequencies(fbMorse);
[frMorse,fMorse] = freqz(fbMorse);
[wvMorse,tMorse] = wavelets(fbMorse);

disp(["Number of Center Frequencies: ",num2str(length(cfMorlet))]);
```

```
    "Number of Center Frequencies: "    "34"
```

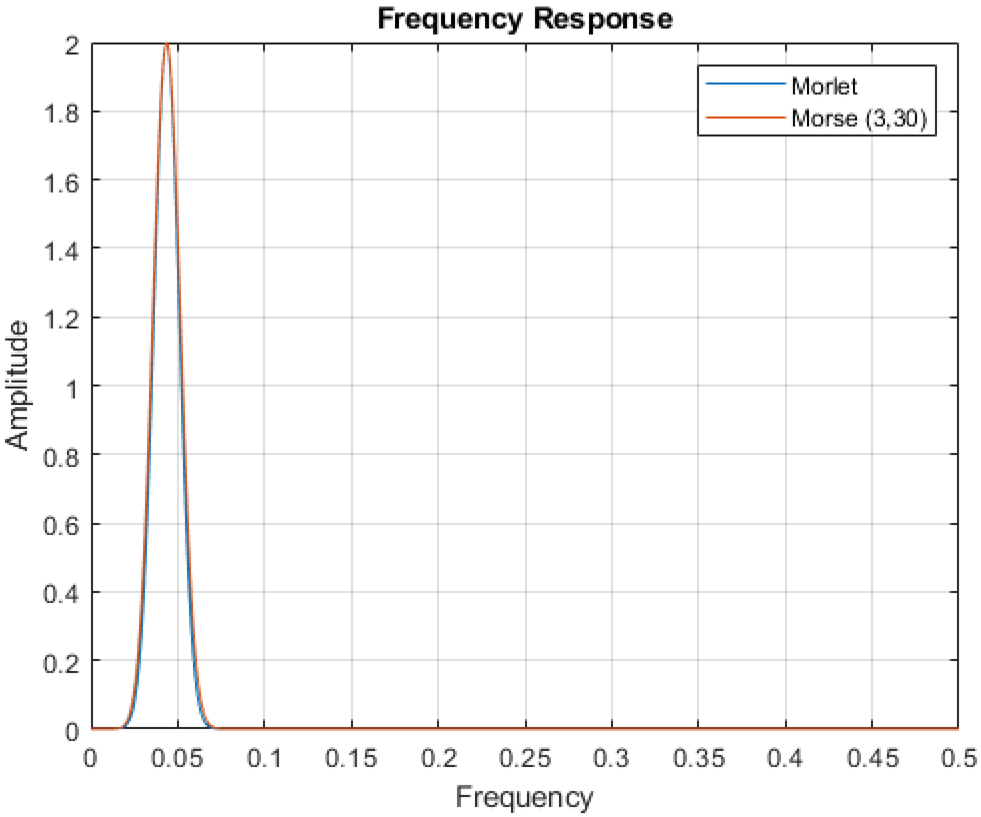
```
disp(["Maximum difference: ",num2str(max(abs(cfMorlet-cfMorse)))]);
```

```
    "Maximum difference: "    "2.7756e-17"
```

Each filter bank contains the same number of wavelets. Choose a center frequency, and plot the frequency response of the associated filter from each filter bank. Confirm the responses are nearly identical.

```
wv = 13;
figure
plot(fMorlet,frMorlet(wv,:));
hold on
plot(fMorse,frMorse(wv,:));
```

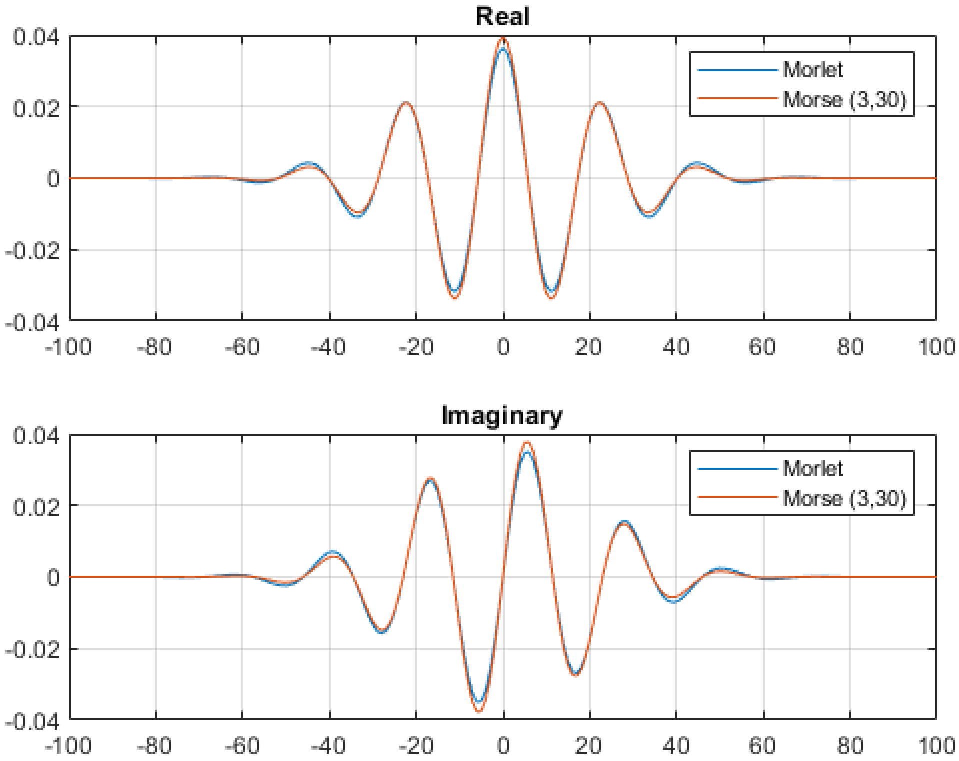
```
grid on
hold off
title("Frequency Response")
xlabel("Frequency")
ylabel("Amplitude")
legend("Morlet","Morse (3,30)")
```



Plot the time-domain wavelets associated with the same center frequency. Confirm they are nearly identical.

```
figure
subplot(2,1,1)
plot(tMorlet,real(wvMorlet(wv,:)))
hold on
plot(tMorse,real(wvMorse(wv,:)))
grid on
hold off
title("Real")
legend("Morlet","Morse (3,30)")
xlim([-100 100])
subplot(2,1,2)
plot(tMorlet,imag(wvMorlet(wv,:)))
hold on
plot(tMorse,imag(wvMorse(wv,:)))
grid on
hold off
title("Imaginary")
```

```
legend("Morlet","Morse (3,30)")
xlim([-100 100])
```



Changing the Time-bandwidth Product

This example shows that increasing the time-bandwidth product P^2 of the Morse wavelet creates a wavelet with more oscillations under its envelope. Increasing P^2 narrows the wavelet in frequency.

Create two filter banks. One filter bank has the default `TimeBandwidth` value of 60. The second filter bank has a `TimeBandwidth` value of 10. The `SignalLength` for both filter banks is 4096 samples.

```
sigLen = 4096;
fb60 = cwtfilterbank(SignalLength=sigLen);
fb10 = cwtfilterbank(SignalLength=sigLen,TimeBandwidth=10);
```

Obtain the time-domain wavelets for the filter banks.

```
[psi60,t] = wavelets(fb60);
[psi10,~] = wavelets(fb10);
```

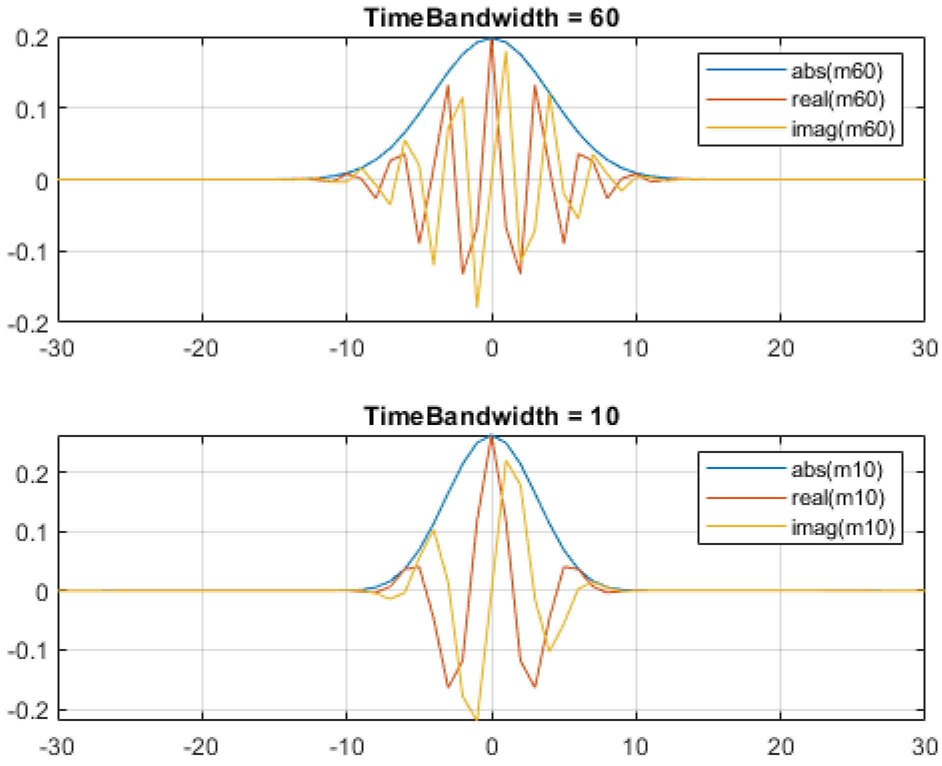
Use the `scales` function to find the mother wavelet for each filter bank.

```
sca60 = scales(fb60);
sca10 = scales(fb10);
```

```
[~,idx60] = min(abs(sca60-1));
[~,idx10] = min(abs(sca10-1));
m60 = psi60(idx60,:);
m10 = psi10(idx10,:);
```

Since the time-bandwidth product is larger for the `fb60` filter bank, verify the `m60` wavelet has more oscillations under its envelope than the `m10` wavelet.

```
subplot(2,1,1)
plot(t,abs(m60))
grid on
hold on
plot(t,real(m60))
plot(t,imag(m60))
hold off
xlim([-30 30])
legend("abs(m60)","real(m60)","imag(m60)")
title("TimeBandwidth = 60")
subplot(2,1,2)
plot(t,abs(m10))
grid on
hold on
plot(t,real(m10))
plot(t,imag(m10))
hold off
xlim([-30 30])
legend("abs(m10)","real(m10)","imag(m10)")
title("TimeBandwidth = 10")
```

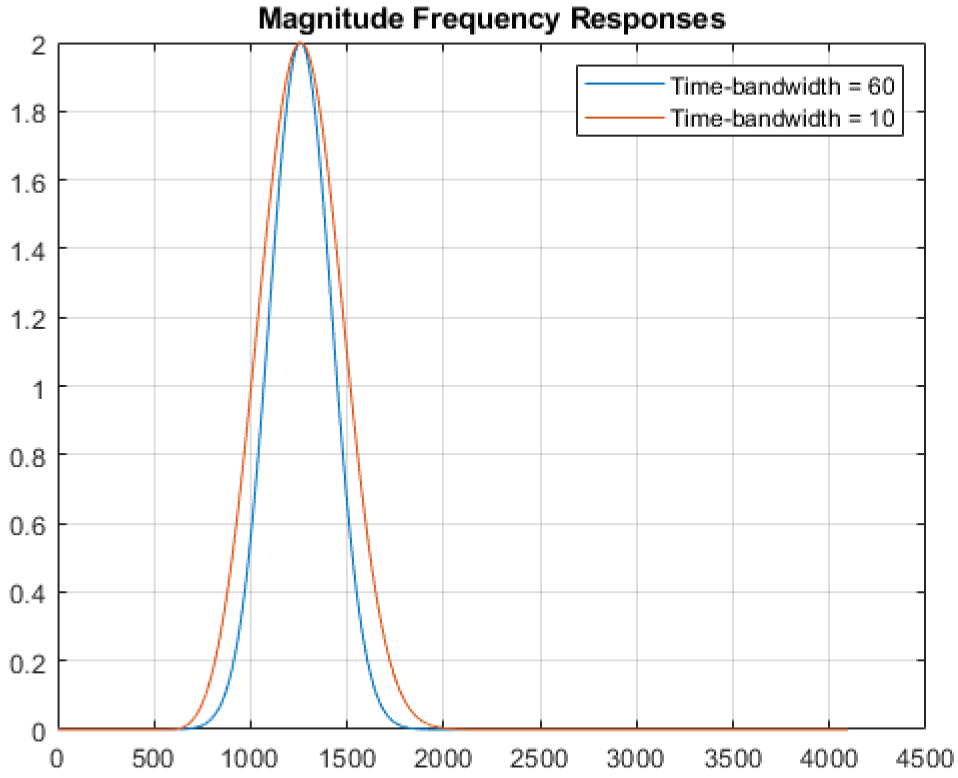


TimeBandwidth = 60

TimeBandwidth = 10

Align the peaks of the m60 and m10 magnitude frequency responses. Verify the frequency response of the m60 wavelet is narrower than the frequency response for the m10 wavelet.

```
cf60 = centerFrequencies(fb60);
cf10 = centerFrequencies(fb10);

m60cFreq = cf60(idx60);
m10cFreq = cf10(idx10);

freqShift = 2*pi*(m60cFreq-m10cFreq);
x10 = m10.*exp(1j*freqShift*(-sigLen/2:sigLen/2-1));

figure
plot([abs(fft(m60)).' abs(fft(x10)).'])
grid on
legend("Time-bandwidth = 60","Time-bandwidth = 10")
title("Magnitude Frequency Responses")
```



Using CWT Filter Bank on Multiple Time Series

This example shows how using a CWT filter bank can improve computational efficiency when taking the CWT of multiple time series.

Create a 100-by-1024 matrix x. Create a CWT filter bank appropriate for signals with 1024 samples.

```
x = randn(100,1024);
fb = cwtfilterbank;
```

Use `cwt` with default settings to obtain the CWT of a signal with 1024 samples. Create a 3-D array that can contain the CWT coefficients of 100 signals, each of which has 1024 samples.

```
cfs = cwt(x(1,:));
res = zeros(100,size(cfs,1),size(cfs,2));
```

Use the `cwt` function and take the CWT of each row of the matrix `x`. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = cwt(x(k,:));
end
toc
```

```
Elapsed time is 0.928160 seconds.
```

Now use the `wt` object function of the filter bank to take the CWT of each row of `x`. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = wt(fb,x(k,:));
end
toc
```

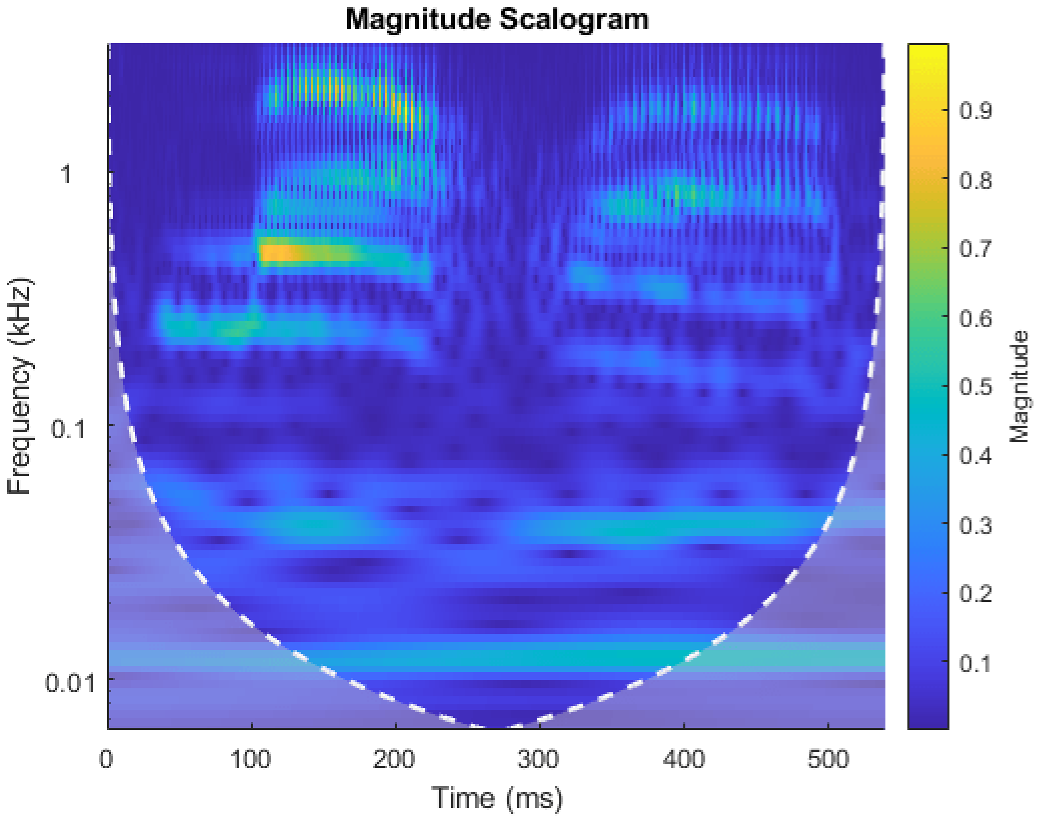
```
Elapsed time is 0.393524 seconds.
```

Plot CWT Scalogram in Subplot

This example shows how to plot the CWT scalogram in a figure subplot.

Load the speech sample. The data is sampled at 7418 Hz. Plot the default CWT scalogram.

```
load mtlb
cwt(mtlb,Fs)
```



Obtain the continuous wavelet transform of the signal, and the frequencies of the CWT.

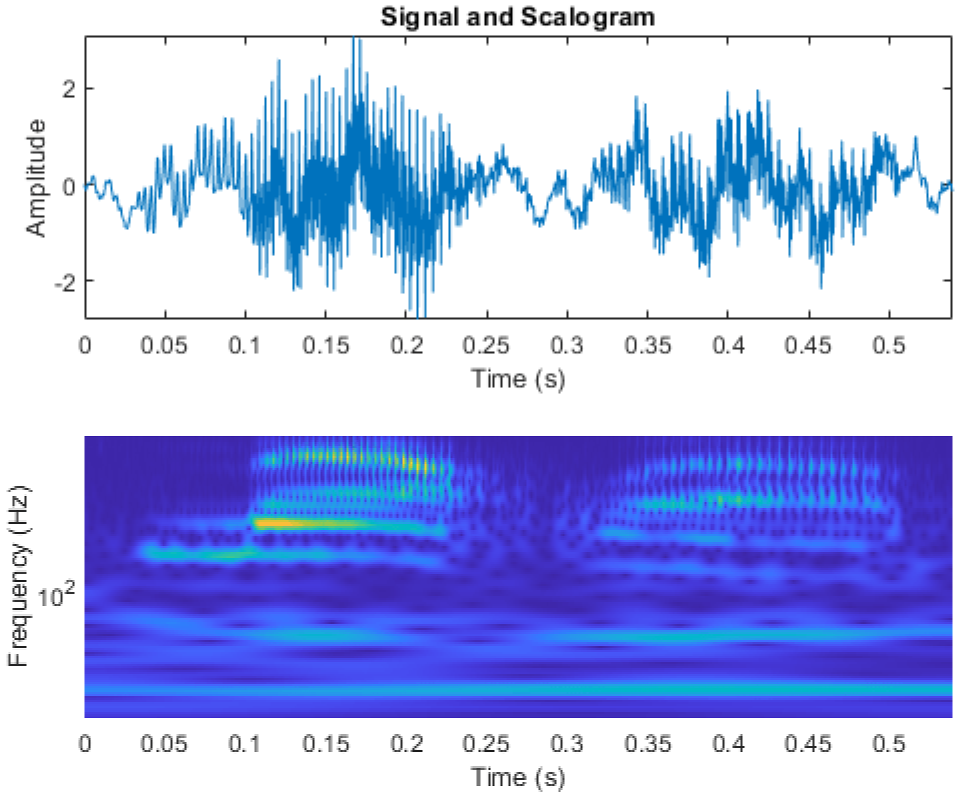
```
[cfs,frq] = cwt(mtlb,Fs);
```

The `cwt` function sets the time and frequency axes in the scalogram. Create a vector representing the sample times.

```
tms = (0:numel(mtlb)-1)/Fs;
```

In a new figure, plot the original signal in the upper subplot and the scalogram in the lower subplot. Plot the frequencies on a logarithmic scale.

```
figure
subplot(2,1,1)
plot(tms,mtlb)
axis tight
title("Signal and Scalogram")
xlabel("Time (s)")
ylabel("Amplitude")
subplot(2,1,2)
surface(tms,frq,abs(cfs))
axis tight
shading flat
xlabel("Time (s)")
ylabel("Frequency (Hz)")
set(gca,"yscale","log")
```



Tips

- The first time you use a filter bank to take the CWT of a signal, the wavelet filters are constructed to have the same datatype as the signal. A warning message is generated when you apply the same filter bank to a signal with a different datatype. Changing datatypes comes with the cost of redesigning or changing the precision of the filter bank. For optimal performance, use a consistent datatype.
- When performing multiple CWTs, for example inside a for-loop, the recommended workflow is to first create a `cwtfilterbank` object and then use the `wt` object function. This workflow minimizes overhead and maximizes performance. See "Using CWT Filter Bank on Multiple Time Series" on page 1-192.

Compatibility Considerations

`BPfrequencies` and `BPperiods` will be removed

Not recommended starting in R2018b

The `BPfrequencies` and `BPperiods` object functions of `cwtfilterbank` have been renamed `centerFrequencies` and `centerPeriods`, respectively. The functionality remains unchanged. `BPfrequencies` and `BPperiods` will be removed in a future release.

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.

[4] Lilly, J. M. “Element analysis: a wavelet-based method for analysing time-localized events in noisy time series.” *Proceedings of the Royal Society A.* Volume 473: 20160776, 2017, pp. 1–28. dx.doi.org/10.1098/rspa.2016.0776.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The following properties are not supported: `SamplingPeriod`, `PeriodLimits`.
- The following object functions support C/C++ code generation:
  - `wt`
  - `freqz`
    - Plotting is not supported.
  - `timeSpectrum`
  - `scaleSpectrum`
    - `PeriodLimits` name-value pair is not supported.
  - `wavelets`
  - `scales`
  - `qfactor`
  - `centerFrequencies`

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps

Wavelet Time-Frequency Analyzer

Functions

`cwt` | `cwtfreqbounds` | `icwt`

Topics

“Practical Introduction to Continuous Wavelet Analysis”
“Using Wavelet Time-Frequency Analyzer App”
“Boundary Effects and the Cone of Influence”
“Morse Wavelets”
“Time-Frequency Gallery”

Introduced in R2018a

cwtfreqbounds

CWT maximum and minimum frequency or period

Syntax

```
[minfreq,maxfreq] = cwtfreqbounds(N)
[minfreq,maxfreq] = cwtfreqbounds(N,Fs)
[maxperiod,minperiod] = cwtfreqbounds(N,Ts)
[ ___ ] = cwtfreqbounds( ___ ,Name=Value)
```

Description

`[minfreq,maxfreq] = cwtfreqbounds(N)` returns the minimum and maximum wavelet bandpass frequencies in cycles/sample for a signal of length `N`. The minimum and maximum frequencies are determined for the default Morse (3,60) wavelet. The minimum frequency is determined so that two time standard deviations of the default wavelet span the `N`-point signal at the coarsest scale. The maximum frequency is such that the highest frequency wavelet bandpass filter drops to ½ of its peak magnitude at the Nyquist frequency.

`[minfreq,maxfreq] = cwtfreqbounds(N,Fs)` returns the bandpass frequencies in hertz for the sampling frequency `Fs`.

`[maxperiod,minperiod] = cwtfreqbounds(N,Ts)` returns the bandpass periods for the sampling period `Ts`. `maxperiod` and `minperiod` are scalar durations with the same format as `Ts`. If the number of standard deviations is set so that `log2(maxperiod/minperiod) < 1/NV` where `NV` is the number of voices per octave, `maxperiod` is adjusted to `minperiod × 2^(1/NV)`.

`[ ___ ] = cwtfreqbounds( ___ ,Name=Value)` returns the minimum and maximum wavelet bandpass frequencies or periods with additional options specified by one or more `Name=Value` arguments. For example, `[minf,maxf] = cwtfreqbounds(1000,TimeBandwidth=30)` sets the time-bandwidth parameter of the default Morse wavelet to 30.

Examples

Wavelet Bandpass Frequencies Using Default Values

Obtain the minimum and maximum wavelet bandpass frequencies for a signal with 1000 samples using the default values.

```
[minfreq,maxfreq] = cwtfreqbounds(1000)
```

```
minfreq = 0.0033

maxfreq = 0.4341
```

Construct CWT Filter Bank With Peak Magnitude at Nyquist

Obtain the minimum and maximum wavelet bandpass frequencies for the default Morse wavelet for a signal of length 10,000 and a sampling frequency of 1 kHz. Set the cutoff to 100% so that the highest frequency wavelet bandpass filter peaks at the Nyquist frequency of 500 Hz.

```
sigLength = 10000;
Fs = 1e3;
[minfreq,maxfreq] = cwtfreqbounds(sigLength,Fs,cutoff=100);
```

Construct a CWT filter bank using the values `cwtfreqbounds` returns. Obtain the frequency responses of the filter bank.

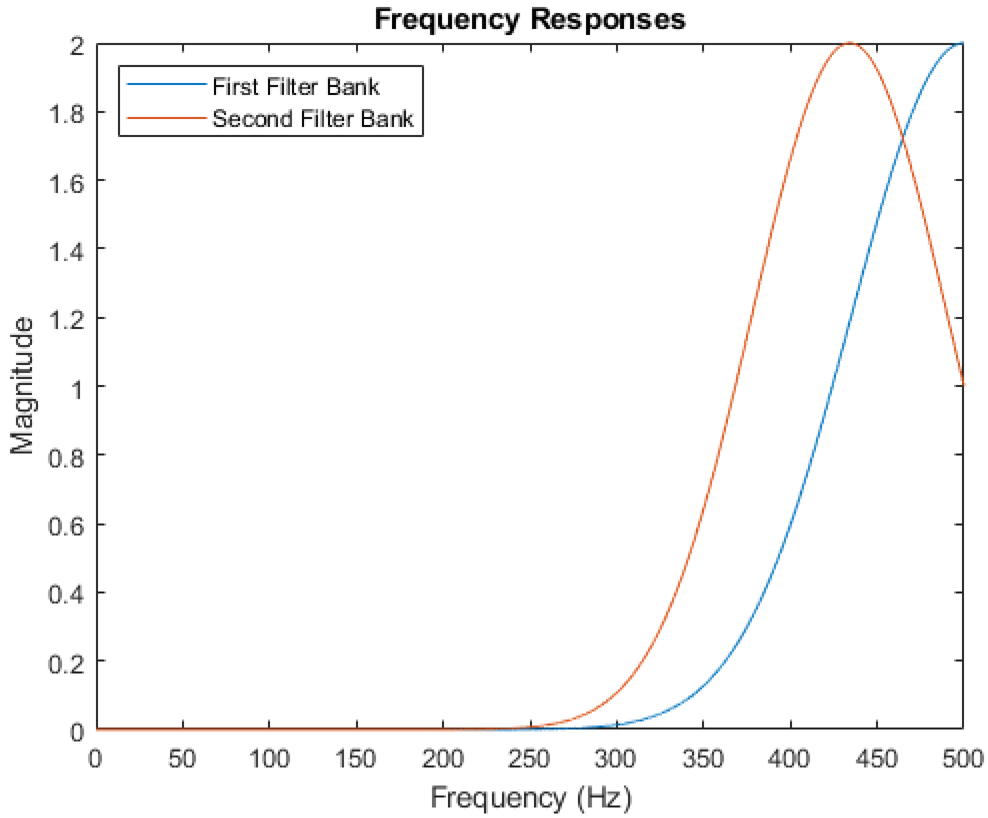
```
fb = cwtfilterbank(SignalLength=sigLength,SamplingFrequency=Fs,...
    FrequencyLimits=[minfreq maxfreq]);
[psidft,f] = freqz(fb);
```

Construct a second CWT filter bank identical to the first, but instead use the default frequency limits. Obtain the frequency responses of the second filter bank.

```
fb2 = cwtfilterbank(SignalLength=sigLength,SamplingFrequency=Fs);
[psidft2,~] = freqz(fb2);
```

For each filter bank, plot the frequency response of the filter with the highest center frequency. Confirm the frequency response from the first filter bank peaks at the Nyquist, and the frequency response from the second filter bank is 50% of the peak magnitude at the Nyquist.

```
plot(f,psidft(1,:))
hold on
plot(f,psidft2(1,:))
hold off
title("Frequency Responses")
xlabel("Frequency (Hz)")
ylabel("Magnitude")
legend("First Filter Bank","Second Filter Bank",...
    Location="NorthWest")
```



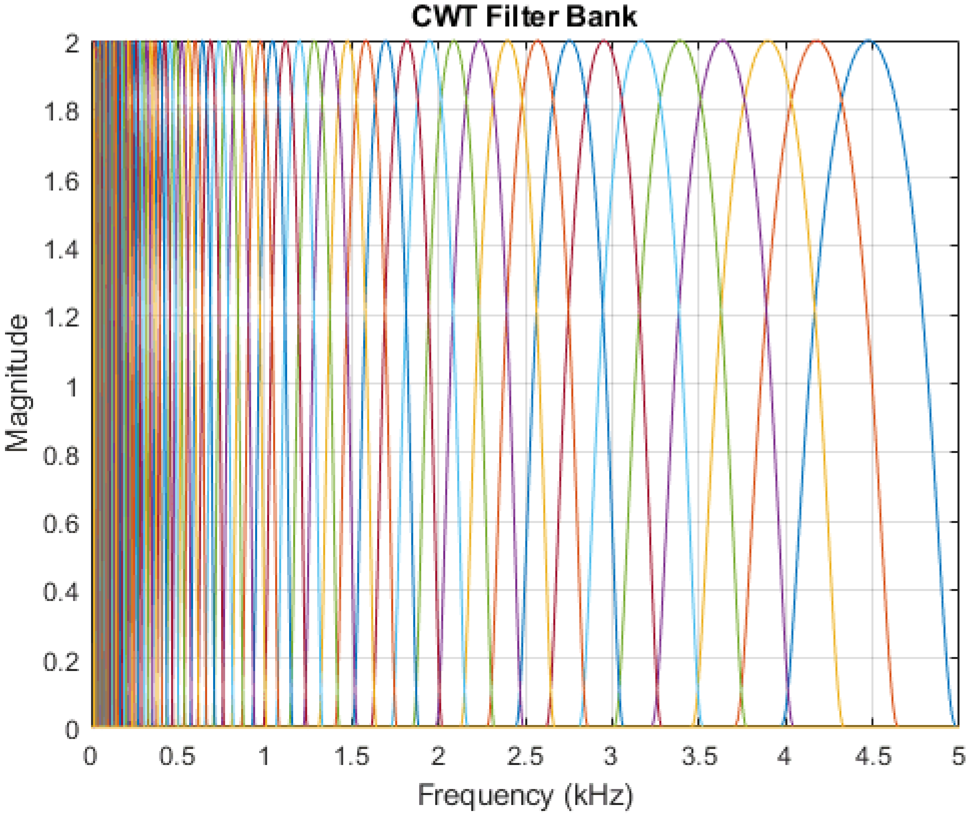
Decay Highest Frequency Wavelet in CWT Filter Bank to Specific Value

Obtain the minimum and maximum frequencies for the bump wavelet for a signal of length 5,000 and a sampling frequency of 10 kHz. Specify a cutoff value of $100 \times 10^{-8}/2$ so that the highest frequency wavelet bandpass filter decays to 10^{-8} at the Nyquist.

```
[minf,maxf] = cwtfreqbounds(5e3,1e4,wavelet="bump",cutoff=100*1e-8/2);
```

Construct the filter bank using the values returned by `cwtfreqbounds`. Plot the frequency responses.

```
fb = cwtfilterbank(SignalLength=5e3,Wavelet="bump",...
    SamplingFrequency=1e4,FrequencyLimits=[minf maxf]);
freqz(fb)
```



Frequency Range for Strictly Zero and Effectively Zero Cutoff Values

Obtain the minimum and maximum wavelet bandpass frequencies for a signal of length 4096. Specify a cutoff of 0. Display the minimum and maximum bandpass frequencies.

```
sLength = 4096;
co = 0;
[minfreq,maxfreq] = cwtfreqbounds(sLength,Cutoff=co);
fprintf("Min Frequency: %f cycles/sample\nMax Frequency: %f cycles/sample", ...
    minfreq,maxfreq)
```

```
Min Frequency: 0.000805 cycles/sample
Max Frequency: 0.103574 cycles/sample
```

Create a filter bank using the frequency limits. Obtain the two-sided wavelet frequency responses.

```
fb = cwtfilterbank(SignalLength=sLength,FrequencyLimits=[minfreq,maxfreq]);
[psif,f] = freqz(fb,FrequencyRange="twosided");
```

Obtain the minimum and maximum wavelet bandpass frequencies for a signal of length 4096, but this time specify a cutoff of $100 \times 10^{-8}/2$. Create a second filter bank using these new frequencies. Confirm the second frequency range is larger than the first frequency range.

```
co = 100*(1e-8/2);
[minfreq2,maxfreq2] = cwtfreqbounds(sLength,Cutoff=co);
```

```
fb2 = cwtfilterbank(SignalLength=sLength,FrequencyLimits=[minfreq2,maxfreq2]);
fprintf("Min Frequency: %f cycles/sample\nMax Frequency: %f cycles/sample", ...
    minfreq2,maxfreq2);
```

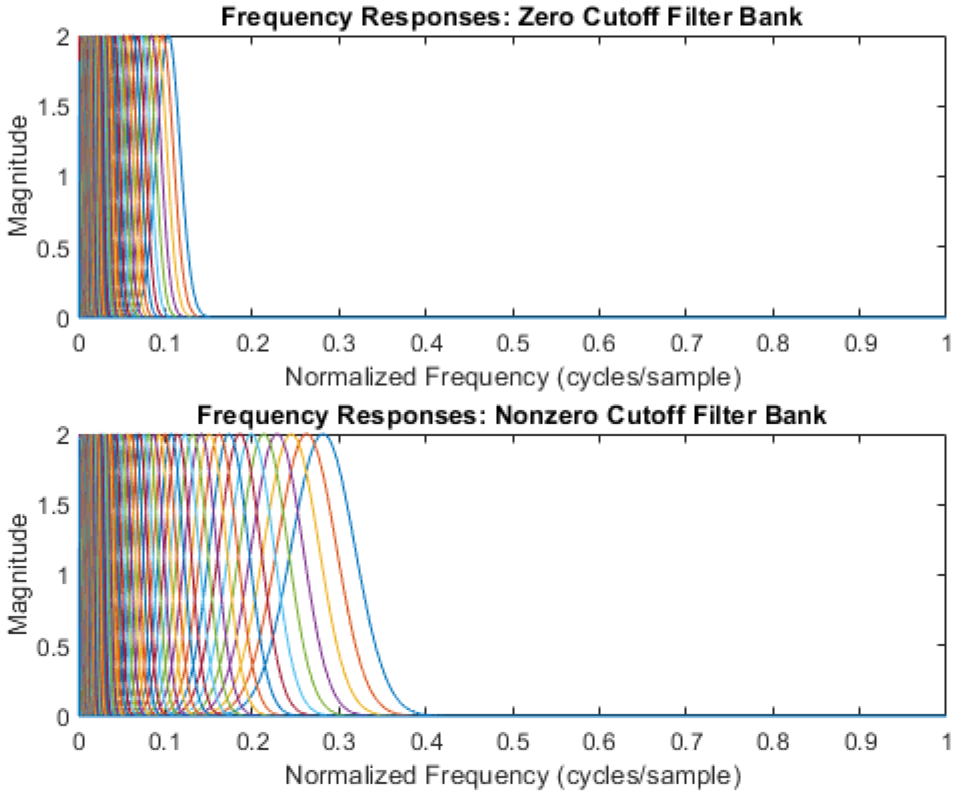
```
Min Frequency: 0.000805 cycles/sample
Max Frequency: 0.281770 cycles/sample
```

Obtain the two-sided wavelet frequency responses of the second filter bank.

```
[psif2,f2] = freqz(fb2,FrequencyRange="twosided");
```

Plot the frequency responses of the filter banks.

```
subplot(2,1,1)
plot(f,psif)
title("Frequency Responses: Zero Cutoff Filter Bank")
ylabel("Magnitude")
xlabel("Normalized Frequency (cycles/sample)")
subplot(2,1,2)
plot(f2,psif2)
title("Frequency Responses: Nonzero Cutoff Filter Bank")
ylabel("Magnitude")
xlabel("Normalized Frequency (cycles/sample)")
```



For the wavelet filter with the highest center frequency in each filter bank, obtain the magnitude of the frequency response at the Nyquist. Observer there is minimal difference between the two values.

```
fprintf("Zero Cutoff / Magnitude at Nyquist: %g",psif(1,floor(size(psif,2)/2)))
```

```
Zero Cutoff / Magnitude at Nyquist: 2.43333e-309
```

```
fprintf("Nonzero Cutoff / Magnitude at Nyquist: %g",psif2(1,floor(size(psif2,2)/2)))
```

```
Nonzero Cutoff / Magnitude at Nyquist: 1.02265e-08
```

Input Arguments

N — Signal length

positive integer ≥ 4

Signal length, specified as a positive integer greater than or equal to 4.

Data Types: `double`

Fs — Sampling frequency

positive scalar

Sampling frequency in hertz, specified as a positive scalar.

Example: `[minf,maxf] = cwtfreqbounds(2048,100)`

Data Types: `double`

Ts — Sampling period

scalar duration

Sampling period, specified as a positive scalar `duration`.

Example: `[minp,maxp] = cwtfreqbounds(2048,seconds(2))`

Data Types: `duration`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[minf,maxf] = cwtfreqbounds(1000,Wavelet="bump",VoicesPerOctave=10)` returns the minimum and maximum bandpass frequencies using the bump wavelet and 10 voices per octave for a signal with 1000 samples.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `[minf,maxf] = cwtfreqbounds(1000,"Wavelet","bump","VoicesPerOctave",10)`

Wavelet — Analysis wavelet

`"Morse"` (default) | `"amor"` | `"bump"`

Analysis wavelet used to determine the minimum and maximum frequencies or periods, specified as `"Morse"`, `"amor"`, or `"bump"`. These strings specify the analytic Morse, Morlet, and bump wavelet, respectively. The default wavelet is the analytic Morse (3,60) wavelet.

For Morse wavelets, you can also parametrize the wavelet using the `TimeBandwidth` or `WaveletParameters` name-value arguments.

Example: `[minp,maxp] = cwtfreqbound(2048,seconds(1),Wavelet="bump")`

`Cutoff` — Percentage of the peak magnitude

`50` for the Morse wavelet, `10` for the analytic Morlet and bump wavelets (default) | scalar between 0 and 100

Percentage of the peak magnitude at the Nyquist, specified as a scalar between 0 and 100. Setting `Cutoff` to `0` indicates that the wavelet frequency response decays to 0 at the Nyquist. Setting `Cutoff` to `100` indicates that the value of the wavelet bandpass filters peaks at the Nyquist.

For `cwtfilterbank`, the analytic wavelets filters peak at a value of 2. As a result, you can ensure the highest frequency wavelet decays to a value of α at the Nyquist frequency by setting `Cutoff` to 100 × α/2. In that case, you must have 0 ≤ α ≤ 2.

Note Unless your application requires a strict cutoff value of 0, consider setting `Cutoff` to a small nonzero value, for example, on the order of 10^{-8}. By specifying a small value, you can increase the frequency range `[minfreq,maxfreq]` and still obtain a wavelet frequency response that effectively decays to 0 at the Nyquist. See “Frequency Range for Strictly Zero and Effectively Zero Cutoff Values” on page 1-201.

Data Types: `double`

`StandardDeviations` — Number of time standard deviations

`2` (default) | positive integer ≥ 2

Number of time standard deviations used to determine the minimum frequency (longest scale), specified as a positive integer greater than or equal to 2. For the Morse, analytic Morlet, and bump wavelets, four standard deviations generally ensures that the wavelet decays to zero at the ends of the signal support. Incrementing `StandardDeviations` by multiples of 4, for example 4**M*, ensures that *M* whole wavelets fit within the signal length. If the number of standard deviations is set so that `log2(minfreq/maxfreq) > -1/NV`, where `NV` is the number of voices per octave, `minfreq` is adjusted to `maxfreq × 2^(-1/NV)`.

Data Types: `double`

`TimeBandwidth` — Time-bandwidth for the Morse wavelet

`60` (default) | scalar greater than 3 and less than or equal to 120

Time-bandwidth for the Morse wavelet, specified as a positive scalar. The symmetry (gamma) of the Morse wavelet is assumed to be 3. The larger the time-bandwidth parameter, the more spread out the wavelet is in time and narrower the wavelet is in frequency. The standard deviation of the Morse wavelet in time is approximately `sqrt(TimeBandwidth/2)`. The standard deviation in frequency is approximately `1/2*sqrt(2/TimeBandwidth)`.

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

Data Types: `double`

`WaveletParameters` — Morse wavelet parameters

`[3,60]` (default) | two-element vector of scalars

Morse wavelet parameters, specified as a two-element vector. The first element is the symmetry parameter (gamma), which must be greater than or equal to 1. The second element is the time-bandwidth parameter, which must be greater than or equal to gamma. The ratio of the time-bandwidth parameter to gamma cannot exceed 40.

When gamma is equal to 3, the Morse wavelet is perfectly symmetric in the frequency domain. The skewness is equal to 0. Values of gamma greater than 3 result in positive skewness, while values of gamma less than 3 result in negative skewness.

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

Data Types: `double`

`VoicesPerOctave` — Number of voices per octave

`10` (default) | integer between 1 and 48

Number of voices per octave to use in determining the necessary separation between the minimum and maximum scales, specified as an integer between 1 and 48. The minimum and maximum scales are equivalent to the minimum and maximum frequencies or maximum and minimum periods, respectively.

Data Types: `double`

Output Arguments

`minfreq` — Minimum wavelet bandpass frequency

scalar

Minimum wavelet bandpass frequency, returned as a scalar. `minfreq` is in cycles/sample if `SamplingFrequency` is not specified. Otherwise, `minfreq` is in hertz.

Data Types: `double`

`maxfreq` — Maximum wavelet bandpass frequency

scalar

Maximum wavelet bandpass frequency, returned as a scalar. `maxfreq` is in cycles/sample if `SamplingFrequency` is not specified. Otherwise, `maxfreq` is in hertz.

Data Types: `double`

`maxperiod` — Maximum wavelet bandpass period

scalar duration

Maximum wavelet bandpass period, returned as a scalar duration with the same format as `Ts`.

If the number of standard deviations is set so that `log2(maxperiod/minperiod) < 1/NV`, where `NV` is the number of voices per octave, `maxperiod` is adjusted to `minperiod × 2^(1/NV)`.

Data Types: `duration`

`minperiod` — Minimum wavelet bandpass period

scalar duration

Minimum wavelet bandpass period, returned as a scalar duration with the same format as `Ts`.

If the number of standard deviations is set so that `log2(maxperiod/minperiod) < 1/NV`, where `NV` is the number of voices per octave, `maxperiod` is adjusted to `minperiod × 2^(1/NV)`

Data Types: `duration`

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The sampling period (`Ts`) input argument is not supported.

See Also

`cwtfilterbank` | `cwt`

Introduced in R2018a

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 `x`. 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 or string scalar *`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 element 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 “More About” on page 1-209 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 ≤ `x1` < `x2` ≤ `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.

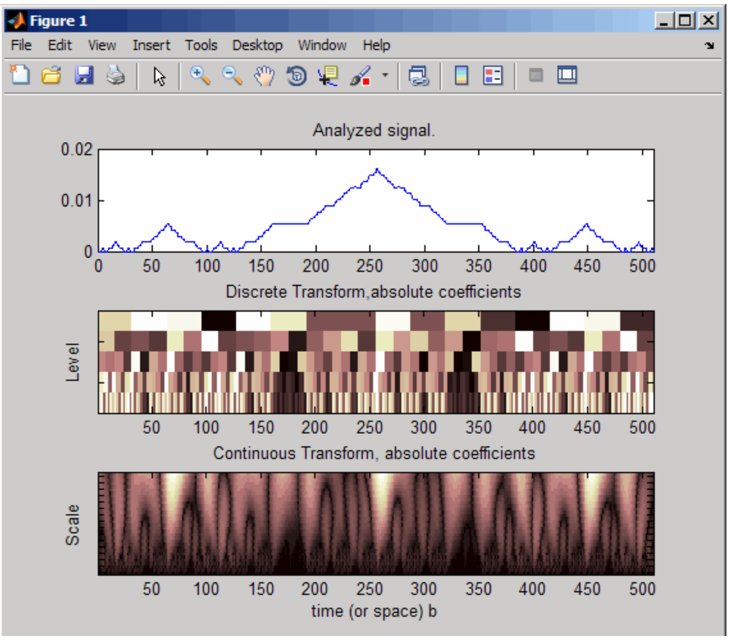
Examples

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');
```



More About

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 1-216 for descriptions of valid analyzing wavelets.

For additional default values, see `scales` in “Name-Value Pair Arguments” on page 1-212.

`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 1-212 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 check box 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 1-216 for descriptions of the default for each wavelet.
2. `nb` — Number of scales. The default `nb` depends on the wavelet. See the More About on page 1-216 for descriptions of the default for each wavelet.
3. `ds` — Spacing between scales. The default `ds` depends on the wavelet. See the More About on page 1-216 for descriptions of the default for each wavelet. You can construct a linear or logarithmic scale vector using `ds`. A logarithmic scale vector is constructed by default. Use the `type` field of `scales` to construct a linear scale vector.
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 1-213 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 1-216 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');
```

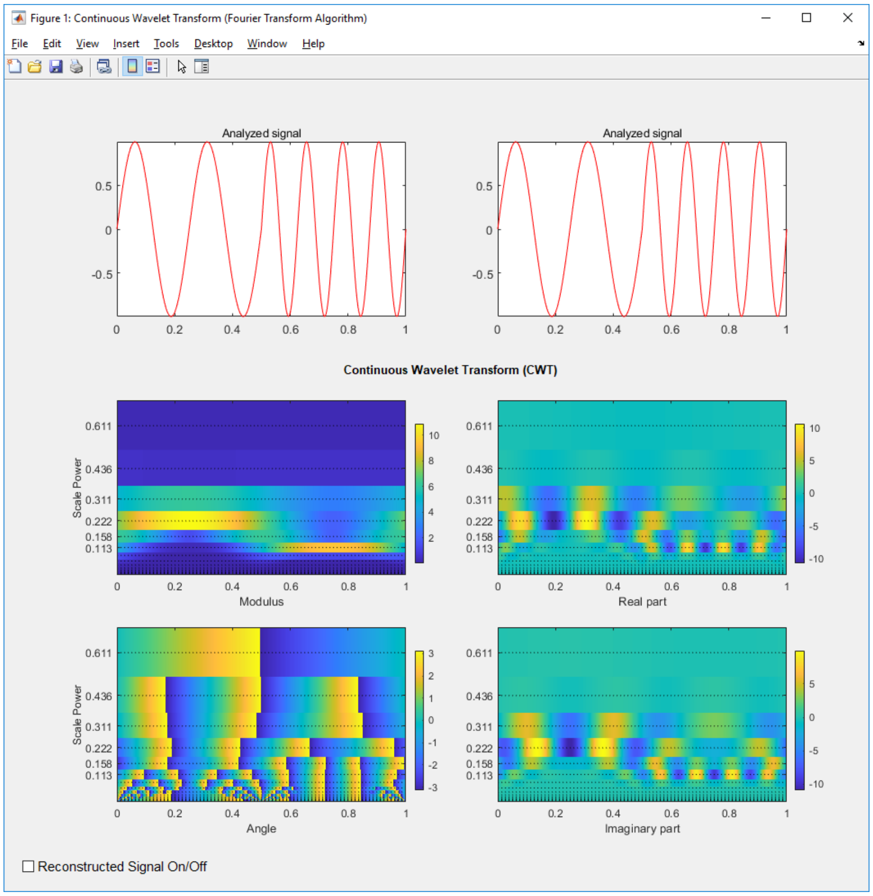


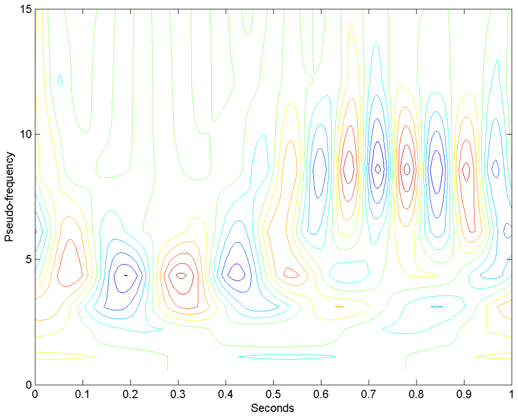
Figure 1: Continuous Wavelet Transform (Fourier Transform Algorithm)
File Edit View Insert Tools Desktop Window Help
Analyzed signal
Analyzed signal
Continuous Wavelet Transform (CWT)
Scale Power
0.611
0.436
0.311
0.222
0.158
0.113
Modulus
Real part
Angle
Imaginary part
0
0.2
0.4
0.6
0.8
1
☐ Reconstructed Signal On/Off

You can display or hide the reconstructed signal using the check box at the bottom left of the figure. When you select the check box, 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.

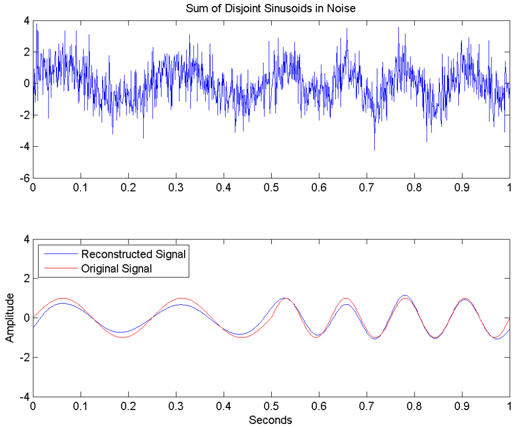
```
rng default % Reset random number generator for reproducible results
N = 1024;
% Sampling interval is 1/1023
t = linspace(0,1,N);
y = sin(2*pi*4*t).*(t<=0.5)+sin(2*pi*8*t).*(t>0.5);
ynoise = y+randn(size(t));
% Because the sampling interval differs from the default
% you must input it along with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



More About

Morlet Wavelet

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

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

where $U(\omega)$ is the Heaviside step function [5] on page 1-218.

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

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

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

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

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:

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

where Γ( ) denotes the gamma function [5] on page 1-218.

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:

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

where *U(ω)* is the Heaviside step function [5] on page 1-218.

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

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

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.

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”
“CWT-Based Time-Frequency Analysis”

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
```

More About

Morlet Wavelet

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

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

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

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

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

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

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

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:

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

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:

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

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

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

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

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
```

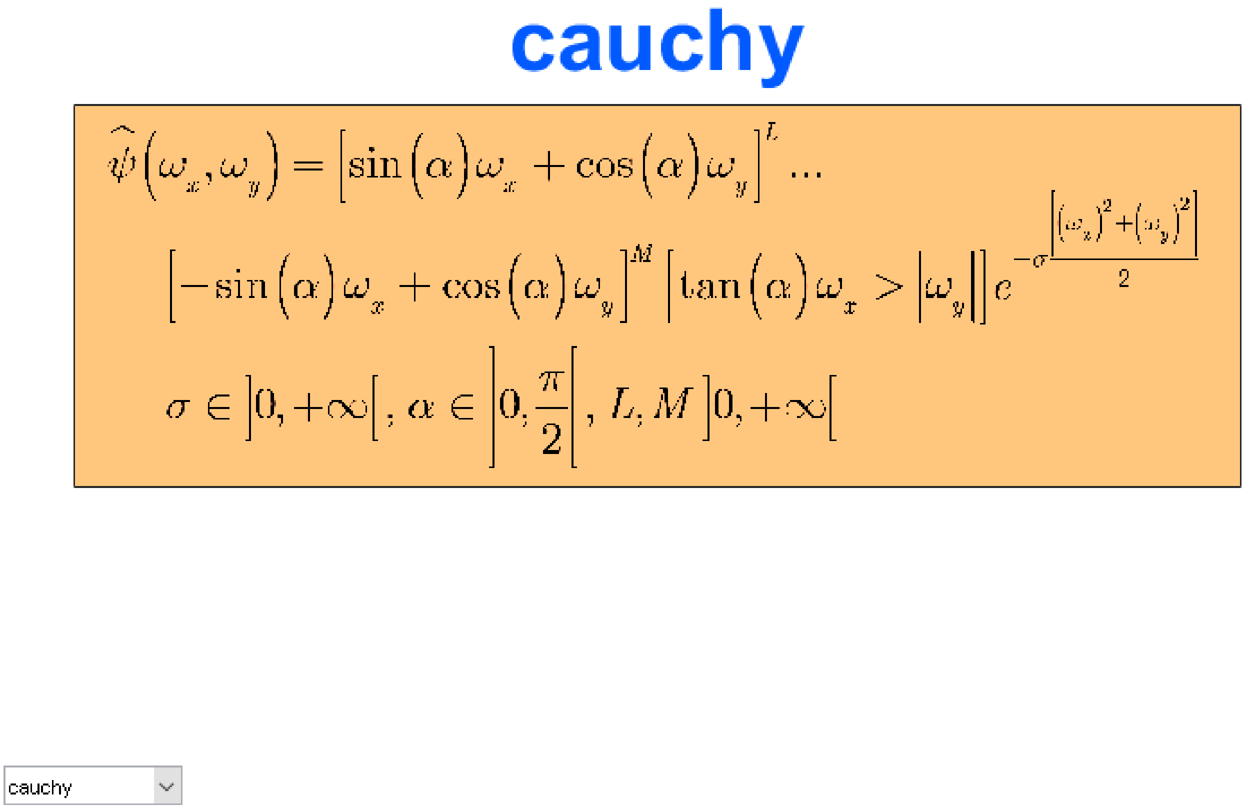
```
 CWTFTINFO2 Information on wavelets for CWTFT2
  CWTFTINFO2 provides information on the available wavelets
  for 2-D Continuous Wavelet Transform using FFT.
  The wavelets are defined by their Fourier transform.

  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')
```



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

$$\left[-\sin\left(\alpha\right)\omega_x + \cos\left(\alpha\right)\omega_y\right]^M \left[\tan\left(\alpha\right)\omega_x > \left|\omega_y\right|\right] e^{-\sigma\frac{\left[\left(\omega_x\right)^2+\left(\omega_y\right)^2\right]}{2}}$$

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

Input Arguments

wname — Wavelet name
character vector | string scalar

Wavelet name, specified as a character vector or string scalar. 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}
'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.

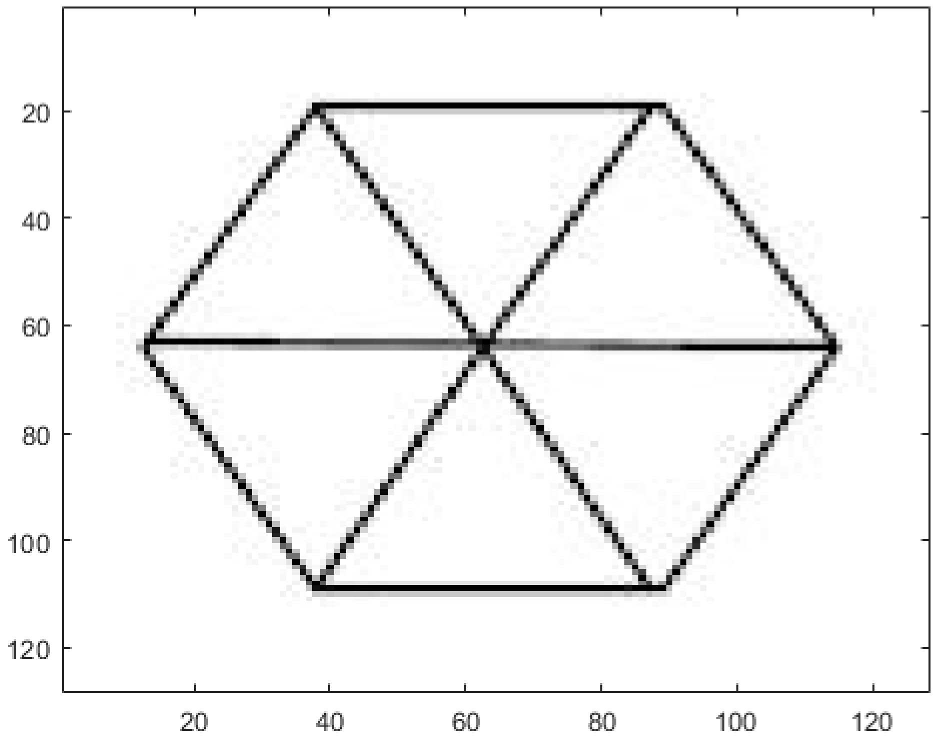
Examples

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);
```

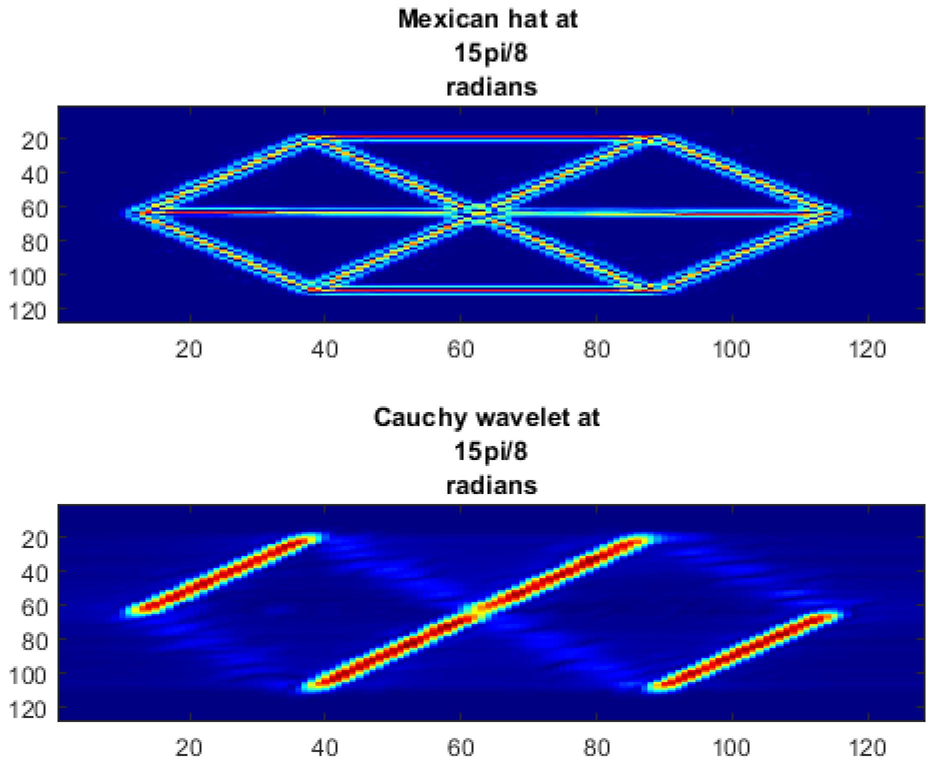


Obtain the scale-one 2-D CWT with both the Mexican hat and Cauchy wavelets. Specify a vector of angles going from 0 to 15?/8 in ?/8 increments.

```
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.

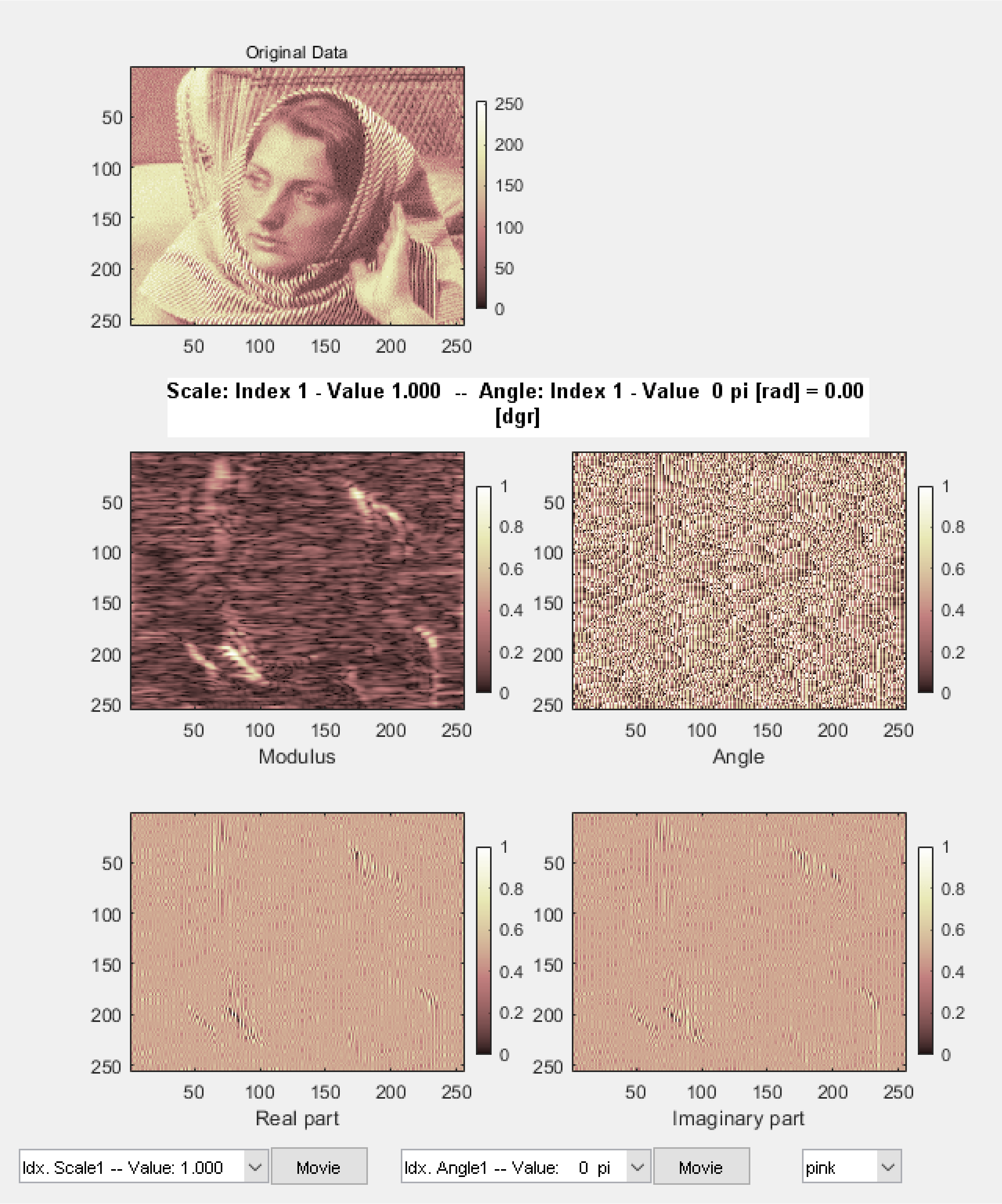
```
angz = {'0', 'pi/8', 'pi/4', '3pi/8', 'pi/2', '5pi/8', '3pi/4', ...
    '7pi/8','pi', '9pi/8', '5pi/4', '11pi/8', '3pi/2', ...
    '13pi/8' '7pi/4', '15pi/8'};
for angn = 1:length(angz)
    subplot(211)
    imagesc(abs(cwtmexh.cfs(:,:,1,1,angn)));
    title(['Mexican hat at ' angz(angn) 'radians']);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



Scale: Index 1 - Value 1.000 -- Angle: Index 1 - Value 0 pi [rad] = 0.00 [dgr]

Idx. Scale1 -- Value: 1.000

Idx. Angle1 -- Value: 0 pi

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);
```

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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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 | string scalar | structure | cell array

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

Syntax

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

Description

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

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.
- For N = 1, 2, and 3, the order N Symlet filters and order N Daubechies filters are identical. See “Extremal Phase” on page 1-238.

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 = 4.2590e-13
```

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

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

```
ans = 4.2590e-13
```

Symlet and Daubechies Scaling Filters

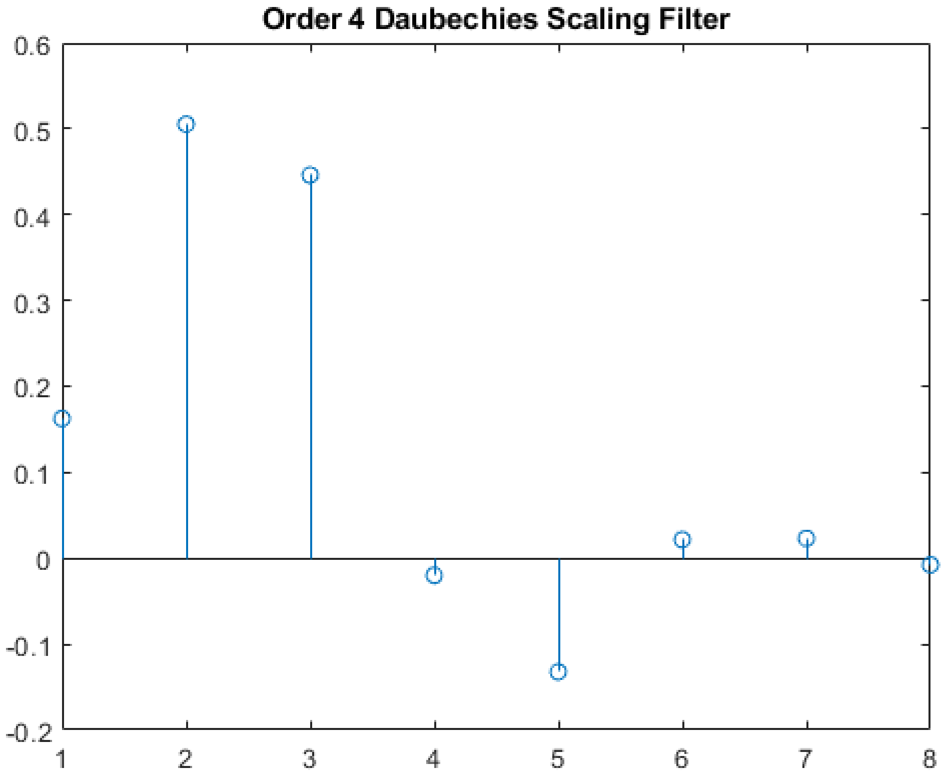
This example shows that symlet and Daubechies scaling filters of the same order are both solutions of the same polynomial equation.

Generate the order 4 Daubechies scaling filter and plot it.

```
wdb4 = dbaux(4)
```

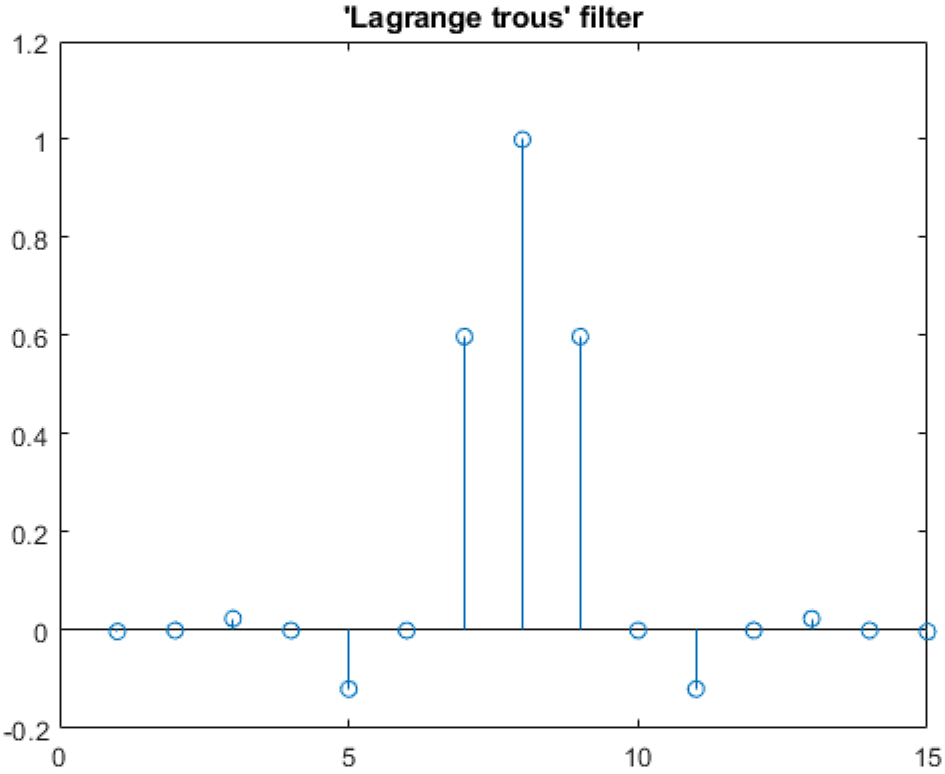
```
wdb4 = 1×8

    0.1629    0.5055    0.4461   -0.0198   -0.1323    0.0218    0.0233   -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')
```

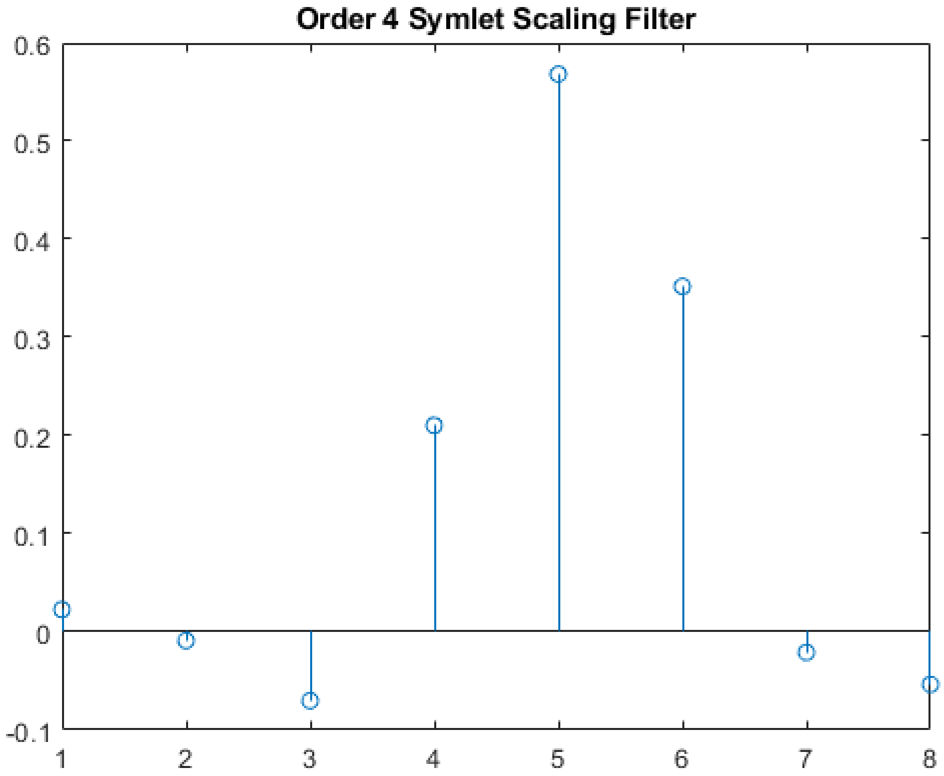


Generate wsym4, the order 4 symlet scaling filter and plot it. The Symlets are the "least asymmetric" Daubechies' wavelets obtained from another choice between the roots of P.

```
wsym4 = symaux(4)
```

```
wsym4 = 1×8

    0.0228   -0.0089   -0.0702    0.2106    0.5683    0.3519   -0.0210   -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 = 1.8677e-15
```

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.

Generate the scaling filter coefficients for the `db15` and `sym15` wavelets. Both wavelets have support of width 2 × 15 − 1 = 29.

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

Next, generate the scaling filter coefficients for the `coif5` wavelet. This wavelet also has support of width 6 × 5 − 1 = 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)
```

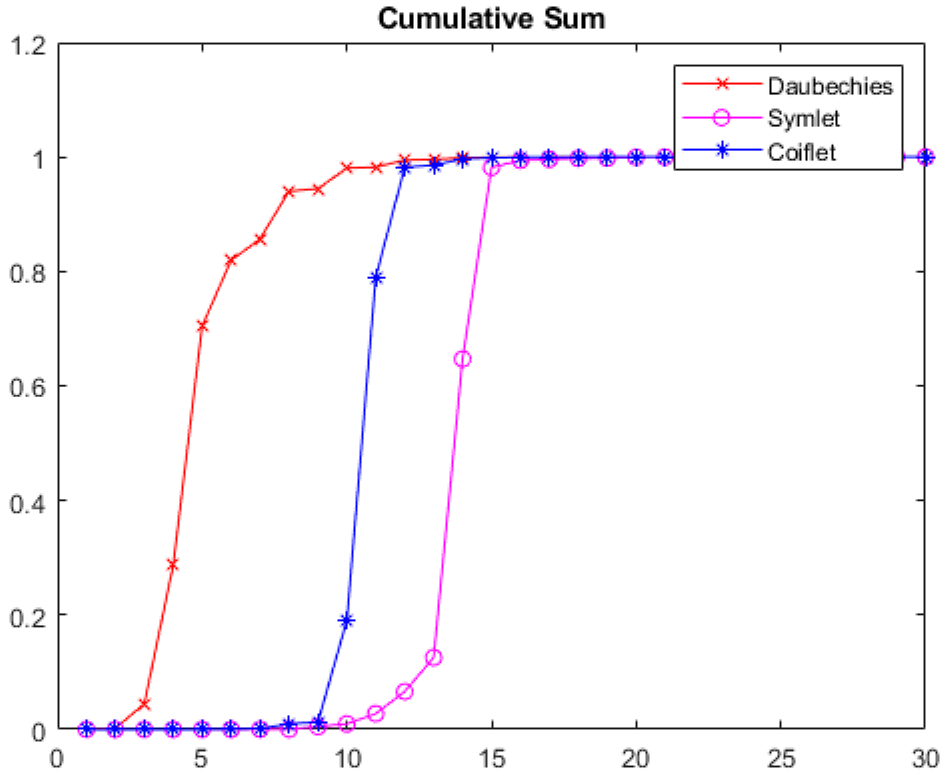
```
ans = 0
```

```
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 1-233 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.

More About

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 $k = 1,...,M$. Then for any other set of scaling coefficients $\{g_k\}$ resulting from a different choice of roots, the following inequality will hold for all $J = 1,...,M$:

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

The $\{h_k\}$ are sometimes called a *minimal delay filter* [2].

The polynomial equation mentioned above depends on the number of vanishing moments *N* for the wavelet. To construct a wavelet basis involves choosing roots of the equation. In the case of least asymmetric wavelets and extremal phase wavelets for orders 1, 2, and 3, there are effectively no choices to make. For *N* = 1, 2, and 3, the `db`*N* and `sym`*N* filters are equal. The example "Symlet and Daubechies Scaling Filters" on page 1-234 shows that two different scaling filters can satisfy the same polynomial equation. For additional information, see Daubechies [1].

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*-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 2*N*.

  The corresponding polynomial has *N* zeros located at −1 and *N−1* zeros less than 1 in modulus.

Note that other methods can be used; see various solutions of the spectral factorization problem in Strang-Nguyen [4] (p. 157).

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: 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

symaux | dbwavf | wfilters

Introduced before R2006a

dbwavf

Daubechies wavelet filter

Syntax

```
f = dbwavf(wname)
```

Description

`f = dbwavf(wname)` returns the scaling filter associated with the Daubechies wavelet specified by `wname`. `f` is a real-valued vector.

Examples

Scaling Filter Associated With Daubechies Wavelet

Specify the order 4 Daubechies wavelet.

```
wname = 'db4';
```

Compute the corresponding scaling filter.

```
f = dbwavf(wname);
f'
```

```
ans = 8×1

    0.1629
    0.5055
    0.4461
   -0.0198
   -0.1323
    0.0218
    0.0233
   -0.0075
```

Input Arguments

`wname` — Daubechies wavelet
`'dbN'`

Daubechies wavelet with *N* vanishing moments, where *N* is a positive integer in the closed interval [1, 45].

See Also

`dbaux` | `waveinfo` | `wfilters`

Introduced before R2006a

ddencmp

Default values for denoising or compression

Syntax

```
[thr,sorh,keepapp] = ddencmp(in1,in2,x)
[ ___ ,crit] = ddencmp(in1,'wp',x)
```

Description

`ddencmp` returns default values for denoising or compression for the critically sampled discrete wavelet or wavelet packet transform.

`[thr,sorh,keepapp] = ddencmp(in1,in2,x)` returns default values for denoising or compression, using wavelets or wavelet packets, of the input data `x`. `x` is a real-valued vector or 2-D matrix. `thr` is the threshold, and `sorh` indicates soft or hard thresholding. `keepapp` can be used as a flag to set whether or not the approximation coefficients are thresholded.

- Set `in1` to `'den'` for denoising or `'cmp'` for compression.
- Set `in2` to `'wv'` to use wavelets or `'wp'` to use wavelet packets.

`[ ___ ,crit] = ddencmp(in1,'wp',x)` also returns the entropy type, `crit`. See `wentropy` for more information.

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. Change the DWT extension mode to periodic. Set the random number generator to the default initial settings for reproducible results.

```
origmode = dwtmode('status','nodisplay');
dwtmode('per','nodisp')
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`.

```
thr
```

```
thr = 3.3639
```

```
thresh
```

```
thresh = 3.3639
```

Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

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 DWT extension mode to periodic. Set the random number generator to the default initial settings for reproducible results.

```
origmode = dwtmode('status','nodisplay');
dwtmode('per','nodisp')
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)
```

```
thr = 0.6424

sorh = 
'h'

keepapp = 1

crit = 
'threshold'
```

Compare with the default values returned for denoising.

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

```
thr = 4.1074

sorh = 
'h'

keepapp = 1

crit = 
'sure'
```

Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

Input Arguments

`in1` — Purpose

`'den'` | `'cmp'`

Purpose of `ddencmp` output, specified as:

- `'den'` — Denoising
- `'cmp'` — Compression

`in2` — Transform type

`'wv'` | `'wp'`

Transform type to be used for denoising or compression, specified as:

- `'wv'` — Critically sampled discrete wavelet transform. This output can be used with `wdencmp`.
- `'wp'` — Critically sampled wavelet packet transform. This output can be used with `wpdencmp`.

`x` — Input data

real-valued vector or matrix

Input data to be denoised or compressed, specified as a real-valued vector or 2-D matrix.

Data Types: `double`

Output Arguments

`thr` — Threshold

real number

Threshold for denoising or compression, returned as a real number. Use this output with `wdencmp` or `wpdencmp`.

`sorh` — Thresholding type

character

Thresholding type for denoising or compression, returned as a character.

- `'s'` — Soft thresholding
- `'h'` — Hard thresholding

Use this output with `wdencmp` or `wpdencmp`.

keepapp — Threshold approximation setting

1 (default)

Threshold approximation setting, returned as 1. Use this output with `wdencmp` or `wpdencmp`. If `keepapp = 1`, the approximation coefficients are not thresholded.

`crit` — Entropy type

character vector

Entropy type when denoising or compressing with wavelet packets, returned as a character vector. Use this output only with `wpdencmp`. See `wentropy` for more information.

References

[1] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*, Vol. 42, Number 3, pp. 613–627, 1995.

[2] Donoho, D. L., and Johnstone, I. M. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*, Vol. 81, pp. 425–455, 1994.

[3] Donoho, D. L., and I. M. Johnstone. "Ideal denoising in an orthonormal basis chosen from a library of bases." *Comptes Rendus Acad. Sci. Paris, Ser. I*, Vol. 319, pp. 1317–1322, 1994.

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` | `wdenoise`

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.

Use `dtfilters` to create the first-stage Farras analysis filters and 6-tap Kingsbury Q-shift analysis filters for the subsequent stages of the multiresolution analysis.

```
df = dtfilters('dtf1');
```

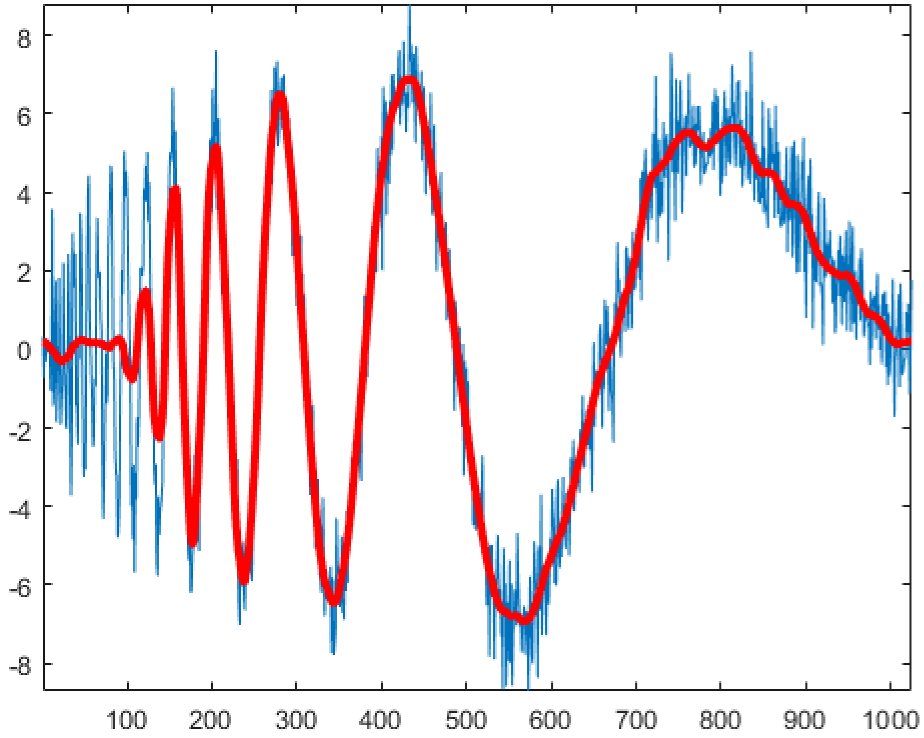
The Farras and Kingsbury filters are in `df{1}` and `df{2}`, respectively. Load the noisy Doppler signal and obtain the complex dual-tree wavelet transform down to level 4.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,4,df{1},df{2});
```

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
```



Using the output of `dtfilters`, or the filter name itself, in `dddtree` is preferable to manually entering truncated filter coefficients. To demonstrate the negative impact on signal reconstruction, create truncated versions of the Farras and Kingsbury analysis filters. Display the differences between the truncated and original filters.

```
  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];

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];

max(max(abs(df{1}{1}-Faf{1})))
```

```
ans = 2.6792e-05
```

```
max(max(abs(df{1}{2}-Faf{2})))
```

```
ans = 2.6792e-05
```

```
max(max(abs(df{2}{1}-af{1})))
```

```
ans = 3.6160e-05
```

```
max(max(abs(df{2}{2}-af{2})))
```

```
ans = 3.6160e-05
```

Obtain the complex dual-tree wavelet transform down to level 4 using the truncated filters. Take the inverse transform and compare the reconstruction with the original signal.

```
wt = dddtree('cplxdt',noisdopp,4,Faf,af);
xrec = idddtree(wt);
max(abs(noisdopp-xrec))
```

```
ans = 0.0024
```

Do the same using the filter name. Confirm the difference is smaller.

```
wt = dddtree('cplxdt',noisdopp,4,'dtf1');
xrec = idddtree(wt);
max(abs(noisdopp-xrec))
```

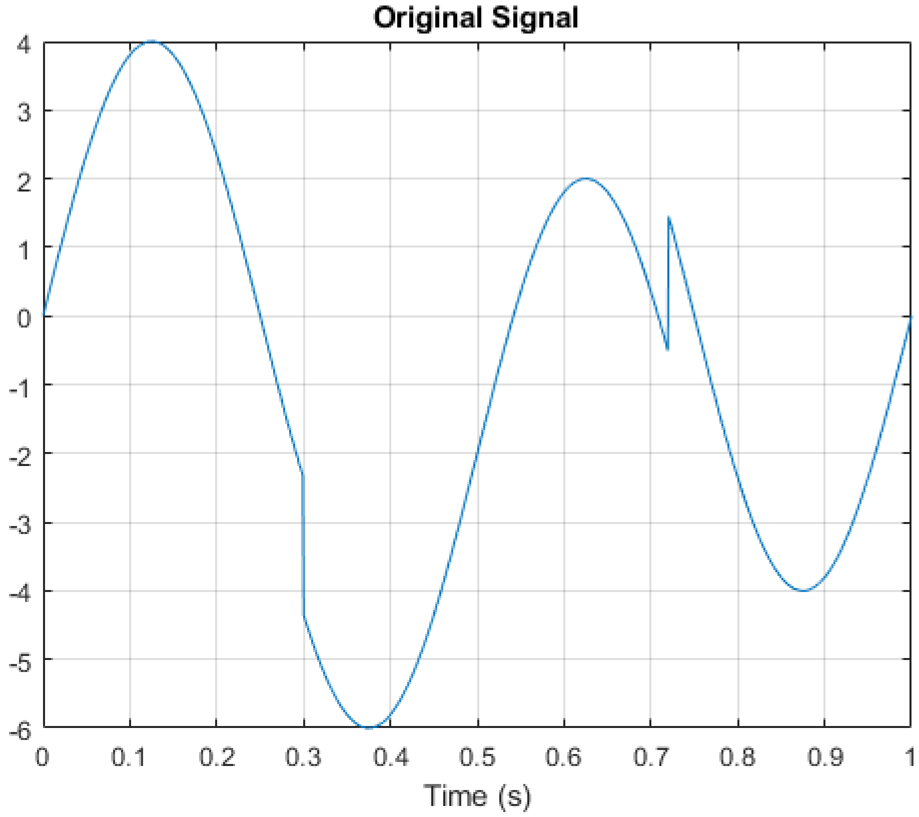
```
ans = 2.1893e-07
```

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.

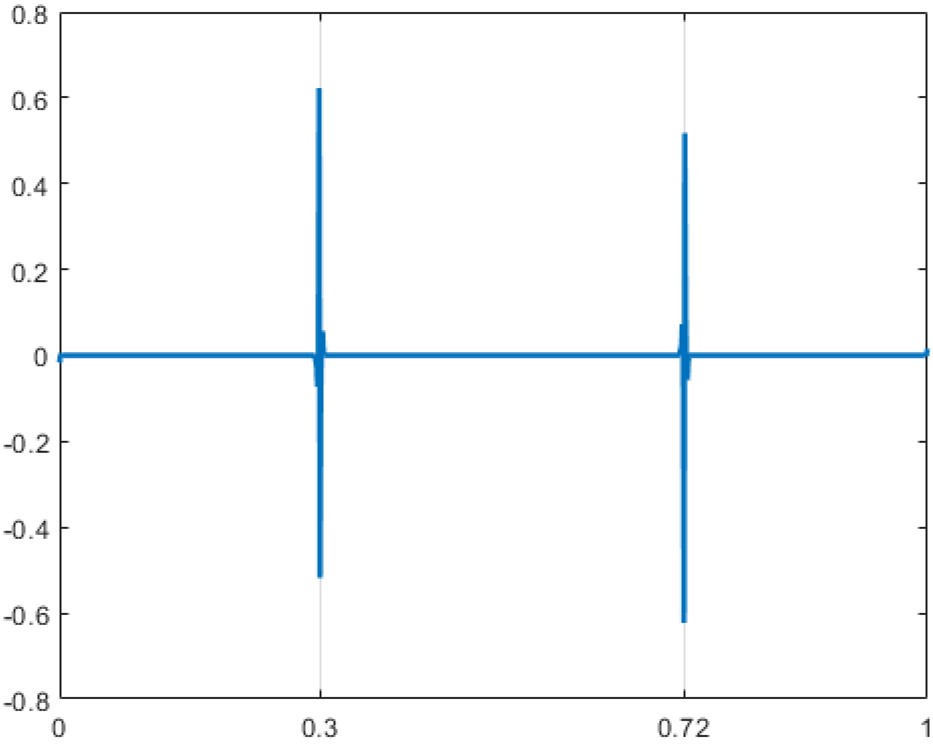
Create a signal consisting of a 2-Hz sine wave with a duration of 1 second. The sine wave has discontinuities at 0.3 and 0.72 seconds.

```
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('Time (s)')
title('Original Signal')
grid on
```



Obtain the double-density wavelet transform of the signal. Reconstruct an approximation based on the level-one detail coefficients by first setting the lowpass (scaling) coefficients equal to 0. Plot the result. Observe features in the reconstruction align with the signal discontinuities.

```
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')
```

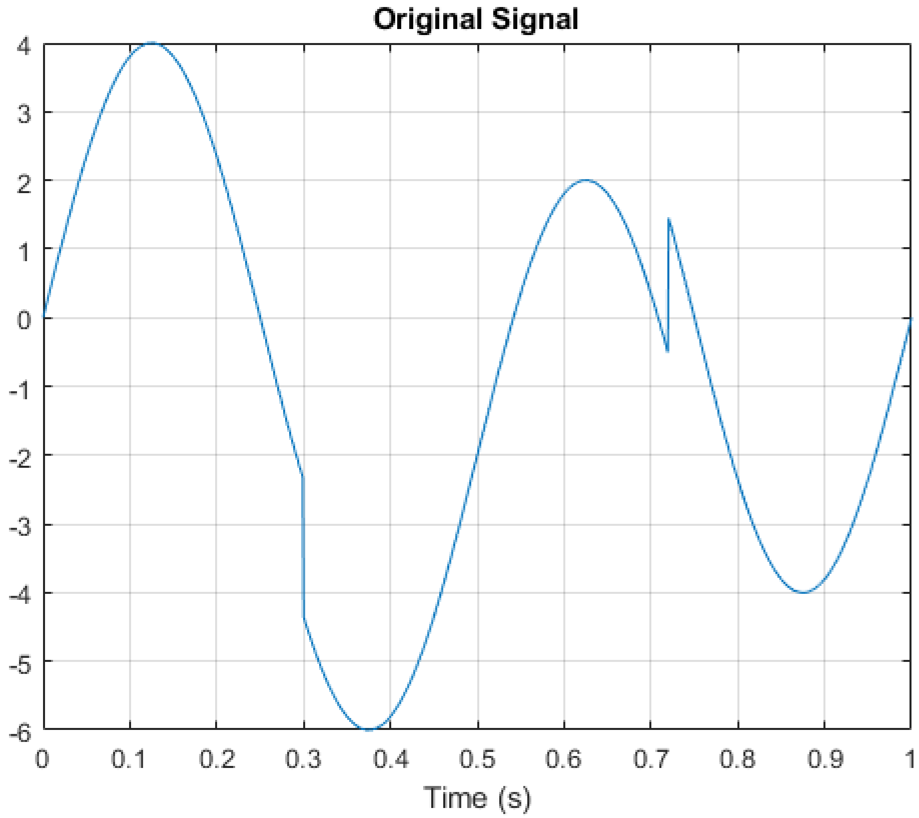


First-Level Detail Coefficients Approximation — Complex Dual-Tree

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

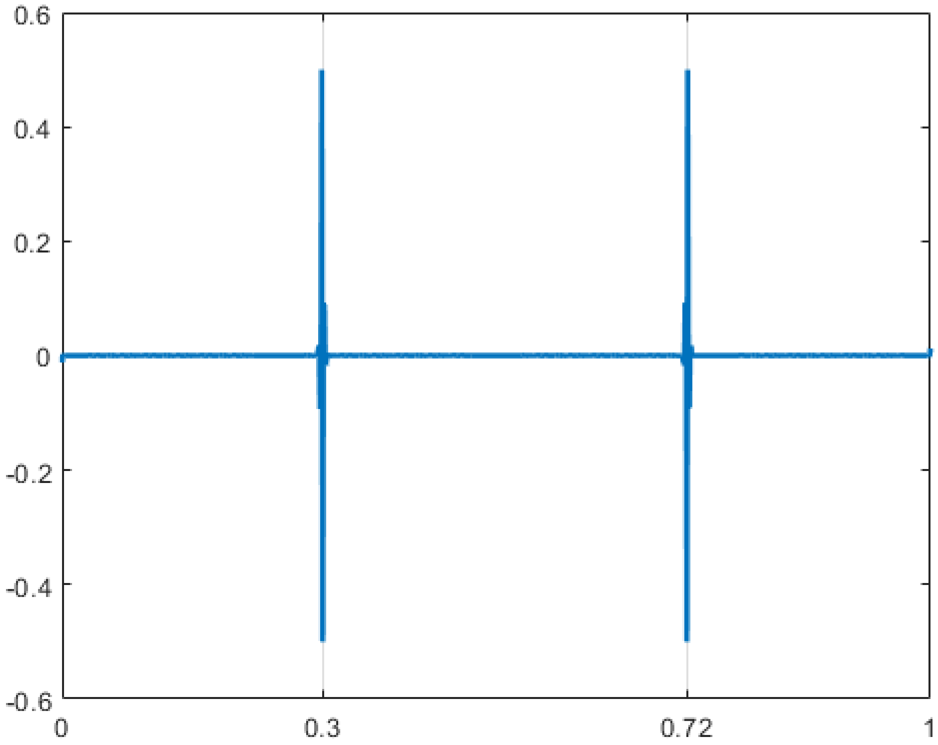
Create a signal consisting of a 2-Hz sine wave with a duration of 1 second. The sine wave has discontinuities at 0.3 and 0.72 seconds.

```
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('Time (s)')
title('Original Signal')
grid on
```



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')
```



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 | string scalar

Filter name, specified as a character vector or string scalar. 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 | string scalar

First-stage filter name, specified as a character vector or string scalar. 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 | string scalar

Filter name for stages > 1, specified as a character vector or string scalar. 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}(:,:,m)` 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}(:,:,m)` are the lowpass, or scaling, coefficients.

See Also

`wfilters` | `dddtree2` | `dddtreecfs` | `dtfilters` | `idddtree` | `dualtree` | `dualtree2`

Topics

“Dual-Tree Complex Wavelet Transforms”

"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 in multiple ways.

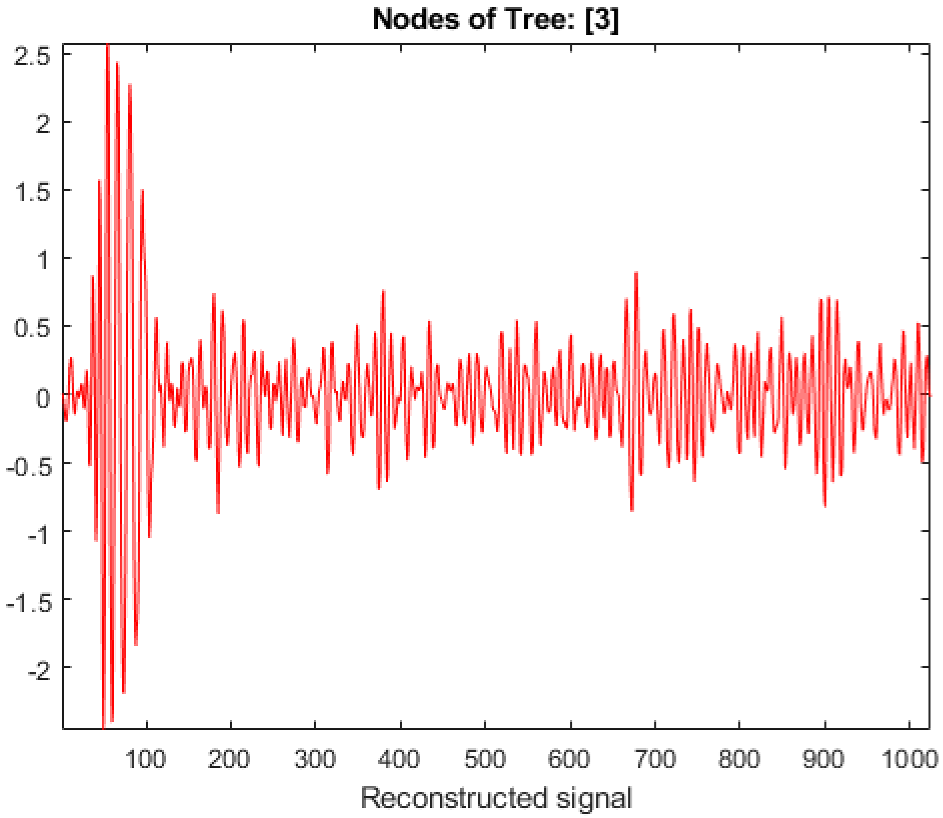
Load the noisy Doppler signal. Obtain the complex dual-tree transform down to level 3.

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

```
wt = struct with fields:
       type: 'cplxdt'
      level: 3
    filters: [1x1 struct]
        cfs: {1x4 cell}
```

Plot a reconstruction of the original signal based on the level-three detail coefficients with `outputspec` set to `'scale'`.

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



The output `xr` is a 1-by-1 cell array. Generate the same reconstruction by using `'cumind'` and the level-three tree nodes. The first element of each vector in the cell array denotes the level, and the second element denotes the tree. Confirm the reconstructions are identical.

```
outputindices = {[3 1];[3 2]};
xr2 = dddtreecfs('r',wt,'cumind',outputindices);
max(abs(xr2-xr{1}))
```

```
ans = 0
```

The output `xr2` is the same datatype as the original signal.

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')
```

```
wt = struct with fields:
       type: 'cplxdt'
      level: 3
    filters: [1×1 struct]
        cfs: {[1×512×2 double]  [1×256×2 double]  [1×128×2 double]  [1×128×2 double]}
```

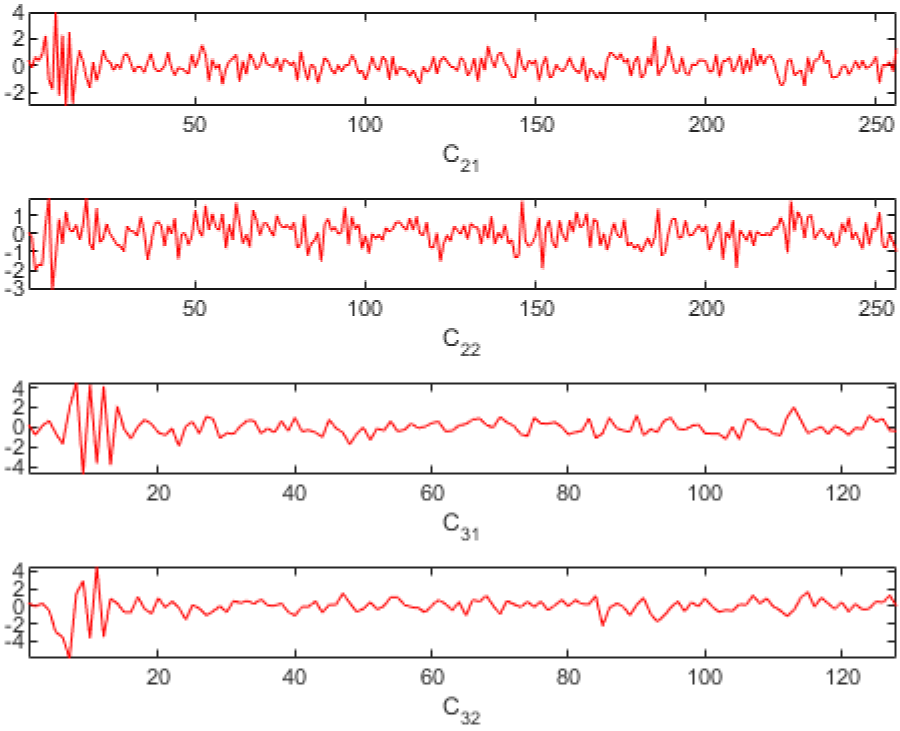
Create a cell array of vectors to obtain the second- and third-level detail coefficients from each of the wavelet filter bank trees.

```
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.

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



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

```
max(abs(wt.cfs{2}(1,:,1)-detailCoeffs{1}))
```

```
ans = 0
```

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');
```

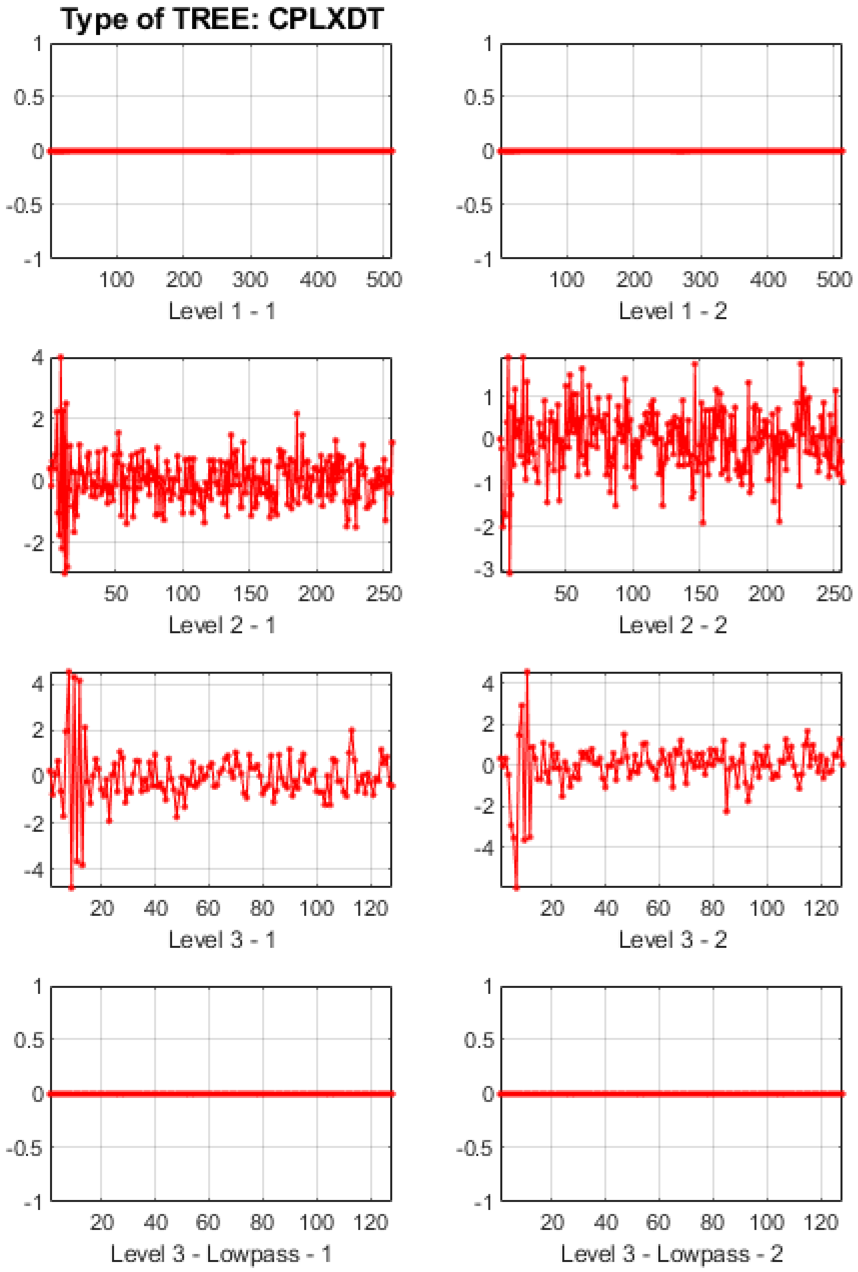
Create a cell array of vectors to obtain the second- and third-level detail coefficients from each of the wavelet filter bank trees.

```
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 second- and third-level detail coefficients.

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



Generate a reconstruction based on the second- and third-level detail coefficients.

```
xr = idddtree(out);
```

Generate two reconstructions, based on the second- and third-level detail coefficients. Confirm the sum of the two reconstructions is identical to `xr`.

```
xr2 = dddtreecfs('r',wt,'scale',{2;3});
max(abs(xr-(xr2{1}+xr2{2})))
```

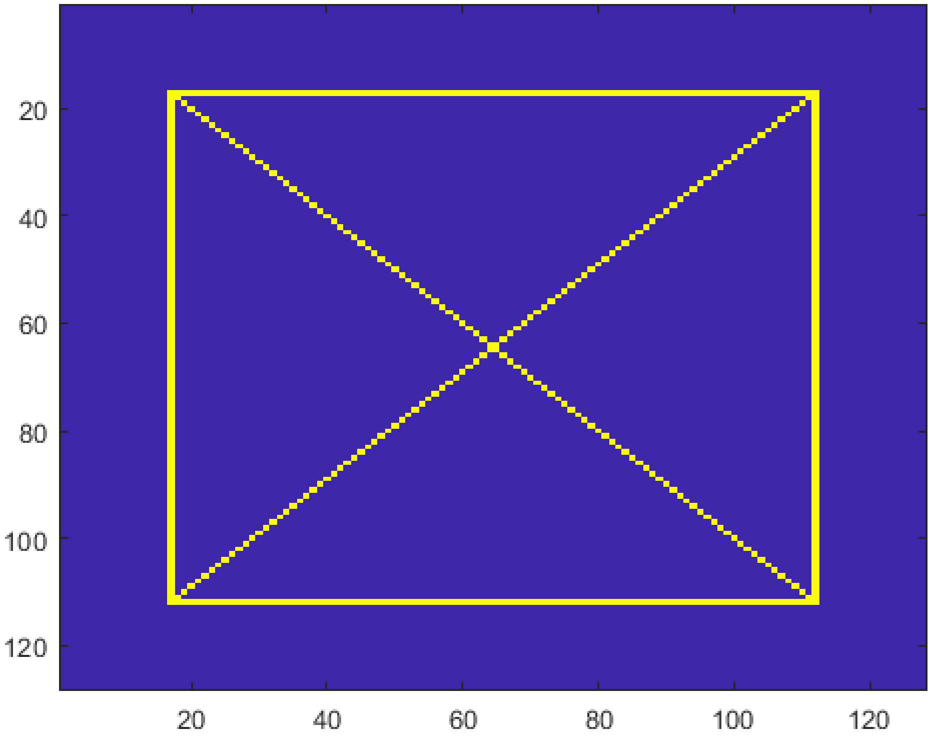
```
ans = 8.8818e-16
```

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)
```

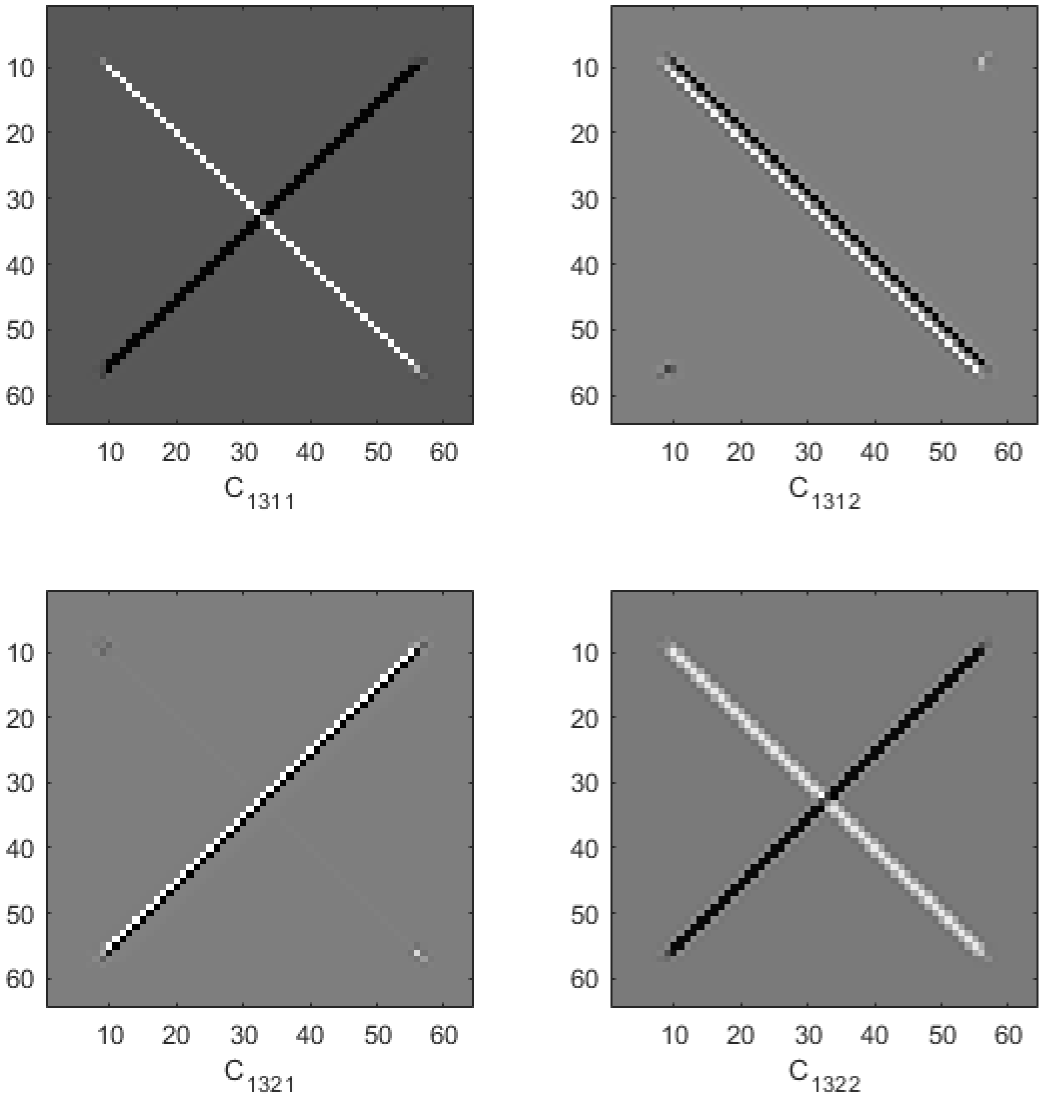


Obtain the complex dual-tree wavelet transform down to level 3.

```
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. Do this by creating a cell array of vectors specifying the tree nodes containing the diagonal details. The first element in the vector specifies the level. The three remaining elements specify the orientation, wavelet tree, and real and imaginary parts, respectively (see `dddtree2`).

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



Distribution of Analysis Coefficients in Wavelet Tree Structure

This example shows how the analysis coefficients are distributed, depending on the transform, in the tree output of `dddtree` and `dddtree2`.

1-D Wavelet Transforms

Load in the noisy Doppler signal. Generate a four-level wavelet decomposition of the signal for each type of transform. Depending on the transform, different dimensions of the coefficient arrays correspond to orientation, wavelet tree, or real and imaginary parts.

Critically Sampled Discrete Wavelet Transform

```
load noisdopp
wt = dddtree('dwt',noisdopp,4,'sym4')
```

```
wt = struct with fields:
       type: 'dwt'
      level: 4
    filters: [1x1 struct]
        cfs: {1x5 cell}
```

This is the usual nonredundant discrete wavelet transform. The first four elements of `wt.cfs` are the wavelet coefficients. The fifth element are the scaling coefficients.

Double-Density Wavelet Transform

```
wt = dddtree('ddt',noisdopp,4,'filters1')
```

```
wt = struct with fields:
       type: 'ddt'
      level: 4
    filters: [1x1 struct]
        cfs: {1x5 cell}
```

The third dimension of the 3-D wavelet coefficient arrays corresponds to the tree. The fifth element are the scaling coefficients.

Dual-Tree Complex Wavelet Transform

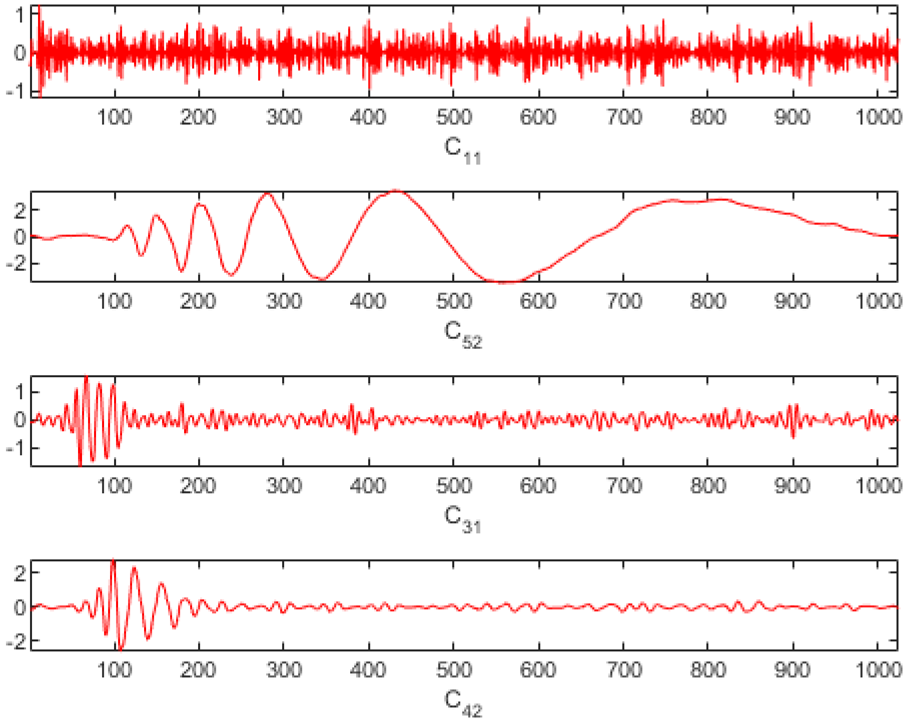
```
wt = dddtree('cplxdt',noisdopp,4,'dtf1')
```

```
wt = struct with fields:
       type: 'cplxdt'
      level: 4
    filters: [1x1 struct]
        cfs: {1x5 cell}
```

The third dimension of all the 3-D arrays in `cfs` corresponds to the real and imaginary parts. The first four elements of `cfs` are the wavelet coefficients, and `cfs{5}` are the scaling coefficients.

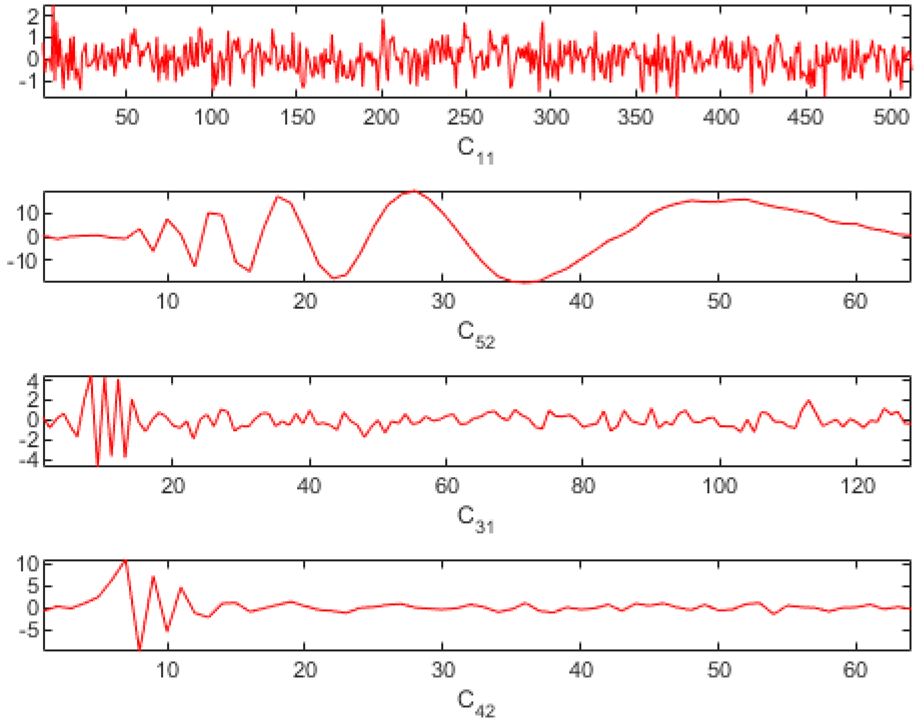
Reconstruct signals from the coefficients at the tree nodes `[1 1]`, `[5 2]`, `[3 1]`, and `[4 2]`. Plot the signals. The output is a cell array containing the reconstructions. The reconstructions are the same length as the original signal.

```
outputindices = {[1 1];[5 2];[3 1];[4 2]};
XR = dddtreecfs('r',wt,'plot','ind',outputindices);
```



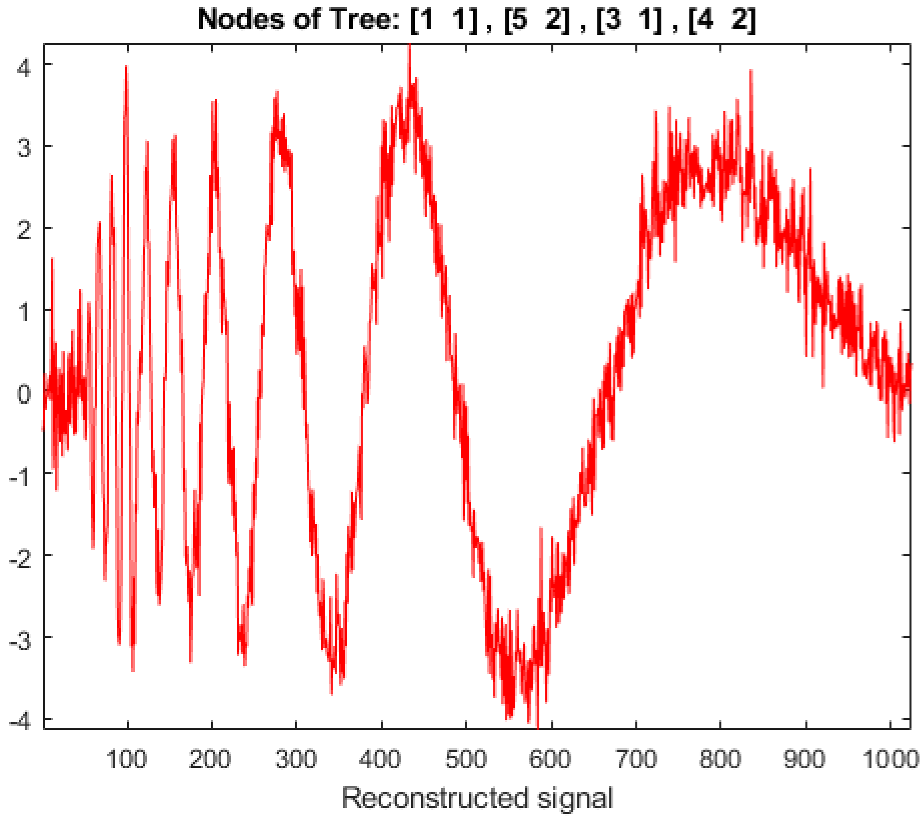
Extract and plot the coefficients used to reconstruct the signals. The output is a cell array containing the coefficients of respective length: 512, 64, 128, and 64.

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

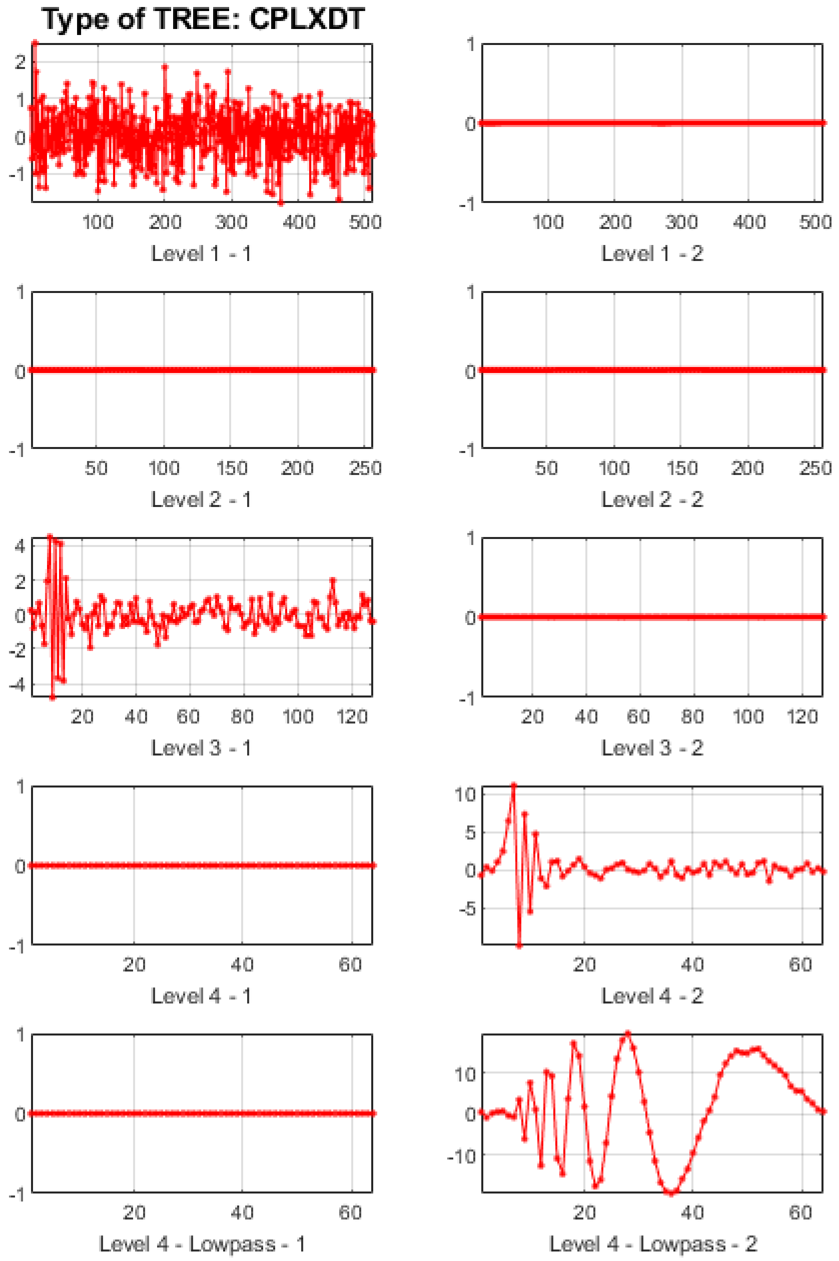


Now use `'cumind'` instead of `'ind'`. The output `XR` is a signal of length 1024 in the first case, and a `'cplxdt'` dual-tree in the second one.

```
XR = dddtreecfs('r',wt,'plot','cumind',outputindices);
```



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



Double-Density Dual-Tree Complex Wavelet Transform

```
wt = dddtree('cplxdddt',noisdopp,4,'dddtf1')
```

```
wt = struct with fields:
       type: 'cplxdddt'
      level: 4
```

```
    filters: [1x1 struct]
        cfs: {1x5 cell}
```

The third dimension of the 4-D wavelet coefficient arrays corresponds to the tree. The fourth dimension in the 4-D wavelet coefficient arrays and third dimension in the 3-D scaling coefficients array corresponds to the real and imaginary parts.

2-D Wavelet Transforms

Load in the 256-by-256 mask image. Generate a two-level wavelet decomposition of the image for each type of transform. Observe the dimensions of the output coefficients.

Critically Sampled Discrete Wavelet Transform

```
load mask
im = X;
wt = dddtree2('dwt',im,3,'sym4')
```

```
wt = struct with fields:
       type: 'dwt'
      level: 3
    filters: [1x1 struct]
        cfs: {1x4 cell}
      sizes: [10x2 double]
```

This is the usual nonredundant 2-D discrete wavelet transform. The third dimension in the 3-D wavelet coefficient arrays corresponds to the orientation. The scaling coefficients are the last element of `cfs`.

Real Oriented Dual-Tree Wavelet Transform

```
wt = dddtree2('realdt',im,3,'dtf1')
```

```
wt = struct with fields:
       type: 'realdt'
      level: 3
    filters: [1x1 struct]
        cfs: {1x4 cell}
      sizes: [11x2 double]
```

The fourth dimension in the 4-D wavelet coefficient arrays and third dimension in the 3-D scaling coefficients array correspond to the tree. The third dimension in the 4-D wavelet coefficient arrays corresponds to orientation.

Complex Oriented Dual-Tree Wavelet Transform

```
wt = dddtree2('cplxdt',im,3,'dtf1')
```

```
wt = struct with fields:
       type: 'cplxdt'
      level: 3
    filters: [1x1 struct]
        cfs: {[5-D double]  [5-D double]  [5-D double]  [32x32x2x2 double]}
      sizes: [11x2 double]
```

```
[size(wt.cfs{1});size(wt.cfs{2});size(wt.cfs{3})]
```

```
ans = 3×5

   128   128     3     2     2
    64    64     3     2     2
    32    32     3     2     2
```

The third dimension of the 5-D wavelet coefficient arrays represents the orientation. The fourth dimension in the 5-D arrays and third dimension in the 4-D scaling coefficients array represents the tree. The fifth dimension in the 5-D arrays and fourth dimension in the 4-D array represents the real and imaginary parts.

Double-Density Wavelet Transform

```
wt = dddtree2('ddt',im,3,'filters1')
```

```
wt = struct with fields:
       type: 'ddt'
      level: 3
    filters: [1x1 struct]
        cfs: {1x4 cell}
      sizes: [26x2 double]
```

The third dimension in the 3-D wavelet coefficient arrays represents the orientation.

Real Oriented Double-Density Wavelet Transform

```
wt = dddtree2('realdddt',im,3,'self1')
```

```
wt = struct with fields:
       type: 'realdddt'
      level: 3
    filters: [1x1 struct]
        cfs: {1x4 cell}
      sizes: [26x2 double]
```

The third dimension in the 4-D wavelet coefficient arrays represents the orientation. The fourth dimension in the 4-D arrays and third dimension in the 3-D scaling coefficients array represent the tree.

Complex Oriented Double-Density Wavelet Transform

```
wt = dddtree2('cplxdddt',im,3,'self1')
```

```
wt = struct with fields:
       type: 'cplxdddt'
      level: 3
    filters: [1x1 struct]
        cfs: {[5-D double]  [5-D double]  [5-D double]  [32x32x2x2 double]}
      sizes: [26x2 double]
```

```
[size(wt.cfs{1}) ; size(wt.cfs{2}) ; size(wt.cfs{3})]
```

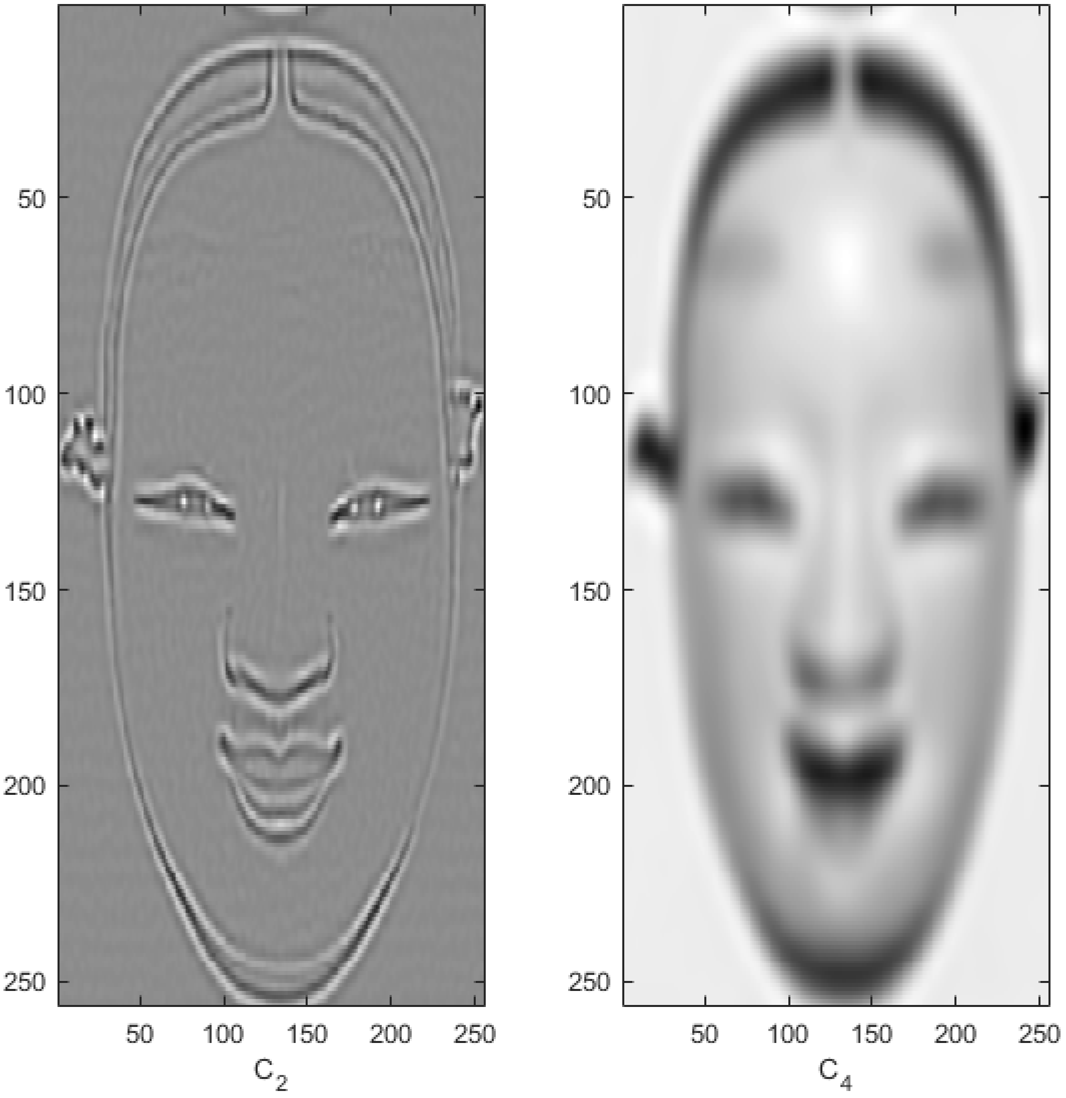
```
ans = 3×5
```

```
   128   128     8     2     2
    64    64     8     2     2
    32    32     8     2     2
```

The third dimension of the 5-D wavelet coefficient arrays represents the orientation. The fourth dimension in the 5-D arrays and third dimension in the 4-D scaling coefficients array represents the tree. The fifth dimension in the 5-D arrays and fourth dimension in the 4-D array represents the real and imaginary parts.

Reconstruct and plot two images based on the second-level detail coefficients and scaling coefficients, respectively.

```
XR = dddtreecfs('r',wt,'plot','scale',{2;4});
```



The output XR is a cell array containing both 256-by-256 images.

Extract the coefficients used to produce the two images. The output is a cell array containing two dual-tree structures, one for each specified scale.

```
XR = dddtreecfs('e',wt,'scale',{2;4});
XR{1}
```

```
ans = struct with fields:
       type: 'cplxdddt'
      level: 3
    filters: [1x1 struct]
        cfs: {[5-D double]  [5-D double]  [5-D double]  [32x32x2x2 double]}
      sizes: [26x2 double]
```

```
XR{2}
```

```
ans = struct with fields:
       type: 'cplxdddt'
      level: 3
    filters: [1x1 struct]
        cfs: {[5-D double]  [5-D double]  [5-D double]  [32x32x2x2 double]}
      sizes: [26x2 double]
```

Confirm the only nonzero coefficients in each structure contained in XR are the level-two wavelet coefficients and scaling coefficients, respectively.

```
dtInd = 1;
[max(abs(XR{dtInd}.cfs{1}(:)));max(abs(XR{dtInd}.cfs{2}(:)));...
    max(abs(XR{dtInd}.cfs{3}(:)));max(abs(XR{dtInd}.cfs{4}(:)))]
```

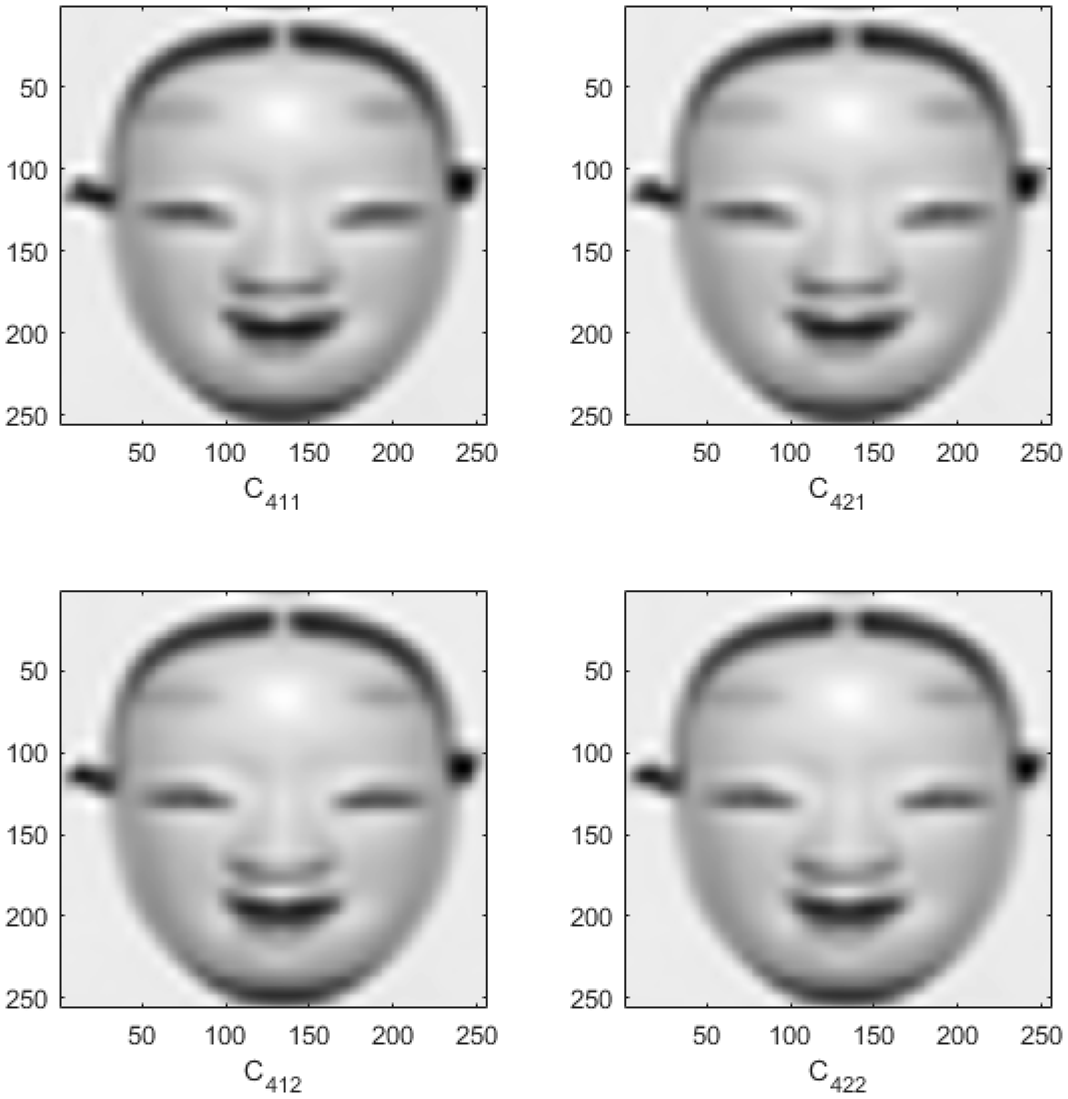
```
ans = 4×1

         0
  143.9924
         0
         0
```

```
dtInd = 2;
[max(abs(XR{dtInd}.cfs{1}(:)));max(abs(XR{dtInd}.cfs{2}(:)));...
    max(abs(XR{dtInd}.cfs{3}(:)));max(abs(XR{dtInd}.cfs{4}(:)))]
```

```
ans = 4×1
10^3 ×

         0
         0
         0
    1.0545
```

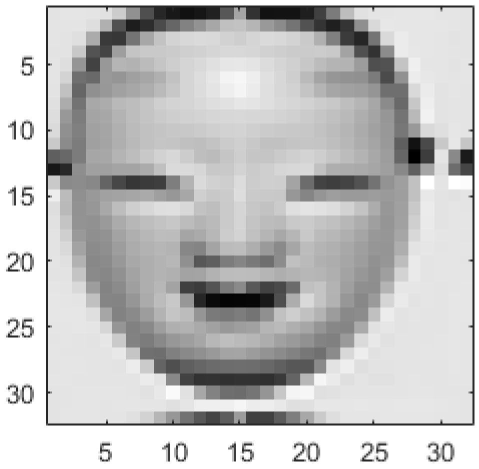
Use 'ind' to reconstruct and display the four images based on the four lowpass components, respectively.

```
outputindices = {[4 1 1];[4 2 1];[4 1 2];[4 2 2]};
XR = dddtreecfs('r',wt,'plot','ind',outputindices);
```

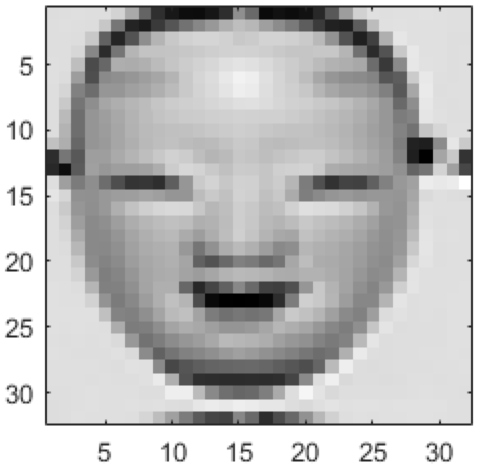


The output XR is a cell array containing the four images. Each image is 256-by-256. Display the coefficients used to reconstruct the images.

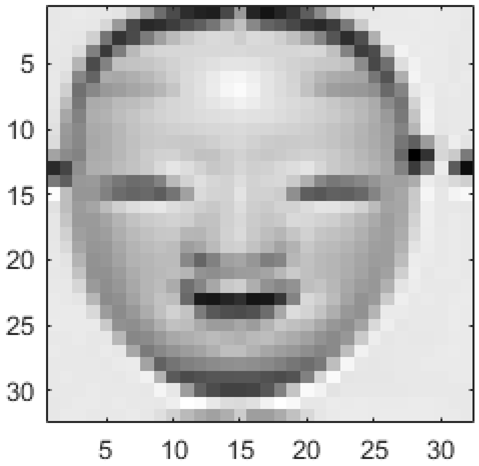
```
XR = dddtreecfs('e',wt,'plot','ind',outputindices);
```



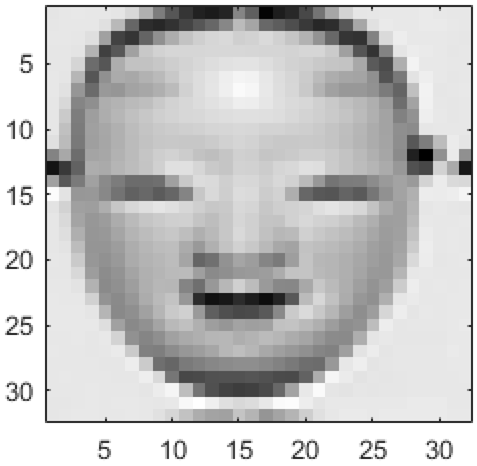
C_{411}



C_{421}



C_{412}



C_{422}

The output XR is a cell array containing the four lowpass components. Each component is 32-by-32.

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. For a description of the subsequent elements, see “Distribution of Analysis Coefficients in Wavelet Tree Structure” on page 1-265. For more information, see `dddtree` and `dddtree2`.

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` | `dualtree` | `dualtree2`

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 Farras 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 6-tap Kingsbury Q-shift 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];
```

To visualize the six directional wavelets, you will modify the wavelet coefficients of a four level real oriented dual-tree wavelet transform of an image of zeros. Create an image of zeros whose size satisfies the following constraints:

- The row and column dimensions are divisible by 2^4.
- The minimum of the row and column size must be greater than or equal to the product of the maximum length of the analysis filters and 2^3.

```
J = 4;
L = 3*2^(J+1);
N = L/2^J;
x = zeros(2*L,3*L);
[numrows,numcols] = size(x)
```

```
numrows = 192

numcols = 288
```

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

```
wt = dddtree2('realdt',x,J,Faf,af)
```

```
wt = struct with fields:
       type: 'realdt'
```

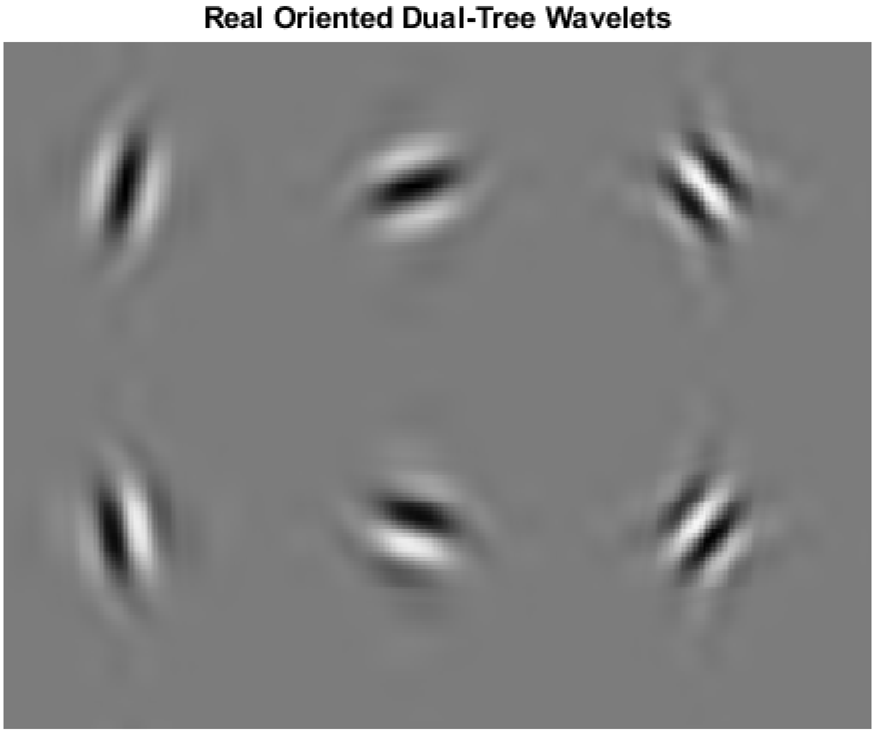
```
      level: 4
    filters: [1x1 struct]
        cfs: {1x5 cell}
      sizes: [14x2 double]
```

The fourth element in `wt.cfs` are the level 4 wavelet coefficients. Insert a 1 in one position of the six wavelet subbands (three orientations × two trees) at the coarsest scale, 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')
```

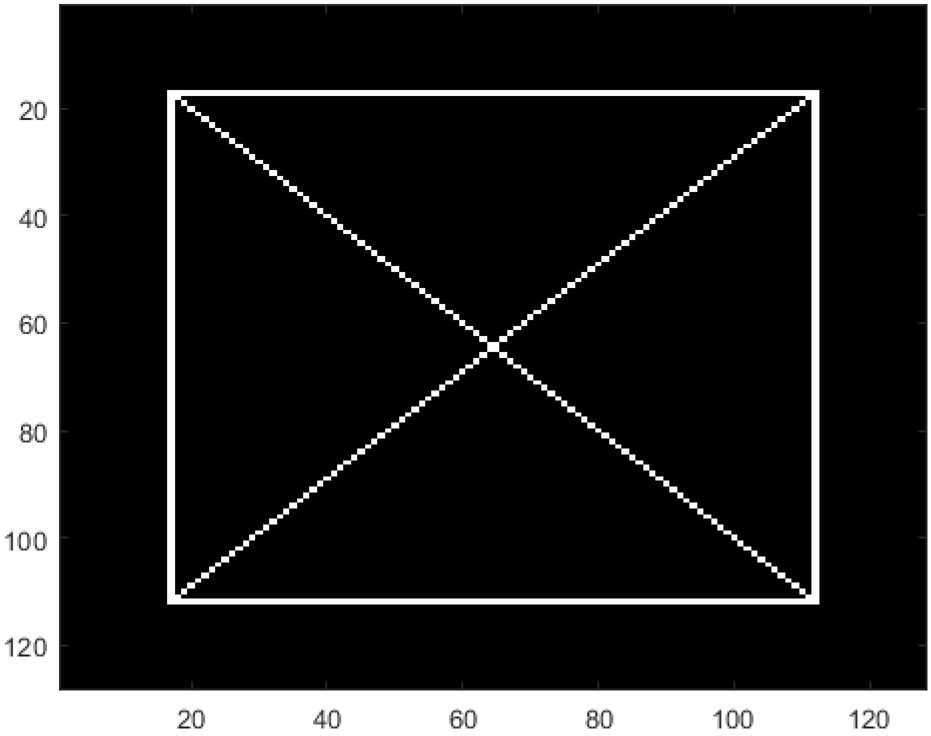


Double-Density Wavelet Transform

Obtain the double-density wavelet transform of an image.

Load the image and obtain the double-density wavelet transform using 6-tap filters (see `dtfilters`).

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

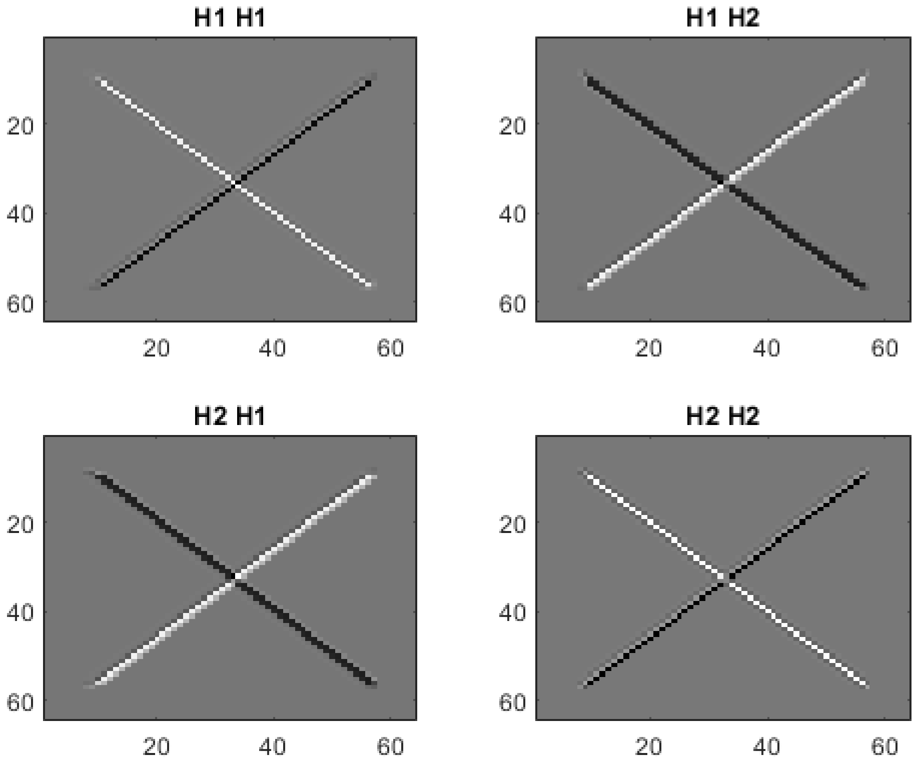


```
wt = dddtree2('ddt',xbox,1,'filters1')
```

```
wt = struct with fields:
       type: 'ddt'
      level: 1
    filters: [1x1 struct]
        cfs: {[64x64x8 double]  [64x64 double]}
      sizes: [10x2 double]
```

In the critically sampled 2-D discrete wavelet transform, there is one highpass filter. Filtering the rows and columns of the image with the highpass filter corresponds to extracting details in the diagonal orientation. In the double-density wavelet transform, there are two highpass filters, H1 and H2. Diagonally oriented details are extracted by filtering the image rows and columns with four combinations of the highpass filters. Visualize the diagonal details in the four wavelet highpass-highpass subbands.

```
H1H1 = wt.cfs{1}(:,:,4);
H1H2 = wt.cfs{1}(:,:,5);
H2H1 = wt.cfs{1}(:,:,7);
H2H2 = wt.cfs{1}(:,:,8);
subplot(2,2,1)
imagesc(H1H1);
title('H1 H1')
colormap gray;
subplot(2,2,2);
imagesc(H1H2);
title('H1 H2')
subplot(2,2,3)
imagesc(H2H1)
title('H2 H1')
subplot(2,2,4)
imagesc(H2H2)
title('H2 H2')
```

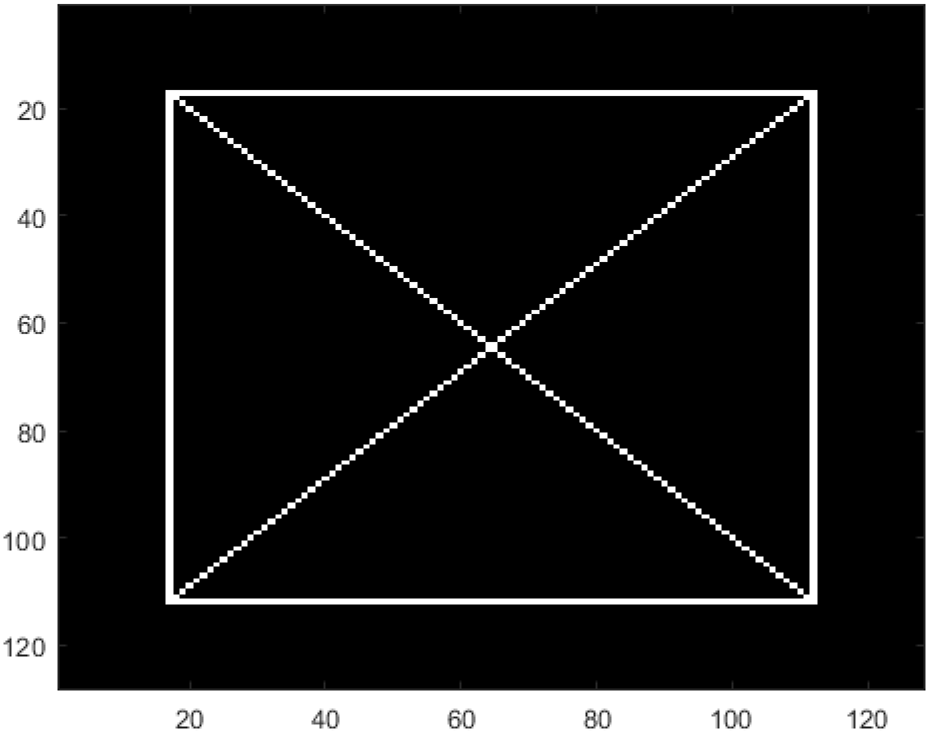


Complex Dual-Tree Wavelet Transform

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
```



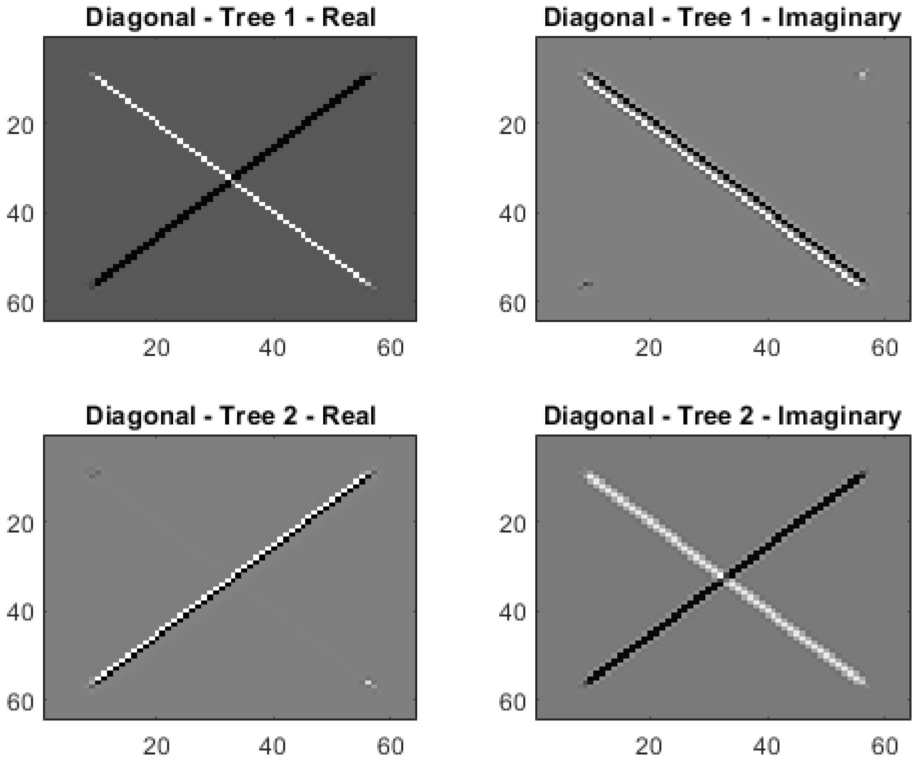
```
wt = dddtree2('cplxdt',xbox,1,'FSfarras','qshift10')
```

```
wt = struct with fields:
       type: 'cplxdt'
      level: 1
    filters: [1x1 struct]
        cfs: {[5-D double]  [64x64x2x2 double]}
      sizes: [5x2 double]
```

Obtain and display the diagonally oriented details from the two trees.

```
waveletcfs = wt.cfs{1};
subplot(2,2,1)
imagesc(waveletcfs(:,:,3,1,1))
title('Diagonal - Tree 1 - Real')
colormap gray
subplot(2,2,2)
imagesc(waveletcfs(:,:,3,1,2))
title('Diagonal - Tree 1 - Imaginary')
subplot(2,2,3)
imagesc(waveletcfs(:,:,3,2,1))
title('Diagonal - Tree 2 - Real')
subplot(2,2,4)
```

```
imagesc(waveletcfs(:,:,3,2,2))
title('Diagonal - Tree 2 - Imaginary')
```



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 | string scalar

Filter name, specified as a character vector or string scalar. 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 | string scalar

First-stage filter name, specified as a character vector or string scalar. 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 | string scalar

Filter name for stages > 1, specified as a character vector or string scalar. 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.
- `'realdt'` — `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}(:,:,k)` 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}(:,:,k,m)` are the lowpass, or scaling, coefficients.
- `'realdddt'` — `cfs{j}(:,:,d,k)`
  - j = 1,2,... `level` is the level.
  - d = 1,2,3,4,5,6,7,8 is the orientation.
  - k = 1,2 is the wavelet transform tree.
  - `cfs{level+1}(:,:,k)` are the lowpass, or scaling, coefficients.
- `'cplxdddt'` — `cfs{j}(:,:,d,k,m)`
  - j = 1,2,... `level` is the level.
  - d = 1,2,3,4,5,6,7,8 is the orientation.
  - k = 1,2 is the wavelet transform tree.
  - m = 1,2 are the real and imaginary parts.

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

Each orientation corresponds to a particular subband. The double-density transforms 'ddt', 'realddt', and 'cplxdddt' generate wavelet coefficients of eight orientations. The other transforms, 'dwt', 'realdt', and 'cplxdt' generate wavelet coefficients of three orientations. The correspondence to subbands is as follows.

typetree	Orientations
'dwt', 'realdt', 'cplxdt'	'L' and 'H', denote the lowpass and highpass filters, respectively. • d=1: 'LH' subband • d=2: 'HL' subband • d=3: 'HH' subband
'ddt', 'realdddt', 'cplxdddt'	'Lo', 'H1', and 'H2' denote the lowpass and two highpass filters, respectively. • d=1: 'Lo H1' subband • d=2: 'Lo H2' subband • d=3: 'H1 Lo' subband • d=4: 'H1 H1' subband • d=5: 'H1 H2' subband • d=6: 'H2 Lo' subband • d=7: 'H2 H1' subband • d=8: 'H2 H2' subband

sizes — Sizes of components
integer-valued matrix

Sizes of components in cfs, returned as an N-by-2 integer-valued matrix. The value of N depends on the level of wavelet decomposition and the type of wavelet decomposition: N = 2 + level × (number of orientations).

- cfs(1,:) = dimensions of input image.
- cfs(2+level,:) = dimensions of scaling coefficients.
- cfs(1+no×(i–1)+(1:no),:) = dimensions of level i detail coefficients, where no is the number of orientations.

See Also

dddtree | dddtreecfs | dtfilters | idddtree2 | dualtree | dualtree2

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

Introduced in R2013b

deletelift

Delete elementary lifting steps

Syntax

```
lsn = deletelift(lscheme)
lsn = deletelift(lscheme,loc)
```

Description

`lsn = deletelift(lscheme)` deletes the last elementary lifting step from the lifting scheme `lsc`.

`lsn = deletelift(lscheme,loc)` deletes the elementary lifting steps at the positions specified by `loc`.

Examples

Delete Elementary Lifting Steps

Create a lifting scheme associated with the `db3` wavelet.

```
lsc = liftingScheme('Wavelet','db3')
```

```
lsc = 
      Wavelet               : 'db3'
     LiftingSteps          : [4 × 1] liftingStep
     NormalizationFactors  : [2.3155 0.4319]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
           Type: 'predict'
    Coefficients: -2.4255
        MaxOrder: 0

           Type: 'update'
    Coefficients: [-0.0793 0.3524]
        MaxOrder: 1

           Type: 'predict'
    Coefficients: [2.8953 -0.5614]
        MaxOrder: -1

           Type: 'update'
    Coefficients: 0.0198
        MaxOrder: 2
```

The lifting scheme has four elementary lifting steps. Delete the second step.

```
lsc2 = deletelift(lsc,2)
```

```
lsc2 = 
      Wavelet               : 'custom'
     LiftingSteps          : [3 × 1] liftingStep
     NormalizationFactors  : [2.3155 0.4319]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
           Type: 'predict'
   Coefficients: -2.4255
       MaxOrder: 0

           Type: 'predict'
   Coefficients: [2.8953 -0.5614]
       MaxOrder: -1

           Type: 'update'
   Coefficients: 0.0198
       MaxOrder: 2
```

Input Arguments

lscheme — Lifting scheme
liftingScheme object

Lifting scheme, specified as a liftingScheme object.

loc — Positions
positive integer | vector

Positions of the elementary lifting steps, specified as a positive integer or vector of positive integers specified in the range [1,*N*], where *N* is the number of steps in the lifting scheme.

Example: lschemeB = deletelift(lschemeA,[2 4]) deletes the second and fourth steps from the lifting scheme lschemeA.

Data Types: double

Output Arguments

lsn — Lifting scheme
liftingScheme object

Lifting scheme, returned as a liftingScheme object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

liftingStep | liftingScheme | addlift

Introduced in R2021a

degree

Degree of Laurent polynomial

Syntax

```
deg = degree(P)
```

Description

`deg = degree(P)` returns the degree of the Laurent polynomial `P`.

If $P(z)$ is a Laurent polynomial $P(z) = \sum_{k=m}^{n} C_k z^k$, where m and n are integers, the degree of $P(z)$ is n-m.

Examples

Degree of Laurent Polynomials

Create two Laurent polynomials:

- $a(z) = z - 1$
- $b(z) = -2z^3 + 6z^2 - 7z + 2$

```
a = laurentPolynomial(Coefficients=[1 -1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[-2 6 -7 2],MaxOrder=3);
```

Multiply $a(z)$ and $b(z)$. Confirm the degree of the product is equal to the sum of the degrees of $a(z)$ and $b(z)$.

```
ab = a*b;
degree(ab)
```

```
ans = 4
```

```
degree(a)+degree(b)
```

```
ans = 4
```

Input Arguments

P — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

Output Arguments

deg — Degree
nonnegative integer

Degree of Laurent polynomial, returned as a nonnegative integer.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

depo2ind

Node depth-position to node index

Syntax

```
N = depo2ind(ORD,[D P])
```

Description

depo2ind is a tree-management utility.

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

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

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

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

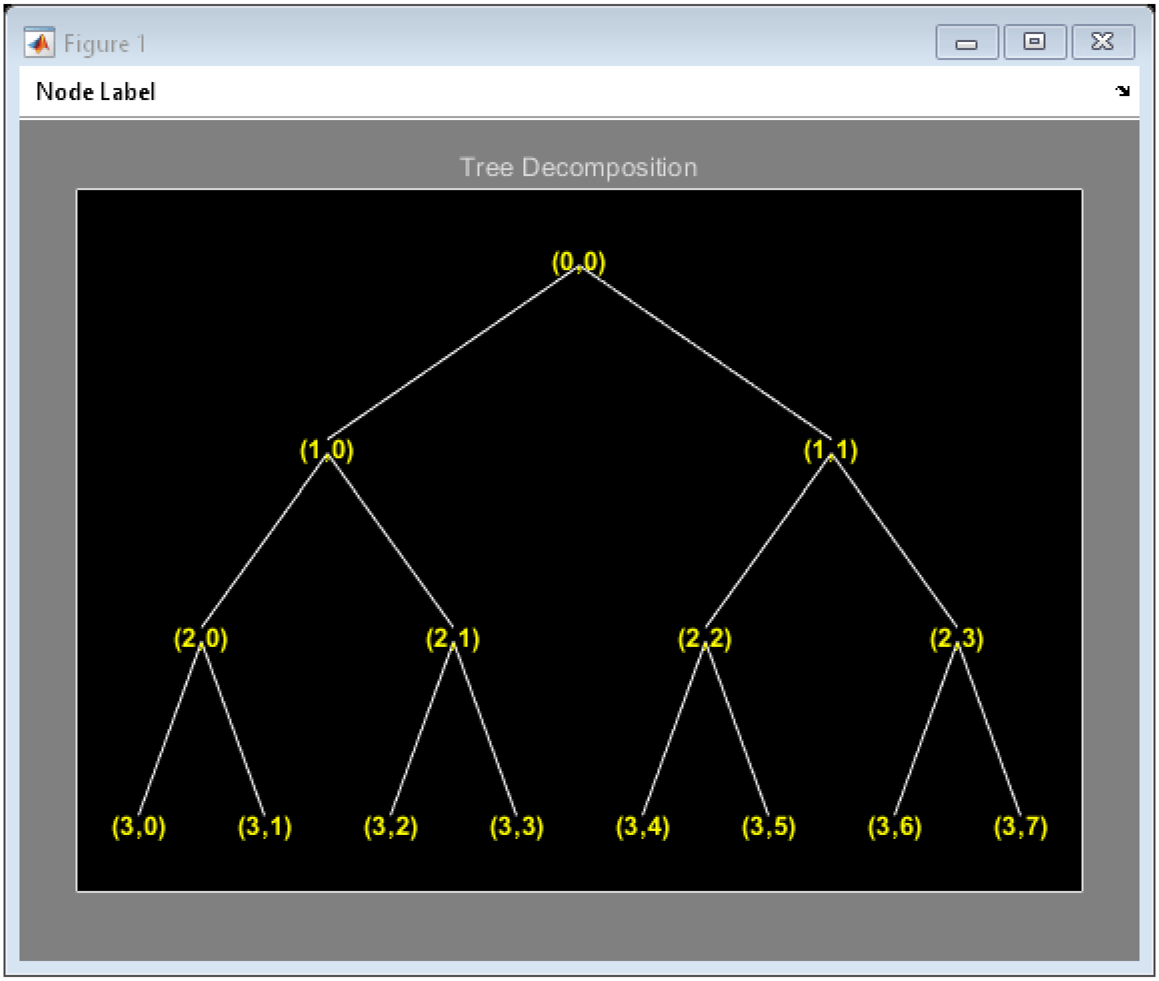
Note that for a column vector X, we have depo2ind(0,X) = X.

Examples

Convert Depth-Position to Node Index

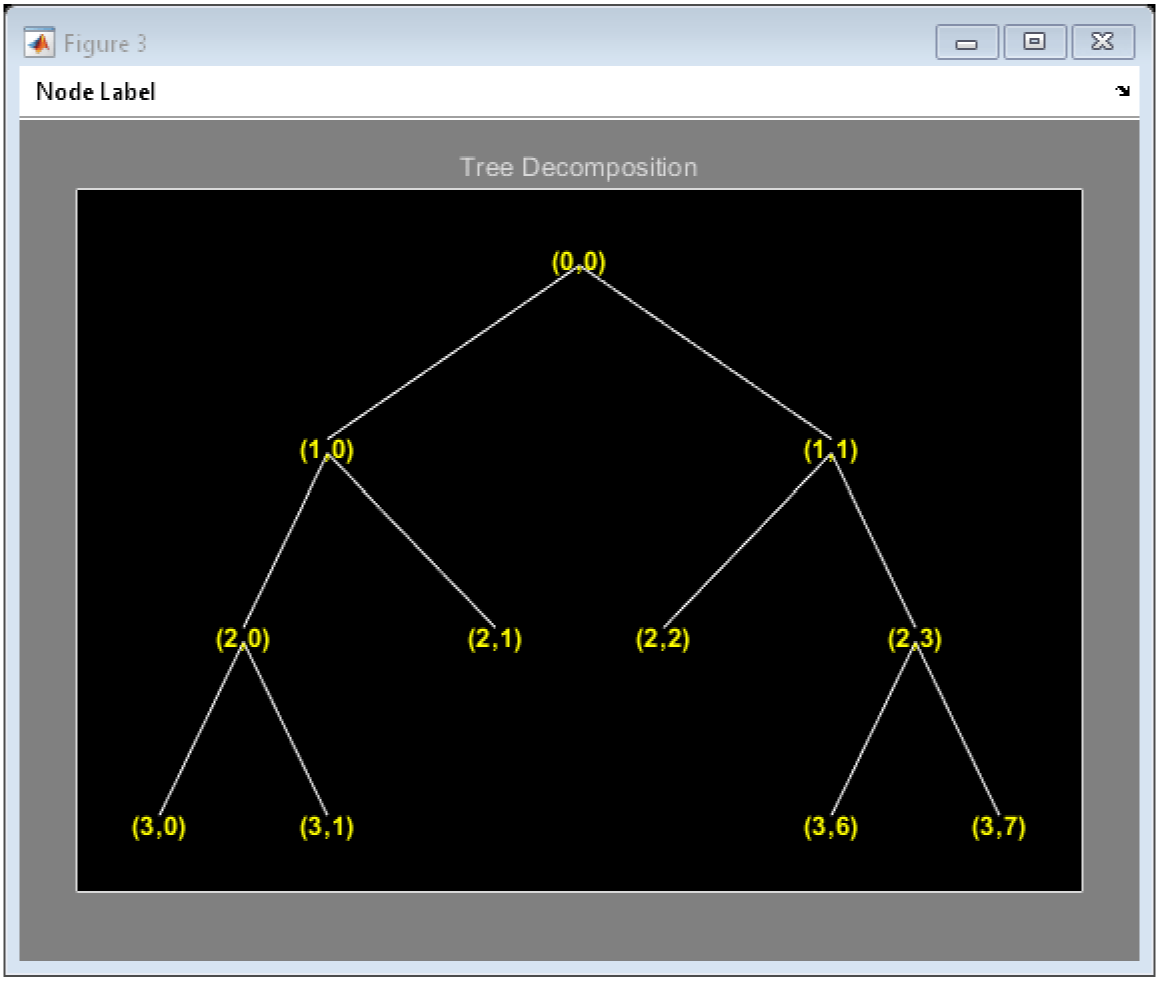
Create a binary tree of depth 3. Plot the tree.

```
ord = 2;
t = ntree(ord,3);
plot(t)
```



Merge the nodes of indices 4 and 5. Plot the new tree.

```
t = nodejoin(t,5);
t = nodejoin(t,4);
figure
plot(t)
```



List the depth-position of the tree nodes.

```
aln_depo = allnodes(t,'deppos')
```

```
aln_depo = 11×2

     0     0
     1     0
     1     1
     2     0
     2     1
     2     2
     2     3
     3     0
     3     1
     3     6
      ⋮
```

Convert the depth-position to index.

```
aln_ind = depo2ind(ord,aln_depo)
aln_ind = 11×1

     0
     1
     2
     3
     4
     5
     6
     7
     8
    13
      ⋮
```

See Also

`ind2depo`

Introduced before R2006a

det

Laurent matrix determinant

Syntax

```
d = det(A)
```

Description

`d = det(A)` returns the determinant of the Laurent matrix `A`.

Examples

Laurent Matrix Determinant

Create two Laurent polynomials:

- $a(z) = 6z^{-2}$
- $b(z) = z^3/6$

```
lpA = laurentPolynomial(Coefficients=[6],MaxOrder=-2);
lpB = laurentPolynomial(Coefficients=[1/6],MaxOrder=3);
```

Create the Laurent matrix $\begin{bmatrix} a(z) & 1 \\ 1 & b(z) \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={lpA 1; 1 lpB});
```

Obtain the determinant of `lmat`. Confirm the determinant is $z - 1$.

```
d = det(lmat)
```

```
d = 
  laurentPolynomial with properties:

    Coefficients: [1 -1]
        MaxOrder: 1
```

Input Arguments

A — Laurent matrix
`laurentMatrix` object

Laurent matrix, specified as a `laurentMatrix` object.

Output Arguments

d — Determinant
`laurentPolynomial` object

Determinant of the Laurent matrix, returned as a `laurentPolynomial` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`inverse`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

detcoef

1-D detail coefficients

Syntax

```
D = detcoef(C,L)
D = detcoef(C,L,N)
D = detcoef(C,L,N,'cells')

[D1,…,Dp] = detcoef(C,L,N)
```

Description

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

`D = detcoef(C,L,N)` extracts the detail coefficients at the level or levels specified by `N`.

`D = detcoef(C,L,N,'cells')` returns a cell array containing the detail coefficients. A minimum of two levels must be specified. The i^{th} element of `D` contains the detail coefficients at the i^{th} specified level.

- If `length(N)>1`, the `D = detcoef(C,L,N)` is equivalent to `D = detcoef(C,L,N,'cells')`.
- `D = detcoef(C,L,'cells')` is equivalent to `D = detcoef(C,L,[1:NMAX])`, where `NMAX = length(L)-2`.

`[D1,…,Dp] = detcoef(C,L,N)` extracts the detail coefficients at the levels specified by `N`. The length of `N` must equal the number of output arguments.

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.

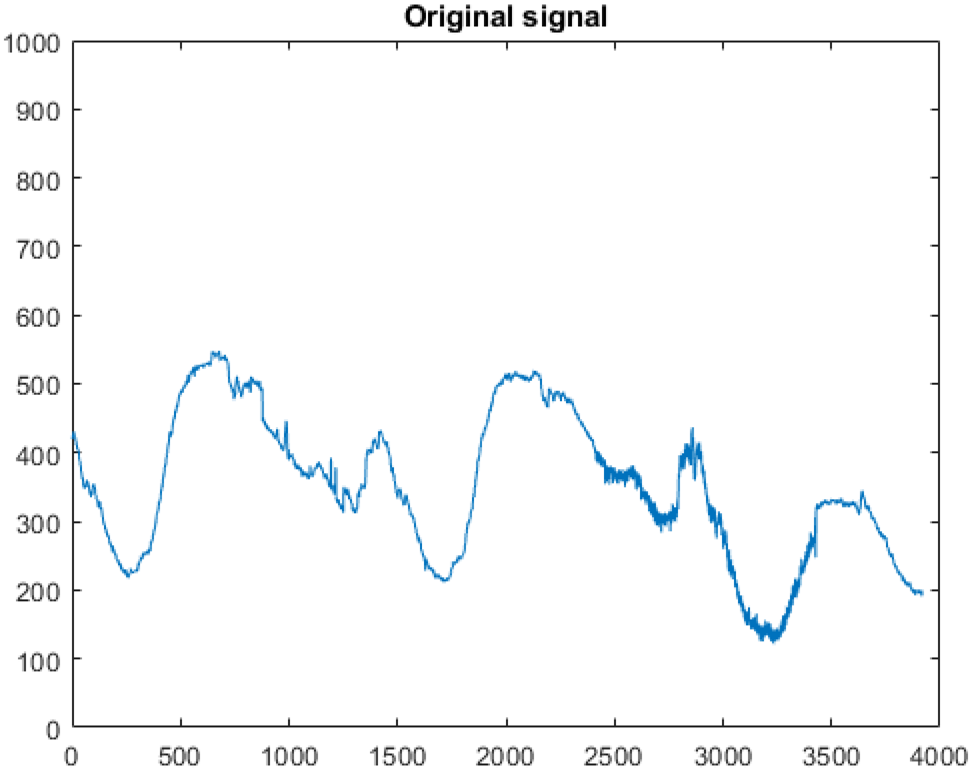
```
origmode = dwtmode('status','nodisplay');
dwtmode('zpd','nodisplay')

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]);
```

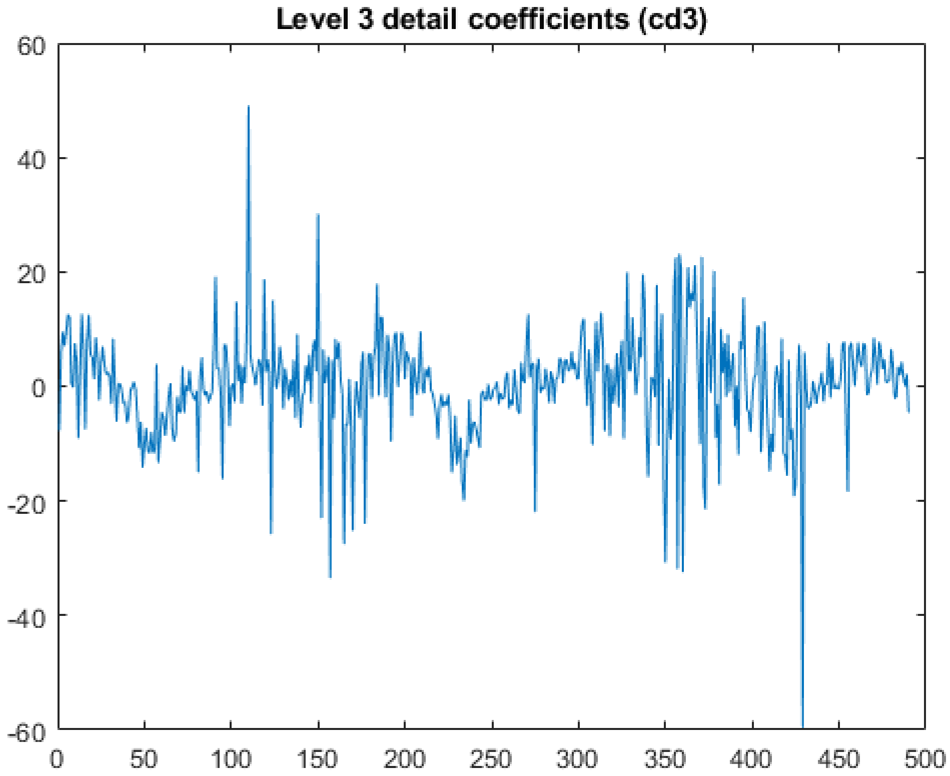
Plot the original signal.

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



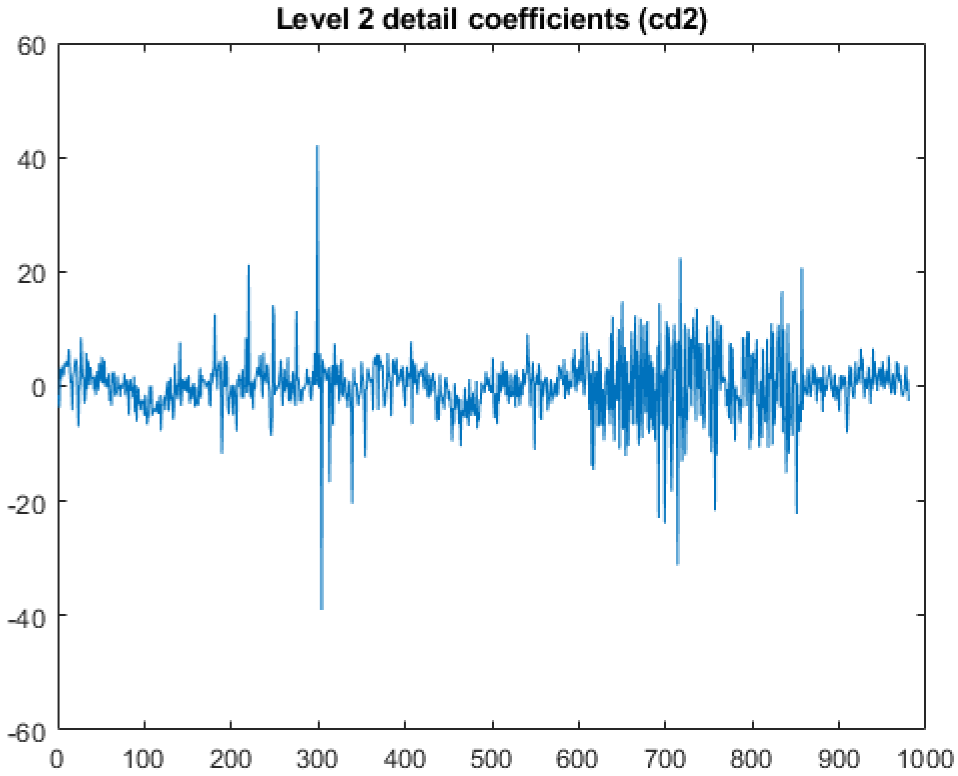
Plot the level 3 detail coefficients.

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



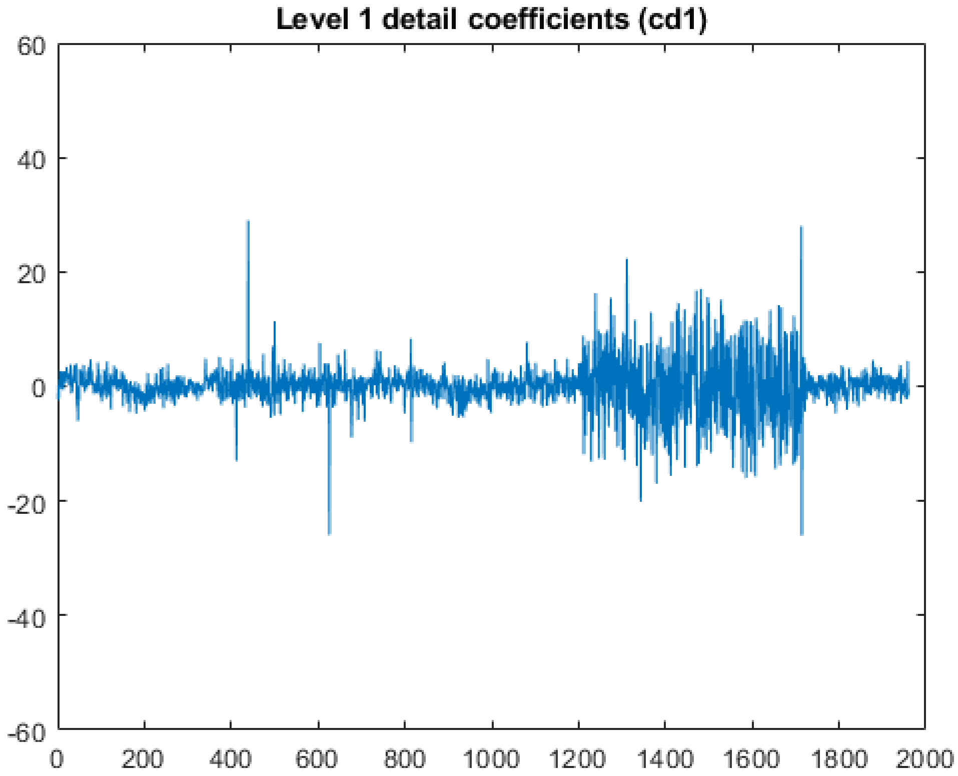
Plot the level 2 detail coefficients.

```
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])
```



Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

Input Arguments

C — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector, specified as a real-valued vector. The vector C is the output of wavedec.

Data Types: single | double

L — Bookkeeping vector

vector of positive integers

Bookkeeping vector, specified as a vector of positive integers. The bookkeeping vector L contains the number of coefficients by level. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition vector C. The vectors C and L are the outputs of wavedec.

Data Types: single | double

N — Detail level

positive integer | vector of positive integers

Detail level to extract from the wavelet decomposition, specified as a positive integer or a vector of positive integers.

- If `N` is an integer, then `N` must be an integer such that `1 ≤ N ≤ NMAX`, where `NMAX = length(L)-2`.
- If `N` is a vector of integers, then `N(j)` must be an integer such that `1 ≤ N(j) ≤ NMAX`, where `j = 1,…,length(N)`.

Output Arguments

`D` — Detail coefficients

real-valued vector | cell array

Detail coefficients, returned as a real-valued vector or a cell array. If `D` is a cell array, the *i*th element of `D` are the detail coefficients at the level specified by the *i*th element of `N`.

`D1,…,Dp` — Detail coefficients

real-valued vectors

Detail coefficients, returned as set of real-valued vectors. The *i*th output argument are the detail coefficients at the level specified by the corresponding element of `N`.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.
- For `gpuArray` inputs, `detcoef` supports only these syntaxes:
  - `D = detcoef(C,L)`
  - `D = detcoef(C,L,N)`

See Also

`appcoef` | `wavedec`

Introduced before R2006a

detcoef2

2-D detail coefficients

Syntax

```
y = detcoef2(o,c,s,n)
[h,v,d] = detcoef2('all',c,s,n)
y = detcoef2('compact',c,s,n)
```

Description

`y = detcoef2(o,c,s,n)` extracts from the wavelet decomposition structure `[c,s]` the detail coefficients of orientation `o` at level `n`. For more information on `c` and `s`, see `wavedec2`.

`[h,v,d] = detcoef2('all',c,s,n)` returns the horizontal `h`, vertical `v`, and diagonal `d` detail coefficients at level `n`.

`detcoef2('a',c,s,n)` is equivalent to `detcoef2('all',c,s,n)`.

`y = detcoef2('compact',c,s,n)` returns all the detail coefficients stored row-wise.

`detcoef2('c',c,s,n)` is equivalent to `detcoef2('compact',c,s,n)`.

If `[H,V,D] = detcoef2('all',c,s,`*N*`)` and `Y = detcoef2('compact',c,s,`*N*`)`, then `Y = [H(:)' V(:)' D(:)']`.

Examples

Extract Detail Coefficients From Image

This example shows how to extract detail coefficients from a discrete wavelet analysis of an image. This example uses zero-padding.

Set the extension mode to zero-padding. Load and display an image.

```
origmode = dwtmode('status','nodisplay');
dwtmode('zpd','nodisplay');

load woman
imagesc(X)
colormap(gray)
```

Obtain the wavelet decomposition of the image down to level two using the Haar wavelet.

```
[c,s] = wavedec2(X,2,'haar');
size(X)
```

```
ans = 1×2

   256   256
```

```
size(c)
```

```
ans = 1×2

           1       65536
```

```
s
```

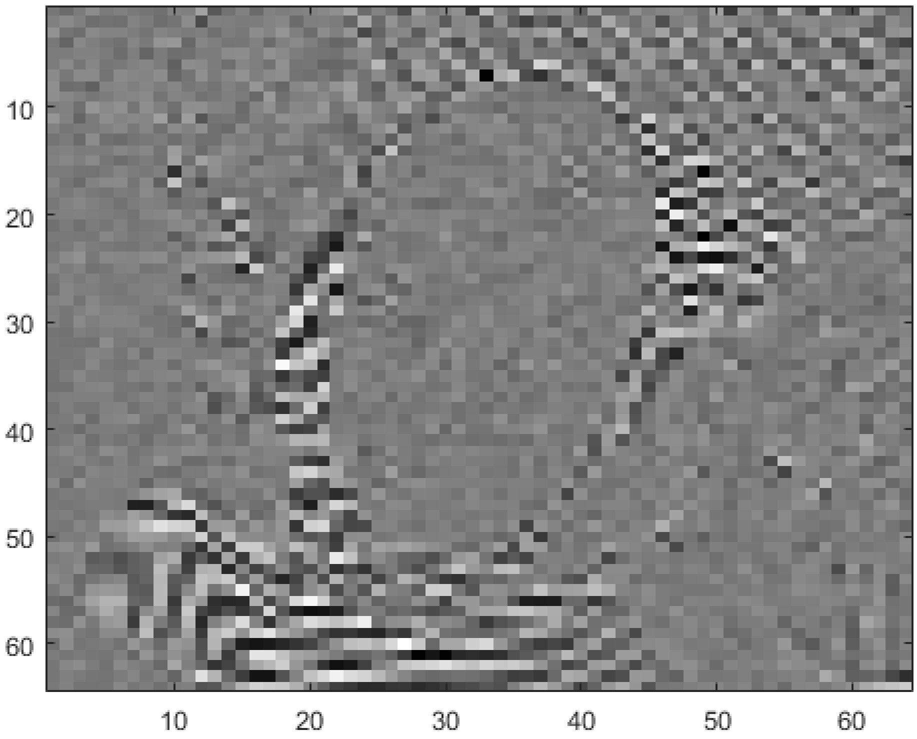
```
s = 4×2

    64    64
    64    64
   128   128
   256   256
```

Extract the detail coefficients at level 2 in each orientation from the wavelet decomposition structure `[c,s]`. Display the diagonal detail coefficients.

```
[chd2,cvd2,cdd2] = detcoef2('all',c,s,2);
size(cdd2)
```

```
ans = 1×2

    64    64
```

```
imagesc(cdd2)
colormap(gray)
```

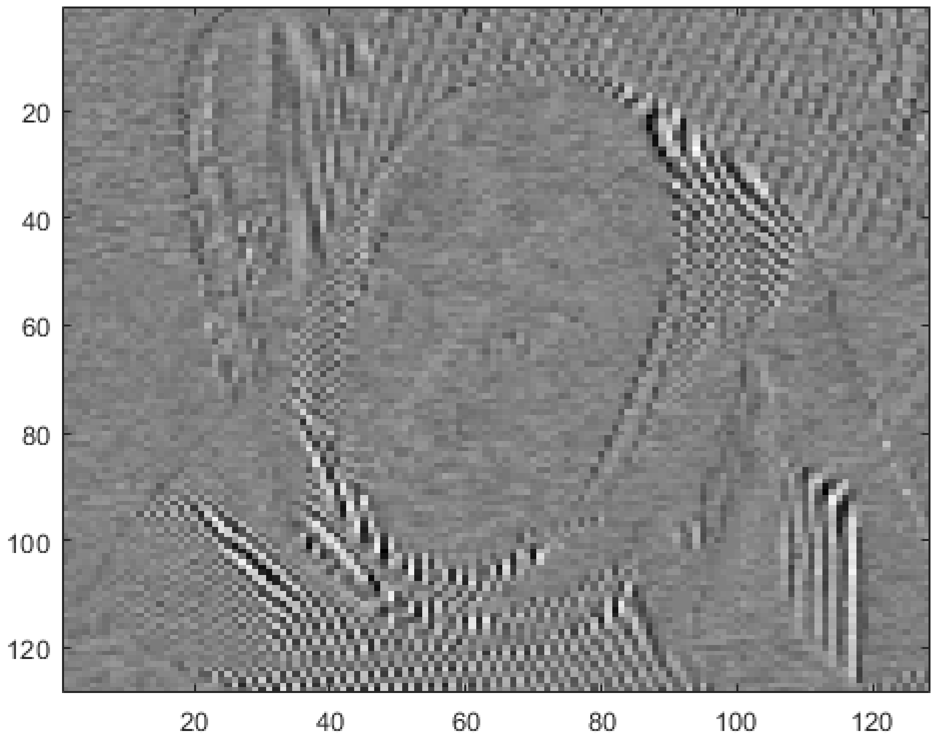


Extract the detail coefficients at level 1 in each orientation. Display the vertical detail coefficients.

```
[chd1,cvd1,cdd1] = detcoef2('all',c,s,1);
size(cvd1)
```

```
ans = 1×2

   128   128
```

```
imagesc(cvd1)
colormap(gray)
```



Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
```

Input Arguments

o — Orientation
'h' | 'v' | 'd'

Orientation of detail coefficients, specified as:

- 'h' — Horizontal
- 'v' — Vertical
- 'd' — Diagonal

c — Wavelet decomposition vector
real-valued vector

Wavelet decomposition vector, specified as a real-valued vector. The vector c contains the approximation and detail coefficients organized by level. The bookkeeping matrix s is used to parse c. See wavedec2.

Data Types: double

s — Bookkeeping matrix

integer-valued matrix

Bookkeeping matrix, specified as an integer-valued matrix. The matrix `s` contains the dimensions of the wavelet coefficients by level and is used to parse the wavelet decomposition vector `c`. See `wavedec2`.

Data Types: `double`

n — Detail level

integer

Detail level to extract from the wavelet decomposition, specified as an integer. The integer `n` must be in the interval `[1,size(s,1)-2]`.

Output Arguments

y — Detail coefficients

vector | matrix

Detail coefficients, returned as a vector or matrix.

Data Types: `double`

h — Horizontal detail coefficients

matrix

Horizontal detail coefficients, returned as a matrix.

Data Types: `double`

v — Vertical detail coefficients

matrix

Vertical detail coefficients, returned as a matrix.

Data Types: `double`

d — Diagonal detail coefficients

matrix

Diagonal detail coefficients, returned as a matrix.

Data Types: `double`

Tips

- If `c` and `s` are obtained from an indexed image analysis or a truecolor image analysis, `y` 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™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.
- For `gpuArray` inputs, `detcoef2` supports only these syntaxes:
  - `y = detcoef2(o,c,s,n)`
  - `[h,v,d] = detcoef2('all',c,s,n)`

See Also

`appcoef2` | `wavedec2` | `waverec2`

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

disp

Display lifting scheme

Syntax

```
disp(lscheme)
```

Description

`disp(lscheme)` displays the properties of the lifting scheme `lscheme`:

- Wavelet name
- Lifting steps
- Lowpass filter coefficients
- Normalization factors

The function also displays the properties of each lifting:

- Type of step
- Laurent polynomial coefficients
- Maximum order of the corresponding Laurent polynomial

Note To display a lifting scheme created using `liftwave`, see `displs`.

Examples

Display Lifting Scheme Properties

Create a lifting scheme associated with the `db3` wavelet.

```
lsc = liftingScheme('Wavelet','db3');
```

Display the lifting scheme properties.

```
disp(lsc)
```

```
      Wavelet               : 'db3'
     LiftingSteps          : [4 × 1] liftingStep
     NormalizationFactors  : [2.3155 0.4319]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'predict'
    Coefficients: -2.4255
        MaxOrder: 0

            Type: 'update'
    Coefficients: [-0.0793 0.3524]
        MaxOrder: 1

            Type: 'predict'
    Coefficients: [2.8953 -0.5614]
        MaxOrder: -1

            Type: 'update'
    Coefficients: 0.0198
        MaxOrder: 2
```

Input Arguments

`lscheme` — Lifting scheme

`liftingScheme` object

Lifting scheme, specified as a `liftingScheme` object.

See Also

`liftingScheme`

Introduced in R2021a

displs

(To be removed) Display lifting scheme

Note displs will be removed in a future release. Use disp and liftingScheme. For more information, see "Compatibility Considerations".

Syntax

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

Description

S = displs(LS,FRM) returns the character array describing the lifting scheme LS. The formatting operator FRM (see sprintf) builds S.

displs(LS) is equivalent to displs(LS,'%12.8f').

Examples

Display Lifting Scheme

Start with the Haar wavelet and get the corresponding lifting scheme.

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

Visualized the lifting scheme.

```
displs(lshaar);
```

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

Add a primal elementary lifting step to the lifting scheme. Display the resulting 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]              []
};
```

Input Arguments

LS — Lifting scheme

lifting scheme

Lifting scheme associated with a wavelet. See `liftwave`.

Example: `LS = liftwave('db4')` returns the lifting scheme associated with the Daubechies wavelet `db4`.

FRM — Formatting operator

formatting operator

Formatting operator used to build `LS`. See `sprintf`.

Example: `'%12.3f'`

Compatibility Considerations

displs will be removed

Not recommended starting in R2021a

`displs` will be removed in a future release. Use `disp` and `liftingScheme`. To update your code, follow these steps:

1. Create a lifting scheme using `liftingScheme`.
2. Display the lifting scheme using `disp`.

See Also

`disp` | `liftingScheme`

Introduced before R2006a

dispMat

Display Laurent matrix

Syntax

```
dispMat(A)
```

Description

`dispMat(A)` displays the Laurent matrix `A`.

Examples

Display Laurent Matrix

Create two Laurent polynomials:

- $a(z) = z^2$
- $b(z) = z^{-3}$

```
lpA = laurentPolynomial(MaxOrder=2);
lpB = laurentPolynomial(MaxOrder=-3);
```

Create the Laurent matrix $\begin{bmatrix} a(z) & 5 \\ 3 & b(z) \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={lpA,5;3,lpB});
```

Display the matrix.

```
dispMat(lmat)
```

```
| z^( 2 )    5.00e+00 |
|                                         |
| 3.00e+00   z^( -3 ) |
```

Input Arguments

A — Laurent matrix

`laurentMatrix` object

Laurent matrix, specified as a `laurentMatrix` object.

See Also

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

dlmodwt

Deep learning maximal overlap discrete wavelet transform and multiresolution analysis

Syntax

```
w = dlmodwt(x)
w = dlmodwt(x,Lo,Hi)
w = dlmodwt(x,Lo,Hi,level)
[w,mra] = dlmodwt( ___ )
[ ___ ] = dlmodwt( ___ ,Name=Value)
```

Description

`w = dlmodwt(x)` returns the maximal overlap discrete wavelet transform (MODWT) of `x` using the lowpass (scaling) and highpass (wavelet) filters associated with the Daubechies least-asymmetric wavelet with four vanishing moments (`'sym4'`) . By default, `dlmodwt` uses periodic boundary extension and computes the MODWT to the maximum level. `dlmodwt` requires Deep Learning Toolbox™.

`w = dlmodwt(x,Lo,Hi)` uses the scaling filter `Lo` and wavelet filter `Hi` in the MODWT computation.

`w = dlmodwt(x,Lo,Hi,level)` computes the MODWT down to the level specified in `level`.

`[w,mra] = dlmodwt( ___ )` returns the multiresolution analysis (MRA) of the MODWT of `x`.

`[ ___ ] = dlmodwt( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntaxes. For example, `BOUNDARY='periodic'` specifies periodic extension at the boundary.

Examples

Deep Learning Maximal Overlap Discrete Wavelet Transform

Load the 23 channel Espiga3 EEG data set. The data is sampled at 200 Hz. There are 995 samples in each channel. The data set is arranged as a 995-by-23(-by-1) array.

```
load Espiga3
```

Store the signal in an unformatted deep learning array.

```
x = dlarray(Espiga3);
```

Obtain the MODWT and MRA of the data. Specify the data format as `'TCB'`.

```
[wt,mra] = dlmodwt(x,DataFormat='TCB');
```

Confirm that both `wt` and `mra` are unformatted `dlarray` objects.

```
whos wt mra
```

```
  Name       Size                      Bytes  Class      Attributes

  mra       10x23x1x995              1830800  dlarray
  wt        10x23x1x995              1830800  dlarray
```

```
dims(wt)
```

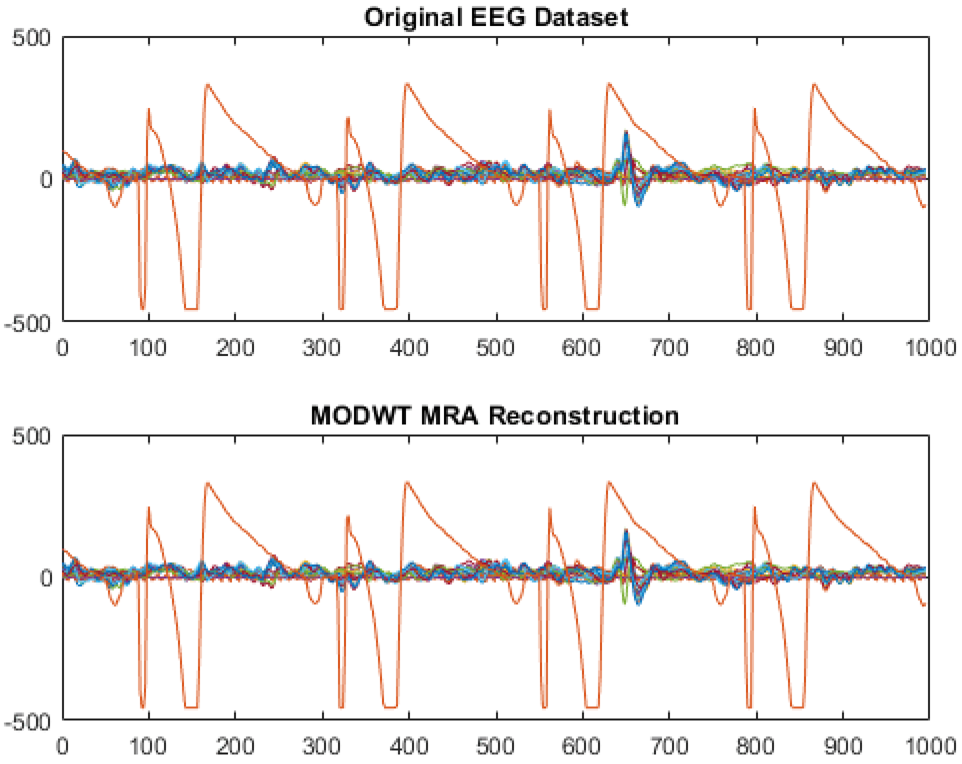
```
ans =

  0x0 empty char array
```

```
dims(mra)
```

```
ans =

  0x0 empty char array
```

Plot the reconstruction based on the MRA. Compare with the original data set.

```
xrec = sum(mra);
subplot(2,1,1)
plot(Espiga3)
title("Original EEG Dataset")
subplot(2,1,2)
plot(extractdata(squeeze(xrec))')
title("MODWT MRA Reconstruction")
```



Input Arguments

x — Input array

dlarray object | numeric array

Input array, specified as an unformatted dlarray, a formatted dlarray in 'CBT' format, or a numeric array.

If x is a numeric array or an unformatted dlarray, x must be compatible with the 'CBT' format. You must specify the 'DataFormat' as some permutation of 'CBT'.

Example: dlarray(cos(pi./[4;2]*(0:159)),'CTB') and dlarray(cos(pi./[4;2]*(0:159))','TCB') both specify one batch observation of a two-channel sinusoid in the 'CBT' format.

Data Types: single | double
Complex Number Support: Yes

Lo,Hi — Filters

numeric vectors | dlarray objects

Filters used in the MODWT computation, specified as a pair of even-length real-valued numeric vectors or unformatted dlarray objects. Lo is the scaling (lowpass) filter, and Hi is the wavelet (highpass) filter.

In order to satisfy the MODWT requirements, Lo and Hi must be the lowpass and highpass filters corresponding to an orthogonal wavelet. The wavelet manager wavemngr designates orthogonal wavelets as type 1 wavelets. Valid built-in orthogonal wavelet families begin with 'haar', 'db*N*', 'fk*N*', 'coif*N*', or 'sym*N*', where *N* is the number of vanishing moments for all families except 'fk'. For 'fk', *N* is the number of filter coefficients. You can determine valid values for *N* by using waveinfo. For example, waveinfo('db'). You can check if your wavelet is orthogonal by using wavemngr('type',*wname*) to see if the function returns 1. For example, wavemngr('type','db2'). If you have Lo and Hi as numeric vectors, you can use dwtfilterbank with the custom wavelet option to determine orthogonality. If unspecified, Lo and Hi default to: [~,~,Lo,Hi] = wfilters('sym4').

```
[Lo,Hi] = wfilters('db2');
fb = dwtfilterbank('Wavelet','custom',...
     'CustomScalingFilter',Lo(:),...
     'CustomWaveletFilter',Hi(:));
isOrthogonal(fb)
```

Data Types: single | double

level — Transform level

floor(log2(T)), where T is the size of x along the time dimension (default) | positive integer

Transform level of the MODWT, specified as a positive integer less than or equal to floor(log2(T)), where T is the size of x along the time dimension. If unspecified, dlmodwt computes the MODWT down to level floor(log2(T)).

Data Types: single | double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `w = dlmodwt(x,DataFormat='TCB')` specifies the data format as `'TCB'`.

`BOUNDARY` — Extension method

`'periodic'` (default) | `'reflection'`

Extension method to apply at the boundary in the computation of the MODWT, specified as one of these:

- `'periodic'` — Extend signal periodically
- `'reflection'` — Extend signal by reflection. The function computes the MODWT using a reflected signal along the `T` dimension twice the original length of `x`. The MODWT transform coefficients are also twice the length of the input.

Example: `w = dlmodwt(x,DataFormat='TCB',BOUNDARY='reflection')` extends the signal by reflection.

`DataFormat` — Data format of input

character vector | string scalar

Data format of input `x`, specified as some permutation of `'CBT'`. This argument is valid only if `x` is unformatted.

Each character in this argument must be one of these labels:

- `C` — Channel
- `B` — Batch
- `T` — Time

The `dlmodwt` function accepts any permutation of `'CBT'`. Each element of the argument labels the matching dimension of `x`.

Example: `w = dlmodwt(x,DataFormat='BCT')` specifies the data format of the unformatted `dlarray` object as `'BCT'`.

Data Types: `char` | `string`

Output Arguments

`w` — Maximal overlap discrete wavelet transform

formatted `dlarray` object | unformatted `dlarray` object

Maximal overlap discrete wavelet transform of `x`, returned as a `'SCBT'` formatted `dlarray`. `w` contains the wavelet coefficients and final-level scaling coefficients of `x`. The MODWT partitions the energy of the signal across the various scales and scaling coefficients. For more information, see `modwt`.

The size of `w` depends on the boundary extension method used in the computation of the MODWT.

- If the signal is extended periodically, then `w` is `level`+1-by-`C`-by-`B`-by-`T`.
- If the signal is extended by reflection, then `w` is `level`+1-by-`C`-by-`B`-by-2×`T`.

`level` is the transform level of the MODWT. `C` and `B` correspond to the channel and batch dimensions, respectively. The *k*th row of `w` contains the wavelet coefficients for the *k*th level. The (`level`+1)th row of `w` contains the approximation coefficients.

If you specify `'DataFormat'`, `w` is an unformatted `dlarray`.

`mra` — Multiresolution analysis
formatted `dlarray` object | unformatted `dlarray` object

Multiresolution analysis of the MODWT of `x`, returned as a `'SCBT'` formatted `dlarray`. `mra` contains the projections of `x` onto wavelet subspaces and a scaling space. For more information, see `modwtmra`.

`mra` is `level`+1-by-`C`-by-`B`-by-`T`, where `level` is the transform level of the MODWT. The *k*th row of `mra` contains the details for the *k*th level. The (`level`+1)th row of `mra` contains the `level`th level smooth.

If you specify `'DataFormat'`, `mra` is an unformatted `dlarray` compatible with `'SCBT'` format.

To learn more about the differences between the MODWT and the MRA, see “Comparing MODWT and MODWTMRA” on page 1-987.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`dlarray` | `dlstft` | `modwt` | `modwtmra`

Introduced in R2022a

drawtree

Draw wavelet packet decomposition tree (GUI)

Syntax

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

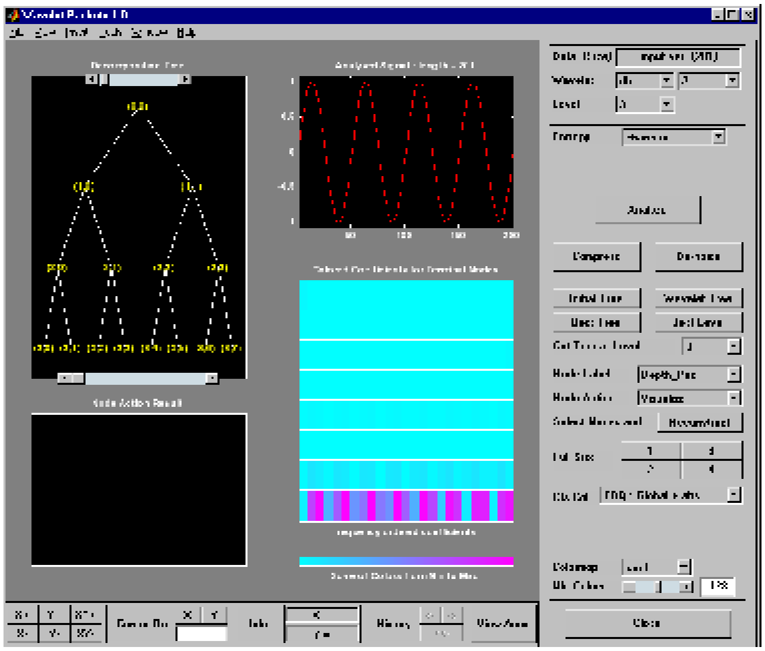
Description

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

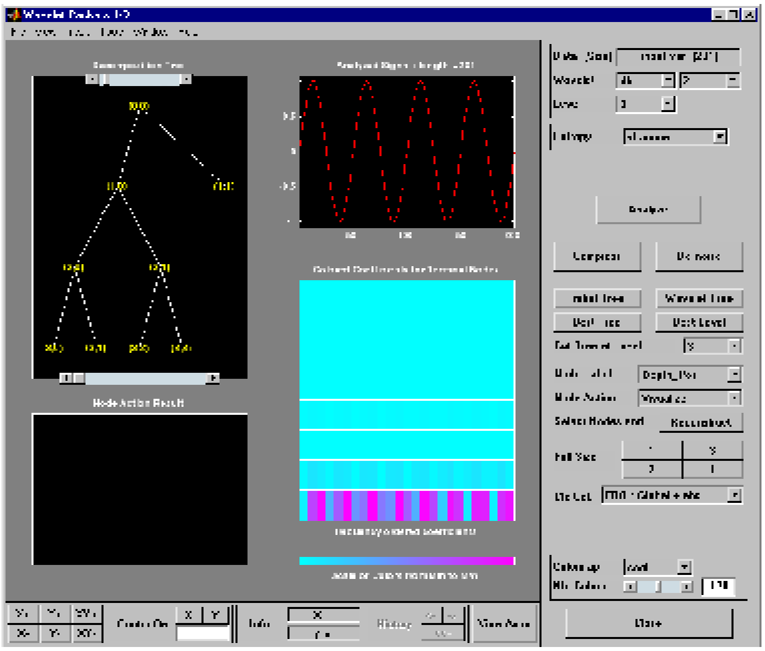
For an existing figure F produced by a previous call to the drawtree function, `drawtree(T,F)` draws the wavelet packet tree T in the figure whose handle is F.

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 `name`. These filters are used most often as input arguments to `dddtree` and `dddtree2`.

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

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 or string scalar. 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

dualtree

Kingsbury Q-shift 1-D dual-tree complex wavelet transform

Syntax

```
[A,D] = dualtree(X)
[ ___ ,Ascale] = dualtree(X)
[ ___ ] = dualtree(X,Name,Value)
```

Description

`[A,D] = dualtree(X)` returns the 1-D dual-tree complex wavelet transform (DTCWT) of `X`. The output `A` is the matrix of real-valued final-level scaling (lowpass) coefficients. The output `D` is an *L*-by-1 cell array of complex-valued wavelet coefficients, where *L* is the level of the transform.

The input `X` must have at least two samples. The DTCWT is obtained by default down to level `floor(log`$_2$*N*`)`, where *N* is the length of `X` if `X` is a vector and the row dimension of `X` if `X` is a matrix. If *N* is odd, `X` is extended by one sample by reflecting the last element of `X`.

By default, `dualtree` uses the near-symmetric biorthogonal filter pair with lengths 5 (scaling filter) and 7 (wavelet filter) for level 1 and the orthogonal Q-shift Hilbert wavelet filter pair of length 10 for levels greater than or equal to 2.

`[ ___ ,Ascale] = dualtree(X)` returns the scaling (lowpass) coefficients at each level.

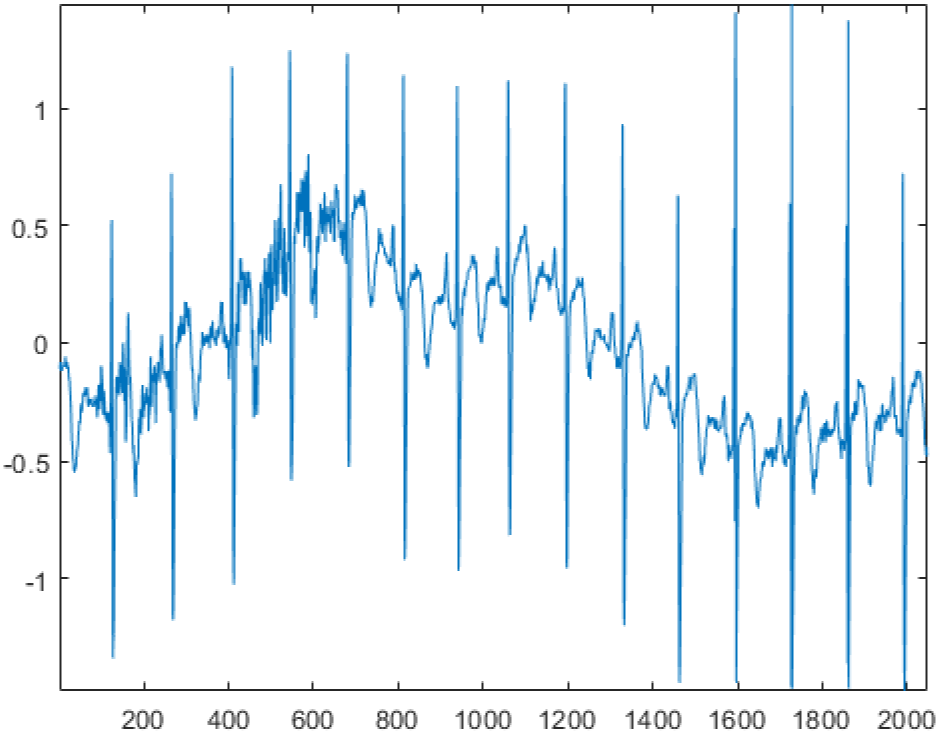
`[ ___ ] = dualtree(X,Name,Value)` specifies additional options using name-value pair arguments. For example, `'Level',10` specifies a decomposition down to level 10.

Examples

Plot Dual-Tree Complex Wavelet Transform Coefficients

Load an ECG signal.

```
load wecg
plot(wecg)
axis tight
```

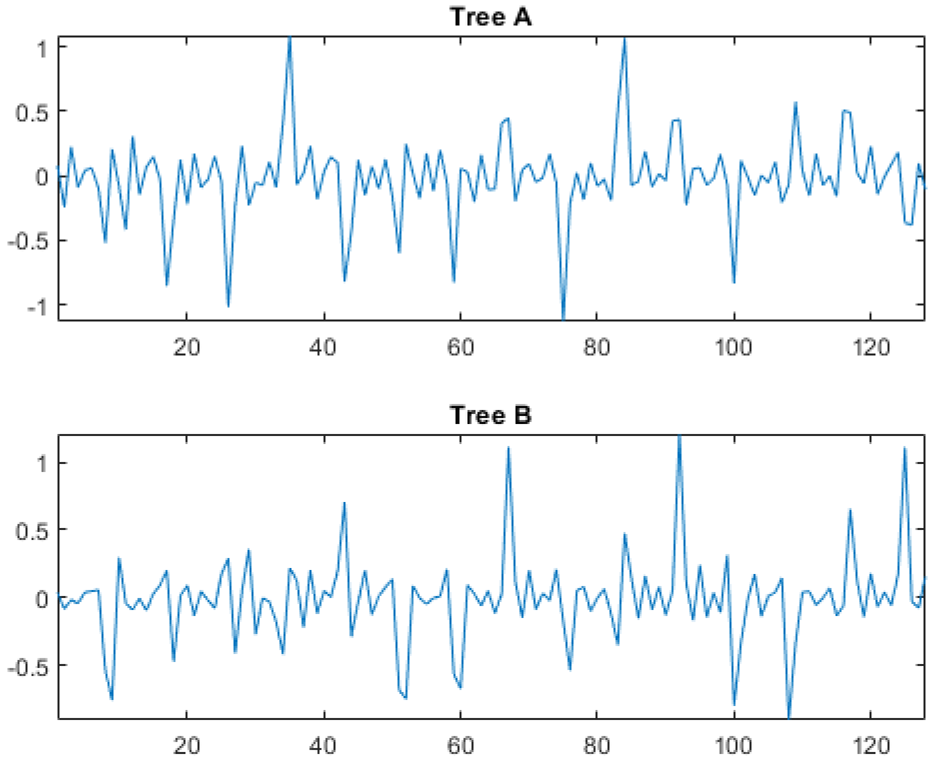


Obtain the 4-level dual-tree transform. Return the approximation (lowpass) coefficients at all levels.

```
[a,d,as] = dualtree(wecg,'Level',4);
```

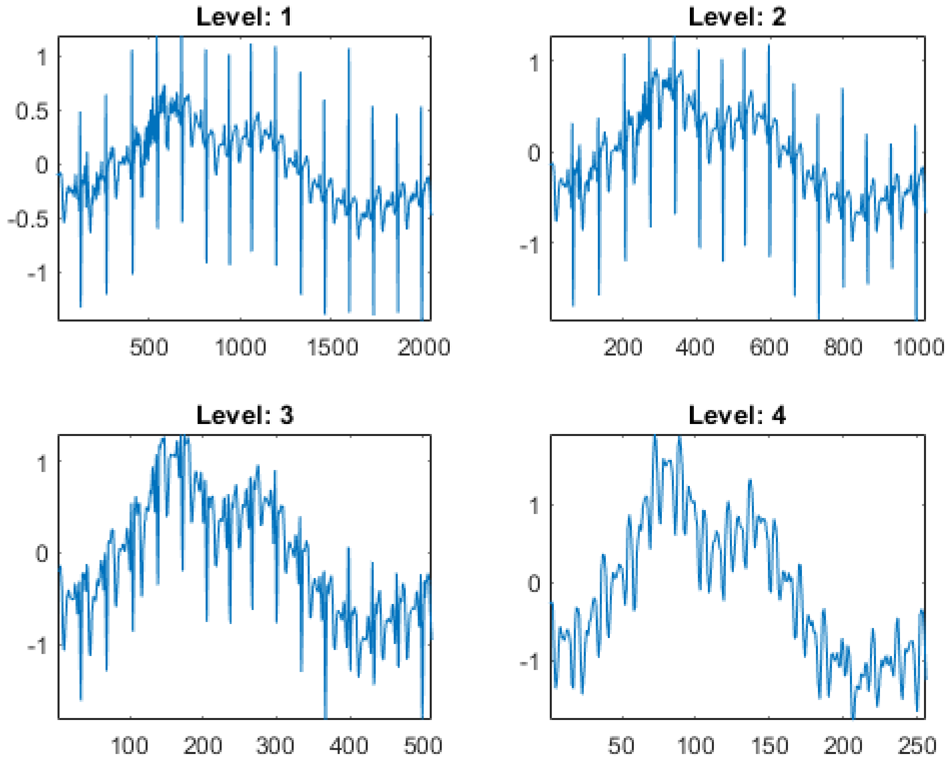
Plot the final-level wavelet coefficients from tree A and tree B.

```
figure
subplot(2,1,1)
plot(real(d{4}))
axis tight
title('Tree A')
subplot(2,1,2)
plot(imag(d{4}))
axis tight
title('Tree B')
```



Plot the lowpass coefficients at each level of the transform.

```
figure
for k=1:4
    subplot(2,2,k)
    plot(as{k})
    axis tight
    title(['Level: ',num2str(k)])
end
```

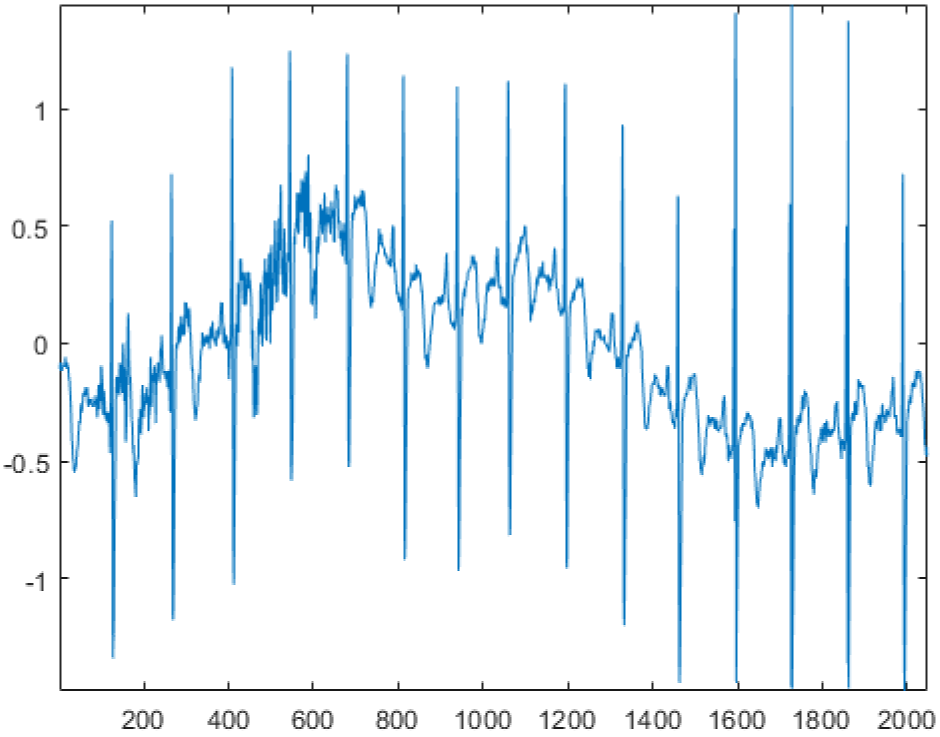


Distribution of Energy Across Scales

This example shows that small signal shifts do not significantly change the distribution of energy among the DTCWT coefficients at different scales.

Load an ECG signal. The signal has 2048 samples.

```
load wecg
len = numel(wecg);
plot(wecg)
axis tight
```



Create two 1-by-3000 zero vectors. Insert the ECG signal into different segments of each zero vector.

```
shift1 = 328;
shift2 = 368;
vec1 = zeros(1,3000);
vec2 = zeros(1,3000);
vec1(shift1+[1:len]) = wecg;
vec2(shift2+[1:len]) = wecg;
```

Obtain the dual-tree transform of both vectors. Use default settings.

```
[a1,d1] = dualtree(vec1);
[a2,d2] = dualtree(vec2);
```

Compute the energy at each scale for both decompositions. Note that the energy distribution of the shifted signals across all scales remains approximately the same.

```
energy1 = cell2mat(cellfun(@(x)(sum(abs(x).^2)),d1,'uni',0));
energy2 = cell2mat(cellfun(@(x)(sum(abs(x).^2)),d2,'uni',0));
levels =cell(numel(energy1),1);
for k=1:numel(energy1)
    levels{k} = sprintf('Level %d',k);
end
energies = table(levels,energy1,energy2)
```

```
energies=11×3 table
       levels        energy1     energy2
    ____________    _______     _______
    {'Level 1' }    16.014     16.014
    {'Level 2' }    19.095     19.095
    {'Level 3' }     35.99      35.99
    {'Level 4' }    25.141     25.065
    {'Level 5' }     16.81     17.452
    {'Level 6' }    9.7078      9.161
    {'Level 7' }    2.3201     2.0513
    {'Level 8' }    8.3808     8.4197
    {'Level 9' }    23.006      22.56
    {'Level 10'}    70.764     73.964
    {'Level 11'}    64.097     59.022
```

Input Arguments

X — Input data

vector | matrix | timetable

Input data, specified as a real-valued vector, matrix, or timetable. The input X must have at least two samples. If X is a timetable, it can contain a single vector or matrix variable, or it can contain multiple variables, each containing a column vector. If X is a matrix, dualtree operates on the columns of X.

Data Types: double | single

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: 'LevelOneFilter','antonini','Level',4

Level — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer less than or equal to floor(log$_2N$), where N is the length of X if X is a vector and the row dimension of X if X is a matrix. If unspecified, Level defaults to floor(log$_2N$).

LevelOneFilter — Biorthogonal filter

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

Biorthogonal filter to use in the first-level analysis, specified as:

- 'legall' — LeGall 5/3 filter [3]
- 'nearsym13_19' — (13,19)-tap near-orthogonal filter [2]
- 'nearsym5_7' — (5,7)-tap near-orthogonal filter [1]
- 'antonini' — (9,7)-tap Antonini filter [1]

By default, dualtree uses 'nearsym5_7', the near-symmetric biorthogonal filter pair with lengths 5 (scaling filter) and 7 (wavelet filter).

FilterLength — Orthogonal Hilbert Q-shift analysis filter pair length
10 (default) | 6 | 14 | 16 | 18

Orthogonal Hilbert Q-shift analysis filter pair length to use for levels 2 and higher, specified as one of the listed values [2]. By default, dualtree uses the orthogonal Q-shift Hilbert wavelet filter pair of length 10.

Output Arguments

A — Final-level approximation coefficients
real-valued vector | real-valued matrix

Final-level approximation coefficients, returned as a real-valued vector if X is a vector, or a matrix if X is a multisignal. The approximation coefficients are the final-level scaling (lowpass) coefficients. If X is a matrix, the column dimensions of X and A are equal.

D — Wavelet coefficients
cell array

Wavelet coefficients, returned as an *L*-by-1 cell array of complex-valued wavelet coefficients, where *L* is the level of the transform. The real parts of the coefficients are from tree A, and the imaginary parts are from tree B. If X is a matrix, each element of D is a matrix whose column dimension equals the column dimension of X.

Ascale — Approximation coefficients
cell array

Approximation coefficients at each level of the transform, returned as an *L*-by-1 cell array of real-valued scaling (lowpass) coefficients, where *L* is the level of the transform. If X is a matrix, each element of D is a matrix whose column dimension equals the column dimension of X.

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Timetable input data is not supported.

See Also

`idualtree` | `dualtree2` | `dualtree3` | `qbiorthfilt` | `qorthwavf`

Topics

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

Introduced in R2020a

dualtree2

Kingsbury Q-shift 2-D dual-tree complex wavelet transform

Syntax

```
[A,D] = dualtree2(X)
[___,Ascale] = dualtree2(X)
[___] = dualtree2(X,Name,Value)
```

Description

`[A,D] = dualtree2(X)` returns the 2-D dual-tree complex wavelet transform (DTCWT) of `X` using Kingsbury Q-shift filters. The output `A` is the matrix of real-valued final-level scaling (lowpass) coefficients. The output `D` is a *L*-by-1 cell array of complex-valued wavelet coefficients, where *L* is the level of the transform. For each element of `D` there are six wavelet subbands.

The DTCWT is obtained by default down to level $\texttt{floor}(\texttt{log}_2(\texttt{min}([H\ W])))$, where *H* and *W* refer to the height (row dimension) and width (column dimension) of `X`, respectively. If any of the row or column dimensions of `X` are odd, `X` is extended along that dimension by reflecting around the last row or column.

By default, `dualtree2` uses the near-symmetric biorthogonal wavelet filter pair with lengths 5 (scaling filter) and 7 (wavelet filter) for level 1 and the orthogonal Q-shift Hilbert wavelet filter pair of length 10 for levels greater than or equal to 2.

`[___,Ascale] = dualtree2(X)` returns the scaling (lowpass) coefficients at each level.

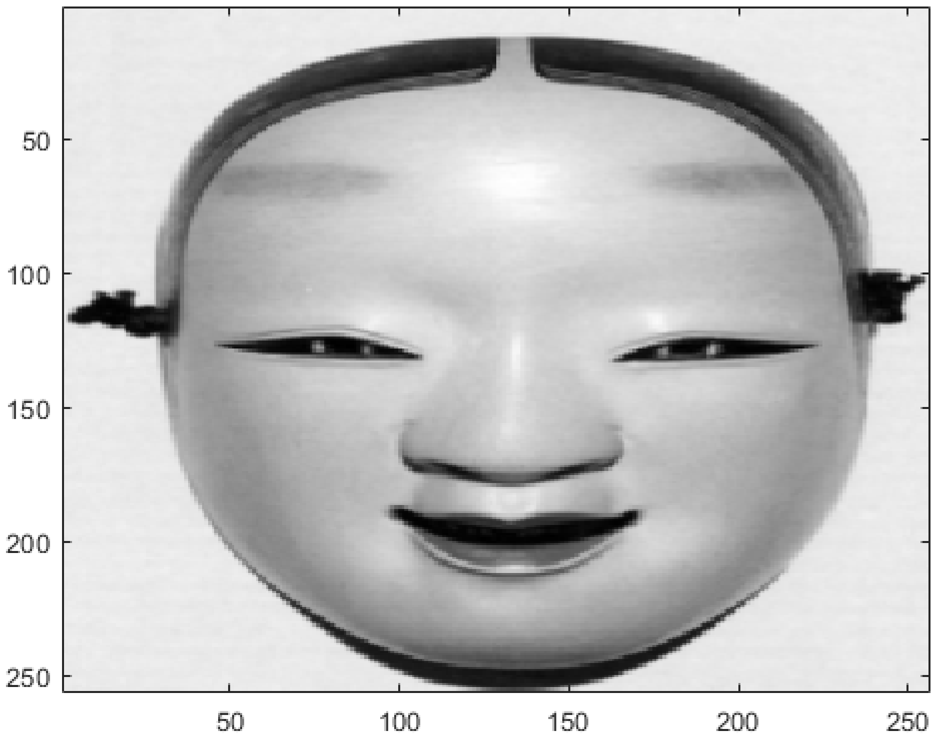
`[___] = dualtree2(X,Name,Value)` specifies additional options using name-value pair arguments. For example, `'LevelOneFilter','antonini'` specifies the (9,7)-tap Antonini filter as the biorthogonal filter to use in the first-level analysis.

Examples

2-D Dual-Tree Complex Wavelet Transform

Load a grayscale image.

```
load mask
imagesc(X)
colormap gray
```

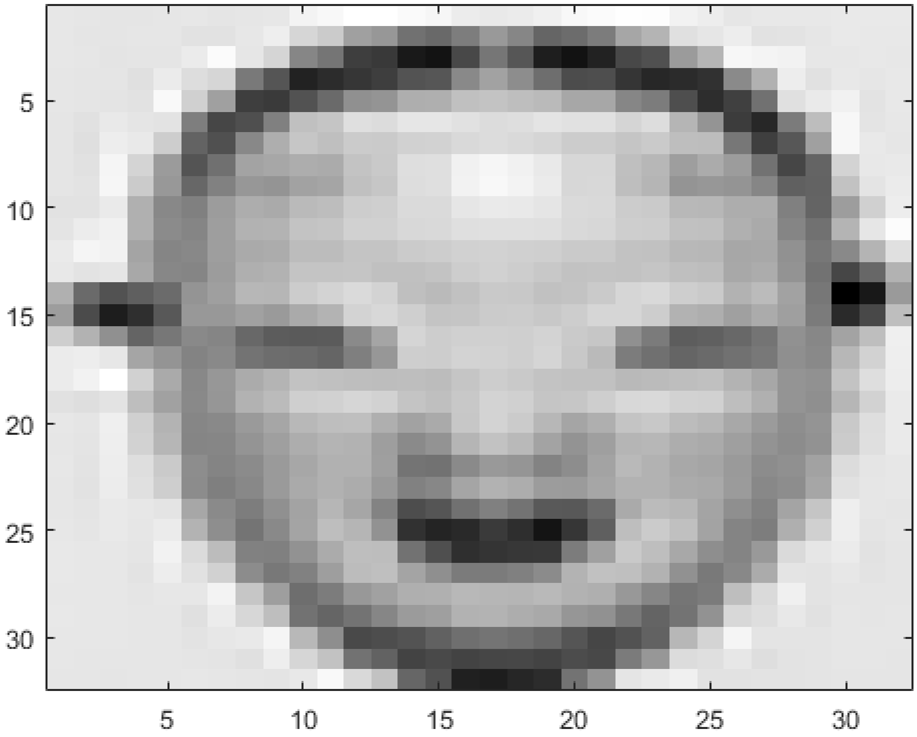


Obtain the dual-tree complex wavelet transform of the image down to four levels of resolution.

```
[a,d] = dualtree2(X,'Level',4);
```

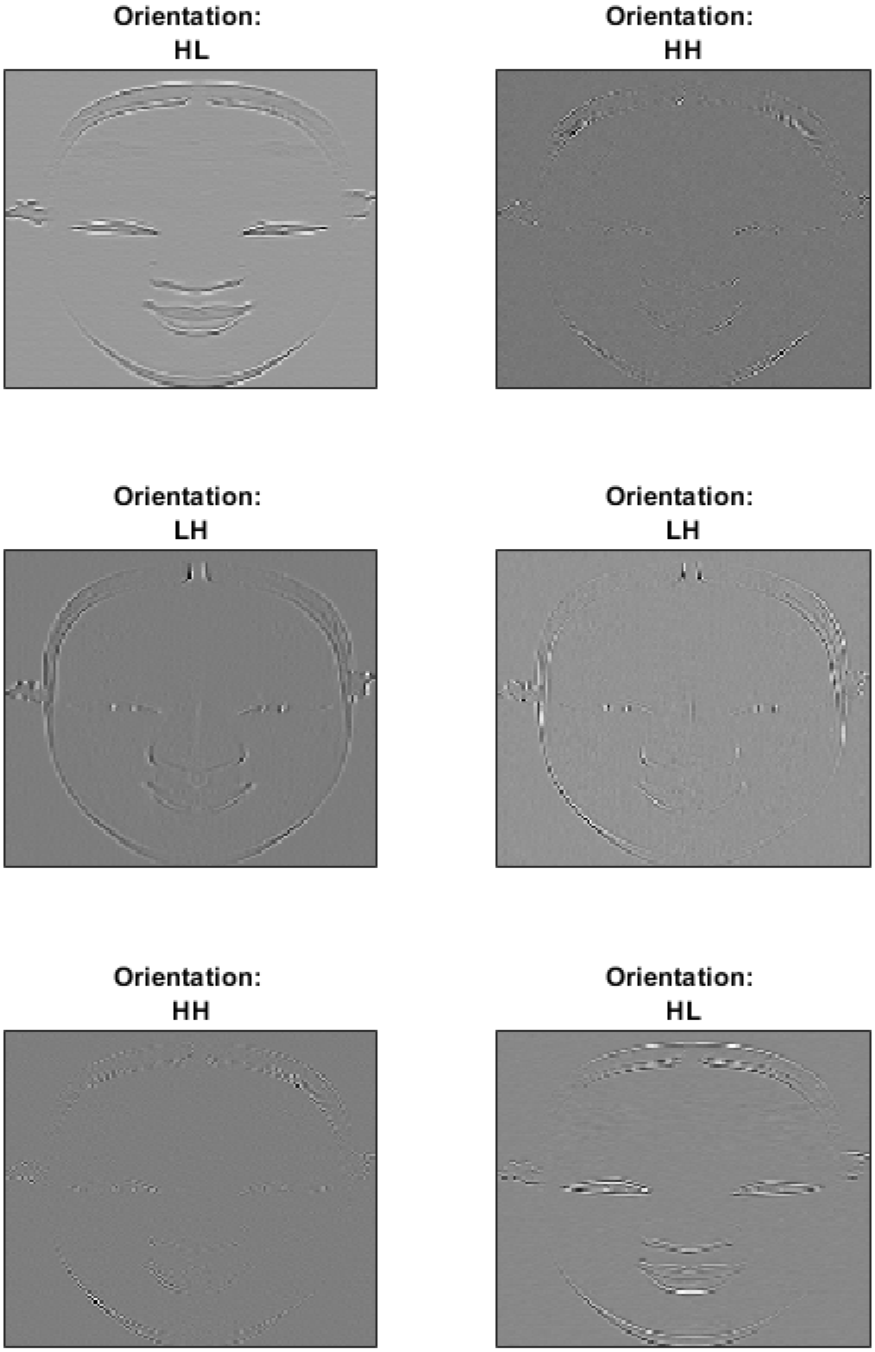
Display the final-level scaling (lowpass) coefficients.

```
imagesc(a)
colormap gray
```



Display the tree B wavelet coefficients at the finest scale. Each subplot title denotes the particular subband ("H" for highpass, "L" for lowpass).

```
orientation = ["HL","HH","LH","LH","HH","HL"];
for k=1:6
    subplot(3,2,k)
    imagesc(imag(d{1}(:,:,k)))
    title(['Orientation: ' orientation(k)])
    set(gca,'xtick',[])
    set(gca,'ytick',[])
end
colormap gray
set(gcf,'Position',[0 0 560 800])
```



Orientation: HL

Orientation: HH

Orientation: LH

Orientation: LH

Orientation: HH

Orientation: HL

Input Arguments

X — Input data

real-valued matrix (default) | real-valued 3-D array | real-valued 4-D array

Input data, specified as a real-valued matrix, 3-D array, or 4-D array. X is a real-valued *H*-by-*W*-by-*C*-by-*N* array, where *H* is the height or row dimension, *W* is the width or column dimension, *C* is the number of channels, and *N* is the number of images. X must have at least two samples in each of the row and column dimensions.

Example: If X is a 256-by-256-by-3-by-2 array, X contains two 256-by-256 RGB images.

Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'LevelOneFilter','antonini','Level',4`

`Level` — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer less than or equal to `floor(log`$_2$`(min([`*H* *W*`])))`, where *H* and *W* refer to the height (row dimension) and width (column dimension) of X, respectively. If unspecified, `Level` defaults to `floor(log`$_2$`(min([`*H* *W*`])))`.

`LevelOneFilter` — Biorthogonal filter

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

Biorthogonal filter to use in the first-level analysis, specified as:

- `'legall'` — LeGall 5/3 filter [3]
- `'nearsym13_19'` — (13,19)-tap near-orthogonal filter [2]
- `'nearsym5_7'` — (5,7)-tap near-orthogonal filter [1]
- `'antonini'` — (9,7)-tap Antonini filter [1]

By default, `dualtree2` uses `'nearsym5_7'`, the near-symmetric biorthogonal filter pair with lengths 5 (scaling filter) and 7 (wavelet filter).

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

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

Orthogonal Hilbert Q-shift analysis filter pair length to use for levels 2 and higher, specified as one of the listed values [2]. By default, `dualtree2` uses the orthogonal Q-shift Hilbert wavelet filter pair of length 10.

Output Arguments

A — Final-level approximation coefficients

real-valued matrix

Final-level approximation coefficients, returned as a real-valued matrix.

`D` — Wavelet coefficients

cell array

Wavelet coefficients, returned as an *L*-by-1 cell array of complex-valued wavelet coefficients, where *L* is the level of the transform. The real parts of the coefficients are from tree A, and the imaginary parts are from tree B. For each element of `D` there are six wavelet subbands.

`Ascale` — Approximation coefficients

cell array

Approximation coefficients at each level of the transform, returned as an *L*-by-1 cell array of real-valued scaling (lowpass) coefficients, where *L* is the level of the transform. If `X` is a matrix, each element of `D` is a matrix whose column dimension equals the column dimension of `X`.

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation

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

See Also

`dualtree` | `idualtree2` | `dualtree3` | `qbiorthfilt` | `qorthwavf`

Topics

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

Introduced in R2020a

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.

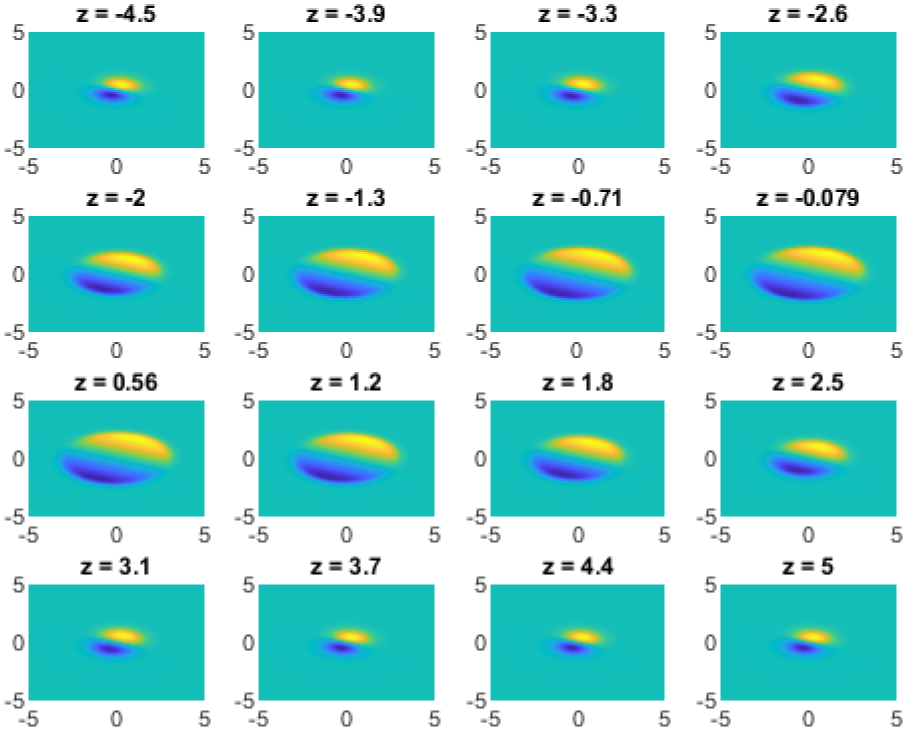
`[a,d] = dualtree3( ___ ,'excludeL1')` excludes the first-level wavelet (detail) coefficients. Excluding the first-level wavelet coefficients can speed up the algorithm and saves memory. The first level does not exhibit the directional selectivity of levels 2 and higher. The perfect reconstruction property 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
```

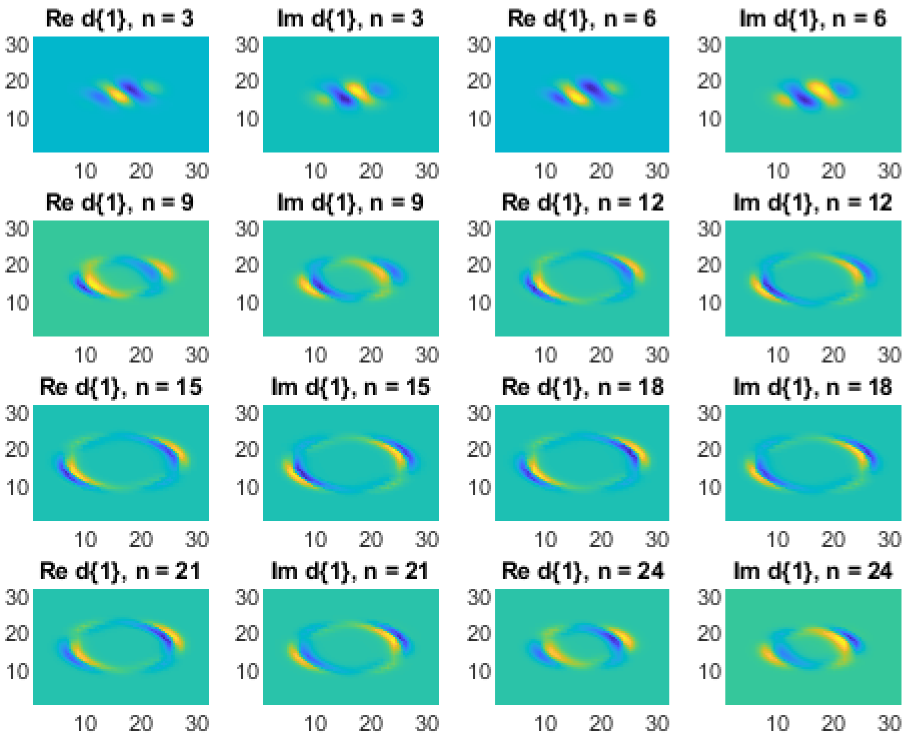


Compute the 3-D dual-tree transform of the data down to level 4. Specify a Hilbert Q-shift filter-pair length of 14.

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

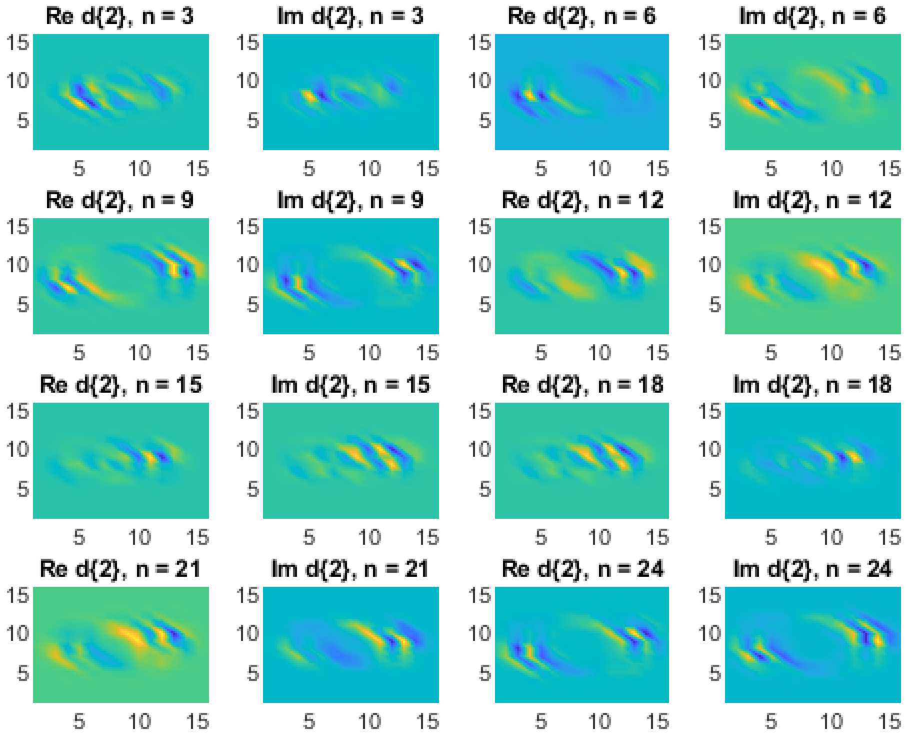
Plot the real and imaginary parts of the first-level wavelet coefficients for selected subbands. The coefficients have the same directionality as the data. The imaginary parts are shifted versions of the real parts.

```
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 = 1.3767e-14
```

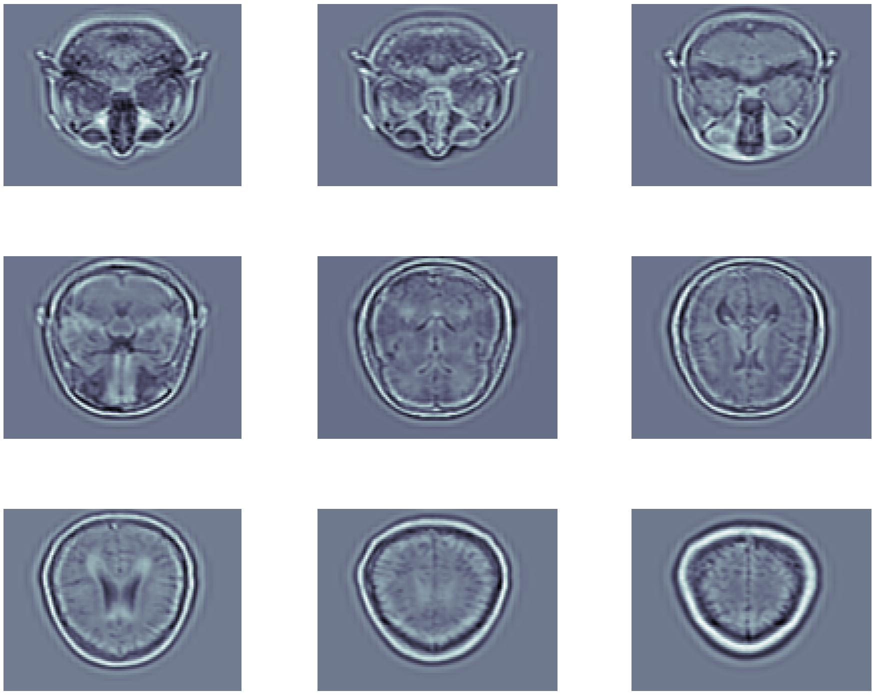
3-D Dual-Tree Transform of MRI Data

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
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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, Number 3, May 2001, pp. 234–253.

See Also

`idualtree3` | `wavedec3` | `waverec3` | `dddtree2` | `qbiorthfilt` | `dualtree2` | `dualtree` | `qorthwavf`

Topics

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

Introduced in R2017a

dwpt

Multisignal 1-D wavelet packet transform

Syntax

```
wpt = dwpt(X)
wpt = dwpt(X,wname)
wpt = dwpt(X,LoD,HiD)
[wpt,l] = dwpt( ___ )
[wpt,l,packetlevels] = dwpt( ___ )
[wpt,l,packetlevels,f] = dwpt( ___ )
[wpt,l,packetlevels,f,re] = dwpt( ___ )
[ ___ ] = dwpt( ___ ,Name,Value)
```

Description

`wpt = dwpt(X)` returns the terminal (final-level) nodes of the discrete wavelet packet transform (DWPT) of `X`. The input `X` is a real-valued vector, matrix, or timetable. By default, the `fk18` wavelet is used, and the decomposition level is `floor(log`$_2$`(`*Ns*`))`, where *Ns* is the number of data samples. The wavelet packet transform `wpt` is a 1-by-*N* cell array, where *N* `= 2^floor(log`$_2$`(`*Ns*`))`.

`wpt = dwpt(X,wname)` uses the wavelet specified by `wname` for the DWPT. `wname` must be recognized by `wavemngr`.

`wpt = dwpt(X,LoD,HiD)` uses the scaling (lowpass) filter, `LoD`, and wavelet (highpass) filter, `HiD`.

`[wpt,l] = dwpt( ___ )` also returns the bookkeeping vector using any of the previous syntaxes. The vector `l` contains the length of the input signal and the number of coefficients by level. The bookkeeping vector is required for perfect reconstruction.

`[wpt,l,packetlevels] = dwpt( ___ )` also returns the transform levels of the nodes of `wpt` using any of the previous syntaxes.

`[wpt,l,packetlevels,f] = dwpt( ___ )` also returns the center frequencies of the approximate passbands in cycles per sample using any of the previous syntaxes.

`[wpt,l,packetlevels,f,re] = dwpt( ___ )` also returns the relative energy for the wavelet packets in `wpt` using any of the previous syntaxes. The relative energy is the proportion of energy contained in each wavelet packet by level.

`[ ___ ] = dwpt( ___ ,Name,Value)` specifies options using name-value pair arguments in addition to the input arguments in the previous syntaxes. For example, `'Level',4` specifies the decomposition level.

Examples

Multichannel Discrete Wavelet Packet Transform

Load the 23-channel EEG data `Espiga3` [3]. The data is sampled at 200 Hz.

```
load Espiga3
```

Compute the 1-D DWPT of the data using the sym3 wavelet down to level 4. Obtain the terminal wavelet packet nodes, bookkeeping vector, and center frequencies of the approximate passbands.

```
[wpt,bk,~,f] = dwpt(Espiga3,'sym3','Level',4);
```

The output wpt is a 1-by-2^4 cell array. Every element of wpt is a matrix. Choose any terminal node, and confirm the size of the matrix is 23-by-*M*, where *M* is the last element of the bookkeeping vector bk.

```
nd = 13;
size(wpt{nd})
```

```
ans = 1×2

    23    66
```

```
bk(end)
```

```
ans = 66
```

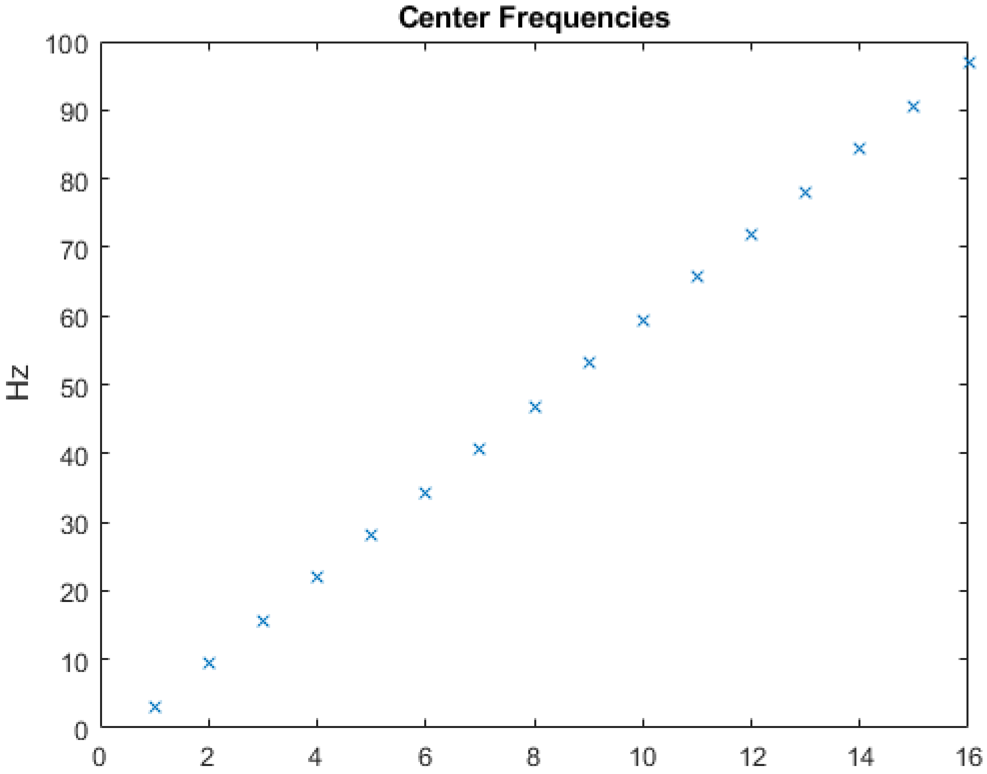
Extract the final-level coefficients of the fifth channel.

```
p5 = cell2mat(cellfun(@(x) x(5,:).',wpt,'UniformOutput',false));
size(p5)
```

```
ans = 1×2

    66    16
```

The terminal nodes are sequency-ordered. Plot the center frequencies of the approximate passbands in hertz, and confirm they are in order of increasing frequency.

```
plot(200*f,'x')
title('Center Frequencies')
ylabel('Hz')
```



PR Biorthogonal Filters

This example shows how to take an expression of a biorthogonal filter pair and construct lowpass and highpass filters to produce a perfect reconstruction (PR) pair in Wavelet Toolbox™.

The LeGall 5/3 filter is the wavelet used in JPEG2000 for lossless image compression. The lowpass (scaling) filters for the LeGall 5/3 wavelet have five and three nonzero coefficients respectively. The expressions for these two filters are:

$$H_0(z) = 1/8(-z^2 + 2z + 6 + 2z^{-1} - z^{-2})$$

$$H_1(z) = 1/2(z + 2 + z^{-1})$$

Create these filters.

```
H0 = 1/8*[-1 2 6 2 -1];
H1 = 1/2*[1 2 1];
```

Many of the discrete wavelet and wavelet packet transforms in Wavelet Toolbox rely on the filters being both even-length and equal in length in order to produce the perfect reconstruction filter bank associated with these transforms. These transforms also require a specific normalization of the coefficients in the filters for the algorithms to produce a PR filter bank. Use the `biorfilt` function on the lowpass prototype functions to produce the PR wavelet filter bank.

```
[LoD,HiD,LoR,HiR] = biorfilt(H0,H1);
```

The sum of the lowpass analysis and synthesis filters is now equal to $\sqrt{2}$.

```
sum(LoD)
```

```
ans = 1.4142
```

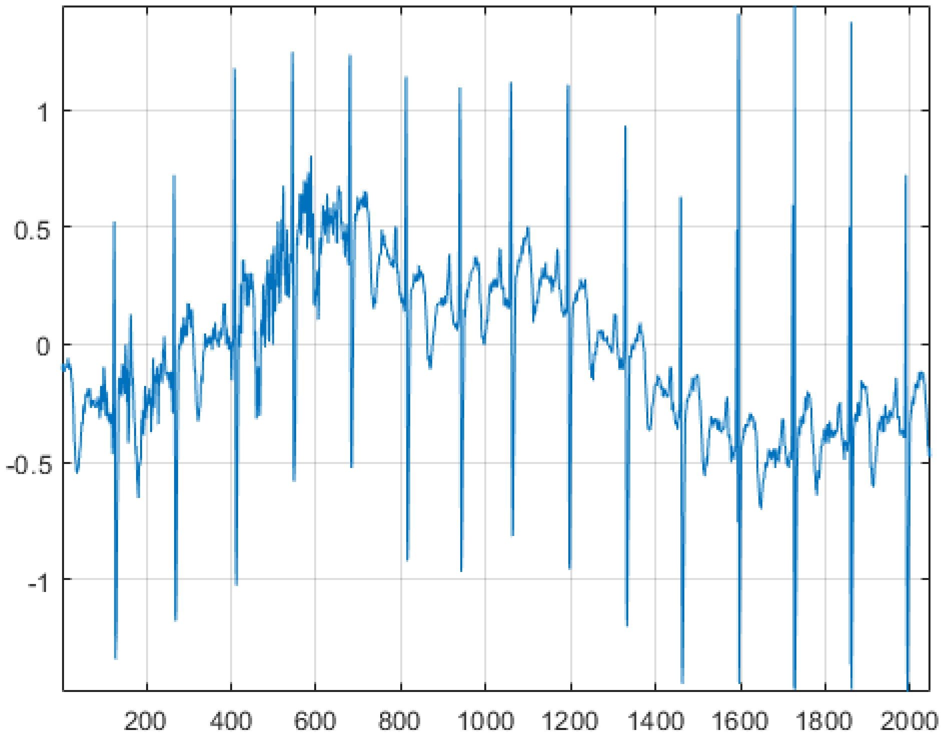
```
sum(LoR)
```

```
ans = 1.4142
```

The wavelet filters sum, as required, to zero. The L2-norms of the lowpass analysis and highpass synthesis filters are equal. The same holds for the lowpass synthesis and highpass analysis filters.

Now you can use these filters in discrete wavelet and wavelet packet transforms and achieve a PR wavelet packet filter bank. To demonstrate this, load and plot an ECG signal.

```
load wecg
plot(wecg)
axis tight
grid on
```

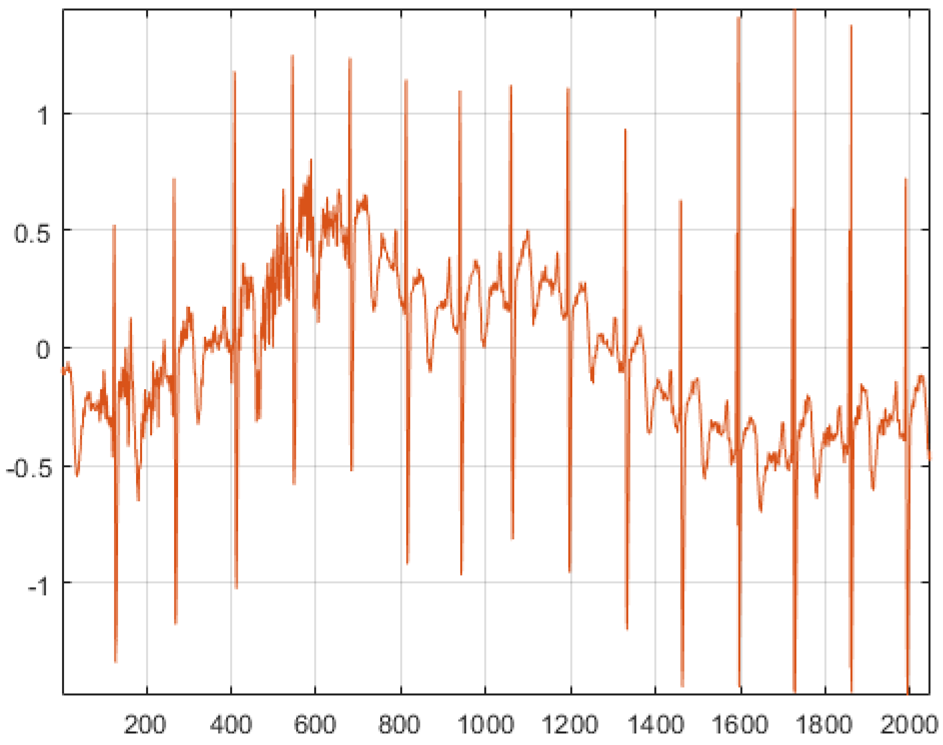


Obtain the discrete wavelet packet transform of the ECG signal using the LeGall 5/3 filter set.

```
[wpt,L] = dwpt(wecg,LoD,HiD);
```

Now use the reconstruction (synthesis) filters to reconstruct the signal and demonstrate perfect reconstruction.

```
xrec = idwpt(wpt,L,LoR,HiR);
plot([wecg xrec])
axis tight, grid on;
```



```
norm(wecg-xrec,'Inf')
```

```
ans = 3.1086e-15
```

You can also use this filter bank in the 1-D and 2-D discrete wavelet transforms. Read and plot an image.

```
im = imread('woodsculp256.jpg');
image(im); axis off;
```

Obtain the 2-D wavelet transform using the LeGall 5/3 analysis filters.

```
[C,S] = wavedec2(im,3,LoD,HiD);
```

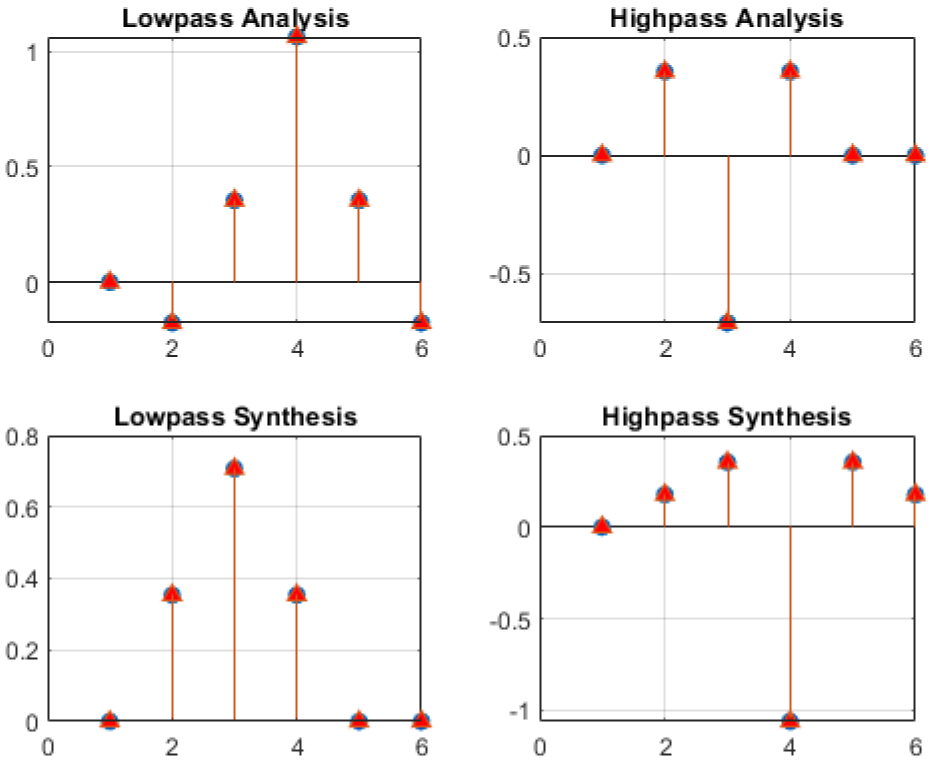
Reconstruct the image using the synthesis filters.

```
imrec = waverec2(C,S,LoR,HiR);
image(uint8(imrec)); axis off;
```

The LeGall 5/3 filter is equivalent to the built-in 'bior2.2' wavelet in Wavelet Toolbox. Use the 'bior2.2' filters and compare with the LeGall 5/3 filters.

```
[LD,HD,LR,HR] = wfilters('bior2.2');
subplot(2,2,1)
hl = stem([LD' LoD']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Lowpass Analysis')
subplot(2,2,2)
hl = stem([HD' HiD']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Highpass Analysis')
subplot(2,2,3)
hl = stem([LR' LoR']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
```

```
title('Lowpass Synthesis')
subplot(2,2,4)
hl = stem([HR' HiR']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Highpass Synthesis')
```



Input Arguments

X — Input data

real-valued vector | real-valued matrix | timetable

Input data, specified as a real-valued vector, matrix, or timetable. If X is a matrix, the transform is applied to each column of X. If X is a timetable, X must either contain a matrix in a single variable or column vectors in separate variables. X must be uniformly sampled.

Data Types: single | double

wname — Wavelet

'fk18' (default) | character vector | string scalar

Wavelet to use in the DWPT, specified as a character vector or string scalar. wname must be recognized by wavemngr.

You cannot specify both wname and a filter pair, LoD and HiD.

Example: wpt = dwpt(data,"sym4") specifies the sym4 wavelet.

LoD,HiD — Wavelet analysis filters
real-valued vectors

Wavelet analysis (decomposition) filters to use in the DWPT, specified as a pair of real-valued vectors. LoD is the scaling (lowpass) analysis filter, and HiD is the wavelet (highpass) analysis filter. You cannot specify both wname and a filter pair, LoD and HiD. See wfilters for additional information.

Note dwpt does not check that LoD and HiD satisfy the requirements for a perfect reconstruction wavelet packet filter bank. See “PR Biorthogonal Filters” on page 1-356 for an example of how to take a published biorthogonal filter and ensure that the analysis and synthesis filters produce a perfect reconstruction wavelet packet filter bank using dwpt.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: wpt = dwpt(x,'sym4','Level',4) specifies a level 4 decomposition using the sym4 wavelet.

Level — Wavelet decomposition level
floor($\log_2$(*Ns*)) (default) | positive integer

Wavelet decomposition level, specified as a positive integer less than or equal to floor($\log_2$(*Ns*)), where *Ns* is the number of samples in the data. If unspecified, Level defaults to floor($\log_2$(*Ns*)).

FullTree — Wavelet packet tree handling
false or 0 (default) | true or 1

Wavelet packet tree handling, specified as a numeric or logical 1 (true) or 0 (false). When set to true, wpt contains the full packet tree. When set to false, wpt contains only the terminal nodes. If unspecified, FullTree defaults to false.

Boundary — Wavelet packet transform boundary handling
'reflection' (default) | 'periodic'

Wavelet packet transform boundary handling, specified as 'reflection' or 'periodic'. Setting to 'reflection' or 'periodic', the wavelet packet coefficients are extended at each level based on the 'sym' or 'per' mode in dwtmode, respectively. If unspecified, Boundary defaults to 'reflection'.

Output Arguments

wpt — Wavelet packet transform
cell array

Wavelet packet transform, returned as a 1-by-*M* cell array. If taking the DWPT of one signal, each element of `wpt` is a vector. Otherwise, each element is a matrix. The coefficients in the *j*th row of the matrix correspond to the signal in the *j*th column of `X`. The packets are sequency-ordered.

If returning the terminal nodes of a level *N* decomposition, `wpt` is a 1-by-2^N cell array. If returning the full wavelet packet tree, `wpt` is a 1-by-($2^{N+1}-2$) cell array.

`l` — Bookkeeping vector

vector of positive integers

Bookkeeping vector, returned as a vector of positive integers. The vector `l` contains the length of the input signal and the number of coefficients by level, and is required for perfect reconstruction.

`packetlevels` — Transform levels

vector of positive integers

Transform levels, returned as a vector of positive integers. The *i*th element of `packetlevels` corresponds to the *i*th element of `wpt`. If `wpt` contains only the terminal nodes, `packetlevels` is a vector with each element equal to the terminal level. If `wpt` contains the full wavelet packet tree, then `packetlevels` is a vector with 2^j elements for each level *j*.

`f` — Center frequencies

real-valued vector

Center frequencies of the approximate passbands in cycles per sample, returned as a real-valued vector. The *j*the element of `f` corresponds to the *j*th wavelet packet node of `wpt`. You can multiply the elements in `f` by a sampling frequency to convert to cycles per unit time.

`re` — Relative energy

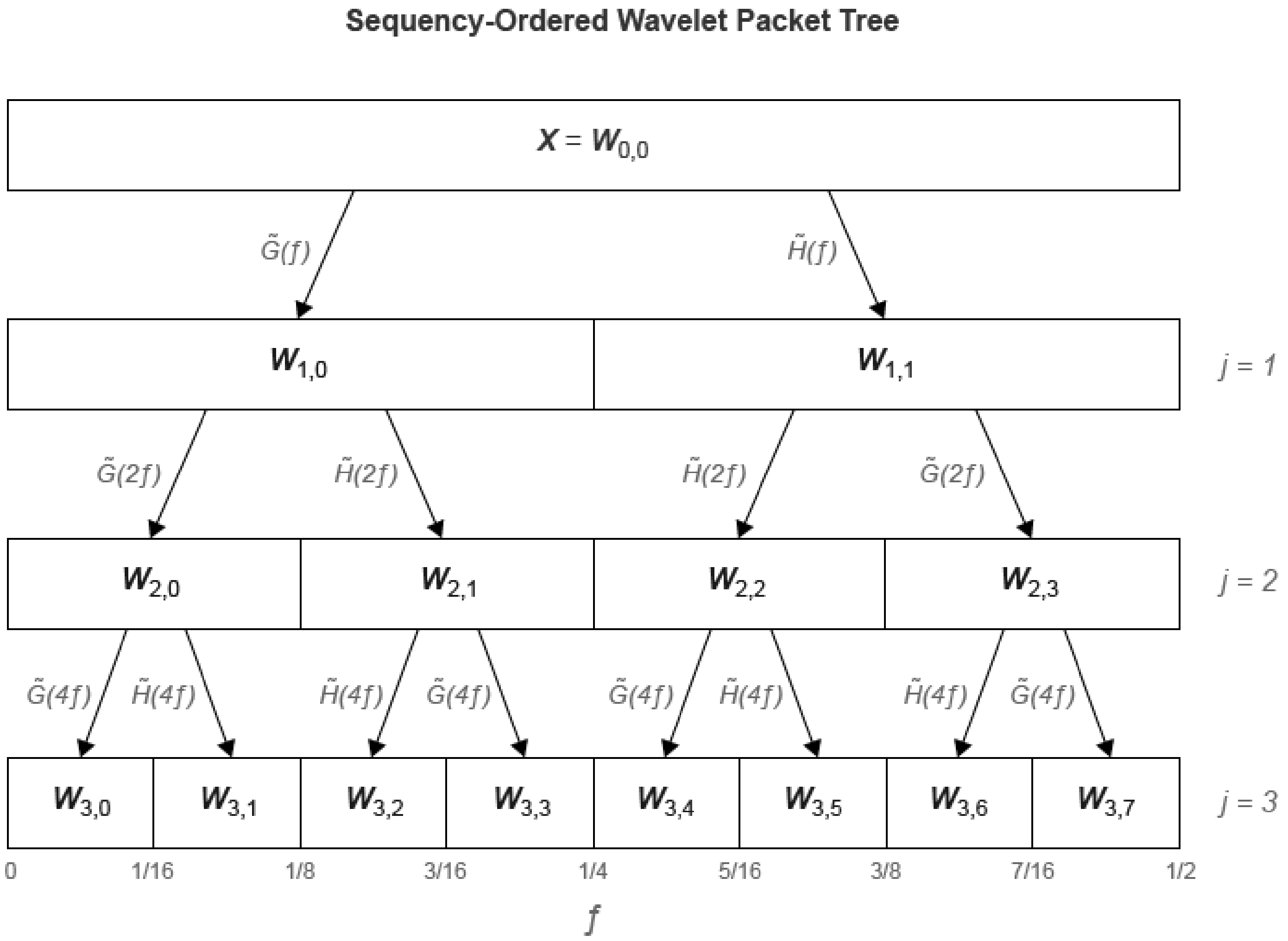
cell array

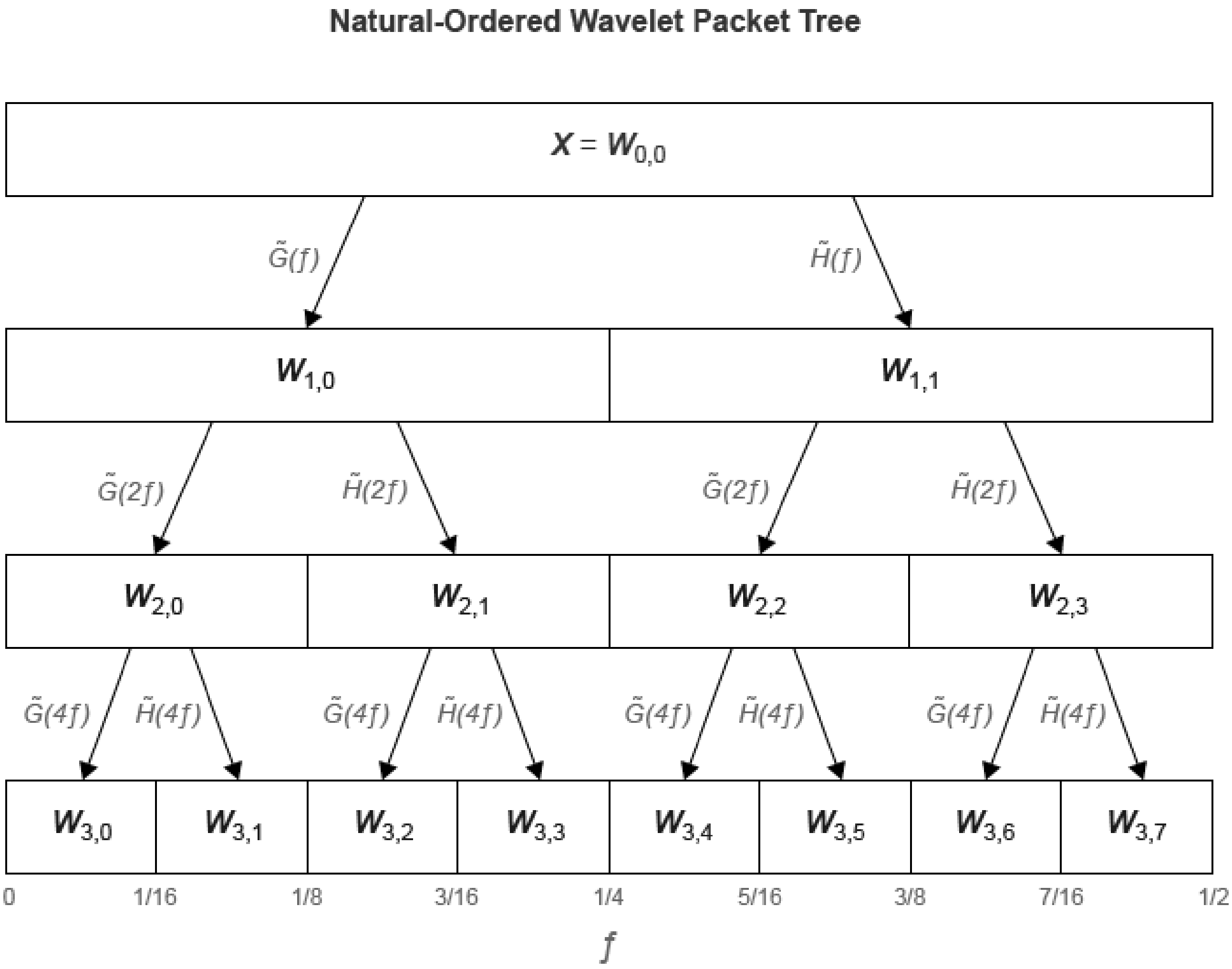
Relative energy for the wavelet packets in `wpt`, returned as a cell array. The relative energy is the proportion of energy contained in each wavelet packet by level. The *j*th element of `re` corresponds to the *j*th wavelet packet node of `wpt`.

Each element of `re` is a scalar when taking the DWPT of one signal. Otherwise, when taking the DWPT of *M* signals, each element of `re` is a *M*-by-1 vector, where the *i*th element is the relative energy of the *i*th signal channel. For each channel, the sum of relative energies in the wavelet packets at a given level is equal to 1.

Algorithms

The `dwpt` function performs a discrete wavelet packet transform and produces a sequency-ordered wavelet packet tree. Compare the sequency-ordered and normal (Paley)-ordered trees. $\widetilde{G}(f)$ is the scaling (lowpass) analysis filter, and $\widetilde{H}(f)$ represents the wavelet (highpass) analysis filter. The labels at the bottom show the partition of the frequency axis [0, ½].


Sequency-Ordered Wavelet Packet Tree
X = W0,0
G̃(f)
H̃(f)
W1,0
W1,1
j = 1
G̃(2f)
H̃(2f)
H̃(2f)
G̃(2f)
W2,0
W2,1
W2,2
W2,3
j = 2
G̃(4f)
H̃(4f)
H̃(4f)
G̃(4f)
G̃(4f)
H̃(4f)
H̃(4f)
G̃(4f)
W3,0
W3,1
W3,2
W3,3
W3,4
W3,5
W3,6
W3,7
j = 3
0
1/16
1/8
3/16
1/4
5/16
3/8
7/16
1/2
f



References

[1] Wickerhauser, Mladen Victor. *Adapted Wavelet Analysis from Theory to Software*. Wellesley, MA: A.K. Peters, 1994.

[2] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[3] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Timetable input data is not supported.
- The input `wname` must be constant.

See Also

`modwpt` | `idwpt`

Topics

“Wavelet Packets: Decomposing the Details”

Introduced in R2020a

dwt

Single-level 1-D discrete wavelet transform

Syntax

```
[cA,cD] = dwt(x,wname)
[cA,cD] = dwt(x,LoD,HiD)
[cA,cD] = dwt( ___ ,'mode',extmode)
```

Description

`[cA,cD] = dwt(x,wname)` returns the single-level discrete wavelet transform (DWT) of the vector `x` using the wavelet specified by `wname`. The wavelet must be recognized by `wavemngr`. `dwt` returns the approximation coefficients vector `cA` and detail coefficients vector `cD` of the DWT.

Note If your application requires a multilevel wavelet decomposition, consider using `wavedec`.

`[cA,cD] = dwt(x,LoD,HiD)` returns the single-level DWT using the specified lowpass and highpass wavelet decomposition filters `LoD` and `HiD`, respectively.

`[cA,cD] = dwt( ___ ,'mode',extmode)` returns the single-level DWT with the specified extension mode `extmode`. For more information, see `dwtmode`. This argument can be added to any of the previous input syntaxes.

Note For `gpuArray` inputs, the supported modes are `'symh'` (`'sym'`) and `'per'`. All `'mode'` options except `'per'` are converted to `'symh'`. See the example “Single-Level Discrete Wavelet Transform on a GPU” on page 1-370.

Examples

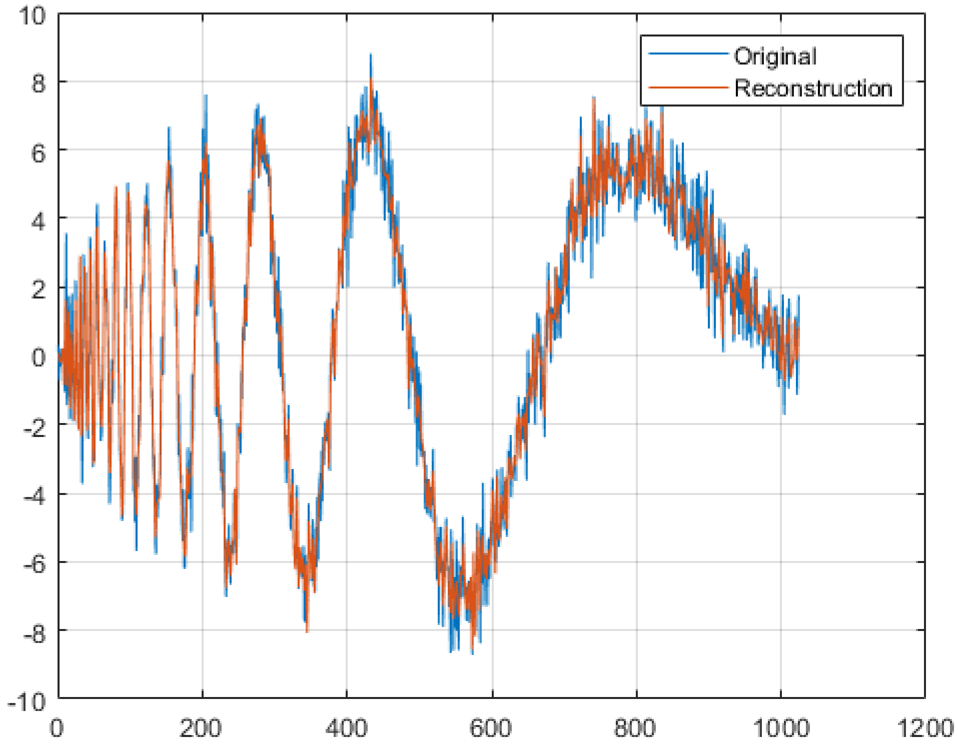
DWT Using Wavelet Name

Obtain the single-level DWT of the noisy Doppler signal using a wavelet name.

```
load noisdopp;
[cA,cD] = dwt(noisdopp,'sym4');
```

Reconstruct a smoothed version of the signal using the approximation coefficients. Plot and compare with the original signal.

```
xrec = idwt(cA,zeros(size(cA)),'sym4');
plot(noisdopp)
hold on
grid on
plot(xrec)
legend('Original','Reconstruction')
```



DWT Using Wavelet and Scaling Filters

Obtain the single-level DWT of a noisy Doppler signal using the wavelet (highpass) and scaling (lowpass) filters.

```
load noisdopp;
[LoD,HiD] = wfilters('bior3.5','d');
[cA,cD] = dwt(noisdopp,LoD,HiD);
```

Create a DWT filter bank that can be applied to the noisy Doppler signal using the same wavelet. Obtain the highpass and lowpass filters from the filter bank.

```
len = length(noisdopp);
fb = dwtfilterbank('SignalLength',len,'Wavelet','bior3.5');
[lo,hi] = filters(fb);
```

For the `bior3.5` wavelet, `lo` and `hi` are 12-by-2 matrices. `lo` are the lowpass filters, and `hi` are the highpass filters. The first columns of `lo` and `hi` are used for analysis and the second columns are used for synthesis. Compare the first column of `lo` and `hi` with `LoD` and `HiD` respectively. Confirm they are equal.

```
disp('Lowpass Analysis Filters')
```

```
Lowpass Analysis Filters
```

```
[lo(:,1) LoD']
```

```
ans = 12×2

   -0.0138   -0.0138
    0.0414    0.0414
    0.0525    0.0525
   -0.2679   -0.2679
   -0.0718   -0.0718
    0.9667    0.9667
    0.9667    0.9667
   -0.0718   -0.0718
   -0.2679   -0.2679
    0.0525    0.0525
      ⋮
```

```
disp('Highpass Analysis Filters')
```

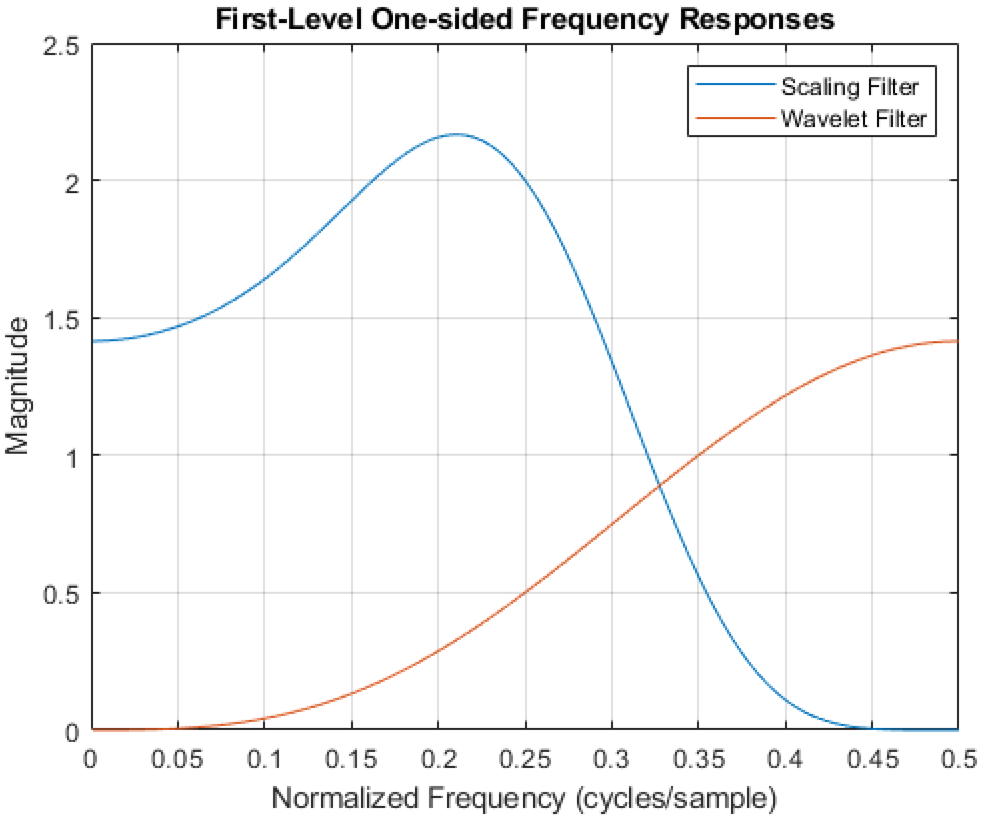
```
Highpass Analysis Filters
```

```
[hi(:,1) HiD']
```

```
ans = 12×2

         0         0
         0         0
         0         0
         0         0
   -0.1768   -0.1768
    0.5303    0.5303
   -0.5303   -0.5303
    0.1768    0.1768
         0         0
         0         0
      ⋮
```

Plot the one-sided magnitude frequency responses of the first-level wavelet and scaling filters.

```
[psidft,f,phidft] = freqz(fb);
level = 1;
plot(f(len/2+1:end),abs(phidft(level,len/2+1:end)))
hold on
plot(f(len/2+1:end),abs(psidft(level,len/2+1:end)))
grid on
legend('Scaling Filter','Wavelet Filter')
title('First-Level One-sided Frequency Responses')
xlabel('Normalized Frequency (cycles/sample)')
ylabel('Magnitude')
```



Single-Level Discrete Wavelet Transform on a GPU

Refer to “GPU Support by Release” (Parallel Computing Toolbox) to see what GPUs are supported.

Load the noisy Doppler signal. Put the signal on the GPU using `gpuArray`. Save the current extension mode.

```
load noisdopp
noisdoppg = gpuArray(noisdopp);
origMode = dwtmode('status','nodisp');
```

Use `dwtmode` to change the extension mode to zero-padding. Obtain the single-level discrete wavelet transform of the signal on the GPU using the `db2` wavelet.

```
dwtmode('zpd','nodisp')
[cA,cD] = dwt(noisdoppg,'db2');
```

The current extension mode `zpd` is not supported for `gpuArray` input. Therefore, the DWT is instead performed using the `sym` extension mode. Confirm this by taking the DWT of `noisdoppg` with the extension mode set to `sym` and compare with the previous result.

```
[cAsym,cDsym] = dwt(noisdoppg,'db2','mode','sym');
[max(abs(cA-cAsym)) max(abs(cD-cDsym))]
```

```
ans =

     0     0
```

An unsupported extension mode specified as an input argument is converted to `'sym'`. Confirm that taking the DWT of `noisdoppg` with `'mode'` set to an unsupported mode also defaults to the `sym` extension mode.

```
[cA,cD] = dwt(noisdoppg,'db2','mode','spd');
[max(abs(cA-cAsym)) max(abs(cD-cDsym))]
```

```
ans =

     0     0
```

Change the current extension mode to periodic. Obtain the single-level discrete wavelet transform of the signal on the GPU using the `db2` wavelet.

```
dwtmode('per','nodisp')
[cA,cD] = dwt(noisdoppg,'db2');
```

Confirm the current extension mode `per` is supported for `gpuArray` input.

```
[cAper,cDper] = dwt(noisdopp,'db2','mode','per');
[max(abs(cA-cAper)) max(abs(cD-cDper))]
```

```
ans =

     0     0
```

Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

x — Input data
vector

Input data, specified as a vector.

Data Types: `single` | `double`

wname — Analyzing wavelet
character vector | string scalar

Analyzing wavelet used to the compute the single-level DWT, specified as a character vector or string scalar. The wavelet must be recognized by `wavemngr`. The analyzing wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

Example: `'db4'`

LoD,HiD — Wavelet decomposition filters
even-length real-valued vectors

Wavelet decomposition filters, specified as a pair of even-length real-valued vectors. `LoD` is the lowpass decomposition filter, and `HiD` is the highpass decomposition filter. The lengths of `LoD` and `HiD` must be equal. See `wfilters` for additional information.

Data Types: `single` | `double`

`extmode` — Extension mode
`'zpd'` | `'sp0'` | `'spd'` | ...

Extension mode used when performing the DWT, specified as one of the following:

mode	DWT Extension Mode
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1
`'sym'` or `'symh'`	Symmetric extension (half point): boundary value symmetric replication
`'symw'`	Symmetric extension (whole point): boundary value symmetric replication
`'asym'` or `'asymh'`	Antisymmetric extension (half point): boundary value antisymmetric replication
`'asymw'`	Antisymmetric extension (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.

The global variable managed by `dwtmode` specifies the default extension mode. See `dwtmode` for extension mode descriptions.

Example: `[cA,cD] = dwt(x,'db4','mode','symw')` returns the single-level DWT of `x` using the order 4 Daubechies extremal phase wavelet and whole point symmetric extension.

Output Arguments

`cA` — Approximation coefficients
vector

Approximation coefficients obtained from the wavelet decomposition, returned as a vector. Convolving the input signal `x` with the scaling filter `LoD`, followed by dyadic decimation, yields the approximation coefficients. Let `sx = size(x)` and `lf =` the length of the decomposition filters.

- If the DWT extension mode is set to periodization, `cA` is a vector of length `ceil(sx/2)`.
- For the other extension modes, `cA` is a vector of length `floor((sx+lf-1)/2)`.

Data Types: `single` | `double`

cD — Detail coefficients

vector

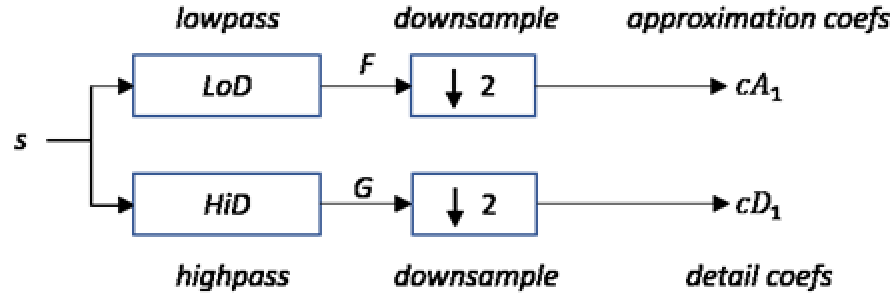
Detail coefficients obtained from the wavelet decomposition, returned as a vector. Convolving the input signal `x` with the wavelet filter `HiD`, followed by dyadic decimation, yields the detail coefficients. Let `sx = size(x)` and `lf =` the length of the decomposition filters.

- If the DWT extension mode is set to periodization, `cD` is a vector of length `ceil(sx/2)`.
- For the other extension modes, `cD` is a vector of length `floor((sx+lf-1)/2)`.

Data Types: `single` | `double`

Algorithms

Starting from a signal s of length N, two sets of coefficients are computed: approximation coefficients cA_1, and detail coefficients cD_1. Convolving s with the scaling filter `LoD`, followed by dyadic decimation, yields the approximation coefficients. Similarly, convolving s with the wavelet filter `HiD`, followed by dyadic decimation, yields the detail coefficients.



where

- $\boxed{X}$ — Convolve with filter X
- $\boxed{\downarrow 2}$ — Downsample (keep the even-indexed elements)

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 floor$\left(\frac{N-1}{2}\right) + n$.

To deal with signal-end effects resulting from a convolution-based algorithm, a global variable managed by `dwtmode` defines the kind of signal extension mode used. The possible options include zero-padding and symmetric extension, which is the default mode.

Note For the same input, the `dwt` function and the DWT block in the DSP System Toolbox™ do not produce the same results. The DWT block is designed for real-time implementation while Wavelet Toolbox software is designed for analysis, so the products 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

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input wname must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input wname must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only 'sym' and 'per' extension modes are supported. See dwtmode.

See Also

wavedec | idwt | dwtmode | waveinfo | dwtfilterbank

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,LoD,HiD)
[cA,cH,cV,cD] = dwt2( ___ ,'mode',extmode)
```

Description

`dwt2` computes the single-level 2-D wavelet decomposition. Compare `dwt2` with `wavedec2` which may be more useful for your application. The decomposition is done with respect to either a particular wavelet (see `wfilters` for more information) or particular wavelet decomposition filters.

`[cA,cH,cV,cD] = dwt2(X,wname)` computes the single-level 2-D discrete wavelet transform (DWT) of the input data `X` using the `wname` wavelet. `dwt2` returns the approximation coefficients matrix `cA` and detail coefficients matrices `cH`, `cV`, and `cD` (horizontal, vertical, and diagonal, respectively).

`[cA,cH,cV,cD] = dwt2(X,LoD,HiD)` computes the single-level 2-D DWT using the wavelet decomposition lowpass filter `LoD` and highpass filter `HiD`. The decomposition filters must have the same length and an even number of samples.

`[cA,cH,cV,cD] = dwt2( ___ ,'mode',extmode)` computes the single-level 2-D DWT with the extension mode `extmode`. Include this argument after all other arguments.

Note For `gpuArray` inputs, the supported modes are `'symh'` (`'sym'`) and `'per'`. All `'mode'` options except `'per'` are converted to `'symh'`. See the example “Single-Level 2-D Discrete Wavelet Transform on a GPU” on page 1-380.

Examples

Single-Level 2-D Discrete Wavelet Transform of an Image

Load and display an image.

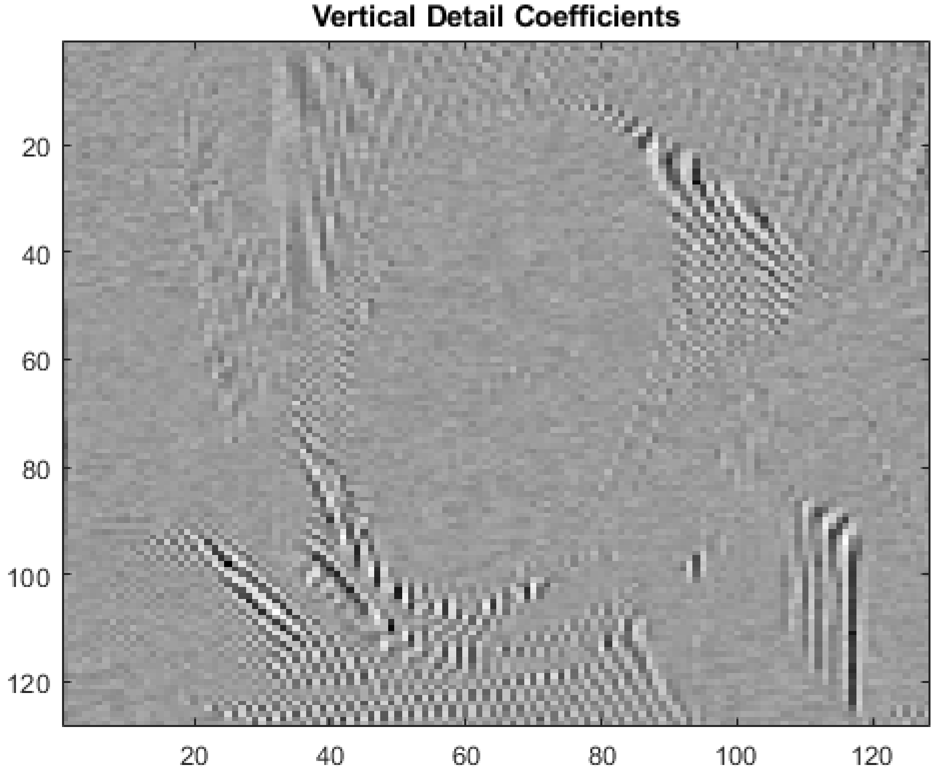
```
load woman
imagesc(X)
colormap(map)
```

Obtain the single-level 2-D discrete wavelet transform of the image using the order 4 symlet and periodic extension.

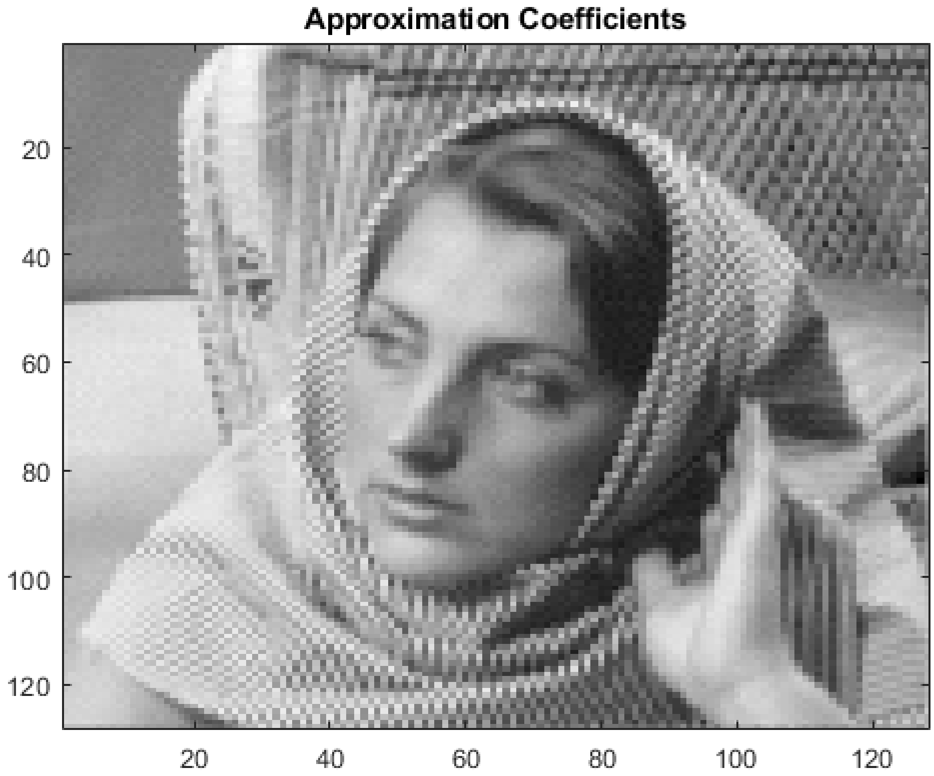
```
[cA,cH,cV,cD] = dwt2(X,'sym4','mode','per');
```

Display the vertical detail coefficients and the approximation coefficients.

```
imagesc(cV)
title('Vertical Detail Coefficients')
```



```
imagesc(cA)
title('Approximation Coefficients')
```



Single-Level 2-D Discrete Wavelet Transform Using Filters

Load and display an image.

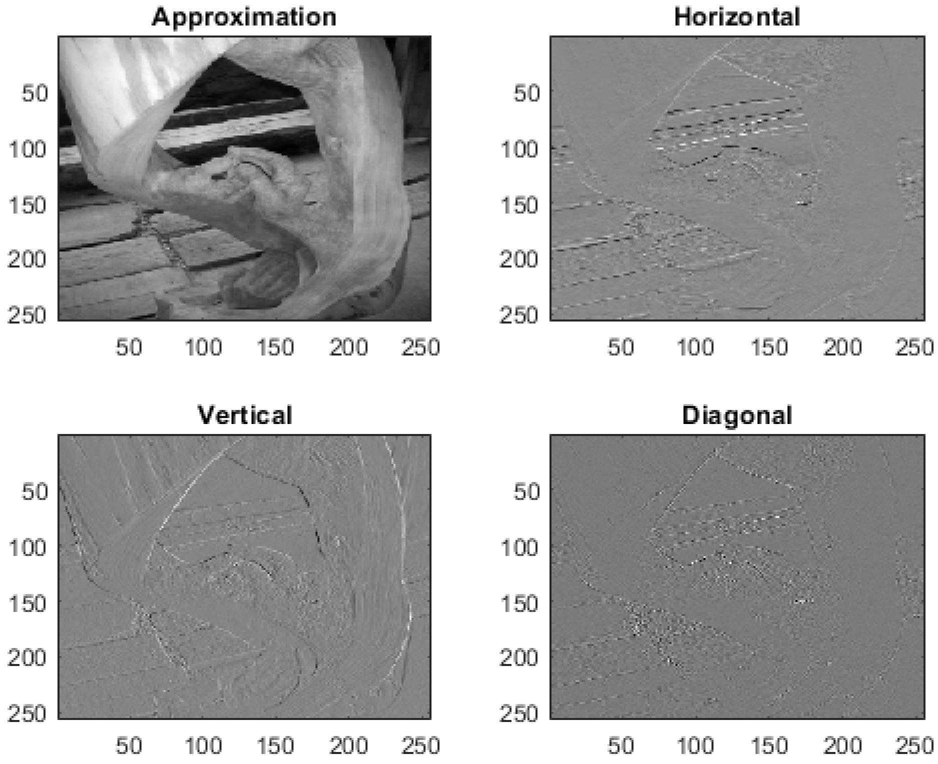
```
load sculpture
imagesc(X)
colormap gray
```

Generate the lowpass and highpass decomposition filters for the Haar wavelet.

```
[LoD,HiD] = wfilters('haar','d');
```

Use the filters to perform a single-level 2-D wavelet decomposition. Use half-point symmetric extension. Display the approximation and detail coefficients.

```
[cA,cH,cV,cD] = dwt2(X,LoD,HiD,'mode','symh');
subplot(2,2,1)
imagesc(cA)
colormap gray
title('Approximation')
subplot(2,2,2)
imagesc(cH)
colormap gray
title('Horizontal')
subplot(2,2,3)
imagesc(cV)
colormap gray
title('Vertical')
subplot(2,2,4)
imagesc(cD)
colormap gray
title('Diagonal')
```



Single-Level 2-D Discrete Wavelet Transform on a GPU

Refer to “GPU Support by Release” (Parallel Computing Toolbox) to see what GPUs are supported.

Load an image. Put the image on the GPU using `gpuArray`. Save the current extension mode.

```
load mask
imgg = gpuArray(X);
origMode = dwtmode('status','nodisp');
```

Use `dwtmode` to change the extension mode to zero-padding. Obtain the single-level 2-D DWT of the image on the GPU using the `db2` wavelet.

```
dwtmode('zpd','nodisp')
[cA,cH,cV,cD] = dwt2(imgg,'db2');
```

The current extension mode `zpd` is not supported for `gpuArray` input. Therefore, the DWT is instead performed using the `sym` extension mode. Confirm this by taking the DWT of `imgg` with the extension mode set to `sym` and compare with the previous result.

```
[cAsym,cHsym,cVsym,cDsym] = dwt2(imgg,'db2','mode','sym');
[max(abs(cA(:)-cAsym(:))) max(abs(cH(:)-cHsym(:))) ...
    max(abs(cV(:)-cVsym(:))) max(abs(cD(:)-cDsym(:)))]
```

```
ans =

     0     0     0     0
```

An unsupported extension mode specified as an input argument is converted to `'sym'`. Confirm that taking the DWT of `imgg` with `'mode'` set to an unsupported mode also defaults to the `sym` extension mode.

```
[cA,cH,cV,cD] = dwt2(imgg,'db2','mode','spd');
[max(abs(cA(:)-cAsym(:))) max(abs(cH(:)-cHsym(:))) ...
    max(abs(cV(:)-cVsym(:))) max(abs(cD(:)-cDsym(:)))]
```

```
ans =

     0     0     0     0
```

Change the current extension mode to periodic. Obtain the single-level DWT of the image on the GPU using the `db2` wavelet.

```
dwtmode('per','nodisp')
[cA,cH,cV,cD] = dwt2(imgg,'db2');
```

Confirm the current extension mode `per` is supported for `gpuArray` input.

```
[cAper,cHper,cVper,cDper] = dwt2(imgg,'db2','mode','per');
[max(abs(cA(:)-cAper(:))) max(abs(cH(:)-cHper(:))) ...
    max(abs(cV(:)-cVper(:))) max(abs(cD(:)-cDper(:)))]
```

```
ans =

     0     0     0     0
```

Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

X — Input data

numeric array | logical array

Input data, specified as a numeric or logical array. `X` can be an `m`-by-`n` array representing an indexed image or an `m`-by-`n`-by-3 array representing a truecolor image. For more information on truecolor images, see “RGB (Truecolor) Images”.

Data Types: `double`

wname — Analyzing wavelet

character vector | string scalar

Analyzing wavelet used to compute the 2-D DWT, specified as a character vector or string scalar. The analyzing wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

LoD — Wavelet decomposition lowpass filter

even-length real-valued vector

Wavelet decomposition lowpass filter, specified as an even-length real-valued vector. LoD must be of the same length as HiD.

Data Types: double

HiD — Wavelet decomposition highpass filter

even-length real-valued vector

Wavelet decomposition highpass filter, specified as an even-length real-valued vector. HiD must be of the same length as LoD.

Data Types: double

extmode — Extension mode

'zpd' | 'sp0' | 'spd' | ...

Extension mode used when performing the DWT, specified as one of the following:

mode	DWT Extension Mode
'zpd'	Zero extension
'sp0'	Smooth extension of order 0
'spd' (or 'sp1')	Smooth extension of order 1
'sym' or 'symh'	Symmetric extension (half point): boundary value symmetric replication
'symw'	Symmetric extension (whole point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric extension (half point): boundary value antisymmetric replication
'asymw'	Antisymmetric extension (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.

The global variable managed by dwtmode specifies the default extension mode.

Example: `[cA,cH,cV,cD] = dwt2(x,'db4','mode','symw');`

Output Arguments

cA — Approximation coefficients

array

Approximation coefficients, returned as an array whose size depends on X. Let sx = size(X) and lf = the length of the decomposition filters.

- If the DWT extension mode is set to periodization, then this output is of size ceil(sx/2).

- For the other extension modes, this output is of size `floor((sx+lf-1)/2)`.

Data Types: `double`

`cH` — Horizontal detail coefficients
array

Horizontal detail coefficients, returned as an array whose size depends on `X`. Let `sx = size(X)` and `lf =` the length of the decomposition filters.

- If the DWT extension mode is set to periodization, then this output is of size `ceil(sx/2)`.
- For the other extension modes, this output is of size `floor((sx+lf-1)/2)`.

Data Types: `double`

`cV` — Vertical detail coefficients
array

Vertical detail coefficients, returned as an array whose size depends on `X`. Let `sx = size(X)` and `lf =` the length of the decomposition filters.

- If the DWT extension mode is set to periodization, then this output is of size `ceil(sx/2)`.
- For the other extension modes, this output is of size `floor((sx+lf-1)/2)`.

Data Types: `double`

`cD` — Diagonal detail coefficients
array

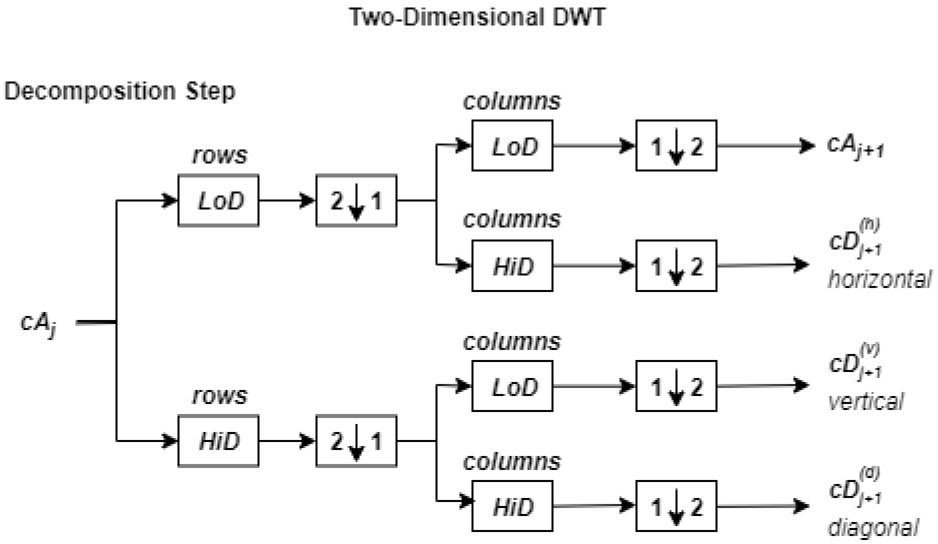
Diagonal detail coefficients, returned as an array whose size depends on `X`. Let `sx = size(X)` and `lf =` the length of the decomposition filters.

- If the DWT extension mode is set to periodization, then this output is of size `ceil(sx/2)`.
- For the other extension modes, this output is of size `floor((sx+lf-1)/2)`.

Data Types: `double`

Algorithms

The 2-D wavelet decomposition algorithm for images is similar to the one-dimensional case. The two-dimensional wavelet and scaling functions are obtained by taking the tensor products of the one-dimensional wavelet and scaling functions. 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.



where

- $\boxed{2\downarrow 1}$ — Downsample columns: keep the even-indexed columns
- $\boxed{1\downarrow 2}$ — Downsample rows: keep the even-indexed rows
- $\overset{rows}{\boxed{X}}$ — Convolve with filter *X* the rows of the entry
- $\overset{columns}{\boxed{X}}$ — Convolve with filter *X* the columns of the entry

The decomposition is initialized by setting the approximation coefficients equal to the image *s*: $cA_0 = s$.

Note To deal with signal-end effects introduced by a convolution-based algorithm, the 1-D and 2-D DWT use a global variable managed by `dwtmode`. This variable defines the kind of signal extension mode used. The possible options include zero-padding and symmetric extension, which is the default mode.

References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S.G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11, no. 7 (July 1989): 674–93. https://doi.org/10.1109/34.192463.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

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

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

`wt = dwt3(x,wname)` returns the single-level three-dimensional wavelet decomposition `wt` of the input data `x` using the `wname` wavelet. The default extension mode of the 3-D discrete wavelet transform (DWT) is `'sym'` (see `dwtmode`).

`wt = dwt3(x,wname,'mode',extM)` uses the extension mode `extM` (see `dwtmode`).

`wt = dwt3(x,w,___)` specifies three wavelets, one for each direction. `w` is a cell array, string array, or structure, and can be followed by `'mode',extM`.

`wt = dwt3(x,wf,___)` specifies four filters, two for decomposition and two for reconstruction, or 3 × 4 filters (one quadruplet by direction). `wf` is a cell array or structure, and can be followed by `'mode',extM`..

Examples

Single-Level Three-Dimensional Wavelet Decomposition

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
```

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}
```

Input Arguments

x — Input data

3-D array

Input data, specified as a 3-D array.

Data Types: `double`

wname — Analyzing wavelet

character vector | string scalar

Analyzing wavelet used to compute the 2-D DWT, specified as a character vector or string scalar. The analyzing wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

w — Analyzing wavelets

cell array of character vectors | string array | structure

Analyzing wavelets to use in the 3-D wavelet decomposition, one for each direction, specified as a cell array of character vectors, a string array, or a structure. `w = {'wname1','wname2','wname3'}`, or `w = ["wname1","wname2","wname3"]`, or `w` is a structure with 3 fields `'w1'`, `'w2'`, `'w3'` containing character vectors or string scalars that are the names of wavelets.

Example: `wt = dwt3(x,["db2","db4","db6"]);`

wf — Wavelet filters

cell array | structure

Wavelet filters to use in the 3-D wavelet decomposition, specified as either a cell array or structure. `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'`.

extM — Extension mode

`'zpd'` | `'sp0'` | `'spd'` | ...

Extension mode used when performing the 3-D DWT, specified as one of the following:

mode	DWT Extension Mode
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1
`'sym'` or `'symh'`	Symmetric extension (half point): boundary value symmetric replication

mode	DWT Extension Mode
'symw'	Symmetric extension (whole point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric extension (half point): boundary value antisymmetric replication
'asymw'	Antisymmetric extension (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.

The global variable managed by dwtmode specifies the default extension mode.

Output Arguments

wt — Single-level 3-D wavelet decomposition
structure

Single-level 3-D wavelet decomposition, returned as a structure with the following fields:

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.

See Also

dwtmode | idwt3 | wavedec3 | waverec3 | waveinfo | wfilters

Introduced in R2010a

dwtfilterbank

Discrete wavelet transform filter bank

Description

Use `dwtfilterbank` to create a discrete wavelet transform (DWT) filter bank

- Visualize wavelets and scaling functions in time and frequency.
- Measure the 3-dB bandwidths of the wavelet and scaling functions. You can also measure energy concentration of the wavelet and scaling functions in the theoretical DWT passbands.
- Create a DWT filter bank using your own custom filters. You can determine whether the filter bank is orthogonal or biorthogonal.
- Determine the frame bounds of the filter bank.

Creation

Syntax

```
fb = dwtfilterbank
fb = dwtfilterbank(Name,Value)
```

Description

`fb = dwtfilterbank` create a discrete wavelet transform (DWT) filter bank. The default filter bank is designed for a signal with 1024 samples. The default filter bank uses the analysis (decomposition) sym4 wavelet and scaling filter with seven resolution levels.

`fb = dwtfilterbank(Name,Value)` creates a DWT filter bank `fb` with properties specified by one or more `Name,Value` pair arguments. Properties can be specified in any order as `Name1,Value1,...,NameN,ValueN`. Enclose each property name in quotes.

For example, `fb = dwtfilterbank('SignalLength',1000,'Wavelet','bior4.4')` creates a DWT filter bank for signals of length 1000 using the biorthogonal `bior4.4` wavelet.

Note You cannot change a property value of an existing filter bank. For example, if you have a filter bank `fb` for the `sym4` wavelet, you must create a second filter bank `fb2` for the `coif5` wavelet. You cannot assign a different `Wavelet` to `fb`.

Properties

`SignalLength` — Signal length
1024 (default) | positive integer greater than or equal to 2

Signal length, specified as a positive integer greater than or equal to 2.

Example: `'SignalLength',768`

Data Types: double

Wavelet — Name of wavelet

'sym4' (default) | 'Custom' | character vector | string scalar

Name of wavelet used to construct the filter bank, specified as a character vector or string scalar. Wavelet is an orthogonal or biorthogonal wavelet recognized by wavemngr or 'Custom'.

To use a wavelet filter not recognized by wavemngr, set the Wavelet property to 'Custom' and specify the “CustomWaveletFilter” on page 1-0 and “CustomScalingFilter” on page 1-0 properties.

Example: 'Wavelet','bior4.4'

Data Types: char | string

FilterType — Wavelet filter type

'Analysis' (default) | 'Synthesis'

Wavelet filter type, specified as one of 'Analysis' or 'Synthesis'. 'Analysis' uses the decomposition filters returned by wfilters. 'Synthesis' uses the reconstruction filters.

Level — Wavelet transform level

7 (default) | positive integer

Wavelet transform level, specified as a positive integer less than or equal to floor(log2(SignalLength)). For a signal of length 1024 and the sym4 wavelet, the default level is 7.

By default the level is equal to floor(log2(SignalLength/(L-1))) where *L* is the length of the wavelet filter associated with Wavelet. For wavelets recognized by wavemngr, the transform level is equal to wmaxlev(SignalLength,Wavelet). If floor(log2(SignalLength/(*L*-1))) is less than or equal to 0, Level defaults to floor(log2(SignalLength)).

SamplingFrequency — Sampling frequency in hertz

1 (default) | positive scalar

Sampling frequency in hertz, specified as a positive scalar. If unspecified, frequencies are in cycles/sample and the Nyquist frequency is ½.

Example: 'SamplingFrequency',5

Data Types: double

CustomWaveletFilter — Custom wavelet filter coefficients

even-length column vector | two-column matrix with even number of rows

Custom wavelet filter coefficients, specified as a real-valued column vector or matrix. CustomWaveletFilter must be an even-length column vector for an orthogonal wavelet or a two-column matrix with an even number of rows for a biorthogonal wavelet.

This property applies only when Wavelet is set to 'Custom'.

CustomScalingFilter — Custom scaling filter coefficients

even-length column vector | two-column matrix with even number of rows

Custom scaling filter coefficients, specified as a real-valued column vector or matrix. `CustomScalingFilter` must be an even-length column vector for an orthogonal wavelet or a two-column matrix with an even number of rows for a biorthogonal wavelet.

This property applies only when `Wavelet` is set to `'Custom'`.

Object Functions

dwtpassbands	DWT filter bank passbands
filters	DWT filter bank filters
framebounds	DWT filter bank frame bounds
freqz	DWT filter bank frequency responses
isBiorthogonal	Determine if DWT filter bank is biorthogonal
isOrthogonal	Determine if DWT filter bank is orthogonal
powerbw	DWT filter bank power bandwidth
qfactor	DWT filter bank quality factor
scalingfunctions	DWT filter bank time-domain scaling functions
wavelets	DWT filter bank time-domain wavelets
waveletsupport	DWT filter bank time supports

Examples

Discrete Wavelet Transform Filter Bank with Default Values

Create a DWT filter bank using default values.

```
fb = dwtfilterbank
```

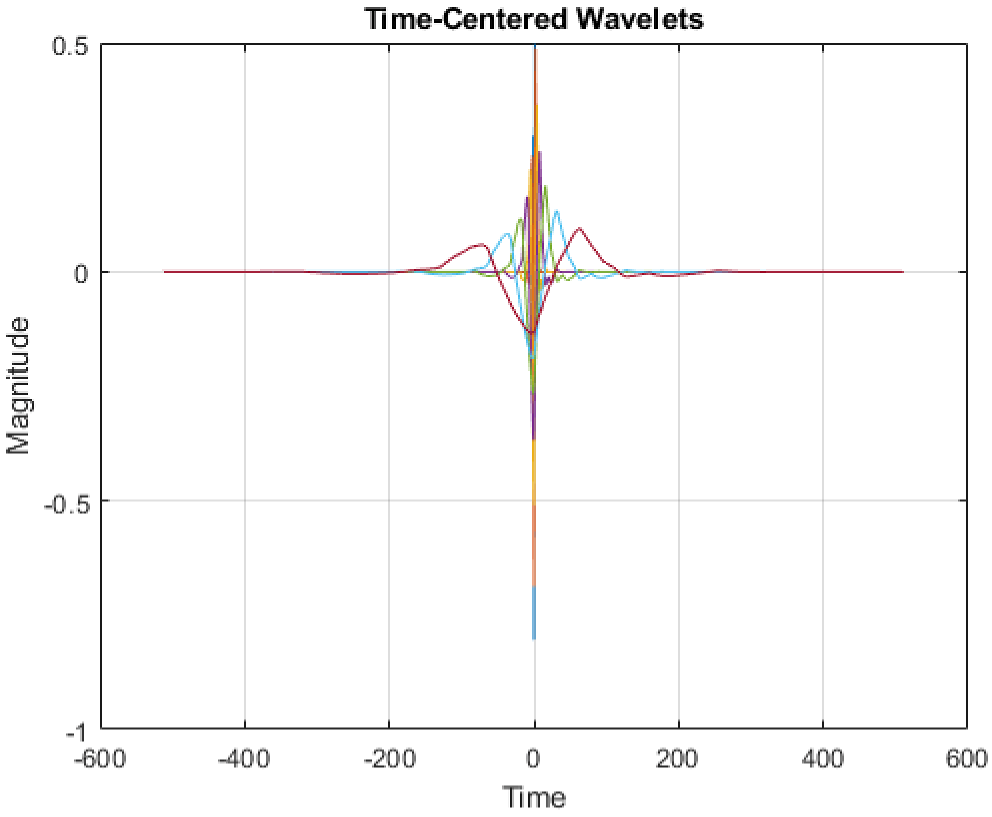
```
fb = 
  dwtfilterbank with properties:

                Wavelet: 'sym4'
           SignalLength: 1024
                  Level: 7
      SamplingFrequency: 1
             FilterType: 'Analysis'
    CustomWaveletFilter: []
    CustomScalingFilter: []
```

Plot the magnitude frequency responses of the wavelets and coarsest-scale scaling function. Open the plot in a separate figure window. The plot legend in the window is interactive. To hide a particular frequency response, click on its name.

```
freqz(fb)
```

Obtain and plot the time-centered wavelets corresponding to the wavelet bandpass filters.

```
[psi,t] = wavelets(fb);
plot(t,psi')
grid on
title('Time-Centered Wavelets')
xlabel('Time')
ylabel('Magnitude')
```



Create DWT Filter Bank Using Custom Filters

This example shows how to create a DWT filter bank using custom biorthogonal wavelet filters.

Two pairs of analysis (decomposition) and synthesis (reconstruction) filters are associated with a biorthogonal wavelet. Each pair consists of a lowpass and highpass filter. Specify the analysis and synthesis filters for the nearly-orthogonal biorthogonal wavelets based on the Laplacian pyramid scheme of Burt and Adelson (Table 8.4 on page 283 in [1]). Because the toolbox requires that all filters associated with a biorthogonal wavelet or an orthogonal wavelet have the same even length, the filters are prepended and appended with 0s.

```
Hd = [0 -1 5 12 5 -1 0 0]/20*sqrt(2);
Gd = [0 3 -15 -73 170 -73 -15 3]/280*sqrt(2);
Hr = [0 -3 -15 73 170 73 -15 -3]/280*sqrt(2);
Gr = [0 -1 -5 12 -5 -1 0 0]/20*sqrt(2);
```

`Hd` and `Gd` are the lowpass and highpass decomposition filters, respectively. `Hr` and `Gr` are the lowpass and highpass reconstruction filters, respectively.

Construct analysis and synthesis DWT filter banks using the biorthogonal filters. Confirm the filter banks are biorthogonal and not orthogonal.

```
fbAna = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',[Hd' Hr'],'CustomWaveletFilter',[Gd' Gr']);
```

```
fbSyn = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',[Hd' Hr'],'CustomWaveletFilter',[Gd' Gr'],...
    'FilterType','Synthesis');
fprintf('fbAna: isOrthogonal = %d\tisBiorthogonal = %d\n',...
    isOrthogonal(fbAna),isBiorthogonal(fbAna));
```

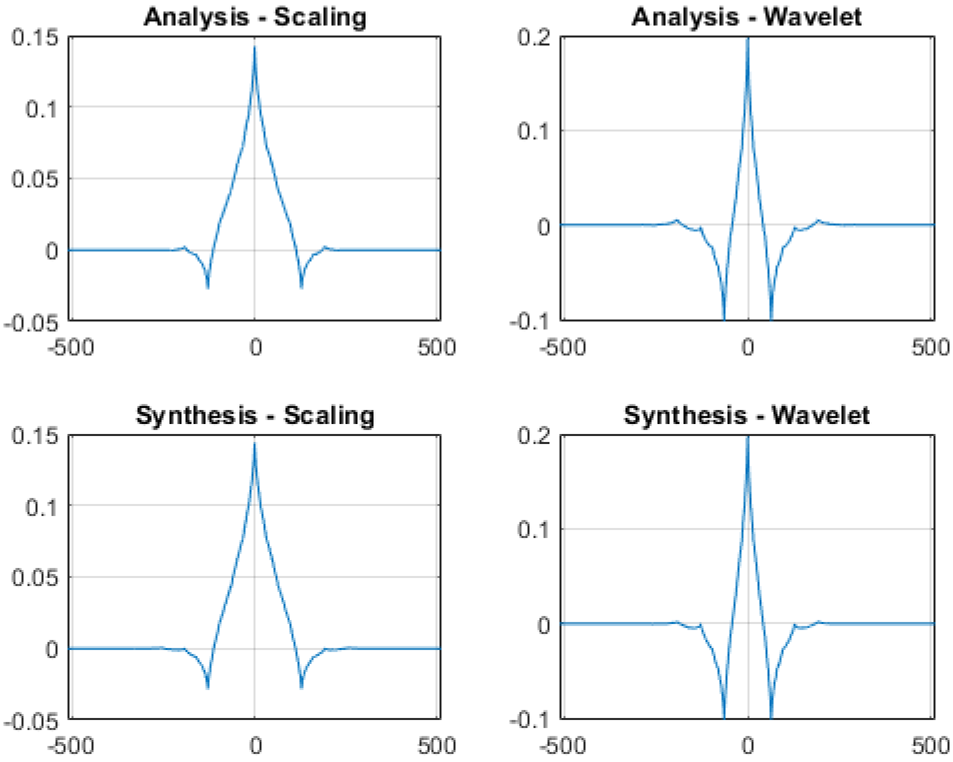
```
fbAna: isOrthogonal = 0    isBiorthogonal = 1
```

```
fprintf('fbSyn: isOrthogonal = %d\tisBiorthogonal = %d\n',...
    isOrthogonal(fbSyn),isBiorthogonal(fbSyn ));
```

```
fbSyn: isOrthogonal = 0    isBiorthogonal = 1
```

Obtain the wavelet and scaling functions of both filter banks. Plot the wavelet and scaling functions at the coarsest scales.

```
[fbAna_phi,t] = scalingfunctions(fbAna);
[fbAna_psi,~] = wavelets(fbAna);
[fbSyn_phi,~] = scalingfunctions(fbSyn);
[fbSyn_psi,~] = wavelets(fbSyn);
subplot(2,2,1)
plot(t,fbAna_phi(end,:))
grid on
title('Analysis - Scaling')
subplot(2,2,2)
plot(t,fbAna_psi(end,:))
grid on
title('Analysis - Wavelet')
subplot(2,2,3)
plot(t,fbSyn_phi(end,:))
grid on
title('Synthesis - Scaling')
subplot(2,2,4)
plot(t,fbSyn_psi(end,:))
grid on
title('Synthesis - Wavelet')
```



Compute the framebounds of the two filter banks. Since the filters are associated with biorthogonal wavelets, the framebounds will not equal 1.

```
[a1,a2] = framebounds(fbAna)
```

```
a1 = 0.9505
```

```
a2 = 1.0211
```

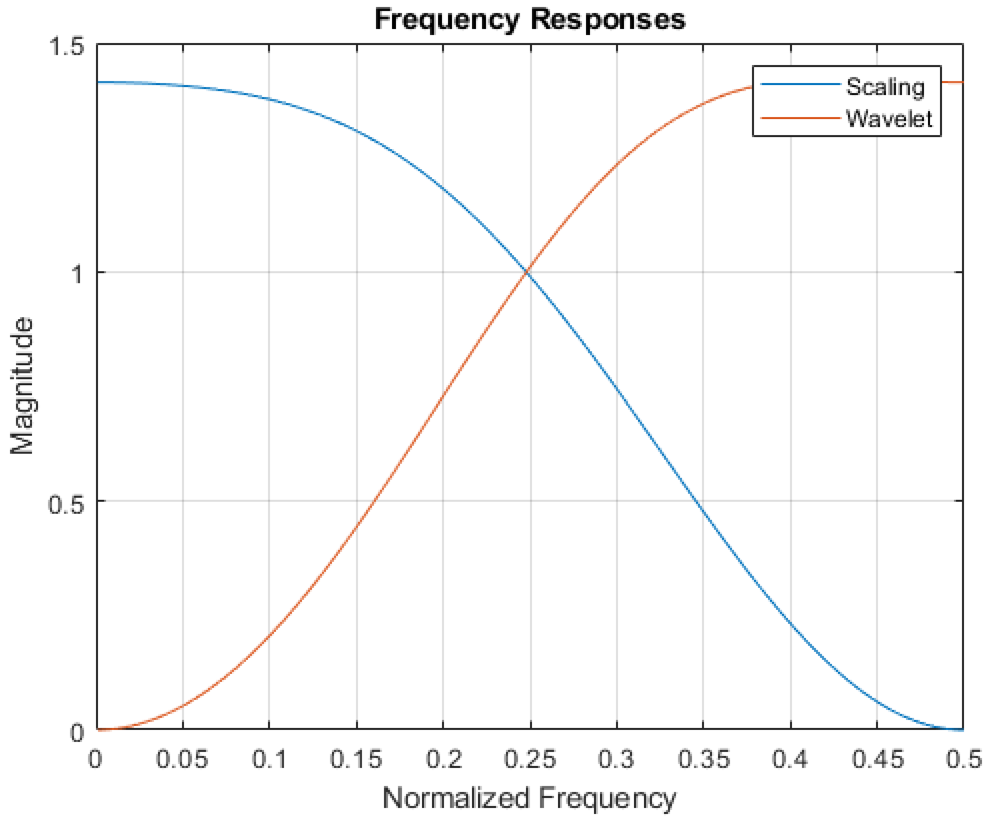
```
[b1,b2] = framebounds(fbSyn)
```

```
b1 = 0.9800
```

```
b2 = 1.0528
```

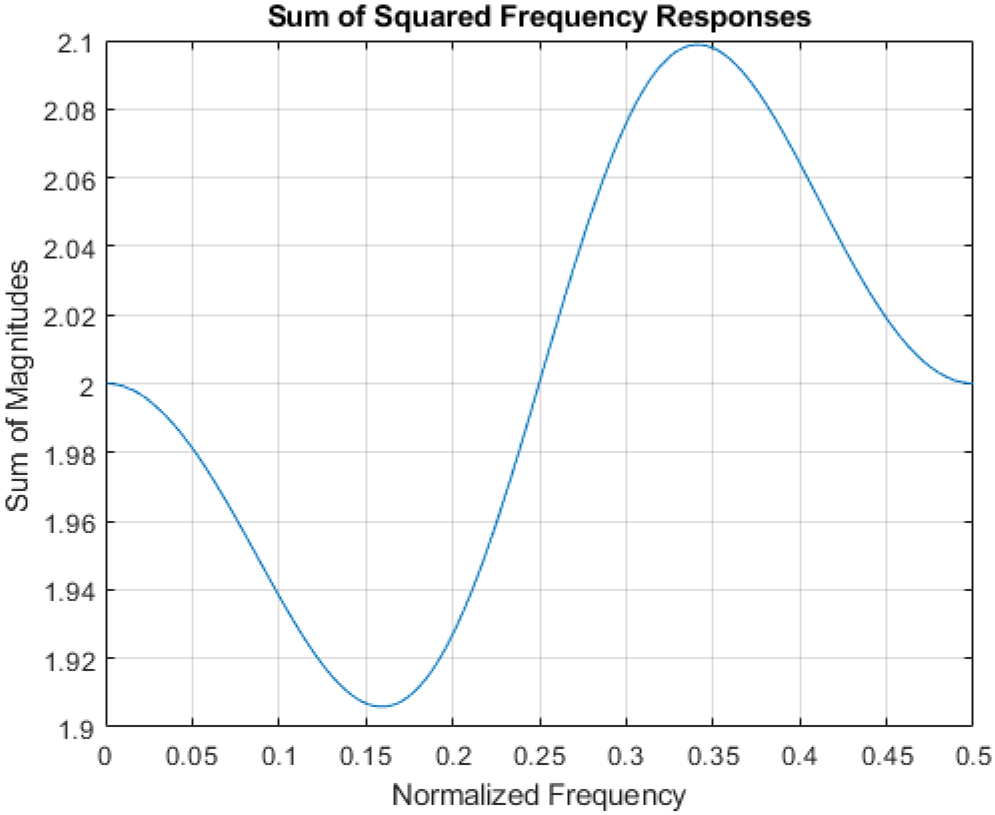
Obtain the frequency responses of the scaling and wavelets filters in the analysis filter bank. Plot up to Nyquist the magnitude frequency responses of the scaling and wavelet filters at the finest scale.

```
[psidft,f,phidft] = freqz(fbAna);
flen = length(f);
figure
plot(f(flen/2+1:end),abs(phidft(1,flen/2+1:end)))
hold on
plot(f(flen/2+1:end),abs(psidft(1,flen/2+1:end)))
grid on
legend('Scaling','Wavelet')
title('Frequency Responses')
xlabel('Normalized Frequency')
ylabel('Magnitude')
```



Zoom in and confirm the magnitude frequency responses at the point of intersection are not magnitude equal to 1. Plot the sum of the squared magnitudes of the frequency responses. Because the scaling (lowpass) and wavelet (highpass) filters do not form an orthogonal quadrature mirror filter pair, the sum does not equal to 2 at all frequencies.

```
figure
plot(f(flen/2+1:end),abs(phidft(1,flen/2+1:end)).^2 + abs(psidft(1,flen/2+1:end)).^2)
grid on
title('Sum of Squared Frequency Responses')
xlabel('Normalized Frequency')
ylabel('Sum of Magnitudes')
```



References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

See Also

`wavemngr` | `dwt` | `wavedec` | `modwt`

Topics

"Add Quadrature Mirror and Biorthogonal Wavelet Filters"

Introduced in R2018a

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.

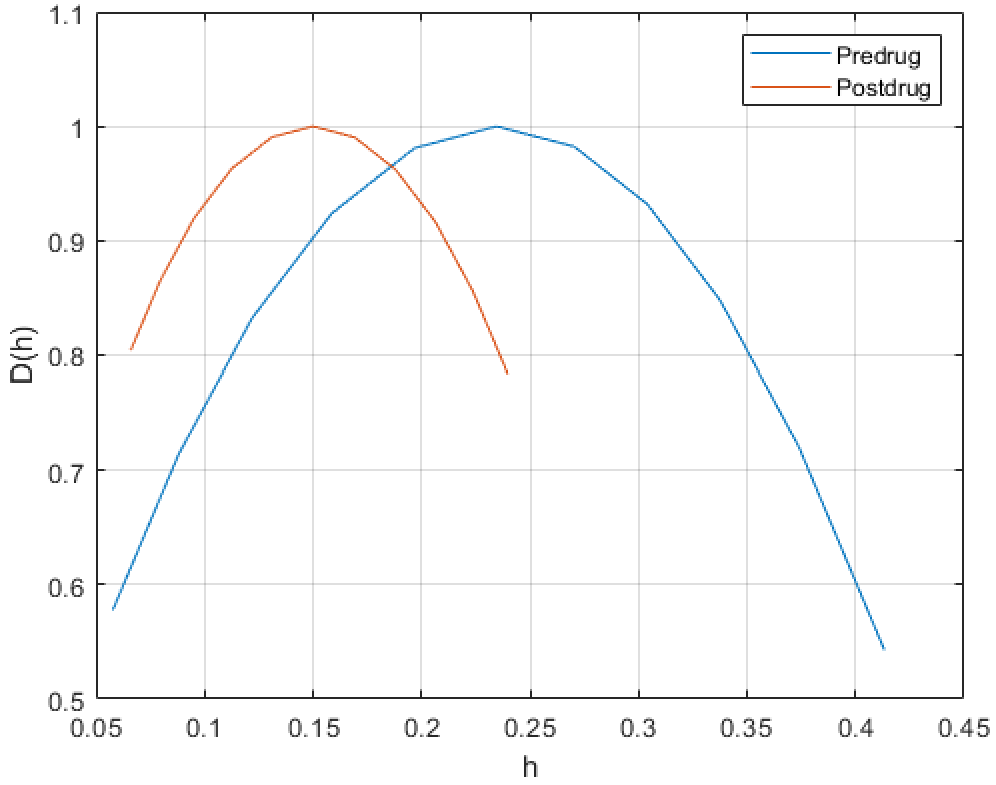
Examples

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

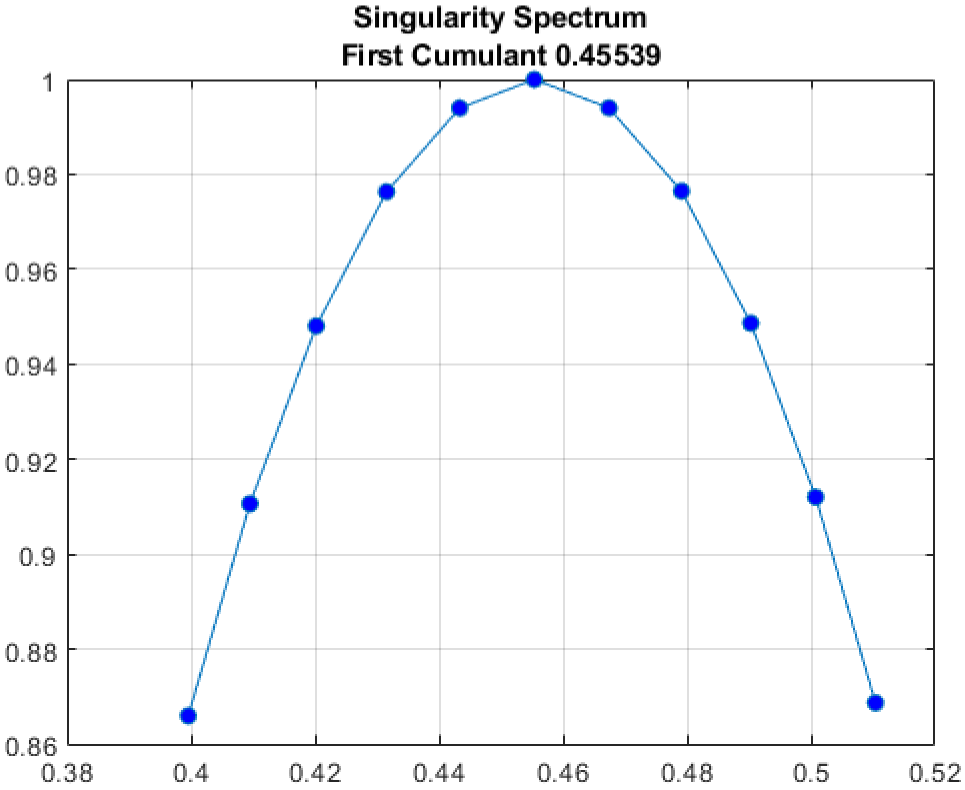
Compute the singularity spectrum and cumulants for a Brownian noise process.

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 = 1×3

    0.4554   -0.0121   -0.0000
```

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.

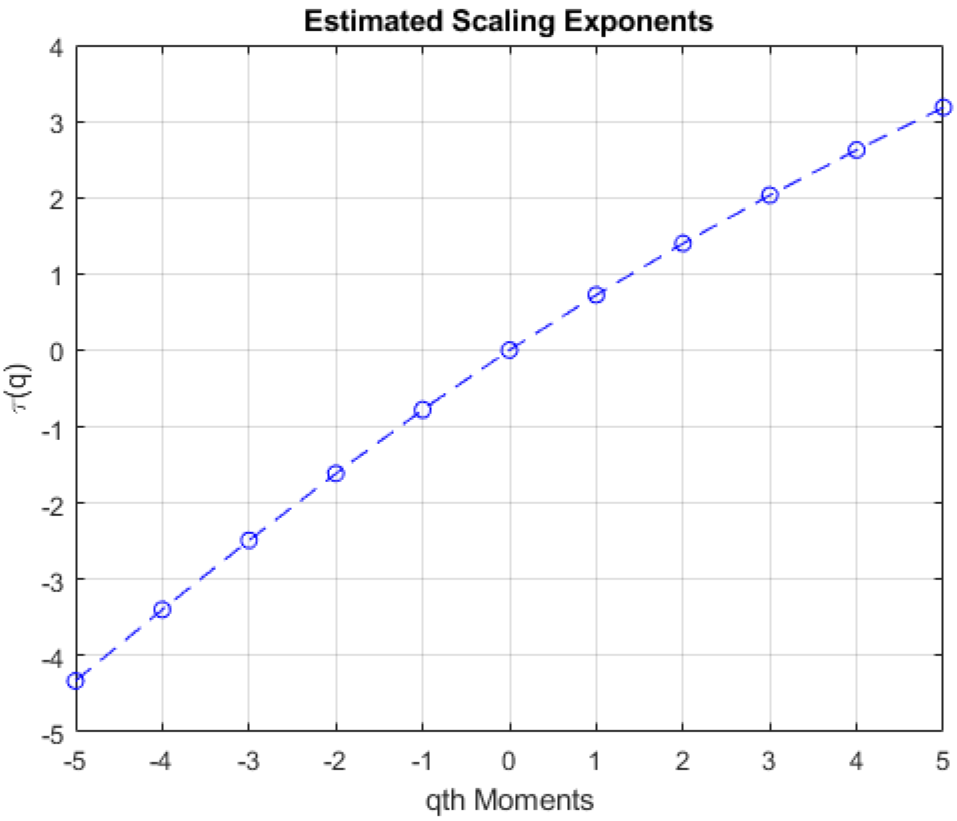
```
[~,~,cp,tauq] = dwtleader(mrw07505);
cp([1 2])
```

```
ans = 1×2

    0.7504   -0.0554
```

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.

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 | string scalar

Wavelet name, specified as a character vector or string scalar. `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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

RegressionWeight — Weight option

`'uniform'` (default) | `'scale'`

Weight option to use in the weighted least-squares regression model to determine the singularity spectrum, Holder exponents, cumulants, and scaling exponents, specified as the comma-separated pair consisting of `'RegressionWeight'` and either `'uniform'` or `'scale'`. The `'uniform'` option applies equal weight to each scale. The `'scale'` option uses the number of wavelet leaders by scale as weights.

Note To duplicate the behavior of `dwtleader` found in releases prior to R2018a, update all instances of `dwtleader` to include the name-value pair argument `'RegressionWeight'` set to `'scale'`.

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 *i*th 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

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

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 *q*th 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.

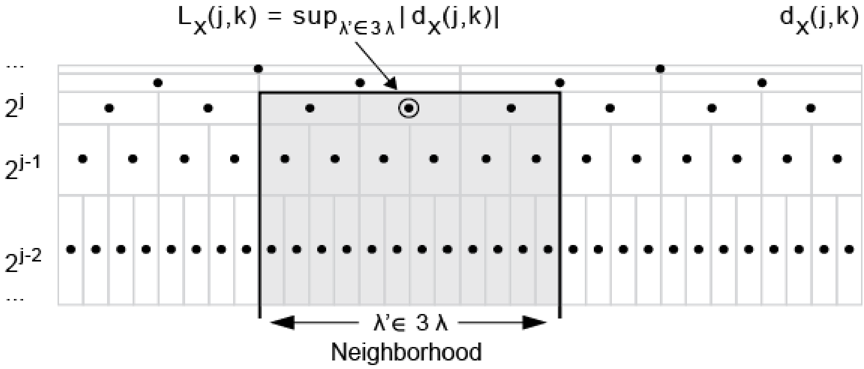
Algorithms

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.

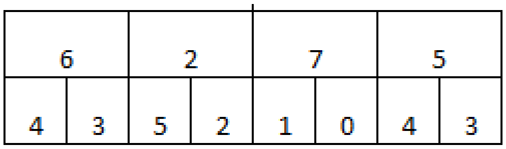


To calculate the wavelet leaders, $L_x(j,k)$:

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.

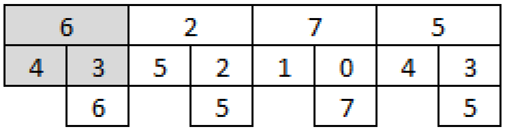
   $$[2^j k, 2^j(k+1)) = [2^{j-1}(2k), 2^{j-1}(2k+2))$$
   $$[2^{j-1}(2k), 2^{j-1}(2k+2)) = [2^{j-1}(2k), 2^{j-1}(2k+1)) \cup [2^{j-1}(2k+1), 2^{j-1}(2k+2))$$
2. Start at the scale that is one level coarser than the finest obtained scale.
3. Compare the first value to all its finer dyadic intervals and 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:



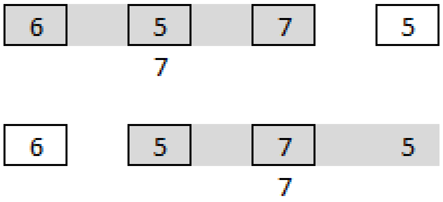
6		2		7		5	
4	3	5	2	1	0	4	3

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.



6		2		7		5	
4	3	5	2	1	0	4	3
	6		5		7		5

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

`wtmm` | `wfbm`

Topics

“Multifractal Analysis”

Introduced in R2016b

dwtmode

Discrete wavelet transform extension mode

Syntax

```
dwtmode(mode)
```

```
dwtmode
dwtmode('status')
st = dwtmode
st = dwtmode('status')
st = dwtmode('status','nodisp')
```

```
dwtmode('save',mode)
dwtmode('save')
dwtmode('save',CURRENTMODE)
```

Description

`dwtmode(mode)` sets the signal or image extension mode for both discrete wavelet and wavelet packet transforms to `mode`. All functions and Wavelet Analyzer app tools involving either the discrete wavelet transform (1-D and 2-D) or wavelet packet transform (1-D and 2-D), use the specified DWT extension mode.

The extension modes provide options for dealing with the problem of border distortion in signal or image analysis. For more information, see “Border Effects”.

Note Functions involving the discrete wavelet transform may not use the current extension mode for `gpuArray` input. Such cases are documented on the function reference page.

`dwtmode` or `dwtmode('status')` display the current mode. If `DWTMODE.DEF` exists in the current path, the default mode is loaded from `DWTMODE.DEF` at the start of the MATLAB session. Otherwise, the file `DWTMODE.CFG` is used.

`st = dwtmode` or `st = dwtmode('status')` display and return the current mode in `st`.

`st = dwtmode('status','nodisp')` returns the current mode `st` and no status or warning text is displayed in the MATLAB command window.

`dwtmode('save',mode)` saves `mode` as the new default mode to the file `DWTMODE.DEF` in the current folder. If `DWTMODE.DEF` already exists in the current folder, the file is overwritten. The new default mode will be active as the default mode in the next MATLAB session.

Note To execute in parallel any functionality that depends on the extension mode, either save the extension mode using `dwtmode('save',mode)` before running your `parfor` loop, or call `dwtmode(mode)` inside your `parfor` loop.

Changing the extension mode in a MATLAB session does not have the desired effect if anything dependent on that mode is called in parallel. In a parallel environment, each worker has its own

MATLAB execution engine, and each worker respects the DWTMODE.CFG file, but not an override in the current session. Therefore, to run in parallel, the extension mode must either be saved to the current folder, or the extension mode must be set for each worker.

Executing for-loop iterations in parallel requires Parallel Computing Toolbox™. For more information, see parfor.

dwtmode('save') is equivalent to dwtmode('save',CURRENTMODE), where CURRENTMODE represents the current extension mode.

Examples

Display and Change Signal Extension Mode

Display the current DWT signal extension mode. If the DWT extension mode global variable does not exist, the default is half-point symmetrization.

```
dwtmode
```

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

Save the current extension mode. Change the extension mode to periodized extension.

```
origmode = dwtmode('status','nodisplay');
dwtmode('per','nodisplay')
```

Display the current DWT signal extension mode.

```
dwtmode
```

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

Restore the original extension mode.

```
dwtmode(origmode,'nodisplay')
dwtmode
```

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

Input Arguments

`mode` — Discrete wavelet transform extension mode

`'zpd'` | `'sp0'` | `'spd'` | ...

DWT extension mode used to extend the input, specified as one of the following values.

`mode`	DWT Extension Mode
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1
`'sym'` or `'symh'`	Symmetric extension (half point): boundary value symmetric replication
`'symw'`	Symmetric extension (whole point): boundary value symmetric replication
`'asym'` or `'asymh'`	Antisymmetric extension (half point): boundary value antisymmetric replication
`'asymw'`	Antisymmetric extension (whole point): boundary value antisymmetric replication
`'ppd'`, `'per'`	Periodized extension If the signal length is odd and `mode` is `'per'`, an extra sample equal to the last value is added to the right and the extension is performed in `'ppd'` mode. If the signal length is even, `'per'` is equivalent to `'ppd'`. This rule also applies to images.

The DWT associated with the symmetric, smooth, zero, and periodic extension modes are slightly redundant. But the inverse DWT ensures a perfect reconstruction for the extensions mentioned.

Note `dwtmode` updates a global variable. Only use `dwtmode` to change the extension mode. Avoid changing the global variable directly.

Output Arguments

`st` — DWT extension mode

character array

DWT extension mode, returned as a character array.

Tips

- For most wavelet applications, either a periodic extension or symmetric extension works fine.

References

[1] Strang, G., and T. 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

dwtpassbands

DWT filter bank passbands

Syntax

```
dwtbands = dwtpassbands(fb)
```

Description

`dwtbands = dwtpassbands(fb)` returns the theoretical discrete wavelet transform (DWT) passbands for the DWT filter bank `fb`.

Examples

DWT Filter Bank Passbands

Obtain the theoretical DWT passbands for a four-level wavelet transform using the Daubechies `db6` wavelet with a sampling frequency of 1 kHz.

```
wv = 'db6';
Fs = 1e3;
fb = dwtfilterbank('Wavelet',wv,'Level',4,'SamplingFrequency',Fs);
dwtpassbands(fb)
```

```
ans = 5×2

  250.0000  500.0000
  125.0000  250.0000
   62.5000  125.0000
   31.2500   62.5000
         0   31.2500
```

Obtain the power bandwidths for the filter bank. Compare the theoretical passbands with the measured wavelet 3 dB bandwidths at all four levels.

```
ptable = powerbw(fb);
ptable(:,1:3)
```

```
ans=4×3 table
    Level       DWTBand         Wavelet3dBBandwidth
    _____    ______________    ____________________

      1        250      500        250         500
      2        125      250      123.2      253.71
      3       62.5      125     61.601      126.78
      4      31.25     62.5     30.815      63.389
```

Input Arguments

`fb` — Discrete wavelet transform filter bank

`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

Output Arguments

`dwtbands` — Theoretical DWT passbands

real-valued matrix

Theoretical DWT passbands for the filter bank `fb`, returned as an L+1-by-2 real-valued matrix, where L is the wavelet transform level of the filter bank.

- The first L rows of `dwtbands` contain the theoretical passband frequencies for the DWT listed in order of decreasing resolution (increasing scale).
- The final row of `dwtbands` contains the theoretical passband for the coarsest resolution scaling filter.
- The first column of `dwtbands` contains the lower frequency limit.
- The final row of `dwtbands` contains the theoretical passband for the coarsest resolution scaling filter.

See Also

`powerbw` | `dwtfilterbank`

Introduced in R2018a

dyaddown

Dyadic downsampling

Syntax

```
Y = dyaddown(X)
Y = dyaddown(X,EVENODD)
Y = dyaddown( ___ ,'type')
```

Description

Y = dyaddown(X) downsamples even-indexed elements of X. Y contains even-index samples of X in this case. Specify X as a vector or matrix. When you specify X as a vector, the function returns a version of X downsampled by 2.

Y = dyaddown(X,EVENODD) downsamples even- or odd-indexed elements of X. Y can contain even- or odd-indexed samples of X depends on the value of EVENODD. Specify X as a vector. When you specify X as a vector, the function returns a version of X downsampled by 2.

Y = dyaddown(___ ,'type') returns a version of X obtained by suppressing columns or rows, or rows and columns of X using 'type' argument. Specify X as a matrix.

Examples

Perform Dyadic Downsampling

Create a vector of data that you want to downsample.

```
X1 = 1:10
```

```
X1 = 1×10

     1     2     3     4     5     6     7     8     9    10
```

Downsample elements with even indices.

```
dse = dyaddown(X1)
```

```
dse = 1×5

     2     4     6     8    10
```

You can also downsample the elements in X1 by setting EVENODD to 0.

```
dse2 = dyaddown(X1,0)
```

```
dse2 = 1×5

     2     4     6     8    10
```

Downsample elements with odd indices.

```
dso = dyaddown(X1,1)
```

```
dso = 1×5

     1     3     5     7     9
```

Create a matrix data that you want to downsample.

```
X = (1:3)'*(1:4)
```

```
X = 3×4

     1     2     3     4
     2     4     6     8
     3     6     9    12
```

Downsample columns with even indices.

```
dec = dyaddown(X,0,'c')
```

```
dec = 3×2

     2     4
     4     8
     6    12
```

Downsample rows with odd indices.

```
der = dyaddown(X,1,'r')
```

```
der = 2×4

     1     2     3     4
     3     6     9    12
```

Downsample rows and columns with odd indices.

```
dem = dyaddown(X,1,'m')
```

```
dem = 2×2

     1     3
     3     9
```

Input Arguments

X — Data to be downsampled

vector | matrix

Data to be downsampled, specified as a vector or matrix. X is a vector when you do not use the 'type' argument in the dyaddown function and X is a matrix when you use the 'type' argument in the dyaddown function.

EVENODD — Even- or odd-indexed elements of X
0 (default) | positive integer

Even- or odd-indexed elements of X, specified as a positive integer.

Y contains the even- or odd-indexed samples of X depending on the value of EVENODD:

- If EVENODD is even, then Y(k) = X(2k).
- If EVENODD is odd, then Y(k) = X(2k+1).

Example: dyaddown(X,0) consists of even-indexed samples.

'type' — Type of downsampling
'c' (default) | 'r' | 'm'

Type of downsampling , specified as one of the following:

- 'c' to downsample columns of X
- 'r' to downsample rows of X
- 'm' to downsample rows and columns of X

Output Arguments

Y — Dyadic downsampled version of X
vector | matrix

Dyadic downsampled version of X, returned as a vector or a matrix.

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also
dyadup

Introduced before R2006a

dyaddown

Dyadic downsampling of Laurent polynomial or Laurent matrix

Syntax

```
Q = dyaddown(P)
```

Description

`Q = dyaddown(P)` downsamples by two the Laurent polynomial or Laurent matrix specified by `P`. If `P` is a Laurent matrix, `dyaddown` downsamples the matrix elements.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `dyaddown`. The input data type determines which version is executed.

Examples

Dyadic Downsampling of Laurent Polynomial

Create the Laurent polynomial $a(z) = \sum_{k = -5}^{6} (-1)^k k z^k$. Obtain the degree of $a(z)$.

```
cfs = (-1).^(-5:6).*(-5:6);
a = laurentPolynomial(Coefficients=fliplr(cfs),MaxOrder=6)
```

```
a = 
  laurentPolynomial with properties:

    Coefficients: [6 -5 4 -3 2 -1 0 1 -2 3 -4 5]
        MaxOrder: 6
```

```
degree(a)
```

```
ans = 11
```

Obtain the degree of the dyadic downsampling of $a(z)$.

```
ddown = dyaddown(a)
```

```
ddown = 
  laurentPolynomial with properties:

    Coefficients: [6 4 2 0 -2 -4]
        MaxOrder: 3
```

```
degree(ddown)
```

```
ans = 5
```

Dyadic Downsampling of Laurent Matrix

Create two Laurent polynomials:

- $a(z) = \sum_{k = 0}^{5} (6 - k)z^{6 - k}$
- $b(z) = \sum_{k = 0}^{5} (k + 1)z^{-k}$

```
lpA = laurentPolynomial(Coefficients=[6:-1:1],MaxOrder=6);
lpB = laurentPolynomial(Coefficients=[1:6],MaxOrder=0);
```

Create the Laurent matrix `matA` $= \begin{bmatrix} a(z) & 1 \\ 2 & b(z) \end{bmatrix}$.

```
matA = laurentMatrix(Elements={lpA,1;2,lpB});
```

Obtain the dyadic downsampling of `matA`.

```
matB = dyaddown(matA);
```

Inspect the elements of `matB`.

```
matB.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [6 4 2]
        MaxOrder: 3
```

```
matB.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

```
matB.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 2
        MaxOrder: 0
```

```
matB.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:
    Coefficients: [1 3 5]
        MaxOrder: 0
```

Input Arguments

P — Laurent polynomial or Laurent matrix
laurentPolynomial object | laurentMatrix object

Laurent polynomial or Laurent matrix, specified as a laurentPolynomial object or a laurentMatrix object, respectively.

Output Arguments

Q — Downsampled Laurent polynomial or Laurent matrix
laurentPolynomial object | laurentMatrix object

Downsampled Laurent polynomial or Laurent matrix, returned as a laurentPolynomial object or a laurentMatrix object. Downsampling a Laurent polynomial $P(z) = \sum_{k = -\infty}^{\infty} C_k z^k$ by two results in the polynomial $Q(z) = \sum_{k = -\infty}^{\infty} C_{2k} z^k$.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
dyadup | polyphase | reflect

Objects
laurentMatrix | laurentPolynomial

Introduced in R2021b

dyadup

Dyadic upsampling

Syntax

```
Y = dyadup(X)
Y = dyadup(X,EVENODD)
Y = dyadup( ___ ,'type')
```

Description

`Y = dyadup(X)` upsamples odd-indexed elements of `X`. `Y` contains odd-index samples of `X` in this case. Specify `X` as a vector or matrix. When you specify `X` as a vector, the function returns an extended copy of vector `X` upsampled by inserting zeros.

`Y = dyadup(X,EVENODD)`, where `X` upsamples even- or odd-indexed elements of `X`. `Y` can contain even- or odd-indexed samples of `X` depends on the value of `EVENODD`. Specify `X` as a vector. When you specify `X` as a vector, the function returns an extended copy of vector `X` obtained by inserting zeros.

`dyadup` implements a simple zero-padding scheme very useful in the wavelet reconstruction algorithm.

`Y = dyadup( ___ ,'type')` returns a extended copies of `X` obtained by inserting columns or rows, or rows and columns of `X` using `'type'` argument. Specify `X` as a matrix.

Examples

Perform Dyadic Upsampling

Create a vector of data that you want to upsample.

```
s = 1:5
```

```
s = 1×5

     1     2     3     4     5
```

Upsample elements at odd indices.

```
dse = dyadup(s)
```

```
dse = 1×11

     0     1     0     2     0     3     0     4     0     5     0
```

You can also upsample the elements in X1 by setting `EVENODD` to `1`.

```
dse1 = dyadup(s,1)
```

```
dse1 = 1×11

     0     1     0     2     0     3     0     4     0     5     0
```

Upsample elements at even indices.

```
dso = dyadup(s,0)
```

```
dso = 1×9

     1     0     2     0     3     0     4     0     5
```

Create a matrix data that you want to upsample.

```
s = (1:2)'*(1:3)
```

```
s = 2×3

     1     2     3
     2     4     6
```

Upsample rows at even indices.

```
der = dyadup(s,1,'r')
```

```
der = 5×3

     0     0     0
     1     2     3
     0     0     0
     2     4     6
     0     0     0
```

Upsample columns at odd indices.

```
doc = dyadup(s,0,'c')
```

```
doc = 2×5

     1     0     2     0     3
     2     0     4     0     6
```

Upsample rows and columns at even indices.

```
dem = dyadup(s,1,'m')
```

```
dem = 5×7

     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 = 1×5

     1     2     3     4     5
```

```
uds = dyaddown(dyadup(s))
```

```
uds = 1×5

     1     2     3     4     5
```

In general reversed identity is false.

Input Arguments

X — Data to be upsampled

vector | matrix

Data to be upsampled, specified as a vector or matrix. `X` is a vector when you do not use the `'type'` argument in the `dyadup` function and `X` is a matrix when you use the `'type'` argument in the `dyadup` function.

EVENODD — Even- or odd-indexed samples of X

`1` (default) | positive integer

Even- or odd-indexed samples of `X`, specified as a positive integer.

`Y` contains the even- or odd-indexed samples of `X` depends on the value of `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)`.

`dyadup` defaults to `EVENODD = 1` (zeros in odd-indexed positions).

'type' — Type of upsampling

`'c'` (default) | `'r'` | `'m'`

Type of upsampling , specified as one of the following:

- `'c'` to upsample columns of `X`
- `'r'` to upsample rows of `X`
- `'m'` to upsample rows and columns of `X`

Output Arguments

Y — Dyadic upsampled version of X

vector | matrix

Dyadic upsampled version of `X`, returned as a vector or a matrix.

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

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).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`dyaddown`

Introduced before R2006a

dyadup

Dyadic upsampling of Laurent polynomial or Laurent matrix

Syntax

```
Q = dyadup(P)
```

Description

`Q = dyadup(P)` upsamples by two the Laurent polynomial or Laurent matrix specified by `P`. If `P` is a Laurent matrix, `dyadup` upsamples the matrix elements. If `P` is a Laurent matrix, `dyadup` upsamples the matrix elements.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `dyadup`. The input data type determines which version is executed.

Examples

Dyadic Upsampling of Laurent Polynomial

Create the Laurent polynomial $a(z) = \sum_{k = -5}^{6} (-1)^k k z^k$. Obtain the degree of $a(z)$.

```
cfs = (-1).^(-5:6).*(-5:6);
a = laurentPolynomial(Coefficients=fliplr(cfs),MaxOrder=6)
```

```
a =
  laurentPolynomial with properties:

    Coefficients: [6 -5 4 -3 2 -1 0 1 -2 3 -4 5]
        MaxOrder: 6
```

```
degree(a)
```

```
ans = 11
```

Obtain the degree of the dyadic upsampling of $a(z)$.

```
dup = dyadup(a)
```

```
dup =
  laurentPolynomial with properties:

    Coefficients: [6 0 -5 0 4 0 -3 0 2 0 -1 0 0 0 1 0 -2 0 3 0 -4 0 5]
        MaxOrder: 12
```

```
degree(dup)
```

```
ans = 22
```

Dyadic Upsampling of Laurent Matrix

Create two Laurent polynomials:

- $a(z) = 2 + 4z^{-1} + 6z^{-2}$
- $b(z) = z + 3 + 5z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[2 4 6],MaxOrder=0);
lpB = laurentPolynomial(Coefficients=[1 3 5],MaxOrder=1);
```

Create the Laurent matrix `matA` $= \begin{bmatrix} a(z) & 2 \\ 3 & b(z) \end{bmatrix}$.

```
matA = laurentMatrix(Elements={lpA,2;3,lpB});
```

Obtain the dyadic upsampling of `matA`.

```
matB = dyadup(matA);
```

Inspect the elements of `matB`.

```
matB.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [2 0 4 0 6]
        MaxOrder: 0
```

```
matB.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 2
        MaxOrder: 0
```

```
matB.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 3
        MaxOrder: 0
```

```
matB.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 0 3 0 5]
```

```
        MaxOrder: 2
```

Input Arguments

P — Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Output Arguments

Q — Upsampled Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Upsampled Laurent polynomial or Laurent matrix, returned as a `laurentPolynomial` object or a `laurentMatrix` object . Upsampling a Laurent polynomial $P(z)$ by two results in the polynomial $Q(z) = P(z^2)$.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dyaddown` | `polyphase` | `reflect`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

editLabelDefinition

Edit label definition properties

Syntax

```
editLabelDefinition(lss,lblname,propname,val)
```

Description

`editLabelDefinition(lss,lblname,propname,val)` changes the `propname` property of the label or sublabel definition `lblname` to `val`.

The function can edit only the “Name” on page 1-0 , “DefaultValue” on page 1-0 , “Tag” on page 1-0 , “Description” on page 1-0 , and “Categories” on page 1-0 properties. To change any other property of the label definition, remove the definition using `removeLabelDefinition` and add a definition with the desired property values using `addLabelDefinitions`.

- If you edit the “DefaultValue” on page 1-0 property, all existing label values remain unchanged. The new default value applies only to new members, new regions, or new points.
- You can edit the “Categories” on page 1-0 property only when the “LabelDataType” on page 1-0 of the target label or sublabel definition is `'Categorical'`.

New specified categories do not replace any existing categories. They are appended to the existing values.

Examples

Edit Label Definition

Load a labeled signal set containing recordings of whale songs. Get the names of the labels.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

```
getLabelNames(lss)
```

```
ans = 3x1 string
    "WhaleType"
    "MoanRegions"
    "TrillRegions"
```

The first label corresponds to the type of whale. Get the types available in the set.

```
lbldefs = getLabelDefinitions(lss);
types = lbldefs(1)
```

```
types = 
  signalLabelDefinition with properties:

             Name: "WhaleType"
        LabelType: "attribute"
    LabelDataType: "categorical"
       Categories: [3x1 string]
     DefaultValue: []
        Sublabels: [0x0 signalLabelDefinition]
              Tag: ""
      Description: "Whale type"

 Use labeledSignalSet to create a labeled signal set.
```

```
types = types.Categories
```

```
types = 3x1 string
    "blue"
    "humpback"
    "white"
```

Modify the label to incorporate sperm whales and killer whales. Verify that the labeled signal set includes the two new whale types.

```
editLabelDefinition(lss,'WhaleType', ...
    'Categories',{'sperm','killer'})

lbldefs = getLabelDefinitions(lss);
types = lbldefs(1).Categories
```

```
types = 5x1 string
    "blue"
    "humpback"
    "white"
    "sperm"
    "killer"
```

The definition for trill regions has a sublabel that identifies peaks.

```
lbldefs(3).Sublabels
```

```
ans = 
  signalLabelDefinition with properties:

                     Name: "TrillPeaks"
                LabelType: "point"
```

```
             LabelDataType: "numeric"
        ValidationFunction: []
    PointLocationsDataType: "double"
              DefaultValue: []
                 Sublabels: [0x0 signalLabelDefinition]
                       Tag: ""
               Description: "Trill peaks"

 Use labeledSignalSet to create a labeled signal set.
```

Change the description of the sublabel.

```
editLabelDefinition(lss,["TrillRegions" "TrillPeaks"],'Description','Peaks of trill regions')

lbldefs = getLabelDefinitions(lss);
lbldefs(3).Sublabels
```

```
ans = 
  signalLabelDefinition with properties:

                      Name: "TrillPeaks"
                 LabelType: "point"
             LabelDataType: "numeric"
        ValidationFunction: []
    PointLocationsDataType: "double"
              DefaultValue: []
                 Sublabels: [0x0 signalLabelDefinition]
                       Tag: ""
               Description: "Peaks of trill regions"

 Use labeledSignalSet to create a labeled signal set.
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`propname` — Property name
`'Name'` | `'DefaultValue'` | `'Tag'` | `'Description'` | `'Categories'`

Property name, specified as `'Name'`, `'DefaultValue'`, `'Tag'`, `'Description'`, or `'Categories'`.

Data Types: `char` | `string`

`val` — Property value
numeric value | logical value | character vector | string | vector of strings | cell array of character vectors

Label values, specified as a numeric or logical value, a character vector or string, a vector of strings, or a cell array of character vectors. `val` must be of the data type specified for `propname`.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

emd

Empirical mode decomposition

Syntax

```
[imf,residual] = emd(x)
[imf,residual,info] = emd(x)
[___] = emd(___,Name,Value)
```

```
emd(___)
```

Description

`[imf,residual] = emd(x)` returns intrinsic mode functions `imf` and residual signal `residual` corresponding to the empirical mode decomposition of `x`. Use `emd` to decompose and simplify complicated signals into a finite number of intrinsic mode functions required to perform Hilbert spectral analysis.

`[imf,residual,info] = emd(x)` returns additional information `info` on IMFs and residual signal for diagnostic purposes.

`[___] = emd(___,Name,Value)` performs the empirical mode decomposition with additional options specified by one or more `Name,Value` pair arguments.

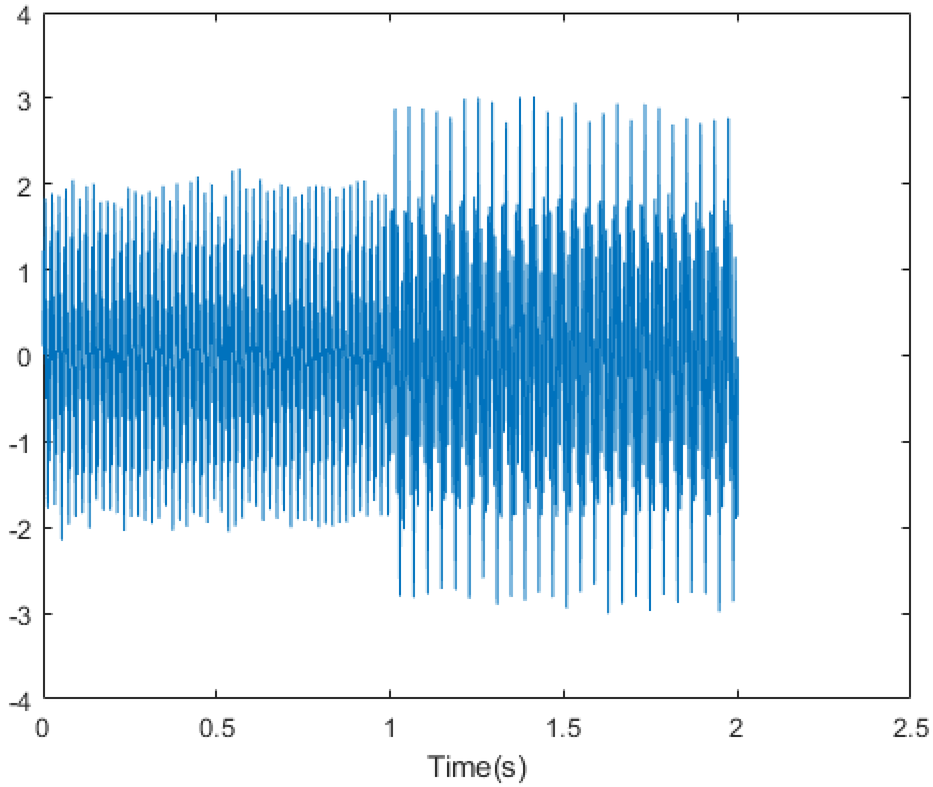
`emd(___)` plots the original signal, IMFs, and residual signal as subplots in the same figure.

Examples

Perform Empirical Mode Decomposition and Visualize Hilbert Spectrum of Signal

Load and visualize a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The vibration of a jackhammer and the sound of fireworks are examples of nonstationary continuous signals. The signal is sampled at a rate `fs`.

```
load('sinusoidalSignalExampleData.mat','X','fs')
t = (0:length(X)-1)/fs;
plot(t,X)
xlabel('Time(s)')
```



The mixed signal contains sinusoidal waves with different amplitude and frequency values.

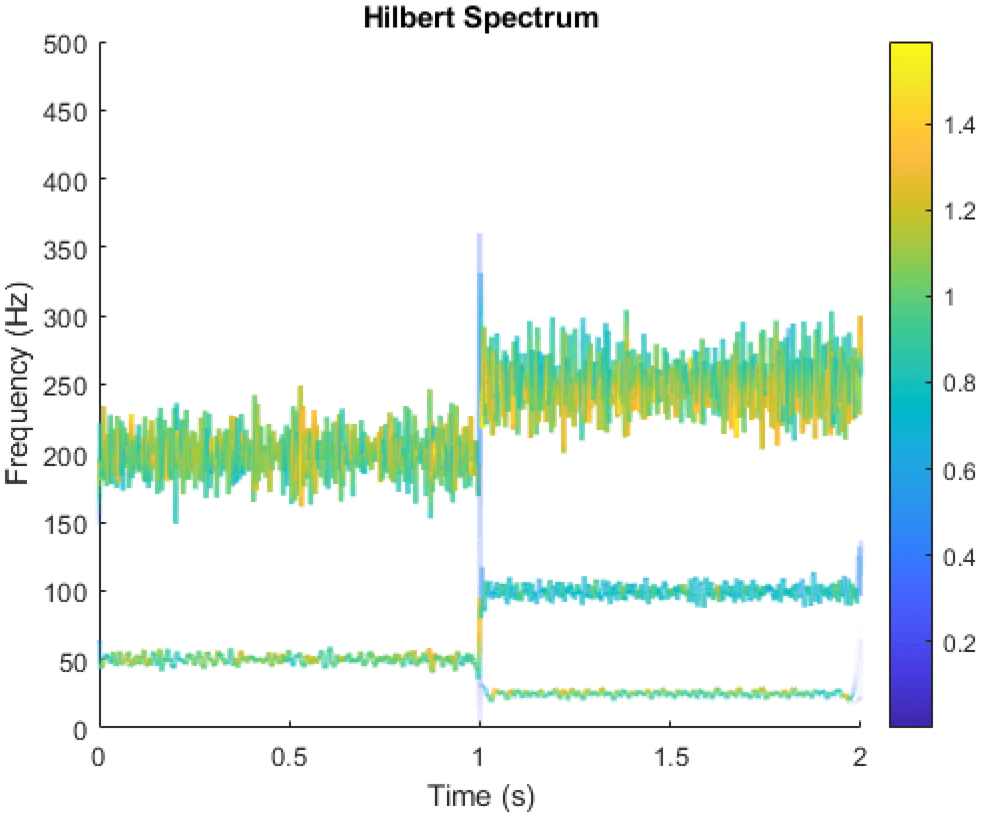
To create the Hilbert spectrum plot, you need the intrinsic mode functions (IMFs) of the signal. Perform empirical mode decomposition to compute the IMFs and residuals of the signal. Since the signal is not smooth, specify 'pchip' as the interpolation method.

```
[imf,residual,info] = emd(X,'Interpolation','pchip');
```

The table generated in the command window indicates the number of sift iterations, the relative tolerance, and the sift stop criterion for each generated IMF. This information is also contained in `info`. You can hide the table by adding the `'Display',0` name value pair.

Create the Hilbert spectrum plot using the `imf` components obtained using empirical mode decomposition.

```
hht(imf,fs)
```



The frequency versus time plot is a sparse plot with a vertical color bar indicating the instantaneous energy at each point in the IMF. The plot represents the instantaneous frequency spectrum of each component decomposed from the original mixed signal. Three IMFs appear in the plot with a distinct change in frequency at 1 second.

Zero Crossings and Extrema in Intrinsic Mode Function of Sinusoid

This trigonometric identity presents two different views of the same physical signal:

$$\frac{5}{2}\cos 2\pi f_1 t + \frac{1}{4}\left(\cos 2\pi (f_1 + f_2)t + \cos 2\pi (f_1 - f_2)t\right) = \left(2 + \cos^2 \pi f_2 t\right)\cos 2\pi f_1 t.$$

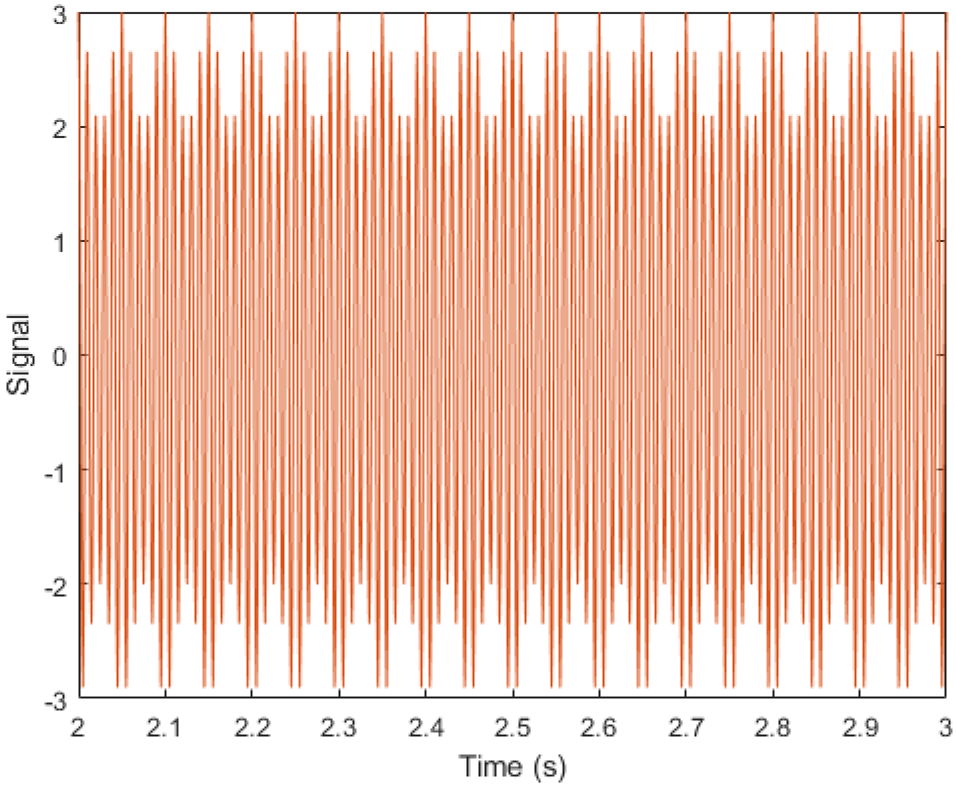
Generate two sinusoids, `s` and `z`, such that `s` is the sum of three sine waves and `z` is a single sine wave with a modulated amplitude. Verify that the two signals are equal by calculating the infinity norm of their difference.

```
t = 0:1e-3:10;
omega1 = 2*pi*100;
omega2 = 2*pi*20;
s = 0.25*cos((omega1-omega2)*t) + 2.5*cos(omega1*t) + 0.25*cos((omega1+omega2)*t);
z = (2+cos(omega2/2*t).^2).*cos(omega1*t);

norm(s-z,Inf)
```

```
ans = 3.2729e-13
```

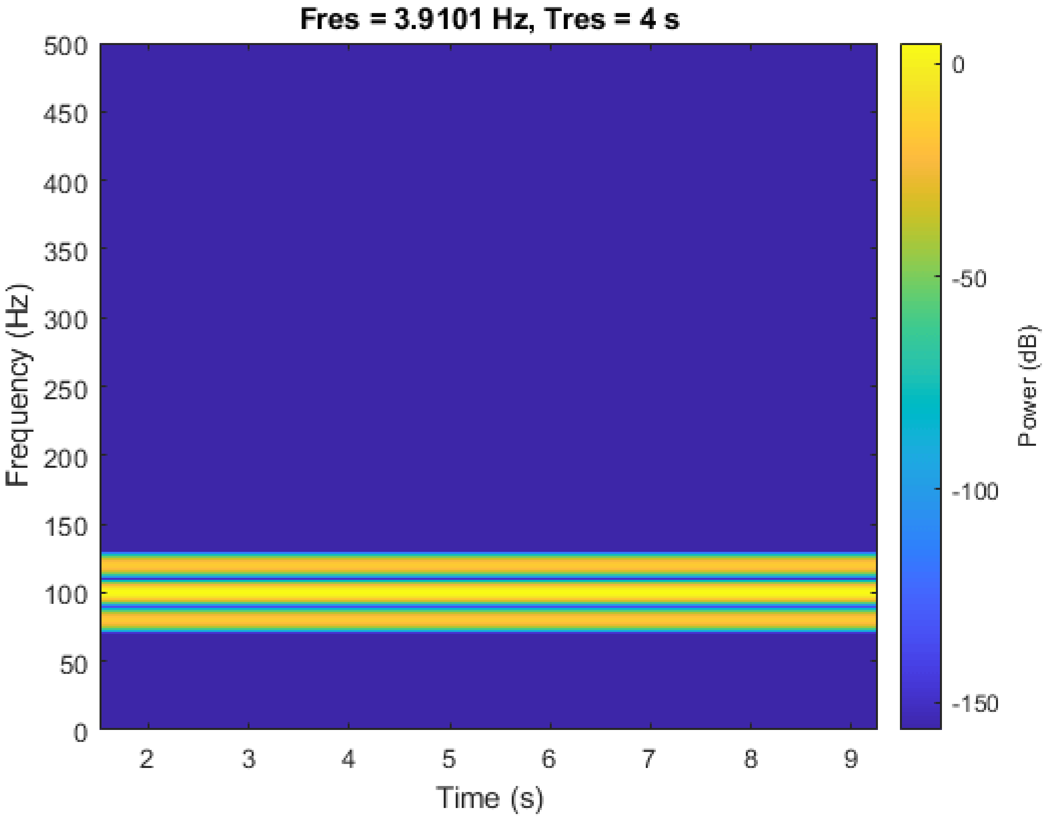
Plot the sinusoids and select a 1-second interval starting at 2 seconds.

```
plot(t,[s' z'])
xlim([2 3])
xlabel('Time (s)')
ylabel('Signal')
```



Obtain the spectrogram of the signal. The spectrogram shows three distinct sinusoidal components. Fourier analysis sees the signals as a superposition of sine waves.

```
pspectrum(s,1000,'spectrogram','TimeResolution',4)
```



Use emd to compute the intrinsic mode functions (IMFs) of the signal and additional diagnostic information. The function by default outputs a table that indicates the number of sifting iterations, the relative tolerance, and the sifting stop criterion for each IMF. Empirical mode decomposition sees the signal as z.

```
[imf,~,info] = emd(s);
```

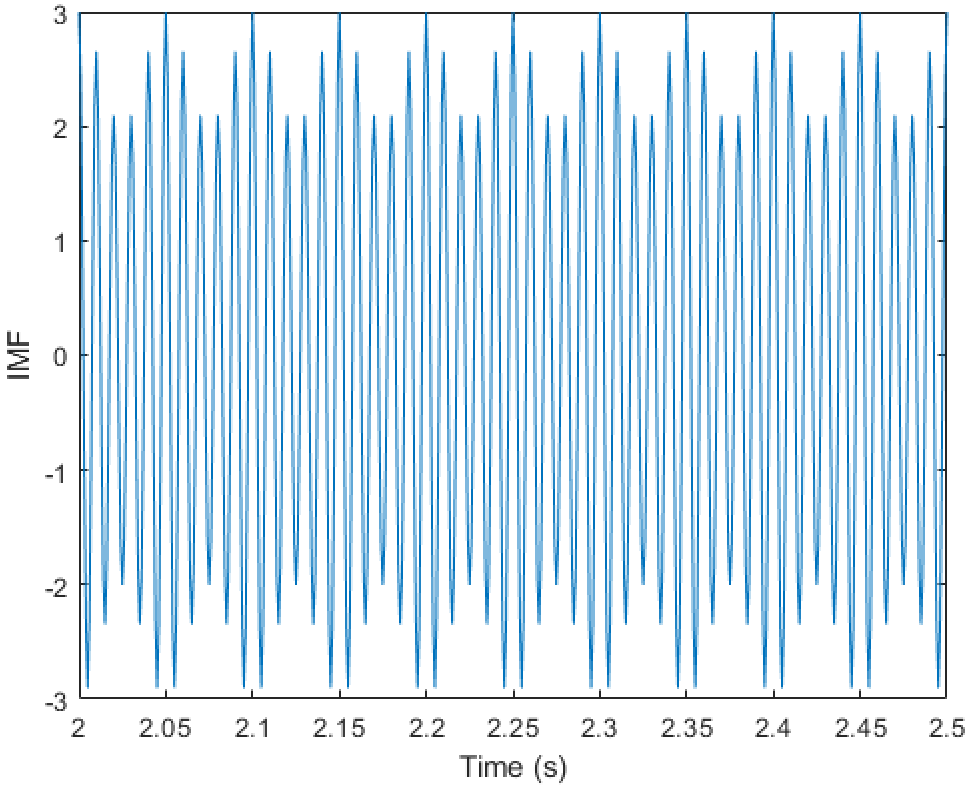
The number of zero crossings and local extrema differ by at most one. This satisfies the necessary condition for the signal to be an IMF.

```
info.NumZerocrossing - info.NumExtrema
```

```
ans = 1
```

Plot the IMF and select a 0.5-second interval starting at 2 seconds. The IMF is an AM signal because emd views the signal as amplitude modulated.

```
plot(t,imf)
xlim([2 2.5])
xlabel('Time (s)')
ylabel('IMF')
```

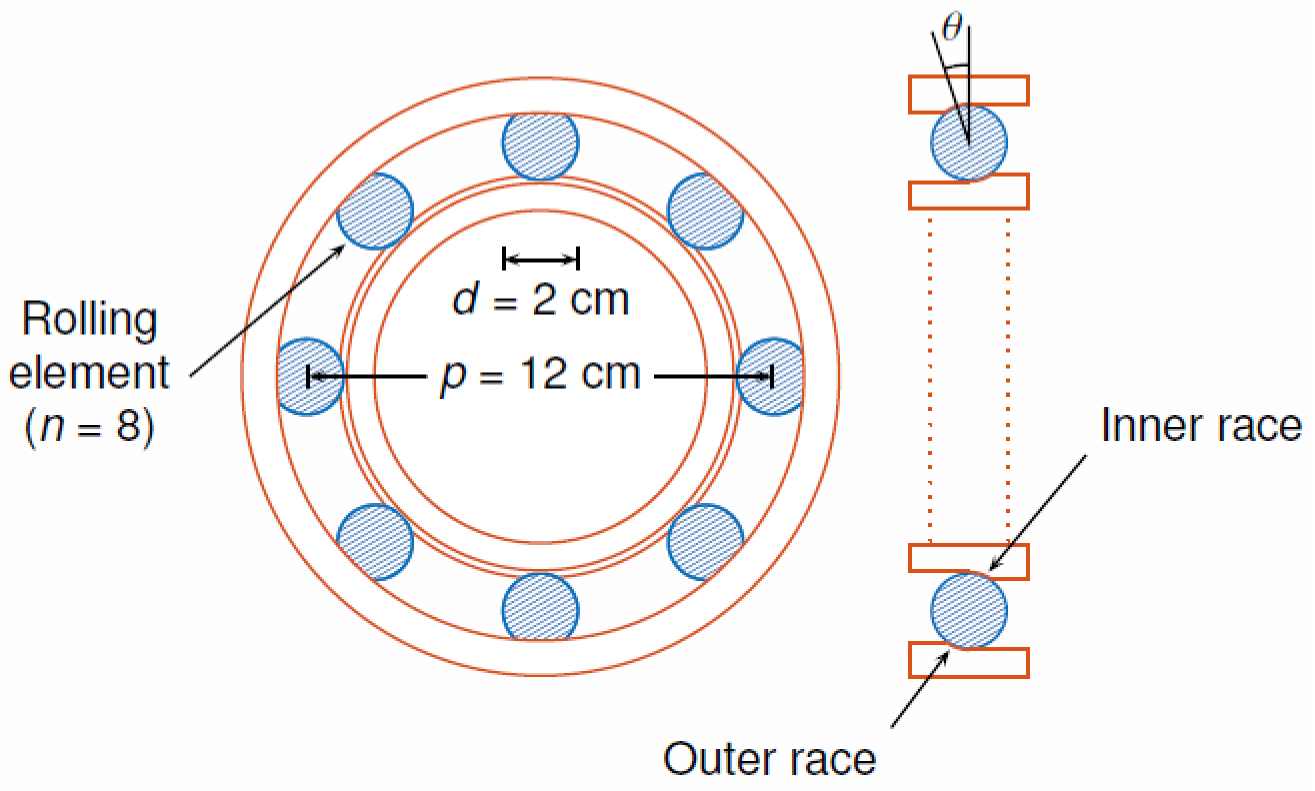


Compute Intrinsic Mode Functions of Vibration Signal

Simulate a vibration signal from a damaged bearing. Perform empirical mode decomposition to visualize the IMFs of the signal and look for defects.

A bearing with a pitch diameter of 12 cm has eight rolling elements. Each rolling element has a diameter of 2 cm. The outer race remains stationary as the inner race is driven at 25 cycles per second. An accelerometer samples the bearing vibrations at 10 kHz.

```
fs = 10000;
f0 = 25;
n = 8;
d = 0.02;
p = 0.12;
```



The vibration signal from the healthy bearing includes several orders of the driving frequency.

```
t = 0:1/fs:10-1/fs;
yHealthy = [1 0.5 0.2 0.1 0.05]*sin(2*pi*f0*[1 2 3 4 5]'.*t)/5;
```

A resonance is excited in the bearing vibration halfway through the measurement process.

```
yHealthy = (1+1./(1+linspace(-10,10,length(yHealthy)).^4)).*yHealthy;
```

The resonance introduces a defect in the outer race of the bearing that results in progressive wear. The defect causes a series of impacts that recur at the ball pass frequency outer race (BPFO) of the bearing:

$$\text{BPFO} = \frac{1}{2}nf_0\left[1 - \frac{d}{p}\cos\theta\right],$$

where f_0 is the driving rate, n is the number of rolling elements, d is the diameter of the rolling elements, p is the pitch diameter of the bearing, and θ is the bearing contact angle. Assume a contact angle of 15° and compute the BPFO.

```
ca = 15;
bpfo = n*f0/2*(1-d/p*cosd(ca));
```

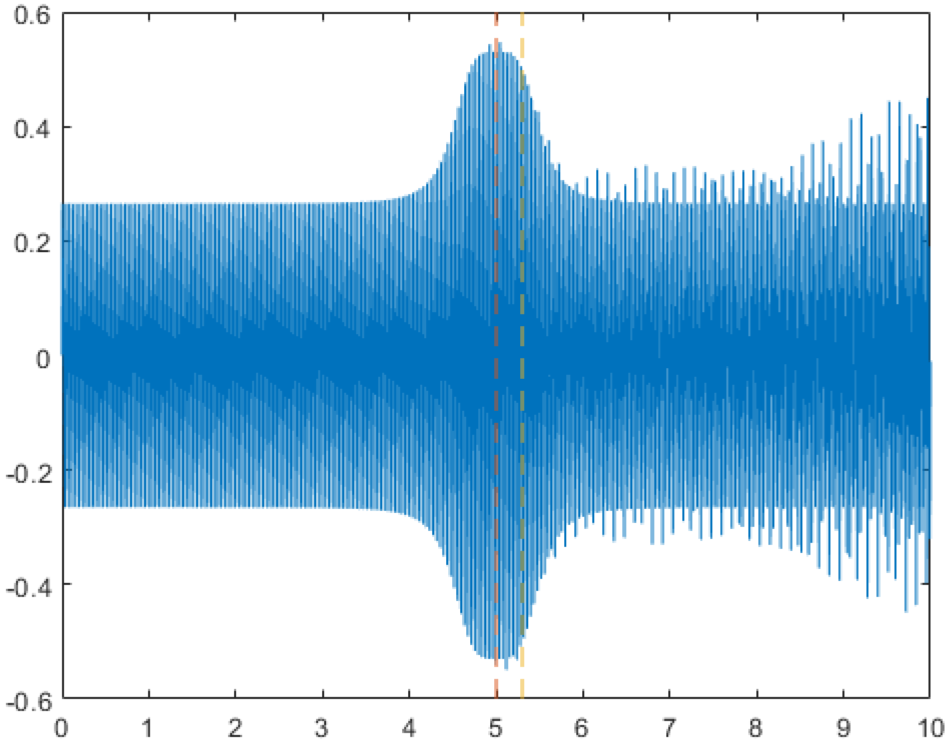
Use the `pulstran` (Signal Processing Toolbox) function to model the impacts as a periodic train of 5-millisecond sinusoids. Each 3 kHz sinusoid is windowed by a flat top window. Use a power law to introduce progressive wear in the bearing vibration signal.

```
fImpact = 3000;
tImpact = 0:1/fs:5e-3-1/fs;
```

```
wImpact = flattopwin(length(tImpact))'/10;
xImpact = sin(2*pi*fImpact*tImpact).*wImpact;

tx = 0:1/bpfo:t(end);
tx = [tx; 1.3.^tx-2];

nWear = 49000;
nSamples = 100000;
yImpact = pulstran(t,tx',xImpact,fs)/5;
yImpact = [zeros(1,nWear) yImpact(1,(nWear+1):nSamples)];
```

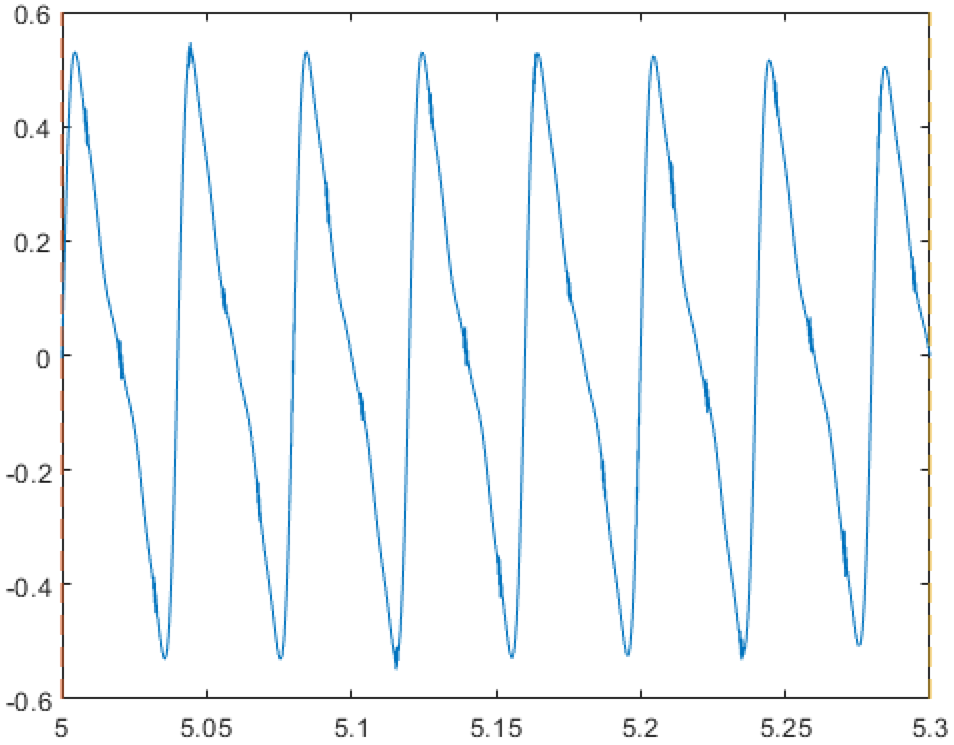
Generate the BPFO vibration signal by adding the impacts to the healthy signal. Plot the signal and select a 0.3-second interval starting at 5.0 seconds.

```
yBPFO = yImpact + yHealthy;

xLimLeft = 5.0;
xLimRight = 5.3;
yMin = -0.6;
yMax = 0.6;

plot(t,yBPFO)

hold on
[limLeft,limRight] = meshgrid([xLimLeft xLimRight],[yMin yMax]);
plot(limLeft,limRight,'--')
hold off
```



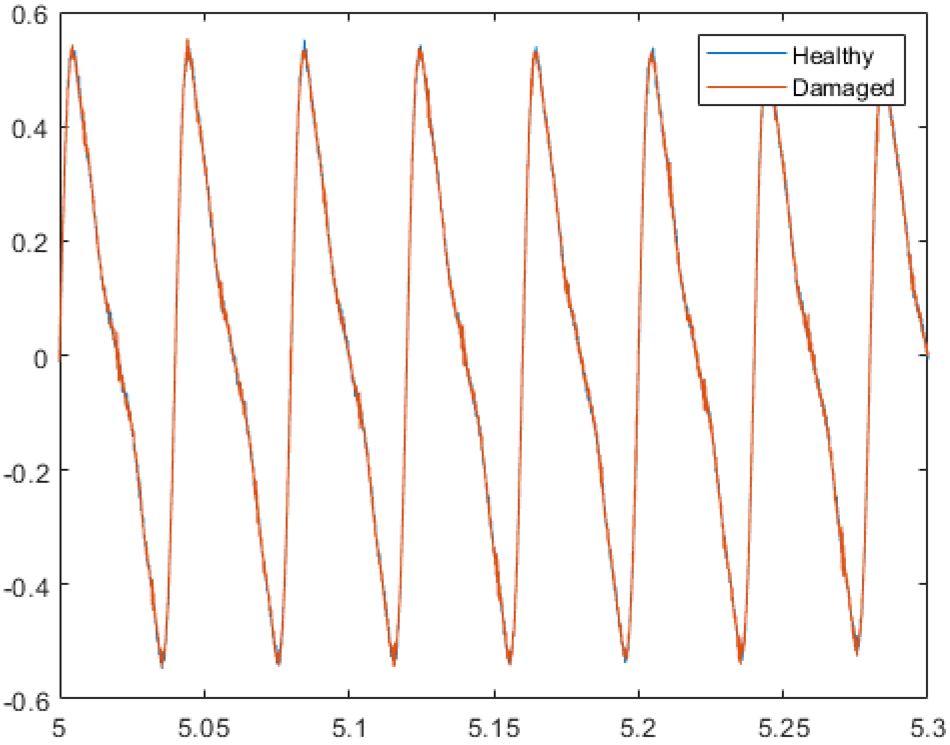
Zoom in on the selected interval to visualize the effect of the impacts.

```
xlim([xLimLeft xLimRight])
```



Add white Gaussian noise to the signals. Specify a noise variance of $1/150^2$.

```
rn = 150;
yGood = yHealthy + randn(size(yHealthy))/rn;
yBad = yBPFO + randn(size(yHealthy))/rn;

plot(t,yGood,t,yBad)
xlim([xLimLeft xLimRight])
legend('Healthy','Damaged')
```



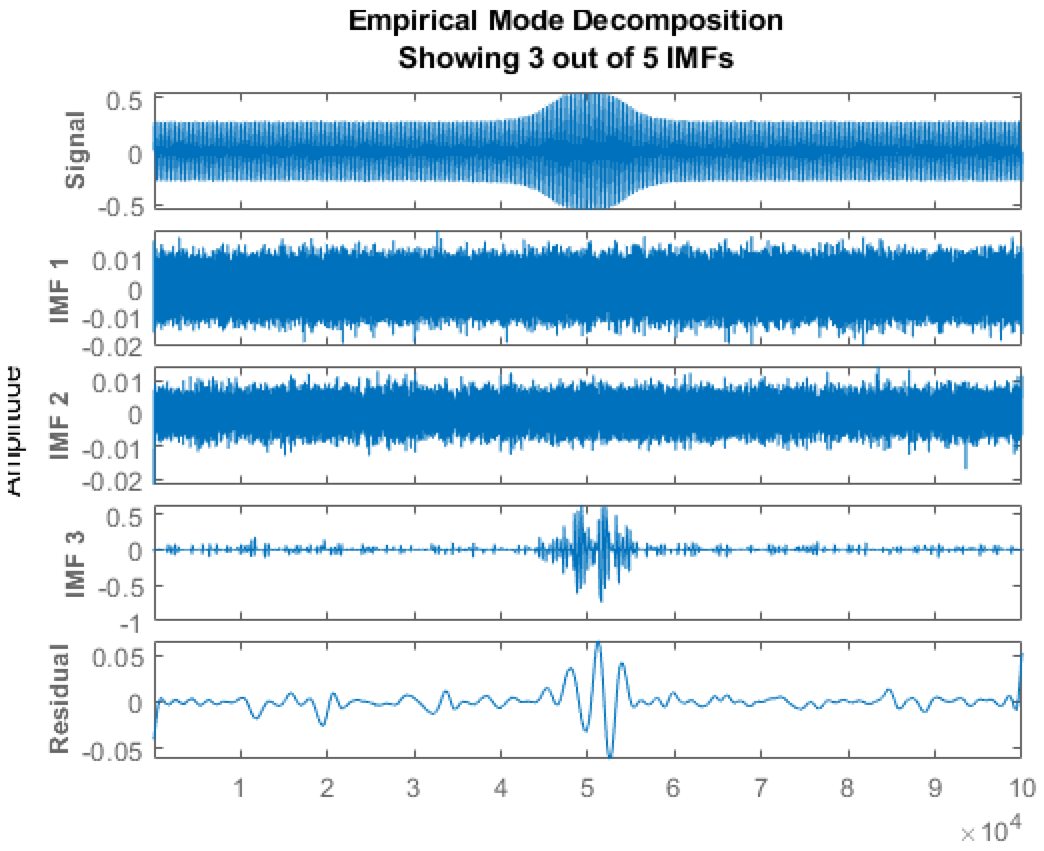
Use emd to perform an empirical mode decomposition of the healthy bearing signal. Compute the first five intrinsic mode functions (IMFs). Use the 'Display' name-value pair to show a table with the number of sifting iterations, the relative tolerance, and the sifting stop criterion for each IMF.

```
imfGood = emd(yGood,'MaxNumIMF',5,'Display',1);
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit
      1      |       3      |     0.017132   |  SiftMaxRelativeTolerance
      2      |       3      |      0.12694   |  SiftMaxRelativeTolerance
      3      |       6      |      0.14582   |  SiftMaxRelativeTolerance
      4      |       1      |     0.011082   |  SiftMaxRelativeTolerance
      5      |       2      |      0.03463   |  SiftMaxRelativeTolerance
Decomposition stopped because maximum number of intrinsic mode functions was extracted.
```

Use emd without output arguments to visualize the first three modes and the residual.

```
emd(yGood,'MaxNumIMF',5)
```

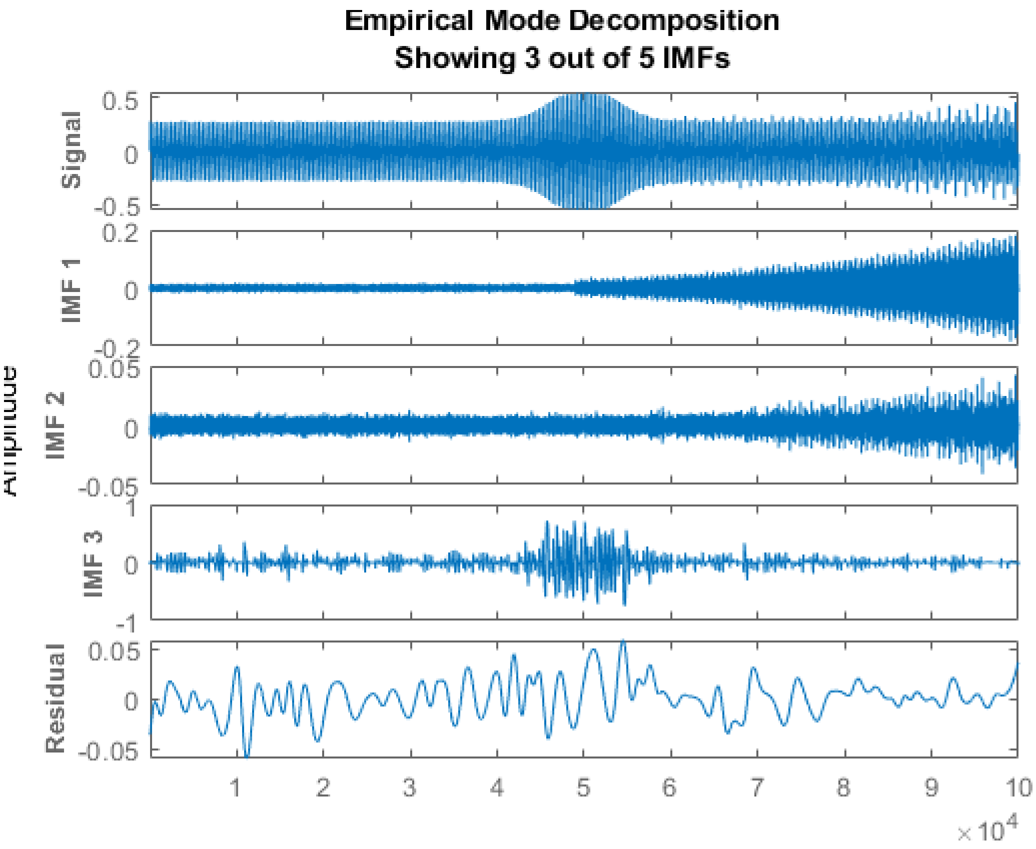


Compute and visualize the IMFs of the defective bearing signal. The first empirical mode reveals the high-frequency impacts. This high-frequency mode increases in energy as the wear progresses. The third mode shows the resonance in the vibration signal.

```
imfBad = emd(yBad,'MaxNumIMF',5,'Display',1);
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit  
      1      |        2     |     0.041274   |  SiftMaxRelativeTolerance
      2      |        3     |      0.16695   |  SiftMaxRelativeTolerance
      3      |        3     |      0.18428   |  SiftMaxRelativeTolerance
      4      |        1     |     0.037177   |  SiftMaxRelativeTolerance
      5      |        2     |     0.095861   |  SiftMaxRelativeTolerance
Decomposition stopped because maximum number of intrinsic mode functions was extracted.
```

```
emd(yBad,'MaxNumIMF',5)
```

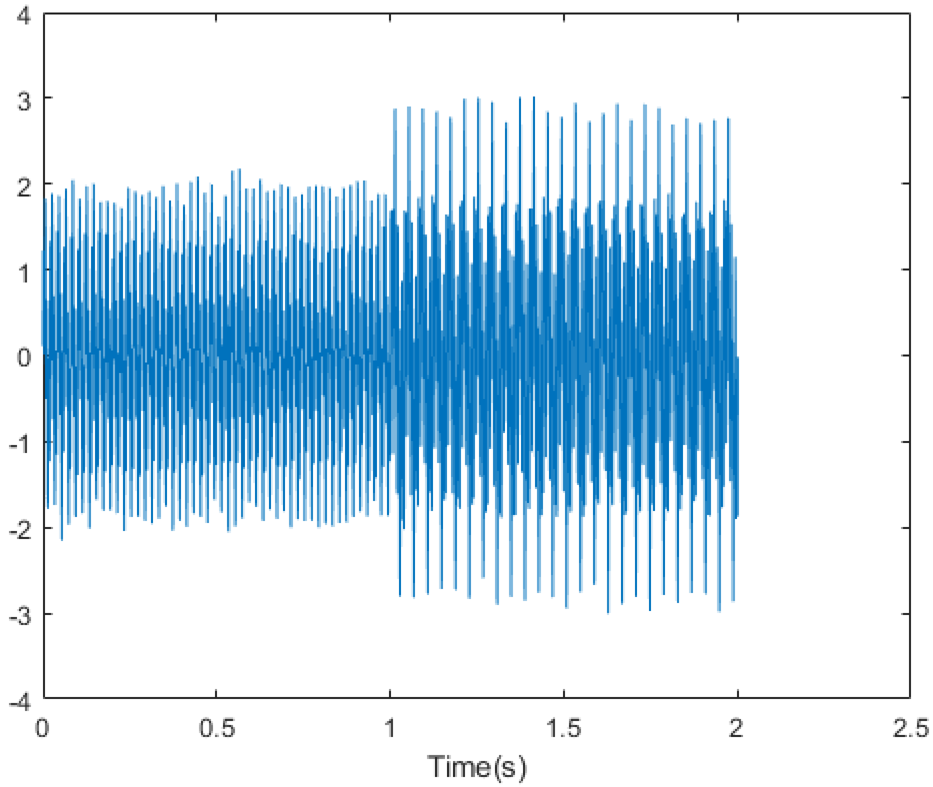


The next step in the analysis is to compute the Hilbert spectrum of the extracted IMFs. For more details, see the “Compute Hilbert Spectrum of Vibration Signal” (Signal Processing Toolbox) example.

Visualize Residual and Intrinsic Mode Functions of Signal

Load and visualize a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The vibration of a jackhammer and the sound of fireworks are examples of nonstationary continuous signals. The signal is sampled at a rate `fs`.

```
load('sinusoidalSignalExampleData.mat','X','fs')
t = (0:length(X)-1)/fs;
plot(t,X)
xlabel('Time(s)')
```

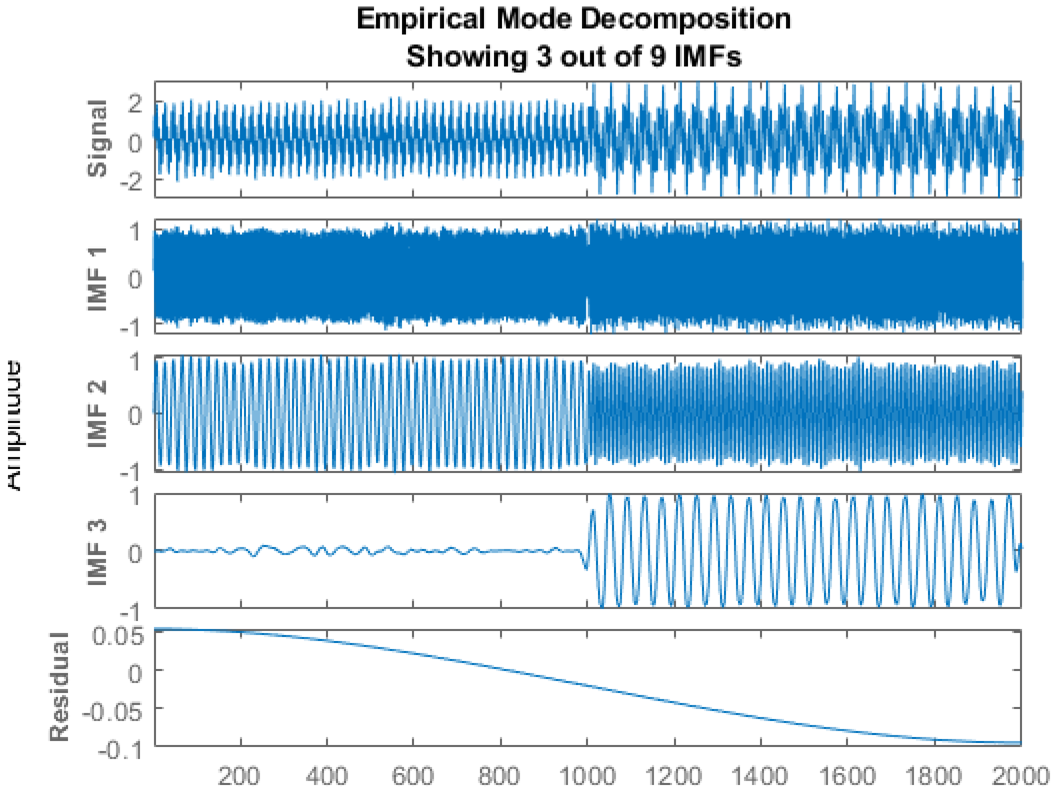


The mixed signal contains sinusoidal waves with different amplitude and frequency values.

Perform empirical mode decomposition to plot the intrinsic mode functions and residual of the signal. Since the signal is not smooth, specify 'pchip' as the interpolation method.

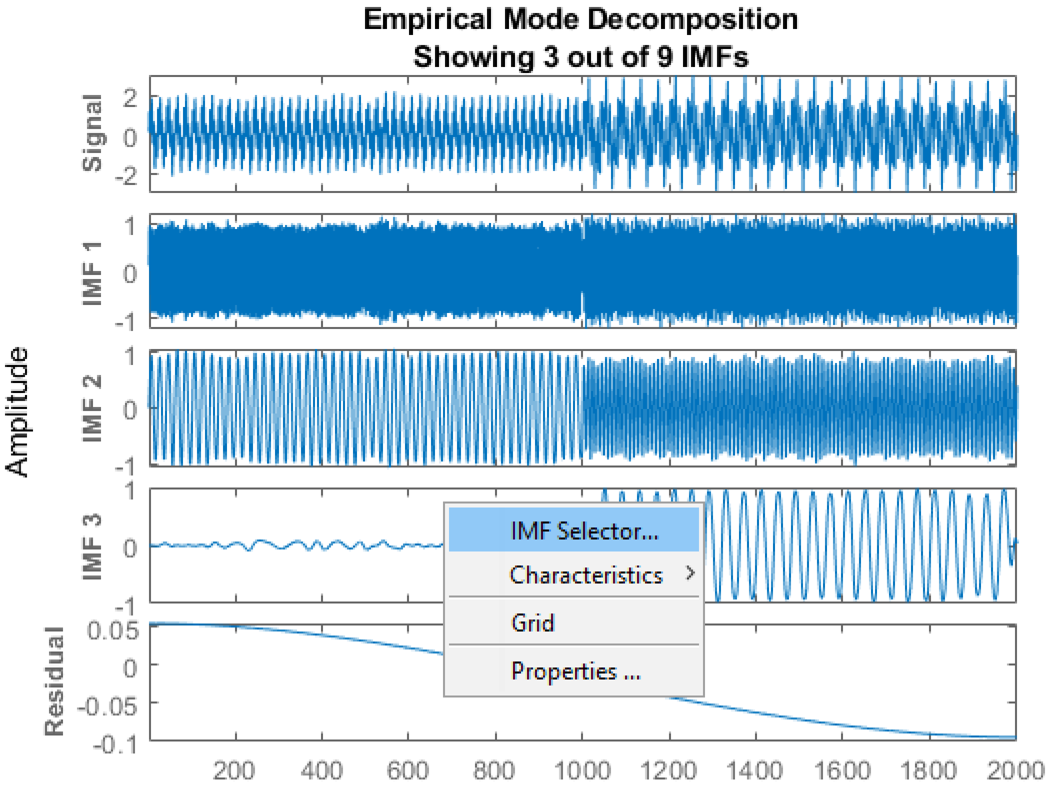
```
emd(X,'Interpolation','pchip','Display',1)
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit
      1      |        2     |    0.026352    |  SiftMaxRelativeTolerance
      2      |        2     |   0.0039573    |  SiftMaxRelativeTolerance
      3      |        1     |    0.024838    |  SiftMaxRelativeTolerance
      4      |        2     |     0.05929    |  SiftMaxRelativeTolerance
      5      |        2     |     0.11317    |  SiftMaxRelativeTolerance
      6      |        2     |     0.12599    |  SiftMaxRelativeTolerance
      7      |        2     |     0.13802    |  SiftMaxRelativeTolerance
      8      |        3     |     0.15937    |  SiftMaxRelativeTolerance
      9      |        2     |     0.15923    |  SiftMaxRelativeTolerance
Decomposition stopped because the number of extrema in the residual signal is less than the 'MaxN
```

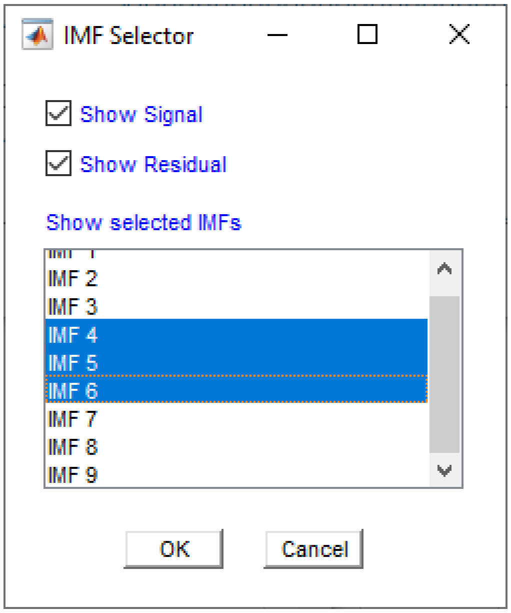


`emd` generates an interactive plot with the original signal, the first 3 IMFs, and the residual. The table generated in the command window indicates the number of sift iterations, the relative tolerance, and the sift stop criterion for each generated IMF. You can hide the table by removing the `'Display'` name-value pair or specifying it as `0`.

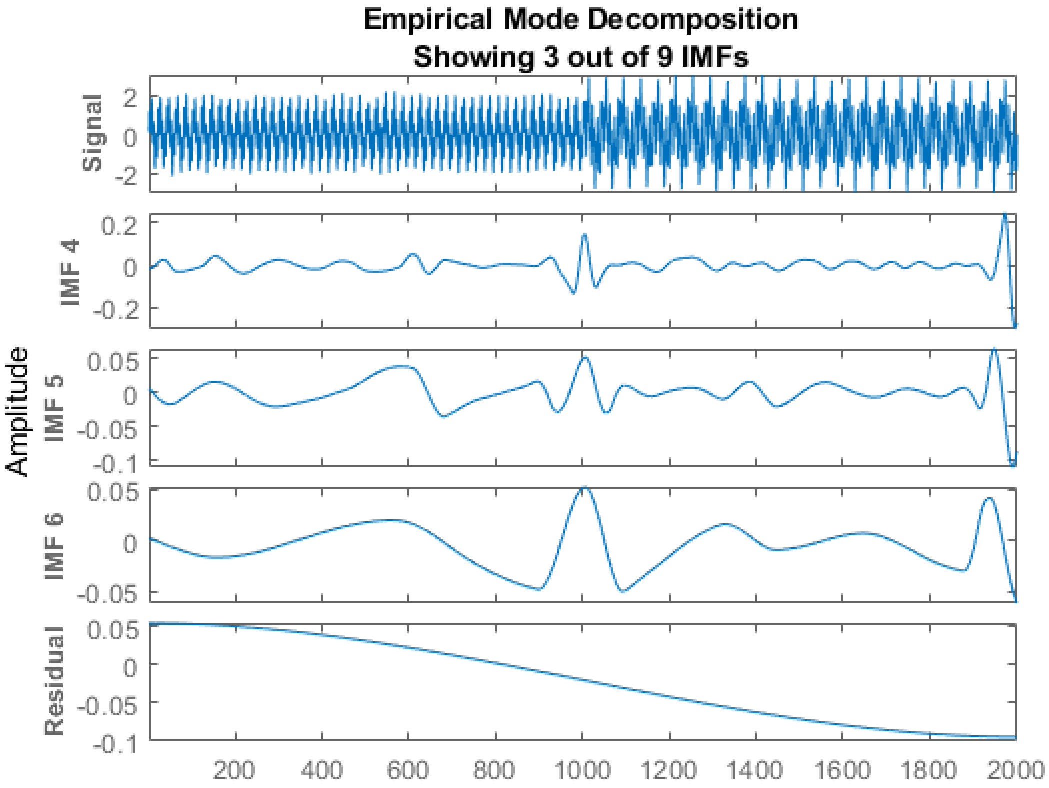
Right-click on the white space in the plot to open the **IMF selector** window. Use **IMF selector** to selectively view the generated IMFs, the original signal, and the residual.



Select the IMFs to be displayed from the list. Choose whether to display the original signal and residual on the plot.



The selected IMFs are now displayed on the plot.



Use the plot to visualize individual components decomposed from the original signal along with the residual. Note that the residual is computed for the total number of IMFs, and does not change based on the IMFs selected in the **IMF selector** window.

Input Arguments

x — Time-domain signal

vector | timetable

Time-domain signal, specified as a real-valued vector, or a single-variable timetable with a single column. If x is a timetable, x must contain increasing, finite row times.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: 'MaxNumIMF',5

SiftRelativeTolerance — Cauchy-type convergence criterion

0.2 (default) | positive scalar

Cauchy-type convergence criterion, specified as the comma-separated pair consisting of `'SiftRelativeTolerance'` and a positive scalar. `SiftRelativeTolerance` is one of the sifting stop criteria, that is, sifting stops when the current relative tolerance is less than `SiftRelativeTolerance`. For more information, see “Sift Relative Tolerance” on page 1-449.

`SiftMaxIterations` — Maximum number of sifting iterations

`100` (default) | positive scalar integer

Maximum number of sifting iterations, specified as the comma-separated pair consisting of `'SiftMaxIterations'` and a positive scalar integer. `SiftMaxIterations` is one of the sifting stop criteria, that is, sifting stops when the current number of iterations is larger than `SiftMaxIterations`.

`SiftMaxIterations` can be specified using only positive whole numbers.

`MaxNumIMF` — Maximum number of IMFs extracted

`10` (default) | positive scalar integer

Maximum number of IMFs extracted, specified as the comma-separated pair consisting of `'MaxNumIMF'` and a positive scalar integer. `MaxNumIMF` is one of the decomposition stop criteria, that is, decomposition stops when number of IMFs generated is equal to `MaxNumIMF`.

`MaxNumIMF` can be specified using only positive whole numbers.

`MaxNumExtrema` — Maximum number of extrema in the residual signal

`1` (default) | positive scalar integer

Maximum number of extrema in the residual signal, specified as the comma-separated pair consisting of `'MaxNumExtrema'` and a positive scalar integer. `MaxNumExtrema` is one of the decomposition stop criteria, that is, decomposition stops when number of extrema is less than `MaxNumExtrema`.

`MaxNumExtrema` can be specified using only positive whole numbers.

`MaxEnergyRatio` — Signal to residual energy ratio

`20` (default) | scalar

Signal to residual energy ratio, specified as the comma-separated pair consisting of `'MaxEnergyRatio'` and a scalar. `MaxEnergyRatio` is the ratio of the energy of the signal at the beginning of sifting and the average envelope energy. `MaxEnergyRatio` is one of the decomposition stop criteria, that is, decomposition stops when current energy ratio is larger than `MaxEnergyRatio`. For more information, see “Energy Ratio” on page 1-450.

`Interpolation` — Interpolation method for envelope construction

`'spline'` (default) | `'pchip'`

Interpolation method for envelope construction, specified as the comma-separated pair consisting of `'Interpolation'` and either `'spline'` or `'pchip'`.

Specify `Interpolation` as:

- `'spline'`, if `x` is a smooth signal
- `'pchip'`, if `x` is a nonsmooth signal

`'spline'` interpolation method uses cubic splines, while `'pchip'` uses piecewise-cubic Hermite interpolating polynomials.

`Display` — Toggle information display in the command window

0 (default) | 1

Toggle information display in the command window, specified as the comma-separated pair consisting of `'Display'` and either 0 or 1. The table generated in the command window indicates the number of sift iterations, the relative tolerance, and the sift stop criterion for each generated IMF. Specify `Display` as 1 to show the table or 0 to hide the table.

Output Arguments

`imf` — Intrinsic mode function

matrix | timetable

Intrinsic mode function (IMF), returned as a matrix or timetable. Each IMF is an amplitude and frequency modulated signal with positive and slowly varying envelopes. To perform spectral analysis of a signal, you can apply the Hilbert-Huang transform to its IMFs. See `hht` and “Intrinsic Mode Functions” on page 1-449.

`imf` is returned as:

- A matrix whose each column is an `imf`, when `x` is a vector
- A timetable, when `x` is a single data column timetable

`residual` — Residual of the signal

column vector | single data column timetable

Residual of the signal, returned as a column vector or a single data column timetable. `residual` represents the portion of the original signal `x` not decomposed by `emd`.

`residual` is returned as:

- A column vector, when `x` is a vector.
- A single data column timetable, when `x` is a single data column timetable.

`info` — Additional information for diagnostics

structure

Additional information for diagnostics, returned as a structure with the following fields:

- `NumIMF` — Number of IMFs extracted

  `NumIMF` is a vector from 1 to *N*, where *N* is the number of IMFs. If no IMFs are extracted, `NumIMF` is empty.
- `NumExtrema` — Number of extrema in each IMF

  `NumExtrema` is a vector equal in length to the number of IMFs. The *k*th element of `NumExtrema` is the number of extrema found in the *k*th IMF. If no IMFs are extracted, `NumExtrema` is empty.
- `NumZerocrossing` — Number of zero crossings in each IMF

  Number of zero crossings in each IMF. `NumZerocrossing` is a vector equal in length to the number of IMFs. The *k*th element of `NumZerocrossing` is the number of zero crossings in the *k*th IMF. If no IMFs are extracted, `NumZerocrossing` is empty.

- `NumSifting` — Number of sifting iterations used to extract each IMF

  `NumSifting` is a vector equal in length to the number of IMFs. The *k*th element of `NumSifting` is the number of sifting iterations used in the extraction of the *k*th IMF. If no IMFs are extracted, `NumSifting` is empty.
- `MeanEnvelopeEnergy` — Energy of the mean of the upper and lower envelopes obtained for each IMF

  If `UE` is the upper envelope and `LE` is the lower envelope, `MeanEnvelopeEnergy` is `mean(((LE+UL)/2).^2)`. `MeanEnvelopeEnergy` is a vector equal in length to the number of IMFs. The *k*th element of `MeanEnvelopeEnergy` is the mean envelope energy for the *k*th IMF. If no IMFs are extracted, `MeanEnvelopeEnergy` is empty.
- `RelativeTolerance` — Final relative tolerance of the residual for each IMF

  The relative tolerance is defined as the ratio of the squared 2-norm of the difference between the residual from the previous sifting step and the residual from the current sifting step to the squared 2-norm of the residual from the *i*th sifting step. The sifting process stops when `RelativeTolerance` is less than `SiftRelativeTolerance`. For additional information, see “Sift Relative Tolerance” on page 1-449. `RelativeTolerance` is a vector equal in length to the number of IMFs. The *k*th element of `RelativeTolerance` is the final relative tolerance obtained for the *k*th IMF. If no IMFs are extracted, `RelativeTolerance` is empty.

More About

Empirical Mode Decomposition

The empirical mode decomposition (EMD) algorithm decomposes a signal $x(t)$ into intrinsic mode functions (IMFs) and a residual in an iterative process. The core component of the algorithm involves *sifting* a function $x(t)$ to obtain a new function $Y(t)$:

- First find the local minima and maxima of $x(t)$.
- Then use the local extrema to construct lower and upper envelopes $s_-(t)$ and $s_+(t)$, respectively, of $x(t)$. Form the mean of the envelopes, $m(t)$.
- Subtract the mean from $x(t)$ to obtain the residual: $Y(t) = x(t) - m(t)$.

An overview of the decomposition is as follows:

1. To begin, let $r_0(t) = x(t)$, where $x(t)$ is the initial signal, and let $i = 0$.
2. Before sifting, check $r_\mathrm{i}(t)$:

   a. Find the total number (TN) of local extrema of $r_i(t)$.

   b. Find the energy ratio (ER) of $r_i(t)$ (see “Energy Ratio” on page 1-450).
3. If (ER > `MaxEnergyRatio`) or (TN < `MaxNumExtrema`) or (number of IMFs > `MaxNumIMF`) then stop the decomposition.
4. Let $r_{i,\mathrm{Prev}}(t) = r_i(t)$.
5. Sift $r_{i,\mathrm{Prev}}(t)$ to obtain $r_{i,\mathrm{Cur}}(t)$.
6. Check $r_{i,\mathrm{Cur}}(t)$

   a. Find the relative tolerance (RT) of $r_{i,\mathrm{Cur}}(t)$ (see “Sift Relative Tolerance” on page 1-449).

   b Get current sift iteration number (IN).

7. If (RT < `SiftRelativeTolerance`) or (IN > `SiftMaxIterations`) then stop sifting. An IMF has been found: $\mathrm{IMF}_i(t) = r_{i,\mathrm{Cur}}(t)$. Otherwise, let $r_{i,\mathrm{Prev}}(t) = r_{i,\mathrm{Cur}}(t)$ and go to Step 5.
8. Let $r_{i+1}(t) = r_i(t) - r_{i,\mathrm{Cur}}(t)$.
9. Let $i = i + 1$. Return to Step 2.

For additional information, see [1] and [3].

Intrinsic Mode Functions

The EMD algorithm decomposes, via an iterative sifting process, a signal $x(t)$ into IMFs $imf_i(t)$ and a residual $r_N(t)$:

$$X(t) = \sum_{i=1}^{N} \mathrm{IMF}_i(t) + r_N(t)$$

When first introduced by Huang et al. [1], an IMF was defined to be a function with two characteristics:

- The number of local extrema — the total number of local minima and local maxima — and the number of zero crossings differ by at most one.
- The mean value of the upper and lower envelopes constructed from the local extrema is zero.

However, as noted in [4], sifting until a strict IMF is obtained can result in IMFs that have no physical significance. Specifically, sifting until the number of zero crossings and local extrema differ by at most one can result in pure-tone like IMFs, in other words, functions very similar to what would be obtained by projection on the Fourier basis. This situation is precisely what EMD strives to avoid, preferring AM-FM modulated components for their physical significance.

Reference [4] proposes options to obtain physically meaningful results. The `emd` function relaxes the original IMF definition by using “Sift Relative Tolerance” on page 1-449, a Cauchy-type stop criterion. The `emd` function iterates to extract natural AM-FM modes. The IMFs generated may fail to satisfy the local extrema-zero crossings criteria. See “Zero Crossings and Extrema in Intrinsic Mode Function of Sinusoid” on page 1-432.

Sift Relative Tolerance

Sift Relative Tolerance is a Cauchy-type stop criterion proposed in [4]. Sifting stops when current relative tolerance is less than `SiftRelativeTolerance`. The current relative tolerance is defined as

$$\text{Relative Tolerance} \triangleq \frac{\left\| r_{\mathrm{prev}}(t) - r_{\mathrm{cur}}(t) \right\|_2^2}{\left\| r_{\mathrm{prev}}(t) \right\|_2^2}.$$

Because the Cauchy criterion does not directly count the number of zero crossings and local extrema, it is possible that the IMFs returned by the decomposition do not satisfy the strict definition of an intrinsic mode function. In those cases, you can try reducing the value of the `SiftRelativeTolerance` from its default value. See [4] for a detailed discussion of stopping criteria. The reference also discusses the advantages and disadvantages of insisting on strictly defined IMFs in empirical mode decomposition.

Energy Ratio

Energy ratio is the ratio of the energy of the signal at the beginning of sifting and the average envelope energy [2]. Decomposition stops when current energy ratio is larger than `MaxEnergyRatio`. For the ith IMF, the energy ratio is defined as

$$\text{Energy Ratio} \triangleq 10\log_{10}\left(\frac{\|X(t)\|_2}{\|r_i(t)\|_2}\right).$$

References

[1] Huang, Norden E., Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry H. Liu. “The Empirical Mode Decomposition and the Hilbert Spectrum for Nonlinear and Non-Stationary Time Series Analysis.” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454, no. 1971 (March 8, 1998): 903–95. https://doi.org/10.1098/rspa.1998.0193.

[2] Rato, R.T., M.D. Ortigueira, and A.G. Batista. “On the HHT, Its Problems, and Some Solutions.” *Mechanical Systems and Signal Processing* 22, no. 6 (August 2008): 1374–94. https://doi.org/10.1016/j.ymssp.2007.11.028.

[3] Rilling, Gabriel, Patrick Flandrin, and Paulo Gonçalves. "On Empirical Mode Decomposition and Its Algorithms." *IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing* 2003. NSIP-03. Grado, Italy. 8–11.

[4] Wang, Gang, Xian-Yao Chen, Fang-Li Qiao, Zhaohua Wu, and Norden E. Huang. “On Intrinsic Mode Function.” *Advances in Adaptive Data Analysis* 02, no. 03 (July 2010): 277–93. https://doi.org/10.1142/S1793536910000549.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Timetables are not supported for code generation.
- If supplied, the interpolation method specified using the `'Interpolation'` name-value pair must be a compile-time constant.

See Also

Apps

Signal Multiresolution Analyzer

Functions

`hht` | `vmd`

Topics

“Time-Frequency Gallery”

Introduced in R2018a

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

eq

Laurent polynomials or Laurent matrices equality test

Syntax

```
tf = eq(A,B)
tf = (A == B)
```

Description

`tf = eq(A,B)` compares the pair of Laurent polynomials or Laurent matrices `A` and `B` and returns `1` (`true`) if the two are identical and `0` (`false`) otherwise.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `eq`. The input data type determines which version is executed.

`tf = (A == B)` is equivalent to `tf = eq(A,B)`.

Examples

Test Equality of Laurent Polynomials

Create two Laurent polynomials:

- $a(z) = 2z^3 - 3z^2 + 4z - 5$
- $b(z) = 4z^3 - 6z^2 + 8z$

```
a = laurentPolynomial(Coefficients=[2 -3 4 -5],MaxOrder=3);
b = laurentPolynomial(Coefficients=[4 -6 8],MaxOrder=3);
```

Confirm $a(z)$ and $b(z)$ are not equal.

```
a ~= b
```

```
ans = logical
   1
```

Confirm $2a(z) + 10$ and $b(z)$ are equal.

```
c = rescale(a,2)+10;
eq(c,b)
```

```
ans = logical
   1
```

Input Arguments

A — Laurent polynomial or Laurent matrix

laurentPolynomial object | laurentMatrix object

Laurent polynomial or Laurent matrix, specified as a laurentPolynomial object or a laurentMatrix object, respectively.

B — Laurent polynomial or Laurent matrix

laurentPolynomial object | laurentMatrix object

Laurent polynomial or Laurent matrix, specified as a laurentPolynomial object or a laurentMatrix object, respectively.

Output Arguments

tf — Equality test result

true or 1 | false or 0

Equality test result, returned as a numeric or logical 1 (true) or 0 (false).

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

ne

Objects

laurentMatrix | laurentPolynomial

Introduced in R2021b

euclid

Euclidean algorithm for Laurent polynomials

Syntax

```
dec = euclid(A,B)
```

Description

`dec = euclid(A,B)` returns an array of structures such that each row of `dec` corresponds to the Euclidean division of the Laurent polynomial `A` by the Laurent polynomial `B`:

`A = B*Q + R`,

where `Q` is the quotient and `R` is the remainder.

Examples

Euclidean Division of Laurent Polynomials

Create two Laurent polynomials:

- $A(z) = z^2 + 3z + 5 + 7z^{-1}$
- $B(z) = 1 + 2z^{-1}$

```
cfa = [1 3 5 7];
cfb = [1 2];
lpA = laurentPolynomial(Coefficients=cfa,MaxOrder=2);
lpB = laurentPolynomial(Coefficients=cfb);
```

Perform Euclidean division of $A(z)$ by $B(z)$. Use the helper function `helperPrintLaurent` to print the quotient and remainder polynomials of each Euclidean division.

```
dec = euclid(lpA,lpB);
numFac = size(dec,1);
for k=1:numFac
    q = helperPrintLaurent(dec(k,1).LP);
    r = helperPrintLaurent(dec(k,2).LP);
    fprintf('Euclidean Division #%d\n',k)
    fprintf('Quotient: %s\n',q)
    fprintf('Remainder:  %s\n \n',r)
end
```

```
Euclidean Division #1

Quotient: z^(2) + z + 3

Remainder:   + z^(-1)


Euclidean Division #2

Quotient: z^(2) + z + 3.5
```

```
Remainder:   - 0.5


Euclidean Division #3

Quotient: z^(2) + 0.75*z + 3.5

Remainder:   + 0.25*z


Euclidean Division #4

Quotient: 1.125*z^(2) + 0.75*z + 3.5

Remainder:  - 0.125*z^(2)
```

For each Euclidean division, confirm that $A(z) = B(z)\ Q_i(z) + R_i(z)$, where $Q_i(z)$ and $R_i(z)$ are the quotient and remainder polynomials, respectively, of the ith division.

```
for k=1:numFac
    q = dec(k,1).LP;
    r = dec(k,2).LP;
    areEqual = (lpA==lpB*q+r);
    fprintf('Euclidean Division #%d: %d\n',k,areEqual)
end
```

```
Euclidean Division #1: 1
Euclidean Division #2: 1
Euclidean Division #3: 1
Euclidean Division #4: 1
```

Input Arguments

A — Laurent polynomial

laurentPolynomial object

Laurent polynomial, specified as a laurentPolynomial object.

B — Laurent polynomial

laurentPolynomial object

Laurent polynomial, specified as a laurentPolynomial object.

Output Arguments

dec — Euclidean algorithm factors

structure array

Euclidean algorithm factors, returned as a N-by-2 structure array, where $N \leq 4$ is the number of decompositions. The ith row of dec contains one Euclidean division of A by B:

```
    A = B*(dec(i,1).LP) + dec(i,2).LP
```

where

- dec(*i*,1).LP is the Laurent polynomial corresponding to the quotient.

- `dec(i,2).LP` is the Laurent polynomial corresponding to the remainder.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

ewt

Empirical wavelet transform

Syntax

```
mra = ewt(x)
[mra,cfs] = ewt(x)
[mra,cfs,wfb] = ewt(x)
[mra,cfs,wfb,info] = ewt(x)
```

```
[___] = ewt(___,Name,Value)
```

```
ewt(___)
```

Description

`mra = ewt(x)` returns the multiresolution analysis (MRA) components corresponding to the empirical wavelet transform (EWT) of `x`. Use `ewt` to decompose signals using an adaptable wavelet subdivision scheme that automatically determines the empirical wavelet and scaling filters and preserves energy.

By default, the number of empirical wavelet filters is automatically determined by identifying peaks in a multitaper power spectral estimate of `x`.

`[mra,cfs] = ewt(x)` returns the EWT analysis coefficients of `x`.

`[mra,cfs,wfb] = ewt(x)` returns the empirical wavelet filter bank used in the analysis of `x`.

`[mra,cfs,wfb,info] = ewt(x)` returns the peak normalized frequencies identified in `x` and the approximate frequency passbands of the wavelet filter bank.

`[___] = ewt(___,Name,Value)` specifies additional options using name-value pair arguments. These arguments can be added to any of the previous input syntaxes. For example, `'MaxNumPeaks',5` specifies a maximum of five peaks used to determine the EWT filter passbands.

`ewt(___)` with no output arguments plots the original signal with the empirical wavelet MRA in the same figure. For complex-valued data, the real part is plotted in the first color in the MATLAB color order matrix and the imaginary part is plotted in the second color.

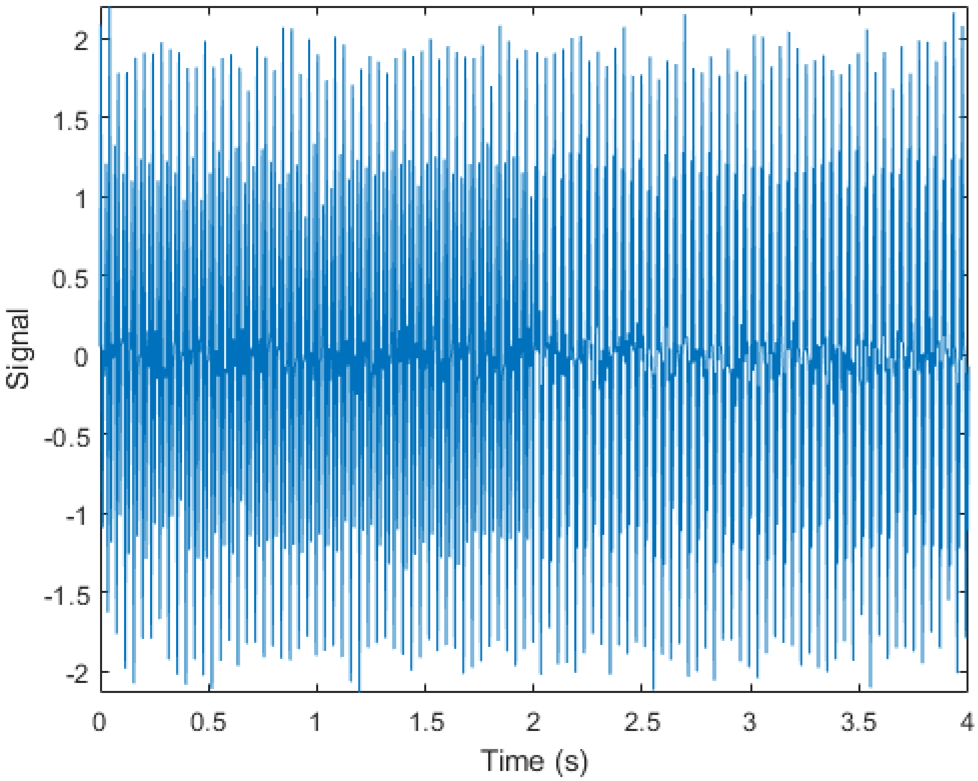
Examples

Perform Empirical Wavelet Transform and Visualize Hilbert Spectrum of Signal

Load and visualize a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The signal is sampled at 250 Hz.

```
fs = 250;
load nonstatdistinct
t = (0:length(nonstatdistinct)-1)/fs;
```

```
plot(t,nonstatdistinct)
xlabel('Time (s)')
ylabel('Signal')
axis tight
```

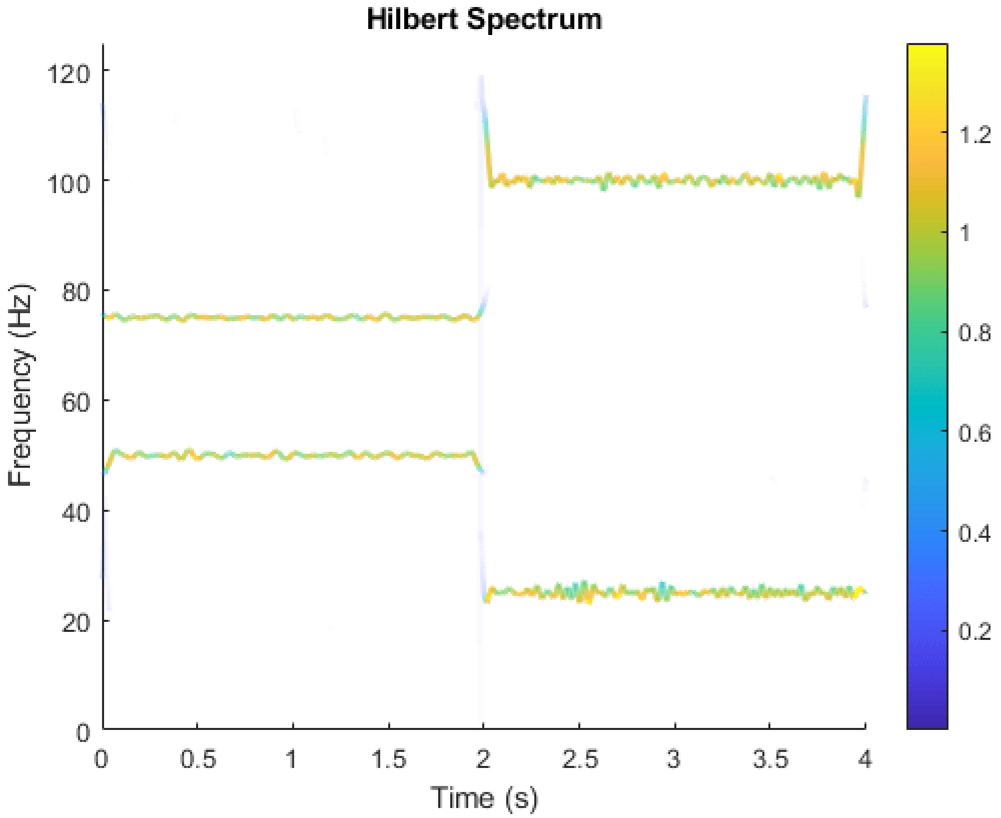


Use ewt to obtain a multiresolution analysis (MRA) of the signal.

```
mra = ewt(nonstatdistinct);
```

Use the MRA components with the hht function and plot the Hilbert spectrum.

```
hht(mra,fs)
```

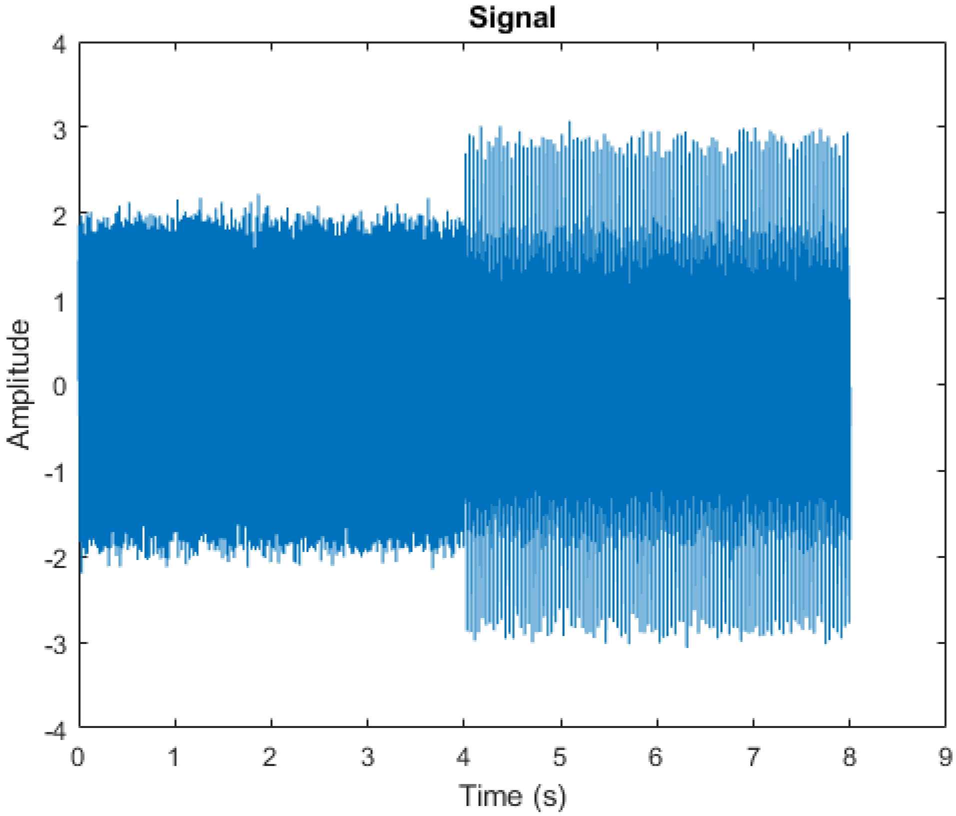


The frequency versus time plot is a sparse plot with a vertical color bar indicating the instantaneous energy at each point in the MRA. The plot represents the instantaneous frequency spectrum of each component decomposed from the original mixed signal.

Visualize Empirical Wavelet Transform Filter Bank

Create a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The signal is sampled at 1000 Hz.

```
Fs = 1000;
t = 0:1/Fs:4;
x1 = sin(2*pi*50*t) + sin(2*pi*200*t);
x2 = sin(2*pi*25*t) + sin(2*pi*100*t) + sin(2*pi*250*t);
x = [x1 x2] + 0.1*randn(1,length(t)*2);
t1 = (0:length(x)-1)/Fs;
plot(t1,x)
xlabel('Time (s)')
ylabel('Amplitude')
title('Signal')
```



Use ewt and obtain the MRA of the signal. Display the normalized peak frequencies identified in the signal, and the approximate frequency passbands of the filter bank. Because the frequencies are in cycles per sample, normalize by the sampling frequency. Note that the peak frequencies correspond to the frequencies of the sinusoidal waves.

```
[mra,~,wfb,info] = ewt(x);
Fs*info.PeakFrequencies
```

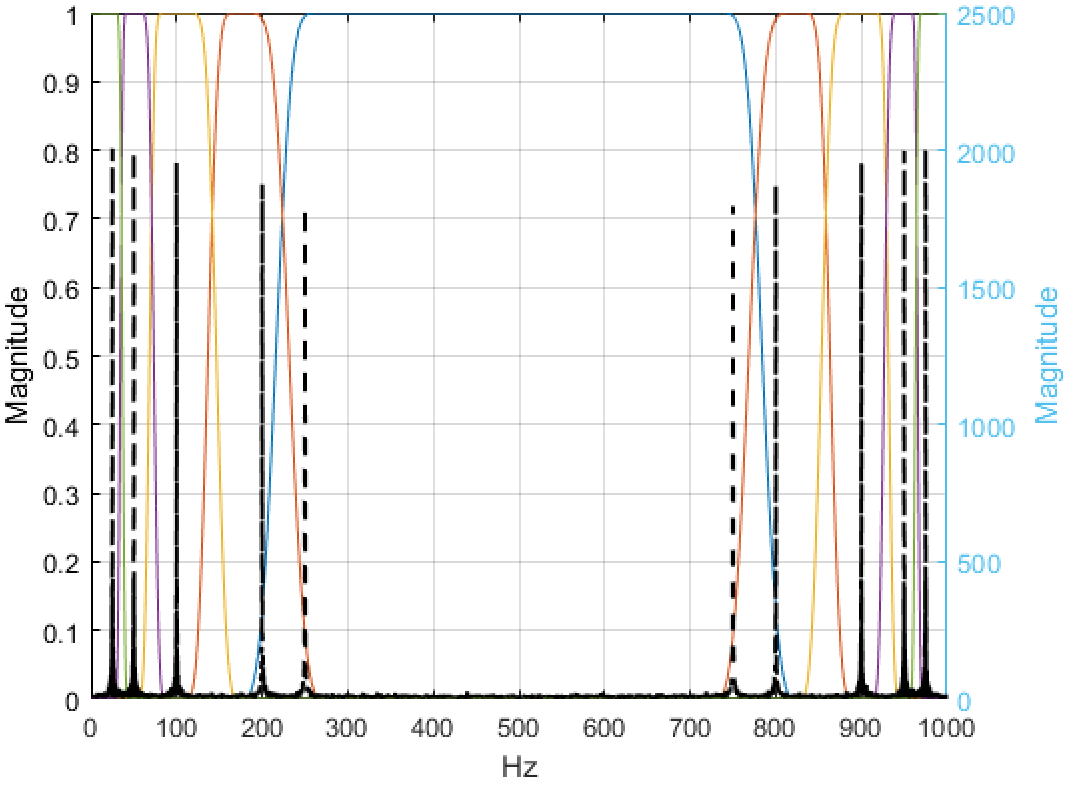
```
ans = 5×1

  249.9375
  200.0750
  100.1000
   50.1125
   25.1187
```

```
Fs*info.FilterBank.Passbands
```

```
ans = 5×2

  223.6941  500.0000
  141.5896  223.6941
   70.8573  141.5896
   35.4911   70.8573
         0   35.4911
```

Plot the magnitude spectrum of the signal, and the filter bank. The locations of the peaks determine the filter passbands.

```
f = 0:Fs/length(x):Fs-1/length(x);
plot(f,wfb)
ylabel('Magnitude')
grid on
yyaxis right
plot(f,abs(fft(x)),'k--','linewidth',1.5)
ylabel('Magnitude')
xlabel('Hz')
```



Because the empirical wavelets form a Parseval tight frame, the analysis filter bank is equal to the synthesis filter bank. Therefore, squaring the magnitudes at each frequency summed over the filters equals 1. If the sum was not equal to 1, perfect reconstruction would not be possible.

Empirical Wavelet Transform of ECG Signal

Load an ECG signal. The signal is sampled at 180 Hz.

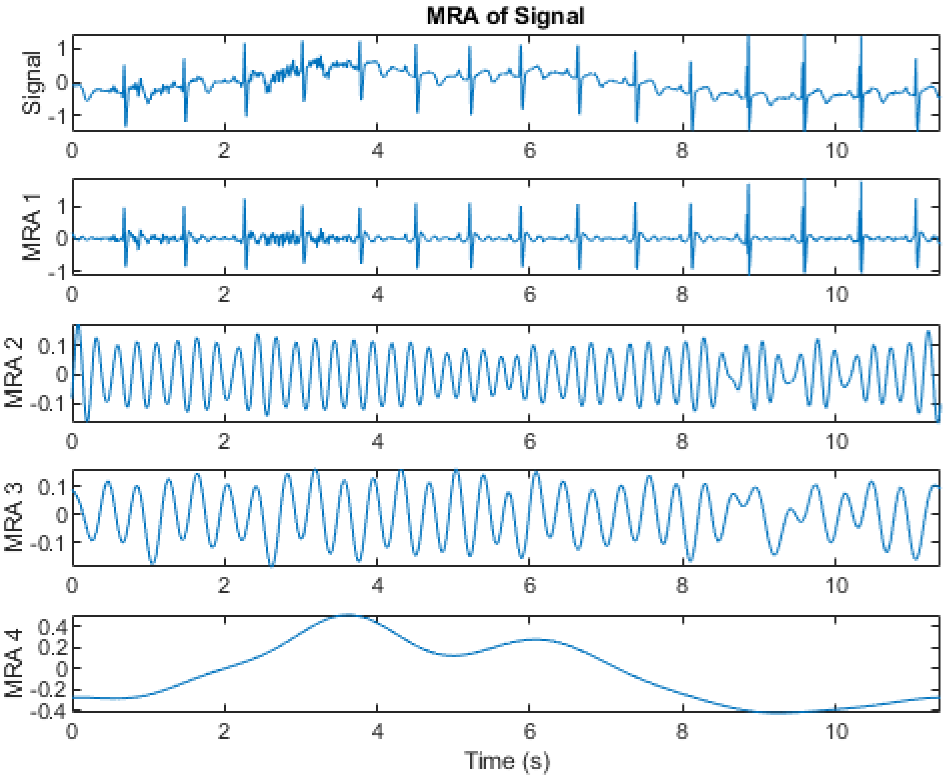
```
load wecg
```

Use `ewt` to obtain a multiresolution analysis (MRA) of the signal, and the corresponding analysis coefficients. Use the four largest peaks to determine the filter passbands.

```
mp = 4;
[mra,cfs] = ewt(wecg,'MaxNumPeaks',mp);
```

Plot the signal and the MRA components.

```
fs = 180;
subplot(mp+1,1,1)
t = (0:length(wecg)-1)/fs;
plot(t,wecg)
title('MRA of Signal')
ylabel('Signal')
axis tight
for k=1:mp
    subplot(mp+1,1,k+1)
    plot(t,mra(:,k))
    ylabel(['MRA ',num2str(k)])
    axis tight
end
xlabel('Time (s)')
```



Verify that summing the MRA components results in perfect reconstruction of the signal.

```
max(abs(wecg-sum(mra,2)))
```

```
ans = 8.8818e-16
```

Verify energy preservation of the EWT analysis coefficients.

```
cfsenergy = sum(sum(abs(cfs).^2));
[cfsenergy norm(wecg,2)^2]

ans = 1×2

  298.2759  298.2759
```

Input Arguments

x — Input data
vector | timetable

Input data, specified as a real- or complex-valued vector or as a single-variable timetable containing a single column vector. x must have at least two samples.

Data Types: single | double
Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: ewt(x,'MaxNumPeaks',5,'SegmentMethod','localmin') obtains the MRA of x using the five largest peaks and the first local minimum between adjacent peaks.

PeakThresholdPercent — Threshold percentage of maximum peak
70 (default) | real number in the interval (0, 100)

Threshold percentage of maximum peak used to determine which peaks to retain in the multitaper power spectrum of x, specified as a real number in the interval (0, 100). Local maxima in the multitaper power spectral estimate of x are normalized to lie in the range [0,1] with the maximum peak equal to 1. All peaks with values strictly greater than PeakThresholdPercent of the maximum peak are retained.

Data Types: single | double

SegmentMethod — Segmentation method
'geomean' (default) | 'localmin'

Segmentation method used to determine the EWT filter passbands, specified as:

- 'geomean' — Geometric mean of adjacent peaks
- 'localmin' — First local minimum between adjacent peaks

If no local minimum is identified between adjacent peaks, the function uses the geometric mean.

MaxNumPeaks — Maximum number of peaks
positive integer

Maximum number of peaks used to determine the EWT filter passbands. If `ewt` finds fewer peaks than the number specified in `MaxNumPeaks`, it uses the maximum number of peaks available. If it does not find any peaks, `ewt` uses a level-one discrete wavelet transform (DWT) filter bank.

You cannot specify both `MaxNumPeaks` and `PeakThresholdPercent`.

Data Types: `single` | `double`

`FrequencyResolution` — Frequency resolution bandwidth
real number less than or equal to `0.25`

Frequency resolution bandwidth of the multitaper power spectral estimate, specified as a real number less than or equal to 0.25.

The value of `FrequencyResolution` determines how many sine tapers are used in the multitaper power spectrum estimate. The bandwidth of a sine multitaper power spectral estimate is $(K+1)/(N+1)$, where K is the number of tapers and N is the length of the signal. The minimum value of `FrequencyResolution` is $2.5/N$, where N is the maximum of the signal length and 64.

Data Types: `single` | `double`

`LogSpectrum` — Logarithm of spectrum
`false` or `0` (default) | `true` or `1`

Logarithm of spectrum logical used to determine the peak frequencies. If `LogSpectrum` is set to `true`, the log of the multitaper power spectrum is used. Consider setting `LogSpectrum` to `true` if using the `PeakThresholdPercent` segmentation method and there is a dominant peak frequency that is significantly larger in magnitude than other peaks.

Output Arguments

`mra` — Multiresolution analysis
matrix | timetable

Multiresolution analysis (MRA), returned as a matrix or timetable.

- When `x` is a vector, `mra` is a matrix where each column stores an extracted MRA component.
    - For real-valued `x`, the MRA components are ordered by decreasing center frequencies. The final column in `mra` corresponds to the lowpass scaling filter.
    - For complex-valued `x`, the MRA components start near −½ cycles per sample and decrease in center frequency until the lowpass scaling coefficients are obtained. The frequency then increases toward +½ cycles per sample.
- When `x` is a timetable, `mra` is a timetable with multiple single variables where each variable stores an MRA component.

See the `info` structure array for a description of the frequency bounds for empirical wavelet and scaling filters.

If `x` has less than 64 samples, `ewt` works on a zero-padded version of `x` of length 64. The MRA components are truncated to the original length.

`cfs` — EWT analysis coefficients
matrix

EWT analysis coefficients, returned as a matrix. If the input data is real-valued, then `cfs` is a real-valued matrix. Otherwise, `cfs` is a complex-valued matrix. Each column of `cfs` stores the EWT analysis coefficients for the corresponding MRA component. The frequency bands of the analysis coefficients are identical to the ordering of the MRA components. If `x` has less than 64 samples, `cfs` contains the analysis coefficients obtained from the zero-padded version of `x`.

Data Types: `single` | `double`

`wfb` — Empirical wavelet filter bank
matrix

Empirical wavelet filter bank, returned as a matrix. The center frequencies of the filters in `wfb` match the order in `mra` and `cfs`. Because the empirical wavelets form a Parseval tight frame, the analysis filter bank is equal to the synthesis filter bank. Therefore, summing the MRA components results in perfect reconstruction of the signal.

Data Types: `single` | `double`

`info` — Filter bank information
structure array

Filter bank information, returned as a structure with the following fields:

- `PeakFrequencies` — The peak normalized frequencies in cycles/sample identified in `x` as a column vector. For real-valued `x`, the frequencies are positive in the interval (0, ½) in decreasing order. For complex-valued `x`, the frequencies are ordered from (−½, ½). If `PeakFrequencies` is empty, `ewt` did not find any peaks and a default one-level discrete wavelet transform (DWT) subdivision is used.
- `FilterBank` — A table with two variables: `MRAComponent` and `Passbands`. `MRAComponent` is the column index of the MRA component in `mra`. `Passbands` is a *L*-by-2 matrix where *L* is the number of MRA components. Each row of `Passbands` is the approximate frequency passband in cycles/sample for the corresponding EWT filter and MRA component.

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

References

[1] Gilles, Jérôme. “Empirical Wavelet Transform.” *IEEE Transactions on Signal Processing* 61, no. 16 (August 2013): 3999–4010. https://doi.org/10.1109/TSP.2013.2265222.

[2] Gilles, Jérôme, Giang Tran, and Stanley Osher. “2D Empirical Transforms. Wavelets, Ridgelets, and Curvelets Revisited.” *SIAM Journal on Imaging Sciences* 7, no. 1 (January 2014): 157–86. https://doi.org/10.1137/130923774.

[3] Gilles, Jérôme, and Kathryn Heal. “A Parameterless Scale-Space Approach to Find Meaningful Modes in Histograms — Application to Image and Spectrum Segmentation.” *International Journal of Wavelets, Multiresolution and Information Processing* 12, no. 06 (November 2014): 1450044. https://doi.org/10.1142/S0219691314500441.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Timetable input data is not supported.

See Also

Apps
Signal Multiresolution Analyzer

Functions
`emd` | `modwtmra` | `vmd` | `hht`

Topics
“Empirical Wavelet Transform”
“Practical Introduction to Multiresolution Analysis”
“Time-Frequency Gallery”

Introduced in R2020b

featureMatrix

Scattering feature matrix

Syntax

```
smat = featureMatrix(sf,x)
[smat,u] = featureMatrix(sf,x)
smat = featureMatrix( ___ ,Name,Value)
```

Description

`smat = featureMatrix(sf,x)` returns the scattering coefficient matrix for the wavelet time scattering network `sf` and the real-valued input data `x`. `x` is a vector, matrix, or 3-D array.

The precision of `smat` depends on the precision specified in the scattering network `sf`.

`[smat,u] = featureMatrix(sf,x)` returns the scalogram coefficients in the cell array of cell arrays, `u`. The number of elements in `u` is equal to the order of the scattering network. The *i*th element of `u` contains the scalogram coefficients for the (*i*-1)th order of the scattering coefficients.

`smat = featureMatrix( ___ ,Name,Value)` returns the scattering feature matrix with additional options specified by one or more `Name,Value` pair arguments.

Examples

Obtain Scattering Feature Matrix

This example shows how to obtain the scattering feature matrix for a wavelet time scattering network and how to compare the matrix with scattering coefficients.

Load an ECG signal sampled at 180 Hz. Create a wavelet time scattering network that can be used with the signal.

```
load wecg
Fs = 180;
sf = waveletScattering('SignalLength',numel(wecg),...
    'SamplingFrequency',Fs);
```

Calculate the scattering feature matrix using the log transformation. Display the dimensions of the matrix.

```
smat = featureMatrix(sf,wecg,'Transform','Log');
size(smat)
```

```
ans = 1×2

   147     8
```

Now calculate the scattering transform of the signal. Obtain the scattering coefficients. The output is a cell array with three elements. Each element is a table. Confirm the total number of rows in the tables is equal to the number of rows in the matrix.

```
S = scatteringTransform(sf,wecg);
t1rows = size(S{1},1);
t2rows = size(S{2},1);
t3rows = size(S{3},1);
disp(['Total Number of Rows: ',num2str(t1rows+t2rows+t3rows)])
```

```
Total Number of Rows: 147
```

Display the base-2 log resolution of the zeroth-order scattering coefficients.

```
disp(['Resolution: ',num2str(S{1}.resolution(1))])
```

```
Resolution: -8
```

Obtain the natural logarithm of the zeroth-order scattering coefficients. Compare the scattering coefficients with the first row in the feature matrix. The number of coefficients in each equals the absolute value of the base-2 log resolution.

```
logS = log(sf,S);
logScat = logS{1}.signals{1};
[smat(1,:)' logScat]
```

```
ans = 8×2

   -1.2914   -1.2914
   -2.4682   -2.4682
   -1.6368   -1.6368
   -1.2716   -1.2716
   -1.6818   -1.6818
   -4.3701   -4.3701
   -1.3199   -1.3199
   -1.0542   -1.0542
```

Input Arguments

sf — Wavelet time scattering network
waveletScattering object

Wavelet time scattering network, specified as a `waveletScattering` object.

x — Input data
vector | matrix | 3-D array

Input data, specified as a real-valued vector, matrix, or 3-D array. If `x` is a vector, the number of samples in `x` must equal the `SignalLength` value of `sf`. If `x` is a matrix or 3-D array, the number of rows in `x` must equal the `SignalLength` value of `sf`. If `x` is 2-D, the first dimension is assumed to be time and the columns of `x` are assumed to be separate channels. If `x` is 3-D, the dimensions of `x` are Time-by-Channel-by-Batch.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `smat = featureMatrix(sf,x,'Transform','log','Normalization','parent')`

`Normalization` — Type of normalization
`'none'` (default) | `'parent'`

Type of normalization to apply to the scattering coefficients, specified as `'none'` or `'parent'`. If specified as `'parent'`, scattering coefficients of order greater than 0 are normalized by their parents along the scattering path.

`Transform` — Type of transformation
`'none'` (default) | `'log'`

Type of transformation to apply to the scattering coefficients, specified as `'none'` or `'log'`.

Output Arguments

`smat` — Scattering coefficients
matrix | 3-D array | 4-D array

Scattering coefficients, returned as a real-valued matrix or array. If `x` is a vector, `smat` is an *Npath*-by-*Nscat* matrix, where *Npath* is the number of scattering paths and *Nscat* is the number of scattering coefficients in each path, or the resolution of the scattering coefficients. If `x` is a matrix, `smat` is *Npath*-by-*Nscat*-by-*Nchan*, where *Nchan* is the number of columns in `x`. If `x` is 3-D, then `smat` is *Npath*-by-*Nscat*-by-*Nchan*-by-*Nbatch*.

The precision of `smat` depends on the precision specified in the scattering network `sf`.

Data Types: `single` | `double`

`u` — Scalogram coefficients
cell array

Scalogram coefficients, returned in a cell array of cell arrays. The number of elements in `u` is equal to the order of the scattering network. The *i*th element of `u` contains the scalogram coefficients for the (*i*-1)th order of the scattering coefficients.

Note that `u{1}{1}` contains the original data.

Data Types: `single` | `double`

Tips

- The `scatteringTransform` function calls `featureMatrix` to generate the scattering and scalogram coefficients. If you only require the coefficients themselves, for improved performance the recommended approach is to use `featureMatrix`. Use `scatteringTransform` if you are also interested in the coefficients metadata.

Compatibility Considerations

featureMatrix function syntax will be deprecated

Not recommended starting in R2021a

One of the featureMatrix syntaxes will be deprecated in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
smat = featureMatrix(sf,s), where s are the scattering coefficients of real-valued data x	Still runs	smat = featureMatrix(sf,x)	Replace all instances of smat = featureMatrix(sf,s) with smat = featureMatrix(sf,x).

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The syntax smat = featureMatrix(sf,s), where s are the scattering coefficients of real-valued data x, is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Run MATLAB Functions on a GPU" (Parallel Computing Toolbox).

See Also

waveletScattering | scatteringTransform

Introduced in R2018b

featureMatrix

Image scattering feature matrix

Syntax

```
smat = featureMatrix(sf,im)
smat = featureMatrix(sf,sc)
smat = featureMatrix( ___ ,'Transform',transformtype)
```

Description

`smat = featureMatrix(sf,im)` returns the scattering feature matrix for the wavelet image scattering network, `sf`, and the input image, `im`. `im` is a real-valued 2-D (*M*-by-*N*) or 3-D matrix (*M*-by-*N*-by-3). If `im` is a 3-D matrix, the size of the third dimension must be 3. If `im` is a 2-D matrix, `smat` is *Np*-by-*Ms*-by-*Ns*, where *Np* is the number of scattering paths and *Ms*-by-*Ns* is the resolution of the scattering coefficients. If `im` is a 3-D matrix, `smat` is *Np*-by-*Ms*-by-*Ns*-by-3.

`smat = featureMatrix(sf,sc)` returns the scattering feature matrix for the cell array of scattering coefficients, `sc`. `sc` is obtained from the `scatteringTransform` method of the wavelet image scattering network.

`smat = featureMatrix( ___ ,'Transform',transformtype)` applies the transformation specified by `transformtype` to the scattering coefficients. Valid options for `transformtype` are `'log'` and `'none'`. If unspecified, `transformtype` defaults to `'none'`. You can use this syntax with any of the previous syntaxes.

Examples

Obtain Feature Matrix for Wavelet Image Scattering Network

This example shows how to obtain the feature matrix for a wavelet image scattering network.

Load the `xbox` image. Create an image scattering network suitable for the image.

```
load xbox
sf = waveletScattering2('ImageSize',size(xbox))
```

```
sf = 
  waveletScattering2 with properties:

              ImageSize: [128 128]
        InvarianceScale: 64
           NumRotations: [6 6]
         QualityFactors: [1 1]
              Precision: "single"
    OversamplingFactor: 0
           OptimizePath: 1
```

Obtain the feature matrix.

```
smat = featureMatrix(sf,xbox);
```

Input Arguments

`sf` — Wavelet image scattering network

`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

`im` — Input image

real-valued matrix

Input image, specified as real-valued 2-D matrix or 3-D matrix. If `im` is 3-D, `im` is assumed to be a color image in the RGB color space, and the size of the third dimension must equal 3. The row and column sizes of `im` must match the `ImageSize` property of `sf`.

`sc` — Scattering coefficients

cell array

Scattering coefficients, specified as a cell array. `sc` is obtained from the `scatteringTransform` method of the image scattering network.

`transformtype` — Transformation

`'none'` (default) | `'log'`

Transformation to apply to the scattering coefficients:

- `'none'`: No transformation is applied to the scattering coefficients.
- `'log'`: The natural logarithm is applied to the scattering coefficients.

Output Arguments

`smat` — Scattering feature matrix

real-valued array

Scattering feature matrix for the 2-D scattering network `sf`, returned as a real-valued array. If `im` is a 2-D matrix, `smat` is *Np*-by-*Ms*-by-*Ns*, where *Np* is the number of scattering paths and *Ms*-by-*Ns* is the resolution of the scattering coefficients. If `im` is a 3-D matrix, `smat` is *Np*-*Ms*-by-*Ns*-by-3.

See Also

`waveletScattering2` | `scatteringTransform`

Introduced in R2019a

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 ≤ *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))
```

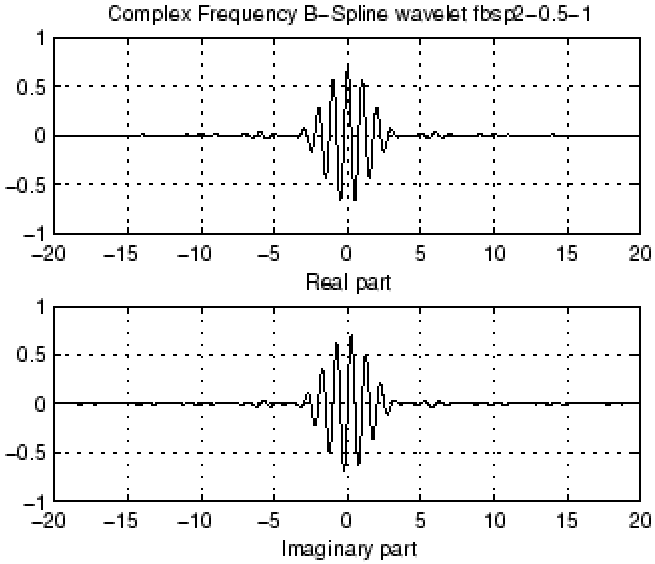
on an *N* point regular grid in the interval `[LB,UB]`.

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*, Birkhäuser, p. 63.

See Also

`waveinfo`

Introduced before R2006a

fejerkorovkin

Fejér-Korovkin wavelet filters

Syntax

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

Description

`Lo = fejerkorovkin(wname)` returns the Fejér-Korovkin scaling filter specified by `wname`. Valid entries for `wname` are `'fk4'`, `'fk6'`, `'fk8'`, `'fk14'`, `'fk18'`, and `'fk22'`. For information on the Fejér–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 = 1×14

    0.2604    0.6869    0.6116    0.0514   -0.2456   -0.0486    0.1243    0.0222   -0.0640   -0.0
```

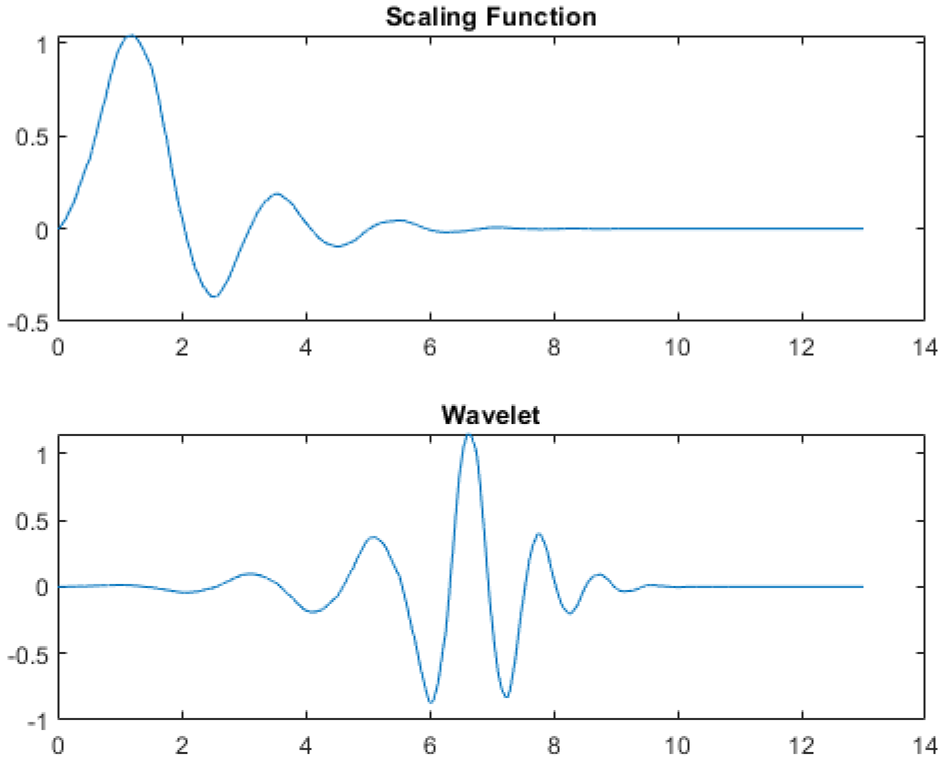
Use the scaling filter to obtain the wavelet filter and display its wavelet filter coefficients.

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

```
Hi = 1×14

    0.0035    0.0093   -0.0033   -0.0298   -0.0051    0.0640    0.0222   -0.1243   -0.0486    0.2
```

`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` — Name

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

Name of desired Fejér-Korovkin scaling filter. The numeric value in each name is the number of Fejér-Korovkin filter coefficients. For example, if `wname` is `'fk14'`, `Lo` has 14 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, Number 1, 2001, pp. 36–52.

See Also

`coifwavf` | `dbwavf` | `symwavf`

Introduced in R2015b

filt2ls

(To be removed) Transform quadruplet of filters to lifting scheme

Note `filt2ls` will be removed in a future release. Use `liftingScheme` instead. For more information, see “Compatibility Considerations”.

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]              []
};
```

Compatibility Considerations

`filt2ls` will be removed

Not recommended starting in R2021a

The `filt2ls` function will be removed in a future release. Set the `CustomLowpassFilter` property of `liftingScheme` instead.

See Also

`liftingScheme` | `ls2filt`

Introduced before R2006a

filterbank

Shearlet system filters

Syntax

```
psi = filterbank(sls)
[psi,scale] = filterbank(sls)
[psi,scale,shear] = filterbank(sls)
[psi,scale,shear,cone] = filterbank(sls)
```

Description

`psi = filterbank(sls)` returns the Fourier transforms of the shearlet filters defined by the shearlet system `sls` as a 3-D real-valued array. The array size is *M*-by-*N*-by-*K*, where *M* and *N* are the values of the two-element row vector “ImageSize” on page 1-0 of `sls`. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1.

`[psi,scale] = filterbank(sls)` returns the scale parameters of the shearlet filters as a *K*-by-1 integer vector. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1.

`[psi,scale,shear] = filterbank(sls)` returns the shearing parameters of the shearlet filters as a *K*-by-1 integer vector. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1.

`[psi,scale,shear,cone] = filterbank(sls)` returns the frequency cones of the shearlet filters as a *K*-by-1 cell array of characters. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1.

Examples

Obtain Shearlet Filters

Load an image. Create a shearlet system that can be used with the image.

```
load clown
sls = shearletSystem('ImageSize',size(X))
```

```
sls = 
  shearletSystem with properties:

         ImageSize: [200 320]
         NumScales: 4
    PreserveEnergy: 0
     TransformType: 'real'
    FilterBoundary: 'periodic'
         Precision: 'double'
```

Obtain the shearlet filters defined by the shearlet system.

```
psi = filterbank(sls);
```

Obtain the shearlet transform of the image.

```
cfs = sheart2(sls,X);
```

Confirm that the size of the third dimension of `psi` is equal to the size of the third dimension of `cfs`.

```
[size(psi,3) size(cfs,3)]
```

```
ans = 1×2

    41    41
```

Plot Shearlet Filters

Create a complex-valued shearlet system for 256-by-256 images with truncated boundaries and four scales.

```
sls = shearletSystem('TransformType','complex',...
    'ImageSize',[256 256],...
    'FilterBoundary','truncated',...
    'NumScales',4);
```

Obtain the shearlet filters and their geometric interpretations.

```
[psi,scale,shear,cone] = filterbank(sls);
```

Different scales can have a different number of shears. The scales in the shearlet system range from 0 to 3 inclusive. Find the range of shears per scale.

```
for k=0:3
    ind = find(scale==k);
    fprintf('scale: %d  shear range: [%d:%d]\n',...
        k,min(shear(ind)),max(shear(ind)))
end
```

```
scale: 0  shear range: [-1:1]
scale: 1  shear range: [-2:2]
scale: 2  shear range: [-2:2]
scale: 3  shear range: [-3:3]
```

Display the unique frequency cone labels.

```
unique(cone)
```

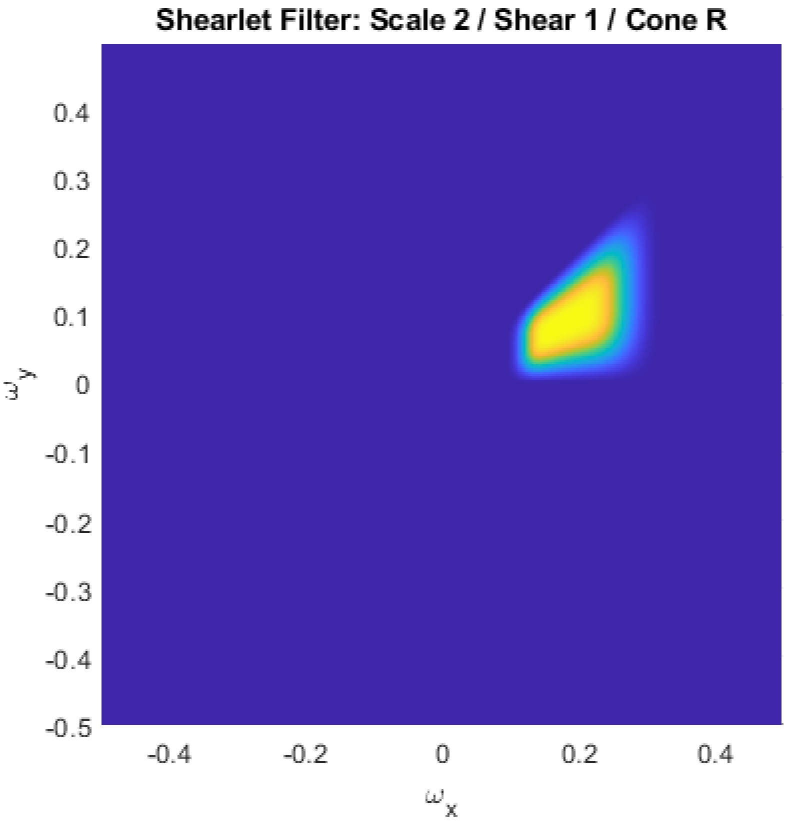
```
ans = 5x1 cell
    {'B'}
    {'L'}
    {'R'}
    {'T'}
    {'X'}
```

Find the shearlet filter at scale 2 with shear parameter equal to 1 and whose support is in the `'R'` frequency cone.

```
vScale = 2;
vShear = 1;
vCone = 'R';
ind = (scale'==vScale)&(shear'==vShear)&([cone{:}]==vCone);
shFilter = psi(:,:,ind);
```

Plot the shearlet filter.

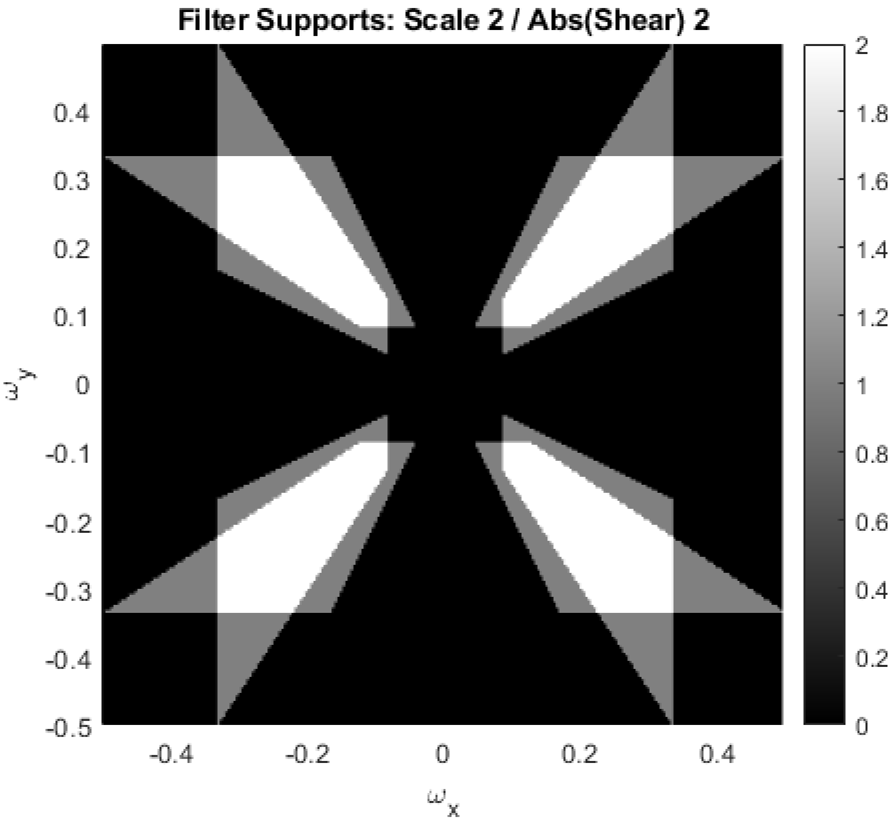
```
omegax = -1/2:1/256:1/2-1/256;
omegay = omegax;
surf(omegax,flip(omegay),shFilter,'EdgeColor','none')
view(0,90)
xlabel('\omega_x')
ylabel('\omega_y')
str = sprintf('Shearlet Filter: Scale %d / Shear %d / Cone %c',...
    vScale,vShear,vCone);
title(str)
axis equal
axis tight
```



Plot the supports of all scale 2 shearlet filters with shear parameters equal to ±2. Areas where filter supports overlap have maximum brightness.

```
vScale = 2;
vShear = 2;
ind = find((scale==vScale).*(abs(shear)==vShear));
tmp = zeros(size(psi,1),size(psi,2));
for k=1:length(ind)
```

```
    tmp = tmp+(psi(:,:,ind(k))~=0);
end
surf(omegax,flip(omegay),double(tmp),'EdgeColor','none')
view(0,90)
xlabel('\omega_x')
ylabel('\omega_y')
str = sprintf('Filter Supports: Scale %d / Abs(Shear) %d',...
    vScale,vShear);
title(str)
axis equal
axis tight
colormap gray
colorbar
```



Input Arguments

sls — Shearlet system

shearletSystem object

Shearlet system, specified as a shearletSystem object.

Output Arguments

psi — Fourier transforms of shearlet filters

real-valued 3-D array

Fourier transforms of the shearlet filters defined by the shearlet system, returned as a 3-D real-valued array. The array size is *M*-by-*N*-by-*K*, where *M* and *N* are the first and second elements, respectively, of the ImageSize value of `sls`. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1. The first element in the third dimension of `psi` corresponds to the lowpass filter. The subsequent elements correspond to the shearlets.

The data type of `psi` matches the Precision value of the shearlet system.

Data Types: `single` | `double`

`scale` — Scale parameters
integer vector

Scale parameters of the shearlet filters defined by the shearlet system, returned as a *K*-by-1 integer vector. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1. The first element of `scale`, −1, corresponds to the lowpass filter. The remaining elements of `scale` correspond to the shearlet filters.

The data type of `scale` matches the Precision value of the shearlet system.

Data Types: `single` | `double`

`shear` — Shearing parameters
integer vector

Shearing parameters of the shearlet filters defined by the shearlet system, returned as a *K*-by-1 integer vector. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1. The shear values at scale *S* range from `−ceil(2^(S/2))` to `ceil(2^(S/2))` inclusive. The first element of `shear`, 0, corresponds to the lowpass filter. The remaining elements of `shear` denote the shears for the corresponding shearlet filters.

The data type of `shear` matches the Precision value of the shearlet system.

Data Types: `single` | `double`

`cone` — Frequency cones
cell array of characters

Frequency cones of the shearlet filters defined by the shearlet system, returned as a *K*-by-1 cell array of characters. *K* is the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1. The first element of `cone`, `'X'`, corresponds to the lowpass filter. The remaining elements of `cone` are the frequency cones in which the corresponding shearlet filters have their frequency support.

For complex-valued shearlets, the frequency plane is divided into four cones: `'R'` (right), `'T'` (top), `'L'` (left), and `'B'` (bottom). For real-valued shearlets, the frequency cones are `'H'` (horizontal) and `'V'` (vertical).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`shearletSystem` | `numshears`

Introduced in R2019b

filterbank

Wavelet time scattering filter banks

Syntax

```
filters = filterbank(sf)
[filters,f] = filterbank(sf)
[filters,f,filterparams] = filterbank(sf)
[___] = filterbank(sf,order)
```

Description

`filters = filterbank(sf)` returns the filter banks used in the computation of the scattering coefficients. `filters` is a cell array of structure arrays with *norder* elements, where *norder* is the number of scattering orders. The first element of `filters` contains the scaling filter, `phift`, used in the computation of the 0th-order scattering coefficients. Subsequent elements of `filters` contain the wavelet filters, `psift`, and scaling filter, `phift`, for the corresponding filter banks of the scattering decomposition.

The precision of `phift` and `psift` depends on the precision specified in the scattering network `sf`.

`[filters,f] = filterbank(sf)` returns the frequencies corresponding to the DFT bins in the `psift` and `phift` fields of `filters`. If you specify a sampling frequency in the construction of `sf`, `f` is measured in hertz. Otherwise, `f` is measured in cycles/sample.

`[filters,f,filterparams] = filterbank(sf)` returns the filter parameters for each element of `filters`. `filterparams` is a cell array with *norder* elements. Each element of `filterparams` is a MATLAB table.

`[___] = filterbank(sf,order)` returns the filter banks used to compute the specified `order` scattering coefficients. `order` is an integer between 0 and *nfilters* inclusive, where *nfilters* is the number of filter banks in the scattering network. These input arguments can be used with any of the output syntaxes shown previously.

Examples

Plot Scattering Network Filter Banks

Create a wavelet time scattering network for a signal sampled at 25 Hz.

```
sf = waveletScattering('SamplingFrequency',25)
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 20.4800
        QualityFactors: [8 1]
              Boundary: 'periodic'
```

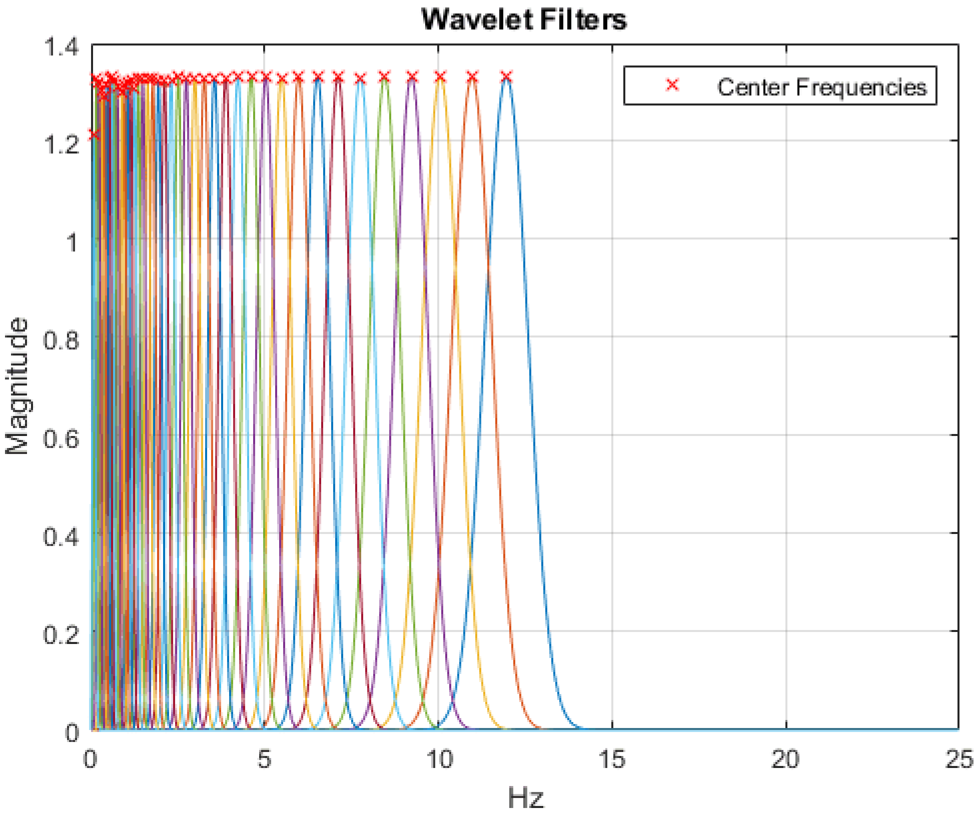
```
      SamplingFrequency: 25
              Precision: 'double'
     OversamplingFactor: 0
           OptimizePath: 0
```

Obtain the filter banks, DFT frequency bins, and filter bank parameters.

```
[filters,f,fparams] = filterbank(sf);
```

Plot the wavelet filters used in computing the first-order coefficients. Plot the wavelet center frequencies as well.

```
coefOrder = 1;
wvFilters = filters{coefOrder+1}.psift;
wvcenFrq = fparams{coefOrder+1}.omegapsi;
plot(f,wvFilters)
hold on
cf = plot(wvcenFrq,max(wvFilters),'rx');
grid on
title('Wavelet Filters')
xlabel('Hz')
ylabel('Magnitude')
legend(cf,'Center Frequencies')
```



Input Arguments

`sf` — Wavelet time scattering network
`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

`order` — Order of scattering coefficients
positive integer

Order of scattering coefficients, specified as a positive integer between 0 and *nfilters* inclusive, where *nfilters* is the number of filter banks in the scattering decomposition `sf`.

Data Types: `double`

Output Arguments

`filters` — Filter banks
cell array

Filter banks using in the computation of the scattering coefficients, returned as a cell array of structure arrays. `filters` has *norder* elements, where *norder* is the number of scattering orders. The first element of `filters` is a structure with the single field `phift`. `phift` contains the scaling filter used in the computation of the 0th-order scattering coefficients. Subsequent elements of `filters` contain the wavelet filters, `psift`, and the scaling filter, `phift`, for the corresponding filter banks of the scattering network in the structure fields.

The precision of `phift` and `psift` depends on the precision specified in the scattering network `sf`.

`f` — Frequencies
real-valued vector

Frequencies corresponding to the DFT bins in the `psift` and `phift` fields of `filters`. If you specify a sampling frequency in the construction of `sf`, `f` is measured in hertz. Otherwise, `f` is measured in cycles/sample.

Data Types: `double`

`filterparams` — Filter bank parameters
cell array

Filter bank parameters for each element of `filters`, returned as a cell array. `filterparams` has *norder* elements, where *norder* is the number of scattering orders.

The first element of `filterparams` is a MATLAB table with the following variables:

- `boundary` — The signal extension used in the filters, returned as either `'periodic'` or `'reflection'`.
- `precision` — The precision used in the filters, returned as `'double'` or `'single'`.
- `sigmaphi` — The time standard deviation of the scaling function, returned as a scalar. If you specify a sampling frequency, `sigmaphi` is in seconds. Otherwise, `sigmaphi` is in samples.
- `freqsigmaphi` — The frequency standard deviation of the scaling function, returned as a scalar. If you specify a sampling frequency, `freqsigmaphi` is in hertz. Otherwise, `freqsigmaphi` is in cycles/sample.

- `phiftsupport` — The frequency support of the scaling function, returned as a scalar. If you specify a sampling frequency, `phiftsupport` is in hertz. Otherwise, `phiftsupport` is in cycles/sample.
- `phi3dBbw` — The 3-dB bandwidth of the scaling function, returned as a scalar.

Subsequent elements of `filterparams` include additional variables for the wavelet parameters:

- `J` — The integer number of logarithmically spaced wavelet filters in the scattering filter bank.
- `omegapsi` — The center frequencies for the wavelet filters in descending order (highest to lowest), returned as a vector. The `omegapsi` variable includes the center frequencies for any linearly spaced filters.
- `freqsigmapsi` — The wavelet frequency standard deviations, returned as a vector.
- `timesigmapsi` — The wavelet time standard deviations, returned as a vector.
- `psi3dBbw` — The wavelet 3-dB bandwidths, returned as a vector.
- `psiftsupport` — The wavelet frequency supports, returned as a vector.

Extended Capabilities

C/C++ Code Generation

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

See Also

`waveletScattering` | `littlewoodPaleySum`

Introduced in R2018b

filterbank

Wavelet and scaling filters

Syntax

```
phif = filterbank(sf)
[phif,psifilters] = filterbank(sf)
[phif,psifilters,f] = filterbank(sf)
[phif,psifilters,f,filterparams] = filterbank(sf)
[ ___ ] = filterbank(sf,fb)
```

Description

`phif = filterbank(sf)` returns the Fourier transform of the scaling filter for the 2-D wavelet scattering network, `sf`. `phif` is a single or double-precision matrix depending on the value of the `Precision` property of the scattering network. `phif` has dimensions *M*-by-*N*, where *M* and *N* are the padded row and column sizes of the scattering network.

`[phif,psifilters] = filterbank(sf)` returns the Fourier transforms for the wavelet filters in `psifilters`. `psifilters` is an *Nfb*-by-1 cell array, where *Nfb* is the number of filter banks in the scattering network. Each element of `psifilters` is a 3-D array. The 3-D arrays are *M*-by-*N*-by-*L*, where *M* and *N* are the padded row and column sizes of the wavelet filters and *L* is the number of wavelet filters for each filter bank. The wavelet filters are ordered by increasing scale with `NumRotations` wavelet filters for each scale. Within a scale, the wavelet filters are ordered by rotation angle.

`[phif,psifilters,f] = filterbank(sf)` returns the center spatial frequencies for the wavelet filters in `psifilters`. `f` is an *Nfb*-by-1 cell array, where *Nfb* is the number of filter banks in `sf`. The *j*th element of `f` contains the center frequencies for the *j*th wavelet filter bank in `psifilters`. Each element of `f` is a *L*-by-2 matrix with each row containing the center frequencies of the corresponding *L*th wavelet.

`[phif,psifilters,f,filterparams] = filterbank(sf)` returns the filter parameters for the 2-D scattering network. `filterparams` is an *Nfb*-by-1 cell array of MATLAB tables, where the *j*th element of `filterparams` is a MATLAB table containing the filter parameters for the *j*th filter bank

`[ ___ ] = filterbank(sf,fb)` returns the desired outputs for the filter banks specified in `fb`. `fb` is a scalar or vector of integers between 1 and `numfilterbanks(sf)` inclusive. If `fb` is a scalar, `psifilters` is an *M*-by-*N*-by-*L* matrix, and `filterparams` is a MATLAB table.

Examples

Plot Wavelet Center Frequencies

This example shows how to plot the scaling filter and the wavelet filter center frequencies for a two-filter bank wavelet image scattering network.

Create a wavelet image scattering network with two filter banks. The first filter bank has a quality factor of 2, and the second filter bank has a quality factor of 1.

```
sf = waveletScattering2('QualityFactors',[2 1])
```

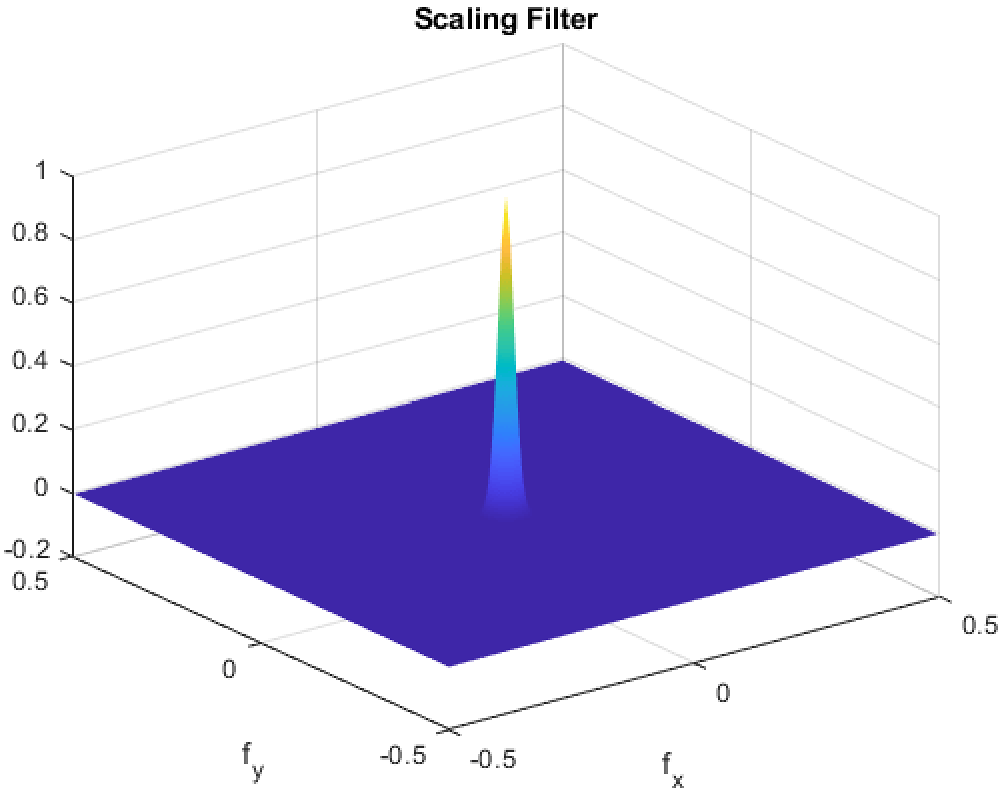
```
sf = 
  waveletScattering2 with properties:

             ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [6 6]
        QualityFactors: [2 1]
             Precision: "single"
    OversamplingFactor: 0
          OptimizePath: 1
```

Obtain the scaling filter, wavelet filters, and wavelet center frequencies for the network.

```
[phif,psifilters,f] = filterbank(sf);
```

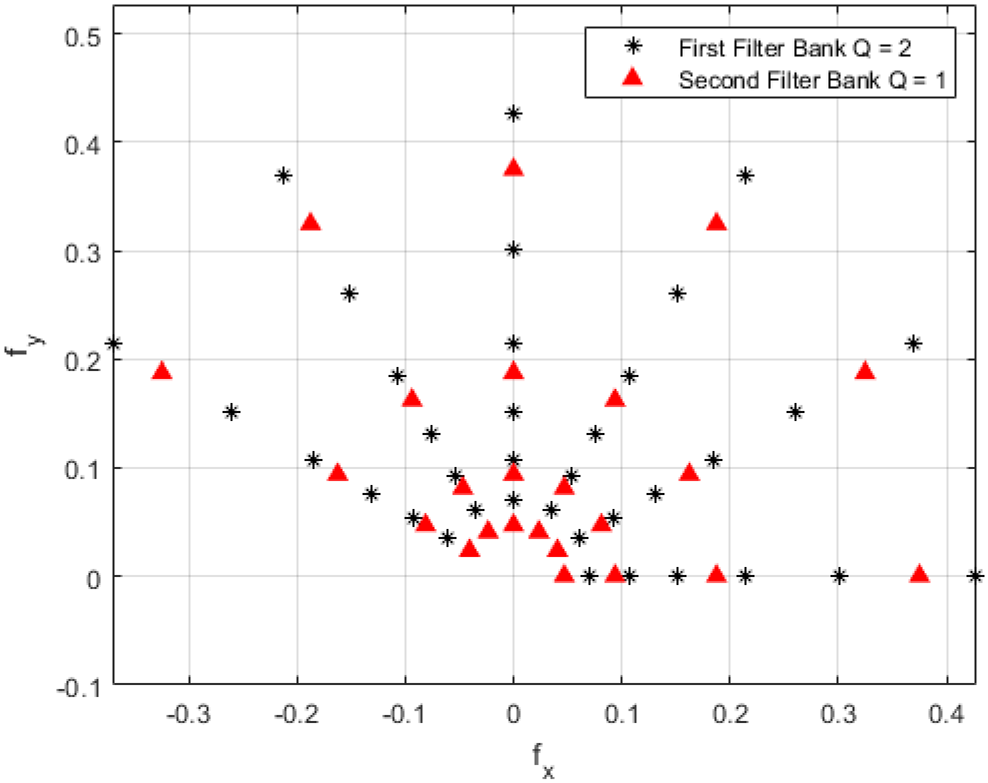
Make a surface plot of the scaling filter.

```
Nx = size(phif,1);
Ny = size(phif,2);
fx = -1/2:1/Nx:1/2-1/Nx;
fy = -1/2:1/Ny:1/2-1/Ny;
surf(fx,fy,fftshift(phif))
shading interp
title('Scaling Filter')
xlabel('f_x')
ylabel('f_y')
```



Plot the wavelet center frequencies for the two filter banks.

```
figure
plot(f{1}(:,1),f{1}(:,2),'k*')
hold on
grid on
plot(f{2}(:,1),f{2}(:,2),'r^','MarkerFaceColor',[1 0 0])
axis equal
xlabel('f_x')
ylabel('f_y')
legend('First Filter Bank Q = 2','Second Filter Bank Q = 1')
```



Plot Specific Wavelet of Image Scattering Network

This example shows how to obtain and plot a specific wavelet of a wavelet image scattering network.

Create a wavelet image scattering network with two filter banks. The first filter bank has a quality factor of 2 and seven rotations per wavelet. The second filter bank has a quality factor of 1 and five rotations per wavelet.

```
sf = waveletScattering2('QualityFactors',[2 1],'NumRotations',[7 5])
```

```
sf = 
  waveletScattering2 with properties:

              ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [7 5]
        QualityFactors: [2 1]
             Precision: "single"
    OversamplingFactor: 0
          OptimizePath: 1
```

Obtain the wavelet filters and center frequencies for the network. Return the dimensions of the two cell arrays.

```
[~,psifilters,f] = filterbank(sf);
psifilters
```

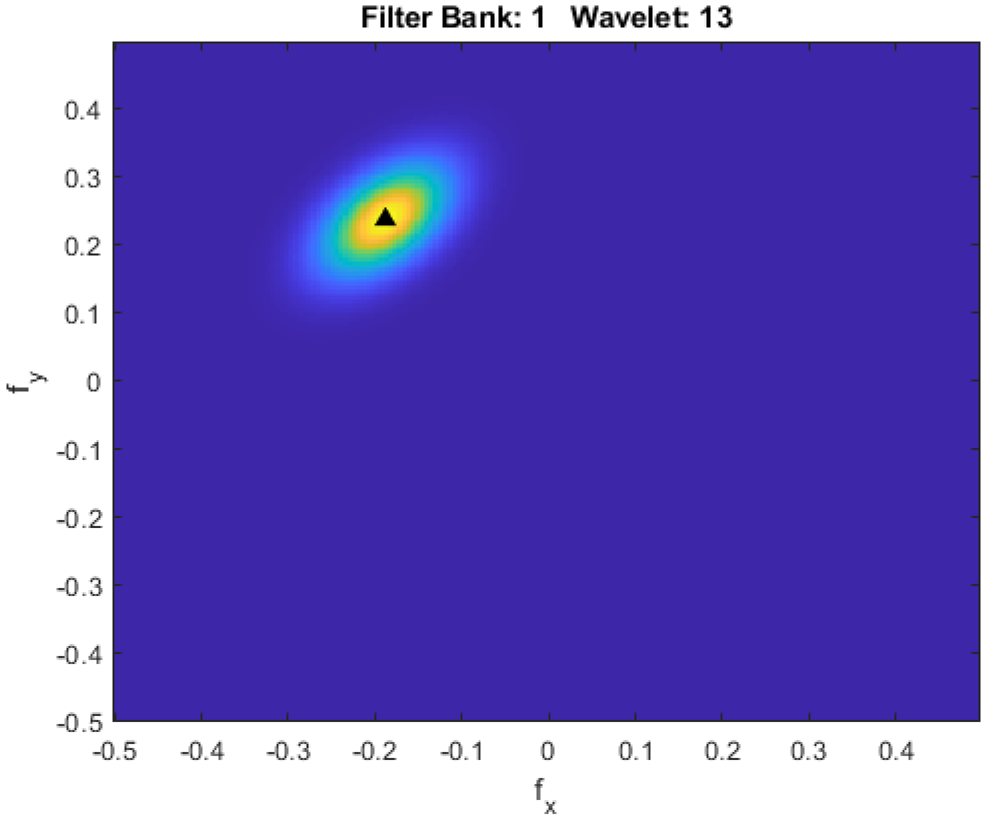
```
psifilters=2×1 cell array
    {192x192x42 single}
    {192x192x20 single}
```

```
f
```

```
f=2×1 cell array
    {42x2 double}
    {20x2 double}
```

The first filter bank has 42 wavelet filters, and the second filter bank has 20 filters. The number of filters in each filter bank is a multiple of the corresponding value in `NumRotations`. Use the helper function `helperPlotWavelet` to plot a specific wavelet and mark its center frequency.

```
figure
whichFilterBank = 1;
whichWavelet = 13;
helperPlotWavelet(psifilters,f,whichFilterBank,whichWavelet)
```



Appendix

The following helper function is used in this example.

```
function helperPlotWavelet(psiFilters,psiFreq,filBank,wvFilter)
Nx = size(psiFilters{filBank},2);
Ny = size(psiFilters{filBank},1);
fx = -1/2:1/Nx:1/2-1/Nx;
fy = -1/2:1/Ny:1/2-1/Ny;
imagesc(fx,fy,fftshift(psiFilters{filBank}(:,:,wvFilter)))
axis xy
hold on
xlabel('f_x')
ylabel('f_y')
plot(psiFreq{filBank}(wvFilter,1),psiFreq{filBank}(wvFilter,2),...
    'k^','markerfacecolor',[0 0 0])
str = sprintf('Filter Bank: %d   Wavelet: %d',filBank,wvFilter);
title(str)
end
```

Determine Wavelet Semi-Major Axis

This example shows how to determine the semi-major axis of a wavelet filter in a 2-D wavelet scattering network.

Create a 2-D wavelet scattering network. The network has two filter banks with quality factors of 2 and 1, respectively. There are seven rotations per wavelet in the first filter bank and five rotations per wavelet in the second filter bank. Return the Fourier transforms of the wavelet filters and their center spatial frequencies, and the filter bank parameters.

```
sf = waveletScattering2('QualityFactors',[2 1],'NumRotations',[7 5]);
[~,psif,f,fparams] = filterbank(sf);
```

The wavelet filters in `psif` are ordered by increasing scale, with `NumRotations` wavelet filters for each scale. Within a scale, the wavelet filters are ordered by rotation.

Return the reported 3 dB bandwidths, rotation angles, and slant parameter of the first filter bank. Return the dimensions of the matrix containing the wavelet filters of the first filter bank. Confirm that (number of rotations) × (number of bandwidths) equals the size of the third dimension of the matrix. The product is the number of wavelet filters in the filter bank. The row and column sizes are the dimensions of the padded wavelet filters. Note that the slant parameter is less than 1.

```
fparams{1}.psi3dBbw
```

```
ans = 1×6

    0.1464    0.1036    0.0732    0.0518    0.0366    0.0366
```

```
fparams{1}.rotations
```

```
ans = 1×7

         0    0.4488    0.8976    1.3464    1.7952    2.2440    2.6928
```

```
fparams{1}.slant
```

```
ans = 0.5817
```

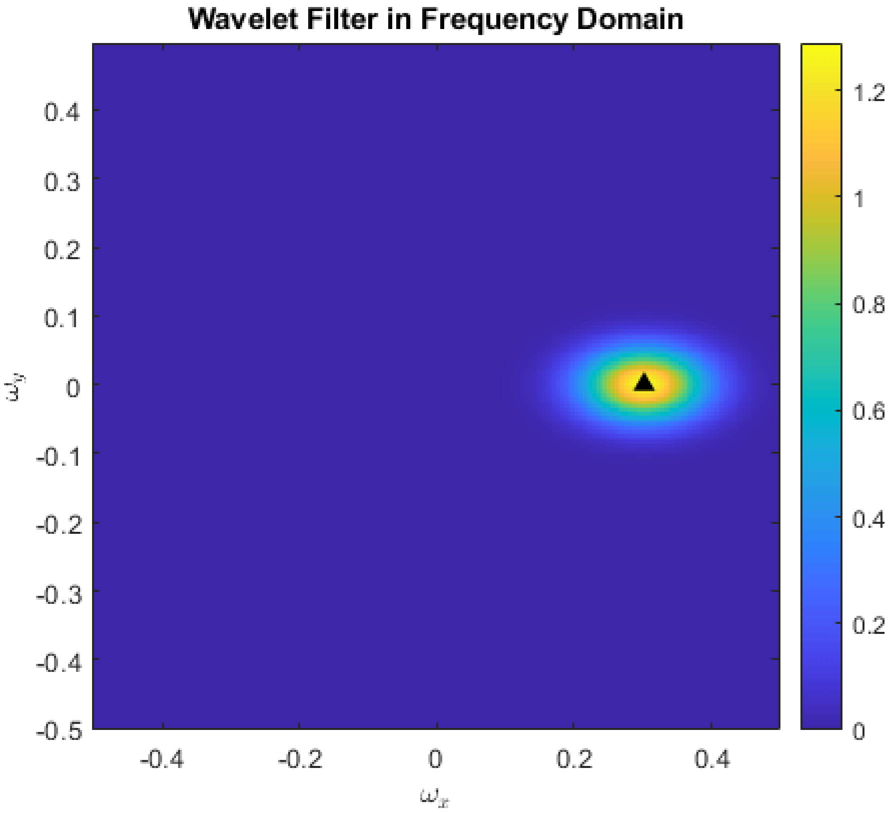
```
size(psif{1})
```

```
ans = 1×3

   192   192    42
```

The vector `fparams{1}.psi3dBbw` has six elements. The number of elements is equal to the number of wavelet scales in the first filter bank.

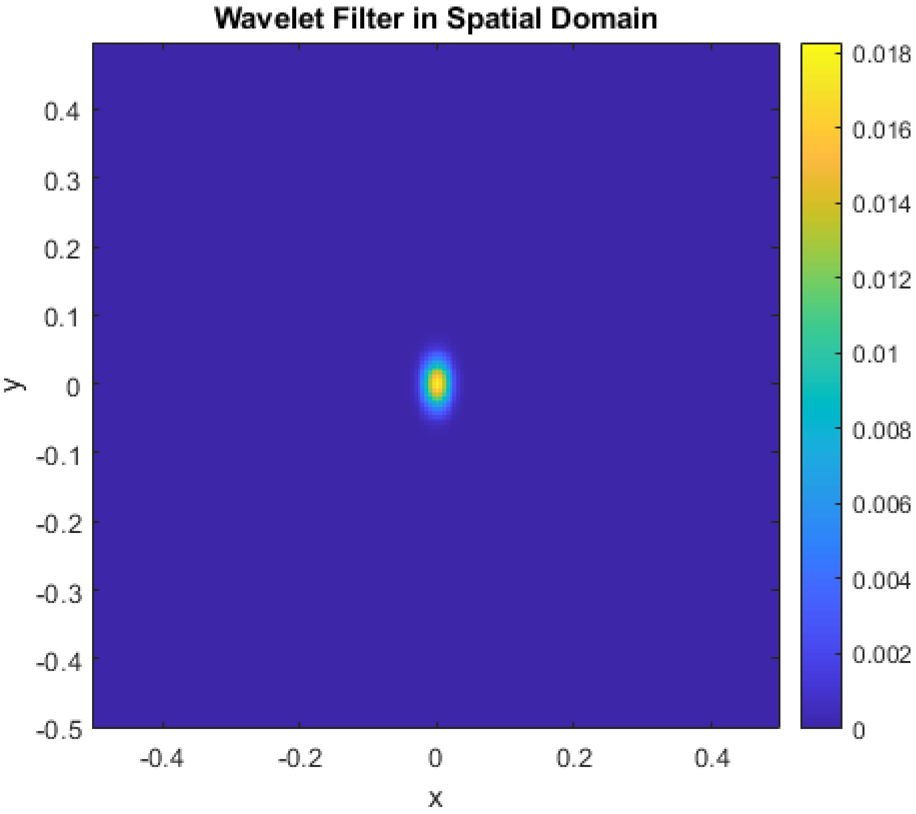
From the first filter bank, obtain the unrotated wavelet filter from the second finest scale. Obtain the center spatial frequency of the wavelet. Use the helper function `helperPlotWaveletFT` to plot the wavelet and mark its center frequency. The code for `helperPlotWaveletFT` is shown at the end of this example.

```
whichFilterBank = 1;
whichScale = 2;
whichRotAngle = 1;
numRot = sf.NumRotations(whichFilterBank);

wvf = psif{whichFilterBank}(:,:,1+(whichScale-1)*numRot+(whichRotAngle-1));
wvfCenFrq = f{whichFilterBank}(1+(whichScale-1)*numRot+(whichRotAngle-1),:);

helperPlotWaveletFT(wvf,wvfCenFrq)
```



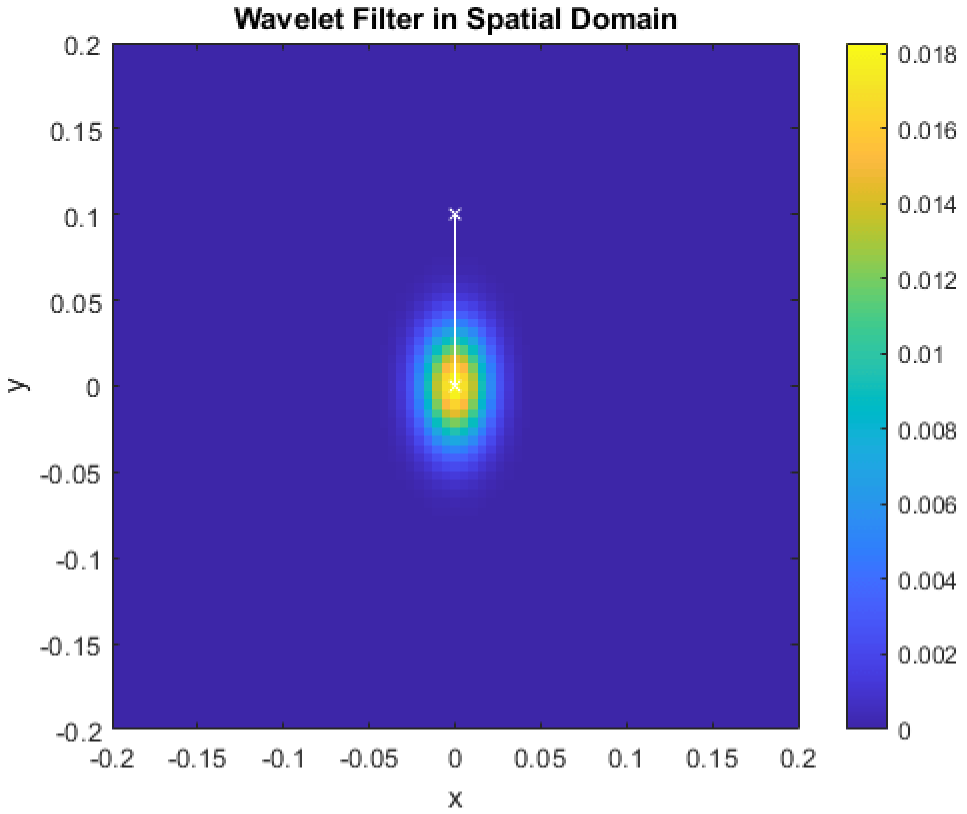
Take the inverse Fourier transform of the wavelet filter. The filter is strictly real, and the inverse Fourier transform is complex-valued. Use the helper function `helperPlotWavelet` to plot the absolute value of the wavelet. The code for `helperPlotWavelet` is shown at the end of this example.

```
wvf_ifft = ifft2(wvf);
figure
helperPlotWavelet(wvf_ifft)
```



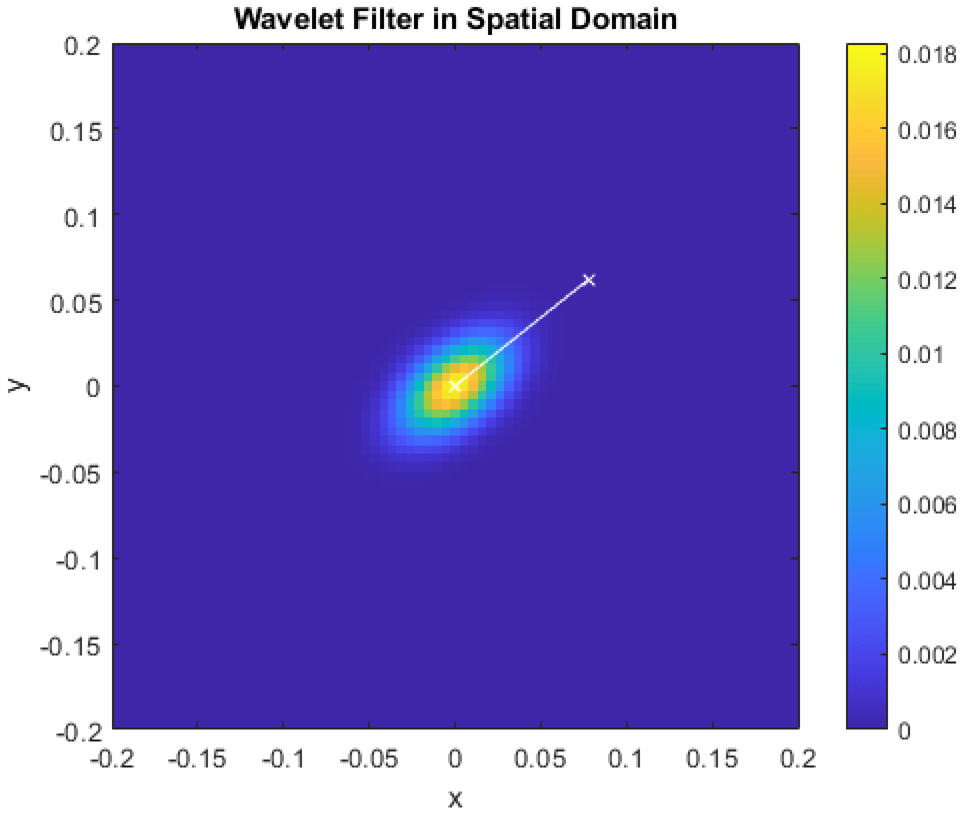
Note that the semi-major axis of the wavelet support is in the *y*-direction. This is consistent with a slant parameter whose value is less than 1. The vector `(0,0.1)` coincides with the semi-major axis. Plot the vector in the previous figure.

```
vec = [0 0.1];
hold on
plot([0 vec(1)],[0 vec(2)],'wx-')
xlim([-0.2 0.2])
ylim([-0.2 0.2])
```



From the same filter bank and scale, choose a rotated wavelet filter. Plot the absolute value of the wavelet in the spatial domain. Use the associated rotation angle in `fparams{1}.rotations`, and rotate clockwise the vector `(0,0.1)` by that amount. Plot the rotated vector in the figure. Confirm that the vector is aligned with the semi-major axis of the rotated wavelet.

```
whichRotAngle = 3;
rotAngle = fparams{whichFilterBank}.rotations(whichRotAngle);
rmat = [cos(rotAngle) sin(rotAngle) ; -sin(rotAngle) cos(rotAngle)];

wvf = psif{whichFilterBank}(:,:,1+(whichScale-1)*numRot+(whichRotAngle-1));
wvf_ifft = ifft2(wvf);

rvec = rmat*vec';

figure
helperPlotWavelet(wvf_ifft)
hold on
plot([0 rvec(1)],[0 rvec(2)],'wx-')
xlim([-0.2 0.2])
ylim([-0.2 0.2])
```



Appendix

The following helper functions are used in this example.

helperPlotWaveletFT — Plot in the frequency domain

```
function helperPlotWaveletFT(wavelet,cenFreq)
Nx = size(wavelet,2);
Ny = size(wavelet,1);
fx = -1/2:1/Nx:1/2-1/Nx;
fy = -1/2:1/Ny:1/2-1/Ny;
imagesc(fx,fy,fftshift(wavelet))
colorbar
axis xy
hold on
xlabel('$\omega_x$','Interpreter',"latex")
ylabel('$\omega_y$','Interpreter',"latex")
axis equal
axis tight
plot(cenFreq(1),cenFreq(2),'k^','markerfacecolor',[0 0 0])
title('Wavelet Filter in Frequency Domain')
end
```

helperPlotWavelet — Plot in the spatial domain

```
function helperPlotWavelet(wavelet)
Nx = size(wavelet,2);
Ny = size(wavelet,1);
```

```
fx = -1/2:1/Nx:1/2-1/Nx;
fy = -1/2:1/Ny:1/2-1/Ny;
imagesc(fx,fy,abs(fftshift(wavelet)))
colorbar
axis xy
hold on
xlabel('x')
ylabel('y')
axis equal
axis tight
title('Wavelet Filter in Spatial Domain')
end
```

Input Arguments

`sf` — Wavelet image scattering network

`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

`fb` — Filter banks

positive integer | vector of positive integers

Filter banks, specified as an integer or vector of integers between 1 and `numfilterbanks(sf)` inclusive. If `fb` is a scalar, `psifilters` is an *M*-by-*N*-by-*L* matrix and `filterparams` is a MATLAB table.

Output Arguments

`phif` — Fourier transform of scaling filter

real-valued matrix

Fourier transform of scaling filter, returned as a real-valued 2-D matrix. The precision of `phif` depends on the value of the `Precision` property of the scattering network. `phif` has dimensions *M*-by-*N*, where *M* and *N* are the padded row and column sizes of the scattering network.

`psifilters` — Fourier transforms of the wavelet filters

cell array

Fourier transforms of the wavelet filters, returned as an *Nfb*-by-1 cell array, where *Nfb* is the number of filter banks in the scattering network. Each element of `psifilters` is a 3-D array. The 3-D arrays are *M*-by-*N*-by-*L*, where *M* and *N* are the padded row and column sizes of the wavelet filters and *L* is the number of wavelet filters for each filter bank. The wavelet filters are ordered by increasing scale with `NumRotations` wavelet filters for each scale.

Example: Note that `size(psifilters,3)` is equal to `size(f,1)`.

`f` — Center spatial frequencies

cell array

Center spatial frequencies of the wavelet filters, returned as a *Nfb*-by-1 cell array where *Nfb* is the number of filter banks in the scattering network. The *j*th element of `f` contains the center frequencies for the *j*th wavelet filter bank in `psifilters`. Each element of `f` is an *L*-by-2 matrix with each row containing the center frequencies of the corresponding *L*th wavelet. The spatial frequencies are in cycles per pixel.

filterparams — Filter parameters

cell array

Filter parameters for the 2-D scattering network, sf. filterparams is an *Nfb*-by-1 cell array of MATLAB tables, where the *j*th element of filterparams is a MATLAB table containing the filter parameters for the *j*th filter bank. Each table contains these variables:

- Q — The quality factor of the filter bank, returned as an integer.
- J — The highest factor used in the dilation of the Morlet wavelets, $2^{J/Q}$, returned as an integer.
- precision — The precision of the scattering network, returned as 'single' or 'double'.
- omegapsi — The wavelet center frequencies in descending order (highest to lowest), returned as a vector.
- freqsigmapsi — The wavelet frequency standard deviations, returned as a vector.
- slant — The slant parameter for the spatial vertical semi-major axis of the wavelet, returned as a real number. The slant parameter, also known as the spatial aspect ratio, characterizes the shape of the support of the wavelet.
- spatialsigmapsi — The wavelet spatial standard deviations, returned as a vector.
- spatialsigmaphi — The scaling filter spatial standard deviation, returned as a real number.
- psi3dBbw — The wavelet 3 dB bandwidths, returned as a vector.
- psiftsupport — The wavelet frequency support, returned as a vector.
- phiftsupport — The scaling filter frequency support, returned as a real number.
- phi3dBbw — The scaling filter 3 dB bandwidth, returned as a real number.
- rotations — The wavelet orientation angles in radians, returned as a vector. The length of rotations equals the NumRotations value associated with the filter bank.

The following vectors in the table have equal length: omegapsi, freqsigmapsi, spatialsigmapsi, psi3dBbw, and psiftsupport.

The total number of wavelet filters in a filter bank is length(omegapsi)×length(rotations). See “Determine Wavelet Semi-Major Axis” on page 1-496.

More About

Slant Parameter

The *slant parameter* or *spatial aspect ratio* controls the shape of the elliptical support of the Morlet wavelet.

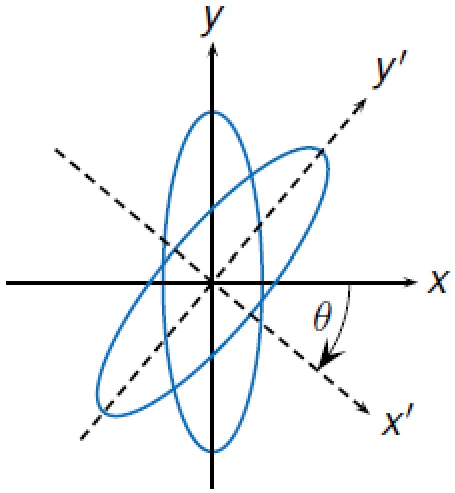
The Morlet wavelet is of the form

$$\psi(x, y) = e^{-(x^2 + \nu^2 y^2)/2\sigma^2} e^{i\omega_\lambda x}$$

where ν is the *slant parameter*. Typically, $\nu < 1$, so that the ellipse $\frac{x^2}{\sigma^2} + \frac{y^2}{\sigma^2/\nu^2}$ is elongated spatially in the *y*-direction. The wavelet is rotated in a clockwise direction: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$.

The rotated Morlet wavelet is of the form

$$\psi(x', y') = e^{-(x'^2 + \nu^2 y'^2)/2\sigma^2} e^{i\omega_\lambda x'}$$



If g(x,y) and G(ω_x,ω_y) form a Fourier pair: $g(x, y) \overset{FT}{\longleftrightarrow} G(\omega_x, \omega_y)$, then so do their rotations:

$$g(x\cos\theta + y\sin\theta, \ -x\sin\theta + y\cos\theta) \overset{FT}{\longleftrightarrow} G(\omega_x\cos\theta + \omega_y\sin\theta, \ -\omega_x\sin\theta + \omega_y\cos\theta)$$

The Fourier transform of the Morlet wavelet is

$$\widehat{\psi}(\omega_x, \omega_y) = \frac{2\pi\sigma^2}{\nu} e^{-\frac{\sigma^2}{\nu}\left((\omega_x - \omega_\lambda)^2 + \frac{\omega_y}{\nu^2}\right)}$$

A given wavelet $\psi(x,y)$ has a reported bandwidth *bw*. The reported bandwidth is dependent on the scale, but not the rotation angle. For a reported 3 dB bandwidth *bw*, the bandwidth along the semi-major spatial axis of the ellipse that describes the wavelet support is *bw* × `slant`.

The semi-major spatial axis depends on the rotation angle. The semi-major spatial axis can be computed from the vector `rotations` in the output argument `filterparams`.

As ordered in the output arguments `f` and `psifilters`, the wavelet filter `psifilters(:,:,1 + k × NumRotations)` for an integer k is the first, unrotated wavelet at a given scale. The wavelet has a center spatial frequency of `f(1 + k × NumRotations,:)`, which is of the form `(`ω_x`,0)`. See "Determine Wavelet Semi-Major Axis" on page 1-496.

See Also

`waveletScattering2`

Introduced in R2019a

filters

DWT filter bank filters

Syntax

```
[Lo,Hi] = filters(fb)
```

Description

`[Lo,Hi] = filters(fb)` returns the lowpass (scaling) and highpass (wavelet) filters, `Lo` and `Hi`, respectively, for the discrete wavelet transform (DWT) filter bank `fb`.

Examples

DWT Filter Bank Filters

Obtain the lowpass and highpass filters for the order-4 symlet.

```
fb = dwtfilterbank('Wavelet','sym4');
[Lo,Hi] = filters(fb)
```

```
Lo = 8×2

   -0.0758    0.0322
   -0.0296   -0.0126
    0.4976   -0.0992
    0.8037    0.2979
    0.2979    0.8037
   -0.0992    0.4976
   -0.0126   -0.0296
    0.0322   -0.0758


Hi = 8×2

   -0.0322   -0.0758
   -0.0126    0.0296
    0.0992    0.4976
    0.2979   -0.8037
   -0.8037    0.2979
    0.4976    0.0992
    0.0296   -0.0126
   -0.0758   -0.0322
```

Confirm the filter bank is orthogonal.

```
isOrthogonal(fb)
```

```
ans = logical
   1
```

Input Arguments

`fb` — Discrete wavelet transform filter bank

`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

Output Arguments

`Lo` — Lowpass (scaling) filters

real-valued matrix

Lowpass (scaling) filters for the DWT filter bank, returned as an *L*-by-2 matrix. *L* is an even positive integer. The first column of `Lo` is the analysis filter, and the second column is the synthesis filter.

For orthogonal wavelets, the lowpass synthesis and lowpass analysis filters are time-reversed versions of each other.

`Hi` — Highpass (wavelet) filters

real-valued matrix

Highpass (wavelet) filters for the DWT filter bank, returned as an *L*-by-2 matrix. *L* is an even positive integer. The first column of `Hi` is the analysis filter, and the second column is the synthesis filter.

For orthogonal wavelets, the highpass synthesis and highpass analysis filters are time-reversed versions of each other.

See Also

`dwtfilterbank` | `wfilters`

Introduced in R2018a

filters2lp

Filters to Laurent polynomials

Syntax

```
[LoDz,HiDz] = filters2lp(Lo)
[ ___ ,LoRz,HiRz] = filters2lp(Lo)
[ ___ ,PRCond,AACond] = filters2lp(Lo)
[ ___ ] = filters2lp(Lo,PmaxLoRz)
[ ___ ] = filters2lp(Lo,PmaxLoRz,AddPOW)
```

Description

`[LoDz,HiDz] = filters2lp(Lo)` returns the Laurent polynomials `LoDz` and `HiDz` that correspond to the z-transform of the lowpass and highpass analysis filters, respectively, associated with the lowpass filter specified by `Lo`.

`[ ___ ,LoRz,HiRz] = filters2lp(Lo)` also returns the Laurent polynomials `LoRz` and `HiRz` that correspond to the z-transform of the lowpass and highpass synthesis filters, respectively. Use this syntax with any of the output arguments in the previous syntax.

`[ ___ ,PRCond,AACond] = filters2lp(Lo)` also returns the perfect reconstruction condition `PRCond` and the anti-aliasing condition `AACond`.

`[ ___ ] = filters2lp(Lo,PmaxLoRz)` sets the maximum order of `LoRz`.

`[ ___ ] = filters2lp(Lo,PmaxLoRz,AddPOW)` sets the maximum order of the Laurent polynomial `HiRz`.

Examples

Laurent Polynomials Associated With Wavelet Filters

Obtain the lowpass filters associated with the biorthogonal `bior1.3` wavelet.

```
[LoD,~,LoR,~] = wfilters("bior1.3");
```

Use `filters2lp` to obtain the Laurent polynomials associated with the wavelet. Also obtain the perfect reconstruction and anti-aliasing conditions.

```
[LoDz,HiDz,LoRz,HiRz,PRC,AAC] = filters2lp({LoR,LoD});
```

Verify the perfect reconstruction condition.

```
eq(LoRz*LoDz + HiRz*HiDz,PRC)
```

```
ans = logical
   1
```

Verify the anti-aliasing condition. Use the helper function `helperMakeLaurentPoly` on page 1-0 to obtain $LoDz(-z)$, where $LoD(z)$ is the Laurent polynomial `LoDz`. Use the helper function `helperMakeLaurentPoly` to obtain $HiDz(-z)$, where $HiD(z)$ is the Laurent polynomial `HiDz`.

```
LoDzm = helperMakeLaurentPoly(LoDz);
HiDzm = helperMakeLaurentPoly(HiDz);
eq(LoRz*LoDzm + HiRz*HiDzm,AAC)
```

```
ans = logical
   1
```

Helper Functions

```
function polyout = helperMakeLaurentPoly(poly)
% This function is only intended to support this example.
% It may change or be removed in a future release.

polyout = poly;
cflen = length(polyout.Coefficients);
cmo = polyout.MaxOrder;
polyneg = (-1).^(mod(cmo,2)+(0:cflen-1));
polyout.Coefficients = polyout.Coefficients.*polyneg;

end
```

Input Arguments

Lo — Wavelet lowpass filter

cell array

Wavelet lowpass filter, specified as a cell array. If the wavelet is orthogonal, then Lo is a one-element cell array that corresponds to `LoR`, the lowpass reconstruction filter. The corresponding highpass filter is `HiR = qmf(LoR)`. For biorthogonal wavelets, `Lo` is a two-element cell array specified as `Lo = {LoR,LoD}`. In this case, `HiR = qmf(fliplr(LoD))`.

Example: If `[LoD,~,LoR,~] = wfilters("bior2.2")`, then `Lo` is specified as `Lo = {LoR,LoD}`.

Data Types: `double`

PmaxLoRz — Maximum power

`0` (default) | integer

Maximum power of the Laurent polynomial `LoRz`, specified as an integer.

Example: If `[~,~,LoRz,HiRz] = filters2lp(Lo,3)`, then the maximum power, or order, of the Laurent polynomial `LoRz` is 3.

Data Types: `double`

AddPOW — Integer

`0` (default) | integer

Integer to set the maximum order of the Laurent polynomial `HiRz`. `PmaxHiRz`, the maximum order of `HiRz`, is

```
    PmaxHiRz = PmaxLoRz+length(HiRz.Coefficients)-2+AddPow.
```

`AddPOW` must be an even integer to preserve the perfect reconstruction condition.

Data Types: `double`

Output Arguments

`LoDz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with the lowpass analysis filter, returned as a `laurentPolynomial` object. `LoDz` is the z-transform of the lowpass analysis filter.

`HiDz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with the highpass analysis filter, returned as a `laurentPolynomial` object. `HiDz` is the z-transform of the highpass analysis filter.

`LoRz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with the lowpass synthesis filter, returned as a `laurentPolynomial` object. `LoRz` is the z-transform of the lowpass synthesis filter.

`HiRz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with the highpass synthesis filter, returned as a `laurentPolynomial` object. `HiRz` is the z-transform of the highpass synthesis filter.

`PRCond,AACond` — Perfect reconstruction and anti-aliasing conditions
`laurentPolynomial` objects

Perfect reconstruction and anti-aliasing conditions, returned as `laurentPolynomial` objects. The perfect reconstruction condition `PRCond` and anti-aliasing condition `AACond` are:

- `PRCond(`*z*`) = LoRz(`*z*`) LoDz(`*z*`) + HiRz(`*z*`) HiDz(`*z*`)`
- `AACond(`*z*`) = LoRz(`*z*`) LoDz(-`*z*`) + HiRz(`*z*`) HiDz(-`*z*`)`

The pairs (`LoRz`, `HiRz`) and (`LoDz`, `HiDz`) are associated with perfect reconstructions filters if and only if:

- `PRCond(`*z*`) = 2`, and
- `AACond(`*z*`) = 0`

If `PRCond(`*z*`) = 2` z^d, a delay is introduced in the reconstruction process.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`lp2filters` | `qmf` | `wave2lp`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

framebounds

DWT filter bank frame bounds

Syntax

```
[a,b] = framebounds(fb)
```

Description

`[a,b] = framebounds(fb)` returns the frame bounds for the discrete wavelet transform (DWT) filter bank `fb`. For an orthogonal wavelet filter bank, the theoretical frame bounds `a` and `b` are equal to 1.

Examples

DWT Filter Bank Frame Bounds

Obtain the frame bounds for the orthogonal Daubechies `db6` wavelet.

```
wv = 'db6';
fb = dwtfilterbank('Wavelet',wv)
```

```
fb = 
  dwtfilterbank with properties:

                Wavelet: 'db6'
           SignalLength: 1024
                  Level: 6
      SamplingFrequency: 1
             FilterType: 'Analysis'
    CustomWaveletFilter: []
    CustomScalingFilter: []
```

```
[a,b] = framebounds(fb)
```

```
a = 1.0000

b = 1.0000
```

The filter bank has the default filter type `Analysis`. Create a second filter bank using the same orthogonal wavelet but with the filter type `Synthesis`. Obtain the frame bounds of this filter bank, which are equal to the previous frame bounds.

```
fbSynthesis = dwtfilterbank('Wavelet',wv,'FilterType','Synthesis');
[a2,b2] = framebounds(fbSynthesis)
```

```
a2 = 1.0000

b2 = 1.0000
```

Create a filter bank for the biorthogonal bior3.9 wavelet. Obtain the frame bounds. The frame bounds are not equal to 1.

```
wv = 'bior3.9';
fbA = dwtfilterbank('Wavelet',wv);
[c,d] = framebounds(fbA)
```

```
c = 0.6250
```

```
d = 3.2982
```

Create a second filter bank using the same biorthogonal wavelet but with the filter type Synthesis. Obtain the frame bounds of this filter bank. Since the wavelet is biorthogonal, the frame bounds change.

```
fbASynthesis = dwtfilterbank('Wavelet',wv,'FilterType','Synthesis');
[c2,d2] = framebounds(fbASynthesis)
```

```
c2 = 0.5502
```

```
d2 = 2.0015
```

Input Arguments

fb — Discrete wavelet transform filter bank

dwtfilterbank object

Discrete wavelet transform (DWT) filter bank, specified as a dwtfilterbank object.

Output Arguments

a — Lower frame bound

positive real number

Lower frame bound of the DWT filter bank fb, returned as a positive real number.

b — Upper frame bound

positive real number

Upper frame bound of the DWT filter bank fb, returned as a positive real number.

See Also

dwtfilterbank | isBiorthogonal | isOrthogonal

Introduced in R2018a

framebounds

Shearlet system frame bounds

Syntax

```
[a,b] = framebounds(sls)
```

Description

`[a,b] = framebounds(sls)` returns the lower and upper frame bounds for the shearlet system `sls`. The energy in the shearlet transform coefficients is bounded by the energy in the input image and the frame bounds. See “Frame Bounds” on page 1-513.

Examples

Shearlet System Frame Bounds

This example shows how the `PreserveEnergy` property affects the frame bounds of a shearlet system.

Load an image and calculate its energy.

```
load xbox
energyIm = norm(xbox,'fro')^2;
```

Create two shearlet systems that can be applied to the image. Set the value of `PreserveEnergy` in the first shearlet system to `true` and in the second shearlet system to `false`.

```
slsT = shearletSystem('ImageSize',size(xbox),'PreserveEnergy',true);
slsF = shearletSystem('ImageSize',size(xbox),'PreserveEnergy',false);
```

Obtain the shearlet transform of the image using both shearlet systems.

```
cfsT = sheart2(slsT,xbox);
cfsF = sheart2(slsF,xbox);
```

Calculate the frame bounds of `slsT`. Confirm that `slsT` is a Parseval frame.

```
[aT,bT] = framebounds(slsT)
```

```
aT = 1
```

```
bT = 1
```

Confirm that using `slsT` preserves energy.

```
energyCfsT = norm(cfsT(:))^2;
abs(energyIm-energyCfsT)
```

```
ans = 6.9849e-10
```

Obtain the frame bounds of `slsF`. Confirm the lower and upper frame bounds are not both equal to 1.

```
[aF,bF] = framebounds(slsF)
```

```
aF = 1.0000
```

```
bF = 8.0000
```

Even though `slsF` is not normalized to be a Parseval frame, confirm the frame inequality is still satisfied.

```
energyCfsF = norm(cfsF(:))^2;
aF*energyIm <= norm(cfsF(:))^2 && norm(cfsF(:))^2 <= bF*energyIm
```

```
ans = logical
   1
```

Input Arguments

`sls` — Shearlet system

`shearletSystem` object

Shearlet system, specified as a `shearletSystem` object.

Output Arguments

`a,b` — Lower and upper frame bounds

positive real numbers

Lower and upper frame bounds of the shearlet system, returned as positive real numbers. If the PreserveEnergy value of `sls` is `true`, then `sls` is a Parseval frame, and both frame bounds are equal to 1. See “Frame Bounds” on page 1-513.

The data types of the frame bounds match the Precision value of the shearlet system.

Note For an image *X*, if `sls` is a Parseval frame and `C = sheart2(sls,X)`, then the energy of *X* and the energy of *C* are equal within round-off error.

Data Types: `single` | `double`

More About

Frame Bounds

The energy in the shearlet transform of an image is bounded by the energy of the image and the lower and upper frame bounds `a,b` of the shearlet system. If *X* is an *M*-by-*N* image and *C*, the shearlet transform of *X*, is *M*-by-*N*-by-*K*, then the frame inequality holds:

$$\mathrm{a}\sum_{i=1}^{M}\sum_{j=1}^{N}\left|x_{ij}\right|^2 \le \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K}\left|c_{ijk}\right|^2 \le \mathrm{b}\sum_{i=1}^{M}\sum_{j=1}^{N}\left|x_{ij}\right|^2 .$$

In a Parseval frame, `a` = `b` = 1, and the shearlet transform preserves energy.

Extended Capabilities

C/C++ Code Generation

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

See Also

`shearletSystem`

Introduced in R2019b

freqz

CWT filter bank frequency responses

Syntax

```
[psidft,f] = freqz(fb)
[ ___ ] = freqz( ___ ,Name=Value)
freqz( ___ )
```

Description

`[psidft,f] = freqz(fb)` returns the frequency responses for the wavelet filters, `psidft`, and the frequency vector, `f`, for the continuous wavelet transform (CWT) filter bank, `fb`. Frequencies are in cycles/sample or Hz. If you specify a sampling period, the frequencies are in cycles/unit time where the time unit is the unit of the duration sampling period.

The frequency responses, `psidft`, are one-sided frequency responses for the positive frequencies. For the analytic wavelets supported by `cwtfilterbank`, the frequency responses are real-valued and are equivalent to the magnitude frequency response.

`[ ___ ] = freqz( ___ ,Name=Value)` specifies one or more additional name-value arguments. For example, `psidft = freqz(fb,FrequencyRange="twosided")` returns the full two-sided frequency responses.

`freqz( ___ )` with no output arguments plots the magnitude frequency responses for the CWT filter bank, `fb`.

Examples

Invert CWT Using Approximate Synthesis Filters

Load the Kobe earthquake data. Create a CWT filter bank with period boundary handling that you can apply to the data.

```
load kobe
fb = cwtfilterbank(SignalLength=numel(kobe),Boundary="periodic");
```

Obtain the two-sided wavelet and scaling filter responses.

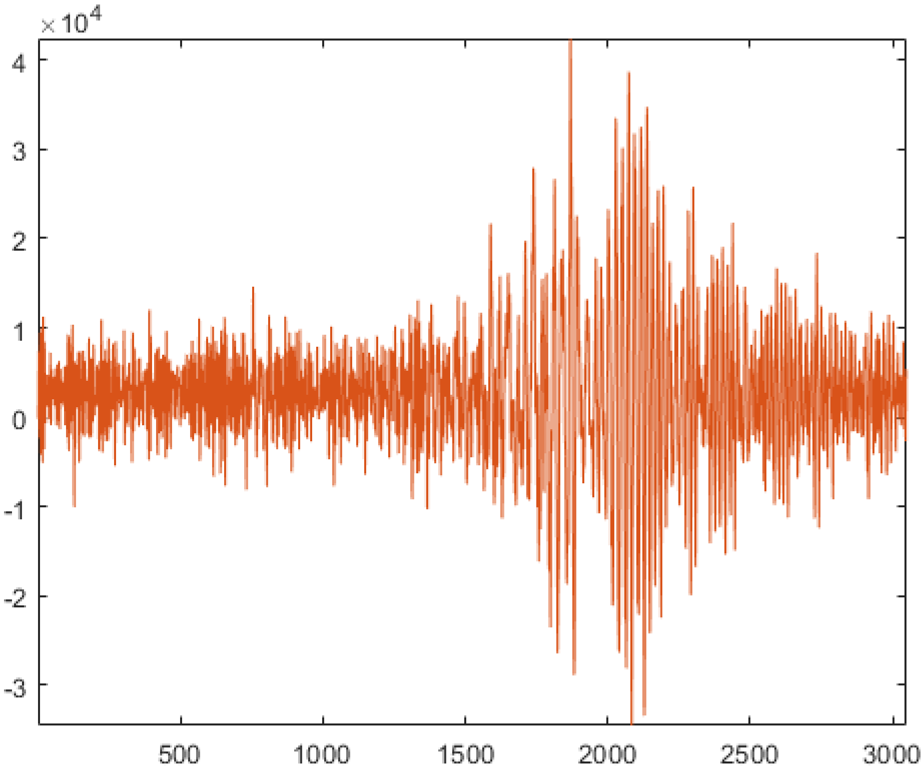
```
[psidft,f] = freqz(fb,IncludeLowpass=true,FrequencyRange="twosided");
```

Obtain the CWT of the data. Also obtain the scaling coefficients.

```
[cfs,~,~,scalcfs] = wt(fb,kobe);
```

Invert the transform using the filter bank and the scaling coefficients.

```
xrec = icwt(cfs,ScalingCoefficients=scalcfs,AnalysisFilterBank=psidft);
plot([kobe(:) xrec(:)])
axis tight
```



Obtain the maximum reconstruction error.

```
norm(kobe(:)-xrec(:),'Inf')
```

```
ans = 2.9104e-11
```

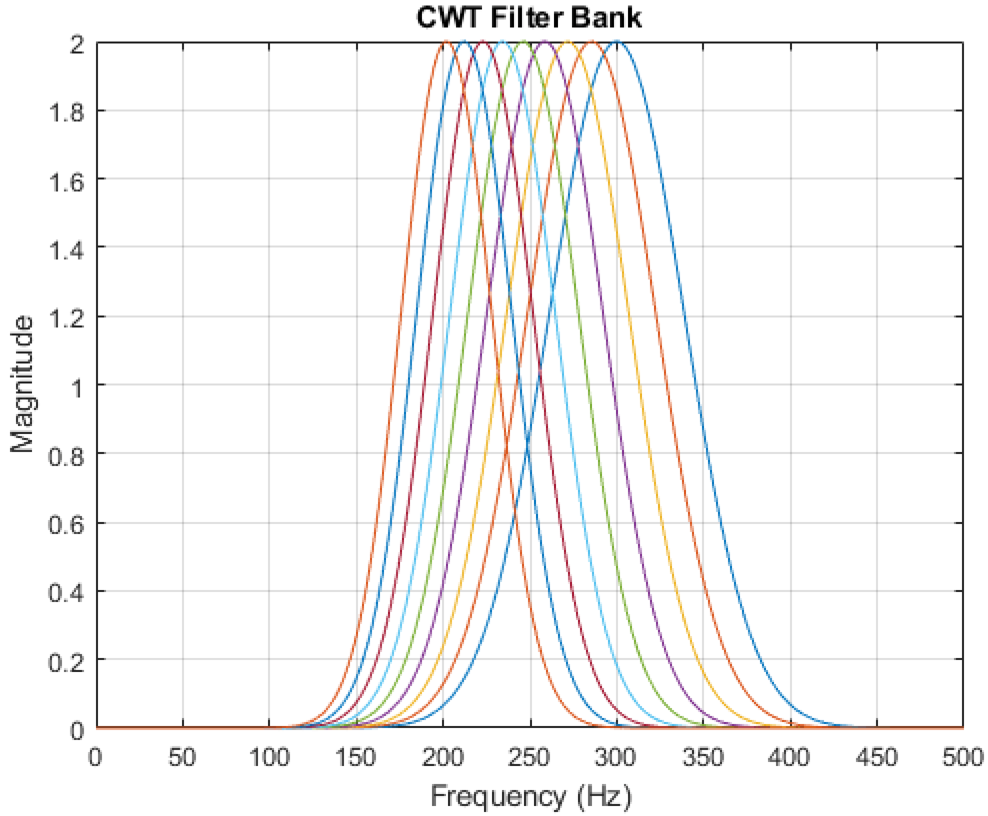
Frequency Responses of Continuous Wavelet Transform Filter Bank

Create a CWT filter bank. Set the voices per octave to 14, the sampling frequency to 1000 Hz, and frequency limits to range from 200 Hz to 300 Hz.

```
fb = cwtfilterbank(VoicesPerOctave=14,...
    SamplingFrequency=1000,FrequencyLimits=[200 300]);
```

Plot the frequency responses.

```
freqz(fb)
```



Boundary Handling and Frequency Range

This example shows how boundary handling and signal length affect the range of frequency responses `freqz` returns.

Reflection / Even Length

Create a CWT filter bank suitable for an even-length signal. Use the default `Boundary` setting `reflection`.

```
sLen = 256;
fb = cwtfilterbank(SignalLength=sLen);
```

Obtain the one-sided frequency responses of the filter bank. Also obtain the frequency vector.

```
[psidft,f] = freqz(fb,FrequencyRange="onesided");
```

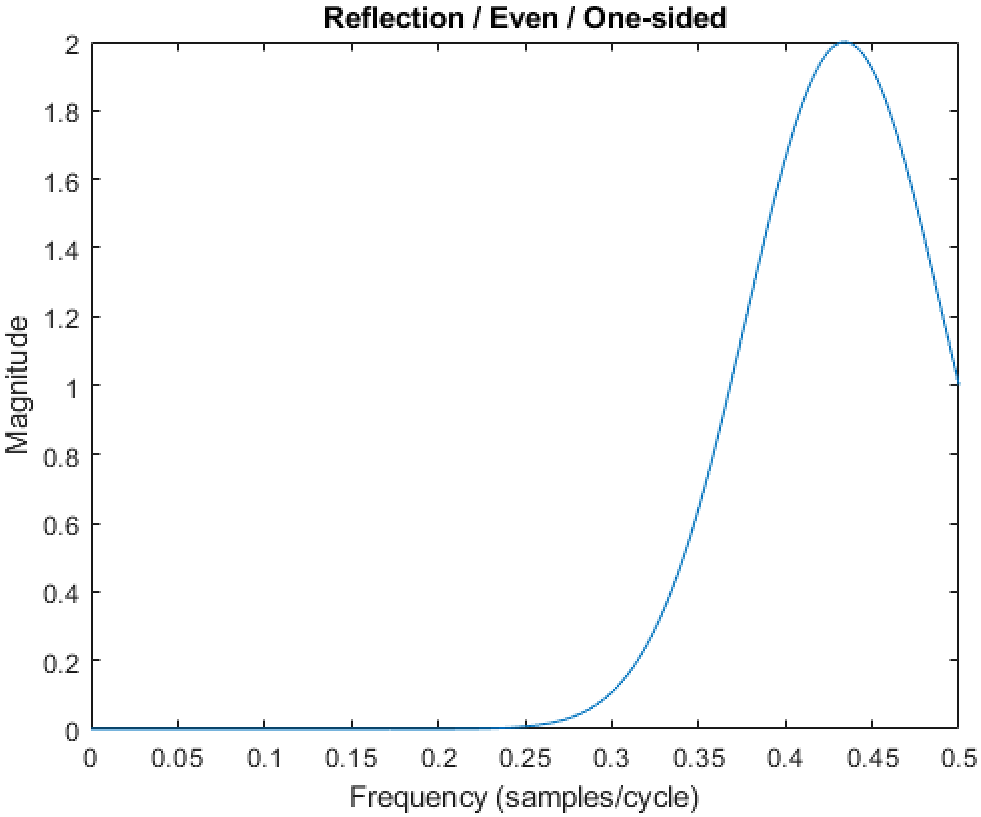
Confirm the range of frequencies includes the Nyquist.

```
f(end)
```

```
ans = 0.5000
```

Plot the frequency response of the filter with the highest center frequency.

```
plot(f,psidft(1,:))
xlabel("Frequency (samples/cycle)")
ylabel("Magnitude")
title("Reflection / Even / One-sided")
```

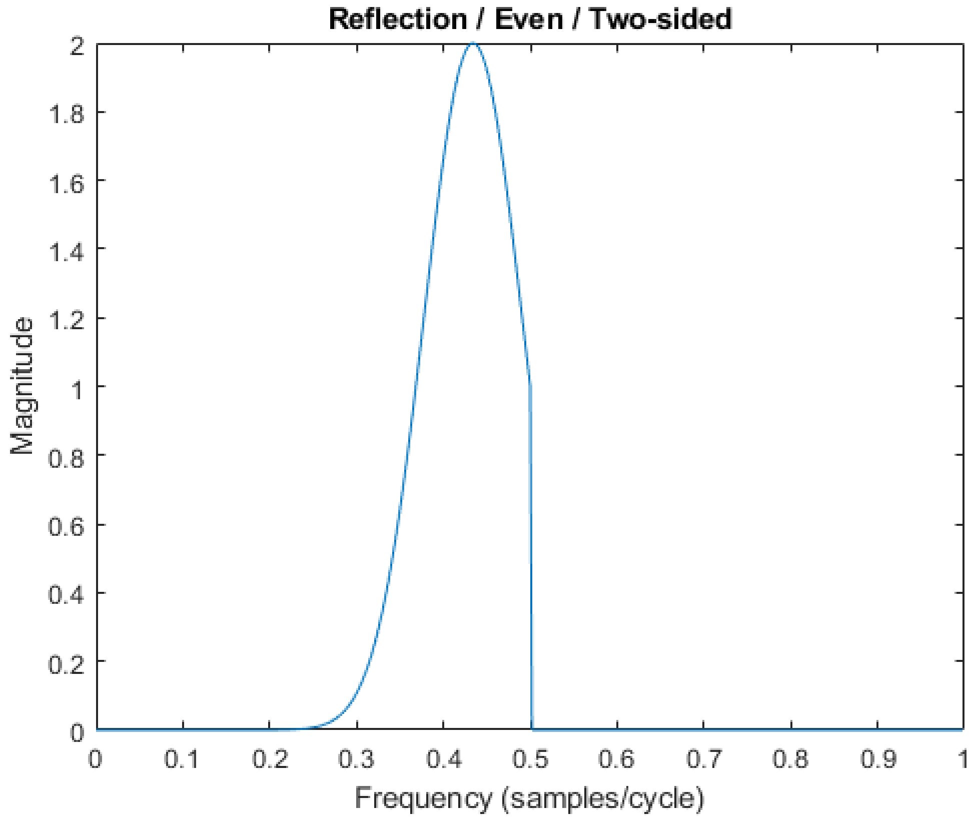


Obtain and plot the two-sided frequency responses. Confirm the frequency range does not include the Nyquist.

```
[psidft,f] = freqz(fb,FrequencyRange="twosided");
f(end)
```

```
ans = 0.9980
```

```
plot(f,psidft(1,:))
xlabel("Frequency (samples/cycle)")
ylabel("Magnitude")
title("Reflection / Even / Two-sided")
```



Reflection / Odd Length

Create a CWT filter bank suitable for an odd-length signal. Use the default `Boundary` setting `reflection`.

```
sLen = 255;
fb = cwtfilterbank(SignalLength=sLen);
```

Obtain the one-sided frequency responses of the filter bank. Confirm the range of frequencies does not include the Nyquist.

```
[~,f] = freqz(fb,FrequencyRange="onesided");
f(end)
```

```
ans = 0.4990
```

Periodic / Even Length

Create a CWT filter bank with periodic boundary handling suitable for an even-length signal

```
sLen = 256;
fb = cwtfilterbank(SignalLength=sLen,Boundary="periodic");
```

Obtain the one-sided frequency responses of the filter bank. Confirm the range of frequencies does include the Nyquist.

```
[~,f] = freqz(fb,FrequencyRange="onesided");
f(end)
```

```
ans = 0.5000
```

Periodic / Odd Length

Create a CWT filter bank with periodic boundary handling suitable for an odd-length signal

```
sLen = 255;
fb = cwtfilterbank(SignalLength=sLen,Boundary="periodic");
```

Obtain the one-sided frequency responses of the filter bank. Confirm the range of frequencies does not include the Nyquist.

```
[~,f] = freqz(fb,FrequencyRange="onesided");
f(end)
```

```
ans = 0.4980
```

Input Arguments

fb — Continuous wavelet transform filter bank

cwtfilterbank object

Continuous wavelet transform (CWT) filter bank, specified as a cwtfilterbank object.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: psidft = freqz(fb,IncludeLowpass=true) appends the lowpass, or scaling filter, frequency response as the final row of psidft.

IncludeLowpass — Append lowpass filter frequency response

false or 0 (default) | true or 1

Option to append lowpass, or scaling filter, frequency response as the final row of psidft, specified as one of these:

- 1 (true) — Include the frequency response symmetrically
- 0 (false) — Do not include the frequency response

For the analytic wavelets supported by cwtfilterbank, the scaling filter frequency response is real-valued and is equivalent to the magnitude frequency response.

Data Types: logical

FrequencyRange — Frequency range for filter responses

"onesided" (default) | "twosided"

Frequency range for the wavelet and scaling function frequency responses, specified as one of "onesided", or "twosided". The frequency ranges corresponding to each option are

- "onesided" — returns the frequency responses from [0,½] when the length of the padded filters is even and [0,½) when the length of the padded filters is odd. Padding is added when the Boundary property of the filter bank is "reflection".

If a sampling frequency *Fs* is specified in the filter bank, the intervals become [0,*Fs*/2] and [0,*Fs*/2) respectively.

- `"twosided"` — returns the full two-sided frequency responses over the range [0,1). If a sampling frequency *Fs* is specified in the filter bank, the interval becomes [0,*Fs*).

Note To use the wavelet and scaling filters in the inverse CWT, set Boundary in the filter bank to `"periodic"`, and use `IncludeLowpass=true` and `FrequencyRange="twosided"` in `freqz`.

Output Arguments

`psidft` — Frequency responses

real-valued 2-D matrix

Frequency responses of a CWT filter bank, returned as a real-valued matrix. Each column of `psidft` is the response at the frequency in the corresponding element of `f`.

By default, frequency responses, `psidft`, are one-sided frequency responses for the positive frequencies. For the analytic wavelets supported by `cwtfilterbank`, the frequency responses are real-valued and are equivalent to the magnitude frequency response.

Data Types: `double`

`f` — Frequencies

real-valued vector

Frequencies, in cycles/sample or hertz, returned as a real-valued vector.

If you specify a sampling period, the frequencies are in cycles/unit time, where the time unit is the unit of the duration `SamplingPeriod`.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Plotting is not supported.

See Also

`cwtfilterbank` | `powerbw` | `centerFrequencies` | `centerPeriods`

Introduced in R2018a

freqz

DWT filter bank frequency responses

Syntax

```
[psidft,f] = freqz(fb)
[psidft,f,phidft] = freqz(fb)
freqz(fb)
```

Description

`[psidft,f] = freqz(fb)` returns the complex-valued frequency responses for the wavelet filters `psidft` and the frequency vector `f` for the discrete wavelet transform (DWT) filter bank `fb`. Frequencies are in cycles/sample or in Hz if a sampling frequency is defined in `fb`. The frequency responses are centered so that the zero frequency is in the middle.

`[psidft,f,phidft] = freqz(fb)` returns the complex-valued frequency responses for the scaling filters `phidft` for the DWT filter bank `fb` at all levels of the decomposition.

`freqz(fb)` plots the one-sided magnitude frequency responses for the wavelet filter bank, `fb`. Magnitude frequency responses are plotted for all wavelet bandpass filters and the coarsest resolution scaling filter. The legend is interactive. To toggle the visibility of the filter magnitude response, click the corresponding line in the legend.

Examples

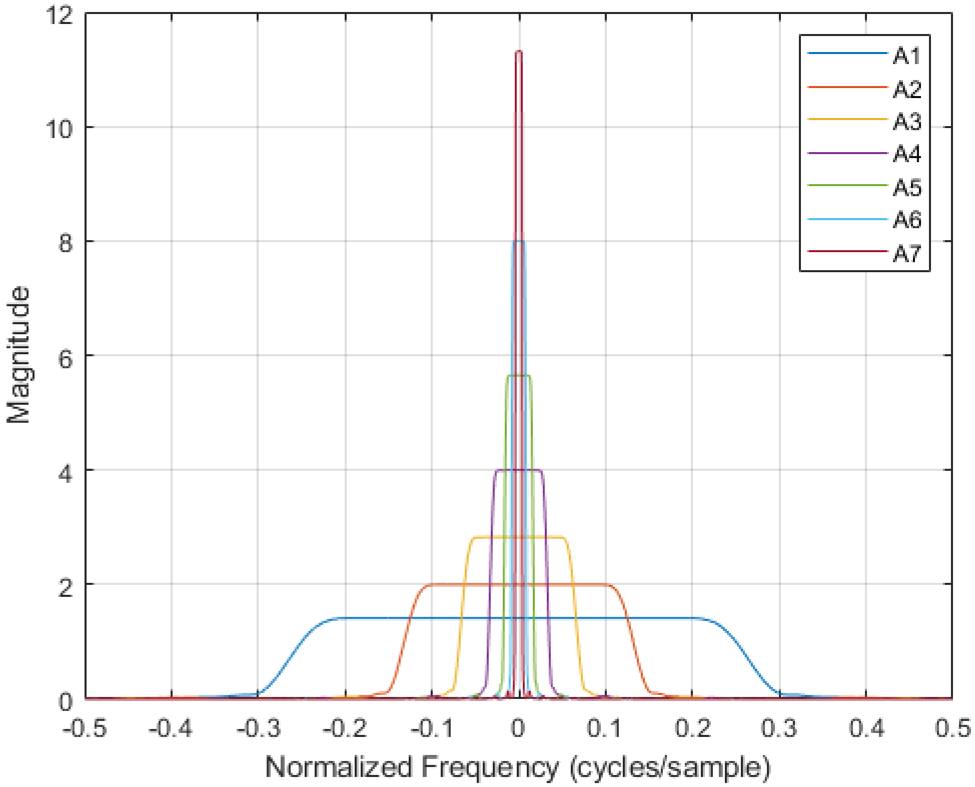
DWT Filter Bank Frequency Responses

Create a DWT filter bank for a length 4096 signal and the Fejér-Korovkin `fk22` wavelet. Plot the magnitude frequency responses of the wavelet filters and final resolution scaling filter.

```
len = 4096;
fb = dwtfilterbank('Wavelet','fk22','SignalLength',len);
freqz(fb)
```

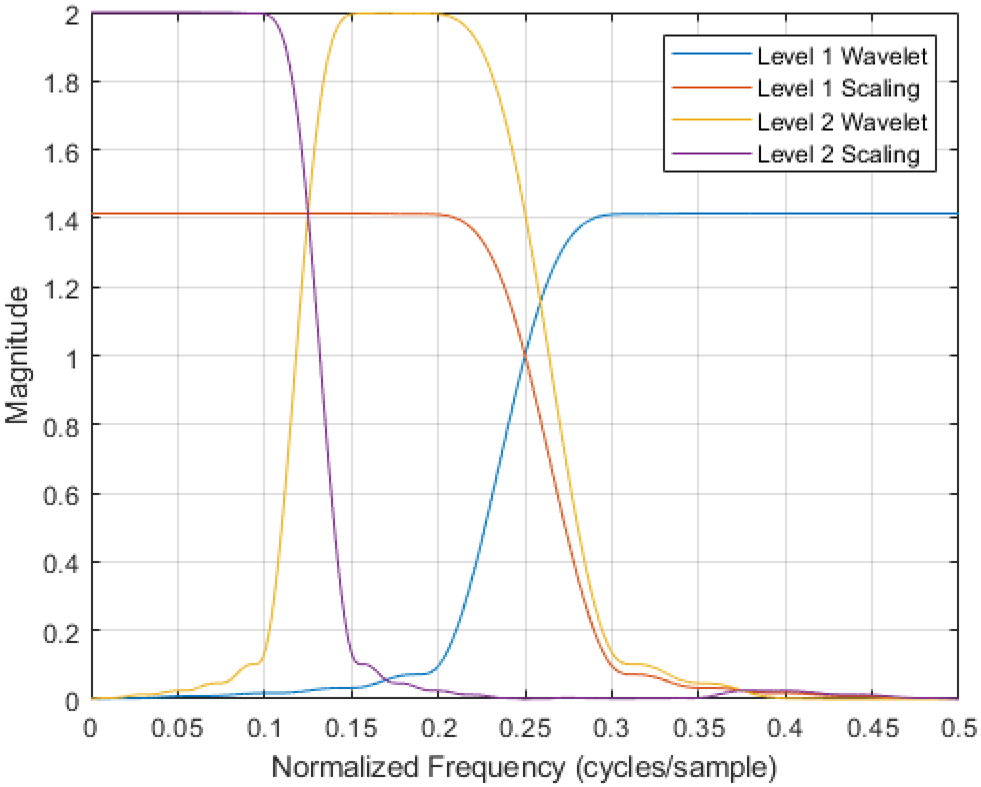
Obtain the frequency responses for the wavelet and scaling filters. Plot the magnitude frequency responses of the scaling filters at all levels of decomposition.

```
[psidft,f,phidft] = freqz(fb);
plot(f,abs(phidft)')
grid on
xlabel('Normalized Frequency (cycles/sample)')
ylabel('Magnitude')
legend('A1','A2','A3','A4','A5','A6','A7')
```



Plot the one-sided magnitude frequency responses of the wavelet and scaling filters at the first two levels of decomposition. Note how the second level frequency responses overlap the magnitude response of the first level scaling filter.

```
plot(f(len/2:end),abs(psidft(1,len/2:end))')
hold on
plot(f(len/2:end),abs(phidft(1,len/2:end))')
plot(f(len/2:end),abs(psidft(2,len/2:end))')
plot(f(len/2:end),abs(phidft(2,len/2:end))')
grid on
xlabel('Normalized Frequency (cycles/sample)')
ylabel('Magnitude')
legend('Level 1 Wavelet','Level 1 Scaling','Level 2 Wavelet','Level 2 Scaling')
```



Input Arguments

fb — Discrete wavelet transform filter bank

dwtfilterbank object

Discrete wavelet transform (DWT) filter bank, specified as a dwtfilterbank object.

Output Arguments

psidft — Wavelet filter frequency responses

complex-valued matrix

Wavelet filter frequency responses for the DWT filter bank fb, returned as an *L*-by-*N* matrix, where *L* is the filter bank Level and *N* is the filter bank SignalLength. The frequency responses are centered so that the zero frequency is centered in the middle.

f — Frequencies

real-valued vector

Frequencies, in cycles/sample or Hz, returned as a real-valued vector of length *N*, where *N* is the filter bank SignalLength. If a sampling frequency is specified in fb, frequencies are in Hz.

Data Types: double

phidft — Scaling function frequency responses
complex-valued matrix

Scaling function frequency responses for the DWT filter bank fb, returned as an *L*-by-*N* matrix, where *L* is the filter bank Level and *N* is the filter bank SignalLength. The frequency responses are centered so that the zero frequency is centered in the middle.

See Also

dwtfilterbank | wavelets | scalingfunctions

Introduced in R2018a

gauswavf

Gaussian wavelet

Syntax

```
[psi,x] = gauswavf(lb,ub,n)
[psi,x] = gauswavf(lb,ub,n,p)
[psi,x] = gauswavf(lb,ub,n,wname)
```

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. `p` is an integer from 1 through 8.

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

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.

`[psi,x] = gauswavf(lb,ub,n,wname)` used the valid wavelet family short name `wname` plus the order of the derivative in a character vector or string scalar, 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.

Examples

Create Gaussian Wavelet

This example shows how to create and plot a Gaussian wavelet of order 8.

Set the initial effective support and grid parameters.

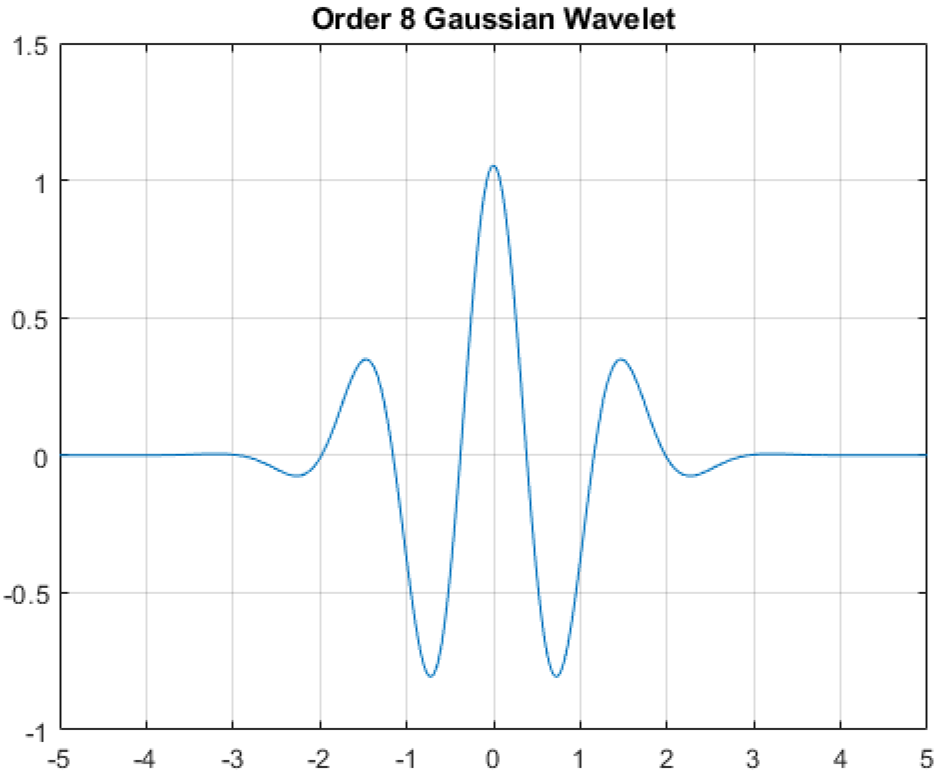
```
lb = -5;
ub = 5;
n = 1000;
```

Compute the Gaussian wavelet of order 8.

```
[psi,x] = gauswavf(lb,ub,n,8);
```

Now plot the wavelet.

```
plot(x,psi)
title('Order 8 Gaussian Wavelet')
grid on
```



Input Arguments

lb — Left endpoint

real number

Left endpoint of the closed interval, specified as a real number. `lb` is strictly less than `ub`.

Data Types: `double`

ub — Right endpoint

real number

Right endpoint of the closed interval, specified as a real number. `ub` is strictly greater than `lb`.

Data Types: `double`

n — Number of regularly spaced points

positive integer

Number of regularly spaced points in the interval [`lb`,`ub`], specified as a positive integer. The derivative of the Gaussian is evaluated at these points.

Data Types: `double`

p — Derivative
positive integer

Positive integer defining the order of the derivative of the Gaussian wavelet, specified as a positive integer. `p` is an integer from 1 through 8.

wname — Gaussian wavelet
character vector | string scalar

Gaussian wavelet to evaluate, specified as a character vector or string scalar. `wname` is of the form `'cgauN'` where *N* is an integer that denotes the order of the derivative of the Gaussian wavelet. *N* is an integer from 1 through 8.

Example: `'gaus4'` denotes the fourth derivative of the Gaussian wavelet.

Output Arguments

psi — Derivative of Gaussian wavelet
real-valued vector

Derivative of the Gaussian wavelet, returned as a real-valued 1-by-`N` vector.

x — Sample points
real-valued vector

Sample points where the derivative of the Gaussian wavelet is evaluated, returned as a real-valued 1-by-`N` vector. The sample points are evenly distributed between `lb` and `ub`.

See Also

`waveinfo` | `wavemngr`

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

getLabelDefinitions

Get label definitions in labeled signal set

Syntax

```
lbldefs = getLabelDefinitions(lss)
getLabelDefinitions(lss,Name=Value)
```

Description

`lbldefs = getLabelDefinitions(lss)` returns a vector of `signalLabelDefinition` objects with the labels of the labeled signal set `lss`.

Changing `lbldefs` does not affect the labeled set. To modify label definitions, use `editLabelDefinition`, `addLabelDefinitions`, and `removeLabelDefinition`.

`getLabelDefinitions(lss,Name=Value)` returns a vector of `signalLabelDefintion` objects that have a label type and frame policy equal to the values you specify using name-value arguments.

Examples

Get Label Definitions

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Retrieve the definitions of the labels in the set.

```
dfs = getLabelDefinitions(lss);

for k = 1:length(dfs)
    dfs(k)
end
```

```
ans = 
  signalLabelDefinition with properties:
```

```
             Name: "WhaleType"
        LabelType: "attribute"
    LabelDataType: "categorical"
       Categories: [3x1 string]
     DefaultValue: []
        Sublabels: [0x0 signalLabelDefinition]
              Tag: ""
      Description: "Whale type"

 Use labeledSignalSet to create a labeled signal set.

ans = 
  signalLabelDefinition with properties:

                  Name: "MoanRegions"
             LabelType: "roi"
         LabelDataType: "logical"
    ValidationFunction: []
     ROILimitsDataType: "double"
          DefaultValue: []
             Sublabels: [0x0 signalLabelDefinition]
                   Tag: ""
           Description: "Regions where moans occur"

 Use labeledSignalSet to create a labeled signal set.

ans = 
  signalLabelDefinition with properties:

                  Name: "TrillRegions"
             LabelType: "roi"
         LabelDataType: "logical"
    ValidationFunction: []
     ROILimitsDataType: "double"
          DefaultValue: []
             Sublabels: [1x1 signalLabelDefinition]
                   Tag: ""
           Description: "Regions where trills occur"

 Use labeledSignalSet to create a labeled signal set.
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `getLabelDefinitions(lss,LabelType="roiFeature",FrameSize=50,FrameOverlapLength=5)` returns `signalLabelDefinition` objects that contain `roiFeature` labels which have a frame size equal to 50 and a frame overlap length equal to 5.

`LabelType` — Label type

`"attribute"` | `"roi"` | `"point"` | `"attributeFeature"` | `"roiFeature"`

Label type, specified as `"attribute"`, `"roi"`, `"point"`, `"attributeFeature"`, or `"roiFeature"`.

Example: `LabelType="attribute"`

Data Types: `char` | `string`

`FrameSize` — Frame size

numeric scalar

Frame size, specified as a numeric scalar. To enable this argument, set `LabelType` to `"roiFeature"`.

Example: `LabelType="roiFeature",FrameSize=50`

Data Types: `double`

`FrameOverlapLength` — Frame overlap length

numeric scalar

Frame overlap length, specified as a numeric scalar. To enable this argument, set `LabelType` to `"roiFeature"`. You cannot specify `FrameOverlapLength` and `FrameRate` simultaneously.

When you specify a frame overlap length, the function returns `signalLabelDefinition` objects that contain `roiFeature` labels that have `FrameSize` and `FrameOverlapLength` equal to the values you specify.

Example: `LabelType="roiFeature",FrameSize=50,FrameOverlapLength=5`

Data Types: `double`

`FrameRate` — Frame rate

numeric scalar

Frame rate, specified as a numeric scalar. To enable this argument, set `LabelType` to `"roiFeature"`. You cannot specify `FrameRate` and `FrameOverlapLength` simultaneously.

When you specify a frame rate, the function returns `signalLabelDefinition` objects that contain `roiFeature` labels that have `FrameSize` and `FrameRate` equal to the values you specify.

Example: `LabelType="roiFeature",FrameSize=50,FrameRate=45`

Data Types: `double`

Output Arguments

lbldefs — Signal label definitions
signalLabelDefinition object

Signal label definitions, returned as a signalLabelDefinition object or a vector of such objects.

See Also

labeledSignalSet | signalLabelDefinition

Introduced in R2018b

getLabeledSignal

Get labeled signals from labeled signal set

Syntax

```
[t,info] = getLabeledSignal(lss)
[t,info] = getLabeledSignal(lss,midx)
```

Description

`[t,info] = getLabeledSignal(lss)` returns a table with all the signals and labeled data in the labeled signal set `lss`.

`[t,info] = getLabeledSignal(lss,midx)` returns a table with the signals specified in `midx`.

Examples

Get Labeled Signal

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get a table with all the signals in `lss`.

```
t = getLabeledSignal(lss)
```

```
t=2×4 table
                        Signal          WhaleType    MoanRegions    TrillRegions
                   ________________    _________    ___________    ____________

    Member{1}    {79572x1 double}      blue       {3x2 table}    {1x3 table} 
    Member{2}    {76579x1 double}      blue       {3x2 table}    {1x3 table} 
```

Identify the sublabels of the trill regions.

```
d = getLabelNames(lss,'TrillRegions')
```

```
d = 
"TrillPeaks"
```

Get the labeled signal corresponding to the second member of the set. Determine the sample rate.

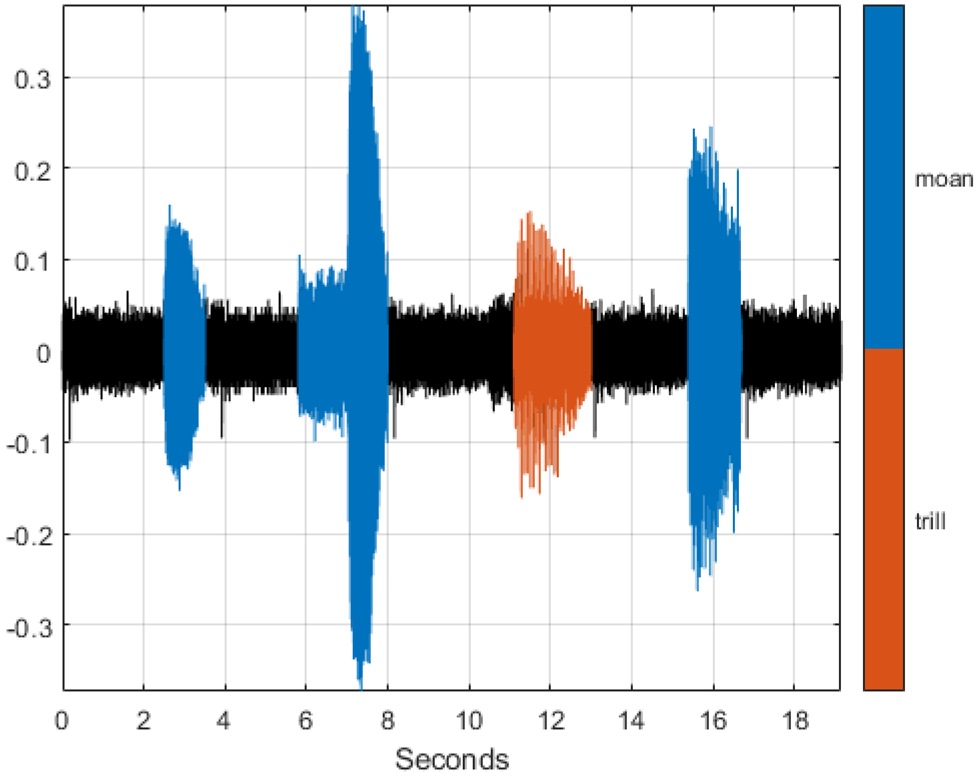
```
idx = 2;

[lbs,info] = getLabeledSignal(lss,idx)
```

```
lbs=1×4 table
                      Signal          WhaleType     MoanRegions     TrillRegions
                 ________________     _________     ___________     ____________

    Member{2}    {76579x1 double}       blue        {3x2 table}     {1x3 table} 


info = struct with fields:
    TimeInformation: "sampleRate"
         SampleRate: 4000

```

```
fs = info.SampleRate;
```

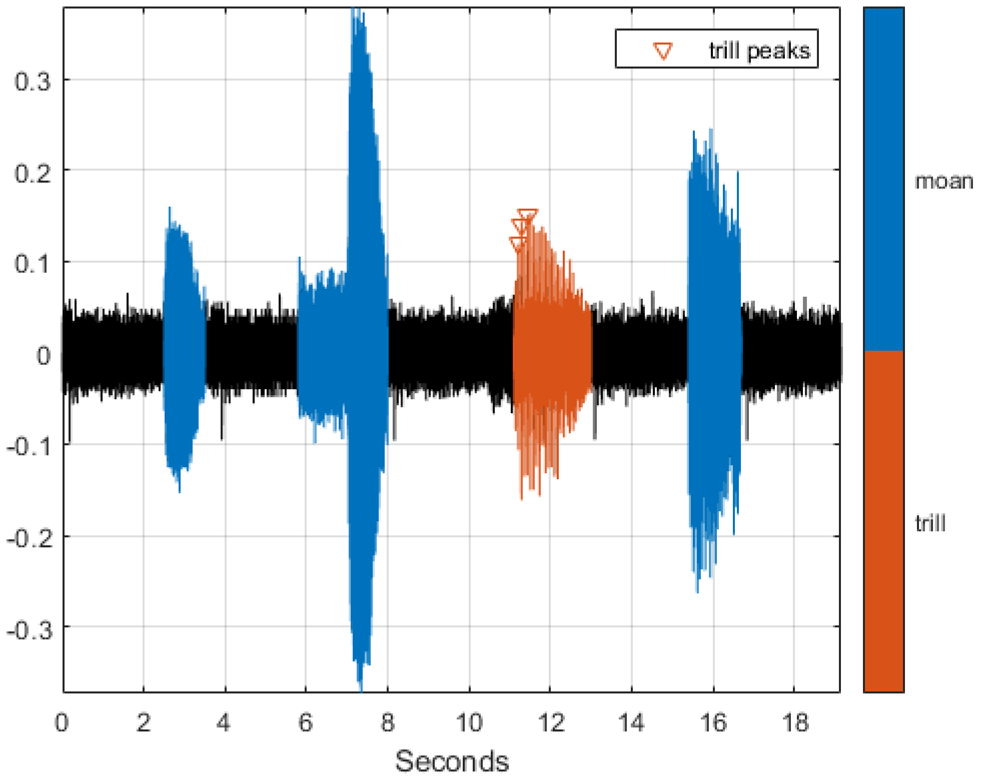
Identify the moan and trill regions of interest. Use a `signalMask` (Signal Processing Toolbox) object to plot the signal and highlight the moans and trills.

```
mvals = getLabelValues(lss,idx,'MoanRegions');
tvals = getLabelValues(lss,idx,'TrillRegions');

tb = [mvals;tvals];
tb.Value = categorical( ...
    [repmat("moan",height(mvals),1);repmat("trill",height(tvals),1)], ...
    ["moan" "trill"]);
sm = signalMask(tb,"SampleRate",fs);
plotsigroi(sm,getSignal(lss,idx))
```



Identify three peaks of the trill region and plot them.

```
peaks = getLabelValues(lss,idx,{'TrillRegions','TrillPeaks'});

hold on
pk = plot(peaks.Location,cell2mat(peaks.Value),'v');
hold off
legend(pk,'trill peaks')
```



Input Arguments

lss — Labeled signal set

labeledSignalSet object

Labeled signal set, specified as a labeledSignalSet object.

Example: labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female')) specifies a two-member set of random signals containing the attribute 'female'.

midx — Member row number

positive integer

Member row number, specified as a positive integer. midx specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

Output Arguments

t — Labeled signal

table

Labeled signal, specified as a table.

`info` — Time information
structure

Time information, returned as a structure.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

getLabelNames

Get label names in labeled signal set

Syntax

```
lblnames = getLabelNames(lss)
sublblnames = getLabelNames(lss,lblname)
getLabelNames(lss,Name=Value)
```

Description

`lblnames = getLabelNames(lss)` returns a string array containing the label names in the labeled signal set `lss`.

`sublblnames = getLabelNames(lss,lblname)` returns a string array containing the sublabel names for the label named `lblname` in the labeled signal set `lss`.

`getLabelNames(lss,Name=Value)` returns a string array containing the label names in the labeled signal set for labels which have a label type and frame policy equal to the values you specify using name-value arguments.

Examples

Get Label Names

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get the names of the labels in the set.

```
str = getLabelNames(lss)
```

```
str = 3x1 string
    "WhaleType"
    "MoanRegions"
    "TrillRegions"
```

Verify that only the `'TrillRegions'` label has sublabels.

```
for kj = 1:length(str)
    sbstr = str{kj};
    sbl = [sbstr getLabelNames(lss,sbstr)]
end
```

```
sbl = 
"WhaleType"

sbl = 
"MoanRegions"

sbl = 1x2 string
    "TrillRegions"    "TrillPeaks"
```

Input Arguments

`lss` — Labeled signal set
`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lblname` — Label name
character vector | string scalar

Label name, specified as a character vector or a string scalar.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Name-Value Arguments

Example: `getLabelNames(lss,LabelType="roiFeature",FrameSize=50,FrameOverlapLength=5)` returns the names of `roiFeature` labels which have a frame size equal to 50 and a frame overlap length equal to 5.

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

`LabelType` — Label type
`"attribute"` | `"roi"` | `"point"` | `"attributeFeature"` | `"roiFeature"`

Label type, specified as `"attribute"`, `"roi"`, `"point"`, `"attributeFeature"`, or `"roiFeature"`.

Example: `LabelType="attribute"`

Data Types: char | string

FrameSize — Frame size
numeric scalar

Frame size, specified as a numeric scalar. To enable this argument, set LabelType to "roiFeature".

Example: LabelType="roiFeature",FrameSize=50

Data Types: double

FrameOverlapLength — Frame overlap length
numeric scalar

Frame overlap length, specified as a numeric scalar. To enable this argument, set LabelType to "roiFeature". You cannot specify FrameOverlapLength and FrameRate simultaneously.

Example: LabelType="roiFeature",FrameSize=50,FrameOverlapLength=5

Data Types: double

FrameRate — Frame rate
numeric scalar

Frame rate, specified as a numeric scalar. To enable this argument, set LabelType to "roiFeature". You cannot specify FrameRate and FrameOverlapLength simultaneously.

Example: LabelType="roiFeature",FrameSize=50,FrameRate=45

Data Types: double

Output Arguments

lblnames — Label names
string array

Label names, returned as a string array.

sublblnames — Sublabel names
string array

Sublabel names, returned as a string array.

See Also

labeledSignalSet | signalLabelDefinition

Introduced in R2018b

getLabelValues

Get label values from labeled signal set

Syntax

```
val = getLabelValues(lss)
val = getLabelValues(lss,midx)

[val,sublbltbl] = getLabelValues(lss,midx,lblname)

[ ___ ] = getLabelValues( ___ ,'LabelRowIndex',ridx)
[ ___ ] = getLabelValues( ___ ,'SublabelRowIndex',sridx)
```

Description

`val = getLabelValues(lss)` returns a table containing the label values for all members of the labeled signal set `lss`.

`val = getLabelValues(lss,midx)` returns a table containing the label values for the member specified by `midx`.

`[val,sublbltbl] = getLabelValues(lss,midx,lblname)` returns the value of the label named `lblname`. If `lblname` has sublabels, then the table `sublbltbl` shows the structure of the label value and its sublabel variables.

`[ ___ ] = getLabelValues( ___ ,'LabelRowIndex',ridx)` specifies the row index, `ridx`, of an ROI or point label whose value you want to get.

`[ ___ ] = getLabelValues( ___ ,'SublabelRowIndex',sridx)` specifies the row index, `sridx`, of an ROI or point sublabel whose value you want to get.

Examples

Get Label Values

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"
```

```
 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get the values of the labels.

```
lbls = getLabelValues(lss)
```

```
lbls=2×3 table
                 WhaleType    MoanRegions    TrillRegions
                 _________    ___________    ____________

    Member{1}      blue       {3x2 table}    {1x3 table} 
    Member{2}      blue       {3x2 table}    {1x3 table} 
```

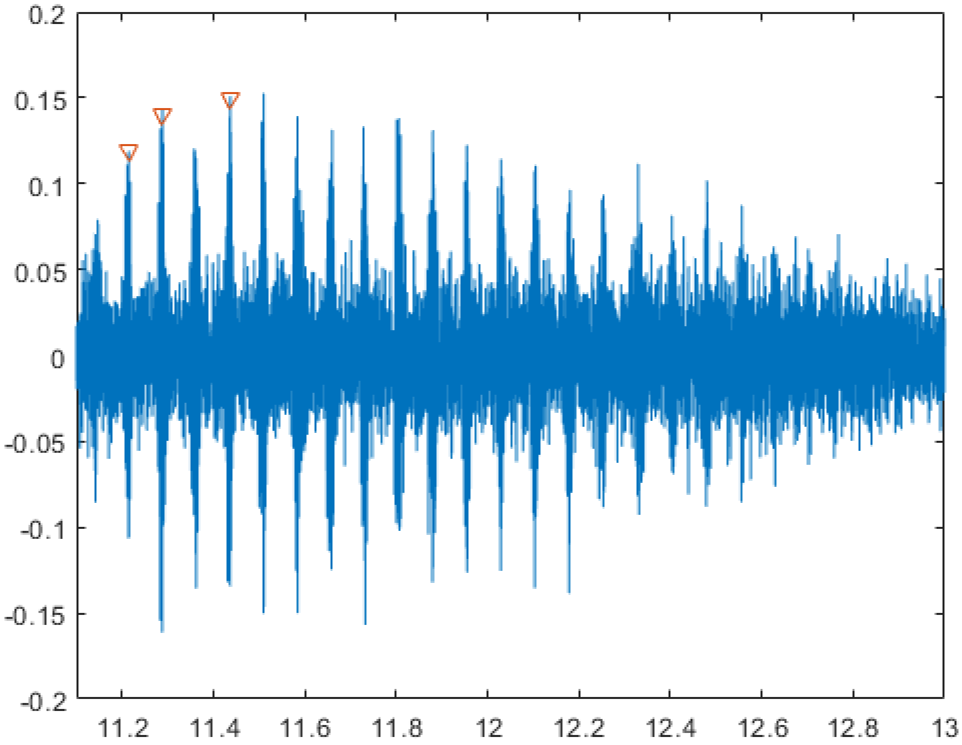
Display the moan ROI limits for the second signal of the set.

```
lbb = getLabelValues(lss,2,'MoanRegions')
```

```
lbb=3×2 table
     ROILimits      Value
    ____________    _____

     2.5     3.5    {[1]}
     5.8       8    {[1]}
    15.4    16.7    {[1]}
```

Plot the trill region of the signal between the ROI limits. Display the labeled trill peaks.

```
tvals = getLabelValues(lss,2,'TrillRegions');
peaks = getLabelValues(lss,2,{'TrillRegions','TrillPeaks'});

sg = getSignal(lss,2);
plot((0:length(sg)-1)/lss.SampleRate,sg)
xlim(tvals.ROILimits)
hold on
plot(peaks.Location,cell2mat(peaks.Value),'v')
hold off
```



Display the coordinates of the third trill peak.

```
pcoor = getLabelValues(lss,2,{'TrillRegions','TrillPeaks'}, ...
    'LabelRowIndex',1,'SublabelRowIndex',3)
```

```
pcoor=1×2 table
    Location       Value
    ________    ___________

     11.437     {[0.1500]}
```

Input Arguments

lss — Labeled signal set
labeledSignalSet object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

midx — Member row number
positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

`lblname` — Label or sublabel name
character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`ridx` — Label row index
positive integer

Label row index, specified as a positive integer. This argument applies only for ROI and point labels.

`sridx` — Sublabel row index
positive integer

Sublabel row index, specified as a positive integer. This argument applies only when a label and sublabel pair has been specified in `lblname` and the sublabel is of type ROI or point.

Output Arguments

`val` — Label values
table

Label values, returned as a table.

`sublbltbl` — Sublabel values
table

Sublabel values, returned as a table showing the structure of the label value and its sublabel variables.

- If `lblname` has no sublabels, then `sublbltbl` is empty.
- If you specify `lblname` as a string or cell array, then `sublbltbl` is empty.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

getMemberNames

Get member names in labeled signal set

Syntax

```
mnames = getMemberNames(lss)
```

Description

`mnames = getMemberNames(lss)` returns a string array containing the member names in the order in which they are stored in the labeled signal set `lss`.

Examples

Get Member Names

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Return a string array with the names of the members.

```
getMemberNames(lss)
```

```
ans = 2x1 string
    "Member{1}"
    "Member{2}"
```

Set the names of the set members to the whales' nicknames.

```
setMemberNames(lss,{'Brutus' 'Lucy'})
```

Verify that the members have the nicknames as names.

```
getMemberNames(lss)
```

```
ans = 2x1 string
    "Brutus"
    "Lucy"
```

Input Arguments

lss — Labeled signal set

labeledSignalSet object

Labeled signal set, specified as a labeledSignalSet object.

Example: labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female')) specifies a two-member set of random signals containing the attribute 'female'.

Output Arguments

mnames — Member names

string array

Member names, returned as a string array.

See Also

labeledSignalSet | signalLabelDefinition

Introduced in R2018b

getSignal

Get signals from labeled signal set

Syntax

```
[s,info] = getSignal(lss,midx)
```

Description

`[s,info] = getSignal(lss,midx)` returns the values for the signals contained in member `midx` of the labeled signal set `lss`.

Examples

Get Signal

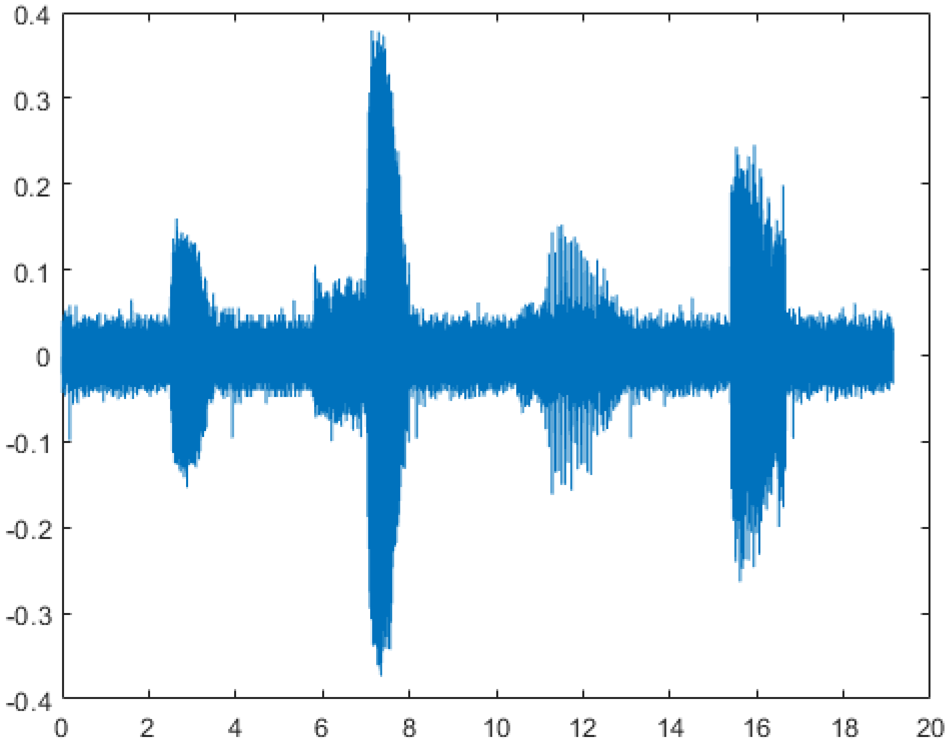
Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Retrieve the second member of the set and plot it.

```
[song,tinfo] = getSignal(lss,2);
t = (0:length(song)-1)/tinfo.SampleRate;
plot(t,song)
```



Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midx` — Member row number

positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

Output Arguments

`s` — Signal values

vector | matrix | timetable | cell array

Signal values, returned as vector, matrix, timetable, or cell array.

`info` — Time information

structure

Time information, returned as a structure.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

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)` performs the 1-D Haar discrete wavelet transform of the even-length vector, `x`. The input `x` can be univariate or multivariate data. If `x` is a matrix, `haart` operates on each column of `x`. If the length of `x` is a power of 2, the Haar transform is obtained down to level `log2(length(x))`. Otherwise, the Haar transform is obtained down to level `floor(log2(length(x)/2))`.

`[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.

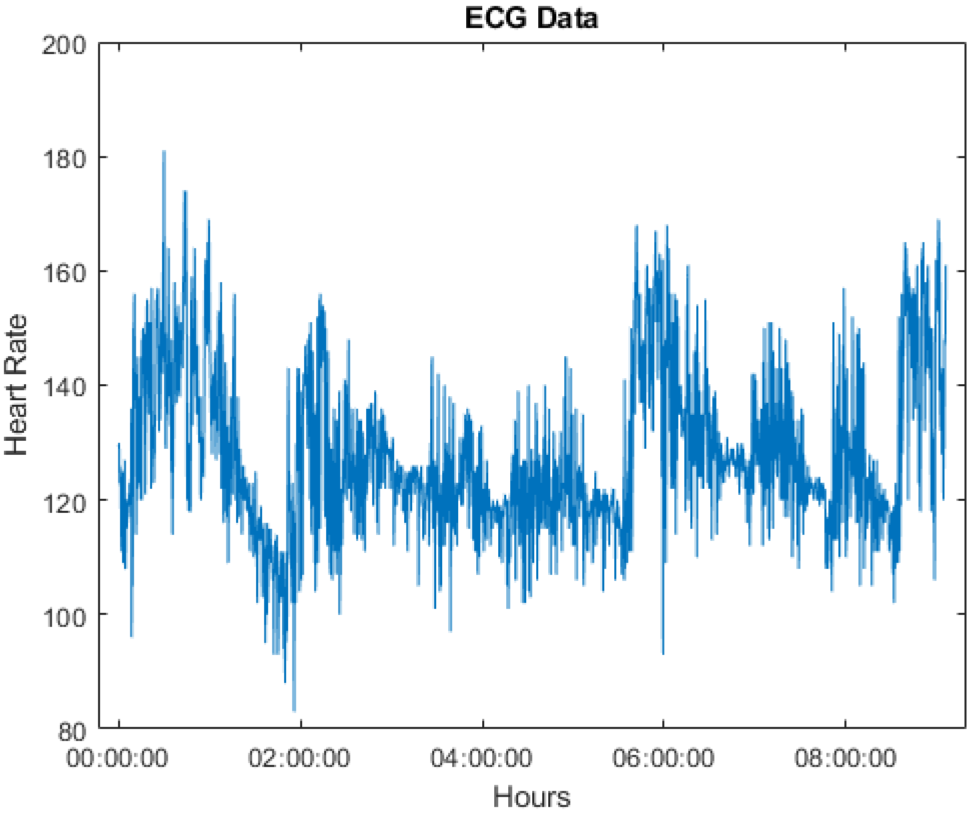
```
load elec35_nor;
signals = signals';
[a,d] = haart(signals,4);
```

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')
```

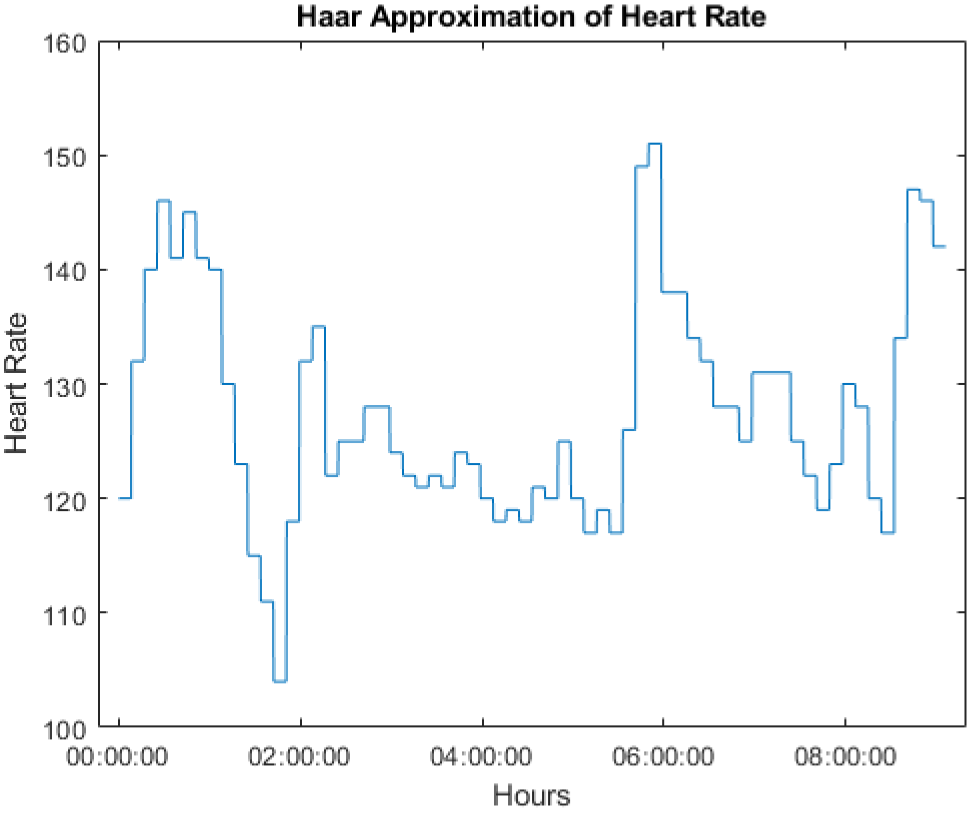


Obtain the Haar transform. Then, obtain the inverse Haar transform approximated at level 5. The scale for this level is 512 seconds, which is 2^5 times the sampling interval (16 seconds).

```
[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')
```



Input Arguments

x — Input signal

vector | matrix

Input signal, specified as a vector or matrix. 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`.

Data Types: `single` | `double`

`level` — Maximum level

positive integer

Maximum level to which to perform the Haar transform, specified as a positive integer.

- If the length of `x` is a power of two, `level` is a positive integer less than or equal to `log2(length(x))`.
- If the length of `x` is even, but not a power of two, `level` is a positive integer less than or equal to `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 either `'noninteger'` or `'integer'`. `'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. If `x` is single precision, the Haar transform coefficients are single precision. For all other numeric type, the numeric type of the coefficients is double precision.

Output Arguments

a — Approximation coefficients

scalar | vector | matrix

Approximation coefficients at the coarsest level, 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: `single` | `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 wavelet coefficients. The number of detail coefficients depends on the selected level and the length of the input. If `d` is a cell array, the elements of `d` are ordered from finest to coarsest resolution.

Note: Generated C and C++ code always returns the wavelet coefficients `d` in a cell array.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`ihaart` | `ihaart2` | `haart2`

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`. `x` is a 2-D, 3-D, or 4-D matrix with even length row and column dimensions. If `x` is 4-D, the dimensions are Spatial-by-Spatial-by-Channel-by-Batch. The Haar transform is always computed along the row and column dimensions of the input. If the row and column dimensions of `x` are powers of two, the Haar transform is obtained down to level `log2(min(size(x,[1 2])))`. If the row or column dimension of `x` is even, but not a power of two, the Haar transform is obtained down to level `floor(log2(min(size(x,[1 2])/2)))`.

`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.

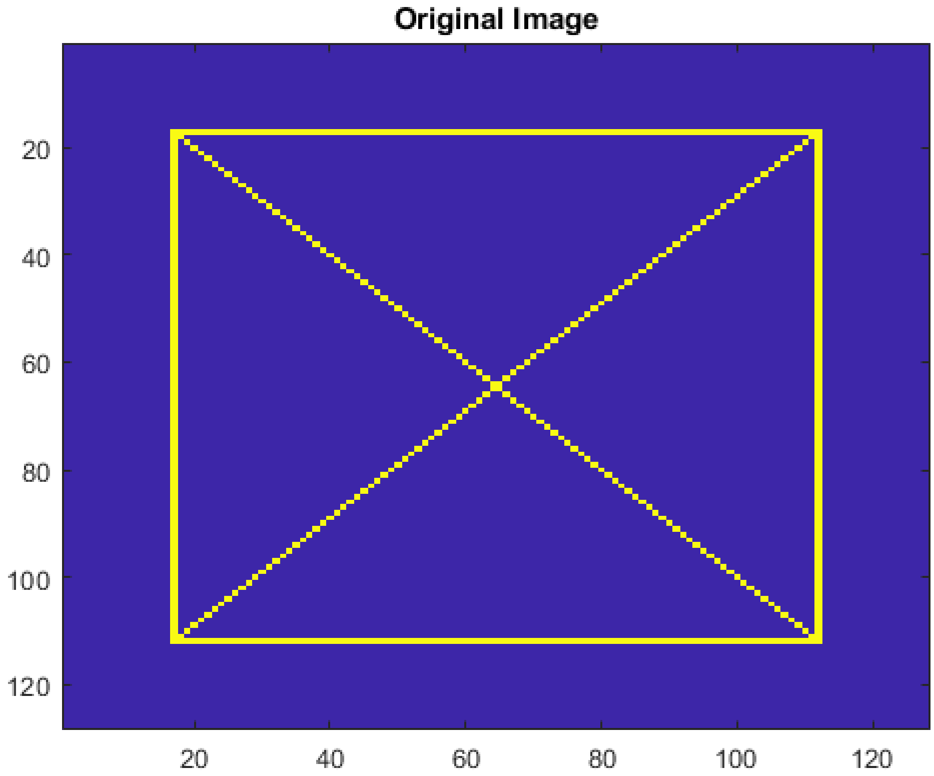
`[a,h,v,d] = haart2( ___ ,integerflag)` specifies how the 2-D Haar transform handles integer-valued data, using any of the previous syntaxes.

Examples

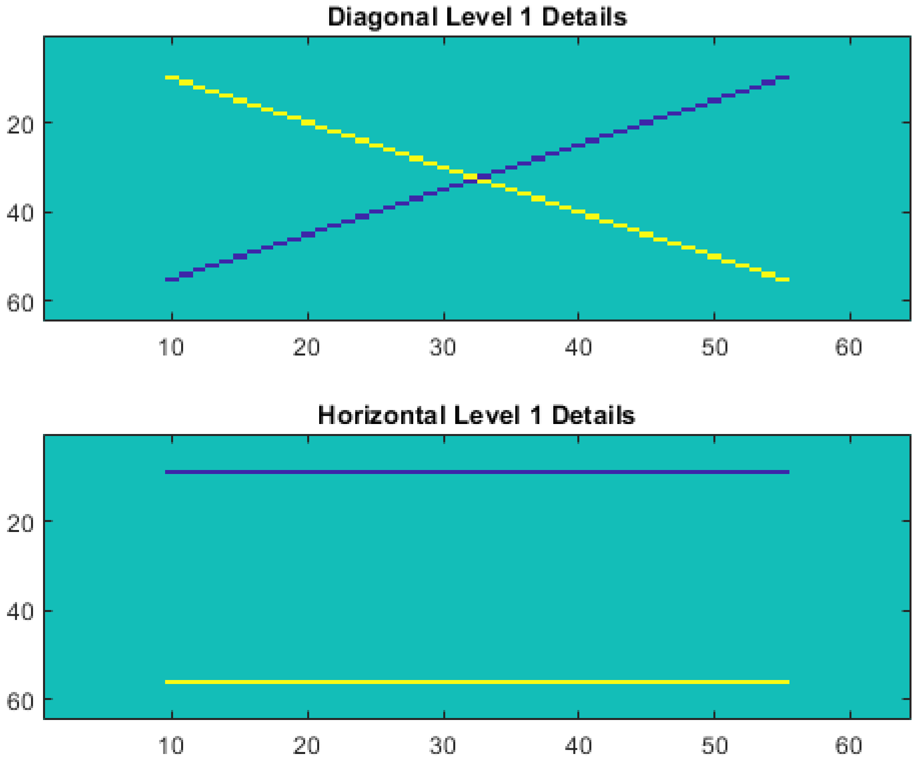
Haar Transform and First Level Details of 2-D Data

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

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



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

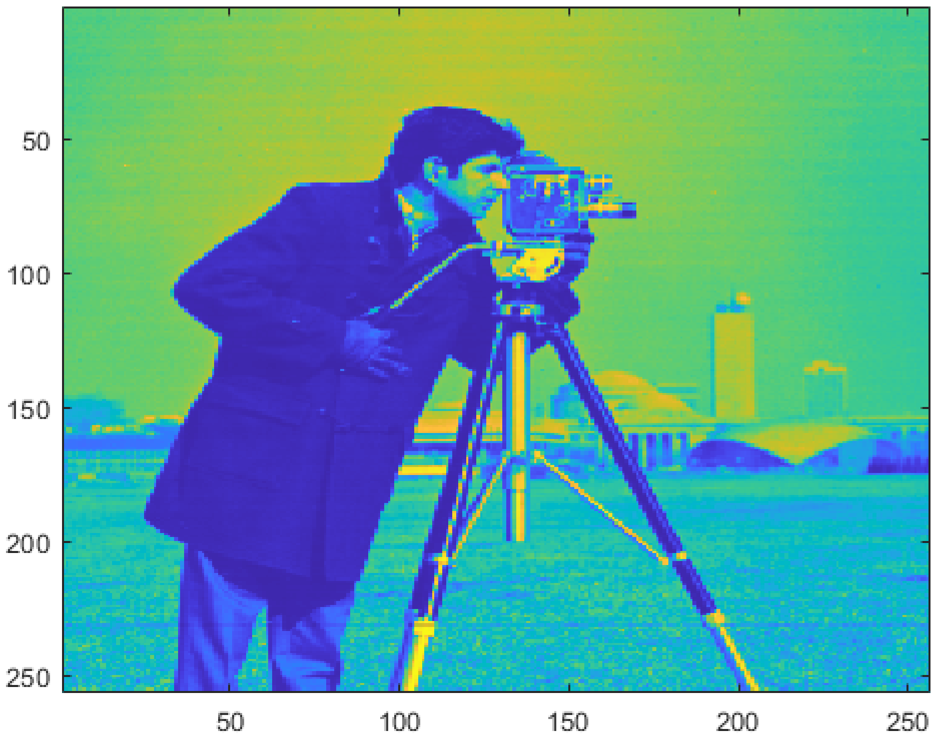


Haar Transform of Image Down to a Specified Level

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

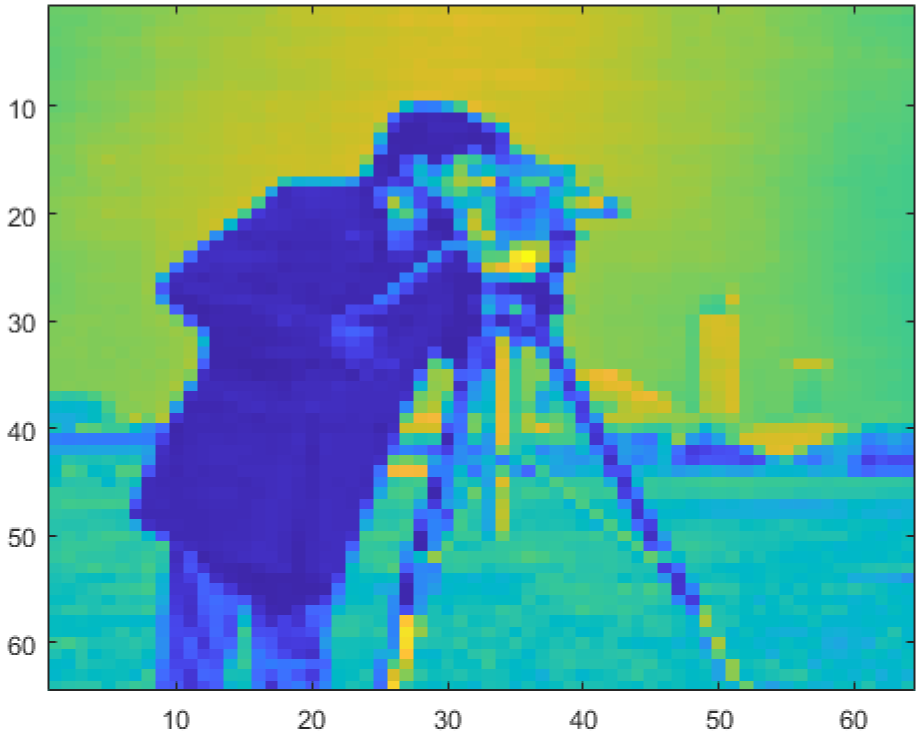
Load and display the image of a cameraman.

```
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
```

Input Arguments

x — Input signal

matrix

Input signal, specified as a 2-D, 3-D, or 4-D real-valued matrix. If x is 4-D, the dimensions are Spatial-by-Spatial-by-Channel-by-Batch. The row and column sizes of x must be even length.

Data Types: single | double

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,[1 2]))).
- 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,[1 2])/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 either 'noninteger' or 'integer'. '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 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. If x is a single-precision input, the numeric type of the Haar transform coefficients is single precision. For all other numeric types, the numeric type of the coefficients is double precision.

Output Arguments

a — Approximation coefficients

scalar | matrix

Approximation coefficients at the coarsest scale, 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: single | 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.

Note: Generated C and C++ code always returns the horizontal detail coefficients h in a cell array.

Data Types: single | 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.

Note: Generated C and C++ code always returns the vertical detail coefficients v in a cell array.

Data Types: single | 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.

Note: Generated C and C++ code always returns the diagonal detail coefficients d in a cell array.

Data Types: single | double

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

ihaart | ihaart2 | haart

Topics

“Haar Transforms for Time Series Data and Images”

Introduced in R2016b

head

Get top rows of labels table

Syntax

```
val = head(lss)
```

Description

`val = head(lss)` returns the top rows of the labels table of the labeled signal set `lss`.

Examples

Top Label Values

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get the top rows of the labels table.

```
head(lss)
```

```
ans=2×3 table
                 WhaleType    MoanRegions    TrillRegions
                 _________    ___________    ____________

    Member{1}      blue       {3x2 table}    {1x3 table} 
    Member{2}      blue       {3x2 table}    {1x3 table} 
```

Input Arguments

`lss` — Labeled signal set
`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

Output Arguments

`val` — Top rows of labels

table

Top rows of labels, returned as a table.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

hht

Hilbert-Huang transform

Syntax

```
hs = hht(imf)
hs = hht(imf,fs)
[hs,f,t] = hht( ___ )
[hs,f,t,imfinsf,imfinse] = hht( ___ )
[ ___ ] = hht( ___ ,Name,Value)

hht( ___ )
hht( ___ ,freqlocation)
```

Description

`hs = hht(imf)` returns the Hilbert spectrum `hs` of the signal specified by intrinsic mode functions `imf`. `hs` is useful for analyzing signals that comprise a mixture of signals whose spectral content changes in time. Use `hht` to perform Hilbert spectral analysis on signals to identify localized features.

`hs = hht(imf,fs)` returns the Hilbert spectrum `hs` of a signal sampled at a rate `fs`.

`[hs,f,t] = hht( ___ )` returns frequency vector `f` and time vector `t` in addition to `hs`. These output arguments can be used with either of the previous input syntaxes.

`[hs,f,t,imfinsf,imfinse] = hht( ___ )` also returns the instantaneous frequencies `imfinsf` and the instantaneous energies `imfinse` of the intrinsic mode functions for signal diagnostics.

`[ ___ ] = hht( ___ ,Name,Value)` estimates Hilbert spectrum parameters with additional options specified by one or more `Name,Value` pair arguments.

`hht( ___ )` with no output arguments plots the Hilbert spectrum in the current figure window. You can use this syntax with any of the input arguments in previous syntaxes.

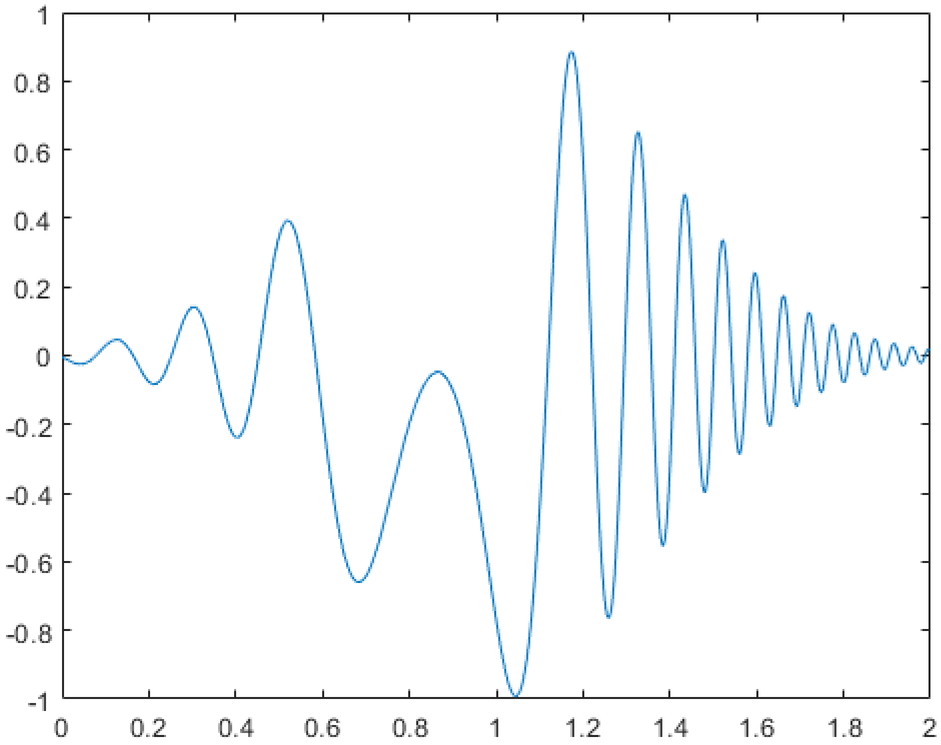
`hht( ___ ,freqlocation)` plots the Hilbert spectrum with the optional `freqlocation` argument to specify the location of the frequency axis. Frequency is represented on the *y*-axis by default.

Examples

Hilbert Spectrum of Quadratic Chirp

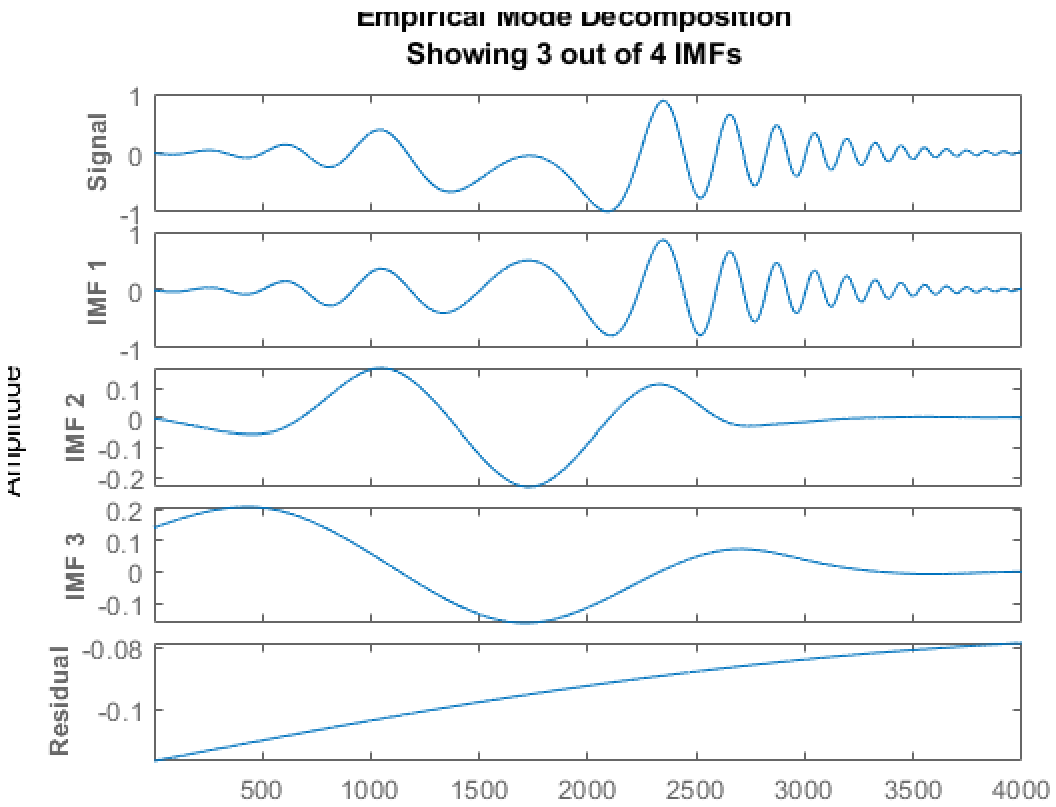
Generate a Gaussian-modulated quadratic chirp. Specify a sample rate of 2 kHz and a signal duration of 2 seconds.

```
fs = 2000;
t = 0:1/fs:2-1/fs;
q = chirp(t-2,4,1/2,6,'quadratic',100,'convex').*exp(-4*(t-1).^2);
plot(t,q)
```



Use `emd` to visualize the intrinsic mode functions (IMFs) and the residual.

```
emd(q)
```



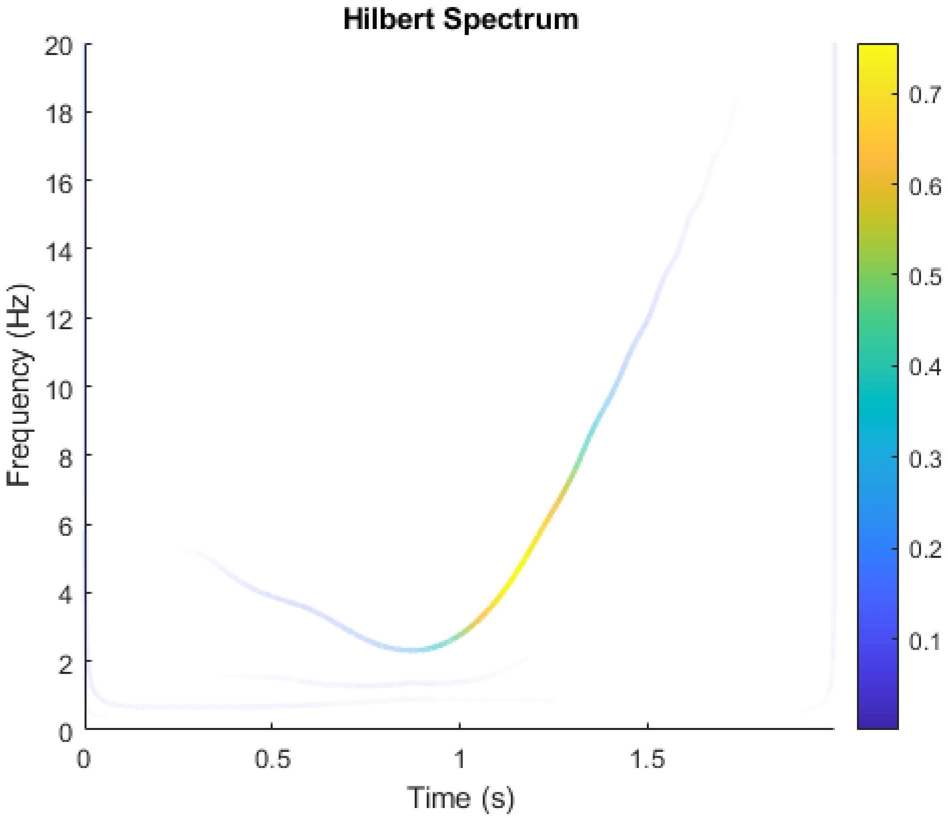
Compute the IMFs of the signal. Use the `'Display'` name-value pair to output a table showing the number of sifting iterations, the relative tolerance, and the sifting stop criterion for each IMF.

```
imf = emd(q,'Display',1);
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit
      1      |        2     |    0.0063952   |  SiftMaxRelativeTolerance
      2      |        2     |       0.1007   |  SiftMaxRelativeTolerance
      3      |        2     |      0.01189   |  SiftMaxRelativeTolerance
      4      |        2     |    0.0075124   |  SiftMaxRelativeTolerance
Decomposition stopped because the number of extrema in the residual signal is less than the 'MaxN
```

Use the computed IMFs to plot the Hilbert spectrum of the quadratic chirp. Restrict the frequency range from 0 Hz to 20 Hz.

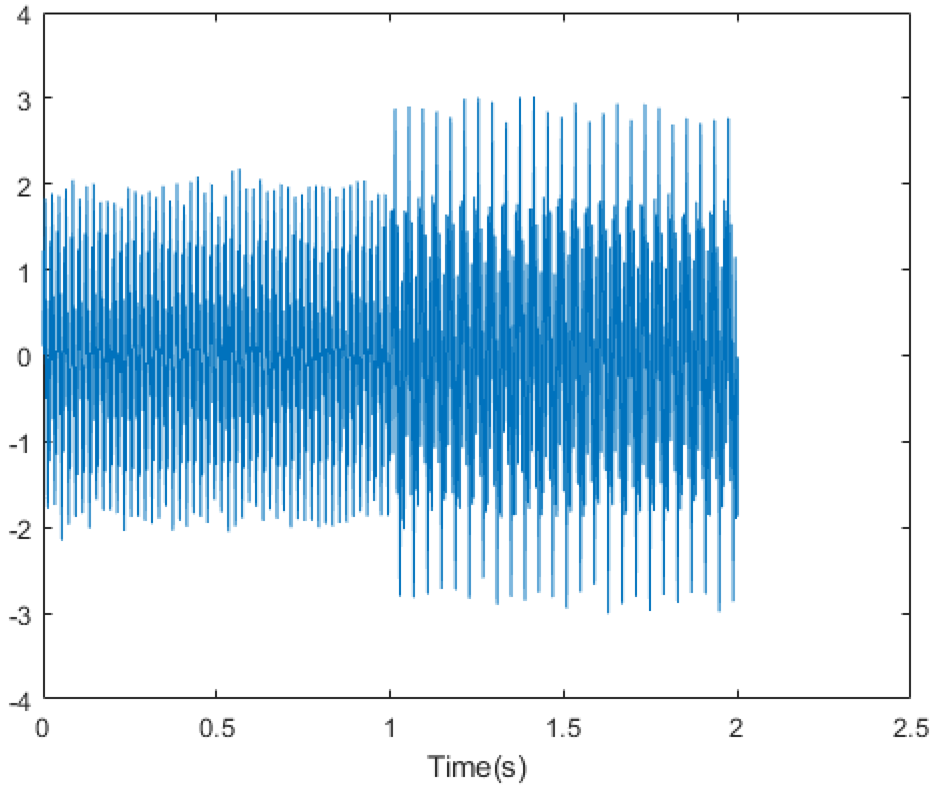
```
hht(imf,fs,'FrequencyLimits',[0 20])
```



Perform Empirical Mode Decomposition and Visualize Hilbert Spectrum of Signal

Load and visualize a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The vibration of a jackhammer and the sound of fireworks are examples of nonstationary continuous signals. The signal is sampled at a rate `fs`.

```
load('sinusoidalSignalExampleData.mat','X','fs')
t = (0:length(X)-1)/fs;
plot(t,X)
xlabel('Time(s)')
```



The mixed signal contains sinusoidal waves with different amplitude and frequency values.

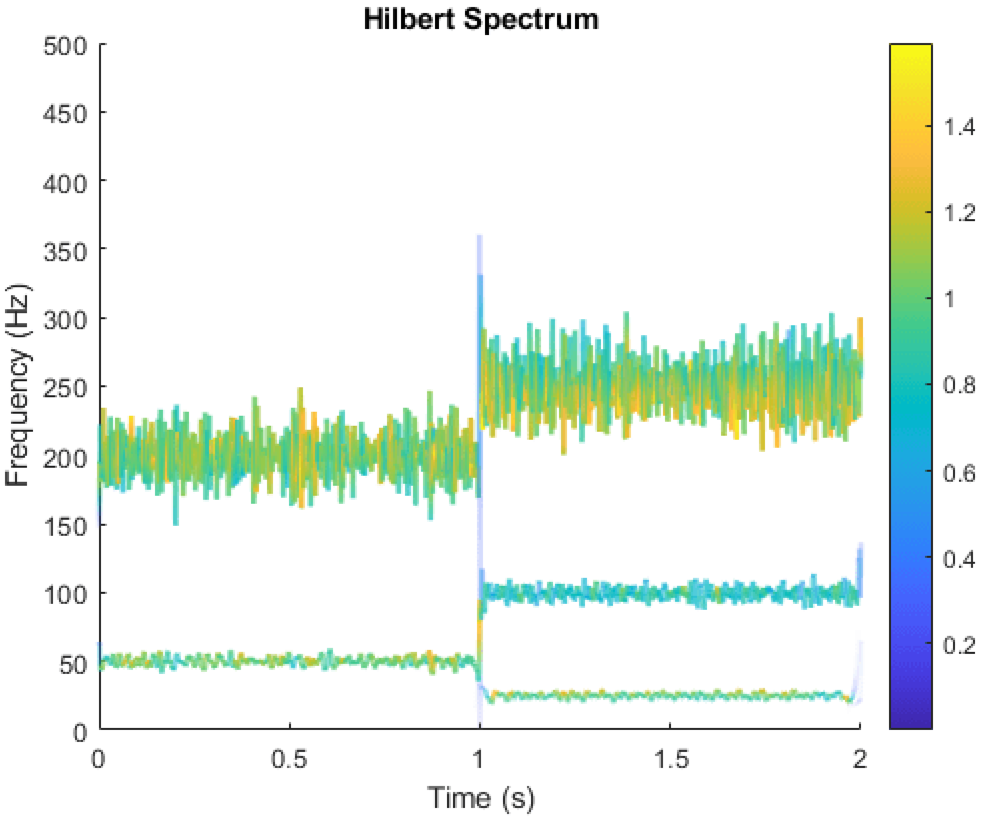
To create the Hilbert spectrum plot, you need the intrinsic mode functions (IMFs) of the signal. Perform empirical mode decomposition to compute the IMFs and residuals of the signal. Since the signal is not smooth, specify 'pchip' as the interpolation method.

```
[imf,residual,info] = emd(X,'Interpolation','pchip');
```

The table generated in the command window indicates the number of sift iterations, the relative tolerance, and the sift stop criterion for each generated IMF. This information is also contained in `info`. You can hide the table by adding the `'Display',0` name value pair.

Create the Hilbert spectrum plot using the `imf` components obtained using empirical mode decomposition.

```
hht(imf,fs)
```

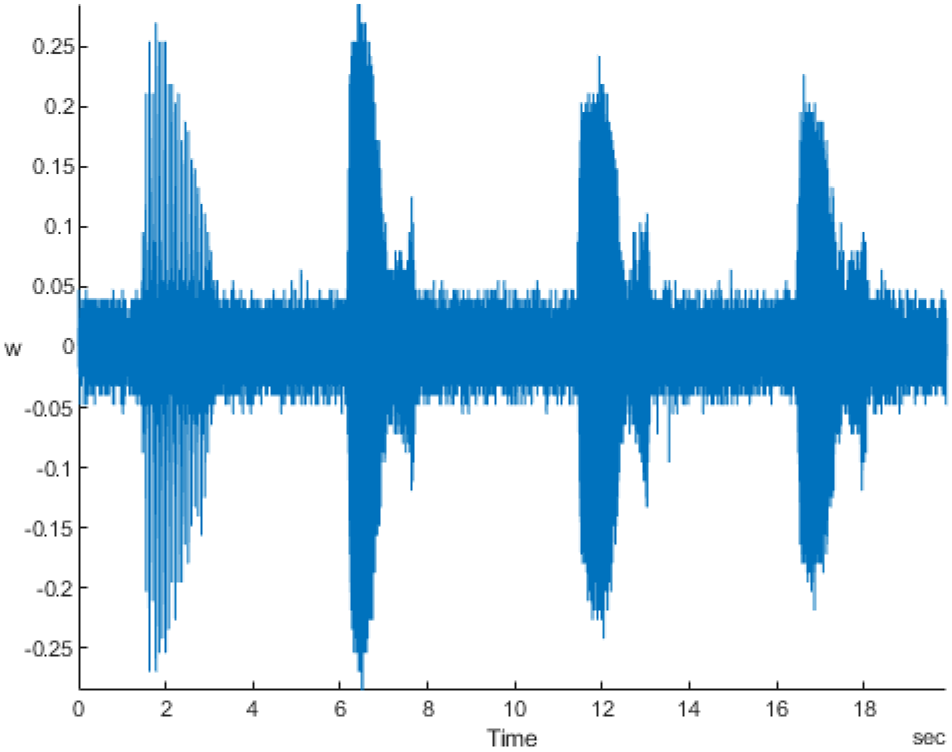


The frequency versus time plot is a sparse plot with a vertical color bar indicating the instantaneous energy at each point in the IMF. The plot represents the instantaneous frequency spectrum of each component decomposed from the original mixed signal. Three IMFs appear in the plot with a distinct change in frequency at 1 second.

Hilbert Spectrum of Whale Song

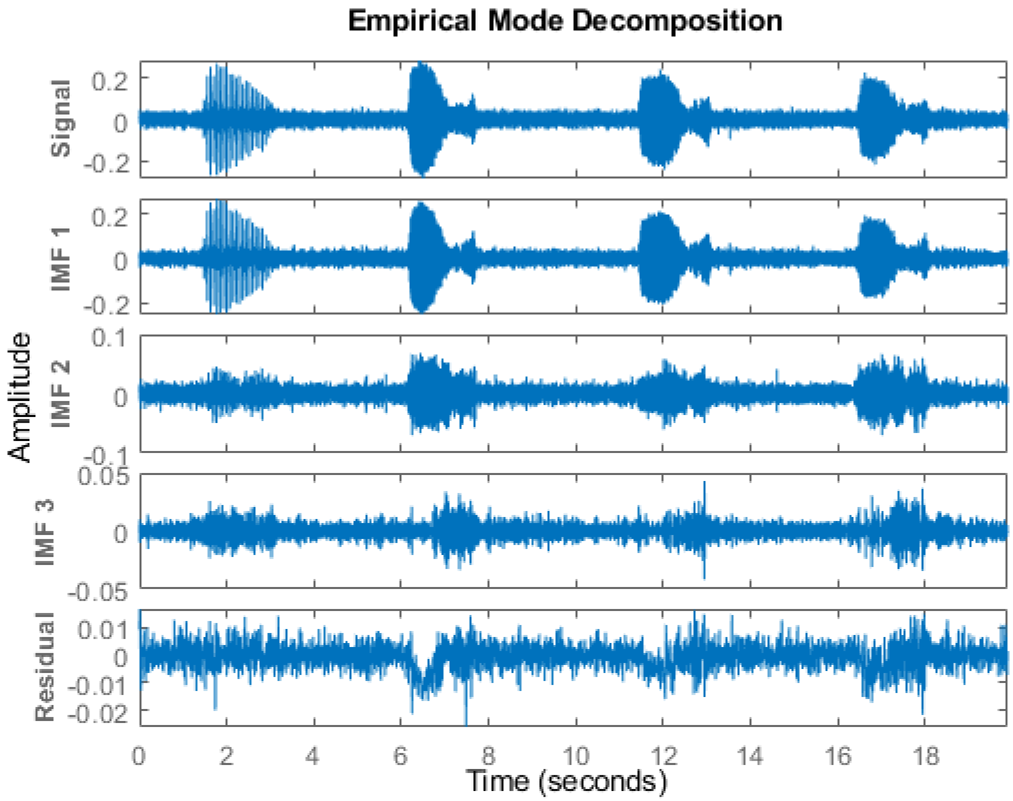
Load a file that contains audio data from a Pacific blue whale, sampled at 4 kHz. The file is from the library of animal vocalizations maintained by the Cornell University Bioacoustics Research Program. The time scale in the data is compressed by a factor of 10 to raise the pitch and make the calls more audible. Convert the signal to a MATLAB® timetable and plot it. Four features stand out from the noise in the signal. The first is known as a *trill*, and the other three are known as *moans*.

```
[w,fs] = audioread('bluewhale.wav');
whale = timetable(w,'SampleRate',fs);
stackedplot(whale);
```



Use emd to visualize the first three intrinsic mode functions (IMFs) and the residual.

```
emd(whale,'MaxNumIMF',3)
```



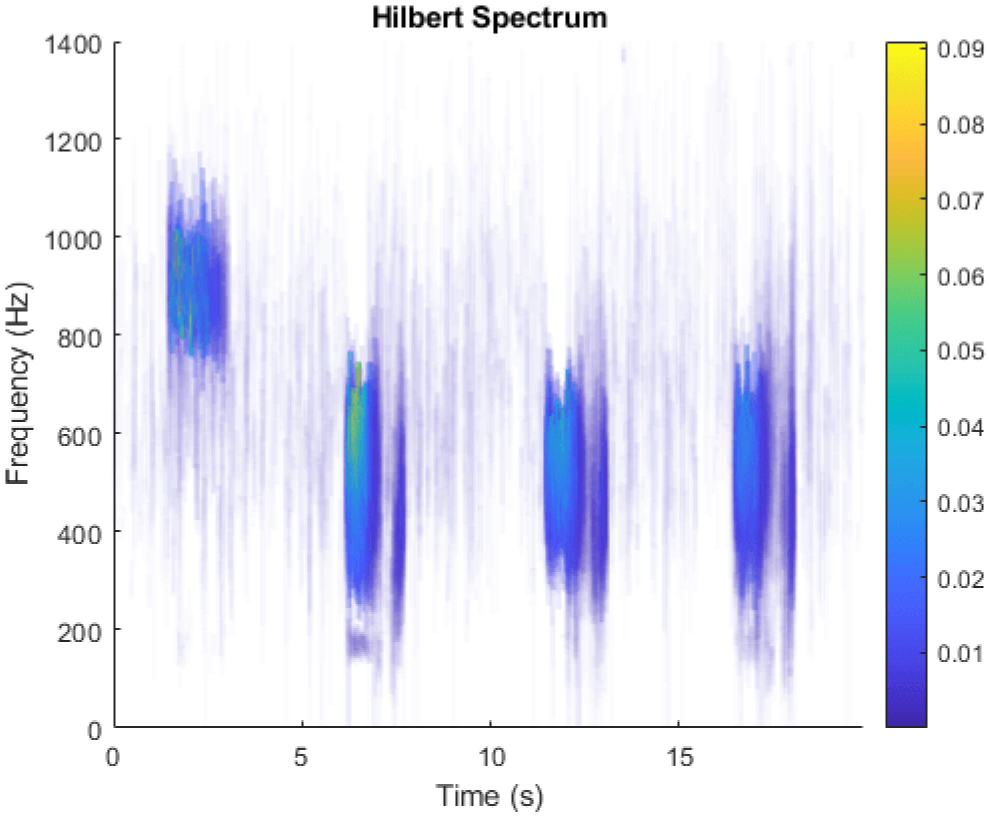
Compute the first three IMFs of the signal. Use the `'Display'` name-value pair to output a table showing the number of sifting iterations, the relative tolerance, and the sifting stop criterion for each IMF.

```
imf = emd(whale,'MaxNumIMF',3,'Display',1);
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit
      1      |        1     |       0.13523  |  SiftMaxRelativeTolerance
      2      |        2     |      0.030198  |  SiftMaxRelativeTolerance
      3      |        2     |       0.01908  |  SiftMaxRelativeTolerance
Decomposition stopped because maximum number of intrinsic mode functions was extracted.
```

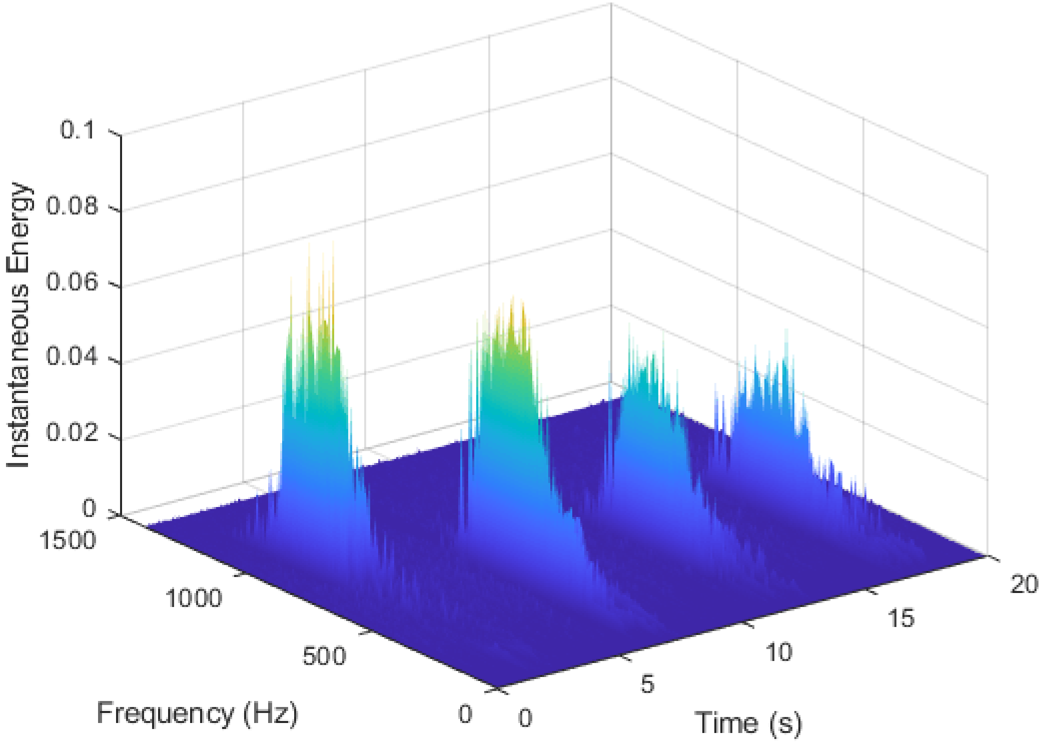
Use the computed IMFs to plot the Hilbert spectrum of the signal. Restrict the frequency range from 0 Hz to 1400 Hz.

```
hht(imf,'FrequencyLimits',[0 1400])
```



Compute the Hilbert spectrum for the same range of frequencies. Visualize the Hilbert spectra of the trill and moans as a mesh plot.

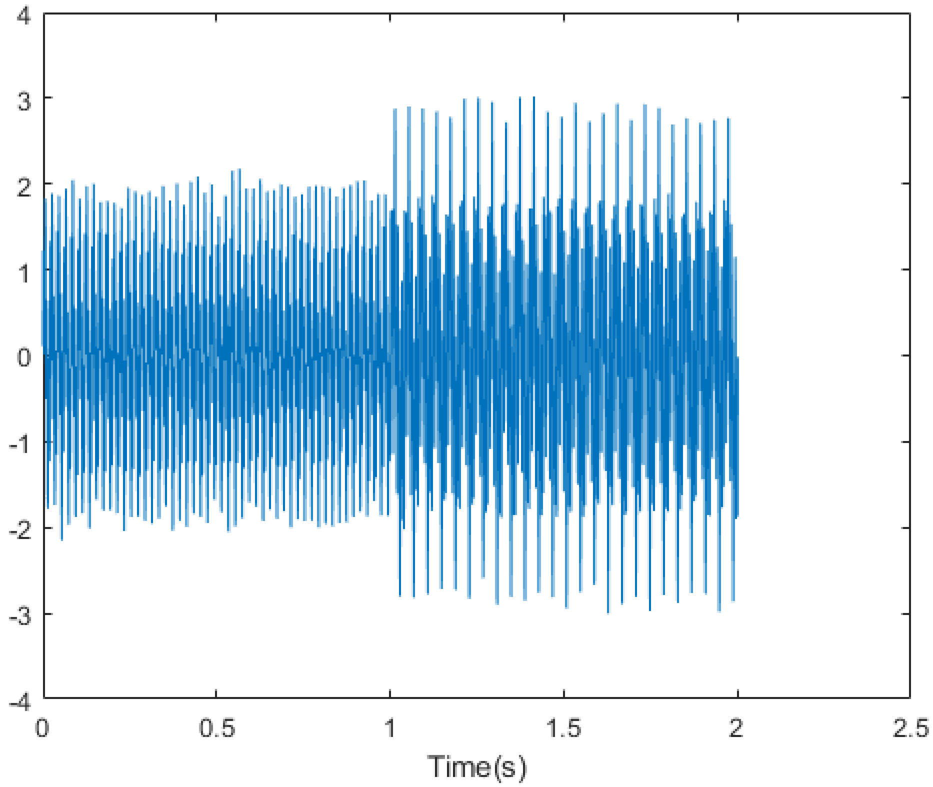
```
[hs,f,t] = hht(imf,'FrequencyLimits',[0 1400]);

mesh(seconds(t),f,hs,'EdgeColor','none','FaceColor','interp')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
zlabel('Instantaneous Energy')
```



Compute Hilbert Spectrum Parameters of Signal

Load and visualize a nonstationary continuous signal composed of sinusoidal waves with a distinct change in frequency. The vibration of a jackhammer and the sound of fireworks are examples of nonstationary continuous signals. The signal is sampled at a rate `fs`.

```
load('sinusoidalSignalExampleData.mat','X','fs')
t = (0:length(X)-1)/fs;
plot(t,X)
xlabel('Time(s)')
```



The mixed signal contains sinusoidal waves with different amplitude and frequency values.

To compute the Hilbert spectrum parameters, you need the IMFs of the signal. Perform empirical mode decomposition to compute the intrinsic mode functions and residuals of the signal. Since the signal is not smooth, specify `'pchip'` as the interpolation method.

```
[imf,residual,info] = emd(X,'Interpolation','pchip');
```

The table generated in the command window indicates the number of sift iterations, the relative tolerance, and the sift stop criterion for each generated IMF. This information is also contained in `info`. You can hide the table by specifying `'Display'` as `0`.

Compute the Hilbert spectrum parameters: Hilbert spectrum `hs`, frequency vector `f`, time vector `t`, instantaneous frequency `imfinsf`, and instantaneous energy `imfinse`.

```
[hs,f,t,imfinsf,imfinse] = hht(imf,fs);
```

Use the computed Hilbert spectrum parameters for time-frequency analysis and signal diagnostics.

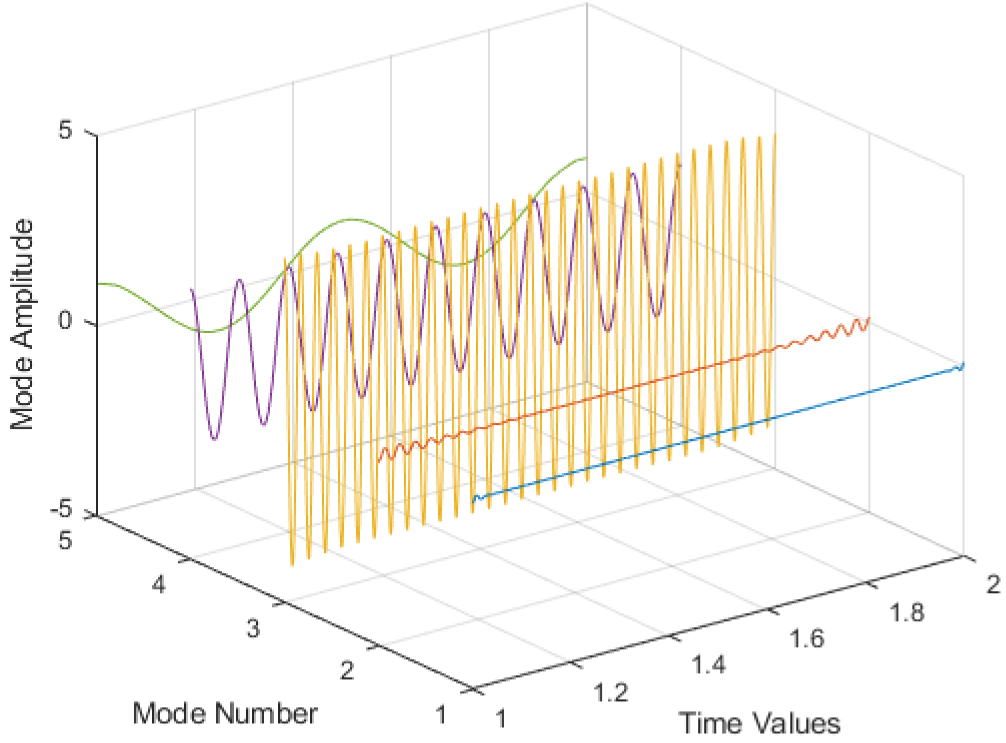
VMD of Multicomponent Signal

Generate a multicomponent signal consisting of three sinusoids of frequencies 2 Hz, 10 Hz, and 30 Hz. The sinusoids are sampled at 1 kHz for 2 seconds. Embed the signal in white Gaussian noise of variance 0.01^2.

```
fs = 1e3;
t = 1:1/fs:2-1/fs;
x = cos(2*pi*2*t) + 2*cos(2*pi*10*t) + 4*cos(2*pi*30*t) + 0.01*randn(1,length(t));
```

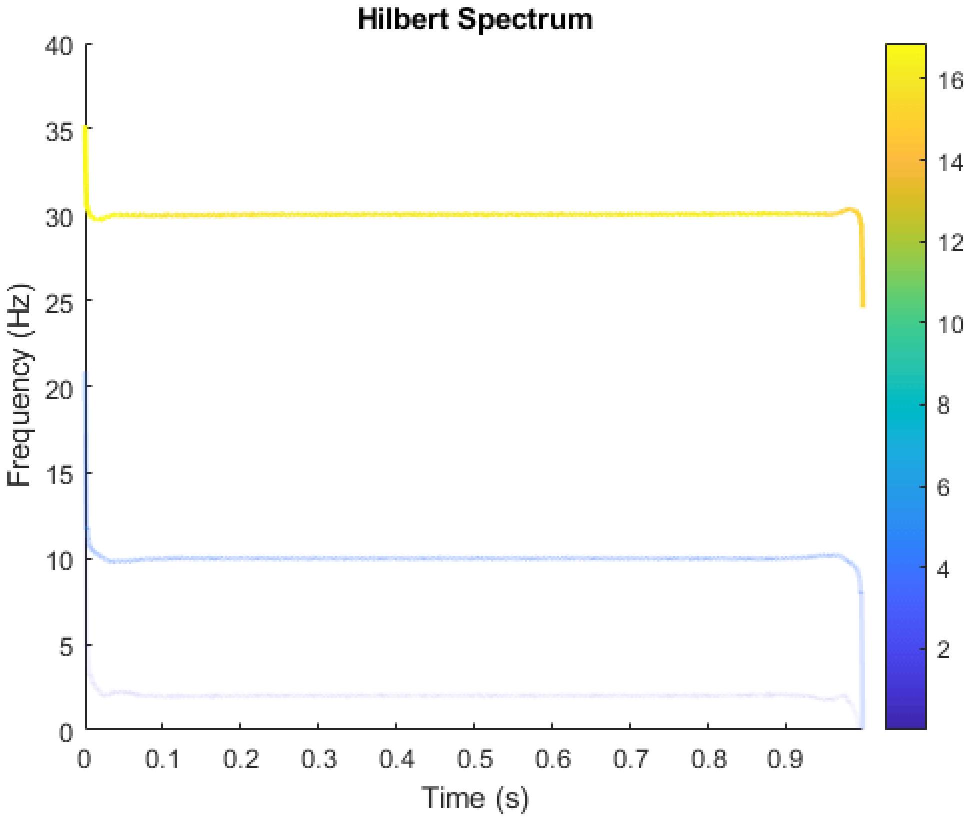
Compute the IMFs of the noisy signal and visualize them in a 3-D plot.

```
imf = vmd(x);
[p,q] = ndgrid(t,1:size(imf,2));
plot3(p,q,imf)
grid on
xlabel('Time Values')
ylabel('Mode Number')
zlabel('Mode Amplitude')
```



Use the computed IMFs to plot the Hilbert spectrum of the multicomponent signal. Restrict the frequency range to [0, 40] Hz.

```
hht(imf,fs,'FrequencyLimits',[0,40])
```

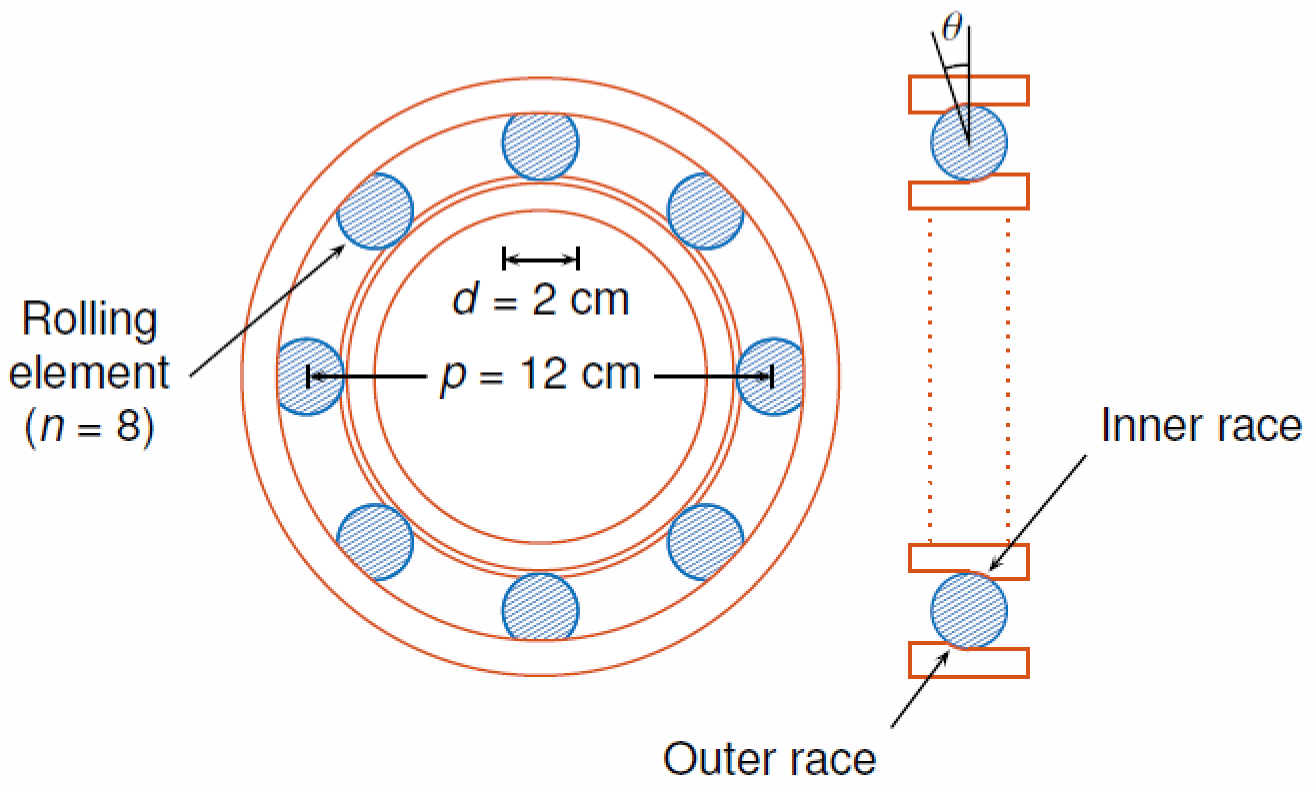


Compute Hilbert Spectrum of Vibration Signal

Simulate a vibration signal from a damaged bearing. Compute the Hilbert spectrum of this signal and look for defects.

A bearing with a pitch diameter of 12 cm has eight rolling elements. Each rolling element has a diameter of 2 cm. The outer race remains stationary as the inner race is driven at 25 cycles per second. An accelerometer samples the bearing vibrations at 10 kHz.

```
fs = 10000;
f0 = 25;
n = 8;
d = 0.02;
p = 0.12;
```



The vibration signal from the healthy bearing includes several orders of the driving frequency.

```
t = 0:1/fs:10-1/fs;
yHealthy = [1 0.5 0.2 0.1 0.05]*sin(2*pi*f0*[1 2 3 4 5]'.*t)/5;
```

A resonance is excited in the bearing vibration halfway through the measurement process.

```
yHealthy = (1+1./(1+linspace(-10,10,length(yHealthy)).^4)).*yHealthy;
```

The resonance introduces a defect in the outer race of the bearing that results in progressive wear. The defect causes a series of impacts that recur at the ball pass frequency outer race (BPFO) of the bearing:

$$\mathrm{BPFO} = \frac{1}{2}nf_0\left[1 - \frac{d}{p}\cos\theta\right],$$

where f_0 is the driving rate, n is the number of rolling elements, d is the diameter of the rolling elements, p is the pitch diameter of the bearing, and θ is the bearing contact angle. Assume a contact angle of 15° and compute the BPFO.

```
ca = 15;
bpfo = n*f0/2*(1-d/p*cosd(ca));
```

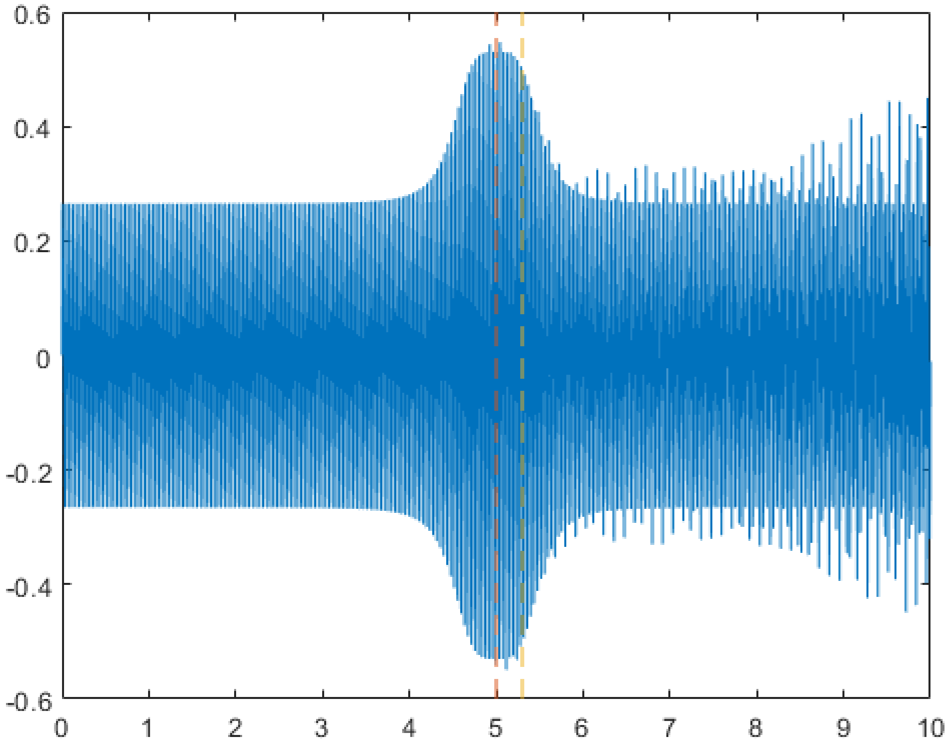
Use the `pulstran` (Signal Processing Toolbox) function to model the impacts as a periodic train of 5-millisecond sinusoids. Each 3 kHz sinusoid is windowed by a flat top window. Use a power law to introduce progressive wear in the bearing vibration signal.

```
fImpact = 3000;
tImpact = 0:1/fs:5e-3-1/fs;
```

```
wImpact = flattopwin(length(tImpact))'/10;
xImpact = sin(2*pi*fImpact*tImpact).*wImpact;

tx = 0:1/bpfo:t(end);
tx = [tx; 1.3.^tx-2];

nWear = 49000;
nSamples = 100000;
yImpact = pulstran(t,tx',xImpact,fs)/5;
yImpact = [zeros(1,nWear) yImpact(1,(nWear+1):nSamples)];
```

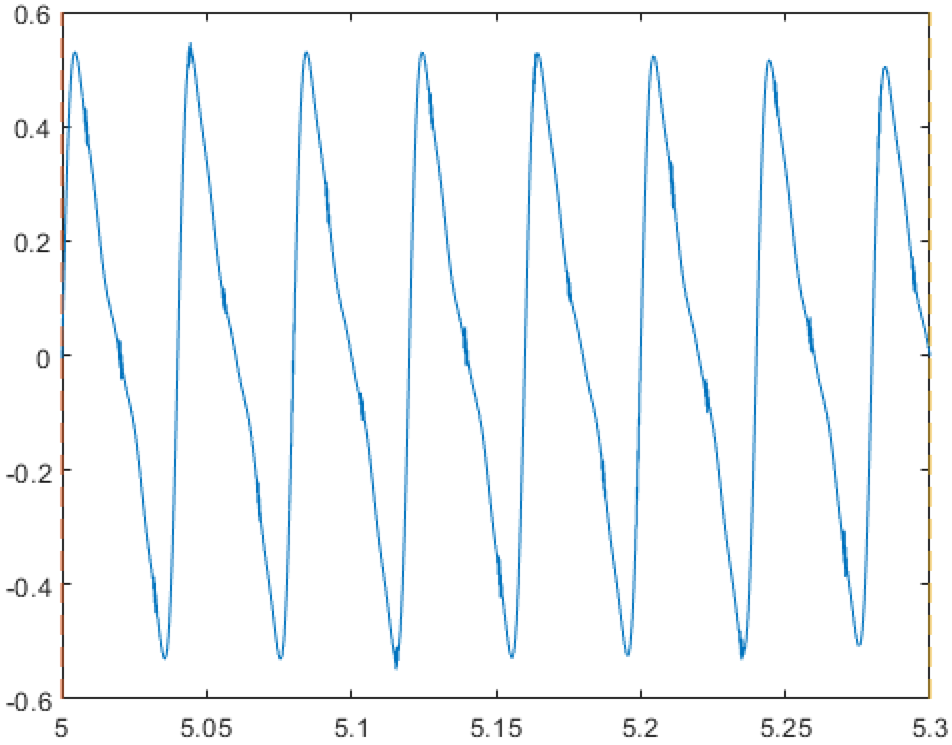
Generate the BPFO vibration signal by adding the impacts to the healthy bearing signal. Plot the signal and select a 0.3-second interval starting at 5.0 seconds.

```
yBPFO = yImpact + yHealthy;

xLimLeft = 5.0;
xLimRight = 5.3;
yMin = -0.6;
yMax = 0.6;

plot(t,yBPFO)

hold on
[limLeft,limRight] = meshgrid([xLimLeft xLimRight],[yMin yMax]);
plot(limLeft,limRight,'--')
hold off
```



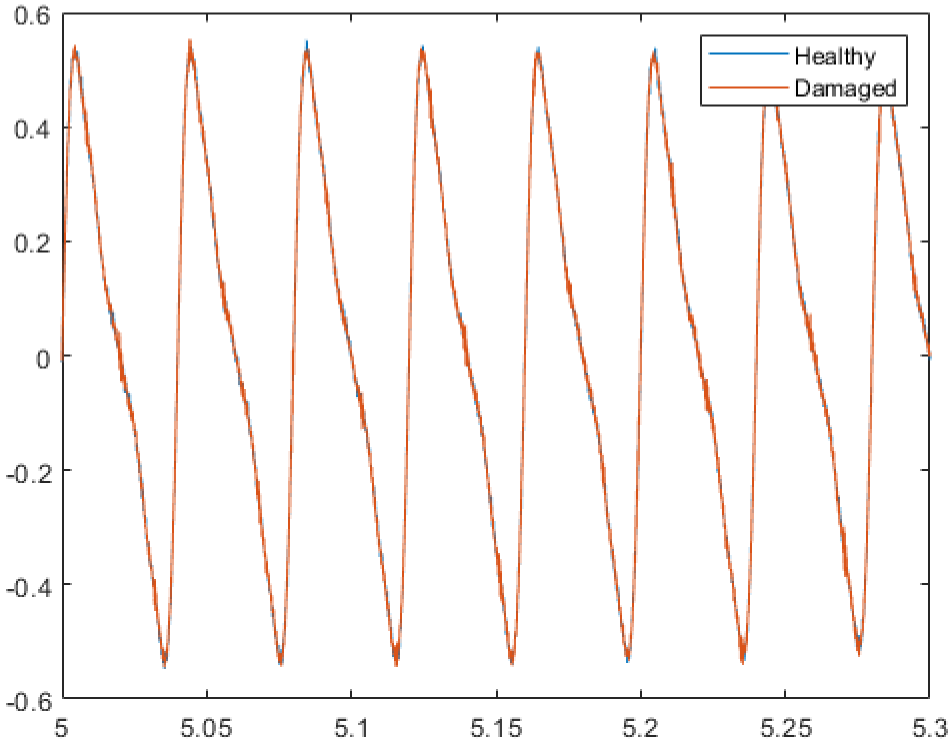
Zoom in on the selected interval to visualize the effect of the impacts.

```
xlim([xLimLeft xLimRight])
```



Add white Gaussian noise to the signals. Specify a noise variance of $1/150^2$.

```
rn = 150;
yGood = yHealthy + randn(size(yHealthy))/rn;
yBad = yBPFO + randn(size(yHealthy))/rn;

plot(t,yGood,t,yBad)
xlim([xLimLeft xLimRight])
legend('Healthy','Damaged')
```



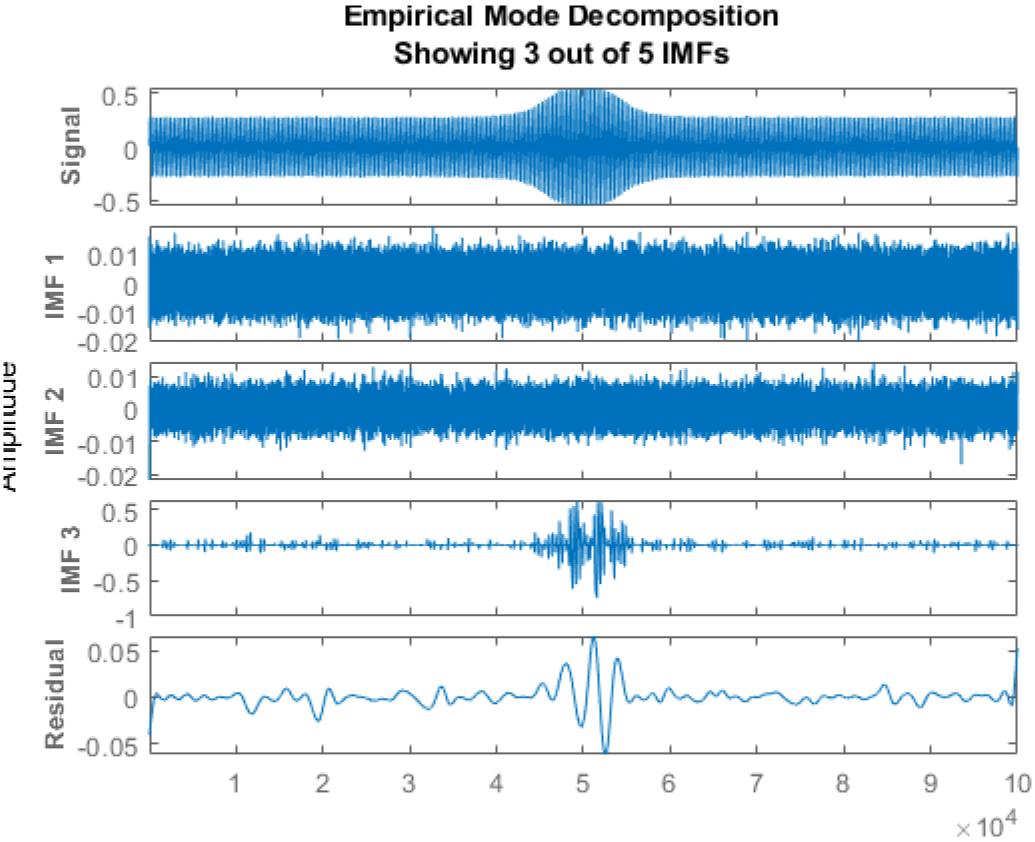
Use emd (Signal Processing Toolbox) to perform an empirical mode decomposition of the healthy bearing signal. Compute the first five intrinsic mode functions (IMFs). Use the `'Display'` name-value argument to output a table showing the number of sifting iterations, the relative tolerance, and the sifting stop criterion for each IMF.

```
imfGood = emd(yGood,'MaxNumIMF',5,'Display',1);
```

```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit  
      1      |        3     |      0.017132  |  SiftMaxRelativeTolerance
      2      |        3     |       0.12694  |  SiftMaxRelativeTolerance
      3      |        6     |       0.14582  |  SiftMaxRelativeTolerance
      4      |        1     |      0.011082  |  SiftMaxRelativeTolerance
      5      |        2     |       0.03463  |  SiftMaxRelativeTolerance
Decomposition stopped because maximum number of intrinsic mode functions was extracted.
```

Use emd without output arguments to visualize the first three IMFs and the residual.

```
emd(yGood,'MaxNumIMF',5)
```

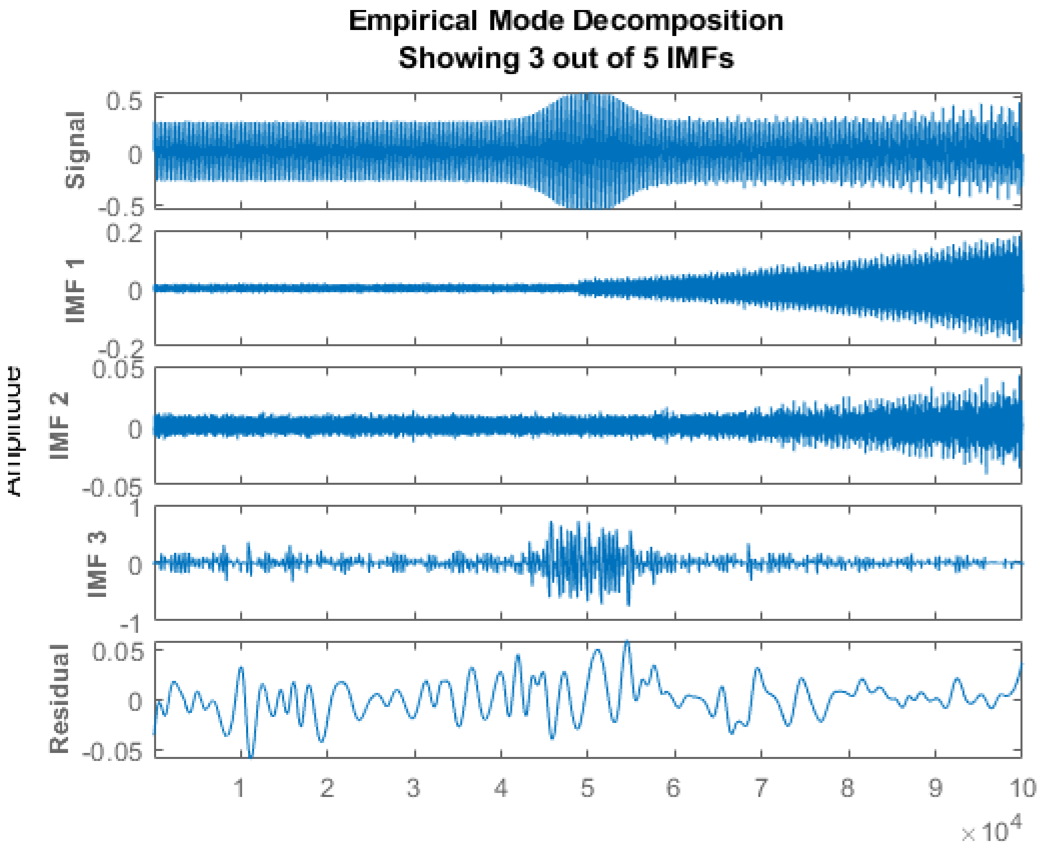


Compute and visualize the IMFs of the defective bearing signal. The first empirical mode reveals the high-frequency impacts. This high-frequency mode increases in energy as the wear progresses.

```
imfBad = emd(yBad,'MaxNumIMF',5,'Display',1);
```

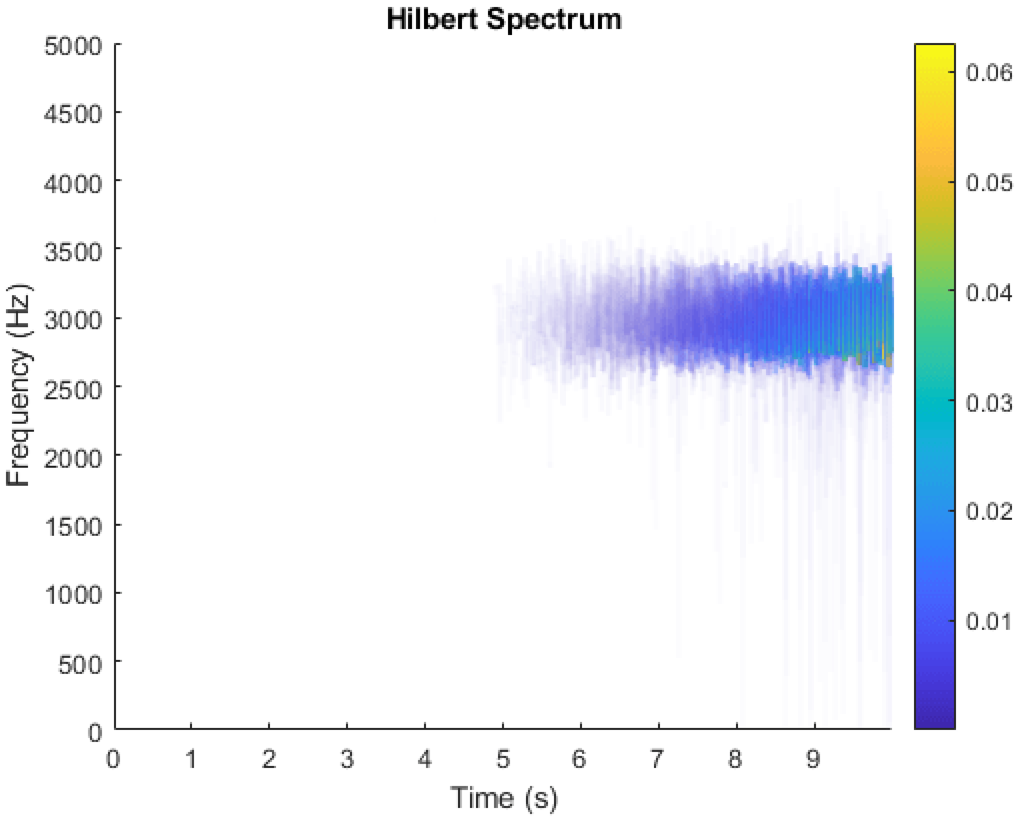
```
Current IMF  |  #Sift Iter  |  Relative Tol  |  Stop Criterion Hit
      1      |        2     |     0.041274   |  SiftMaxRelativeTolerance
      2      |        3     |      0.16695   |  SiftMaxRelativeTolerance
      3      |        3     |      0.18428   |  SiftMaxRelativeTolerance
      4      |        1     |     0.037177   |  SiftMaxRelativeTolerance
      5      |        2     |     0.095861   |  SiftMaxRelativeTolerance
Decomposition stopped because maximum number of intrinsic mode functions was extracted.
```

```
emd(yBad,'MaxNumIMF',5)
```



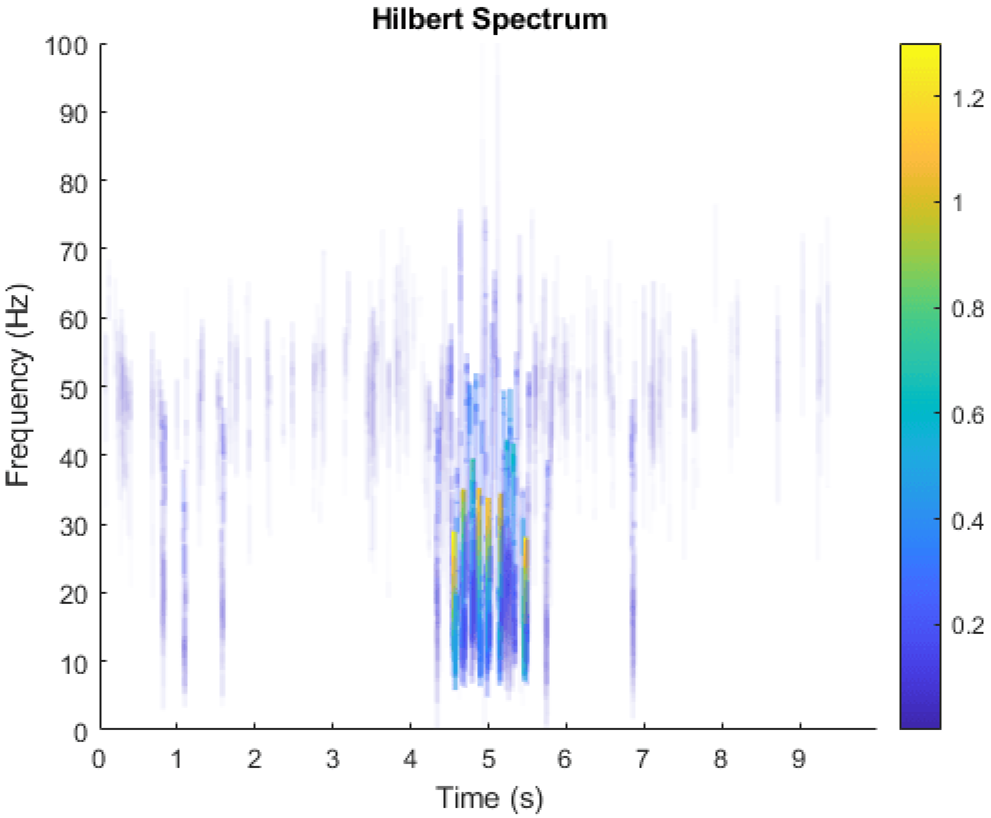
Plot the Hilbert spectrum of the first empirical mode of the defective bearing signal. The first mode captures the effect of high-frequency impacts. The energy of the impacts increases as the bearing wear progresses.

```
figure
hht(imfBad(:,1),fs)
```



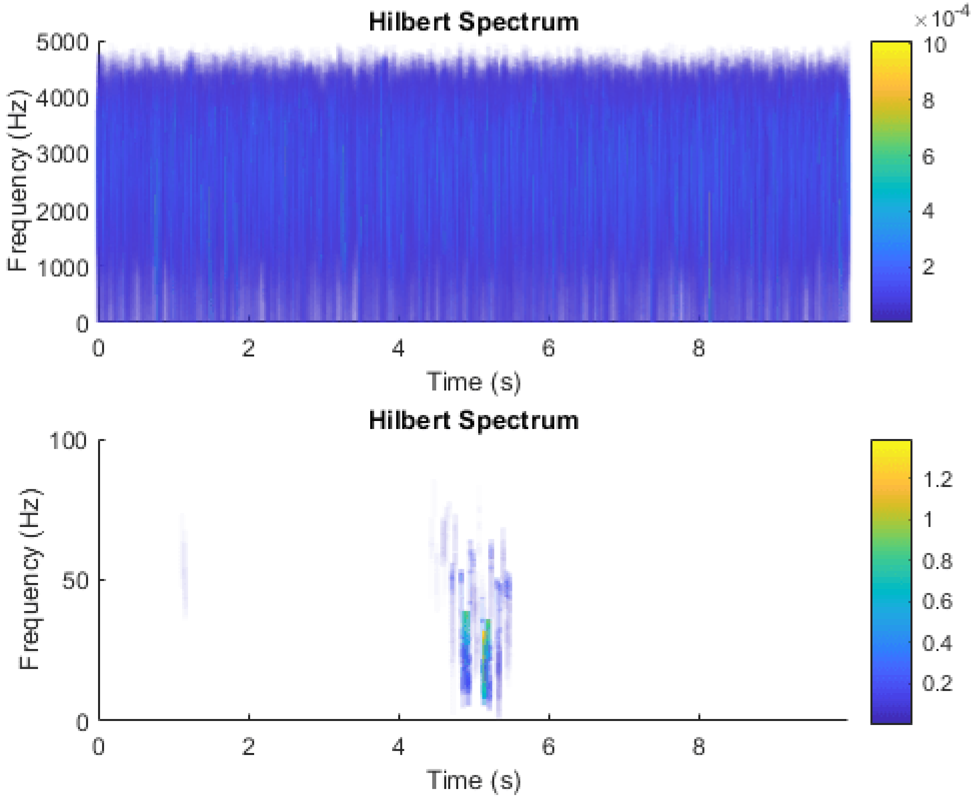
The Hilbert spectrum of the third mode shows the resonance in the vibration signal. Restrict the frequency range from 0 Hz to 100 Hz.

```
hht(imfBad(:,3),fs,'FrequencyLimits',[0 100])
```



For comparison, plot the Hilbert spectra of the first and third modes of the healthy bearing signal.

```
subplot(2,1,1)
hht(imfGood(:,1),fs)
subplot(2,1,2)
hht(imfGood(:,3),fs,'FrequencyLimits',[0 100])
```



Input Arguments

`imf` — Intrinsic mode function

matrix | timetable

Intrinsic mode function, specified as a matrix or timetable. `imf` is any signal whose envelope is symmetric with respect to zero and whose numbers of extrema and zero crossings differ by at most one. `emd` is used to decompose and simplify complicated signals into a finite number of intrinsic mode functions required to perform Hilbert spectral analysis.

`hht` treats each column in `imf` as an intrinsic mode function. For more information on computing `imf`, see `emd`.

`fs` — Sample Rate

2π (default) | positive scalar

Sample rate, specified as a positive scalar. If `fs` is not supplied, a normalized frequency of 2π is used to compute the Hilbert spectrum. If `imf` is specified as a timetable, the sample rate is inferred from it.

`freqlocation` — Location of frequency axis on plot

`'yaxis'` (default) | `'xaxis'`

Location of frequency axis on the plot, specified as `'yaxis'` or `'xaxis'`. To display frequency data on the *y*-axis or *x*-axis of the plot, specify `freqlocation` as `'yaxis'` or `'xaxis'` respectively.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FrequencyResolution',1`

`FrequencyLimits` — Frequency limits to compute Hilbert spectrum

`[0,fs/2]` (default) | 1-by-2 integer-valued vector

Frequency limits to compute Hilbert spectrum, specified as the comma-separated pair consisting of '`FrequencyLimits`' and a 1-by-2 integer-valued vector. `FrequencyLimits` is specified in Hz.

`FrequencyResolution` — Frequency resolution to discretize frequency range

(f_high-f_low)/100 (default) | positive scalar

Frequency resolution to discretize frequency limits, specified as the comma-separated pair consisting of '`FrequencyResolution`' and a positive scalar.

Specify `FrequencyResolution` in Hz. If `'FrequencyResolution'` is not specified, a value of (f_{high}-f_{low})/100 is inferred from `FrequencyLimits`. Here, f_{high} is the upper limit of `FrequencyLimits` and f_{low} is the lower limit.

`MinThreshold` — Minimum threshold value of Hilbert spectrum

`-inf` (default) | scalar

Minimum threshold value of Hilbert spectrum, specified as the comma-separated pair consisting of '`MinThreshold`' and a scalar.

`MinThreshold` sets elements of `hs` to 0 when the corresponding elements of $10\log_{10}(hs)$ are less than `MinThreshold`.

Output Arguments

`hs` — Hilbert spectrum of signal

sparse matrix

Hilbert spectrum of the signal, returned as a sparse matrix. Use `hs` for time-frequency analysis and to identify localized features in the signal.

`f` — Frequency values

vector

Frequency values of the signal, returned as a vector. `hht` uses the frequency vector `f` and the time vector `t` to create the Hilbert spectrum plot.

Mathematically, `f` is denoted as: $f = f_{low} : f_{res} : f_{high}$, where f_{res} is the frequency resolution.

`t` — Time values

vector | `duration` array

Time values of the signal, returned as a vector or a `duration` array. `hht` uses the time vector `t` and the frequency vector `f` to create the Hilbert spectrum plot.

`t` is returned as:

- An array, if `imf` is specified as an array.
- A `duration` array, if `imf` is specified as a uniformly sampled timetable.

`imfinsf` — Instantaneous frequency of each IMF
vector | matrix | timetable

Instantaneous frequency of each IMF, returned as a vector, a matrix, or a timetable.

`imfinsf` has the same number of columns as `imf` and is returned as:

- A vector, if `imf` is specified as a vector.
- A matrix, if `imf` is specified as a matrix.
- A timetable, if `imf` is specified as a uniformly sampled timetable.

`imfinse` — Instantaneous energy of each IMF
vector | matrix | timetable

Instantaneous energy of each IMF, returned as a vector, a matrix, or a timetable.

`imfinse` has the same number of columns as `imf` and is returned as:

- A vector, if `imf` is specified as a vector.
- A matrix, if `imf` is specified as a matrix.
- A timetable, if `imf` is specified as a uniformly sampled timetable.

Algorithms

The Hilbert-Huang transform is useful for performing time-frequency analysis of nonstationary and nonlinear data. The Hilbert-Huang procedure consists of the following steps:

1. `emd` or `vmd` decomposes the data set x into a finite number of intrinsic mode functions.
2. For each intrinsic mode function, x_i, the function `hht`:
   - **a** Uses `hilbert` to compute the analytic signal, $z_i(t) = x_i(t) + jH\{x_i(t)\}$, where $H\{x_i\}$ is the Hilbert transform of x_i.
   - **b** Expresses z_i as $z_i(t) = a_i(t)\, e^{j\theta_i(t)}$, where $a_i(t)$ is the instantaneous amplitude and $\theta_i(t)$ is the instantaneous phase.
   - **c** Computes the instantaneous energy, $|a_i(t)|^2$, and the instantaneous frequency, $\omega_i(t) \equiv d\theta_i(t)/dt$. If given a sample rate, `hht` converts $\omega_i(t)$ to a frequency in Hz.
   - **d** Outputs the instantaneous energy in `imfinse` and the instantaneous frequency in `imfinsf`.
3. When called with no output arguments, `hht` plots the energy of the signal as a function of time and frequency, with color proportional to amplitude.

References

[1] Huang, Norden E, and Samuel S P Shen. *Hilbert–Huang Transform and Its Applications*. 2nd ed. Vol. 16. Interdisciplinary Mathematical Sciences. WORLD SCIENTIFIC, 2014. https://doi.org/10.1142/8804.

[2] Huang, Norden E., Zhaohua Wu, Steven R. Long, Kenneth C. Arnold, Xianyao Chen, and Karin Blank. “ON INSTANTANEOUS FREQUENCY.” *Advances in Adaptive Data Analysis* 01, no. 02 (April 2009): 177–229. https://doi.org/10.1142/S1793536909000096.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Arguments specified using name-value pairs must be compile-time constants.
- Timetables are not supported for code generation.

See Also

`emd` | `vmd`

Topics

“Time-Frequency Gallery”

Introduced in R2018a

horzcat

Horizontal concatenation of Laurent polynomials

Syntax

```
H = horzcat(P1,…,PN)
```

Description

`H = horzcat(P1,…,PN)` returns the horizontal concatenation of the Laurent polynomials `P1,…,PN`.

Examples

Laurent Polynomial Concatenation

Create two Laurent polynomials:

- $a(z) = z - 1$
- $b(z) = -2z^3 + 6z^2 - 7z + 2$

```
a = laurentPolynomial(Coefficients=[1 -1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[-2 6 -7 2],MaxOrder=3);
```

Obtain the vertical and horizontal concatenations of $a(z)$ and $b(z)$.

```
v = vertcat(a,b)
```

```
v=2×1 cell array
    {1x1 laurentPolynomial}
    {1x1 laurentPolynomial}
```

```
h = horzcat(a,b)
```

```
h=1×2 cell array
    {1x1 laurentPolynomial}    {1x1 laurentPolynomial}
```

Input Arguments

P1,…,PN — Input polynomials
`laurentPolynomial` objects

Input polynomials, specified as `laurentPolynomial` objects.

Example: `horzcat(P1,P2,P3)` returns the horizontal concatenation of the three Laurent polynomials `P1`, `P2`, and `P3`.

Output Arguments

`H` — Horizontal cell array

cell array

Horizontal cell array of Laurent polynomials. `H` is a 1-by-*N* cell array, where *N* is the number of Laurent polynomials.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`vertcat`

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

horzcat

Horizontal concatenation of two sensing dictionaries

Syntax

```
Anew = horzcat(A,B)
Anew = [A,B]
```

Description

`Anew = horzcat(A,B)` creates a custom sensing dictionary by appending the columns in `B` after the columns in `A`. The dictionaries `A` and `B` must have the same number of rows.

`Anew = [A,B]` is equivalent to `Anew = horzcat(A,B)`.

Examples

Concatenate Two Sensing Dictionaries

Create two sensing dictionaries of size 100-by-100. Set the basis type of one dictionary to `'dct'` and to `'walsh'` for the other dictionary.

```
A = sensingDictionary(Type={'dct'});
B = sensingDictionary(Type={'walsh'});
```

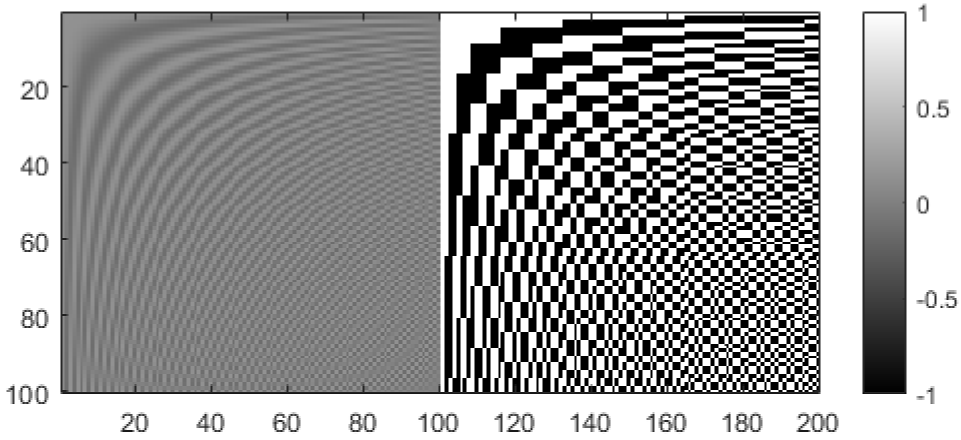
Concatenate the two sensing dictionaries.

```
C = [A B]
```

```
C = 
  sensingDictionary with properties:

                Type: {'dct'  'walsh'}
                Name: {''  ''}
               Level: [0 0]
    CustomDictionary: []
                Size: [100 200]
```

Extract the entire matrix from sensing dictionary `C`. Visualize the matrix.

```
Cmat = subdict(C,1:C.Size(1),1:C.Size(2));
imagesc(Cmat)
axis equal
axis tight
colorbar
colormap gray
```



Input Arguments

A — Sensing dictionary

sensingDictionary object

Sensing dictionary, specified as a sensingDictionary object.

B — Sensing dictionary

sensingDictionary object | matrix

Sensing dictionary, specified as a sensingDictionary object.

Data Types: single | double
Complex Number Support: Yes

Output Arguments

Anew — Sensing dictionary

sensingDictionary object

Sensing dictionary, returned as a sensingDictionary object. Depending on A and B, Anew has the following properties:

- sensingDictionary A, matrix B:

  - `Anew.Type = {A.type,'custom'}`
  - `Anew.CustomDictionary = [A.CustomDictionary B]`
- `sensingDictionary` A, `sensingDictionary` B:
  - `Anew.Type = {A.Type,B.Type}`
  - `Anew.CustomDictionary = [A.CustomDictionary B.CustomDictionary]`

See Also

`sensingDictionary`

Introduced in R2022a

icqt

Inverse constant-Q transform using nonstationary Gabor frames

Syntax

```
xrec = icqt(cfs,g,fshifts)
xrec = icqt( ___ ,'SignalType',sigtype)
[xrec,gdual] = icqt( ___ )
```

Description

`xrec = icqt(cfs,g,fshifts)` returns the inverse constant-Q transform, `xrec`, of the coefficients `cfs`. `cfs` is a matrix, cell array, or structure array. `g` is the cell array of nonstationary Gabor constant-Q analysis filters used to obtain the coefficients `cfs`. `fshifts` is a vector of frequency bin shifts for the constant-Q bandpass filters in `g`. `icqt` assumes by default that the original signal was real-valued. To indicate the original input signal was complex-valued, use the `'SignalType'` name-value pair. If the input to `cqt` was a single signal, then `xrec` is a vector. If the input to `cqt` was a multichannel signal, then `xrec` is a matrix. `cfs`, `g`, and `fshifts` must be outputs of `cqt`.

`xrec = icqt( ___ ,'SignalType',sigtype)` designates whether the signal was real-valued or complex-valued. Valid options for `sigtype` are `'real'` or `'complex'`. If unspecified, `sigtype` defaults to `'real'`.

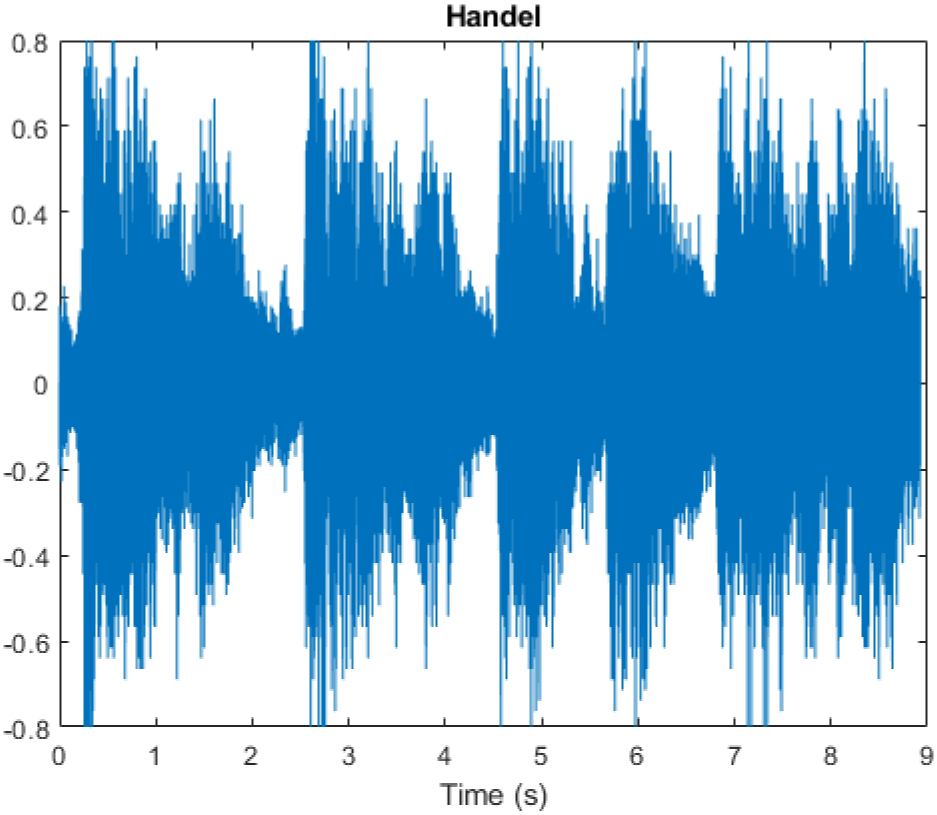
`[xrec,gdual] = icqt( ___ )` returns the dual frames of `xrec` as a cell array the same size as `g`. The dual frames are the canonical dual frames derived from the analysis filters.

Examples

Perfect Reconstruction of Constant-Q Transform

Load and plot the Handel signal.

```
load handel
t = (0:length(y)-1)/Fs;
plot(t,y)
title('Handel')
xlabel('Time (s)')
```



Obtain the constant-Q transform of the signal using the `sparse` transform option. Because the transform will be inverted, you must also return the Gabor frames and frequency shifts used in the analysis.

```
[cfs,~,g,fshifts] = cqt(y,'SamplingFrequency',Fs,'TransformType','sparse');
```

Invert the constant-Q transform and demonstrate perfect reconstruction by showing the maximum absolute reconstruction error and the relative energy error in dB.

```
xrec = icqt(cfs,g,fshifts);
maxAbsError = max(abs(xrec-y))
```

```
maxAbsError = 7.6328e-16
```

```
relEnergyError = 20*log10(norm(xrec-y)/norm(y))
```

```
relEnergyError = -301.4461
```

Input Arguments

`cfs` — Constant-Q coefficients

matrix | cell array | structure array

Constant-Q coefficients of a signal or multichannel signal, specified as a matrix, cell array, or structure array. `cfs` must be the output of `cqt`.

`g` — Nonstationary Gabor constant-Q analysis filters

cell array

Nonstationary Gabor constant-Q analysis filters used to obtain the coefficients `cfs`, specified as a cell array. `cfs` must be the output of `cqt`.

`fshifts` — Frequency bin shifts

real-valued vector

Frequency bin shifts for the constant-Q bandpass filters in `g`, specified as a real-valued vector. `fshifts` must be the output of `cqt`.

`sigtype` — Signal type

`'real'` (default) | `'complex'`

Signal type of the original signal, specified as `'real'` or `'complex'`. Use `sigtype` to designate whether the original signal was real-valued or complex-valued. If unspecified, `sigtype` defaults to `'real'`.

Output Arguments

`xrec` — Inverse constant-Q transform

vector | matrix

Inverse constant-Q transform, returned as a vector or matrix. If the input to `cqt` was a single signal, then `xrec` is a vector. If the input to `cqt` was a multichannel signal, then `xrec` is a matrix.

`gdual` — Dual frames

cell array

Dual frames used in the synthesis of `xrec`, returned as a cell array the same size as `g`. The dual frames are the canonical dual frames derived from the analysis filters.

Algorithms

The theory of nonstationary Gabor (NSG) frames for frequency-adaptive analysis and efficient algorithms for analysis and synthesis using NSG frames are due to Dörfler, Holighaus, Grill, and Velasco [1],[2]. The algorithms used in `cqt` and `icqt` were developed by Dörfler, Holighaus, Grill, and Velasco and are described in [1],[2]. In [3], Schörkhuber, Klapuri, Holighaus, and Dörfler develop and provide algorithms for a phase-corrected CQT transform which matches the CQT coefficients that would be obtained by naïve convolution. The Large Time-Frequency Analysis Toolbox (https://github.com/ltfat) provides an extensive suite of algorithms for nonstationary Gabor frames [4].

References

[1] Holighaus, N., M. Dörfler, G. A. Velasco, and T. Grill. "A framework for invertible real-time constant-Q transforms." *IEEE Transactions on Audio, Speech, and Language Processing.* Vol. 21, No. 4, 2013, pp. 775–785.

[2] Velasco, G. A., N. Holighaus, M. Dörfler, and T. Grill. "Constructing an invertible constant-Q transform with nonstationary Gabor frames." In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11).* Paris, France: 2011.

[3] Schörkhuber, C., A. Klapuri, N. Holighaus, and M. Dörfler. "A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution." Submitted to the *AES 53rd International Conference on Semantic Audio*. London, UK: 2014.

[4] Průša, Z., P. L. Søndergaard, N. Holighaus, C. Wiesmeyr, and P. Balazs. *The Large Time-Frequency Analysis Toolbox 2.0*. Sound, Music, and Motion, Lecture Notes in Computer Science 2014, pp 419-442.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- To minimize compilation time, define the nonstationary Gabor constant-Q analysis filters `g` as variable size using `coder.typeof`. If you define the filters `g` as fixed size, compilation time is significant with minimal gain in execution efficiency.

See Also

`cqt`

Topics

"Nonstationary Gabor Frames and the Constant-Q Transform"
"Time-Frequency Gallery"

Introduced in R2018a

icwt

Inverse continuous 1-D wavelet transform

Syntax

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

Description

`xrec = icwt(cfs)` inverts the continuous wavelet transform (CWT) coefficient matrix `cfs` using Morlet's single integral formula. `icwt` assumes that you obtained the CWT using `cwt` with the default analytic Morse (3,60) wavelet. This wavelet has a symmetry of 3 and a time bandwidth of 60. `icwt` also assumes that the CWT uses default scales.

`xrec = icwt(cfs,wname)` uses the analytic wavelet `wname` to invert the CWT. The specified wavelet must be the same wavelet used in `cwt`.

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

`xrec = icwt( ___ ,period,periodrange)` inverts the CWT over the range of periods specified in `periodrange`. `p` 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)` specifies one or more additional name-value arguments. For example, `xrec = icwt(cfs,TimeBandwidth=40,VoicesPerOctave=20)` specifies a time-bandwidth product of 40 and 20 voices per octave.

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
cfs = cwt(mtlb);
xrec = icwt(cfs);
```

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 the CWT.

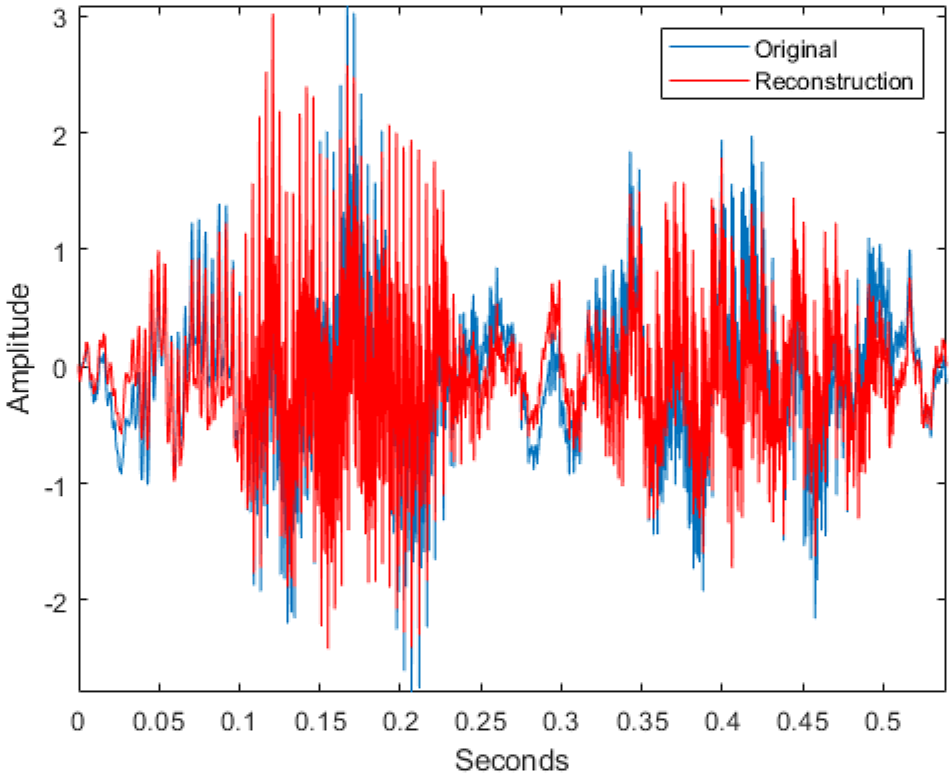
```
bumpmtlb = cwt(mtlb,Fs,"bump");
```

Obtain the inverse CWT. Add the signal mean to the output.

```
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")
hold off
axis tight
legend("Original","Reconstruction")
```



If your computer has a sound card, you can listen to the original and reconstructed signals.

```
% To play the original signal, uncomment the next two lines
% p = audioplayer(mtlb,Fs);
% play(p)
% To play the reconstructed signal, uncomment the next two lines
% 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 sampling frequency is 1 Hz. The extracted information corresponds to frequencies in the range [0.030 0.070] Hz.

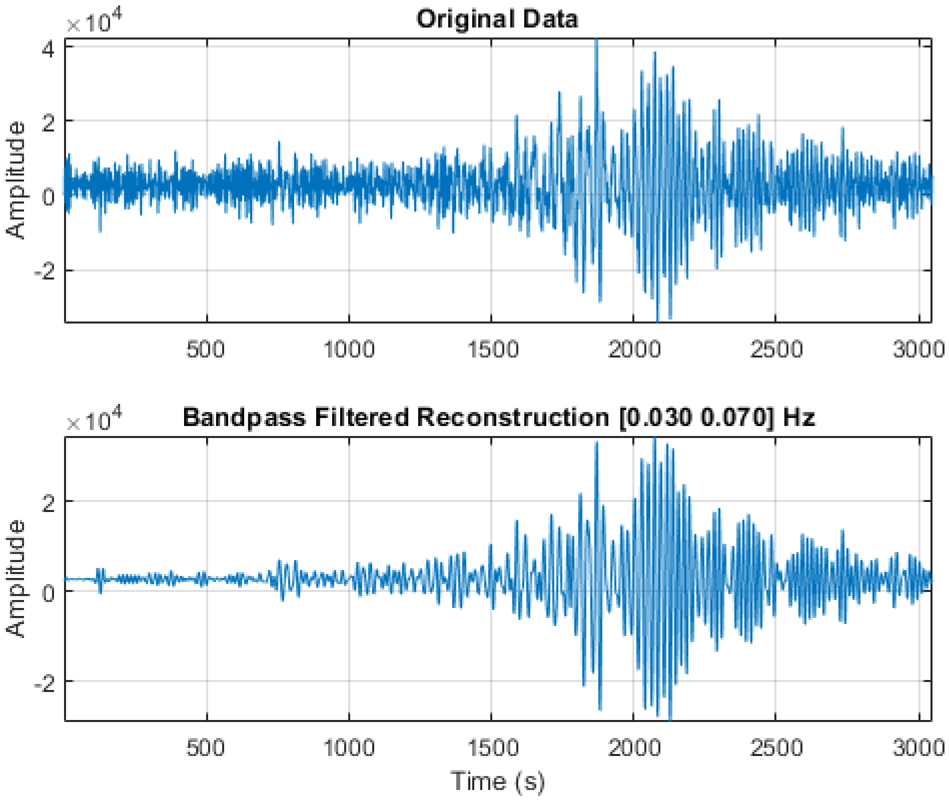
```
load kobe
```

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.

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

Plot the original and reconstructed data.

```
subplot(2,1,1)
plot(kobe)
grid on
title("Original Data")
ylabel("Amplitude")
axis tight

subplot(2,1,2)
plot(xrec)
grid on
title("Bandpass Filtered Reconstruction [0.030 0.070] Hz");
xlabel("Time (s)")
ylabel("Amplitude")
axis tight
```



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.

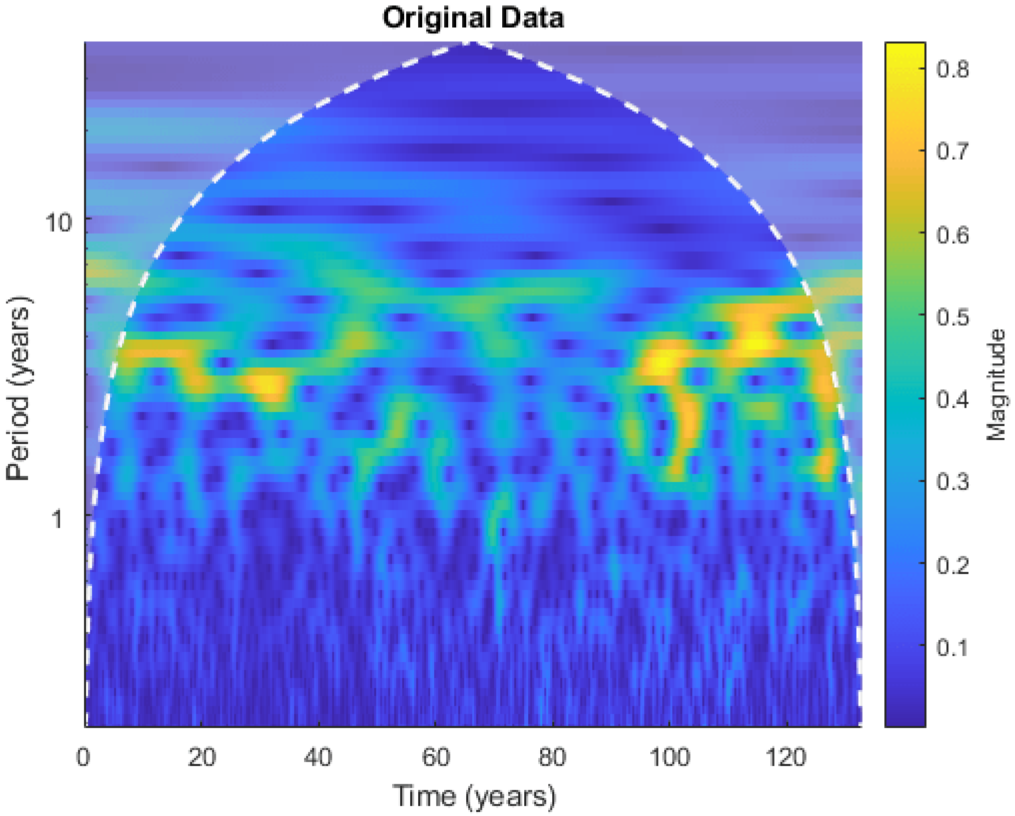
```
load ninoairdata
[cfs,period] = cwt(nino,years(1/12));
```

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

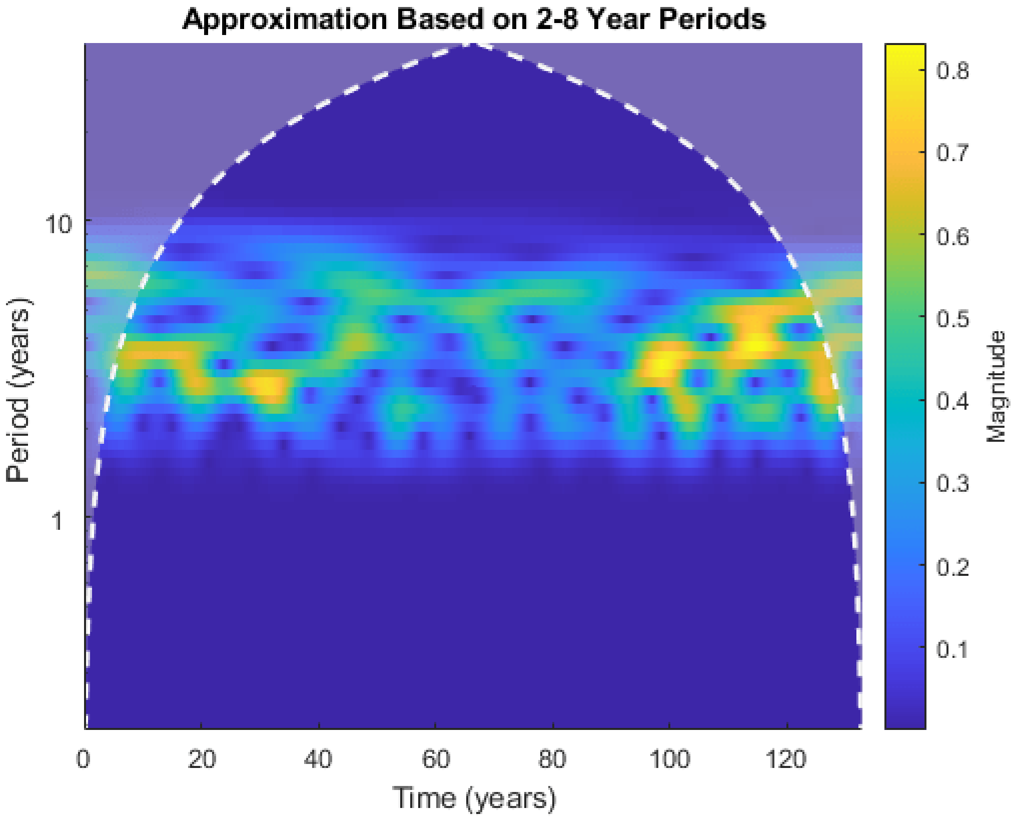
```
xrec = icwt(cfs,[],period,[years(2) years(8)]);
```

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")
```

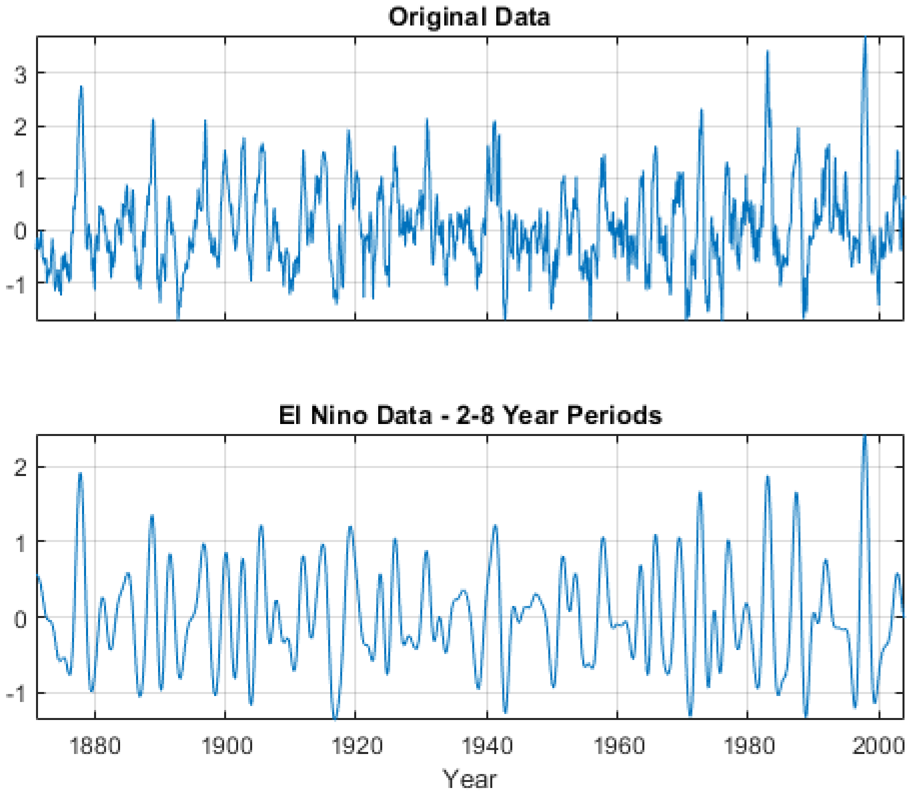


```
figure
cwt(xrec,years(1/12))
title("Approximation Based on 2-8 Year Periods")
```



Compare the original data with the reconstructed data in time.

```
figure
subplot(2,1,1)
plot(datayear,nino)
grid on
ax = gca;
ax.XTickLabel = '';
axis tight
title("Original Data")

subplot(2,1,2)
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.

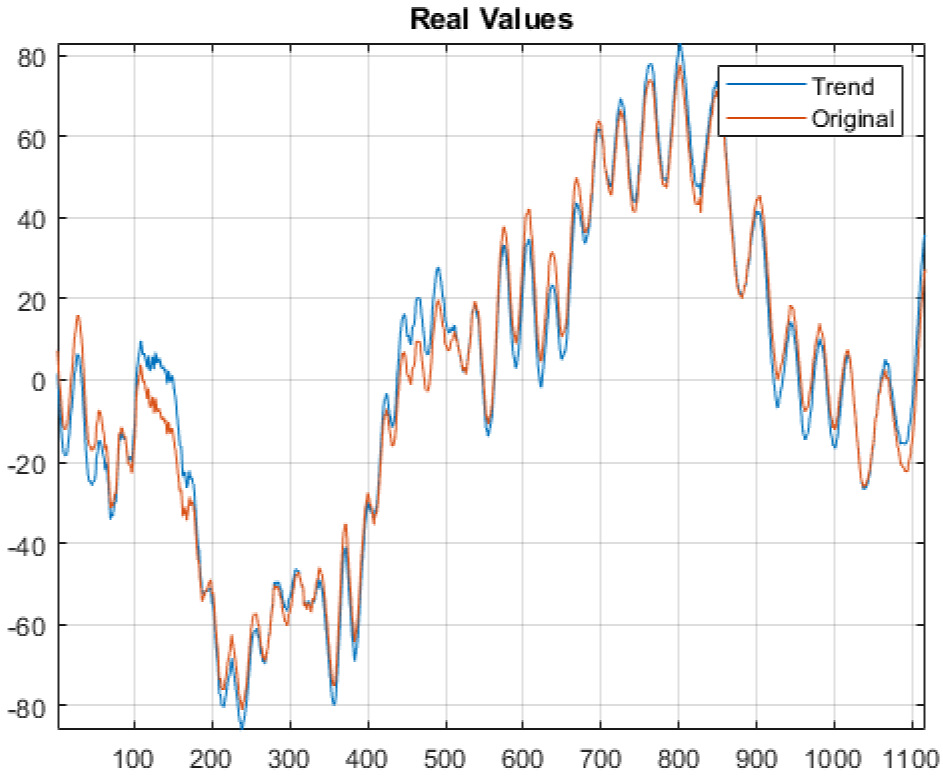
```
load npg2006.mat
cfs = cwt(npg2006.cx);
```

Create a time-varying trend derived from the data.

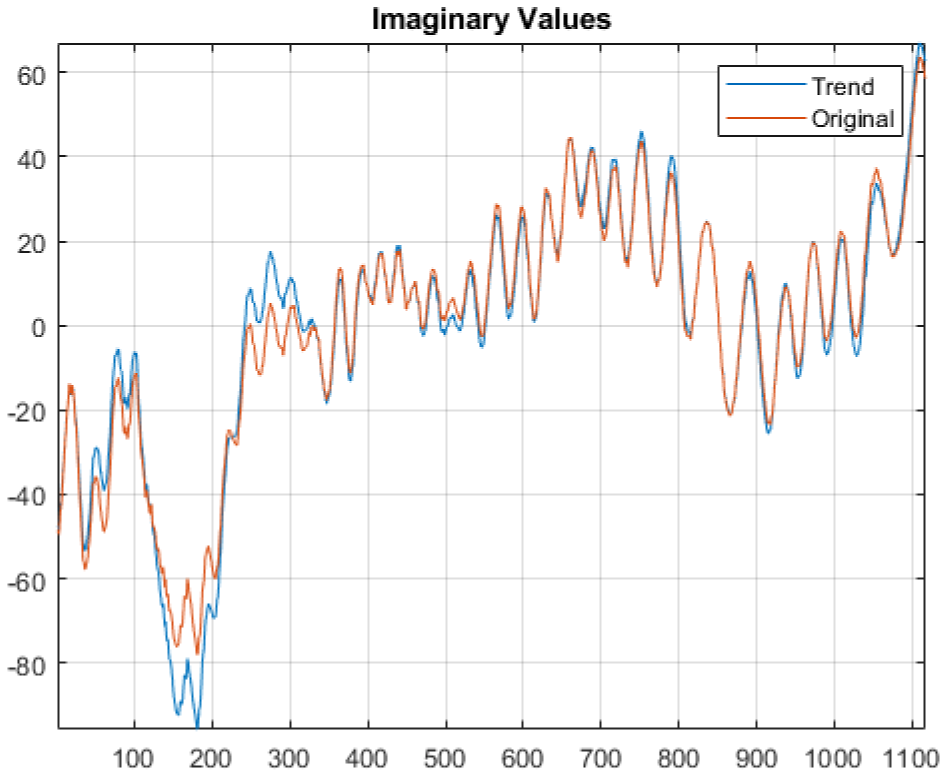
```
trend = smoothdata(npg2006.cx,"movmean",100);
```

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

```
xrec = icwt(cfs,SignalMean=trend);
plot([real(xrec)' real(npg2006.cx)])
grid on
title("Real Values")
legend("Trend","Original")
axis tight
```



```
figure
plot([imag(xrec)' imag(npg2006.cx)])
grid on
title("Imaginary Values")
legend("Trend","Original")
axis tight
```



Reconstruct Signal Using Analysis Filter Bank

Load an ECG waveform. Create a CWT filter bank with periodic boundary handling that you can apply to the waveform.

```
load wecg
fb = cwtfilterbank(SignalLength=length(wecg),Boundary="periodic");
```

Obtain the two-sided frequency responses for the wavelet and scaling filters in the filter bank.

```
psif = freqz(fb,FrequencyRange="twosided",IncludeLowpass=true);
```

Use the filter bank to obtain the CWT of the waveform. Also obtain the scaling coefficients for the transform.

```
[cfs,~,~,scalcfs] = wt(fb,wecg);
```

Use the analysis filter bank to reconstruct the input. The approximate synthesis filters, or dual frame, are used to invert the transform.

```
xrecAN = icwt(cfs,[],ScalingCoefficients=scalcfs,...
    AnalysisFilterBank=psif);
```

Reconstruct the input using the default Morlet single integral formula.

```
xrecSI = icwt(cfs,[],ScalingCoefficients=scalcfs);
```

Compare the maximum reconstruction errors.

```
errAN = norm(xrecAN'-wecg,Inf)
```

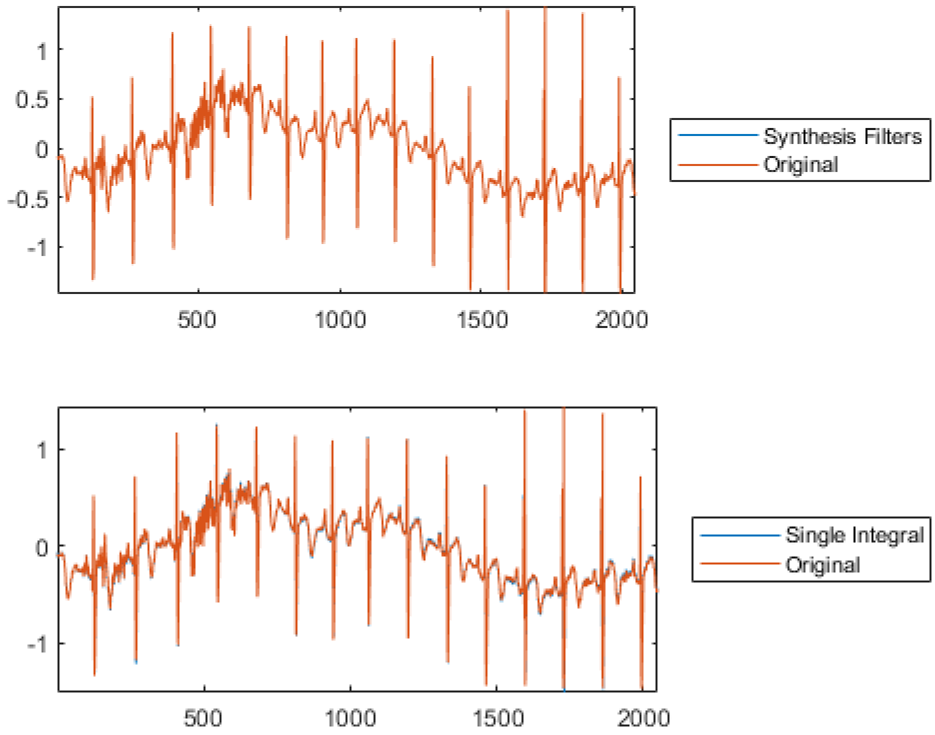
```
errAN = 4.4409e-16
```

```
errSI = norm(xrecSI'-wecg,Inf)
```

```
errSI = 0.4037
```

Plot both reconstructions.

```
subplot(2,1,1)
plot([xrecAN' wecg])
axis tight
    legend("Synthesis Filters","Original",Location="eastoutside")
subplot(2,1,2)
plot([xrecSI' wecg])
axis tight
legend("Single Integral","Original",Location="eastoutside")
```



Input Arguments

cfs — Continuous wavelet transform coefficients

matrix

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

If `cfs` is a 2-D matrix, `icwt` assumes that the CWT was obtained from a real-valued signal. If `cfs` 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 `cfs` is the CWT of the positive (counterclockwise) component and the second page of `cfs` is the negative (clockwise) component. The pages represent the analytic and anti-analytic parts of the CWT, respectively.

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

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

Analytic wavelet used to invert the CWT, specified as one of these:

- `"morse"` — Morse wavelet
- `"amor"` — Morlet wavelet
- `"bump"` — bump wavelet

The specified wavelet must be the same wavelet used to obtain the CWT. The default Morse wavelet uses a symmetry parameter, γ, that is 3 and a 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, `cfs`. If you specify `f`, you must also specify `freqrange`.

Data Types: `single` | `double`

`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 `cfs` is a 2-D matrix, `freqrange` must be a two-element vector.
- If `cfs` is a 3-D matrix, `freqrange` can be a two-element vector or a 2-by-2 matrix.
  - If `freqrange` is a vector, `icwt` inverts the CWT over the same frequency range in both the positive (analytic) and negative (anti-analytic) components of `cfs`.
  - If `freqrange` is a 2-by-2 matrix, the first row contains the frequency range for the positive part of `cfs` (first page) and the second row contains the frequency range for the negative part of `cfs` (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.

Data Types: `single` | `double`

`period` — Time periods
vector

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

Data Types: `duration`

`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 `cfs` is a 2-D matrix, `periodrange` must be a two-element vector of durations.
- If `cfs` 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, `icwt` inverts the CWT over the same frequency range in both the positive (analytic) and negative (anti-analytic) components of `cfs`.
  - If `periodrange` is a 2-by-2 matrix of durations, the first row contains the period range for the positive part of `cfs` (first page) and the second row contains the period range for the negative part of `cfs` (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.

Data Types: `duration`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `xrec = icwt(cfs,"bump",VoicesPerOctave=10)` returns the inverse CWT of `cfs` using the bump wavelet and 10 voices per octave.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `xrec = icwt(cfs,"WaveletParameters",[3 40],"SignalMean",sigmean)` inverts the CWT using the Morse (3,40) wavelet and signal mean `sigmean`.

`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 scalar greater than 3 and less than or equal to 120. The specified time bandwidth must be the same time-bandwidth value used in the `cwt`. The symmetry of the Morse wavelet is assumed to be 3.

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

This syntax is not valid if you specify the `AnalysisFilterBank` name-value argument.

Data Types: `single` | `double`

`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 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 values used in the CWT.

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

This syntax is not valid if you specify the `AnalysisFilterBank` name-value argument.

Data Types: `single` | `double`

`SignalMean` — Signal mean
scalar | vector

Signal mean to add to the `icwt` output, specified as a scalar or vector. If the signal mean is a vector, it must be the same length as the column size of the wavelet coefficient matrix `cfs`.

- If `cfs` is a 2-D matrix, the signal mean must be a real-valued scalar or vector.
- If `cfs` is a 3-D matrix, the signal mean must be a complex-valued scalar or vector.

Because `cwt` does not preserve the signal mean, the inverse CWT is a zero-mean signal by default. Adding a non-zero signal mean to a frequency- or period-limited reconstruction adds a zero-frequency component to the reconstruction.

This syntax is not valid if you specify the `AnalysisFilterBank` name-value argument.

Data Types: `single` | `double`

`ScalingCoefficients` — Scaling coefficients
real- or complex-valued vector

Scaling coefficients to use in the inverse CWT, specified as a real- or complex-valued vector, obtained as an optional output of `cwt`. The length of `ScalingCoefficients` is equal to the column size of `cfs`.

- If you only specify `ScalingCoefficients` without the `AnalysisFilterBank` name-value argument, the single-integral approximation is used to obtain the inverse CWT.
- If you specify `ScalingCoefficients` with the `AnalysisFilterBank` name-value argument, the synthesis filters are used to obtain the inverse CWT.

You cannot specify both `SignalMean` and `ScalingCoefficients` name-value arguments.

Data Types: `single` | `double`

`AnalysisFilterBank` — Analysis filters
real-valued matrix

Bank of analysis filters used in inverting the CWT, specified as a matrix. The approximate synthesis filters, or dual frame, are used in the inversion. In most cases, use of the approximate synthesis filters results in a more accurate signal reconstruction. The wavelet name input is ignored if you specify the analysis filters.

To use the analysis filters, you must obtain the CWT with `ExtendSignal` set to `false` in `cwt`, or equivalently, `Boundary` set to `"periodic"` in `cwtfilterbank`. Obtain the analysis filters from the `freqz` object function of the filter bank with `FrequencyRange="twosided"` and `IncludeLowpass=true`.

Data Types: `single` | `double`

`VoicesPerOctave` — Number of voices per octave
`10` (default) | integer from 1 to 48

Number of voices per octave used in inverting the CWT, specified as an integer from 1 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 value used to obtain the CWT.

You cannot specify the number of voices per octave if you specify either the frequency, `f`, or duration, `period`. This syntax is not valid if you specify the `AnalysisFilterBank` name-value argument.

Data Types: `single` | `double`

Output Arguments

`xrec` — Inverse 1-D continuous wavelet transform
real- or complex-valued row vector

Inverse 1-D continuous wavelet transform, returned as a real- or complex-valued row vector.

Data Types: `single` | `double`

More About

Inverse Continuous Wavelet Transform — Single Integral Formula

By default, `icwt` computes the inverse CWT based on a discretized version of the single integral formula due to Morlet [5]. For a brief description of the theoretical foundation for the single integral formula, see “Inverse Continuous Wavelet Transform”. For additional theoretical information, see section 2.4 of [6]. The discretized version of this integral is presented in [7].

Compatibility Considerations

`icwt` behavior change

Behavior changed in R2022a

If you invert the CWT over a specified frequency range or range of periods, you must precede those inputs either by a wavelet name or an empty input for the default Morse wavelet.

Functionality	Result	Use Instead	Compatibility Considerations
`xrec = icwt(cfs,f,freqrange)`	Errors	`xrec = icwt(cfs,[],f,freqrange)` or `xrec = icwt(cfs,"morse",f,freqrange)`	You do not have to specify the default Morse wavelet if you are only setting `Name=Value` arguments. For example, `xrec = icwt(cfs,TimeBandwidth=40)`.
`xrec = icwt(cfs,f,freqrange,Name=Value)`	Errors	`xrec = icwt(cfs,[],f,freqrange,Name=Value)` or `xrec = icwt(cfs,"morse",f,freqrange,Name=Value)`	
`xrec = icwt(cfs,period,periodrange)`	Errors	`xrec = icwt(cfs,[],period,periodrange)` or `xrec = icwt(cfs,"morse",period,periodrange)`	
`xrec = icwt(cfs,period,periodrange,Name=Value)`	Errors	`xrec = icwt(cfs,[],period,periodrange,Name=Value)` or `xrec = icwt(cfs,"morse",period,periodrange,Name=Value)`	

Data type of wavelet and scaling coefficients must match for `icwt`

Behavior changed in R2022a

If you specify `ScalingCoefficients`, the scaling coefficients must have the same data type as the wavelet coefficients `cfs`. Both sets of coefficients must be either single or double precision.

Note that the wavelet and scaling coefficient outputs of `cwt` and the `wt` method of `cwtfilterbank` always have the same data type.

References

[1] Lilly, J. M., and S. C. Olhede. "Generalized Morse Wavelets as a Superfamily of Analytic Wavelets." *IEEE Transactions on Signal Processing* 60, no. 11 (November 2012): 6036–41. https://doi.org/10.1109/TSP.2012.2210890.

[2] Lilly, J.M., and S.C. Olhede. "Higher-Order Properties of Analytic Wavelets." *IEEE Transactions on Signal Processing* 57, no. 1 (January 2009): 146–60. https://doi.org/10.1109/TSP.2008.2007607.

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

[4] Lilly, J. M., and J.-C. Gascard. "Wavelet Ridge Diagnosis of Time-Varying Elliptical Signals with Application to an Oceanic Eddy." *Nonlinear Processes in Geophysics* 13, no. 5 (September 14, 2006): 467–83. https://doi.org/10.5194/npg-13-467-2006.

[5] Duval-Destin, M., M. A. Muschietti, and B. Torresani. “Continuous Wavelet Decompositions, Multiresolution, and Contrast Analysis.” *SIAM Journal on Mathematical Analysis* 24, no. 3 (May 1993): 739–55. https://doi.org/10.1137/0524045.

[6] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

[7] Torrence, Christopher, and Gilbert P. Compo. “A Practical Guide to Wavelet Analysis.” *Bulletin of the American Meteorological Society* 79, no. 1 (January 1, 1998): 61–78. https://doi.org/10.1175/1520-0477(1998)079<0061:APGTWA>2.0.CO;2.

[8] Holschneider, M., and Ph. Tchamitchian. “Pointwise Analysis of Riemann’s 'Nondifferentiable' Function.” *Inventiones Mathematicae* 105, no. 1 (December 1991): 157–75. https://doi.org/10.1007/BF01232261.

Extended Capabilities

C/C++ Code Generation

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

See Also

`cwt` | `cwtfilterbank` | `cwtfreqbounds` | `wcoherence` | `wsst`

Topics

“Continuous and Discrete Wavelet Transforms”
“CWT-Based Time-Frequency Analysis”
“Morse Wavelets”
“Time-Frequency Gallery”

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 check box in the bottom left corner of the plot. Enabling the check box superimposes the plot of the input signal `SIG` on the plot of the reconstructed signal. By default the check box 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`

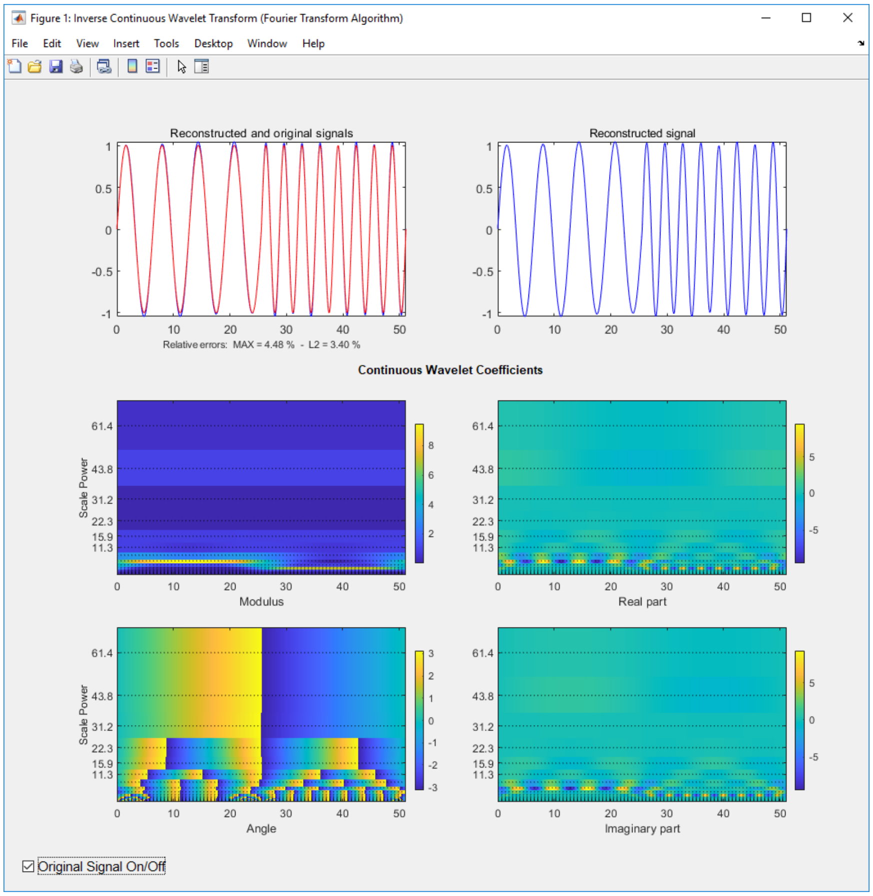
Reconstructed signal

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 check box in the bottom left corner of the plot.

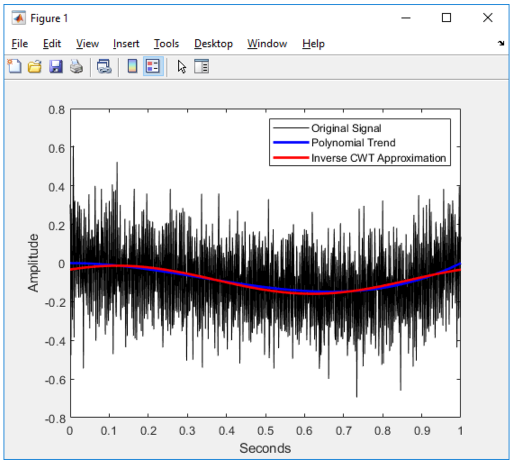


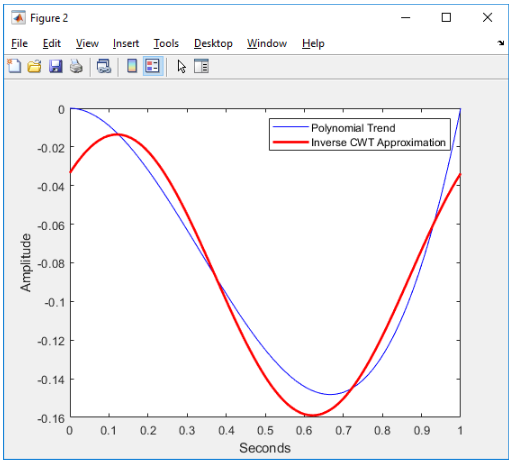
Relative errors: MAX = 4.48 % - L2 = 3.40 %

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',{'bump',[4 0.7]});
% Zero out all but the nine coarsest scale CWT coefficients
cwty.cfs(1:80,:) = 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);
```

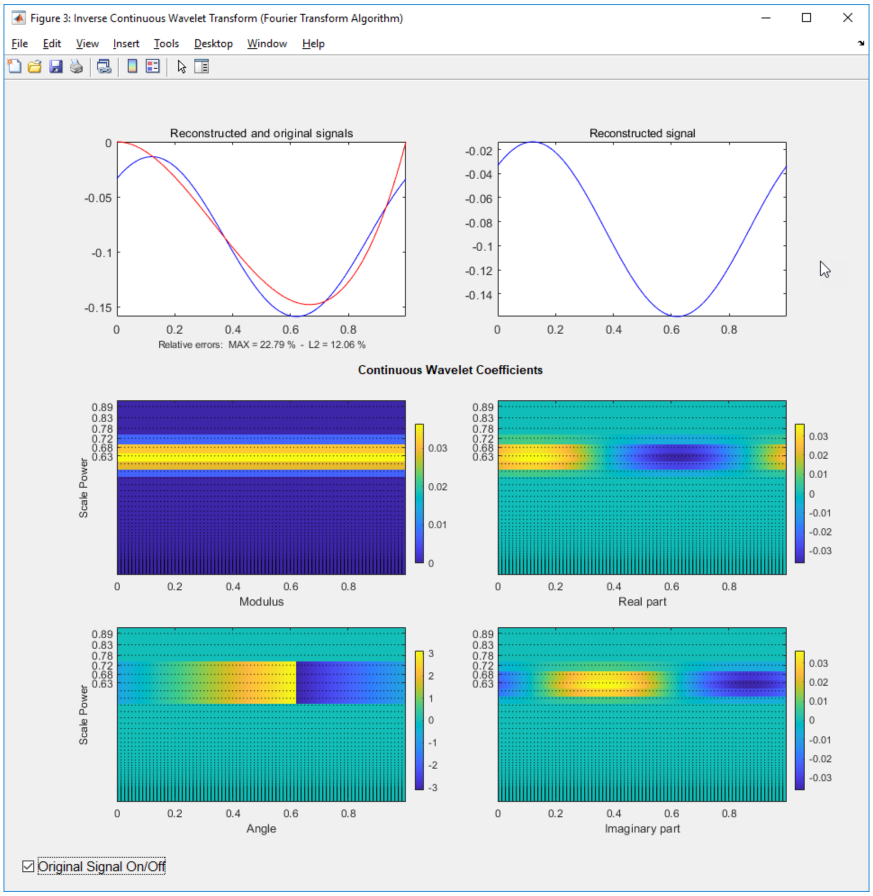
```
legend('Original Signal','Polynomial Trend',...
    'Inverse CWT Approximation');
figure
plot(t,x,'b'); hold on;
xlabel('Seconds'); ylabel('Amplitude');
plot(t,xrec,'r','linewidth',2);
legend('Polynomial Trend','Inverse CWT Approximation');
```





You can also use the following syntax to plot the approximation. Select the box 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');
```



More About

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 1-622 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 check box in the bottom-left corner of the plot. Enabling the check box 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');
```

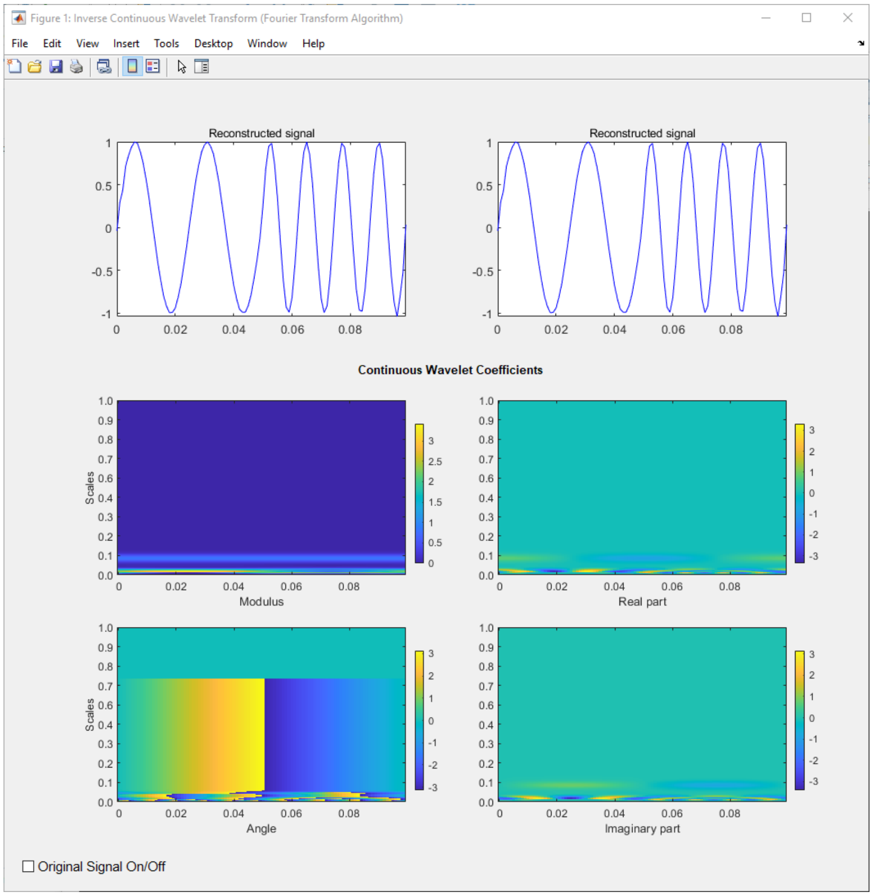


Figure 1: Inverse Continuous Wavelet Transform (Fourier Transform Algorithm)
File Edit View Insert Tools Desktop Window Help
Reconstructed signal
Reconstructed signal
Continuous Wavelet Coefficients
Scales
Modulus
Real part
Scales
Angle
Imaginary part
□ Original Signal On/Off

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);

% 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** check box in the bottom-left corner.

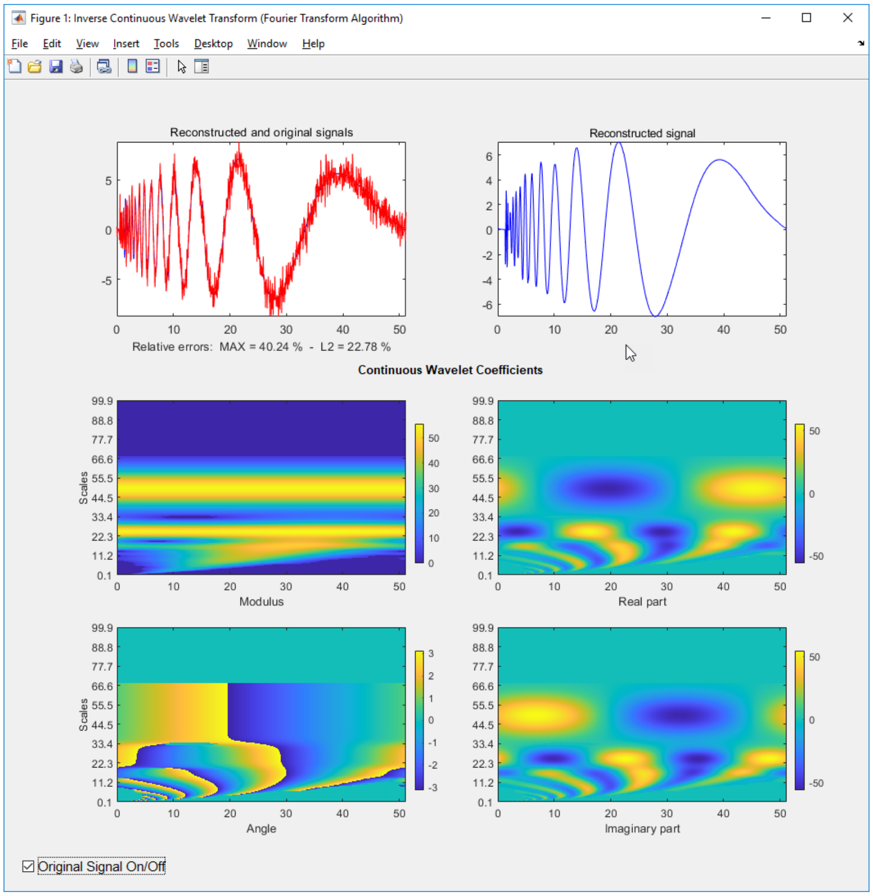


Figure 1: Inverse Continuous Wavelet Transform (Fourier Transform Algorithm)
File Edit View Insert Tools Desktop Window Help
Reconstructed and original signals
5
0
-5
0 10 20 30 40 50
Relative errors: MAX = 40.24 % - L2 = 22.78 %
Reconstructed signal
6
4
2
0
-2
-4
-6
0 10 20 30 40 50
Continuous Wavelet Coefficients
Scales
99.9 88.8 77.7 66.6 55.5 44.5 33.4 22.3 11.2 0.1
50 40 30 20 10 0
Modulus
Real part
50 0 -50
Angle
3 2 1 0 -1 -2 -3
Imaginary part
☑ Original Signal On/Off

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

`icwtft` | `cwtft` | `cwt`

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
```

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
```

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

idualtree

Kingsbury Q-shift 1-D inverse dual-tree complex wavelet transform

Syntax

```
xrec = idualtree(A,D)
xrec = idualtree( ___ ,Name,Value)
```

Description

`xrec = idualtree(A,D)` returns the inverse 1-D complex dual-tree transform of the final-level approximation coefficients, `A`, and cell array of wavelet coefficients, `D`. `A` and `D` are outputs of `dualtree`. For the reconstruction, `idualtree` uses two sets of filters:

- Orthogonal Q-shift filter of length 10
- Near-symmetric biorthogonal filter pair with lengths 7 (scaling synthesis filter) and 5 (wavelet synthesis filter)

`xrec = idualtree( ___ ,Name,Value)` specifies additional options using name-value pair arguments. For example, `'LowpassGain',0.1` applies a gain of 0.1 to the final-level approximation coefficients.

Examples

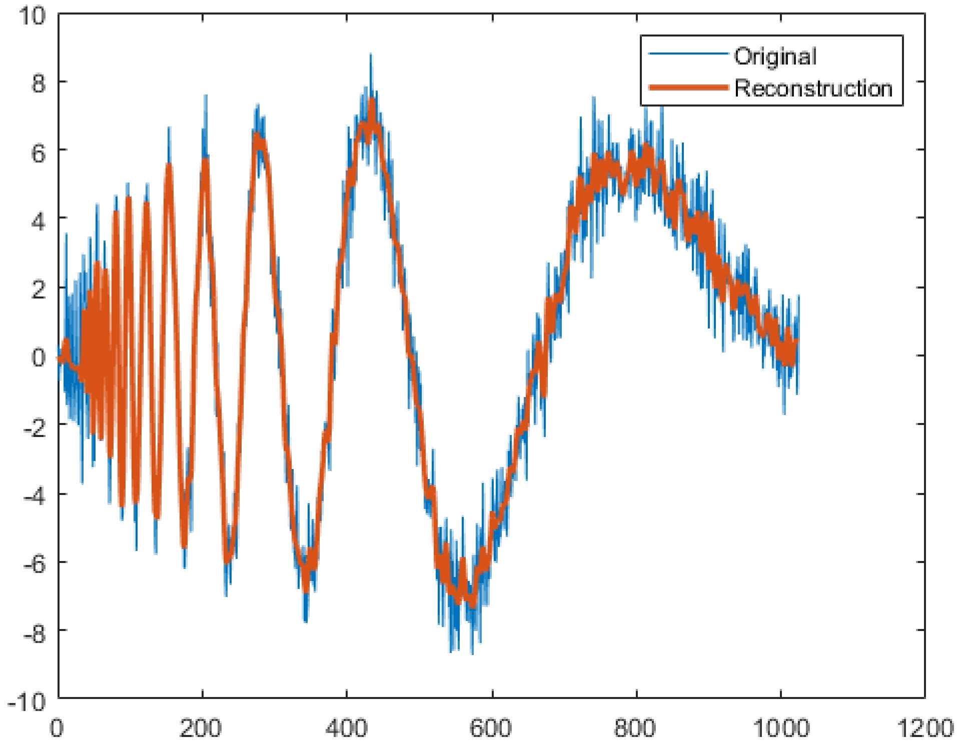
Inverse 1-D Dual-Tree Complex Wavelet Transform

Load a signal, and obtain its dual-tree transform.

```
load noisdopp
[a,d] = dualtree(noisdopp);
```

Reconstruct an approximation using all but the two finest-detail wavelet subbands.

```
dgain = ones(numel(d),1);
dgain(1:2) = 0;
xrec = idualtree(a,d,'DetailGain',dgain);
plot(noisdopp)
hold on
plot(xrec,'LineWidth',2);
legend('Original','Reconstruction')
```



Input Arguments

A — Final-level approximation coefficients

real-valued vector | real-valued matrix

Final-level approximation coefficients, specified as a real-valued vector or real-valued matrix. The approximation coefficients are the output of `dualtree`.

Data Types: `double` | `single`

D — Wavelet coefficients

cell array

Approximation coefficients, specified as a cell array. The wavelet coefficients are the output of `dualtree`.

Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: 'LevelOneFilter','antonini','LowpassGain',0.5

LevelOneFilter — Biorthogonal filter
'nearsym5_7' (default) | 'nearsym13_19' | 'antonini' | 'legall'

Biorthogonal filter to use in the first-level synthesis, specified by one of the values listed here. For perfect reconstruction, the first-level synthesis filters must match the first-level analysis filters used in dualtree.

- 'legall' — LeGall 5/3 filter
- 'nearsym13_19' — (13,19)-tap near-orthogonal filter
- 'nearsym5_7' — (5,7)-tap near-orthogonal filter
- 'antonini' — (9,7)-tap Antonini filter

FilterLength — Orthogonal Hilbert Q-shift synthesis filter pair length
10 (default) | 6 | 14 | 16 | 18

Orthogonal Hilbert Q-shift synthesis filter pair length to use for levels 2 and higher, specified as one of the listed values. For perfect reconstruction, the filter length must match the filter length used in dualtree.

DetailGain — Wavelet coefficients subband gains
real-valued vector

Wavelet coefficients subband gains, specified as a real-valued vector of length L, where L is the number of elements in D. The elements of DetailGain are real numbers in the interval [0, 1]. The k^{th} element of DetailGain is the gain (weighting) applied to the k^{th} wavelet subband. By default, DetailGain is a vector of L ones.

LowpassGain — Gain
1 (default) | real number

Gain to apply to final-level approximation (lowpass, scaling) coefficients, specified as a real number in the interval [0, 1].

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation

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

See Also

`dualtree` | `qbiorthfilt` | `qorthwavf` | `dualtree3` | `dualtree2`

Topics

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

Introduced in R2020a

idualtree2

Kingsbury Q-shift 2-D inverse dual-tree complex wavelet transform

Syntax

```
imrec = idualtree2(A,D)
imrec = idualtree2( ___ ,Name,Value)
```

Description

`imrec = idualtree2(A,D)` returns the inverse 2-D complex dual-tree transform of the final-level approximation coefficients, `A`, and cell array of wavelet coefficients, `D`. `A` and `D` are outputs of `dualtree2`. For the reconstruction, `idualtree2` uses two sets of filters:

- Orthogonal Q-shift filter of length 10
- Near-symmetric biorthogonal filter pair with lengths 7 (scaling synthesis filter) and 5 (wavelet synthesis filter)

`imrec = idualtree2( ___ ,Name,Value)` specifies additional options using name-value pair arguments. For example, `'LowpassGain',0.1` applies a gain of 0.1 to the final-level approximation coefficients.

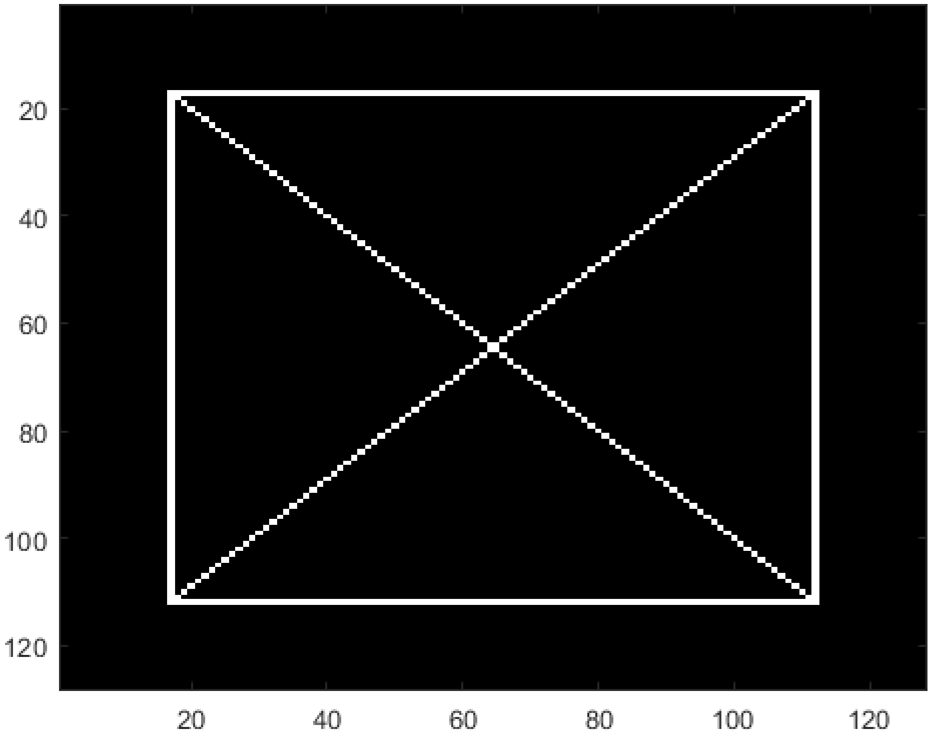
Examples

Inverse 2-D Dual-Tree Wavelet Transform Using Specific Subbands

This example shows how to reconstruct an approximation based on a subset of the wavelet subbands.

Load a 128-by-128 grayscale image.

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



Obtain the dual-tree wavelet transform of the image down to level 2

```
lev = 2;
[a,d] = dualtree2(xbox,'Level',lev);
```

Since there are six wavelet subbands in each level of the decomposition, create a 2-by-6 matrix of zeros.

```
dgains = zeros(lev,6);
```

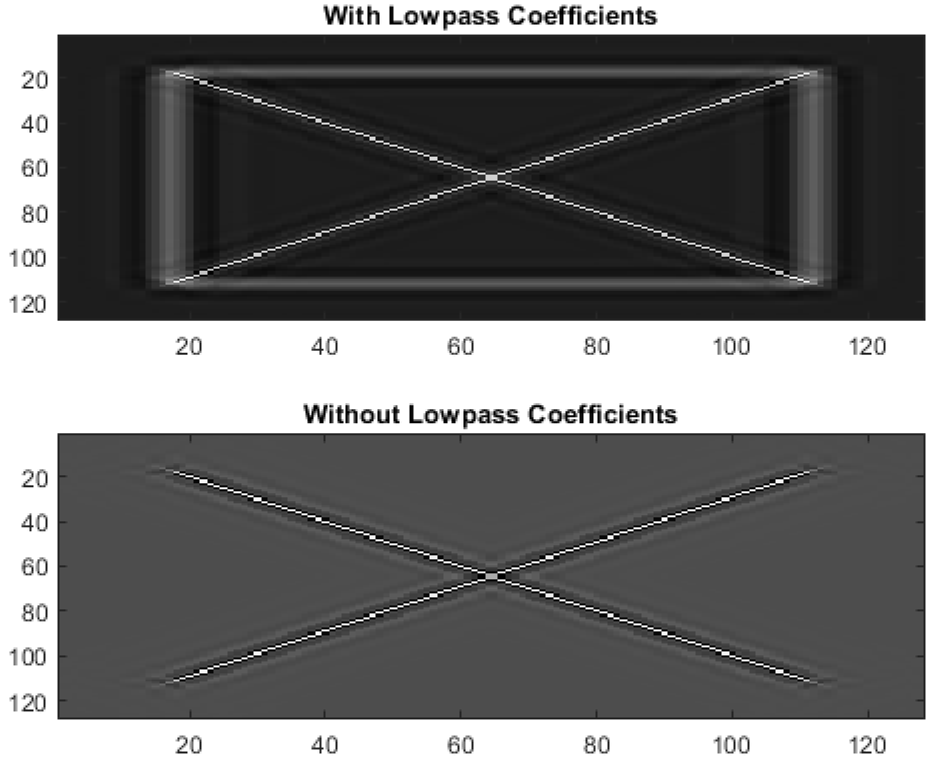
To reconstruct an approximation based on the 2nd and 5th wavelet subbands, set the second and fifth rows of `dgains` equal to 1. The 2nd and 5th wavelet subbands correspond to the highpass filtering of the rows and columns of the image.

```
dgains(:,[2 5]) = 1;
```

Obtain two reconstructions using the specified wavelet subbands. Include the scaling (lowpass) coefficients only in the first reconstruction.

```
imrec = idualtree2(a,d,'DetailGain',dgains);
imrec2 = idualtree2(a,d,'DetailGain',dgains,'LowpassGain',0);
figure
subplot(2,1,1)
imagesc(imrec)
title('With Lowpass Coefficients')
subplot(2,1,2)
imagesc(imrec2)
```

```
title('Without Lowpass Coefficients')
colormap gray
```



Input Arguments

A — Final-level approximation coefficients

real-valued array

Final-level approximation coefficients, specified as a real-valued array. The approximation coefficients are the output of `dualtree2`.

Data Types: `double` | `single`

D — Wavelet coefficients

cell array

Approximation coefficients, specified as a cell array. The wavelet coefficients are the output of `dualtree2`.

Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'LevelOneFilter','antonini','LowpassGain',0.5`

`LevelOneFilter` — Biorthogonal filter

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

Biorthogonal filter to use in the first-level synthesis, specified by one of the values listed here. For perfect reconstruction, the first-level synthesis filters must match the first-level analysis filters used in `dualtree2`.

- `'legall'` — LeGall 5/3 filter
- `'nearsym13_19'` — (13,19)-tap near-orthogonal filter
- `'nearsym5_7'` — (5,7)-tap near-orthogonal filter
- `'antonini'` — (9,7)-tap Antonini filter

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

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

Orthogonal Hilbert Q-shift synthesis filter pair length to use for levels 2 and higher, specified as one of the listed values. For perfect reconstruction, the filter length must match the filter length used in `dualtree2`.

`DetailGain` — Wavelet coefficients subband gains

real-valued matrix

Wavelet coefficients subband gains, specified as a real-valued matrix with a row dimension of *L*, where *L* is the number of elements in `D`. There are six columns in `DetailGain` for each of the six wavelet subbands. The elements of `DetailGain` are real numbers in the interval [0, 1]. The k^{th} column elements of `DetailGain` are the gains (weightings) applied to the k^{th} wavelet subband. By default, `DetailGain` is a *L*-by-6 matrix of ones.

`LowpassGain` — Gain

`1` (default) | real number

Gain to apply to final-level approximation (lowpass, scaling) coefficients, specified as a real number in the interval [0, 1].

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation

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

See Also

`dualtree2` | `qbiorthfilt` | `qorthwavf` | `dualtree3` | `dualtree`

Topics

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

Introduced in R2020a

idualtree3

3-D dual-tree complex 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);
```

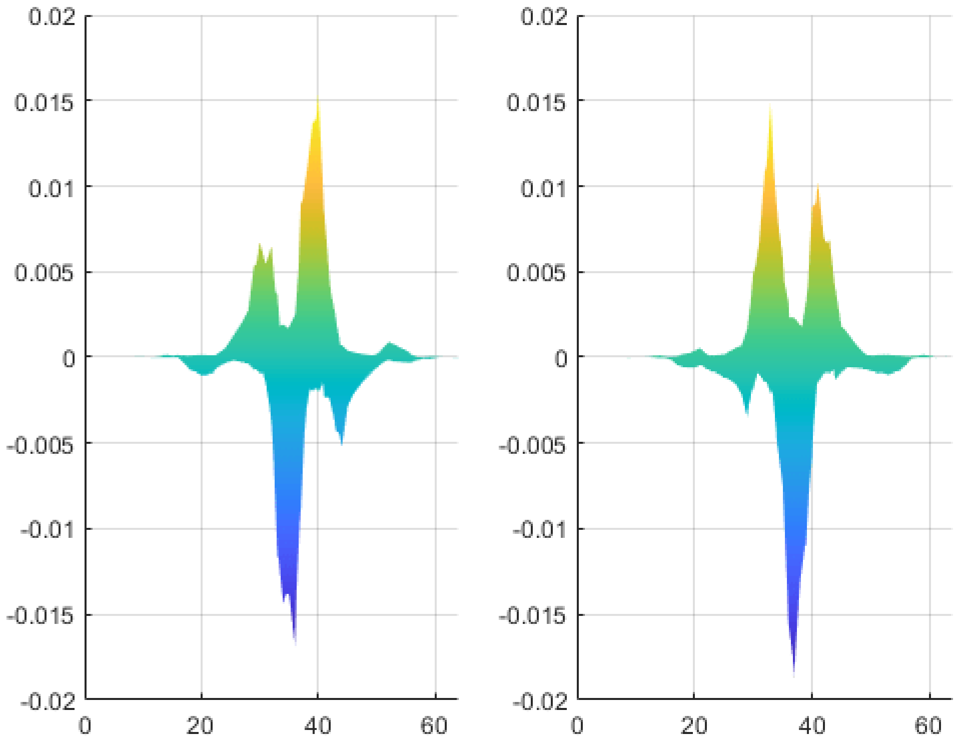
```
d{3}(4,5,19) = 1j;
```

```
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
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

`dualtree3` | `wavedec3` | `waverec3` | `dddtree2` | `dualtree2` | `dualtree`

Topics

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

Introduced in R2017a

idwpt

Multisignal 1-D inverse wavelet packet transform

Syntax

```
xrec = idwpt(wpt,l)
xrec = idwpt(wpt,l,wname)
xrec = idwpt(wpt,l,LoR,HiR)
xrec = idwpt( ___ , 'Boundary',ExtensionMode)
```

Description

`xrec = idwpt(wpt,l)` inverts the discrete wavelet packet transform (DWPT) of the terminal node wavelet packet tree `wpt` using the bookkeeping vector `l`. The `idwpt` function assumes that you obtained `wpt` and `l` using `dwpt` with the `fk18` wavelet and default settings.

If the input to `dwpt` was one signal, `xrec` is a column vector. If the input was a multichannel signal, `xrec` is a matrix, where each matrix column corresponds to a channel.

`xrec = idwpt(wpt,l,wname)` uses the wavelet specified by `wname` to invert the DWPT. `wname` must be recognized by `wavemngr`. The specified wavelet must be the same wavelet used to obtain the DWPT.

`xrec = idwpt(wpt,l,LoR,HiR)` uses the scaling (lowpass) filter, `LoR`, and wavelet (highpass) filter, `HiR`. The synthesis filter pair `LoR` and `HiR` must be associated with the same wavelet used in the DWPT.

`xrec = idwpt( ___ , 'Boundary',ExtensionMode)` specifies the mode to extend the signal. `ExtensionMode` can be either `'reflection'` (default) and `'periodic'`. By setting `ExtensionMode` to `'periodic'` or `'reflection'`, the wavelet packet coefficients at each level are extended based on the modes `'per'` or `'sym'` in `dwtmode`, respectively. `ExtensionMode` must be the same mode used in the DWPT.

Examples

Inverse Wavelet Packet Transform

This example shows how to perform the inverse wavelet packet transform using synthesis filters.

Obtain the DWPT of an ECG signal using `dwpt` with default settings.

```
load wecg
[wpt,l] = dwpt(wecg);
```

By default, `dwpt` uses the `fk18` wavelet. Obtain the synthesis (reconstruction) filters associated with the wavelet.

```
[~,~,lor,hir] = wfilters('fk18');
```

Invert the DWPT using the synthesis filters and demonstrate perfect reconstruction.

```
xrec = idwpt(wpt,l,lor,hir);
norm(wecg-xrec,'inf')
```

```
ans =

   4.9236e-11
```

Change Boundary Extension Mode

Obtain the DWPT of an ECG signal using dwpt and periodic extension.

```
load wecg
[wpt,l] = dwpt(wecg,'Boundary','periodic');
```

By default, idwpt uses symmetric extension. Invert the DWPT using periodic and symmetric extension modes.

```
xrecA = idwpt(wpt,l,'Boundary','periodic');
xrecB = idwpt(wpt,l);
```

Demonstrate perfect reconstruction only when the extension modes of the forward and inverse DWPT agree.

```
fprintf('Periodic/Periodic : %f\n',norm(wecg-xrecA,'inf'))
```

```
Periodic/Periodic : 0.000000
```

```
fprintf('Periodic/Symmetric: %f\n',norm(wecg-xrecB,'inf'))
```

```
Periodic/Symmetric: 1.477907
```

PR Biorthogonal Filters

This example shows how to take an expression of a biorthogonal filter pair and construct lowpass and highpass filters to produce a perfect reconstruction (PR) pair in Wavelet Toolbox™.

The LeGall 5/3 filter is the wavelet used in JPEG2000 for lossless image compression. The lowpass (scaling) filters for the LeGall 5/3 wavelet have five and three nonzero coefficients respectively. The expressions for these two filters are:

$$H_0(z) = 1/8(- z^2 + 2z + 6 + 2z^{-1} - z^{-2})$$

$$H_1(z) = 1/2(z + 2 + z^{-1})$$

Create these filters.

```
H0 = 1/8*[-1 2 6 2 -1];
H1 = 1/2*[1 2 1];
```

Many of the discrete wavelet and wavelet packet transforms in Wavelet Toolbox rely on the filters being both even-length and equal in length in order to produce the perfect reconstruction filter bank associated with these transforms. These transforms also require a specific normalization of the coefficients in the filters for the algorithms to produce a PR filter bank. Use the `biorfilt` function on the lowpass prototype functions to produce the PR wavelet filter bank.

```
[LoD,HiD,LoR,HiR] = biorfilt(H0,H1);
```

The sum of the lowpass analysis and synthesis filters is now equal to $\sqrt{2}$.

```
sum(LoD)
```

```
ans = 1.4142
```

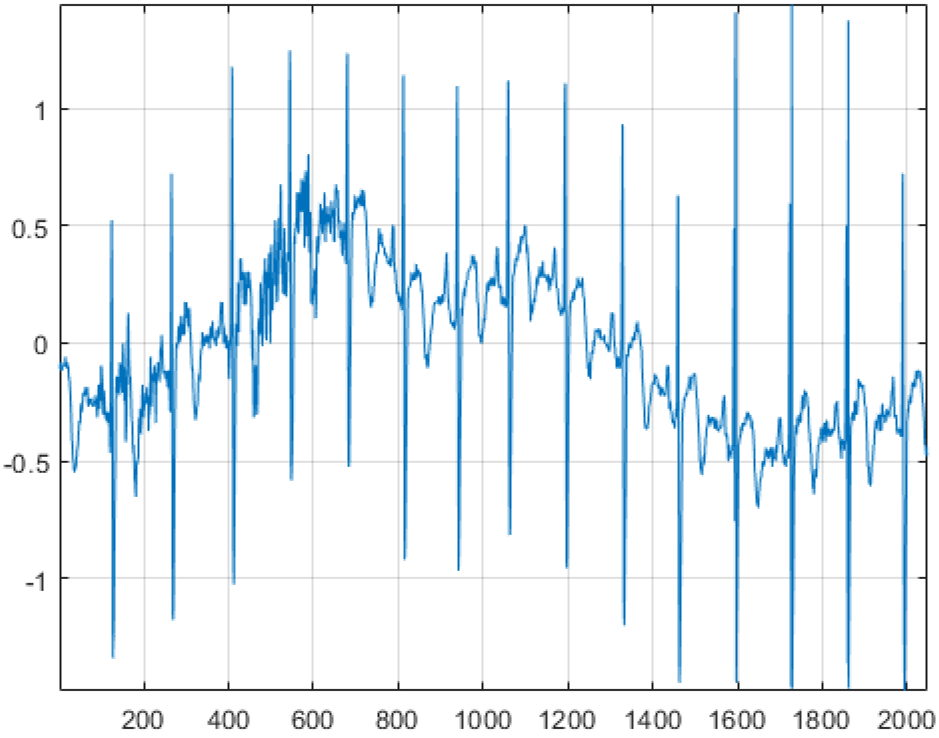
```
sum(LoR)
```

```
ans = 1.4142
```

The wavelet filters sum, as required, to zero. The L2-norms of the lowpass analysis and highpass synthesis filters are equal. The same holds for the lowpass synthesis and highpass analysis filters.

Now you can use these filters in discrete wavelet and wavelet packet transforms and achieve a PR wavelet packet filter bank. To demonstrate this, load and plot an ECG signal.

```
load wecg
plot(wecg)
axis tight
grid on
```

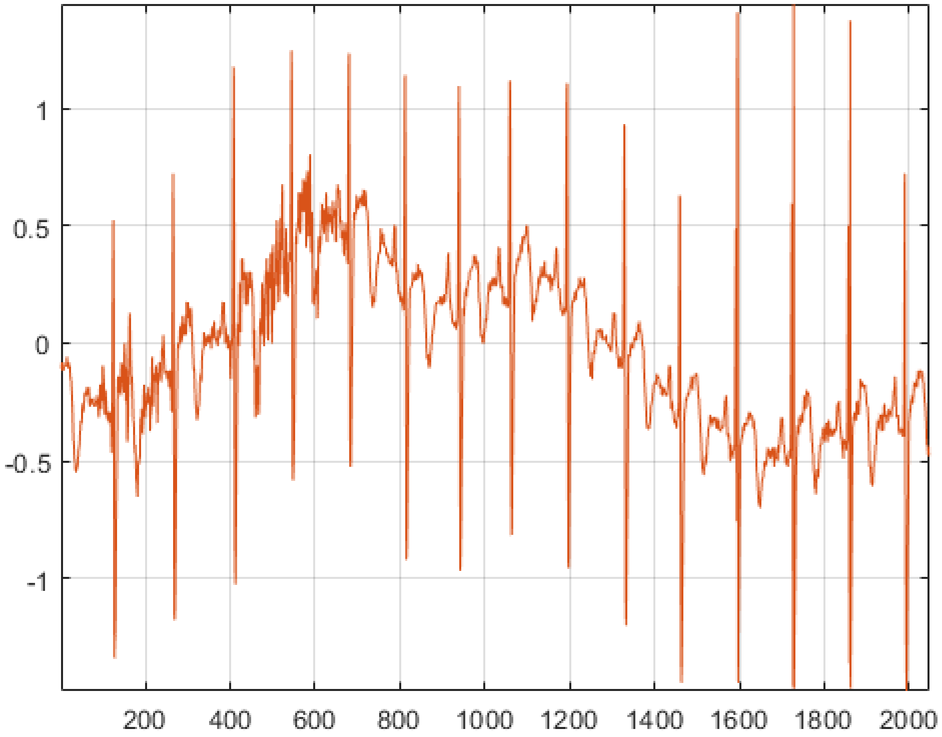


Obtain the discrete wavelet packet transform of the ECG signal using the LeGall 5/3 filter set.

```
[wpt,L] = dwpt(wecg,LoD,HiD);
```

Now use the reconstruction (synthesis) filters to reconstruct the signal and demonstrate perfect reconstruction.

```
xrec = idwpt(wpt,L,LoR,HiR);
plot([wecg xrec])
axis tight, grid on;
```



```
norm(wecg-xrec,'Inf')
```

```
ans = 3.1086e-15
```

You can also use this filter bank in the 1-D and 2-D discrete wavelet transforms. Read and plot an image.

```
im = imread('woodsculp256.jpg');
image(im); axis off;
```

Obtain the 2-D wavelet transform using the LeGall 5/3 analysis filters.

```
[C,S] = wavedec2(im,3,LoD,HiD);
```

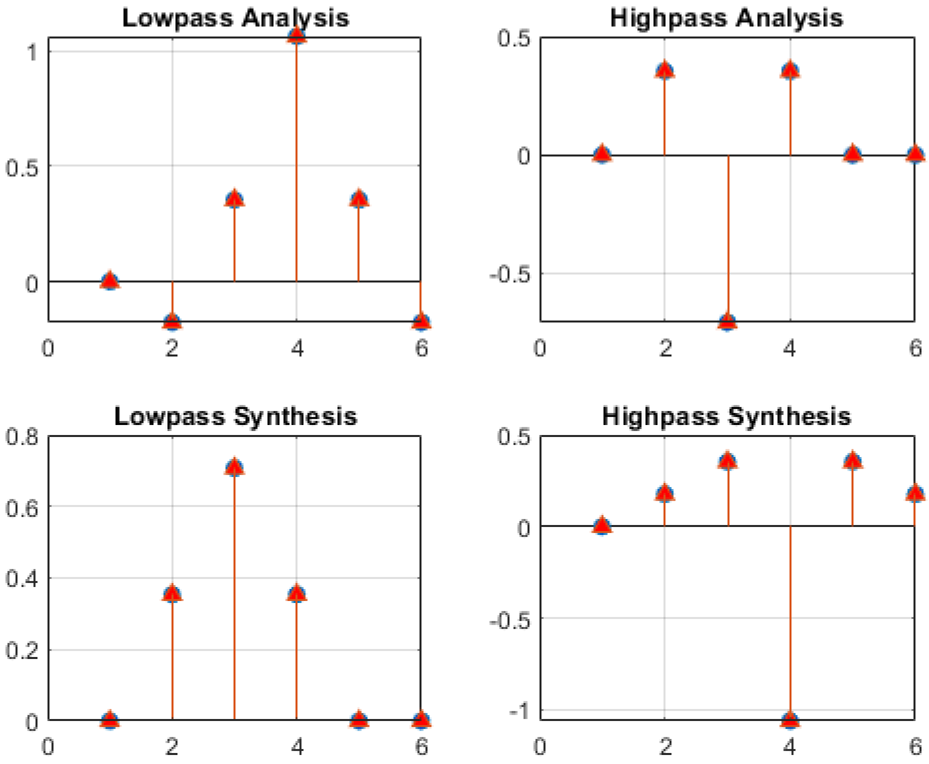
Reconstruct the image using the synthesis filters.

```
imrec = waverec2(C,S,LoR,HiR);
image(uint8(imrec)); axis off;
```

The LeGall 5/3 filter is equivalent to the built-in `'bior2.2'` wavelet in Wavelet Toolbox. Use the `'bior2.2'` filters and compare with the LeGall 5/3 filters.

```
[LD,HD,LR,HR] = wfilters('bior2.2');
subplot(2,2,1)
hl = stem([LD' LoD']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Lowpass Analysis')
subplot(2,2,2)
hl = stem([HD' HiD']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Highpass Analysis')
subplot(2,2,3)
hl = stem([LR' LoR']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
```

```
title('Lowpass Synthesis')
subplot(2,2,4)
hl = stem([HR' HiR']);
hl(1).MarkerFaceColor = [0 0 1];
hl(1).Marker = 'o';
hl(2).MarkerFaceColor = [1 0 0];
hl(2).Marker = '^';
grid on
title('Highpass Synthesis')
```



Input Arguments

wpt — Terminal node wavelet packet tree

cell array

Terminal node wavelet packet tree, specified as a cell array. `wpt` is the output of `dwpt` with the `'FullTree'` value set to `false`.

Example: `[wpt,l] = dwpt(X,'Level',3,'FullTree',false)` returns the terminal node wavelet packet tree of the three-level wavelet packet decomposition of X.

Data Types: `single` | `double`

l — Bookkeeping vector

vector of positive integers

Bookkeeping vector, specified as a vector of positive integers. The vector `l` is the output of `dwpt`. The bookkeeping vector contains the length of the input signal and the number of coefficients by level, and is required for perfect reconstruction.

Data Types: `single` | `double`

`wname` — Wavelet
`'fk18'` (default) | character vector | string scalar

Wavelet to use in the inverse DWPT, specified as a character vector or string scalar. `wname` must be recognized by `wavemngr`. The specified wavelet must be the same wavelet used to obtain the DWPT.

You cannot specify both `wname` and a filter pair, `LoD` and `HiD`.

Example: `xrec = idwpt(wpt,l,"sym4")` specifies the `sym4` wavelet.

`LoR,HiR` — Wavelet synthesis filters
real-valued vectors

Wavelet synthesis (reconstruction) filters to use in the inverse DWPT, specified as a pair of real-valued vectors. `LoR` is the scaling (lowpass) synthesis filter, and `HiR` is the wavelet (highpass) synthesis filter. The synthesis filter pair must be associated with the same wavelet as used in the DWPT. You cannot specify both `wname` and a filter pair, `LoR` and `HiR`. See `wfilters` for additional information.

Note `idwpt` does not check that `LoR` and `HiR` satisfy the requirements for a perfect reconstruction wavelet packet filter bank. See “PR Biorthogonal Filters” on page 1-648 for an example of how to take a published biorthogonal filter and ensure that the analysis and synthesis filters produce a perfect reconstruction wavelet packet filter bank using `idwpt`.

`ExtensionMode` — Wavelet packet transform boundary handling
`'reflection'` (default) | `'periodic'`

Wavelet packet transform boundary handling, specified as `'reflection'` or `'periodic'`. When set to `'reflection'` or `'periodic'`, the wavelet packet coefficients are extended at each level based on the `'sym'` or `'per'` mode in `dwtmode`, respectively. `ExtensionMode` must be the same mode used in the DWPT. If unspecified, `ExtensionMode` defaults to `'reflection'`.

References

[1] Wickerhauser, Mladen Victor. *Adapted Wavelet Analysis from Theory to Software.* Wellesley, MA: A.K. Peters, 1994.

[2] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[3] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

See Also

`dwpt` | `imodwpt`

Introduced in R2020a

idwt

Single-level inverse discrete 1-D wavelet transform

Syntax

```
x = idwt(cA,cD,wname)
x = idwt(cA,cD,LoR,HiR)
x = idwt( ___ ,l)
x = idwt( ___ ,'mode',mode)

x = idwt(cA,[], ___ )
x = idwt([],cD, ___ )
```

Description

`x = idwt(cA,cD,wname)` returns the single-level one-dimensional wavelet reconstruction `x` based on the approximation and detail coefficients `cA` and `cD`, respectively, using the wavelet specified by `wname`. For more information, see `dwt`.

Let `la` be the length of `cA` (which also equals the length of `cD`), and `lf` the length of the reconstruction filters associated with `wname` (see `wfilters`). If the DWT extension mode is set to periodization, then the length of `x` is equal to `2la`. Otherwise, the length of `x` is equal to `2la- 2lf+2`. For more information, see `dwtmode`.

`x = idwt(cA,cD,LoR,HiR)` uses the specified lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

`x = idwt( ___ ,l)` returns the length-`l` central portion of the reconstruction. This argument can be added to any of the previous input syntaxes

`x = idwt( ___ ,'mode',mode)` uses the specified DWT extension mode `mode`. For more information, see `dwtmode`. This argument can be added to any of the previous syntaxes.

`x = idwt(cA,[], ___ )` returns the single-level reconstructed approximation coefficients based on the approximation coefficients `cA`.

`x = idwt([],cD, ___ )` returns the single-level reconstructed detail coefficients based on the detail coefficients `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 Biorthogonal 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
```

Input Arguments

`cA` — Approximation coefficients
vector

Approximation coefficients, specified as a vector. `cA` is expected to be the output of `dwt`.

Data Types: `single` | `double`

`cD` — Detail coefficients
vector

Detail coefficients, specified as a vector. `cD` is expected to be the output of `dwt`.

Data Types: `single` | `double`

`wname` — Wavelet
character vector | string scalar

Wavelet used to compute the single-level inverse discrete wavelet transform (IDWT), specified as a character vector or string scalar. The wavelet must be recognized by `wavemngr`. The wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

The wavelet specified must be the same wavelet used to obtain the approximation and detail coefficients.

Example: `'db4'`

`LoR,HiR` — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. `LoR` is the lowpass reconstruction filter, and `HiR` is the highpass reconstruction filter. The lengths of `LoR` and `HiR` must be equal. See `wfilters` for additional information.

Data Types: `single` | `double`

`l` — Length of central portion
positive integer

Length of central portion of reconstruction, specified as a positive integer. If `xrec = idwt(cA,cD,wname)`, then `l` cannot exceed `length(xrec)`.

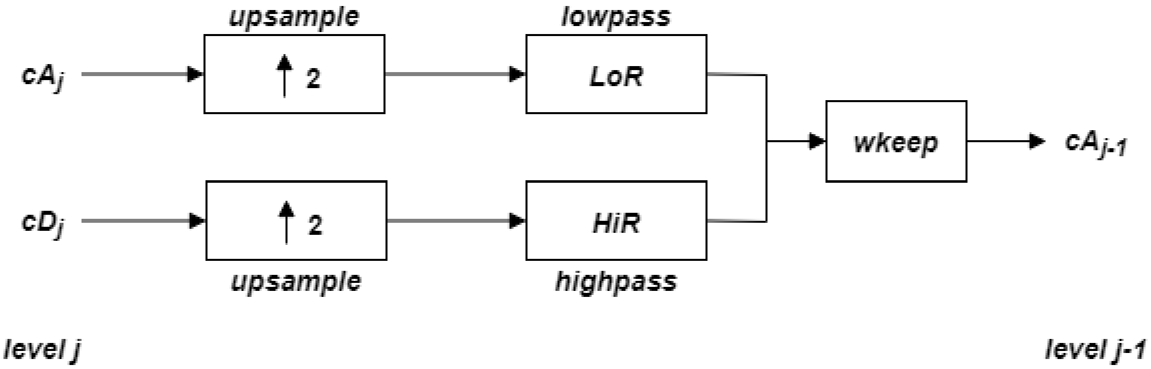
Data Types: `single` | `double`

`mode` — DWT extension mode
character vector | string scalar

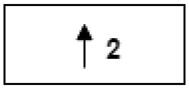
DWT extension mode used in the wavelet reconstruction, specified as a character vector or string scalar. For possible extension modes, see `dwtmode`.

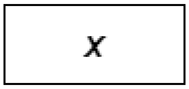
Algorithms

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.



where

- 

— Insert zeros at even-indexed elements
- 

— Convolve with filter *X*
-

— Take the central part of *U* with the convenient length

References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`dwt` | `dwtmode` | `wfilters`

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,LoR,HiR)
x = idwt2( ___ ,s)
x = idwt2( ___ ,'mode',mode)

x = idwt2(cA,[],[],[], ___ )
x = idwt2([],cH,[],[], ___ )
x = idwt2([],[],cV,[], ___ )
x = idwt2([],[],[],cD, ___ )
```

Description

`x = idwt2(cA,cH,cV,cD,wname)` performs a single-level two-dimensional wavelet reconstruction based on the approximation matrix `cA` and details matrices `cH`, `cV`, and `cD` (horizontal, vertical, and diagonal, respectively) using the wavelet specified by `wname`. For additional information, see `dwt2`.

Let `sa = size(cA) = size(cH) = size(cV) = size(cD)`, and let `lf` equal the length of the reconstruction filters associated with `wname`. If the DWT extension mode is set to periodization, the size of `x`, `sx` is equal to `2*sa`. For other extension modes, `sx = 2*sa-lf+2`. For additional information, see `dwtmode`.

`x = idwt2(cA,cH,cV,cD,LoR,HiR)` uses the specified lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

`x = idwt2( ___ ,s)` returns the size-`s` central portion of the reconstruction using any of the previous syntaxes.

`x = idwt2( ___ ,'mode',mode)` computes the wavelet reconstruction using the specified extension mode `mode`. For additional information, see `dwtmode`. This syntax can be used with any of the previous syntaxes.

`x = idwt2(cA,[],[],[], ___ )` returns the single-level reconstructed approximation coefficients matrix `x` based on the approximation coefficients matrix `cA`.

`x = idwt2([],cH,[],[], ___ )` returns the single-level reconstructed approximation coefficients matrix `x` based on horizontal detail coefficients matrix `cH`.

`x = idwt2([],[],cV,[], ___ )` returns the single-level reconstructed approximation coefficients matrix `x` based on vertical detail coefficients matrix `cV`.

`x = idwt2([],[],[],cD, ___ )` returns the single-level reconstructed approximation coefficients matrix `x` based on diagonal detail coefficients matrix `cD`.

Examples

Single-Level 2-D Wavelet Reconstruction

Load an image.

```
load woman
whos X
```

```
  Name         Size               Bytes  Class     Attributes

  X          256x256             524288  double              
```

The workspace variable X contains the image. Perform a single-level wavelet decomposition of X use the db4 wavelet.

```
[cA1,cH1,cV1,cD1] = dwt2(X,'db4');
```

Invert the decomposition of X using the coefficients at level 1.

```
A0 = idwt2(cA1,cH1,cV1,cD1,'db4');
```

Check for perfect reconstruction.

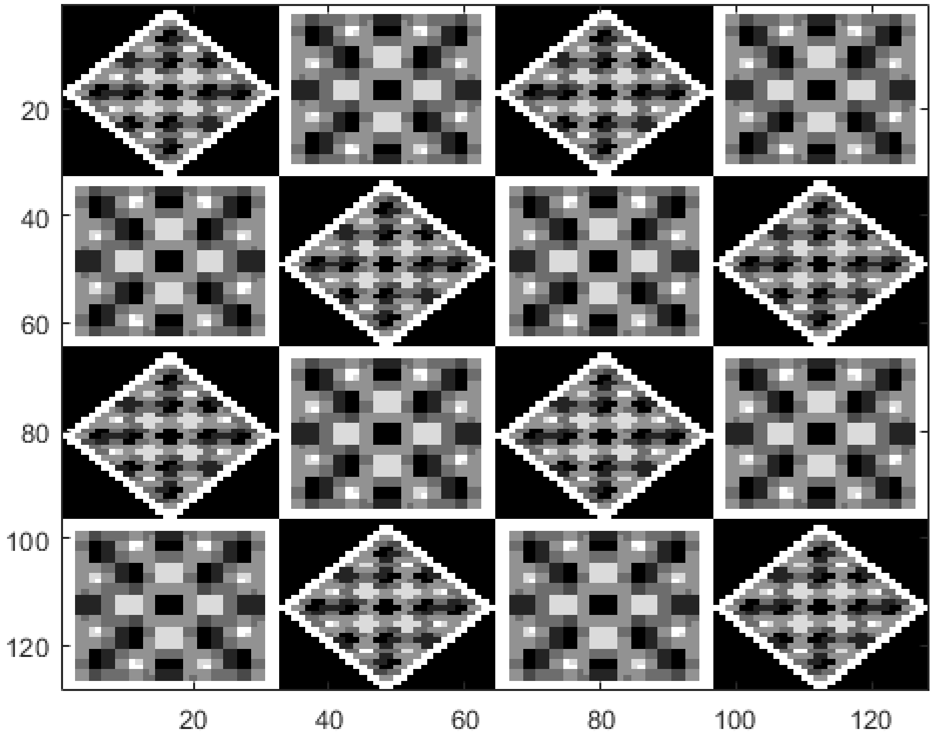
```
max(abs(X(:)-A0(:)))
```

```
ans = 3.4174e-10
```

Wavelet Reconstruction of Detail Coefficients

Load an image.

```
load tartan
imagesc(X)
colormap(gray)
```



Perform a single-level wavelet decomposition using the `db4` wavelet.

```
[cA,cH,cV,cD] = dwt2(X,'db4');
```

Obtain the wavelet reconstruction using only the diagonal detail coefficients.

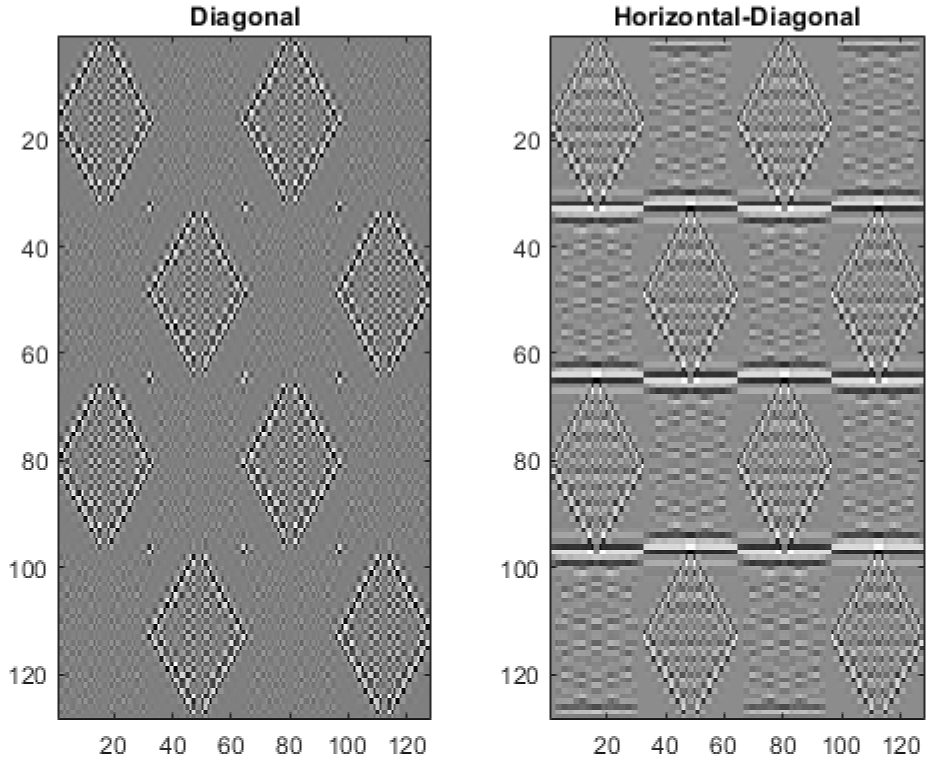
```
xrecD = idwt2([],[],[],cD,'db4');
```

Obtain a second wavelet reconstruction, this time using the horizontal and diagonal detail coefficients.

```
xrecHD = idwt2([],cH,[],cD,'db4');
```

Display both reconstructions.

```
subplot(1,2,1)
imagesc(xrecD)
title('Diagonal')
subplot(1,2,2)
imagesc(xrecHD)
title('Horizontal-Diagonal')
colormap(gray)
```



Input Arguments

cA — Approximation coefficients

array

Approximation coefficients, specified as an array. cA is expected to be the output of dwt2.

Data Types: double

cH — Horizontal detail coefficients

array

Horizontal detail coefficients, specified as an array. cD is expected to be the output of dwt2.

Data Types: double

cV — Vertical detail coefficients

array

Vertical detail coefficients, specified as an array. cV is expected to be the output of dwt2.

Data Types: double

cD — Diagonal detail coefficients

array

Diagonal detail coefficients, specified as an array. cD is expected to be the output of dwt2.

Data Types: `double`

`wname` — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. `idwt2` supports only orthogonal or biorthogonal wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

The wavelet specified must be the same wavelet used to obtain the approximation and details coefficients.

`LoR,HiR` — Wavelet reconstruction filters

even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. `LoR` is the lowpass reconstruction filter, and `HiR` is the highpass reconstruction filter. The lengths of `LoR` and `HiR` must be equal. See `wfilters` for additional information.

Data Types: `double`

`s` — Size of central portion

two-element vector

Size of central portion of reconstruction to return, specified as a two element vector of positive integers. `s` must be less than `sx`, the size of `x`.

Data Types: `double`

`mode` — DWT extension mode

character vector | string scalar

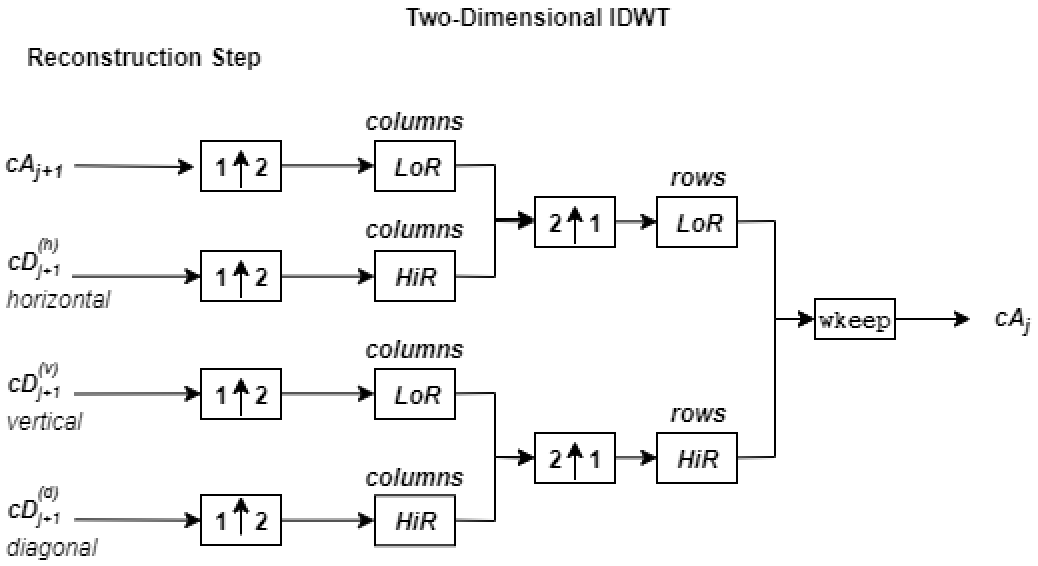
DWT extension mode used in the wavelet reconstruction, specified as a character vector or string scalar. For possible extension modes, see `dwtmode`.

Tips

- If `cA`, `cH`, `cV`, and `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 `image` and `imfinfo`.

Algorithms

The 2-D wavelet reconstruction algorithm for images is similar to the one-dimensional case. The two-dimensional wavelet and scaling functions are obtained by taking the tensor products of the one-dimensional wavelet and scaling functions. This kind of two-dimensional inverse DWT leads to a reconstruction of approximation coefficients at level j from four components: the approximation at level $j+1$, and the details in three orientations (horizontal, vertical, and diagonal). The following chart describes the basic reconstruction steps for images.



where

- $\boxed{2\uparrow 1}$ — Upsample columns: insert zeros at odd-indexed columns
- $\boxed{1\uparrow 2}$ — Upsample rows: insert zeros at odd-indexed rows
- rows $\boxed{X}$ — Convolve with filter *X* the rows of the entry
- columns $\boxed{X}$ — Convolve with filter *X* the columns of the entry

References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S.G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11, no. 7 (July 1989): 674–93. https://doi.org/10.1109/34.192463.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

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'))
```

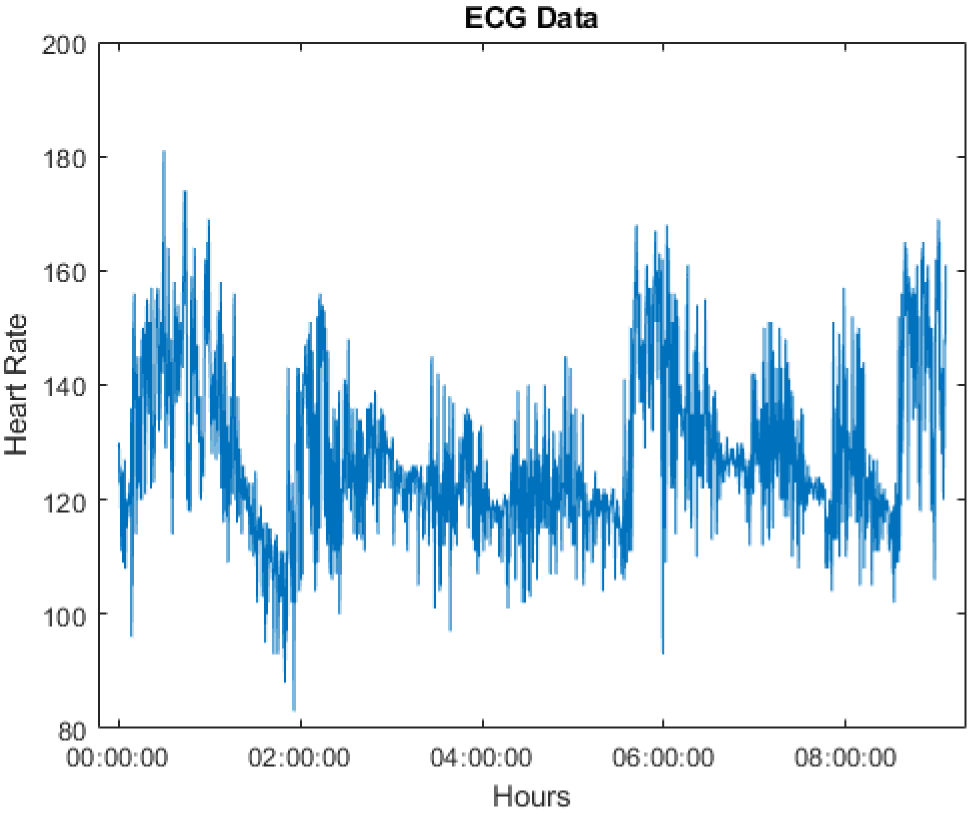
```
ans = 4.4409e-15
```

Inverse Haar Transform of ECG Data

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

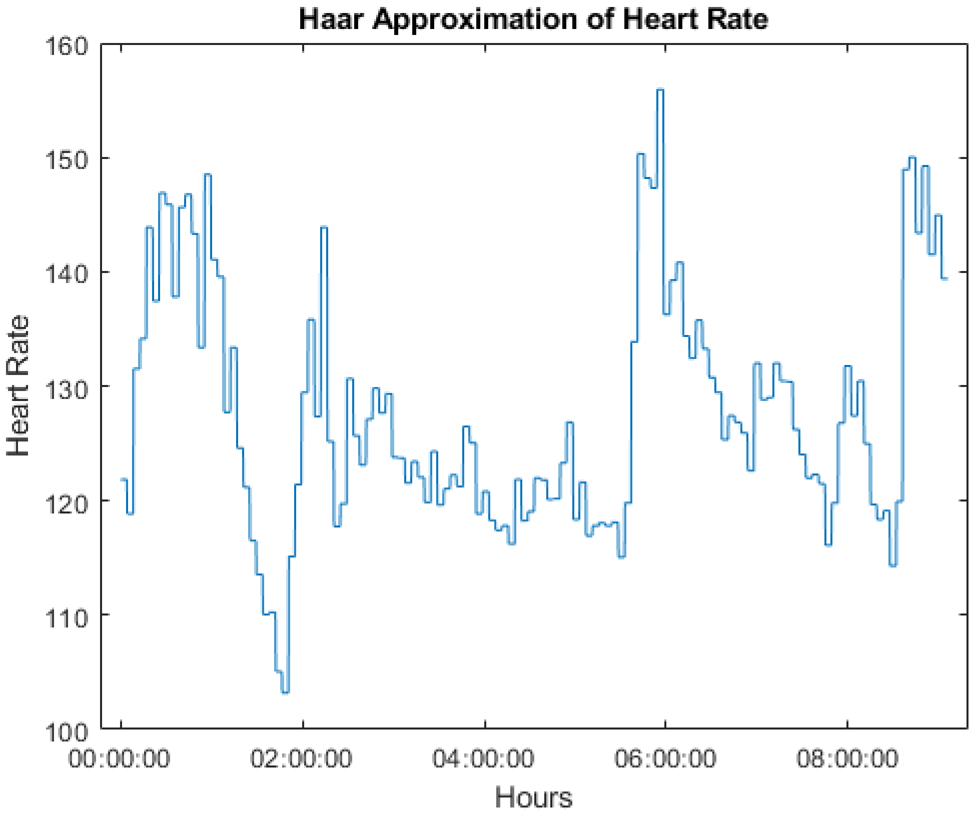
Load and plot the ECG data.

```
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.

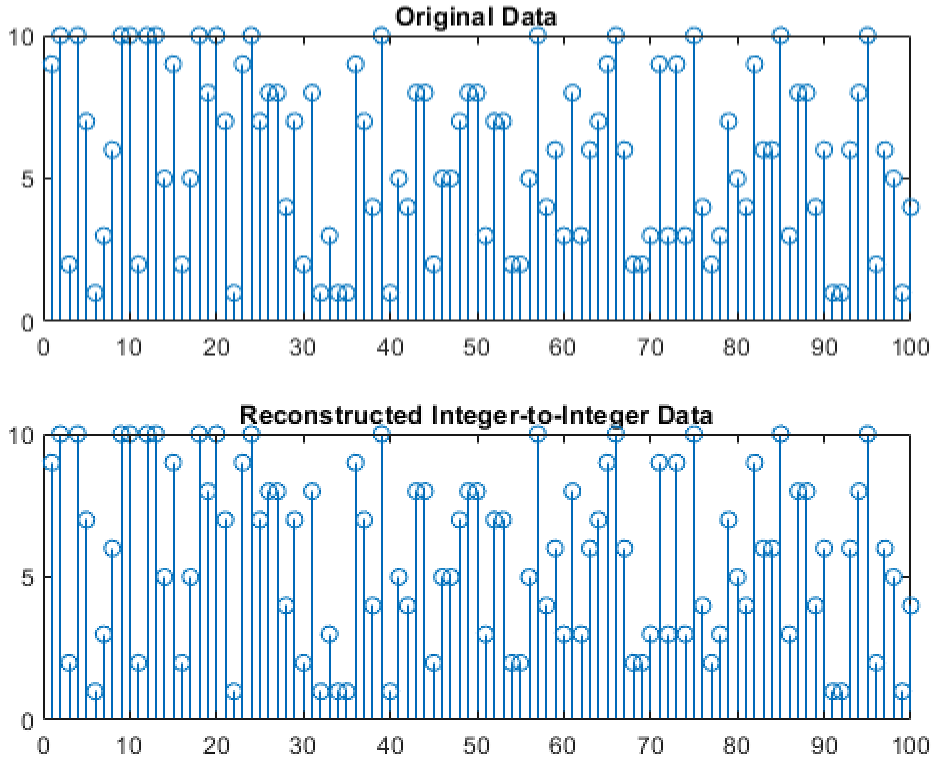
```
x = randi(10,100,1);
```

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
```

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: `single` | `double`

d — Detail coefficients

scalar | vector | matrix | cell array

Detail coefficients, specified as a scalar, vector, matrix, or cell array of wavelet coefficients. `d` is an output from the `haart` function. The number of detail coefficients depends on the selected level and the length of the input. If d is a cell array, the elements of d are ordered from finest to coarsest resolution.

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.

If d is a vector or matrix, the Haar transform was computed only down to one level coarser in resolution.

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: single | 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 either 'noninteger' or 'integer'. '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: single | double

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

haart | ihaart2 | haart2

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.

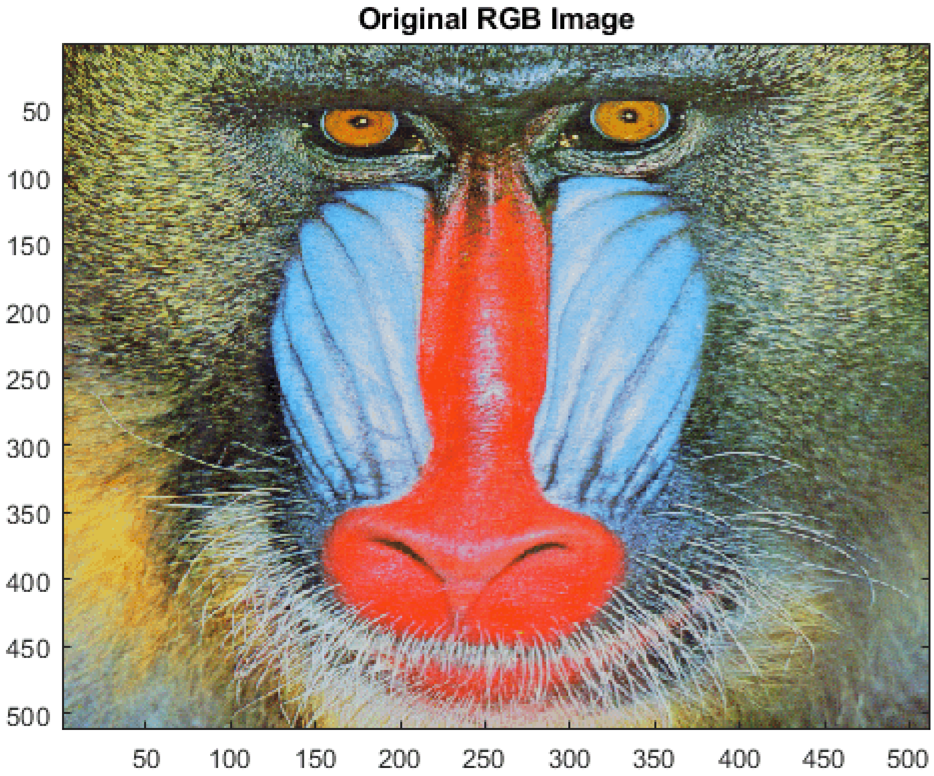
```
im = imread('mandrill.png');
[a,h,v,d] = haart2(im);
```

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

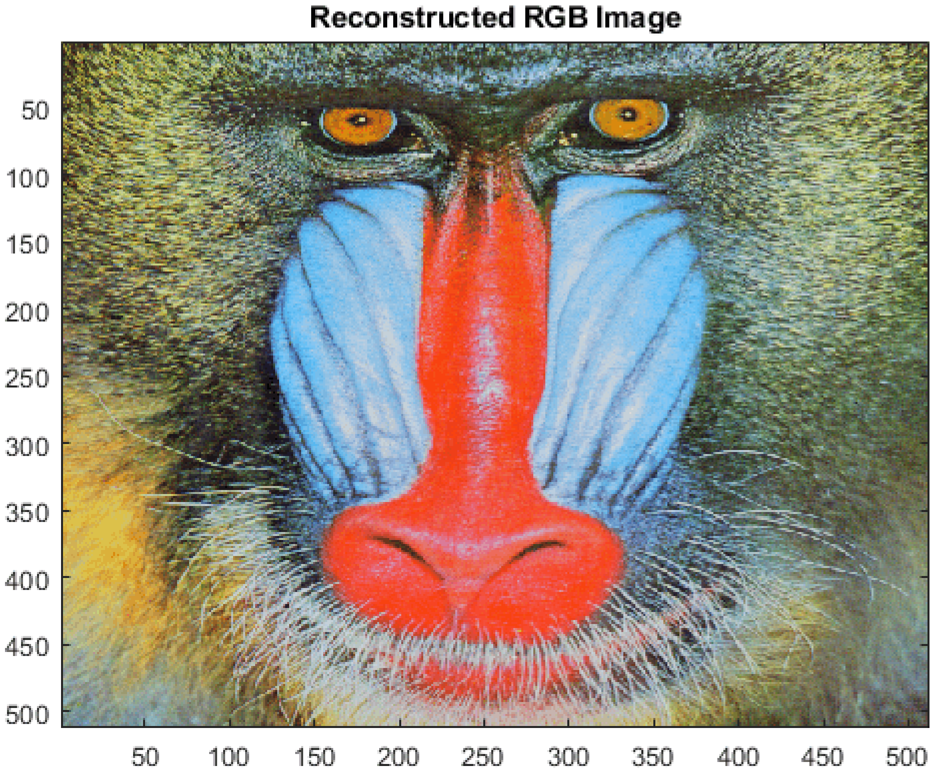
```
xrec = ihaart2(a,h,v,d);
```

Compare the original and reconstructed images.

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



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

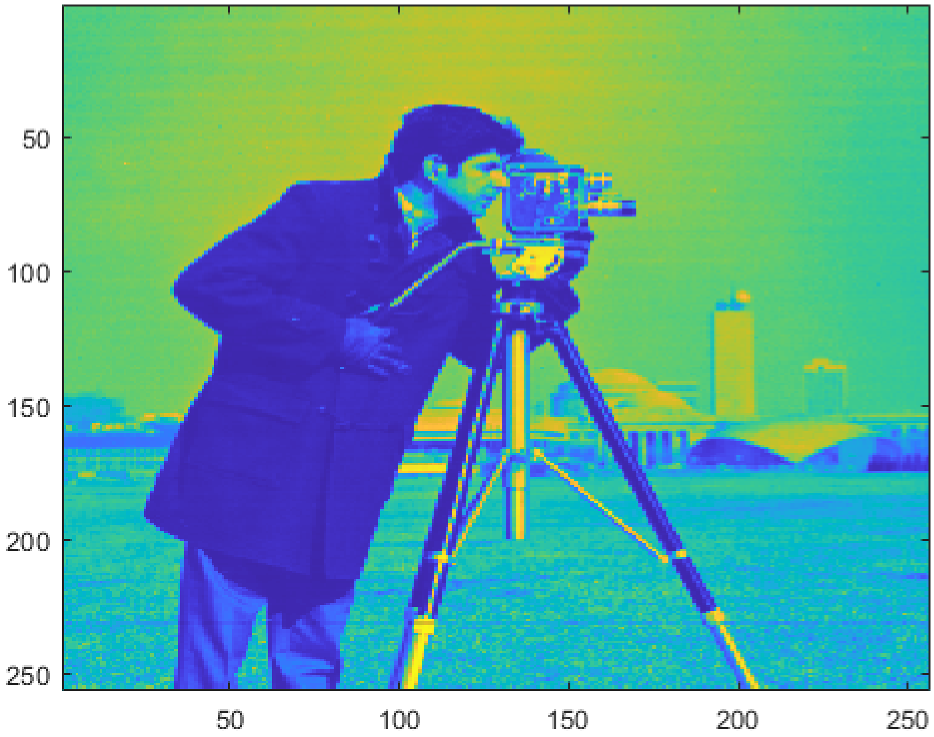


Inverse 2-D Haar Transform of Image Limited to Specified Level

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)
```

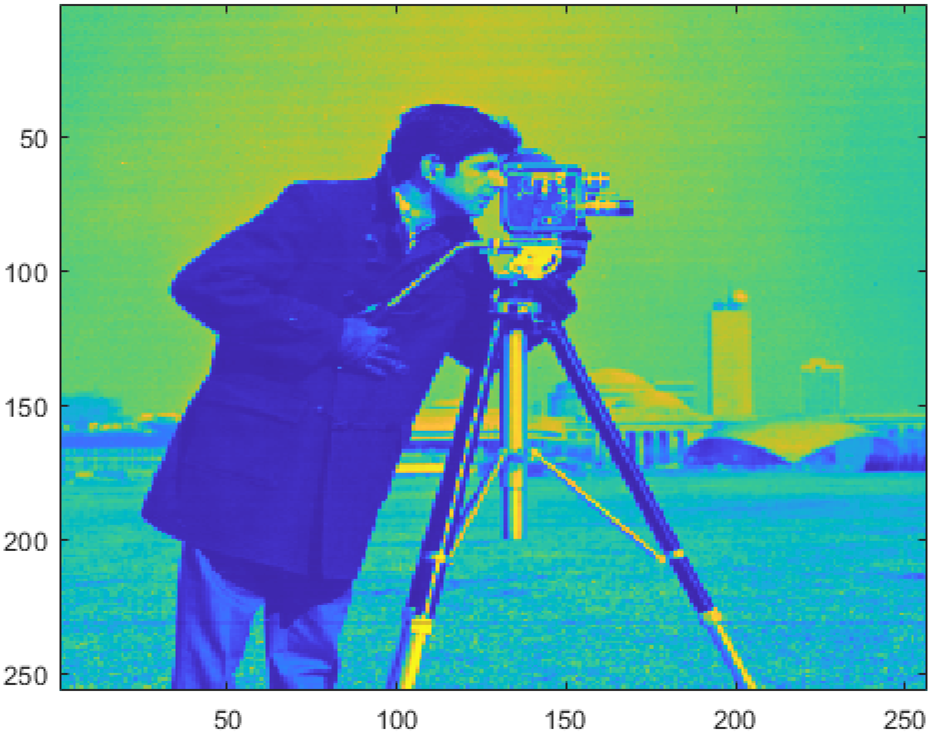


Obtain the 2-D Haar transform using the default maximum number of levels.

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

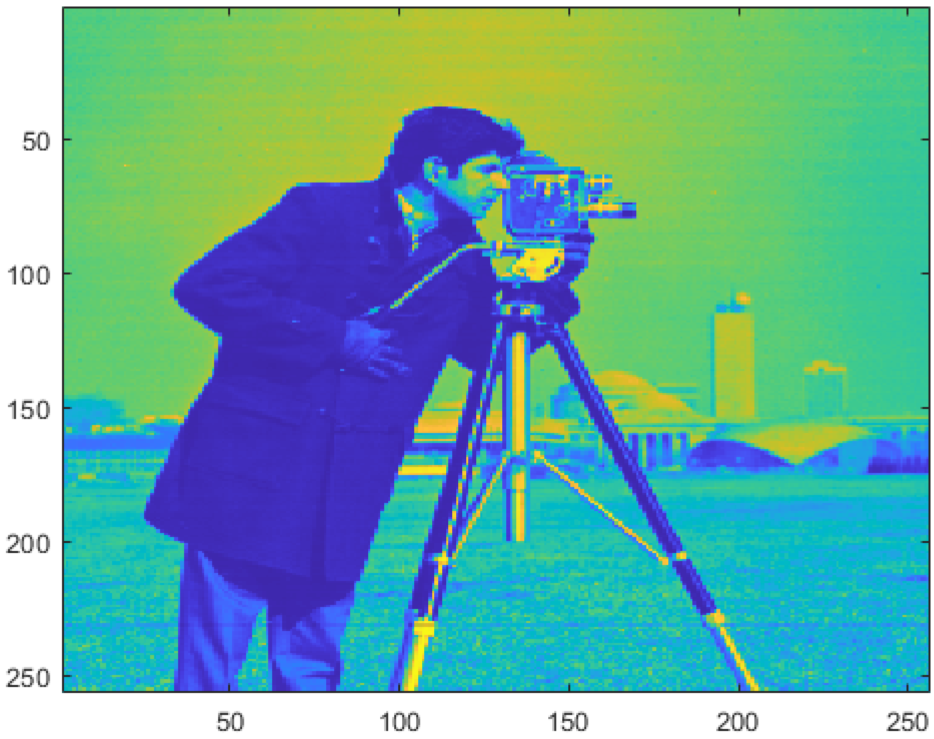
Reconstruct the image using the inverse 2-D Haar transform and view the image. Notice the near-perfect reconstruction.

```
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.

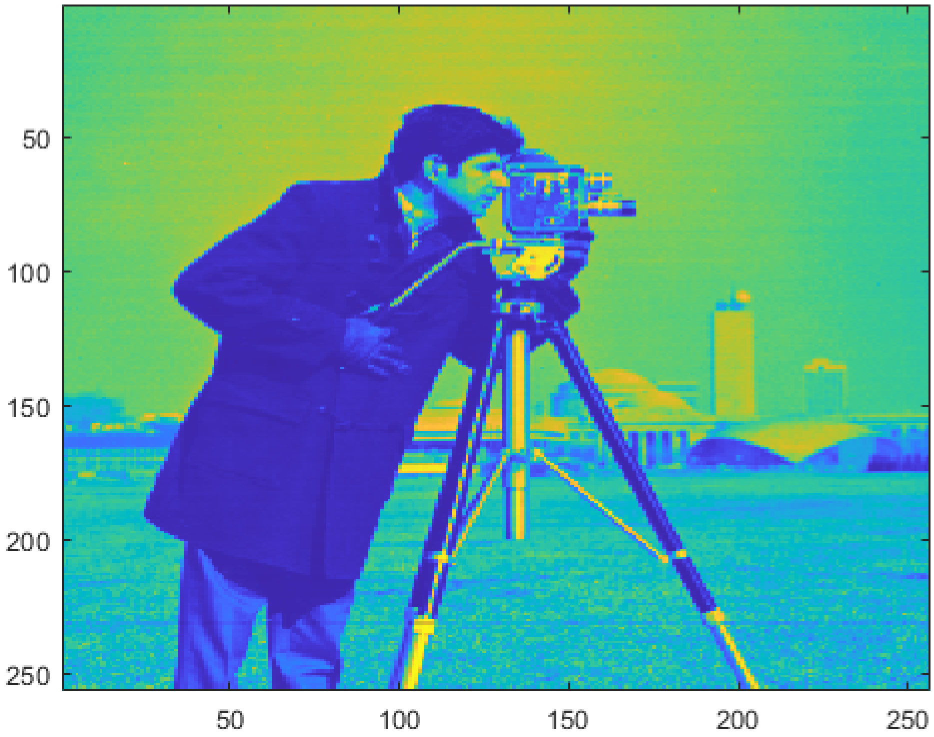
```
im = imread('cameraman.tif');
```

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.

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: single | 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: single | 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: `single` | `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: `single` | `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 either `'noninteger'` or `'integer'`. `'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: `single` | `double`

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`haart` | `ihaart` | `haart2`

Topics

"Haar Transforms for Time Series Data and Images"

Introduced in R2016b

ilwt

Inverse 1-D lifting wavelet transform

Syntax

```
xr = ilwt(ca,cd)
xr = ilwt(ca,cd,Name,Value)
```

Description

`xr = ilwt(ca,cd)` returns the 1-D inverse wavelet transform based on the approximation coefficients, `ca`, and cell array of detail coefficients, `cd`. By default, `ilwt` assumes you used the lifting scheme associated with the `db1` wavelet to obtain `ca` and `cd`. If you do not modify the coefficients, `xr` is a perfect reconstruction of the signal.

`xr = ilwt(ca,cd,Name,Value)` specifies options using one or more name-value arguments. For example, `xr = ilwt(ca,cd,'Wavelet','db2')` specifies the orthogonal wavelet `db2`.

For perfect reconstruction, all name-value arguments must match those used in `lwt` to obtain `ca` and `cd`.

Examples

Inverse LWT of Integer-Valued Signal

Create a lifting scheme associated with the `db3` wavelet. Specify an integer-valued signal whose length is a power of 2.

```
lsc = liftingScheme('Wavelet','db3');
n = 8;
sig = 1:2^n;
```

Use the lifting scheme to obtain the integer-valued LWT of the signal down to the maximum decomposition level.

```
[ca,cd] = lwt(sig,'LiftingScheme',lsc,'Int2Int',true);
```

Confirm the detail coefficients `cd` are a cell array whose length is equal to the exponent of 2.

```
length(cd)
```

```
ans = 8
```

Obtain the inverse LWT up to level 0. Confirm perfect reconstruction.

```
xrec0 = ilwt(ca,cd,'LiftingScheme',lsc,'Int2Int',true,'Level',0);
max(abs(xrec0(:)-sig(:)))
```

```
ans = 0
```

Obtain the inverse LWT up to level 1.

```
xrec1 = ilwt(ca,cd,'LiftingScheme',lsc,'Int2Int',true,'Level',1);
```

Obtain the level 1 decomposition of the signal. Confirm the approximation coefficients are equal to `xrec1`.

```
[ca,cd] = lwt(sig,'LiftingScheme',lsc,'Int2Int',true,'Level',1);
max(abs(ca(:)-xrec1(:)))
```

```
ans = 0
```

Inverse LWT of Multichannel Signal

Load the 23 channel EEG data `Espiga3`. The channels are arranged column-wise.

```
load Espiga3
size(Espiga3)
```

```
ans = 1×2

   995    23
```

Obtain the LWT of the multichannel signal using the `db4` wavelet down to the default maximum decomposition level.

```
wv = 'db4';
[ca,cd] = lwt(Espiga3,'Wavelet',wv);
```

Reconstruct the multichannel signal.

```
xrec = ilwt(ca,cd,'Wavelet',wv);
```

Because the original signal has an odd number of samples in each channel, confirm the reconstruction has one more row than the original signal.

```
size(xrec)
```

```
ans = 1×2

   996    23
```

Confirm the last row in the reconstruction is equal to the previous row.

```
max(abs(xrec(end-1,:)-xrec(end,:)))
```

```
ans = 5.6843e-14
```

Delete the last row from the reconstruction. Confirm the result is equal to the original signal.

```
xrec(end,:) = [];
max(abs(Espiga3(:)-xrec(:)))
```

```
ans = 4.5475e-13
```

Input Arguments

ca — Approximation coefficients

scalar | vector | matrix

Approximation (lowpass) coefficients at the coarsest level, specified as a scalar, vector, or matrix. The coefficients are the output of `lwt`.

If `ca` and the elements of `cd` are matrices, `xr` is a matrix where each column is the inverse wavelet transform of the corresponding columns in `ca` and `cd`.

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

cd — Detail coefficients

cell array

Detail coefficients, specified as an *L*-by-1 cell array, where *L* is the level of the transform. The elements of `cd` are in order of decreasing resolution. The coefficients are the output of `lwt`.

If `ca` and the elements of `cd` are matrices, `xr` is a matrix where each column is the inverse wavelet transform of the corresponding columns in `ca` and `cd`.

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

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `xr = ilwt(ca,cd,'LiftingScheme',lsc,'Level',1)` uses the `lsc` lifting scheme to perform an inverse wavelet transform up to level 1.

Wavelet — Wavelet

`'db1'` (default) | character vector | string scalar

Orthogonal or biorthogonal wavelet to use in the inverse LWT, specified as a character vector or string scalar. See the Wavelet property of `liftingScheme` for the list of supported wavelets. For perfect reconstruction, the specified wavelet must be the same wavelet that was used to obtain the coefficients `ca` and `cd`.

You cannot specify `'Wavelet'` and `'LiftingScheme'` name-value arguments at the same time.

Example: `xr = ilwt(ca,cd,'Wavelet','bior3.5')` uses the `bior3.5` biorthogonal wavelet.

Data Types: `char` | `string`

LiftingScheme — Lifting scheme

`liftingScheme` object

Lifting scheme to use in the inverse LWT, specified as a `liftingScheme` object. For perfect reconstruction, the specified lifting scheme must be the same lifting scheme that was used to obtain the coefficients `ca` and `cd`.

You cannot specify `'Wavelet'` and `'LiftingScheme'` name-value arguments at the same time.

Example: `xr = ilwt(ca,cd,'LiftingScheme',lScheme)` uses the `lScheme` lifting scheme.

`Level` — Reconstruction level

`0` (default) | positive integer

Reconstruction level, specified as a nonnegative integer less than or equal to `length(cd)`-1. If unspecified, the reconstruction level defaults to 0 and `xr` is a perfect reconstruction of the signal.

Example: `xr = ilwt(ca,cd,'Level',1)` reconstructs the signal up to level 1.

Data Types: `double`

`Extension` — Extension mode

`'periodic'` (default) | `'zeropad'` | `'symmetric'`

Extension mode to use in the inverse LWT, specified as a `'periodic'` (default), `'zeropad'`, or `'symmetric'`. The value of `'Extension'` specifies how to extend the signal at the boundaries.

Example: `xr = ilwt(ca,cd,'Extension','symmetric')` specifies the symmetric extension mode.

`Int2Int` — Integer-valued data handling

`false` or `0` (default) | `true` or `1`

Integer-valued data handling, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `1` (`true`) — Preserve integer-valued data
- `0` (`false`) — Do not preserve integer-valued data

Specify the `'Int2Int'` name-value argument only if all elements of the input are integers.

Example: `xr = ilwt(ca,cd,'Int2Int',true)` preserves integer-valued data.

Output Arguments

`xr` — Inverse wavelet transform

vector | matrix

Inverse wavelet transform of `ca` and `cd`, returned as a vector or matrix. If `ca` is a scalar or vector, and the elements of `cd` are vectors, `xr` is a vector. If `ca` and the elements of `cd` are matrices, `xr` is a matrix where each column is the inverse wavelet transform of the corresponding columns in `ca` and `cd`.

Data Types: `single` | `double`

Compatibility Considerations

`ilwt` input syntax has changed

Behavior changed in R2021a

The `ilwt` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`X = ilwt(CA,CD,W)`	Errors	`X = ilwt(CA,CD,'Wavelet',W)`	You can also set the `LiftingScheme` name-value argument to obtain the inverse LWT.
`X = ilwt(CA,CD,W,LEVEL)`	Errors	`X = ilwt(CA,CD,'Wavelet',W,'Level',LEVEL)`	You can also set the `ExtensionMode` and `Int2Int` name-value arguments.
`X = ilwt(AD_In_Place,W)`	Errors	NA	In-place transforms are no longer supported.

Extended Capabilities

C/C++ Code Generation

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

See Also

`liftingScheme` | `haart` | `lwt` | `ihaart` | `lwtcoef`

Introduced in R2021a

ilwt2

Inverse 2-D lifting wavelet transform

Syntax

```
xr = ilwt2(ll,lh,hl,hh)
xr = ilwt2(ll,lh,hl,hh,Name=Value)
```

Description

`xr = ilwt2(ll,lh,hl,hh)` returns the 2-D inverse wavelet transform based on the approximation coefficients, `ll`, and the horizontal (`lh`), vertical (`hl`), and diagonal (`hh`) wavelet coefficients. By default, `ilwt2` assumes that you used the lifting scheme associated with the `db1` wavelet to obtain the coefficients. If you have not modified the coefficients, `xr` is a perfect reconstruction of the signal.

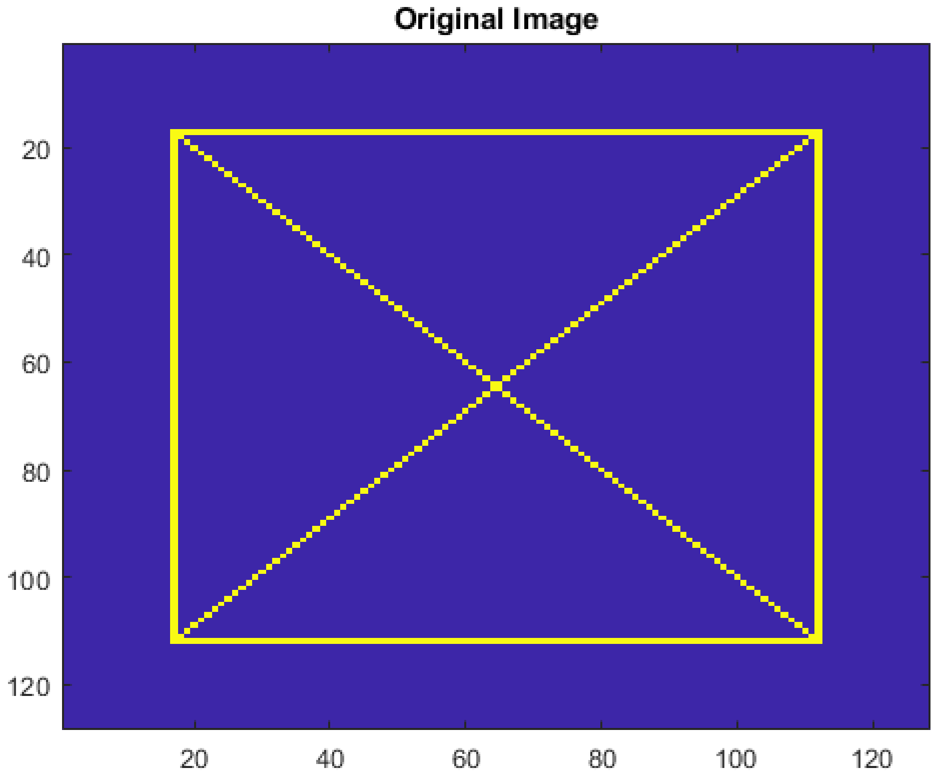
`xr = ilwt2(ll,lh,hl,hh,Name=Value)` specifies options using one or more name-value arguments. For example, `ilwt2(ll,lh,hl,hh,LiftingScheme=lscheme,Level=3)` specifies the `lscheme` lifting scheme and the inverse transform up to level 3.

Examples

Inverse 2-D Lifting Wavelet Transform

Load and display the 128-by-128 `xbox` image.

```
load xbox
imagesc(xbox)
title("Original Image")
```



Obtain the 2-D LWT of the image using default settings. Preserve integer values.

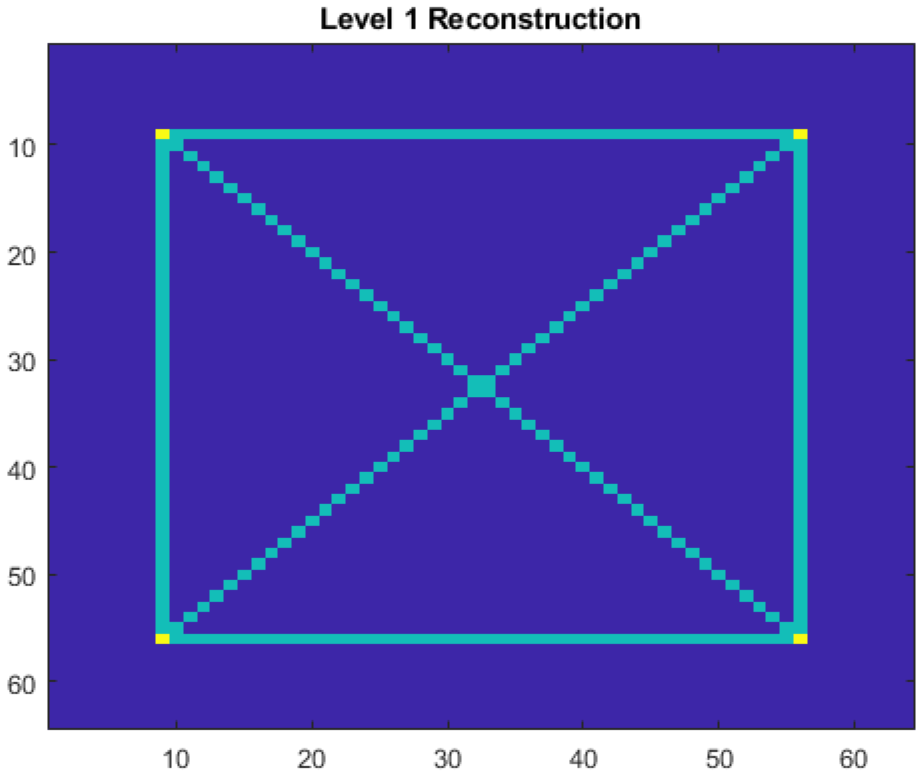
```
[ll,lh,hl,hh] = lwt2(xbox,Int2Int=true);
```

Obtain the inverse LWT up to level 1. Confirm the size of the reconstruction is 64-by-64.

```
xr = ilwt2(ll,lh,hl,hh,Level=1,Int2Int=true);
size(xr)
```

```
ans = 1×2

    64    64
```

```
imagesc(xr)
title("Level 1 Reconstruction")
```



Obtain the inverse LWT using default settings. Confirm perfect reconstruction.

```
xr = ilwt2(ll,lh,hl,hh,Int2Int=true);
max(abs(xr(:)-xbox(:)))
```

```
ans = 0
```

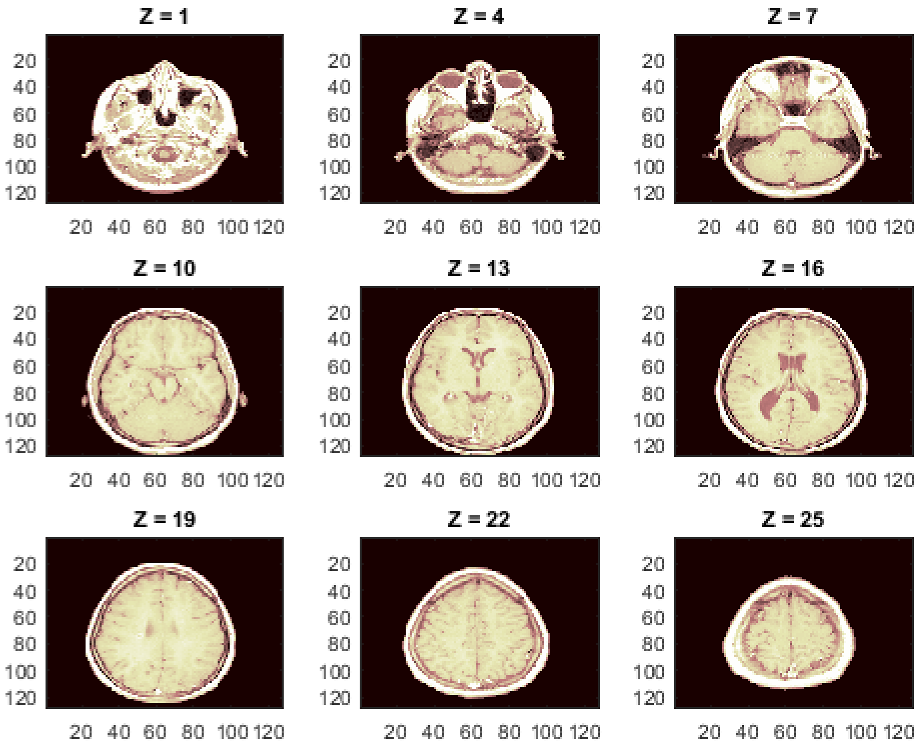
Inverse LWT of 2-D Multisignal to Specified Level

Load the 3-D `wmri` data set. The data consists of 27 128-by-128 magnetic resonance images (MRI) arranged in a 128-by-128-by-27 array.

```
load wmri
```

Display some of the slices along the Z-orientation of the original data set.

```
map = pink(90);
idxImages = 1:3:size(X,3);
figure("DefaultAxesXTick",[],"DefaultAxesYTick",[],...
    "DefaultAxesFontSize",8,"Color","w")
colormap(map)
for k = 1:9
    j = idxImages(k);
    subplot(3,3,k)
    image(X(:,:,j))
    str = sprintf("Z = %d",j);
    title(str)
end
```



By default, `lwt2` performs the wavelet decomposition along the rows and columns of the input data. Use `lwt2` to obtain the 2-D LWT of each 128-by-128 slice in the 3-D data set using the lifting scheme associated with the `bior3.5` wavelet. Preserve the integer-valued data.

```
lscheme = liftingScheme(Wavelet="bior3.5");
[ll,lh,hl,hh] = lwt2(X,LiftingScheme=lscheme,Int2Int=true);
```

Inspect the dimensions of a detail coefficients cell array. Confirm the coefficients at each level is a 3-D array, and the size of the third dimension is 27.

```
hh
```

```
hh=7×1 cell array
    {64x64x27 double}
    {32x32x27 double}
    {16x16x27 double}
    { 8x8x27  double}
    { 4x4x27  double}
    { 2x2x27  double}
    { 1x1x27  double}
```

Obtain the inverse 2-D LWT up to level 1. Confirm the size of the 3-D reconstruction is 64-by-64-by-27.

```
xr = ilwt2(ll,lh,hl,hh,LiftingScheme=lscheme,Int2Int=true,Level=1);
size(xr)
```

```
ans = 1×3

    64    64    27
```

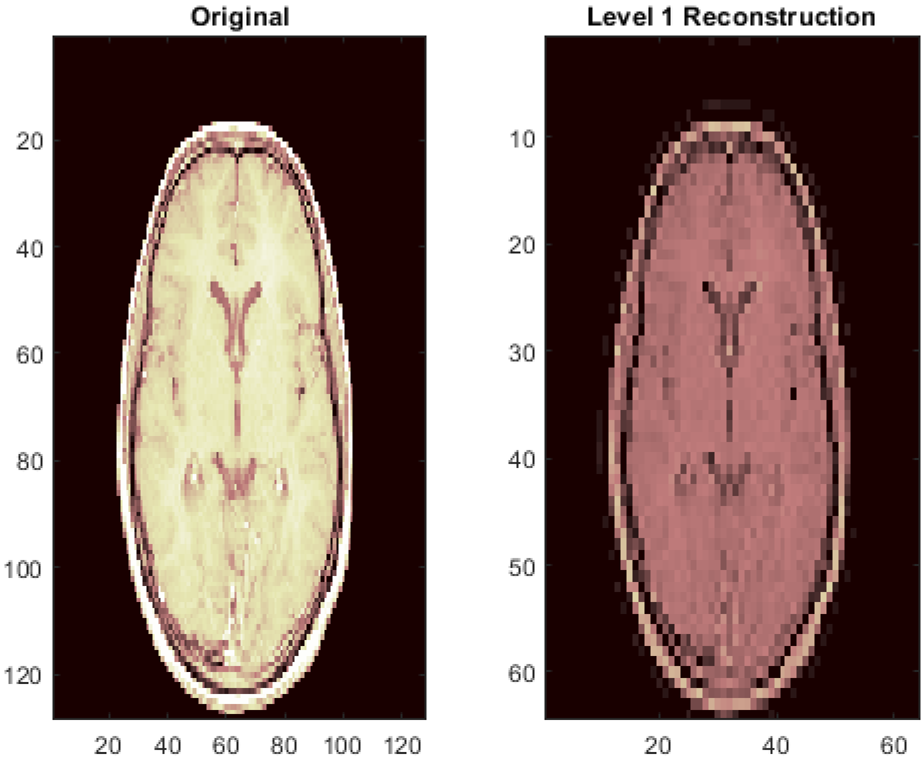
Choose any slice from the original data set, and perform the same LWT operations on that slice. Confirm the reconstruction is equal to the corresponding slice in the 3-D reconstruction array.

```
num = 13;
slice = X(:,:,num);
[lls,lhs,hls,hhs] = lwt2(slice,LiftingScheme=lscheme,Int2Int=true);
xrs = ilwt2(lls,lhs,hls,hhs,LiftingScheme=lscheme,Int2Int=true,Level=1);
max(max(abs(xrs-xr(:,:,num))))
```

```
ans = 0
```

Compare the reconstruction of the slice with the original version.

```
figure
colormap(map)
subplot(1,2,1)
image(X(:,:,num))
title("Original")
subplot(1,2,2)
image(xrs)
title("Level 1 Reconstruction")
```



Input Arguments

`ll` — Approximation coefficients

scalar | vector | matrix

Approximation coefficients at the coarsest scale, specified as a scalar, vector, or matrix. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`lh` — Horizontal detail coefficients

cell array

Horizontal detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `lh` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`hl` — Vertical detail coefficients

cell array

Vertical detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hl` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`hh` — Diagonal detail coefficients

cell array

Diagonal detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hh` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `xr = ilwt2(ll,lh,hl,hh,Wavelet="db2",Int2Int=true)`

`Wavelet` — Wavelet

`"db1"` (default) | character vector | string scalar

Orthogonal or biorthogonal wavelet to use in the inverse LWT, specified as a character vector or string scalar. See the Wavelet property of `liftingScheme` for the list of supported wavelets. For perfect reconstruction, you must specify the same wavelet that you used to obtain the coefficients `ll`, `lh`, `hl`, and `hh`.

You cannot specify `Wavelet` and `LiftingScheme` at the same time.

Example: `xr = ilwt2(ll,lh,hl,hh,Wavelet="bior3.5")` uses the `bior3.5` biorthogonal wavelet.

Data Types: `char` | `string`

`LiftingScheme` — Lifting scheme
`liftingScheme` object

Lifting scheme to use in the inverse LWT, specified as a `liftingScheme` object. For perfect reconstruction, you must specify the same lifting scheme that you used to obtain the coefficients `ll`, `lh`, `hl`, and `hh`.

You cannot specify `LiftingScheme` and `Wavelet` at the same time.

Example: `xr = ilwt2(ll,lh,hl,hh,LiftingScheme=lScheme)` uses the `lScheme` lifting scheme.

`Level` — Reconstruction level
`0` (default) | positive integer

Reconstruction level, specified as a nonnegative integer less than or equal to `length(hh)`-1. If you do not specify a level, the function sets the reconstruction level to `0` and `xr` is a perfect reconstruction of the signal.

Example: `xr = ilwt2(ll,lh,hl,hh,Level=2)` reconstructs the signal up to level 2.

Data Types: `double`

`Extension` — Extension mode
`"periodic"` (default) | `"zeropad"` | `"symmetric"`

Extension mode to use in the inverse LWT, specified as one of these:

- `"periodic"` — Periodized extension
- `"zeropad"` — Zero extension
- `"symmetric"` — Symmetric extension

This argument specifies how to extend the signal at the boundaries.

Example: `xr = ilwt2(ll,lh,hl,hh,Extension="zeropad")` specifies zero extension.

`Int2Int` — Handling integer-valued data
`false` or `0` (default) | `true` or `1`

Integer-valued data handling, specified as one of these:

- `1` (`true`) — Preserve integer-valued data
- `0` (`false`) — Do not preserve integer-valued data

Specify `Int2Int` only if all coefficients are integers.

Example: `xr = ilwt2(ll,lh,hl,hh,Int2Int=true)` preserves integer-valued data.

Output Arguments

`xr` — Inverse wavelet transform
matrix

Inverse wavelet transform, returned as a matrix. `xr` has the same dimensionality as the input used by the `lwt2` function to generate the approximation and details coefficients.

Compatibility Considerations

`ilwt2` input syntax has changed

Behavior changed in R2021b

The `ilwt2` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`X = ilwt2(CA,CH,CV,CD,W)`	Errors	`X = ilwt2(CA,CH,CV,CD,Wavelet=W)`	You can also set the `LiftingScheme` name-value argument to obtain the inverse LWT.
`X = ilwt2(CA,CH,CV,CD,W,LEVEL)`	Errors	`X = ilwt2(CA,CH,CV,CD,Wavelet=W,Level=LEVEL)`	You can also set the `Extension` and `Int2Int` name-value arguments.
`X = ilwt2(AD_In_Place,W)`	Errors	NA	In-place transforms are no longer supported.

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

[2] Sweldens, Wim. “The Lifting Scheme: A Construction of Second Generation Wavelets.” *SIAM Journal on Mathematical Analysis* 29, no. 2 (March 1998): 511–46. https://doi.org/10.1137/S0036141095289051.

Extended Capabilities

C/C++ Code Generation

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

See Also

`lwt2` | `lwtcoef2` | `haart2` | `ihaart2` | `liftingScheme`

Introduced in R2021b

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.

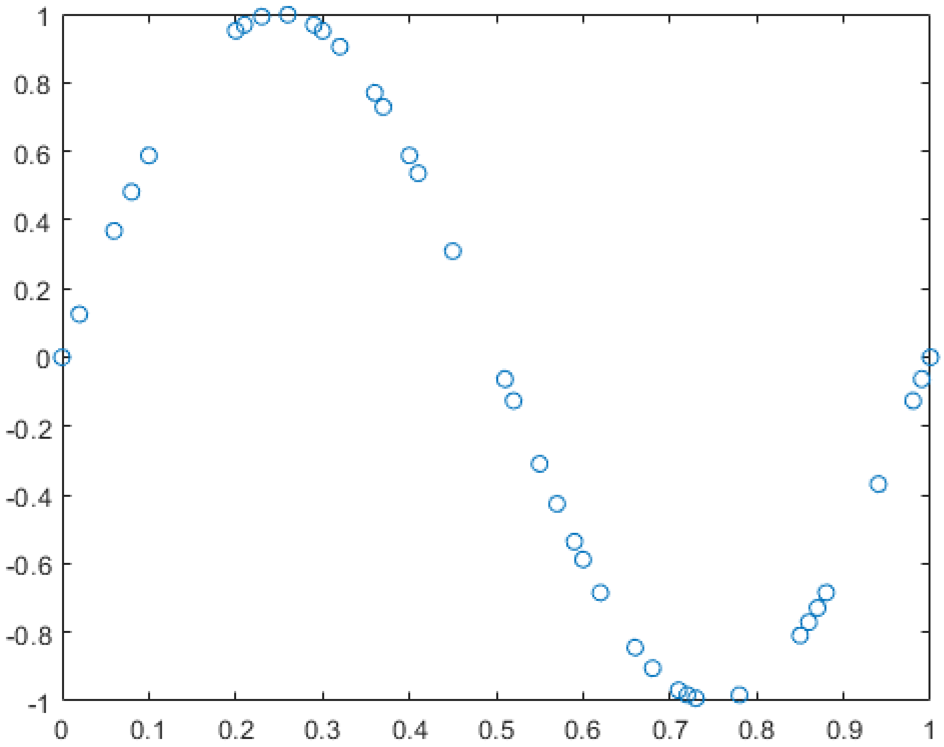
Examples

Multiscale Local 1-D Polynomial Transform and Inverse Transform

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

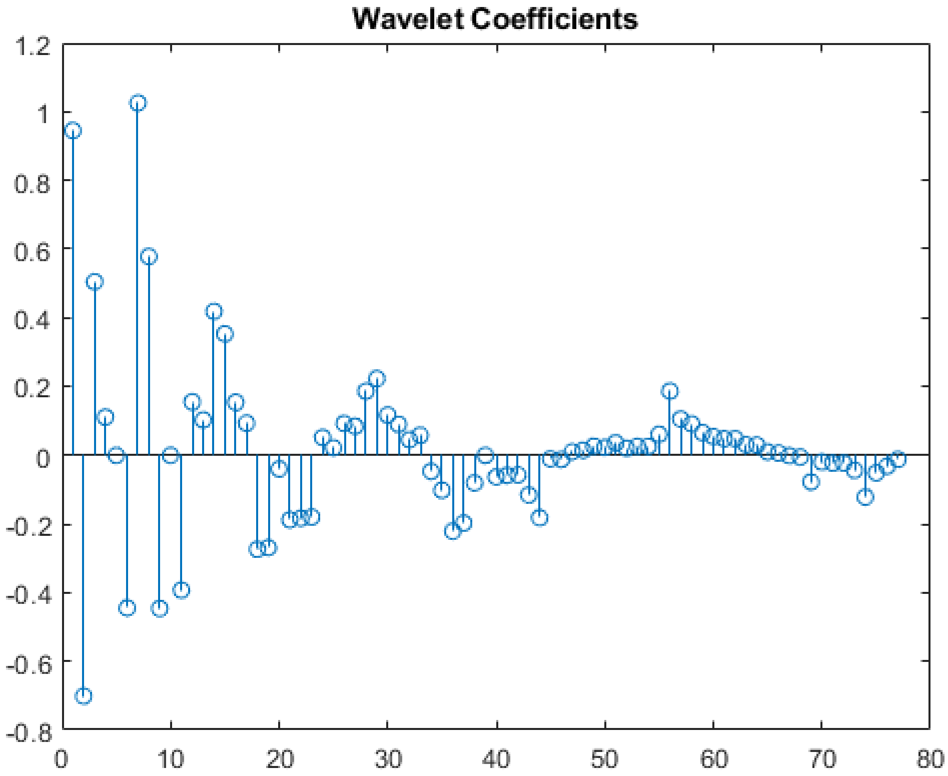
Create and plot a sine wave with non-uniform sampling.

```
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
```



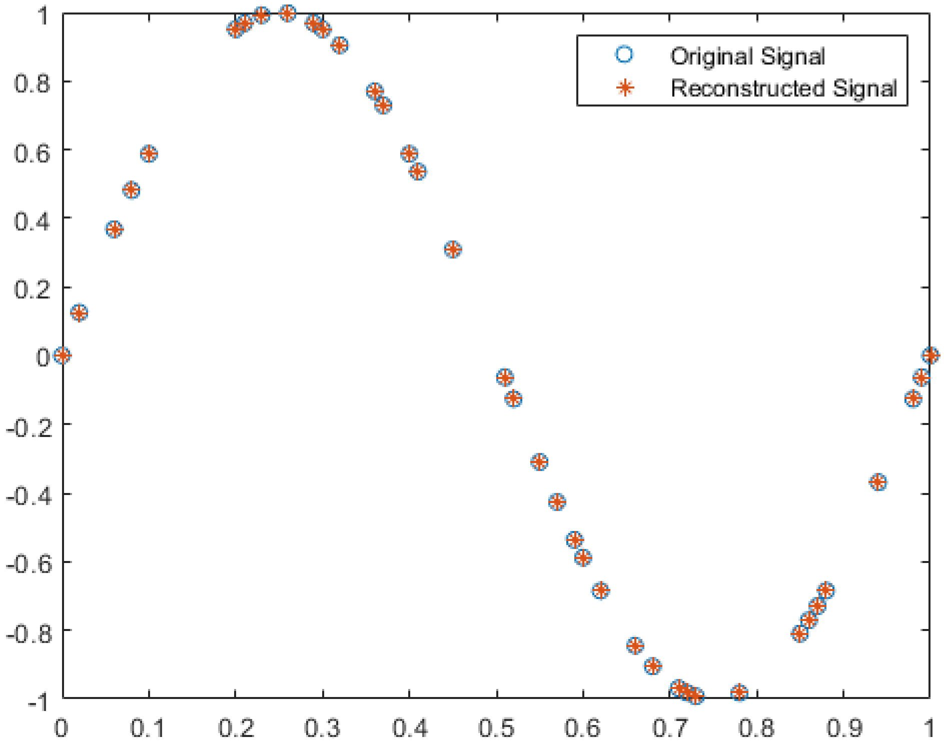
Perform the multiscale local 1-D polynomial transform (`mlpt`) on the signal. Visualize the coefficients.

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

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



Perform the inverse multiscale local 1-D polynomial transform (`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))
```

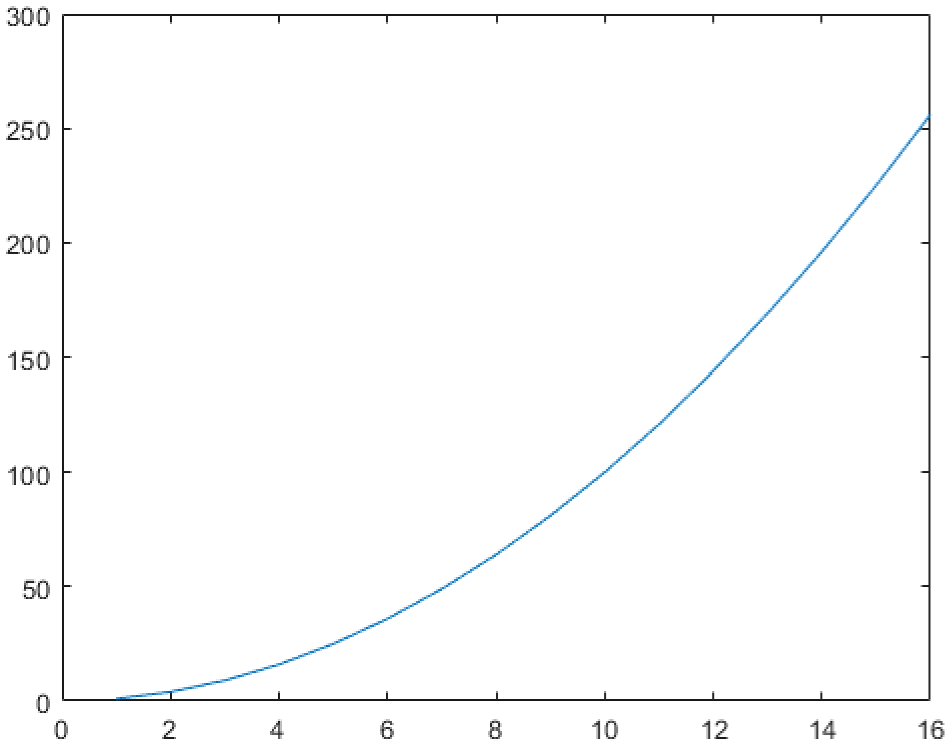
```
reconstructionError = 1.7552e-15
```

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:
```

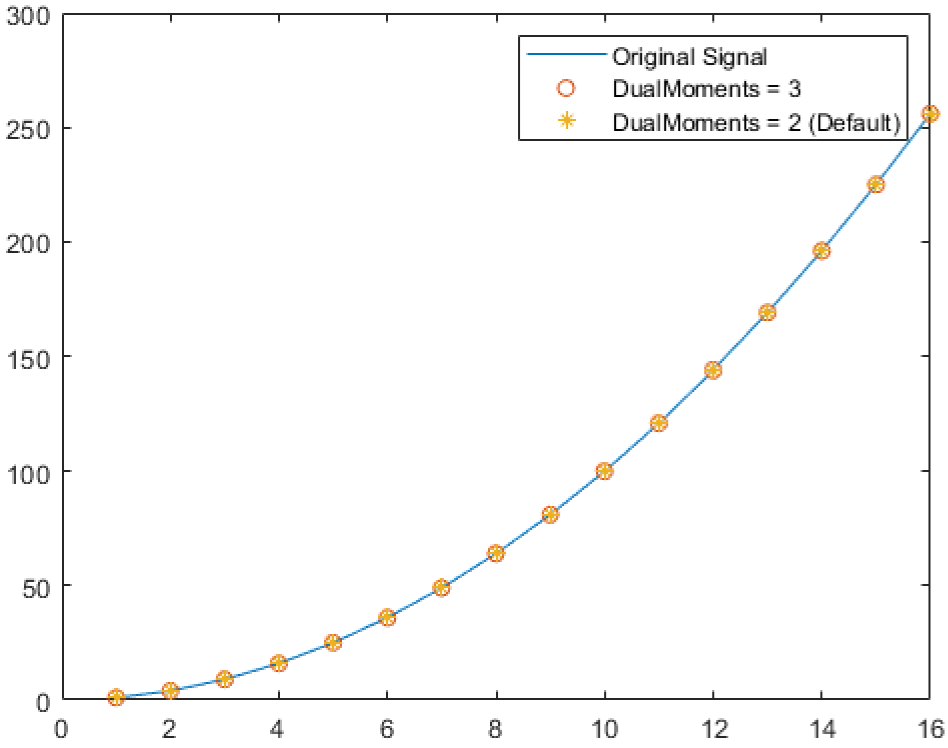
```
fprintf('  - Nondefault dual moments: %0.2f\n',mean(abs(y3-x)));
```

```
  - Nondefault dual moments: 0.00
```

```
fprintf('  - Default dual moments: %0.2f\n\n',mean(abs(y2-x)));
```

```
  - Default dual moments: 0.00
```

```
hold off
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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, Maarten. “Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data.” *IEEE Transactions on Signal Processing* 61, no. 3 (February 2013): 545–55. https://doi.org/10.1109/TSP.2012.2225059.

[2] Jansen, Maarten, and Mohamed Amghar. “Multiscale Local Polynomial Decompositions Using Bandwidths as Scales.” *Statistics and Computing* 27, no. 5 (September 2017): 1383–99. https://doi.org/10.1007/s11222-016-9692-8.

[3] Jansen, Maarten, and Patrick Oonincx. *Second Generation Wavelets and Applications*. London ; New York: 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 Fejér-Korovkin (`'fk18'`) wavelet.

`xrec = imodwpt(coefs,wname)` returns the inverse MODWPT using the orthogonal filter specified by `wname`. This filter must be the same filter used in `modwpt`.

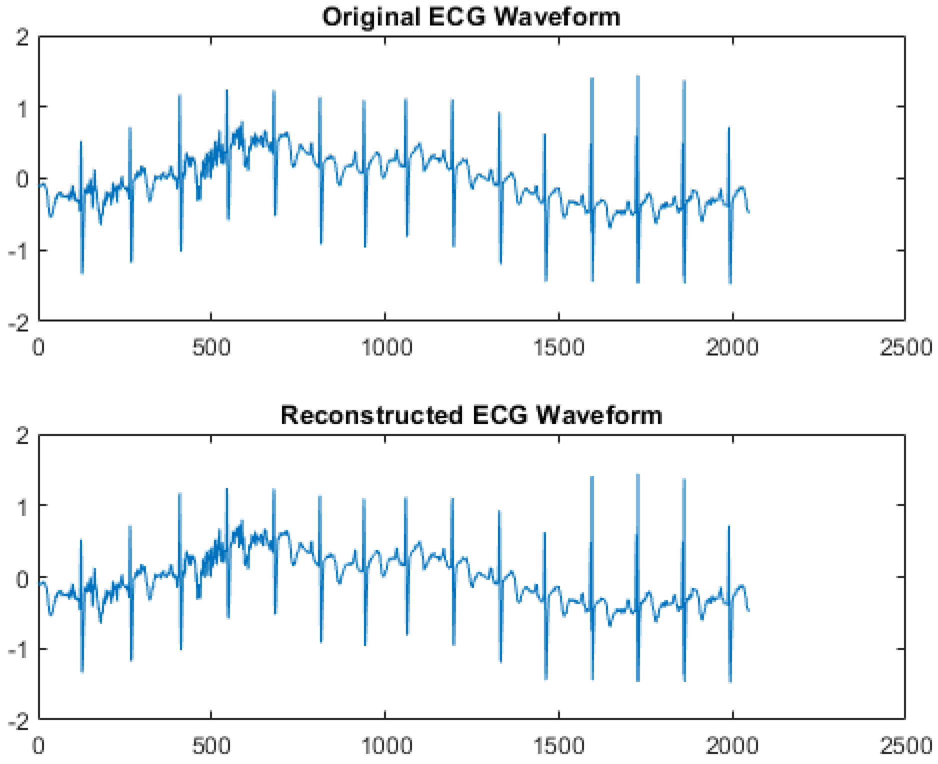
`xrec = imodwpt(coefs,lo,hi)` returns the inverse MODWPT using the orthogonal scaling filter, `lo`, and wavelet filter, `hi`.

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

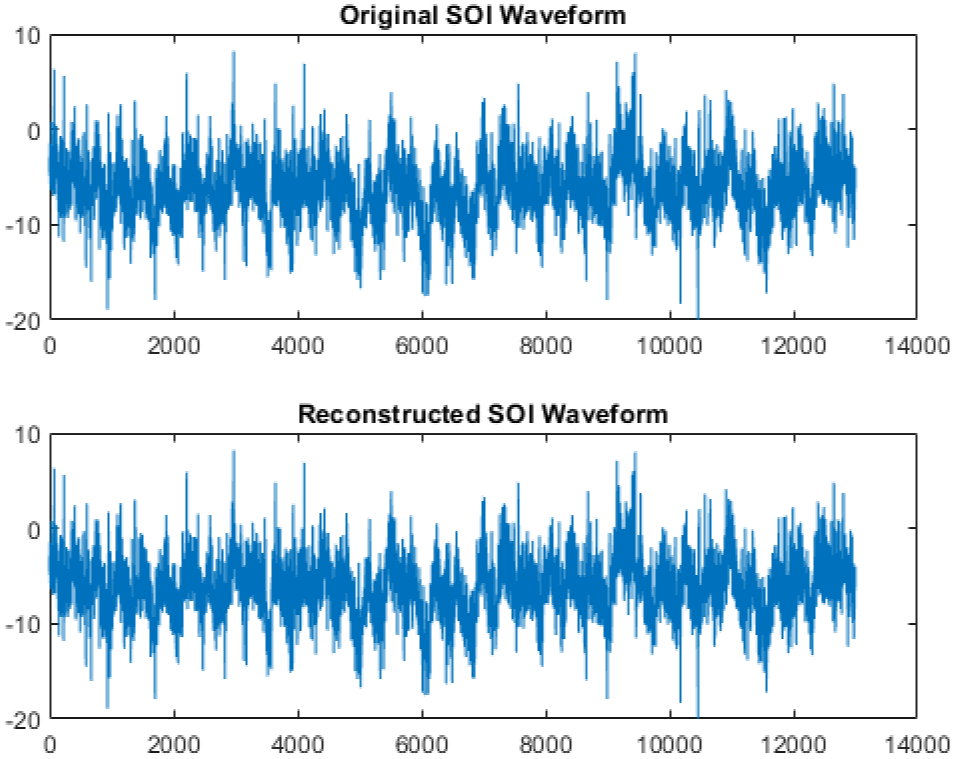
Obtain the MODWPT of Southern Oscillation Index data using specified scaling and wavelets filters with the Daubechies extremal phase wavelet with two vanishing moments (`'db2'`).

```
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')
```



Original SOI Waveform

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 | string scalar

Synthesizing wavelet filter used to invert the MODWPT, specified as a character vector or string scalar. 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™.

Usage notes and limitations:

- The input `wname` must be constant.

See Also

`modwpt` | `modwptdetails` | `dwpt`

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)` reconstructs the signal based on the maximal overlap discrete wavelet transform (MODWT) coefficients in `w`. By default, `imodwt` assumes that you obtained `w` using the `'sym4'` wavelet with periodic boundary handling. If you do not modify the coefficients, `xrec` is a perfect reconstruction of the signal.

`xrec = imodwt(w,wname)` reconstructs the signal using the orthogonal wavelet `wname`. `wname` must be the same wavelet used to analyze the signal input to `modwt`.

`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`.

`xrec = imodwt( ___ ,lev)` reconstructs the signal up to level `lev`. `xrec` is a projection onto the scaling space at level `lev`. The default level is 0, which results in perfect reconstruction if you do not modify the coefficients.

`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.

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 `x`.

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.

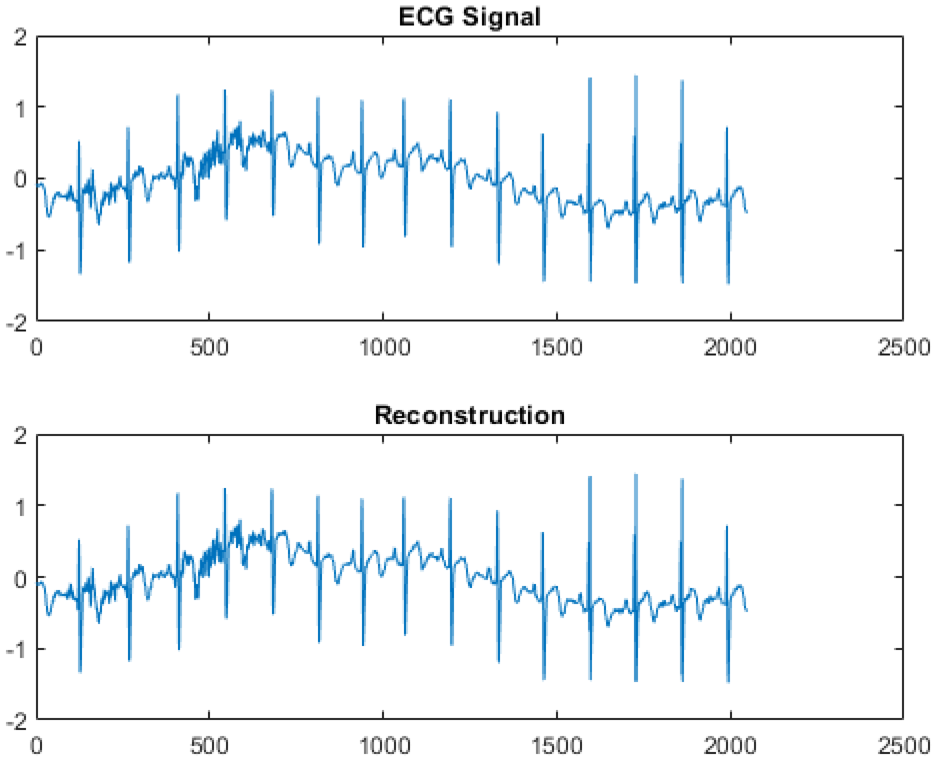
```
[Lo,Hi] = wfilters('fk8');
```

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.

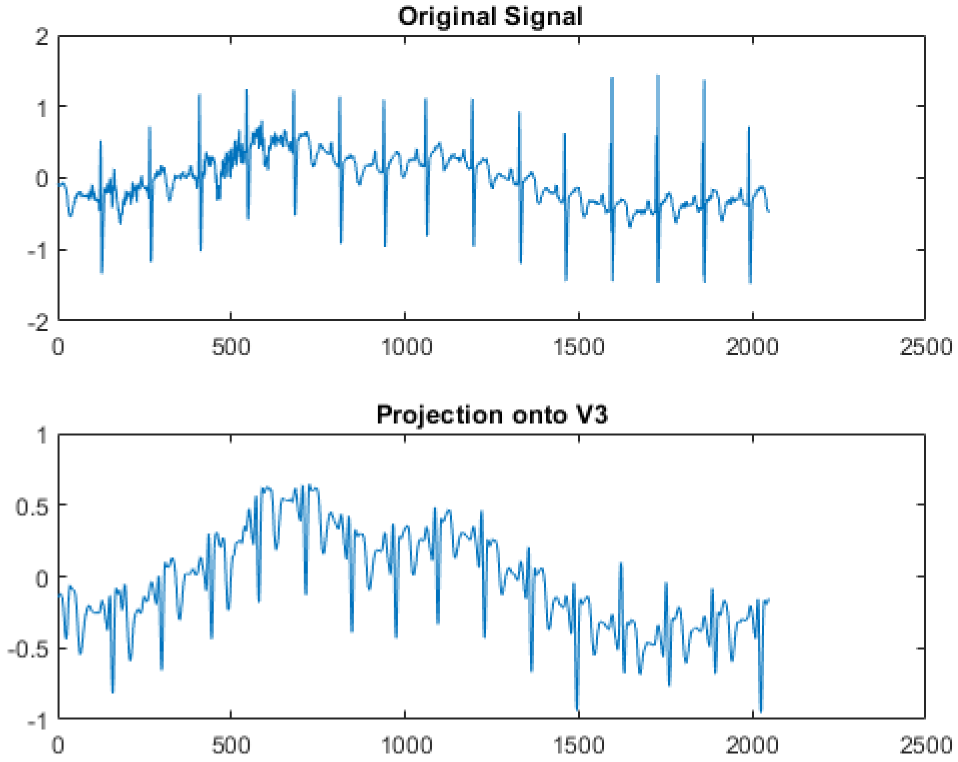
```
wtecg = modwt(wecg);
```

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.

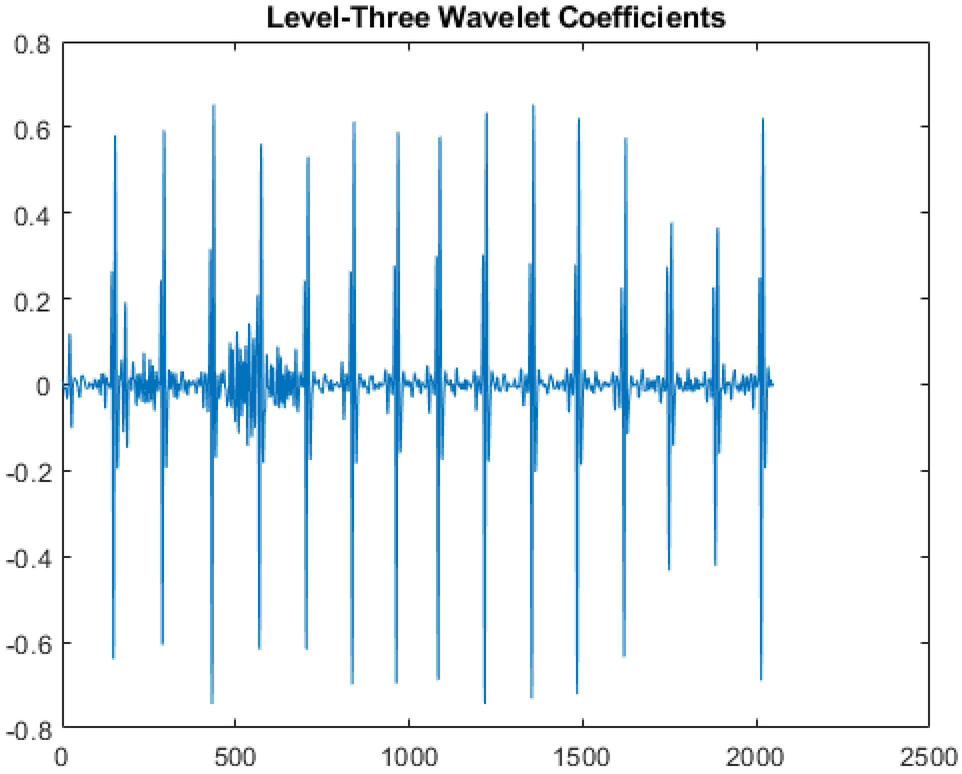
```
subplot(2,1,1)
plot(wecg)
title('Original Signal')
subplot(2,1,2)
plot(v3proj)
title('Projection onto V3')
```



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 wavelet 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
```

Inverse MODWT of Multisignal

Load the 23 channel EEG data `Espiga3` [2]. The channels are arranged column-wise. The data is sampled at 200 Hz.

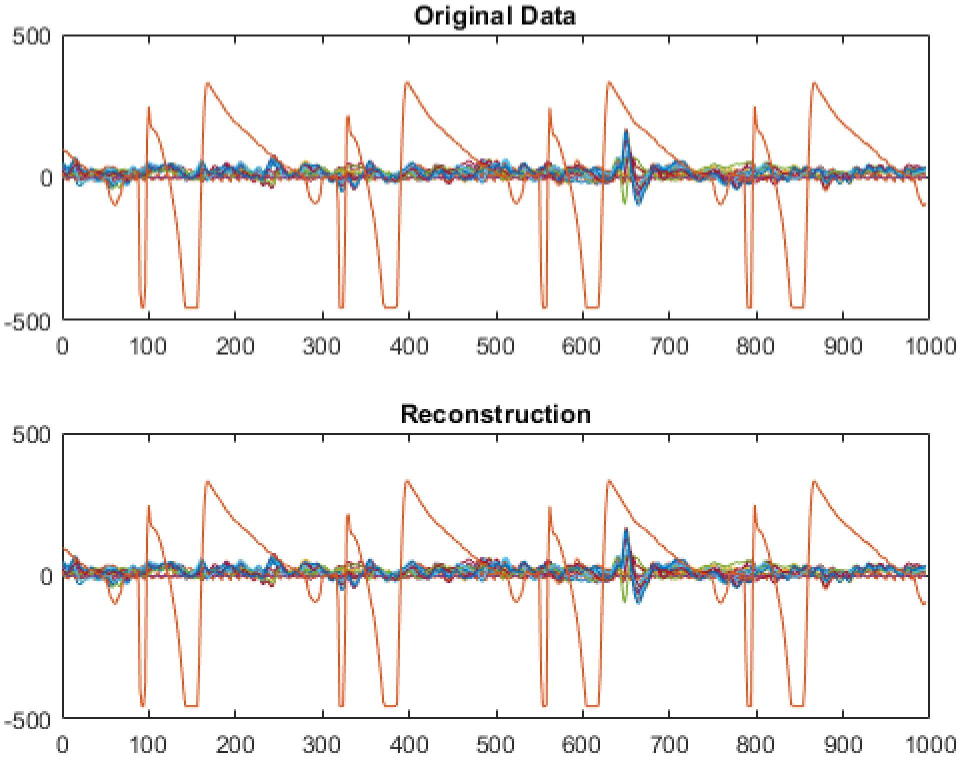
```
load Espiga3
```

Obtain the maximal overlap discrete wavelet transform down to the maximum level.

```
w = modwt(Espiga3);
```

Reconstruct the multichannel signal. Plot the original data and reconstruction.

```
xrec = imodwt(w);
subplot(2,1,1)
plot(Espiga3)
title('Original Data')
subplot(2,1,2)
plot(xrec)
title('Reconstruction')
```



Input Arguments

w — MODWT transform
matrix | 3-D array

MODWT transform of a signal or multisignal down to level *L*, specified as a matrix or 3-D array, respectively. `w` is an *L*+1-by-*N* matrix for the MODWT of an *N*-point signal, and an *L*+1-by-*N*-by-*NC* array for the MODWT of an *N*-by-*NC* multisignal. By default, `imodwt` assumes that you obtained the MODWT using the `'sym4'` wavelet with periodic boundary handling.

Data Types: `single` | `double`

`wname` — Synthesis wavelet
`'sym4'` (default) | `'db`*N*`'` | `'coif`*N*`'` | `'haar'` | `'fk`*N*`'` | `'sym`*N*`'`

Synthesis wavelet, specified as one of the following:

- `'haar'` — Haar wavelet
- `'db`*N*`'` — Extremal phase Daubechies wavelet with `N` vanishing moments, where `N` is a positive integer from 1 to 45.
- `'sym`*N*`'` — Symlets wavelet with `N` vanishing moments, where `N` is a positive integer from 2 to 45.
- `'coif`*N*`'` — Coiflets wavelet with `N` vanishing moments, where `N` is a positive integer from 1 to 5.
- `'fk`*N*`'` — Fejér-Korovkin wavelet with `N` coefficients, where `N = 4, 6, 8, 14, 18` and `22`.

The synthesis wavelet must be the same wavelet used in the analysis with `modwt`.

`Lo,Hi` — Filters
even-length real-valued vectors

Filters, specified as a pair of even-length real-valued vectors. `Lo` is the scaling filter, and `Hi` is the wavelet filter. `Lo` and `Hi` must be the same filters used in the analysis with `modwt`. The filters must satisfy the conditions for an orthogonal wavelet. The lengths of `Lo` and `Hi` must be equal. See `wfilters` for additional information. You cannot specify both a wavelet `wname` and filter pair `Lo,Hi`.

`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
vector | matrix

Reconstructed version of the original signal or multisignal based on the MODWT and the level of reconstruction, returned as a vector or matrix.

References

[1] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[2] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés,

3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps
Signal Multiresolution Analyzer

Functions
`modwt` | `modwtmra`

Topics
“Practical Introduction to Multiresolution Analysis”
“Time-Frequency Gallery”
“Wavelet Analysis of Financial Data”

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

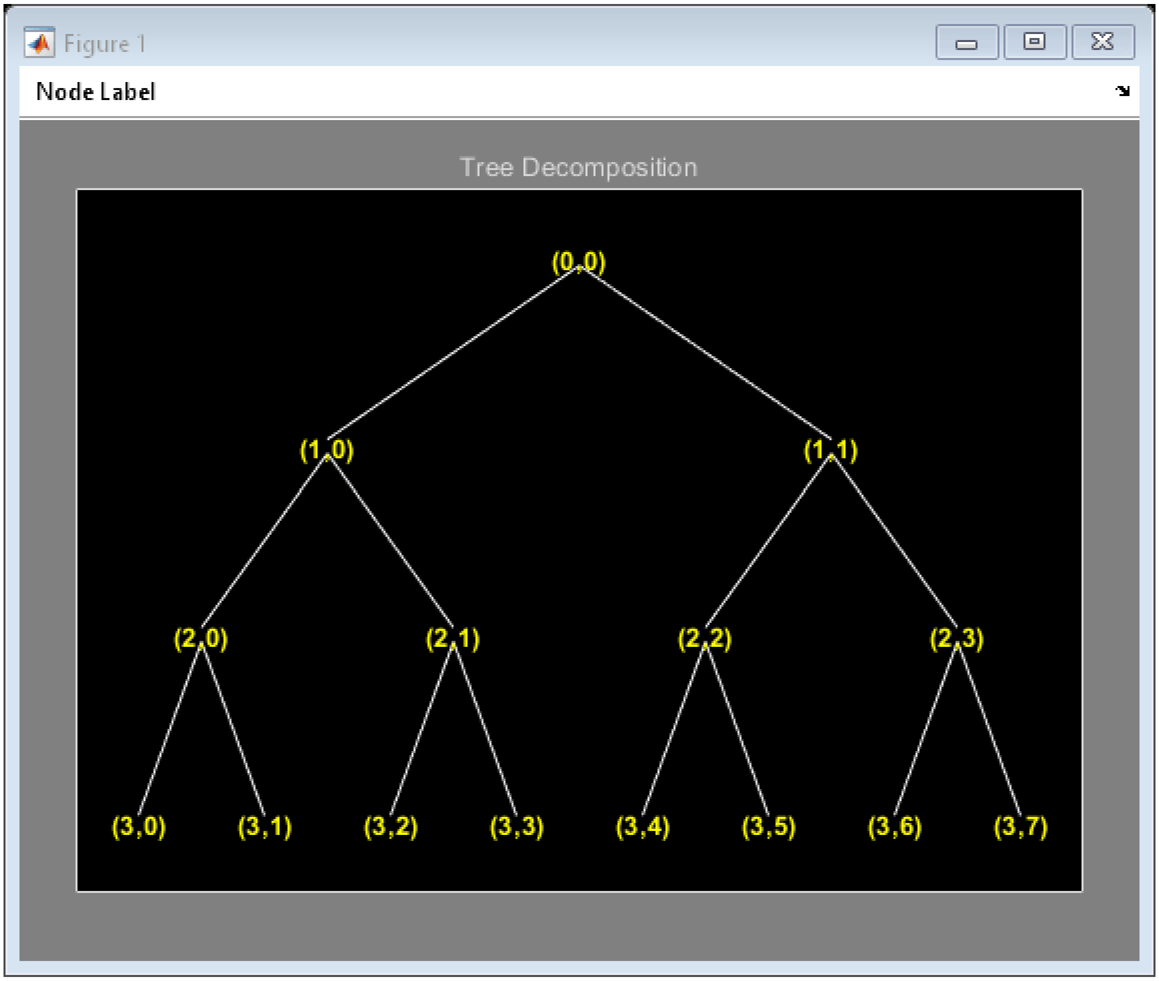
Depth and Position in Wavelet Packet Tree

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);
```

Convert the indices to depth-position format.

```
[depth,pos] = ind2depo(Ord,idx);
table(depth,pos)
```

```
ans=15×2 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 −∞ 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.

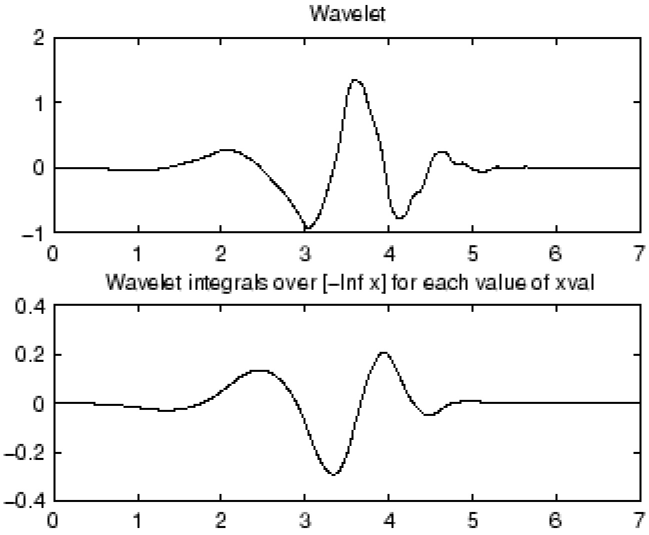
When `IN2` is equal to the special value 0, `intwave('wname',0)` is equivalent to `intwave('wname',8,IN3)`.

`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

inverse

Laurent matrix inverse

Syntax

```
R = inverse(M)
```

Description

`R = inverse(M)` returns the inverse of the Laurent matrix `M` if `M` has a nonzero monomial determinant.

Examples

Laurent Matrix Inverse

Create the Laurent polynomials:

- $a(z) = z + 1$
- $b(z) = z^2 + z + z^{-1}$
- $c(z) = z$
- $d(z) = z^2 + z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[1 1],MaxOrder=1);
lpB = laurentPolynomial(Coefficients=[1 1 0 1],MaxOrder=2);
lpC = laurentPolynomial(Coefficients=[1],MaxOrder=1);
lpD = laurentPolynomial(Coefficients=[1 0 0 1],MaxOrder=2);
```

Create the matrix `lmat` = $\begin{bmatrix} a(z) & b(z) \\ c(z) & d(z) \end{bmatrix}$. Obtain the determinant of `lmat`.

```
lmat = laurentMatrix(Elements={lpA,lpB;lpC,lpD});
det(lmat)
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: -1
```

The determinant is a nonzero monomial. Obtain the inverse of `lmat`. Inspect the elements of the inverse.

```
lmatinv = inverse(lmat);
lmatinv.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:
    Coefficients: [1 0 0 1]
        MaxOrder: 3
```

```
lmatinv.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [-1 -1 0 -1]
        MaxOrder: 3
```

```
lmatinv.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: -1
        MaxOrder: 2
```

```
lmatinv.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 1]
        MaxOrder: 2
```

Confirm the product of `lmat` and its inverse is equal to the identity matrix.

```
matprod = lmat*lmatinv;
matprod.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

```
matprod.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 0
        MaxOrder: 0
```

```
matprod.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 0
```

```
        MaxOrder: 0
```

```
matprod.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

Input Arguments

M — Laurent matrix
laurentMatrix object

Laurent matrix, specified as a laurentMatrix object.

Output Arguments

R — Inverse
laurentMatrix object

Inverse of a Laurent matrix, returned as a laurentMatrix object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
det

Objects
laurentMatrix | laurentPolynomial

Introduced in R2021b

isBiorthogonal

Determine if DWT filter bank is biorthogonal

Syntax

```
tf = isBiorthogonal(fb)
tf = isBiorthogonal(fb,tol)
```

Description

`tf = isBiorthogonal(fb)` returns `true` if the discrete wavelet transform (DWT) filter bank `fb` is a biorthogonal filter bank and `false` otherwise.

To determine if a DWT filter bank is orthogonal, use `isOrthogonal`.

`tf = isBiorthogonal(fb,tol)` uses the positive real-valued tolerance `tol` to determine the biorthogonality of the filter bank `fb`. `tol` is a small positive number in the interval (0, 10^{-2}]. If unspecified, `tol` defaults to 10^{-5}.

Examples

Biorthogonality Test of DWT Filter Bank

Check whether a filter bank is biorthogonal.

```
fb = dwtfilterbank('Wavelet','bior4.4');
isBiorthogonal(fb)
```

```
ans = logical
   1
```

Input Arguments

fb — Discrete wavelet transform filter bank
`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

tol — Tolerance
10^{-5} (default) | positive scalar

Tolerance to use to determine biorthogonality of the filter bank, specified as a positive scalar in the interval (`0`,10^{-2}]. The sum of both scaling filters must be within `tol` of √2 and the sum of both wavelet filters must be less than `tol`.

See Also

`dwtfilterbank` | `isOrthogonal`

Introduced in R2018a

isheart2

Inverse shearlet transform

Syntax

```
imrec = isheart2(sls,cfs)
```

Description

`imrec = isheart2(sls,cfs)` returns the inverse shearlet transform or shearlet synthesis based on the shearlet system `sls` and the shearlet transform coefficients `cfs`. The `isheart2` function assumes `sls` is the same shearlet system used to obtain the transform coefficients `cfs`.

Examples

Perfect Reconstruction of Shearlet Transform

Load an image and create a shearlet system that can be applied to the image.

```
load shapes
[numRows,numCols] = size(shapes);
sls = shearletSystem('ImageSize',[numRows numCols],'NumScales',4)
```

```
sls = 
  shearletSystem with properties:

         ImageSize: [512 512]
         NumScales: 4
    PreserveEnergy: 0
     TransformType: 'real'
    FilterBoundary: 'periodic'
         Precision: 'double'
```

Obtain the shearlet coefficients of the image.

```
cfs = sheart2(sls,shapes);
```

Take the inverse transform of the coefficients. Check for perfect reconstruction.

```
imrec = isheart2(sls,cfs);
norm(imrec-shapes,'fro')
```

```
ans = 8.2562e-14
```

Input Arguments

`sls` — Shearlet system
`shearletSystem` object

Shearlet system, specified as a `shearletSystem` object.

`cfs` — Shearlet transform coefficients
3-D array

Shearlet transform coefficients, specified as a real- or complex-valued 3-D array. The 3-D array `cfs` is an *M*-by-*N*-by-*K* matrix where *M* and *N* are equal to the row and column dimensions of the original image. The size of the third dimension, *K*, is equal to the number of shearlets including the lowpass filter, *K* = `numshears(sls)` + 1.

The `isheart2` function assumes `sls` is the same shearlet system used to obtain the transform coefficients `cfs`.

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

Output Arguments

`imrec` — Inverse shearlet transform
real-valued matrix

Inverse shearlet transform or shearlet synthesis, based on the shearlet system `sls` and the shearlet transform coefficients `cfs`. The size of `imrec` is equal to the size of the original image. The data type of `imrec` matches the Precision value of the shearlet system.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`sheart2` | `shearletSystem`

Topics
“Shearlet Systems”

Introduced in R2019b

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.

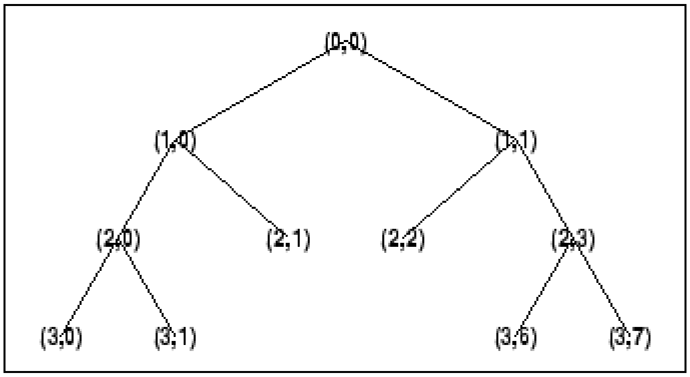
N can be a column vector containing the indices of nodes or a matrix, that contains the depths and positions of nodes.

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.

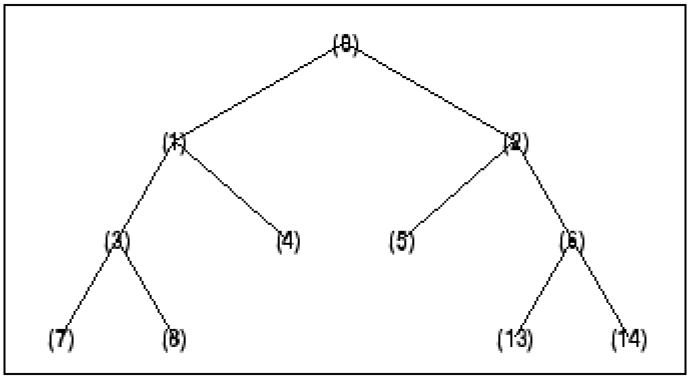
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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

isOrthogonal

Determine if DWT filter bank is orthogonal

Syntax

```
tf = isOrthogonal(fb)
tf = isOrthogonal(fb,tol)
```

Description

`tf = isOrthogonal(fb)` returns `true` if the discrete wavelet transform (DWT) filter bank `fb` is an orthogonal filter bank and `false` otherwise.

To determine if a DWT filter bank is biorthogonal, use `isBiorthogonal`.

`tf = isOrthogonal(fb,tol)` uses the positive real-valued tolerance `tol` to determine the orthogonality of the filter bank `fb`. `tol` is a small positive number in the interval (0,10^{-2}]. If unspecified, `tol` defaults to 10^{-5}.

Examples

Orthogonality Test of DWT Filter Bank

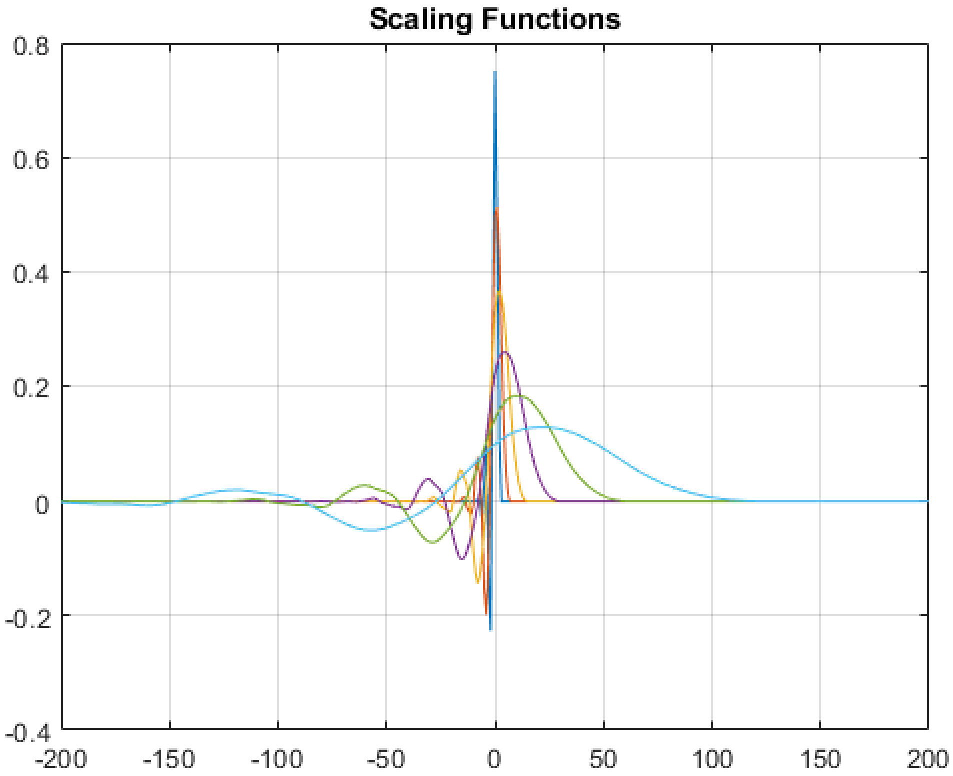
Create a DWT filter bank using the Daubechies `db6` wavelet. Confirm the filter bank is orthogonal.

```
fb = dwtfilterbank('Wavelet','db6');
isOrthogonal(fb)
```

```
ans = logical
   1
```

Plot the time-domain and centered scaling functions for each level in the filter bank.

```
[phi,t] = scalingfunctions(fb);
psi = wavelets(fb);
plot(t,phi')
grid on
xlim([-200 200])
title('Scaling Functions')
```



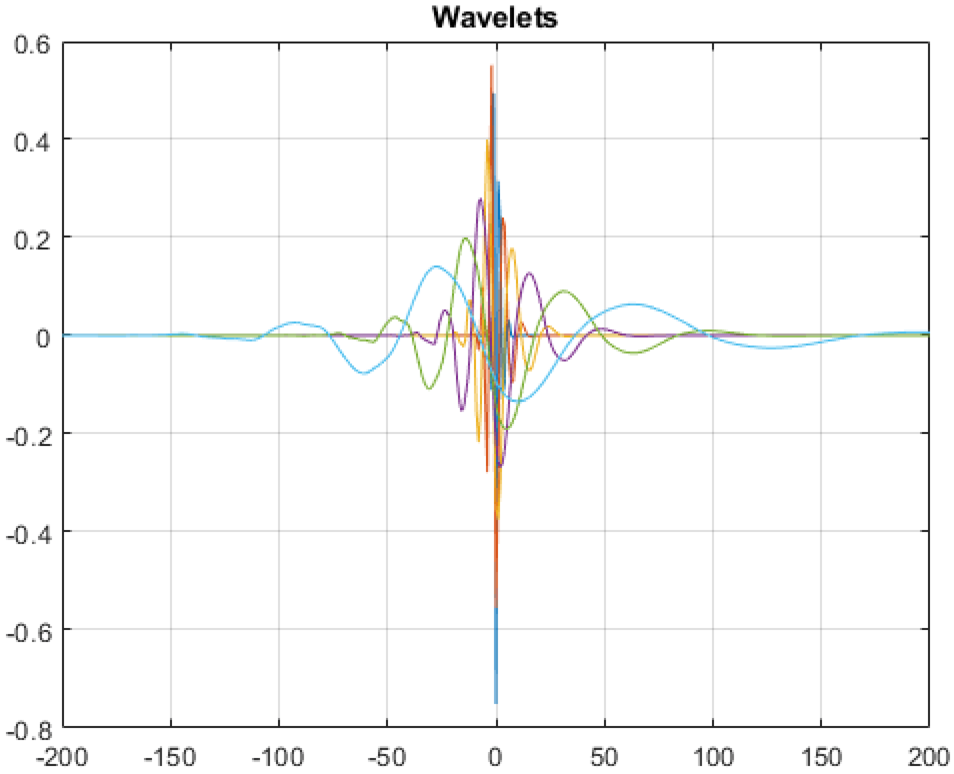
Confirm the scaling functions have norm square equal to 1.

```
sum(phi.^2,2)
```

```
ans = 6×1

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Plot the time-domain and centered wavelets corresponding to the wavelet passband filters.

```
plot(t,psi')
grid on
xlim([-200 200])
title('Wavelets')
```



Confirm the wavelets have norm square equal to 1.

```
sum(psi.^2,2)
```

```
ans = 6×1

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Input Arguments

fb — Discrete wavelet transform filter bank
dwtfilterbank object

Discrete wavelet transform (DWT) filter bank, specified as a dwtfilterbank object.

tol — Tolerance
10^{-5} (default) | positive scalar

Tolerance to use to determine orthogonality of the filter bank, specified as a positive scalar in the interval $(0,10^{-2}]$.

See Also

`dwtfilterbank` | `isBiorthogonal`

Introduced in R2018a

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.

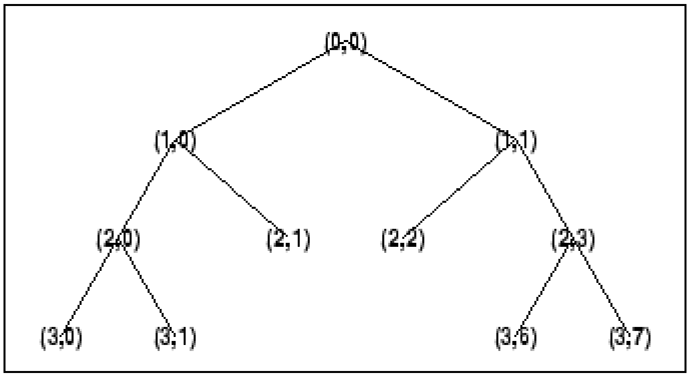
N can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes.

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.

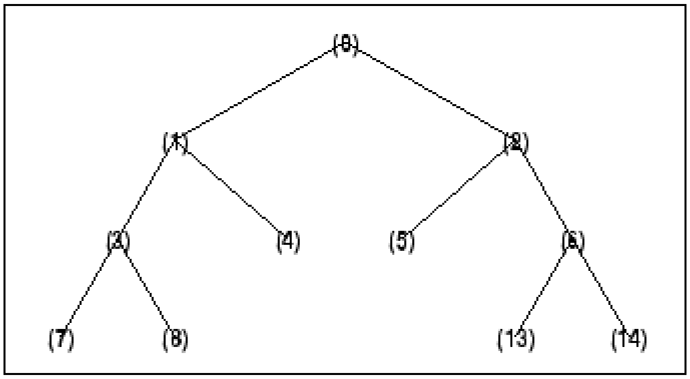
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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(swc,LoR,HiR)
x = iswt(swa,swd,LoR,HiR)
```

Description

`x = iswt(swc,wname)` reconstructs the 1-D signal `x` based on the multilevel stationary wavelet decomposition `swc` using the wavelet specified by `wname`. `swc` is expected to be the output of the `swt` function. The `wname` wavelet must be the same wavelet used to obtain the `swc` structure.

`x = iswt(swa,swd,wname)` uses the approximation coefficients `swa` and detail coefficients `swd` to reconstruct the 1-D signal. The real-valued matrices `swa` and `swd` are expected to be the outputs of the `swt` function.

The syntax `iswt(swa(end,:),swd,wname)` is equivalent to `iswt(swa,swd,wname)`.

`x = iswt(swc,LoR,HiR)` uses the scaling filter `LoR` and wavelet filter `HiR`. The filters are expected to be the reconstruction filters associated with the wavelet used to create the `swc` structure. For more information, see `wfilters`.

`x = iswt(swa,swd,LoR,HiR)` uses the scaling filter `LoR` and wavelet filter `HiR`. The filters are expected to be the reconstruction filters associated with the wavelet used to create the `swc` structure. For more information, see `wfilters`.

The syntax `iswt(swa(end,:),swd,LoR,HiR)` is equivalent to `iswt(swa,swd,LoR,HiR)`.

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
```

Input Arguments

swc — Multilevel stationary wavelet decomposition

real-valued matrix

Multilevel stationary wavelet decomposition, specified as a real-valued matrix. swc is the output of swt.

Data Types: double

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. iswt supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See wfilters for a list of orthogonal and biorthogonal wavelets.

swa — Approximation coefficients

real-valued matrix

Approximation coefficients, specified as a real-valued matrix. swa is the output of swt.

Data Types: double

swd — Detail coefficients

real-valued matrix

Detail coefficients, specified as a real-valued matrix. swd is the output of swt.

Data Types: double

LoR,HiR — Wavelet reconstruction filters

even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. LoR is the scaling (lowpass) reconstruction filter, and HiR is the wavelet (highpass) reconstruction filter. The lengths of LoR and HiR must be equal. See wfilters for additional information.

Data Types: double

References

[1] Nason, G. P., and B. W. Silverman. “The Stationary Wavelet Transform and Some Statistical Applications.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:281–99. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_17.

[2] Coifman, R. R., and D. L. Donoho. “Translation-Invariant De-Noising.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:125–50. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_9.

[3] Pesquet, J.-C., H. Krim, and H. Carfantan. “Time-Invariant Orthonormal Wavelet Representations.” *IEEE Transactions on Signal Processing* 44, no. 8 (August 1996): 1964–70. https://doi.org/10.1109/78.533717.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

See Also

`idwt` | `swt` | `waverec`

Introduced before R2006a

iswt2

Inverse discrete stationary 2-D wavelet transform

Syntax

```
X = iswt2(swc,wname)
X = iswt2(swc,LoR,HiR)

X = iswt2(A,H,V,D,wname)
X = iswt2(A,H,V,D,LoR,HiR)
```

Description

`X = iswt2(swc,wname)` returns the inverse discrete stationary 2-D wavelet transform of the wavelet decomposition `swc` using the wavelet `wname`. The decomposition `swc` is the output of `swt2`.

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.

`X = iswt2(swc,LoR,HiR)` uses the specified lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

`X = iswt2(A,H,V,D,wname)` uses the approximation coefficients array `A` and detail coefficient arrays `H`, `V`, and `D`. The arrays `H`, `V`, and `D` contain the horizontal, vertical, and diagonal detail coefficients, respectively. The arrays are the output of `swt2`.

- If the decomposition `swc` or the coefficient arrays `A`, `H`, `V`, and `D` were generated from a multilevel decomposition of a 2-D matrix, the syntax `X = iswt2(A(:,:,end),H,V,D,wname)` reconstructs the 2-D matrix.
- If the decomposition `swc` or the coefficient arrays `A`, `H`, `V`, and `D` were generated from a single-level decomposition of a 3-D array, the syntax `X = iswt2(A(:,:,1,:),H,V,D,wname)` reconstructs the 3-D array.

`X = iswt2(A,H,V,D,LoR,HiR)` uses the lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

- If the decomposition `swc` or the coefficient arrays `A`, `H`, `V`, and `D` were generated from a multilevel decomposition of a 2-D matrix, the syntax `X = iswt2(A(:,:,end),H,V,D,LoR,HiR)` reconstructs the 2-D matrix.
- If the decomposition `swc` or the coefficient arrays `A`, `H`, `V`, and `D` were generated from a single-level decomposition of a 3-D array, the syntax `X = iswt2(A(:,:,1,:),H,V,D,LoR,HiR)` reconstructs the 3-D array.

Examples

Multilevel 2-D Stationary Wavelet Reconstruction

Show 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)
```

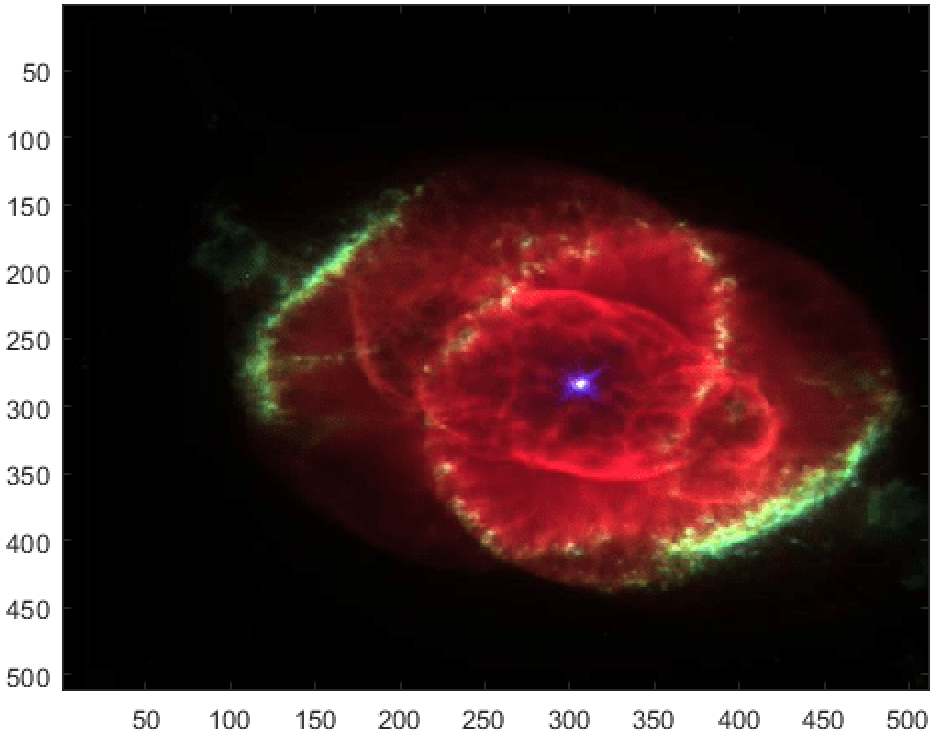
```
ans = 1.0126e-08
```

Inverse Stationary Wavelet Transform of RGB Image

This example shows how to reconstruct an RGB image from a multilevel stationary wavelet decomposition using approximation and detail coefficient arrays.

Load an RGB image. An RGB image is also referred to as a *truecolor* image. The image is a 3-D array of type uint8. Since swt2 requires that 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,:);
image(x)
```

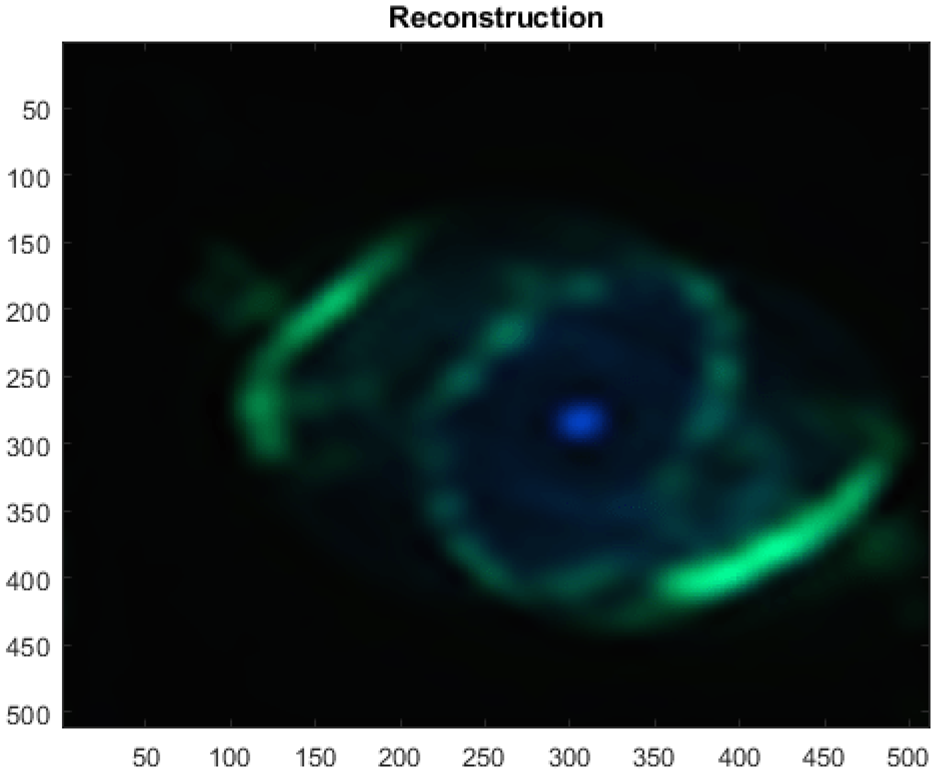


Obtain the level 4 stationary wavelet decomposition of the image using the db4 wavelet. Return the approximation coefficients and horizontal, vertical, and detail coefficients as separate arrays.

```
[a,h,v,d] = swt2(x,4,'db4');
```

Reconstruct an image using the green and blue components of the approximation coefficients. Display the reconstruction.

```
a2 = zeros(size(a));
a2(:,:,2:3,4)=a(:,:,2:3,4);
xrec = iswt2(a2,0*h,0*v,0*d,'db4');
xrec2 = (xrec-min(xrec(:)))/(max(xrec(:))-min(xrec(:)));
image(xrec2)
title('Reconstruction')
```



Input Arguments

swc — Stationary wavelet decomposition

3-D array | 4-D array

Stationary wavelet decomposition, specified as a 3-D or 4-D array. The decomposition contains the approximation and detail coefficients of the 2-D stationary wavelet transform (SWT). The stationary wavelet decomposition is the output of swt2.

Data Types: double

wname — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. iswt2 supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See wfilters for a list of orthogonal and biorthogonal wavelets. The specified wavelet must be the same wavelet used to obtain the approximation and detail coefficients.

LoR,HiR — Wavelet reconstruction filters

even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter. The lengths of LoR and HiR must be equal. See wfilters for additional information.

A — Approximation coefficients

2-D matrix | 3-D array | 4-D array

Approximation coefficients, specified as a multidimensional array. The array is the output of swt2.

Data Types: double

H,V,D — Detail coefficients

2-D matrix | 3-D array | 4-D array

Detail coefficients, specified as multidimensional arrays of equal size. H, V, and D contain the horizontal, vertical, and diagonal detail coefficients, respectively. The arrays are the output of swt2.

Data Types: double

Output Arguments

X — Reconstruction

2-D matrix | 3-D array

Reconstruction, returned as a 2-D matrix or 3-D array.

If swc or (A,H,V,D) are obtained from an indexed image analysis or a truecolor (RGB) image analysis, then X is an *m*-by-*n* matrix or an *m*-by-*n*-by-3 array, respectively.

Compatibility Considerations

Distinguish Single-Level Truecolor Image from Multilevel Indexed Image Decompositions

Behavior changed in R2017b

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.

- **Migrate from Previous Releases to R2017b**

  Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make swt2 coefficient arrays from previous releases compatible with R2017b coefficient arrays. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Input: Truecolor image
• Single-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
• Multi-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`
	`ca2 = reshape(ca1,[m,n,1,3]);`
	`chd2 = reshape(chd1,[m,n,1,3]);`
	`cvd2 = reshape(cvd1,[m,n,1,3]);`
	`cdd2 = reshape(cdd1,[m,n,1,3]);`
	• Multi-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`

- **Migrate from R2017b to Previous Releases**

  Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make R2017b `swt2` coefficient arrays compatible with the coefficient arrays from previous releases. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Input: Truecolor image
• Single-level: No compatibility issues
• Multi-level: No compatibility issues
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from R2017b, execute:
	`ca1 = single(squeeze(ca));`
	`chd1 = single(squeeze(chd));`
	`cvd1 = single(squeeze(cvd));`
	`cdd1 = single(squeeze(cdd));`
	• Multi-level: No compatibility issues

References

[1] Nason, G. P., and B. W. Silverman. “The Stationary Wavelet Transform and Some Statistical Applications.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges

Oppenheim, 103:281–99. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_17.

[2] Coifman, R. R., and D. L. Donoho. “Translation-Invariant De-Noising.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:125–50. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_9.

[3] Pesquet, J.-C., H. Krim, and H. Carfantan. “Time-Invariant Orthonormal Wavelet Representations.” *IEEE Transactions on Signal Processing* 44, no. 8 (August 1996): 1964–70. https://doi.org/10.1109/78.533717.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input wavelet name must be constant.

See Also

`idwt2` | `swt2` | `waverec2`

Introduced before R2006a

itqwt

Inverse tunable Q-factor wavelet transform

Syntax

```
xrec = itqwt(wt,n)
xrec = itqwt(wt,n,Name=Value)
```

Description

`xrec = itqwt(wt,n)` returns the inverse tunable Q-factor transform (TQWT) of the analysis coefficients in `wt`. `wt` is a cell array containing the wavelet subband and lowpass coefficients obtained from `tqwt` using the default quality factor of 1. `n` is the original signal length specified as a positive scalar.

`xrec = itqwt(wt,n,Name=Value)` specifies one or more additional name-value arguments. For example, `xrec = itqwt(x,1024,QualityFactor=2)` specifies a quality factor of 2.

Examples

Inverse Tunable Q-factor Wavelet Transform

Load an ECG signal. Obtain the tunable Q-factor wavelet transform to the default maximum decomposition level using a quality factor of 3.

```
load wecg
qf = 3;
wt = tqwt(wecg,QualityFactor=qf);
```

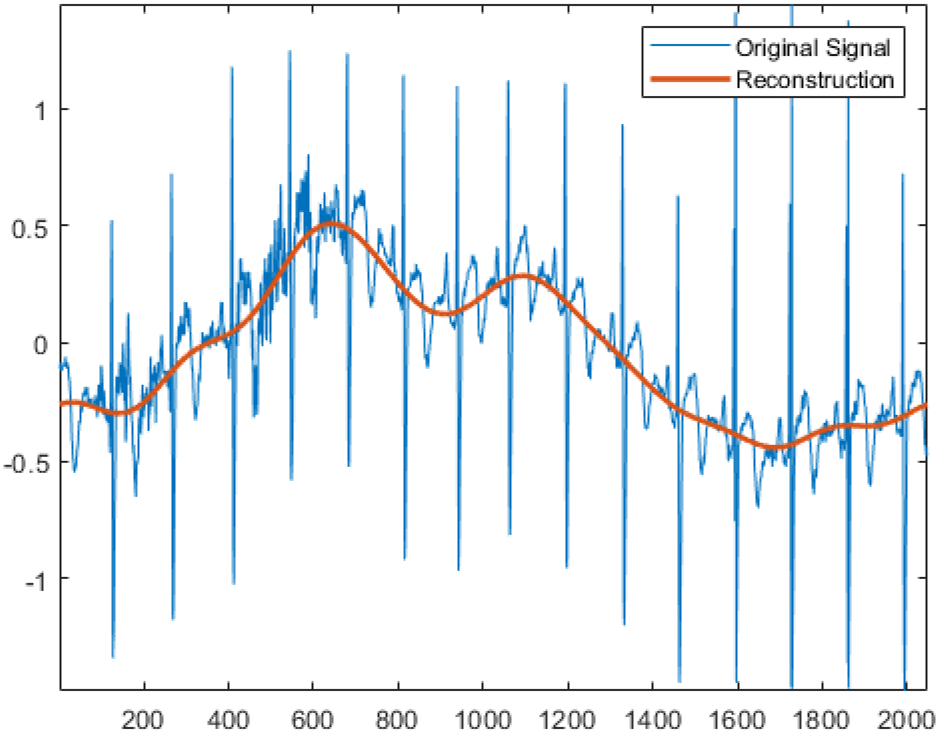
Reconstruct the data from the subband coefficients and confirm perfect reconstruction.

```
xrec = itqwt(wt,length(wecg),QualityFactor=qf);
max(abs(xrec-wecg))
```

```
ans = 5.5511e-16
```

Return the inverse TQWT at the coarsest scale.

```
xrec = itqwt(wt,length(wecg),QualityFactor=qf,Level=length(wt)-2);
plot(wecg)
hold on
plot(xrec,LineWidth=2)
hold off
axis tight
legend("Original Signal","Reconstruction")
```



Input Arguments

wt — Tunable Q-factor wavelet transform

cell array

Tunable Q-factor wavelet transform, specified as a cell array. The elements of wt contain the wavelet subband and lowpass coefficients. wt is expected to be the output of tqwt.

Data Types: single | double
Complex Number Support: Yes

n — Original signal length

positive integer

Original signal length in samples, specified as a positive integer. If n is odd, n+1 is used internally to invert the TQWT and the first n samples are returned.

Data Types: single | double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: xrec = itqwt(wt,2048,Level=3,QualityFactor=2)

`Level` — Reconstruction level

`0` (default) | nonnegative integer

Reconstruction level of analysis coefficients, specified as a nonnegative integer less than or equal to `length(wt)-2`. If unspecified, `itqwt` reconstructs the TQWT up to the resolution level of the original signal.

Example: `xrec = itqwt(x,Level=1)` returns the inverse TQWT at level 1.

Data Types: `single` | `double`

`QualityFactor` — Quality factor

`1` (default) | positive scalar

Quality factor, specified as a positive scalar greater than or equal to 1. The quality factor must match the value used in the TQWT.

Example: `xrec = itqwt(x,QualityFactor=2)` specifies a quality factor of 2.

Data Types: `single` | `double`

Output Arguments

`xrec` — Inverse tunable-Q wavelet transform

vector | matrix | 3-D array

Inverse tunable-Q wavelet transform, returned as a vector, matrix, or 3-D array.

Data Types: `single` | `double`

Tips

- To reconstruct only the scaling coefficients, specify a reconstruction level of `length(wt)-1`.

References

[1] Selesnick, Ivan W. “Wavelet Transform With Tunable Q-Factor.” *IEEE Transactions on Signal Processing* 59, no. 8 (August 2011): 3560–75. https://doi.org/10.1109/TSP.2011.2143711.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

- The TQWT array `wt` is the only supported input argument.

See Also

Apps

Signal Multiresolution Analyzer

Functions
`tqwt` | `tqwtmra`

Topics
"Time-Frequency Gallery"
"Tunable Q-factor Wavelet Transform"

Introduced in R2021b

iwsst

Inverse wavelet synchrosqueezed transform

Syntax

```
xrec = iwsst(sst)
xrec = iwsst(sst,f,freqrange)
xrec = iwsst(sst,iridge)
xrec = iwsst( ___ ,wav)
xrec = iwsst( ___ ,iridge,'NumFrequencyBins',numBins)
```

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 synchrosqueezed 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,'NumFrequencyBins',numBins)` returns the inverse synchrosqueezed transform with `numBins`-many additional frequency bins included on either side of each `iridge` index bin.

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.

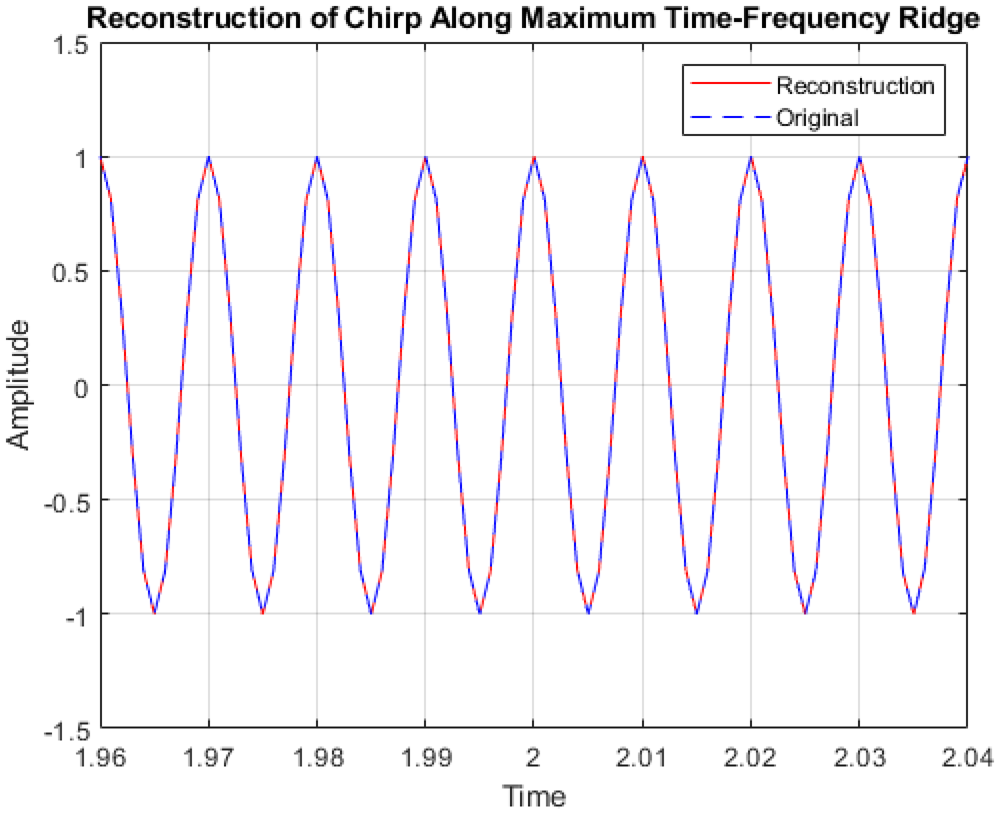
```
load quadchirp;
sstchirp = wsst(quadchirp,'ExtendSignal',true);
```

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

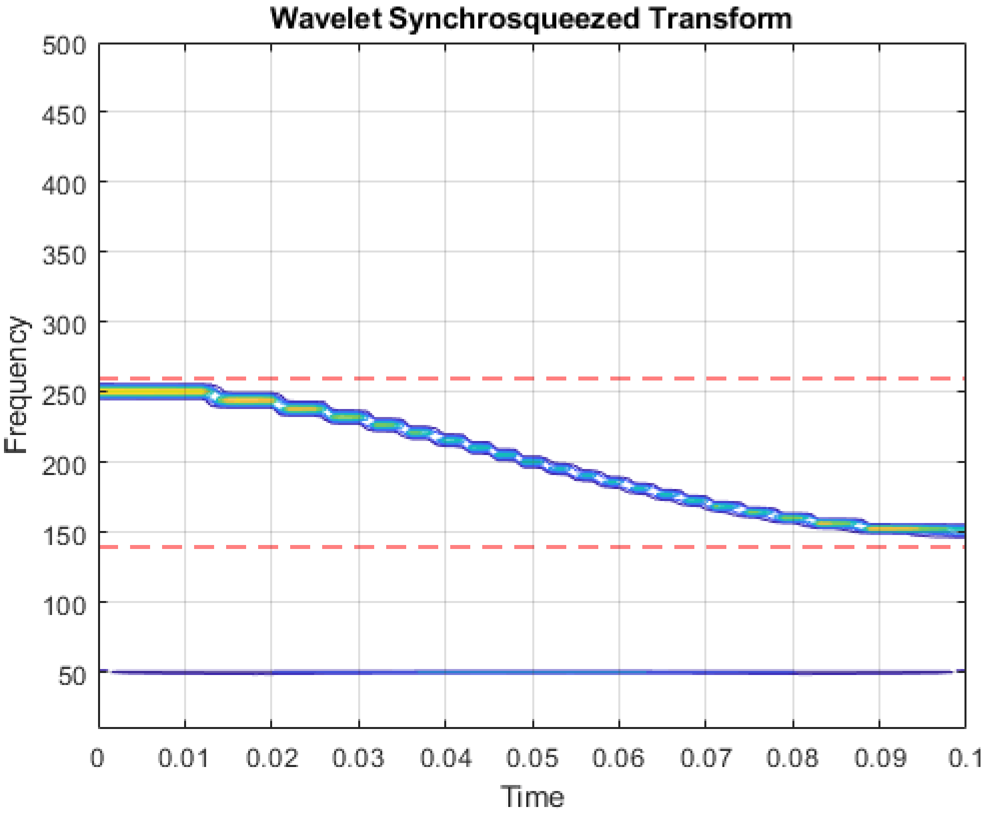
Obtain the inverse synchrosqueezed transform for a specified frequency range of a two-component signal. The input is a combination of an amplitude-modulated and a frequency-modulated signal.

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;
```

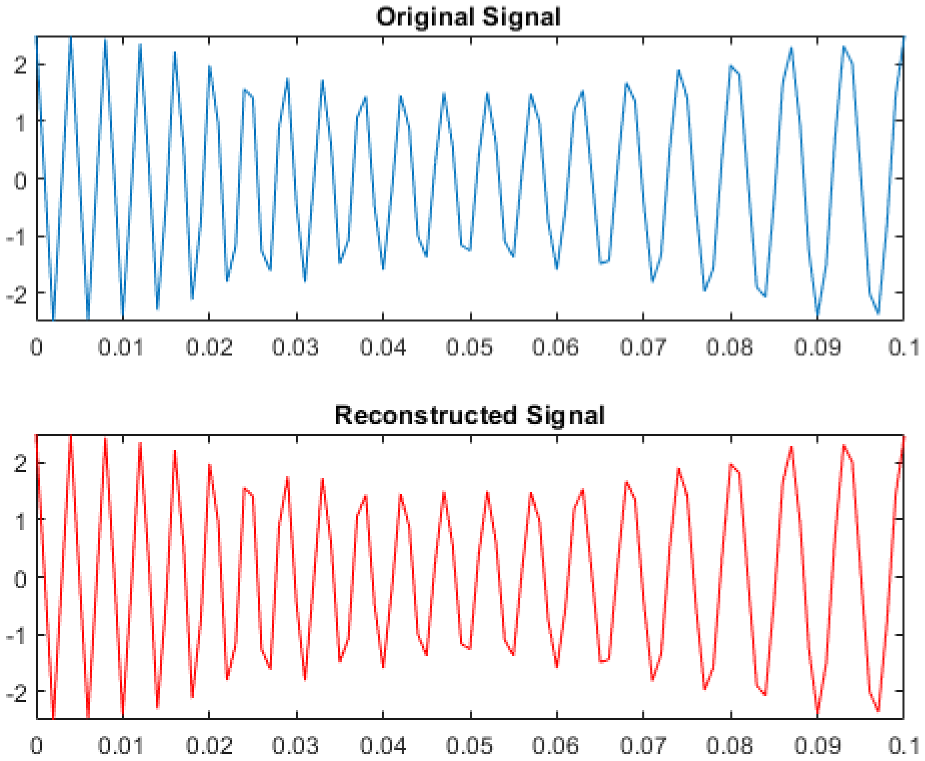
Obtain the wavelet synchrosqueezed transform and plot the resulting two frequency components.

```
[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');
```


Original Signal

Reconstructed Signal

Synchrosqueezed and Inverse Synchrosqueezed Transform of Speech Signal

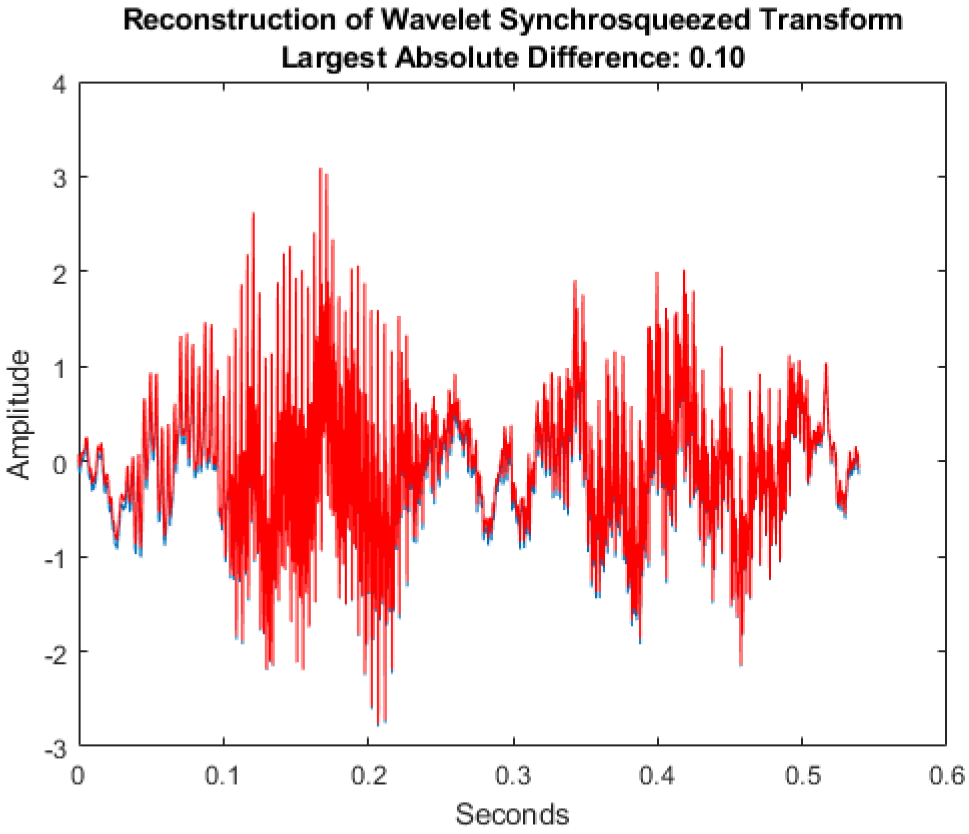
Obtain the wavelet synchrosqueezed transform and inverse synchrosqueezed transform of a speech sample using the bump wavelet.

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.

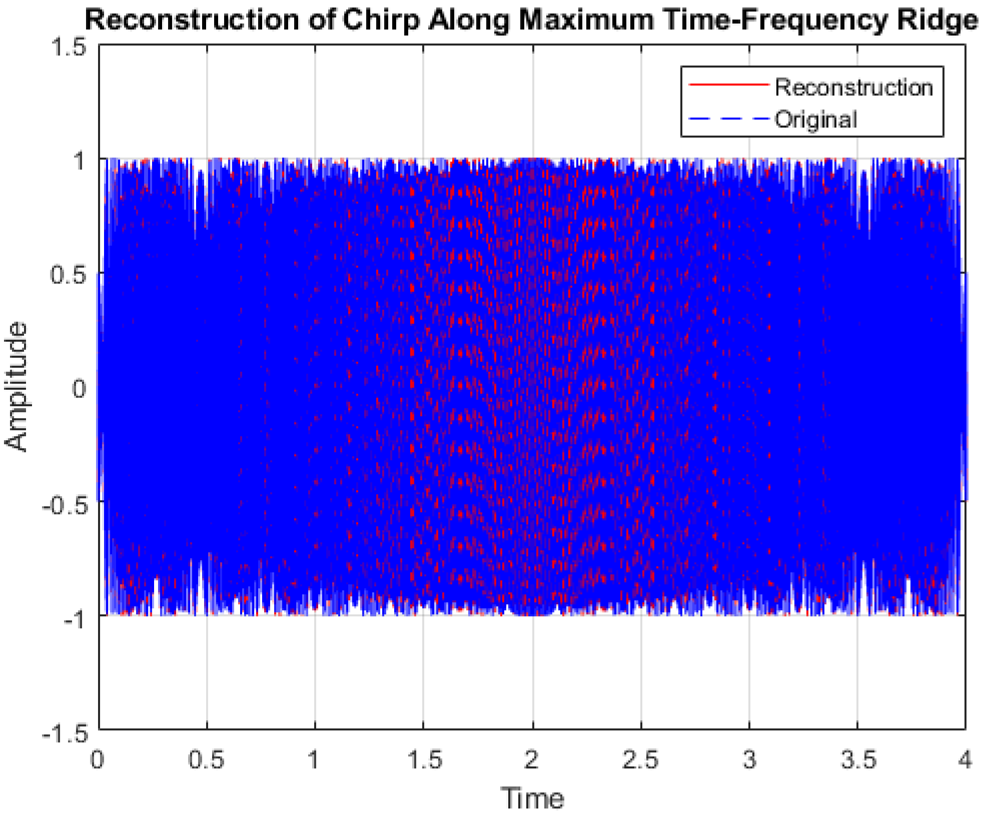
```
load quadchirp;
sstchirp = wsst(quadchirp,'ExtendSignal',true);
```

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');
```



Input Arguments

sst — Synchrosqueezed transform
matrix

Synchrosqueezed transform, specified as a matrix. `sst` is the output from the `wsst` function.

f — Synchrosqueezed 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
`'amor'` (default) | `'bump'`

Analytic wavelet used to compute the inverse synchrosqueezed transform, specified as one of the following:

- `'amor'` — Analytic Morlet wavelet
- `'bump'` — Bump wavelet

You must use the same wavelet in the reconstruction that you used to compute the synchrosqueezed transform, `sst`.

`numBins` — Number of additional frequency bins
16 (default) | positive integer

Number of additional frequency bins to include on either side of each `iridge` index bin, specified as 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.

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., J. Lu, and H. T. Wu. "Synchrosqueezed Wavelet Transforms: an Empirical Mode Decomposition-like Tool." *Applied and Computational Harmonic Analysis*, Vol. 30, Number 2, 2011, 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, Number 5, 2013, pp. 1079–1094.

See Also

`wsst` | `wsstridge`

Topics

"Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing"
"Wavelet Synchrosqueezing"
"Time-Frequency Gallery"

Introduced in R2016a

labelDefinitionsHierarchy

Get hierarchical list of label and sublabel names

Syntax

```
str = labelDefinitionsHierarchy(lbldefs)
str = labelDefinitionsHierarchy(lss)
```

Description

`str = labelDefinitionsHierarchy(lbldefs)` returns a character array with a hierarchical list of label and sublabel names contained in `lbldefs`, a vector of `signalLabelDefinition` objects.

`str = labelDefinitionsHierarchy(lss)` returns a character array with a hierarchical list of label and sublabel names contained in the `labeledSignalSet` object `lss`.

Examples

Label Hierarchy

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Visualize the label hierarchy of the set.

```
labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: []
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: TrillPeaks
     '
```

Input Arguments

`lbldefs` — Signal label definitions

`signalLabelDefinition` object | vector of `signalLabelDefinition` objects

Signal label definitions, specified as a `signalLabelDefinition` object or a vector of `signalLabelDefinition` objects.

Example: `signalLabelDefinition("Asleep",'LabelType','roi','LabelDataType','logical')` can label a region of a signal in which a patient is asleep.

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

Output Arguments

`str` — List of label and sublabel names

character array

List of label and sublabel names, returned as a character array.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

labelDefinitionsSummary

Get summary table of signal label definitions

Syntax

```
T = labelDefinitionsSummary(lbldefs)
T = labelDefinitionsSummary(lss)
```

```
T = labelDefinitionsSummary( ___ ,lblname)
T = labelDefinitionsSummary( ___ ,lblname,'sublbls')
```

Description

`T = labelDefinitionsSummary(lbldefs)` returns a table, `T`, with the properties of the label definitions contained in `lbldefs`, a vector of `signalLabelDefinition` objects.

`T = labelDefinitionsSummary(lss)` returns a table, `T`, with the properties of the label definitions contained in the `labeledSignalSet` object `lss`.

`T = labelDefinitionsSummary( ___ ,lblname)` returns a table with the properties of the label `lblname`.

`T = labelDefinitionsSummary( ___ ,lblname,'sublbls')` returns a table of the properties of the sublabels defined for `lblname`.

Examples

Label Properties

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Visualize the label properties of the set.

```
labelDefinitionsSummary(lss)
```

```
ans=3×9 table
      LabelName          LabelType      LabelDataType       Categories        ValidationFunction      Defau
    ______________     ___________     _____________     ____________     __________________     ______

    "WhaleType"        "attribute"     "categorical"     {3x1 string}         {["N/A"   ]}        {0x0
    "MoanRegions"      "roi"           "logical"         {["N/A"   ]}         {0x0 double}        {0x0
    "TrillRegions"     "roi"           "logical"         {["N/A"   ]}         {0x0 double}        {0x0
```

Visualize the properties of the `TrillRegions` label.

```
labelDefinitionsSummary(lss,"TrillRegions")
```

```
ans=1×9 table
      LabelName        LabelType     LabelDataType     Categories      ValidationFunction     DefaultVa
    ______________     _________     _____________     __________     __________________     _________

    "TrillRegions"       "roi"         "logical"       {["N/A"]}        {0x0 double}         {0x0 doub
```

Visualize the properties of the `TrillRegions` sublabels.

```
labelDefinitionsSummary(lss,"TrillRegions",'sublbls')
```

```
ans=1×8 table
     LabelName       LabelType     LabelDataType     Categories      ValidationFunction     DefaultValu
    ____________     _________     _____________     __________     __________________     ___________

    "TrillPeaks"      "point"        "numeric"       {["N/A"]}        {0x0 double}         {0x0 double
```

Input Arguments

`lbldefs` — Signal label definitions

`signalLabelDefinition` object | vector of `signalLabelDefinition` objects

Signal label definitions, specified as a `signalLabelDefinition` object or a vector of `signalLabelDefinition` objects.

Example:
`signalLabelDefinition("Asleep",'LabelType','roi','LabelDataType','logical')` can label a region of a signal in which a patient is asleep.

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

Output Arguments

T — Summary table

table

Summary table with the properties of a label.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

labeledSignalSet

Create labeled signal set

Description

Use `labeledSignalSet` to store labeled signals along with the label definitions. Create signal label definitions using `signalLabelDefinition`.

Creation

Syntax

```
lss = labeledSignalSet

lss = labeledSignalSet(src)
lss = labeledSignalSet(src,lbldefs)

lss = labeledSignalSet(src,lbldefs,'MemberNames',mnames)
lss = labeledSignalSet(src,lbldefs,Name,Value)
```

Description

`lss = labeledSignalSet` creates an empty labeled signal set. Use `addMembers` to add signals to the set. Use `addLabelDefinitions` to add label definitions to the set.

`lss = labeledSignalSet(src)` creates a labeled signal set for the input data source `src`. Use `addLabelDefinitions` to add label definitions to the set.

`lss = labeledSignalSet(src,lbldefs)` creates a labeled signal set for the input data source `src` using the signal label definitions `lbldefs`. Use `signalLabelDefinition` to create signal label definitions.

`lss = labeledSignalSet(src,lbldefs,'MemberNames',mnames)` creates a labeled signal set for the input data source `src` and specifies names for the members of the set. Use `setMemberNames` to modify the member names. `lbldefs` is optional.

`lss = labeledSignalSet(src,lbldefs,Name,Value)` sets “Properties” on page 1-766 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in quotes. `lbldefs` is optional.

Input Arguments

`src` — Input data source
matrix | cell array | timetable | `signalDatastore` object | `audioDatastore` object

Input data source, specified as a matrix, a cell array, a timetable, a `signalDatastore` object, or an `audioDatastore` object. `src` implicitly specifies the number of members of the set, the number of signals in each member, and the data in each signal.

Example: {randn(10,3),randn(17,9)} has two members. The first member contains three 10-sample signals. The second member contains nine 17-sample signals.

Example: {{randn(10,1)},{randn(17,1),randn(27,1)}} has two members. The first member contains one 10-sample signal. The second member contains a 17-sample signal and a 27-sample signal.

Example:
{{timetable(seconds(1:10)',randn(10,3)),timetable(seconds(1:7)',randn(7,2))},{timetable(seconds(1:3)',randn(3,1))}} has two members. The first member contains three signals sampled at 1 Hz for 10 seconds and two signals sampled at 1 Hz for 7 seconds. The second member contains one signal sampled at 1 Hz for 3 seconds.

Example: signalDatastore Object Pointing to Files

Specify the path to a set of sample sound signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'
```

```
nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to Fs, which is common to all files. Generate a subset of the datastore that excludes the file mtlb.mat, which differs from the other files in that the signal variable is not called y.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sdss = subset(sds,~strcmp(nms,"mtlb.mat"));
```

Use the subset datastore as the source for a labeledSignalSet object.

```
lss = labeledSignalSet(sdss)
```

```
lss = 
  labeledSignalSet with properties:

             Source: [1x1 signalDatastore]
         NumMembers: 6
    TimeInformation: "inherent"
             Labels: [6x0 table]
        Description: ""

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

lbldefs — Label definitions
vector of signalLabelDefinition objects

Label definitions, specified as a vector of `signalLabelDefinition` objects.

`mnames` — Member names
character vector | string scalar | cell array of character vectors | string array

Member names, specified as a character vector, a string scalar, a cell array of character vectors, or a string array.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},'MemberNames',{'llama' 'alpaca'})` specifies a set of random signals with two members, `'llama'` and `'alpaca'`.

Properties

`Description` — Labeled signal set description
character vector | string scalar

Labeled signal set description, specified as a character vector or string scalar.

Example: `'Description','Sleep test patients by sex and age'`

Data Types: `char` | `string`

`SampleRate` — Sample rate values
positive scalar | vector

This property is read-only.

Sample rate values, specified as a positive scalar or a vector. This property is valid only when the data source does not contain inherent time information.

- Set `SampleRate` to a positive numeric scalar to specify the same sample rate for all signals in the labeled set.
- Set `SampleRate` to a vector to specify that each member of the labeled set has signals sampled at the same rate, but the sample rates differ from member to member. The vector must have a number of elements equal to the number of members of the set. If a member of a set has signals with different sample rates, then specify the sample rates using timetables.

Example: `'SampleRate',[1e2 1e3]` specifies that the signals in the first member of a set are sampled at a rate of 100 Hz and the signals in the second member are sampled at 1 kHz.

`SampleTime` — Sample time values
positive scalar | vector | `duration` scalar | `duration` vector

This property is read-only.

Sample time values, specified as a positive scalar, a vector, a `duration` scalar, or a `duration` vector. This property is valid only when the data source does not contain inherent time information.

- Set `SampleTime` to a numeric or `duration` scalar to specify the same sample time for all signals in the labeled set.
- Set `SampleTime` to a numeric or `duration` vector to specify that each member of the labeled set has signals with the same time interval between samples, but the intervals differ from member to member. The vector must have a number of elements equal to the number of members of the set. If a member of a set has signals with different sample times, then specify the sample times using timetables.

Example: `'SampleTime',seconds([1e-2 1e-3])` specifies that the signals in the first member of a set have 0.01 second between samples, and the signals in the second member have 1 millisecond between samples.

`TimeValues` — Time values

vector | `duration` vector | matrix | cell array

This property is read-only.

Time values, specified as a vector, a `duration` vector, a matrix, or a cell array. This property is valid only when the data source does not contain inherent time information. Time values must be unique and increasing.

- Set `TimeValues` to a numeric or `duration` vector to specify the same time values for all signals in the labeled set. The vector must have the same length as all the signals in the set.
- Set `TimeValues` to a numeric or `duration` matrix or cell array to specify that each member of the labeled set has signals with the same time values, but the time values differ from member to member.

  - If `TimeValues` is a matrix, then it must have a number of columns equal to the number of members of the set. All signals in the set must have a length equal to the number of rows of the matrix.
  - If `TimeValues` is a cell array, then it must contain a number of vectors equal to the number of members of the set. All signals in a member must have a length equal to the number of elements of the corresponding vector in the cell array.

If a member of a set has signals with different time values, then specify the time values using timetables.

Example: `'TimeValues',[1:1000;0:1/500:2-1/500]'` specifies that the signals in the first member of a set are sampled 1 Hz for 1000 seconds. The signals in the second member are sampled at 500 Hz for 2 seconds.

Example: `'TimeValues',seconds([1:1000;0:1/500:2-1/500]')` specifies that the signals in the first member of a set are sampled 1 Hz for 1000 seconds. The signals in the second member are sampled at 500 Hz for 2 seconds.

Example: `'TimeValues',{1:1000,0:1/500:2-1/500}` specifies that the signals in the first member of a set are sampled 1 Hz for 1000 seconds. The signals in the second member are sampled at 500 Hz for 2 seconds.

Example: `'TimeValues',{seconds(1:1000),seconds(0:1/500:2-1/500)}` specifies that the signals in the first member of a set are sampled 1 Hz for 1000 seconds. The signals in the second member are sampled at 500 Hz for 2 seconds.

`NumMembers` — Number of members in set

positive integer

This property is read-only.

Number of members in set, specified as a positive integer.

`Labels` — Labels table

table

This property is read-only.

Labels table, specified as a MATLAB table. Each variable of `Labels` corresponds to a label defined for the set. Each row of `Labels` corresponds to a member of the data source. The row names of `Labels` are the member names.

Data Types: `table`

`TimeInformation` — Time information of source
`'none'` | `'sampleRate'` | `'sampleTime'` | `'timeValues'` | `'inherent'`

Time information of source, specified as one of the following:

- `'none'` — The signals in the source have no time information.
- `'sampleRate'` — The signals in the source are sampled at a specified rate.
- `'sampleTime'` — The signals in the source have a specified time interval between samples.
- `'timeValues` — The signals in the source have a time value corresponding to each sample.
- `'inherent'` — The signals in the source contain inherent time information. MATLAB timetables are an example of such signals.

Data Types: `char` | `string`

`Source` — Data source of labeled signal set
matrix | cell array | timetable

This property is read-only.

Data source of labeled signal set, specified as a matrix, a timetable, a cell array, or an audio datastore.

- If `Source` is a numeric matrix, then the labeled signal set has one member that contains a number of signals equal to the number of matrix columns.

  Example: `labeledSignalSet(randn(10,3))` has one member that contains three 10-sample signals.
- If `Source` is a cell array of matrices, then the labeled signal set has a number of members equal to the number of matrices in the cell array. Each member contains a number of signals equal to the number of columns of the corresponding matrix.

  Example: `labeledSignalSet({randn(10,3),randn(17,9)})` has two members. The first member contains three 10-sample signals. The second member contains nine 17-sample signals.
- If `Source` is a cell array, and each element of the cell array is a cell array of numeric vectors, then the labeled signal set has a number of members equal to the number of cell array elements. Each signal within a member can have any length.

  Example: `labeledSignalSet({{randn(10,1)},{randn(17,1),randn(27,1)}})` has two members. The first member contains one 10-sample signal. The second member contains a 17-sample signal and a 27-sample signal.
- If `Source` is a timetable with variables containing numeric values, then the labeled signal set has one member that contains a number of signals equal to the number of variables. The time values of the timetable must be of type `duration`, unique, and increasing.

  Example: `labeledSignalSet(timetable(seconds(1:10)',randn(10,3)))` has one member that contains three signals sampled at 1 Hz for 10 seconds.

- If `Source` is a cell array of timetables, and each timetable has an arbitrary number of variables with numeric values, then the labeled signal set has a number of members equal to the number of timetables. Each member contains a number of signals equal to the number of variables in the corresponding timetable.

  Example:
  `labeledSignalSet({timetable(seconds(1:10)',randn(10,3)),timetable(seconds(1:5)',randn(5,13))})` has two members. The first member contains three signals sampled at 1 Hz for 10 seconds. The second member contains 13 signals sampled at 1 Hz for 5 seconds.
- If `Source` is a cell array, and each element of the cell array is a cell array of timetables, then the labeled signal set has a number of members equal to the number of cell array elements. Each member can have any number of timetables, and each timetable within a member can have any number of variables.

  Example:
  `labeledSignalSet({{timetable(seconds(1:10)',randn(10,3)),timetable(seconds(1:7)',randn(7,2))},{timetable(seconds(1:3)',randn(3,1))}})` has two members. The first member contains three signals sampled at 1 Hz for 10 seconds and two signals sampled at 1 Hz for 7 seconds. The second member contains one signal sampled at 1 Hz for 3 seconds.
- If the input data source, `src`, is an audio datastore, then the labeled signal set has a number of members equal to the number of files to which the datastore points. The `Source` property contains a cell array of character vectors with the file names. Each member contains all the signals returned by the read of the corresponding datastore file.

Object Functions

addLabelDefinitions	Add label definitions to labeled signal set
addMembers	Add members to labeled signal set
countLabelValues	Count label values
createDatastores	Create datastores pointing to signal and label data
createFeatureData	Create feature table or matrix and response vectors
editLabelDefinition	Edit label definition properties
getAlternateFileSystemRoots	Get alternate file system roots when data source of labeled signal set is a datastore
getLabelDefinitions	Get label definitions in labeled signal set
getLabeledSignal	Get labeled signals from labeled signal set
getLabelIndices	Get label indices pointing to label definitions in labeled signal set
getLabelNames	Get label names in labeled signal set
getLabelValues	Get label values from labeled signal set
getMemberNames	Get member names in labeled signal set
getSignal	Get signals from labeled signal set
head	Get top rows of labels table
labelDefinitionsHierarchy	Get hierarchical list of label and sublabel names
labelDefinitionsSummary	Get summary table of signal label definitions
merge	Merge two or more labeled signal sets
removeLabelDefinition	Remove label definition from labeled signal set
removeMembers	Remove members from labeled signal set
removePointValue	Remove row from point label
removeRegionValue	Remove row from ROI label
resetLabelValues	Reset labels to default values
setAlternateFileSystemRoots	Set alternate file system roots when data source of labeled signal set is a datastore

setLabelValue	Set label value in labeled signal set
setMemberNames	Set member names in labeled signal set
subset	Get new labeled signal set with subset of members

Examples

Label Definitions for Whale Songs

Consider a set of whale sound recordings. The recorded whale sounds consist of trills and moans. *Trills* sound like series of clicks. *Moans* are low-frequency cries similar to the sound made by a ship's horn. You want to look at each signal and label it to identify the whale type, the trill regions, and the moan regions. For each trill region, you also want to label the signal peaks higher than a certain threshold.

Signal Label Definitions

Define an attribute label to store whale types. The possible categories are blue whale, humpback whale, and white whale.

```
dWhaleType = signalLabelDefinition('WhaleType',...
   'LabelType','attribute',...
   'LabelDataType','categorical',...
   'Categories',{'blue','humpback','white'},...
   'Description','Whale type');
```

Define a region-of-interest (ROI) label to capture moan regions. Define another ROI label to capture trill regions.

```
dMoans = signalLabelDefinition('MoanRegions',...
   'LabelType','roi',...
   'LabelDataType','logical',...
   'Description','Regions where moans occur');

dTrills = signalLabelDefinition('TrillRegions',...
   'LabelType','roi',...
   'LabelDataType','logical',...
   'Description','Regions where trills occur');
```

Finally, define a point label to capture the trill peaks. Set this label as a sublabel of the `dTrills` definition.

```
dTrillPeaks = signalLabelDefinition('TrillPeaks',...
   'LabelType','point',...
   'LabelDataType','numeric',...
   'Description','Trill peaks');

dTrills.Sublabels = dTrillPeaks;
```

Labeled Signal Set

Create a `labeledSignalSet` with the whale signals and the label definitions. Add label values to identify the whale type, the moan and trill regions, and the peaks of the trills.

```
load labelwhalesignals
lbldefs = [dWhaleType dMoans dTrills];
```

```
lss = labeledSignalSet({whale1 whale2},lbldefs,'MemberNames',{'Whale1','Whale2'}, ...
   'SampleRate',Fs,'Description','Characterize whale song regions');
```

Visualize the label hierarchy and label properties using `labelDefinitionsHierarchy` and `labelDefinitionsSummary`.

```
labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: []
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: TrillPeaks
     '
```

```
labelDefinitionsSummary(lss)
```

```
ans=3×9 table
      LabelName         LabelType       LabelDataType      Categories       ValidationFunction     Defa
    ______________    ___________    _____________    ____________    __________________    ____

    "WhaleType"       "attribute"    "categorical"    {3x1 string}       {["N/A"   ]}       {0x0
    "MoanRegions"     "roi"          "logical"        {["N/A"   ]}       {0x0 double}       {0x0
    "TrillRegions"    "roi"          "logical"        {["N/A"   ]}       {0x0 double}       {0x0
```

The signals in the loaded data correspond to songs of two blue whales. Set the `'WhaleType'` values for both signals.

```
setLabelValue(lss,1,'WhaleType','blue');
setLabelValue(lss,2,'WhaleType','blue');
```

Visualize the `'Labels'` property. The table has the newly added `'WhaleType'` values for both signals.

```
lss.Labels
```

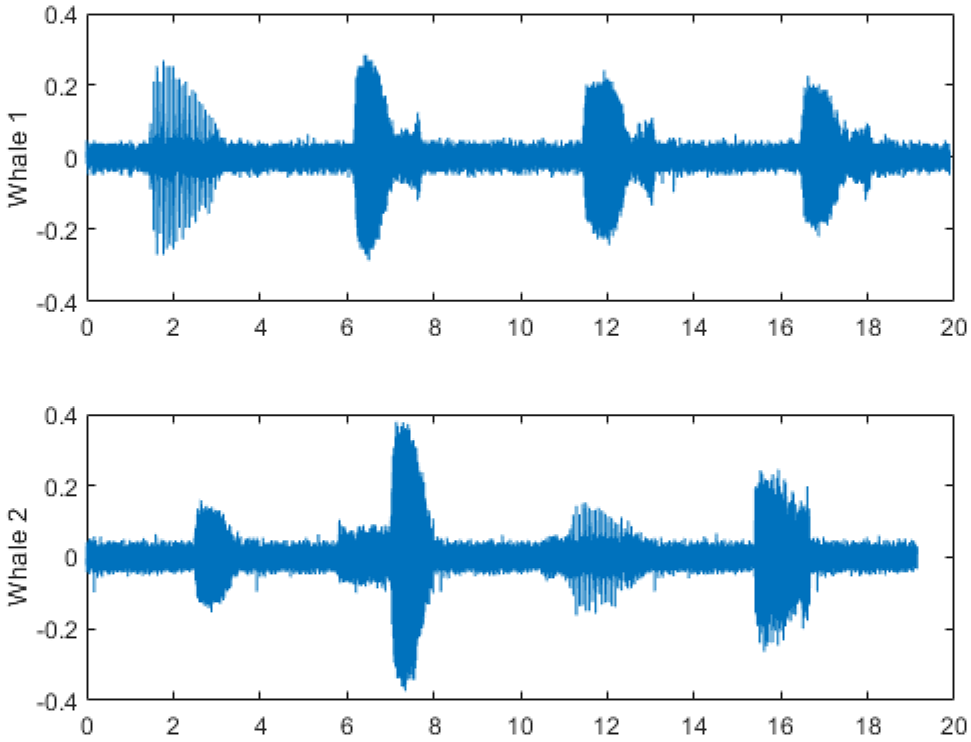
```
ans=2×3 table
              WhaleType     MoanRegions     TrillRegions
              _________    ___________    ____________

    Whale1      blue       {0x2 table}     {0x3 table}
    Whale2      blue       {0x2 table}     {0x3 table}
```

Visualize Region Labels

Visualize the whale songs to identify the trill and moan regions.

```
subplot(2,1,1)
plot((0:length(whale1)-1)/Fs,whale1)
ylabel('Whale 1')

subplot(2,1,2)
```

```
plot((0:length(whale2)-1)/Fs,whale2)
ylabel('Whale 2')
```



Moan regions are sustained low-frequency wails.

- `whale1` has moans centered at about 7 seconds, 12 seconds, and 17 seconds.
- `whale2` has moans centered at about 3 seconds, 7 seconds, and 16 seconds.

Add the moan regions to the labeled set. Specify the ROI limits in seconds and the label values.

```
moanRegionsWhale1 = [6.1 7.7; 11.4 13.1; 16.5 18.1];
mrsz1 = [size(moanRegionsWhale1,1) 1];
setLabelValue(lss,1,'MoanRegions',moanRegionsWhale1,true(mrsz1));

moanRegionsWhale2 = [2.5 3.5; 5.8 8; 15.4 16.7];
mrsz2 = [size(moanRegionsWhale2,1) 1];
setLabelValue(lss,2,'MoanRegions',moanRegionsWhale2,true(mrsz2));
```

Trill regions have distinct bursts of sound punctuated by silence.

- `whale1` has a trill centered at about 2 seconds.
- `whale2` has a trill centered at about 12 seconds.

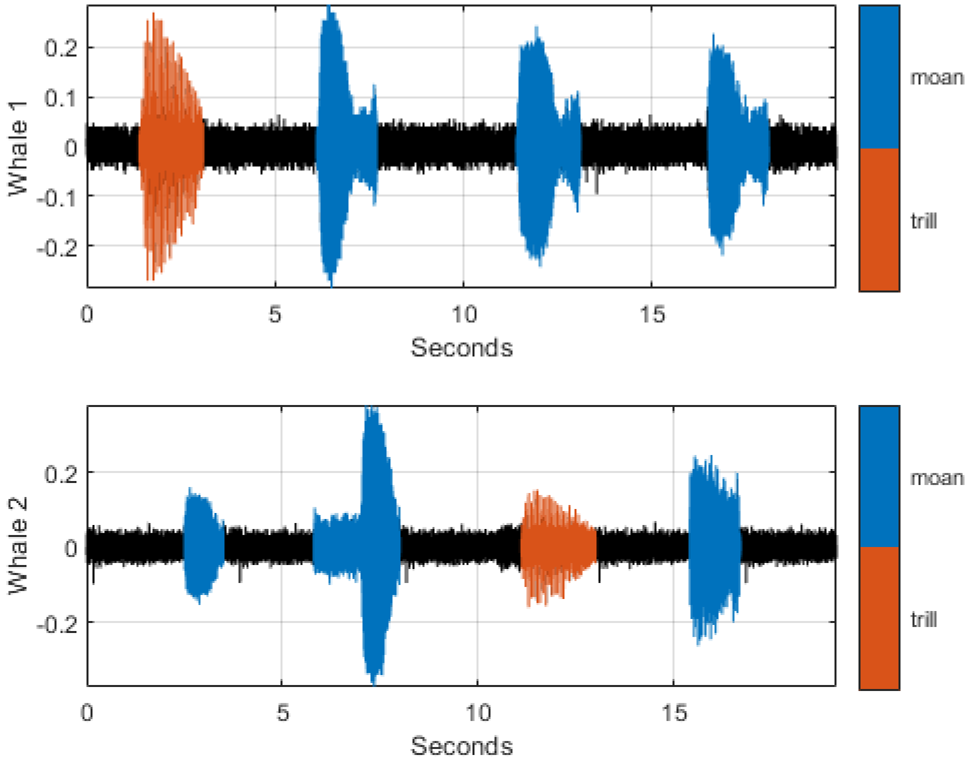
Add the trill regions to the labeled set.

```
trillRegionWhale1 = [1.4 3.1];
trsz1 = [size(trillRegionWhale1,1) 1];
```

```
setLabelValue(lss,1,'TrillRegions',trillRegionWhale1,true(trsz1));

trillRegionWhale2 = [11.1 13];
trsz2 = [size(trillRegionWhale1,1) 1];
setLabelValue(lss,2,'TrillRegions',trillRegionWhale2,true(trsz2));
```

Create a `signalMask` (Signal Processing Toolbox) object for each whale song and use it to visualize and label the different regions. For better visualization, change the label values from logical to categorical.

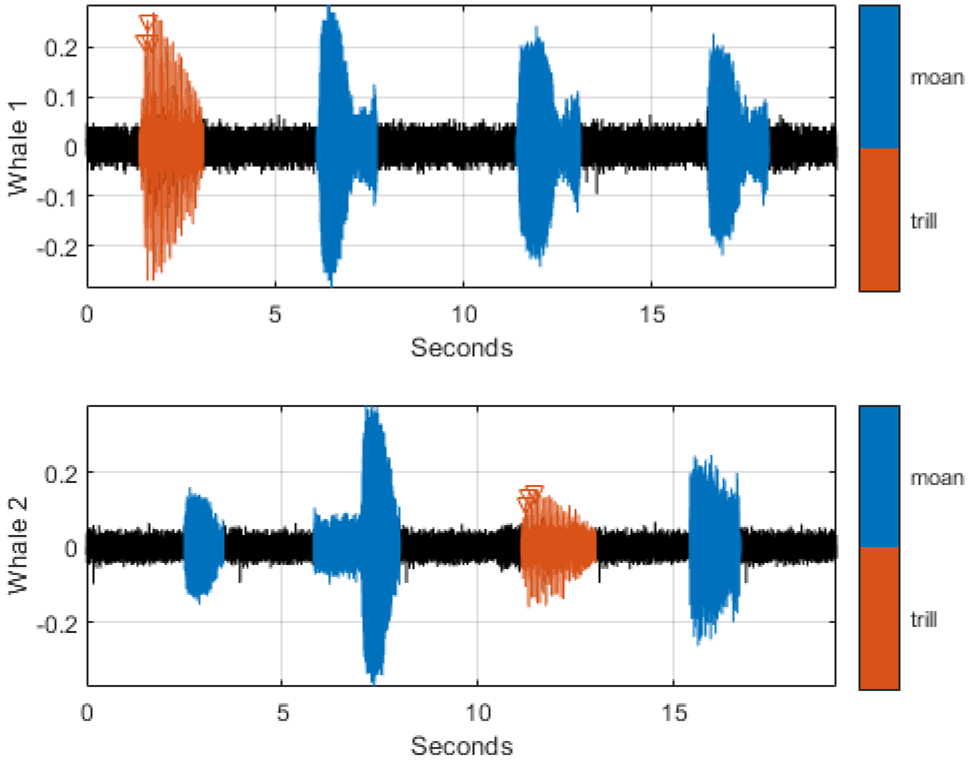
```
mr1 = getLabelValues(lss,1,'MoanRegions');
mr1.Value = categorical(repmat("moan",mrsz1));
tr1 = getLabelValues(lss,1,'TrillRegions');
tr1.Value = categorical(repmat("trill",trsz1));

msk1 = signalMask([mr1;tr1],'SampleRate',Fs);

subplot(2,1,1)
plotsigroi(msk1,whale1)
ylabel('Whale 1')
hold on

mr2 = getLabelValues(lss,2,'MoanRegions');
mr2.Value = categorical(repmat("moan",mrsz2));
tr2 = getLabelValues(lss,2,'TrillRegions');
tr2.Value = categorical(repmat("trill",trsz2));

msk2 = signalMask([mr2;tr2],'SampleRate',Fs);

subplot(2,1,2)
plotsigroi(msk2,whale2)
ylabel('Whale 2')
hold on
```



Visualize Point Labels

Label three peaks for each trill region. For point labels, you specify the point locations and the label values. In this example, the point locations are in seconds.

```
peakLocsWhale1 = [1.553 1.626 1.7];
peakValsWhale1 = [0.211 0.254 0.211];

setLabelValue(lss,1,{'TrillRegions','TrillPeaks'}, ...
   peakLocsWhale1,peakValsWhale1,'LabelRowIndex',1);

subplot(2,1,1)
plot(peakLocsWhale1,peakValsWhale1,'v')
hold off

peakLocsWhale2 = [11.214 11.288 11.437];
peakValsWhale2 = [0.119 0.14 0.15];

setLabelValue(lss,2,{'TrillRegions','TrillPeaks'}, ...
   peakLocsWhale2,peakValsWhale2,'LabelRowIndex',1);

subplot(2,1,2)
plot(peakLocsWhale2,peakValsWhale2,'v')
hold off
```



Explore Label Values

Explore the label values using `getLabelValues`.

```
getLabelValues(lss)
```

```
ans=2×3 table
                 WhaleType     MoanRegions      TrillRegions
                 _________    ____________     ____________

    Whale1         blue       {3x2 table}      {1x3 table}
    Whale2         blue       {3x2 table}      {1x3 table}
```

Retrieve the moan regions for the first member of the labeled set.

```
getLabelValues(lss,1,'MoanRegions')
```

```
ans=3×2 table
      ROILimits        Value
    ____________       _____

     6.1      7.7      {[1]}
    11.4     13.1      {[1]}
    16.5     18.1      {[1]}
```

Use a second output argument to list the sublabels of a label.

```
[value,valueWithSublabel] = getLabelValues(lss,1,'TrillRegions')
```

```
value=1×2 table
    ROILimits     Value
    __________    _____

    1.4    3.1    {[1]}


valueWithSublabel=1×3 table
    ROILimits     Value     Sublabels
                           TrillPeaks
    __________    _____    ___________

    1.4    3.1    {[1]}    {3x2 table}
```

To retrieve the values in a sublabel, express the label name as a two-element array.

```
getLabelValues(lss,1,{'TrillRegions','TrillPeaks'})
```

```
ans=3×2 table
    Location       Value
    ________    __________

     1.553      {[0.2110]}
     1.626      {[0.2540]}
       1.7      {[0.2110]}
```

Find the value of the third trill peak corresponding to the second member of the set.

```
getLabelValues(lss,2,{'TrillRegions','TrillPeaks'}, ...
    'LabelRowIndex',1,'SublabelRowIndex',3)
```

```
ans=1×2 table
    Location       Value
    ________    __________

     11.437     {[0.1500]}
```

Count Label Values and Create Datastores

Specify the path to a set of audio signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'
```

```
nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to `Fs`, which is common to all files. Generate a subset of the datastore that excludes the file `mtlb.mat`. Use the subset datastore as the source for a `labeledSignalSet` object.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sds = subset(sds,~strcmp(nms,"mtlb.mat"));
lss = labeledSignalSet(sds);
```

Create three label definitions to label the signals:

- Define a logical attribute label that is true for signals that contain human voices.
- Define a numeric point label that marks the location and amplitude of the maximum of each signal.
- Define a categorical region-of-interest (ROI) label to pick out nonoverlapping, uniform-length random regions of each signal.

Add the signal label definitions to the labeled signal set.

```
vc = signalLabelDefinition("Voice",'LabelType','attribute', ...
    'LabelDataType','logical','DefaultValue',false);
mx = signalLabelDefinition("Maximum",'LabelType','point', ...
    'LabelDataType','numeric');
rs = signalLabelDefinition("RanROI",'LabelType','ROI', ...
    'LabelDataType','categorical','Categories',["ROI" "other"]);
addLabelDefinitions(lss,[vc mx rs])
```

Label the signals:

- Label `'handel.mat'` and `'laughter.mat'` as having human voices.
- Use the `islocalmax` function to find the maximum of each signal. Label its location and value.
- Use the `randROI` on page 1-0 function to generate as many regions of length *N*/10 samples as can fit in a signal of length *N* given a minimum separation of *N*/6 samples between regions. Label their locations and assign them to the `ROI` category.

When labeling points and regions, convert sample values to time values. Subtract 1 to account for MATLAB® array indexing and divide by the sample rate.

```
kj = 1;
while hasdata(sds)

    [sig,info] = read(sds);
    fs = info.SampleRate;

    [~,fn] = fileparts(info.FileName);
    if fn=="handel" || fn=="laughter"
        setLabelValue(lss,kj,"Voice",true)
    end

    xm = find(islocalmax(sig,'MaxNumExtrema',1));
    setLabelValue(lss,kj,"Maximum",(xm-1)/fs,sig(xm))

    N = length(sig);
    rois = randROI(N,round(N/10),round(N/6));
    setLabelValue(lss,kj,"RanROI",(rois-1)/fs,repelem("ROI",size(rois,1)))

    kj = kj+1;

end
```

Verify that only two signals contain voices.

```
countLabelValues(lss,"Voice")
```

```
ans=2×3 table
    Voice    Count    Percent
    _____    _____    _______

    false      4      66.667 
    true       2      33.333 
```

Verify that two signals have a maximum amplitude of 1.

```
countLabelValues(lss,"Maximum")
```

```
ans=5×4 table
           Maximum             Count    Percent    MemberCount
    ______________________    _____    _______    ___________

    0.80000000000000004441      1      16.667          1      
    0.89113331915798421612      1      16.667          1      
    0.94730769230769229505      1      16.667          1      
    1                           2      33.333          2      
    1.0575668990330560071       1      16.667          1      
```

Verify that each signal has four nonoverlapping random regions of interest.

```
countLabelValues(lss,"RanROI")
```

```
ans=2×4 table
    RanROI    Count    Percent    MemberCount
    ______    _____    _______    ___________

    ROI        24        100           6     
    other       0          0           0     
```

Create two datastores with the data in the labeled signal set:

- The `signalDatastore` (Signal Processing Toolbox) object `sd` contains the signal data.
- The `arrayDatastore` object `ld` contains the labeling information. Specify that you want to include the information corresponding to all the labels you created.

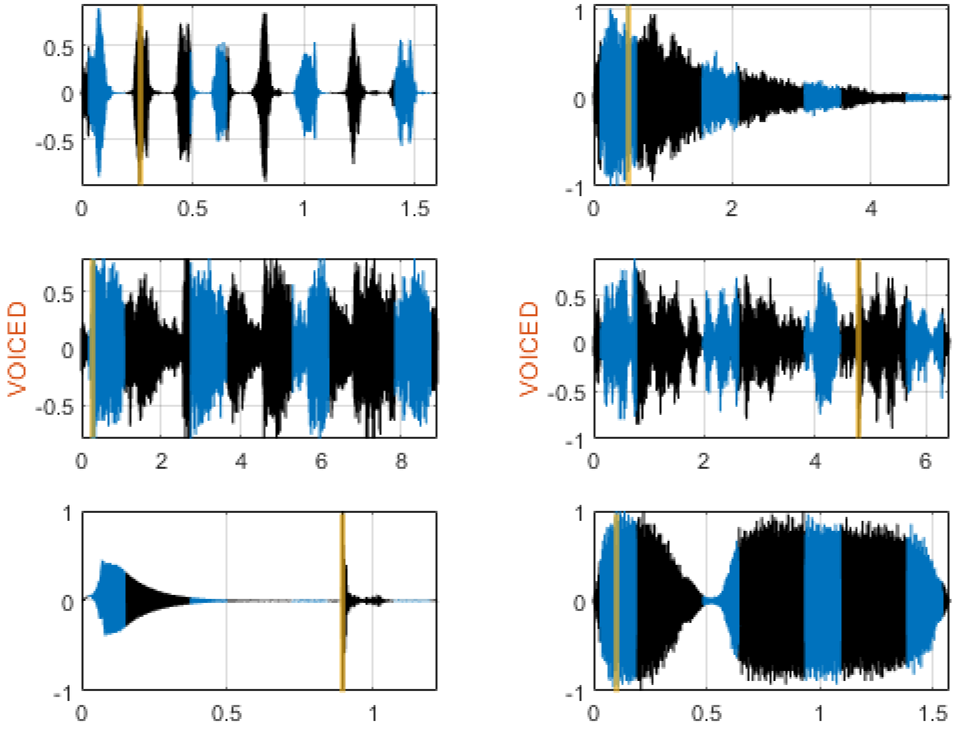
```
[sd,ld] = createDatastores(lss,["Voice" "RanROI" "Maximum"]);
```

Use the information in the datastores to plot the signals and display their labels.

- Use a `signalMask` (Signal Processing Toolbox) object to highlight the regions of interest in blue.
- Plot yellow lines to mark the locations of the maxima.

- Add a red axis label to the signals that contain human voices.

```
tiledlayout flow

while hasdata(sd)

    [sg,nf] = read(sd);

    lbls = read(ld);

    nexttile

    msk = signalMask(lbls{:}.RanROI{:},'SampleRate',nf.SampleRate);
    plotsigroi(msk,sg)
    colorbar off
    xlabel('')

    xline(lbls{:}.Maximum{:}.Location, ...
        'LineWidth',2,'Color','#EDB120')

    if lbls{:}.Voice{:}
        ylabel('VOICED','Color','#D95319')
    end

end
```



```
function roilims = randROI(N,wid,sep)
```

```
num = floor((N+sep)/(wid+sep));
hq = histcounts(randi(num+1,1,N-num*wid-(num-1)*sep),(1:num+2)-1/2);
roilims = (1 + (0:num-1)*(wid+sep) + cumsum(hq(1:num)))' + [0 wid-1];

end
```

See Also

Apps
Signal Labeler

Objects
signalLabelDefinition | signalMask

Introduced in R2018b

laurentMatrix

Create Laurent matrix

Description

Use the `laurentMatrix` object to create a matrix with `laurentPolynomial` elements. You can perform mathematical operations on the matrices.

Creation

Syntax

```
lmat = laurentMatrix
lmat = laurentMatrix(Elements=entries)
```

Description

`lmat = laurentMatrix` creates a Laurent matrix that is a 2-by-2 identity matrix.

`lmat = laurentMatrix(Elements=entries)` creates a Laurent matrix with elements specified by the value of the `Elements` property.

Properties

`Elements` — Laurent matrix elements
2-by-2 identity matrix (default) | cell array

Laurent matrix elements, specified as a cell array that has at most two rows and two columns. You can specify an element as a real-valued scalar or `laurentPolynomial` object. `laurentMatrix` converts all real-valued scalars into `laurentPolynomial` objects internally.

Example: `lmat = laurentMatrix(Elements={2,4;lpA,lpB})` creates a 2-by-2 Laurent matrix, where `lpA` and `lpB` are both `laurentPolynomial` objects.

Object Functions

Specific to laurentMatrix

ctranspose	Laurent matrix transpose
det	Laurent matrix determinant
dispMat	Display Laurent matrix
inverse	Laurent matrix inverse

Common to laurentMatrix and laurentPolynomial

dyaddown	Dyadic downsampling of Laurent polynomial or Laurent matrix
dyadup	Dyadic upsampling of Laurent polynomial or Laurent matrix

eq	Laurent polynomials or Laurent matrices equality test
plus	Laurent polynomial or Laurent matrix addition
minus	Laurent polynomial or Laurent matrix subtraction
mtimes	Laurent polynomial or Laurent matrix multiplication
reflect	Laurent polynomial or Laurent matrix reflection
uminus	Unary minus for Laurent polynomial or Laurent matrix

Examples

Create Laurent Matrix

Create two Laurent polynomials:

- $a(z) = 2 + 4z^{-1} + 6z^{-2}$
- $b(z) = z^2 + 3z + 5$

```
lpA = laurentPolynomial(Coefficients=[2 4 6]);
lpB = laurentPolynomial(Coefficients=[1 3 5],MaxOrder=2);
```

Create the Laurent matrix $\begin{bmatrix} -1 & a(z) \\ b(z) & 7 \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={-1 lpA; lpB 7});
```

Display the elements of the matrix.

```
for j=1:2
    for k=1:2
        fprintf("===================\nlmat(%d,%d):\n",j,k);
        element = lmat.Elements{j,k}
    end
end
```

```
===================
lmat(1,1):

element = 
  laurentPolynomial with properties:

    Coefficients: -1
        MaxOrder: 0


===================
lmat(1,2):

element = 
  laurentPolynomial with properties:

    Coefficients: [2 4 6]
        MaxOrder: 0


===================
lmat(2,1):
```

```
element = 
  laurentPolynomial with properties:

    Coefficients: [1 3 5]
        MaxOrder: 2


===================
lmat(2,2):

element = 
  laurentPolynomial with properties:

    Coefficients: 7
        MaxOrder: 0
```

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The `dispMat` object function is not supported.

See Also

`laurentPolynomial` | `liftingScheme`

Introduced in R2021b

laurentPolynomial

Create Laurent polynomial

Description

Use the `laurentPolynomial` object to create a Laurent polynomial with real-valued polynomial coefficients. You can specify the maximum order of the polynomial. You can perform mathematical and logical operations on Laurent polynomials. You can also create a lifting scheme associated with a pair of Laurent polynomials.

Creation

Syntax

```
lpoly = laurentPolynomial
lpoly = laurentPolynomial(Name=Value)
```

Description

`lpoly = laurentPolynomial` creates the constant Laurent polynomial, where the constant is equal to 1 and the maximum order is equal to 0.

`lpoly = laurentPolynomial(Name=Value)` creates a Laurent polynomial with “Properties” on page 1-784 specified by name-value arguments. For example, `laurentPolynomial(MaxOrder=2)` creates a Laurent polynomial with maximum order equal to 2. You can specify multiple name-value arguments.

Properties

`Coefficients` — Laurent polynomial coefficients
`1` (default) | real-valued vector

Laurent polynomial coefficients, specified as a real-valued vector. If k is the length of the vector C, then `lpoly = laurentPolynomial(Coefficients=C)` represents the Laurent polynomial

$$\mathrm{lpoly}(z) = \sum_{m=1}^{k} C(m) z^{1-m} .$$

Example: If `C = [4 3 2 1]`, then `P = laurentPolynomial(Coefficients=C)` represents the Laurent polynomial $\mathrm{P}(z) = 4 + 3z^{-1} + 2z^{-2} + z^{-3}$.

Data Types: `double`

`MaxOrder` — Maximum order
`0` (default) | integer

Maximum order of the Laurent polynomial, specified as an integer. If k is the length of the vector C and d is an integer, then `lpoly = laurentPolynomial(Coefficients=C,MaxOrder=d)` represents the Laurent polynomial

$$\mathrm{lpoly}(z) = \sum_{m=1}^{k} C(m) z^{d-m+1} .$$

Example: If `C = [2 4 6 8]`, then `P = laurentPolynomial(Coefficients=C,MaxOrder=1)` represents the Laurent polynomial $\mathrm{P}(z) = 2z + 4 + 6z^{-1} + 8z^{-2}$.

Data Types: `double`

Object Functions

Specific to laurentPolynomial

degree	Degree of Laurent polynomial
euclid	Euclidean algorithm for Laurent polynomials
polyphase	Polyphase components of Laurent polynomial
mpower	Laurent polynomial exponentiation
horzcat	Horizontal concatenation of Laurent polynomials
vertcat	Vertical concatenation of Laurent polynomials
lp2filters	Laurent polynomials to filters
lp2LS	Laurent polynomials to lifting steps and normalization factors
ne	Laurent polynomials inequality test
rescale	Rescale Laurent polynomial

Common to laurentPolynomial and laurentMatrix

dyaddown	Dyadic downsampling of Laurent polynomial or Laurent matrix
dyadup	Dyadic upsampling of Laurent polynomial or Laurent matrix
eq	Laurent polynomials or Laurent matrices equality test
plus	Laurent polynomial or Laurent matrix addition
minus	Laurent polynomial or Laurent matrix subtraction
mtimes	Laurent polynomial or Laurent matrix multiplication
reflect	Laurent polynomial or Laurent matrix reflection
uminus	Unary minus for Laurent polynomial or Laurent matrix

Examples

Basic Mathematical Operations Applied to Laurent Polynomials

Create three Laurent polynomials:

- $a(z) = 1 + z^{-1}$
- $b(z) = z^2 + 3z + z^{-1}$
- $c(z) = z^3 + 3z^2 + 5z + 7$

```
a = laurentPolynomial(Coefficients=[1 1])
```

```
a = 
  laurentPolynomial with properties:
    Coefficients: [1 1]
        MaxOrder: 0
```

```
b = laurentPolynomial(Coefficients=[1 3 0 1],MaxOrder=2)
```

```
b = 
  laurentPolynomial with properties:

    Coefficients: [1 3 0 1]
        MaxOrder: 2
```

```
c = laurentPolynomial(Coefficients=[1 3 5 7],MaxOrder=3)
```

```
c = 
  laurentPolynomial with properties:

    Coefficients: [1 3 5 7]
        MaxOrder: 3
```

Addition

Add the two polynomials $a(z)$ and $b(z)$. Use the helper function `helperPrintLaurent` to print the result in algebraic form.

```
polySum = plus(a,b)
```

```
polySum = 
  laurentPolynomial with properties:

    Coefficients: [1 3 1 2]
        MaxOrder: 2
```

```
res = helperPrintLaurent(polySum);
disp(res)
```

```
z^(2) + 3*z + 1 + 2*z^(-1)
```

Add 2 to $b(z)$.

```
consSum = b+2;
res = helperPrintLaurent(consSum);
disp(res)
```

```
z^(2) + 3*z + 2 + z^(-1)
```

Subtraction

Subtract $a(z)$ from $b(z)$.

```
polyDiff = minus(b,a);
res = helperPrintLaurent(polyDiff);
disp(res)
```

```
z^(2) + 3*z - 1
```

Subtract $a(z)$ from 1.

```
consDiff = 1-a;
res = helperPrintLaurent(consDiff);
disp(res)
```

```
- z^(-1)
```

Multiplication

Multiply $a(z)$ and $b(z)$.

```
polyProd = mtimes(a,b);
res = helperPrintLaurent(polyProd);
disp(res)
```

```
z^(2) + 4*z + 3 + z^(-1) + z^(-2)
```

Compute $a(z)c(z) - b(z)$.

```
polyProd2 = a*c-b;
res = helperPrintLaurent(polyProd2);
disp(res)
```

```
z^(3) + 3*z^(2) + 5*z + 12 + 6*z^(-1)
```

To multiply a Laurent polynomial by a constant, use the `rescale` function.

```
consProd = rescale(b,7);
res = helperPrintLaurent(consProd);
disp(res)
```

```
7*z^(2) + 21*z + 7*z^(-1)
```

Exponentiation

Raise $a(z)$ to the fourth power.

```
polyPow = mpower(a,4);
res = helperPrintLaurent(polyPow);
disp(res)
```

```
1 + 4*z^(-1) + 6*z^(-2) + 4*z^(-3) + z^(-4)
```

Compute $b^2(z) - c(z)$.

```
polyPow2 = b^2-c;
res = helperPrintLaurent(polyPow2);
disp(res)
```

```
z^(4) + 5*z^(3) + 6*z^(2) - 3*z - 1 + z^(-2)
```

Properties of Laurent Polynomials

Create two Laurent polynomials:

- $a(z) = z - 1$

- $b(z) = -2z^3 + 6z^2 - 7z + 2$

```
a = laurentPolynomial(Coefficients=[1 -1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[-2 6 -7 2],MaxOrder=3);
```

Reflection

Obtain the reflection of $b(z)$.

```
br = reflect(b);
res = helperPrintLaurent(br);
disp(res)
```

```
2 - 7*z^(-1) + 6*z^(-2) - 2*z^(-3)
```

Unary Minus

Confirm the sum of $b(z)$ and its unary negation is equal to 0.

```
b+uminus(b)
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 0
        MaxOrder: 0
```

Degree

Multiply $a(z)$ and $b(z)$. Confirm the degree of the product is equal to the sum of the degrees of $a(z)$ and $b(z)$.

```
ab = a*b;
degree(ab)
```

```
ans = 4
```

```
degree(a)+degree(b)
```

```
ans = 4
```

Exponentiation

Raise $a(z)$ to the third power. Confirm the result is not equal to $b(z)$.

```
a3 = a^3;
a3 ~= b
```

```
ans = logical
   1
```

Rescale

Confirm $a(z)$ raised to the third power is equal to $-b(z)/2 - z/2$.

```
zt = laurentPolynomial(Coefficients=[-1/2],MaxOrder=1);
b2 = rescale(b,-1/2)+zt;
eq(a3,b2)
```

```
ans = logical
   1
```

Dyadic Operations

Create the Laurent polynomial $c(z) = \sum_{k = -3}^{4} (-1)^k k z^k$. Obtain the degree of $c(z)$.

```
cfs = (-1).^(-3:4).*(-3:4);
c = laurentPolynomial(Coefficients=fliplr(cfs),MaxOrder=4);
res = helperPrintLaurent(c);
disp(res)
```

```
4*z^(4) - 3*z^(3) + 2*z^(2) - z + z^(-1) - 2*z^(-2) + 3*z^(-3)
```

```
degree(c)
```

```
ans = 7
```

Obtain the dyadic upsampling and downsampling of $c(z)$. Obtain the degree of both polynomials.

```
dUp = dyadup(c)
```

```
dUp = 
  laurentPolynomial with properties:

    Coefficients: [4 0 -3 0 2 0 -1 0 0 0 1 0 -2 0 3]
        MaxOrder: 8
```

```
degree(dUp)
```

```
ans = 14
```

```
dDown = dyaddown(c)
```

```
dDown = 
  laurentPolynomial with properties:

    Coefficients: [4 2 0 -2]
        MaxOrder: 2
```

```
degree(dDown)
```

```
ans = 3
```

Extended Capabilities

C/C++ Code Generation

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

See Also

`laurentMatrix` | `liftingScheme`

Introduced in R2021b

laurmat

(To be removed) Laurent matrices constructor

Note `laurmat` will be removed in a future release. Use `laurentMatrix` instead. For more information, see “Compatibility Considerations”.

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
```

Compatibility Considerations

`laurmat` will be removed

Not recommended starting in R2021b

`laurmat` will be removed in a future release. Use `laurentMatrix` instead.

Functionality	Result	Use Instead	Compatibility Considerations
`M = laurmat(V)`	Still runs	`M = laurentMatrix(Elements=V)`	You can also perform mathematical operations on the matrices.

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

`laurentMatrix`

Introduced before R2006a

laurpoly

(To be removed) Laurent polynomials constructor

Note `laurpoly` will be removed in a future release. Use `laurentPolynomial` instead. For more information, see “Compatibility Considerations”.

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)
```

Compatibility Considerations

`laurpoly` will be removed

Not recommended starting in R2021b

`laurpoly` will be removed in a future release. Use `laurentPolynomial` instead.

Functionality	Result	Use Instead	Compatibility Considerations
`P = laurpoly(C,d)` and `P = laurpoly(C,'dmax',d)`	Still runs	`P = laurentPolynomial(Coefficients=C,MaxOrder=d)`	You can also create a lifting scheme associated with a pair of Laurent polynomials.
`P = laurpoly(C,'dmin',d)`	Still runs	`P = laurentPolynomial(Coefficients=C,MaxOrder=`*N*`+d-1)` where *N* is the length of `C`.	

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

`laurentPolynomial`

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.

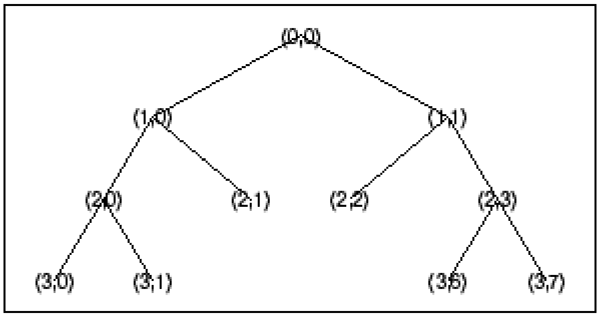
`N = leaves(T,'dp')` returns a matrix `N`, which contains the depths and positions of terminal nodes.

`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

tnodes | noleaves

Introduced before R2006a

liftingScheme

Create lifting scheme for lifting wavelet transform

Description

Use the `liftingScheme` object to create a lifting scheme that you can efficiently apply to data.

Creation

Syntax

```
lscheme = liftingScheme
lscheme = liftingScheme(Name,Value)
```

Description

`lscheme = liftingScheme` creates the lifting scheme for the `'lazy'` wavelet with normalization set to 1.

`lscheme = liftingScheme(Name,Value)` creates a lifting scheme with "Properties" on page 1-797 specified by name-value pairs. Enclose the property name in quotes. You can create a lifting scheme using one of the following syntaxes:

- `lscheme = liftingScheme('Wavelet',wname)`
- `lscheme = liftingScheme('CustomLowpassFilter',filter)`
- `lscheme = liftingScheme('LiftingSteps',liftingSteps,'NormalizationFactors',normFactors)`

Properties

`Wavelet` — Orthogonal or biorthogonal wavelet
`'lazy'` (default) | `'haar'` | `'db1'` | `'db2'` | ...

Orthogonal or biorthogonal wavelet associated with the lifting scheme, specified as one of these.

Wavelet Family	**Wavelet**
Daubechies	`'lazy'`, `'haar'`, `'db1'`, `'db2'`, `'db3'`, `'db4'`, `'db5'`, `'db6'`, `'db7'`, and `'db8'`
Symlet	`'sym2'`, `'sym3'`, `'sym4'`, `'sym5'`, `'sym6'`, `'sym7'`, and `'sym8'`

Wavelet Family	Wavelet
Cohen-Daubechies-Feauveau	'cdf1.1', 'cdf1.3', 'cdf1.5', 'cdf2.2', 'cdf2.4', 'cdf2.6', 'cdf3.1', 'cdf3.3', 'cdf3.5', 'cdf4.2', 'cdf4.4', 'cdf4.6', 'cdf5.1', 'cdf5.3', 'cdf5.5', 'cdf6.2', 'cdf6.4', and 'cdf6.6'
Coiflet	'coif1', and 'coif2'
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', 'bs3', and '9.7'
Reverse Biorthogonal	'rbs3', 'r9.7', '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', and 'rbio6.8'

Example: `lscheme = liftingScheme('Wavelet','bior3.7')` creates the lifting scheme associated with the `'bior3.7'` biorthogonal wavelet.

CustomLowpassFilter — Lowpass filters

cell array

Lowpass filters associated with the lifting scheme, specified as a cell array.

- To create a lifting scheme associated with an orthogonal wavelet, set `CustomLowpassFilter` to `{LoD}`, where `LoD` is the lowpass filter associated with wavelet.
- To create a lifting scheme associated with a biorthogonal wavelet, set `CustomLowpassFilter` to `{LoPrimal,LoDual}`, where `LoPrimal` and `LoDual` are the lowpass filters associated with the biorthogonal wavelet.

When you specify filter coefficients, the `Wavelet` property is automatically set to `'custom'`.

Example: `lscheme = liftingScheme('CustomLowpassFilter',{[sqrt(2)/2 sqrt(2)/2]})` creates a lifting scheme associated with the Haar wavelet.

Data Types: `single` | `double`

LiftingSteps — Lifting steps

`liftingStep` structure | array of `liftingStep` structures

Lifting steps associated with the lifting scheme, specified as an array of structures obtained from `liftingStep`. To create a lifting scheme using `LiftingSteps`, you must also set the `NormalizationFactors` property. When you set these two properties, the `Wavelet` property is automatically set to `'custom'`.

Example: `lscheme = liftingScheme('LiftingSteps',ELS,'NormalizationFactors',NF)` creates a lifting scheme using the `liftingStep` structures specified in `ELS` and factors specified in `NF`.

`NormalizationFactors` — Normalization factors

non-zero scalar | vector

Normalization factors associated with the lifting scheme, specified as K or `[K 1/K]`, where K is a non-zero scalar. The factor K specifies the diagonal elements of the 2-by-2 normalization matrix. If specified as a vector, the product of the vector elements must equal 1 to within precision.

To create a lifting scheme using `NormalizationFactors`, you must also set the `LiftingSteps` property. When you set these two properties, the `Wavelet` property is automatically set to `'custom'`.

Data Types: `double`

Object Functions

addlift	Add elementary lifting steps
deletelift	Delete elementary lifting steps
ls2filt	Extract wavelet filters from lifting scheme
disp	Display lifting scheme

Examples

Apply Lifting Scheme to Signal

Create the lifting scheme associated with the Haar wavelet.

```
lscheme = liftingScheme('Wavelet','haar')
```

```
lscheme = 
      Wavelet               : 'haar'
     LiftingSteps          : [2 × 1] liftingStep
     NormalizationFactors  : [1.4142 0.7071]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'predict'
    Coefficients: -1
        MaxOrder: 0

            Type: 'update'
    Coefficients: 0.5000
        MaxOrder: 0
```

Obtain the level 2 wavelet decomposition of a signal using the lifting scheme. Inspect the approximation and detail coefficients.

```
sig = 0:7;
[appC,detC]=lwt(sig,'LiftingScheme',lscheme,'Level',2);
appC
```

```
appC = 2×1

    3.0000
   11.0000
```

```
detC{1}
```

```
ans = 4×1

    0.7071
    0.7071
    0.7071
    0.7071
```

```
detC{2}
```

```
ans = 2×1

    2.0000
    2.0000
```

Obtain the inverse transform and demonstrate perfect reconstruction.

```
xrec = ilwt(appC,detC,'LiftingScheme',lscheme);
max(abs(xrec(:)-sig(:)))
```

```
ans = 2.6645e-15
```

Create Lifting Scheme Using Custom Lowpass Filter

Create a lifting scheme using the lowpass filters associated with the `db4` wavelet.

```
wv = 'db4';
[~,~,LoR,~] = wfilters(wv);
LS = liftingScheme('CustomLowpassFilter',{LoR});
```

Demonstrate Wavelet Orthogonality

Create the lifting scheme associated with the biorthogonal `bior2.2` wavelet.

```
lscheme = liftingScheme('Wavelet','bior2.2');
```

A wavelet with N vanishing moments is orthogonal to degree $N-1$ polynomials. The `bior2.2` wavelet has two vanishing moments. Create a signal by sampling a polynomial of degree 1.

```
sig = 1:16;
```

Apply the lifting scheme to the signal. Inspect the detail coefficients at the finest scale. The `bior2.2` wavelet is orthogonal to the degree 1 polynomial. Confirm that except for the nonzero coefficient at the boundary, the detail coefficients are zero.

```
[A,D] = lwt(sig,'LiftingScheme',lscheme);
D{1}
```

```
ans = 8×1

         0
         0
         0
         0
         0
         0
         0
    5.6569
```

Now create the lifting scheme associated with the Haar wavelet.

```
lschemeH = liftingScheme('Wavelet','haar');
```

Apply the lifting scheme to the signal. Confirm the detail coefficients are all nonzero. Because the Haar wavelet has only one vanishing moment, the wavelet is not orthogonal to the degree 1 polynomial.

```
[AH,DH] = lwt(sig,'LiftingScheme',lschemeH);
DH{1}
```

```
ans = 8×1

    0.7071
    0.7071
    0.7071
    0.7071
    0.7071
    0.7071
    0.7071
    0.7071
```

Compatibility Considerations

`CustomLowpassFilter` name-value argument in `liftingScheme` must be a cell array

Behavior changed in R2021b

Starting this release, to use a lowpass filter to create a lifting scheme associated with an orthogonal wavelet, you must specify `CustomLowpassFilter` as a cell array. If you specify `CustomLowpassFilter` as a vector, `liftingScheme` will generate an error.

To update your code, change instances of `'CustomLowpassFilter',lpass`, where `lpass` is the vector, to `'CustomLowpassFilter',{lpass}`.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The `disp` object function is not supported.

See Also

`liftingStep` | `lwt` | `ilwt` | `lwt2` | `ilwt2`

Introduced in R2021a

liftingStep

Create elementary lifting step

Syntax

```
Lstep = liftingStep
Lstep = liftingStep(Name,Value)
```

Description

`Lstep = liftingStep` returns an elementary lifting step as a structure array with default field values. You can add the lifting step to a `liftingScheme` object. For more information, see `addlift`.

`Lstep = liftingStep(Name,Value)` sets field values using name-value arguments. For example, `Lstep = liftingStep('Type','update')` creates a lifting step of type `'update'`. You can specify multiple name-value arguments. Enclose each field name in quotes.

Examples

Apply Lifting Scheme with User-Specified Lifting Steps

This example shows how to apply a lifting scheme with user-specified lifting steps to a signal.

Create two lifting steps. Concatenate the steps in a single array.

```
els1 = liftingStep('Type','update',...
    'Coefficients',[-sqrt(3) 1],'MaxOrder',0);

els2 = liftingStep('Type','predict',...
    'Coefficients',[1 sqrt(3)/4+(sqrt(3)-2)/4],'MaxOrder',1);

stepArray = [els1;els2];
```

Specify normalization constants.

```
K = [(sqrt(3)+1)/sqrt(2) (sqrt(3)-1)/sqrt(2)];
```

Create a lifting scheme using the array of lifting steps and the normalization constants.

```
lScheme = liftingScheme('LiftingSteps',stepArray,'NormalizationFactors',K)
```

```
lScheme = 
      Wavelet               : 'custom'
     LiftingSteps          : [2 × 1] liftingStep
     NormalizationFactors  : [1.9319 0.5176]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'update'
    Coefficients: [-1.7321 1]
```

```
        MaxOrder: 0

            Type: 'predict'
    Coefficients: [1 0.3660]
        MaxOrder: 1
```

Create a signal. Apply the lifting scheme to the signal.

```
sig = 0:20;
[ca,cd] = lwt(sig,'LiftingScheme',lScheme);
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ls = liftingStep('MaxOrder',2,'Type','update','Coefficients',[1 2 3])`

Type — Type of lifting step
`'predict'` | `'update'`

Type of elementary lifting step, specified as `'predict'` or `'update'`.

Data Types: `char` | `string`

`Coefficients` — Laurent polynomial coefficients
vector

Laurent polynomial coefficients that correspond to the z-transform of the lifting filter, specified as a real-valued vector. The order of the first element of `Coefficients` is `MaxOrder`.

Data Types: `single` | `double`

`MaxOrder` — Maximum order
`0` (default) | integer

Maximum order of the Laurent polynomial coefficient, specified as an integer.

Data Types: `double`

Output Arguments

`Lstep` — Elementary lifting step
structure array

Elementary lifting step, returned as a structure. `Lstep` has three fields:

- `Type` — Type of lifting step, returned as a character array.
- `Coefficients` — Laurent polynomial coefficients, returned as a real-valued vector.

- `MaxOrder` — Maximum order of the Laurent polynomial, returned as an integer.

Data Types: `struct`

Extended Capabilities

C/C++ Code Generation

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

See Also

`liftingScheme` | `lwt` | `ilwt` | `lwtcoef` | `lwt2` | `ilwt2` | `lwtcoef2`

Introduced in R2021a

liftfilt

Apply elementary lifting steps on filters

Syntax

```
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,LiftingSteps=ELS)
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,NormalizationFactor=NF)
liftfilt( ___ )
```

Description

`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,LiftingSteps=ELS)` returns the four filters obtained by adding an array of elementary lifting steps (`ELS`) starting from the two filters `LoD` and `LoR`.

`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,NormalizationFactor=NF)` scales the filters by the normalization factor `NF`.

`liftfilt( ___ )` with no output arguments plots the successive biorthogonal pairs. A scaling function and a wavelet comprise each pair.

Examples

Generate Biorthogonal Wavelet Filters From Haar Filters

This example shows how to obtain the `bior1.3` wavelet filters using Haar filters and elementary lifting steps.

Obtain the Haar lowpass decomposition and reconstruction filters.

```
[LoD,~,LoR,~] = wfilters("haar");
```

Use `liftingStep` to create two elementary lifting steps of type `update`. Create an array consisting of the two steps.

```
els1 = liftingStep(Type="update",...
    Coefficients=[0.125 -0.125],MaxOrder=0);
els2 = liftingStep(Type="update",...
    Coefficients=[0.125 -0.125],MaxOrder=1);
elsBoth = [els1;els2];
```

Apply the lifting steps to the Haar filters to obtain new filters.

```
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,LiftingSteps=elsBoth);
```

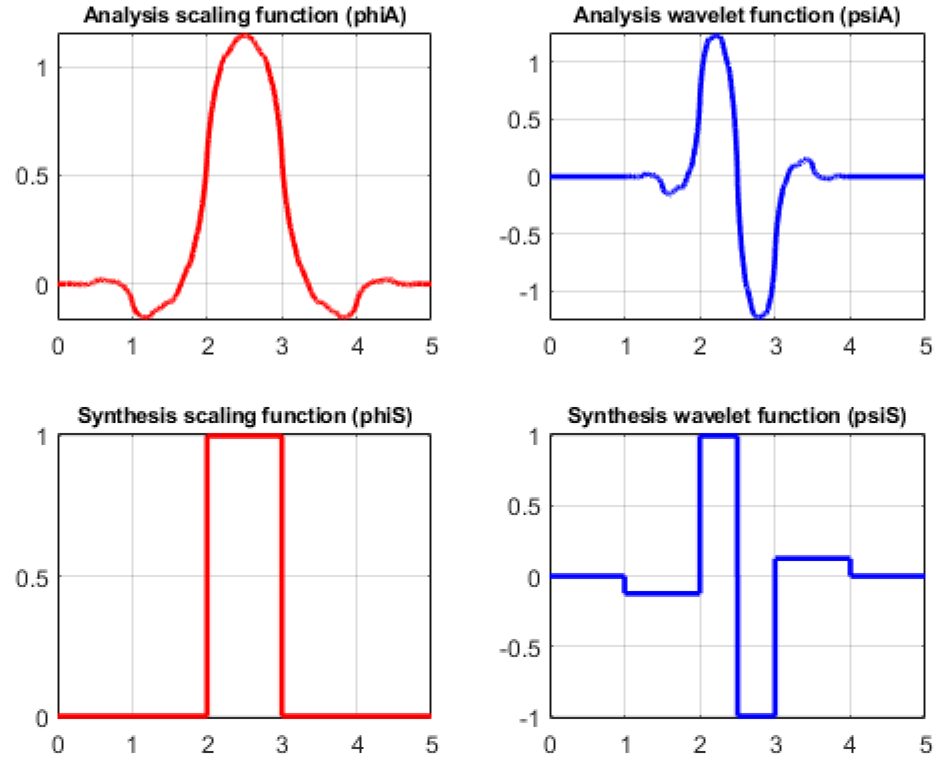
Obtain the `bior1.3` wavelet filters. Confirm that up to a sign change, the wavelet filters are equal to the filters `liftfilt` returns.

```
[LoDw,HiDw,LoRw,HiRw] = wfilters("bior1.3");
samewavelet = ...
isequal([LoDw,HiDw,LoRw,HiRw],[LoDN,-HiDN,LoRN,HiRN])
```

```
samewavelet = logical
   1
```

Use `liftfilt` to plot the successive biorthogonal pairs of scaling functions and wavelets.

```
liftfilt(LoD,LoR,LiftingSteps=elsBoth)
```



Input Arguments

`LoD`,`LoR` — Lowpass filters

real-valued vectors

Lowpass filters associated with a wavelet, specified as real-valued vectors. `LoD` is the lowpass decomposition filter. `LoR` is the lowpass reconstruction filters.

Example: For `[LoD,~,LoR,~] = wfilters("db4")`, `liftfilt(LoD,LoR,LiftingSteps=lsteps)` applies the elementary lifting steps specified in `lsteps` to the `db4` filters.

Data Types: `double`

`ELS` — Lifting steps

structure array

Lifting steps, specified as a structure array consisting of elementary lifting steps.

- If `liftingStep.Type="update"`, `LoR` and `HiD` are unchanged, where `HiD` is the associated highpass decomposition filter.
- If `liftingStep.Type="predict"`, `LoD` and `HiR` are unchanged, where `HiR` is the associated highpass decomposition filter.

Example: `liftfilt(LoD,LoR,LiftingSteps=ELS)` applies the elementary lifting steps specified in `lsteps` to the filters `LoD` and `LoR`.

Data Types: `struct`

`NF` — Normalization factor
nonzero scalar

Normalization factor, specified as a nonzero scalar.

Example: `[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,NF=2)` scales the filters by 2.

Data Types: `double`

Output Arguments

`LoDN,HiDN` — Decomposition filters
real-valued vectors

Decomposition filters, returned as a pair of real-valued vectors. `LoDN` and `HiDN` correspond to the lowpass and highpass filters, respectively.

Data Types: `double`

`LoRN,HiRN` — Reconstruction filters
real-valued vectors

Reconstruction filters, returned as a pair of real-valued vectors. `LoRN` and `HiRN` correspond to the lowpass and highpass filters, respectively.

Data Types: `double`

Compatibility Considerations

`liftfilt` input syntax has changed
Behavior changed in R2021b

The `liftfilt` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)`	Errors	`[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,LoR,LiftingSteps=ELS)`, where `ELS` is a structure array consisting of elementary lifting steps.	You can also scale the filters by a normalization factor. For more information about elementary lifting steps, see `liftingStep`.

Functionality	Result	Use Instead	Compatibility Considerations
`liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)`	Errors	NA	This syntax is no longer supported.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Plotting is not supported.

See Also

Functions
`liftingStep`

Objects
`laurentPolynomial`

Introduced in R2021b

liftwave

(To be removed) Lifting schemes

Note `liftwave` will be removed in a future release. Use `liftingScheme`. For more information, see “Compatibility Considerations”.

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 `lwt2` and `ilwt2`.

`LS = liftwave(WNAME,'Int2Int')` performs an integer-to-integer wavelet transform. Using `'Int2Int'` produces an `LS` such that when you use `[CA,CH,CV,CD] = lwt2(X,LS)` or `Y = lwt2(X,LS)` and `X` is an array of integers, the resulting `CA`, `CH`, `CV`, `CD`, and `Y` are matrices of integers. If you omit `'Int2Int'` then `lwt2` produces matrices 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'`

`'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]              []
};
```

Compatibility Considerations

`liftwave` will be removed

Not recommended starting in R2021a

`liftwave` will be removed in a future release. Use `liftingScheme`.

Functionality	Result	Use Instead	Compatibility Considerations
`LS = liftwave(WNAME)`	Still runs	`LS = liftingScheme('Wavelet',WNAME)`	You can also use `liftingScheme` to create a lifting scheme by specifying lowpass filter coefficients or customized lifting steps.

`LS = liftwave(WNAME,'Int2Int')`	Still runs	`LS = liftingScheme('Wavelet',WNAME)`
		`[CA,CD] = lwt(X,'LiftingScheme',LS,'Int2Int',true)`
			To preserve integer-valued data, set the `Int2Int` name-value pair of the functions `lwt` or `lwt2` to `true`.

See Also

`liftingScheme`

Introduced before R2006a

littlewoodPaleySum

Littlewood-Paley sum

Syntax

```
lpsum = littlewoodPaleySum(sf)
lpsum = littlewoodPaleySum(sf,fb)
[lpsum,f] = littlewoodPaleySum( ___ )
```

Description

`lpsum = littlewoodPaleySum(sf)` returns the Littlewood-Paley sum for the scattering filter banks in `sf`, the wavelet time scattering network. `lpsum` is an *M*-by-*L* matrix, where *M* is the number of elements in the Fourier transform of the scattering filters, and *L* is the number of scattering filter banks. The columns of `lpsum` are ordered by the position of the filter bank in the scattering network. For example, the first column of `lpsum` corresponds to the filter bank used for the first-order scattering coefficients.

Since the scattering transform is contractive, the Littlewood-Paley sums will not exceed one.

`lpsum = littlewoodPaleySum(sf,fb)` returns the Littlewood-Paley sum for the specified filter bank `fb` in `sf`. The argument `fb` is a positive integer between 1 and the number of filter banks in `sf` inclusive. The number of filter banks in `sf` is equal to the number of specified `QualityFactors` in `sf`.

`[lpsum,f] = littlewoodPaleySum( ___ )` returns the frequencies for the Littlewood-Paley sum. If you specify a sampling frequency in `sf`, `f` is in hertz. If you do not specify a sampling frequency, `f` is in cycles/sample. You can use these output arguments with any of the input syntaxes shown previously.

Examples

Plot Littlewood-Paley Sum

Create a wavelet time scattering network with three filter banks for data sampled at 25 Hz.

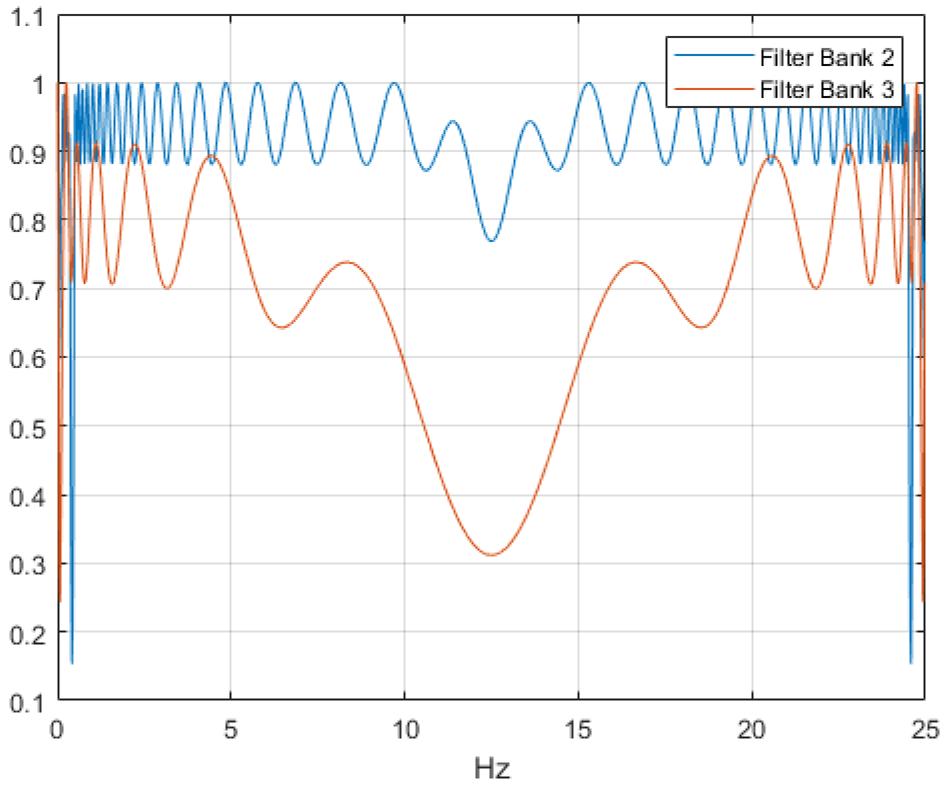
```
sf = waveletScattering('QualityFactors',[8 4 1],...
    'SamplingFrequency',25)
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 20.4800
        QualityFactors: [8 4 1]
              Boundary: 'periodic'
     SamplingFrequency: 25
             Precision: 'double'
    OversamplingFactor: 0
```

```
          OptimizePath: 0
```

Plot the Littlewood-Paley sums for the second and third filter banks. Note that the sums do not exceed 1. This shows the filters have been normalized so that the scattering transform is contractive.

```
[lpsum,f] = littlewoodPaleySum(sf);
plot(f,lpsum(:,2:3))
grid on
legend('Filter Bank 2','Filter Bank 3')
xlabel('Hz')
```



Input Arguments

`sf` — Wavelet time scattering network

`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

`fb` — Filter bank index

positive integer

Filter bank index in the wavelet time scattering network, specified as a positive integer between 1 and the number of filter banks in `sf` inclusive. The number of filter banks in `sf` is equal to the number of specified `QualityFactors` in `sf`.

Data Types: `double`

Output Arguments

`lpsum` — Littlewood-Paley sum

real-valued matrix

Littlewood-Paley sum for the filter banks in the scattering network `sf`, returned as a real-valued matrix. `lpsum` is an *M*-by-*L* matrix, where *M* is the number of elements in the Fourier transform of the scattering filters and *L* is the number of scattering filter banks. For example, the first column of `lpsum` corresponds to the filter bank used for the first-order scattering coefficients.

`f` — Frequencies

real-valued vector

Frequencies for the Littlewood-Paley sum, returned as a real-valued vector. If you specify a sampling frequency in `sf`, `f` is in hertz. If you do not specify a sampling frequency, `f` is in cycles/sample.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`waveletScattering`

Introduced in R2018b

littlewoodPaleySum

Littlewood-Paley sum

Syntax

```
lpsum = littlewoodPaleySum(sf)
lpsum = littlewoodPaleySum(sf,fb)
[lpsum,f] = littlewoodPaleySum( ___ )
```

Description

`lpsum = littlewoodPaleySum(sf)` returns the Littlewood-Paley sum for the 2-D filter banks in the 2-D wavelet scattering network `sf`. `lpsum` is an *M*-by-*N*-by-*Nfb* matrix, where *M*-by-*N* is the matrix size of the padded filters and *Nfb* is the number of filter banks.

Since the scattering transform is contractive, the Littlewood-Paley sums do not exceed 1.

`lpsum = littlewoodPaleySum(sf,fb)` returns the Littlewood-Paley sum for the specified filter banks `fb`. `fb` is a positive integer or vector of positive integers between 1 and `numfilterbanks(sf)` inclusive. `lpsum` is an *M*-by-*N*-by-*L* matrix, where *L* is the number of unique elements in `fb`.

`[lpsum,f] = littlewoodPaleySum( ___ )` returns the spatial frequencies for the Littlewood-Paley sum. `f` is a two-column matrix with the first column containing the spatial frequencies in the *x*-direction and the second column containing the spatial frequencies in the *y*-direction.

Examples

Littlewood-Paley Sum of Image Scattering Network

This example shows how to obtain and display the Littlewood-Paley sum of an image scattering network.

Create a scattering network with two filter banks and quality factors of 2 and 1, respectively.

```
sf = waveletScattering2('QualityFactors',[2 1]);
```

Obtain the Littlewood-Paley sums and spatial frequencies of the two filter banks. Display the maximum value of the sums. Since the scattering transform is contractive, the sums do not exceed 1.

```
[lpsum,f] = littlewoodPaleySum(sf);
max(max(lpsum(:,:,1)))
```

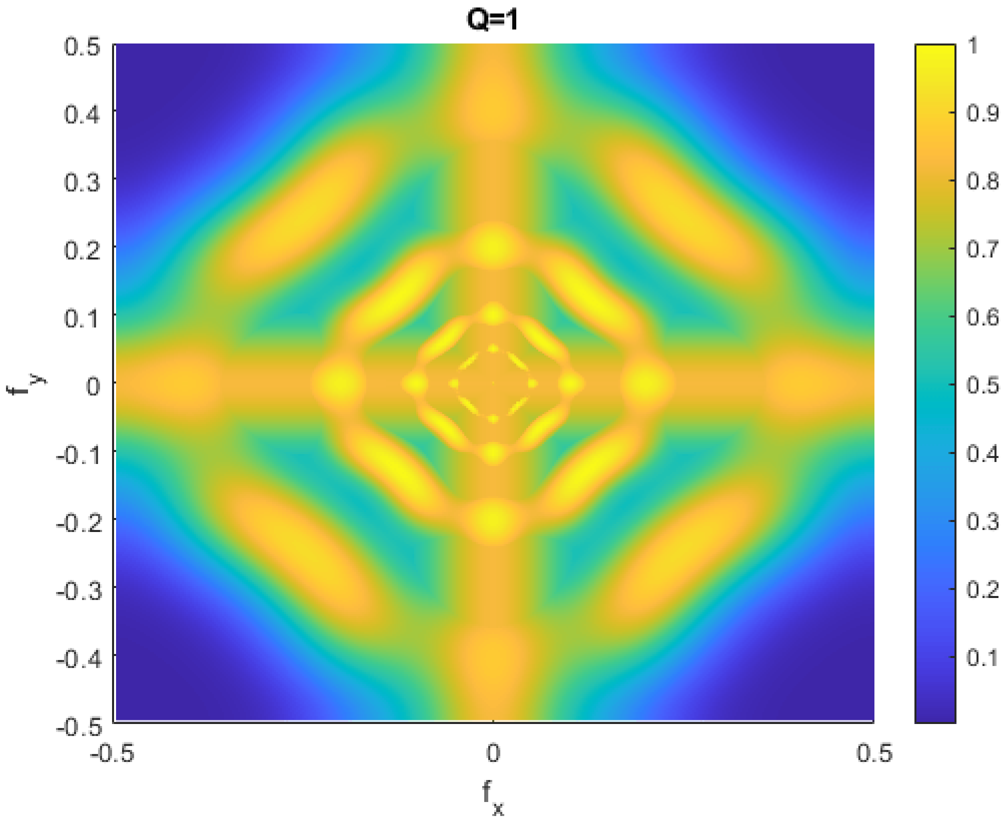
```
ans = 1.0000
```

```
max(max(lpsum(:,:,2)))
```

```
ans = 1.0000
```

Display the Littlewood-Paley sum of the second filter bank with the zero frequency centered. Note the 2-D Morlet filter bank used in the scattering transform is not designed to capture the highest spatial frequencies jointly in the *x*- and *y*-directions.

```
f(f>1/2) = f(f>1/2)-1;
surf(fftshift(f(:,1)),fftshift(f(:,2)),fftshift(lpsum(:,:,2)))
shading interp
view(0,90)
xlabel('f_x')
ylabel('f_y')
colorbar
title('Q=1')
```



Input Arguments

`sf` — Wavelet image scattering network

`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

`fb` — Filter bank index

positive integer | vector of positive integers

Filter bank index in the image scattering network, specified as a positive integer or vector of positive integers between 1 and `numfilterbanks(sf)` inclusive. The number of filter banks in `sf` is equal to the number of specified `QualityFactors` in `sf`.

Output Arguments

`lpsum` — Littlewood-Paley sum

real-valued 3-D matrix

Littlewood-Paley sum for the filter banks in the image scattering network `sf`, returned as a real-valued 3-D matrix. `lpsum` is an *M*-by-*N*-by-*L* matrix, where *M*-by-*N* is the matrix size of the padded filters and *L* does not exceed the number of filter banks in `sf`.

`f` — Frequencies

real-valued two-column matrix

Frequencies for the Littlewood-Paley sum, returned as a real-valued two-column matrix. Frequencies are in cycles per pixel. The first column of `f` contains the spatial frequencies in the *x*-direction, and the second column contains the spatial frequencies in the *y*-direction. In this convention, the Fourier transform is 1-periodic in both Fourier variables.

See Also

`waveletScattering2`

Introduced in R2019a

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)
```

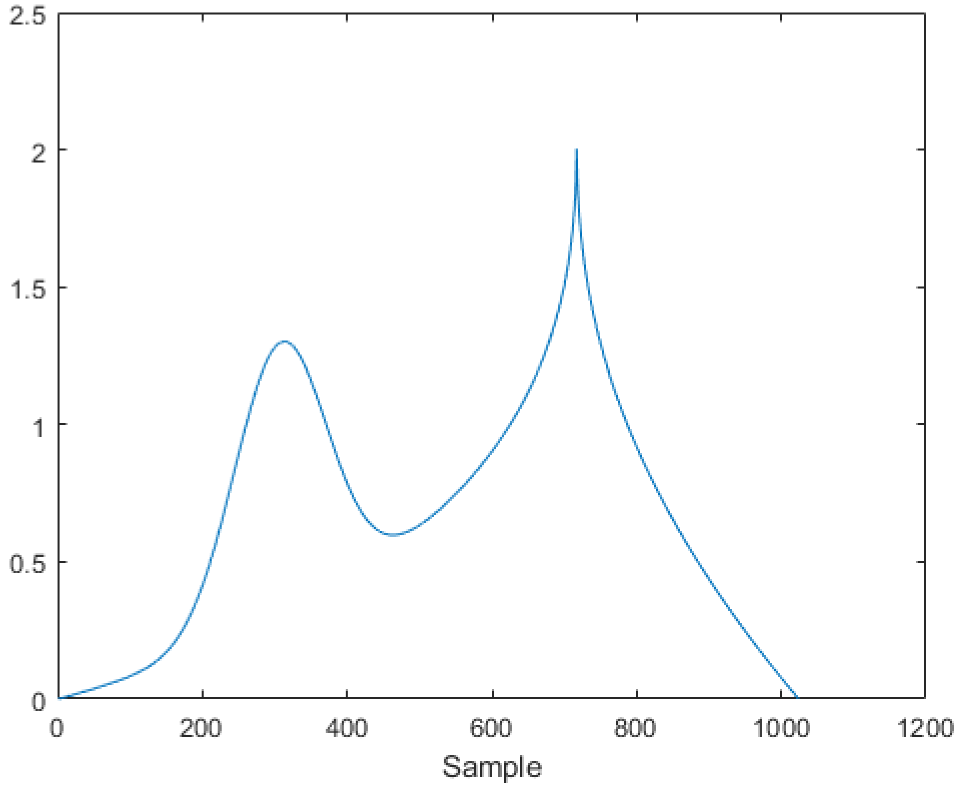
CWT Coefficient Moduli and Maxima Lines

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;
```

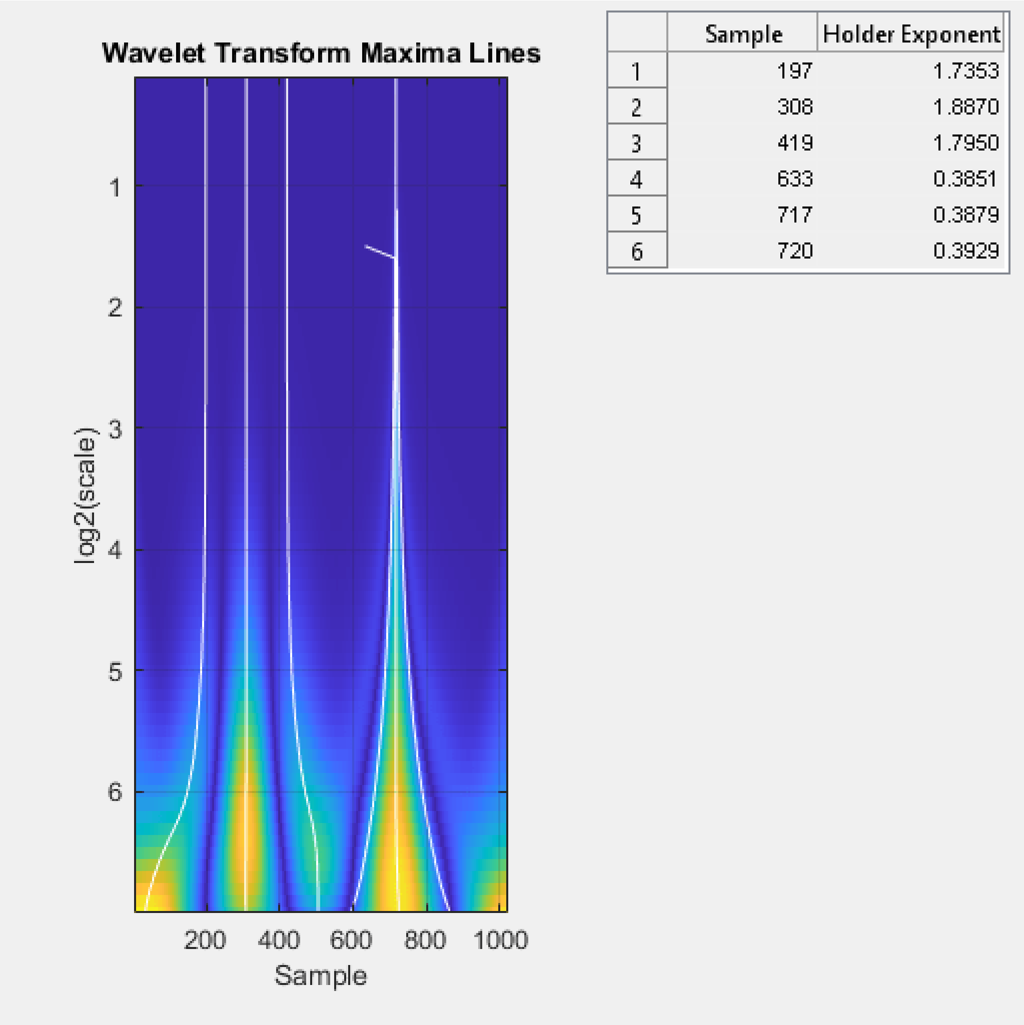
Plot the `cuspamax` signal and notice the shape of the signal near samples 300 and 700. The signal shows a cusp near sample 700.

```
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 exponents 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

log

Natural logarithm of scattering transform

Syntax

```
slog = log(sf,s)
ulog = log(sf,u)
xlog = log(sf,x)
```

Description

`slog = log(sf,s)` returns the natural logarithm of the scattering coefficients in the cell array `s`. `s` is the output of `scatteringTransform` and is a cell array of structure arrays with a `signals` field.

The precision of `slog` depends on the precision specified in the wavelet time scattering network `sf`.

`ulog = log(sf,u)` returns the natural logarithm of the scalogram coefficients in the cell array `u`. `u` is the output of `scatteringTransform` and is a cell array of structure arrays with a `coefficients` field.

The precision of `ulog` depends on the precision specified in the wavelet time scattering network `sf`.

`xlog = log(sf,x)` returns the natural logarithm of the 2-D matrix or 3-D array `x`. `x` is the output of `featureMatrix`.

The precision of `xlog` depends on the precision specified in the wavelet time scattering network `sf`.

Examples

Natural Logarithm of Scattering Coefficients

This example shows how to obtain the natural logarithm of scattering coefficients.

Load a noisy Doppler signal and create a wavelet time scattering network that can be used with the signal. Return the scattering coefficients.

```
load noisdopp
sf = waveletScattering('SignalLength',numel(noisdopp));
S = scatteringTransform(sf,noisdopp);
```

Calculate the natural logarithm of the scattering coefficients. Display the number of rows in the table containing the first-order scattering coefficients.

```
slog = log(sf,S);
coefOrder = 1;
display(['Number of rows: ',...
    num2str(size(S{coefOrder+1},1))])
```

```
Number of rows: 41
```

Choose a row from the first-order scattering coefficients table. Take the natural logarithm of the absolute value of the scattering coefficients in that row. Compare with the corresponding row in `slog` and confirm they are equal.

```
row = 23;
tmp1 = slog{coefOrder+1}.signals{row};
tmp2 = log(abs(S{coefOrder+1}.signals{row}));
disp(['Max Difference of Scattering Coefficients: ',...
    num2str(max(abs(tmp1(:)-tmp2(:))))])
```

```
Max Difference of Scattering Coefficients: 0
```

Input Arguments

`sf` — Wavelet time scattering network

`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

`s` — Scattering coefficients

cell array

Scattering coefficients, specified as a cell array of structure arrays. `s` is the output of `scatteringTransform` for the scattering network `sf`.

`u` — Scalogram coefficients

cell array

Scalogram coefficients, specified as a cell array of structure arrays. `u` is the output of `scatteringTransform` for the scattering network `sf`.

`x` — Scattering feature matrix

real-valued matrix | real-valued array

Scattering feature matrix, specified as a real-valued 2-D matrix or 3-D array. `x` is the output of `featureMatrix` for the scattering network `sf`.

Output Arguments

`slog` — Natural logarithm of scattering coefficients

cell array

Natural logarithm of scattering coefficients, returned as a cell array. The dimensions of `slog` are equal to the dimensions of `s`.

The precision of `slog` depends on the precision specified in the scattering network `sf`.

`ulog` — Natural logarithm of scalogram coefficients

cell array

Natural logarithm of scalogram coefficients, returned as a cell array. The dimensions of `ulog` are equal to the dimensions of `u`.

The precision of `ulog` depends on the precision specified in the scattering network `sf`.

xlog — Natural logarithm of scattering feature matrix
real-valued matrix | real-valued array

Natural logarithm of scattering feature matrix, returned as a real-valued matrix or array. The dimensions of `xlog` are equal to the dimensions of `x`.

The precision of `xlog` depends on the precision specified in the scattering network `sf`.

Algorithms

`log` returns the natural logarithm of the absolute value of the input argument.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`waveletScattering` | `scatteringTransform`

Introduced in R2018b

log

Natural logarithm of 2-D scattering transform

Syntax

```
slog = log(sf,s)
ulog = log(sf,u)
xlog = log(sf,x)
```

Description

`slog = log(sf,s)` returns the natural logarithm of the scattering coefficients in the cell array `s`. `s` is the output of `scatteringTransform` and is a cell array of structure arrays with an `images` field.

The precision of `slog` depends on the precision specified in the network `sf`.

`ulog = log(sf,u)` returns the natural logarithm of the scalogram coefficients in the cell array `u`. `u` is the output of `scatteringTransform` and is a cell array of structure arrays with a `coefficients` field.

The precision of `ulog` depends on the precision specified in the network `sf`.

`xlog = log(sf,x)` returns the natural logarithm of the 3-D matrix or 4-D tensor `x`. `x` is the output of `featureMatrix`.

The precision of `xlog` depends on the precision specified in the network `sf`.

Examples

Natural Logarithm of Scattering Coefficients

This example shows how to obtain the natural logarithm of scattering coefficients.

Load the `xbox` image. Create an image scattering network that can be applied to the image.

```
load xbox
sf = waveletScattering2('ImageSize',size(xbox),...
    'InvarianceScale',min(size(xbox)))
```

```
sf = 
  waveletScattering2 with properties:

              ImageSize: [128 128]
        InvarianceScale: 128
           NumRotations: [6 6]
         QualityFactors: [1 1]
              Precision: "single"
    OversamplingFactor: 0
           OptimizePath: 1
```

Obtain the scattering transform of the image and then the natural logarithm of the scattering coefficients. Display the number of rows in the table containing the first-order scattering coefficients.

```
S = scatteringTransform(sf,xbox);
Slog = log(sf,S);
coefOrder = 1;
display(['Number of rows: ',num2str(size(S{coefOrder+1},1))])
```

```
Number of rows: 30
```

Choose a row from the first-order scattering coefficients table. Take the natural logarithm of the absolute value of the scattering coefficients in that row. Compare with the corresponding row in `Slog` and confirm they are equal.

```
row = 11;
tmp1 = Slog{coefOrder+1}.images{row};
tmp2 = log(abs(S{coefOrder+1}.images{row}));
disp(['Max Difference of Scattering Coefficients: '...
    num2str(max(abs(tmp1(:)-tmp2(:))))])
```

```
Max Difference of Scattering Coefficients: 0
```

Input Arguments

sf — Wavelet image scattering network

waveletScattering2 object

Wavelet image scattering network, specified as a `waveletScattering2` object.

s — Scattering coefficients

cell array

Scattering coefficients, specified as a cell array of structure arrays. `s` is the output of `scatteringTransform` for the image scattering network `sf`.

u — Scalogram coefficients

cell array

Scalogram coefficients, specified as a cell array of structure arrays. `u` is the output of `scatteringTransform` for the image scattering network `sf`.

x — Scattering feature matrix

real-valued matrix | real-valued 4-D tensor

Scattering feature matrix, specified as a real-valued 3-D matrix or a real-valued 4-D tensor. `x` is the output of `featureMatrix` for the image scattering network `sf`.

Output Arguments

slog — Natural logarithm of scattering coefficients

cell array

Natural logarithm of scattering coefficients, returned as a cell array. The dimensions of `slog` are equal to the dimensions of `s`.

The precision of `slog` depends on the precision specified in the network `sf`.

`ulog` — Natural logarithm of scalogram coefficients
cell array

Natural logarithm of scalogram coefficients, returned as a cell array. The dimensions of `ulog` are equal to the dimensions of `u`.

The precision of `ulog` depends on the precision specified in the network `sf`.

`xlog` — Natural logarithm of scattering feature matrix
real-valued 3-D matrix | real-valued 4-D tensor

Natural logarithm of scattering feature matrix, returned as a real-valued matrix or tensor. The dimensions of `xlog` are equal to the dimensions of `x`.

The precision of `xlog` depends on the precision specified in the network `sf`.

Algorithms

`log` returns the natural logarithm of the absolute value of the input argument.

See Also

`waveletScattering2` | `featureMatrix` | `scatteringTransform`

Introduced in R2019a

lp2filters

Laurent polynomials to filters

Syntax

```
[LoD,HiD,LoR,HiR] = lp2filters(LoDz,HiDz,LoRz,HiRz)
[LoD,HiD,LoR,HiR] = lp2filters( ___ ,signFLAG)
```

Description

`[LoD,HiD,LoR,HiR] = lp2filters(LoDz,HiDz,LoRz,HiRz)` returns the filters associated with the Laurent polynomials `LoDz`, `HiDz`, `LoRz`, and `HiRz`. The polynomials are associated to the filters as follows:

- `LoDz` — *Z*`(LoD)`
- `HiDz` — *Z*`(HiD)`
- `LoRz` — *Z*`(LoR)`
- `HiDz` — *Z*`(HiD)`

where *Z*`(.)` is the z-transform of the corresponding filter.

`[LoD,HiD,LoR,HiR] = lp2filters( ___ ,signFLAG)` changes the signs of the two highpass filters, `HiD` and `HiR`, when `signFLAG` is equal to `1`. The default value for `signFLAG` is `0`.

Examples

Filters Associated With Laurent Polynomials

Obtain the filters associated with the orthogonal `db4` wavelet.

```
wv = "db4";
[LoD,HiD,LoR,HiR] = wfilters(wv);
```

Use the `filters2lp` function to obtain Laurent polynomials associated with the lowpass filter.

```
[LoDz,HiDz,LoRz,HiRz] = filters2lp({LoR});
```

Use the `lp2filters` function to obtain a new set of filters. Confirm the first and second set of filters are identical.

```
[LoD2,HiD2,LoR2,HiR2] = lp2filters(LoDz,HiDz,LoRz,HiRz);
max(abs(LoD-LoD2))
```

```
ans = 0
```

```
max(abs(HiD-HiD2))
```

```
ans = 0
```

```
max(abs(LoR-LoR2))
```

```
ans = 0
```

```
max(abs(HiR-HiR2))
```

```
ans = 0
```

Confirm that for orthogonal wavelets, the reflection of `LoDz` is equal to `LoRz`.

```
areEqual = (reflect(LoDz)==LoRz)
```

```
areEqual = logical
   1
```

Input Arguments

LoDz — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

HiDz — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

LoRz — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

HiRz — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

signFLAG — Change sign flag
`0` (default) | `1`

Change sign flag, specified as `0` or `1`. If `signFLAG` is equal to `1`, the signs of the highpass filters `HiD` and `HiR` change.

Output Arguments

LoD — Lowpass filter
real-valued vector

Lowpass filter associated with the Laurent polynomial `LoDz`, returned as a real-valued vector.

Data Types: `double`

HiD — Highpass filter
real-valued vector

Highpass filter associated with the Laurent polynomial `HiDz`, returned as a real-valued vector.

Data Types: `double`

LoR — Lowpass filter
real-valued vector

Lowpass filter associated with the Laurent polynomial `LoRz`, returned as a real-valued vector.

Data Types: `double`

`HiR` — Highpass filter
real-valued vector

Highpass filter associated with the Laurent polynomial `HiRz`, returned as a real-valued vector.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`filters2lp` | `wave2lp`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

lp2LS

Laurent polynomials to lifting steps and normalization factors

Syntax

```
[lsteps,k] = lp2LS(wavetype,LoRz,HiRz,factmode)
```

Description

`[lsteps,k] = lp2LS(wavetype,LoRz,HiRz,factmode)` returns the lifting scheme steps `lsteps` and normalization factors `k` associated with the Laurent polynomials `LoRz` and `HiRz`. `wavetype` specifies the wavelet type corresponding to the Laurent polynomials, and `factmode` specifies the factorization mode.

Examples

Obtain Lifting Scheme from Laurent Polynomials

Create a lifting scheme associated with the `db2` wavelet.

```
wv = "db2";
lsw = liftingScheme(Wavelet=wv);
```

Obtain the Laurent polynomials associated with the wavelet, and then obtain the lifting steps and normalization factors associated with the Laurent polynomials.

```
[~,~,LoRz,HiRz] = wave2lp(wv);
[lsteps,k] = lp2LS("o",LoRz,HiRz,"s");
```

Create a lifting scheme using the lifting steps and normalization factors.

```
lscheme = liftingScheme(LiftingSteps=lsteps,NormalizationFactors=k)
```

```
lscheme = 
      Wavelet               : 'custom'
     LiftingSteps          : [3 × 1] liftingStep
     NormalizationFactors  : [3.3461 0.2989]
     CustomLowpassFilter   : [  ]


 Details of LiftingSteps :
            Type: 'update'
    Coefficients: -0.5774
        MaxOrder: 0

            Type: 'predict'
    Coefficients: [-2.7990 0.4330]
        MaxOrder: 1

            Type: 'update'
    Coefficients: 0.3333
    MaxOrder: -1
```

Input Arguments

`wavetype` — Wavelet type
`"o"` | `"b"`

Wavelet type associated with the Laurent polynomials `LoRz` and `HiRz`, specified as one of:

- `"o"` — Orthogonal wavelet
- `"b"` — Biorthogonal wavelet

Data Types: `char` | `string`

`LoRz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with a lowpass filter, specified as a `laurentPolynomial` object.

`HiRz` — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial associated with a highpass filter, specified as a `laurentPolynomial` object.

`factmode` — Factorization mode
`"analysis"` (default) | `"synthesis"`

Factorization mode of lifting steps and normalization factors, specified as one of:

- `"analysis"` — Analysis factorization
- `"synthesis"` — Synthesis factorization

Output Arguments

`lsteps` — Lifting steps
array of `liftingStep` structures

Lifting steps, returned as an array of `liftingStep` structures.

`k` — Normalization factors
real-valued vector

Normalization factors, returned as a 1-by-2 real-valued vector.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`liftingStep`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

ls2filt

(To be removed) Transform lifting scheme to quadruplet of filters

Note This version of `ls2filt` will be removed in a future release. Use the new version of `ls2filt` and `liftingScheme`. For more information, see “Compatibility Considerations”.

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
```

Compatibility Considerations

`ls2filt` will be removed

Not recommended starting in R2021a

`ls2filt` will be removed in a future release. Use `ls2filt`, the new version of `ls2filt`, and `liftingScheme`. To update your code, follow these steps:

1. Create a lifting scheme using `liftingScheme`.
2. Extract the wavelet filters using `ls2filt`.

See Also

`liftingScheme` | `ls2filt`

Introduced before R2006a

ls2filt

Extract wavelet filters from lifting scheme

Syntax

```
[lod,hid,lor,hir] = ls2filt(lscheme)
```

Description

`[lod,hid,lor,hir] = ls2filt(lscheme)` returns the wavelet decomposition and reconstruction filters associated with the lifting scheme `lscheme`.

Examples

Compare Lifting Scheme Filters

Create a lifting scheme associated with the `db4` wavelet.

```
wv = 'db4';
lsc = liftingScheme('Wavelet',wv);
```

Use `ls2filt` to extract from the lifting scheme the corresponding wavelet filters. Compare with the filters generated by `wfilters`. Confirm they are equal.

```
[lod,hid,lor,hir] = ls2filt(lsc);
[lod2,hid2,lor2,hir2] = wfilters(wv);
fprintf('Lowpass Decomposition\n ls2filt: %s\nwfilters: %s\n',num2str(lod),num2str(lod2))
```

```
Lowpass Decomposition
 ls2filt: -0.010597    0.032883    0.030841    -0.18703   -0.027984     0.63088     0.71485     0
wfilters: -0.010597    0.032883    0.030841    -0.18703   -0.027984     0.63088     0.71485     0
```

```
fprintf('Highpass Decomposition\n ls2filt: %s\nwfilters: %s\n',num2str(hid),num2str(hid2))
```

```
Highpass Decomposition
 ls2filt: -0.23038     0.71485    -0.63088   -0.027984     0.18703    0.030841   -0.032883   -0.0
wfilters: -0.23038     0.71485    -0.63088   -0.027984     0.18703    0.030841   -0.032883   -0.0
```

```
fprintf('Lowpass Reconstruction\n ls2filt: %s\nwfilters: %s\n',num2str(lor),num2str(lor2))
```

```
Lowpass Reconstruction
 ls2filt: 0.23038     0.71485     0.63088   -0.027984    -0.18703    0.030841    0.032883   -0.01
wfilters: 0.23038     0.71485     0.63088   -0.027984    -0.18703    0.030841    0.032883   -0.01
```

```
fprintf('Highpass Reconstruction\n ls2filt: %s\nwfilters: %s\n',num2str(hir),num2str(hir2))
```

```
Highpass Reconstruction
 ls2filt: -0.010597   -0.032883    0.030841     0.18703   -0.027984    -0.63088     0.71485    -0
wfilters: -0.010597   -0.032883    0.030841     0.18703   -0.027984    -0.63088     0.71485    -0
```

Now create a lifting scheme associated with the `bior2.2` wavelet.

```
wv = 'bior2.2';
lsc = liftingScheme('Wavelet',wv);
```

Use `ls2filt` to extract from the lifting scheme the corresponding wavelet filters. Compare with the filters generated by `wfilters`. Observe that `wfilters` includes the missing powers of the associated Laurent series as zeros so that all filters have equal even length. Except for the prepended and appended zeros, the filters coefficients generated by `wfilters` equal the coefficients returned by `ls2filt`.

```
[lod,hid,lor,hir] = ls2filt(lsc);
[lod2,hid2,lor2,hir2] = wfilters(wv);
fprintf('Lowpass Decomposition\n ls2filt: %s\nwfilters: %s\n',num2str(lod),num2str(lod2))
```

```
Lowpass Decomposition
 ls2filt: -0.17678     0.35355      1.0607     0.35355    -0.17678
wfilters: 0    -0.17678     0.35355      1.0607     0.35355    -0.17678
```

```
fprintf('Highpass Decomposition\n ls2filt: %s\nwfilters: %s\n',num2str(hid),num2str(hid2))
```

```
Highpass Decomposition
 ls2filt: 0.35355    -0.70711     0.35355
wfilters: 0     0.35355    -0.70711     0.35355           0           0
```

```
fprintf('Lowpass Reconstruction\n ls2filt: %s\nwfilters: %s\n',num2str(lor),num2str(lor2))
```

```
Lowpass Reconstruction
 ls2filt: 0.35355     0.70711     0.35355
wfilters: 0     0.35355     0.70711     0.35355           0           0
```

```
fprintf('Highpass Reconstruction\n ls2filt: %s\nwfilters: %s\n',num2str(hir),num2str(hir2))
```

```
Highpass Reconstruction
 ls2filt: 0.17678     0.35355     -1.0607     0.35355     0.17678
wfilters: 0     0.17678     0.35355     -1.0607     0.35355     0.17678
```

Input Arguments

`lscheme` — Lifting scheme

`liftingScheme` object

Lifting scheme, specified as a `liftingScheme` object.

Output Arguments

`lod,hid` — Decomposition filters

vectors

Decomposition filters associated with the lifting scheme, returned as vectors. `lod` is the lowpass decomposition filter. `hid` is the highpass decomposition filter.

Data Types: `double`

`lor,hir` — Reconstruction filters

vectors

Reconstruction filters associated with the lifting scheme, returned as vectors. `lor` is the lowpass decomposition filter. `hir` is the highpass decomposition filter.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`liftingScheme` | `wavedec` | `wavedec2`

Introduced in R2021a

lsinfo

(To be removed) Lifting schemes information

Note `lsinfo` will be removed in a future release. Use `disp` and `liftingScheme` instead. For more information, see “Compatibility Considerations”.

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]     []
     }
```

Compatibility Considerations

`lsinfo` will be removed

Not recommended starting in R2021a

`lsinfo` will be removed in a future release. For lifting, use `disp` to display information of a lifting scheme created by `liftingScheme`.

See Also

`disp` | `liftingScheme`

Introduced before R2006a

lwt

1-D lifting wavelet transform

Syntax

```
[ca,cd] = lwt(x)
[ca,cd] = lwt( ___ ,Name,Value)
```

Description

`[ca,cd] = lwt(x)` returns the wavelet decomposition of `x`. `lwt` uses the lifting scheme associated with the `db1` wavelet and does not preserve integer-valued data. `x` is a vector or matrix. If `x` is a matrix, `lwt` operates along the first dimension of `x`. `x` must have at least two samples. If `x` is of even length, the wavelet transform is obtained down to level `floor(log2(N))`, where *N* is the length of `x` if `x` is a vector, and the row dimension of `x` if `x` is a matrix. If *N* is odd, `x` is extended by one sample by duplicating the last element of `x`.

`[ca,cd] = lwt( ___ ,Name,Value)` specifies options using one or more name-value arguments. For example, `[ca,cd] = lwt(x,'Level',2)` specifies a level 2 wavelet decomposition.

Examples

Lifting Wavelet Transform of Integer-Valued Signal

Specify an integer-valued signal. Create a lifting scheme associated with the `db2` wavelet.

```
sig = 1:10;
lsc = liftingScheme('Wavelet','db2');
```

Obtain the level 2 lifting wavelet transform (LWT) using the lifting scheme. Display the approximation and detail coefficients.

```
wv = 'db2';
[ca,cd] = lwt(sig,'LiftingScheme',lsc,'Level',2);
ca
```

```
ca = 3×1

    5.8038
   14.0801
   16.5801

```

```
cd{1}
```

```
ans = 5×1

    3.5355
         0
    0.0000
    0.0000
    0.0000
```

```
cd{2}
```

```
ans = 3×1

    5.0311
   -0.0000
   -1.0311
```

Obtain the decomposition again, but this time preserve the integer-valued data.

```
[ca,cd] = lwt(sig,'LiftingScheme',lsc,'Level',2,'Int2Int',true);
ca
```

```
ca = 3×1

     2
     4
     4
```

```
cd{1}
```

```
ans = 5×1

     6
     0
     0
     0
     0
```

```
cd{2}
```

```
ans = 3×1

     5
     1
     0
```

LWT of Multichannel Signal

Load the 23 channel EEG data `Espiga3`. The channels are arranged column-wise.

```
load Espiga3
size(Espiga3)
```

```
ans = 1×2

   995    23
```

Obtain the LWT of the multichannel signal using the db4 wavelet down to the default maximum decomposition level.

```
wv = 'db4';
[ca,cd] = lwt(Espiga3,'Wavelet',wv);
```

Confirm the number of columns in ca is equal to the number of channels in the multichannel signal, and that the detail coefficients are an *N*-by-1 cell array, where *N* is equal to floor(log2(size(Espiga3,1))).

```
size(ca)
```

```
ans = 1×2

     2    23
```

```
floor(log2(size(Espiga3,1)))
```

```
ans = 9
```

```
size(cd)
```

```
ans = 1×2

     9     1
```

Input Arguments

x — Signal

vector | matrix

Signal, specified as a vector or matrix. If x is a matrix, lwt operates along the first dimension of x. x must have at least two samples. If x has an odd number of samples, x is extended by one sample by duplicating the last element of x.

Data Types: single | double
Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: [ca,cd] = lwt(x,'Wavelet','db3','Level',4) uses the db3 wavelet to perform a level 4 wavelet decomposition.

Wavelet — Wavelet

character vector | string scalar

Orthogonal or biorthogonal wavelet to use in the LWT, specified as a character vector or string scalar. See the Wavelet property of liftingScheme for the list of supported wavelets.

You cannot specify 'Wavelet' and 'LiftingScheme' name-value arguments at the same time.

Example: [ca,cd] = lwt(x,'Wavelet','bior3.5') uses the bior3.5 biorthogonal wavelet.

LiftingScheme — Lifting scheme
liftingScheme object

Lifting scheme to use in the LWT, specified as a liftingScheme object.

You cannot specify 'LiftingScheme' and 'Wavelet' name-value arguments at the same time.

Example: [ca,cd] = lwt(x,'LiftingScheme',lScheme) uses the lScheme lifting scheme.

Level — Level of decomposition
positive integer

Level of wavelet decomposition, specified as a positive integer less than or equal to floor(log2(*N*)), where *N* is the length of x if x is a vector, or the row dimension of x if x is a matrix.

Example: [ca,cd] = lwt(x,'Level',4) specifies a level 4 wavelet decomposition.

Data Types: double

Extension — Extension mode
'periodic' (default) | 'zeropad' | 'symmetric'

Extension mode to use in the LWT, specified as 'periodic' (default), 'zeropad', or 'symmetric'. The value of 'Extension' specifies how to extend the signal at the boundaries.

Example: [ca,cd] = lwt(x,'Extension','symmetric') specifies the symmetric extension mode.

Int2Int — Integer-valued data handling
false or 0 (default) | true or 1

Integer-valued data handling, specified as a numeric or logical 1 (true) or 0 (false).

- 1 (true) — Preserve integer-valued data
- 0 (false) — Do not preserve integer-valued data

Specify the 'Int2Int' name-value argument only if all elements of the input are integers.

Example: [ca,cd] = lwt(1:8,'Int2Int',true) preserves integer-valued data.

Output Arguments

ca — Approximation coefficients
scalar | vector | matrix

Approximation (lowpass) coefficients at the coarsest level, returned as a scalar, vector, or matrix. The dimension of ca depends on the signal dimension.

Data Types: single | double

cd — Detail coefficients
cell array

Detail coefficients, returned as an *L*-by-1 cell array, where *L* is the level of the transform. The elements of `cd` are in order of decreasing resolution.

Data Types: `single` | `double`

Compatibility Considerations

`lwt` input syntax has changed

Behavior changed in R2021a

The `lwt` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CD] = lwt(X,W)`	Errors	`[CA,CD] = lwt(X,'Wavelet',W)`	You can also obtain the lifting wavelet transform (LWT) of a 1-D signal using a lifting scheme by setting the `LiftingScheme` name-value argument.
`[CA,CD] = lwt(X,W,LEVEL)`	Errors	`[CA,CD] = lwt(X,'Wavelet',W, 'Level',LEVEL)`	You can also specify the extension mode by setting the `ExtensionMode` name-value argument.
`[CA,CD] = lwt(X,W,LEVEL,'typeDEC','wp')`	Errors	NA	The wavelet packet decomposition option is no longer provided.
`X_InPlace = lwt(X,W)`	Errors	NA	In-place transforms are no longer supported.

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

[2] Sweldens, Wim. “The Lifting Scheme: A Construction of Second Generation Wavelets.” *SIAM Journal on Mathematical Analysis* 29, no. 2 (March 1998): 511–46. https://doi.org/10.1137/S0036141095289051.

Extended Capabilities

C/C++ Code Generation

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

See Also

`liftingScheme` | `haart` | `ilwt` | `ihaart` | `lwtcoef`

Introduced in R2021a

lwt2

2-D Lifting wavelet transform

Syntax

```
[ll,lh,hl,hh] = lwt2(x)
[ ___ ] = lwt2(x,Name=Value)
```

Description

`[ll,lh,hl,hh] = lwt2(x)` performs the 2-D lifting wavelet transform (LWT) of the real- or complex-valued matrix `x` using the `db1` wavelet. The function performs the decomposition first along the rows in `x` and then along the columns. The default decomposition level depends on the size of `x`. For more information, see 'Level'. The function returns the approximation coefficients at the coarsest scale and the horizontal, vertical, and diagonal detail coefficients by level.

If `x` is a single-precision input, the numeric type of the coefficients is single precision. Otherwise, the numeric type is double precision.

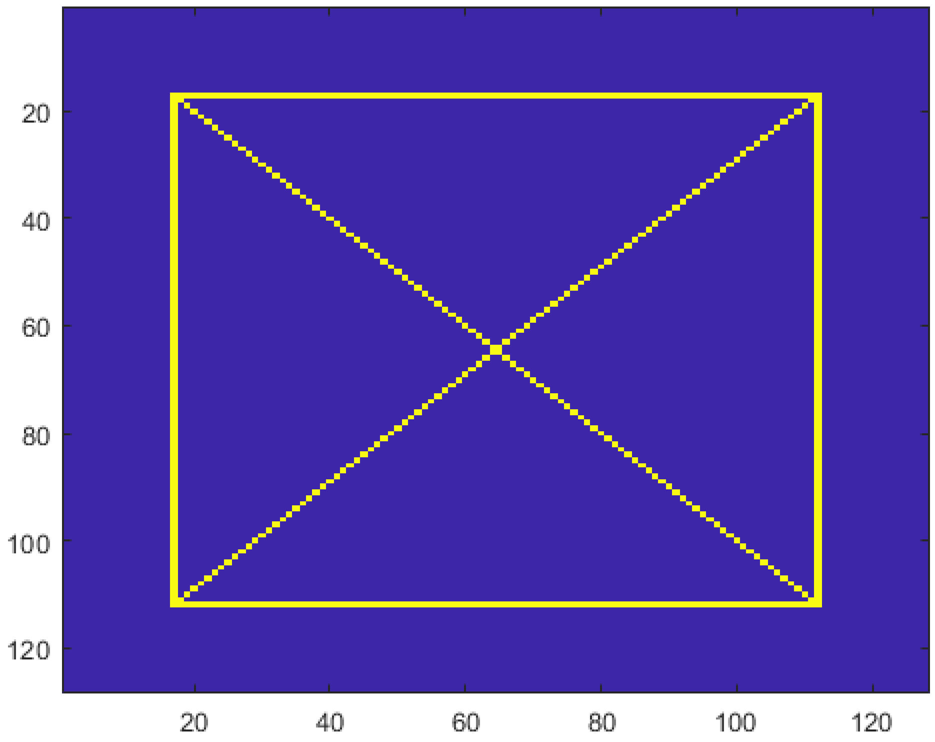
`[ ___ ] = lwt2(x,Name=Value)` specifies options using one or more name-value arguments. For example, `lwt2(x,Wavelet="db2",Level=3)` performs 2-D LWT using the `db2` wavelet and a level 3 decomposition.

Examples

Lifting Wavelet Transform of 2-D Data

Load and display the `xbox` image.

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

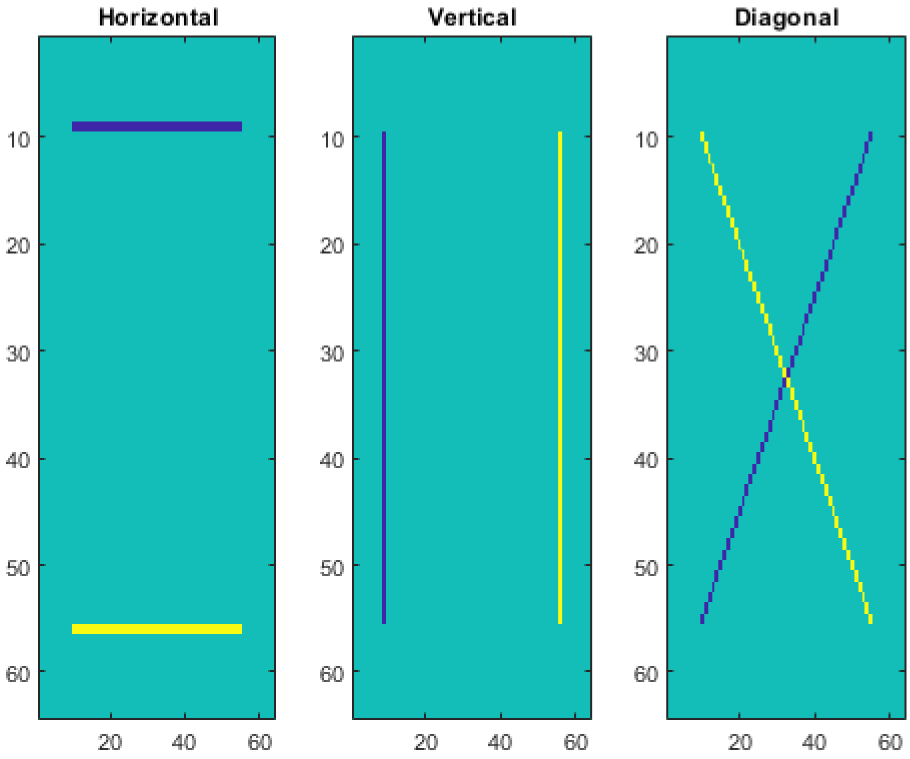


Obtain the 2-D LWT of the image using default settings.

```
[ll,lh,hl,hh] = lwt2(xbox);
```

Display the first level detail coefficients.

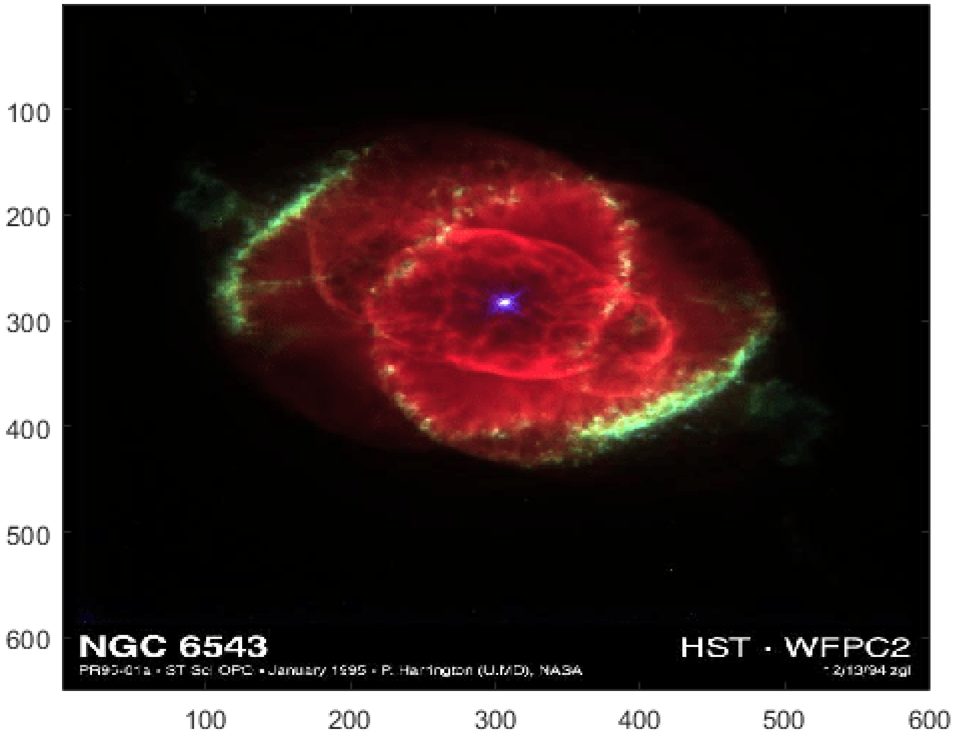
```
subplot(1,3,1)
imagesc(lh{1})
title("Horizontal")
subplot(1,3,2)
imagesc(hl{1})
title("Vertical")
subplot(1,3,3)
imagesc(hh{1})
title("Diagonal")
```



Lifting Wavelet Transform of RGB Image Using Lifting Scheme

Load an RGB image. An RGB image is also known as a *truecolor* image. The image is a 3-D array of type `uint8`.

```
x = imread("ngc6543a.jpg");
image(x)
```



Create the lifting scheme associated with the `bior3.7` wavelet. Obtain the level 3 LWT of the image using the lifting scheme. Preserve the integer-valued data.

```
lvl = 3;
lScheme = liftingScheme("Wavelet","bior3.7");
[ll,lh,hl,hh] = lwt2(x,LiftingScheme=lScheme,Level=lvl,Int2Int=true);
```

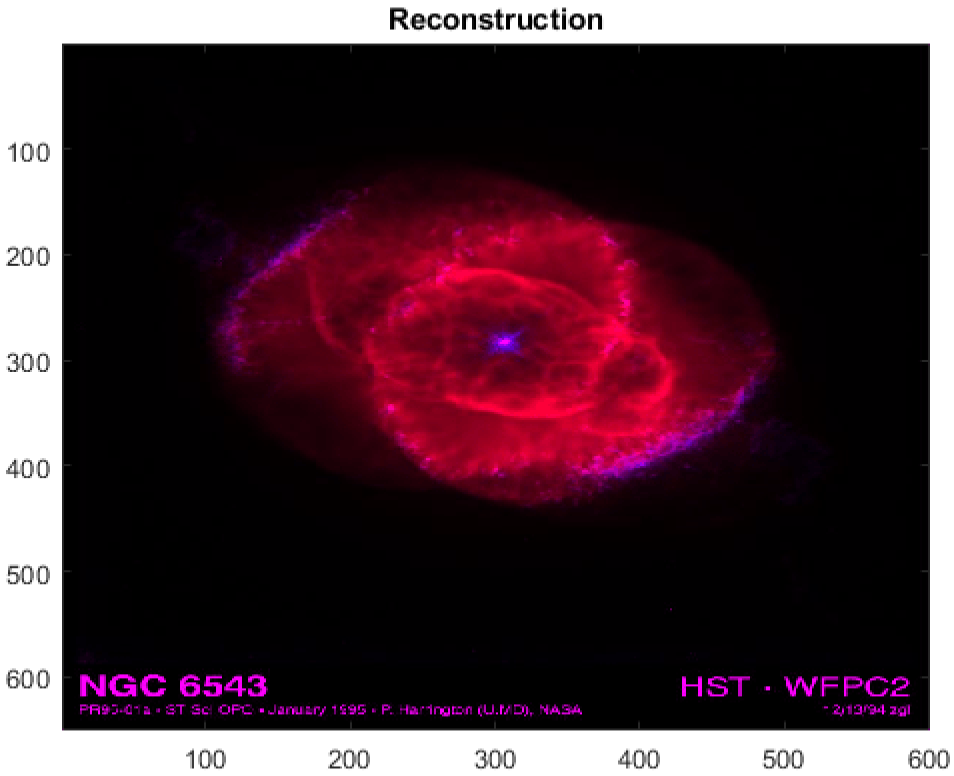
Confirm the approximation coefficients are all integer valued. Choose a level and confirm all the detail coefficients at that level are integer valued.

```
approxDiffs = ll-floor(ll);
max(abs(approxDiffs(:)))

ans = 0
```

```
lev = 2;
horizDiffs = lh{lev}-floor(lh{lev});
vertDiffs = hl{lev}-floor(hl{lev});
diagDiffs = hh{lev}-floor(hh{lev});
[max(abs(horizDiffs(:))) max(abs(vertDiffs(:))) max(abs(diagDiffs(:)))]

ans = 1×3

     0     0     0
```

Reconstruct the image using the red and blue components of the coefficients. Display the reconstruction.

```
llx = ll;
llx(:,:,2) = 0;
for k=1:lvl
    lhx{k} = lh{k};
    hlx{k} = hl{k};
    hhx{k} = hh{k};
    lhx{k}(:,:,2) = 0;
    hlx{k}(:,:,2) = 0;
    hhx{k}(:,:,2) = 0;
end
xrec = ilwt2(llx,lhx,hlx,hhx,LiftingScheme=lScheme,Int2Int=true);
imagesc(uint8(xrec))
title("Reconstruction")
```



Confirm the reconstruction is integer valued.

```
recDiffs = xrec-floor(xrec);
max(abs(recDiffs(:)))
```

```
ans = 0
```

Input Arguments

x — Input data

matrix

Input data, specified as a real- or complex-valued 2-D, 3-D, or 4-D matrix. The input `x` must have at least two samples in the row and column dimensions.

- If `size(x,1)` is odd, the function extends `x` by duplicating the last row.
- If `size(x,2)` is odd, the function extends the last column of `x` by duplicating the last column.

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

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[ll,lh,hl,hh] = lwt2(x,LiftingScheme=lscheme,Level=2)`

`Wavelet` — Wavelet

`"db1"` (default) | character vector | string scalar

Orthogonal or biorthogonal wavelet to use in the 2-D LWT, specified as a character vector or string scalar. See the Wavelet property of `liftingScheme` for the list of supported wavelets.

You cannot specify `Wavelet` and `LiftingScheme` at the same time.

Example: `[ll,~,~,hh] = lwt2(x,Wavelet="bior3.5")` uses the `bior3.5` biorthogonal wavelet.

Data Types: `char` | `string`

`LiftingScheme` — Lifting scheme

`liftingScheme` object

Lifting scheme to use in the 2-D LWT, specified as a `liftingScheme` object.

You cannot specify `LiftingScheme` and `Wavelet` at the same time.

Example: `[~,lh,hl,~] = lwt2(x,LiftingScheme=lScheme)` uses the `lScheme` lifting scheme.

`Level` — Decomposition level

positive integer

Decomposition level of the 2-D LWT, specified as a positive integer less than or equal to `floor(log2(`*`N`*`))`, where *`N`* `= min(size(x,[1 2])/2)`.

The default decomposition level depends on the number of rows and columns in `x`.

- If the number of both the rows and the columns is a power of two, the function performs 2-D LWT down to level `log2(min(size(x,[1 2])))`.
- If the number of both the rows and the columns is even but at least one is not a power of two, the function performs 2-D LWT down to `floor(log2(`*`N`*`))`, where *`N`* `= min(size(x,[1 2])/2)`.

Example: `[ll,~,hl,~] = lwt2(x,Level=4)` specifies a level 4 wavelet decomposition.

Data Types: `double`

`Extension` — Extension mode
`"periodic"` (default) | `"zeropad"` | `"symmetric"`

Extension mode to use in the LWT, specified as one of these:

- `"periodic"` — Periodized extension
- `"zeropad"` — Zero extension
- `"symmetric"` — Symmetric extension

This argument specifies how `lwt2` extends the input at the boundaries.

Example: `[~,lh,~,hh] = lwt2(x,Extension="symmetric")` specifies the symmetric extension mode.

Data Types: `char` | `string`

`Int2Int` — Handling integer-valued data
`false` or `0` (default) | `true` or `1`

Handling integer-valued data, specified as one of these:

- `1` (`true`) — Preserve integer-valued data
- `0` (`false`) — Do not preserve integer-valued data

Specify `Int2Int` only if all elements of the input are integers.

Example: `[~,lh,hl,hh] = lwt2(x,Int2Int=true)` preserves integer-valued data.

Output Arguments

`ll` — Approximation coefficients
scalar | vector | matrix

Approximation coefficients at the coarsest scale, returned as a scalar, vector, or matrix.

Data Types: `single` | `double`

`lh` — Horizontal detail coefficients
cell array

Horizontal detail coefficients by level, returned as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `lh` are in order of decreasing resolution.

Data Types: `single` | `double`

`hl` — Vertical detail coefficients
cell array

Vertical detail coefficients by level, returned as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hl` are in order of decreasing resolution.

Data Types: `single` | `double`

hh — Diagonal detail coefficients

cell array

Diagonal detail coefficients by level, returned as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hh` are in order of decreasing resolution.

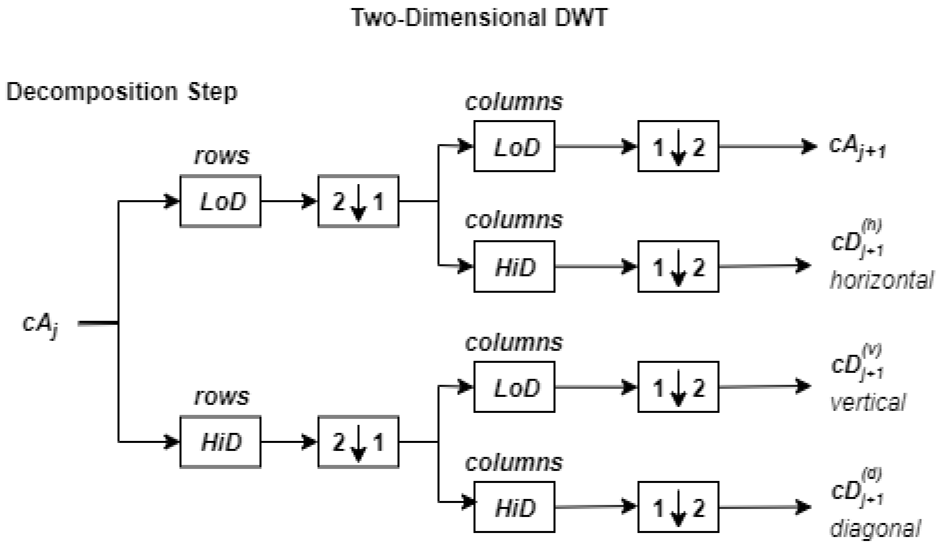
Data Types: `single` | `double`

Algorithms

At each stage of a 2-D wavelet decomposition, the approximation coefficients at level *j* are decomposed into four components: the approximation at level *j*+1 and the details in three orientations (horizontal, vertical, and diagonal). Each component is the result of convolving the rows and columns of the level *j* approximation with the appropriate combination of lowpass and highpass filters, *LoD* and *HiD*, respectively, followed by downsampling:

- Approximation — Convolve the rows and columns with a lowpass filter (`ll`)
- Horizontal — Convolve the rows with a lowpass filter, and convolve the columns with a highpass filter (`lh`)
- Vertical — Convolve the rows with a highpass filter, and convolve the columns with a lowpass filter (`hl`)
- Diagonal — Convolve the rows and columns with a highpass filter (`hh`)

The following chart describes the basic decomposition steps.



where

- $\boxed{2\downarrow 1}$ — Downsample columns: keep the even-indexed columns
- $\boxed{1\downarrow 2}$ — Downsample rows: keep the even-indexed rows

- rows $\boxed{X}$ — Convolve the rows of the entry with filter *X*
- columns $\boxed{X}$ — Convolve the columns of the entry with filter *X*

The decomposition is initialized by setting the approximation coefficients equal to the image *s*: $cA_0 = s$.

Compatibility Considerations

`lwt2` input syntax has changed

Behavior changed in R2021b

The `lwt2` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`[CA,CH,CV,CD] = lwt2(X,W)`	Errors	`[CA,CH,CV,CD] = lwt2(X,Wavelet=W)`	You can also obtain the lifting wavelet transform (LWT) using a lifting scheme by setting the `LiftingScheme` name-value argument.
`[CA,CH,CV,CD] = lwt2(X,W,LEVEL)`	Errors	`[CA,CH,CV,CD] = lwt2(X,Wavelet=W,Level=LEVEL)`	You can also specify the extension mode by setting the `Extension` name-value argument.
`[CA,CD] = lwt2(X,W,LEVEL,'typeDEC','wp')`	Errors	NA	The wavelet packet decomposition option is no longer provided.
`X_InPlace = lwt2(X,LS)`	Errors	NA	In-place transforms are no longer supported.

References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S.G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11, no. 7 (July 1989): 674–93. https://doi.org/10.1109/34.192463.

[3] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

[4] Sweldens, Wim. “The Lifting Scheme: A Construction of Second Generation Wavelets.” *SIAM Journal on Mathematical Analysis* 29, no. 2 (March 1998): 511–46. https://doi.org/10.1137/S0036141095289051.

Extended Capabilities

C/C++ Code Generation

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

See Also

`ilwt2` | `lwtcoef2` | `haart2` | `ihaart2` | `liftingScheme`

Introduced in R2021b

lwtcoef

Extract or reconstruct 1-D LWT wavelet coefficients and orthogonal projections

Syntax

```
y = lwtcoef(ca,cd)
y = lwtcoef(ca,cd,Name,Value)
```

Description

`y = lwtcoef(ca,cd)` returns the level 1 approximation coefficients that correspond to the approximation and detail coefficients, `ca` and `cd`, respectively. `ca` and `cd` are outputs of `lwt`.

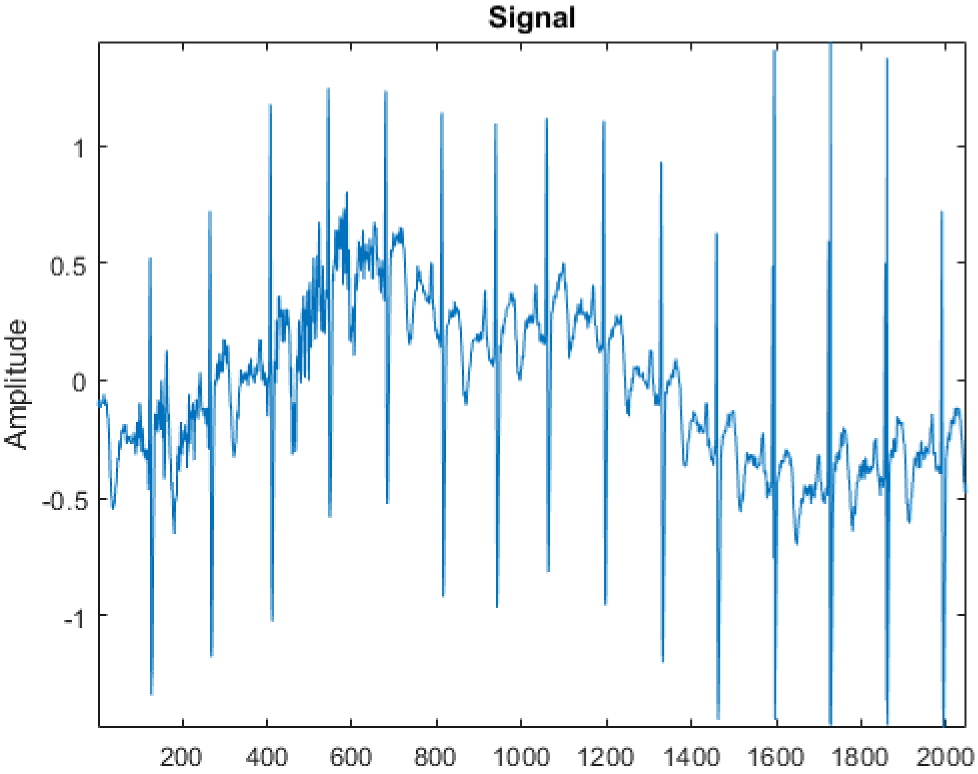
`y = lwtcoef(ca,cd,Name,Value)` specifies options using one or more name-value arguments. For example, `y = lwtcoef(ca,cd,'OutputType','coefficients')` specifies coefficients output.

Examples

Reconstruct Signal from Orthogonal Projections

Load a 1-D signal of length 2048. Plot the signal.

```
load wecg
plot(wecg)
title('Signal')
ylabel('Amplitude')
axis tight
```



Create a lifting scheme associated with the db4 wavelet. Use the lifting scheme to obtain the wavelet decomposition of the signal to the maximum level. Confirm the length of the detail coefficients cell array equals floor(log2(N)), where *N* is the length of the signal.

```
wv = 'db4';
lsc = liftingScheme('Wavelet',wv);
[ca,cd] = lwt(wecg,'LiftingScheme',lsc);
[length(cd) floor(log2(length(wecg)))]
```

```
ans = 1×2

    11    11
```

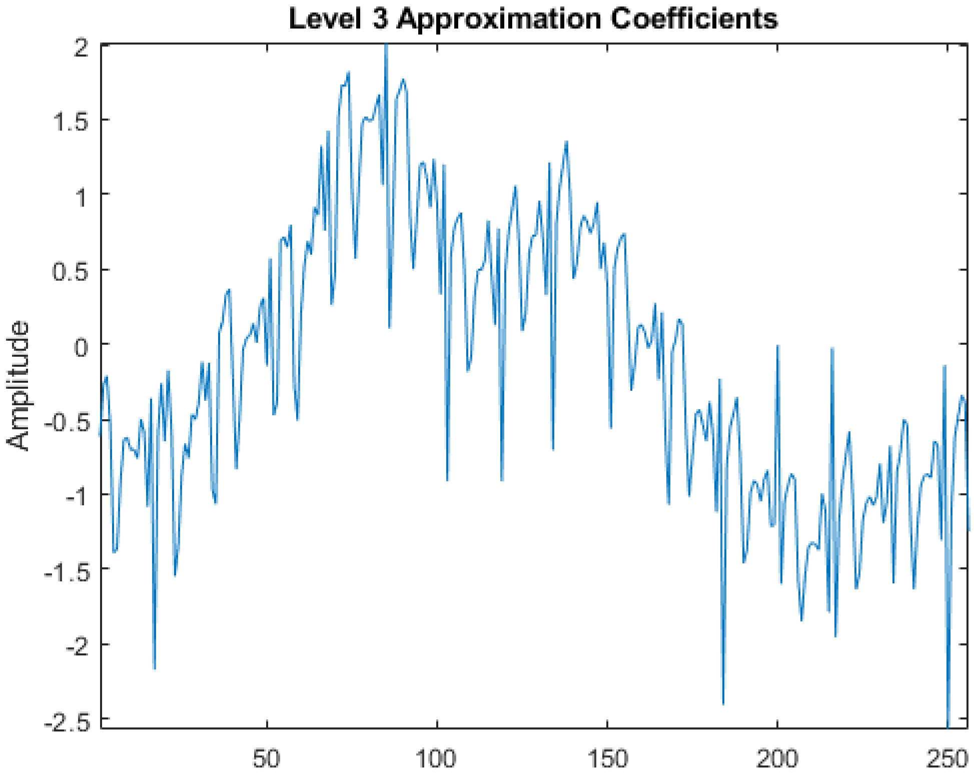
Extract and plot the approximation coefficients at level 3. Confirm the length of the extraction is one-eighth the length of the original signal.

```
approxCf = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','coefficients','Level',3);
[2048/(2^3) length(approxCf)]
```

```
ans = 1×2

   256   256
```

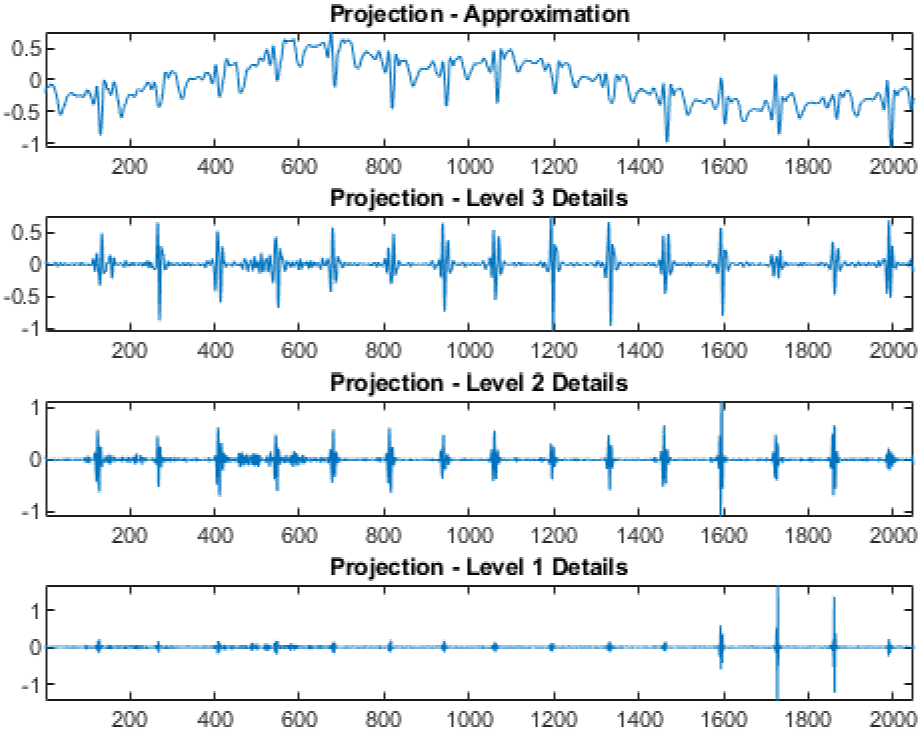
```
plot(approxCf)
title('Level 3 Approximation Coefficients')
```

```
ylabel('Amplitude')
axis tight
```



Obtain the orthogonal projection of the level 3 approximation coefficients. Also obtain the orthogonal projections of the detail coefficients at levels 1, 2, and 3. Plot the results.

```
approx3 = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','projection','Level',3);
det3 = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','projection','Level',3,'Type','detail');
det2 = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','projection','Level',2,'Type','detail');
det1 = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','projection','Level',1,'Type','detail');
subplot(4,1,1)
plot(approx3)
title('Projection - Approximation')
axis tight
subplot(4,1,2)
plot(det3)
title('Projection - Level 3 Details')
axis tight
subplot(4,1,3)
plot(det2)
title('Projection - Level 2 Details')
axis tight
subplot(4,1,4)
plot(det1)
title('Projection - Level 1 Details')
axis tight
```



Confirm the sum of the four projections equals the original signal.

```
max(abs(wecg-(approx3+det3+det2+det1)))
```

```
ans = 1.3323e-15
```

Input Arguments

ca — Approximation coefficients

scalar | vector | matrix

Approximation (lowpass) coefficients at the coarsest level, specified as a scalar, vector, or matrix. The coefficients are the output of `lwt`.

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

cd — Detail coefficients

cell array

Detail coefficients, specified as an *L*-by-1 cell array, where *L* is the level of the transform. The elements of `cd` are in order of decreasing resolution. The coefficients are the output of `lwt`.

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

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `y = lwtcoef(ca,cd,'LiftingScheme',lsc,'OutputType','coefficients','Level',2)` uses the lifting scheme `lsc` to extract the approximation coefficients at level 2.

`Wavelet` — Wavelet
`'db1'` (default) | character vector | string scalar

Orthogonal or biorthogonal wavelet, specified as a character vector or string scalar. See the Wavelet property of `liftingScheme` for the list of supported wavelets. For perfect reconstruction, the specified wavelet must match the wavelet you used to generate `ca` and `cd`.

You cannot specify `'Wavelet'` and `'LiftingScheme'` name-value arguments at the same time.

`LiftingScheme` — Lifting scheme
`liftingScheme` object

Lifting scheme to use, specified as a `liftingScheme` object. For perfect reconstruction, the specified lifting scheme must match the lifting scheme you used to generate `ca` and `cd`.

You cannot specify `'Wavelet'` and `'LiftingScheme'` name-value arguments at the same time.

`OutputType` — Output type
`'coefficients'` (default) | `'projection'`

Output type, specified as one of:

- `'coefficients'` — Extract the approximation or details coefficients
- `'projection'` — Return the projection (reconstruction) of the approximation or details coefficients

Example: `y = lwtcoef(ca,cd,'OutputType','projection','Type','detail')` returns the projection corresponding to the detail coefficients at the finest scale.

`Type` — Type of coefficients
`'approximation'` (default) | `'detail'`

Type of coefficients to extract or reconstruct, specified as `'approximation'` or `'detail'`.

Example: `y = lwtcoef(ca,cd,'Type','detail')` extracts the detail coefficients at the finest scale.

`Level` — Level
`1` (default) | integer

Level of coefficients to extract or reconstruct, specified as an integer in the range `[1,`N`]`, where N is the length of `cd`.

Example: `y = lwtcoef(ca,cd,'LiftingScheme',lsc,'Level',3)` uses the lifting scheme `lsc` to extract the approximation coefficients at level 3.

Data Types: `double`

`Int2Int` — Handling integer-valued data

`false` or `0` (default) | `true` or `1`

Handling integer-valued data, specified as one of these:

- `1` (`true`) — Preserve integer-valued data
- `0` (`false`) — Do not preserve integer-valued data

`Int2Int` must match the value you used to generate `ca` and `cd`.

Example: `y = lwtcoef(ca,cd,Int2Int=true)` preserves integer-valued data.

`Extension` — Extension mode

`"periodic"` (default) | `"zeropad"` | `"symmetric"`

Extension mode to use to extract or reconstruct the coefficients, specified as one of these:

- `"periodic"` — Periodized extension
- `"zeropad"` — Zero extension
- `"symmetric"` — Symmetric extension

This argument specifies how to extend the signal at the boundaries. The extension mode must match the value you used to generate `ca` and `cd`.

Example: `y = lwtcoef(ca,cd,Extension="zeropad")` specifies zero extension.

Output Arguments

`y` — Extracted coefficients or projection

vector | matrix

Extracted coefficients or projection, returned as a vector or matrix. If `ca` is a scalar or vector, and the elements of `cd` are vectors, then `y` is a vector. If `ca` and the elements of `cd` are matrices, then `y` is a matrix, where each column is the extraction or projection of the corresponding columns in `ca` and `cd`.

Data Types: `single` | `double`

Compatibility Considerations

`lwtcoef` input syntax has changed

Behavior changed in R2021a

The `lwtcoef` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`Y = lwtcoef(TYPE,XDEC,LS,LEVEL,LEVEXT)`	Errors	`Y = lwtcoef(CA,CD,Name,Value)` with the lifting decomposition `CA` and `CD` in place of `XDEC`, and the following name-value arguments: • Replace `LS` with `'LiftingScheme'`, where `'LiftingScheme'` is a `liftingScheme` object. • Replace `LEVEXT` with `'Level'`. • Replace `TYPE` with the `Type` and `OutputType` name-value arguments. • `LEVEL` is no longer needed.	According to the value of `TYPE`, set the `Type` and `OutputType` name-value arguments as listed: • `'a'` — `'Type','approximation'` and `'OutputType','projection'` • `'ca'` — `'Type','approximation'` and `'OutputType','coefficients'` • `'d'` — `'Type','detail'` and `'OutputType','projection'` • `'cd'` — `'Type','detail'` and `'OutputType','coefficients'`

Extended Capabilities

C/C++ Code Generation

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

See Also

`lwt` | `ilwt` | `liftingScheme`

Introduced in R2021a

lwtcoef2

Extract 2-D LWT wavelet coefficients and orthogonal projections

Syntax

```
y = lwtcoef2(ll,lh,hl,hh)
y = lwtcoef2(ll,lh,hl,hh,Name=Value)
```

Description

`y = lwtcoef2(ll,lh,hl,hh)` returns the level 1 reconstructed approximation coefficients that correspond to the approximation coefficients `ll` and the horizontal (`lh`), vertical (`hl`), and diagonal (`hh`) wavelet coefficients. The coefficients in `ll`, `lh`, `hl`, and `hh` are the outputs of `lwt2` using default values.

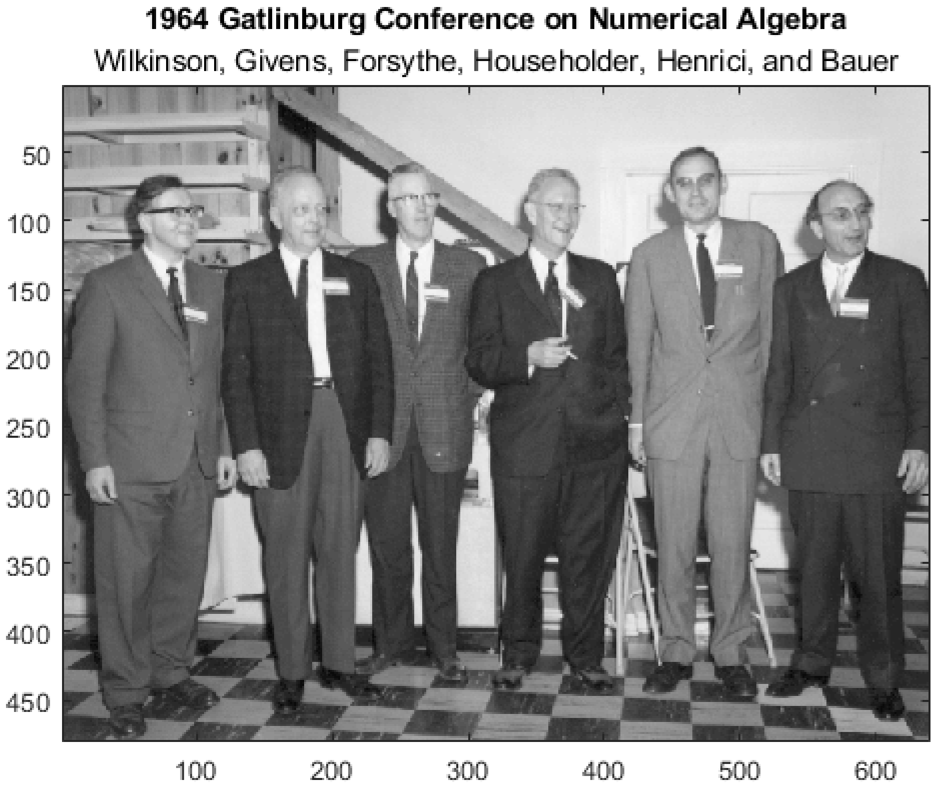
`y = lwtcoef2(ll,lh,hl,hh,Name=Value)` specifies options using one or more name-value arguments. For example, `lwtcoef2(ll,lh,hl,hh,Type="detail",OutputType="projection")` returns the projection of the detail coefficients at the finest scale using the `db1` wavelet.

Examples

Reconstruct Image from Orthogonal Projections

Load and plot a grayscale image.

```
load gatlin
figure
image(X)
colormap(map)
title("1964 Gatlinburg Conference on Numerical Algebra",...
    "Wilkinson, Givens, Forsythe, Householder, Henrici, and Bauer")
```

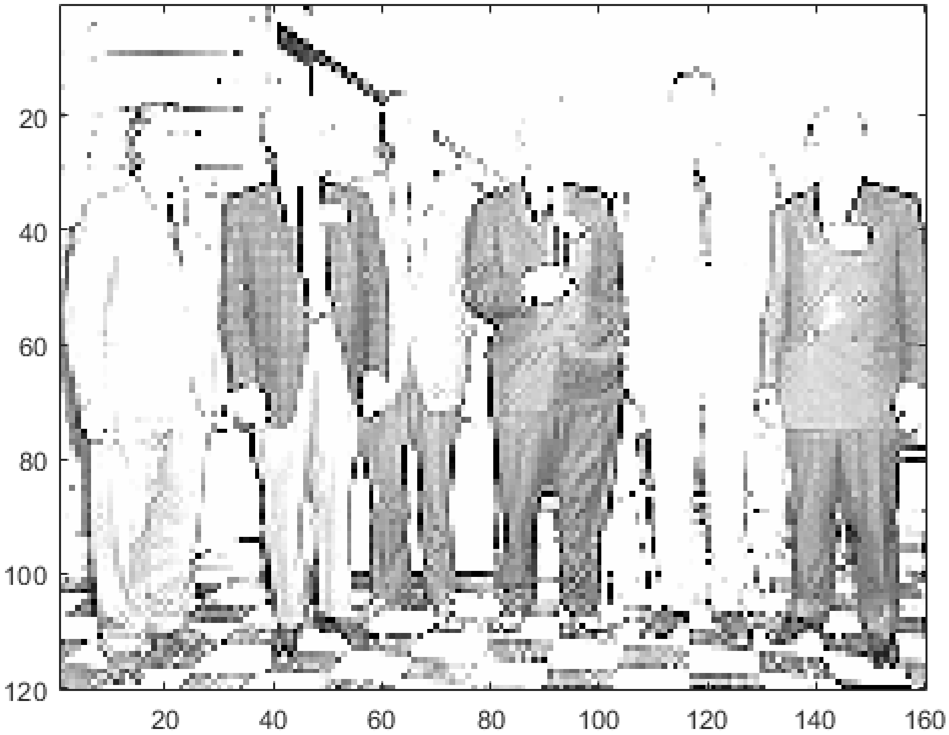

1964 Gatlinburg Conference on Numerical Algebra
Wilkinson, Givens, Forsythe, Householder, Henrici, and Bauer
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600

Create a lifting scheme associated with the `bior3.7` wavelet. Use the lifting scheme to obtain the wavelet decomposition of the image to the maximum level.

```
lscheme = liftingScheme(Wavelet="bior3.7");
[ll,lh,hl,hh] = lwt2(X,LiftingScheme=lscheme);
```

Extract and display the approximation coefficients at level 2. Confirm the row and column dimensions are one-quarter the size of those of the original image.

```
approxCF = lwtcoef2(ll,lh,hl,hh,...
    LiftingScheme=lscheme,OutputType="coefficients",Level=2);
figure
image(approxCF)
colormap(map)
```

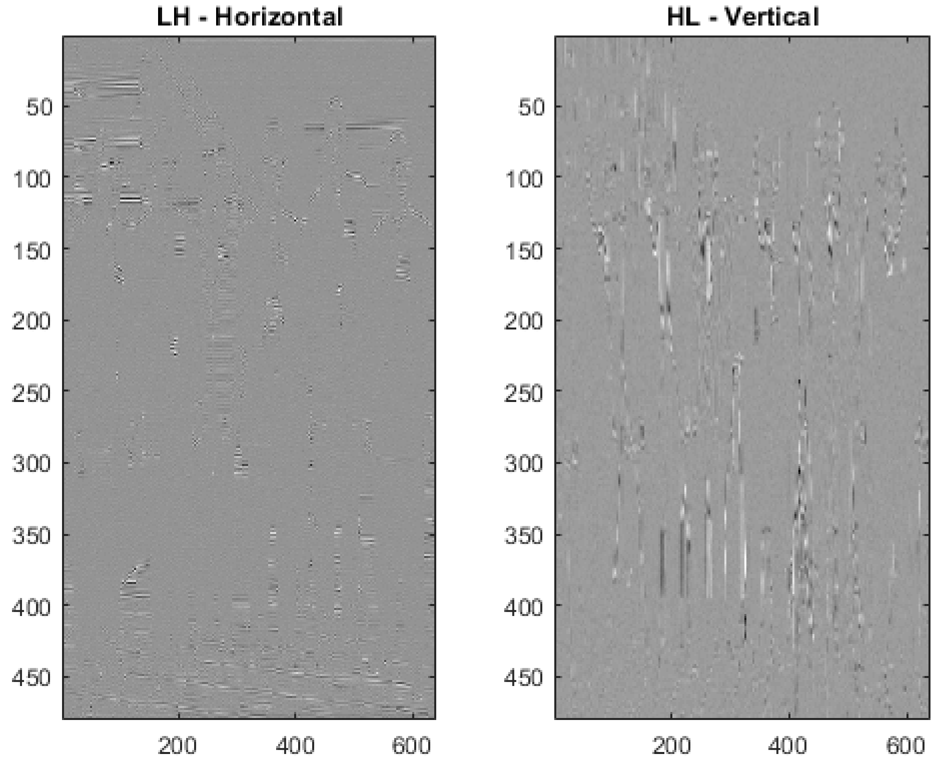


```
size(X)./size(approxCF)
```

```
ans = 1×2

     4     4
```

Obtain the orthogonal projection of the level 1 approximation coefficients. Also obtain the orthogonal projections of the detail coefficients at level 1. Display the projections corresponding to the LH and HL detail coefficients. Observe that the prominent features in the LH- and HL-derived images correspond to the horizontal, and vertical features, respectively, of the original image.

```
approx = lwtcoef2(ll,lh,hl,hh,...
    LiftingScheme=lscheme,OutputType="projection",Level=1);
dLH = lwtcoef2(ll,lh,hl,hh,...
    LiftingScheme=lscheme,OutputType="projection",Level=1,Type="LH");
dHL = lwtcoef2(ll,lh,hl,hh,...
    LiftingScheme=lscheme,OutputType="projection",Level=1,Type="HL");
dHH = lwtcoef2(ll,lh,hl,hh,...
    LiftingScheme=lscheme,OutputType="projection",Level=1,Type="HH");
subplot(1,2,1)
imagesc(dLH)
title("LH - Horizontal")
subplot(1,2,2)
imagesc(dHL)
title("HL - Vertical")
```



Confirm the sum of the four projections equals the original image.

```
max(max(abs(X-(approx+dLH+dHL+dHH))))
```

```
ans = 2.3448e-13
```

Input Arguments

`ll` — Approximation coefficients

scalar | vector | matrix

Approximation coefficients at the coarsest scale, specified as a scalar, vector, or matrix. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`lh` — Horizontal detail coefficients

cell array

Horizontal detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `lh` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`hl` — Vertical detail coefficients

cell array

Vertical detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hl` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

`hh` — Diagonal detail coefficients

cell array

Diagonal detail coefficients by level, specified as a *LEV*-by-1 cell array, where *LEV* is the level of the decomposition. The elements of `hh` are in order of decreasing resolution. The coefficients are the output of `lwt2`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `y = lwtcoef2(ll,lh,hl,hh,OutputType="projection",LiftingScheme=lscheme)`

`Wavelet` — Wavelet

`"db1"` (default) | character vector | string scalar

Orthogonal or biorthogonal wavelet, specified as a character vector or string scalar. See the Wavelet property of `liftingScheme` for the list of supported wavelets. The specified wavelet must match the value that you used to obtain the coefficients `ll`, `lh`, `hl`, and `hh`.

You cannot specify `Wavelet` and `LiftingScheme` at the same time.

Example: `y = lwtcoef2(ll,lh,hl,hh,Wavelet="bior3.5")` uses the `bior3.5` biorthogonal wavelet.

Data Types: `char` | `string`

`LiftingScheme` — Lifting scheme

`liftingScheme` object

Lifting scheme, specified as a `liftingScheme` object. The specified lifting scheme must be the same lifting scheme that you used to obtain the coefficients `ll`, `lh`, `hl`, and `hh`.

You cannot specify `LiftingScheme` and `Wavelet` at the same time.

Example: `y = lwtcoef2(ll,lh,hl,hh,LiftingScheme=lScheme)` uses the `lScheme` lifting scheme.

`OutputType` — Output type

`"coefficients"` (default) | `"projection"`

Output type, specified as one of these:

- `"coefficients"` — Extract the approximation or details coefficients
- `"projection"` — Return the projection (reconstruction) of the approximation or details coefficients

Example: `y = lwtcoef2(ll,lh,hl,hh,OutputType="projection",Type="detail")` returns the projection corresponding to the detail coefficients at the finest scale.

Type — Type of coefficients

`"ll"` (default) | `"lh"` | `"hl"` | `"hh"`

Type of coefficients to extract or reconstruct, specified as one of these:

- `"ll"` — Approximation coefficients
- `"lh"` — Horizontal coefficients
- `"hl"` — Vertical coefficients
- `"hh"` — Diagonal coefficients

Example: `y = lwtcoef2(ll,lh,hl,hh,Type="hh")` extracts the diagonal coefficients at the finest scale.

Level — Level

`1` (default) | positive integer

Level of coefficients to extract or reconstruct, specified as a positive integer less than or equal to `length(hh)`.

Example: `y = lwtcoef2(ll,lh,hl,hh,LiftingScheme=lsc,Level=3)` uses the lifting scheme `lsc` to extract the approximation coefficients at level 3.

Data Types: `double`

Extension — Extension mode

`"periodic"` (default) | `"zeropad"` | `"symmetric"`

Extension mode to use to extract or reconstruct the coefficients, specified as one of these:

- `"periodic"` — Periodized extension
- `"zeropad"` — Zero extension
- `"symmetric"` — Symmetric extension

This argument specifies how to extend the signal at the boundaries. The extension mode must match the value you used to generate `ll`, `lh`, `hl`, and `hh`.

Example: `y = lwtcoef2(ll,lh,hl,hh,Extension="zeropad")` specifies zero extension.

Int2Int — Handling integer-valued data

`false` or `0` (default) | `true` or `1`

Handling integer-valued data, specified as one of these:

- `1` (`true`) — Preserve integer-valued data
- `0` (`false`) — Do not preserve integer-valued data

`Int2Int` must match the value you used to generate `ll`, `lh`, `hl`, and `hh`.

Example: `y = lwtcoef2(ll,lh,hl,hh,Int2Int=true)` preserves integer-valued data.

Output Arguments

y — Extracted coefficients or projection

matrix

Extracted coefficients or projection, returns as a matrix. `y` has the same dimensionality as the input used by the `lwt2` function to generate the approximation and details coefficients.

Data Types: `single` | `double`

Compatibility Considerations

`lwtcoef2` input syntax has changed

Behavior changed in R2021b

The `lwtcoef2` input syntax has changed. Use name-value arguments instead.

Functionality	Result	Use Instead	Compatibility Considerations
`Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT)`	Errors	`Y = lwtcoef2(CA,CH,CV,CD,Name=Value)` with the lifting decomposition `CA`, `CH`, `CV`, and `CD` in place of `XDEC`, and the following name-value arguments: • Replace `LS` with `LiftingScheme`, where `LiftingScheme` is a `liftingScheme` object. • Replace `LEVEXT` with `Level`. • Replace `TYPE` with the `Type` and `OutputType` name-value arguments. • `LEVEL` is no longer needed.	According to the value of `TYPE`, set the `Type` and `OutputType` name-value arguments as listed: • `'a'` — `Type="approximation"` and `OutputType="projection"` • `'ca'` — `Type="approximation"` and `OutputType="coefficients"` • `'d'` — `Type="detail"` and `OutputType="projection"` • `'cd'` — `Type="detail"` and `OutputType="coefficients"`

Extended Capabilities

C/C++ Code Generation

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

See Also

`lwt2` | `ilwt2` | `liftingScheme`

Introduced in R2021b

matchingPursuit

Recover sparse signal using matching pursuit algorithm

Syntax

```
[Xr,YI,I,R] = matchingPursuit(A,Y)
[Xr,YI,I,R] = matchingPursuit( ___ ,Name=Value)
```

Description

`[Xr,YI,I,R] = matchingPursuit(A,Y)` recovers the sparse signal `Xr` using the `sensingDictionary` `A` and sensor measurement `Y`. By default, the sparse recovery algorithm is matching pursuit. The `I` output is the support of `Xr` identified by the matching pursuit algorithm. The `YI` output is the best fit for `Y` corresponding to the bases indexed by the elements of `I`, and `R` is the residual.

`[Xr,YI,I,R] = matchingPursuit( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the input argument in the previous syntax. For example, `[Xr,YI,I,R] = matchingPursuit(A,Y,Algorithm="WMP")` specifies the weak matching pursuit algorithm.

Examples

Obtain Sparse Approximation

Load a signal.

```
load cuspamax
```

Create a sensing dictionary consisting of the basis types `poly` and `walsh` that can be applied to the signal.

```
lsig = length(cuspamax);
A = sensingDictionary(Size=lsig,Type={'poly','walsh'})
```

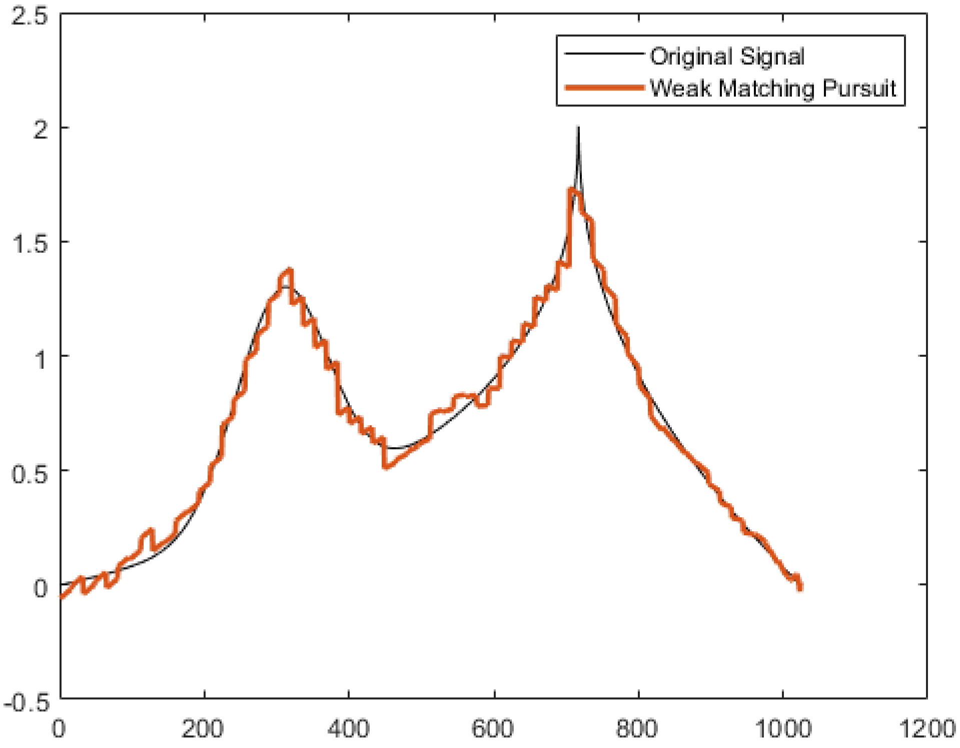
```
A = 
  sensingDictionary with properties:

                Type: {'poly'  'walsh'}
                Name: {''  ''}
               Level: [0 0]
    CustomDictionary: []
                Size: [1024 2048]
```

Use the dictionary to obtain a sparse approximation of the signal using weak matching pursuit.

```
[Xr,YI,I,R] = matchingPursuit(A,cuspamax, ...
    Algorithm="WMP",maxerr={"L2",1});
```

Plot the original signal and the approximation.

```
plot(cuspamax,"k")
hold on
plot(YI,LineWidth=2)
legend("Original Signal","Weak Matching Pursuit")
hold off
```



Extract the vectors from the dictionary that correspond to the approximation. Multiply with the associated coefficients and confirm the result is equal to the approximation.

```
Amat = subdict(A,1:1024,I);
x = Amat*Xr(I,:);
max(abs(x-YI))
```

```
ans = 0
```

Input Arguments

A — Sensing dictionary

`sensingDictionary` object

Sensing dictionary, specified as a `sensingDictionary` object.

Y — Sensor measurements

vector

Sensor measurements, specified as a vector `Y` such that `Y = AX`, where `X` is a sparse signal.

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

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Xr = matchingPursuit(D,Y,Algorithm="WMP")` recovers the sparse signal using weak matching pursuit.

`Algorithm` — Recovery algorithm

`"BMP"` (default) | `"OMP"` | `"WMP"`

Recovery algorithm, specified as one of these:

- `"BMP"` — Matching pursuit
- `"OMP"` — Orthogonal matching pursuit
- `"WMP"` — Weak matching pursuit

For more information, see “Matching Pursuit Algorithms”.

Example: `[Xr,YI,I,R] = matchingPursuit(D,Y,Algorithm="OMP")` recovers the sparse signal using orthogonal matching pursuit.

`wmpcfs` — Optimality factor

`0.6` (default) | positive scalar

Optimality factor for weak orthogonal matching pursuit, specified as a scalar in the interval (0,1]. This option is valid only when `Algorithm` is `"WMP"`.

Example: `[Xr,YI,I,R] = matchingPursuit(D,Y,Algorithm="WMP",wmpcfs=0.7)` specifies an optimality factor of `0.7`.

Data Types: `single` | `double`

`maxIterations` — Maximum number of iterations

`25` (default) | positive integer

Maximum number of iterations to recover the sparse signal, specified as a positive integer. The pursuit algorithm stops if the number of iterations reaches `maxIterations`. Note that the number of iterations `matchingPursuit` performs to recover the sparse signal is equal to the length of the index vector `I`.

Example: `[Xr,YI,I,R] = matchingPursuit(D,Y,maxIterations=15)` iterates at most 15 times to recover the sparse signal.

Data Types: `single` | `double`

`maxerr` — Maximum error criteria

`{"L2",1}` (default) | cell

Maximum error criteria used to recover the sparse signal, specified as cell array `{NORME,ME}`. `NORME` specifies the norm used in the error computation. Valid options are `"L1"`, `"L2"`, and `"Linf"`. `ME` is a positive scalar in the interval (0,100] that specifies the maximum percentage of the relative admissible value.

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 `maxIterations`.
- The relative error falls below the percentage you specify with the `maxerr` name-value argument.

Example: `[Xr,YI,I,R] = matchingPursuit(D,Y,maxerr={"L1",20})` specifies the L1 norm and a relative error of 20%.

Data Types: `cell`

Output Arguments

`Xr` — Sparse signal

vector

Sparse signal recovered, returned as a vector.

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

`YI` — Best fit

vector

Best fit to the sensor measurements, returned as a vector. `YI` is the best fit for `Y` corresponding to the bases indexed by the elements of `I`. The best fit is defined as `YI = A(:,I)*Xr(I,:)`.

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

`I` — Index

vector

Index of basis elements identified by the matching pursuit algorithm, returned as a vector. For matching pursuit algorithms, the length of `I` corresponds to the number of iterations the algorithm needed before termination.

Data Types: `double`

`R` — Residual

vector

Residual, returned as a vector. The vectors `R` and `Y` are equal in size. The residual is defined as `R = Y-(A(:,I)*Xr(I,:)) = Y-YI`.

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

See Also

`basisPursuit` | `sensingDictionary`

Topics

"Signal Deconvolution and Impulse Denoising Using Pursuit Methods"
"Matching Pursuit Algorithms"

Introduced in R2022a

mdwtcluster

Multisignals 1-D clustering

Syntax

```
s = mdwtcluster(x)
s = mdwtcluster( ___ ,Name,Value)
```

Description

`s = mdwtcluster(x)` clusters data using hierarchical clustering. The input matrix `x` is decomposed in the row direction using the discrete wavelet transform (DWT) with the Haar wavelet and the maximum allowed level `fix(log2(size(x,2)))`.

Note `mdwtcluster` requires Statistics and Machine Learning Toolbox™.

`s = mdwtcluster( ___ ,Name,Value)` specifies options using name-value pair arguments in addition to the input argument in the previous syntax. For example, `'level',4` specifies the decomposition level.

Examples

Cluster 1-D Multisignal

Load the 1-D multisignal `elecsig10`.

```
load elecsig10
```

Compute the structure resulting from multisignal clustering.

```
lst2clu = {'s','ca1','ca3','ca6'};
S = mdwtcluster(signals,'maxclust',4,'lst2clu',lst2clu)
```

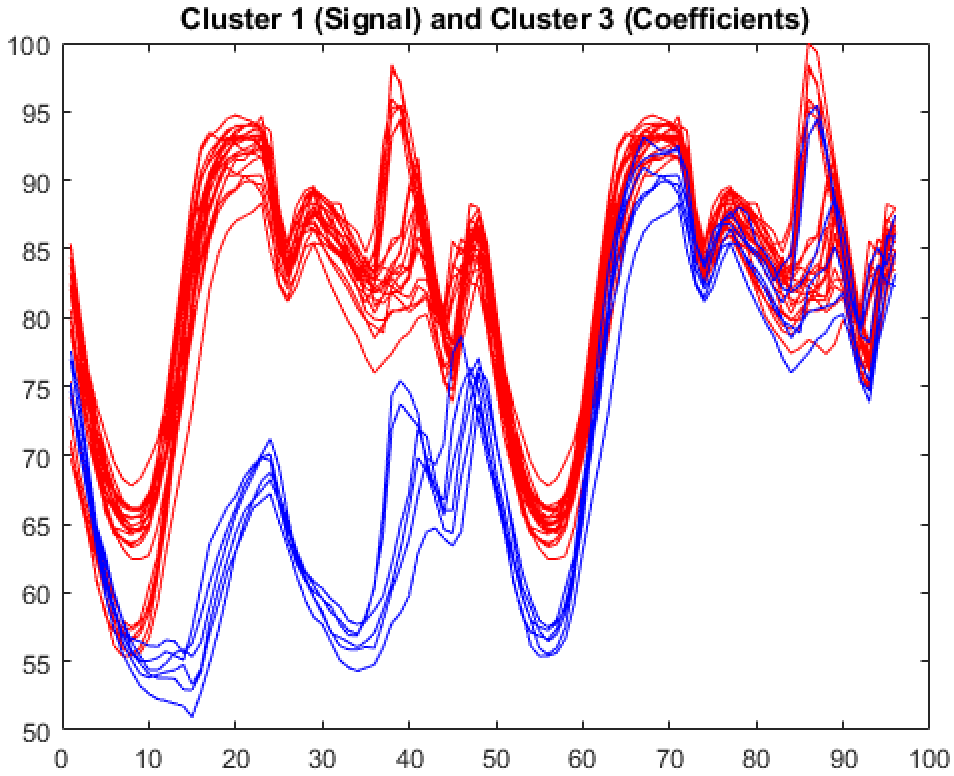
```
S = struct with fields:
    IdxCLU: [70x4 double]
    Incons: [69x4 double]
      Corr: [0.7920 0.7926 0.7947 0.7631]
```

Retrieve the cluster indices.

```
IdxCLU = S.IdxCLU;
```

Plot the first and third clusters.

```
plot(signals(IdxCLU(:,1)==1,:)','r')
hold on
plot(signals(IdxCLU(:,1)==3,:)','b')
hold off
title('Cluster 1 (Signal) and Cluster 3 (Coefficients)')
```



Check the equality of partitions. Confirm we obtain the same partitions using coefficients of approximation at level 3 instead of the original signals. Much less information is then used.

```
equalPART = isequal(IdxCLU(:,1),IdxCLU(:,3))
```

```
equalPART = logical
   1
```

Input Arguments

x — Input data

matrix

Input data, specified as a matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `s = mdwtcluster(signals,'maxclust',4,'wname','db4')` specifies four clusters and the wavelet `db4`.

`dirDec` — Direction of decomposition

`'r'` (default) | `'c'`

Direction of decomposition, specified as `'r'` (row) or `'c'` (column).

`level` — Level of DWT decomposition

`fix(log2(size(x,d)))` (default) | positive integer

Level of DWT decomposition, specified as a positive integer. The default value is `fix(log2(size(x,d)))`, where `d=1` or `d=2`, depending on the `dirDec` value.

`wname` — Wavelet

`'haar'` (default) | character vector | string scalar

Wavelet used for the DWT, specified as a character vector or string scalar. The default value is the Haar wavelet, `'haar'`.

`dwtEXTM` — DWT extension mode

character vector | string scalar

DWT extension mode, specified as a character vector or string scalar. See `dwtmode`.

`pdist` — Distance metric

`'euclidean'` (default) | character vector | string scalar | function handle

Distance metric, specified as a character vector, string scalar, or function handle. The default value is `'euclidean'`. See `pdist`.

`linkage` — Algorithm for computing the distance between clusters

`'ward'` (default) | `'average'` | `'centroid'` | `'complete'` | ...

Algorithm for computing the distance between clusters, specified as one of the values in this table.

Method	Description
`'average'`	Unweighted average distance (UPGMA)
`'centroid'`	Centroid distance (UPGMC), appropriate for Euclidean distances only
`'complete'`	Farthest distance
`'median'`	Weighted center of mass distance (WPGMC), appropriate for Euclidean distances only
`'single'`	Shortest distance
`'ward'`	Inner squared distance (minimum variance algorithm), appropriate for Euclidean distances only
`'weighted'`	Weighted average distance (WPGMA)

See `linkage`.

`maxclust` — Number of clusters

6 (default) | integer | vector

Number of clusters, specified as an integer or vector.

`lst2clu` — Cell array that contains the list of data to classify

cell array | string vector

Cell array of character vectors or string vector which contains the list of data to classify. If *N* is the level of decomposition, the allowed name values for the cells are:

- `'s'` — Signal
- `'a`*`j`*`'` — Approximation at level *j*
- `'d`*`j`*`'` — Detail at level *j*
- `'ca`*`j`*`'` — Coefficients of approximation at level *j*
- `'cd`*`j`*`'` — Coefficients of detail at level *j*

with *`j`*` = 1, …, `*`N`*.

The default value is `{'s';'ca1';...;'ca`*`N`*`'}` or `["s" "cal" ... "caN"]`.

Output Arguments

`s` — Output structure

structure

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`.
`S.Incons(:,`*`j`*`)`	Contains the inconsistent values of each non-leaf node in the hierarchical cluster tree. See `inconsistent`.
`S.Corr(`*`j`*`)`	Contains the cophenetic correlation coefficients of the partition. See `cophenet`.

Note If `maxclust` 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.

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 1-D discrete wavelet decomposition at level `lev` of each row or each column of the matrix `x`, using the wavelet `wname`.

`dec = mdwtdec(dirdec,x,lev,LoD,HiD,LoR,HiR)` uses the specified lowpass and highpass wavelet decomposition filters `LoD` and `HiD`, respectively, and the lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

`dec = mdwtdec(___,'mode',extmode)` uses the specified discrete wavelet transform (DWT) extension mode `extmode`. For more information, see `dwtmode`. This syntax can be used with any of the previous syntaxes.

Examples

Decompose Multisignals

This example shows how to return the wavelet decomposition of a multisignal using a wavelet name and wavelet filters.

Load the 23 channel EEG data `Espiga3` [4]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
size(Espiga3)
```

```
ans = 1×2

   995    23
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
```

```
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

Compute the filters associated with the db2 wavelet.

```
[LoD,HiD,LoR,HiR] = wfilters('db2');
```

Perform a decomposition at level 2 using the filters.

```
decBIS = mdwtdec('c',Espiga3,2,LoD,HiD,LoR,HiR)
```

```
decBIS = struct with fields:
        dirDec: 'c'
         level: 2
         wname: ''
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

Confirm the approximation and detail coefficients of both decompositions are identical.

```
max(abs(dec.ca(:)-decBIS.ca(:)))
```

```
ans = 0
```

```
max(abs(dec.cd{1}(:)-decBIS.cd{1}(:)))
```

```
ans = 0
```

```
max(abs(dec.cd{2}(:)-decBIS.cd{2}(:)))
```

```
ans = 0
```

Input Arguments

dirdec — Direction indicator

'r' | 'c'

Direction indicator of the wavelet decomposition, specified as:

- 'r': Take the 1-D wavelet decomposition of each row of x
- 'c': Take the 1-D wavelet decomposition of each column of x

x — Input data

real-valued matrix

Input data, specified as a real-valued matrix.

lev — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer. mdwtdec does not enforce a maximum level restriction. Use wmaxlev to ensure that the wavelet coefficients are free from boundary effects. If boundary effects are not a concern, a good rule is to set lev less than or equal to fix(log2(length(*N*))), where *N* is the number of samples in the 1-D data.

wname — Analyzing wavelet
character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. 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', 'db*N*', 'fk*N*', 'coif*N*', or 'sym*N*', where *N* is the number of vanishing moments for all families except fk. For fk, *N* is the number of filter coefficients.
- Valid biorthogonal wavelet families begin with 'bior*Nr*.*Nd*' or 'rbio*Nd*.*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).

LoD,HiD — Wavelet decomposition filters
even-length real-valued vectors

Wavelet decomposition filters, specified as a pair of even-length real-valued vectors. LoD is the lowpass decomposition filter, and HiD is the highpass decomposition filter. The lengths of LoD and HiD must be equal. See wfilters for additional information.

LoR,HiR — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter. The lengths of LoR and HiR must be equal. See wfilters for additional information.

extmode — Extension mode
'zpd' | 'sp0' | 'spd' | ...

Extension mode used when performing the wavelet decomposition, specified as:

mode	DWT Extension Mode
'zpd'	Zero extension
'sp0'	Smooth extension of order 0
'spd' (or 'sp1')	Smooth extension of order 1
'sym' or 'symh'	Symmetric extension (half point): boundary value symmetric replication
'symw'	Symmetric extension (whole point): boundary value symmetric replication

mode	DWT Extension Mode
'asym' or 'asymh'	Antisymmetric extension (half point): boundary value antisymmetric replication
'asymw'	Antisymmetric extension (whole point): boundary value antisymmetric replication
'ppd', 'per'	Periodized extension If the signal length is odd and mode is 'per', an extra sample equal to the last value is added to the right and the extension is performed in 'ppd' mode. If the signal length is even, 'per' is equivalent to 'ppd'. This rule also applies to images.

The global variable managed by dwtmode specifies the default extension mode. Use dwtmode to determine the extension modes.

Output Arguments

dec — Wavelet decomposition
structure

Wavelet decomposition of the multisignal x, returned as a structure with the following fields:

- dirDec — Direction indicator: 'r' (row) or 'c' (column)
- level — Level of wavelet decomposition
- wname — Wavelet name
- dwtFilters — Structure with four fields: LoD, HiD, LoR, and HiR
- dwtEXTM — DWT extension mode
- 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

The coefficients ca and cd{*k*}, for *k* from 1 to lev, are matrices and are stored in rows if dirdec = 'r' or in columns if dirdec = 'c'.

References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

[4] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés,

3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input wname must be constant.
- The input lev must be defined as a scalar during compilation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input wname must be constant.
- The level of decomposition, lev must be scalar and a compile-time constant.

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')
d = mdwtrec(dec,'d')
ca = mdwtrec(dec,'ca')

cd = mdwtrec(dec,'cd',mode)
cfs = mdwtrec(dec,'cfs',mode)

y = mdwtrec( ___ ,idxsig)
```

Description

`x = mdwtrec(dec)` reconstructs the original matrix of signals from the wavelet decomposition structure `dec`.

`x = mdwtrec(dec,idxsig)` reconstructs the signals whose indices are specified in the vector `idxsig`.

`y = mdwtrec(dec,type,lev)` extracts or reconstructs the detail or approximation coefficients at level `lev` depending on the value of `type`.

`a = mdwtrec(dec,'a')` returns the reconstructed approximation coefficients.

`d = mdwtrec(dec,'d')` returns a matrix containing the sum of all the details, so that `x = a + d`.

`ca = mdwtrec(dec,'ca')` returns a matrix containing the extracted approximation coefficients.

`cd = mdwtrec(dec,'cd',mode)` returns a matrix containing all the detail coefficients concatenated in the order specified by `mode`.

`cfs = mdwtrec(dec,'cfs',mode)` returns a matrix containing all the coefficients in the order specified by `mode`.

`y = mdwtrec( ___ ,idxsig)` extracts or reconstructs the coefficients whose indices are specified in the vector `idxsig`.

Examples

Reconstruct Multisignals

This example shows how to reconstruct a multisignal and a user-specified signal within the multisignal.

Load the 23 channel EEG data `Espiga3` [4]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
size(Espiga3)
```

```
ans = 1×2

   995    23
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2');
```

Reconstruct the original matrix of signals using the decomposition structure `dec`.

```
XR = mdwtrec(dec);
```

Compute the reconstruction error.

```
errREC = max(abs(Espiga3(:)-XR(:)))
```

```
errREC = 3.5442e-10
```

Reconstruct the original signal at index 17, the corresponding approximation at level 2, and details at levels 1 and 2.

```
idx = 17;
Y = mdwtrec(dec,idx);
A2 = mdwtrec(dec,'a',2,idx);
D2 = mdwtrec(dec,'d',2,idx);
D1 = mdwtrec(dec,'d',1,idx);
```

Compute the reconstruction error for signal 17.

```
errREC = max(abs(Y-A2-D2-D1))
```

```
errREC = 1.3242e-17
```

Input Arguments

dec — Wavelet decomposition

structure

Wavelet decomposition of a multisignal, specified as a structure with the following fields:

- `dirDec` — Direction indicator: `'r'` (row) or `'c'` (column)
- `level` — Level of wavelet decomposition
- `wname` — Wavelet name
- `dwtFilters` — Structure with four fields: `LoD`, `HiD`, `LoR`, and `HiR`
- `dwtEXTM` — DWT extension mode
- `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`

The format of `dec` matches the output of `mdwtdec`.

`idxsig` — Indices

positive integer-valued vector

Indices of signals to reconstruct, specified as a positive integer-valued vector.

Example: If *S* is a matrix of 100 signals and `dec = mdwtdec('r',S,3,'db2')`, then `mdwtrec(dec,[1 20 98])` reconstructs the signals whose row indices are 1, 20, and 98.

`lev` — Level

nonnegative integer

Level of coefficients to extract or reconstruct, specified as a nonnegative integer.

- If `type` is `'a'` or `'ca'`, then `lev` must be an integer in the interval [0, `levdec`], where `levdec = dec.level`.
- If `type` is `'d'` or `'cd'`, then `lev` must be an integer in the interval [1, `levdec`], where `levdec = dec.level`.

`type` — Output type

`'cd'` | `'ca'` | `'d'` | `'a'`

Output type, specified as one of the following:

- `'cd'` – detail coefficients of level `lev` are extracted
- `'d'` – detail coefficients of level `lev` are reconstructed
- `'ca'` – approximation coefficients of level `lev` are extracted
- `'a'` – approximation coefficients of level `lev` are reconstructed

`mode` — Order of concatenation

`'descend'` (default) | `'ascend'`

Order of concatenation, specified as `'descend'` or `'ascend'`. For `mode = 'descend'`, the coefficients are concatenated from level `levdec` to level 1, where `levdec = dec.level`. If `mode = 'ascend'`, the coefficients are concatenated from level 1 to level `levdec`. The concatenation is made row-wise if `dec.dirDEC = 'r'` or column-wise if `dec.dirDEC = 'c'`.

Output Arguments

x — Reconstructed signals

real-valued matrix

Reconstructed signals, returned as a real-valued matrix.

y — Decomposition coefficients

real-valued matrix

Decomposition coefficients, returned as a real-valued matrix, depending on `type`:

- `'cd'` – extracted detail coefficients
- `'ca'` – extracted approximation coefficients
- `'d'` – reconstructed detail coefficients
- `'a'` – reconstructed approximation coefficients

a — Reconstructed approximation coefficients

real-valued matrix

Reconstructed approximation coefficients, returned as a real-valued matrix.

d — Reconstructed detail coefficients

real-valued matrix

Reconstructed detail coefficients, returned as a real-valued matrix.

ca — Extracted approximation coefficients

real-valued matrix

Extracted approximation coefficients, returned as a real-valued matrix.

cd — Extracted detail coefficients

real-valued matrix

Extracted detail coefficients, returned as a real-valued matrix.

`cfs` — Extracted approximation and detail coefficients

real-valued matrix

Extracted approximation and detail coefficients, returned as a real-valued matrix.

References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

[4] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `type` must be constant.

See Also

`mdwtdec` | `waverec`

Introduced in R2007a

measerr

Quality metrics of signal or image approximation

Syntax

```
[PSNR,MSE,MAXERR,L2RAT] = measerr(X,XAPP)
[PSNR,MSE,MAXERR,L2RAT] = measerr(X,XAPP,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`.

`[PSNR,MSE,MAXERR,L2RAT] = measerr(X,XAPP,BPS)` uses the bits per sample, `BPS`, to determine the peak signal-to-noise ratio.

Examples

Measure Approximation Quality in RGB Image

Approximate an RGB image and compute the quality metrics.

Load an RGB image. Return the image dimensions and minimum and maximum values.

```
X = imread('africasculpt.jpg');
size(X)
```

```
ans = 1×3

   512   512     3
```

```
[min(X(:)) max(X(:))]
```

```
ans = 1x2 uint8 row vector

     0   236
```

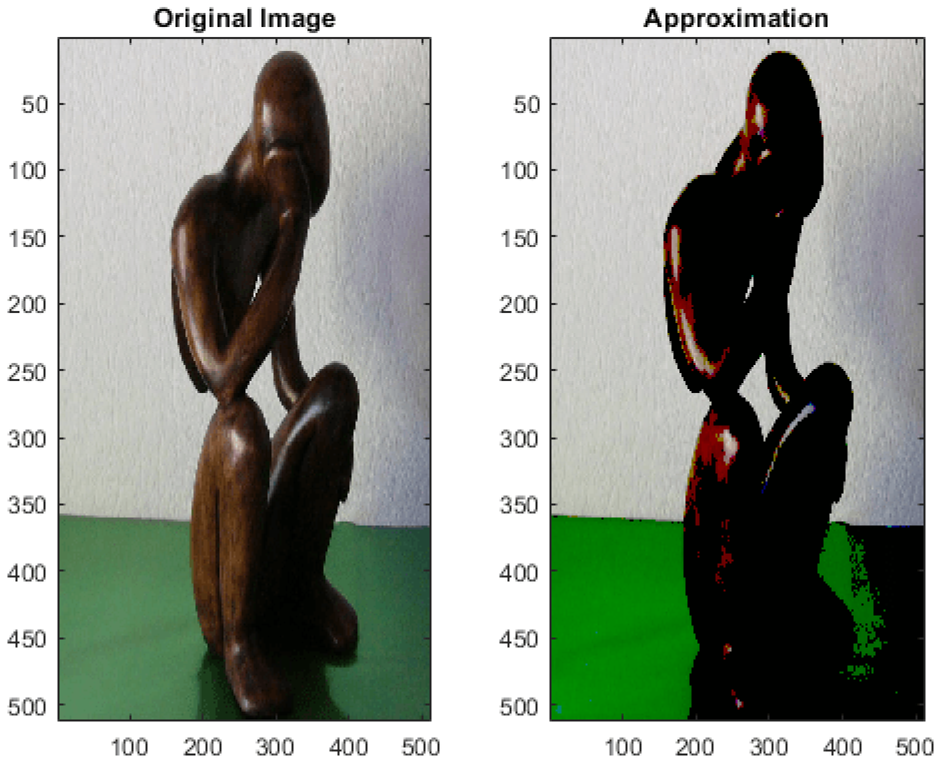
Define the image approximation by setting equal to 1 all RGB values less than or equal to 100.

```
Xapp = X;
Xapp(X<=100) = 1;
```

Display the image and its approximation.

```
subplot(1,2,1)
image(X)
title('Original Image')
subplot(1,2,2)
```

```
image(Xapp)
title('Approximation')
```



Compute the quality metrics of the image approximation.

```
[psnr,mse,maxerr,L2rat] = measerr(X,Xapp)
```

```
psnr = 17.5287
```

```
mse = 1.1487e+03
```

```
maxerr = 99
```

```
L2rat = 0.9398
```

Measure Approximation Quality in Grayscale Image

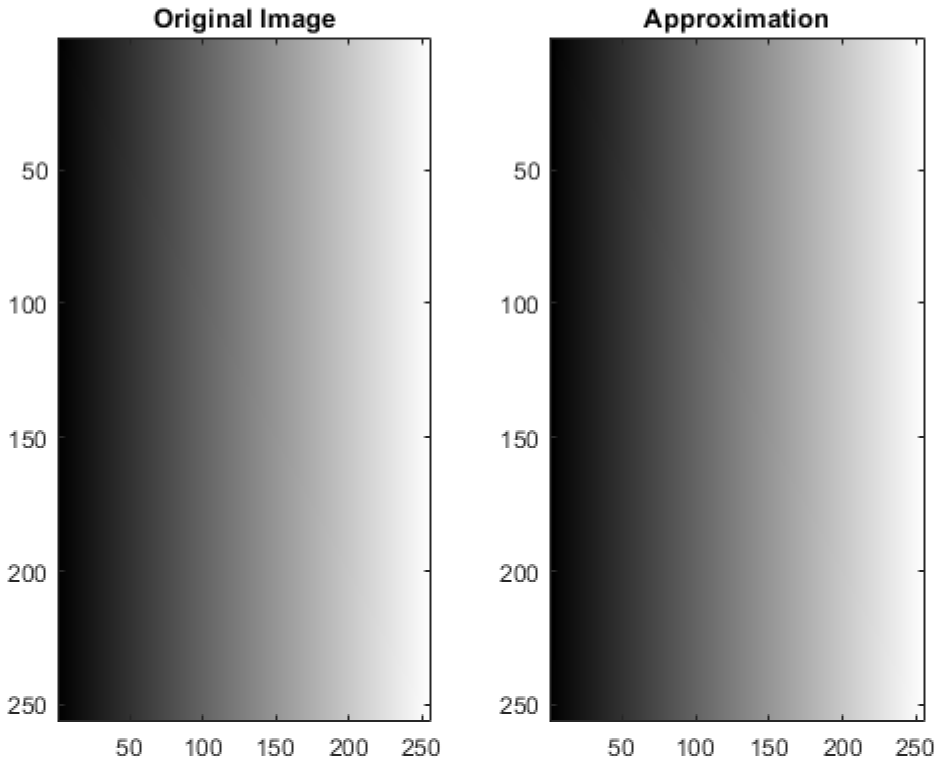
Approximate a grayscale image and calculate approximation quality metrics.

Create a 256-by-256 grayscale image with intensities between 0 and $2^{16} - 1$.

```
val = 0:2^16-1;
X = reshape(val,256,256);
```

There are 16 bits per sample. Define the image approximation by setting equal to 1 all grayscale values less than or equal to 1000. Display the image and its approximation.

```
Xapp = X;
Xapp(X<=1000) = 1;
colormap(gray(2^16))
subplot(1,2,1)
image(X)
title('Original Image')
subplot(1,2,2)
image(Xapp)
title('Approximation')
```



There are 16 bits per sample. Compute the quality metrics of the grayscale approximation.

```
bps = 16;
[psnr,mse,maxerr,L2rat] = measerr(X,Xapp)
```

```
psnr = 11.0733

mse = 5.0786e+03

maxerr = 999

L2rat = 1.0000
```

Input Arguments

X — Input signal or image

real-valued array

Input signal or image, specified as a real-valued array.

XAPP — Approximation of signal or image
real-valued array

Approximation of signal or image X, specified as a real-valued array. XAPP is the same size as X.

BPS — Bits per sample
8 (default) | positive integer

Bits per sample of the input data, specified as a positive integer. The default value is 8, so the maximum possible pixel value of an image (MAXI) is 255. More generally, when samples are represented using linear Pulse Code Modulation with B bits per sample, MAXI is 2^B-1.

Output Arguments

PSNR — Peak signal-to-noise ratio
positive real number

Peak signal-to-noise ratio (PSNR) in decibels, returned as a positive real number. 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 — Mean square error
positive real number

Mean square error, returned as a positive real number. MSE is the squared norm of the difference between X and XAPP divided by the number of elements.

MAXERR — Maximum absolute squared deviation
positive real number

Maximum absolute squared deviation of the data X from the approximation XAPP, returned as a positive real number.

L2RAT — Energy ratio
positive real number

Energy ratio between the approximation XAPP and input data X, returned as a positive real number. L2RAT is the ratio of the squared norm of XAPP to X.

More About

Peak Signal to Noise Ratio

The peak signal-to-noise ratio (PSNR) in decibels between a signal and its approximation is

$$20\log_{10}\left(\frac{2^B-1}{\sqrt{MSE}}\right)$$

where MSE represents the mean square error, and B represents the bits per sample.

Mean Square Error

The mean square error (MSE) between a signal or image, *X*, and an approximation, *Y*, is

$$\frac{||X-Y||^2}{N}$$

where *N* is the number of elements in the signal.

References

[1] Huynh-Thu, Q. and M. Ghanbari. "Scope of Validity of PSNR in Image/Video Quality Assessment." *Electronics Letters*. Vol. 44, Issue 13, 2008, pp. 800–801.

See Also

`wdenoise` | `wden` | `wdencmp`

Topics

"Wavelet Data Compression"
"Wavelet Denoising and Nonparametric Function Estimation"

Introduced in R2010b

merge

Merge two or more labeled signal sets

Syntax

```
lssnew = merge(lss1,...,lssN)
```

Description

`lssnew = merge(lss1,...,lssN)` merges *N* labeled signal set objects, `lss1,...,lssN`, and returns a labeled signal set `lssnew` containing all the members and label values of the input sets.

Examples

Merge Labeled Signal Sets

Load a labeled signal set containing recordings of whale songs. Display the names of the set's members and a summary of its label definitions.

```
load whales

getMemberNames(lss)
```

```
ans = 2x1 string
    "Member{1}"
    "Member{2}"
```

```
labelDefinitionsSummary(lss)
```

```
ans=3×9 table
      LabelName         LabelType      LabelDataType      Categories      ValidationFunction     Defau
    ______________    ___________    _____________    ____________    __________________    _____

    "WhaleType"       "attribute"    "categorical"    {3x1 string}       {["N/A"   ]}       {0x0
    "MoanRegions"     "roi"          "logical"        {["N/A"   ]}       {0x0 double}       {0x0
    "TrillRegions"    "roi"          "logical"        {["N/A"   ]}       {0x0 double}       {0x0
```

Create a new signal set with the same data source, time information, and labels as `lss`. Remove the first member of the new set and change the name of the remaining one. Display the names of the new set's members.

```
newlss = copy(lss);

removeMembers(newlss,1)
setMemberNames(newlss,"YoungOne")

getMemberNames(newlss)
```

```
ans = 
"YoungOne"
```

Create a label definition that specifies whether a signal corresponds to a calf or to an adult whale. Add the definition to the new labeled signal set and label the member. Remove the label that specifies the moan regions. Display a summary of the new member's label definitions

```
calf = signalLabelDefinition('Calf','LabeldataType','logical','DefaultValue',false, ...
    'Description','Is the specimen a calf, or an adult?');

addLabelDefinitions(newlss,calf)
setLabelValue(newlss,1,"Calf",true)

removeLabelDefinition(newlss,"MoanRegions")
labelDefinitionsSummary(newlss)
```

```
ans=3×9 table
      LabelName         LabelType      LabelDataType     Categories      ValidationFunction     Defau
    ______________    ___________    _____________    ____________    __________________    _____

    "WhaleType"       "attribute"    "categorical"    {3x1 string}      {["N/A"   ]}        {0x0
    "TrillRegions"    "roi"          "logical"        {["N/A"   ]}      {0x0 double}        {0x0
    "Calf"            "attribute"    "logical"        {["N/A"   ]}      {0x0 double}        {[
```

Merge the two labeled signal sets. Verify that the merged set contains the members, definitions, and labels of the original sets.

```
lssmerge = merge(lss,newlss);

getMemberNames(lssmerge)
```

```
ans = 3x1 string
    "Member{1}"
    "Member{2}"
    "YoungOne"
```

```
labelDefinitionsSummary(lssmerge)
```

```
ans=4×9 table
      LabelName         LabelType      LabelDataType     Categories      ValidationFunction     Defau
    ______________    ___________    _____________    ____________    __________________    _____

    "WhaleType"       "attribute"    "categorical"    {3x1 string}      {["N/A"   ]}        {0x0
    "MoanRegions"     "roi"          "logical"        {["N/A"   ]}      {0x0 double}        {0x0
    "TrillRegions"    "roi"          "logical"        {["N/A"   ]}      {0x0 double}        {0x0
    "Calf"            "attribute"    "logical"        {["N/A"   ]}      {0x0 double}        {[
```

Input Arguments

`lss1,...,lssN` — Input labeled signal sets

`labeledSignalSet` objects

Input labeled signal sets, specified as `labeledSignalSet` objects. All input sets must have the same time information settings and data source type.

Output Arguments

`lssnew` — Merged labeled signal set

`labeledSignalSet` object

Merged labeled signal set, returned as a `labeledSignalSet` object. The set `lssnew` contains a signal source, label definitions, and label values that are independent of those in the input labeled signal sets.

- Changing any of the input labeled signal sets does not affect the merged labeled signal set.
- Changing the merged labeled signal set does not affect any of the input labeled signal sets.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2020a

mexihat

Mexican hat (Ricker) wavelet

Syntax

```
[psi,x] = mexihat(lb,ub,n)
```

Description

`[psi,x] = mexihat(lb,ub,n)` returns the Mexican hat wavelet `psi` evaluated at `x`, an `n`-point regular grid in the interval `[lb, ub]`. The Mexican hat wavelet is also known as the Ricker wavelet.

The Mexican hat wavelet has the interval [-5, 5] as effective support. Nearly 100% of the wavelet's energy is in the interval. 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.

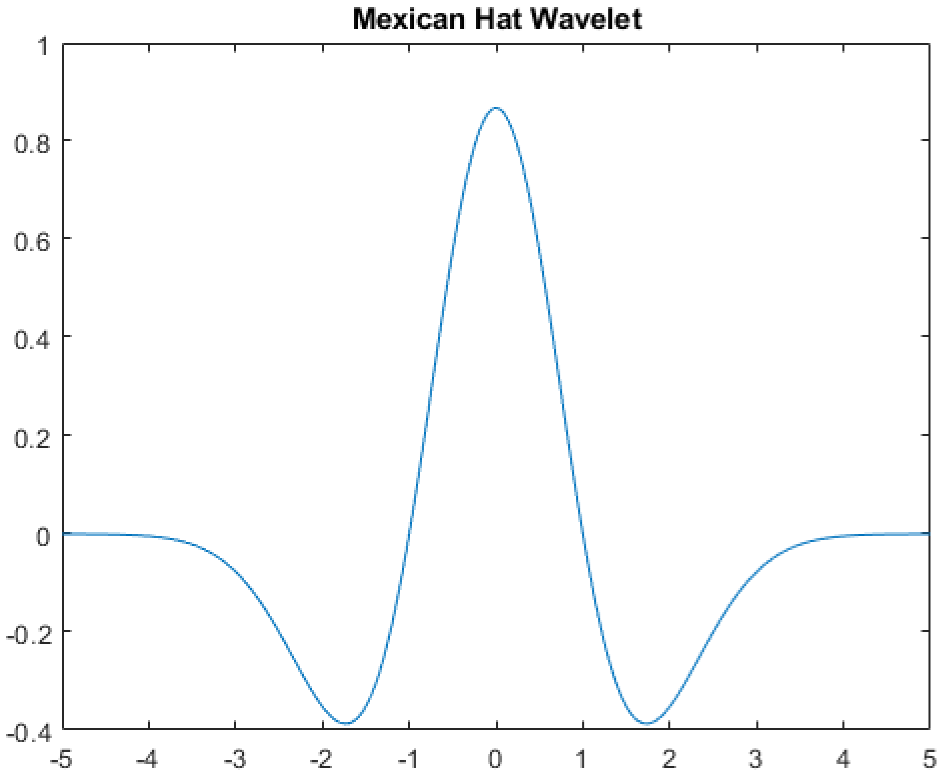
Note You can use `gauswavf` to obtain a second order derivative of a Gaussian wavelet. If you use the negative of this normalized derivative, the resulting wavelet resembles the Mexican hat wavelet.

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')
```



Input Arguments

`lb` — Lower limit

real-valued scalar

Lower limit of interval, specified as a real-valued scalar.

`ub` — Upper limit

real-valued scalar

Upper limit of interval, specified as a real-valued scalar.

`n` — Number of sample points

positive integer

Number of sample points, specified as a positive integer.

Output Arguments

`psi` — Mexican hat wavelet

real-valued vector

Mexican hat wavelet, returned as a real-valued vector of length `n`.

x — Sampling instants
real-valued vector

Sampling instants, returned as a real-valued vector of length `n`.

See Also

`waveinfo` | `gauswavf`

Introduced before R2006a

meyer

Meyer wavelet

Syntax

```
[phi,psi,t] = meyer(lb,ub,n)
[phi,t] = meyer(lb,ub,n,'phi')
[psi,t] = meyer(lb,ub,n,'psi')
[phi,psi] = meyer(lb,ub,n,S)
```

Description

`[phi,psi,t] = meyer(lb,ub,n)` returns the Meyer scaling and wavelet functions, `phi` and `psi` respectively, evaluated at `t`, an `n`-point regular grid in the interval `[lb, ub]`. Both functions have the interval [-8, 8] as effective support.

Note `meyer` uses the auxiliary function `meyeraux`. If you change `meyeraux`, you get a family of different wavelets.

`[phi,t] = meyer(lb,ub,n,'phi')` returns only the Meyer scaling function.

`[psi,t] = meyer(lb,ub,n,'psi')` returns only the Meyer wavelet.

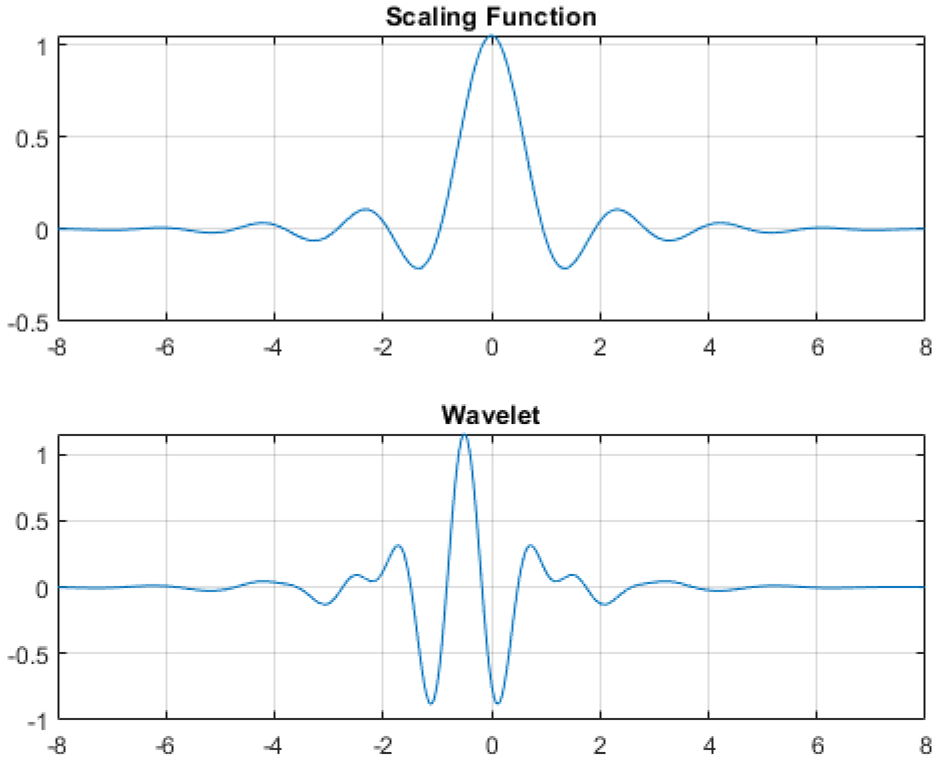
`[phi,psi] = meyer(lb,ub,n,S)` returns the Meyer scaling function and wavelet if `S` is not equal to `'phi'` or `'psi'`.

Examples

Plot Meyer Wavelet and Scaling Functions

Plot the Meyer wavelet and scaling functions.

```
lb = -8;
ub = 8;
n = 1024;
[phi,psi,x] = meyer(lb,ub,n);
subplot(2,1,1)
plot(x,phi)
grid on
title('Scaling Function')
subplot(2,1,2)
plot(x,psi)
grid on
title('Wavelet')
```



Input Arguments

`lb` — Lower limit

real-valued scalar

Lower limit of interval, specified as a real-valued scalar.

`ub` — Upper limit

real-valued scalar

Upper limit of interval, specified as a real-valued scalar.

`n` — Number of points

positive integer

Number of points, specified as a positive integer. `n` must be a power of 2.

Output Arguments

`phi` — Meyer scaling function

real-valued vector

Meyer scaling function, returned as a real-valued vector of length `n`.

psi — Meyer wavelet

real-valued vector

Meyer wavelet, returned as a real-valued vector of length n.

t — Sampling instants

real-valued vector

Sampling instants, returned as a real-valued vector of length n.

Algorithms

The Meyer wavelet and scaling functions are defined in the Fourier domain. Starting from an explicit form of the Fourier transform $\widehat{\phi}$ of the scaling function ϕ, meyer computes the values of $\widehat{\phi}$ on a regular grid. The values of ϕ are computed using an inverse Fourier transform.

The procedure for the wavelet ψ is identical to the procedure for the scaling function.

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

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 `X`. The input `X` is a vector or matrix of real values. The function is

$$y = 35x^4 - 84x^5 + 70x^6 - 20x^7 .$$

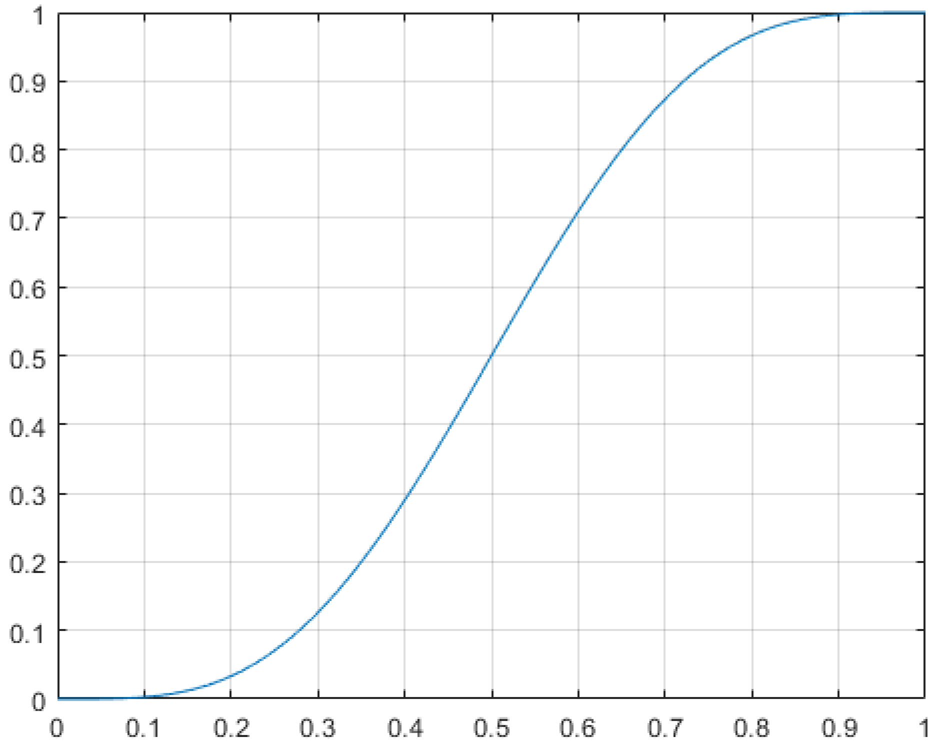
`X` and `Y` have the same dimensions. The range of `meyeraux` is the closed interval [0, 1].

Examples

Plot Meyer Auxiliary Function

Plot the Meyer auxiliary function.

```
x = linspace(0,1,100);
y = meyeraux(x);
plot(x,y)
grid on
```



Input Arguments

X — Sample points
real-valued vector | real-valued matrix

Sample points at which to evaluate the Meyer auxiliary function, specified as a vector or matrix of real values.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`meyer`

Introduced before R2006a

minus

Laurent polynomial or Laurent matrix subtraction

Syntax

```
Q = minus(A,B)
Q = A - B
```

Description

`Q = minus(A,B)` subtracts `B` from `A`, where `A` and `B` are a pair of Laurent polynomials or Laurent matrices.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `minus`. The input data type determines which version is executed.

`Q = A - B` is equivalent to `Q = minus(A,B)`.

Examples

Laurent Polynomial Subtraction

Create two Laurent polynomials:

- $a(z) = 2z$
- $b(z) = 8z^3 + 4z^2 + 2z + 1$

```
a = laurentPolynomial(Coefficients=[2],MaxOrder=1);
b = laurentPolynomial(Coefficients=[8 4 2 1],MaxOrder=3);
```

Subtract $a(z)$ from $b(z)$.

```
c = minus(b,a)
```

```
c = 
  laurentPolynomial with properties:

    Coefficients: [8 4 0 1]
        MaxOrder: 3
```

Subtract $a^3(z) + a^2(z)$ from $b(z)$.

```
d = b-(mpower(a,3)+mpower(a,2))
```

```
d = 
  laurentPolynomial with properties:
    Coefficients: [2 1]
        MaxOrder: 1
```

Laurent Matrix Subtraction

Create the Laurent polynomials:

- $a(z) = 5z^2 + 8z + 3$
- $b(z) = 8z + 3 + 2z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[5 8 3],MaxOrder=2);
lpB = laurentPolynomial(Coefficients=[8 3 2],MaxOrder=1);
```

Create the Laurent matrices:

- $\mathtt{lmatA} = \begin{bmatrix} a(z) & 2 \\ 4 & 6 \end{bmatrix}$
- $\mathtt{lmatB} = \begin{bmatrix} b(z) & 1 \\ 3 & 5 \end{bmatrix}$

```
lmatA = laurentMatrix(Elements={lpA,2;4,6});
lmatB = laurentMatrix(Elements={lpB,1;3,5});
```

Subtract `lmatB` from `lmatA`.

```
lmatC = lmatA-lmatB;
lmatC.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [5 0 0 -2]
        MaxOrder: 2
```

```
lmatC.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

```
lmatC.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

```
lmatC.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 1
        MaxOrder: 0
```

Input Arguments

A — Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

B — Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Output Arguments

Q — Difference
`laurentPolynomial` object

Difference of two Laurent polynomials or two Laurent matrices, returned as a `laurentPolynomial` object or a `laurentMatrix` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`mtimes` | `plus`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

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.

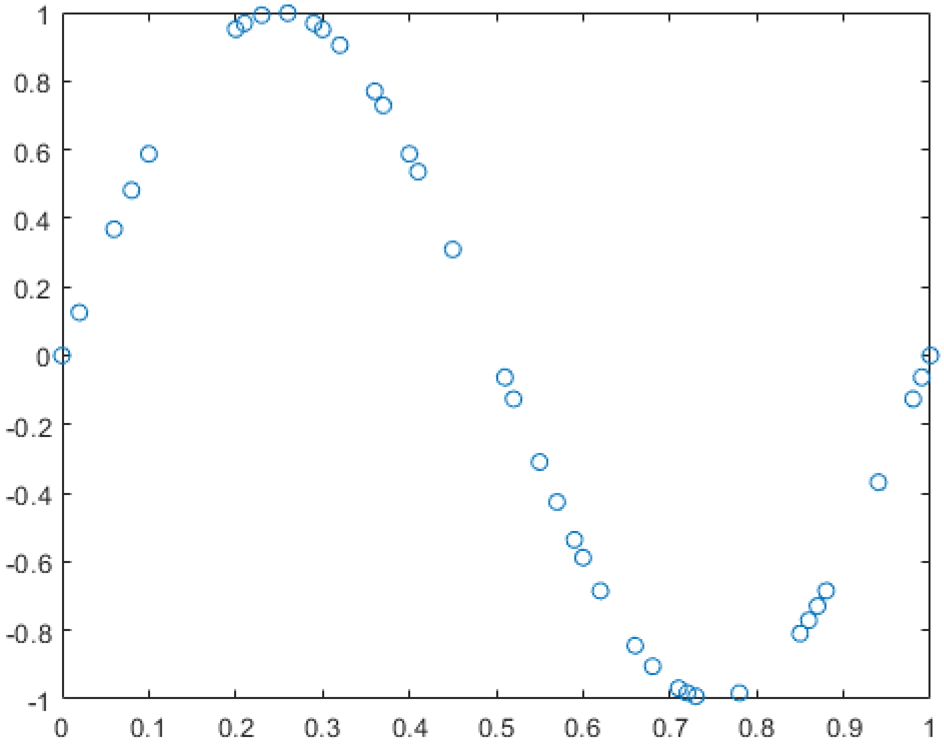
Examples

Multiscale Local 1-D Polynomial Transform and Inverse Transform

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

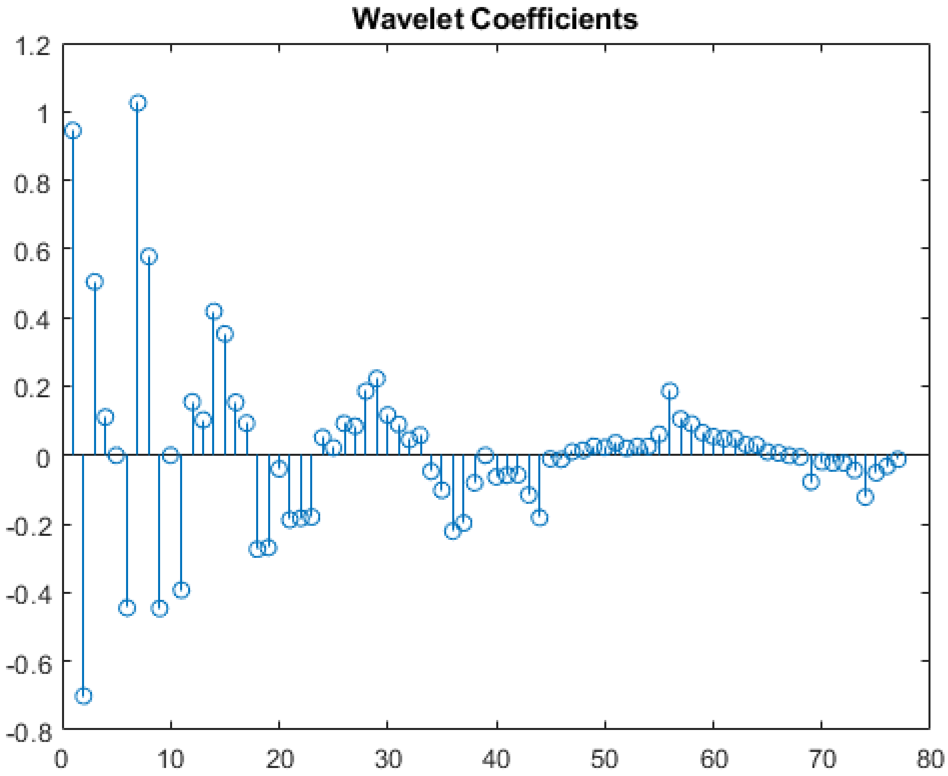
Create and plot a sine wave with non-uniform sampling.

```
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
```



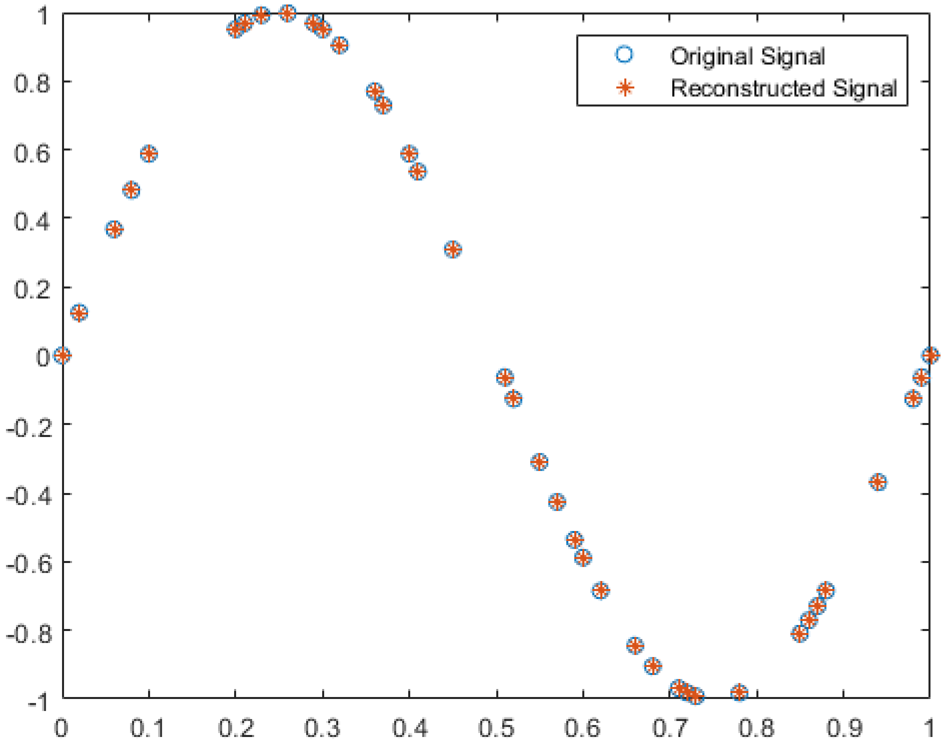
Perform the multiscale local 1-D polynomial transform (`mlpt`) on the signal. Visualize the coefficients.

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

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



Perform the inverse multiscale local 1-D polynomial transform (`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))
```

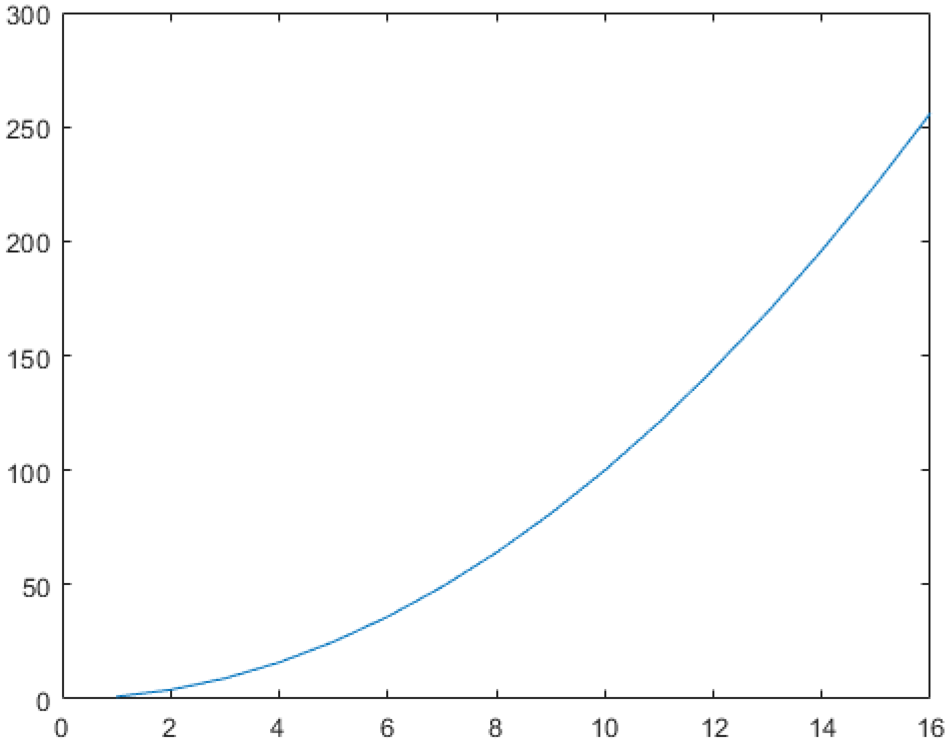
```
reconstructionError = 1.7552e-15
```

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:
```

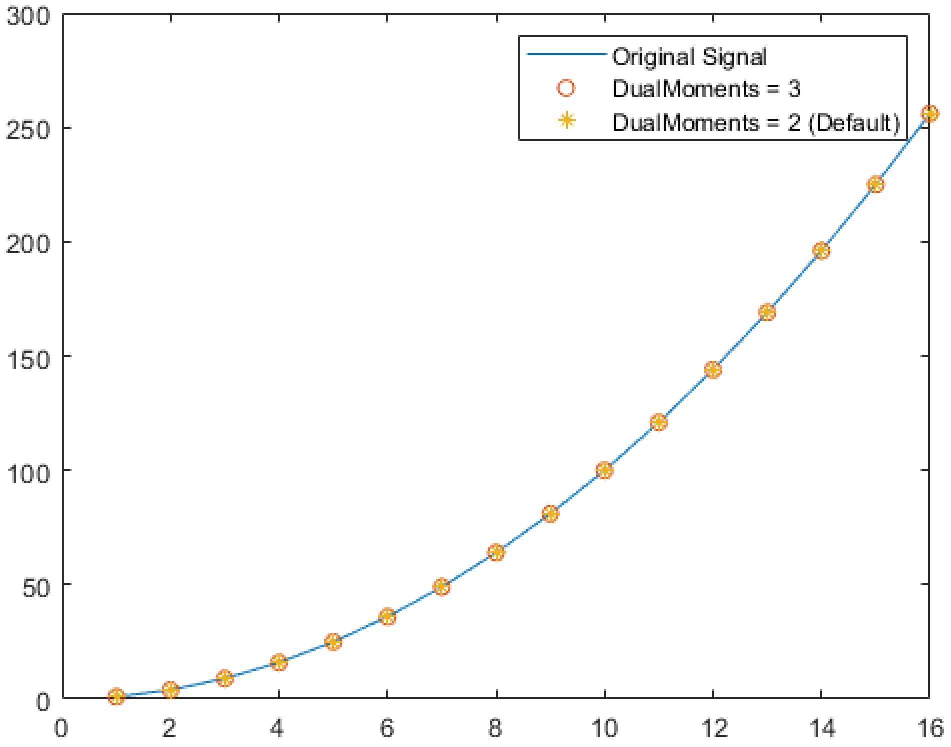
```
fprintf('  - Nondefault dual moments: %0.2f\n',mean(abs(y3-x)));
```

```
  - 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: 3.96
```

```
tic
[w13,~,nj13,m13] = mlpt(x,t,13);
computationTime13 = toc;
fprintf('Level thirteen computation time: %0.2f\n',computationTime13)
```

```
Level thirteen computation time: 5.98
```

Use Default Time Instants

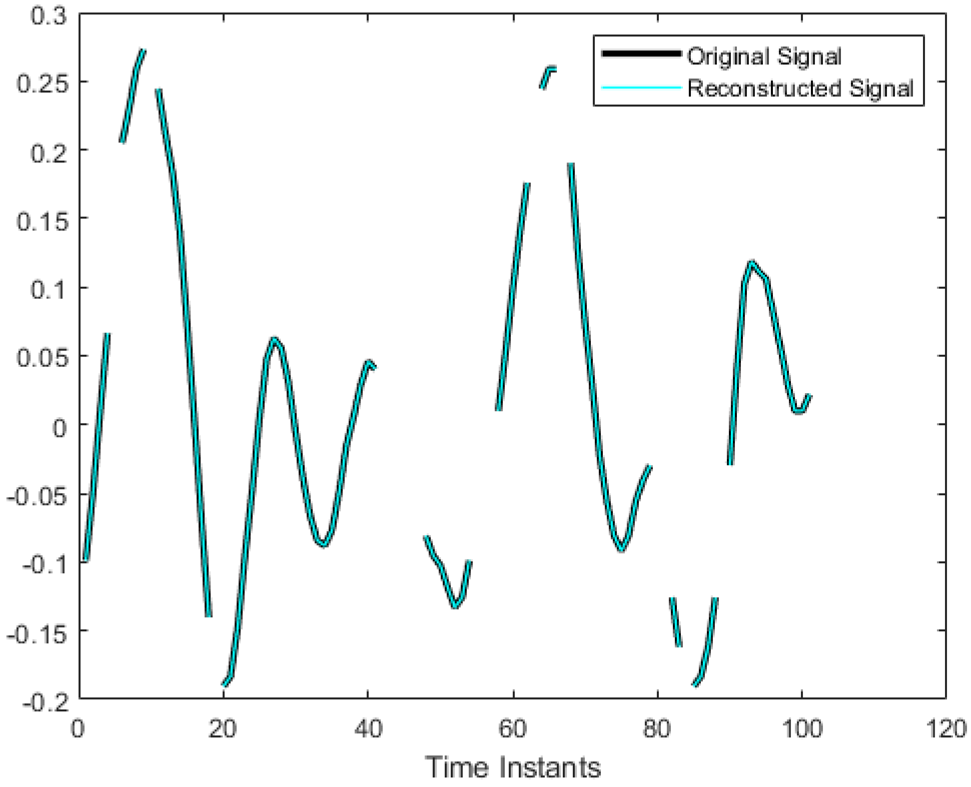
If your time instants are not known or specified, you can calculate the MLPT using default time instants.

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('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')
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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, Maarten. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data." *IEEE Transactions on Signal Processing* 61, no. 3 (February 2013): 545–55. https://doi.org/10.1109/TSP.2012.2225059.

[2] Jansen, Maarten, and Mohamed Amghar. "Multiscale Local Polynomial Decompositions Using Bandwidths as Scales." *Statistics and Computing* 27, no. 5 (September 2017): 1383–99. https://doi.org/10.1007/s11222-016-9692-8.

[3] Jansen, Maarten, and Patrick Oonincx. *Second Generation Wavelets and Applications*. London ; New York: 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.

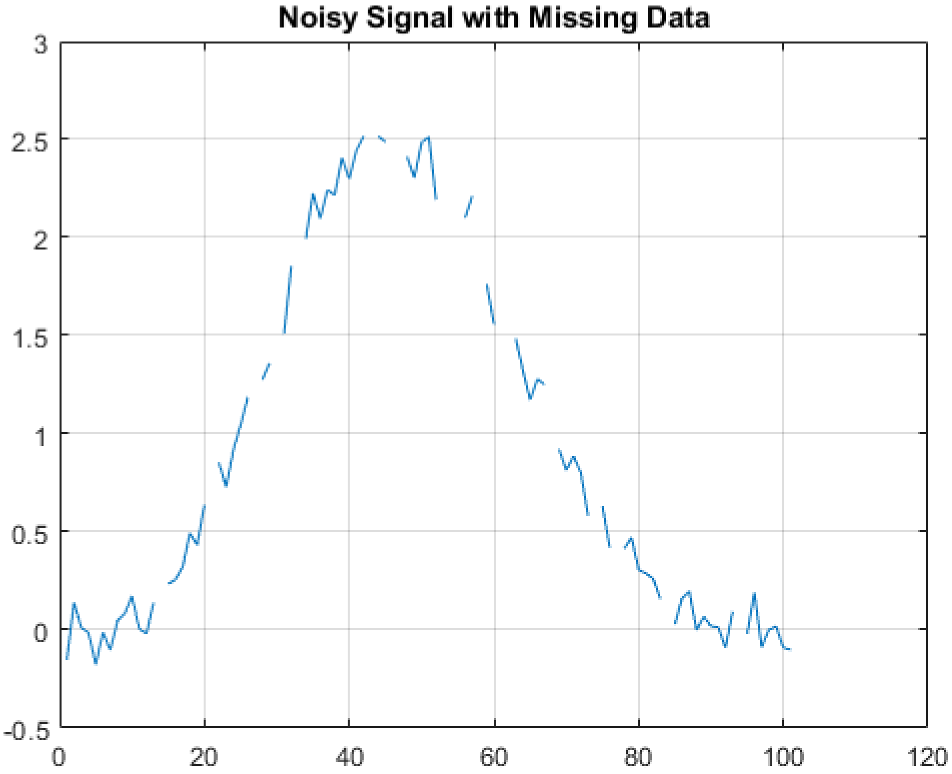
Examples

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')
```

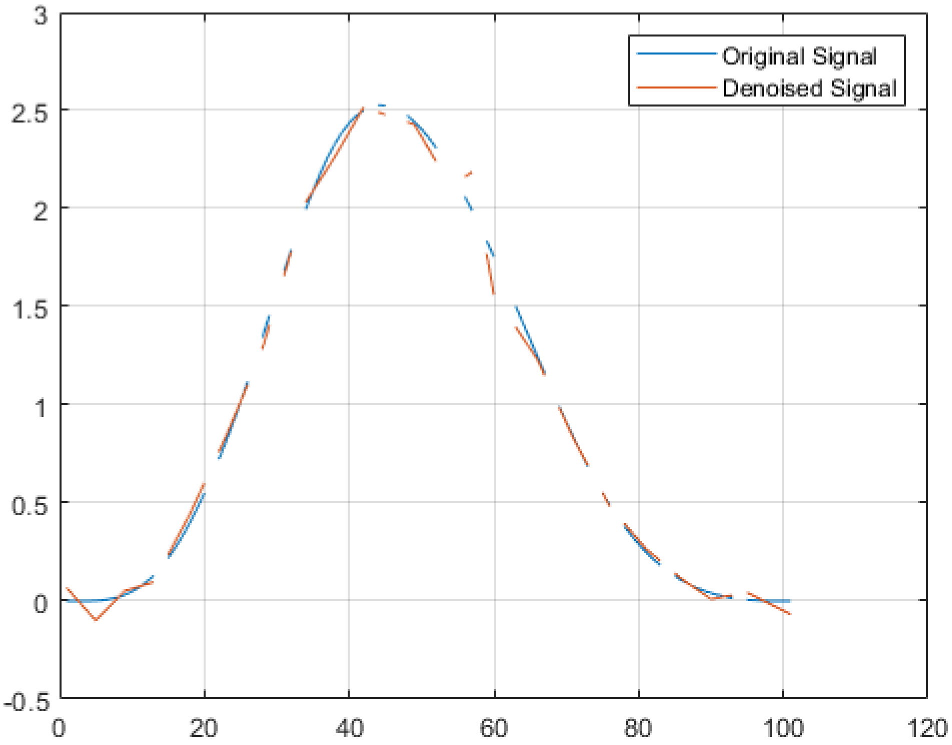


Denoise the data using the median denoising method.

```
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');
```

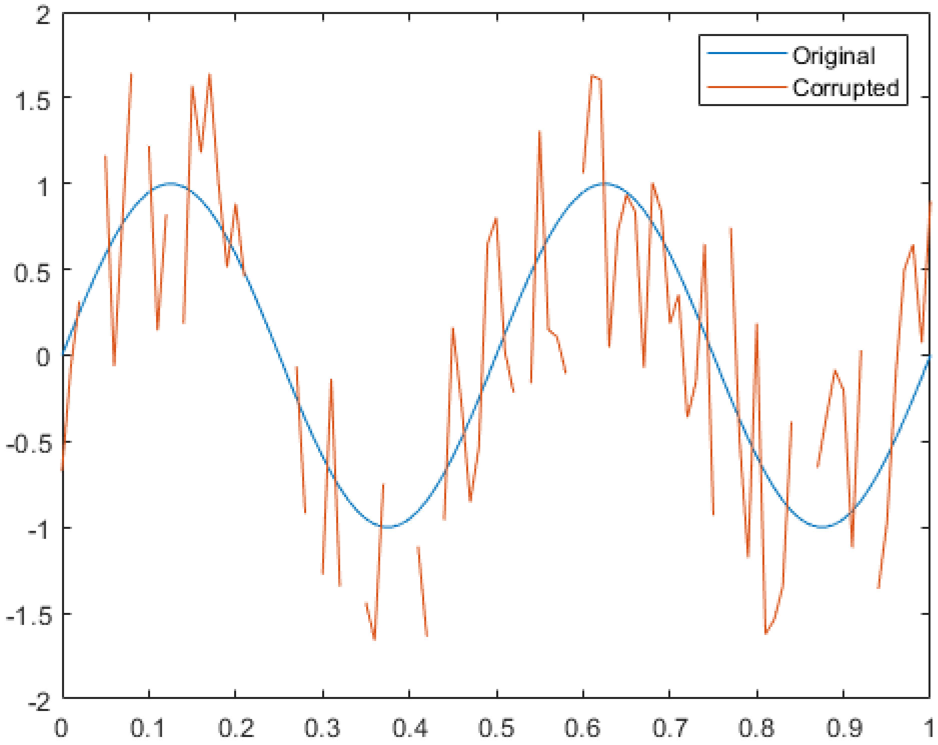


Denoise Using Multiscale Local Polynomial Transform

Reduce noise of signal using the multiscale local polynomial transform (MLPT).

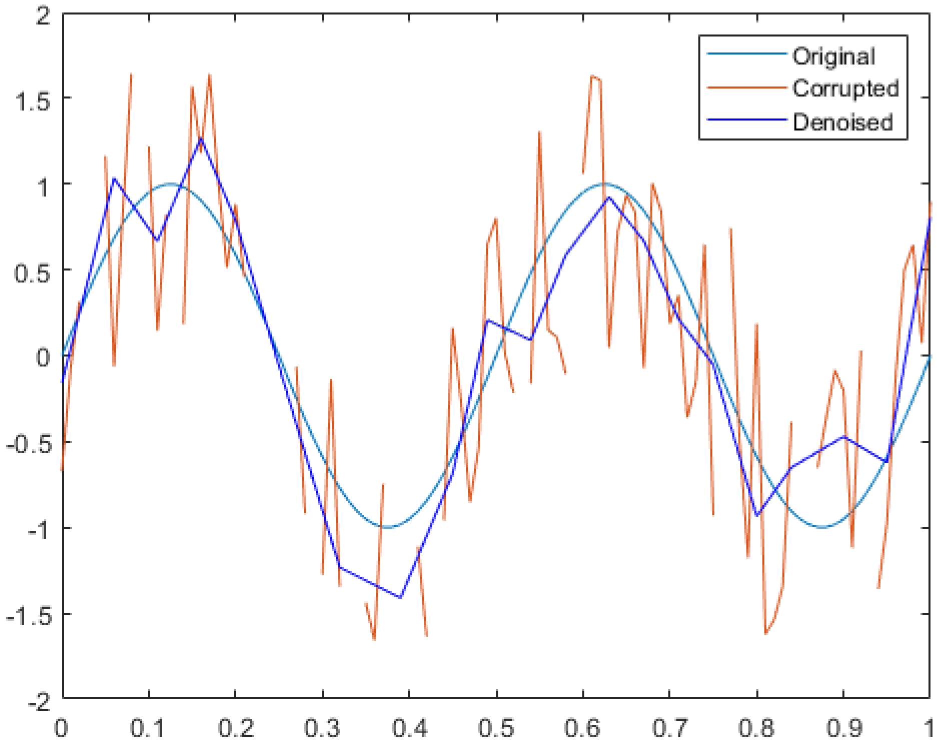
Load a pure sine wave signal with uniform sampling, and a corrupted version of the signal.

```
load('InputSamples.mat')

plot(t,x)
hold on
plot(tCorrupt,xCorrupt)
legend('Original','Corrupted')
```



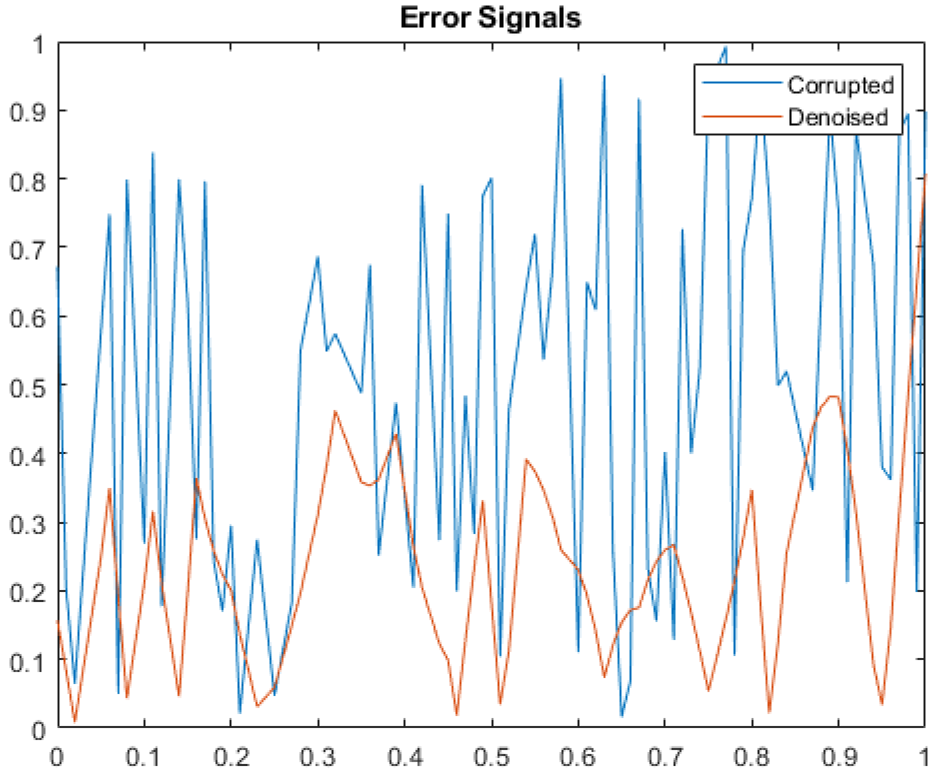
Use `mlptdenoise` to denoise the corrupted signal. Visually compare the corrupted and denoised signals against the original signal.

```
[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);
```

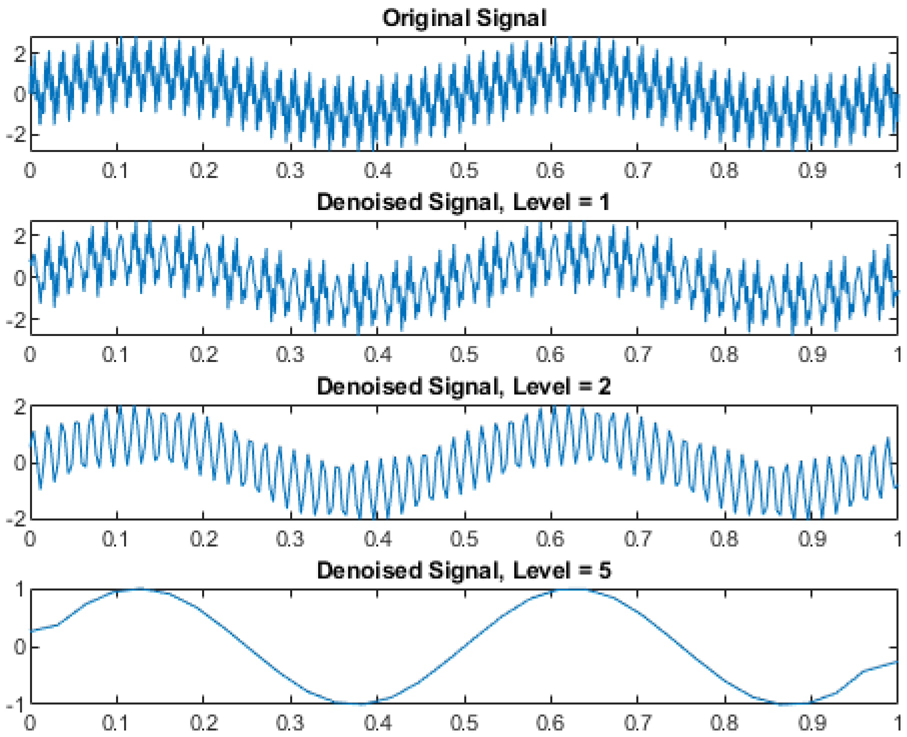
Denoise the signal to levels one, two, and five.

```
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

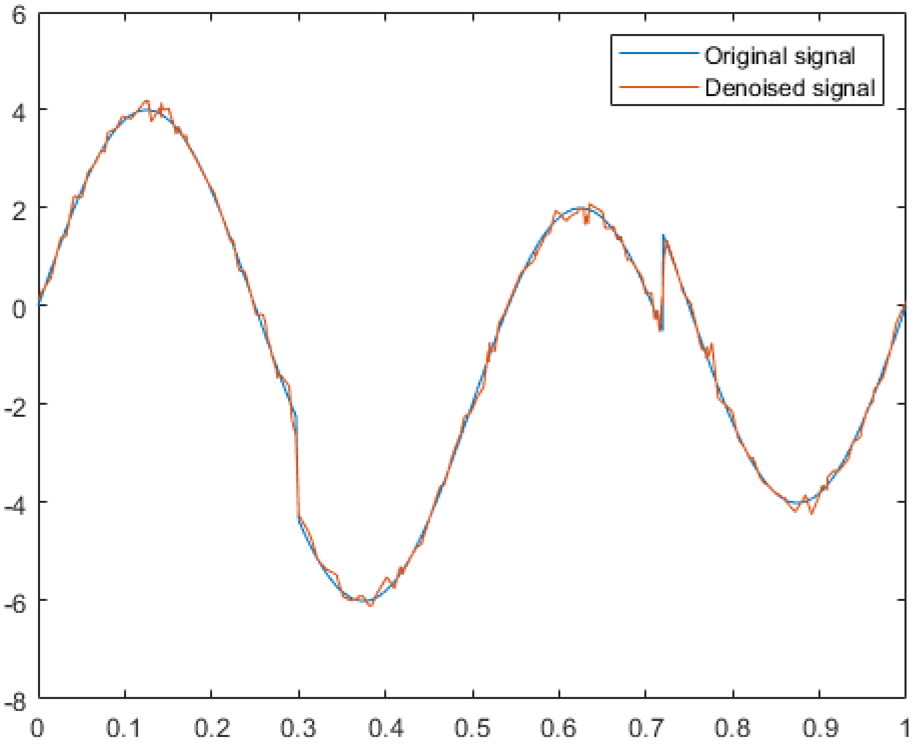
The `mlptdenoise` function performs the forward MLPT, thresholds the coefficients as specified by the `'DenoisingMethod'` name-value pair. Then `mlptdenoise` performs the inverse MLPT to return a denoised signal in the domain of your original signal.

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.

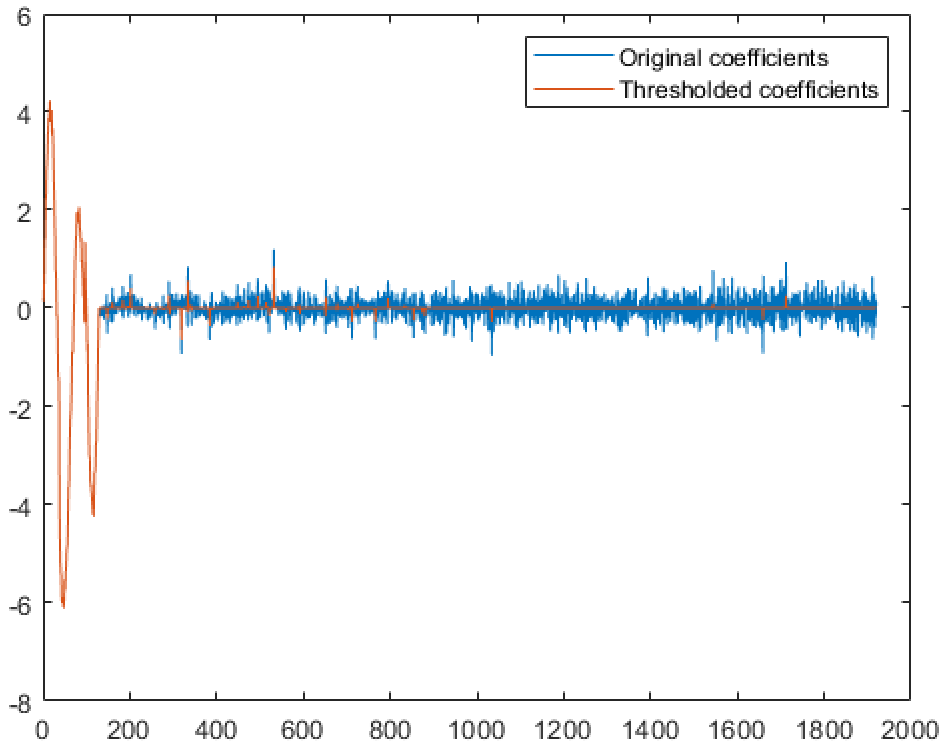
```
load nonuniformheavisine;
```

```
[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')
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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, Maarten. “Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data.” *IEEE Transactions on Signal Processing* 61, no. 3 (February 2013): 545–55. https://doi.org/10.1109/TSP.2012.2225059.

[2] Jansen, Maarten, and Mohamed Amghar. “Multiscale Local Polynomial Decompositions Using Bandwidths as Scales.” *Statistics and Computing* 27, no. 5 (September 2017): 1383–99. https://doi.org/10.1007/s11222-016-9692-8.

[3] Jansen, Maarten, and Patrick Oonincx. *Second Generation Wavelets and Applications*. London ; New York: 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.

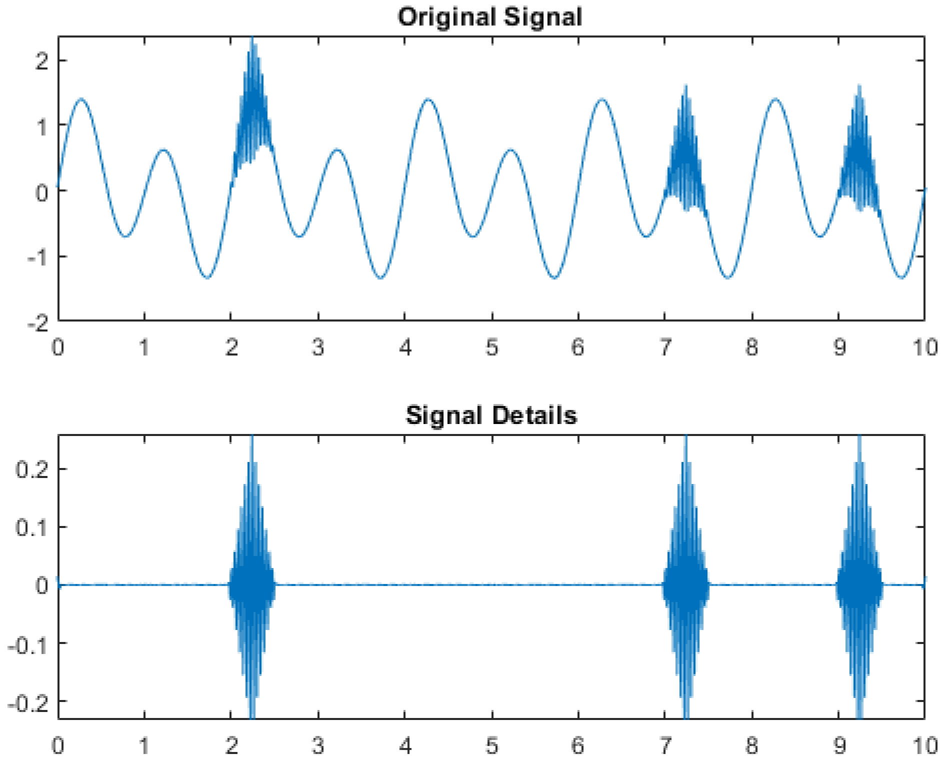
```
t = (0:0.01:10)';
x = sin(2*pi.*t) + 0.5*sin(pi.*t+0.1);
bliptime = (0:0.01:0.5)';
n = numel(bliptime);
z0 = 2*(1:(n+1)/2)/(n+1);
trng = [z0 z0((n-1)/2:-1:1)]';
blip = sin(50*pi.*bliptime).*trng;
for i = [200,700,900]
    x(i:i+numel(bliptime)-1) = x(i:i+numel(bliptime)-1)+blip;
end
```

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')
```

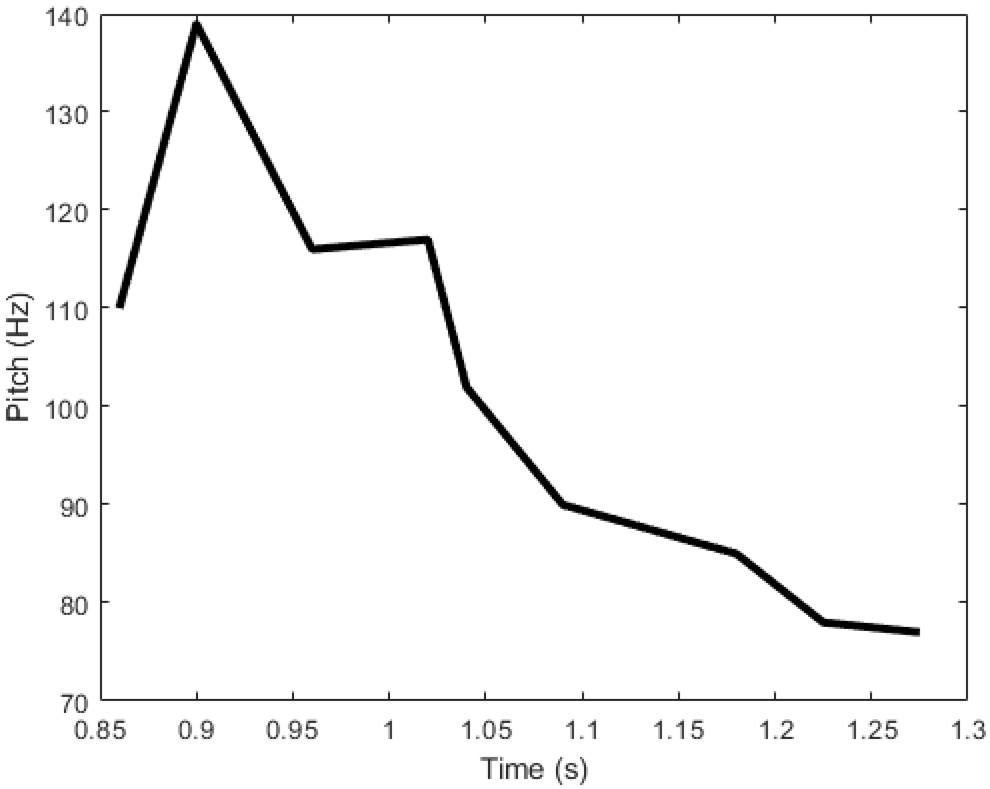


Approximate Data by Choosing Reconstruction Coefficients

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('CorruptedPitchData.mat');
plot(time,pitchContour,'k','linewidth',3)
hold on
xlabel('Time (s)')
ylabel('Pitch (Hz)')
```



Compute the MLPT of the pitch contour.

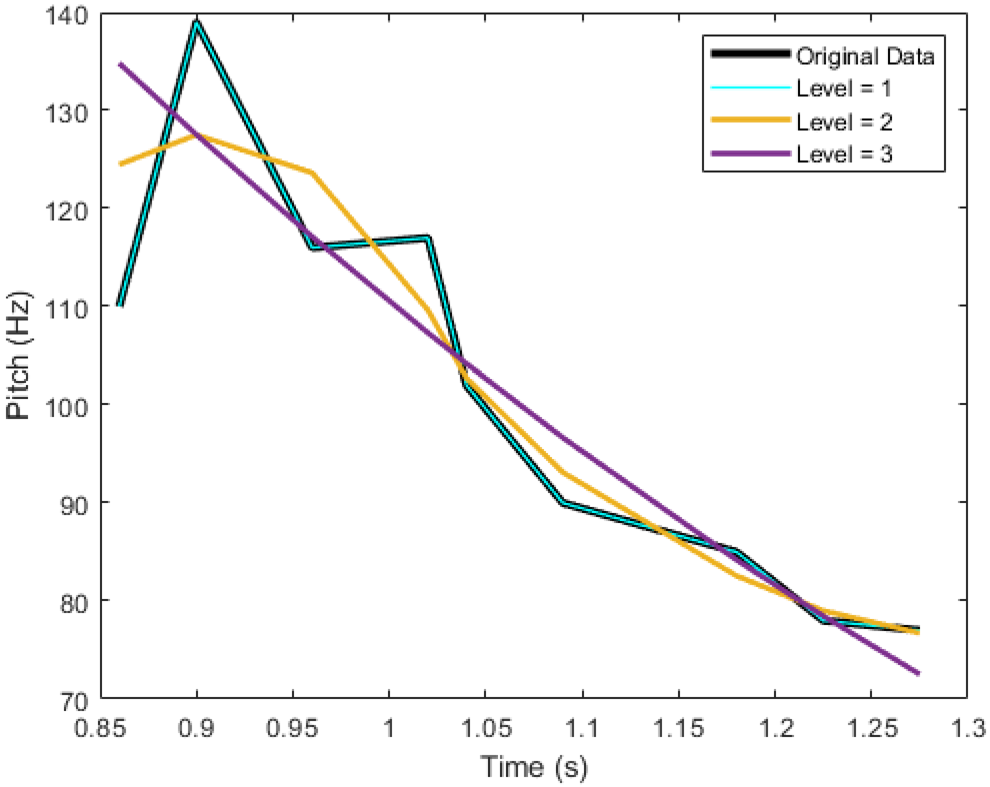
```
[w,t,nj,scalingMoments] = mlpt(pitchContour,time, ...
    'DualMoments',3, ...
    'PrimalMoments',4, ...
    'PreFilter','none');
```

Use `mlptrecon` to reconstruct the signal using the approximation coefficients 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
```



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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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, Maarten. “Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data.” *IEEE Transactions on Signal Processing* 61, no. 3 (February 2013): 545–55. https://doi.org/10.1109/TSP.2012.2225059.

[2] Jansen, Maarten, and Mohamed Amghar. “Multiscale Local Polynomial Decompositions Using Bandwidths as Scales.” *Statistics and Computing* 27, no. 5 (September 2017): 1383–99. https://doi.org/10.1007/s11222-016-9692-8.

[3] Jansen, Maarten, and Patrick Oonincx. *Second Generation Wavelets and Applications*. London ; New York: 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

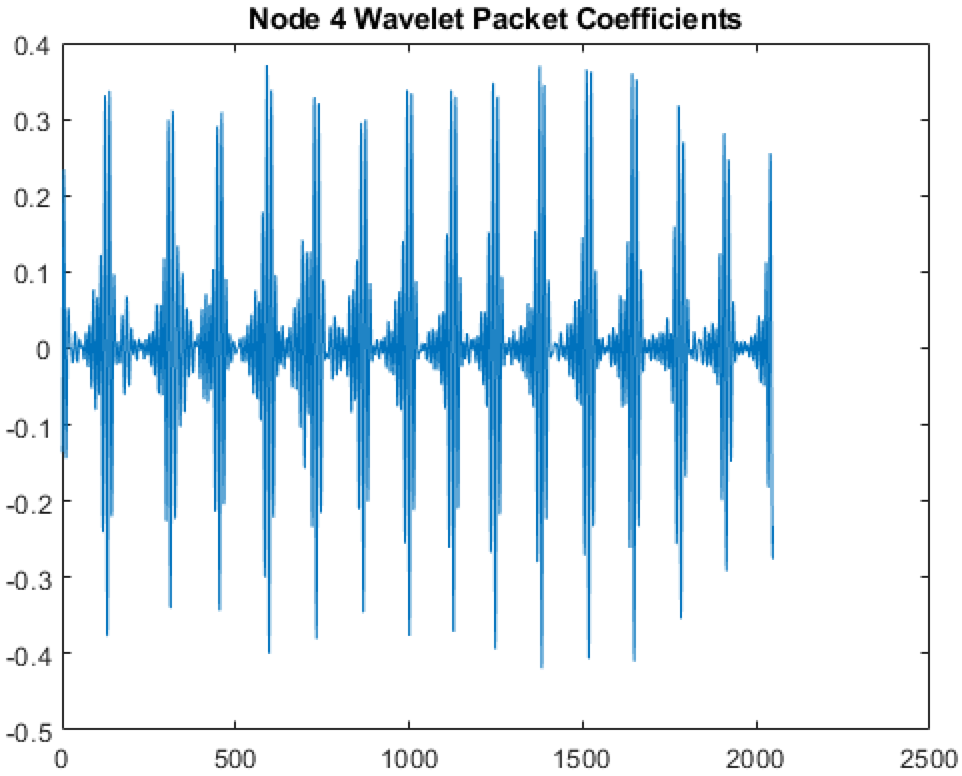
MODWPT Using Default Wavelet

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 = 1×2

          16         12998
```

MODWPT Using Scaling and Wavelet Filters

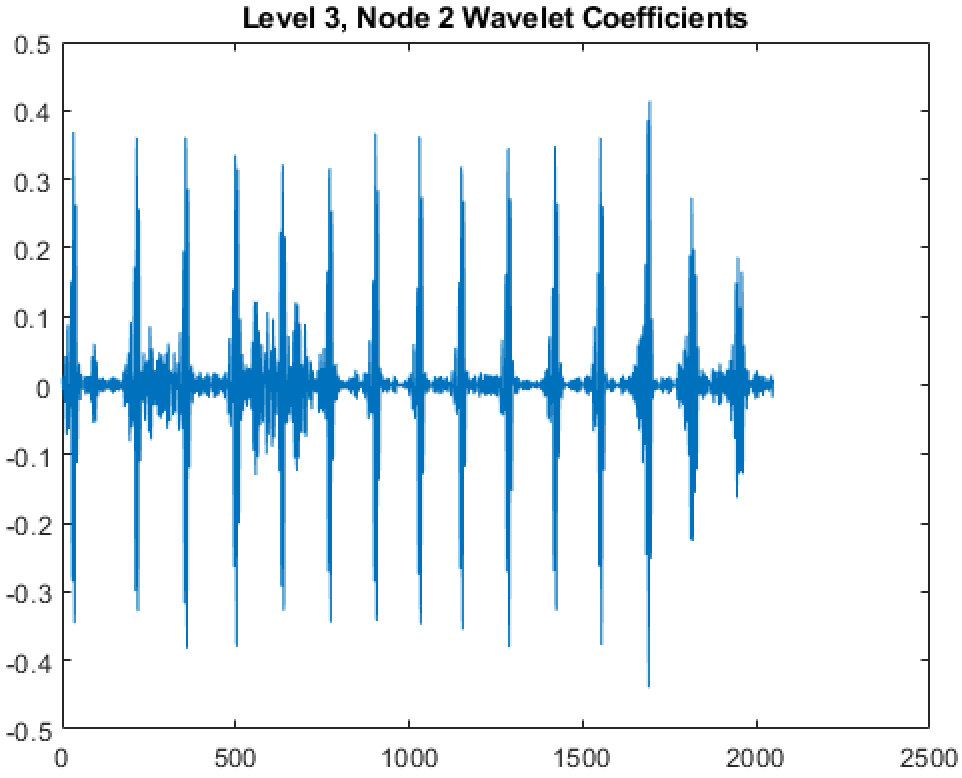
Obtain the MODWPT of an ECG waveform using the Fejer-Korovkin length 18 scaling and wavelet filters.

```
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 = 8×1

    0.0312
    0.0938
    0.1562
    0.2188
    0.2812
    0.3438
    0.4062
    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);
```

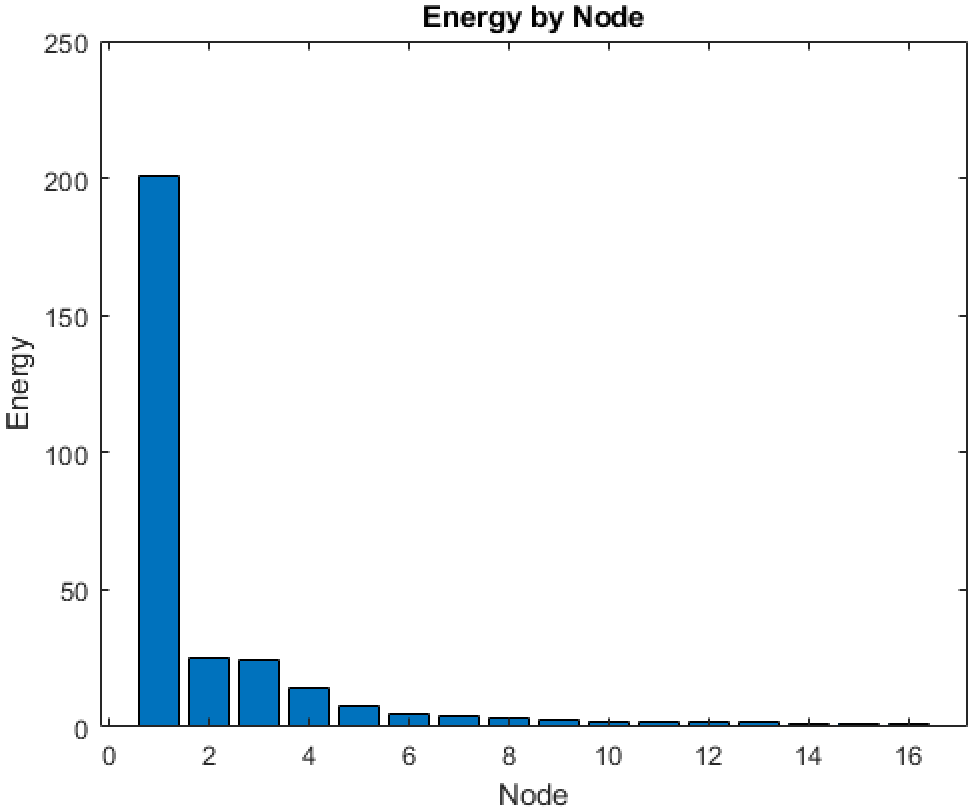
Show that the sum of the MODWPT energies is equal to the sum of the energy in the original signal. The difference between the total MODWPT energy and the signal energy is small enough to be considered insignificant.

```
disp(['Difference between MODWPT energy and signal energy: ',num2str(sum(energy)-sum(wecg.^2))])
```

```
Difference between MODWPT energy and signal energy: 3.6122e-09
```

Plot the MODWPT energy by node.

```
figure
bar(1:16,energy)
xlabel('Node')
ylabel('Energy')
title('Energy by Node')
```

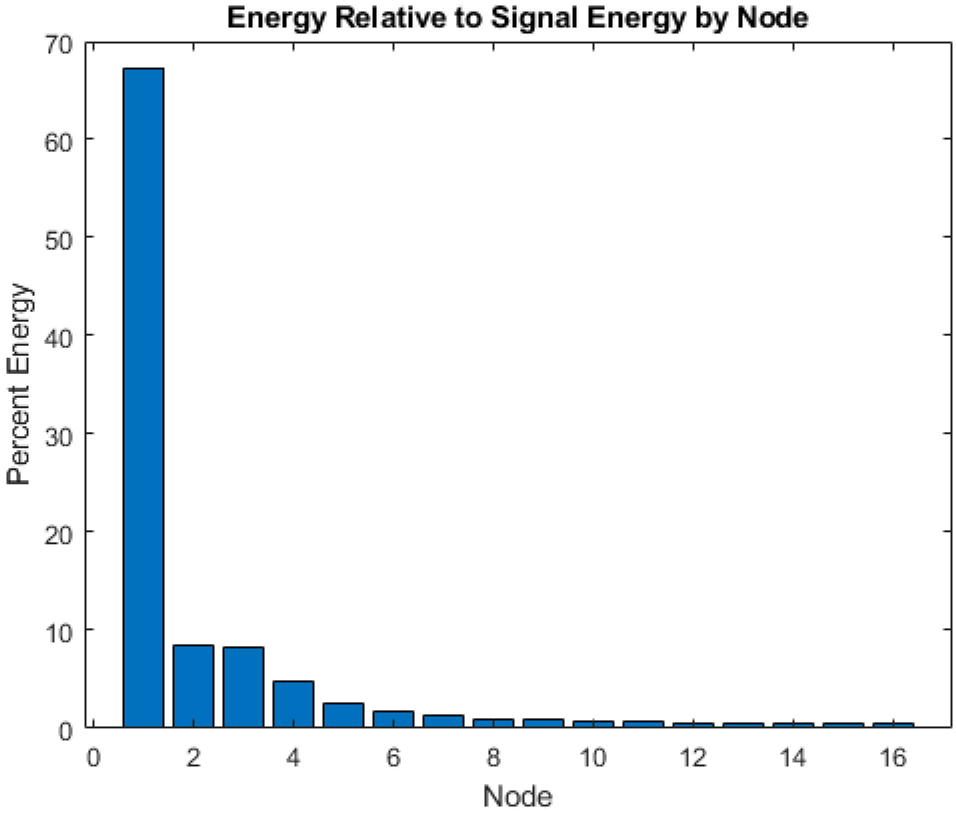


```
disp(['Total power in passband: ',num2str(energy(1))])
```

```
Total power in passband: 200.8446
```

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')
```



```
disp(['Percentage of signal power in passband: ',num2str(relenergy(1)*100)])
```

```
Percentage of signal power in passband: 67.3352
```

Time-Aligned MODWPT

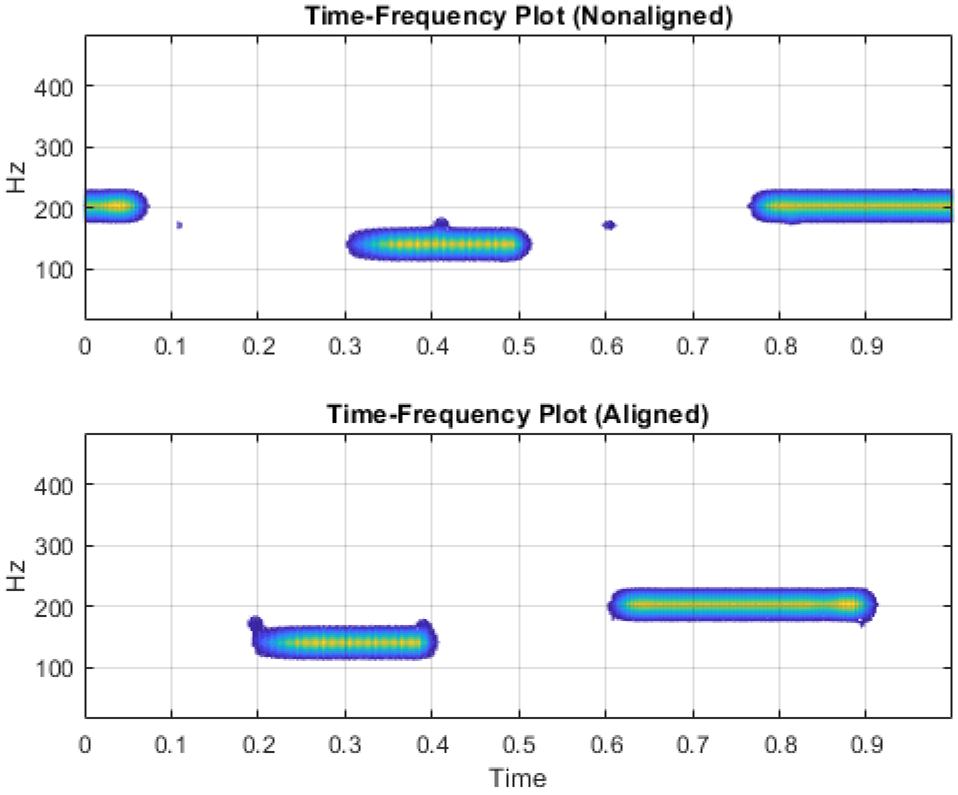
Obtain the time-aligned MODWPT of two intermittent sine waves in noise. The sine wave frequencies are 150 Hz and 200 Hz. The data is sampled at 1000 Hz.

```
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) | `'haar'` | `'db2'` | ...

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, 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'`.

- `'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 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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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 *n*th 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.

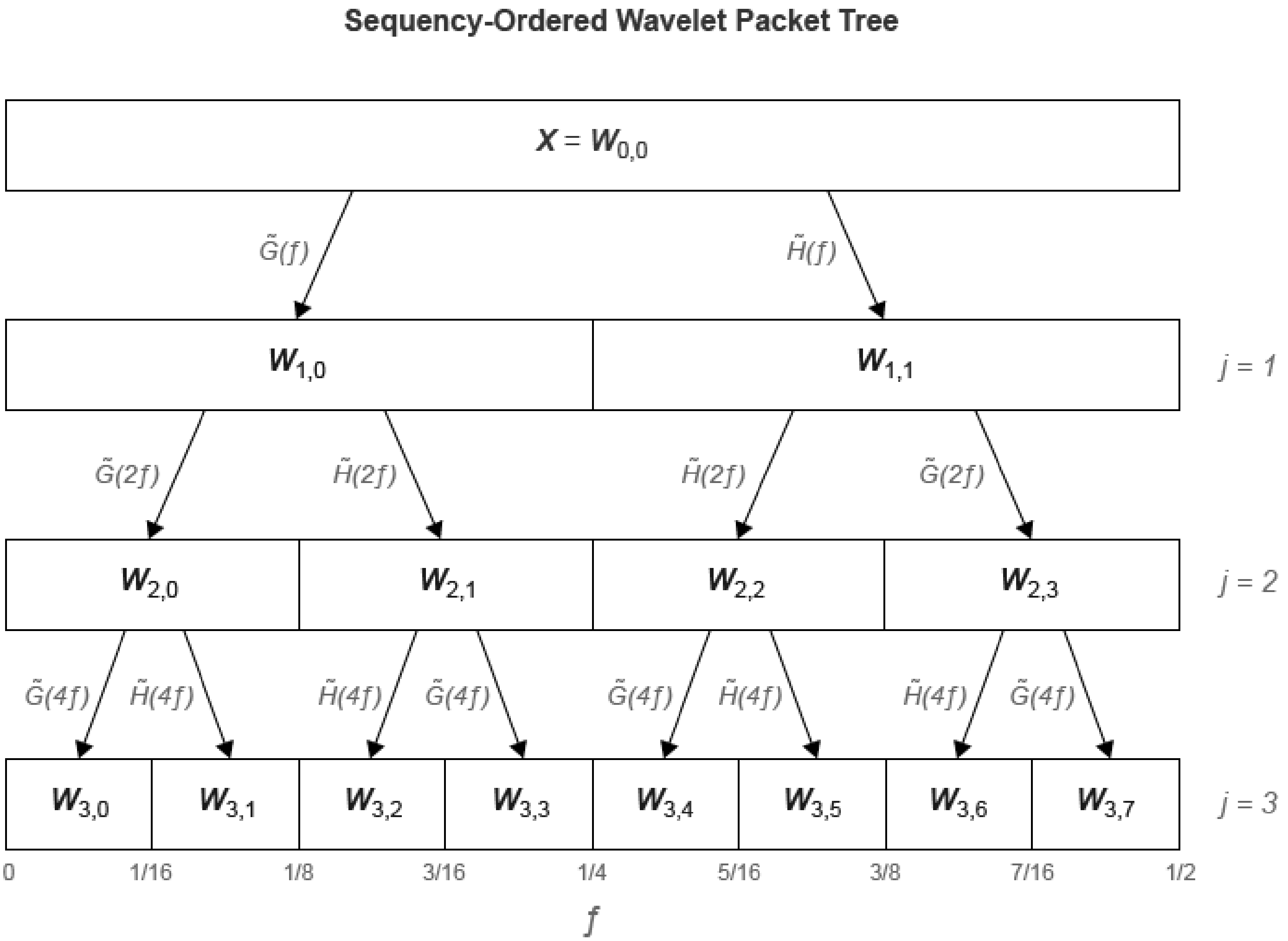
relenergy — Relative energy

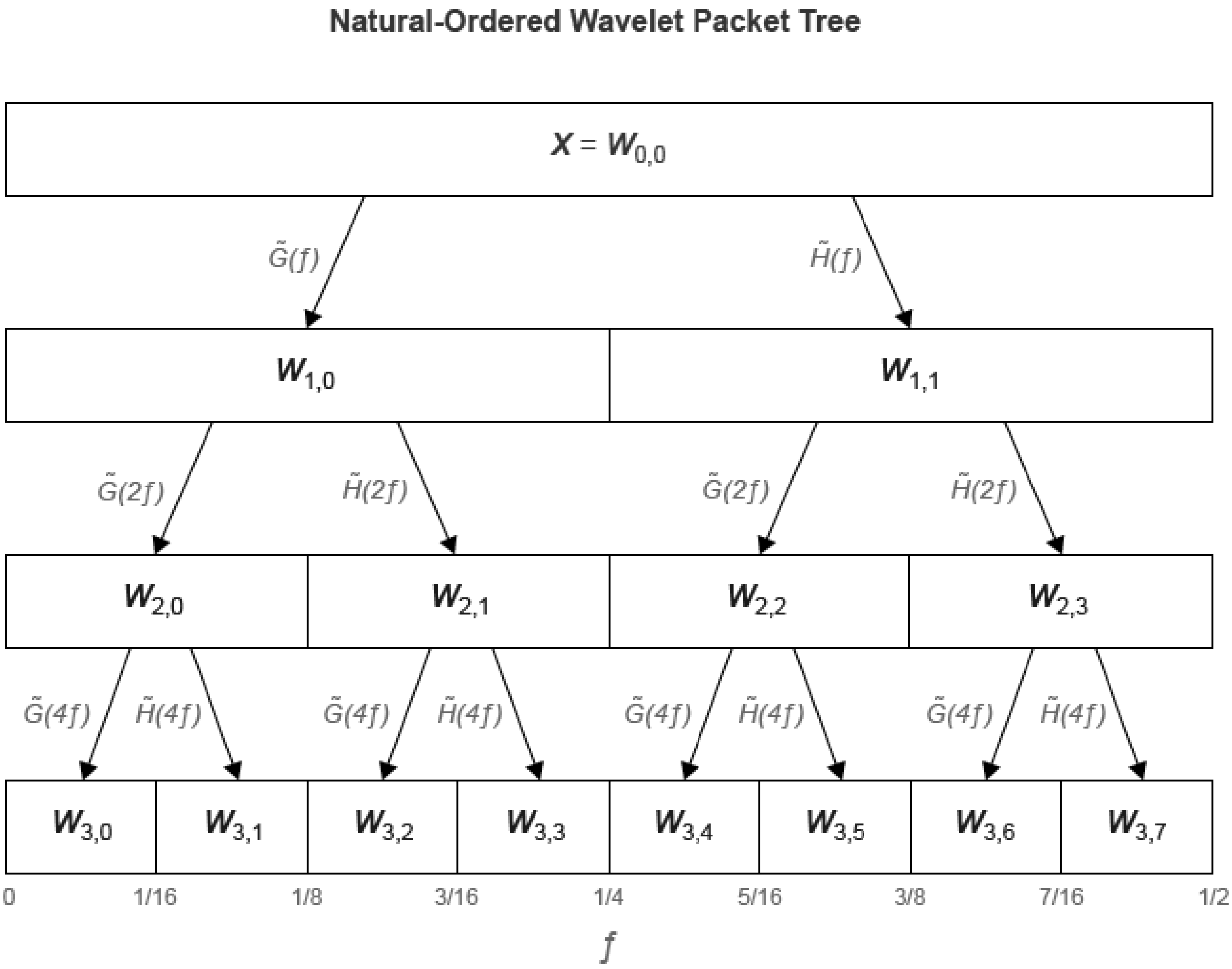
vector

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.


Sequency-Ordered Wavelet Packet Tree
X = W0,0
G̃(f)
H̃(f)
W1,0
W1,1
j = 1
G̃(2f)
H̃(2f)
H̃(2f)
G̃(2f)
W2,0
W2,1
W2,2
W2,3
j = 2
G̃(4f)
H̃(4f)
H̃(4f)
G̃(4f)
G̃(4f)
H̃(4f)
H̃(4f)
G̃(4f)
W3,0
W3,1
W3,2
W3,3
W3,4
W3,5
W3,6
W3,7
j = 3
0
1/16
1/8
3/16
1/4
5/16
3/8
7/16
1/2
f



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™.

Usage notes and limitations:

- The input `wname` must be constant.

See Also

`imodwpt` | `modwptdetails` | `dwpt`

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( ___ ,'FullTree',tf)
```

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 1-961.

`w = modwptdetails(x,wname)` uses the orthogonal wavelet filter specified by `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 `cfreq`, the center frequencies of the approximate passbands corresponding to the MODWPT details in `w`.

`[ ___ ] = modwptdetails( ___ ,'FullTree',tf)`, where `tf` is `false`, 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. The default for `tf` is `false`.

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
```

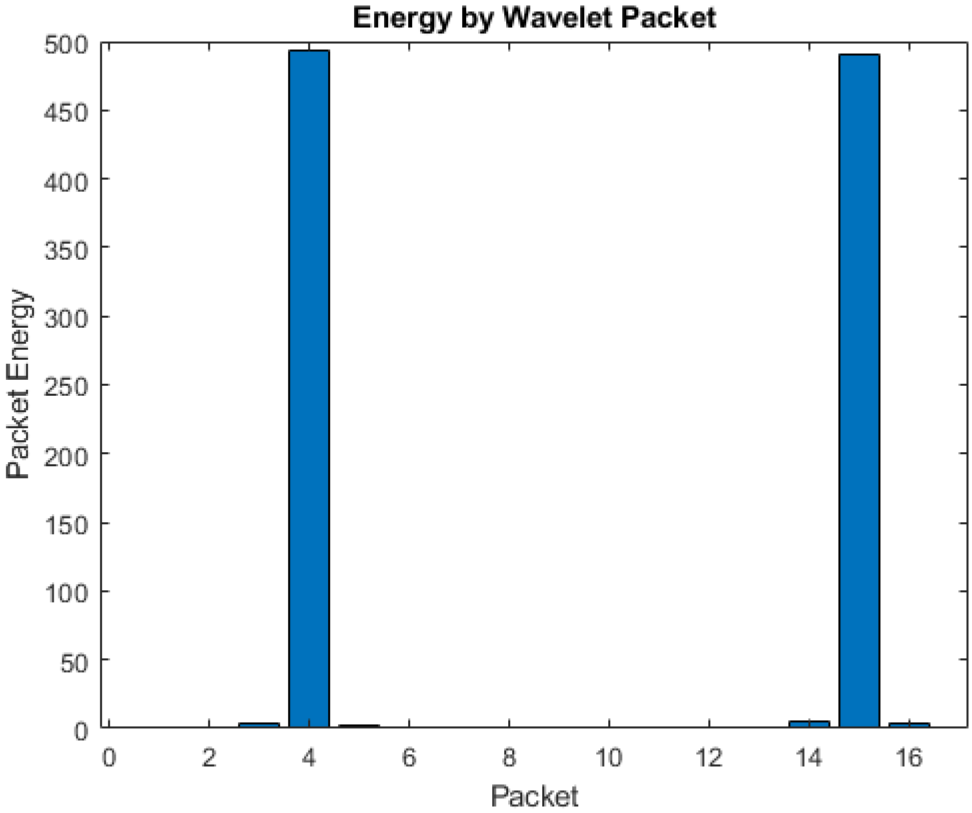
MODWPT Details for Two Sine Waves

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);
```

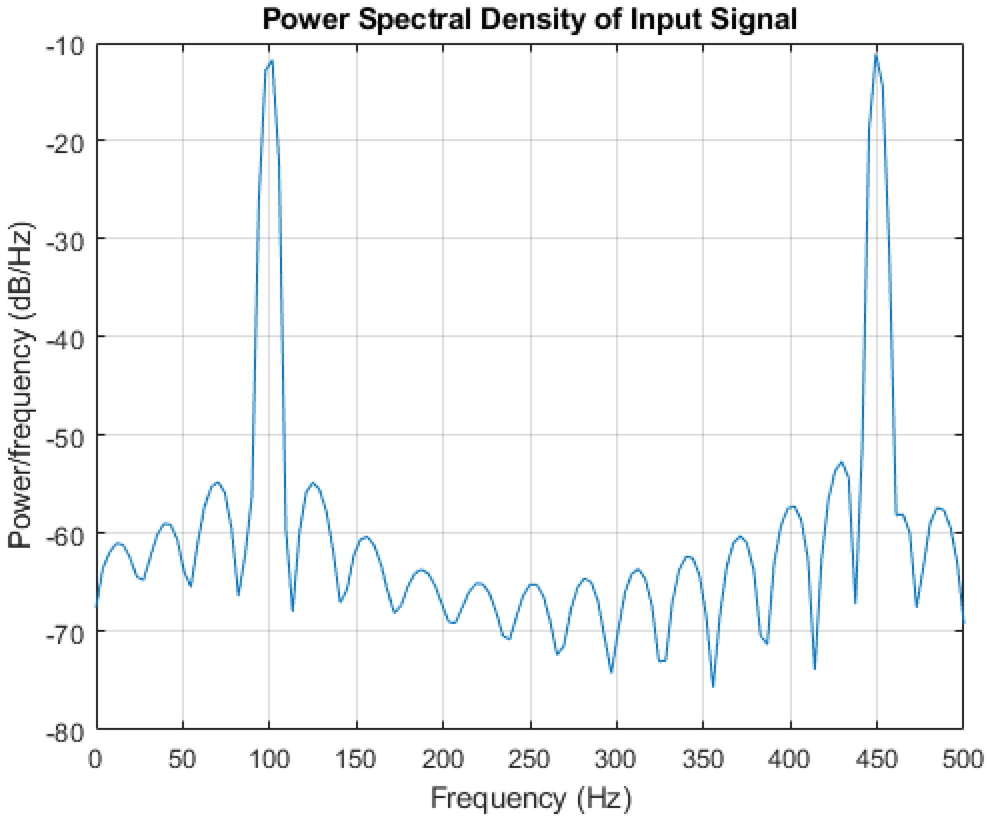
Use `modwpt` to obtain the energy and center frequencies of the signal. Plot the energy in the wavelet packets. The fourth and fifteenth frequency bands contain most of the energy. Other frequency bands have significantly less energy. The frequency ranges of fourth and fifteenth bands are approximately 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')
```



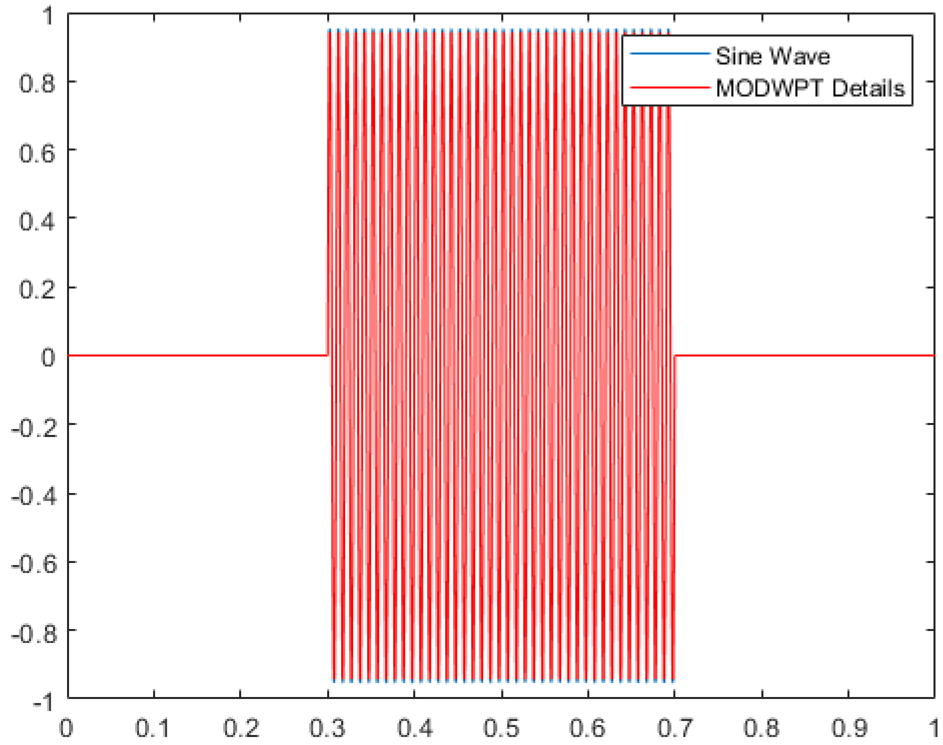
Plot the power spectral density of the input signal.

```
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')
```



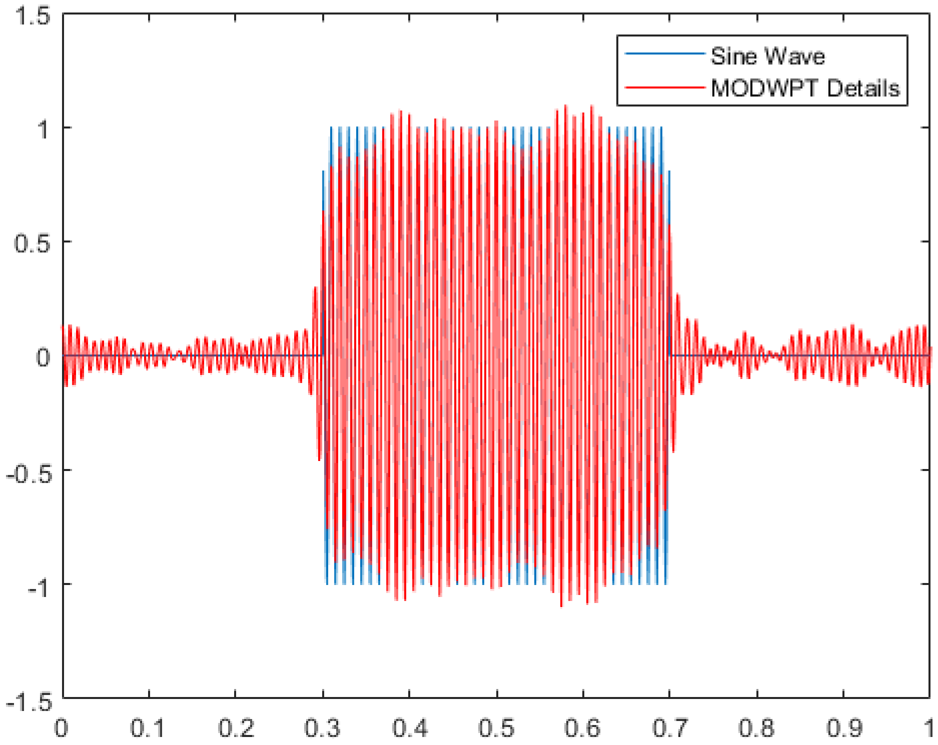
MODWPT Details for Noisy Sine Wave

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

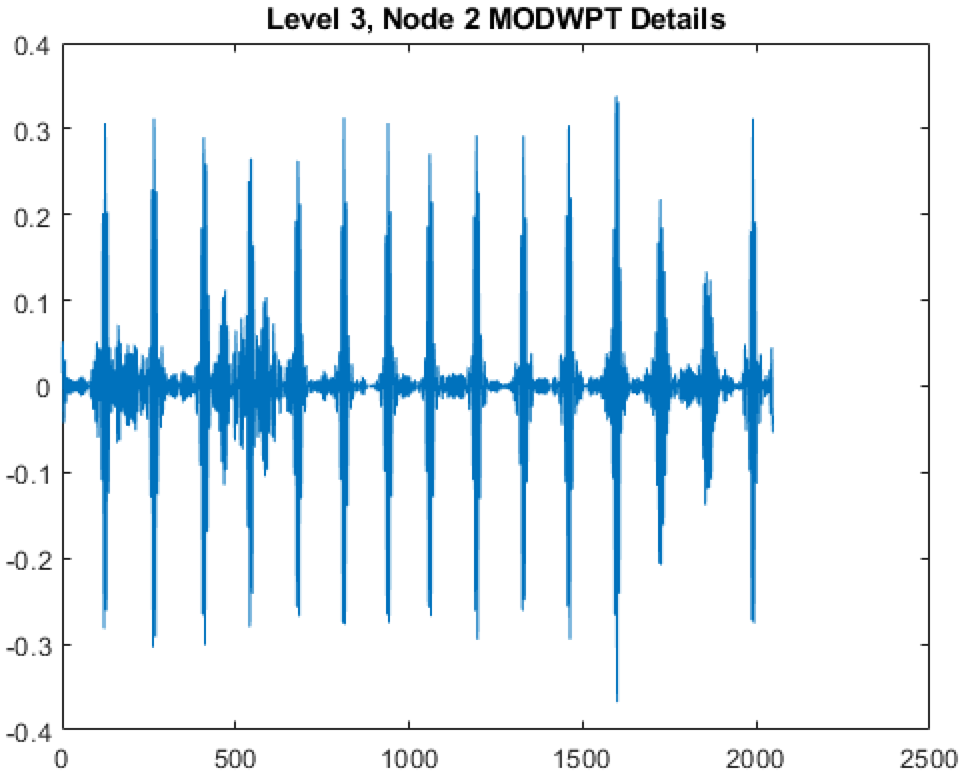
Obtain the MODWPT details of an ECG waveform using the length 18 Fejer-Korovkin scaling and wavelet filters.

```
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) | `'dbN'` | `'coifN'` | `'haar'` | `'fkN'` | `'symN'`

Analyzing wavelet, specified as one of the following:

- `'haar'` — Haar wavelet
- `'dbN'` — Extremal phase Daubechies wavelet with *N* vanishing moments, where *N* is a positive integer from 1 to 45.
- `'symN'` — Symlets wavelet with *N* vanishing moments, where *N* is a positive integer from 2 to 45.
- `'coifN'` — Coiflets wavelet with *N* vanishing moments, where *N* is a positive integer from 1 to 5.
- `'fkN'` — Fejér-Korovkin wavelet with *N* coefficients, where `N = 4, 6, 8, 14, 18` and `22`.

`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)))`.

`tf` — Return tree option

`false` (default) | `true`

Return tree option, specified as `false` or `true`. If `tf` is `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™.

Usage notes and limitations:

- The input `wname` must be constant.

See Also

`modwpt` | `imodwpt` | `waveinfo` | `wavemngr`

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 x. x can be a real- or complex-valued vector or matrix. If x is a matrix, modwt operates on the columns of x. modwt computes the wavelet transform down to level floor(log2(length(x))) if x is a vector and floor(log2(size(x,1))) if x is a matrix. By default, modwt uses the Daubechies least-asymmetric wavelet with four vanishing moments ('sym4') and periodic boundary handling.

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 both wname and a filter pair, Lo and Hi.

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 terminal end to twice the signal length. 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.

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 x.

Examples

MODWT Using Default Wavelet

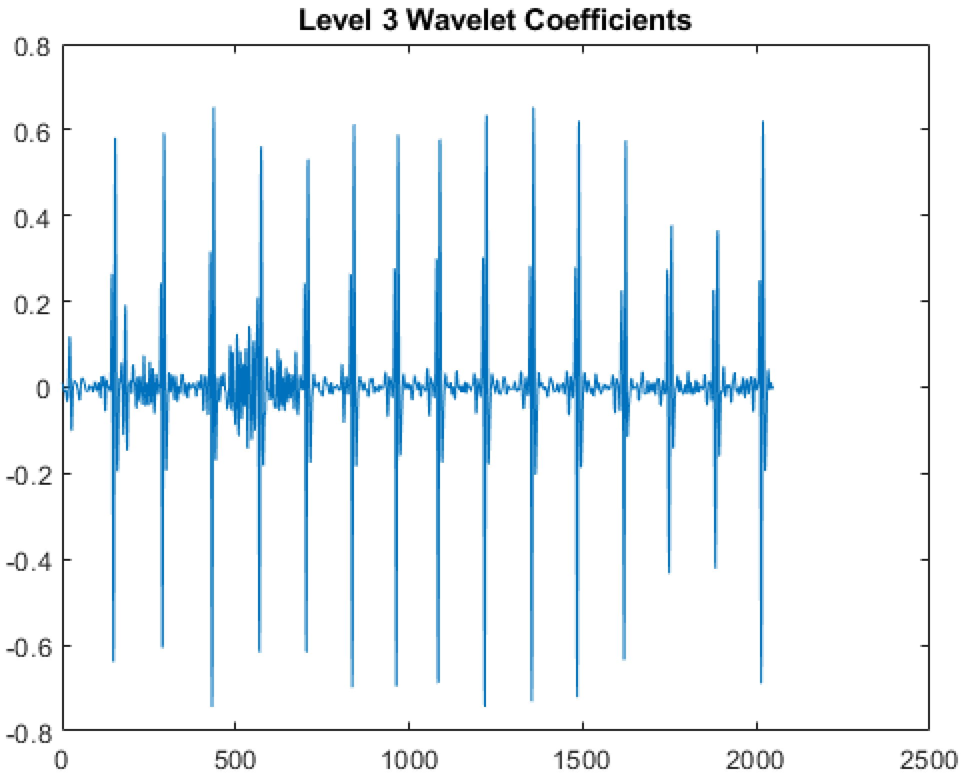
Obtain the MODWT of an electrocardiogram (ECG) signal using the default sym4 wavelet down to the maximum level. The data are taken from Percival & Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg;
wtecg = modwt(wecg);
whos wtecg
```

```
  Name         Size                Bytes  Class     Attributes

  wtecg       12x2048             196608  double
```

The first eleven rows of `wtecg` 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. The data are taken from Percival & Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg;
wtecg = modwt(wecg,4);
whos wecg wtecg
```

```
  Name           Size                Bytes  Class     Attributes

  wecg        2048x1                 16384  double              
  wtecg          5x2048              81920  double              
```

The row size of `wtecg` is L+1, where, in this case, the level (L) is 4. The column size matches 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. The data are taken from Percival & Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg;
wtecg = modwt(wecg,4,'reflection');
whos wecg wtecg
```

```
  Name           Size                 Bytes  Class     Attributes

  wecg        2048x1                  16384  double              
  wtecg          5x4096              163840  double              
```

`wtecg` has 4096 columns, which is twice the length of the input signal, `wecg`.

MODWT of Multisignal

Load the 23 channel EEG data `Espiga3` [3]. The channels are arranged column-wise. The data is sampled at 200 Hz.

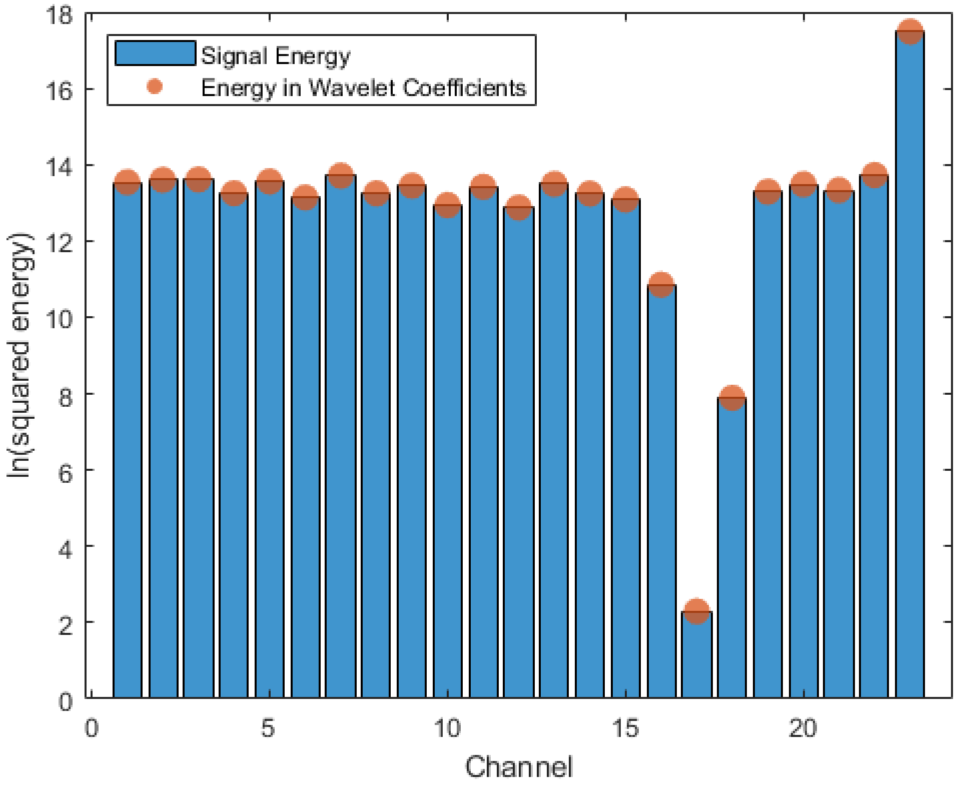
```
load Espiga3
```

Compute the maximal overlap discrete wavelet transform down to the maximum level.

```
wt = modwt(Espiga3);
```

Obtain the squared signal energies and compare them against the squared energies obtained from summing the wavelet coefficients over all levels. Use the log-squared energy due to the disproportionately large energy in one component.

```
sigN2 = vecnorm(Espiga3).^2;
wtN2 = sum(squeeze(vecnorm(wt,2,2).^2));
bar(1:23,log(sigN2))
hold on
scatter(1:23,log(wtN2),'filled','SizeData',100)
alpha(0.75)
legend('Signal Energy','Energy in Wavelet Coefficients', ...
       'Location','NorthWest')
xlabel('Channel')
ylabel('ln(squared energy)')
```

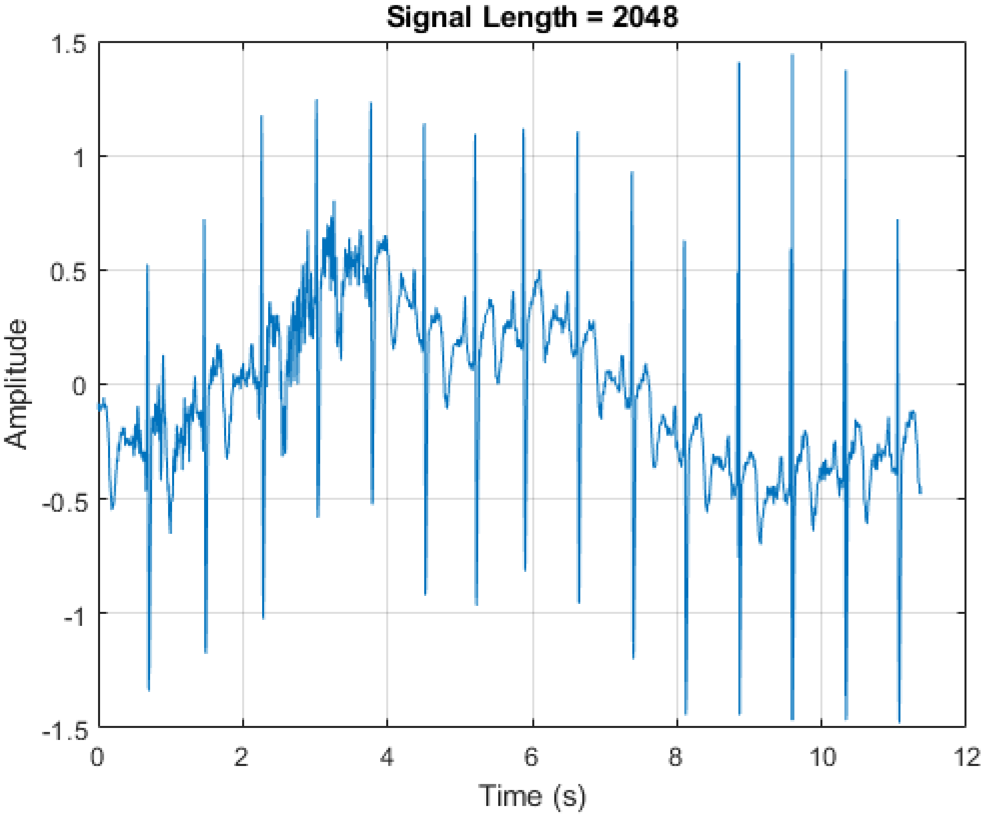


Comparing MODWT and MODWTMRA

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 electrocardiogram (ECG) signal. The sampling frequency for the ECG signal is 180 hertz. The data are taken from Percival and Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg
t = (0:numel(wecg)-1)/180;
wv = 'sym6';
plot(t,wecg)
grid on
title(['Signal Length = ',num2str(numel(wecg))])
xlabel('Time (s)')
ylabel('Amplitude')
```



Take the MODWT of the signal.

```
wtecg = modwt(wecg,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 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 $N = 2048$, $J_0 = \text{floor}(\log2(2048)) = 11$ and the number of rows in `wtecg` is $J_0 + 1 = 11 + 1 = 12$.

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';'A11'};
energy_table = table(Levels,energy_by_scales);
disp(energy_table)
```

```
    Levels     energy_by_scales
    _______    ________________

    {'D1' }         14.063     
    {'D2' }         20.612     
    {'D3' }         37.716     
    {'D4' }         25.123     
    {'D5' }         17.437     
    {'D6' }         8.9852     
    {'D7' }         1.2906     
    {'D8' }         4.7278     
    {'D9' }         12.205     
    {'D10'}         76.428     
    {'D11'}         76.268     
    {'A11'}         3.4192     
```

```
energy_total = varfun(@sum,energy_table(:,2))
```

```
energy_total=table
    sum_energy_by_scales
    ____________________

           298.28       
```

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(wecg.^2);
max(abs(energy_total.sum_energy_by_scales-energy_ecg))
```

```
ans = 7.4414e-10
```

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))-wecg(time_point))
```

```
ans = 3.0849e-13
```

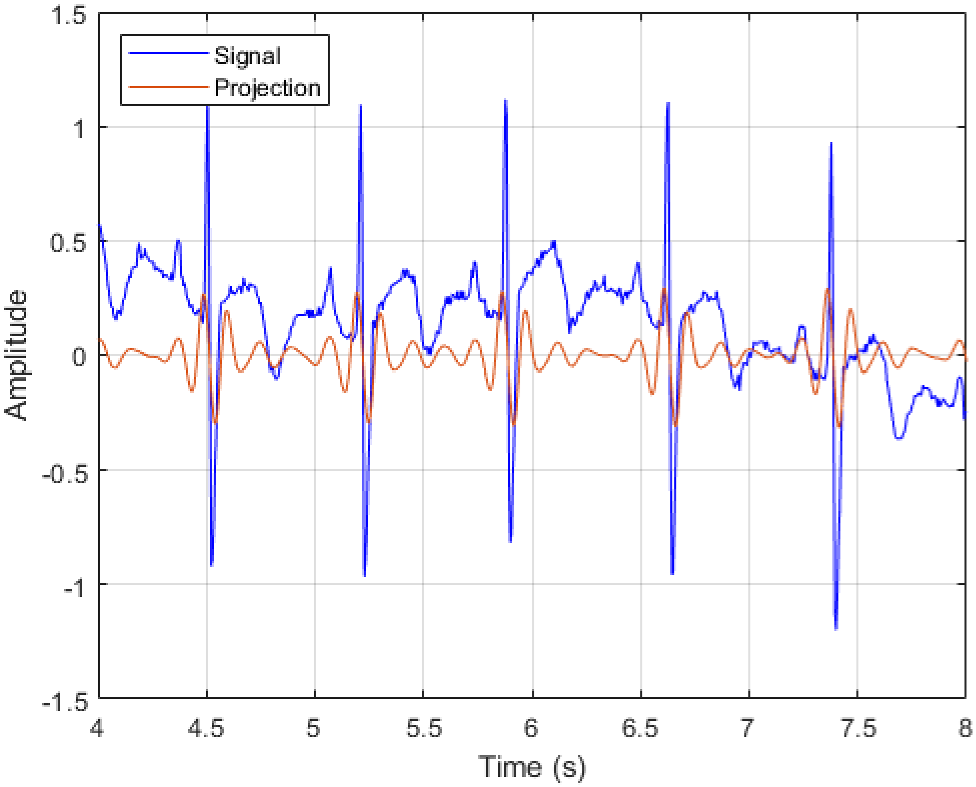
Confirm that, unlike MODWT, MODWTMRA is not an energy-preserving transform.

```
energy_ecg = sum(wecg.^2);
energy_mra_scales = sum(mraecg.^2,2);
energy_mra = sum(energy_mra_scales);
max(abs(energy_mra-energy_ecg))
```

```
ans = 115.7053
```

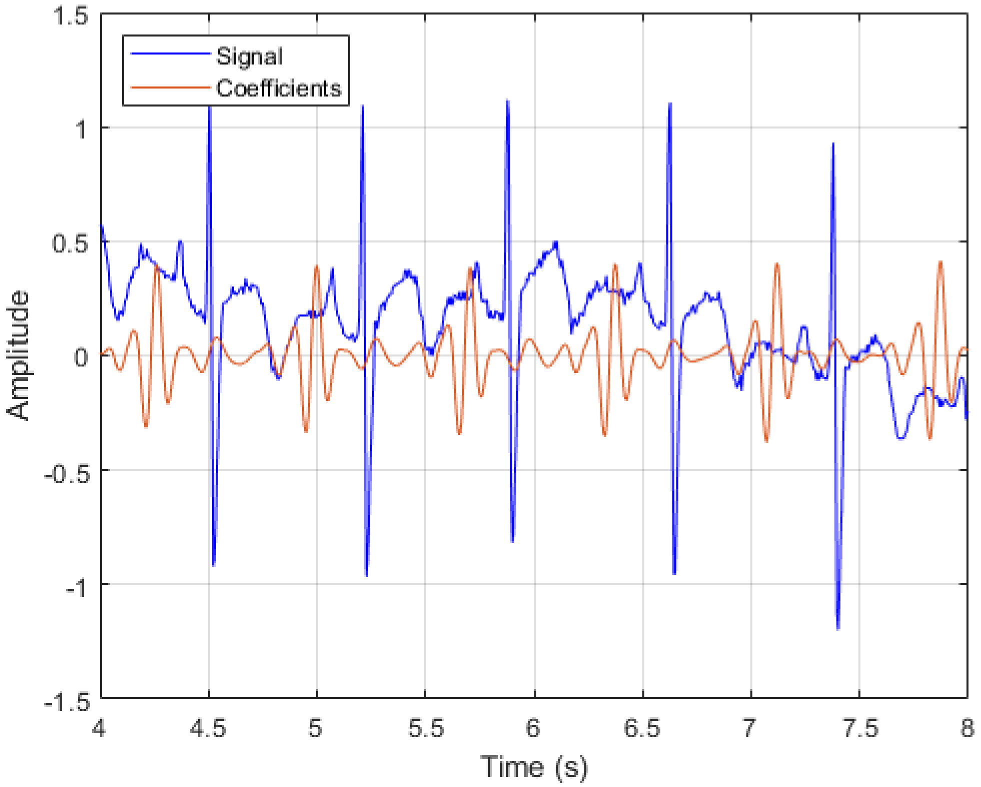
The MODWTMRA is a zero-phase filtering of the signal. Features will be time-aligned. Demonstrate this by plotting the original signal and one of its projections. To better illustrate the alignment, zoom in.

```
plot(t,wecg,'b')
hold on
plot(t,mraecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Projection','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



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.

```
plot(t,wecg,'b')
hold on
plot(t,wtecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Coefficients','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



Input Arguments

x — Input signal

vector | matrix

Input signal, specified as a vector or matrix. If `x` is a vector, `x` must have at least two elements. If `x` is a matrix, the row dimension of `x` must be at least 2.

Data Types: `single` | `double`

`wname` — Analyzing wavelet

`'sym4'` (default) | `'haar'` | `'db`*N*`'` | `'sym`*N*`'` | `'coif`*N*`'` | `'fk`*N*`'`

Analyzing wavelet, specified as one of the following:

- `'haar'` — Haar wavelet
- `'db`*N*`'` — Extremal phase Daubechies wavelet with `N` vanishing moments, where `N` is a positive integer from 1 to 45.
- `'sym`*N*`'` — Symlets wavelet with `N` vanishing moments, where `N` is a positive integer from 2 to 45.
- `'coif`*N*`'` — Coiflets wavelet with `N` vanishing moments, where `N` is a positive integer from 1 to 5.
- `'fk`*N*`'` — Fejér-Korovkin wavelet with `N` coefficients, where `N = 4, 6, 8, 14, 18` and `22`.

`Lo,Hi` — Filters

even-length real-valued vectors

Filters, specified as a pair of even-length real-valued vectors. `Lo` is the scaling filter, and `Hi` is the wavelet filter. The filters must satisfy the conditions for an orthogonal wavelet. The lengths of `Lo` and `Hi` must be equal. See `wfilters` for additional information. You cannot specify both a wavelet `wname` and filter pair `Lo,Hi`.

`lev` — Transform level
positive integer

Transform level, specified as a positive integer less than or equal to `floor(log2(N))`, where `N = length(x)` if `x` is a vector, or `N = size(x,1)` if `x` is a matrix. If unspecified, `lev` defaults to `floor(log2(N))`.

Output Arguments

`w` — MODWT transform
matrix | 3-D array

MODWT transform of `x`. `w` contains the wavelet coefficients and final-level scaling coefficients of `x`. If `x` is a vector, `w` is a `lev`+1-by-*N* matrix. If `x` is a matrix, `w` is a `lev`+1-by-*N*-by-*NC* array, where *NC* is the number of columns in `x`. *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 *k*th row of the array, `w`, contains the wavelet coefficients for scale 2^k (wavelet scale $2^{(k-1)}$). The final, (`lev`+1)th, row contains the scaling coefficients for scale 2^{lev}.

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

$$G_{j,k} = \prod_{m=0}^{j-1} G_{2^{m}k\,\mathrm{mod}\,N}$$

References

[1] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[2] Percival, Donald B., and Harold O. Mofjeld. “Analysis of Subtidal Coastal Sea Level Fluctuations Using Wavelets.” *Journal of the American Statistical Association* 92, no. 439 (September 1997): 868–80. https://doi.org/10.1080/01621459.1997.10474042.

[3] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps

Signal Multiresolution Analyzer

Functions

`imodwt` | `modwtmra` | `modwtcorr` | `modwtvar` | `modwtxcorr`

Topics

“Practical Introduction to Multiresolution Analysis”
“Time-Frequency Gallery”
“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')

[ ___ ] = modwtcorr( ___ ,'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.

[___] = modwtcorr(___ ,'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 remove 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 exchange 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 = 7×1

    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 = 10×1

    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 vanishing moments ('db2') 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,'db2');
[wcorr wcorrci]
```

```
ans = 7×3

    0.5854    0.4780    0.6756
    0.5748    0.4133    0.7013
    0.6264    0.4016    0.7800
    0.4948    0.0803    0.7634
    0.3787   -0.3295    0.8142
    0.9072    0.1247    0.9939
    0.7976   -0.2857    0.9860
```

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 = 7×3

    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 vanishing moments ('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 = 7×2

     2.694174887029436e-17     4.889927419958426e-16
     7.125460513473893e-09     6.466355415977557e-08
     7.012389783536670e-06     4.242495819039685e-05
     2.258540027996925e-02     1.024812537703605e-01
     2.805930327935258e-01     7.275376493146417e-01
```

```
    3.348079529469863e-02     1.215352869197560e-01
    1.059217509938030e-01     3.204132967562542e-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 vanishing moments ('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=7×6 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.

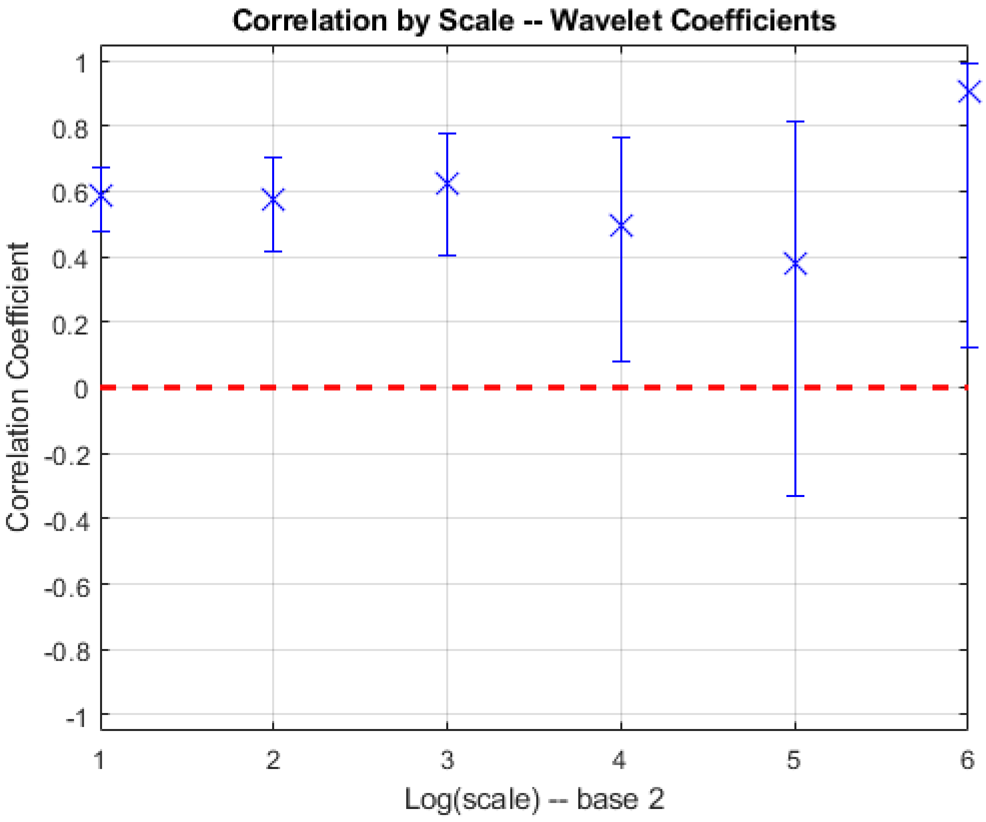
```
load soi
load truk
wsoi = modwt(soi,'fk4',6,'reflection');
wtruk = modwt(truk,'fk4',6,'reflection');
corrtable = modwtcorr(wsoi,wtruk,'fk4',0.95,'reflection','table')
```

```
corrtable=7×6 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')
```



Input Arguments

w1 — MODWT transform of signal 1
matrix

MODWT transform of signal 1, specified as a matrix. w1 is the output of modwt. w1 and w2 must be the same size and both 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. w1 and w2 must be the same size and both must have been obtained using the same analyzing wavelet.

wav — Wavelet

'sym4' (default) | character vector | string scalar | positive even scalar

Wavelet, specified as a character vector or string scalar 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 symlet wavelet with four vanishing moments, '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 vanishing moments, '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 N in the critically sampled DWT, `floor(size(w1,2)/2^LEV)`, where `LEV` is the level of the wavelet transform. `modwtcorr` returns NaNs when (N – 2) 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 A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: 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, pp. 14941–14962, 2000.

[3] Benjamini, Y., and Yekutieli, D. “The Control of the False Discovery Rate in Multiple Testing Under Dependency.” *Annals of Statistics*, Vol. 29, Number 4, pp. 1165–1188, 2001.

See Also

`modwtxcorr` | `modwtvar` | `modwt` | `modwtmra` | `imodwt`

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. By default, `modwtmra` assumes that you obtained `w` using the `'sym4'` wavelet with periodic boundary handling.

`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 column dimension of `w` is even and equals twice the length of the original signal.

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 `x`. By default, `modwtmra` uses periodic extension at the boundary.

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
```

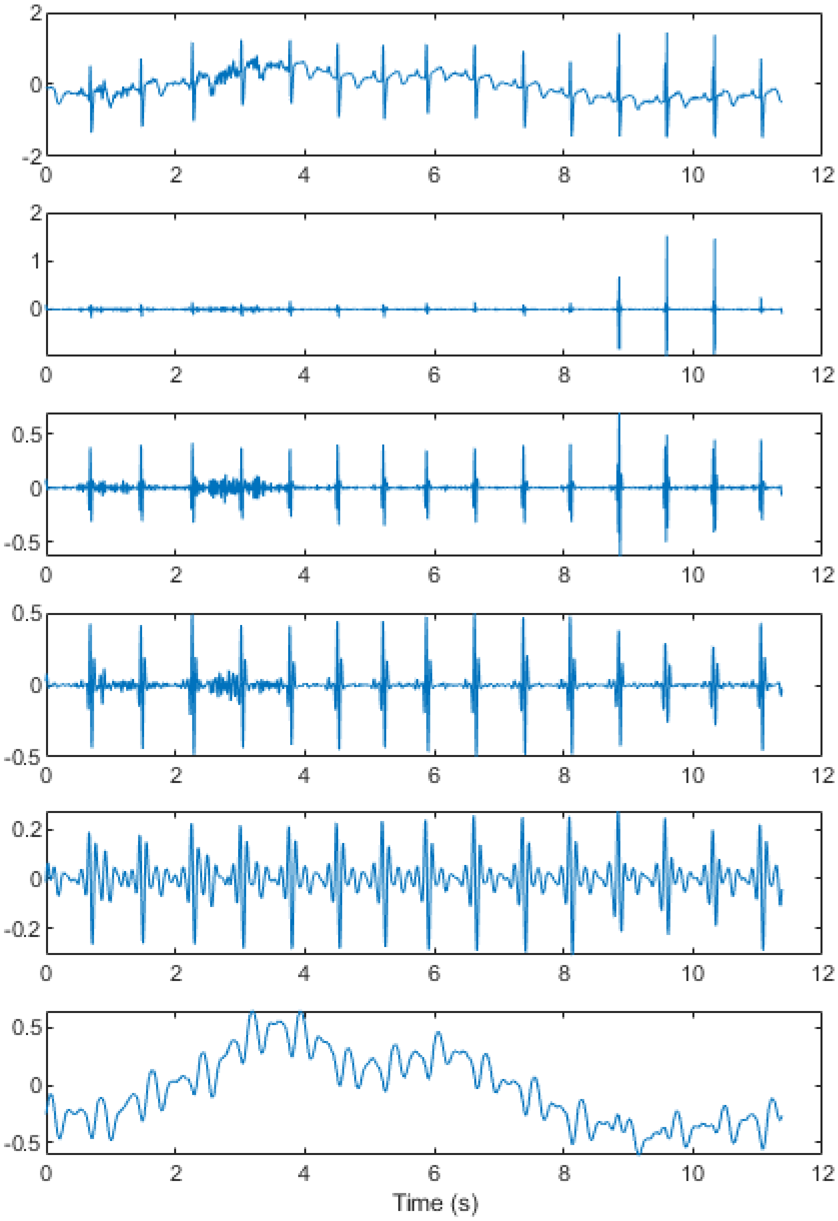
MRA Using Non-Default Wavelet

Construct an MRA of an ECG signal down to level four using the db2 wavelet. The data are taken from Percival & Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington). The sampling frequency for the ECG signal is 180 hertz.

```
load wecg;
lev = 4;
wtecg = modwt(wecg,'db2',lev);
mra = modwtmra(wtecg,'db2');
```

Plot the ECG waveform and the MRA.

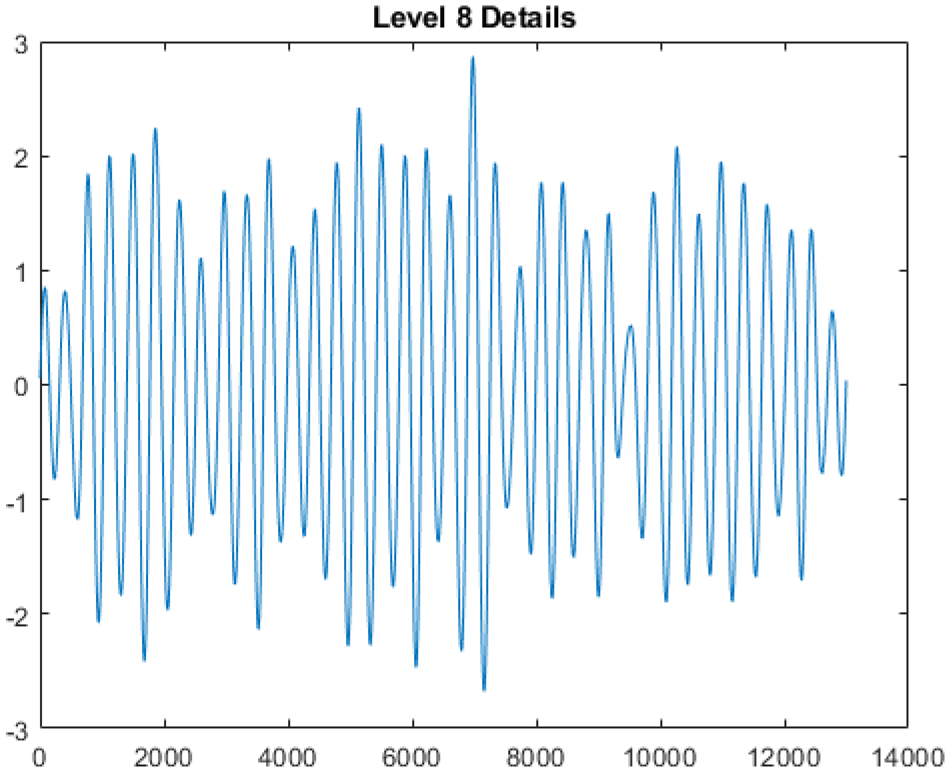
```
t = (0:numel(wecg)-1)/180;
subplot(6,1,1)
plot(t,wecg)
for kk = 2:lev+2
    subplot(6,1,kk)
    plot(t,mra(kk-1,:))
end
xlabel('Time (s)')
set(gcf,'Position',[0 0 500 700])
```



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. The data are taken from Percival & Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
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
```

MRA of Multisignal

Load the 23 channel EEG data `Espiga3` [3]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Obtain the MRA of the multisignal.

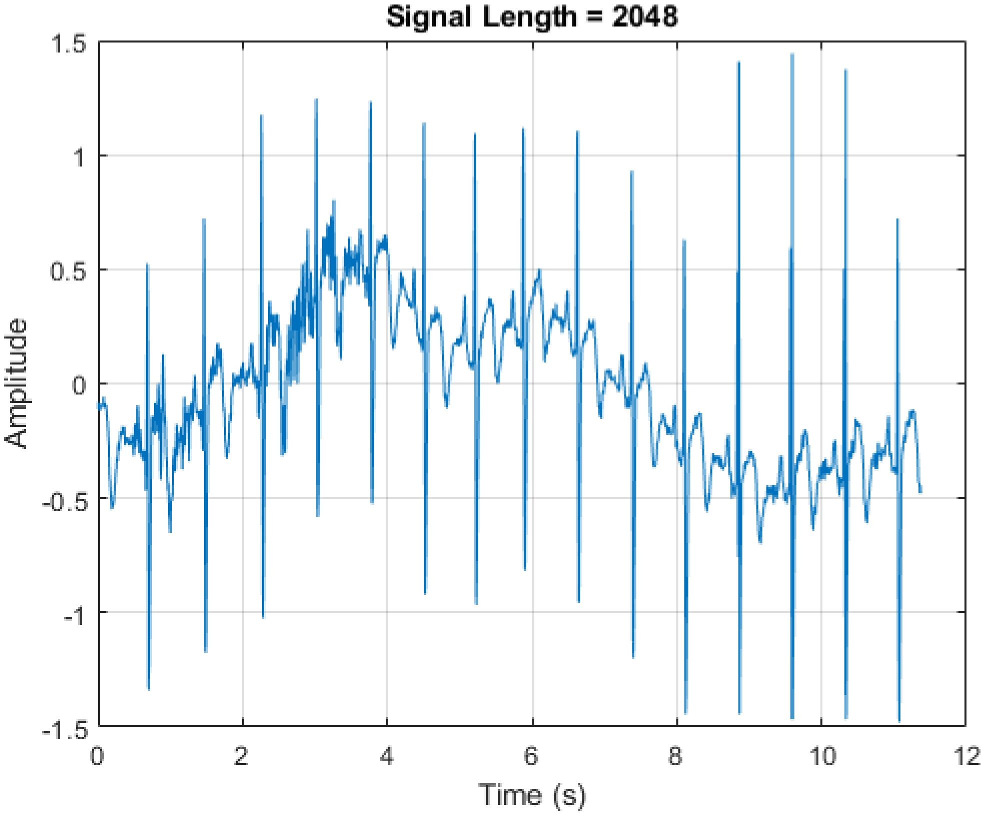
```
w = modwt(Espiga3);
mra = modwtmra(w);
```

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 electrocardiogram (ECG) signal. The sampling frequency for the ECG signal is 180 hertz. The data are taken from Percival and Walden (2000), p.125 (data originally provided by William Constantine and Per Reinhall, University of Washington).

```
load wecg
t = (0:numel(wecg)-1)/180;
wv = 'sym6';
plot(t,wecg)
grid on
title(['Signal Length = ',num2str(numel(wecg))])
xlabel('Time (s)')
ylabel('Amplitude')
```



Take the MODWT of the signal.

```
wtecg = modwt(wecg,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 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 $N = 2048$, $J_0 = \text{floor}(\text{log2}(2048)) = 11$ and the number of rows in `wtecg` is $J_0 + 1 = 11 + 1 = 12$.

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';'A11'};
energy_table = table(Levels,energy_by_scales);
disp(energy_table)
```

```
    Levels     energy_by_scales
    _______    ________________

    {'D1' }         14.063     
    {'D2' }         20.612     
    {'D3' }         37.716     
    {'D4' }         25.123     
    {'D5' }         17.437     
    {'D6' }         8.9852     
    {'D7' }         1.2906     
    {'D8' }         4.7278     
    {'D9' }         12.205     
    {'D10'}         76.428     
    {'D11'}         76.268     
    {'A11'}         3.4192     
```

```
energy_total = varfun(@sum,energy_table(:,2))
```

```
energy_total=table
    sum_energy_by_scales
    ____________________

           298.28       
```

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(wecg.^2);
max(abs(energy_total.sum_energy_by_scales-energy_ecg))
```

```
ans = 7.4414e-10
```

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))-wecg(time_point))
```

```
ans = 3.0849e-13
```

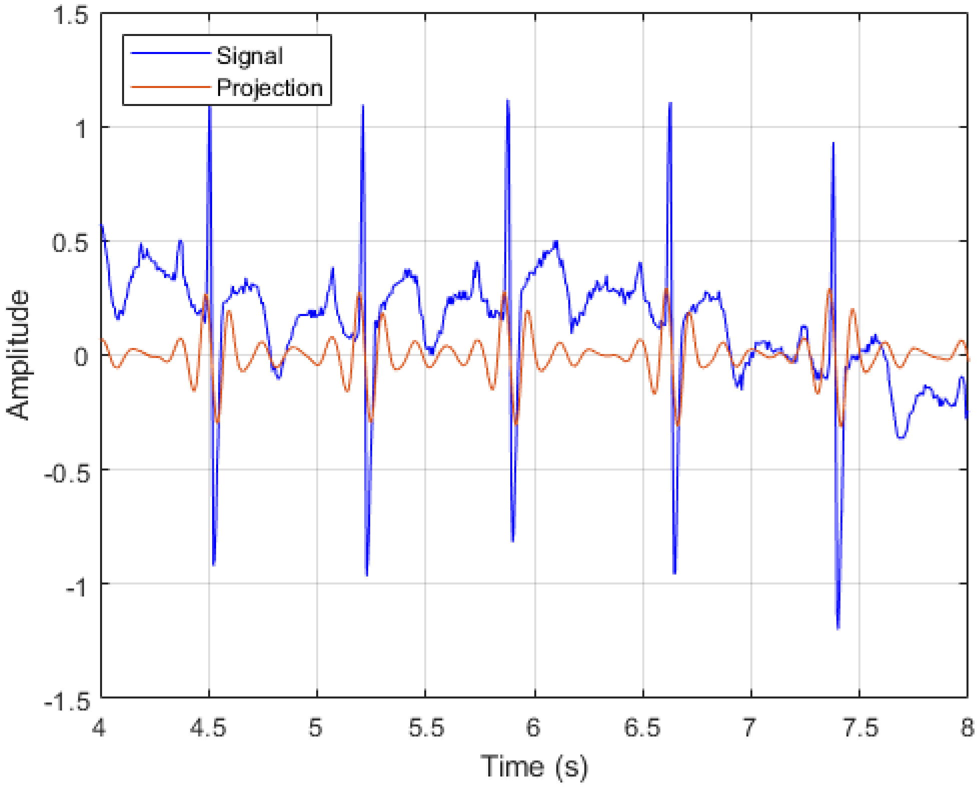
Confirm that, unlike MODWT, MODWTMRA is not an energy-preserving transform.

```
energy_ecg = sum(wecg.^2);
energy_mra_scales = sum(mraecg.^2,2);
energy_mra = sum(energy_mra_scales);
max(abs(energy_mra-energy_ecg))
```

```
ans = 115.7053
```

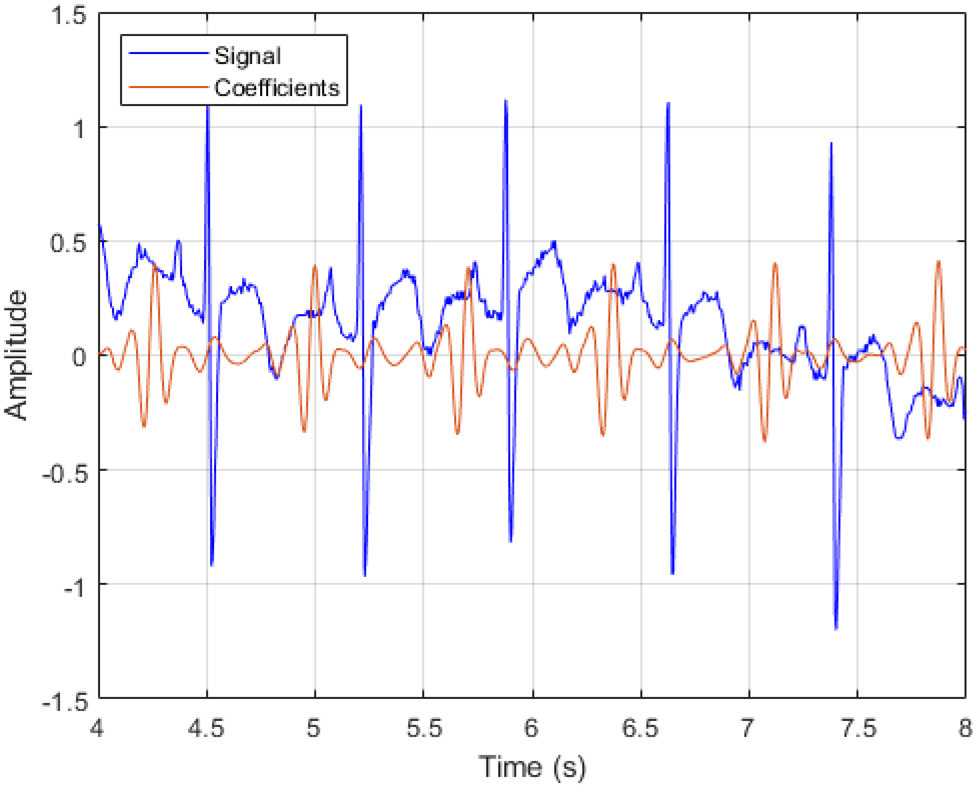
The MODWTMRA is a zero-phase filtering of the signal. Features will be time-aligned. Demonstrate this by plotting the original signal and one of its projections. To better illustrate the alignment, zoom in.

```
plot(t,wecg,'b')
hold on
plot(t,mraecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Projection','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



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.

```
plot(t,wecg,'b')
hold on
plot(t,wtecg(4,:),'-')
hold off
grid on
xlim([4 8])
legend('Signal','Coefficients','Location','northwest')
xlabel('Time (s)')
ylabel('Amplitude')
```



Input Arguments

w — MODWT transform

matrix

MODWT transform of a signal or multisignal down to level *LEV*, specified as a matrix or 3-D array, respectively. w is an *LEV*+1-by-*N* matrix for the MODWT of an *N*-point signal, and an *LEV*+1-by-*N*-by-*NC* array for the MODWT of an *N*-by-*NC* multisignal. By default, imodwt assumes that you obtained the MODWT using the 'sym4' wavelet with periodic boundary handling.

Data Types: single | double

wname — Synthesis wavelet

'sym4' (default) | character vector | string scalar

Synthesis wavelet, specified as a character vector or string scalar. The synthesis wavelet must be the same wavelet used to obtain the MODWT with the `modwt` function.

`Lo,Hi` — Filters

even-length real-valued vectors

Filters, specified as a pair of even-length real-valued vectors. `Lo` is the scaling filter, and `Hi` is the wavelet filter. `Lo` and `Hi` must be the same filters used in the analysis with `modwt`. The filters must satisfy the conditions for an orthogonal wavelet. The lengths of `Lo` and `Hi` must be equal. See `wfilters` for additional information. You cannot specify both a wavelet `wname` and filter pair `Lo,Hi`.

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.

Output Arguments

`mra` — Multiresolution analysis

matrix | 3-D array

Multiresolution analysis, returned as a matrix or 3-D array. `mra` is a *LEV*+1-by-*N* matrix or *LEV*+1-by-*N*-by-*NC* array where *LEV* is the level of the MODWT and *N* is the length of the analyzed signal. The k^{th} row of `mra` contains the details for the k^{th} level. The $(LEV+1)^{th}$ row of `mra` contains the LEV^{th} level smooth.

By default, `mra` is the same size as the input `w`. If you specify reflection boundary handling, then `mra` has one half the size of the column dimension as the input `w`.

References

[1] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[2] Whitcher, Brandon, Peter Guttorp, and Donald B. Percival. "Wavelet Analysis of Covariance with Application to Atmospheric Time Series." *Journal of Geophysical Research: Atmospheres* 105, no. D11 (June 16, 2000): 14941–62. https://doi.org/10.1029/2000JD900110.

[3] Mesa, Hector. "Adapted Wavelets for Pattern Detection." In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps

Signal Multiresolution Analyzer

Functions

`modwt` | `imodwt`

Topics

“Practical Introduction to Multiresolution Analysis”
“Time-Frequency Gallery”
“Wavelet Analysis of Financial Data”

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 moments. Compute the unbiased estimates of the wavelet variance by scale.

```
load soi
wsoi = modwt(soi);
wvar = modwtvar(wsoi)
```

```
wvar = 10×1

    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.

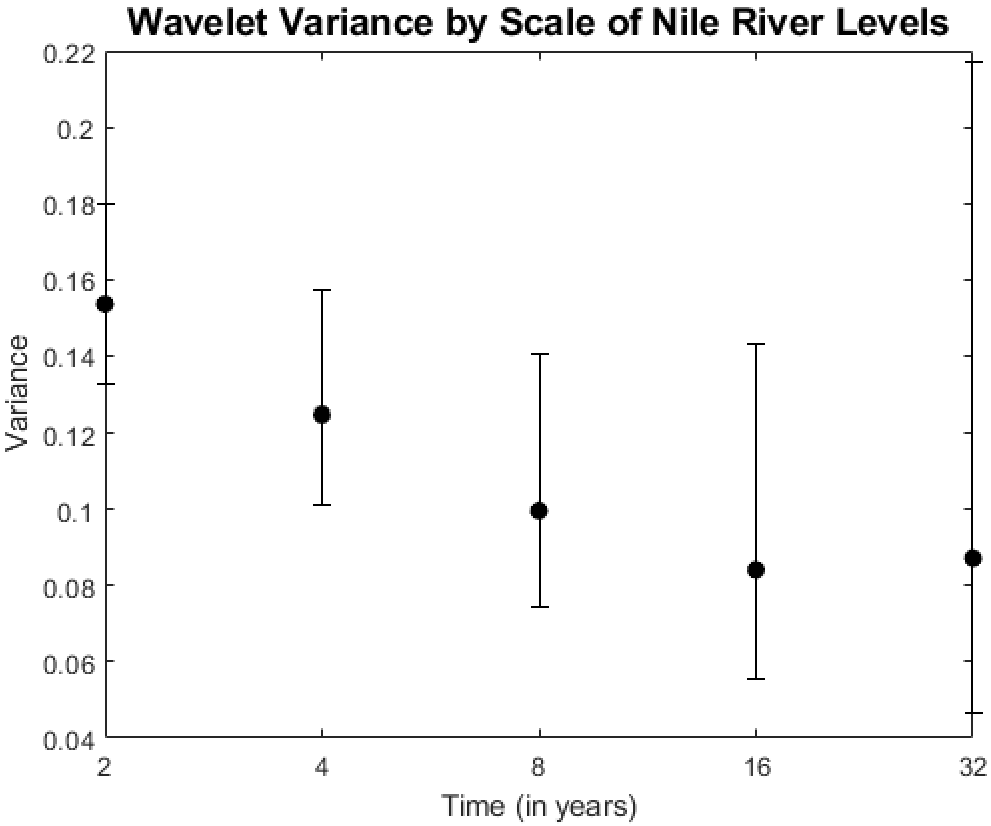
```
load soi
wsoi = modwt(soi,'db2');
wvar = modwtvar(wsoi,'db2')
```

```
wvar = 12×1

    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 = 10×3

    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');
```

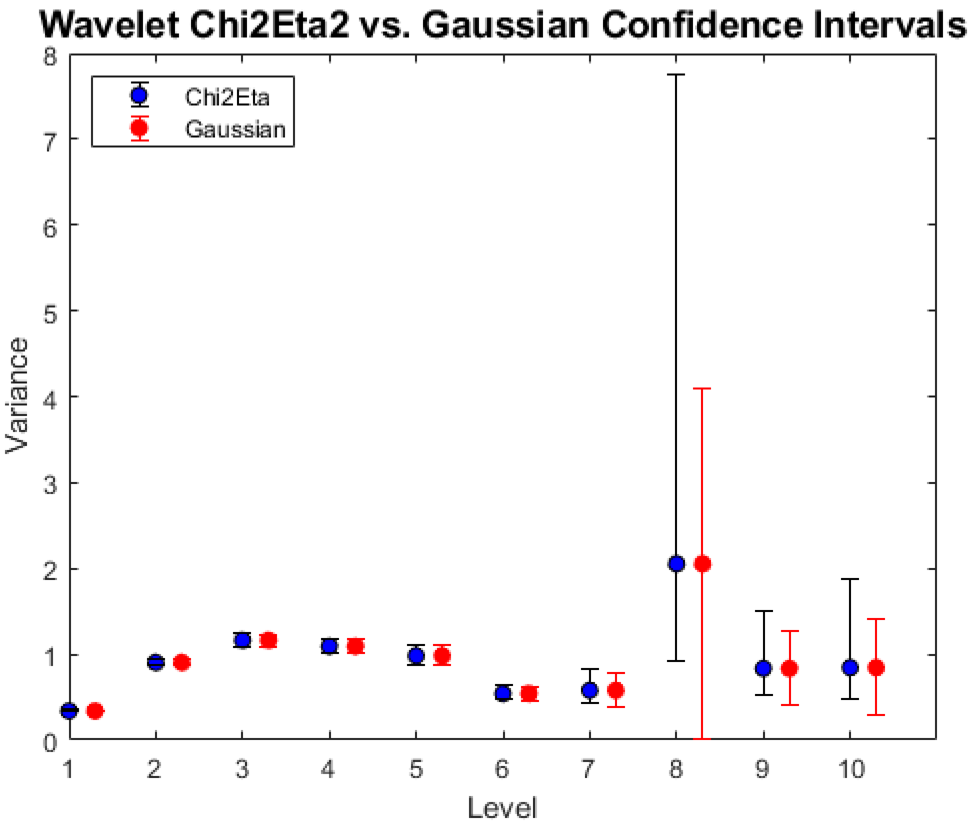
Use the Chi2Eta and Gaussian confidence methods to obtain the variances and confidence interval bounds for each method.

```
[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 = 10×2

       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=10×4 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 confidence level bounds, and the variance estimate for each level.

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 | string scalar | positive even scalar

Wavelet, specified as a character vector or string scalar 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. If you specify `'table'` as an input, the confidence levels are also shown in `wvartable`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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 1-1002 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

$$\eta_1 = \frac{N_j \widehat{v}_j^4}{\widehat{A}_j}$$

where

$$\widehat{A}_j = \frac{1}{2} \int_{-1/2}^{1/2} \left[\widehat{S}_j^{(p)}(f) \right]^2 df\,.$$

In this equation, $\widehat{S}_j^{(p)}$ is the spectral density function estimate of the wavelet coefficients at level j.

The chi-square statistic is

$$\frac{\eta_1 N_j \widehat{v}_j^2}{v_j^2} \sim X_{\eta_1}^2$$

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

$$\frac{\eta_3 N_j \widehat{v}_j^2}{v_j^2} \sim X_{\eta_3}^2$$

For the Gaussian method, the statistic

$$\frac{N_j^{1/2}\left((\widehat{v}_j^2 - v_j^2)\right)}{\left(2\widehat{A}_j\right)^{1/2}}$$

is distributed as `N(0,1)`. The variable $\widehat{A}_j$ 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.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.
- The input `conflevel` must be defined as a scalar.
- Plotting is not supported.

See Also

`modwtcorr` | `modwtxcorr` | `modwt` | `modwtmra` | `imodwt`

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

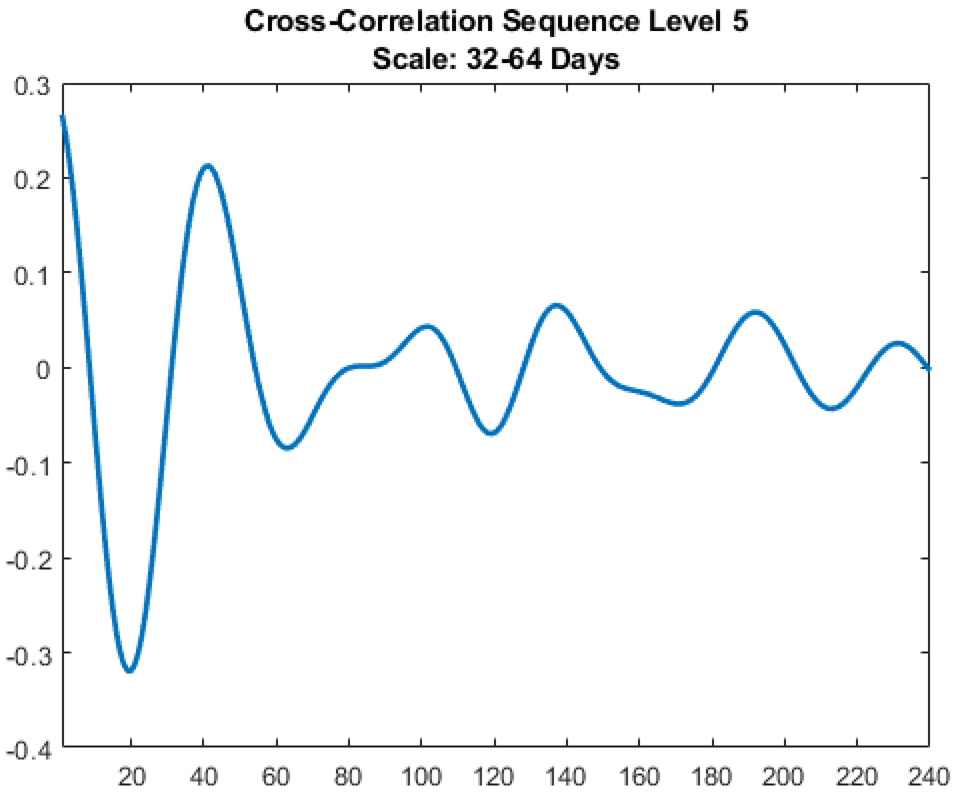
Cross-Correlation Sequence

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
```



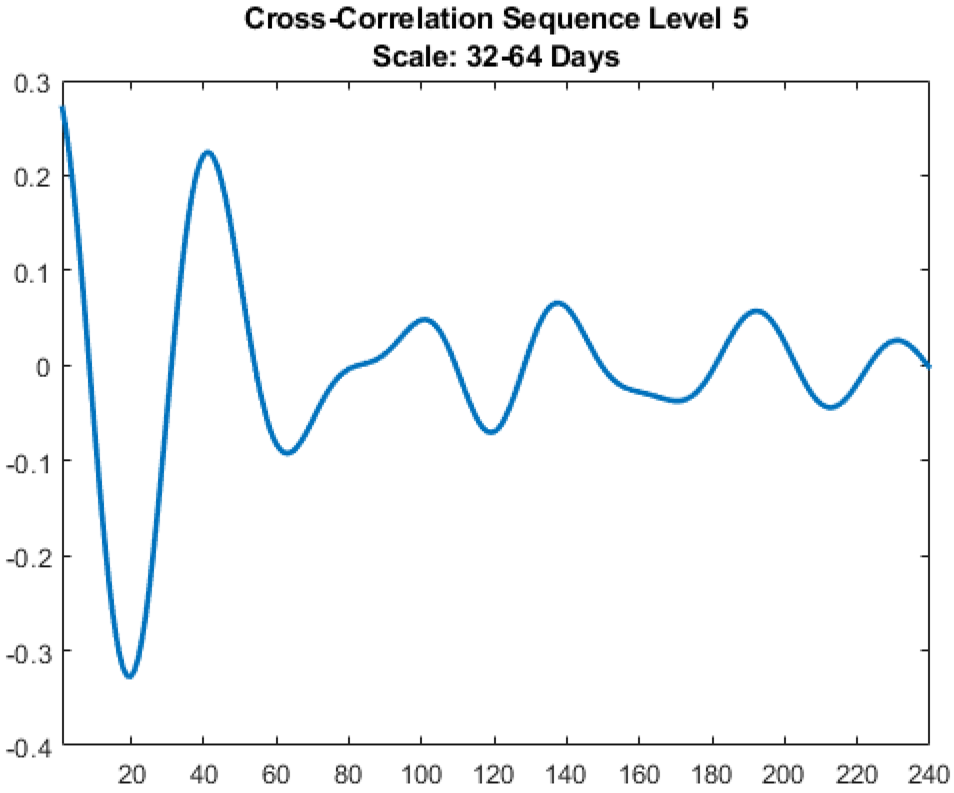
Cross-Correlation Sequence with Fejér-Korovkin Wavelet

Obtain the MODWT of the Southern Oscillation Index and Truk Islands pressure data using the Fejér-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
```

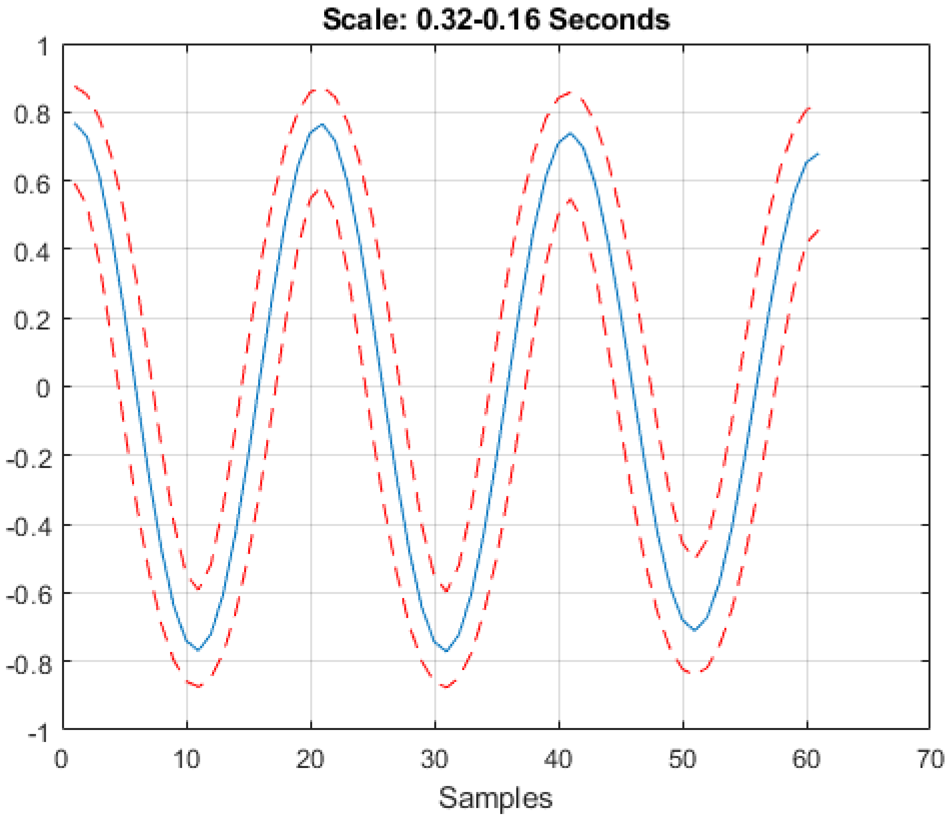


Cross-Correlation Confidence Intervals by Scale

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');
```



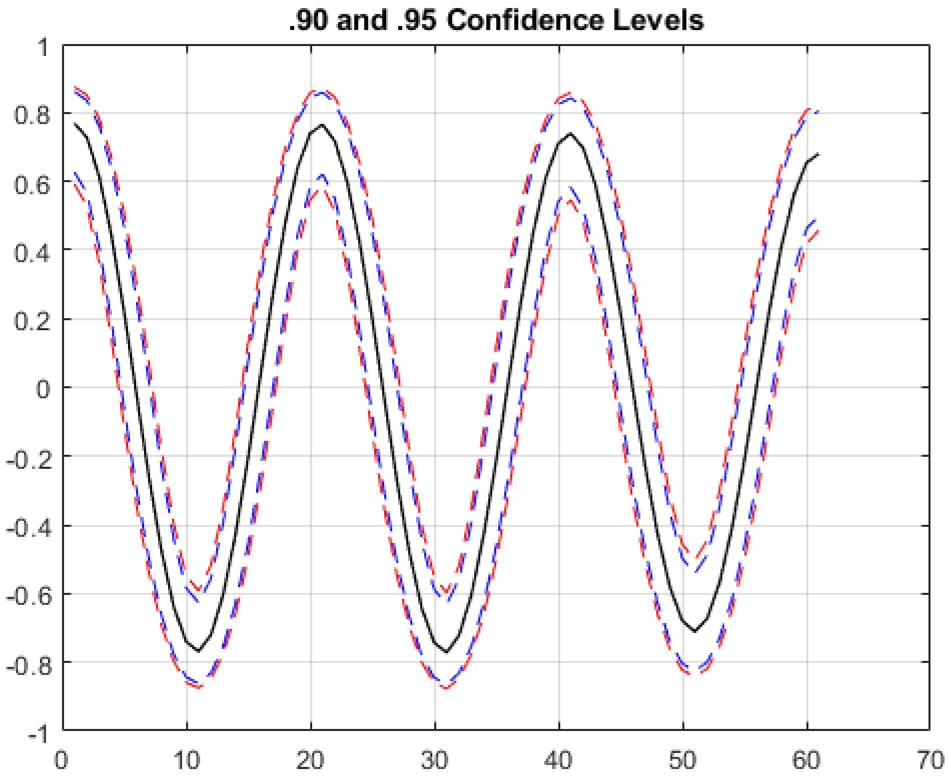
Cross-Correlation .90 and .95 Confidence Intervals Comparison

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 Fejér-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')
```

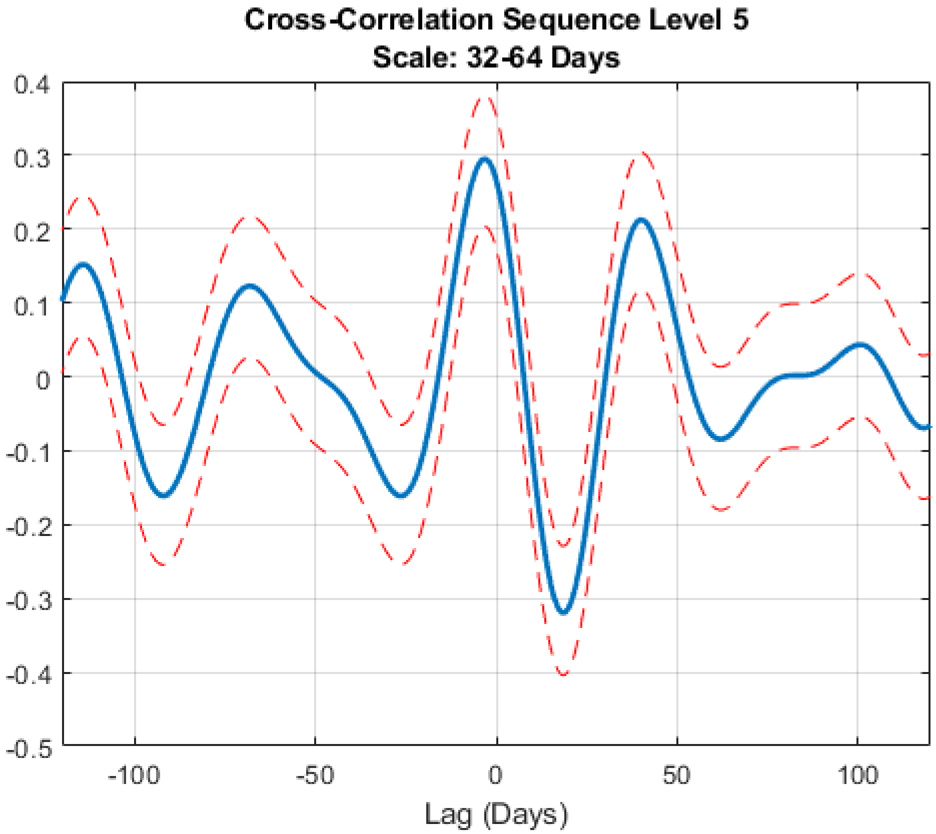


Notice that the .95 confidence interval width (in red) is larger than the .90 confidence interval width (in blue).

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` function 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 = 10×1

       25981
       25953
       25897
       25785
       25561
       25113
       24217
       22425
       18841
       11673
```

```
cellfun(@numel,xcseq_default)
```

```
ans = 10×1

       25981
       25953
       25897
       25785
       25561
       25113
       24217
       22425
       18841
       11673
```

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 | string scalar | positive even integer

Wavelet, specified as a character vector or string scalar, 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 2*NJ*-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 (2*NJ*-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 0th lag element is located at the index floor((2*NJ-1)/2)+1.

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: 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

`modwtcorr` | `modwtvar` | `modwt` | `modwtmra` | `imodwt`

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 the Morlet wavelet `psi` evaluated at `x`, an `n`-point regular grid in the interval `[lb, ub]`. The Morlet wavelet is defined as

$$\psi(x) = e^{-x^2/2}\cos(5x)$$

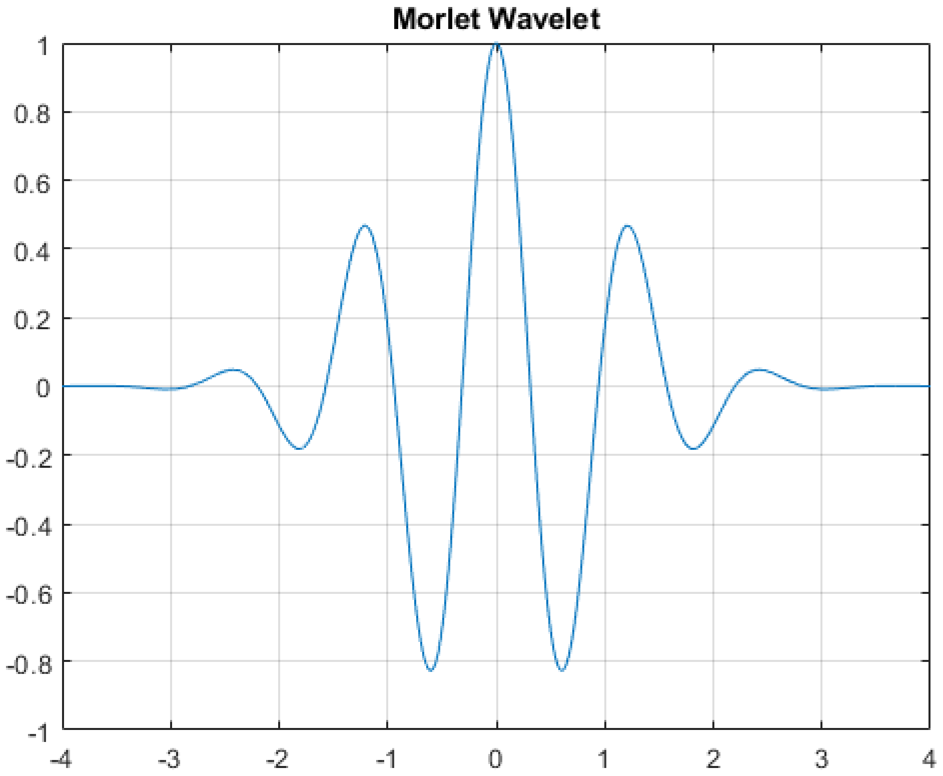
The Morlet wavelet has the interval [-4, 4] as effective support. Nearly 100% of the wavelet's energy is in the interval. Although [-4, 4] is the correct theoretical effective support, a wider effective support, [-8, 8], is used in the computation to provide more accurate results.

Examples

Morlet Wavelet

This example shows how to create a Morlet wavelet. The wavelet has an effective support of [-4, 4]. Use 1000 sample points.

```
lb = -4;
ub = 4;
n = 1000;
[psi,xval] = morlet(lb,ub,n);
plot(xval,psi)
grid on
title('Morlet Wavelet')
```



Compute the wavelet's energy in the interval. Normalize by the difference between sample points.

```
e1 = sum(psi.^2)*diff(xval(1:2));
fprintf('%.15f',e1)
```

```
0.886226920745597
```

Create a second Morlet wavelet with support on [-8, 8] using 1000 sample points. Compute the second wavelet's energy, normalized by the difference between sample points. Return the ratio of the two energies.

```
[psi2,xval2] = morlet(-8,8,1000);
e2 = sum(psi2.^2)*diff(xval2(1:2));
fprintf('%.15f',e1/e2)
```

```
0.999999994674672
```

Input Arguments

lb — Lower limit
real-valued scalar

Lower limit of interval, specified as a real-valued scalar.

ub — Upper limit
real-valued scalar

Upper limit of interval, specified as a real-valued scalar.

n — Number of points

positive integer

Number of sample points, specified as a positive integer.

Output Arguments

psi — Morlet wavelet

real-valued vector

Morlet wavelet, returned as a real-valued vector of length n.

x — Sampling instants

real-valued vector

Sampling instants, returned as a real-valued vector of length n.

See Also

waveinfo

Introduced before R2006a

mswcmp

Multisignal 1-D compression using wavelets

Syntax

```
[xc,deccmp,thresh] = mswcmp('cmp',dec,mthd)
[xc,deccmp,thresh] = mswcmp('cmp',dec,mthd,param)
```

```
[xc,thresh] = mswcmp('cmpsig',___)
[deccmp,thresh] = mswcmp('cmpdec',___)
thresh = mswcmp('thr',___)
```

```
[___] = mswcmp(option,dirdec,x,wname,lev,mthd)
[___] = mswcmp(option,dirdec,x,wname,lev,mthd,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,mthd)` returns a compressed version `xc` of the original multisignal `x`, whose wavelet decomposition structure is `dec`. The compression method is specified by `mthd`. The output `xc` is obtained by thresholding the wavelet coefficients. The output `deccmp` is the wavelet decomposition associated with `xc`, and `thresh` is the matrix of threshold values.

`[xc,deccmp,thresh] = mswcmp('cmp',dec,mthd,param)` uses the parameter `param` associated with `mthd`, if required.

`[xc,thresh] = mswcmp('cmpsig',___)` returns the compressed multisignal and computed thresholds if `'cmp'` in the first or second syntaxes is replaced with `'cmpsig'`.

`[deccmp,thresh] = mswcmp('cmpdec',___)` returns the wavelet decomposition associated with the compressed multisignal and computed thresholds if `'cmp'` in the first or second syntaxes is replaced with `'cmpdec'`.

`thresh = mswcmp('thr',___)` returns the computed thresholds if `'cmp'` in the first or second syntaxes is replaced with `'thr'`.

`[___] = mswcmp(option,dirdec,x,wname,lev,mthd)` decomposes the multisignal `x` to level `lev` using the wavelet specified by `wname` in the direction `dirdec` before performing a compression or computing the thresholds.

`[___] = mswcmp(option,dirdec,x,wname,lev,mthd,param)` uses the parameter `param` associated with `mthd`, if required.

`[___] = mswcmp(___,s_or_h)` applies the threshold rule specified by `s_or_h`.

`[___] = mswcmp(___,s_or_h,keepapp)` either keeps the approximation coefficients (`true`) or does not (`false`).

`[___] = mswcmp(___,s_or_h,keepapp,idxsig)` is a vector, which contains the indices of the initial signals.

Examples

Compress Multisignal

Load the 23-channel EEG data `Espiga3` [8]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the `db2` wavelet.

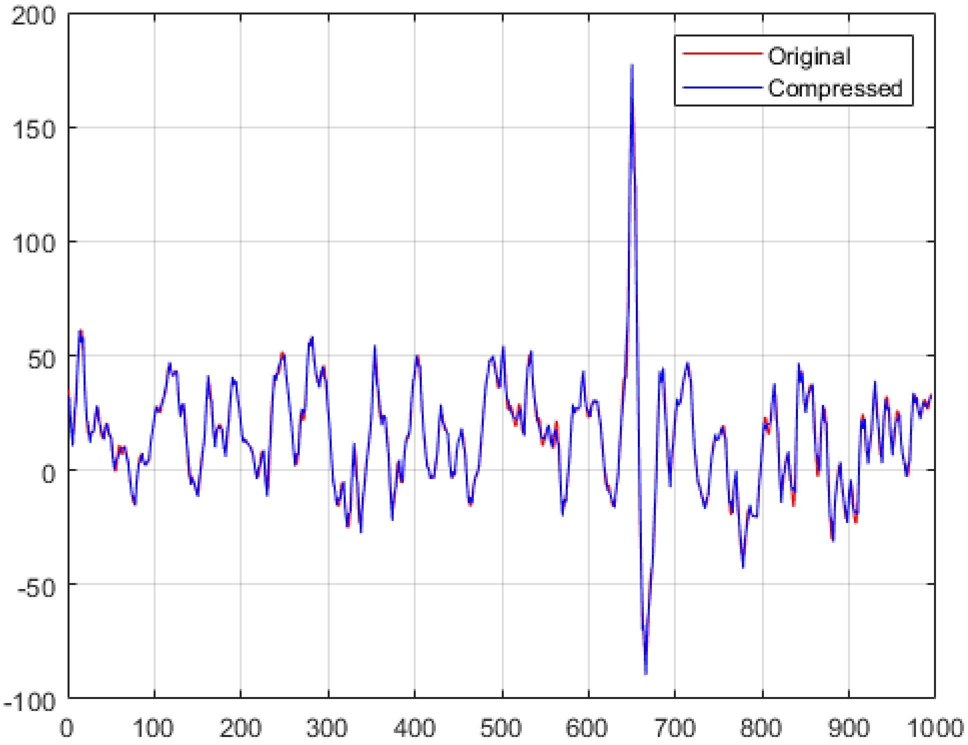
```
dec = mdwtdec('c',Espiga3,2,'db2');
```

Compress the signals to obtain a percentage of zeros near 95% for the wavelet coefficients.

```
[xr,deccmp,thresh] = mswcmp('cmp',dec,'N0_perf',95);
```

Plot an original signal, and the corresponding compressed signal.

```
idx = 3;
plot(Espiga3(:,idx),'r')
hold on
plot(xr(:,idx),'b')
grid on
legend('Original','Compressed')
```



Input Arguments

dec — Wavelet decomposition

structure

Wavelet decomposition, specified as a structure. `dec` is the output of `mdwtdec`.

`mthd` — Compression method

`'rem_n0'` | `'bal_sn'` | `'sqrtbal_sn'` | `'scarce'` | `'scarcehi'` | `'scarceme'` | `'scarcelo'` | `'L2_perf'` | `'N0_perf'` | `'glb_thr'` | `'man_thr'`

Compression method, specified as one of the values listed here. For methods that use an associated parameter, the range of allowable `param` values is shown.

For methods listed in the following table, `param` is a sparsity parameter, and it should be specified such that 1 ≤ `param` ≤ 10. For the `'scarce'` method no control is done.

method	Description
`'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

method	Description
'rem_n0'	Remove near 0
'bal_sn'	Balance sparsity-norm
'sqrtbal_sn'	Balance sparsity-norm (sqrt)

For methods listed in the following table, param is a real number, which represents the required performance: 0 ≤ param ≤ 100.

method	Description
'L2_perf'	Energy ratio
'N0_perf'	Zero coefficients ratio

To apply a global threshold for compression, specify the method 'glb_thr' and any positive real number param.

To apply a manual compression method, specify the method 'man_thr', and specify param as an *NbSig*-by-*NbLev* or an *NbSig*-by-(*NbLev*+1) real-valued matrix, where *NbSig* is the number of signals, and *NbLev* the number of levels of decomposition.

- param(*i*,*j*) is the threshold for the detail coefficients of level *j* for the *i*th signal (1 ≤ *j* ≤ *NbLev*).
- param(*i*,*NbLev*+1) is the threshold for the approximation coefficients for the *i*th signal (if keepapp is 0).

param — Parameter

real number | matrix

Parameter associated with the compression method mthd, specified as a real number or a real-valued matrix. For additional information, see mthd.

option — Compression outputs option

'cmp' | 'cmpsig' | 'cmpdec' | 'thr'

Compression outputs option, specified as one of the values listed here.

option	Description
'cmp'	Return the compressed signal, the associated wavelet decomposition, and the thresholds.
'cmpsig'	Return the compressed signal, and the thresholds.
'cmpdec'	Return the wavelet decomposition associated with the compressed signal, and the thresholds.
'thr'	Return the thresholds.

dirdec — Direction indicator

'r' | 'c'

Direction indicator of the wavelet decomposition, specified as one of the following:

- 'r': Take the 1-D wavelet decomposition of each row of x
- 'c': Take the 1-D wavelet decomposition of each column of x

x — Multisignal

real-valued matrix

Multisignal, specified as a real-valued matrix.

Data Types: `double`

`wname` — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. 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'`, `'db`*`N`*`'`, `'fk`*`N`*`'`, `'coif`*`N`*`'`, or `'sym`*`N`*`'`, where *N* is the number of vanishing moments for all families except `fk`. For `fk`, *N* is the number of filter coefficients.
- Valid biorthogonal wavelet families begin with `'bior`*`Nr`*`.`*`Nd`*`'` or `'rbio`*`Nd`*`.`*`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).

`lev` — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer. `mdwtdec` does not enforce a maximum level restriction. Use `wmaxlev` to ensure that the wavelet coefficients are free from boundary effects. If boundary effects are not a concern, a good rule is to set `lev` less than or equal to `fix(log2(length(`*`N`*`)))`, where *N* is the number of samples in the 1-D data.

`s_or_h` — Type of thresholding

`'h'` (default) | `'s'`

Type of thresholding to perform, specified as either of the following:

- `'s'` — Soft thresholding
- `'h'` — Hard thresholding

keepapp — Threshold approximation

`false` or `0` (default) | `true` or `1`

Threshold approximation setting:

- `0` — Approximation coefficients are thresholded
- `1` — Approximation coefficients are not thresholded

`idxsig` — Indices of initial signals

`'all'` (default) | vector of positive integers

Indices of initial signals, specified as a vector of positive integers, or `'all'`.

Output Arguments

`xc` — Compressed multisignal

real-valued matrix

Compressed multisignal, returned as a real-valued matrix.

`deccmp` — Wavelet decomposition

structure

Wavelet decomposition of the compressed multisignal `x`, returned as a structure with the following fields:

- `dirDec` — Direction indicator: `'r'` (row) or `'c'` (column)
- `level` — Level of wavelet decomposition
- `wname` — Wavelet name
- `dwtFilters` — Structure with four fields: `LoD`, `HiD`, `LoR`, and `HiR`
- `dwtEXTM` — DWT extension mode
- `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`

The coefficients `ca` and `cd{k}`, for *k* from 1 to `lev`, are matrices and are stored in rows if `dirdec = 'r'` or in columns if `dirdec = 'c'`.

`thresh` — Threshold values

real-valued matrix

Threshold values used in the compression, returned as a real-valued matrix.

References

[1] Birgé, L., and P. Massart. “From Model Selection to Adaptive Estimation.” *Festschrift for Lucien Le Cam: Research Papers in Probability and Statistics* (E. Torgersen, D. Pollard, and G. Yang, eds.). New York: Springer-Verlag, 1997, pp. 55–88.

[2] DeVore, R. A., B. Jawerth, and B. J. Lucier. “Image Compression Through Wavelet Transform Coding.” *IEEE Transactions on Information Theory*. Vol. 38, Number 2, 1992, pp. 719–746.

[3] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[4] Donoho, D. L., and I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*. Vol. 81, pp. 425–455, 1994.

[5] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society, series B*, Vol. 57, No. 2, pp. 301–369, 1995.

[6] Donoho, D. L., and I. M. Johnstone. “Ideal denoising in an orthonormal basis chosen from a library of bases.” *C. R. Acad. Sci. Paris, Ser. I*, Vol. 319, pp. 1317–1322, 1994.

[7] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*. Vol. 42, Number 3, pp. 613–627, 1995.

[8] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

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 row-wise if DEC.dirDec is equal to 'r' or column-wise 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 ≤ j ≤ NbCFS), and L2SCR(:,1)=100.
- N0SCR(i,j) is the percentage of zeros corresponding to a threshold equal to CFS(i,j-1) (2 ≤ j ≤ 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 'all'.

The defaults are, respectively, 'h', false and 'all'.

Examples

Multisignal Compression Scores

Load the 23 channel EEG data `Espiga3` [4]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

Compute the compression performances for soft and hard thresholding.

```
[THR_S,L2SCR_S,N0SCR_S] = mswcmpscr(dec,'s');
[THR_H,L2SCR_H,N0SCR_H] = mswcmpscr(dec,'h');
```

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

[4] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

See Also

`mdwtdec` | `mdwtrec` | `ddencmp` | `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 `'all'`.

The defaults are, respectively, `'h'`, false and `'all'`.

Examples

Multisignal Compression Thresholds and Performance

Load the 23 channel EEG data `Espiga3` [4]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

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

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

[4] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

See Also

`mdwtdec` | `mdwtrec` | `ddencmp` | `wdencmp`

Introduced in R2007a

mswden

Multisignal 1-D denoising using wavelets

Note mswden is no longer recommended. Use wdenoise instead.

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 'all'.

The defaults are, respectively, 'h', false and 'all'.

Examples

Multisignal 1-D Wavelet Denoising

Load the 23 channel EEG data Espiga3 [8]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

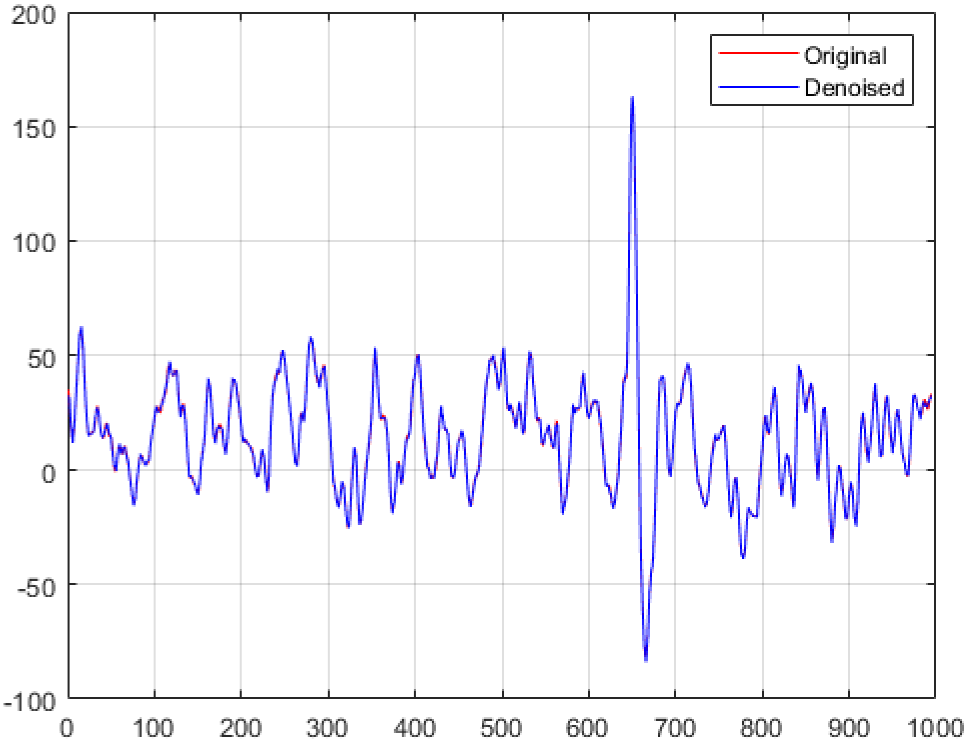
```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

Denoise the signals using the universal method of thresholding (`sqtwolog`) and the `'sln'` threshold rescaling (with a single estimation of level noise, based on the first level coefficients).

```
[xd,decden,thresh] = mswden('den',dec,'sqtwolog','sln');
```

Plot an original signal, and the corresponding denoised signal.

```
idxA = 3;
plot(Espiga3(:,idxA),'r')
hold on
plot(xd(:,idxA),'b')
grid on
legend('Original','Denoised')
```



References

[1] Birgé, L., and P. Massart. “From Model Selection to Adaptive Estimation.” *Festschrift for Lucien Le Cam: Research Papers in Probability and Statistics* (E. Torgersen, D. Pollard, and G. Yang, eds.). New York: Springer-Verlag, 1997, pp. 55–88.

[2] DeVore, R. A., B. Jawerth, and B. J. Lucier. “Image Compression Through Wavelet Transform Coding.” *IEEE Transactions on Information Theory*. Vol. 38, Number 2, 1992, pp. 719–746.

[3] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[4] Donoho, D. L., and I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*. Vol. 81, pp. 425–455, 1994.

[5] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society, series B*, Vol. 57, No. 2, pp. 301–369, 1995.

[6] Donoho, D. L., and I. M. Johnstone. “Ideal denoising in an orthonormal basis chosen from a library of bases.” *C. R. Acad. Sci. Paris, Ser. I*, Vol. 319, pp. 1317–1322, 1994.

[7] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*. Vol. 42, Number 3, pp. 613–627, 1995.

[8] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

See Also

Functions
`mdwtdec` | `mdwtrec` | `mswthresh` | `wthresh` | `wdenoise`

Apps
Wavelet Signal Denoiser

Introduced in R2007a

mswthresh

Perform multisignal 1-D thresholding

Syntax

```
Y = mswthresh(X,sorh,T)
Y = mswthresh(X,sorh,T,'c')
```

Description

`Y = mswthresh(X,sorh,T)` returns the soft or hard `T`-thresholding of the matrix `X`. `T` can be a single value, a matrix the same size as `X`, or a vector. If `T` is a vector, thresholding is performed row-wise, and `LT = length(T)` must be such that `size(X,1) ≤ LT`. Only the first `size(X,1)` values of `T` are used.

`Y = mswthresh(X,sorh,T,'c')` performs thresholding column-wise, and `LT = length(T)` must be such that `size(X,2) ≤ LT`. Only the first `size(X,2)` values of `T` are used.

Examples

Multisignal Thresholding

Create a 3-by-3 matrix and a 1-by-3 vector of threshold values.

```
mat = [1 1 3; 1 1 3; 2 2 3]
```

```
mat = 3×3

     1     1     3
     1     1     3
     2     2     3
```

```
thr = [1 2 3]
```

```
thr = 1×3

     1     2     3
```

Apply soft thresholding to the matrix row-wise. The *k*th threshold in `thr` is applied to the *k*th row of `mat`.

```
mswthresh(mat,'s',thr)
```

```
ans = 3×3

     0     0     2
     0     0     1
     0     0     0
```

Apply soft thresholding to the matrix column-wise. The *k*th threshold in `thr` is applied to the *k*th column of `mat`.

```
mswthresh(mat,'s',thr,'c')
```

```
ans = 3×3

     0     0     0
     0     0     0
     1     0     0
```

Apply hard thresholding to the matrix row-wise.

```
mswthresh(mat,'h',thr)
```

```
ans = 3×3

     0     0     3
     0     0     3
     0     0     0
```

Apply hard thresholding to the matrix column-wise.

```
mswthresh(mat,'h',thr,'c')
```

```
ans = 3×3

     0     0     0
     0     0     0
     2     0     0
```

Input Arguments

X — Input data
real-valued matrix

Input data to threshold, specified as a real-valued matrix.

Data Types: `double`

`sorh` — Type of thresholding
`'s'` | `'h'`

Type of thresholding to perform, specified as:

- `'s'` — Soft thresholding
- `'h'` — Hard thresholding

T — Threshold value
real-valued scalar or vector

Threshold value, specified as a real-valued scalar or vector.

Data Types: `double`

Output Arguments

Y — Thresholded data

real-valued matrix

Thresholded data, returned as a real-valued matrix. Y has the same dimensions as X.

Algorithms

- If `sorh` is `'s'`, Y is the soft thresholding of X: $Y = \text{sign}(X) \cdot (|X| - T)_+$ where

  $$(x)_+ = \begin{cases} x & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

  Soft thresholding is wavelet shrinkage.
- If `sorh` is `'h'`, Y is the hard thresholding of X: $Y = X \cdot \mathbf{1}_{(|X| > T)}$ where

  $$\mathbf{1}_{(|X| > T)} = \begin{cases} 1 & \text{if} \quad |X| > T \\ 0 & \text{otherwise} \end{cases}$$

  Hard thresholding is cruder than soft thresholding.

See Also

`wdenoise` | `mswden` | `mswcmp` | `wthresh` | `wden` | `wdencmp` | `wpdencmp`

Introduced in R2007a

mtimes

Laurent polynomial or Laurent matrix multiplication

Syntax

```
Q = mtimes(A,B)
Q = A * B
```

Description

`Q = mtimes(A,B)` returns the product of the pair of Laurent polynomials or Laurent matrices `A` and `B`.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `mtimes`. The input data type determines which version is executed.

`Q = A * B` is equivalent to `Q = mtimes(A,B)`.

Examples

Laurent Polynomial Multiplication

Create three Laurent polynomials:

- $a(z) = 4z + z^{-1}$
- $b(z) = 2z^2 + 3z + z^{-1}$
- $c(z) = z^3 + 3z^2 + 5z + 7$

```
a = laurentPolynomial(Coefficients=[4 0 1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[2 3 0 1],MaxOrder=2);
c = laurentPolynomial(Coefficients=[1 3 5 7],MaxOrder=3);
```

Multiply $a(z)$ and $b(z)$.

```
ab = mtimes(a,b)
```

```
ab = 
  laurentPolynomial with properties:

    Coefficients: [8 12 2 7 0 1]
        MaxOrder: 3
```

Compute $a(z)c(z) - b(z)$.

```
d = a*c-b
```

```
d = 
  laurentPolynomial with properties:
    Coefficients: [4 12 19 28 5 6]
        MaxOrder: 4
```

Laurent Matrix Multiplication

Create two Laurent polynomials:

- $a(z) = z + 1$
- $b(z) = z^2 - z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[1 1],MaxOrder=1);
lpB = laurentPolynomial(Coefficients=[1 0 0 -1],MaxOrder=2);
```

Create two Laurent matrices:

- $\texttt{lmatA} = \begin{bmatrix} a(z) & 1 \\ 1 & 0 \end{bmatrix}$
- $\texttt{lmatB} = \begin{bmatrix} 0 & 2 \\ 3 & b(z) \end{bmatrix}$

```
lmatA = laurentMatrix(Elements={lpA,1;1,0});
lmatB = laurentMatrix(Elements={0,2;3,lpB});
```

Multiply the matrices.

```
lmat = lmatA*lmatB;
lmat.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 3
        MaxOrder: 0
```

```
lmat.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 2 2 -1]
        MaxOrder: 2
```

```
lmat.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 0
        MaxOrder: 0
```

```
lmat.Elements{2,2}

ans = 
  laurentPolynomial with properties:

    Coefficients: 2
        MaxOrder: 0
```

Input Arguments

A — Laurent polynomial or Laurent matrix

laurentPolynomial object | laurentMatrix object

Laurent polynomial or Laurent matrix, specified as a laurentPolynomial object or a laurentMatrix object, respectively.

B — Laurent polynomial or Laurent matrix

laurentPolynomial object | laurentMatrix object

Laurent polynomial or Laurent matrix, specified as a laurentPolynomial object or a laurentMatrix object, respectively.

Output Arguments

Q — Product

laurentPolynomial object | laurentMatrix object

Product of two Laurent polynomials or two Laurent matrices, returned as a laurentPolynomial object or a laurentMatrix object.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

plus | minus

Objects

laurentMatrix | laurentPolynomial

Introduced in R2021b

mpower

Laurent polynomial exponentiation

Syntax

```
Q = mpower(P,pow)
Q = P^pow
```

Description

`Q = mpower(P,pow)` raises the Laurent polynomial `P` to the power `pow`.

`Q = P^pow` is equivalent to `Q = mpower(p,pow)`.

Examples

Laurent Polynomial Exponentiation

Create two Laurent polynomials:

- $a(z) = z - 1$
- $b(z) = -2z^3 + 6z^2 - 6z + 2$

```
a = laurentPolynomial(Coefficients=[1 -1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[-2 6 -6 2],MaxOrder=3);
```

Raise $a(z)$ to the third power. Confirm the result is not equal to $b(z)$.

```
a3 = a^3;
a3 ~= b
```

```
ans = logical
   1
```

Confirm $a(z)$ raised to the third power is equal to $-b(z)/2$.

```
b2 = rescale(b,-1/2);
a3 == b2
```

```
ans = logical
   1
```

Input Arguments

P — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

pow — Power
integer

Power, specified as an integer. If `pow` is negative, `P` must be a monomial.

Example: `Q = mpower(lp,3)` raises the Laurent polynomial `lp` to the third power.

Data Types: `double`

Output Arguments

Q — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial raised to a nonzero power, returned as a `laurentPolynomial` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

ne

Laurent polynomials inequality test

Syntax

```
tf = ne(A,B)
tf = (A ~= B)
```

Description

`tf = ne(A,B)` compares Laurent polynomials `A` and `B` and returns `1` (`true`) if the two are unequal and `0` (`false`) otherwise.

`tf = (A ~= B)` is equivalent to `tf = ne(A,B)`.

Examples

Test Equality of Laurent Polynomials

Create two Laurent polynomials:

- $a(z) = 2z^3 - 3z^2 + 4z - 5$
- $b(z) = 4z^3 - 6z^2 + 8z$

```
a = laurentPolynomial(Coefficients=[2 -3 4 -5],MaxOrder=3);
b = laurentPolynomial(Coefficients=[4 -6 8],MaxOrder=3);
```

Confirm $a(z)$ and $b(z)$ are not equal.

```
a ~= b
```

```
ans = logical
   1
```

Confirm $2a(z) + 10$ and $b(z)$ are equal.

```
c = rescale(a,2)+10;
eq(c,b)
```

```
ans = logical
   1
```

Input Arguments

A — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

B — Laurent polynomial

`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

Output Arguments

tf — Inequality test result

`true` or `1` | `false` or `0`

Inequality test result, returned as a numeric or logical `1` (`true`) or `0` (`false`).

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`eq`

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

nodeasc

Node ascendants

Syntax

```
A = nodeasc(T,N)
```

Description

`nodeasc` is a tree-management utility.

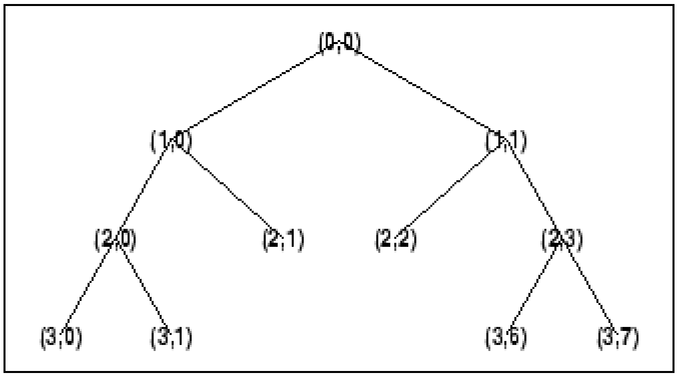
`A = nodeasc(T,N)` returns the indices of all the ascendants of the node *N* in the tree *T* where `N` can be the index node or the depth and position of the node. `A` is a column vector with `A`(1) = index of node *N*.

`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.

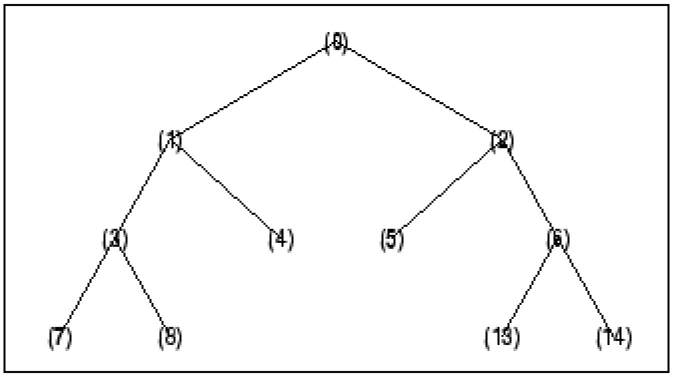
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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.

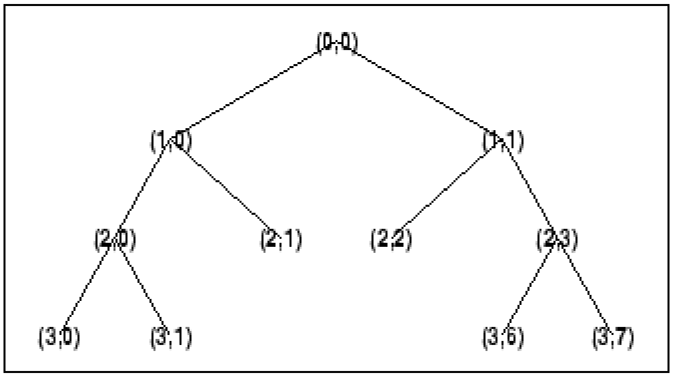
`D = nodedesc(T,N)` returns the indices of all the descendants of the node *N* in the tree *T* where *N* can be the index node or the depth and position of node. `D` is a column vector with `D(1)` = index of node *N*.

`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.

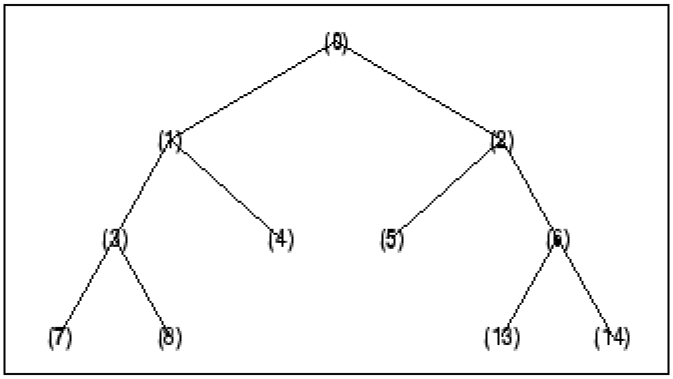
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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.

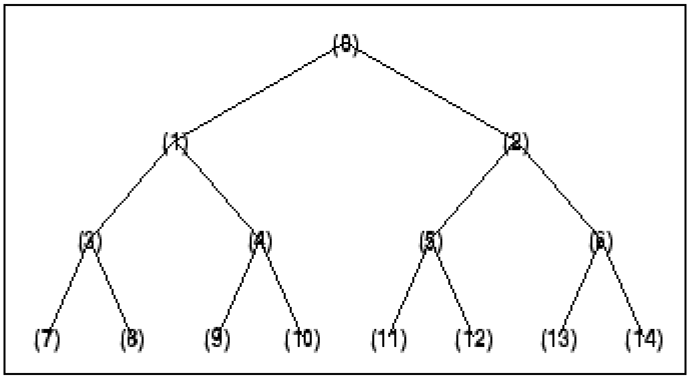
`T = nodejoin(T,N)` returns the modified tree *T* corresponding to a recomposition of the node *N*.

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

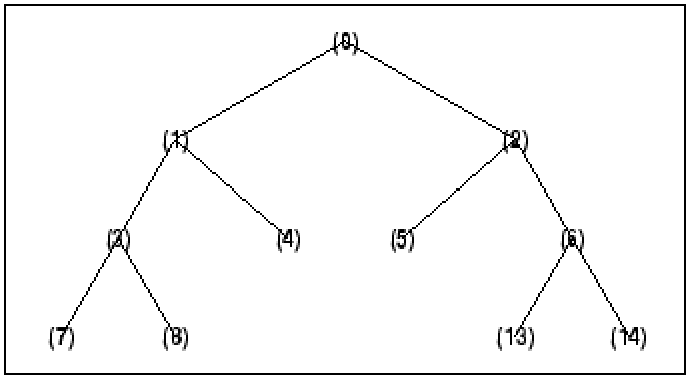
`T = nodejoin(T)` is equivalent to `T = nodejoin(T,0)`.

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.

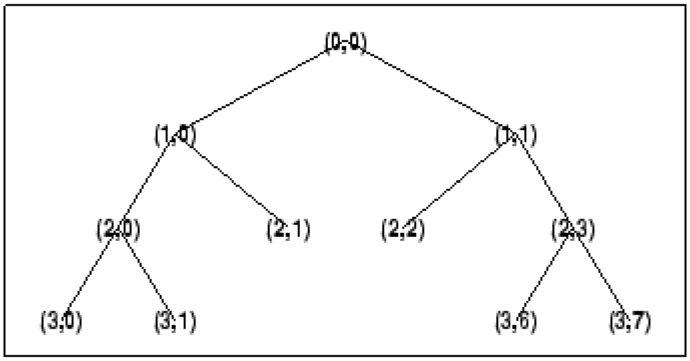
nodepar(T,0) or nodepar(T,[0,0]) returns -1.

nodepar(T,0,'deppos') or nodepar(T,[0,0],'deppos') returns [-1,0].

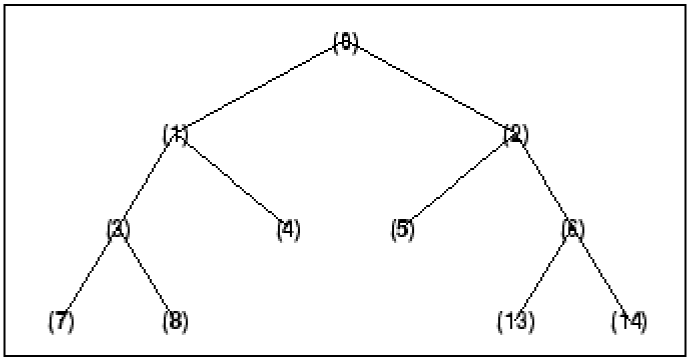
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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

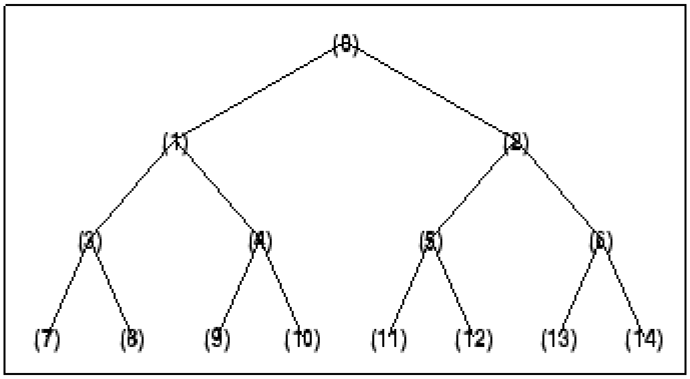
`nodesplt` is a tree-management utility.

`T = nodesplt(T,N)` returns the modified tree *T* corresponding to the decomposition of the node *N*.

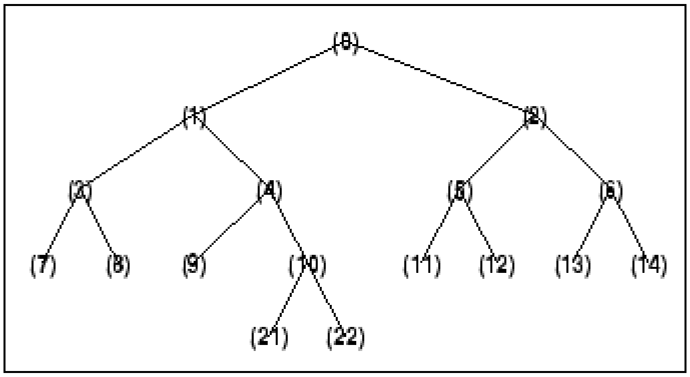
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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.

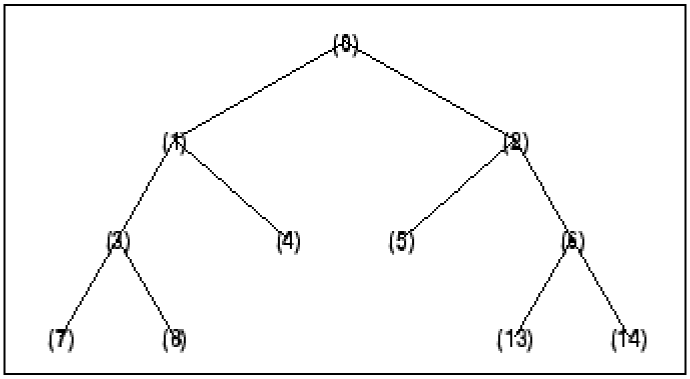
The nodes are ordered from left to right as in tree *T*.

`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

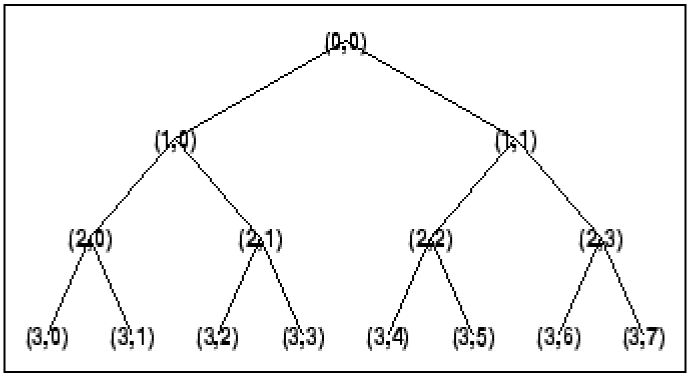
`ntnode` is a tree-management utility.

`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`.

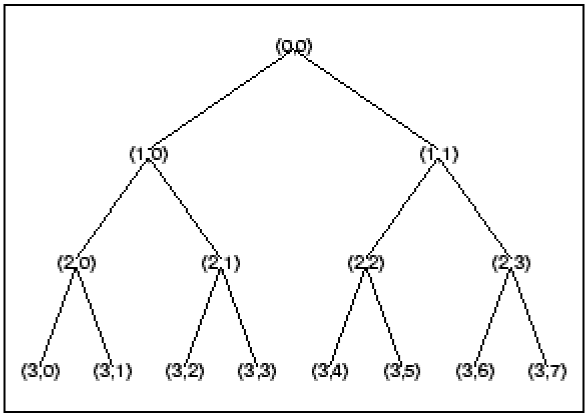
Class NTREE (Parent class: WTBO)

Fields

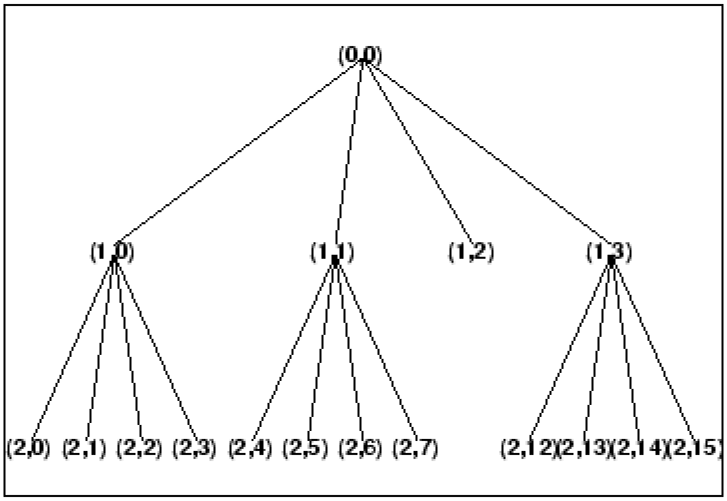
`wtbo`	Parent object
`order`	Tree order
`depth`	Tree depth
`spsch`	Split scheme for nodes
`tn`	Column vector with terminal node indices

Examples

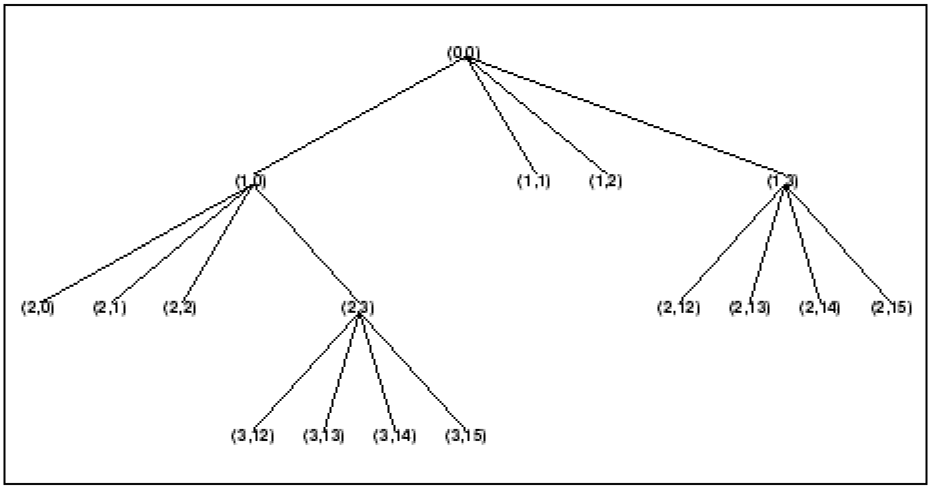
```
% Create binary tree (tree of order 2) of depth 3.
t2 = ntree(2,3);

% 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

numCoefficients

Number of wavelet scattering coefficients

Syntax

```
ncf = numCoefficients(sf)
```

Description

`ncf = numCoefficients(sf)` returns the number of scattering coefficients for each scattering path in the wavelet time scattering network `sf`. The number of scattering coefficients depends on the values of the “SignalLength” on page 1-0 , “InvarianceScale” on page 1-0 , and “OversamplingFactor” on page 1-0 properties of `sf`.

Examples

Oversample 1-D Wavelet Scattering Transform

This example shows how to oversample a 1-D wavelet scattering transform.

Load an ECG signal sampled at 180 Hz, and create a wavelet time scattering network to process the signal. To perform a critically downsampled wavelet scattering transform, do not change the value of the `OversamplingFactor` property in `sf`. Return the number of scattering coefficients for the scattering network.

```
load wecg
Fs = 180;
sf = waveletScattering('SignalLength',numel(wecg),'SamplingFrequency',Fs);
ncf = numCoefficients(sf)
```

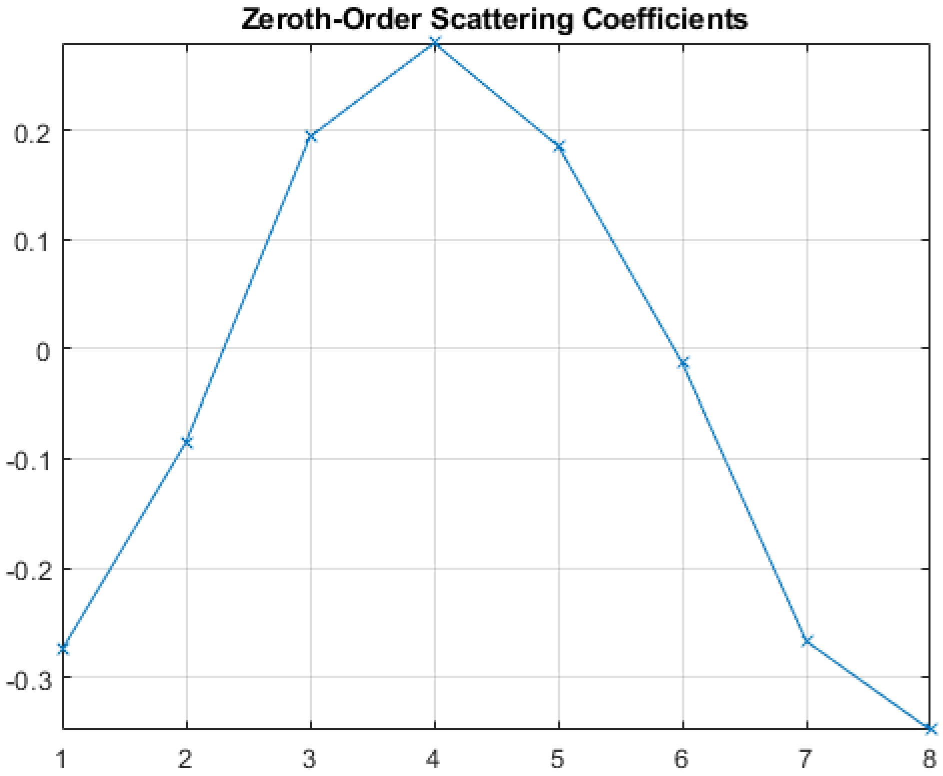
```
ncf = 8
```

Return the 1-D wavelet scattering transform of `wecg`, and plot the zeroth-order scattering coefficients. Confirm the number of zeroth-order scattering coefficients is equal to `ncf`.

```
s = scatteringTransform(sf,wecg);
display(['Number of zeroth-order scattering coefficients: ',...
  num2str(numel(s{1}.signals{1}))])
```

```
Number of zeroth-order scattering coefficients: 8
```

```
plot(s{1}.signals{1},'x-')
grid on
axis tight
title('Zeroth-Order Scattering Coefficients')
```



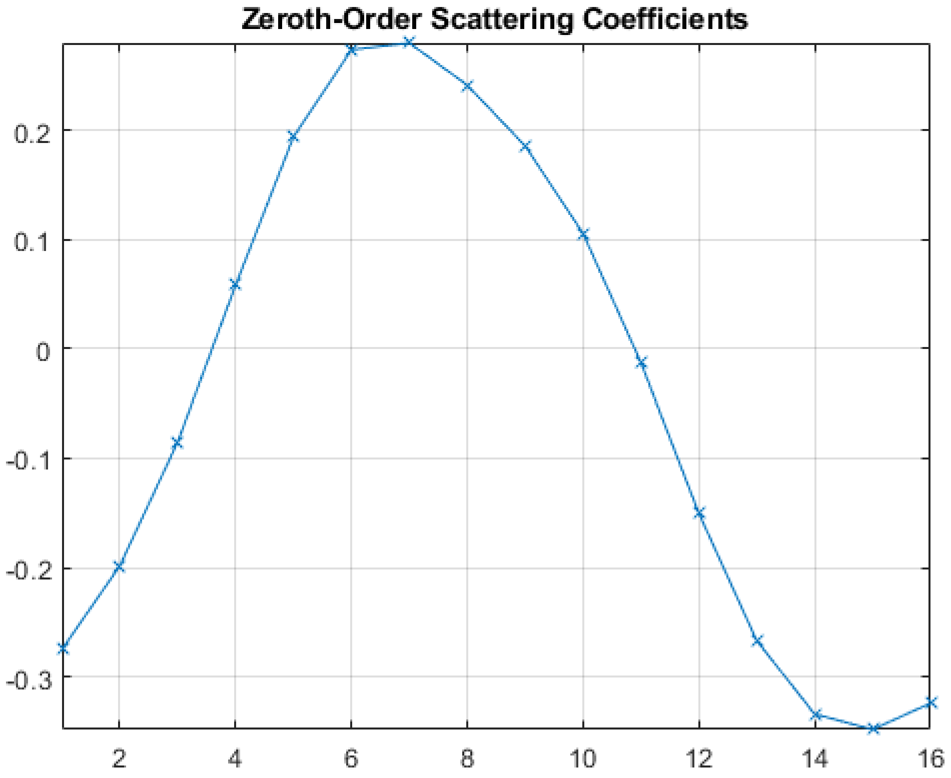
To oversample the scattering coefficients by a factor of 2, set the `OversamplingFactor` property of `sf` equal to 1 (because $\log_2 2 = 1$). Return the number of scattering coefficients for the edited network. Confirm the number of scattering coefficients has doubled.

```
sf.OversamplingFactor = 1;
ncf = numCoefficients(sf)
```

```
ncf = 16
```

Return the wavelet scattering transform of `wecg` using the edited network, and plot the zeroth-order scattering coefficients. Since the number of coefficients in the critically sampled transform is equal to 8, confirm that the number of zeroth-order coefficients in the oversampled transform is equal to 16.

```
s = scatteringTransform(sf,wecg);
figure
plot(s{1}.signals{1},'x-')
grid on
axis tight
title('Zeroth-Order Scattering Coefficients')
```



Input Arguments

sf — Wavelet time scattering network
waveletScattering object

Wavelet time scattering network, specified as a `waveletScattering` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`waveletScattering`

Introduced in R2019a

numfilterbanks

Number of scattering filter banks

Syntax

```
nfb = numfilterbanks(sf)
```

Description

`nfb = numfilterbanks(sf)` returns the number of filter banks in the wavelet time scattering network, `sf`. The number of filter banks in a scattering network is equal to *ord* − 1 where *ord* is the number of scattering orders.

Examples

Number of Filter Banks in Wavelet Scattering Network

Calculate the number of filter banks for the default wavelet scattering network.

```
sf = waveletScattering
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 512
        QualityFactors: [8 1]
              Boundary: 'periodic'
     SamplingFrequency: 1
             Precision: 'double'
    OversamplingFactor: 0
          OptimizePath: 0
```

```
Nfb = numfilterbanks(sf)
```

```
Nfb = 2
```

Input Arguments

`sf` — Wavelet time scattering network
`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`waveletScattering` | `waveletScattering2`

Introduced in R2018b

numfilterbanks

Number of scattering filter banks

Syntax

```
nfb = numfilterbanks(sf)
```

Description

`nfb = numfilterbanks(sf)` returns the number of filter banks in the wavelet image scattering network, `sf`. The number of filter banks in the network is equal to $ord - 1$ where *ord* is the number of scattering orders.

Examples

Number of Filter Banks in 2-D Wavelet Scattering Network

Calculate the number of filter banks for the default 2-D wavelet scattering network.

```
sf = waveletScattering2
```

```
sf = 
  waveletScattering2 with properties:

             ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [6 6]
        QualityFactors: [1 1]
             Precision: 'single'
    OversamplingFactor: 0
          OptimizePath: 1
```

```
Nfb = numfilterbanks(sf)
```

```
Nfb = 2
```

Input Arguments

`sf` — Wavelet image scattering network
`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

See Also

`waveletScattering` | `waveletScattering2`

Introduced in R2018b

numorders

Number of scattering orders

Syntax

```
no = numorders(sf)
```

Description

`no = numorders(sf)` returns the number of orders for the wavelet time scattering network, `sf`. The number of orders is equal to *Nfb* + 1, where *Nfb* is the number of filter banks in `sf`.

Examples

Number of Orders in Wavelet Time Scattering Network

Calculate the number of orders for the default wavelet time scattering network.

```
sf = waveletScattering
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 512
        QualityFactors: [8 1]
              Boundary: 'periodic'
     SamplingFrequency: 1
             Precision: 'double'
    OversamplingFactor: 0
          OptimizePath: 0
```

```
no = numorders(sf)
```

```
no = 3
```

Input Arguments

`sf` — Wavelet time scattering network
`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`waveletScattering` | `waveletScattering2`

Introduced in R2018b

numorders

Number of scattering orders

Syntax

```
no = numorders(sf)
```

Description

`no = numorders(sf)` returns the number of orders for the wavelet image scattering network, `sf`. The number of orders is equal to *Nfb* + 1, where *Nfb* is the number of filter banks in `sf`.

Examples

Number of Orders in Wavelet Image Scattering Network

Calculate the number of orders for the default wavelet image scattering network.

```
sf = waveletScattering2
```

```
sf = 
  waveletScattering2 with properties:

             ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [6 6]
        QualityFactors: [1 1]
             Precision: 'single'
    OversamplingFactor: 0
          OptimizePath: 1
```

```
no = numorders(sf)
```

```
no = 3
```

Input Arguments

`sf` — Wavelet image scattering network
`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

See Also

`waveletScattering` | `waveletScattering2`

Introduced in R2018b

numshears

Number of shearlets

Syntax

```
NS = numshears(sls)
```

Description

`NS = numshears(sls)` returns the number of shearlets in the shearlet system `sls`. The number of shearlets does not include the lowpass filter, which is not sheared. The total filter size of the shearlet system is *M*-by-*N*-by-`NS`+1. *M* and *N* are the first and second elements, respectively, of the ImageSize value of `sls`.

The data type of `NS` matches the Precision value of the shearlet system.

Examples

Number of Shearlets in Shearlet System

Create a complex-valued shearlet system that can be applied to 256-by-256 images. The system has four scales.

```
sls = shearletSystem('ImageSize',[256 256],'TransformType','complex',...
    'NumScales',4);
```

Obtain the number of shearlets in the shearlet system.

```
num = numshears(sls)
```

```
num = 80
```

Input Arguments

`sls` — Shearlet system
`shearletSystem` object

Shearlet system, specified as a `shearletSystem` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`shearletSystem` | `filterbank`

Introduced in R2019b

orthfilt

Orthogonal wavelet filters

Syntax

```
[LoD,HiD,LoR,HiR] = orthfilt(W)
```

Description

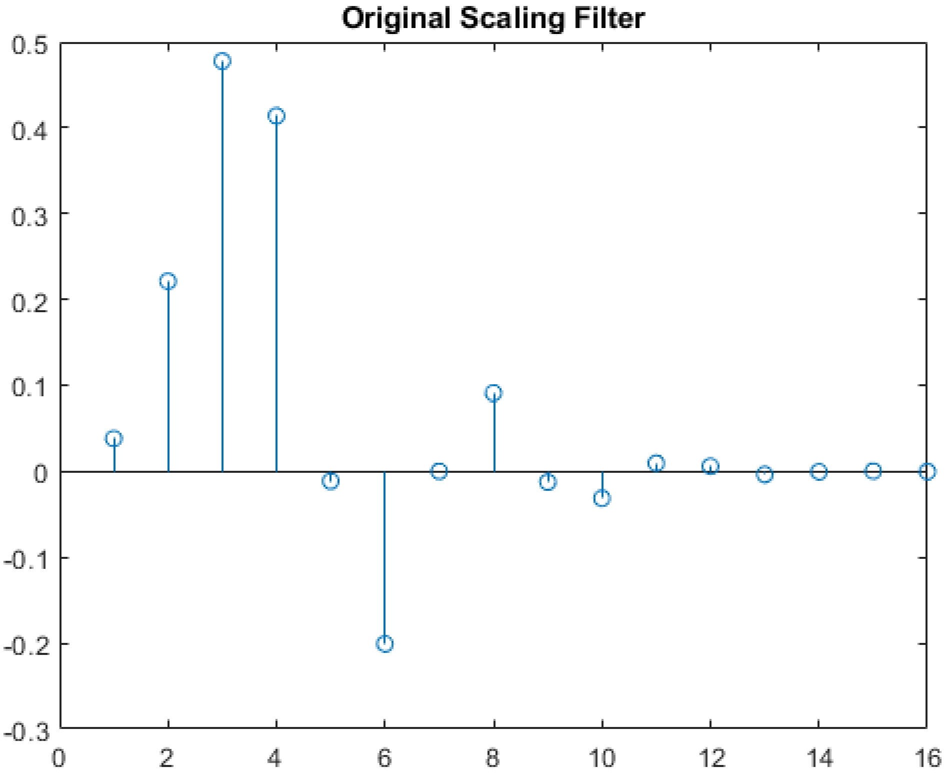
`[LoD,HiD,LoR,HiR] = orthfilt(W)` computes the four lowpass and highpass, decomposition and reconstruction filters associated with the scaling filter `W` corresponding to a wavelet.

Examples

Compute Orthogonal Wavelet Filters of Scaling Filter

Create a scaling filter associated with the Daubechies `db8` wavelet.

```
W = dbwavf("db8");
stem(W)
title("Original Scaling Filter")
```

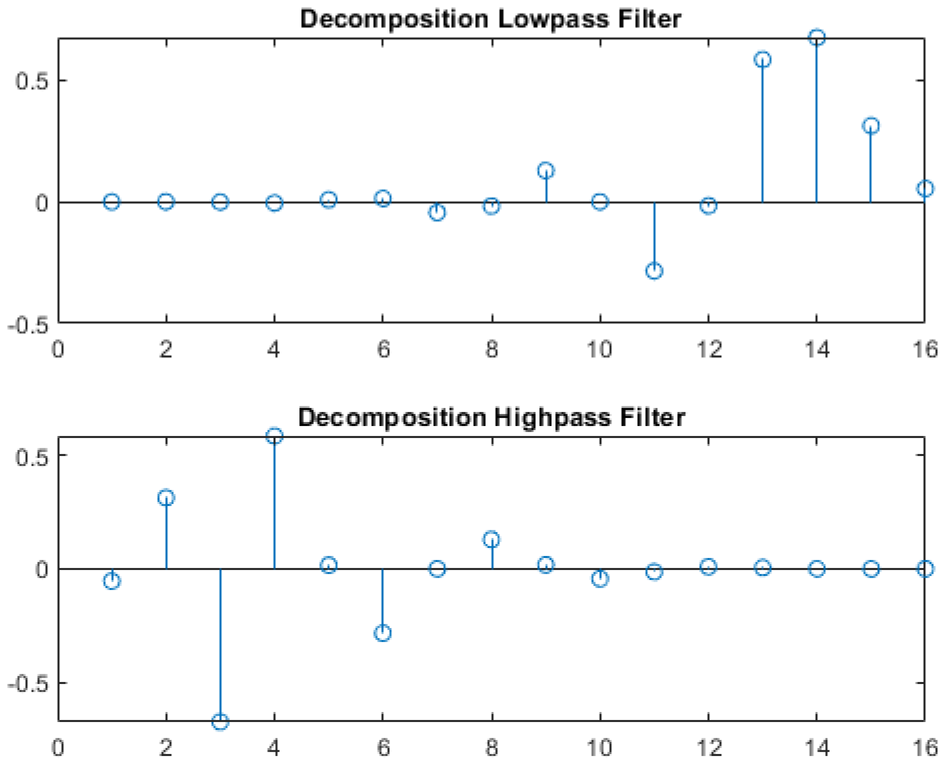


Compute the four filters associated with the scaling filter.

```
[LoD,HiD,LoR,HiR] = orthfilt(W);
```

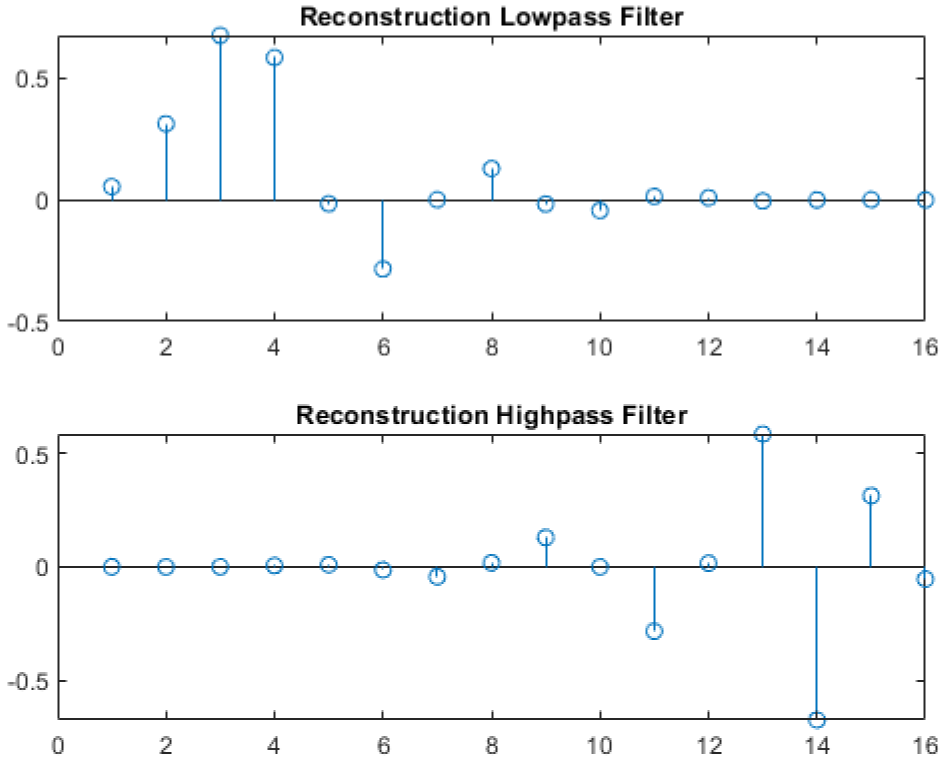
Plot the decomposition lowpass and highpass filters.

```
subplot(2,1,1)
stem(LoD)
title("Decomposition Lowpass Filter")
subplot(2,1,2)
stem(HiD)
title("Decomposition Highpass Filter")
```



Plot the reconstruction lowpass and highpass filters.

```
subplot(2,1,1)
stem(LoR)
title("Reconstruction Lowpass Filter")
subplot(2,1,2)
stem(HiR)
title("Reconstruction Highpass Filter")
```



Check for orthonormality in the decomposition filters.

```
df = [LoD;HiD];
rf = [LoR;HiR];
id = df*df'
```

```
id = 2×2

    1.0000         0
         0    1.0000
```

Check for orthonormality in the reconstruction filters.

```
id2 = rf*rf'
```

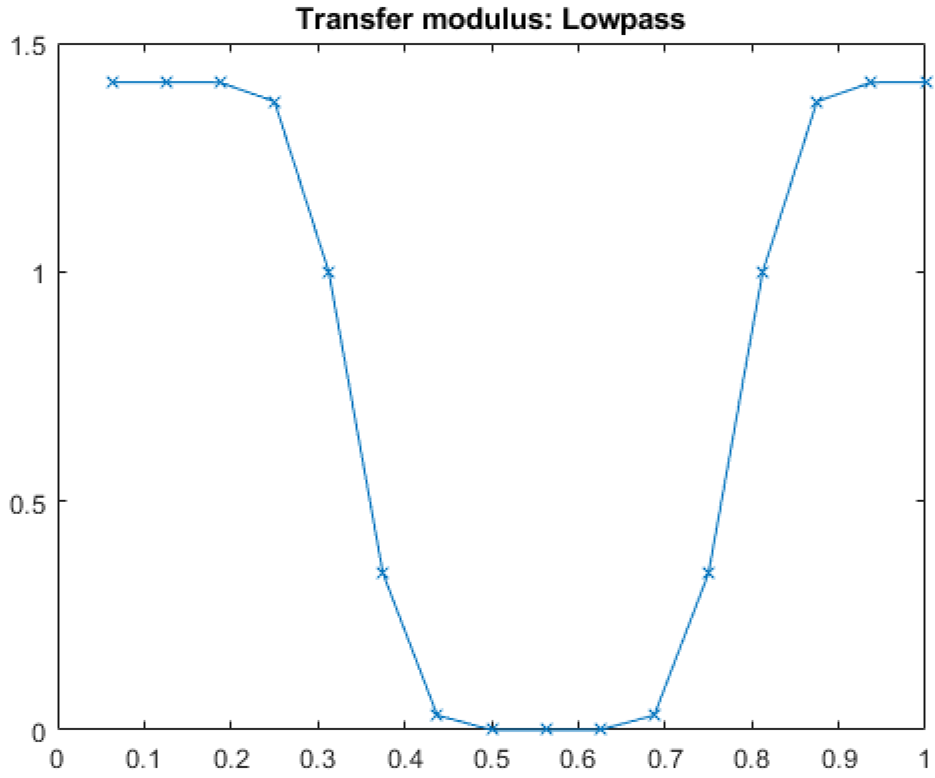
```
id2 = 2×2

    1.0000         0
         0    1.0000
```

Check for orthogonality by dyadic translation.

```
df = [LoD 0 0;HiD 0 0];
dft = [0 0 LoD; 0 0 HiD];
zer = df*dft'
zer = 2×2
10^-12 ×

   -0.1895   -0.0000
         0   -0.1895
```

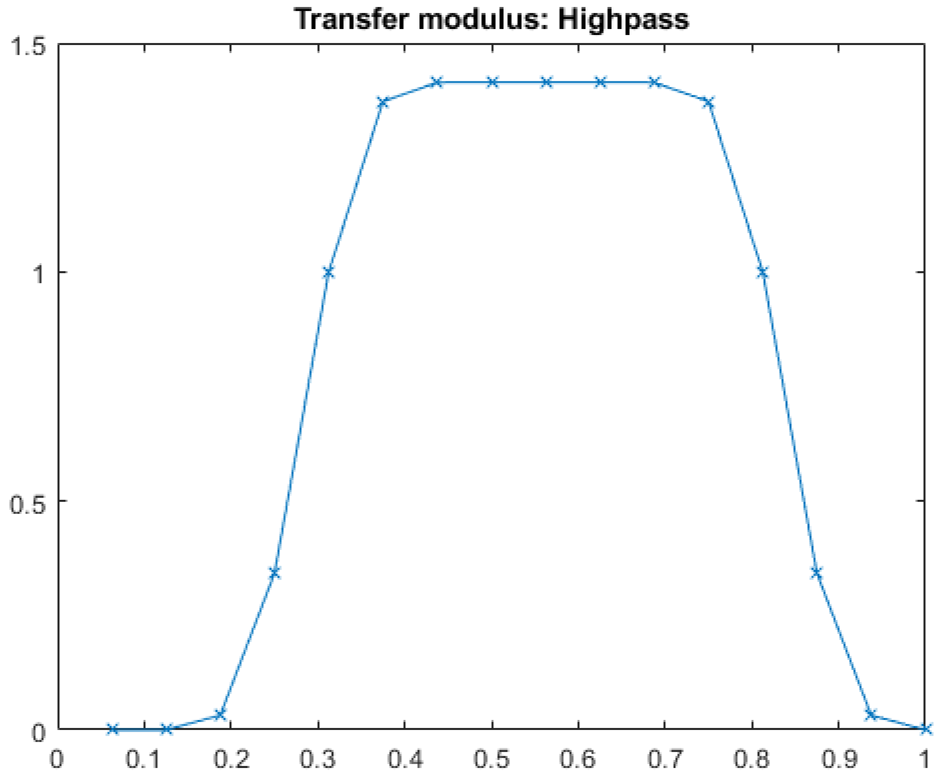
Plot the low-frequency transfer modulus.

```
fftld = fft(LoD);
freq = [1:length(LoD)]/length(LoD);
figure
plot(freq,abs(fftld),"x-")
title("Transfer modulus: Lowpass")
```



Plot the high-frequency transfer modulus.

```
ffthd = fft(HiD);
plot(freq,abs(ffthd),"x-")
title("Transfer modulus: Highpass")
```



Input Arguments

`W` — Scaling filter
real-valued vector

Scaling filter corresponding to a wavelet, specified as a real-valued vector.

Output Arguments

`LoD` — Decomposition lowpass filter
real-valued vector

Decomposition lowpass filter associated with the scaling filter `W`, returned as a real-valued vector.

`HiD` — Decomposition highpass filter
real-valued vector

Decomposition highpass filter associated with the scaling filter `W`, returned as a real-valued vector.

`LoR` — Reconstruction lowpass filter
real-valued vector

Reconstruction lowpass filter associated with the scaling filter `W`, returned as a real-valued vector.

HiR — Reconstruction highpass filter
real-valued vector

Reconstruction highpass filter associated with the scaling filter `W`, returned as a real-valued vector.

Algorithms

For an orthogonal wavelet in the multiresolution framework, 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 the `dwt` and `idwt` functions are intimately related to the sequence $(w_n)_{n \in Z}$. If ϕ is compactly supported, the sequence (w_n) is finite and can be viewed as an FIR filter. The scaling filter `W` is a lowpass FIR filter of length 2N, with the sum 1, and with the norm of 1/√2.

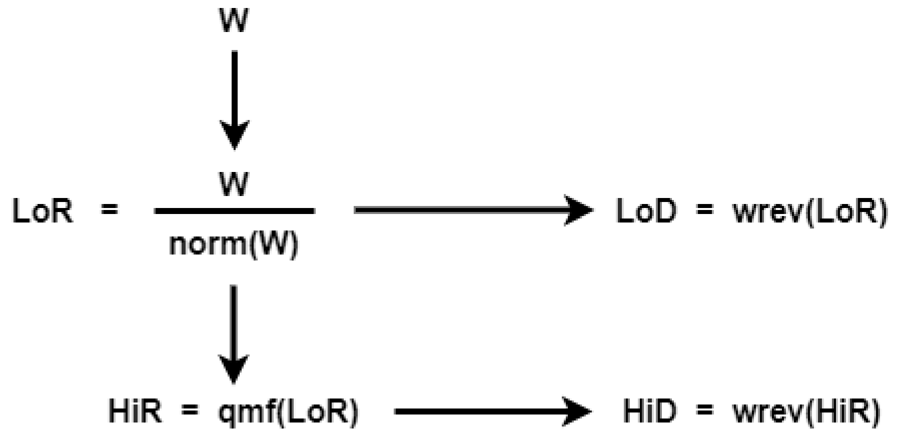
For example, for a `db3` scaling filter,

```
w = dbwavf("db3")
w = 0.2352 0.5706 0.3252 -0.0955 -0.0604 0.0249

sum(w)
 = 1.000
norm(w)
 = 0.7071
```

Define four FIR filters from filter `W` of length 2N and norm 1.

Filters	Low-Pass	High-Pass
Decomposition	`LoD`	`HiD`
Reconstruction	`LoR`	`HiR`

The function computes the four filters using the following scheme.



`HiR` and `LoR` are quadrature mirror filters: `HiR(k) = (-1)`^{k}`LoR(2N + 1 - k)`, for `k = 1, 2, … , 2N`. Because `wrev` reverses vectors, `HiD` and `LoD` are also quadrature mirror filters.

References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

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 = 4×1

     3
     4
     5
     6


Tn_Seq = 4×1

     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)
```

```
ans = logical
   1
```

```
isequal(Tn_Seq,Tn_Pal(I))
```

```
ans = logical
   1
```

Order Terminal Nodes by Depth and Position

Order terminal nodes by depth and position.

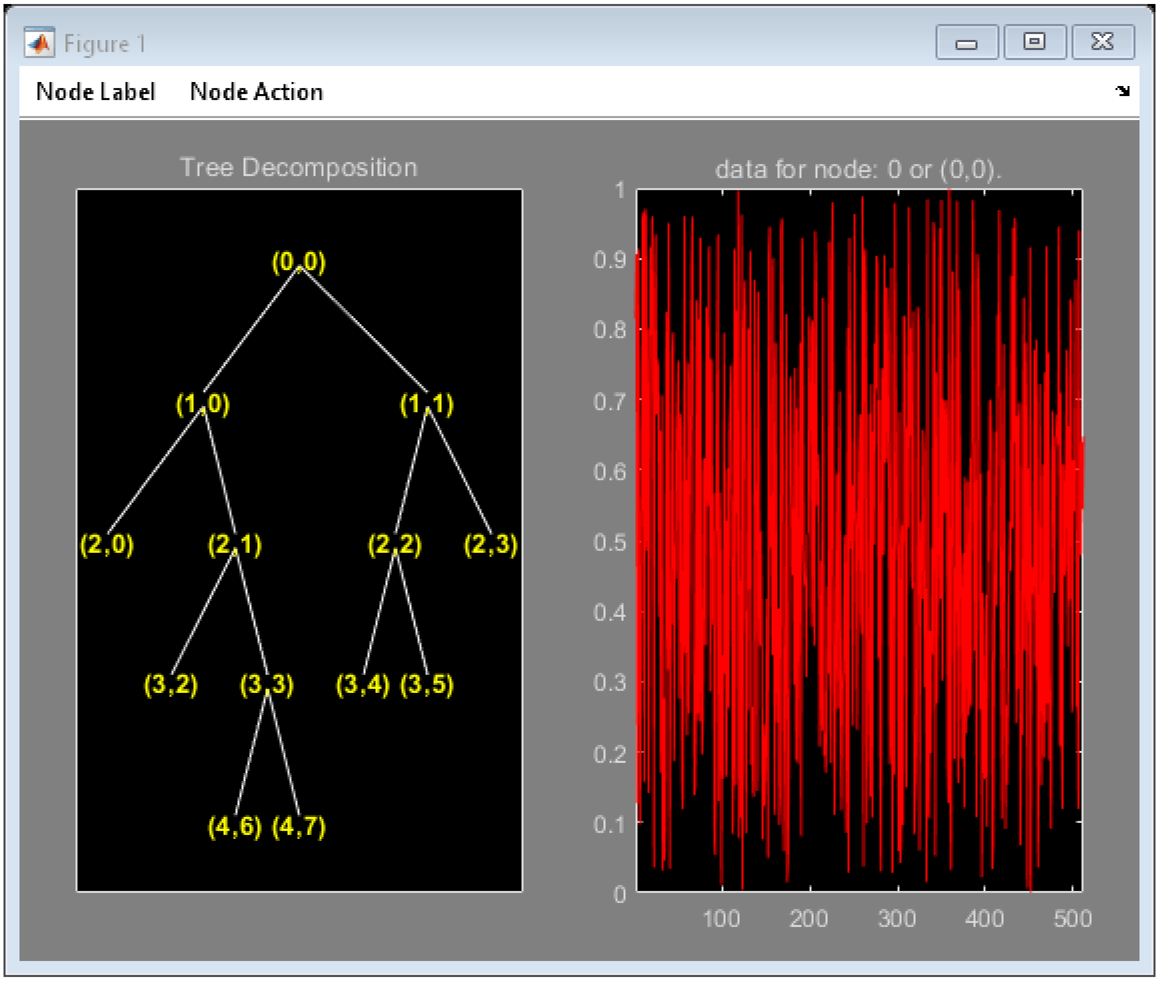
```
x = randn(8,1);
wpt = wpdec(x,2,'haar');
[DP_Pal,DP_Seq] = otnodes(wpt,'dp')
```

```
DP_Pal = 4×2

     2     0
     2     1
     2     2
     2     3


DP_Seq = 4×2

     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)
```

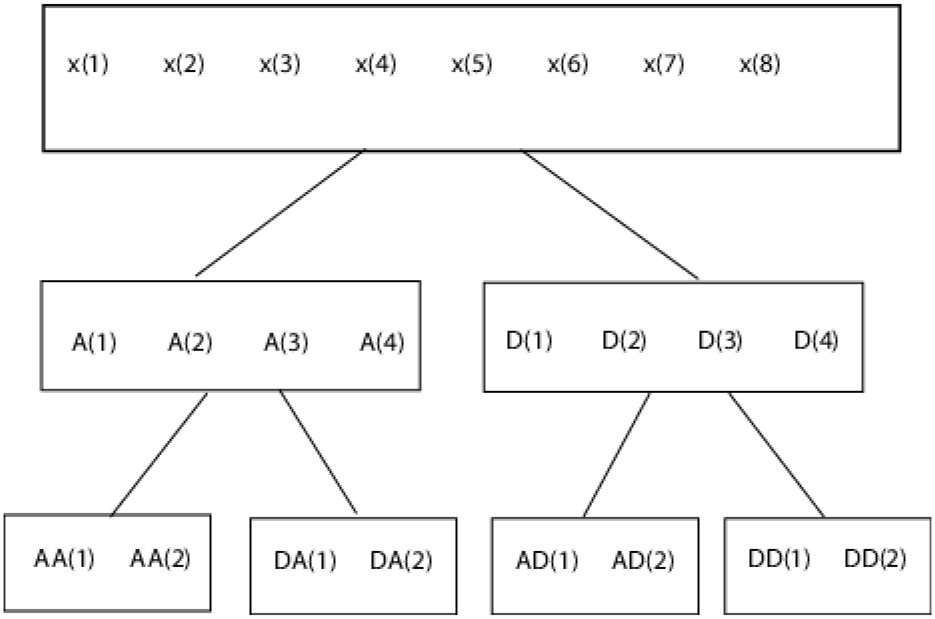
```
tn_Pal = 7×1

     3
     9
    21
    22
    11
    12
     6


tn_Seq = 7×1

     3
    21
    22
     9
     6
    12
    11
```

```
I = 7×1

     1
     3
     4
     2
     7
     6
     5


J = 7×1

     1
     4
     2
     3
     7
     6
     5
```

More About

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)` returns an admissible wavelet `psi` for the continuous wavelet transform (CWT) adapted to the pattern `ypat`. The wavelet `psi` is evaluated at `xval`, a regular grid in the interval [0,1], and has L_2-norm equal to 1.

The constant `nc` is such that `nc×psi` approximates `ypat` on the interval [0,1] by least-squares fitting using the method specified by `method`, and a polynomial of degree `poldegree` with boundary constraints specified by `regularity`.

Examples

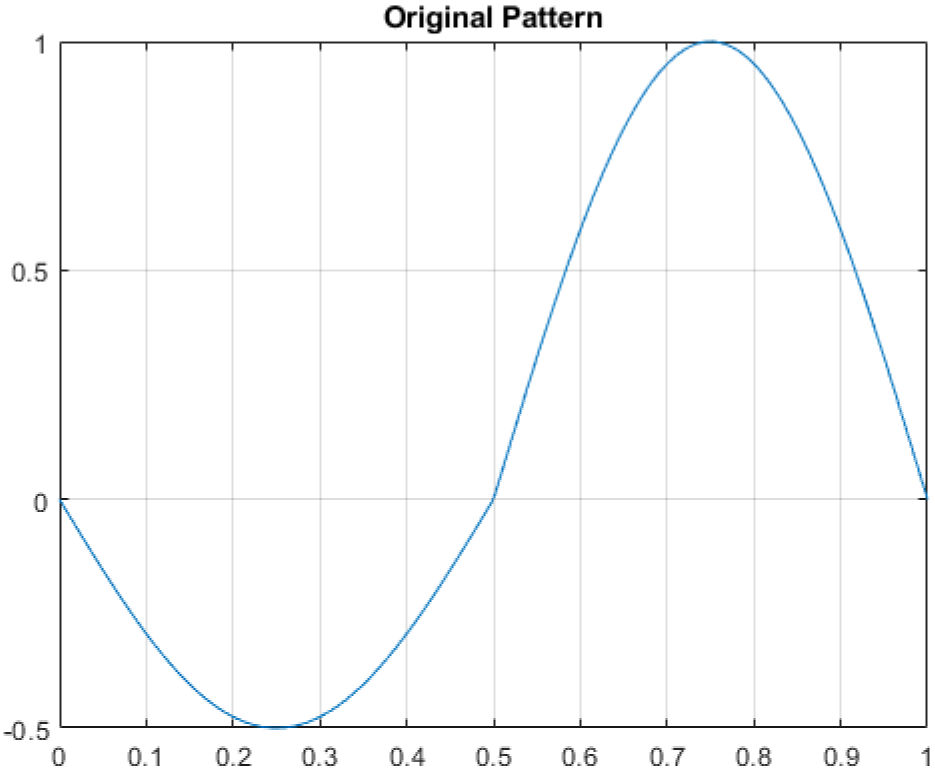
Create Wavelet for Continuous Wavelet Transform

This example illustrates how to generate a new wavelet starting from a pattern.

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 [1].

Load and plot a pattern.

```
load ptpssin1
plot(X,Y)
grid on
title('Original Pattern')
```



Integrate the pattern over the interval. The integral does not equal 0. However, the pattern is a good candidate since it oscillates like a wavelet.

```
dX = X(2)-X(1);
patternInt = dX*sum(Y);
disp(['Integral: ',num2str(patternInt)]);
```

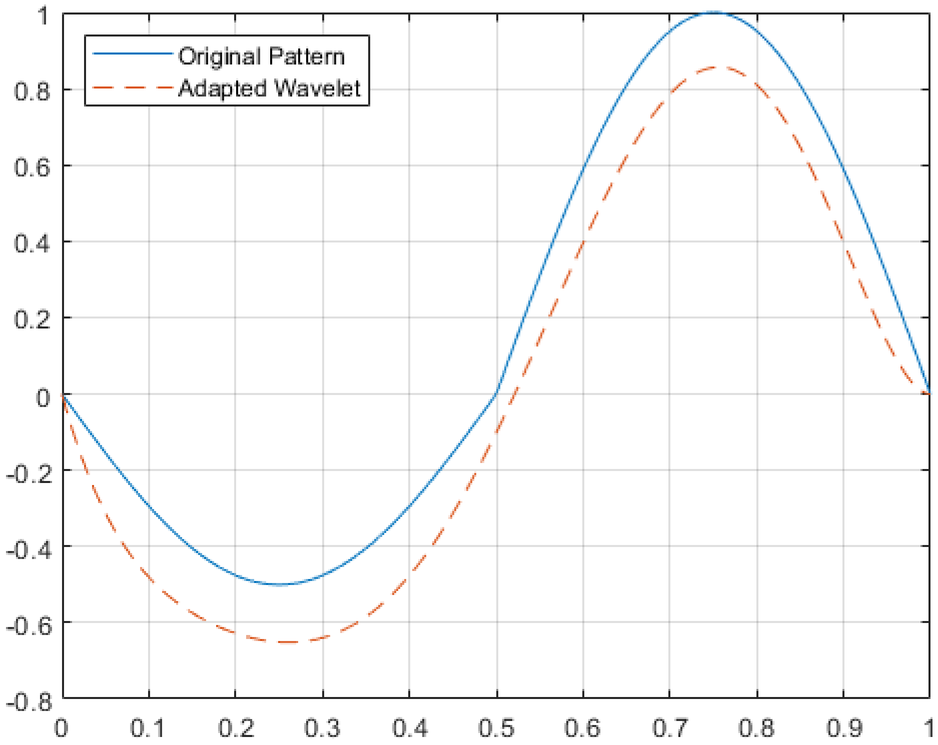
```
Integral: 0.15915
```

To synthesize a new wavelet adapted to the given pattern, use a least-squares polynomial approximation of degree 6 with constraints of continuity at the beginning and the end of the pattern.

```
[psi,xval,nc] = pat2cwav(Y,'polynomial',6,'continuous');
```

Plot the new wavelet.

```
plot(X,Y,'-',xval,nc*psi,'--')
grid on
legend('Original Pattern','Adapted Wavelet','Location','NorthWest')
```



Check that `psi` satisfies the definition of a wavelet by confirming that it integrates to zero and has L_2 norm is equal to 1.

```
dxval = xval(2)-xval(1);
psiIntegral = dxval*sum(psi);
disp(['Integral: ',num2str(psiIntegral)])
```

```
Integral: 1.9626e-05
```

```
psiSqN = dxval*sum(psi.^2);
disp(['L2-norm: ',num2str(psiSqN)])
```

```
L2-norm: 1
```

Input Arguments

ypat — Pattern

real-valued vector

Pattern to approximate, specified as a real-valued vector.

`method` — Least-squares fitting method

`'polynomial'` | `'orthconst'`

Least-squares fitting method to use to approximate the pattern, specified as one of the following:

- 'polynomial' — Use a polynomial of degree poldegree
- 'orthconst' — Use a projection onto the space of functions orthogonal to constants

Note Specifying the 'orthconst' option does not produce an orthogonal wavelet. Any wavelet psi produced using pat2cwav is a type 4 wavelet (wavelet without a scaling function) in wavemngr.

poldegree — Degree of polynomial

integer

Degree of polynomial to use in least-squares fitting, specified as an integer.

regularity — Boundary constraints

'continuous' | 'differentiable' | 'none'

Boundary constraints at the points 0 and 1, specified as 'continuous', 'differentiable', or '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.

Output Arguments

psi — Admissible wavelet

real-valued vector

Admissible wavelet for CWT, returned as a real-valued vector. The length of psi equals the length of ypat. The wavelet psi integrates to zero and has L_2-norm equal to 1.

xval — Sampling instants

real-valued vector

Sampling instants where psi is evaluated, returned as a real-valued vector. The sampling instants xval are a regular *n*-point grid spanning the interval [0,1], where *n* is the length of ypat: xval = linspace(0,1,length(ypat)).

nc — Normalizing constant

scalar

Normalizing constant, returned as a scalar. The constant nc is such that nc×psi approximates ypat on the interval [0,1] by least-squares fitting using the method specified by method.

References

[1] Misiti, M., Y. Misiti, G. Oppenheim, and J.-M. Poggi. *Les ondelettes et leurs applications*. France: Hermes Science/Lavoisier, 2003.

See Also

wavemngr

Introduced before R2006a

paths

Scattering network paths

Syntax

```
spaths = paths(sf)
[spaths,npaths] = paths(sf)
```

Description

`spaths = paths(sf)` returns the scattering paths for the scattering network, `sf`. `spaths` is a *NO*-by-1 cell array of MATLAB tables, where *NO* is the number of orders in the network.

`[spaths,npaths] = paths(sf)` returns the number of wavelet scattering paths by order. `npaths` is a *NO*-by-1 vector, where *NO* is the number of orders in the scattering network. The *i*th element of `npaths` contains the number of scattering paths in the (*i*-1)th order.

Examples

Wavelet Scattering Paths

Create two wavelet scattering networks, both for a signal of length 500. In the second network, set the `OptimizePath` value to `true`.

```
sf = waveletScattering('SignalLength',500);
sfOpt = waveletScattering('SignalLength',500,'OptimizePath',true);
```

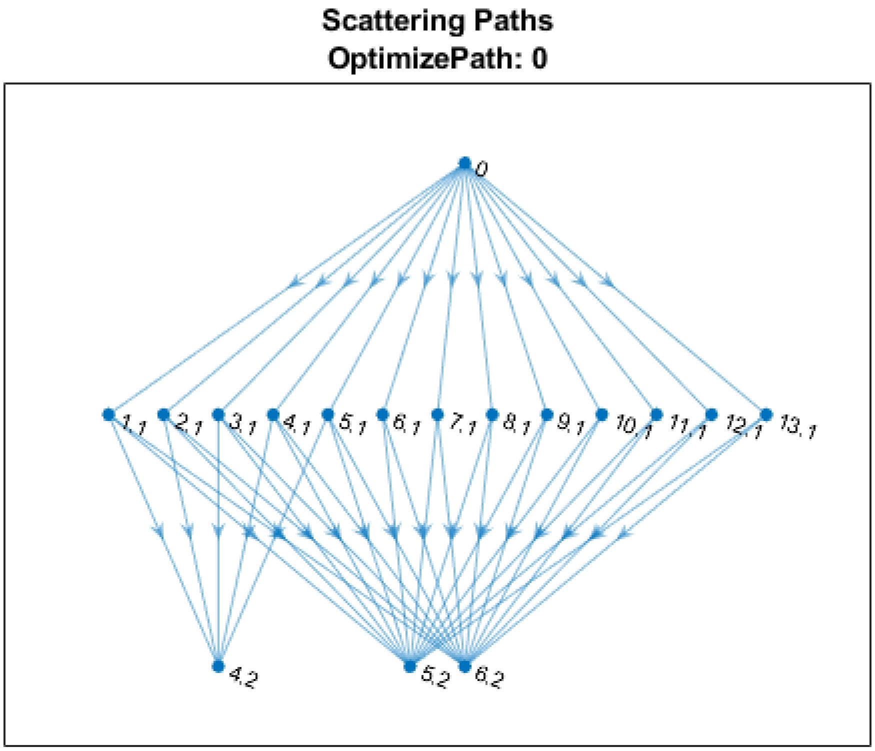
Obtain the path information of both networks. Determine the total number of scattering paths in both networks.

```
[spaths,npaths] = paths(sf);
[spathsOpt,npathsOpt] = paths(sfOpt);
str = sprintf('Paths in default network: %d\nPaths in path-optimized network: %d\n',...
    sum(npaths),sum(npathsOpt));
fprintf(str)
```

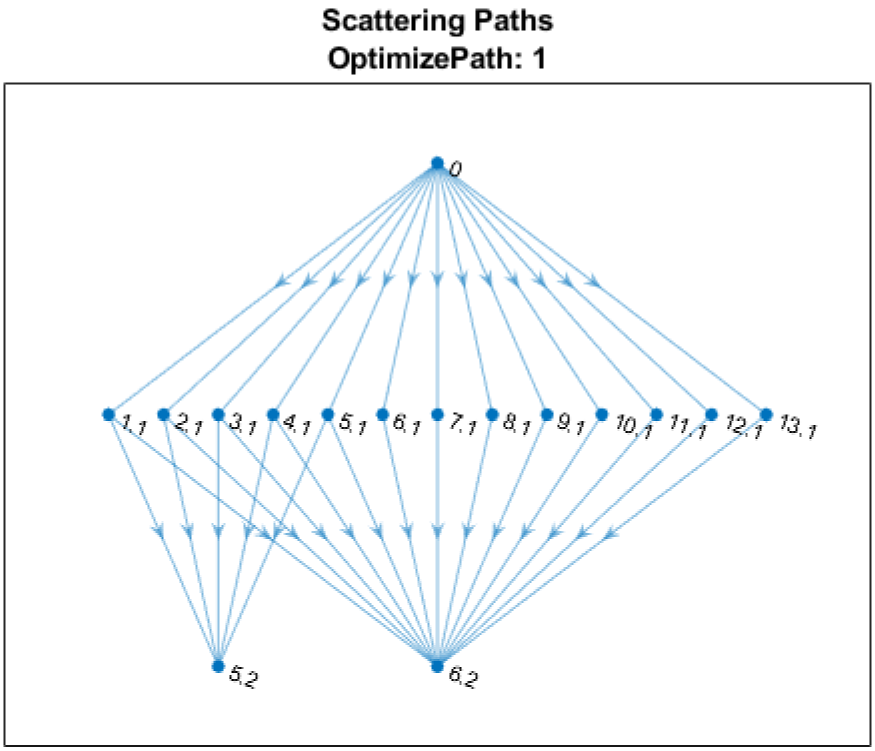
```
Paths in default network: 65
Paths in path-optimized network: 52
```

Both networks have two filter banks. Visualize the scattering paths that include the wavelets in the second filter bank. Create a directed graph. For every wavelet filter that is on at least one path, label the corresponding node as *waveletNumber.filterbank*. For each path, connect the corresponding nodes. Use the helper function `helperPlotScatteringGraph` to construct the graphs. Plot the graphs of both networks.

```
scatGraph = helperPlotScatteringGraph(spaths);
plot(scatGraph)
title({'Scattering Paths',['OptimizePath: ',num2str(sf.OptimizePath)]})
```



```
figure
scatGraphOpt = helperPlotScatteringGraph(spathsOpt);
plot(scatGraphOpt)
title({'Scattering Paths',['OptimizePath: ',num2str(sfOpt.OptimizePath)]})
```



Supporting Functions

plotScatteringGraph

```
function dirGraph = helperPlotScatteringGraph(networkPaths)
% This function is intended for use only in this example. It may change or
% be removed in a future release.

path = networkPaths{3}.path;
% set to 0 if want to show the multiple paths between 0 and each
% first level node
mkunique = 1;

if mkunique == 1
    f1 = path(:,1:2);
    c = unique(f1,'rows');
else
    c = path(:,1:2);
end

p1 = string(c(:,1));
p2 = string(c(:,2)+.1);
p3 = string(path(:,2)+.1);
p4 = string(path(:,3)+.2);
dirGraph = digraph([p1;p3],[p2;p4]);
```

```
end
```

Input Arguments

sf — Wavelet time scattering network

waveletScattering object

Wavelet time scattering network, specified as a `waveletScattering` object.

Output Arguments

spaths — Scattering paths

cell array

Scattering paths, returned as a *NO*-by-1 cell array of MATLAB tables, where *NO* is the number of orders of the scattering network.

Each MATLAB table in `spaths` contains three variables:

- `path` — Scattering network paths. In the *k*th element of `spaths`, `path` is a *N*-by-*k* matrix where each row contains a path from the input data through the (*k*-1)th wavelet filter bank. For example, when *k* equals 1, *N* is equal to 1 and the only path is 0 denoting the input data. When k equals 2, *N* is equal to the number of wavelet filters in the first filter bank and `path` is a *N*-by-2 matrix describing the path from the input data, 0, through the wavelet filters in the first filter bank. The second column of `path` contains the wavelet filters in the first filter bank ordered by decreasing center frequency.
- `log2ds` — The incremental base-2 log downsampling factor for the scalogram coefficients corresponding to the cumulative path in the same row.
- `log2res` — The base-2 log resolution of the scalogram coefficients corresponding to the cumulative path in the same row.

npaths — Number of wavelet scattering paths

vector

Number of wavelet scattering paths in the network by order, returned as a vector. `npaths` is a *NO*-by-1 vector where *NO* is the number of orders in the network. The *i*th element of `npaths` contains the number of scattering paths in the (*i*-1)th order. The sum of the elements of `npaths` is the total number of scattering paths.

Extended Capabilities

C/C++ Code Generation

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

See Also

`waveletScattering` | `filterbank`

Introduced in R2021a

paths

Scattering paths

Syntax

```
spaths = paths(sf)
[spaths,npaths] = paths(sf)
```

Description

`spaths = paths(sf)` returns the scattering paths for all orders of the scattering network, `sf`. `spaths` is a cell array of MATLAB tables with *n* elements, where *n* is the number of orders in the scattering network.

`[spaths,npaths] = paths(sf)` returns the number of paths in each order as *n*-by-1 column vector, where *n* is the number of orders in the scattering network. The sum of the elements of `npaths` is the total number of scattering paths.

Examples

Scattering Paths of Wavelet Image Scattering Network

Create an image scattering network with an image size of 256-by-256 and invariance scale equal to the minimum of the image size. The default `OptimizePath` value is 1 (`true`).

```
sf = waveletScattering2('ImageSize',[256 256],'InvarianceScale',128)
```

```
sf = 
  waveletScattering2 with properties:

            ImageSize: [256 256]
      InvarianceScale: 128
         NumRotations: [6 6]
       QualityFactors: [1 1]
            Precision: "single"
   OversamplingFactor: 0
         OptimizePath: 1
```

Obtain the number of scattering paths in each order. Display the total number of scattering paths.

```
[spaths,npaths] = paths(sf);
sum(npaths)
```

```
ans = 391
```

Set the `OptimizePath` value of the network to `false`. Display the total number of scattering paths. For the modified network, the scattering transform does not reduce the number of paths to compute based on a bandwidth consideration.

```
sf.OptimizePath = false;
[spaths,npaths] = paths(sf);
sum(npaths)
```

```
ans = 571
```

Wavelets on Scattering Path

This example shows how the `OptimizePath` property can affect the scattering paths that include a specific wavelet.

Create the default wavelet image scattering network. Obtain all the wavelet filters and center spatial frequencies for the network. Obtain all the scattering paths. Display the total number of paths.

```
sf = waveletScattering2
[~,psifilters,f] = filterbank(sf);
[spaths,npaths] = paths(sf);
disp(['Total Number of Paths: ',num2str(sum(npaths))])
```

```
sf = 

  waveletScattering2 with properties:

             ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [6 6]
        QualityFactors: [1 1]
             Precision: 'single'
    OversamplingFactor: 0
          OptimizePath: 1

Total Number of Paths: 241
```

Display the number of wavelet filters in each filter bank.

```
disp(['Filter Bank 1: ',num2str(size(psifilters{1},3))]);
disp(['Filter Bank 2: ',num2str(size(psifilters{2},3))]);
```

```
Filter Bank 1: 24
Filter Bank 2: 24
```

Choose a wavelet from the first filter bank and display its spatial center frequency. Use `spaths` to find all the three-element paths that include the chosen wavelet. Display the paths.

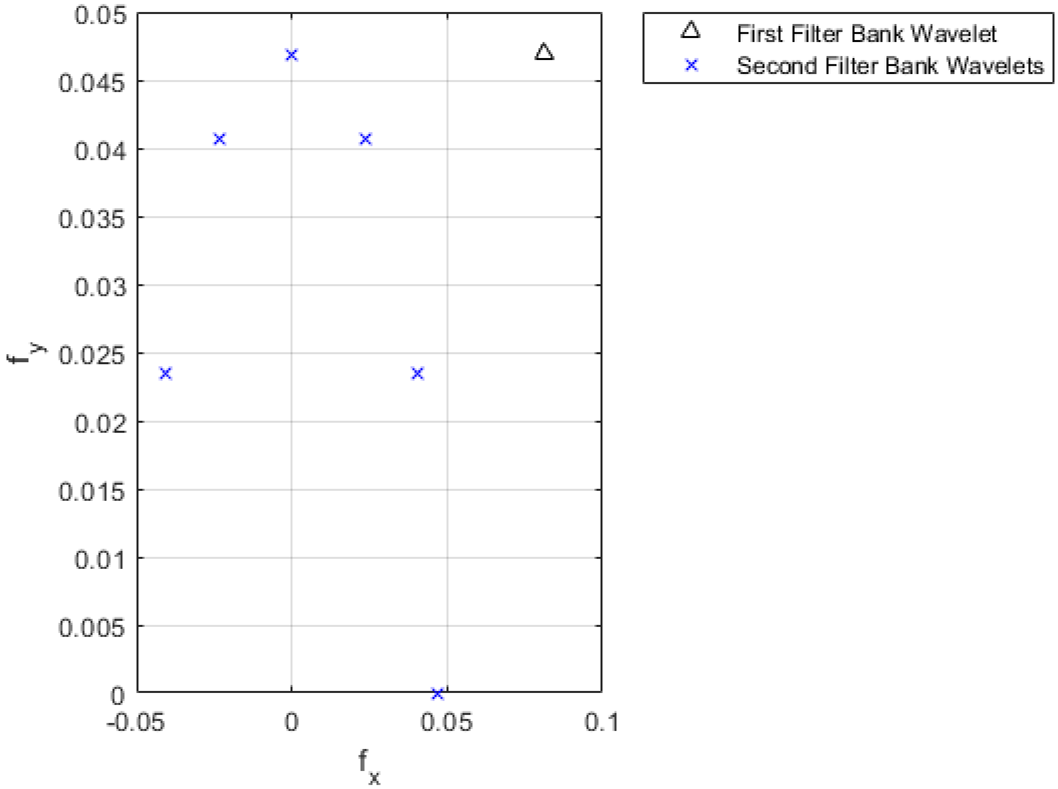
```
waveletA = 14;
disp(['Center Frequency: ',num2str(f{1}(waveletA,:))]);
ind = find(spaths{3}.path(:,2)==waveletA);
spaths{3}(ind,:)
```

```
Center Frequency: 0.08119    0.046875

ans =

  6x1 table
```

```
        path
    _____________

    0    14    19
    0    14    20
    0    14    21
    0    14    22
    0    14    23
    0    14    24
```

Plot the center frequencies of the wavelet filters on the paths.

```
plot(f{1}(waveletA,1),f{1}(waveletA,2),'k^');
xlabel('f_x')
ylabel('f_y')
hold on
waveletBs = spaths{3}.path(ind,3);
plot(f{2}(waveletBs,1),f{2}(waveletBs,2),'bx');
grid on
legend('First Filter Bank Wavelet','Second Filter Bank Wavelets',...
    'Location','northeastoutside')
```



Now set the `OptimizePath` property of the scattering network `sf` to `false`. Obtain the wavelet filters, center spatial frequencies, and scattering paths of the network.

```
sf.OptimizePath = false
[~,psifilters2,f2] = filterbank(sf);
```

```
[spaths2,npaths2] = paths(sf);
disp(['Total Number of Paths: ',num2str(sum(npaths2))])
```

```
sf = 

  waveletScattering2 with properties:

              ImageSize: [128 128]
        InvarianceScale: 64
           NumRotations: [6 6]
         QualityFactors: [1 1]
              Precision: 'single'
    OversamplingFactor: 0
           OptimizePath: 0

Total Number of Paths: 385
```

Choose the same wavelet as above. To confirm it is the same wavelet, display its spatial center frequency. Use `spaths` to find all the three-element paths that include the wavelet. Because `OptimizePath` is set to `false`, the wavelet filter has more children.

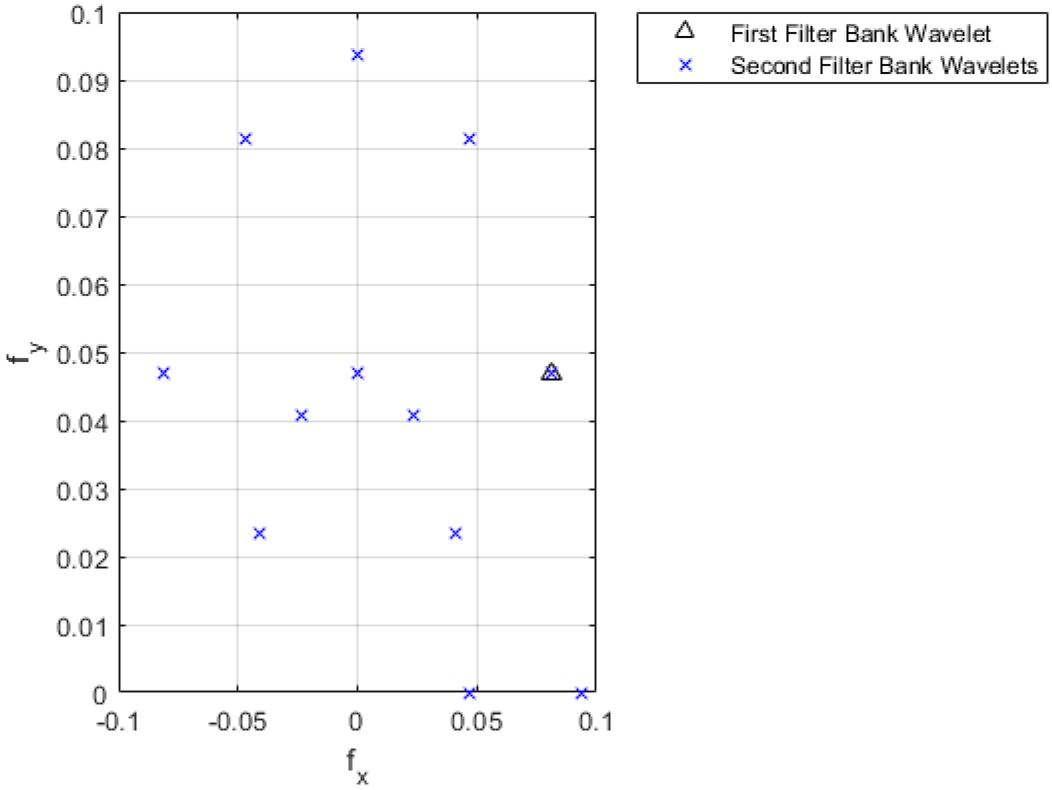
```
waveletA = 14;
disp(['Center Frequency: ',num2str(f2{1}(waveletA,:))]);
ind = find(spaths2{3}.path(:,2)==waveletA);
spaths2{3}(ind,:)
```

```
Center Frequency: 0.08119    0.046875

ans =

  12x1 table

        path
    _____________

    0    14    13
    0    14    14
    0    14    15
    0    14    16
    0    14    17
    0    14    18
    0    14    19
    0    14    20
    0    14    21
    0    14    22
    0    14    23
    0    14    24
```

Plot the center frequencies of the wavelet filters on the paths. Some of child filters have center frequencies higher than the chosen wavelet.

```
figure
plot(f2{1}(waveletA,1),f2{1}(waveletA,2),'k^');
xlabel('f_x')
ylabel('f_y')
hold on
```

```
waveletBs = spaths2{3}.path(ind,3);
plot(f2{2}(waveletBs,1),f2{2}(waveletBs,2),'bx');
grid on
legend('First Filter Bank Wavelet','Second Filter Bank Wavelets',...
    'Location','northeastoutside')
```



Input Arguments

sf — Wavelet image scattering network

waveletScattering2 object

Wavelet image scattering network, specified as a `waveletScattering2` object.

Output Arguments

spaths — Scattering paths

cell array

Scattering paths of all orders of the scattering network, returned as a cell array of MATLAB tables. `spaths` has n elements, where n is the number of orders in the scattering network.

Each MATLAB table in `spaths` contains a single variable, `path`. The variable `path` is a row vector with one column for each element of the path. The scalar 0 denotes the original image. Positive integers in the Lth column denote the corresponding wavelet filter in the $(L-1)$th filter bank. Wavelet bandpass filters are ordered by decreasing center frequency. There are `NumRotations` wavelets per center frequency pair.

`npaths` — Number of scattering paths
column vector

Number of scattering paths in each order of the scattering network. `npaths` is a *no*-by-1 column vector where *no* is the number of orders in the scattering network. The sum of the elements of `npaths` is the total number of scattering paths.

See Also

`waveletScattering2` | `coefficientSize`

Introduced in R2019a

plot

Plot tree GUI

Syntax

```
plot(t)
fig = plot(t)
newt = plot(trobj,'read',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.

- You can change the **Node Label** to **Depth_Position** (default) or **Index**.
- You can change the **Node Action** to **Visualize** (default) or **Split-Merge**.

You can click the nodes to execute the current **Node Action**.

`fig = plot(t)` returns the handle to the figure.

`newt = plot(trobj,'read',fig)` returns the tree plotted in `fig`. You can use this syntax to return the tree after performing some split or merge actions.

Examples

Plot Wavelet Packet Trees

This example shows how to plot wavelet packet trees.

Load a 1-D signal.

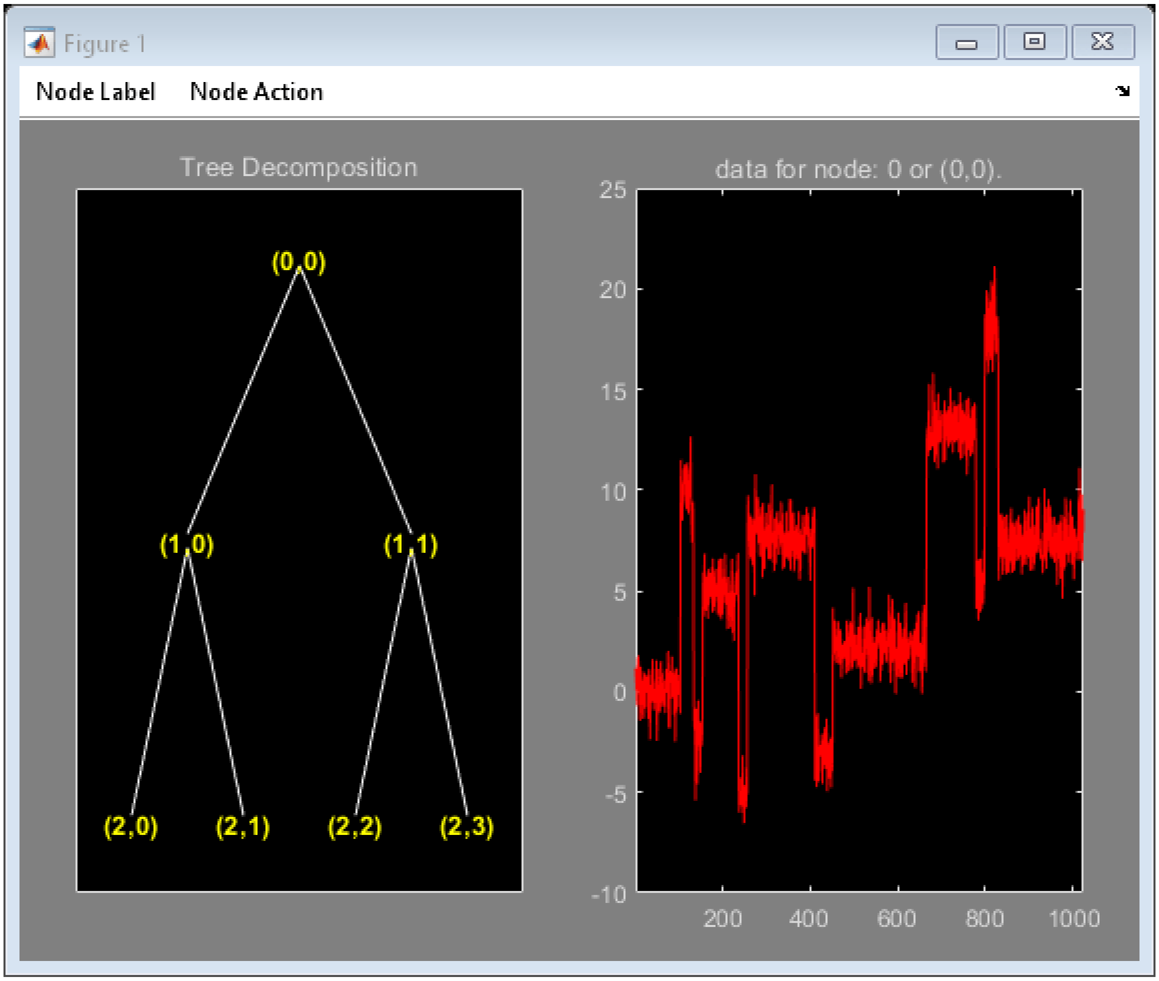
```
load noisbloc
```

Obtain the `wptree` object that corresponds to a level 2 wavelet packet decomposition of the signal using the `db2` wavelet.

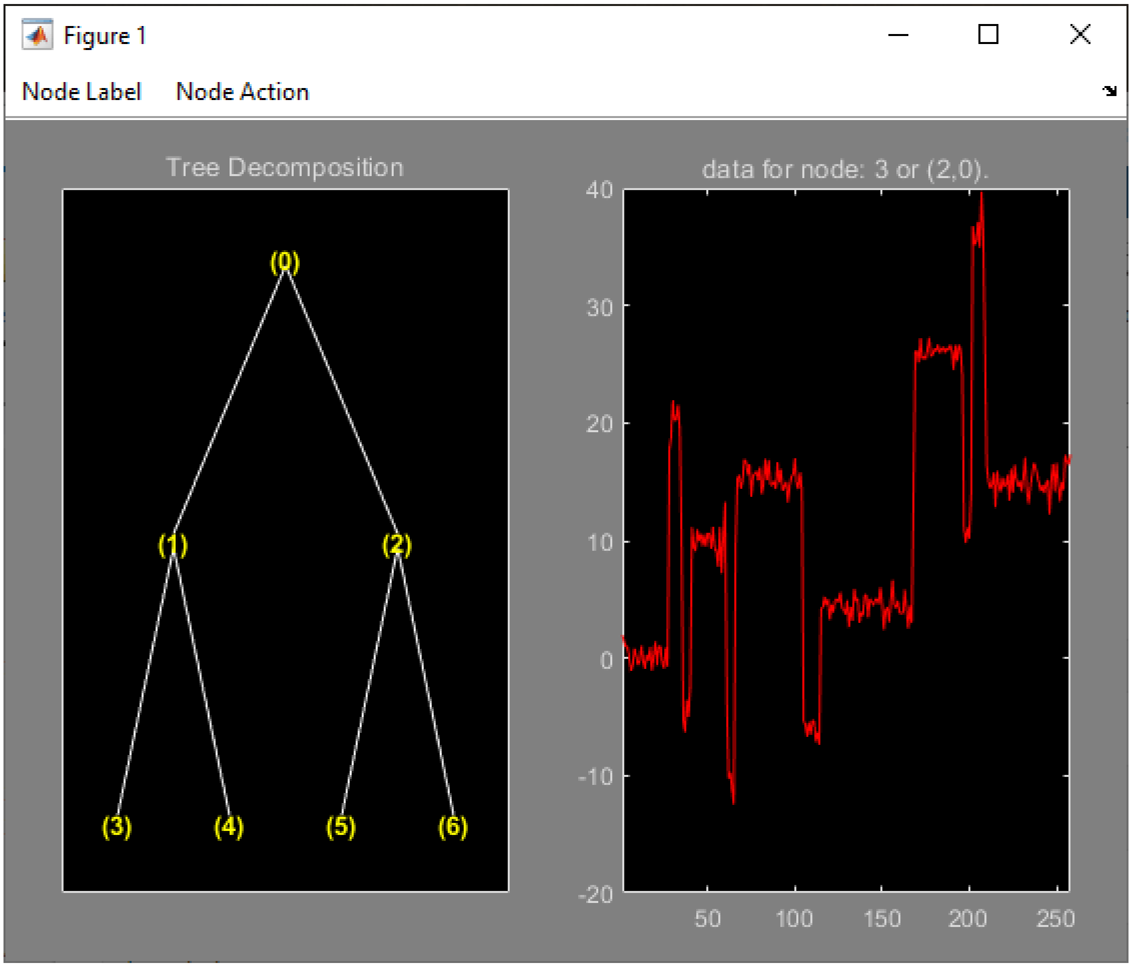
```
x = noisbloc;
t = wpdec(x,2,'db2');
```

Plot the tree.

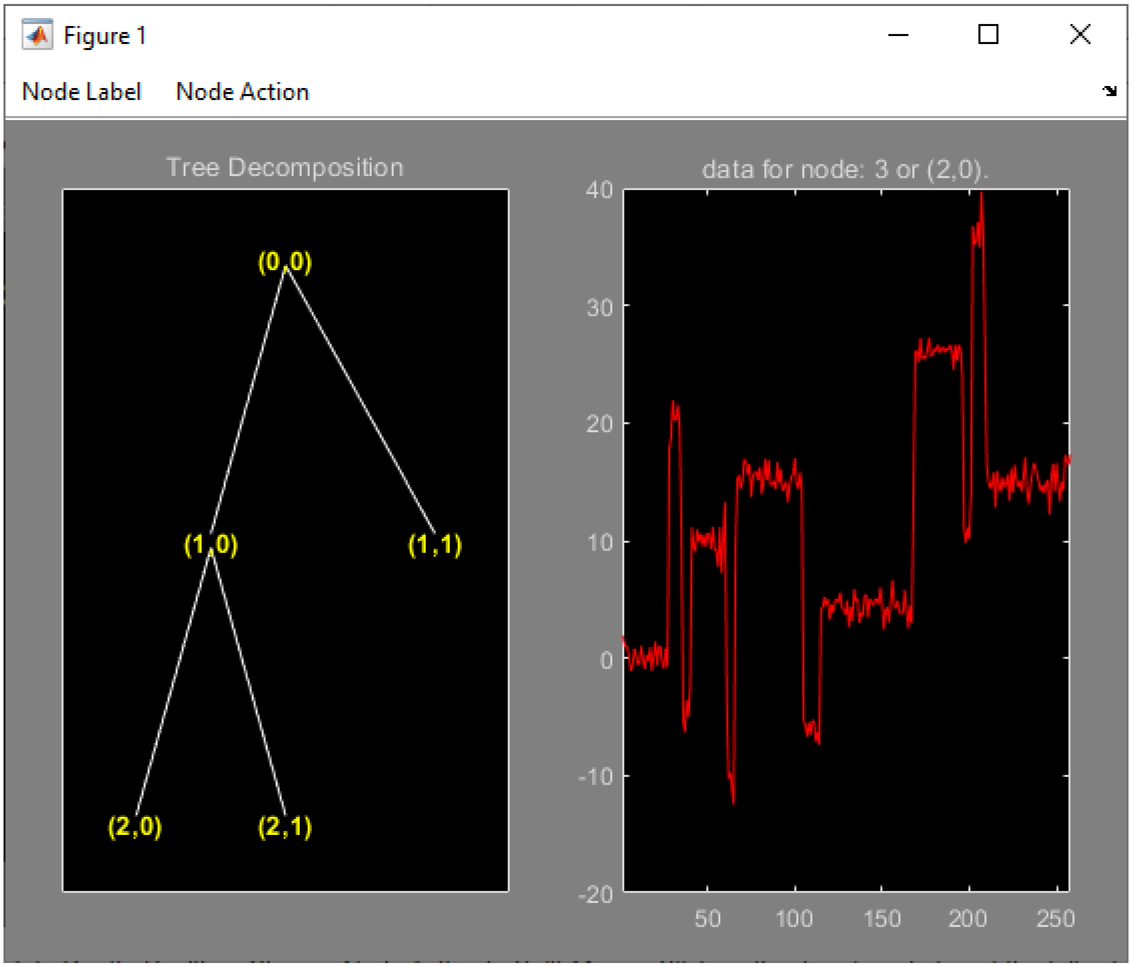
```
plot(t)
```



Change **Node Label** from **Depth_Position** to **Index**. Click node (3) to obtain the following figure.



Set the **Node Label** back to **Depth_Position**. Change **Node Action** to **Split-Merge**. Click on the `(1,1)` node to get the following figure. The figure shows the discrete wavelet transform down to level 2.

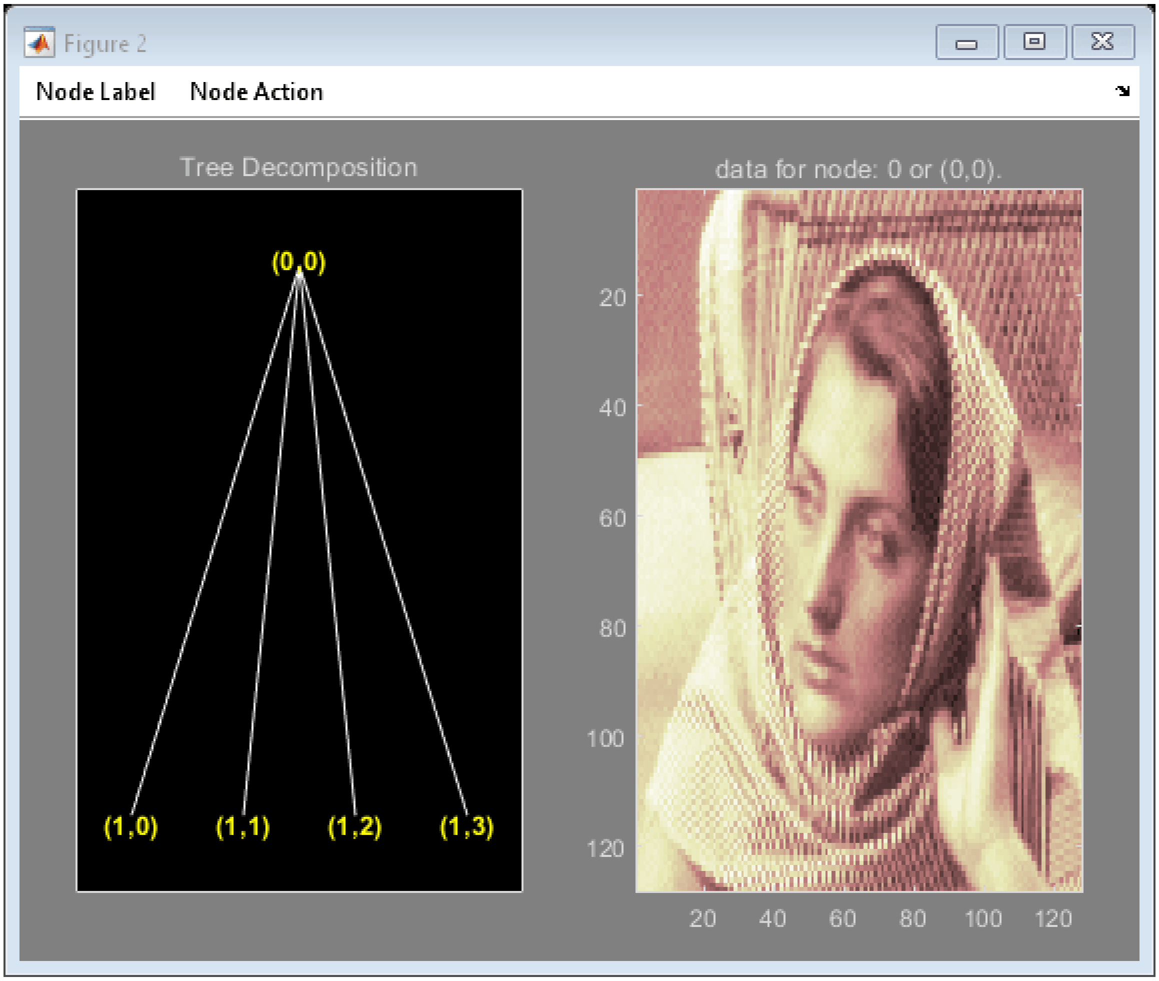


Load an image. Obtain the `wptree` object that corresponds to a level 1 wavelet packet decomposition of the image using the `sym4` wavelet.

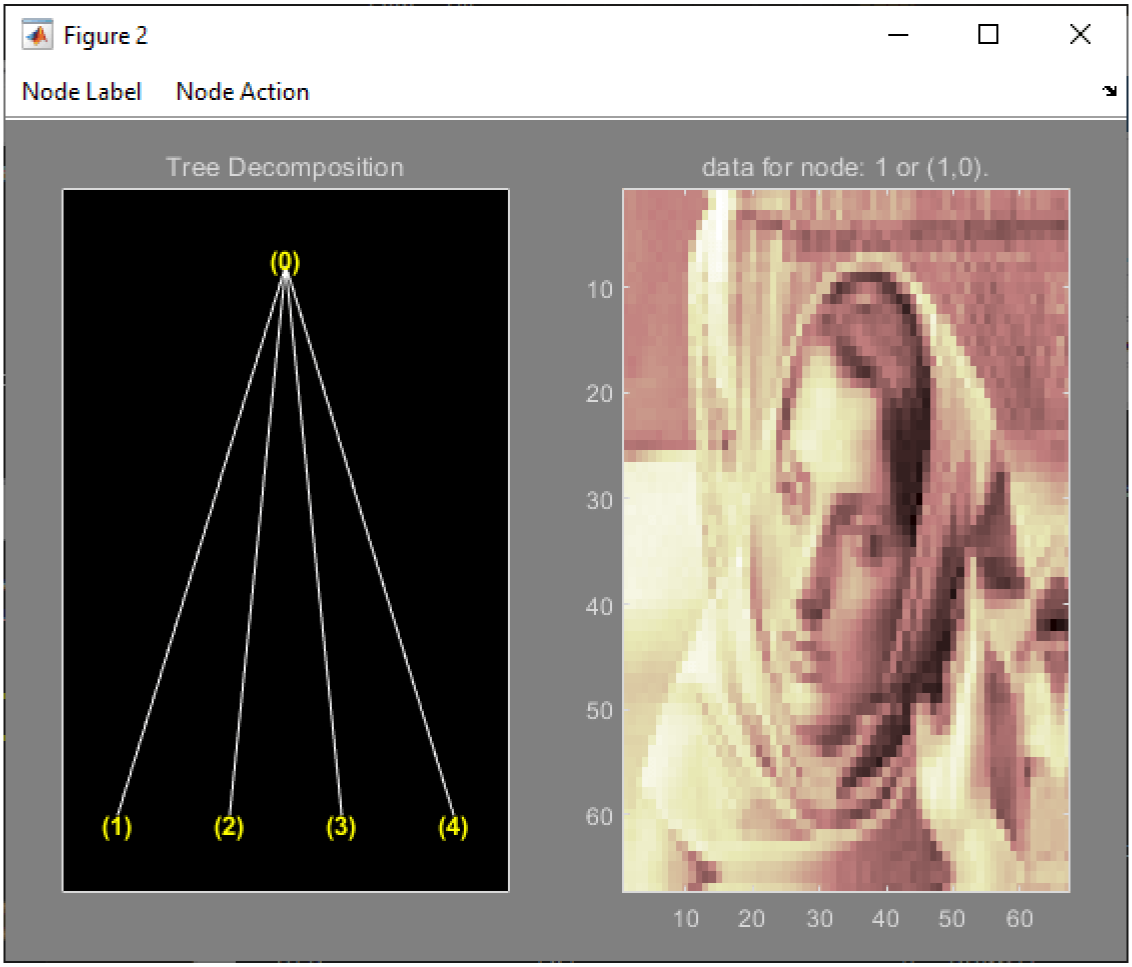
```
load woman2
t = wpdec2(X,1,'sym4');
```

Plot the tree.

```
plot(t)
```



Change **Node Label** from **Depth_Position** to **Index**. Click the node `(1)`. You get the following figure.



Input Arguments

t — Tree
ntree object | dtree object | wptree object

Tree, specified as an `ntree`, `dtree`, or `wptree` object.

`trobj` — Object
NTREE-parented object

Object, specified as an NTREE-parented object. `trobj` is an object constructor name returning an NTREE-parented object.

Example: `newt = plot(ntree,'read',fig)`, `newt = plot(dtree,'read',fig)`, `newt = plot(wptree,'read',fig)`

Output Arguments

`fig` — Figure
Figure object

Figure containing plot, returned as a `Figure` object.

See Also

`ntree` | `dtree` | `wptree`

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

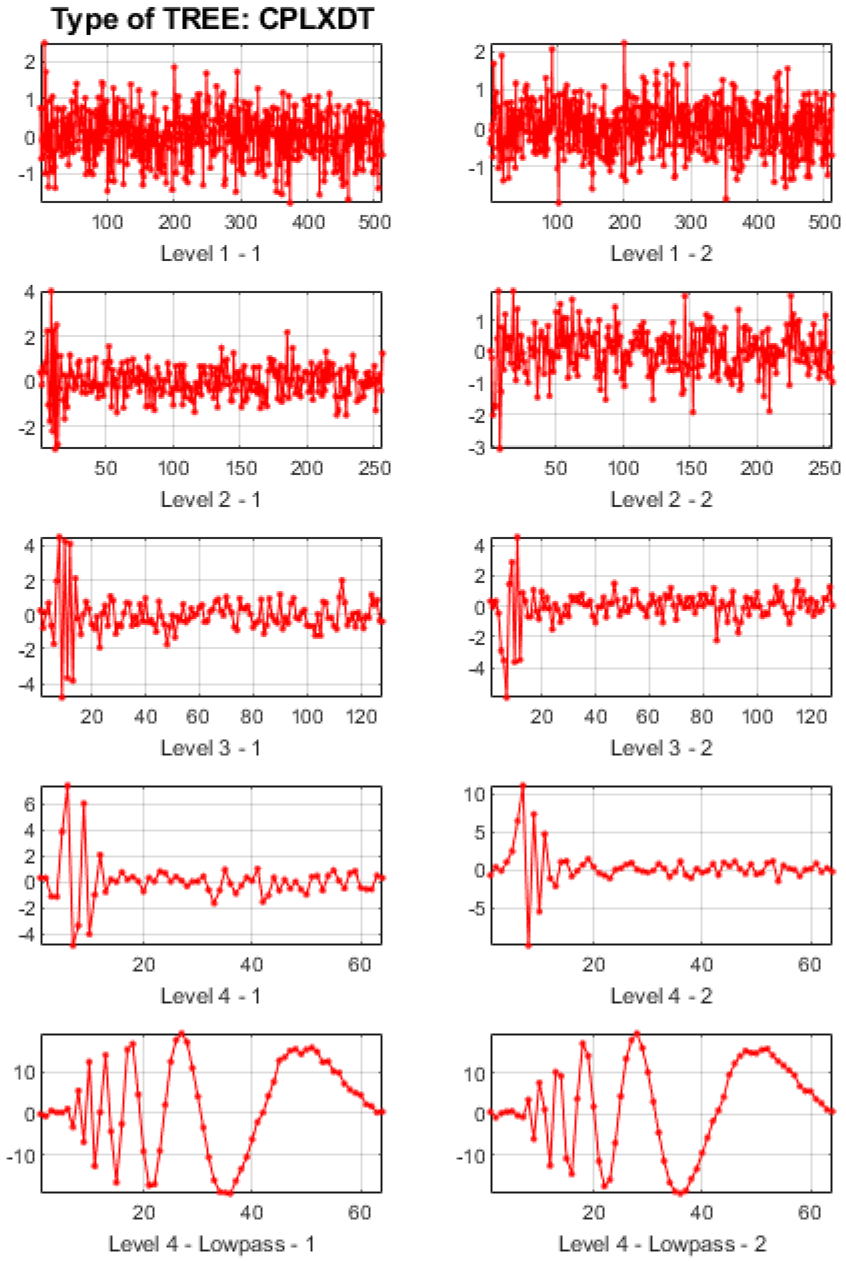
Plot the complex dual-tree wavelet transform of the noisy Doppler signal.

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)
```



Plot Complex Oriented Dual-Tree Wavelet Transform of 2-D Image

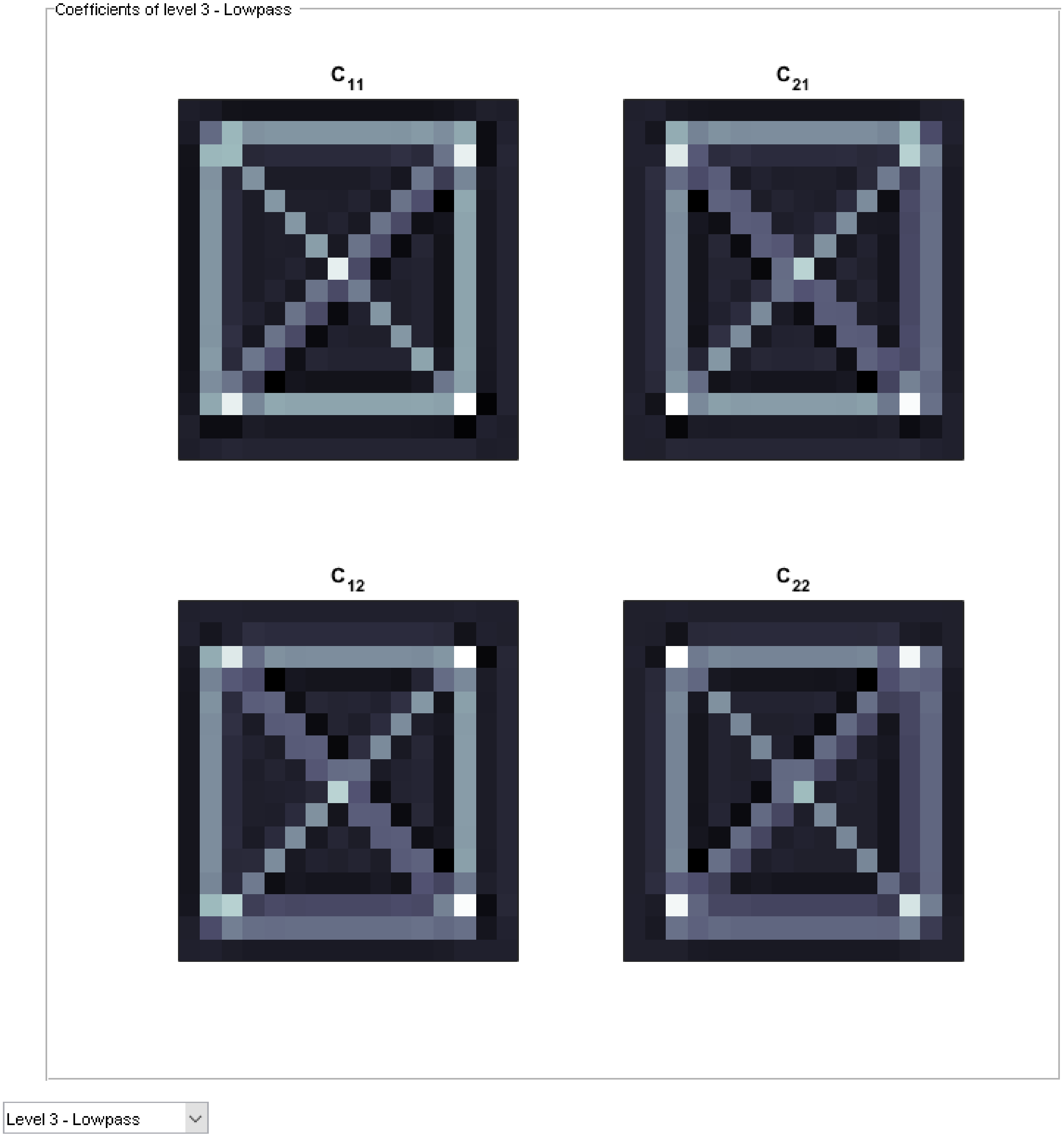
Plot the complex oriented dual-tree wavelet transform of an image.

Load the xbox image. Obtain the complex oriented dual-tree wavelet transform down to level 3.

```
load xbox;
wt = dddtree2('cplxdt',xbox,3,'dtf1');
```

Plot the coefficients.

```
plotdt(wt)
```



Select the desired level detail coefficients from the drop-down list.

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` | `dualtree` | `dualtree2`

Topics

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

Introduced in R2013b

plus

Laurent polynomial or Laurent matrix addition

Syntax

```
Q = plus(A,B)
Q = A + B
```

Description

`Q = plus(A,B)` returns the sum of the pair of Laurent polynomials or Laurent matrices `A` and `B`.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `plus`. The input data type determines which version is executed.

`Q = A + B` is equivalent to `Q = plus(A,B)`.

Examples

Laurent Polynomial Addition

Create two Laurent polynomials:

- $a(z) = z^2 + 2z + 3 + 5z^{-1} + 8z^{-2} + 13z^{-3}$
- $b(z) = 8z + 4 + 2z^{-1} + z^{-2}$

```
a = laurentPolynomial(Coefficients=[1 2 3 5 8 13],MaxOrder=2);
b = laurentPolynomial(Coefficients=[8 4 2 1],MaxOrder=1);
```

Add $a(z)$ and $b(z)$.

```
c = a+b
```

```
c = 
  laurentPolynomial with properties:

    Coefficients: [1 10 7 7 9 13]
        MaxOrder: 2
```

Add $a(z)$ and the negative of $b(z)$.

```
d = plus(a,-b)
```

```
d = 
  laurentPolynomial with properties:

    Coefficients: [1 -6 -1 3 7 13]
```

```
        MaxOrder: 2
```

Laurent Matrix Addition

Create two Laurent polynomials:

- $a(z) = z + 1$
- $b(z) = z^2 - z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[1 1],MaxOrder=1);
lpB = laurentPolynomial(Coefficients=[1 0 0 -1],MaxOrder=2);
```

Create two Laurent matrices:

- $\texttt{lmatA} = \begin{bmatrix} a(z) & 1 \\ 1 & 0 \end{bmatrix}$
- $\texttt{lmatB} = \begin{bmatrix} 0 & 2 \\ 3 & b(z) \end{bmatrix}$

```
lmatA = laurentMatrix(Elements={lpA,1;1,0});
lmatB = laurentMatrix(Elements={0,2;3,lpB});
```

Sum the matrices.

```
lmat = lmatA+lmatB;
lmat.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 1]
        MaxOrder: 1
```

```
lmat.Elements{1,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 3
        MaxOrder: 0
```

```
lmat.Elements{2,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 4
        MaxOrder: 0
```

```
lmat.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 0 0 -1]
        MaxOrder: 2
```

Input Arguments

A — Laurent polynomial or Laurent matrix

`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

B — Laurent polynomial or Laurent matrix

`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Output Arguments

Q — Sum

`laurentPolynomial` object | `laurentMatrix` object

Sum of two Laurent polynomials or two Laurent matrices, returned as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`minus` | `mtimes`

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

polyphase

Polyphase components of Laurent polynomial

Syntax

```
[E,O] = polyphase(P)
```

Description

`[E,O] = polyphase(P)` returns the even part `E` and odd part `O` of the Laurent polynomial `P`.

Examples

Polyphase Decomposition of Laurent Polynomial

Create the Laurent polynomial $b(z) = z^3 + 3z^2 - 1 + 2z^{-1}$.

```
b = laurentPolynomial(Coefficients=[1 3 0 -1 0 2],MaxOrder=3);
```

Use the `polyphase` function to obtain the even and odd parts of $b(z)$. Use the helper function `helperPrintLaurent` to print the Laurent polynomials in algebraic form.

```
[evenP,oddP] = polyphase(b);
resE = helperPrintLaurent(evenP);
disp(resE)
```

```
3*z - 1 + 2*z^(-1)
```

```
resO = helperPrintLaurent(oddP);
disp(resO)
```

```
z^(2)
```

Confirm the identity $E(z^2) + z^{-1}O(z^2) = \; = b(z)$, where $E(z)$ and $O(z)$ are the even and odd parts, respectively, of $b(z)$.

```
evenPz2 = dyadup(evenP);
oddPz2 = dyadup(oddP);
lpz = laurentPolynomial(Coefficients=1,MaxOrder=-1);
leftSide = evenPz2+(lpz*oddPz2);
areEqual = (leftSide == b)
```

```
areEqual = logical
   1
```

Input Arguments

P — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

Output Arguments

E — Even part

`laurentPolynomial` object

Even part of the Laurent polynomial `P`, returned as a `laurentPolynomial` object. The polynomial `E` is such that:

$$E(z^2) = [P(z) + P(-z)]/2.$$

O — Odd part

`laurentPolynomial` object

Odd part of the Laurent polynomial `P`, returned as a `laurentPolynomial` object. The polynomial `O` is such that:

$$O(z^2) = [P(z) - P(-z)]/ [2 z^{-1}].$$

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`dyaddown` | `dyadup`

Objects

`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

powerbw

CWT filter bank 3 dB bandwidths

Syntax

```
bw = powerbw(fb)
```

Description

`bw = powerbw(fb)` returns 3 dB (half-power) bandwidths for the wavelet filters in the filter bank `fb`. `bw` is a *Ns*-by-4 MATLAB `table`, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales). For every filter in `fb`, the table contains the corresponding bandpass frequency, the 3 dB bandwidth, and the lower frequency and upper frequency limits of the 3 dB bandwidth.

The 3 dB bandwidth limits mark where the filter power is half its peak value. The magnitude frequency response at the limits is equal to 1/√2 times the peak magnitude. Since the passbands in `fb` are normalized with peak magnitudes approximately equal to 2, the magnitude frequency response at each limit is approximately equal to 2/√2. The 3 dB bandwidth is also known as the half-power bandwidth because $20\log_{10}\frac{1}{\sqrt{2}} \approx -3$.

Examples

Half-Power Wavelet Bandwidths

Create a CWT filter bank.

```
fb = cwtfilterbank;
```

Obtain the 3 dB (half-power) bandwidths of the filter bank. Obtain the frequency responses of the wavelets.

```
bw = powerbw(fb);
[psidft,f] = freqz(fb);
```

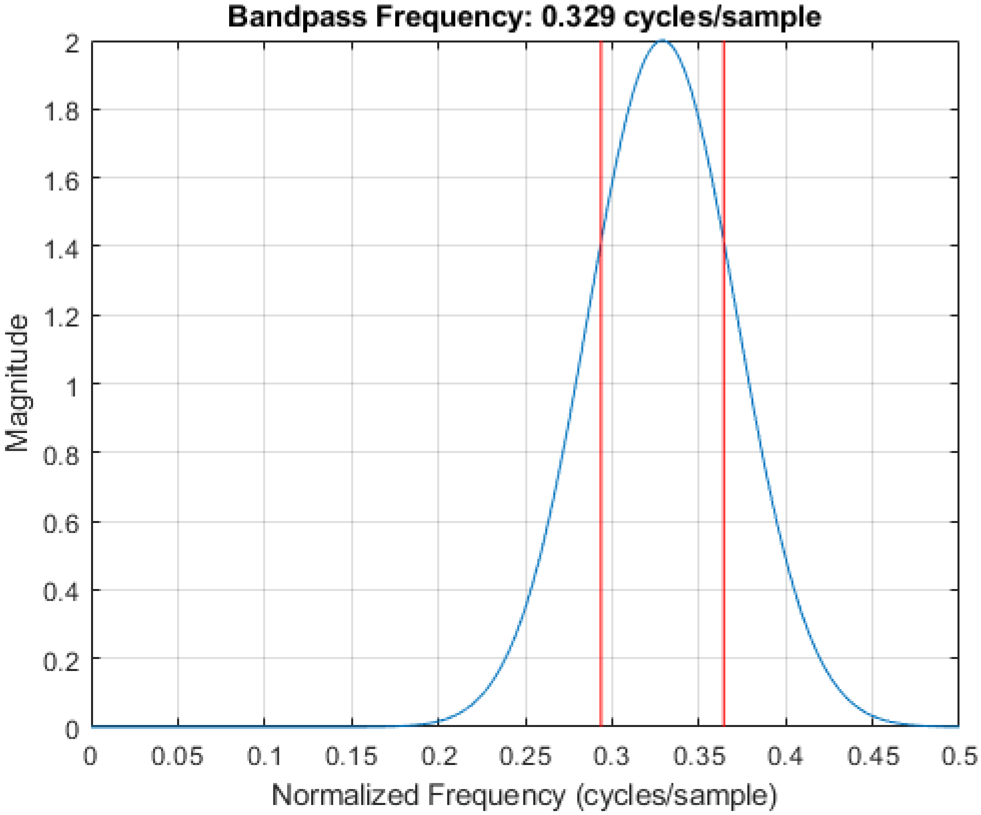
Choose a wavelet bandpass filter from the filter bank. Extract from the table `bw` the 3 dB limits of the bandpass filter.

```
wv = 5;
frq = bw.Frequencies(wv);
lfb = bw.LowFrequencyBorder(wv);
hfb = bw.HighFrequencyBorder(wv);
```

Plot the frequency response and 3 dB limits. Since the frequency response is scaled to have a maximum value equal to 2, inspect the plot to confirm the lower and upper frequency borders intersect the frequency response at $\sqrt{2}$.

```
plot(f,psidft(wv,:))
grid on
```

```
hold on
plot([lfb lfb],[0 2],'r')
plot([hfb hfb],[0 2],'r')
xlabel('Normalized Frequency (cycles/sample)')
ylabel('Magnitude')
title(['Bandpass Frequency: ' num2str(frq) ' cycles/sample'])
```



Input Arguments

fb — Continuous wavelet transform filter bank
`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

bw — 3 dB (half-power) bandwidths
`table`

3 dB (half-power) bandwidths, returned as an *Ns*-by-4 `table`, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales). The table has four variables:

`Frequencies` — Bandpass frequency
positive scalar

Bandpass frequency, returned as a positive scalar (see `centerFrequencies`).

Data Types: `double`

`HalfPowerBandwidth` — Half-power bandwidth
positive scalar

Half-power bandwidth, returned as a positive scalar.

Data Types: `double`

`LowFrequencyBorder` — Lower frequency edge
positive scalar

Lower frequency edge of the 3 dB bandwidth, returned as a positive scalar.

Data Types: `double`

`HighFrequencyBorder` — High frequency edge
positive scalar

High frequency edge of the 3 dB bandwidth, returned as a positive scalar.

Data Types: `double`

Data Types: `table`

See Also
`cwtfilterbank` | `freqz` | `centerFrequencies`

Introduced in R2018a

powerbw

DWT filter bank power bandwidth

Syntax

```
bwtable = powerbw(fb)
```

Description

`bwtable = powerbw(fb)` returns a MATLAB table `bwtable` containing the theoretical and measured bandwidths of the discrete wavelet transform (DWT) filter bank `fb`. The table contains the following variables by level:

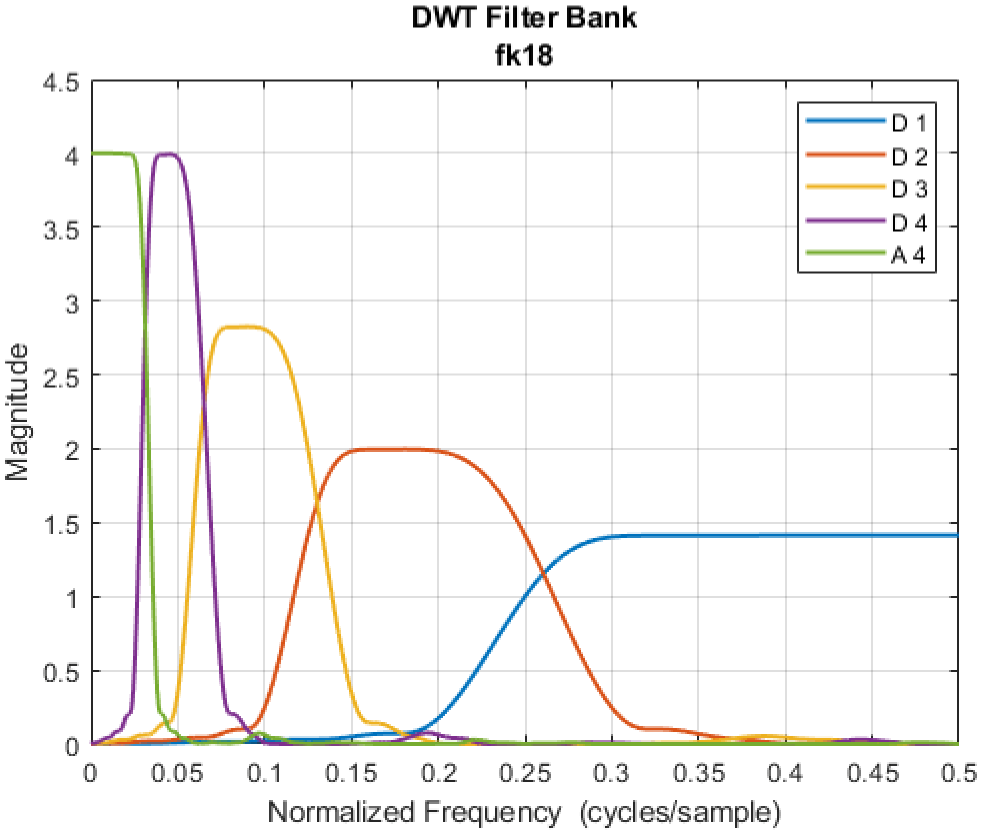
- DWT frequency bands
- Measured wavelet and scaling filter 3 dB bandwidths
- Proportions of the total energy in the reported bands

Examples

DWT Filter Bank Power Bandwidth

Obtain the 3 dB bandwidths of a level-4 discrete wavelet transform with the Fejér-Korovkin `fk18` wavelet. Obtain the frequency responses of the wavelets. Plot the one-sided frequency responses for the wavelet filters.

```
fb = dwtfilterbank('Wavelet','fk18','Level',4);
bw = powerbw(fb);
[psidft,f] = freqz(fb);
freqz(fb)
```

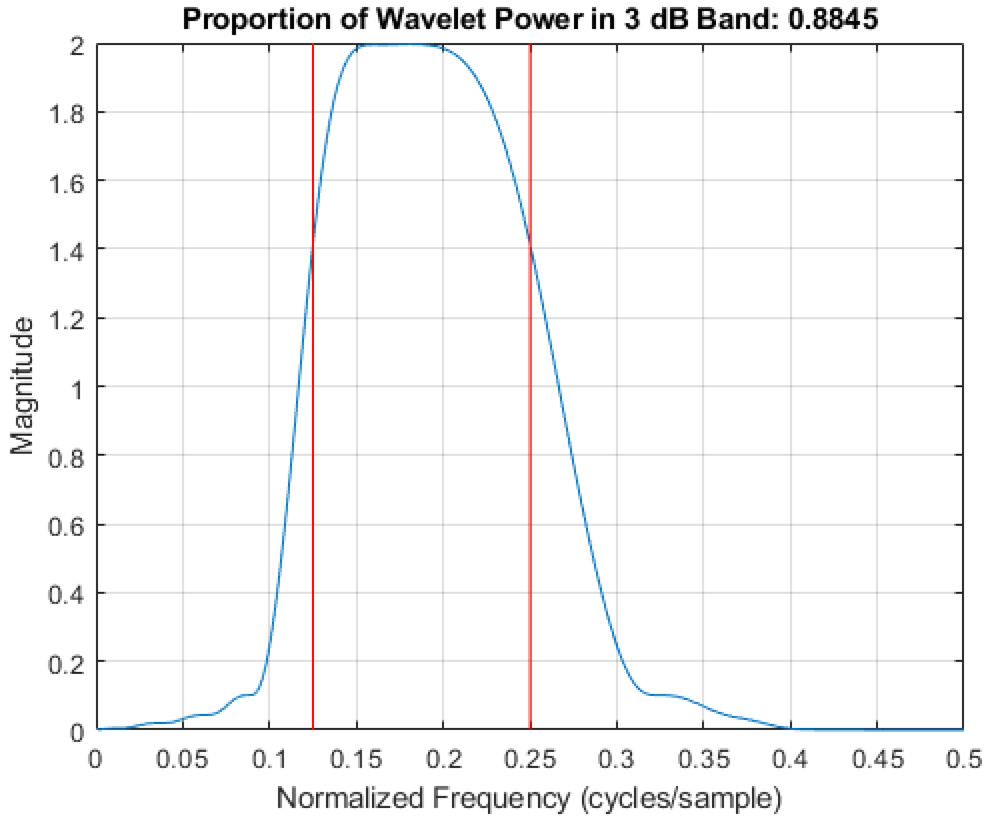


Choose the wavelet bandpass filter whose peak magnitude is equal to 2. Obtain the lower and upper bounds of the 3 dB bandwidth of the filter.

```
wv = 2;
wvBw = bw.Wavelet3dBBandwidth(wv,:);
```

Plot the magnitude frequency response of the filter and the 3 dB limits. Since the frequency response has a maximum value equal to 2, confirm the lower and upper frequency bounds intersect the frequency response at $\sqrt{2}$.

```
filLength = size(psidft,2);
plot(f(filLength/2+1:end),abs(psidft(wv,filLength/2+1:end)))
hold on
plot([wvBw(1) wvBw(1)],[0 2],'r')
plot([wvBw(2) wvBw(2)],[0 2],'r')
grid on
title(['Proportion of Wavelet Power in 3 dB Band: ',num2str(bw.WaveletPowerIn3dBBand(wv))])
xlabel('Normalized Frequency (cycles/sample)')
ylabel('Magnitude')
```



Input Arguments

`fb` — Discrete wavelet transform filter bank

`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

Output Arguments

`bwtable` — Theoretical and measured bandwidths

table

Theoretical and measured bandwidths of the DWT filter bank `fb`, returned as a MATLAB table. `bwtable` is *L*-by-8, where *L* is the wavelet transform level of the filter bank. Levels are ordered by decreasing resolution. `bwtable` has the following eight variables:

`Level` — Level of DWT decomposition

positive integer

Level of DWT decomposition, returned as a positive integer.

`DWTBand` — Theoretical DWT frequency bands

two-element real-valued vector

Theoretical DWT frequency bands by level, returned as a two-element real-valued vector.

`Wavelet3dBBandwidth` — Measured wavelet 3 dB bandwidths
two-element real-valued vector

Measured wavelet 3 dB bandwidths by level, returned as a two-element real-valued vector.

`Scaling3dBBandwidth` — Measured scaling filter 3 dB bandwidths
two-element real-valued vector

Measured scaling filter 3 dB bandwidths by level, returned as a two-element real-valued vector.

`WaveletPowerIn3dBBand` — Proportion of total wavelet power
positive scalar

Proportion of total wavelet power in the measured 3 dB band by level, returned as a positive scalar.

`ScalingPowerIn3dBBand` — Proportion of total scaling filter power
positive scalar

Proportion of total scaling filter power in the measured 3 dB band by level, returned as a positive scalar.

`WaveletPowerInDWTBand` — Proportion of total wavelet power
positive scalar

Proportion of total wavelet power in the theoretical DWT band by level, returned as a positive scalar.

`ScalingPowerInDWTBand` — Proportion of total scaling filter power
positive scalar

Proportion of total scaling filter power in the theoretical DWT band by level, returned as a positive scalar.

See Also

`dwtfilterbank` | `dwtpassbands`

Introduced in R2018a

qbiorthfilt

First-level dual-tree biorthogonal filters

Syntax

```
[LoD,HiD,LoR,HiR] = qbiorthfilt(name)
```

Description

`[LoD,HiD,LoR,HiR] = qbiorthfilt(name)` returns the first-level biorthogonal filters for Kingsbury's Q-shift complex dual-tree transform specified by `name`.

Examples

DTCWT First-Level Biorthogonal Filters

Obtain the decomposition and reconstruction filters associated with the biorthogonal wavelet `nearsym5_7`.

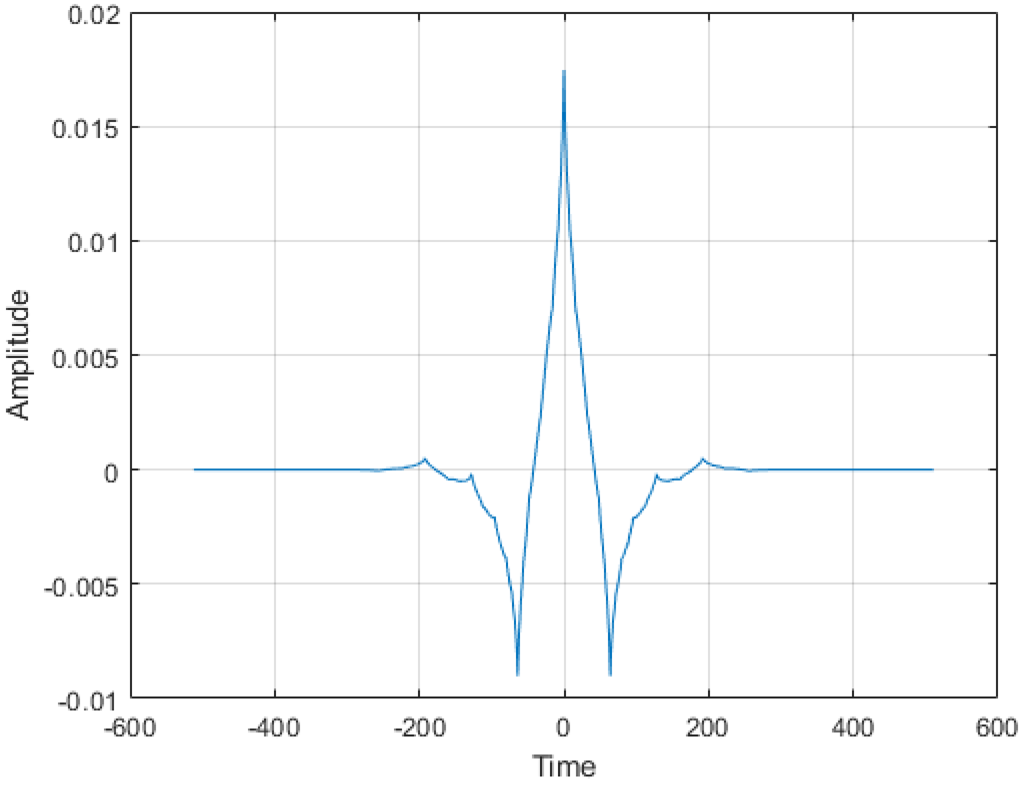
```
fname = 'nearsym5_7';
[LoD,HiD,LoR,HiR] = qbiorthfilt(fname);
```

Use the `dwtfilterbank` function to create a 7-level discrete wavelet transform filter bank with the biorthogonal filters. Specify the wavelet filter type as `analysis`. Because the filters are not of even lengths, extend the filters appropriately to match powers of their *z*-transforms.

```
scal(:,1) = [0 0 LoD' 0];
scal(:,2) = [0 LoR'];
wavf(:,1) = [0 HiD'];
wavf(:,2) = [0 0 HiR' 0];
fb = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',scal,...
    'CustomWaveletFilter',wavf,...
    'Level',7,...
    'FilterType','analysis');
```

Obtain the time-domain wavelets corresponding to the wavelet passband filters. Plot the coarsest-scale wavelet.

```
[psi,t] = wavelets(fb);
plot(t,psi(end,:))
grid on
xlabel('Time')
ylabel('Amplitude')
```



Input Arguments

name — First-level biorthogonal filter

'nearsym5_7' | 'nearsym13_19' | 'antonini' | 'legall'

First-level biorthogonal filter used in Kingsbury's Q-shift complex dual-tree transform, specified by one of the values listed here.

- 'nearsym5_7' — (5,7)-tap near-orthogonal filter [1]
- 'nearsym13_19' — (13,19)-tap near-orthogonal filter [2]
- 'antonini' — (9,7)-tap Antonini filter [1]
- 'legall' — LeGall 5/3 filter [3]

Output Arguments

LoD — Lowpass analysis filter

real-valued vector

Lowpass (scaling) analysis filter associated with the biorthogonal filter name, returned as a real-valued vector. The length of LoD does not equal the length of HiD.

HiD — Highpass analysis filter

real-valued vector

Highpass (wavelet) analysis filter associated with the biorthogonal filter `name`, returned as a real-valued vector. The length of `LoD` does not equal the length of `HiD`.

`LoR` — Lowpass synthesis filter

real-valued vector

Lowpass (scaling) synthesis filter associated with the biorthogonal filter `name`, returned as a real-valued vector. The length of `LoR` does not equal the length of `HiR`.

`HiR` — Highpass synthesis filter

real-valued vector

Highpass (wavelet) synthesis filter associated with the biorthogonal filter `name`, returned as a real-valued vector. The length of `LoR` does not equal the length of `HiR`.

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation

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

See Also

`dualtree3` | `dualtree2` | `dualtree` | `qorthwavf`

Topics

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

Introduced in R2020a

qfactor

CWT filter bank quality factor

Syntax

```
qf = qfactor(fb)
```

Description

`qf = qfactor(fb)` returns the quality factor for the wavelet bandpass filters in `fb`. The quality factor is the ratio of the 3-dB bandwidth to the center frequency, where the center frequency is the geometric mean of the bandwidth frequencies. The larger the quality factor, the more frequency localized the wavelet. For reference, a half-band filter has a quality factor of `sqrt(2)`.

Examples

Quality Factor of CWT Filter Bank

Create a CWT filter bank using the default analytic Morse (3,60) wavelet.

```
fb = cwtfilterbank;
```

Compute the quality factor of the filter bank.

```
qf = qfactor(fb)
```

```
qf = 4.6296
```

Create a CWT filter bank using the analytic Morse (3,10) wavelet. Compute the quality factor of the filter bank. The analytic Morlet (3,10) wavelet is not localized in frequency as well as the Morse (3,60) wavelet. Confirm that the quality factor of the second filter bank is smaller than the first filter bank.

```
fb2 = cwtfilterbank('Timebandwidth',10);
qf2 = qfactor(fb2)
```

```
qf2 = 1.8445
```

Input Arguments

fb — Continuous wavelet transform filter bank
`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

qf — Quality factor
positive number

Quality factor, returned as a positive real number.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`cwtfilterbank` | `powerbw`

Introduced in R2018a

qfactor

DWT filter bank quality factor

Syntax

```
qf = qfactor(fb)
```

Description

`qf = qfactor(fb)` returns the quality factor for the discrete wavelet transform (DWT) filter bank `fb`.

The quality factor `qf` is defined to be the geometric mean frequency of the lower and upper 3 dB bandwidth frequencies divided by the 3 dB bandwidth. For orthogonal wavelets, the measured quality factor approximates the theoretical value of √2.

Examples

DWT Filter Bank Quality Factor

Obtain the quality factor for the Coiflet `coif4`. Since the wavelet is orthogonal, confirm the quality factor approximates the theoretical value of $\sqrt{2}$.

```
wvOrth = 'coif4';
fb = dwtfilterbank('Wavelet',wvOrth);
orthogAnalysis = qfactor(fb);
abs(orthogAnalysis-sqrt(2))
```

```
ans = 5.7311e-11
```

Compare with the quality factor for the biorthogonal wavelet `bior6.8`. Since the wavelet is biorthogonal, confirm the quality factor does not approximate $\sqrt{2}$.

```
wvBior = 'bior6.8';
fb2 = dwtfilterbank('Wavelet',wvBior);
biorthogAnalysis = qfactor(fb2);
abs(biorthogAnalysis-sqrt(2))
```

```
ans = 0.1339
```

By default, `fb` and `fb2` filter banks have the default filter type `Analysis`. Create two new filter banks of filter type `Synthesis` for the same wavelets. Compare the quality factors with the filter type `Analysis` filter banks. Confirm the quality factors using the orthogonal wavelet are equal.

```
fb3 = dwtfilterbank('Wavelet',wvOrth,'FilterType','Synthesis');
fb4 = dwtfilterbank('Wavelet',wvBior,'FilterType','Synthesis');
orthogSynthesis = qfactor(fb3);
abs(orthogSynthesis-sqrt(2))
```

```
ans = 5.7311e-11
```

```
biorthogSynthesis = qfactor(fb4);
abs(biorthogSynthesis-sqrt(2))
```

```
ans = 0.1141
```

Input Arguments

fb — Discrete wavelet transform filter bank

dwtfilterbank object

Discrete wavelet transform (DWT) filter bank, specified as a dwtfilterbank object.

See Also

dwtfilterbank

Introduced in R2018a

qmf

Scaling and wavelet filter

Syntax

```
Y = qmf(X)
Y = qmf(X,P)
```

Description

`Y = qmf(X)` changes the signs of the even-indexed elements of the reversed vector filter coefficients `X`.

`Y = qmf(X,P)` changes the signs of the even-indexed elements of the reversed vector filter coefficients `X` if `P` is `0`. If `P` is `1`, the signs of the odd-indexed elements are reversed. Changing `P` changes the phase of the Fourier transform of the resulting wavelet filter by π radians.

Examples

Create Quadrature Mirror Filter

This example shows how to create a quadrature mirror filter associated with the `db10` wavelet.

Obtain the scaling filter associated with the `db10` wavelet.

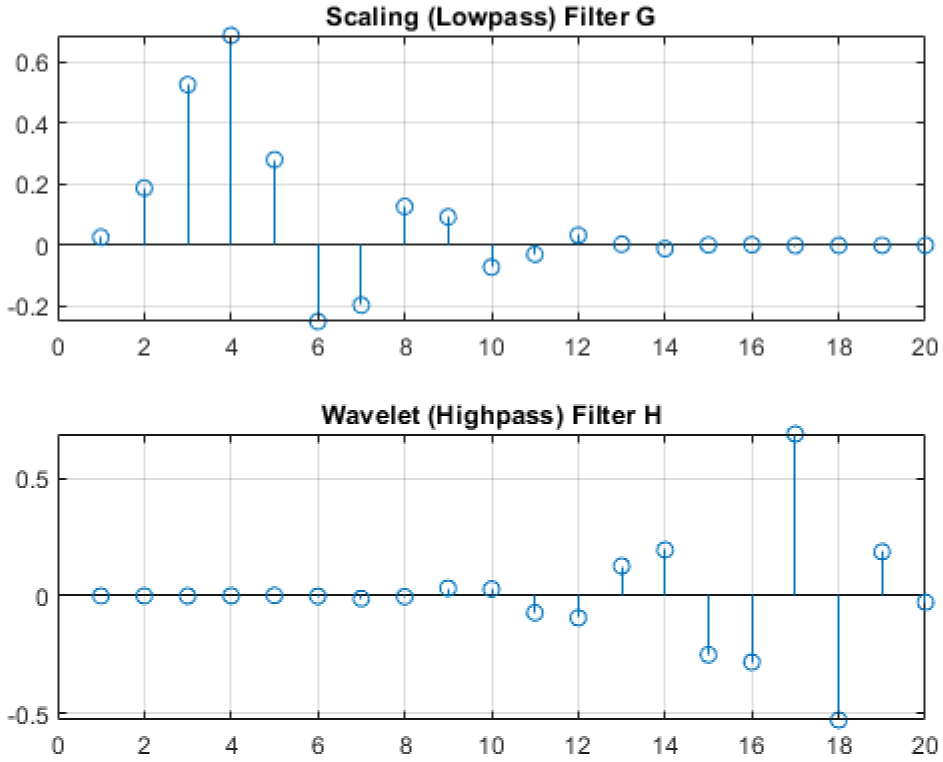
```
sF = dbwavf("db10");
```

`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
```



Scaling (Lowpass) Filter G

Wavelet (Highpass) Filter H

Save the current extension mode. Set the extension mode to Periodization. Generate a random signal of length 64. Perform a single-level wavelet decomposition of the signal using G and H. For purposes of reproducibility, set the random seed to the default value.

```
origmode = dwtmode("status","nodisplay");
dwtmode("per","nodisplay")
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)]
```

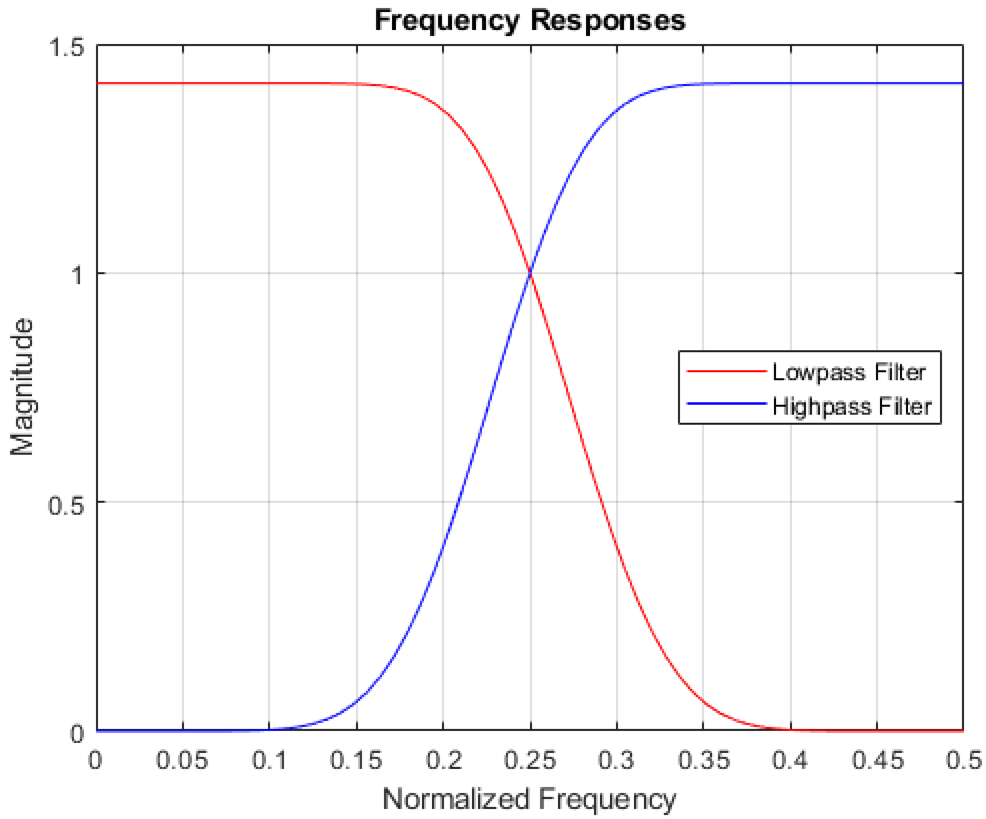
```
ans = 1×2

   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")
hold off
```



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 = 1×2

    2.0000    2.0000
```

Confirm that the filters are orthonormal.

```
df = [G;H];
id = df*df'
id = 2×2

    1.0000    0.0000
    0.0000    1.0000
```

Restore the original extension mode.

```
dwtmode(origmode,"nodisplay")
```

Controlling Phase of a Quadrature Mirror Filter

This example shows the effect of setting the phase parameter of the `qmf` function.

Obtain the decomposition lowpass filter associated with a Daubechies wavelet.

```
lowfilt = wfilters("db4");
```

Use the `qmf` function to obtain the decomposition lowpass 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)
```

```
p0 = 1×8

    0.2304   -0.7148    0.6309    0.0280   -0.1870   -0.0308    0.0329    0.0106
```

```
p1 = qmf(lowfilt,1)
```

```
p1 = 1×8

   -0.2304    0.7148   -0.6309   -0.0280    0.1870    0.0308   -0.0329   -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 = 1×8

     0     0     0     0     0     0     0     0
```

Input Arguments

X — Filter coefficients
vector

Filter coefficients, specified as a vector.

Data Types: `single` | `double`

P — Phase parameter
0 (default) | 1

Phase parameter, specified as follows.

- 0 — Change signs of even-indexed elements of the reversed vector X
- 1 — Change signs of odd-indexed elements of the reversed vector X

Data Types: single | double

More About

Quadrature Mirror Filters

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 is defined by $x_0 = F_0(x)$ and $y_0(n) = x_0(2n)$. 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 [1] 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:

$$|F_0(z)|^2 + |F_1(z)|^2 = 2.$$

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

wfilters | dwtfilterbank

Topics
“Add Quadrature Mirror and Biorthogonal Wavelet Filters”

Introduced before R2006a

qorthwavf

Kingsbury Q-shift filters

Syntax

```
[LoDa,LoDb,HiDa,HiDb,LoRa,LoRb,HiRa,HiRb] = qorthwavf(num)
```

Description

[LoDa,LoDb,HiDa,HiDb,LoRa,LoRb,HiRa,HiRb] = qorthwavf(num) returns the Kingsbury Q-shift filters for the Q-shift complex dual-tree transform. The integer num refers to the number of nonzero coefficients (taps) in the filter. Valid options for num are 6, 10, 14, 16, and 18. All filters are of even lengths and the tree B filters are the time reverse of the tree A filters.

Examples

Kingsbury Q-shift Filters

Obtain the Q-shift filters for the case with 10 nonzero coefficients.

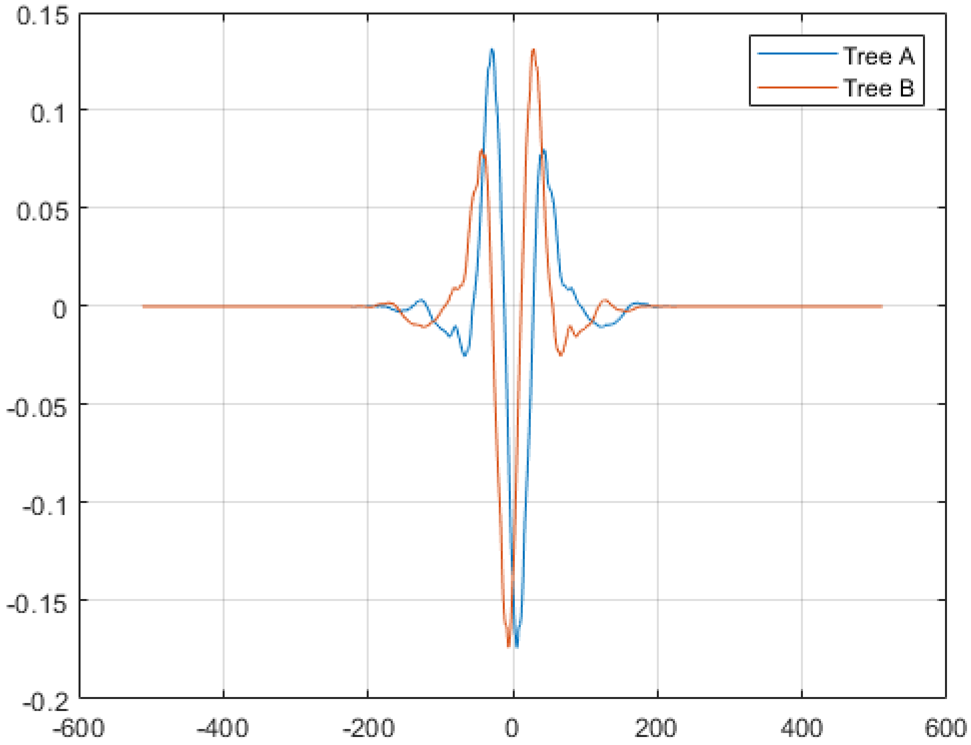
```
[LoDa,LoDb,HiDa,HiDb,LoRa,LoRb,HiRa,HiRb] = qorthwavf(10);
```

Use the dwtfilterbank function and create two discrete wavelet transform filter banks. Use the tree A analysis filters in the first filter bank, and the tree B analysis filters in the second filter bank.

```
fbTreeA = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',LoDa,...
    'CustomWaveletFilter',HiDa);
fbTreeB = dwtfilterbank('Wavelet','Custom',...
    'CustomScalingFilter',LoDb,...
    'CustomWaveletFilter',HiDb);
```

Plot the coarsest-scale wavelets of each filter bank.

```
[psiA,t] = wavelets(fbTreeA);
[psiB,~] = wavelets(fbTreeB);
plot(t,psiA(end,:))
hold on
plot(t,psiB(end,:))
grid on
hold off
legend('Tree A','Tree B')
```



Confirm both filter banks are orthogonal.

```
isOrthogonal(fbTreeA)
```

```
ans = logical
   1
```

```
isOrthogonal(fbTreeB)
```

```
ans = logical
   1
```

Input Arguments

num — Number of nonzero coefficients

6 | 10 | 14 | 16 | 18

Number of nonzero coefficients in the Kingsbury Q-shift filters, specified as one of the listed values.

Output Arguments

LoDa — Tree A lowpass analysis filter

real-valued vector

Tree A lowpass (scaling) analysis filter associated with the Q-shift filter, returned as a real-valued vector.

LoDb — Tree B lowpass analysis filter

real-valued vector

Tree B lowpass (scaling) analysis filter associated with the Q-shift filter, returned as a real-valued vector.

HiDa — Tree A highpass analysis filter

real-valued vector

Tree A highpass (wavelet) analysis filter associated with the Q-shift filter, returned as a real-valued vector.

HiDb — Tree B highpass analysis filter

real-valued vector

Tree B highpass (wavelet) analysis filter associated with the Q-shift filter, returned as a real-valued vector.

LoRa — Tree A lowpass synthesis filter

real-valued vector

Tree A lowpass (scaling) synthesis filter associated with the Q-shift filter, returned as a real-valued vector.

LoRb — Tree B lowpass synthesis filter

real-valued vector

Tree B lowpass (scaling) synthesis filter associated with the Q-shift filter, returned as a real-valued vector.

HiRa — Tree A highpass synthesis filter

real-valued vector

Tree A highpass (wavelet) synthesis filter associated with the Q-shift filter, returned as a real-valued vector.

HiRb — Tree B highpass synthesis filter

real-valued vector

Tree B highpass (wavelet) synthesis filter associated with the Q-shift filter, returned as a real-valued vector.

References

[1] Antonini, M., M. Barlaud, P. Mathieu, and I. Daubechies. “Image Coding Using Wavelet Transform.” *IEEE Transactions on Image Processing* 1, no. 2 (April 1992): 205–20. https://doi.org/10.1109/83.136597.

[2] Kingsbury, Nick. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Applied and Computational Harmonic Analysis* 10, no. 3 (May 2001): 234–53. https://doi.org/10.1006/acha.2000.0343.

[3] Le Gall, D., and A. Tabatabai. “Sub-Band Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques.” In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, 761–64. New York, NY, USA: IEEE, 1988. https://doi.org/10.1109/ICASSP.1988.196696.

Extended Capabilities

C/C++ Code Generation

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

See Also

`qbiorthfilt` | `dualtree3` | `dualtree2` | `dualtree`

Topics

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

Introduced in R2020a

rbiowavf

Reverse biorthogonal spline wavelet filters

Syntax

```
[RF,DF] = rbiowavf(wname)
```

Description

`[RF,DF] = rbiowavf(wname)` returns the reconstruction (synthesis) and decomposition (analysis) scaling filters, `RF` and `DF`, respectively, associated with the reverse biorthogonal wavelet specified by `wname`.

Examples

Reverse Biorthogonal Scaling Filter

Obtain the reverse biorthogonal reconstruction and decomposition scaling filters for the `'rbio3.1'` wavelet. The `'rbio3.1'` wavelet has three vanishing moments for the decomposition (analysis) wavelet and one 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
```

Input Arguments

wname — Name of reverse biorthogonal wavelet
character vector | string scalar

Name of reverse biorthogonal wavelet, specified as `'rbioNd.Nr'` where possible values for `Nd` and `Nr` are as follows:

`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

Nd and Nr are the numbers of vanishing moments for the decomposition and reconstruction filters, respectively.

Example: 'rbiowavf3.7'

Output Arguments

RF — Reconstruction filter

real-valued vector

Reconstruction filter associated with the reverse biorthogonal wavelet wname, returned as a real-valued vector.

DF — Decomposition filter

real-valued vector

Decomposition filter associated with the reverse biorthogonal wavelet wname, returned as a real-valued vector.

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', )
```

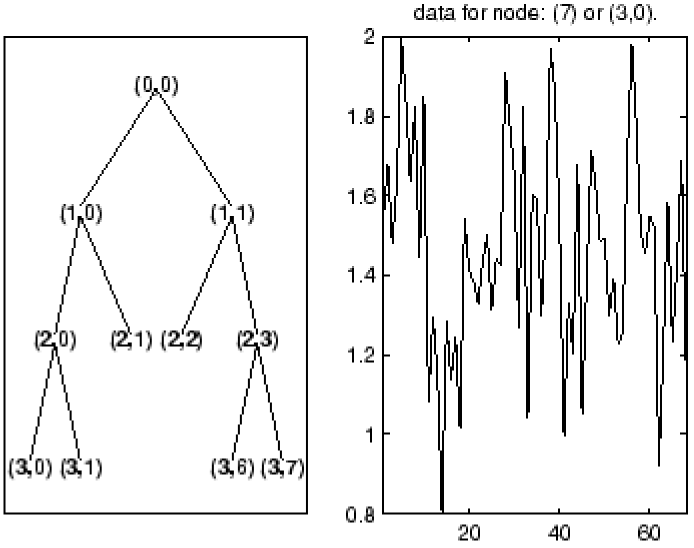
where `'PropParam'` is optional.

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`.
`'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

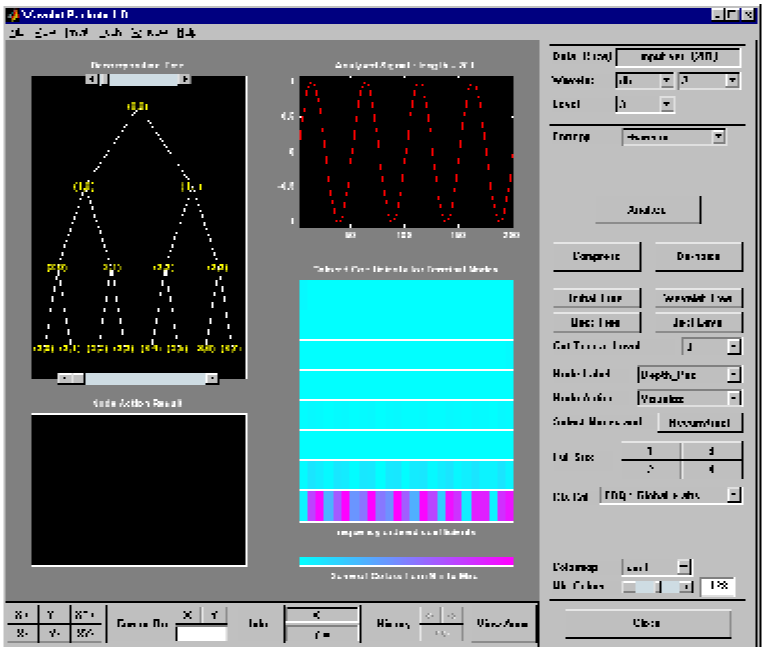
```
T = readtree(F)
```

Description

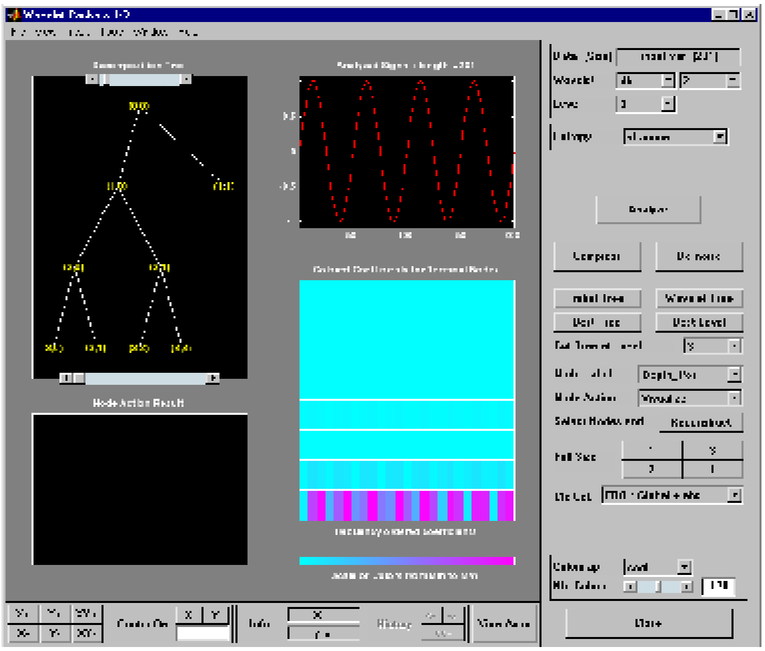
`T = readtree(F)` reads the wavelet packet decomposition tree from the figure whose handle is *F*.

Examples

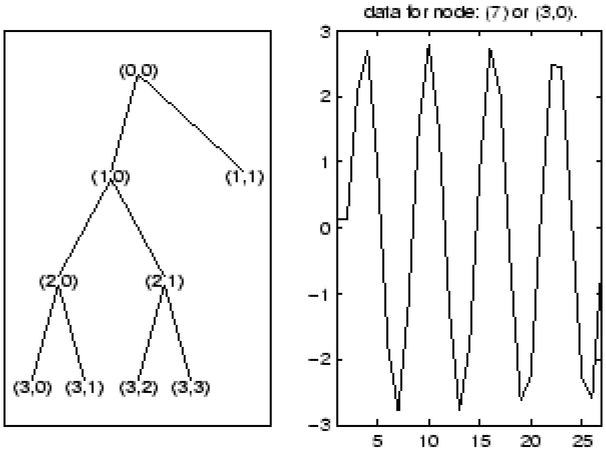
```
% Create a wavelet packet tree.
x   = sin(8*pi*[0:0.005:1]);
t   = wpdec(x,3,'db2');

% 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

reflect

Laurent polynomial or Laurent matrix reflection

Syntax

```
Q = reflect(P)
```

Description

`Q = reflect(P)` returns the reflection of the Laurent polynomial or the Laurent matrix specified by `P`. If `P` is a Laurent matrix, the function reflects the matrix elements.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `reflect`. The input data type determines which version is executed.

Examples

Laurent Polynomial Reflection

Create a Laurent polynomial $a(z) = 5z^6 + 4z^5 + 3z^4 + 2z^3$.

```
a = laurentPolynomial(Coefficients=[5 4 3 2],MaxOrder=6)
```

```
a = 
  laurentPolynomial with properties:

    Coefficients: [5 4 3 2]
        MaxOrder: 6
```

Obtain the reflection of $a(z)$. Confirm the maximum order of the reflection is –3.

```
b = reflect(a)
```

```
b = 
  laurentPolynomial with properties:

    Coefficients: [2 3 4 5]
        MaxOrder: -3
```

Laurent Matrix Reflection

Create two Laurent polynomials:

- $a(z) = -z^3 + 2z^2 - 3z + 4$

- $b(z) = 5z^2 - z - z^{-1} + z^{-2}$

```
lpA = laurentPolynomial(Coefficients=[-1 2 -3 4],MaxOrder=3);
lpB = laurentPolynomial(Coefficients=[5 -1 0 -1 1],MaxOrder=1);
```

Create the Laurent matrix $\begin{bmatrix} a(z) & 0 \\ 1 & b(z) \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={lpA,0;1,lpB});
```

Obtain the reflection of the matrix. Inspect the diagonal elements of the reflection.

```
lmatref = reflect(lmat);
lmatref.Elements{1,1}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [4 -3 2 -1]
        MaxOrder: 0
```

```
lmatref.Elements{2,2}
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: [1 -1 0 -1 5]
        MaxOrder: 3
```

Input Arguments

P — Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Output Arguments

Q — Reflection
`laurentPolynomial` object | `laurentMatrix` object

Reflection of a Laurent polynomial or a Laurent matrix, returned as a `laurentPolynomial` object or a `laurentMatrix` object. The reflection of a Laurent polynomial $P(z)$ is the Laurent polynomial $Q(z) = P(1/z)$.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
uminus

Objects
laurentMatrix | laurentPolynomial

Introduced in R2021b

removeLabelDefinition

Remove label definition from labeled signal set

Syntax

```
removeLabelDefinition(lss,lblname)
```

Description

`removeLabelDefinition(lss,lblname)` removes the label definition `lblname` from the labeled signal set `lss`. If you want to remove a sublabel, specify `lblname` as a two-element string array or two-element cell array of character vectors:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Examples

Remove Label Definition

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Retrieve a hierarchical list of labels and sublabels.

```
labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: []
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: TrillPeaks
```

```
      '
```

Remove the sublabel that labels peaks in the trill regions.

```
removeLabelDefinition(lss,{'TrillRegions' 'TrillPeaks'})
```

```
labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: []
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: []
      '
```

Remove the label that specifies the whale type.

```
removeLabelDefinition(lss,"WhaleType")
```

```
getLabelNames(lss)
```

```
ans = 2x1 string
    "MoanRegions"
    "TrillRegions"
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

removeMembers

Remove members from labeled signal set

Syntax

```
removeMembers(lss,midxvect)
```

Description

`removeMembers(lss,midxvect)` removes the members specified in `midxvect` from the labeled signal set `lss`.

Examples

Remove Member

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Remove the second member of the set.

```
removeMembers(lss,2)
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {[79572x1 double]}
         NumMembers: 1
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [1x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
Use setLabelValue to add data to the set.
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midxvect` — Subset member row numbers

vector of positive integers

Subset member row numbers, specified as a vector of positive integers. Each element of `midxvect` specifies a member row number as it appears in the “Labels” on page 1-0 table of the `labeledSignalSet` object `lss`.

Example: `[2 3 5 7 11 13 17]` chooses a subset of signals indexed by prime numbers.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

removePointValue

Remove row from point label

Syntax

```
removePointValue(lss,midx,lblname)
removePointValue(lss,midx,lblname,'LabelRowIndex',ridx)
removePointValue(lss,midx,lblname,'SublabelRowIndex',sridx)
removePointValue(lss,midx,lblname,'LabelRowIndex',ridx,'SublabelRowIndex',
sridx)
```

Description

`removePointValue(lss,midx,lblname)` removes all rows of the point label `lblname` for the member specified by `midx`.

- If `lblname` is a character vector or a string scalar, the function targets a parent label.
- If `lblname` is a two-element string array or a two-element cell array of character vectors, the function:
  - Interprets the first element as the name of a parent label.
  - Interprets the second element as the sublabel name of a point label.
  - Removes all the points of the sublabel.

`removePointValue(lss,midx,lblname,'LabelRowIndex',ridx)` removes a row, specified by `ridx`, of the point label `lblname` for the member `midx`.

If `lblname` is a two-element string array or a two-element cell array of character vectors, the function:

- Interprets the first element as the name of a parent label.
- Interprets the second element as the sublabel name of a point label.
- Removes all the points of the sublabel contained in row `ridx`.

`removePointValue(lss,midx,lblname,'SublabelRowIndex',sridx)` removes the sublabel row specified by `sridx`. In this case, `lblname` must be a two-element string array or a two-element cell array of character vectors:

- The first element is the name of a parent attribute label.
- The second element is the sublabel name of a point label.

`removePointValue(lss,midx,lblname,'LabelRowIndex',ridx,'SublabelRowIndex',sridx)` removes the sublabel row specified by `sridx` of the ROI or point label row specified by `ridx`. In this case, `lblname` must be a two-element string array or a two-element cell array of character vectors:

- The first element is the name of a parent ROI or point label.
- The second element is the sublabel name of a point label.

Examples

Remove Point Value

Load a labeled signal set containing recordings of whale songs. Get the names of the labels and the number of members.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

```
nm = lss.NumMembers;
```

Define a point label associated with the signal maximum.

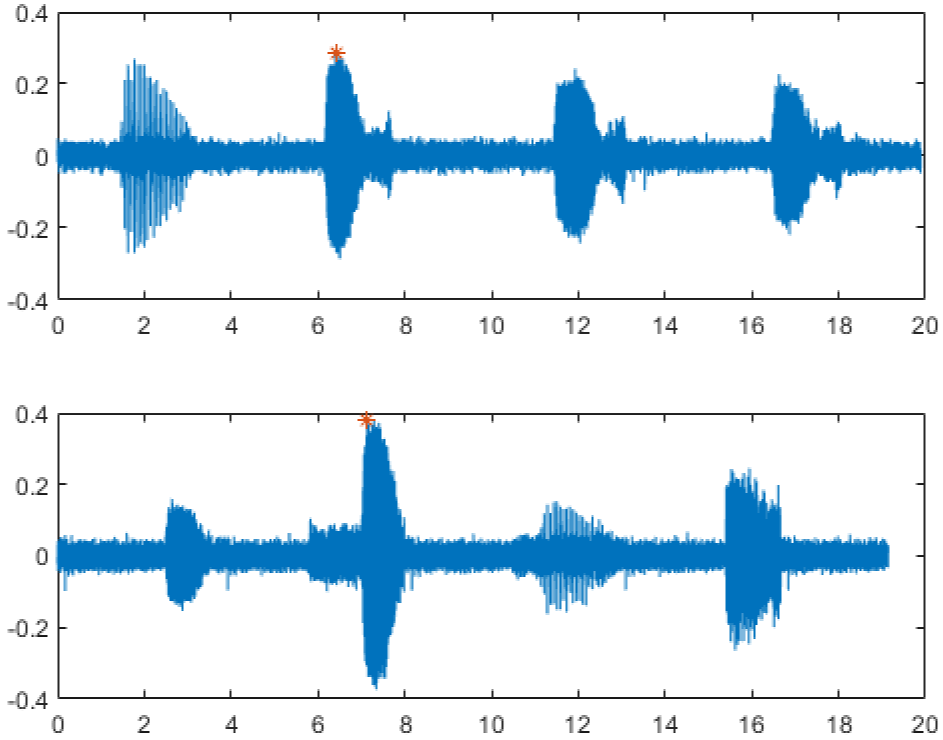
```
themax = signalLabelDefinition('Maximum','LabelType','point', ...
    'LabelDataType','numeric')
```

```
themax = 
  signalLabelDefinition with properties:

                      Name: "Maximum"
                 LabelType: "point"
             LabelDataType: "numeric"
        ValidationFunction: []
    PointLocationsDataType: "double"
              DefaultValue: []
                 Sublabels: [0x0 signalLabelDefinition]
                       Tag: ""
               Description: ""

 Use labeledSignalSet to create a labeled signal set.
```

```
addLabelDefinitions(lss,themax)
```

Find the maxima of the signals and add their values to the labeled set.

```
figure
for idx = 1:nm
    sg = getSignal(lss,idx);
    [mx,ix] = max(sg);
    setLabelValue(lss,idx,'Maximum',ix,mx)

    subplot(nm,1,idx)
    plot((0:length(sg)-1)/lss.SampleRate,sg,ix/lss.SampleRate,mx,'*')
end
```



Verify that the set includes the new point label.

```
getLabelValues(lss)
```

```
ans=2×4 table
                 WhaleType    MoanRegions    TrillRegions      Maximum
                 _________    ___________    ____________    ___________

    Member{1}      blue       {3x2 table}    {1x3 table}     {1x2 table}
    Member{2}      blue       {3x2 table}    {1x3 table}     {1x2 table}
```

Remove the `'Maximum'` value for the first member of the set. Verify that the label is empty for the first member.

```
removePointValue(lss,1,'Maximum')
```

```
getLabelValues(lss,1)
```

```
ans=1×4 table
                 WhaleType    MoanRegions    TrillRegions      Maximum
                 _________    ___________    ____________    ___________

    Member{1}      blue       {3x2 table}    {1x3 table}     {0x2 table}
```

Input Arguments

`lss` — Labeled signal set
`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midx` — Member row number
positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

`lblname` — Label or sublabel name
character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`ridx` — Label row index
positive integer

Label row index, specified as a positive integer. This argument applies only for ROI and point labels.

`sridx` — Sublabel row index
positive integer

Sublabel row index, specified as a positive integer. This argument applies only when a label and sublabel pair has been specified in `lblname` and the sublabel is of type ROI or point.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

removeRegionValue

Remove row from ROI label

Syntax

```
removeRegionValue(lss,midx,lblname)
removeRegionValue(lss,midx,lblname,'LabelRowIndex',ridx)
removeRegionValue(lss,midx,lblname,'SublabelRowIndex',sridx)
removeRegionValue(lss,midx,lblname,'LabelRowIndex',ridx,'SublabelRowIndex',
sridx)
```

Description

`removeRegionValue(lss,midx,lblname)` removes all rows of the ROI label `lblname` for the member specified by `midx`.

- If `lblname` is a character vector or a string scalar, the function targets a parent label.
- If `lblname` is a two-element string array or a two-element cell array of character vectors, the function:
  - Interprets the first element as the name of a parent label.
  - Interprets the second element as the sublabel name of an ROI label.
  - Removes all the regions of the sublabel.

`removeRegionValue(lss,midx,lblname,'LabelRowIndex',ridx)` removes a row, specified by `ridx`, of the ROI label `lblname` for the member `midx`.

If `lblname` is a two-element string array or a two-element cell array of character vectors, the function:

- Interprets the first element as the name of a parent label.
- Interprets the second element as the sublabel name of an ROI label.
- Removes all the regions of the sublabel contained in row `ridx`.

`removeRegionValue(lss,midx,lblname,'SublabelRowIndex',sridx)` removes the sublabel row specified by `sridx`. In this case, `lblname` must be a two-element string array or a two-element cell array of character vectors:

- The first element is the name of a parent attribute label.
- The second element is the sublabel name of an ROI label.

`removeRegionValue(lss,midx,lblname,'LabelRowIndex',ridx,'SublabelRowIndex',sridx)` removes the sublabel row specified by `sridx` of the ROI or point label row specified by `ridx`. In this case, `lblname` must be a two-element string array or a two-element cell array of character vectors:

- The first element is the name of a parent ROI or point label.
- The second element is the sublabel name of an ROI label.

Examples

Remove Region Value

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Get the names and values of the labels in the set. For the following, concentrate on the second member of the set.

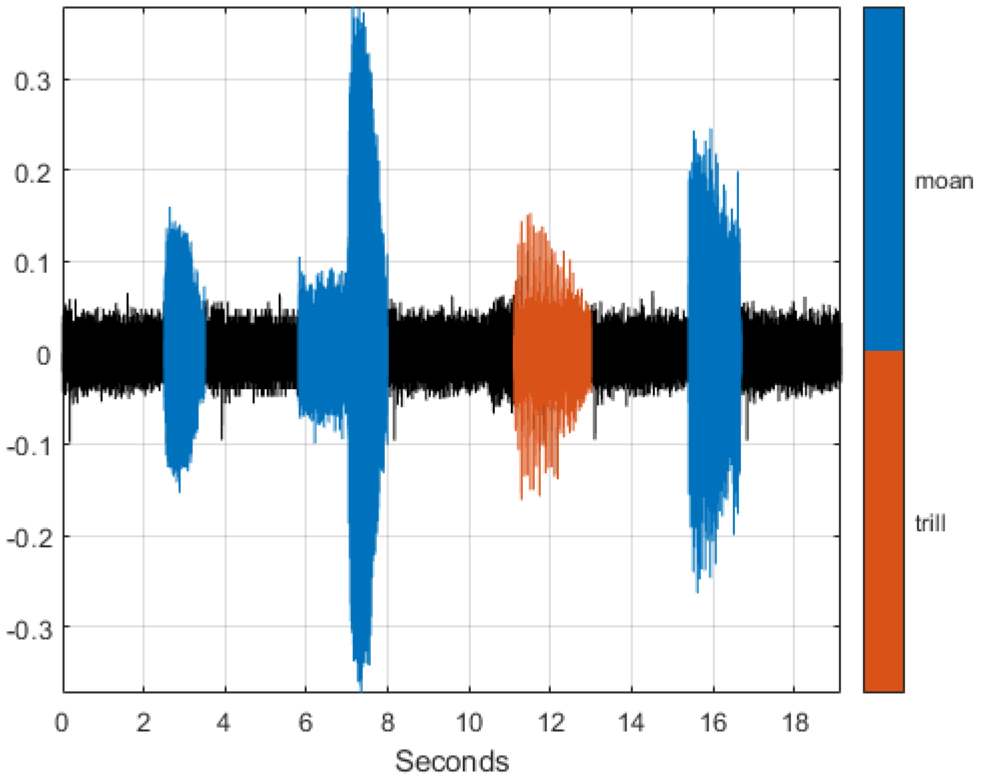
```
lbldefs = getLabelValues(lss)
```

```
lbldefs=2×3 table
                 WhaleType    MoanRegions    TrillRegions
                 _________    ___________    ____________

    Member{1}      blue       {3x2 table}    {1x3 table} 
    Member{2}      blue       {3x2 table}    {1x3 table} 
```

```
idx = 2;
```

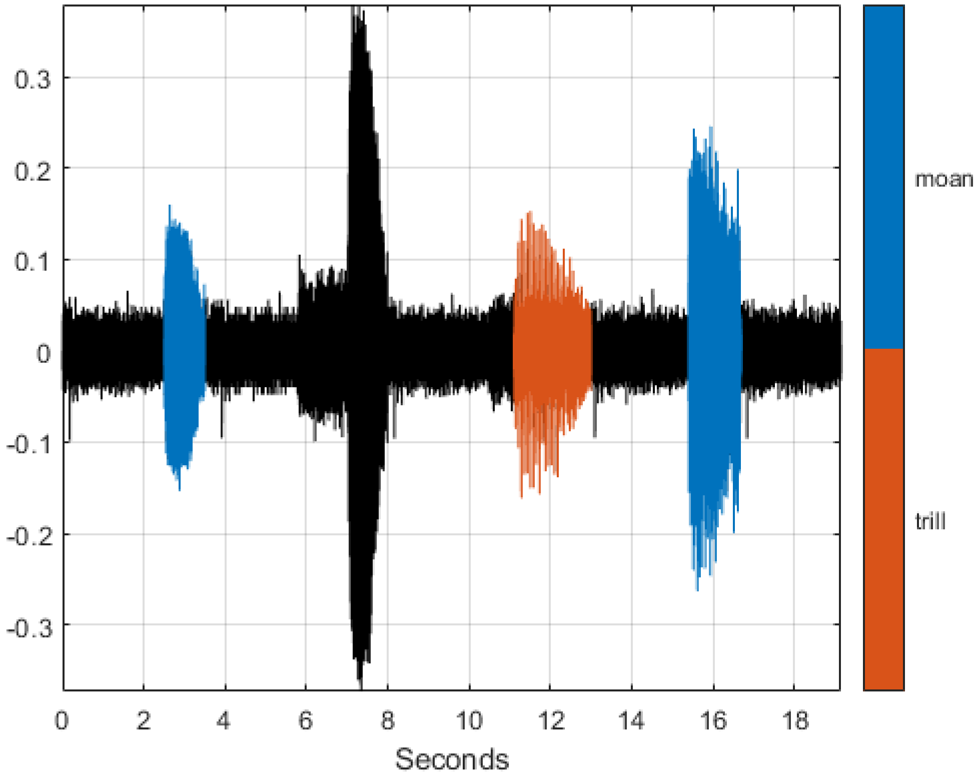
Retrieve the moan and trill regions. Use a `signalMask` (Signal Processing Toolbox) object to plot the signal and highlight the moans and trills.

```
mvals = getLabelValues(lss,idx,'MoanRegions');
tvals = getLabelValues(lss,idx,'TrillRegions');

tb = [mvals;tvals];
tb.Value = categorical( ...
    [repmat("moan",height(mvals),1);repmat("trill",height(tvals),1)], ...
    ["moan" "trill"]);

sm = signalMask(tb,"SampleRate",lss.SampleRate);

plotsigroi(sm,getSignal(lss,idx))
```



Remove the second moan from the labels. Plot the signal again. Highlight the moans and trills.

```
removeRegionValue(lss,idx,'MoanRegions','LabelRowIndex',2)

mvals = getLabelValues(lss,idx,'MoanRegions');

tb = [mvals;tvals];
tb.Value = categorical( ...
    [repmat("moan",height(mvals),1);repmat("trill",height(tvals),1)], ...
    ["moan" "trill"]);

sm = signalMask(tb,"SampleRate",lss.SampleRate);

plotsigroi(sm,getSignal(lss,idx))
```



Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midx` — Member row number

positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the "Labels" on page 1-0 table of a labeled signal set.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.

- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`ridx` — Label row index
positive integer

Label row index, specified as a positive integer. This argument applies only for ROI and point labels.

`sridx` — Sublabel row index
positive integer

Sublabel row index, specified as a positive integer. This argument applies only when a label and sublabel pair has been specified in `lblname` and the sublabel is of type ROI or point.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

rescale

Rescale Laurent polynomial

Syntax

```
Q = rescale(P,c)
```

Description

`Q = rescale(P,c)` scales the coefficients of the Laurent polynomial `P` by the nonzero scalar `c`.

Examples

Rescale Laurent Polynomial

Create the Laurent polynomial $a(z) = 4z + 6 + 10z^{-1} + 14z^{-2} + 22z^{-3} + 26z^{-4}$.

```
a = laurentPolynomial(Coefficients=[4 6 10 14 22 26],MaxOrder=1);
```

Divide the coefficients of $a(z)$ by 2.

```
b = rescale(a,1/2)
```

```
b = 
  laurentPolynomial with properties:

    Coefficients: [2 3 5 7 11 13]
        MaxOrder: 1
```

Input Arguments

P — Laurent polynomial
`laurentPolynomial` object

Laurent polynomial, specified as a `laurentPolynomial` object.

c — Scale factor
nonzero scalar

Scale factor, specified as a nonzero scalar.

Example: `Q = rescale(P,5)` multiplies the coefficients of `P` by 5.

Data Types: `double`

Output Arguments

Q — Scaled Laurent polynomial
`laurentPolynmial` object

Scaled Laurent polynomial, returned as a `laurentPolynmial` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

resetLabelValues

Reset labels to default values

Syntax

```
resetLabelValues(lss)
resetLabelValues(lss,midx)

resetLabelValues(lss,midx,lblname)
resetLabelValues( ___ ,'LabelRowIndex',ridx)
```

Description

`resetLabelValues(lss)` resets all label values for all members of the labeled signal set `lss`.

`resetLabelValues(lss,midx)` resets all label values for the signals in the member specified by `midx`.

`resetLabelValues(lss,midx,lblname)` resets the values of label `lblname` for the signals in the member specified by `midx`. To reset a sublabel, make `lblname` a two-element string array or a two-element cell array of character vectors, with the first element containing the parent label name and the second element containing the sublabel name.

By default, the function resets all sublabels of a parent label. To target a sublabel of an ROI or point parent label, specify the parent label row index using `ridx`.

`resetLabelValues( ___ ,'LabelRowIndex',ridx)` specifies the row index of the ROI or point parent label for which you want to reset a sublabel value.

Examples

Reset Label Values

Load a labeled signal set containing recordings of whale songs. Get the names of the labels.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

```
getLabelNames(lss)

ans = 3x1 string
    "WhaleType"
    "MoanRegions"
    "TrillRegions"
```

Get the label values corresponding to the trill regions for the second signal in the set.

```
idx = 2;
getLabelValues(lss,idx,'TrillRegions')

ans=1×2 table
     ROILimits      Value
    ____________    _____

    11.1      13    {[1]}
```

Reset the values. Verify that `'TrillRegions'` becomes an empty array.

```
resetLabelValues(lss,idx,'TrillRegions')

getLabelValues(lss,idx,'TrillRegions')

ans =

  0x2 empty table

getLabelValues(lss,idx)

ans=1×3 table
                 WhaleType     MoanRegions     TrillRegions
                 _________    ___________     ____________

    Member{2}      blue        {3x2 table}     {0x3 table}
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midx` — Member row number

positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`ridx` — Label row index

positive integer

Label row index, specified as a positive integer. This argument applies only for ROI and point labels.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

scal2frq

Scale to frequency

Syntax

```
frq = scal2frq(A,wname,delta)
frq = scal2frq(A,wname)
```

Description

`frq = scal2frq(A,wname,delta)` returns the pseudo-frequencies corresponding to the scales given by `A` and the wavelet specified by `wname` and the sampling period `delta`. The output `frq` is real-valued and has the same dimensions as `A`.

`frq = scal2frq(A,wname)` is equivalent to `frq = scal2frq(A,wname,1)`.

Examples

Scales and Pseudo-Frequencies

This example shows how the pseudo-frequency changes as you double the scale.

Construct a vector of scales with 10 voices per octave over five octaves.

```
vpo = 10;
no = 5;
a0 = 2^(1/vpo);
ind = 0:vpo*no;
sc = a0.^ind;
```

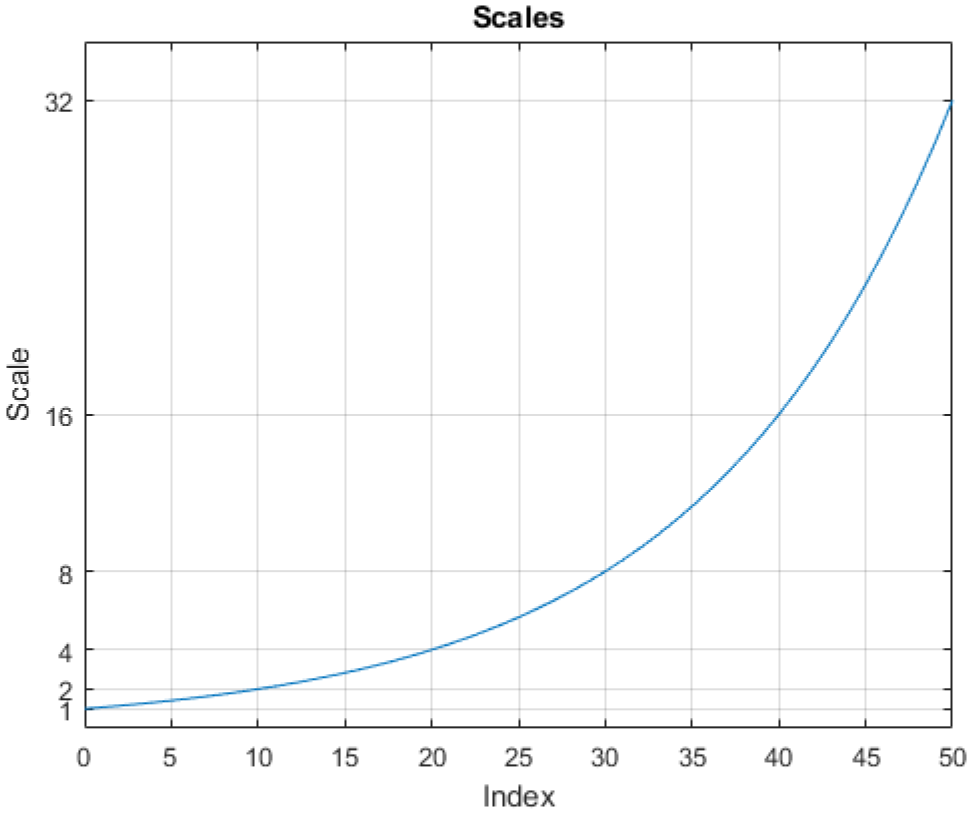
Verify that the range of scales covers five octaves.

```
log2(max(sc)/min(sc))
```

```
ans = 5.0000
```

If you plot the scales, you can use a data cursor to confirm that the scale at index $n + 10$ is twice the scale at index n. Set the y-ticks to mark each octave.

```
plot(ind,sc)
title('Scales')
xlabel('Index')
ylabel('Scale')
grid on
set(gca,'YTick',2.^(0:5))
```



Convert the scales to pseudo-frequencies for the real-valued Morlet wavelet. First, assume the sampling period is 1.

```
pf = scal2frq(sc,"morl");
T = [sc(:) pf(:)];
T = array2table(T,'VariableNames',{'Scale','Pseudo-Frequency'});
disp(T)
```

```
    Scale     Pseudo-Frequency
    ______    ________________

         1         0.8125     
    1.0718        0.75809     
    1.1487        0.70732     
    1.2311        0.65996     
    1.3195        0.61576     
    1.4142        0.57452     
    1.5157        0.53605     
    1.6245        0.50015     
    1.7411        0.46666     
    1.8661        0.43541     
         2        0.40625     
    2.1435        0.37904     
    2.2974        0.35366     
    2.4623        0.32998     
     2.639        0.30788     
    2.8284        0.28726     
    3.0314        0.26803     
     3.249         0.25008
    3.4822         0.23333
    3.7321          0.2177
         4         0.20313
    4.2871         0.18952
    4.5948         0.17683
    4.9246         0.16499
     5.278         0.15394
    5.6569         0.14363
    6.0629         0.13401
     6.498         0.12504
    6.9644         0.11666
    7.4643         0.10885
         8         0.10156
    8.5742        0.094761
    9.1896        0.088415
    9.8492        0.082494
    10.556         0.07697
    11.314        0.071816
    12.126        0.067006
    12.996        0.062519
    13.929        0.058332
    14.929        0.054426
        16        0.050781
    17.148        0.047381
    18.379        0.044208
    19.698        0.041247
    21.112        0.038485
    22.627        0.035908
    24.251        0.033503
    25.992         0.03126
    27.858        0.029166
    29.857        0.027213
        32        0.025391
```

Assume that data is sampled at 100 Hz. Construct a table with the scales, the corresponding pseudo-frequencies, and periods. Since there are 10 voices per octave, display every tenth row in the table. Observe that for each doubling of the scale, the pseudo-frequency is cut in half.

```
Fs = 100;
DT = 1/Fs;
pf = scal2frq(sc,"morl",DT);
T = [sc(:)/Fs pf(:) 1./pf(:)];
T = array2table(T,'VariableNames',{'Scale','Pseudo-Frequency','Period'});
T(1:vpo:end,:)
```

```
ans=6×3 table
    Scale    Pseudo-Frequency     Period 
    _____    ________________    ________

    0.01           81.25         0.012308
    0.02          40.625         0.024615
    0.04          20.313         0.049231
    0.08          10.156         0.098462
    0.16          5.0781          0.19692
    0.32          2.5391          0.39385
```

Note the presence of the $\Delta t = \frac{1}{\mathrm{Fs}}$ factor in `scal2frq`. This is necessary in order to achieve the proper scale-to-frequency conversion. The Δt is needed to adjust the raw scales properly. For example, with:

```
f = scal2frq(1,'morl',0.01);
```

You are really asking what happens to the center frequency of the mother Morlet wavelet, if you dilate the wavelet by 0.01. In other words, what is the effect on the center frequency if instead of $\psi(t)$, you look at $\psi(t/0.01)$. The Δt provides the correct adjustment factor on the scales.

You could have obtained the same results by first converting the scales to their adjusted sizes and then using `scal2frq without specifying` Δt.

```
scadjusted = sc.*0.01;
pf2 = scal2frq(scadjusted,'morl');
max(pf-pf2)
```

```
ans = 0
```

Plot CWT with Frequencies in a Contour Plot

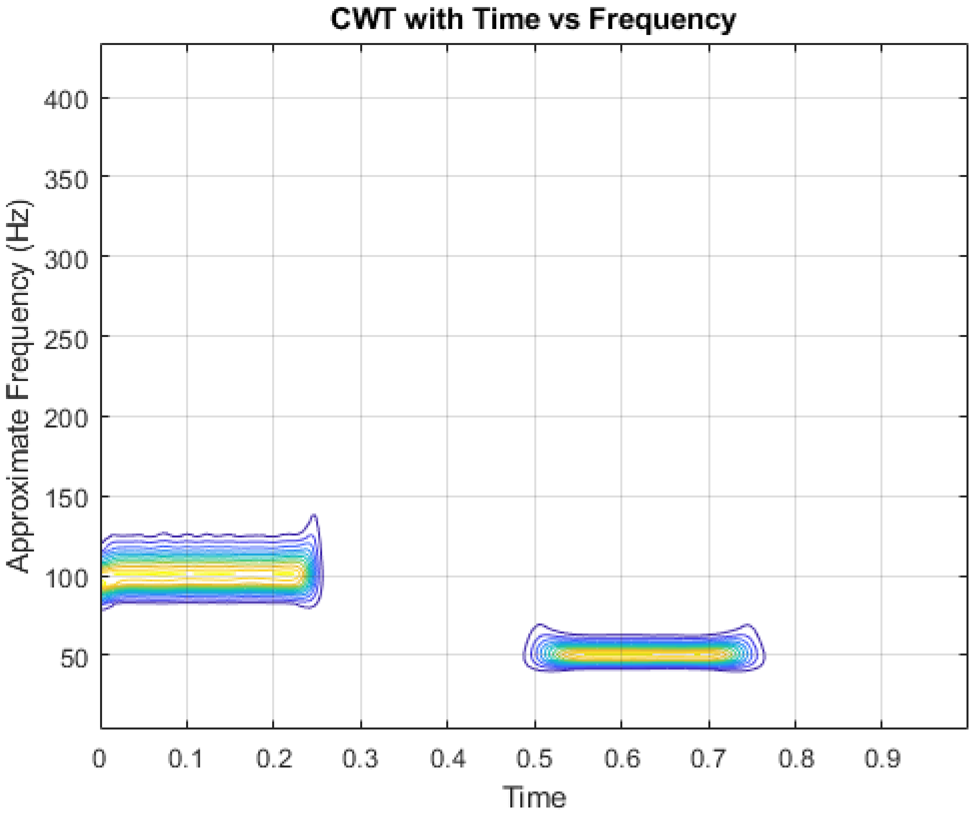
The example shows how to create a contour plot of the CWT using approximate frequencies in Hz.

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');
```



Input Arguments

A — Scales

positive real-valued vector

Scales, specified as a positive real-valued vector.

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. See `wavefun` for more information.

delta — Sampling period

1 (default) | positive real-valued scalar

Sampling period, specified as a real-valued scalar.

Example: `pf = scal2frq([1:5],"db4",0.01)`

More About

Pseudo-Frequencies

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:

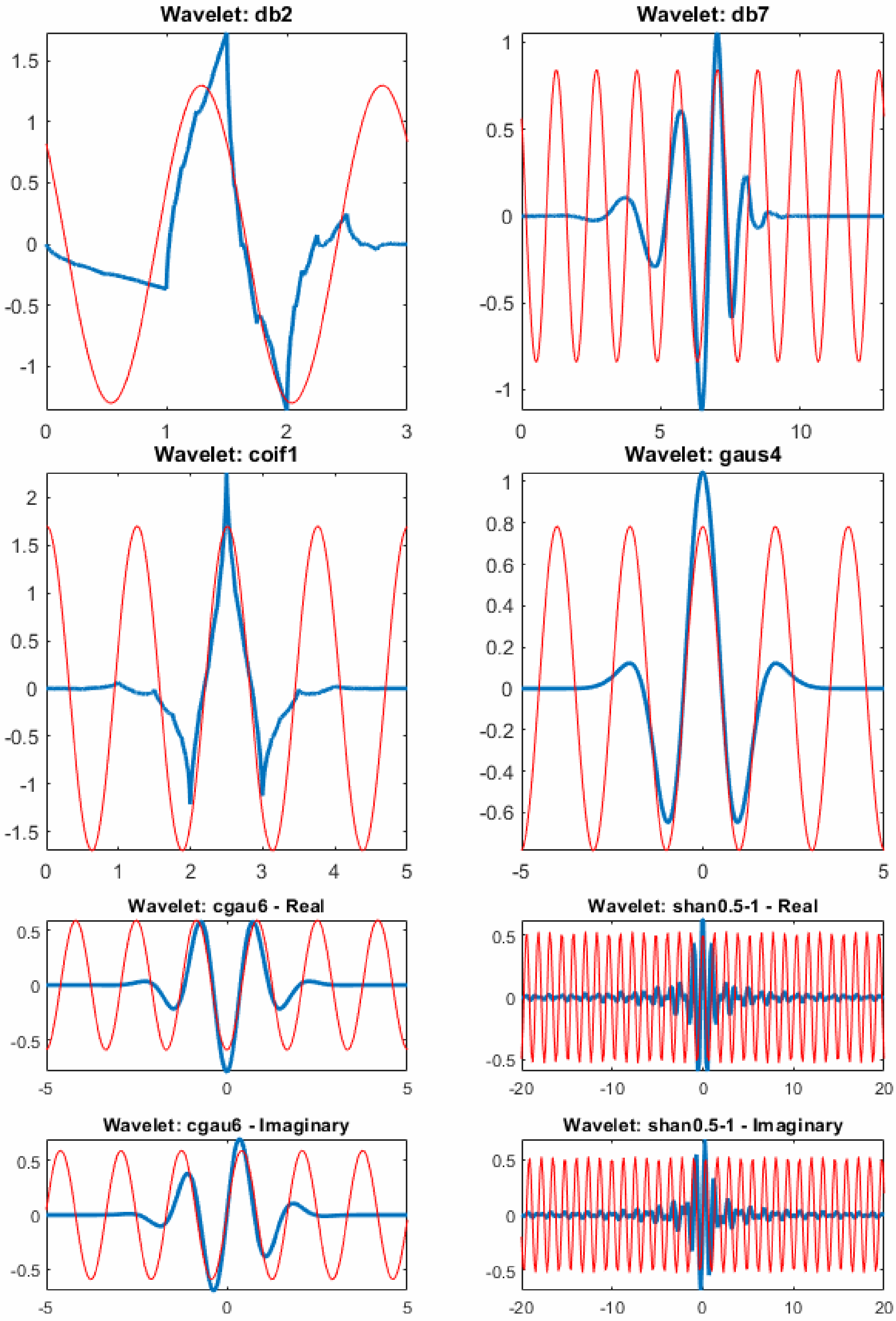
$$F_a = \frac{F_c}{a}$$

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. The `centfrq` function 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.


Wavelet: db2
Wavelet: db7
Wavelet: coif1
Wavelet: gaus4
Wavelet: cgau6 - Real
Wavelet: shan0.5-1 - Real
Wavelet: cgau6 - Imaginary
Wavelet: shan0.5-1 - Imaginary

As you can see, the center frequency-based approximation (red) captures the main wavelet oscillations (blue). The center frequency is a convenient and simple characterization of the dominant frequency of the wavelet.

References

[1] Abry, P. *Ondelettes et turbulence. Multirésolutions, algorithmes de décomposition, invariance d'échelles et signaux de pression*. Diderot, Editeurs des sciences et des arts, Paris, 1997.

See Also

`centfrq`

Introduced before R2006a

scales

CWT filter bank scales

Syntax

```
rs = scales(fb)
[rs,cs] = scales(fb)
```

Description

`rs = scales(fb)` returns the raw (unitless) scales used in creating the wavelet bandpass filters. Scales are ordered from finest to coarsest.

`[rs,cs] = scales(fb)` returns the wavelet scales converted to units of the sampling frequency or sampling period.

Examples

CWT Filter Bank Scales

Create a CWT filter bank with sampling period equal to 0.001 seconds.

```
fb = cwtfilterbank('SamplingPeriod',seconds(0.001));
```

Obtain the raw and converted scales used in creating the wavelet bandpass filters.

```
[rs,cs] = scales(fb);
```

Obtain the filter bank bandpass center periods.

```
P = centerPeriods(fb);
```

Compare the finest converted scale with the smallest bandpass center period normalized by the sampling period.

```
min(cs)
```

```
ans = 2.3035
```

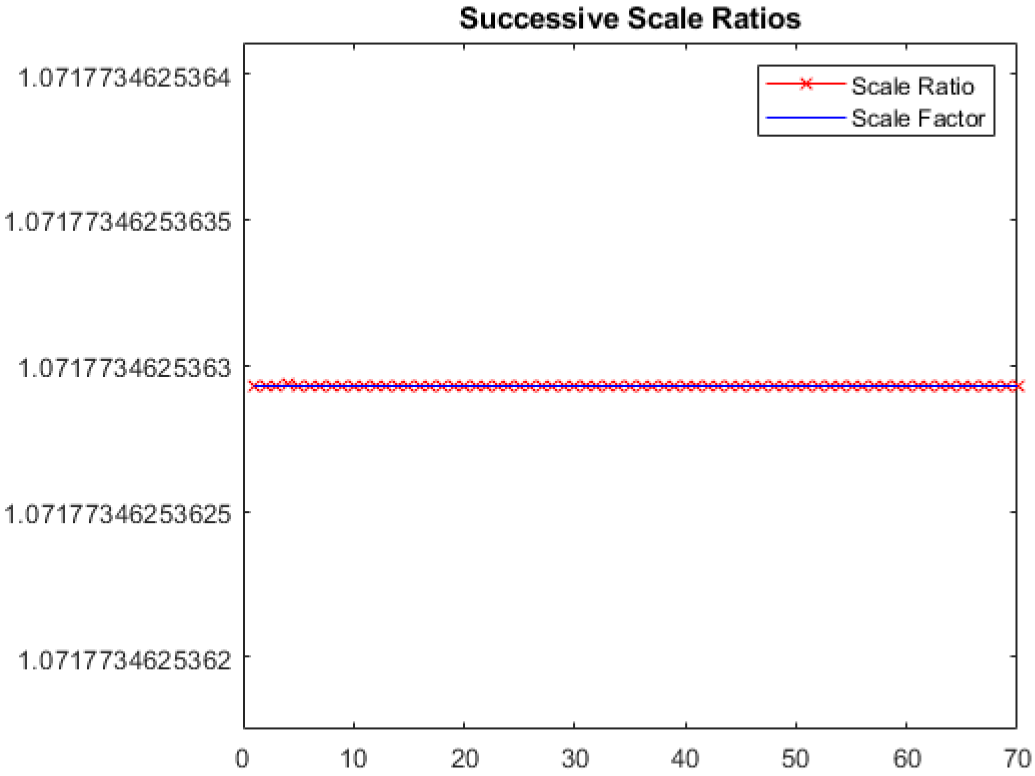
```
min(P)/seconds(0.001)
```

```
ans = 2.3035
```

The scales should increase by a factor of approximately $2^{1/|NV|}$, where NV is the number of voices per octave. The default value of NV is 10. Plot the ratios of successive scales and compare with $2^{1/10}$.

```
len = length(rs);
plot(rs(2:len)./rs(1:len-1),'rx-')
hold on
plot(1:len-1,2^(1/10)*ones(1,len-1),'b')
```

```
title('Successive Scale Ratios')
legend('Scale Ratio','Scale Factor')
```



Input Arguments

fb — Continuous wavelet transform filter bank
`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

rs — Raw scales
real-valued vector

Raw scales used in creating the wavelet bandpass filters, returned as a real-valued vector of length *Ns*, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales).

Data Types: `double`

cs — Converted scales
real-valued vector

Converted scales used in creating the wavelet bandpass filters, returned as a real-valued vector of length *Ns*, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales). `cs` is in units of the sampling frequency or sampling period.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`cwtfilterbank` | `centerPeriods` | `centerFrequencies`

Introduced in R2018a

scaleSpectrum

Scale-averaged wavelet spectrum

Syntax

```
savgp = scaleSpectrum(fb,x)
savgp = scaleSpectrum(fb,cfs)
[savgp,scidx] = scaleSpectrum( ___ )

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

scaleSpectrum( ___ )
```

Description

`savgp = scaleSpectrum(fb,x)` returns the scale-averaged wavelet power spectrum of the signal `x` using the CWT filter bank `fb`. By default, `savgp` is obtained by scale-averaging the magnitude-squared scalogram over all scales.

`savgp = scaleSpectrum(fb,cfs)` returns the scale-averaged wavelet spectrum for the CWT coefficients `cfs`.

Note When using this syntax, the power of the scale-averaged wavelet spectrum is normalized to equal the variance of the last signal processed by the filter bank object function `wt`.

`[savgp,scidx] = scaleSpectrum( ___ )` also returns the scale indices over which the scale-averaged wavelet spectrum is computed. If you do not specify `FrequencyLimits` or `PeriodLimits`, `scidx` is a vector from 1 to the number of scales.

`[ ___ ] = scaleSpectrum( ___ ,Name,Value)` specifies additional options using name-value pair arguments. These arguments can be added to any of the previous input syntaxes. For example, `'Normalization','none'` specifies no normalization of the scale-averaged wavelet spectrum.

`scaleSpectrum( ___ )` with no output arguments plots the scale-averaged wavelet power spectrum in the current figure.

Examples

Scale-Averaged Wavelet Spectrum

Load an audio file containing a fragment of Handel's "Hallelujah Chorus" sampled at 8192 Hz.

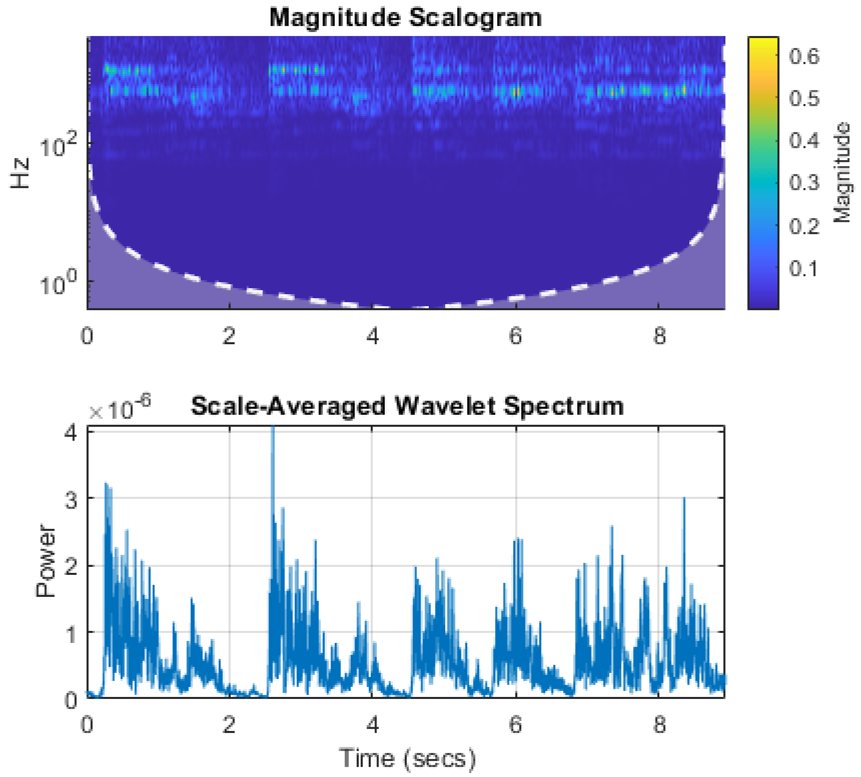
```
load handel         % To hear, type soundsc(y,Fs)
```

Create a CWT filter bank that can be applied to the signal. Use the default Morse wavelet.

```
fb = cwtfilterbank('SignalLength',length(y),'SamplingFrequency',Fs);
```

Plot the scalogram and scale-averaged wavelet power spectrum using the default settings.

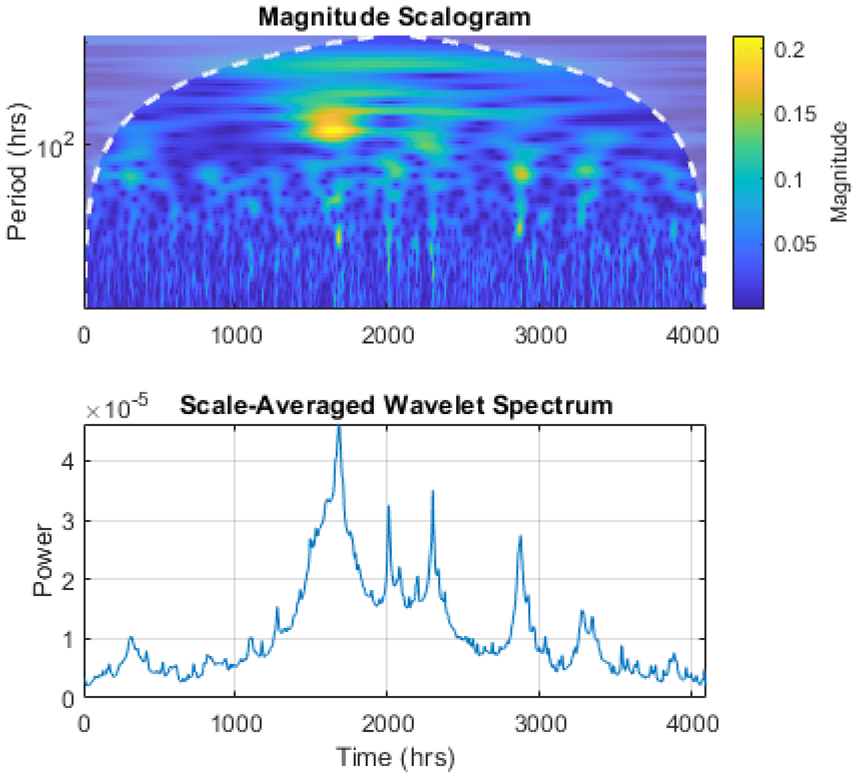
```
scaleSpectrum(fb,y)
```



Normalize Scale-Averaged Wavelet Spectrum

Load a time series of solar magnetic field magnitudes recorded hourly over the south pole of the sun by the Ulysses spacecraft from 21:00 UT on December 4, 1993 to 12:00 UT on May 24, 1994. See [2] pp. 218–220 for a complete description of this data. Create a CWT filter bank that can be applied to the data. Plot the scalogram and the scale-averaged wavelet spectrum.

```
load solarMFmagnitudes
fb = cwtfilterbank('SignalLength',length(sm),'SamplingPeriod',hours(1));
scaleSpectrum(fb,sm)
```



Obtain the scale-averaged wavelet spectrum of the signal using default values. By default, `scaleSpectrum` normalizes the power of the scale-averaged wavelet spectrum to equal the variance of the signal. Verify that the sum of the spectrum equals the variance of the signal.

```
savg = scaleSpectrum(fb,sm);
[var(sm) sum(savg)]
```

```
ans = 1×2

    0.0448    0.0447
```

Obtain the scale-averaged wavelet spectrum of the signal, but instead normalize the power as a probability density function. Verify that the sum is equal to 1.

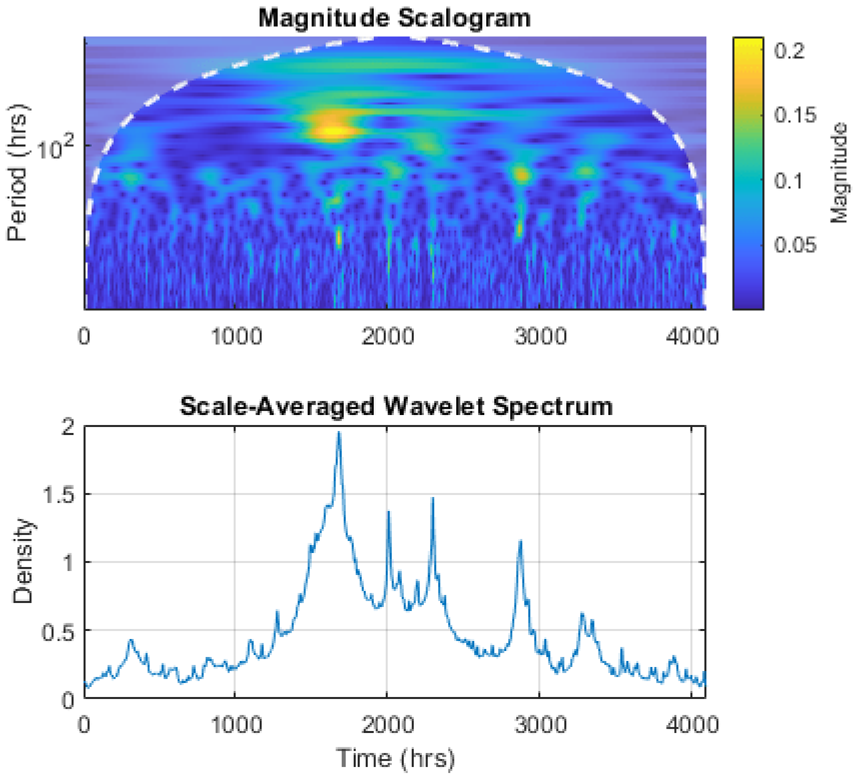
```
savg = scaleSpectrum(fb,sm,'Normalization','pdf');
sum(savg)
```

```
ans = 1.0000
```

If you set `SpectrumType` to `'density'`, `scaleSpectrum` normalizes the weighted integral of the wavelet spectrum according to the value of `Normalization`. In this case, the spectrum mimics a probability density function whose integral, numerically evaluated, equals the value specified by `Normalization`.

Plot the scalogram and the scale-averaged wavelet spectrum with spectrum type `'density'` and `'pdf'` normalization.

```
figure
scaleSpectrum(fb,sm,'SpectrumType','density','Normalization','pdf')
```



To confirm the integral of the spectrum equals 1, first obtain the scale-averaged wavelet spectrum with `'density'` spectrum type and `'pdf'` normalization.

```
savg = scaleSpectrum(fb,sm,'SpectrumType','density','Normalization','pdf');
```

By default, the filter bank uses the analytic Morse (3,60) wavelet. Obtain the admissibility constant for the wavelet and numerically integrate the wavelet spectrum using the trapezoidal rule. Confirm that the integral equals 1.

```
ga = 3;
tbw = 60;

be = tbw/ga;
anorm = 2*exp(be/ga*(1+(log(ga)-log(be))));
cPsi = anorm^2/(2*ga).*(1/2)^(2*(be/ga)-1)*gamma(2*be/ga);

numInt = 2/cPsi*1/length(sm)*trapz(1:length(savg),savg)
```

```
numInt = 1
```

Input Arguments

fb — Continuous wavelet transform filter bank

`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

`x` — Input data
vector

Input data, specified as a real- or complex-valued vector. The input data `x` must have at least four samples.

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

`cfs` — CWT coefficients
matrix | 3-D array

CWT coefficients, specified as a 2-D matrix or as an *M*-by-*N*-by-2 array. `cfs` should be the output of the `wt` object function of the CWT filter bank `fb`.

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

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `scaleSpectrum(fb,x,'FrequencyLimits',[0.2 0.4])` returns the scale-averaged wavelet spectrum averaged over the frequency limits `[0.2 0.4]`.

`Normalization` — Normalization
`'var'` (default) | `'pdf'` | `'none'`

Normalization of the scale-averaged wavelet spectrum, specified as a comma-separated pair consisting of `'Normalization'` and one of the following:

- `'var'` — Normalize to equal the variance of the time series `x`. If you provide the `cfs` input, the `scaleSpectrum` function uses the variance of the last time series processed by the filter bank object function `wt`.
- `'pdf'` — Normalize to equal 1.
- `'none'` — No normalization is applied.

`SpectrumType` — Type of wavelet spectrum
`'power'` (default) | `'density'`

Type of wavelet spectrum to return, specified as a comma-separated pair consisting of `'SpectrumType'` and either `'power'` or `'density'`. If specified as `'power'`, the averaged sum of the scale-averaged wavelet spectrum over all scales is normalized according to the value specified in '`Normalization`'. If specified as `'density'`, the weighted integral of the wavelet spectrum over all scales is normalized according to the value specified in '`Normalization`'.

`FrequencyLimits` — Frequency limits
two-element vector

Frequency limits over which the magnitude-squared scalogram is averaged, specified as a comma-separated pair consisting of `'FrequencyLimits'` and a two-element vector with nondecreasing elements. The `FrequencyLimits` values must lie between the lowest and highest center frequencies returned by the `centerFrequencies` object function of `fb`. The base 2 logarithm of the ratio of the maximum frequency to the minimum frequency must be greater than or equal to 1/*NV*, where *NV* is the value of the '`VoicesPerOctave`' property of the filter bank `fb`.

If a region of the specified limits falls outside the frequency limits of the filter bank `fb`, `scaleSpectrum` truncates computations to within the range specified by `centerFrequencies(fb)`. `FrequencyLimits` cannot be completely outside of the Nyquist range.

`PeriodLimits` — Period limits
two-element vector

Period limits over which the magnitude-squared scalogram is averaged, specified as a comma-separated pair consisting of `'PeriodsLimits'` and a two-element vector with nondecreasing durations. The elements of `PeriodLimits` agree in type and format with the '`SamplingPeriod`' property of the filter bank `fb`. The `SamplingPeriod` values must lie between the lowest and highest center periods returned by the `centerPeriods` object function of `fb`. The base 2 logarithm of the ratio of the minimum period to the maximum period must be less than or equal to -1/*NV*, where *NV* is the value of the '`VoicesPerOctave`' property of the filter bank `fb`.

If a region of the specified limits falls outside the period limits of the filter bank `fb`, `scaleSpectrum` truncates computations to within the range specified by `centerPeriods(fb)`. `SamplingPeriod` cannot be completely outside the Nyquist range of [2*Ts,*N**Ts], where `Ts` is the '`SamplingPeriod`' and *N* is the signal length.

Output Arguments

`savgp` — Scale-averaged wavelet power spectrum
real-valued vector | real-valued 3-D array

Scale-averaged wavelet power spectrum, returned as a real-valued vector or real-valued 3-D array. If `x` is real-valued, `savgp` is a 1-by-*N* vector where *N* is the length of `x`. If `x` is complex-valued, `savgp` is a 1-by-*N*-by-2 array, where the first page is the scale-averaged wavelet spectrum for the positive scales (analytic part or counterclockwise component), and the second page is the scale-averaged wavelet spectrum for the negative scales (anti-analytic part or clockwise component).

`scidx` — Scale indices
vector

Scale indices over which the scale-average wavelet spectrum is computed, returned as a vector. If you do not specify '`FrequencyLimits`' or '`PeriodLimits`', `scidx` is a vector from 1 to the number of scales.

References

[1] Torrence, Christopher, and Gilbert P. Compo. “A Practical Guide to Wavelet Analysis.” *Bulletin of the American Meteorological Society* 79, no. 1 (January 1, 1998): 61–78. https://doi.org/10.1175/1520-0477(1998)079<0061:APGTWA>2.0.CO;2.

[2] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[3] Lilly, J.M., and S.C. Olhede. “Higher-Order Properties of Analytic Wavelets.” IEEE Transactions on Signal Processing 57, no. 1 (January 2009): 146–60. https://doi.org/10.1109/TSP.2008.2007607.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- `PeriodLimits` name-value pair is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`cwtfilterbank` | `timeSpectrum`

Introduced in R2020b

scalingfunctions

DWT filter bank time-domain scaling functions

Syntax

```
phi = scalingfunctions(fb)
[phi,t] = scalingfunctions(fb)
```

Description

`phi = scalingfunctions(fb)` returns the time-centered scaling functions for each level of the discrete wavelet transform (DWT) filter bank `fb`.

`[phi,t] = scalingfunctions(fb)` returns the sampling instants, `t`.

Examples

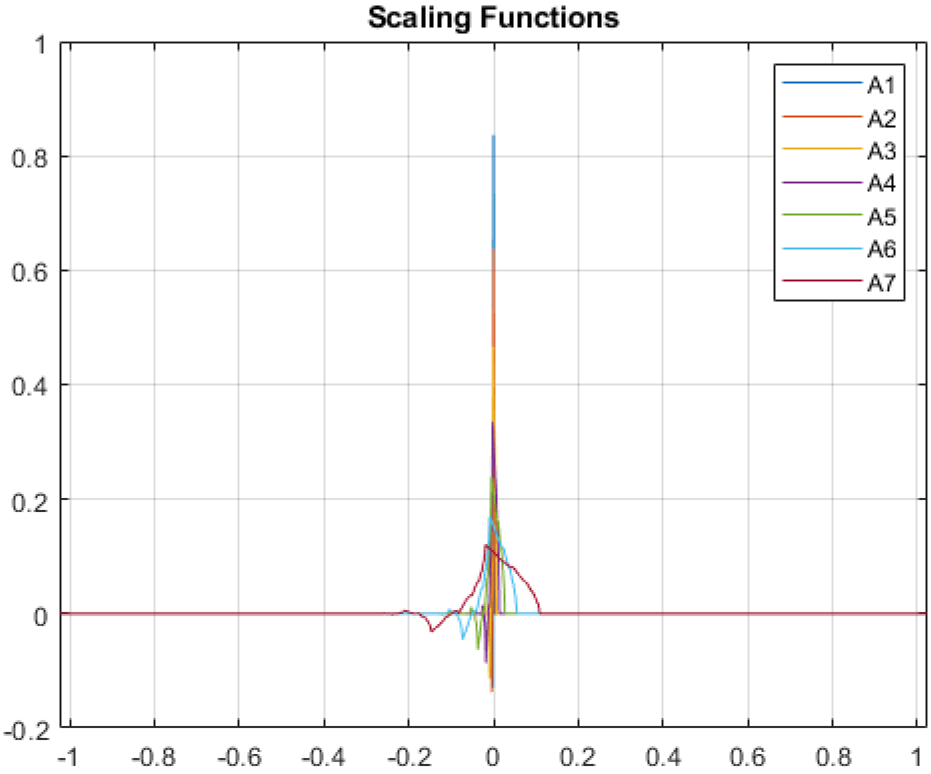
DWT Filter Bank Scaling Functions

Create a seven-level DWT filter bank for a length 2048 signal, using the Daubechies `db2` wavelet and a sampling frequency of 1 kHz.

```
wv = "db2";
len = 2048;
Fs = 1e3;
lev = 7;
fb = dwtfilterbank('SignalLength',len,'Wavelet',wv,'Level',lev,'SamplingFrequency',Fs);
```

Plot the scaling functions for each level of the filter bank.

```
[phi,t] = scalingfunctions(fb);
plot(t,phi')
grid on
xlim([-len/2*1e-3 len/2*1e-3])
title('Scaling Functions')
legend('A1','A2','A3','A4','A5','A6','A7')
```



Input Arguments

`fb` — Discrete wavelet transform filter bank

`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

Output Arguments

`phi` — Time-centered scaling functions

real-valued matrix

Time-centered scaling functions of the filter bank `fb`, returned as a real-valued L-by-N matrix, where L is the filter bank `Level` and N is the `SignalLength`. The scaling functions are ordered in `phi` from the finest scale resolution to the coarsest scale resolution.

`t` — Sampling instants

real-valued vector

Sampling instants, returned as a real-valued vector `t` of length N, where N is the filter bank `SignalLength`. Sampling instants lie in the interval $[-\frac{1}{2}NDT,\ \frac{1}{2}NDT)$, where DT is the filter bank sampling period (reciprocal of the filter bank sampling frequency).

See Also

`dwtfilterbank` | `wavelets` | `freqz`

Introduced in R2018a

scattergram

Visualize scattering or scalogram coefficients

Syntax

```
img = scattergram(sf,S)
img = scattergram(sf,U)
img = scattergram( ___ ,Name,Value)
scattergram( ___ )
```

Description

`img = scattergram(sf,S)` returns the scattergram as a matrix for the first-order scattering coefficients, `S`. The matrix `S` is the output of `scatteringTransform` computed using the wavelet time scattering network, `sf`.

`img = scattergram(sf,U)` returns the scattergram as a matrix for the first-order scalogram coefficients, `U`. The matrix `U` is the output of `scatteringTransform` computed using the wavelet time scattering network, `sf`.

`img = scattergram( ___ ,Name,Value)` returns the scattergram with additional options specified by one or more `Name,Value` pair arguments. You can use this syntax with any of the input syntaxes shown previously.

`scattergram( ___ )` with no output arguments plots the scattergram in the current figure. You can use any of the input syntaxes shown previously.

Examples

Visualize Scattergram

Load an ECG signal sampled at 180 Hz. Create a wavelet time scattering network that can be used with the signal.

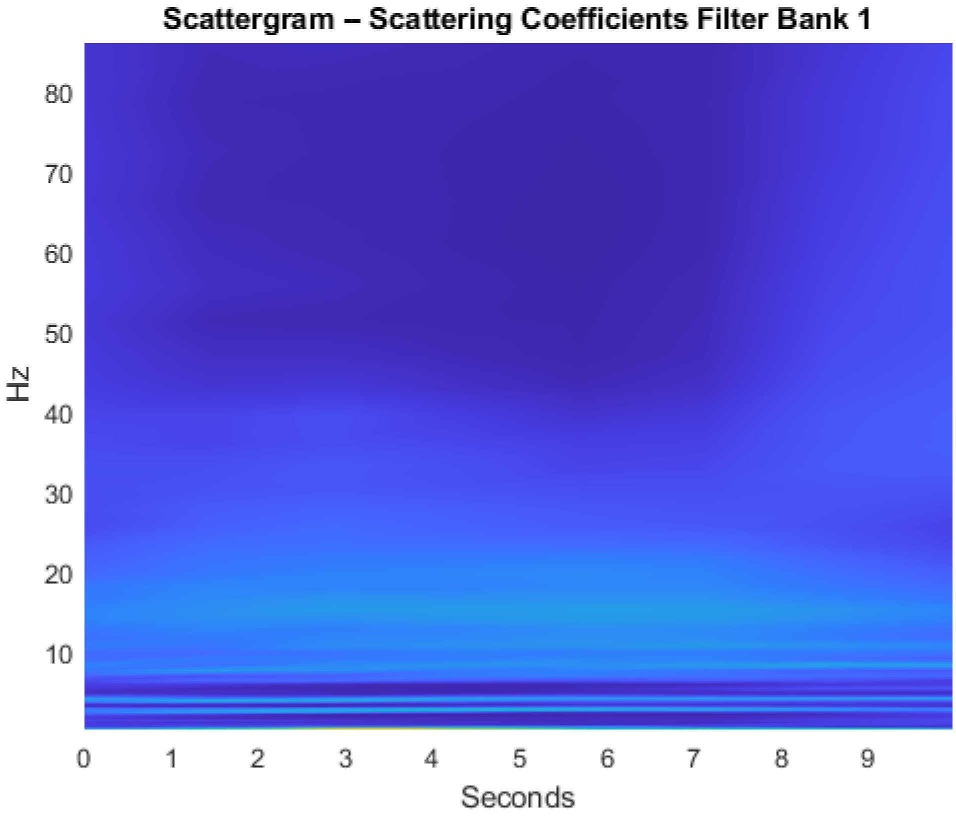
```
load wecg
Fs = 180;
sf = waveletScattering('SignalLength',numel(wecg),...
    'SamplingFrequency',Fs);
```

Calculate the scattering transform of the signal.

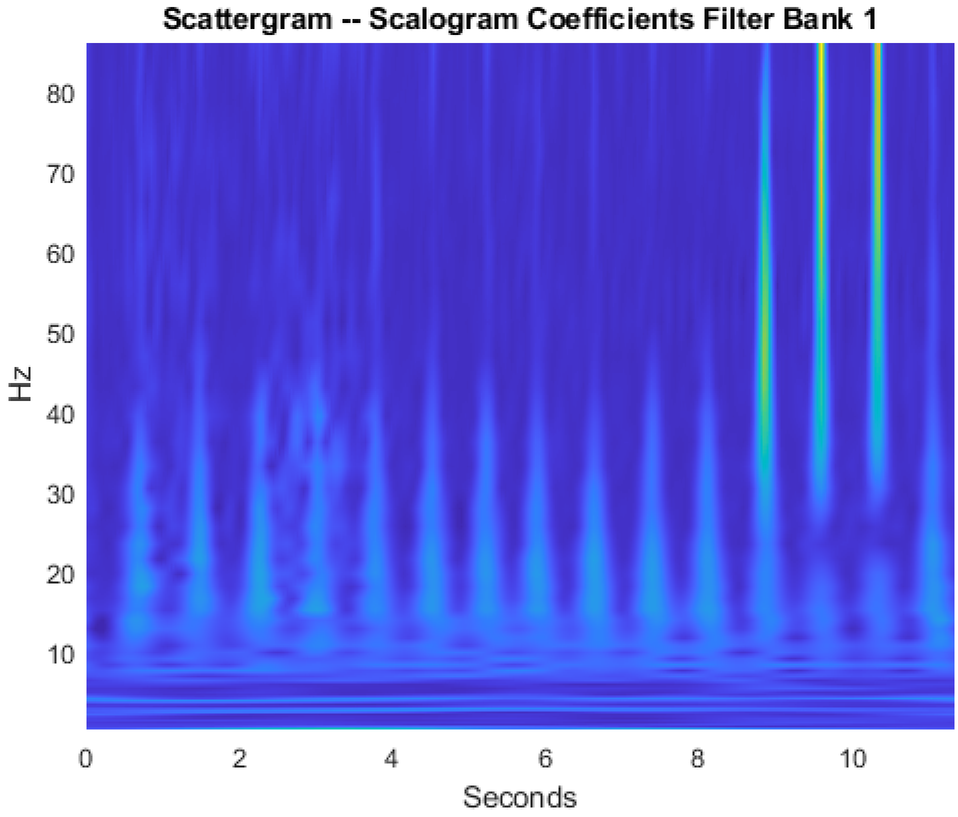
```
[S,U] = scatteringTransform(sf,wecg);
```

Visualize the scattergram for the first-order scattering and scalogram coefficients.

```
scattergram(sf,S,'FilterBank',1)
```



```
figure
scattergram(sf,U,'FilterBank',1)
```



Input Arguments

sf — Wavelet time scattering network
`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

S — Scattering coefficients
cell array

Scattering coefficients, specified as a cell array. `S` is the output of `scatteringTransform` computed using the scattering network, `sf`. For more information, see `scatteringTransform`.

U — Scalogram coefficients
cell array

Scalogram coefficients, specified as a cell array. `U` is the output of `scatteringTransform` computed using the scattering network, `sf`. For more information, see `scatteringTransform`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FilterBank',1` specifies the first filter bank.

`FilterBank` — Filter bank index
positive integer between 1 and the number of filter banks in `sf` inclusive

Filter bank index, specified as a positive number between 1 and the number of filter banks in `sf` inclusive. `scattergram` returns the scattergram for the specified filter bank in `sf`. The number of filter banks in `sf` is equal to the number of specified `QualityFactors` in `sf`.

If `FilterBank` is greater than 1, `scattergram` averages the scalogram or scattering coefficients over all paths terminating at each wavelet bandpass filter. To obtain paths with a common parent, use the `'Parent'` name-value pair.

`P` — Path parent index
nonnegative integer

Path parent index, specified as a nonnegative integer. The scalar `P` is a nonnegative integer representing the `P`-th wavelet filter at the filter bank `FilterBank` − 1. `scattergram` returns the scattergram for the path at the specified filter bank with parent `P`. If `FilterBank` is equal to 1, the zeroth filter bank corresponds to the input signal in the case of the scalogram coefficients and the lowpass filtering of the input signal with the scaling function in the case of the scattering coefficients. Lower values of `P` correspond to wavelets with higher bandpass frequencies.

If you specify `P`, you must specify the `FilterBank` name-value pair.

If you specify a value for `P` which results in a single child, the output `img` is a vector. The scattergram of a single child is a line plot. If you specify a value for `P` that results in no children, `scattergram` returns the scattergram for the filter bank specified by `FilterBank`.

Output Arguments

`img` — Scattergram
real-valued matrix | real-valued vector

Scattergram, returned as a real-valued matrix or vector. If you use the `Parent` name-value pair and specify a value which results in a single child, `img` is a vector. If the parent has more than one child, `img` is a matrix.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`waveletScattering`

Introduced in R2018b

scatteringTransform

Wavelet 1-D scattering transform

Syntax

```
s = scatteringTransform(sf,x)
[s,u] = scatteringTransform(sf,x)
```

Description

`s = scatteringTransform(sf,x)` returns the wavelet 1-D scattering transform of `x` with metadata for the wavelet time scattering network, `sf`. `x` is a real-valued vector, matrix, or 3-D array.

The precision of the scattering coefficients depends on the precision specified in the scattering network `sf`.

`[s,u] = scatteringTransform(sf,x)` also returns the scalogram coefficients for each of the scattering orders.

The precision of the scalogram coefficients depends on the precision specified in the scattering network `sf`.

Examples

Scattering Transform of ECG Signal

This example shows how to return the wavelet 1-D scattering transform of a real-valued signal.

Load an ECG signal sampled at 180 Hz.

```
load wecg
Fs = 180;
```

Create a wavelet time scattering network to apply to the signal. Compute the scattering transform of the signal.

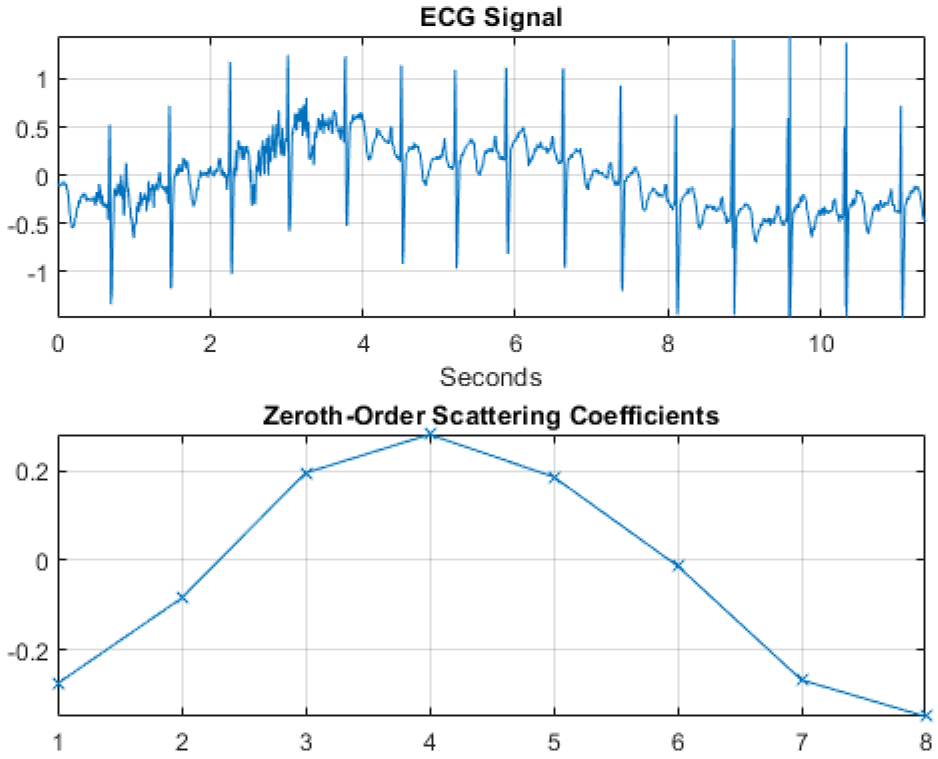
```
sf = waveletScattering('SignalLength',numel(wecg),...
    'SamplingFrequency',Fs)
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 2048
       InvarianceScale: 5.6889
        QualityFactors: [8 1]
              Boundary: 'periodic'
     SamplingFrequency: 180
             Precision: 'double'
    OversamplingFactor: 0
          OptimizePath: 0
```

```
[S,U] = scatteringTransform(sf,wecg);
```

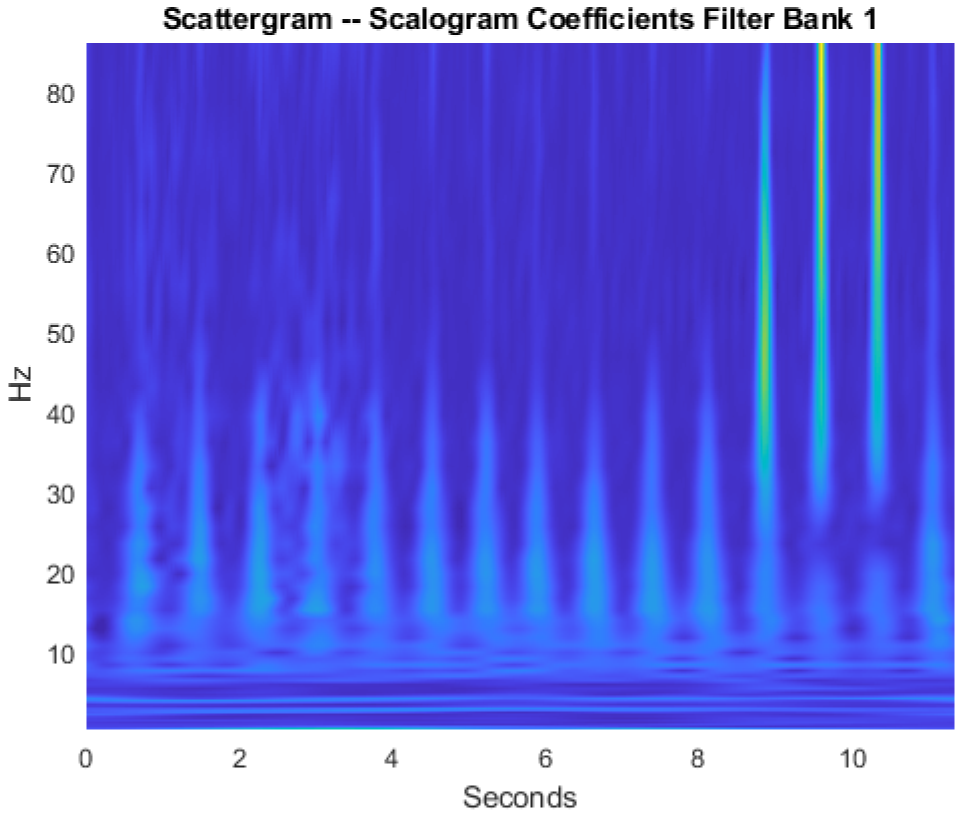
Plot the signal and the zeroth-order scattering coefficients. Note that the invariance scale is one half the duration of the signal.

```
t = [0:length(wecg)-1]/Fs;
subplot(2,1,1)
plot(t,wecg)
grid on
axis tight
xlabel('Seconds')
title('ECG Signal')
subplot(2,1,2)
plot(S{1}.signals{1},'x-')
grid on
axis tight
title('Zeroth-Order Scattering Coefficients')
```



Visualize the scattergram for the first-order scalogram coefficients.

```
figure
scattergram(sf,U,'FilterBank',1)
```



Input Arguments

`sf` — Wavelet time scattering network

`waveletScattering` object

Wavelet time scattering network, specified as a `waveletScattering` object.

`x` — Input data

vector | matrix | 3-D array

Input data, specified as a real-valued vector, matrix, or 3-D array. If `x` is a vector, the number of samples in `x` must equal the `SignalLength` value of `sf`. If `x` is a matrix or 3-D array, the number of rows in `x` must equal the `SignalLength` value of `sf`. If `x` is 2-D, the first dimension is assumed to be time and the columns of `x` are assumed to be separate channels. If `x` is 3-D, the dimensions of `x` are Time-by-Channel-by-Batch.

Data Types: `single` | `double`

Output Arguments

`s` — Scattering coefficients

cell array

Scattering coefficients, returned as a *NO*-by-1 cell array, where *NO* is the number of orders in `sf`.

Each element of `s` is a MATLAB table with the following variables:

`signals` — Scattering coefficients
cell array

Scattering coefficients, returned as a cell array. If `x` is a vector, each element of `signals` is a *Ns*-by-1 vector, where *Ns* is the number of scattering coefficients. If `x` is 2-D, each element of `signals` is a *Ns*-by-*Nc* matrix, where *Nc* is the number of channels in `x`. If `x` is 3-D, each element of `signals` is a *Ns*-by-*Nc*-by-*Nb* array, where *Nb* is the number of batches in `x`.

Data Types: `single` | `double`

`path` — Scattering path
row vector

Scattering path used to obtain the scattering coefficients, returned as a row vector. Each column of `path` corresponds to one element of the path. The scalar 0 denotes the original signal. Positive integers in the L^{th} column denote the corresponding wavelet filter in the $(L\text{-}1)^{\text{th}}$ filter bank. Wavelet bandpass filters are ordered by decreasing center frequency.

Data Types: `double`

`bandwidth` — Bandwidth of scattering coefficients
scalar

Bandwidth of the scattering coefficients, returned as a scalar. If you specify a sampling frequency in the scattering network, the bandwidth is in hertz. Otherwise, the bandwidth is in cycles/sample.

Data Types: `double`

`resolution` — Base-2 log resolution
scalar

Base-2 log resolution of the scattering coefficients, returned as a scalar.

Data Types: `double`

`u` — Scalogram coefficients
cell array

Scalogram coefficients, returned as a *NO*-by-1 cell array, where *NO* is the number of orders in `sf`. The *i*th element of `u` are the scalogram coefficients for the *i*th row of `s`.

Each element of `u` is a MATLAB table with the following variables:

`coefficients` — Scalogram coefficients
cell array

Scalogram coefficients, returned as a cell array. If `x` is a vector, each element of `coefficients` is a *Nu*-by-1 vector, where *Nu* is the number of scalogram coefficients. If `x` is 2-D, each element of `coefficients` is a *Nu*-by-*Nc* matrix, where *Nc* is the number of channels in `x`. If `x` is 3-D, each element of `coefficients` is a *Nu*-by-*Nc*-by-*Nb* array, where *Nb* is the number of batches in `x`.

Note that `u{1}` contains the original data in the coefficients variable.

Data Types: `single` | `double`

`path` — Scattering path
row vector

Scattering path used to obtain the scalogram coefficients, returned as a row vector. Each column of `path` corresponds to one element of the path. The scalar 0 denotes the original signal. Positive integers in the L^{th} column denote the corresponding wavelet filter in the $(L\text{-}1)^{th}$ filter bank. Wavelet bandpass filters are ordered by decreasing center frequency.

Data Types: `double`

`bandwidth` — Bandwidth of scalogram coefficients
scalar

Bandwidth of the scalogram coefficients, returned as a scalar. If you specify a sampling frequency in the scattering network, the bandwidth is in hertz. Otherwise, the bandwidth is in cycles/sample.

Data Types: `double`

`resolution` — Base-2 log resolution
scalar

Base-2 log resolution of the scalogram coefficients, returned as a scalar.

Data Types: `double`

Tips

- The `scatteringTransform` function calls `featureMatrix` to generate the scattering and scalogram coefficients. If you only require the coefficients themselves, for improved performance the recommended approach is to use `featureMatrix`. Use `scatteringTransform` if you are also interested in the coefficients metadata.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`waveletScattering` | `featureMatrix`

Introduced in R2018b

scatteringTransform

Wavelet 2-D scattering transform

Syntax

```
s = scatteringTransform(sf,im)
[s,u] = scatteringTransform(sf,im)
```

Description

`s = scatteringTransform(sf,im)` returns the wavelet 2-D scattering transform of `im` for `sf`, the image scattering network. `im` is a real-valued 2-D matrix or 3-D matrix. If `im` is 3-D, the size of the third dimension must equal 3. The row and column sizes of `im` must match the `ImageSize` value of `sf`. The output `s` is a cell array with *Nfb*+1 elements, where *Nfb* is the number of filter banks in the scattering network. *Nfb* is equal to the number of elements in the `QualityFactors` property of `sf`. Equivalently, the number of elements in `s` is equal to the number of orders in the scattering network. Each element of `s` is a MATLAB table.

`[s,u] = scatteringTransform(sf,im)` also returns the wavelet scalogram coefficients for `im`. The output `u` is a cell array with *Nfb*+1 elements, where *Nfb* is the number of filter banks in the scattering network. *Nfb* is equal to the number of elements in the `QualityFactors` property of `sf`. Equivalently, the number of elements in `u` is equal to the number of orders in the scattering network. Each element of `u` is a MATLAB table.

Examples

Compare Scattering and Scalogram Coefficients

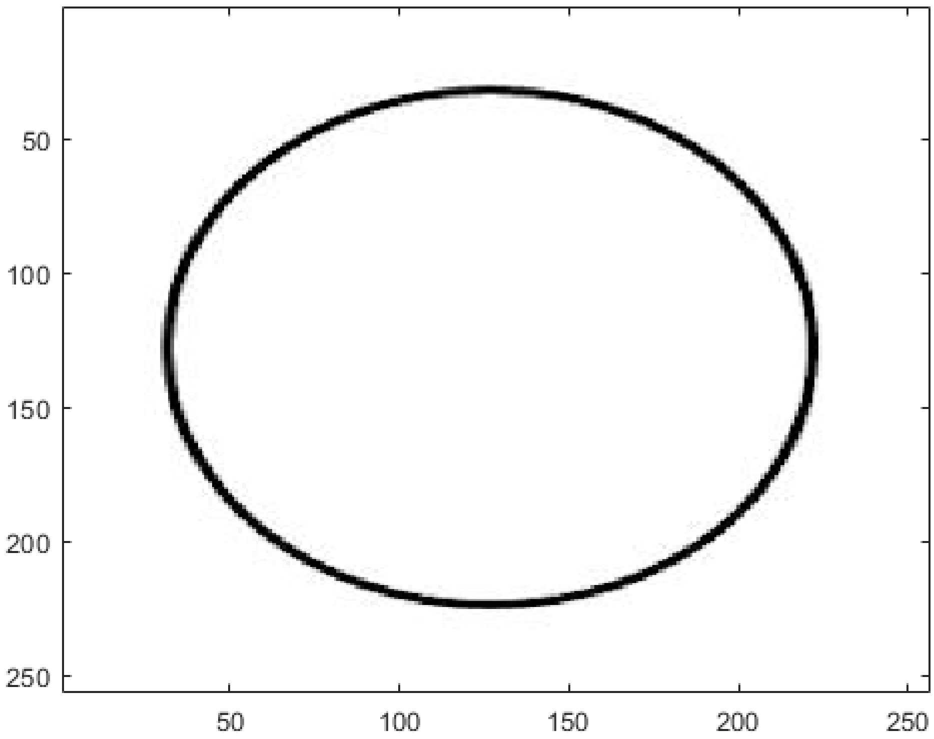
This example shows that scattering coefficients are lowpassed versions of scalogram coefficients.

Load an RGB image. Display the red channel.

```
im = imread('circle.jpg');
size(im)
```

```
ans = 1×3

   256   256     3
```

```
figure
imagesc(im(:,:,1))
colormap gray;
```



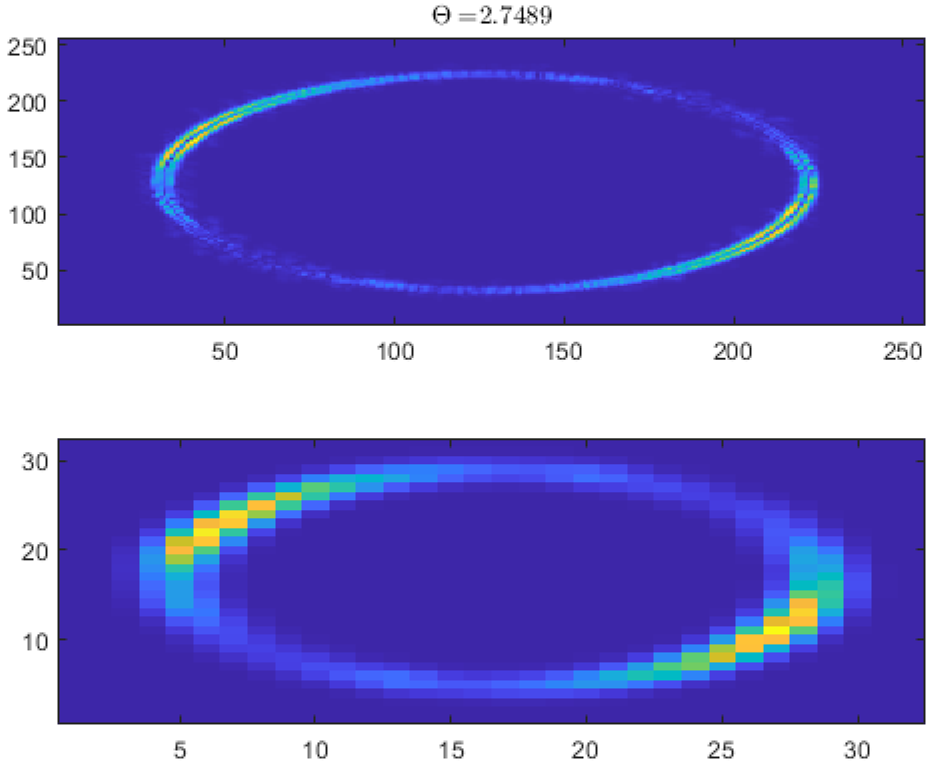
For RGB images, the size of the third dimension must be 3. You only have to specify the row and column sizes of the image when you create the scattering network. Create a scattering network to apply to the image and take the scattering transform.

```
sf = waveletScattering2('ImageSize',[256 256],'InvarianceScale',32,...
    'NumRotations',[8 8]);
[S,U] = scatteringTransform(sf,im);
```

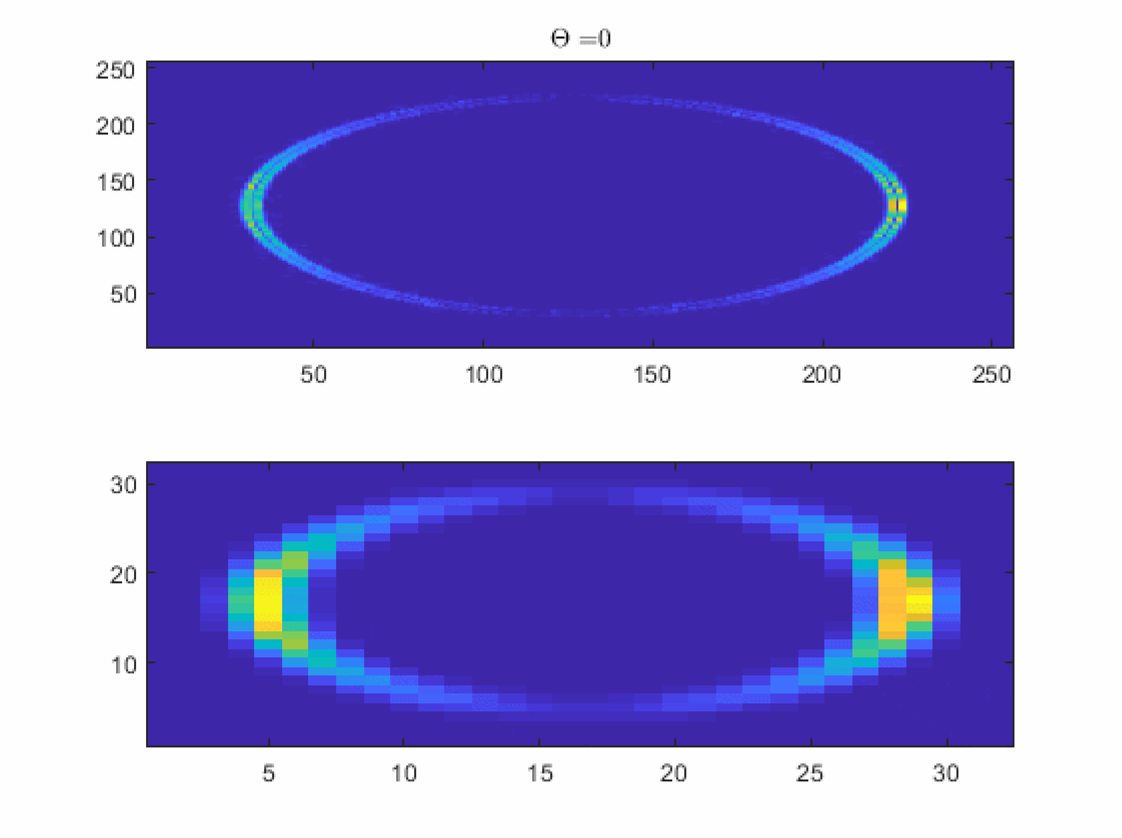
The image and coefficient fields in S and U are *M*-by-*N*-by-3. The *M*-by-*N* dimensions are constant only in the scattering images because the scaling function has fixed bandwidth, while the wavelets have different bandwidths.

Use a for-loop and plot the red channel for the scalogram and scattering coefficients for the 8 rotation angles in the scattering transform. Note how the scattering coefficients are lowpass versions of the scalogram coefficients.

```
[~,~,~,filterparams] = sf.filterbank();
theta = filterparams{1}.rotations;
figure
for k = 1:numel(theta)
    subplot(2,1,1)
    imagesc(U{2}.coefficients{k}(:,:,1));
    axis xy
    title(['$$\Theta =  $$' num2str(theta(k))],'Interpreter','Latex');
    subplot(2,1,2)
    imagesc(S{2}.images{k}(:,:,1));
    axis xy
    pause(1)
end
```



The above for-loop results in an animation identical to the one below.



Input Arguments

`sf` — Wavelet image scattering network

`waveletScattering2` object

Wavelet image scattering network, specified as a `waveletScattering2` object.

`im` — Input image

real-valued matrix

Input image, specified as a real-valued 2-D matrix or 3-D matrix. If `im` is 3-D, `im` is assumed to be a color image in the RGB color space, and the size of the third dimension must equal 3. The row and column sizes of `im` must match the `ImageSize` property of `sf`.

Output Arguments

`s` — Scattering coefficients

cell array

Scattering coefficients, returned as a cell array. `s` is a cell array with *Nfb*+1 elements where *Nfb* is the number of filter banks in the scattering network. *Nfb* is equal to the number of elements in the `QualityFactors` property of `sf`. Equivalently, the number of elements in `s` is equal to the number of orders in the scattering network. Each element of `s` is a MATLAB table with these variables:

images — Scattering coefficients
cell array

Scattering coefficients, returned as a cell array. Each element of `images` is an *M*-by-*N* or *M*-by-*N*-by-3 matrix.

path — Scattering path
row vector

Scattering path used to obtain the scattering coefficients, returned as a row vector. Each column of `path` corresponds to one element of the path. The scalar 0 denotes the original image. Positive integers in the Lth column denote the corresponding wavelet filter in the $(L-1)$th filter bank. Wavelet bandpass filters are ordered by decreasing center frequency.

There are `NumRotations` wavelets per center frequency pair.

bandwidth — Bandwidth of scattering coefficients
scalar

Bandwidth of scattering coefficients, returned as a scalar. The bandwidth is symmetric in the *x* and *y* directions.

resolution — Base-2 log resolution
scalar

Base-2 log resolution of the scattering coefficients, returned as a scalar.

u — Scalogram coefficients
cell array

Scalogram coefficients, returned as a cell array. `u` is a cell array with $Nfb+1$ elements, where *Nfb* is the number of filter banks in the scattering network. *Nfb* is equal to the number of elements in the `QualityFactors` property of `sf`. Equivalently, the number of elements in `u` is equal to the number of orders in the scattering network. Each element of `u` is a MATLAB table with these variables:

coefficients — Scalogram coefficients
cell array

Scalogram coefficients, returned as a cell array. Each element of `coefficients` is an *M*-by-*N* or *M*-by-*N*-by-3 matrix.

path — Scattering path
row vector

Scattering path used to obtain the scalogram coefficients, returned as a row vector. Each column of `path` corresponds to one element of the path. The scalar 0 denotes the original image. Positive integers in the Lth column denote the corresponding wavelet filter in the $(L-1)$th filter bank. Wavelet bandpass filters are ordered by decreasing center frequency.

There are `NumRotations` wavelets per center frequency pair.

bandwidth — Bandwidth of scalogram coefficients
scalar

Bandwidth of scalogram coefficients, returned as a scalar.

`resolution` — Base-2 log resolution
scalar

Base-2 log resolution of the scattering coefficients, returned as a scalar.

See Also

`waveletScattering2`

Introduced in R2019a

sensingDictionary

Sensing dictionary for sparse signal recovery

Description

Use `sensingDictionary` to create a sensing dictionary object for sparse approximations of 1-D signals. The `sensingDictionary` function provides built-in support for a variety of frames, including wavelet, discrete cosine transform (DCT), Fourier, and Gaussian and Bernoulli random distributions. You can also create and use custom dictionaries. You can apply your dictionary for signal sparse recovery using matching pursuit or basis pursuit. Additionally, the basis pursuit algorithm supports custom dictionaries created using tall arrays. You can apply these custom dictionaries to tall array inputs.

Creation

Syntax

```
A = sensingDictionary
A = sensingDictionary(Name=Value)
```

Description

`A = sensingDictionary` creates a sensing dictionary that corresponds to the 100-by-100 identity matrix.

`A = sensingDictionary(Name=Value)` creates a sensing dictionary with properties on page 1-1206 specified by name-value arguments. For example, `A = sensingDictionary(Type={'dct'})` creates a sensing dictionary corresponding to the `dct` basis type. You can specify multiple name-value arguments.

Properties

`Size` — Sensing dictionary size
`[100 100]` (default) | positive integer | two-element vector

Sensing dictionary size, specified as a positive integer or two-element vector of positive integers. If you specify the size as `[m,n]`, the number of rows in the sensing dictionary is *m*, and the number of columns is *n*. If you specify the size as a scalar *m*, the number of rows in the dictionary is *m*, and the number of columns depends on the type of dictionary. For random dictionaries, you must specify the size as a two-element vector.

Note You cannot change the size of an existing sensing dictionary. For example, if the size of the sensing dictionary `A` is `[100 100]`, you cannot assign a different `Size` to `A`.

Example: `A = sensingDictionary(Size=[50 50],Type={'dct','poly'})` creates a sensing dictionary A consisting of the basis types `'dct'` and `'poly'`. The size of the matrix of each type is 50-by-50 and `A.Size = [50 100]`.

Data Types: `double`

Type — Dictionary basis type
cell array of character vectors

Dictionary basis type, specified as a cell array of character vectors. Each character vector specifies a basis type in the sensing dictionary `A`. For each basis type, the basis elements are arranged columnwise in the matrix. Each basis element is length *N*, where `N = A.Size(1)`. The `sensingDictionary` object supports the following basis types:

- `'eye'` (default) — Dictionary corresponds to an identity matrix.
- `'dct'` — Dictionary corresponds to the discrete cosine transform-II basis. The DCT-II orthonormal basis is:

$$\phi_k(n) = \begin{cases} \dfrac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\dfrac{2}{N}}\cos\left(\dfrac{\pi}{N}\left(n + \dfrac{1}{2}\right)k\right) & k = 1, 2, \ldots, N - 1. \end{cases}$$

- `'dwt'` — Dictionary corresponds to a specific wavelet basis from a certain level of decomposition.
- `'fourier'` — Dictionary corresponds to the Fourier basis. The Fourier basis is $\phi_k(t) = \dfrac{e^{2\pi ikt/N}}{\sqrt{N}}$, where `k = 0, …, N-1`, and `t = 0, …, N-1`.
- `'poly'` — The *k*th column of the dictionary matrix corresponds to monomials of the form `t.^(k-1)`, where `t` is the time interval specified by `linspace(0,1,N)` and `k = 1,...,N`.
- `'rand'` — Dictionary matrix entries are either an independent identically distributed (IID) Gaussian matrix (default) or a Bernoulli matrix.
- `'walsh'` — Dictionary matrix entries are generated from Walsh code.

Example: `A = sensingDictionary('Type',{'eye','dct'})` creates a sensing dictionary with basis types `'eye'` and `'dct'`.

Data Types: `char` | `string`

Name — Basis name
cell array of character vectors

Basis name, specified as a cell array of character vectors. `Name` is supported only for the following basis types.

- `'dwt'` — Character vectors are the names of orthogonal wavelets supported by the `modwt` function:
  - `'haar'` or `'db1'` (default) — Haar wavelet.
  - `'dbN'` — Extremal phase Daubechies wavelet with *N* vanishing moments, where *N* is a positive integer from 2 to 45.
  - `'symN'` — Symlets wavelet with *N* vanishing moments, where *N* is a positive integer from 2 to 45.

  - `'coifN'` — Coiflets wavelet with *N* vanishing moments, where *N* is a positive integer from 1 to 5.
  - `'fkN'` — Fejér-Korovkin wavelet with *N* coefficients, where `N = 4, 6, 8, 14, 18` and `22`.
- `'rand'` — Character vectors are the names of random matrices:
  - `'Gaussian'` (default) — IID Gaussian matrix.
  - `'Bernoulli'` — Bernoulli matrix.

If not all basis types specified in `Type` support `Name`, insert an empty character vector in the corresponding positions in `Name`.

Command	Result

`A = sensingDictionary(Size=[50 50],...`
`    Type={'dwt','rand'},...`
`    Name={'coif4','Bernoulli'});`
	Creates a sensing dictionary that corresponds to the `coif4` wavelet basis and a random Bernoulli matrix.

`A = sensingDictionary(Size=[50 50],...`
`    Type={'dwt','dct','rand'},...`
`    Name={'coif4','','Bernoulli'});`
	Creates a sensing dictionary that corresponds to the `coif4` wavelet basis, DCT basis, and a random Bernoulli matrix.

Data Types: `char` | `string`

`Level` — Decomposition level
`floor(log2(A.Size(1)))` (default) | integer | vector

Decomposition level of the wavelet basis, specified as an integer or vector of positive integers. You can specify `Level` only if you specify at least one wavelet basis in `Type`. If you specify wavelet and other basis types in `Type`, you can either set the level of the other basis to `0`, or omit it. The number of nonzero elements you specify must be less than or equal to the number of `'dwt'` types.

Command	Result

`A = sensingDictionary(Type={'dwt'},...`
`    Name={'db2'},...`
`    Level=[3]);`
	Creates a sensing dictionary that corresponds to the level 3 `db2` wavelet details.

`A = sensingDictionary(Type={'dwt','fourier'},...`
`    Name={'db2'},...`
`    Level=[3 0]);`
or
`A = sensingDictionary(Type={'dwt','fourier'},...`
`    Name={'db2'},...`
`    Level=[3]);`
	Creates a sensing dictionary that corresponds to the level 3 `db2` wavelet details and the Fourier basis.

Data Types: `double`

`CustomDictionary` — Custom sensing dictionary matrix
matrix

Custom sensing dictionary matrix, specified as a matrix. You cannot simultaneously specify `CustomDictionary` and any other property.

Example: `A = sensingDictionary(CustomDictionary=xmat)` creates a sensing dictionary using the custom matrix `xmat`.

Data Types: single | double
Complex Number Support: Yes

Object Functions

matchingPursuit	Recover sparse signal using matching pursuit algorithm
basisPursuit	Recover sparse signal using the basis pursuit algorithm
horzcat	Horizontal concatenation of two sensing dictionaries
subdict	Extract submatrix from a sensing dictionary

Examples

Create Sensing Dictionary

Create the sensing dictionary that is a concatenation of the level 2 db1 wavelet and DCT bases.

```
A = sensingDictionary(Size=100,Type={'dwt','dct'},...
    Name={'db1'},...
    Level=[2 0])
```

```
A = 
  sensingDictionary with properties:

                Type: {'dwt'  'dct'}
                Name: {'db1'  ''}
               Level: [2 0]
    CustomDictionary: []
                Size: [100 200]
```

Obtain Best Fit Using Matching Pursuit

Create a sensing dictionary consisting of basis types eye and poly. The size of each basis type is 150-by-150.

```
A = sensingDictionary(Size=150,Type={'eye','poly'});
```

Use the subdict object function to extract one basis vector from each basis type. Create a signal by multiplying the sum of the vectors by 2.

```
indA = 20;        % basis vector in 'eye'
indB = 160;       % basis vector in 'poly'
btv = subdict(A,1:150,[indA indB]);
sig = 2*sum(btv,2);
```

Obtain the best fit of the signal using the sensing dictionary A and matching pursuit algorithm with default settings.

```
[Xr,YI,I,R] = matchingPursuit(A,sig);
norm(sig-YI)
```

```
ans = 6.1693e-09
```

Inspect the indices.

```
I
```

```
I = 1×2

    20   160
```

Inspect the coefficients.

```
Xr(I)
```

```
ans = 2×1

    2.0000
    2.0000
```

Extended Capabilities

Tall Arrays

Calculate with arrays that have more rows than fit in memory.

This function fully supports tall arrays. For more information, see “Tall Arrays”.

See Also

Topics

“Signal Deconvolution and Impulse Denoising Using Pursuit Methods”
“Matching Pursuit Algorithms”

Introduced in R2022a

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

setLabelValue

Set label value in labeled signal set

Syntax

```
setLabelValue(lss,midx,lblname,val)
setLabelValue(lss,midx,lblname,limits,val)
setLabelValue(lss,midx,lblname,locs,val)
setLabelValue( ___ ,'LabelRowIndex',ridx)
setLabelValue( ___ ,'SublabelRowIndex',sridx)
```

Description

`setLabelValue(lss,midx,lblname,val)` sets the attribute label `lblname` to value `val`, for the member of labeled signal set `lss` specified in `midx`. Omit `val` if `lblname` has a default value and you want to set the label to the default value.

`setLabelValue(lss,midx,lblname,limits,val)` adds regions delimited by `limits` to the ROI label named `lblname`. The number of rows of `limits` specifies the number of added regions.

`setLabelValue(lss,midx,lblname,locs,val)` adds points to the point label named `lblname`. `locs` specifies the number of added points and their locations.

`setLabelValue( ___ ,'LabelRowIndex',ridx)` specifies the row index, `ridx`, of an ROI or point label. The specified value replaces the current value of that row. If you omit this argument, the function appends ROI or point values to any existing label values.

`setLabelValue( ___ ,'SublabelRowIndex',sridx)` specifies the row index, `sridx`, of an ROI or point sublabel. The specified value replaces the current value of that sublabel row.

Examples

Set Label Value

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

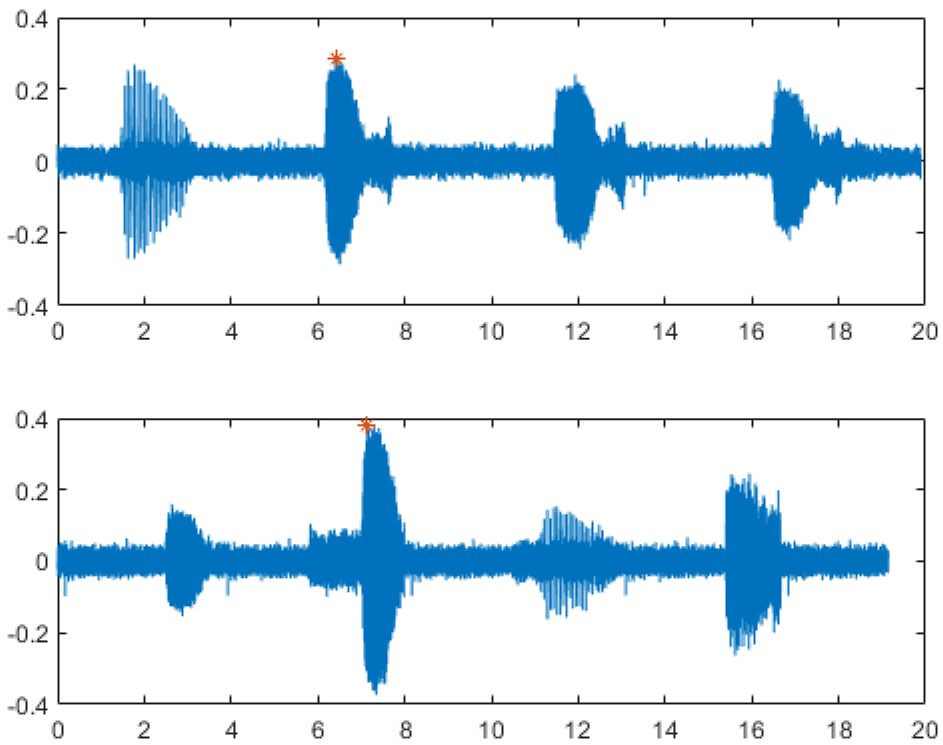
```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
Use setLabelValue to add data to the set.
```

Add a new label to the signal set, corresponding to the maximum value of each member.

```
theMax = signalLabelDefinition('Maximum', ...
    'LabelDataType','numeric', ...
    'Description','Maximum value of the signal');
addLabelDefinitions(lss,theMax)
```

For each labeled signal, set the value of the new label to the signal maximum. Plot the signals and their maxima.

```
fs = lss.SampleRate;
for k = 1:lss.NumMembers
    sg = getSignal(lss,k);
    [mx,ix] = max(sg);

    setLabelValue(lss,k,'Maximum',mx)

    subplot(2,1,k)
    plot((0:length(sg)-1)/fs,sg,ix/fs,mx,'*')
end
```



Display the names and values of the labels in the set.

```
lbldefs = getLabelValues(lss)
```

```
lbldefs=2×4 table
                 WhaleType    MoanRegions    TrillRegions     Maximum  
                 _________    ___________    ____________    __________

    Member{1}      blue       {3x2 table}    {1x3 table}     {[0.2850]}
    Member{2}      blue       {3x2 table}    {1x3 table}     {[0.3791]}
```

Decide that the signal maximum is better represented as a point label than as an attribute. Remove the numeric definition and redefine the maximum.

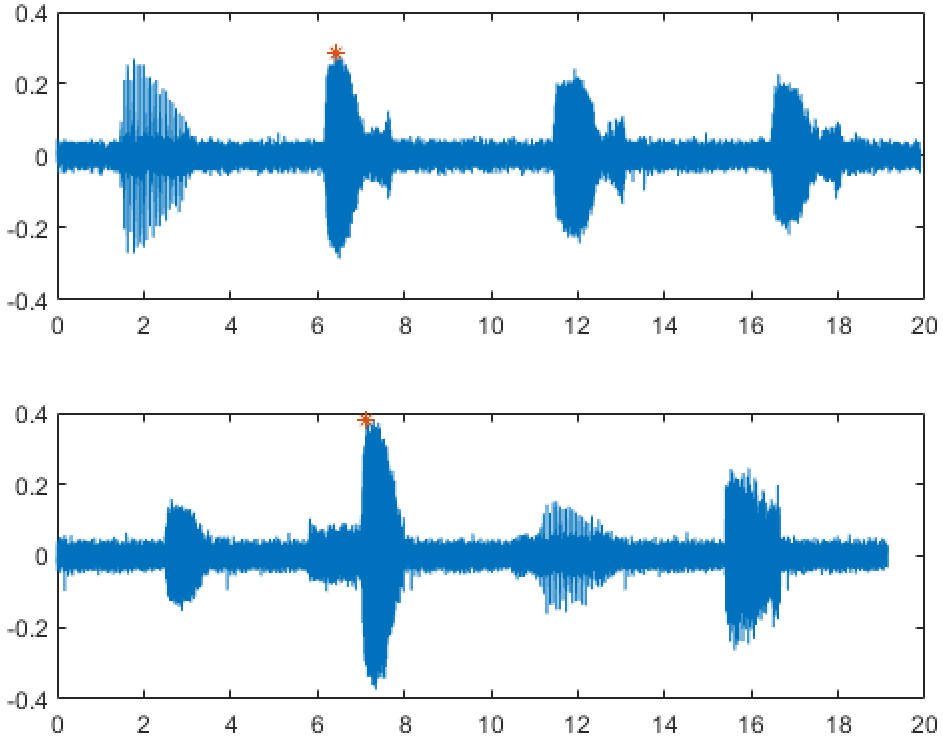
```
removeLabelDefinition(lss,'Maximum')
theMax = signalLabelDefinition('Maximum', ...
    'LabelType','point','LabelDataType','numeric', ...
    'Description','Maximum value of the signal');
addLabelDefinitions(lss,theMax)
```

For each labeled signal, set the value of the new label to the signal maximum.

```
for k = 1:lss.NumMembers
    sg = getSignal(lss,k);
    [mx,ix] = max(sg);
    setLabelValue(lss,k,'Maximum',ix/fs,mx)
end
```

Plot the signals and their maxima.

```
for k = 1:lss.NumMembers
    subplot(2,1,k)
    sg = getSignal(lss,k);
    peaks = getLabelValues(lss,k,'Maximum');
    plot((0:length(sg)-1)/fs,sg, ...
        peaks.Location,cell2mat(peaks.Value),'*')
end
```



Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midx` — Member row number

positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

`lblname` — Label or sublabel name

character vector | string scalar | cell array of character vectors | string array

Label name, specified as a character vector or string scalar.

Label or sublabel name. To specify a label, use a character vector or a string scalar. To specify a sublabel, use a two-element cell array of character vectors or a two-element string array:

- The first element is the name of the parent label.
- The second element is the name of the sublabel.

When targeting a sublabel of an ROI or point label, you must also specify the `'LabelRowIndex'` of the parent label whose label you want to set. The row of the parent must already exist before you can set a sublabel value to it.

Example: `signalLabelDefinition("Asleep",'LabelType','roi')` specifies a label of name `"Asleep"` for a region of a signal in which a patient is asleep during a clinical trial.

Example: `{'Asleep' 'REM'}` or `["Asleep" "REM"]` specifies a region of a signal in which a patient undergoes REM sleep.

`val` — Label values

numeric value or array | logical value or array | categorical value or array | character vector or cell array of character vectors | string or string array | table or table array | timetable or timetable array

Label values, specified as a numeric, logical, or categorical value, as a string, as a table, or as a timetable. `val` can also be an array of any of the previous types. `val` must be of the data type specified for `lblname`.

- If you specify `locs`, then `val` must have the same number of elements as `locs`.
- If you specify `limits`, then `val` must have a number of elements equal to the number of rows in `limits`.
  - If `limits` has more than one row, and `lblname` is of type `'numeric'` or `'logical'`, then `val` must be a vector or a cell array.
  - If `limits` has more than one row, and `lblname` is of type `'string'` or `'categorical'`, then `val` must be a string array or a cell array of character vectors.
  - If `limits` has more than one row, and `lblname` is of type `'table'` or `'timetable'`, then `val` must be a cell array of tables or timetables.

Assign Nonscalar Label Values

To assign nonscalar label values to several points or regions of interest, you must use cell arrays. For example, given the labeled signal set

```
lss = labeledSignalSet(randn(10,1), [...
    signalLabelDefinition('pl','LabelType','point', ...
                               'LabelDataType','numeric') ...
    signalLabelDefinition('rl','LabelType','ROI', ...
                               'LabelDataType','numeric')]);
```

the commands

```
setLabelValue(lss,1,'pl',5,{[3 4]'})
setLabelValue(lss,1,'rl',[2 3; 8 9],{[2 1]' [6 7]})
```

label point 5 with the column vector `[3 4]'`, the region limited by 2 and 3 with the column vector `[2 1]'`, and the region limited by 8 and 9 with the row vector `[6 7]`.

`limits` — Region limits

two-column matrix

Region limits, specified as a two-column matrix.

- If `lss` does not have time information, then `limits` defines the minimum and maximum indices over which the regions are defined.
- If `lss` has time information, then `limits` defines the minimum and maximum instants over which the regions are defined.

`limits` must be of the data type specified by the “ROILimitsDataType” on page 1-0 property of the label definition for `lblname`.

Example: `seconds([0:3;1:4]')`

Example: `[0:3;1:4]'`

`locs` — Point locations

vector

Point locations, specified as a vector.

- If `lss` does not have time information, then `locs` defines the indices corresponding to the point locations.
- If `lss` has time information, then `locs` defines the instants corresponding to the point locations.

`locs` must be of the data type specified by the “PointLocationsDataType” on page 1-0 property of the label definition for `lblname`.

`ridx` — Label row index

positive integer

Label row index, specified as a positive integer. This argument applies only for ROI and point labels.

`sridx` — Sublabel row index

positive integer

Sublabel row index, specified as a positive integer. This argument applies only when a label and sublabel pair has been specified in `lblname` and the sublabel is of type ROI or point.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

setMemberNames

Set member names in labeled signal set

Syntax

```
setMemberNames(lss,mnames)
setMemberNames(lss,mnames,midx)
```

Description

`setMemberNames(lss,mnames)` sets the names of the members of the labeled signal set `lss` to `mnames`. The length of `mnames` must be equal to the number of members.

`setMemberNames(lss,mnames,midx)` sets the name of the member specified by `midx`.

Examples

Set Member Names

Load a labeled signal set containing recordings of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Set the names of the set members to the whales' nicknames.

```
setMemberNames(lss,{'Brutus' 'Lucy'})
```

Return a string array with the names of the members.

```
getMemberNames(lss)
```

```
ans = 2x1 string
    "Brutus"
    "Lucy"
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`mnames` — Member names

character vector | string scalar | cell array of character vectors | string array

Member names, specified as a character vector, a string scalar, a cell array of character vectors, or a string array.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},'MemberNames',{'llama' 'alpaca'})` specifies a set of random signals with two members, `'llama'` and `'alpaca'`.

`midx` — Member row number

positive integer

Member row number, specified as a positive integer. `midx` specifies the member row number as it appears in the “Labels” on page 1-0 table of a labeled signal set.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2019a

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.

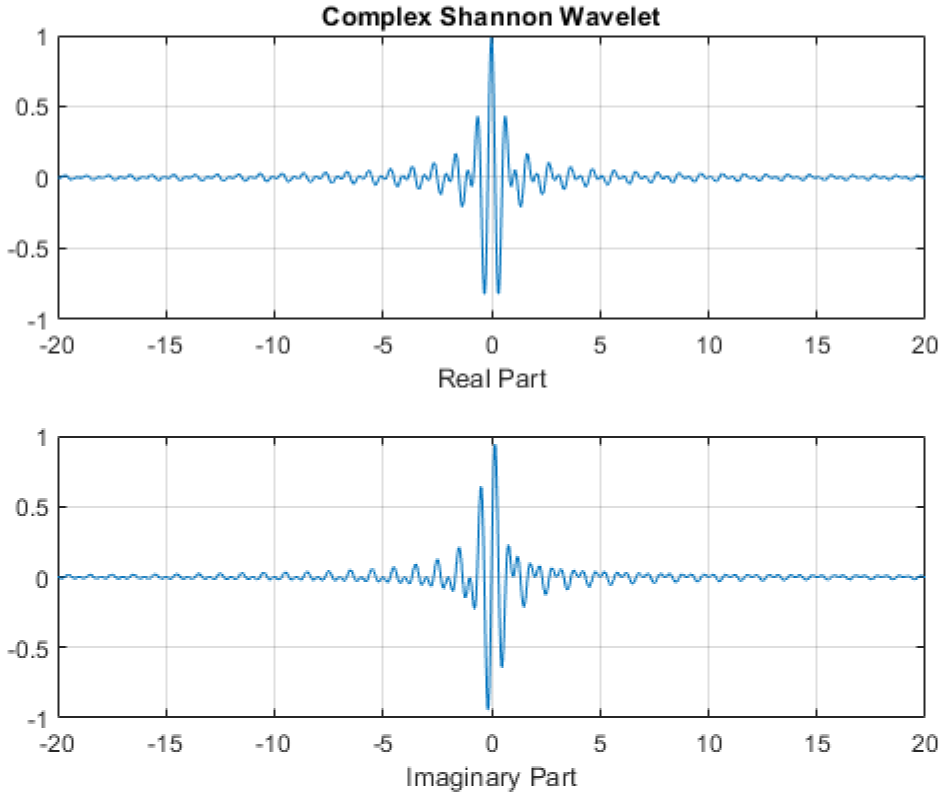
```
fb = 1;
fc = 1.5;
```

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(2,1,1)
plot(x,real(psi))
title('Complex Shannon Wavelet')
xlabel('Real Part')
grid on
subplot(2,1,2)
plot(x,imag(psi))
xlabel('Imaginary Part')
grid on
```



References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhäuser, p. 62.

See Also

`waveinfo`

Introduced before R2006a

shearletSystem

Cone-adapted bandlimited shearlet system

Description

The `shearletSystem` object represents a cone-adapted bandlimited shearlet system. After you create the shearlet system, you can use `sheart2` to obtain the shearlet transform of a real-valued 2-D image. You can also use `isheart2` to obtain the inverse transform. Additional “Object Functions” on page 1-1224 are provided.

Creation

Syntax

```
sls = shearletSystem
sls = shearletSystem(Name,Value)
```

Description

`sls = shearletSystem` creates a cone-adapted real-valued bandlimited shearlet system for a real-valued image of size 128-by-128 with the number of scales equal to 4. The system `sls` is a nondecimated shearlet system. Shearlets extending beyond the 2-D frequency bounds are periodically extended. Using real-valued shearlets with periodic boundary conditions results in real-valued shearlet coefficients.

The implementation of `shearletSystem` follows the approach described in Häuser and Steidl [6]

`sls = shearletSystem(Name,Value)` creates a cone-adapted bandlimited shearlet system with “Properties” on page 1-1223 specified by one or more `Name,Value` pairs. For example, `shearletSystem('ImageSize',[100 100])` creates a shearlet system for images of size 100-by-100. Properties can be specified in any order as `Name1,Value1,...,NameN,ValueN`. Enclose each property name in single quotes (`' '`) or double quotes (`" "`).

Note Property values of a shearlet system are fixed. For example, if the shearlet system `SLS` is created with an `ImageSize` of [128 128], you cannot change that `ImageSize` to [200 200].

Properties

`ImageSize` — Image size
`[128 128]` (default) | two-element integer-valued vector

Image size for the shearlet system, specified as a two-element integer-valued vector `[`*`numrows numcolumns`*`]`. Images must be at least 16-by-16.

Example: `sls = shearletSystem('ImageSize',[100 200])` creates a shearlet system for 100-by-200 images.

Data Types: single | double

NumScales — Number of scales
4 (default) | positive integer

Number of scales in the shearlet system, specified as a positive integer less than or equal to $\log_2(\min([M\ N]))$–3, where M and N are the row and column dimensions of the input image. For a 16-by-16 input image, $\log_2(\min([16\ 16]))$–3 = 4–3 = 1, so the smallest image compatible with `shearletSystem` has a minimum dimension of 16. For the default image size 128-by-128, the number of scales equals 4.

Example: `sls = shearletSystem('NumScales',1)` creates a shearlet system with `NumScales` equal to 1.

Data Types: single | double

TransformType — Shearlet system type
'real' (default) | 'complex'

Shearlet system type, specified as `'real'` or `'complex'`. Real-valued shearlets have two-sided 2-D frequency spectra, while complex-valued shearlets have one-sided 2-D spectra. If FilterBoundary is set to `'periodic'`, shearlets at the finest spatial scales have energy that wraps around in the 2-D frequency response. For both `'real'` and `'complex'` shearlet systems, the Fourier transforms of the shearlets are real valued.

FilterBoundary — Shearlet filter boundary handling
'periodic' (default) | 'truncated'

Shearlet filter boundary handling, specified as `'periodic'` or `'truncated'`. When set to `'periodic'`, shearlets extending beyond the 2-D frequency boundaries are periodically extended. When set to `'truncated'`, shearlets are truncated at the 2-D frequency boundaries.

PreserveEnergy — Shearlet system analysis normalization
false or 0 (default) | true or 1

Shearlet system analysis normalization, specified as a numeric or logical `1` (`true`) or `0` (`false`). When set to `true`, the shearlet system is normalized to be a Parseval frame, and the energy of the input image is preserved in the shearlet transform coefficients.

Example: `sls = shearletSystem('PreserveEnergy',true)`

Data Types: logical

Precision — Shearlet system precision
'double' (default) | 'single'

Shearlet system precision, specified as `'double'` or `'single'`. All computations are done using the specified precision.

Note To obtain the shearlet transform of an image, the precision of the image must match the precision of the shearlet system.

Object Functions

sheart2	Shearlet transform

isheart2	Inverse shearlet transform
framebounds	Shearlet system frame bounds
filterbank	Shearlet system filters
numshears	Number of shearlets

Examples

Create Energy-Preserving Shearlet System

Load an image. Create two real-valued shearlet systems that can be applied to the image. Normalize the first system so that energy is preserved in the shearlet transform coefficients. Leave the second shearlet system with the default (`false`) normalization.

```
load mask
[numRows,numCols] = size(X);
slsA = shearletSystem('ImageSize',[numRows numCols],'PreserveEnergy',true);
slsB = shearletSystem('ImageSize',[numRows numCols]);
```

Take the shearlet transform of the image using both shearlet systems.

```
cfA = sheart2(slsA,X);
cfB = sheart2(slsB,X);
```

Determine the energy of the input image and both sets of transform coefficients. Confirm that only the first shearlet system preserved energy.

```
energyA = sum(cfA(:).^2);
energyB = sum(cfB(:).^2);
energyImage = sum(X(:).^2)
```

```
energyImage = 2.4655e+09
```

```
diffSystemA = abs(energyImage-energyA)
```

```
diffSystemA = 4.7684e-07
```

```
diffSystemB = abs(energyImage-energyB)
```

```
diffSystemB = 1.4869e+07
```

Limitations

- Boundary effects of a real-valued shearlet transform of a non-square image can result in complex-valued coefficients. As implemented, `shearletSystem` constructs shearlets in the 2-D Fourier domain. For a real-valued shearlet transform, the shearlets in the 2-D Fourier domain should be symmetric in the positive and negative 2-D frequency plane. Shearlets constructed for square images are symmetric. However, as the image aspect ratio increases, the shearlets constructed become less symmetric. If the support of the lowpass filter in the 2-D frequency plane is too large, boundary effects can increase. Whenever possible, use square images. See “Boundary Effects in Real-Valued Bandlimited Shearlet Systems” for additional information and strategies to mitigate boundary effects.

References

[1] Guo, K., G. Kutyniok, and D. Labate. "Sparse multidimensional representations using anisotropic dilation and shear operators." In *Wavelets and Splines: Athens 2005* (G. Chen, and M.-J. Chen, eds.), 189–201. Brentwood, TN: Nashboro Press, 2006.

[2] Guo, K., and D. Labate. "Optimally Sparse Multidimensional Representation Using Shearlets." *SIAM Journal on Mathematical Analysis*. Vol. 39, Number 1, 2007, pp. 298–318.

[3] Kutyniok, G., and W.-Q Lim. "Compactly supported shearlets are optimally sparse." *Journal of Approximation Theory*. Vol. 163, Number 11, 2011, pp. 1564–1589.

[4] *Shearlets: Multiscale Analysis for Multivariate Data* (G. Kutyniok, and D. Labate, eds.). New York: Springer, 2012.

[5] *ShearLab*. `https://www3.math.tu-berlin.de/numerik/www.shearlab.org/`.

[6] Häuser, S., and G. Steidl. "Fast Finite Shearlet Transform: a tutorial." arXiv preprint arXiv:1202.1773 (2014).

Extended Capabilities

C/C++ Code Generation

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

See Also

`cwtft2` | `dddtree2`

Topics

“Shearlet Systems”
“Boundary Effects in Real-Valued Bandlimited Shearlet Systems”

Introduced in R2019b

sheart2

Shearlet transform

Syntax

```
coefs = sheart2(sls,im)
```

Description

`coefs = sheart2(sls,im)` returns the shearlet transform or shearlet analysis of the real-valued 2-D image `im` for the shearlet system `sls`. If the shearlet system is real-valued with periodic boundary conditions, then `coefs` is real-valued. Otherwise, `coefs` is complex-valued. The size and class (data type) of `im` must match the ImageSize and Precision values, respectively, of `sls`.

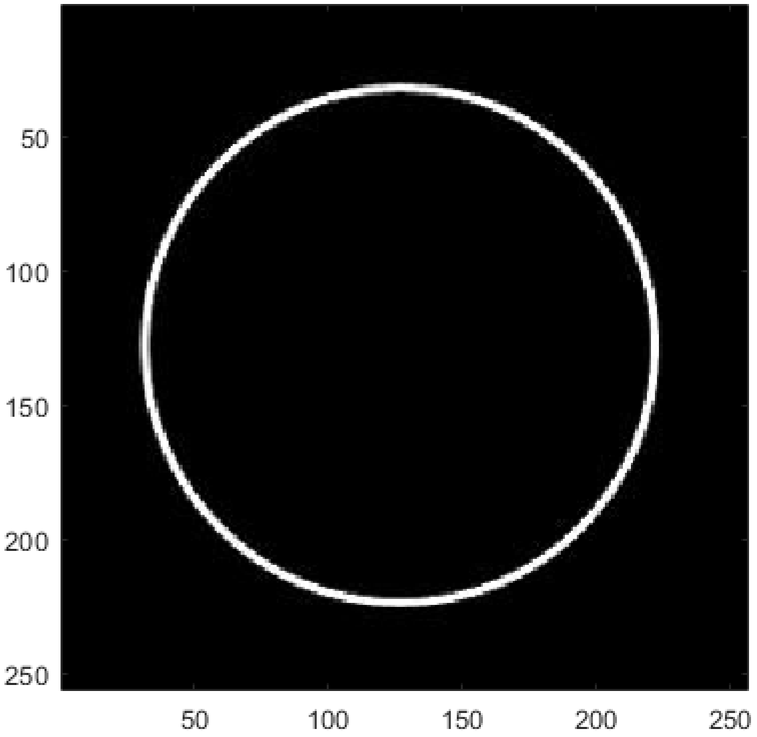
Examples

Shearlet Transform of Circle

This example shows how to take the shearlet transform of an image and reconstruct the image using only coefficients corresponding to zero shearing.

Load and display an image of a circle.

```
load circleGS
imagesc(circleGS)
colormap gray
axis equal
axis tight
```



Create a shearlet system that can be used with the image. Obtain the shearlet filters defined by the system, as well as their geometric interpretations.

```
[numRows,numCols] = size(circleGS);
sls = shearletSystem('ImageSize',[numRows numCols],'FilterBoundary','truncated');
[psi,scale,shear,cone] = filterbank(sls);
```

Obtain the shearlet transform of the image.

```
cfs = sheart2(sls,circleGS);
```

Find the indices of the shearlet filters that correspond to zero shearing. Keep in mind that the lowpass filter also corresponds to zero shearing.

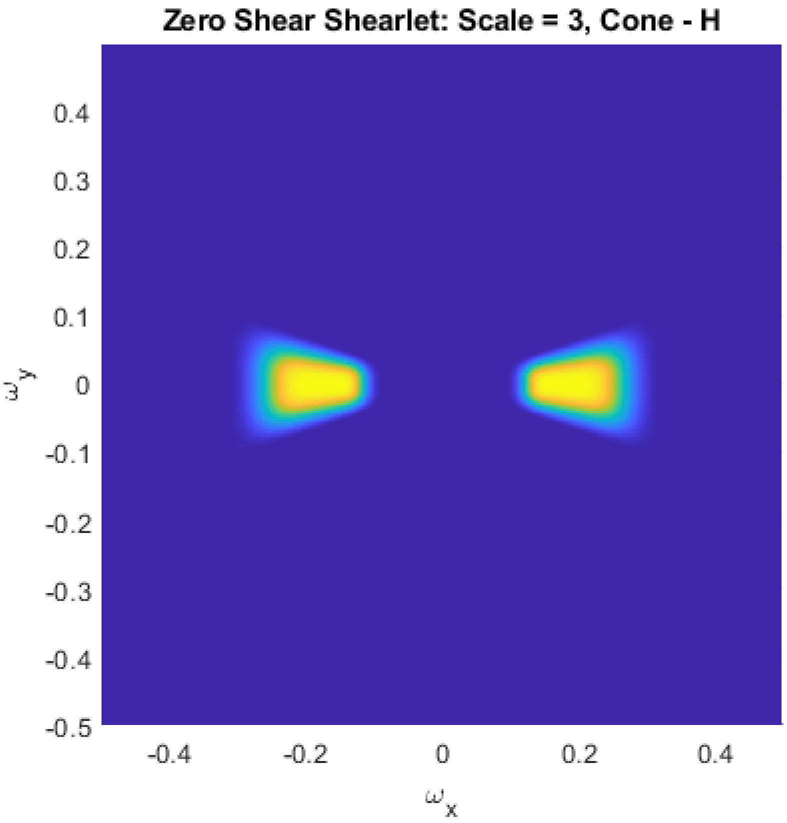
```
ind = find((shear==0).*(scale~=-1))'
```

```
ind = 1×10

     3     6    10    15    20    25    31    38    46    55
```

Plot one of the shearlets in the frequency plane. Because the shearlet corresponds to zero shearing, confirm the frequency response is concentrated along either the horizontal or vertical axis.

```
sh = 31;
omegax = -1/2:1/numCols:1/2-1/numCols;
omegay = omegax;
figure
```

```
surf(omegax,flip(omegay),psi(:,:,sh),'EdgeColor','none')
view(0,90)
xlabel('\omega_x')
ylabel('\omega_y')
axis equal
axis tight
title(['Zero Shear Shearlet: Scale = ',num2str(scale(sh)),', Cone - ',cone{sh}])
```

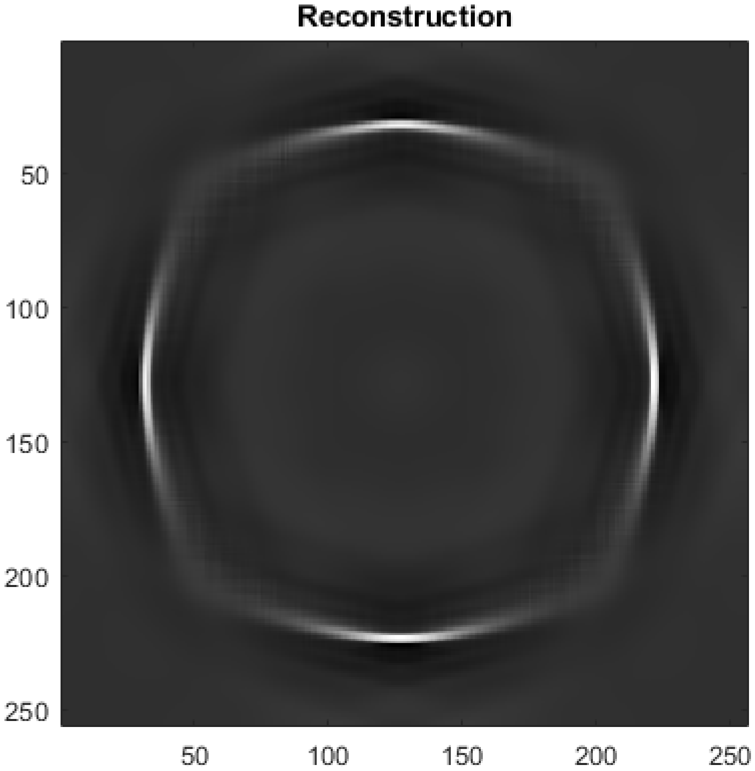


Create an array that only contains the shearlet coefficients that correspond to the zero shearing filters.

```
cfsx = zeros(size(cfs));
for k=1:length(ind)
    cfsx(:,:,ind(k)) = cfs(:,:,ind(k));
end
```

Reconstruct the image using the new coefficients array. Because the only nonzero shearlet coefficients are those that correspond to zero shearing, the horizontal and vertical portions of the circle are emphasized in the reconstruction.

```
rec = isheart2(sls,cfsx);
imagesc(rec)
axis equal
axis tight
colormap gray
title('Reconstruction')
```



Input Arguments

sls — Shearlet system
shearletSystem object

Shearlet system, specified as a `shearletSystem` object.

im — Input image
real-valued matrix

Input image, specified a real-valued matrix. The size and data type of `im` must match the ImageSize and Precision values, respectively, of `sls`.

Data Types: `single` | `double`

Output Arguments

coefs — Shearlet coefficients
3-D array

Shearlet coefficients, returned as a 3-D array. The size of `coefs` is *M*-by-*N*-by-*K*, where *M* and *N* are the row and column dimensions of the input image, respectively. The size of the third dimension, *K*, equals the number of shearlets in `sls`, including the lowpass filter, *K* = `numshears(sls)` + 1.

For example, if `cfs = sheart2(sls,im)` and `psi = filterbank(sls)`, then the shearlet corresponding to `cfs(:,:,k)` is `psi(:,:,k)`. The data type of `coefs` matches the Precision value of the shearlet system.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

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

See Also

`shearletSystem` | `isheart2`

Topics

“Shearlet Systems”

Introduced in R2019b

signalLabelDefinition

Create signal label definition

Description

Use `signalLabelDefinition` to create signal label definitions for data sets. The labels can correspond to attributes, regions, or points of interest. Use a vector of `signalLabelDefinition` objects to create a `labeledSignalSet`.

Creation

Syntax

```
sld = signalLabelDefinition(name)
sld = signalLabelDefinition(name,Name=Value)
```

Description

`sld = signalLabelDefinition(name)` creates a signal label definition object, `sld`, with the "Name" on page 1-0 property set to `name` and other properties set to default values.

`sld = signalLabelDefinition(name,Name=Value)` sets "Properties" on page 1-1232 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in quotes.

Input Arguments

`name` — Label name
character vector | string scalar

Label name, specified as a character vector or string scalar.

Data Types: `char` | `string`

Properties

`Name` — Name of label
character vector | string scalar

Name of label, specified as a character vector or string scalar.

Data Types: `char` | `string`

`LabelType` — Type of label
`"attribute"` (default) | `"roi"` | `"point"` | `"attributeFeature"` | `"roiFeature"`

Type of label, specified as one of these:

- "attribute" — Define signal characteristics.
- "roi" — Define signal characteristics over regions of interest.
- "point" — Define signal characteristics over points of interest.
- "attributeFeature" — Define signal characteristics that correspond to features.
- "roiFeature" — Define signal characteristics over regions of interest that correspond to features.

Data Types: char | string

LabelDataType — Data type of label
"logical" (default) | "categorical" | "numeric" | "string" | "table" | "timetable"

Data type of label, specified as "logical", "categorical", "numeric", "string", "table", or "timetable". When you set this property to "categorical", use the “Categories” on page 1-0 property to specify the array of categories. The object does not support timetable and table data types for attributeFeature and roiFeature labels.

Data Types: char | string

Categories — Label category names
string array | cell array of character vectors

Label category names, specified as a string array or a cell array of character vectors. The array must have unique elements. This property applies only when the “LabelDataType” on page 1-0 property is set to "categorical".

Example: LabelDataType="categorical",Categories=["apple","orange"]

Data Types: char | string

ROILimitsDataType — Data type of ROI limits
"double" (default) | "duration"

Data type of ROI limits, specified as either "double" or "duration". This property applies only when “LabelType” on page 1-0 is set to "roi".

Data Types: char | string

PointLocationsDataType — Data type of point locations
"double" (default) | "duration"

Data type of point locations, specified as either "double" or "duration". This property applies only when “LabelType” on page 1-0 is set to "point".

Data Types: char | string

ValidationFunction — Validation function
function handle

Validation function, specified as a function handle and used when setting label values in a labeledSignalSet object. This property applies only when “LabelDataType” on page 1-0 is set to "categorical", "logical", "numeric", "table", or "timetable". If not specified, the function checks only that its input values are of the correct data type. If “LabelDataType” on page 1-0 is set to "categorical", the function checks that the input is one of the values specified using “Categories” on page 1-0. The function takes an input value and returns true if the value is valid and false if the value is invalid.

Example: `LabelDataType="numeric",DefaultValue=1,ValidationFunction=@(x)x<2`

Data Types: `function_handle`

`DefaultValue` — Default value of label

`[]` (default) | `LabelDataType` value

Default value of label, specified as a value of the type specified using “LabelDataType” on page 1-0. If “LabelDataType” on page 1-0 is set to `"categorical"`, then “DefaultValue” on page 1-0 must be one of the values specified using “Categories” on page 1-0.

Example:
`LabelDataType="categorical",Categories=["apple","orange"],DefaultValue="apple"`

Data Types: `char` | `double` | `logical` | `string` | `table`

`Description` — Label description

character vector | string scalar

Label description, specified as a character vector or string scalar.

Example: `Description="Patient is asleep"`

Data Types: `char` | `string`

`Tag` — Label tag identifier

character vector | string scalar

Label tag identifier, specified as a character vector or string scalar. Use this property to identify the same label in a larger labeling scheme or public labeling set.

Example: `Tag="Peak1"`

Data Types: `char` | `string`

`Sublabels` — Array of sublabels

signal label definition object

Array of sublabels, specified as a signal label definition object. To specify more than one sublabel, set this property to a vector of signal label definition objects. Use this property to create a relationship between a parent label and its children. If “LabelType” (Signal Processing Toolbox) is set to `"attributeFeature"` or `"roiFeature"`, then this property does not apply.

Note Sublabels cannot have sublabels.

Example:
`Sublabels=[signalLabelDefinition("negative"),signalLabelDefinition("positive")]`

`FrameSize` — Frame size

numeric scalar

Frame size, specified as a numeric scalar. You must specify `FrameSize` when “LabelType” (Signal Processing Toolbox) is set to `"roiFeature"`.

Example: `FrameSize=50`

Data Types: `double`

`FrameOverlapLength` — Overlap length of adjacent frames

`0` (default) | numeric scalar

Overlap length of adjacent frames, specified as a numeric scalar. To enable this property, set “LabelType” (Signal Processing Toolbox) to `"roiFeature"`. You cannot specify `FrameOverlapLength` and `FrameRate` simultaneously. If you do not specify `FramerOverlapLength`, then the object assumes the overlap length to be zero.

Example: `FrameSize=50,FrameOverlapLength=5`

Data Types: `double`

`FrameRate` — Frame rate

`0` (default) | numeric scalar

Frame rate, specified as a numeric scalar. To enable this property, set “LabelType” (Signal Processing Toolbox) to `"roiFeature"`. You cannot specify `FrameRate` and `FrameOverlapLength` simultaneously. If you do not specify `FrameRate`, then the object assumes no overlap between frames.

Example: `FrameSize=50,FrameRate=45`

Data Types: `double`

Object Functions

labelDefinitionsHierarchy	Get hierarchical list of label and sublabel names
labelDefinitionsSummary	Get summary table of signal label definitions

Examples

Label Definitions for Whale Songs

Consider a set of whale sound recordings. The recorded whale sounds consist of trills and moans. *Trills* sound like series of clicks. *Moans* are low-frequency cries similar to the sound made by a ship's horn. You want to look at each signal and label it to identify the whale type, the trill regions, and the moan regions. For each trill region, you also want to label the signal peaks higher than a certain threshold.

Signal Label Definitions

Define an attribute label to store whale types. The possible categories are blue whale, humpback whale, and white whale.

```
dWhaleType = signalLabelDefinition('WhaleType',...
   'LabelType','attribute',...
   'LabelDataType','categorical',...
   'Categories',{'blue','humpback','white'},...
   'Description','Whale type');
```

Define a region-of-interest (ROI) label to capture moan regions. Define another ROI label to capture trill regions.

```
dMoans = signalLabelDefinition('MoanRegions',...
   'LabelType','roi',...
   'LabelDataType','logical',...
   'Description','Regions where moans occur');

dTrills = signalLabelDefinition('TrillRegions',...
   'LabelType','roi',...
   'LabelDataType','logical',...
   'Description','Regions where trills occur');
```

Finally, define a point label to capture the trill peaks. Set this label as a sublabel of the `dTrills` definition.

```
dTrillPeaks = signalLabelDefinition('TrillPeaks',...
   'LabelType','point',...
   'LabelDataType','numeric',...
   'Description','Trill peaks');

dTrills.Sublabels = dTrillPeaks;
```

Labeled Signal Set

Create a `labeledSignalSet` with the whale signals and the label definitions. Add label values to identify the whale type, the moan and trill regions, and the peaks of the trills.

```
load labelwhalesignals
lbldefs = [dWhaleType dMoans dTrills];

lss = labeledSignalSet({whale1 whale2},lbldefs,'MemberNames',{'Whale1','Whale2'}, ...
   'SampleRate',Fs,'Description','Characterize whale song regions');
```

Visualize the label hierarchy and label properties using `labelDefinitionsHierarchy` and `labelDefinitionsSummary`.

```
labelDefinitionsHierarchy(lss)
```

```
ans = 
    'WhaleType
       Sublabels: []
     MoanRegions
       Sublabels: []
     TrillRegions
       Sublabels: TrillPeaks
     '
```

```
labelDefinitionsSummary(lss)
```

```
ans=3×9 table
      LabelName         LabelType      LabelDataType      Categories       ValidationFunction     Defau
    ______________    ___________    ______________    ____________    __________________     _____

    "WhaleType"       "attribute"    "categorical"     {3x1 string}       {["N/A"    ]}       {0x0
    "MoanRegions"     "roi"          "logical"         {["N/A"    ]}      {0x0 double}        {0x0
    "TrillRegions"    "roi"          "logical"         {["N/A"    ]}      {0x0 double}        {0x0
```

The signals in the loaded data correspond to songs of two blue whales. Set the `'WhaleType'` values for both signals.

```
setLabelValue(lss,1,'WhaleType','blue');
setLabelValue(lss,2,'WhaleType','blue');
```

Visualize the `'Labels'` property. The table has the newly added `'WhaleType'` values for both signals.

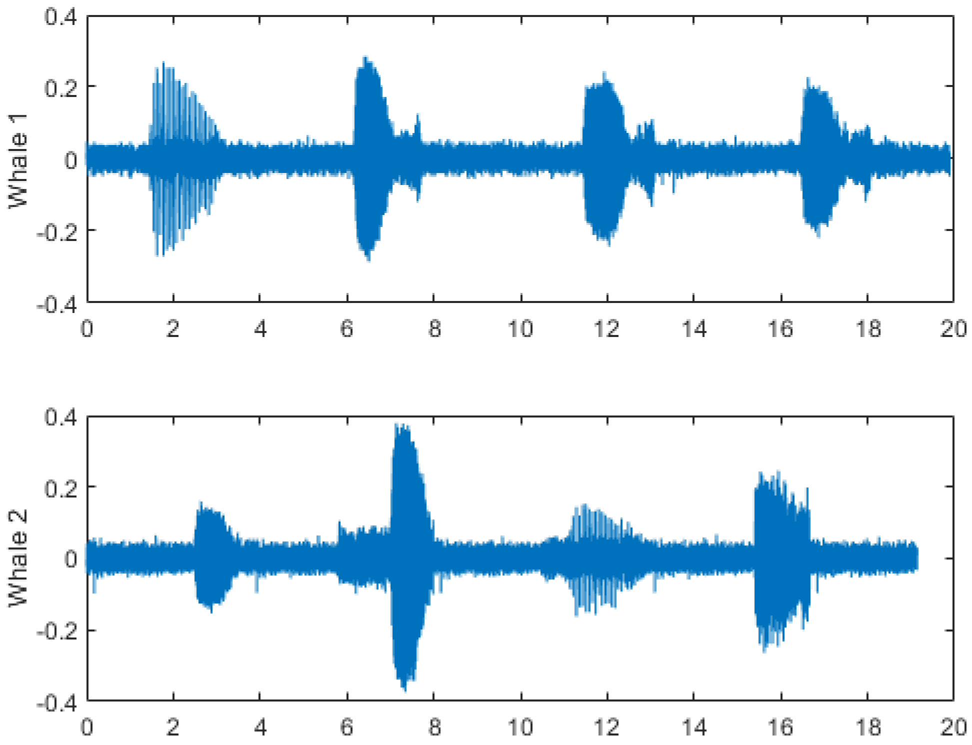
```
lss.Labels
```

```
ans=2×3 table
                 WhaleType     MoanRegions     TrillRegions 
                 _________    _____________    _____________

    Whale1         blue       {0x2 table}      {0x3 table}
    Whale2         blue       {0x2 table}      {0x3 table}
```

Visualize Region Labels

Visualize the whale songs to identify the trill and moan regions.

```
subplot(2,1,1)
plot((0:length(whale1)-1)/Fs,whale1)
ylabel('Whale 1')

subplot(2,1,2)
plot((0:length(whale2)-1)/Fs,whale2)
ylabel('Whale 2')
```



Moan regions are sustained low-frequency wails.

- whale1 has moans centered at about 7 seconds, 12 seconds, and 17 seconds.
- whale2 has moans centered at about 3 seconds, 7 seconds, and 16 seconds.

Add the moan regions to the labeled set. Specify the ROI limits in seconds and the label values.

```
moanRegionsWhale1 = [6.1 7.7; 11.4 13.1; 16.5 18.1];
mrsz1 = [size(moanRegionsWhale1,1) 1];
setLabelValue(lss,1,'MoanRegions',moanRegionsWhale1,true(mrsz1));

moanRegionsWhale2 = [2.5 3.5; 5.8 8; 15.4 16.7];
mrsz2 = [size(moanRegionsWhale2,1) 1];
setLabelValue(lss,2,'MoanRegions',moanRegionsWhale2,true(mrsz2));
```

Trill regions have distinct bursts of sound punctuated by silence.

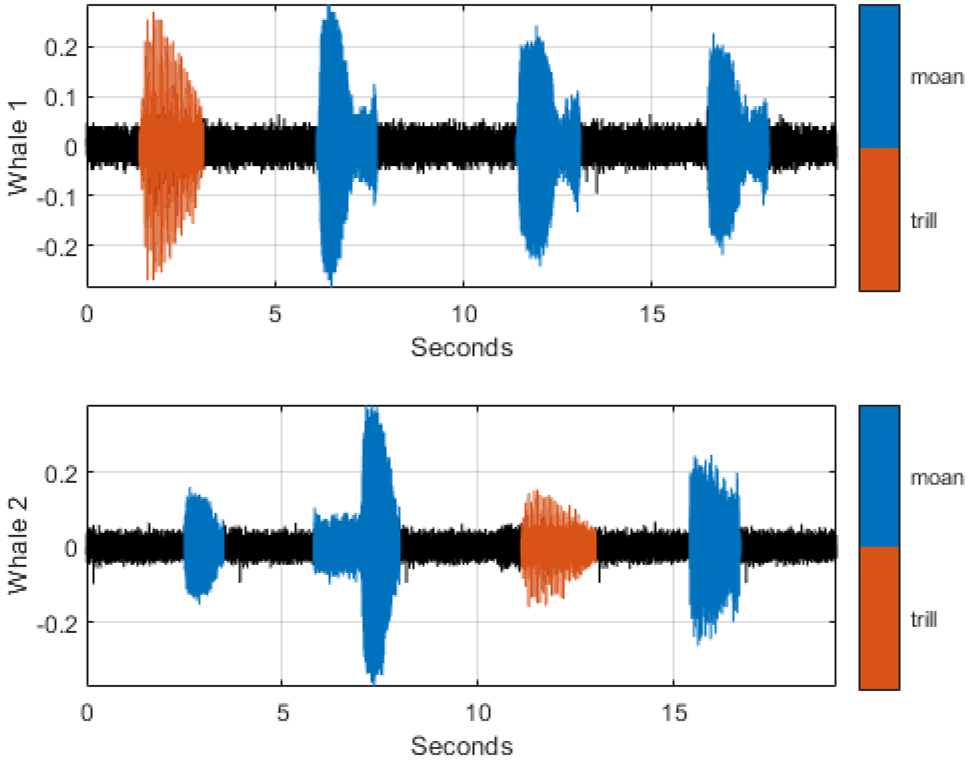
- whale1 has a trill centered at about 2 seconds.
- whale2 has a trill centered at about 12 seconds.

Add the trill regions to the labeled set.

```
trillRegionWhale1 = [1.4 3.1];
trsz1 = [size(trillRegionWhale1,1) 1];
setLabelValue(lss,1,'TrillRegions',trillRegionWhale1,true(trsz1));

trillRegionWhale2 = [11.1 13];
trsz2 = [size(trillRegionWhale1,1) 1];
setLabelValue(lss,2,'TrillRegions',trillRegionWhale2,true(trsz2));
```

Create a signalMask (Signal Processing Toolbox) object for each whale song and use it to visualize and label the different regions. For better visualization, change the label values from logical to categorical.

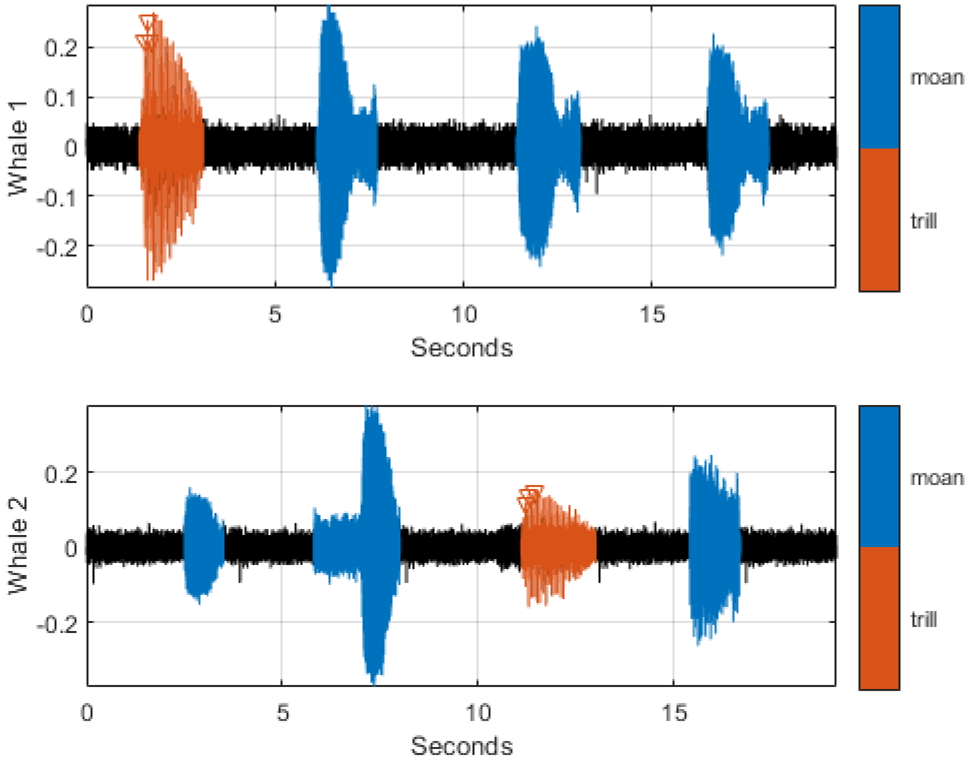
```
mr1 = getLabelValues(lss,1,'MoanRegions');
mr1.Value = categorical(repmat("moan",mrsz1));
tr1 = getLabelValues(lss,1,'TrillRegions');
tr1.Value = categorical(repmat("trill",trsz1));

msk1 = signalMask([mr1;tr1],'SampleRate',Fs);

subplot(2,1,1)
plotsigroi(msk1,whale1)
ylabel('Whale 1')
hold on

mr2 = getLabelValues(lss,2,'MoanRegions');
mr2.Value = categorical(repmat("moan",mrsz2));
tr2 = getLabelValues(lss,2,'TrillRegions');
tr2.Value = categorical(repmat("trill",trsz2));

msk2 = signalMask([mr2;tr2],'SampleRate',Fs);

subplot(2,1,2)
plotsigroi(msk2,whale2)
ylabel('Whale 2')
hold on
```



Visualize Point Labels

Label three peaks for each trill region. For point labels, you specify the point locations and the label values. In this example, the point locations are in seconds.

```
peakLocsWhale1 = [1.553 1.626 1.7];
peakValsWhale1 = [0.211 0.254 0.211];

setLabelValue(lss,1,{'TrillRegions','TrillPeaks'}, ...
   peakLocsWhale1,peakValsWhale1,'LabelRowIndex',1);

subplot(2,1,1)
plot(peakLocsWhale1,peakValsWhale1,'v')
hold off

peakLocsWhale2 = [11.214 11.288 11.437];
peakValsWhale2 = [0.119 0.14 0.15];

setLabelValue(lss,2,{'TrillRegions','TrillPeaks'}, ...
   peakLocsWhale2,peakValsWhale2,'LabelRowIndex',1);

subplot(2,1,2)
plot(peakLocsWhale2,peakValsWhale2,'v')
hold off
```



Explore Label Values

Explore the label values using `getLabelValues`.

```
getLabelValues(lss)
```

```
ans=2×3 table
                  WhaleType    MoanRegions    TrillRegions
                  _________    ___________    ____________

    Whale1          blue       {3x2 table}    {1x3 table} 
    Whale2          blue       {3x2 table}    {1x3 table} 
```

Retrieve the moan regions for the first member of the labeled set.

```
getLabelValues(lss,1,'MoanRegions')
```

```
ans=3×2 table
     ROILimits       Value
    ____________     _____

     6.1     7.7     {[1]}
    11.4    13.1     {[1]}
    16.5    18.1     {[1]}
```

Use a second output argument to list the sublabels of a label.

```
[value,valueWithSublabel] = getLabelValues(lss,1,'TrillRegions')
```

```
value=1×2 table
    ROILimits     Value
    __________    _____

    1.4    3.1    {[1]}


valueWithSublabel=1×3 table
    ROILimits     Value      Sublabels
                            TrillPeaks

    __________    _____    ___________

    1.4    3.1    {[1]}    {3x2 table}
```

To retrieve the values in a sublabel, express the label name as a two-element array.

```
getLabelValues(lss,1,{'TrillRegions','TrillPeaks'})
```

```
ans=3×2 table
    Location       Value
    ________    __________

     1.553      {[0.2110]}
     1.626      {[0.2540]}
       1.7      {[0.2110]}
```

Find the value of the third trill peak corresponding to the second member of the set.

```
getLabelValues(lss,2,{'TrillRegions','TrillPeaks'}, ...
    'LabelRowIndex',1,'SublabelRowIndex',3)
```

```
ans=1×2 table
    Location       Value
    ________    __________

     11.437     {[0.1500]}
```

Count Label Values and Create Datastores

Specify the path to a set of audio signals included as MAT-files with MATLAB®. Each file contains a signal variable and a sample rate. List the names of the files.

```
folder = fullfile(matlabroot,"toolbox","matlab","audiovideo");
lst = dir(append(folder,"/*.mat"));
nms = {lst(:).name}'
```

```
nms = 7x1 cell
    {'chirp.mat'   }
    {'gong.mat'    }
    {'handel.mat'  }
    {'laughter.mat'}
    {'mtlb.mat'    }
    {'splat.mat'   }
    {'train.mat'   }
```

Create a signal datastore that points to the specified folder. Set the sample rate variable name to `Fs`, which is common to all files. Generate a subset of the datastore that excludes the file `mtlb.mat`. Use the subset datastore as the source for a `labeledSignalSet` object.

```
sds = signalDatastore(folder,"SampleRateVariableName","Fs");
sds = subset(sds,~strcmp(nms,"mtlb.mat"));
lss = labeledSignalSet(sds);
```

Create three label definitions to label the signals:

- Define a logical attribute label that is true for signals that contain human voices.
- Define a numeric point label that marks the location and amplitude of the maximum of each signal.
- Define a categorical region-of-interest (ROI) label to pick out nonoverlapping, uniform-length random regions of each signal.

Add the signal label definitions to the labeled signal set.

```
vc = signalLabelDefinition("Voice",'LabelType','attribute', ...
    'LabelDataType','logical','DefaultValue',false);
mx = signalLabelDefinition("Maximum",'LabelType','point', ...
    'LabelDataType','numeric');
rs = signalLabelDefinition("RanROI",'LabelType','ROI', ...
    'LabelDataType','categorical','Categories',["ROI" "other"]);
addLabelDefinitions(lss,[vc mx rs])
```

Label the signals:

- Label `'handel.mat'` and `'laughter.mat'` as having human voices.
- Use the `islocalmax` function to find the maximum of each signal. Label its location and value.
- Use the `randROI` on page 1-0 function to generate as many regions of length *N*/10 samples as can fit in a signal of length *N* given a minimum separation of *N*/6 samples between regions. Label their locations and assign them to the `ROI` category.

When labeling points and regions, convert sample values to time values. Subtract 1 to account for MATLAB® array indexing and divide by the sample rate.

```
kj = 1;
while hasdata(sds)

    [sig,info] = read(sds);
    fs = info.SampleRate;

    [~,fn] = fileparts(info.FileName);
    if fn=="handel" || fn=="laughter"
        setLabelValue(lss,kj,"Voice",true)
    end

    xm = find(islocalmax(sig,'MaxNumExtrema',1));
    setLabelValue(lss,kj,"Maximum",(xm-1)/fs,sig(xm))

    N = length(sig);
    rois = randROI(N,round(N/10),round(N/6));
    setLabelValue(lss,kj,"RanROI",(rois-1)/fs,repelem("ROI",size(rois,1)))

    kj = kj+1;

end
```

Verify that only two signals contain voices.

```
countLabelValues(lss,"Voice")
```

```
ans=2×3 table
    Voice    Count    Percent
    _____    _____    _______

    false      4      66.667 
    true       2      33.333 
```

Verify that two signals have a maximum amplitude of 1.

```
countLabelValues(lss,"Maximum")
```

```
ans=5×4 table
           Maximum             Count    Percent    MemberCount
    ______________________    _____    _______    ___________

    0.80000000000000004441      1      16.667          1      
    0.89113331915798421612      1      16.667          1      
    0.94730769230769229505      1      16.667          1      
    1                           2      33.333          2      
    1.0575668990330560071       1      16.667          1      
```

Verify that each signal has four nonoverlapping random regions of interest.

```
countLabelValues(lss,"RanROI")
```

```
ans=2×4 table
    RanROI    Count    Percent    MemberCount
    ______    _____    _______    ___________

    ROI        24        100           6     
    other       0          0           0     
```

Create two datastores with the data in the labeled signal set:

- The `signalDatastore` (Signal Processing Toolbox) object `sd` contains the signal data.
- The `arrayDatastore` object `ld` contains the labeling information. Specify that you want to include the information corresponding to all the labels you created.

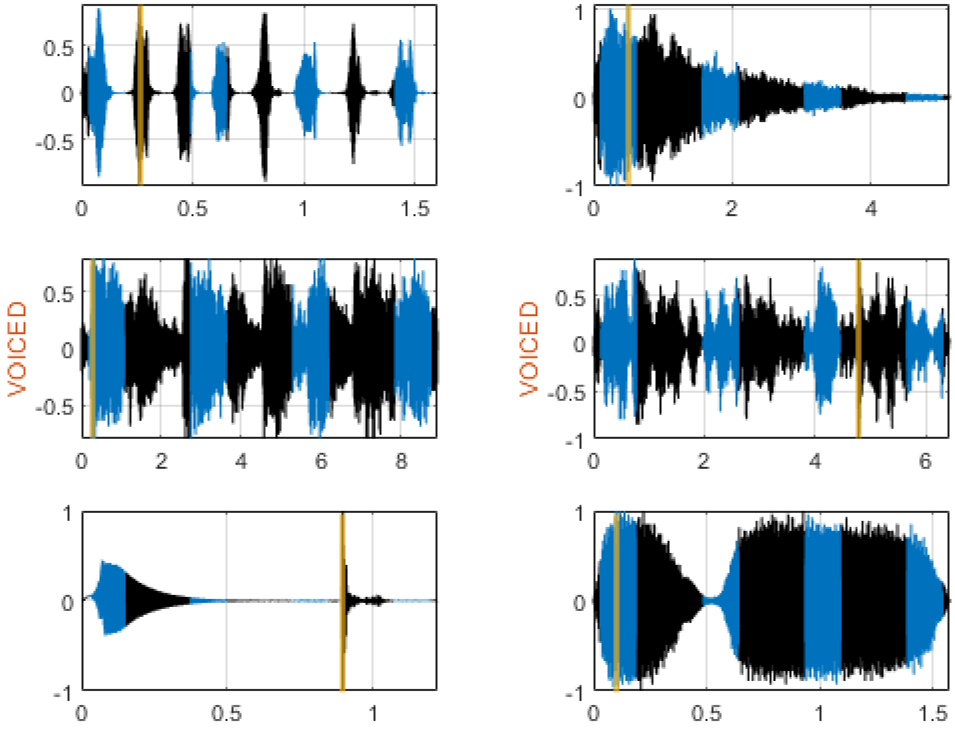
```
[sd,ld] = createDatastores(lss,["Voice" "RanROI" "Maximum"]);
```

Use the information in the datastores to plot the signals and display their labels.

- Use a `signalMask` (Signal Processing Toolbox) object to highlight the regions of interest in blue.
- Plot yellow lines to mark the locations of the maxima.

- Add a red axis label to the signals that contain human voices.

```
tiledlayout flow

while hasdata(sd)

    [sg,nf] = read(sd);

    lbls = read(ld);

    nexttile

    msk = signalMask(lbls{:}.RanROI{:},'SampleRate',nf.SampleRate);
    plotsigroi(msk,sg)
    colorbar off
    xlabel('')

    xline(lbls{:}.Maximum{:}.Location, ...
        'LineWidth',2,'Color','#EDB120')

    if lbls{:}.Voice{:}
        ylabel('VOICED','Color','#D95319')
    end

end
```



```
function roilims = randROI(N,wid,sep)
```

```
num = floor((N+sep)/(wid+sep));
hq = histcounts(randi(num+1,1,N-num*wid-(num-1)*sep),(1:num+2)-1/2);
roilims = (1 + (0:num-1)*(wid+sep) + cumsum(hq(1:num)))' + [0 wid-1];

end
```

See Also

Apps
Signal Labeler

Objects
labeledSignalSet | signalMask

Introduced in R2018b

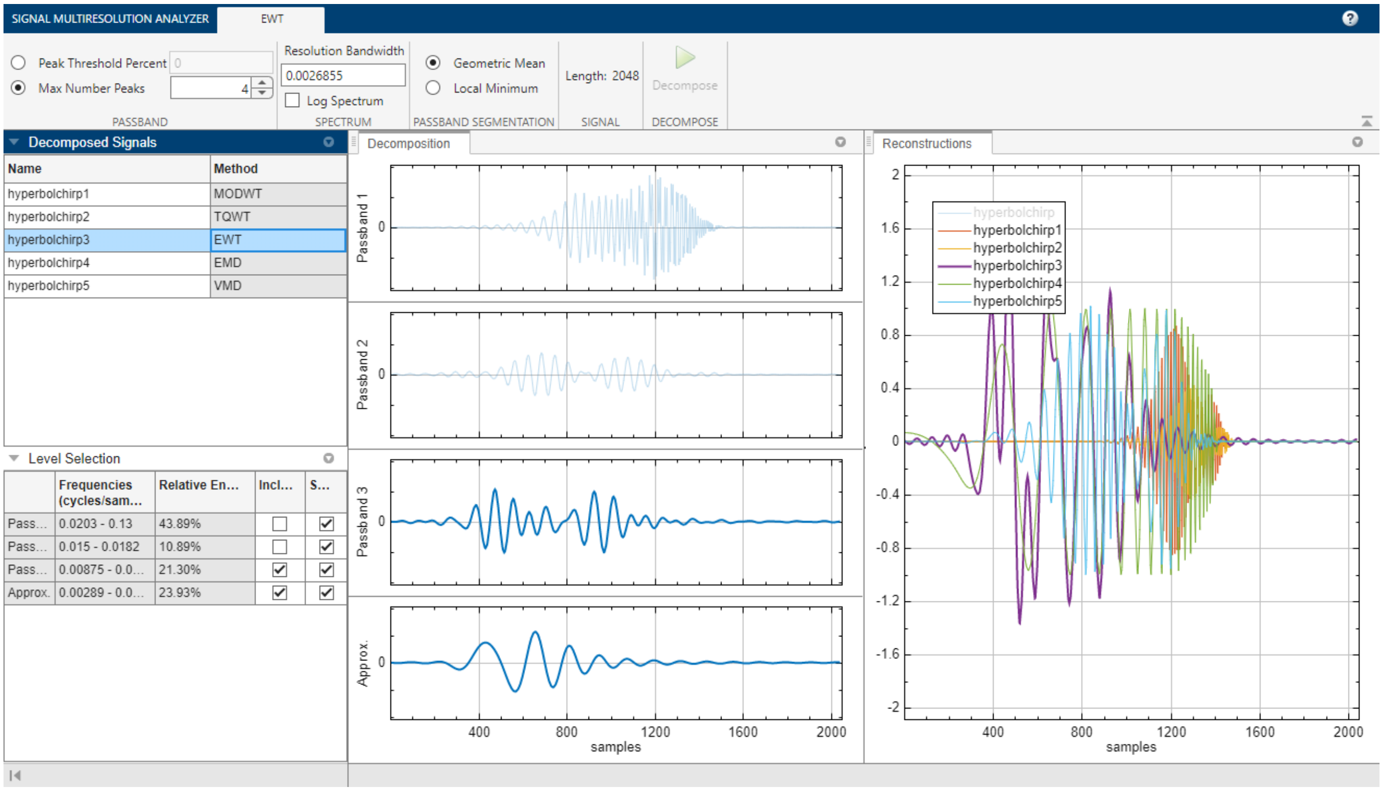
Signal Multiresolution Analyzer

Decompose signals into time-aligned components

Description

The **Signal Multiresolution Analyzer** app is an interactive tool for visualizing multilevel wavelet- and data adaptive-based decompositions of real-valued 1-D signals and comparing results. The app supports single- and double-precision data. With the app, you can:

- Access all the real-valued 1-D signals in your MATLAB workspace.
- Generate decompositions using fixed-bandwidth and data-adaptive multiresolution analysis (MRA) methods:
  - Fixed-bandwidth: Maximal overlap discrete wavelet transform (MODWT) (default), and empirical mode decomposition (EMD)
  - Data-adaptive: Empirical wavelet transform (EWT), tunable Q-factor wavelet transform (TQWT), and variational mode decomposition (VMD)
- Adjust default parameters, and visualize and compare multiple decompositions.
- Choose decomposition levels to include in the signal reconstruction.
- Obtain frequency ranges of the decomposition levels.
- Determine the relative energy of the signal across levels.
- Export reconstructed signals and decompositions to your workspace.
- Recreate decompositions in your workspace by generating MATLAB scripts.



○ Peak Threshold Percent 0

● Max Number Peaks 4

Resolution Bandwidth 0.0026855

□ Log Spectrum

● Geometric Mean

○ Local Minimum

Length: 2048

Decomposed Signals

Name	Method
hyperbolchirp1	MODWT
hyperbolchirp2	TQWT
hyperbolchirp3	EWT
hyperbolchirp4	EMD
hyperbolchirp5	VMD

Level Selection

	Frequencies (cycles/sam...	Relative En...	Incl...	S...
Pass...	0.0203 - 0.13	43.89%	□	☑
Pass...	0.015 - 0.0182	10.89%	□	☑
Pass...	0.00875 - 0.0...	21.30%	☑	☑
Approx.	0.00289 - 0.0...	23.93%	☑	☑

Open the Signal Multiresolution Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `signalMultiresolutionAnalyzer`.

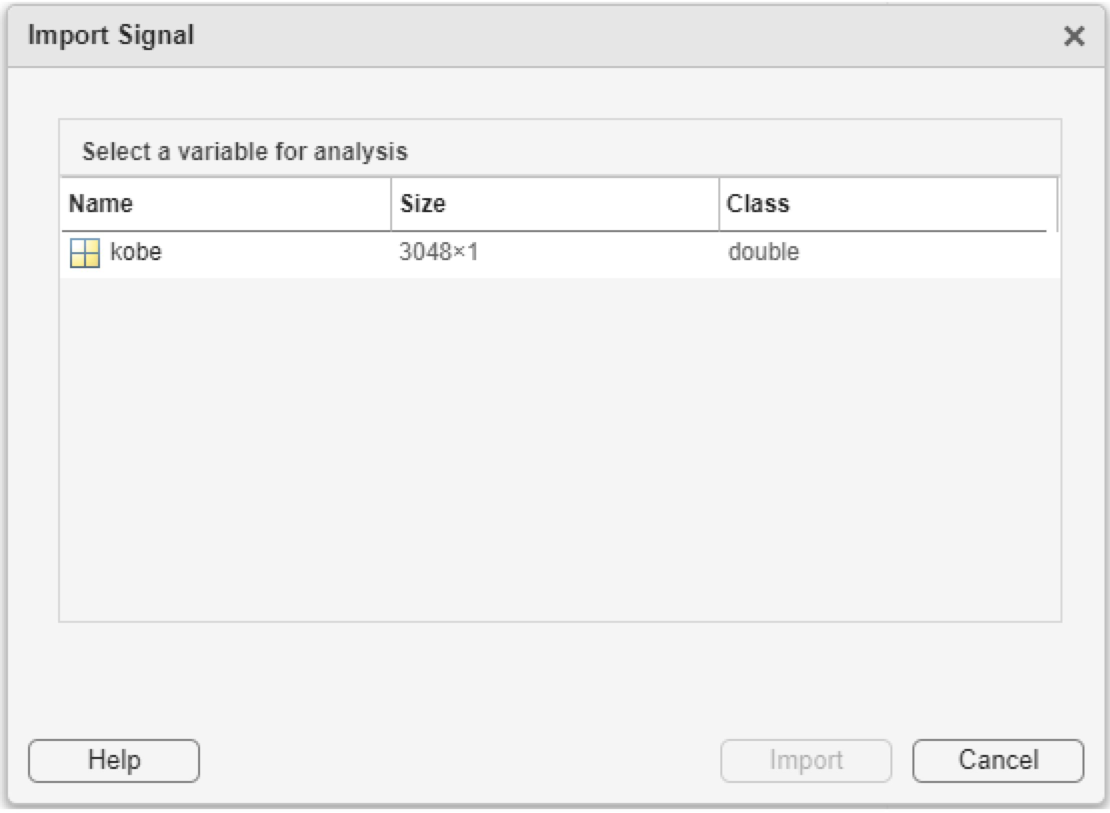
Examples

Visualize Time-Aligned MODWTMRA Decomposition

Load in the Kobe earthquake data. The data are seismograph measurements (vertical acceleration in nm/sec^2) recorded at Tasmania University, Hobart, Australia, on 16 January 1995, beginning at 20:56:51 (GMT) and continuing for 51 minutes at one second intervals.

```
load kobe
```

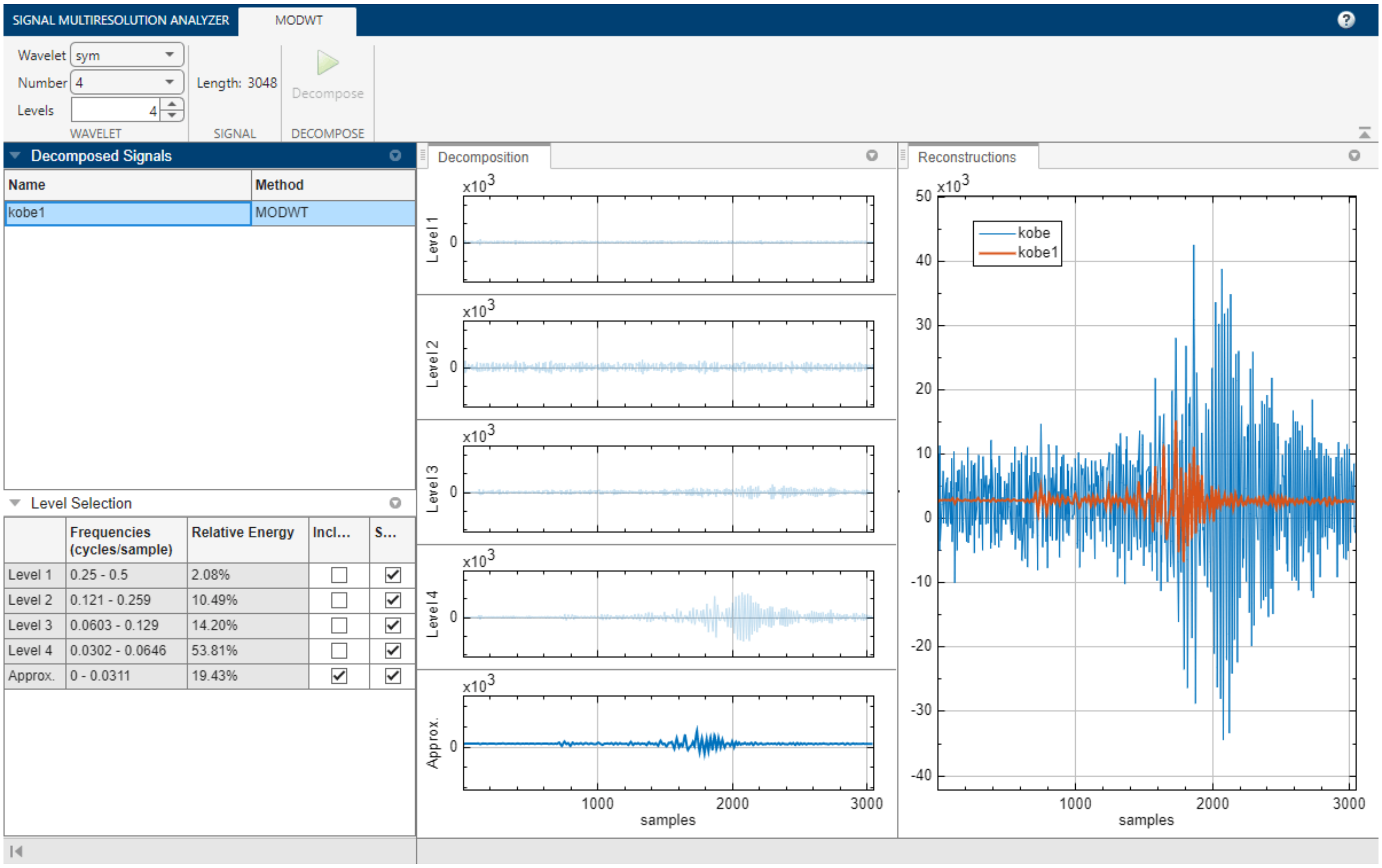
Open **Signal Multiresolution Analyzer** and click **Import**. A window appears listing all the workspace variables the app can process.



Select a variable for analysis

Name	Size	Class
▦ kobe	3048×1	double

Select the Kobe data from the dialog box and click **Import**. By default, a four-level MODWTMRA decomposition of the signal appears in the **MODWT** tab. The decomposition is obtained using the `modwt` and `modwtmra` functions with default settings. The plots in the **Decomposition** pane are the projections of the wavelet decompositions of the signal at each scale on the original signal subspace. The decomposed signal is named `kobe1` in the **Decomposed Signals** pane. The method `MODWT` identifies the decomposition. The original signal, `kobe`, and the reconstruction, `kobe1`, are plotted in the **Reconstructions** pane.



Wavelet sym
Number 4
Levels 4
Length: 3048

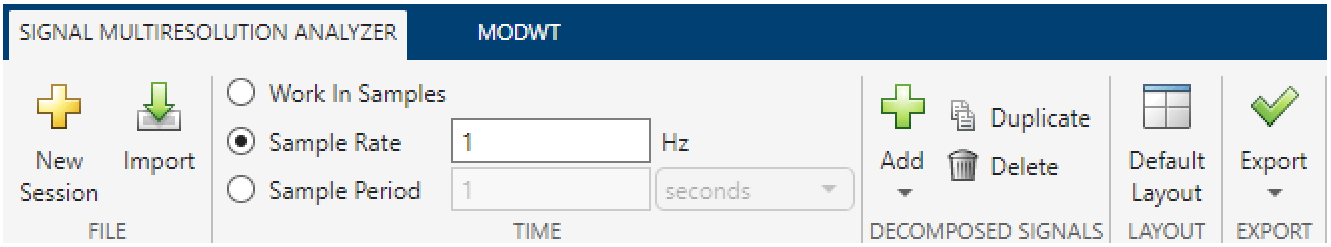
Decomposed Signals

Name	Method
kobe1	MODWT

Level Selection

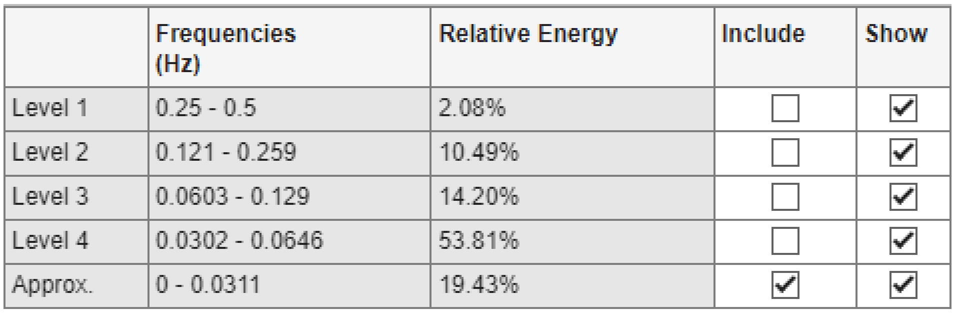
	Frequencies (cycles/sample)	Relative Energy	Incl...	S...
Level 1	0.25 - 0.5	2.08%	□	☑
Level 2	0.121 - 0.259	10.49%	□	☑
Level 3	0.0603 - 0.129	14.20%	□	☑
Level 4	0.0302 - 0.0646	53.81%	□	☑
Approx.	0 - 0.0311	19.43%	☑	☑

By default, plots are with respect to sample index and frequencies are in cycles per sample. To plot with respect to time and display frequencies in hertz, select the **Sample Rate** radio button on the **Signal Multiresolution Analyzer** tab. The default sample rate is 1 hertz. The plots and frequencies update to use the sample rate.



○ Work In Samples
● Sample Rate 1 Hz
○ Sample Period 1 seconds

The **Level Selection** pane shows the relative energies of the signal across scales, as well as the frequency bands.



	Frequencies (Hz)	Relative Energy	Include	Show
Level 1	0.25 - 0.5	2.08%	☐	☑
Level 2	0.121 - 0.259	10.49%	☐	☑
Level 3	0.0603 - 0.129	14.20%	☐	☑
Level 4	0.0302 - 0.0646	53.81%	☐	☑
Approx.	0 - 0.0311	19.43%	☑	☑

A check box in the **Show** column controls whether or not that level is displayed in the **Decomposition** pane. A check box in the **Include** column controls whether or not to include that level of the wavelet decomposition in the reconstruction. Clicking a plot in the **Decomposition** pane is another way to include or exclude that level in the signal reconstruction.

To generate a new decomposition, change one of the wavelet parameters in the toolstrip on the **MODWT** tab and click **Decompose**.

- **Wavelet** - Wavelet family
- **Number** - Wavelet filter number
- **Level** - Wavelet decomposition level

Changing any parameter in the toolstrip enables the **Decompose** button.

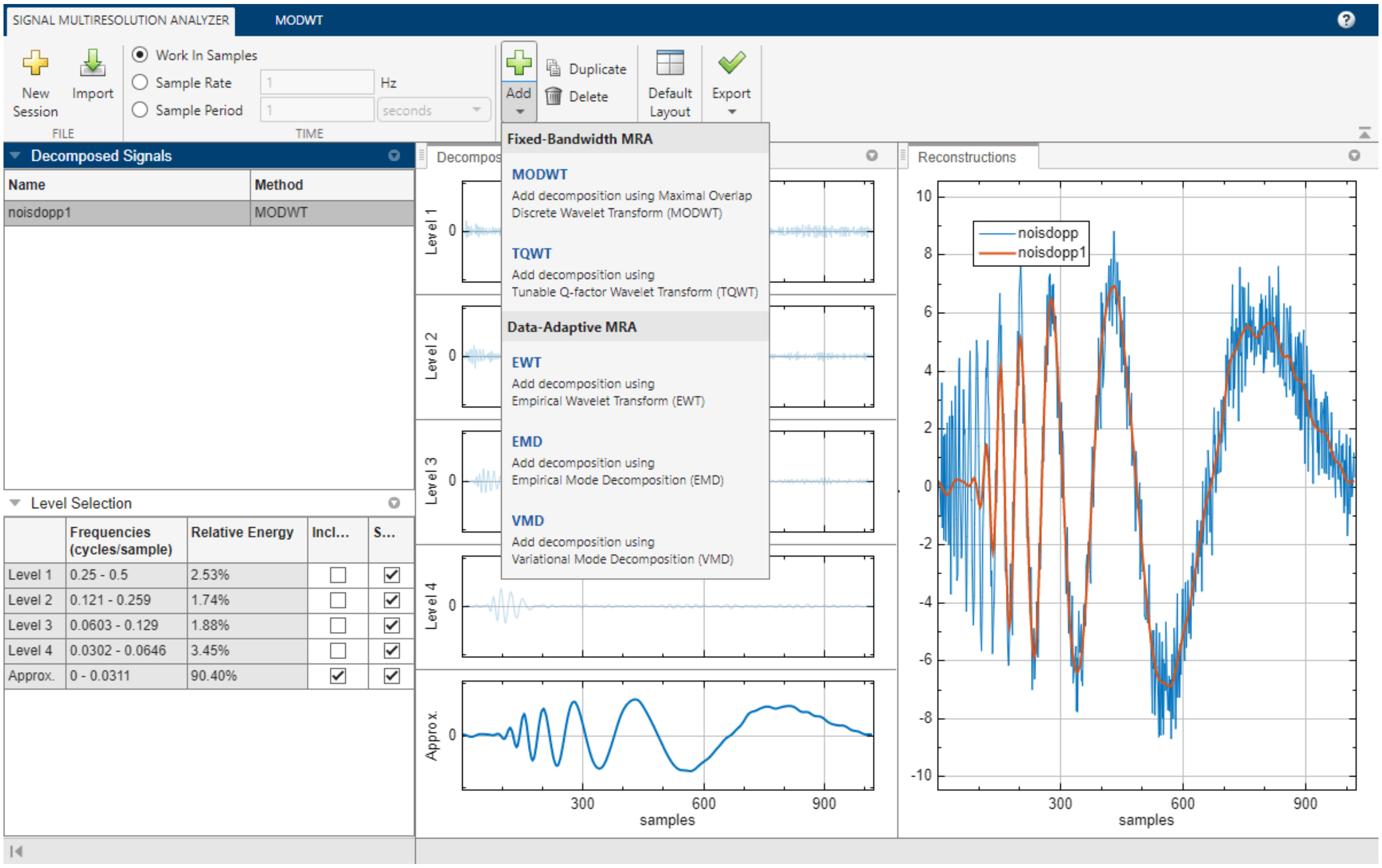
Compare MODWTMRA and EMD Decompositions

Load the noisy Doppler signal. The signal is a noisy version of the Doppler test signal of Donoho and Johnstone [1].

```
load noisdopp
```

Open **Signal Multiresolution Analyzer** and import the signal into the app. By default, the app creates a four-level MODWTMRA decomposition of the signal in the **MODWT** tab. In the **Decomposed Signals** pane, the wavelet decomposition is named `noisdopp1`. The **Reconstructions** pane shows the original and reconstructed signals plotted in two different colors.

To add the EMD decomposition, first switch to the **Signal Multiresolution Analyzer** tab, then click **Add ▼** and select **EMD**.


SIGNAL MULTIRESOLUTION ANALYZER
MODWT
New Session
Import
FILE
● Work In Samples
○ Sample Rate 1 Hz
○ Sample Period 1 seconds
TIME
Add
Duplicate
Delete
Default Layout
Export
Fixed-Bandwidth MRA
MODWT
Add decomposition using Maximal Overlap Discrete Wavelet Transform (MODWT)
TQWT
Add decomposition using Tunable Q-factor Wavelet Transform (TQWT)
Data-Adaptive MRA
EWT
Add decomposition using Empirical Wavelet Transform (EWT)
EMD
Add decomposition using Empirical Mode Decomposition (EMD)
VMD
Add decomposition using Variational Mode Decomposition (VMD)
Decompos
Level 1
Level 2
Level 3
Level 4
Approx.
0
300
600
900
samples
Reconstructions
noisdopp
noisdopp1
10
8
6
4
2
0
-2
-4
-6
-8
-10
300
600
900
samples

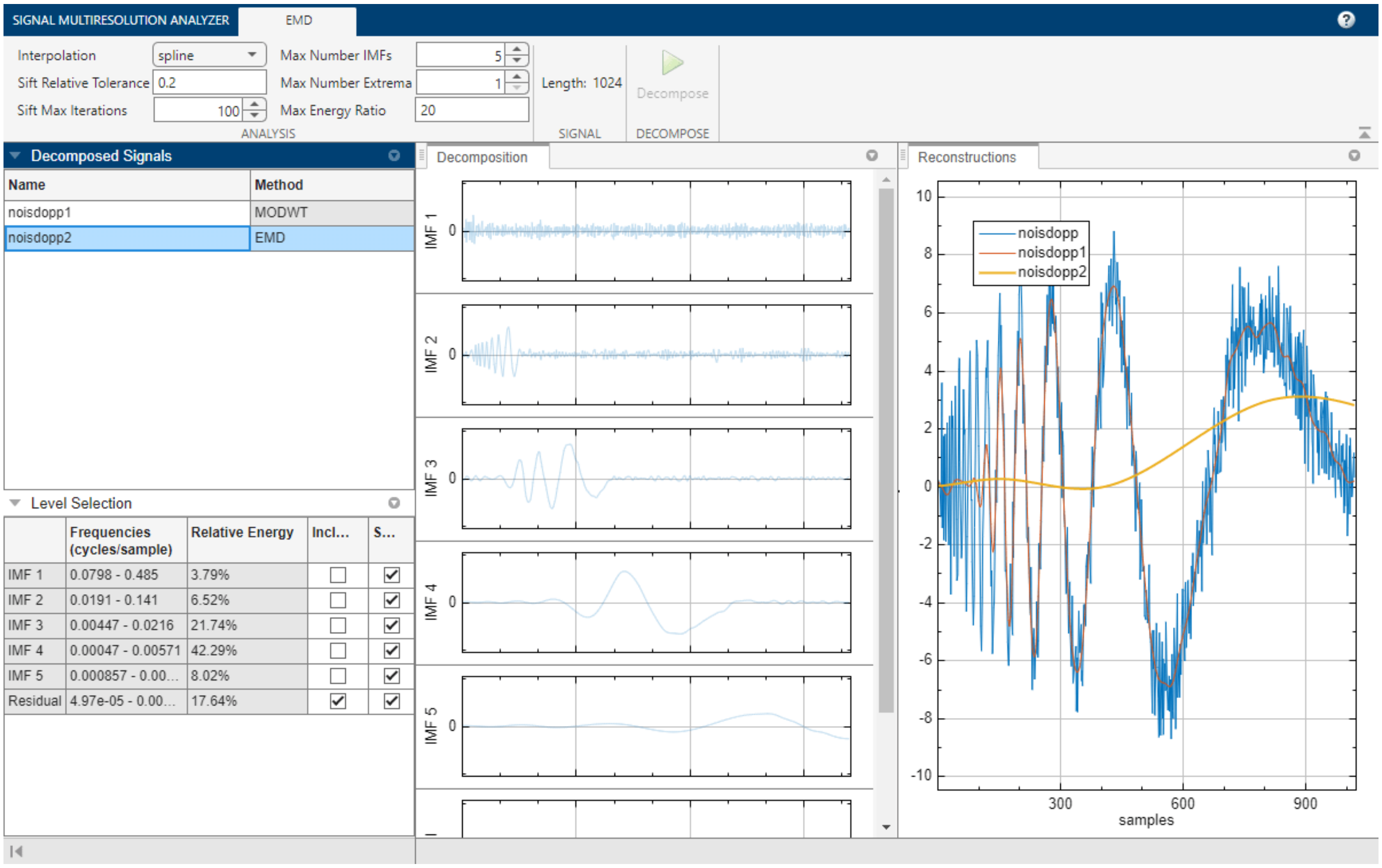
Decomposed Signals

Name	Method
noisdopp1	MODWT

Level Selection

	Frequencies (cycles/sample)	Relative Energy	Incl...	S...
Level 1	0.25 - 0.5	2.53%	□	☑
Level 2	0.121 - 0.259	1.74%	□	☑
Level 3	0.0603 - 0.129	1.88%	□	☑
Level 4	0.0302 - 0.0646	3.45%	□	☑
Approx.	0 - 0.0311	90.40%	☑	☑

After a few moments the EMD decomposition `noisdopp2` appears in the **EMD** tab. The decomposition is obtained using the `emd` function with default settings. The residual is now the thickest plot in the **Reconstructions** pane. You can change the parameters in the toolstrip and click **Decompose** to obtain a different EMD decomposition. To learn more about the parameters and the EMD algorithm, see `emd`.



Interpolation: spline
Sift Relative Tolerance: 0.2
Sift Max Iterations: 100
Max Number IMFs: 5
Max Number Extrema: 1
Max Energy Ratio: 20
Length: 1024

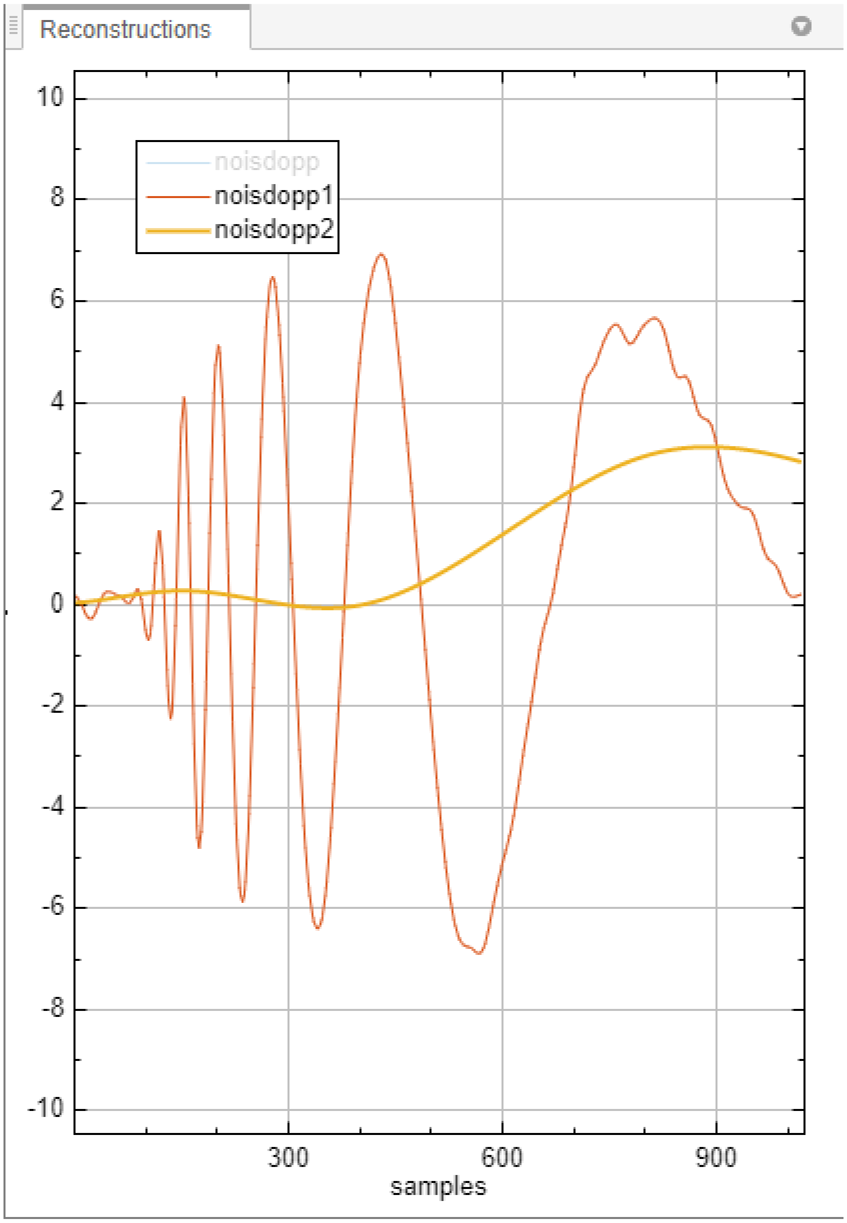
Decomposed Signals

Name	Method
noisdopp1	MODWT
noisdopp2	EMD

Level Selection

	Frequencies (cycles/sample)	Relative Energy	Incl...	S...
IMF 1	0.0798 - 0.485	3.79%	□	☑
IMF 2	0.0191 - 0.141	6.52%	□	☑
IMF 3	0.00447 - 0.0216	21.74%	□	☑
IMF 4	0.00047 - 0.00571	42.29%	□	☑
IMF 5	0.000857 - 0.00...	8.02%	□	☑
Residual	4.97e-05 - 0.00...	17.64%	☑	☑

To more easily see the differences between the two reconstructions, click `noisdopp` in the plot legend. The text fades, and the plot of the original signal is hidden. You can use the legend to hide any plot in the **Reconstructions** pane.



Duplicate Decomposition and Generate Script

This example shows how to duplicate a decomposition for modification. The example also shows how to generate a script to recreate the decomposition in your workspace.

Load the Kobe earthquake data into your workspace. The data are seismograph measurements (vertical acceleration in nm/sec^2) recorded at Tasmania University, Hobart, Australia, on 16 January 1995, beginning at 20:56:51 (GMT) and continuing for 51 minutes at one second intervals.

```
load kobe
```

Open **Signal Multiresolution Analyzer** and import the earthquake data into the app. By default, the app creates a four-level MODWTMRA decomposition of the signal called `kobe1` using the `modwt` and `modwtmra` functions with default settings. To show plots with respect to time and express frequencies in Hz, click the **Sample Rate** radio button in the **Signal Multiresolution Analyzer** tab.

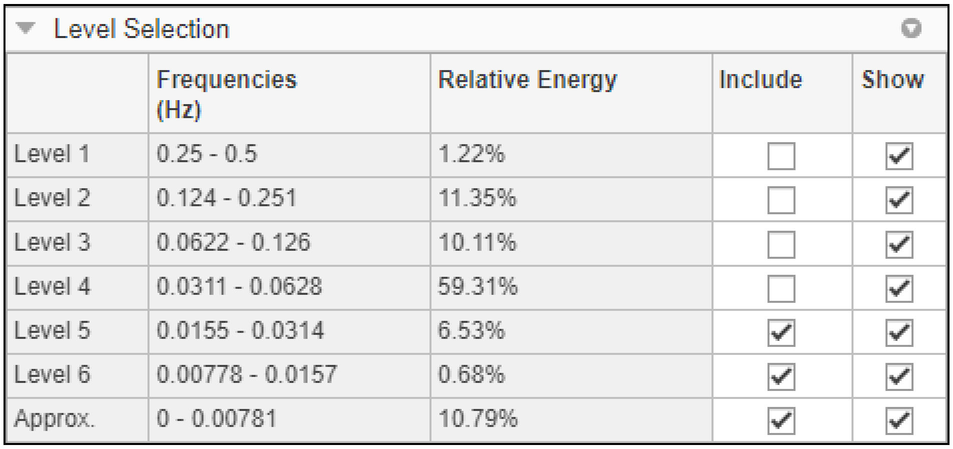
Duplicate Decomposition

Create a new six-level decomposition using the order 4 Coiflet. In the **Signal Multiresolution Analyzer** tab, click **Duplicate** in the toolstrip. Since `kobe1` is the currently selected item in **Decomposed Signals**, a duplicate of the first decomposition is created. The duplicate is called `kobe1Copy`. The plots in **Reconstructions** are updated to include the new decomposition. Except for the color, the duplicate is identical with the first decomposition. You can change the name of the duplicate by right-clicking on the name in **Decomposed Signals**.

In the **MODWT** tab, change the settings in the toolstrip to the following values and then click **Decompose**.

- **Wavelet**: `coif`
- **Number**: 4
- **Level**: 6

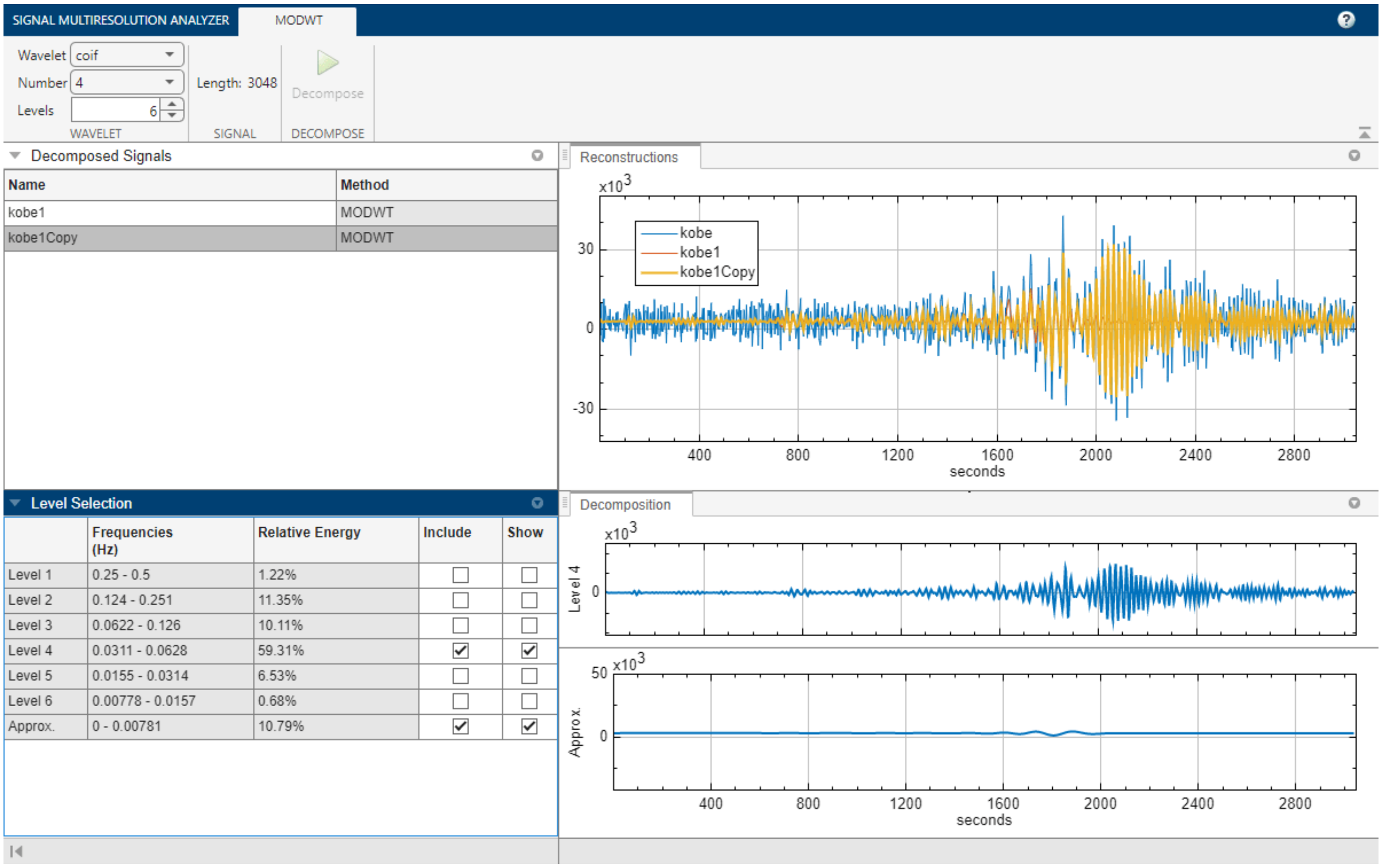
In **Level Selection**, note which components of the decomposition are included in the reconstruction: the approximation and the level 5 and level 6 details.



Level Selection

	Frequencies (Hz)	Relative Energy	Include	Show
Level 1	0.25 - 0.5	1.22%	☐	☑
Level 2	0.124 - 0.251	11.35%	☐	☑
Level 3	0.0622 - 0.126	10.11%	☐	☑
Level 4	0.0311 - 0.0628	59.31%	☐	☑
Level 5	0.0155 - 0.0314	6.53%	☑	☑
Level 6	0.00778 - 0.0157	0.68%	☑	☑
Approx.	0 - 0.00781	10.79%	☑	☑

Level 4 has approximately 60% of the total energy. Remove levels 5 and 6 from the reconstruction, and include level 4. Show only the approximation and level 4 details in the **Decomposition** pane. To approximately align the decomposition with the reconstruction, drag the **Decomposition** pane beneath the **Reconstructions** pane.



Wavelet: coif
Number: 4
Levels: 6
Length: 3048

Decomposed Signals

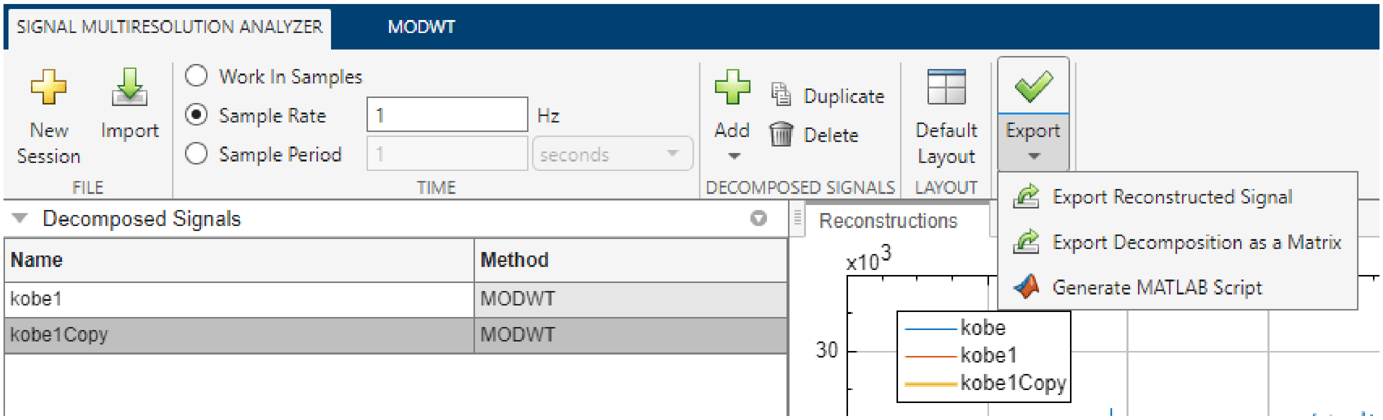
Name	Method
kobe1	MODWT
kobe1Copy	MODWT

Level Selection

	Frequencies (Hz)	Relative Energy	Include	Show
Level 1	0.25 - 0.5	1.22%	□	□
Level 2	0.124 - 0.251	11.35%	□	□
Level 3	0.0622 - 0.126	10.11%	□	□
Level 4	0.0311 - 0.0628	59.31%	☑	☑
Level 5	0.0155 - 0.0314	6.53%	□	□
Level 6	0.00778 - 0.0157	0.68%	□	□
Approx.	0 - 0.00781	10.79%	☑	☑

Generate MODWT Script

You have three export options. You can export the reconstruction or the entire decomposition of the selected decomposed signal to your workspace, or you can export a MATLAB™ script to recreate the decomposition in your workspace. To generate a script, in the **Signal Multiresolution Analyzer** tab click **Export** > **Generate MATLAB Script**.



○ Work In Samples
● Sample Rate 1 Hz
○ Sample Period 1 seconds

Decomposed Signals

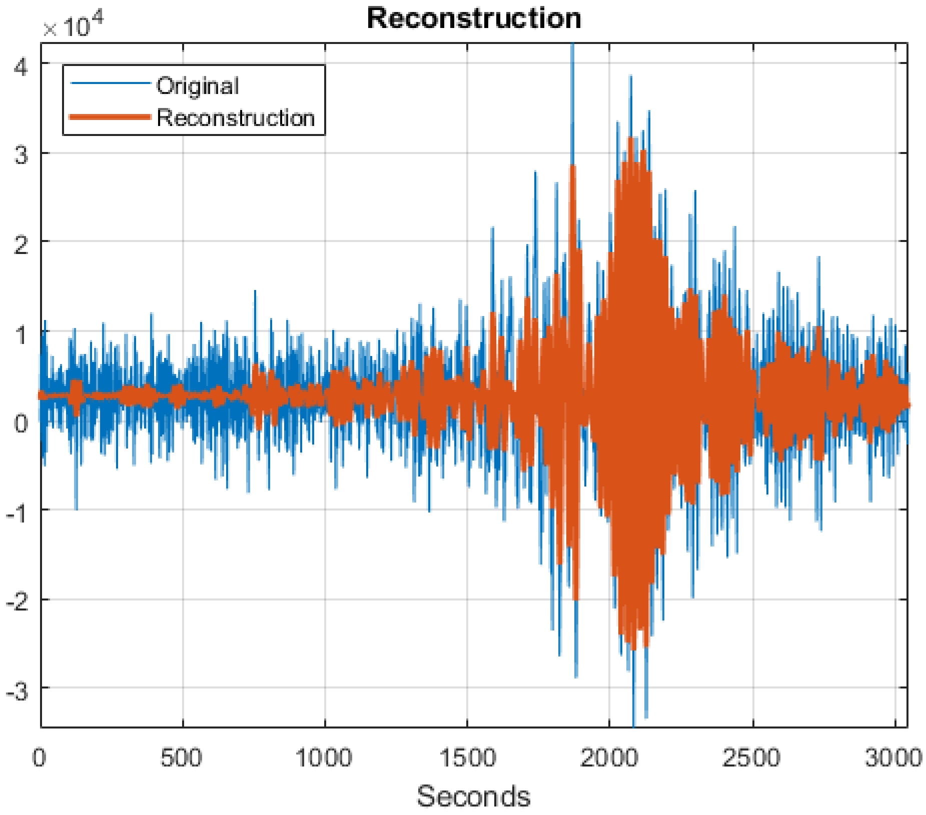
Name	Method
kobe1	MODWT
kobe1Copy	MODWT

An untitled script opens in your editor with the following executable code. The true-false values in `levelForReconstruction` correspond to which `Include` boxes are checked in **Level Selection**. You can save the script as is, or modify it to apply the same decomposition settings to other signals. Run the code.

```
% Logical array for selecting reconstruction elements
levelForReconstruction = [false,false,false,true,false,false,true];

% Perform the decomposition using modwt
wt = modwt(kobe,'coif4',6);

% Construct MRA matrix using modwtmra
mra = modwtmra(wt,'coif4');

% Sum down the rows of the selected multiresolution signals
kobe1Copy = sum(mra(levelForReconstruction,:),1);
```

Plot the original signal and reconstruction. Except for possibly the colors, the plot will match the `kobe1Copy` reconstruction shown in the app.

```
t = 0:numel(kobe)-1;
plot(t,kobe)
grid on
hold on
plot(t,kobe1Copy,LineWidth=2)
xlabel("Seconds")
title("Reconstruction")
legend("Original","Reconstruction",Location="northwest")
axis tight
hold off
```



Generate EMD Script

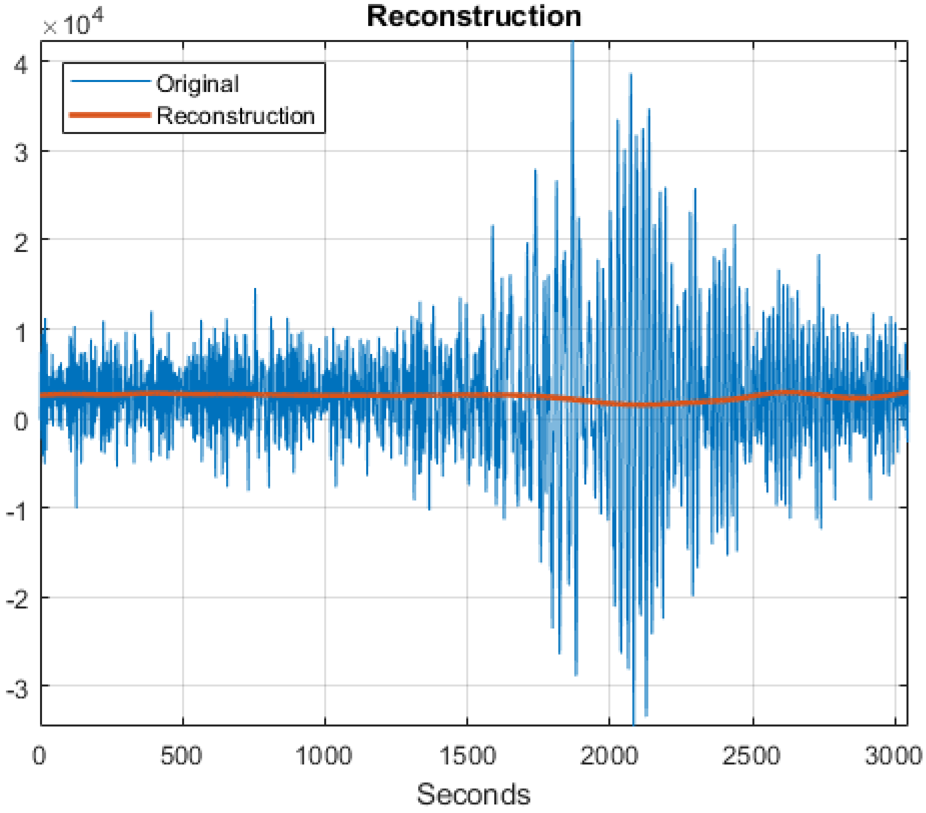
Add the EMD decomposition of the Kobe data by clicking **Add ▼** and selecting **EMD** in the **Signal Multiresolution Analyzer** tab. The name of the decomposed signal in the **Decomposed Signals** pane is `kobe3`. By default, the reconstruction consists only of the residual. The decomposition is obtained by using the `emd` function with default settings.

Generate a script that creates the EMD decomposition by clicking **Export** > **Generate MATLAB Script**. An untitled script opens in your editor with the following executable code. Run the code.

```
% Logical array for selecting reconstruction elements
levelForReconstruction = [false,false,false,false,false,true];

% Perform the decomposition using EMD
[imf,residual,info] = emd(kobe, ...
    SiftRelativeTolerance=0.2, ...
    SiftMaxIterations=100, ...
    MaxNumIMF=5, ...
    MaxNumExtrema=1, ...
    MaxEnergyRatio=20, ...
    Interpolation='spline');

% Construct MRA matrix by appending IMFs and residual
mra = [imf residual].';

% Sum down the rows of the selected multiresolution signals
kobe3 = sum(mra(levelForReconstruction,:),1);
```

Compare the reconstruction `kobe3` with the original signal. In this case, the reconstruction only consists of the residual.

```
plot(t,kobe)
grid on
hold on
plot(t,kobe3,LineWidth=2)
xlabel("Seconds")
title("Reconstruction")
legend("Original","Reconstruction",Location="northwest")
axis tight
hold off
```



- "Visualize and Recreate EWT Decomposition"
- "Visualize and Recreate TQWT Decomposition"
- "Visualize and Recreate VMD Decomposition"
- "Comparing MODWT and MODWTMRA" on page 1-965

Parameters

Wavelet — Orthogonal wavelet family

sym (default) | coif | db | fk

Orthogonal wavelet family to use to generate the multiresolution analysis (default), specified as:

- sym — Symlets
- coif — Coiflets
- db — Daubechies wavelets
- fk — Fejér-Korovkin wavelets

The Wavelet parameter is applicable only for generating a multiresolution analysis.

For more information about the wavelets, use the waveinfo function. For example, to learn more about Daubechies wavelets, enter waveinfo('db').

`Interpolation` — Interpolation method

`spline` (default) | `pchip`

Interpolation method to use for envelope construction in empirical mode decomposition, specified as one of the following:

- `spline` — Cubic spline interpolation
- `pchip` — Piecewise cubic Hermite interpolating polynomial method

The `Interpolation` parameter is applicable only for generating an empirical mode decomposition. You can change other options with the app when creating empirical mode decompositions. For more information, see `emd`.

Programmatic Use

`signalMultiresolutionAnalyzer` opens the **Signal Multiresolution Analyzer** app. Once the app initializes, import a signal for analysis by clicking **Import**.

`signalMultiresolutionAnalyzer(sig)` opens the **Signal Multiresolution Analyzer** app and imports, decomposes, and plots the multiresolution analysis of `sig` using `modwtmra` and `modwt` with the `sym4` wavelet and default settings.

`sig` is a variable in the workspace. `sig` can be:

- A 1-by-*N* or *N*-by-1 real-valued vector.
- Single or double precision.

By default, the app plots the decomposition levels as functions of sample index. To plot with respect to time, you can set a sample rate or sample period using the app.

Tips

- To decompose more than one signal simultaneously, you can run multiple instances of the **Signal Multiresolution Analyzer** app.
- For the MODWT and TQWT decomposition methods, the script generated by the **Signal Multiresolution Analyzer** app supports `gpuArray` (Parallel Computing Toolbox) inputs.

Algorithms

Decomposition Methods

To generate the decompositions, **Signal Multiresolution Analyzer** uses these functions:

- EMD — `emd`
- EWT — `ewt`
- MODWT — `modwt` and `modwtmra`
- TQWT — `tqwt` and `tqwtmra`
- VMD — `vmd`

Passband Frequencies

For the fixed-bandwidth methods, EMD and MODWT, **Signal Multiresolution Analyzer** reports the theoretical frequency ranges of the decomposition levels. For the data-adaptive methods, EWT, TQWT, and VMD, the app reports the measured bandwidth.

References

[1] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

See Also

Apps
Wavelet Signal Denoiser | **Wavelet Time-Frequency Analyzer**

Functions
`emd` | `ewt` | `modwt` | `modwtmra` | `tqwt` | `tqwtmra` | `vmd`

Topics
"Visualize and Recreate EWT Decomposition"
"Visualize and Recreate TQWT Decomposition"
"Visualize and Recreate VMD Decomposition"
"Comparing MODWT and MODWTMRA" on page 1-965
"Empirical Wavelet Transform"
"Tunable Q-factor Wavelet Transform"
"Practical Introduction to Multiresolution Analysis"
"Time-Frequency Gallery"

Introduced in R2018b

subdict

Extract submatrix from a sensing dictionary

Syntax

```
Ar = subdict(A,rowIndices,colIndices)
```

Description

`Ar = subdict(A,rowIndices,colIndices)` returns the submatrix `Ar` that corresponds to the rows and columns specified by `rowIndices` and `colIndices`, respectively.

Examples

Extract and Visualize Submatrix

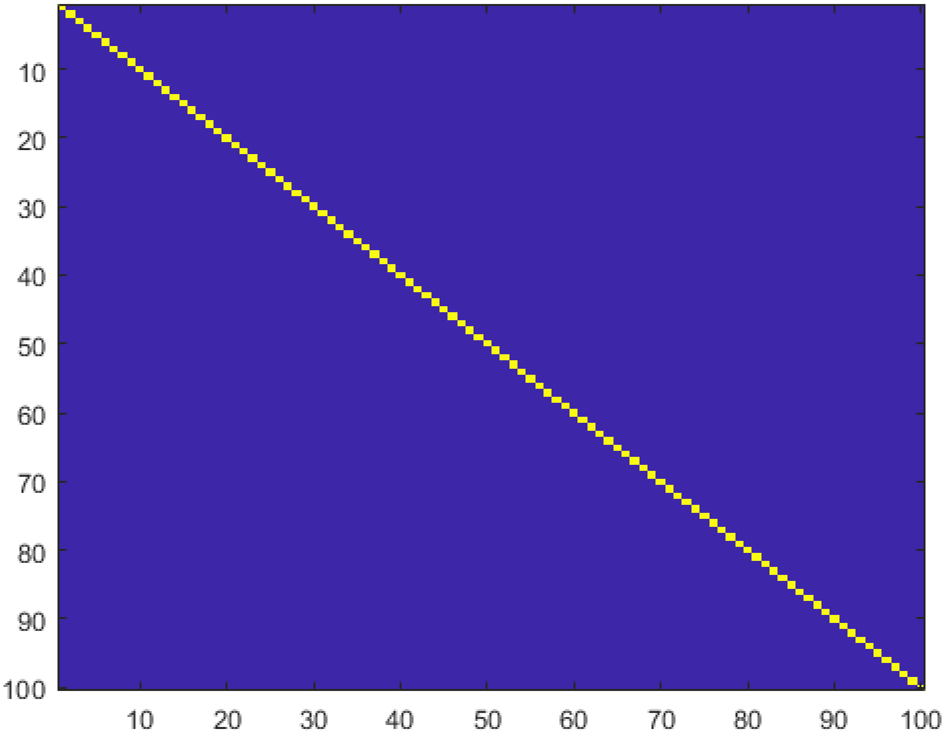
Create a sensing dictionary. Set the type of the sensing dictionary to `'fourier'` and `'eye'`. The size of each basis type is 100-by-100.

```
A = sensingDictionary(Size=100,Type={'fourier','eye'})
```

```
A = 
  sensingDictionary with properties:

                Type: {'fourier'  'eye'}
                Name: {''  ''}
               Level: [0 0]
    CustomDictionary: []
                Size: [100 200]
```

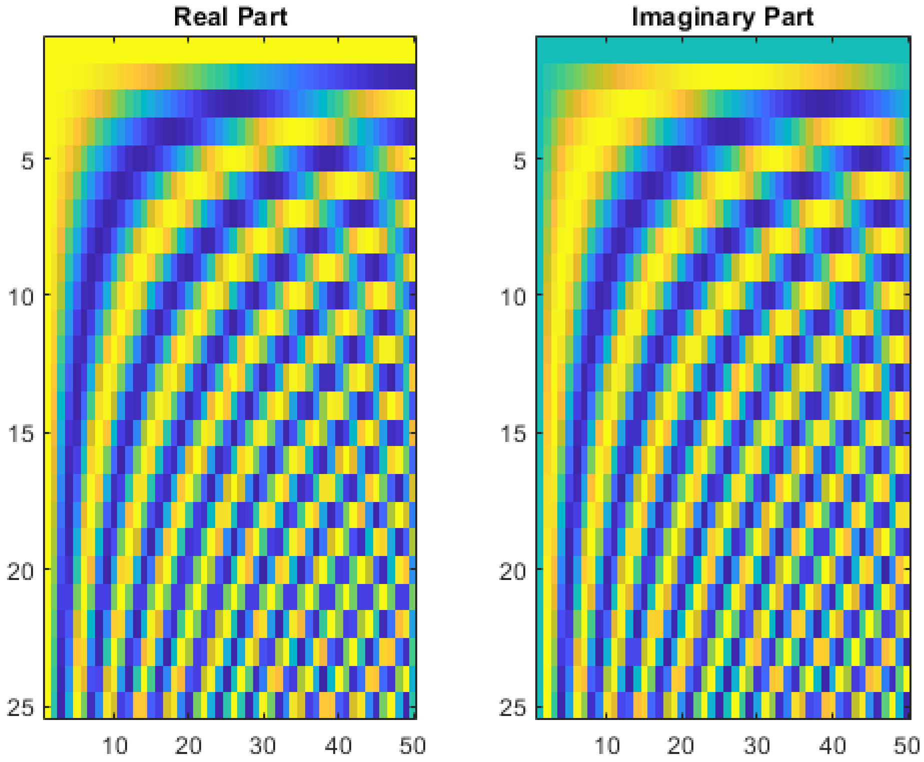
Extract the entire submatrix that is associated with the `'eye'` basis type. Visualize the submatrix.

```
Bmat = subdict(A,1:100,101:200);
imagesc(Bmat)
```



Extract a 25-by-50 submatrix associated with the `'fourier'` basis type. Visualize the real and imaginary parts of the submatrix.

```
Cmat = subdict(A,1:25,1:50);
subplot(1,2,1)
imagesc(real(Cmat))
title("Real Part")
subplot(1,2,2)
imagesc(imag(Cmat))
title("Imaginary Part")
```



Input Arguments

A — Sensing dictionary
`sensingDictionary` object

Sensing dictionary, specified as a `sensingDictionary` object.

`rowIndices` — Row indices
vector

Row indices to extract, specified as a vector.

Example: `Ar = subdict(A,1:256,1:100)` returns the 256-by-100 submatrix that corresponds to the rows indexed by `1:256` and columns indexed by `[1:100]`.

Data Types: `double`

`colIndices` — Column indices
vector

Column indices to extract, specified as a vector.

Example: `Ar = subdict(A,1:128,[2 3 5 8 13])` returns the 128-by-5 submatrix that corresponds to the rows indexed by `1:128` and columns indexed by `[2 3 5 8 13]`.

Data Types: `double`

Output Arguments

`Ar` — Submatrix

matrix

Submatrix extracted from the `sensingDictionary` `A`, returned as a matrix. The matrix `Ar` is *M*-by-*N*, where *M* equals the length of `rowIndices`, and *N* equals the length of `colIndices`.

Data Types: `double`

See Also

`sensingDictionary`

Introduced in R2022a

subset

Get new labeled signal set with subset of members

Syntax

```
lssnew = subset(lss,midxvect)
```

Description

`lssnew = subset(lss,midxvect)` returns a new labeled signal set containing the members specified in `midxvect`.

Examples

Labeled Subset

Load a labeled signal set of whale songs.

```
load whales
lss
```

```
lss = 
  labeledSignalSet with properties:

             Source: {2x1 cell}
         NumMembers: 2
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [2x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Create a new labeled signal set consisting of the second member of the original set.

```
lssnew = subset(lss,2)
```

```
lssnew = 
  labeledSignalSet with properties:

             Source: {[76579x1 double]}
         NumMembers: 1
    TimeInformation: "sampleRate"
         SampleRate: 4000
             Labels: [1x3 table]
        Description: "Characterize wave song regions"

 Use labelDefinitionsHierarchy to see a list of labels and sublabels.
 Use setLabelValue to add data to the set.
```

Input Arguments

`lss` — Labeled signal set

`labeledSignalSet` object

Labeled signal set, specified as a `labeledSignalSet` object.

Example: `labeledSignalSet({randn(100,1) randn(10,1)},signalLabelDefinition('female'))` specifies a two-member set of random signals containing the attribute `'female'`.

`midxvect` — Subset member row numbers

vector of positive integers

Subset member row numbers, specified as a vector of positive integers. Each element of `midxvect` specifies a member row number as it appears in the “Labels” on page 1-0 table of the `labeledSignalSet` object `lss`.

Example: `[2 3 5 7 11 13 17]` chooses a subset of signals indexed by prime numbers.

Output Arguments

`lssnew` — New labeled signal set

`labeledSignalSet` object

New labeled signal set, returned as a `labeledSignalSet` object.

See Also

`labeledSignalSet` | `signalLabelDefinition`

Introduced in R2018b

swt

Discrete stationary wavelet transform 1-D

Syntax

```
swc = swt(x,n,wname)
swc = swt(x,n,LoD,HiD)
[swa,swd] = swt( ___ )
```

Description

`swc = swt(x,n,wname)` returns the stationary wavelet decomposition of the signal `x` at level `n` using the wavelet `wname`.

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.

`swc = swt(x,n,LoD,HiD)` returns the stationary wavelet decomposition using the specified lowpass and highpass wavelet decomposition filters `LoD` and `HiD`, respectively.

`[swa,swd] = swt( ___ )` returns the approximation coefficients `swa` and stationary wavelet coefficients `swd` using either of the previous syntaxes.

Examples

Multilevel Stationary Wavelet Decomposition

Perform a multilevel stationary wavelet decomposition of a signal.

Load a one-dimensional signal and acquire its length.

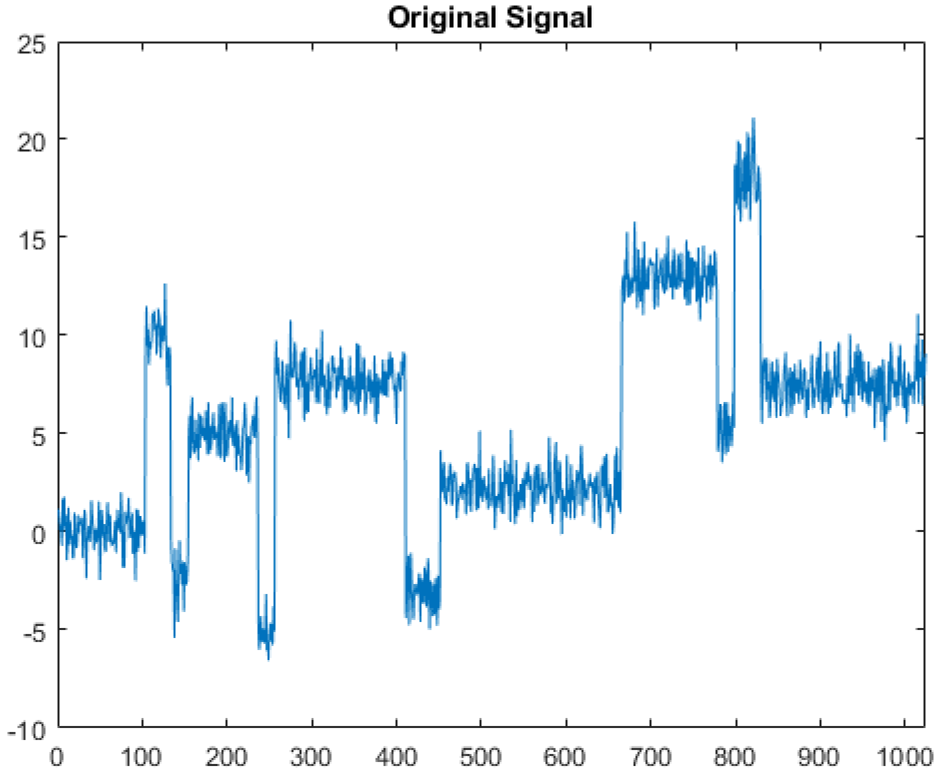
```
load noisbloc
s = noisbloc;
sLen = length(s);
```

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,:);
```

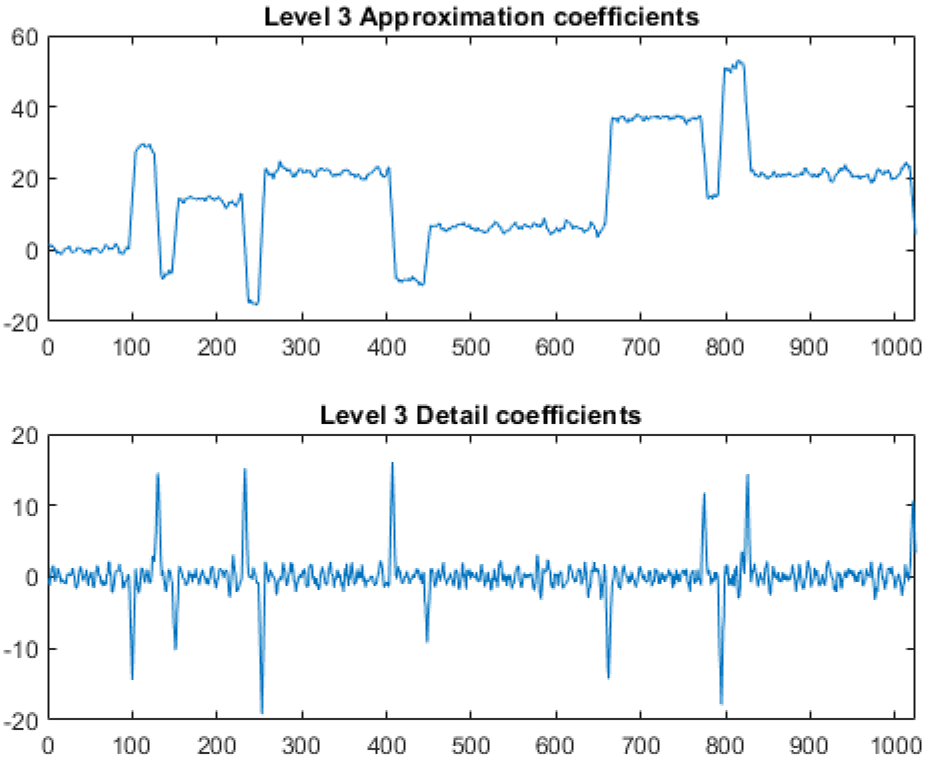
Plot the output of the decomposition.

```
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')
```



Input Arguments

x — Input signal

real-valued vector

Input signal, specified as a real-valued vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

n — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer. 2^n must divide the length of `x`. Use `wmaxlev` to determine the maximum level of decomposition.

Data Types: `double`

`wname` — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. `swt` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoD,HiD` — Wavelet decomposition filters

even-length real-valued vectors

Wavelet decomposition filters, specified as a pair of even-length real-valued vectors. LoD is the lowpass decomposition filter, and HiD is the highpass decomposition filter. The lengths of LoD and HiD must be equal. See wfilters for additional information.

Output Arguments

swc — Stationary wavelet decomposition

real-valued matrix

Stationary wavelet decomposition, returned as a real-valued matrix. The coefficients are stored row-wise:

- For 1 ≤ *i* ≤ n, the *i*th row of swc contains the detail coefficients of level *i*.
- swc(n+1,:) contains the approximation coefficients of level n.

Data Types: double

swa — Approximation coefficients

real-valued matrix

Approximation coefficients, returned as a real-valued matrix. For 1 ≤ *i* ≤ n, the *i*th row of swa contains the approximation coefficients of level *i*.

Data Types: double

swd — Detail coefficients

real-valued matrix

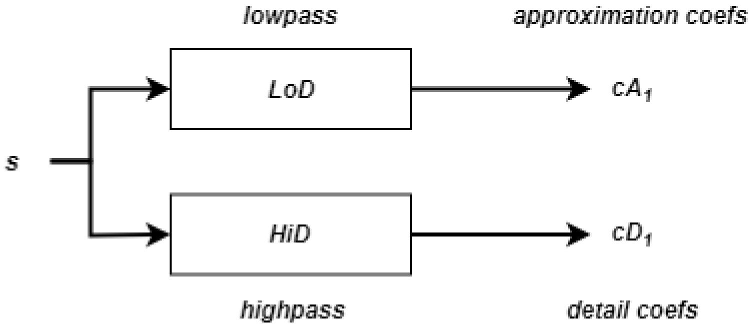
Detail coefficients, returned as a real-valued matrix. For 1 ≤ *i* ≤ n, the *i*th row of swd contains the detail coefficients of level *i*.

Data Types: double

Algorithms

Given a signal *s* of length *N*, the first step of the stationary wavelet transform (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 lowpass filter LoD for approximation, and with the highpass filter HiD for detail.

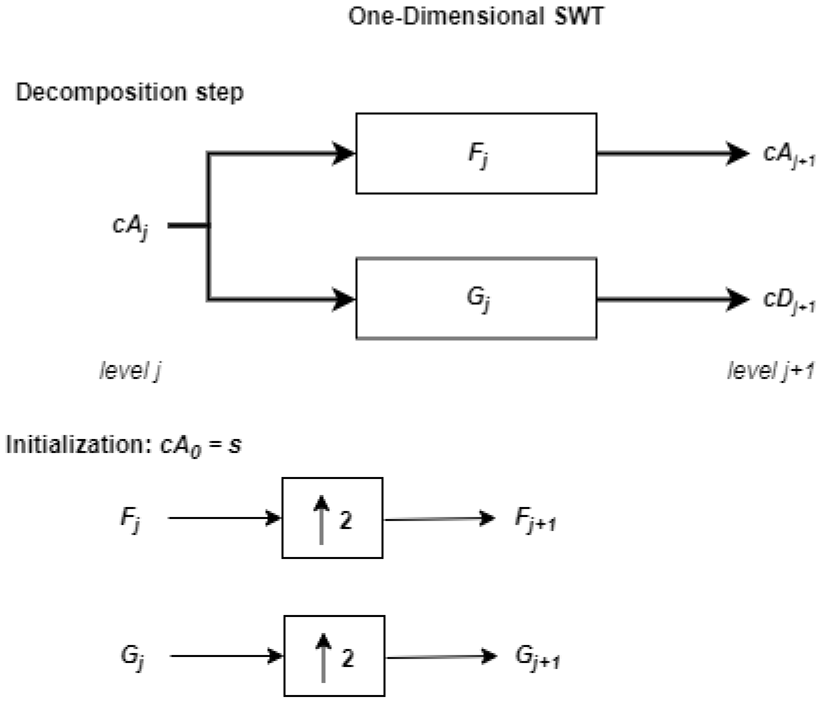
More precisely, the first step is



where $\boxed{X}$ denotes convolution with the filter *X*.

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,



where

- $F_0 = LoD$
- $G_0 = HiD$
- $\boxed{\uparrow 2}$ — Upsample (insert zeros between elements)

References

[1] Nason, G. P., and B. W. Silverman. “The Stationary Wavelet Transform and Some Statistical Applications.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:281–99. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_17.

[2] Coifman, R. R., and D. L. Donoho. “Translation-Invariant De-Noising.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:125–50. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_9.

[3] Pesquet, J.-C., H. Krim, and H. Carfantan. “Time-Invariant Orthonormal Wavelet Representations.” *IEEE Transactions on Signal Processing* 44, no. 8 (August 1996): 1964–70. https://doi.org/10.1109/78.533717.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.
- The level of decomposition `n` must be defined as a scalar during compilation.

See Also

`dwt` | `iswt` | `wavedec` | `modwt`

Introduced before R2006a

swt2

Discrete stationary 2-D wavelet transform

Syntax

```
[A,H,V,D] = swt2(X,N,wname)
[A,H,V,D] = swt2(X,N,LoD,HiD)
```

```
swc = swt2( ___ )
```

Description

`[A,H,V,D] = swt2(X,N,wname)` returns the approximation coefficients `A` and the horizontal, vertical, and diagonal detail coefficients `H`, `V`, and `D`, respectively, of the stationary 2-D wavelet decomposition of the image `X` at level `N` using the wavelet `wname`.

Note

- `swt2` is uses periodic extension.
- `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.

`[A,H,V,D] = swt2(X,N,LoD,HiD)` uses the specified lowpass and highpass wavelet decomposition filters `LoD` and `HiD`, respectively.

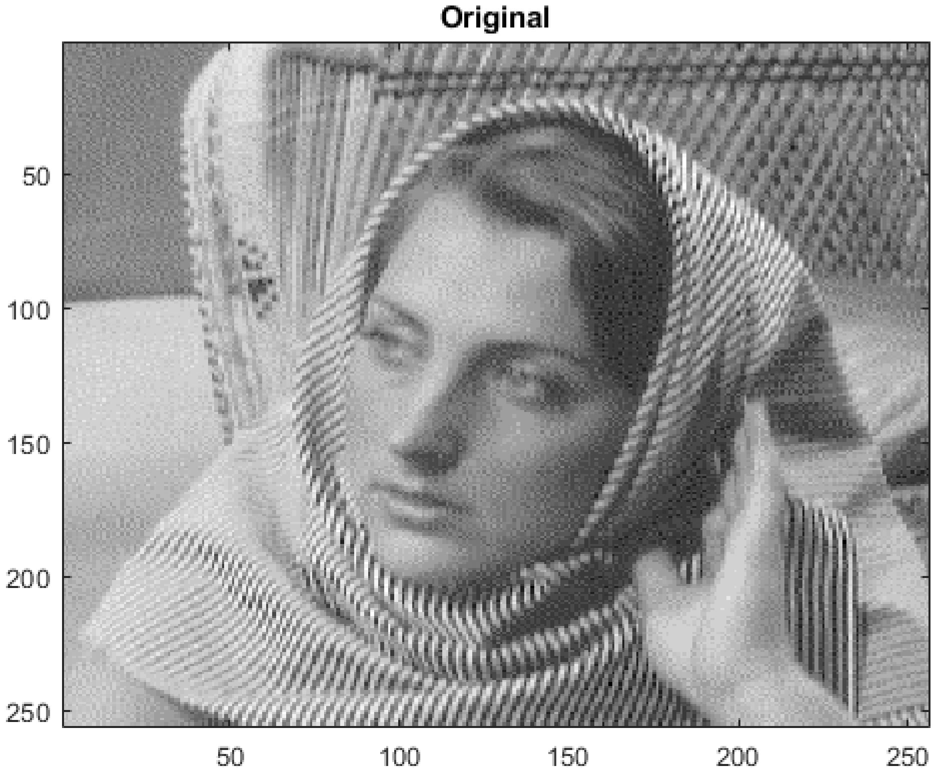
`swc = swt2( ___ )` returns the approximation and detail coefficients in `swc`.

Examples

Extract and Display 2-D Stationary Wavelet Decomposition

Load and display an image.

```
load woman
imagesc(X)
colormap(map)
title('Original')
```



Perform the stationary wavelet decomposition of the image at level 2 using db6.

```
[ca,chd,cvd,cdd] = swt2(X,2,'db6');
```

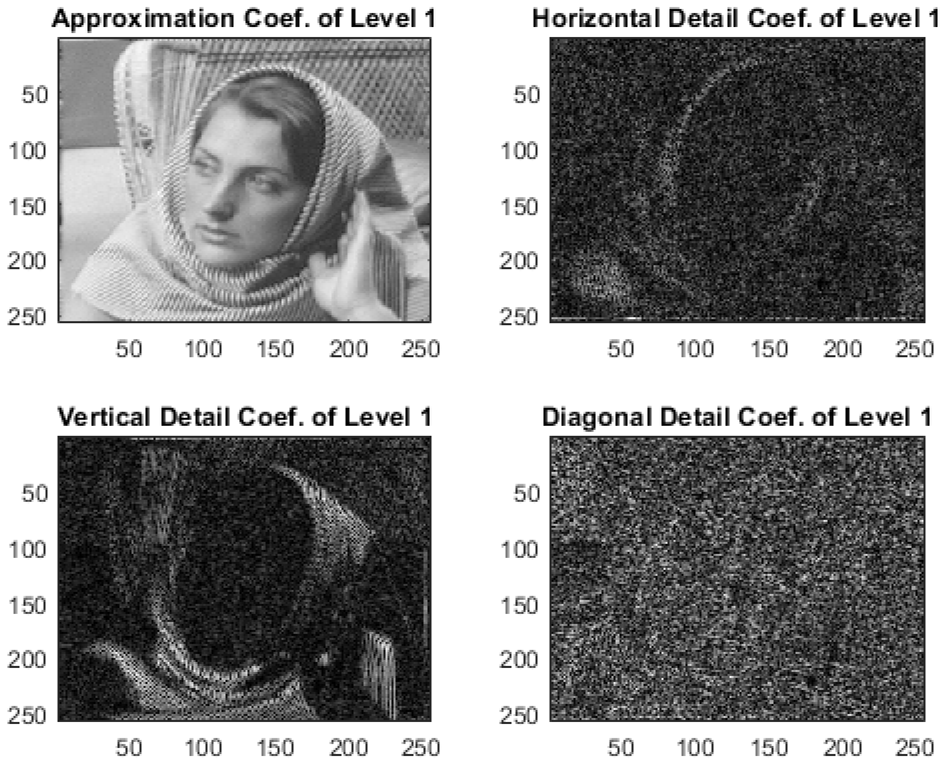
Extract the level 1 and level 2 approximation and detail coefficients from the decomposition.

```
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);
```

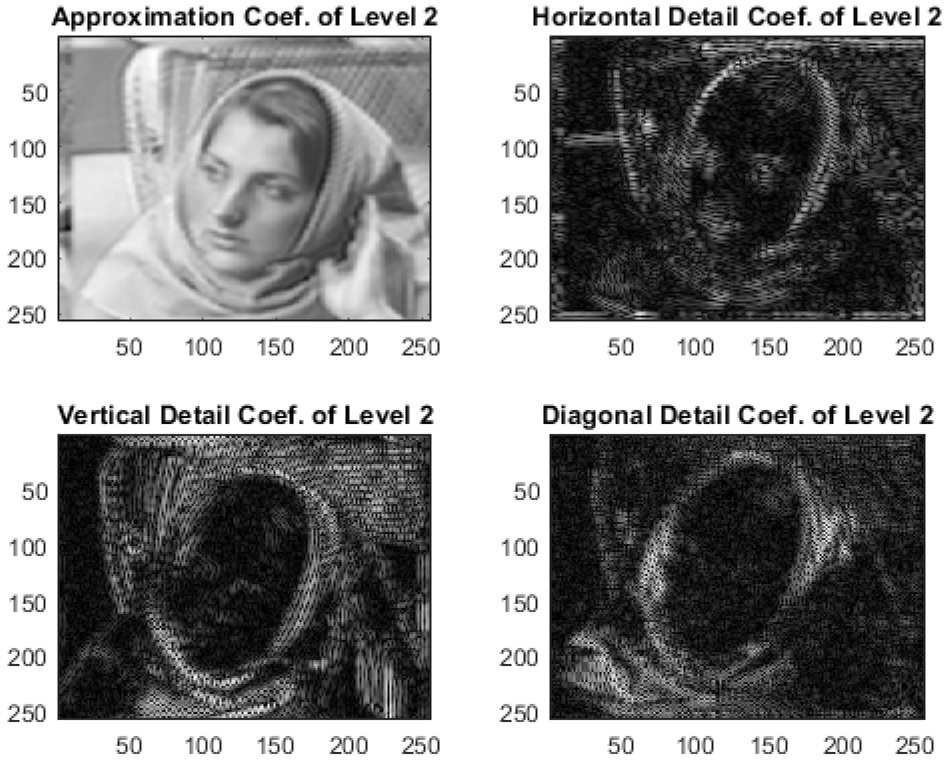
Display the approximation and detail coefficients from the two levels.

```
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')
```

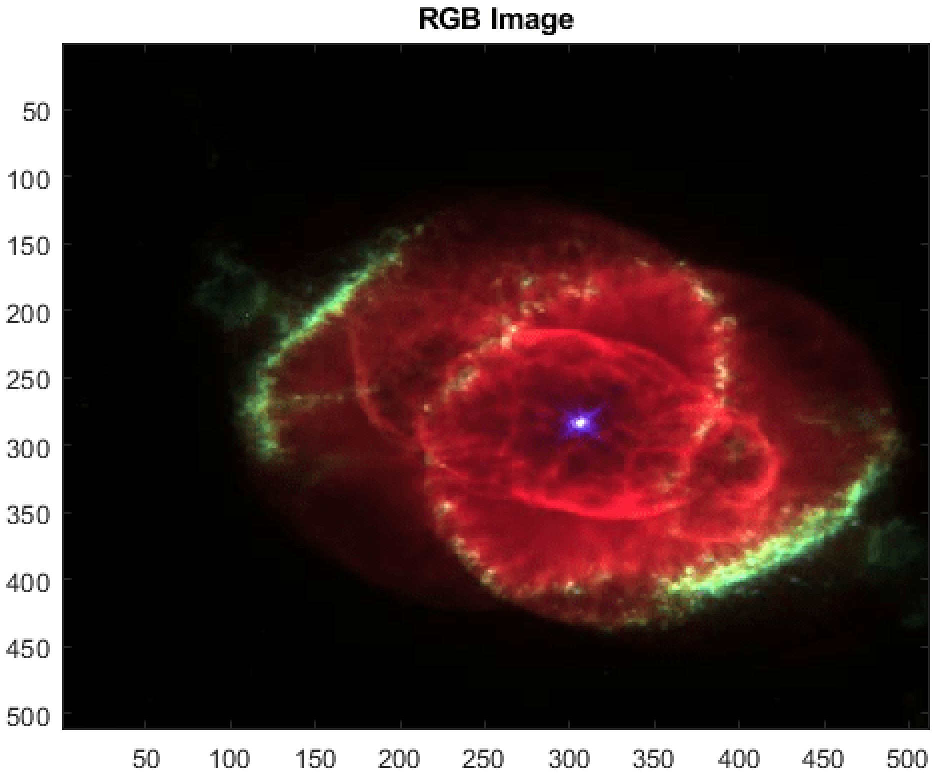


Stationary Wavelet Transform of RGB Image

This example shows how to obtain single-level and multilevel stationary wavelet decompositions of an RGB image.

Load and view an RGB image. The image is a 3-D array of type `uint8`. Since `swt2` requires that the first and second dimensions both be divisible by a power of 2, extract a portion of the image.

```
imdata = imread('ngc6543a.jpg');
x = imdata(1:512,1:512,:);
image(x)
title('RGB Image')
```



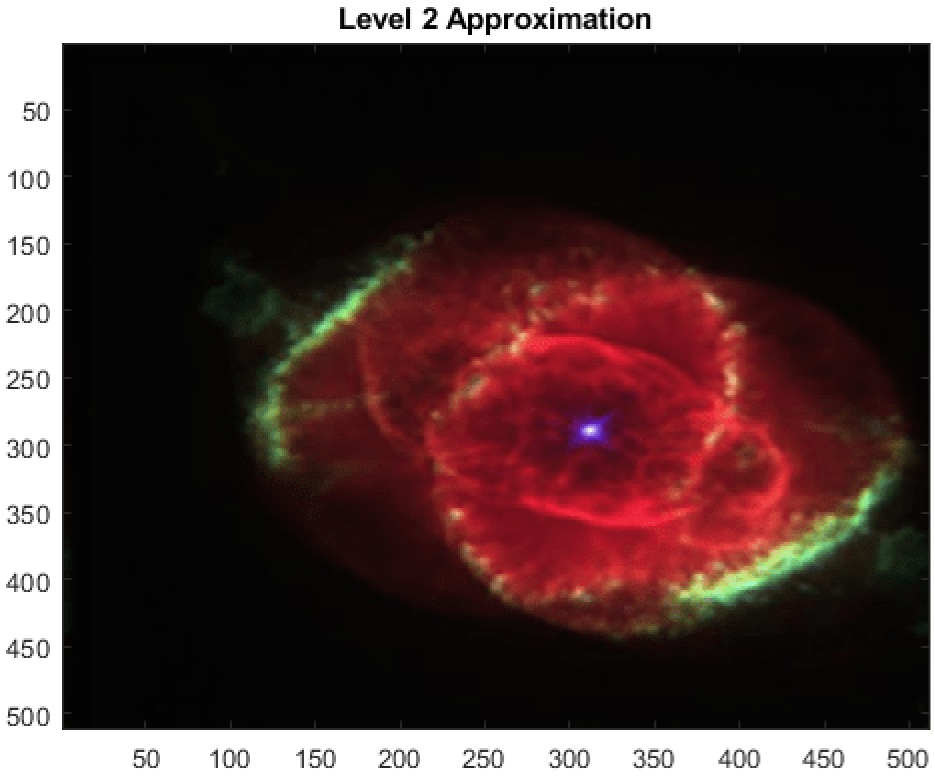
Obtain the level 4 stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients. Note the dimensions of the coefficients array.

```
[a,~,~,~] = swt2(x,4,'db4');
size(a)
```

```
ans = 1×4

   512   512     3     4
```

The coefficients are all of type `double`. In an RGB array of type `double`, each color component is a value between 0 and 1. Rescale the level 2 approximation coefficients to values between 0 and 1 and view the result.

```
a2 = a(:,:,:,2);
a2 = (a2-min(a2(:)))/(max(a2(:))-min(a2(:)));
image(a2)
title('Level 2 Approximation')
```



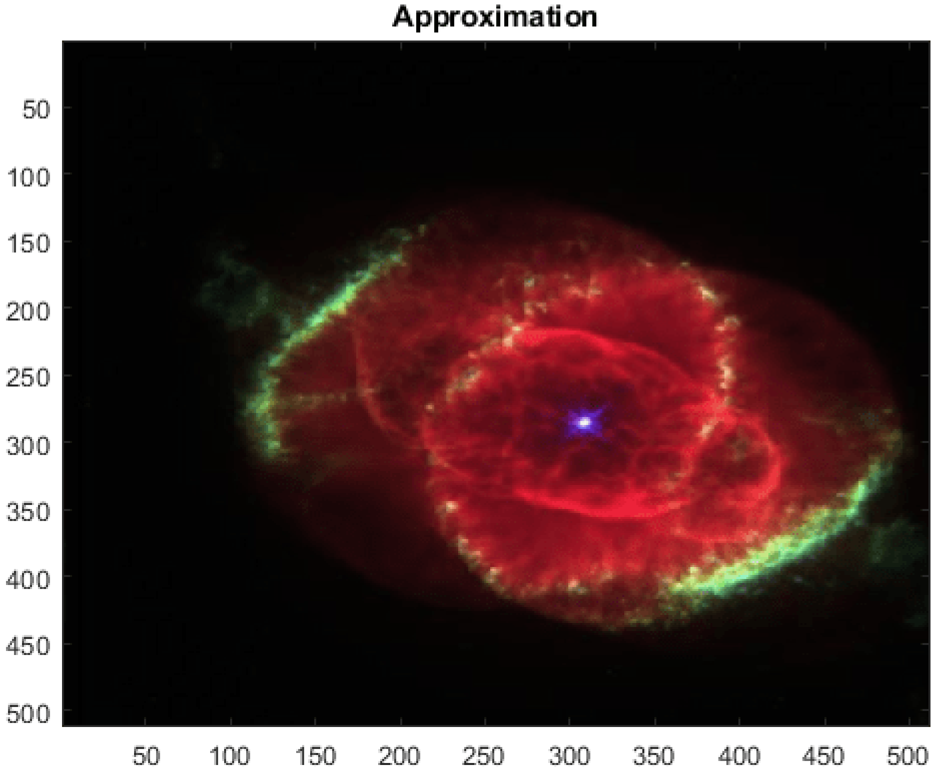
Obtain the single-level stationary wavelet decomposition of the image using the db4 wavelet. Return the approximation coefficients. In a single-level decomposition of an RGB image, the third dimension is singleton.

```
[a,~,~,~] = swt2(x,1,'db4');
size(a)
```

```
ans = 1×4

   512   512     1     3
```

View the approximation coefficients. To prevent an error when using image, squeeze the approximation coefficients array to remove the singleton dimension.

```
a2 = squeeze(a);
a2 = (a2-min(a(:)))/(max(a(:))-min(a(:)));
image(a2)
title('Approximation')
```



Input Arguments

X — Input image
2-D matrix | 3-D array

Input image, specified as a real-valued 2-D matrix or real-valued 3-D array. If X is 3-D, X is assumed to be an RGB image, also referred to as a *truecolor* image, and the third dimension of X must equal 3. For more information on truecolor images, see “Image Types”.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

N — Level of decomposition
positive integer

Level of decomposition, specified as a positive integer. 2^N must divide `size(X,1)` and `size(X,2)`. Use `wmaxlev` to determine the maximum level of decomposition.

wname — Analyzing wavelet
character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. `swt2` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoD,HiD` — Wavelet decomposition filters
even-length real-valued vectors

Wavelet decomposition filters, specified as a pair of even-length real-valued vectors. `LoD` is the lowpass decomposition filter, and `HiD` is the highpass decomposition filter. The lengths of `LoD` and `HiD` must be equal. See `wfilters` for additional information.

Output Arguments

`A` — Approximation coefficients
2-D matrix | 3-D array | 4-D array

Approximation coefficients, returned as a multidimensional array. The dimensions of `A` depend on the dimensions of the input `X` and the level of decomposition `N`.

- If `X` is *m*-by-*n*:
  - If `N` is greater than 1, then `A` is *m*-by-*n*-by-`N`. For 1 ≤ *i* ≤ `N`, `A(:,:,`*`i`*`)` contains the approximation coefficients at level *i*.
  - If `N` is equal to 1, then `A` is *m*-by-*n*.
- If `X` is *m*-by-*n*-by-3:
  - If `N` is greater than 1, then `A` is *m*-by-*n*-by-3-by-`N`. For 1 ≤ *i* ≤ `N` and *`j`*` = 1, 2, 3`, `A(:,:,`*`j`*`,`*`i`*`)` contains approximation coefficients at level *i*.
  - If `N` is equal to 1, then `A` is *m*-by-*n*-by-1-by-3. Since MATLAB removes singleton last dimensions by default, the third dimension is singleton.

Data Types: `double`

`H,V,D` — Detail coefficients
2-D matrix | 3-D array | 4-D array

Detail coefficients, returned as multidimensional arrays of equal size. `H`, `V`, and `D` contain the horizontal, vertical, and diagonal detail coefficients, respectively. The dimensions of the arrays depend on the dimensions of the input `X` and the level of decomposition `N`.

- If `X` is *m*-by-*n*:
  - If `N` is greater than 1, the arrays are *m*-by-*n*-by-`N`. For 1 ≤ *i* ≤ `N`, `H(:,:,`*`i`*`)`, `V(:,:,`*`i`*`)`, and `D(:,:,`*`i`*`)` contain the detail coefficients at level *i*.
  - If `N` is equal to 1, the arrays are *m*-by-*n*.
- If `X` is *m*-by-*n*-by-3:
  - If `N` is greater than 1, the arrays are *m*-by-*n*-by-3-by-`N`. For 1 ≤ *i* ≤ `N` and *`j`*` = 1, 2, 3`, `H(:,:,`*`j`*`,`*`i`*`)`, `V(:,:,`*`j`*`,`*`i`*`)`, and `D(:,:,`*`j`*`,`*`i`*`)` contain the detail coefficients at level *i*.
  - If `N` is equal to 1, the arrays are *m*-by-*n*-by-1-by-3. For *`j`*` = 1, 2, 3`, `H(:,:,1,`*`j`*`)`, `V(:,:,1,`*`j`*`)` and `D(:,:,1,`*`j`*`)` contain the detail coefficients. Since MATLAB removes singleton last dimensions by default, the third dimension is singleton.

Data Types: `double`

`swc` — Stationary wavelet decomposition
3-D array | 4-D array

Stationary wavelet decomposition, returned as a multidimensional array. `swc` is the concatenation of the approximation coefficients `A` and detail coefficients `H`, `V`, and `D`.

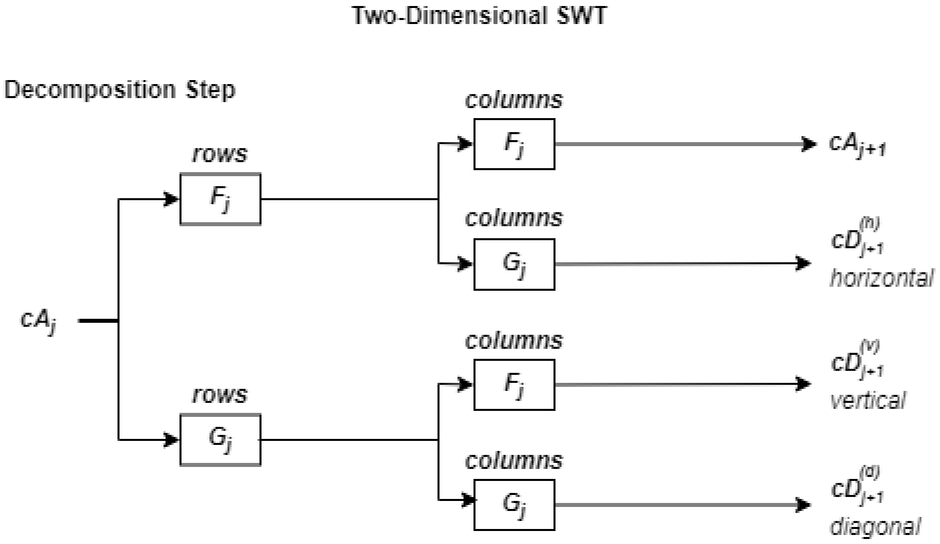
- If `X` is *m*-by-*n* and `N` is greater than 1, then `swc = cat(3,H,V,D,A(:,:,N))`.
- If `X` is *m*-by-*n* and `N` is equal to 1, then `swc = cat(3,H,V,D,A)`.
- If `X` is *m*-by-*n*-by-3 and `N` is greater than 1, then `swc = cat(4,H,V,D,A(:,:,:,N))`.
- If `X` is *m*-by-*n*-by-3 and `N` is equal to1, then `swc = cat(4,H,V,D,A)`.

Algorithms

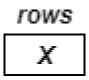
2-D Discrete Stationary Wavelet Transform

For images, a stationary wavelet transform (SWT) algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional functions by tensor product. This kind of two-dimensional SWT leads to a decomposition of approximation coefficients at level *j* into four components: the approximation at level *j*+1, and the details in three orientations (horizontal, vertical, and diagonal).

This chart describes the basic decomposition step for images.



where

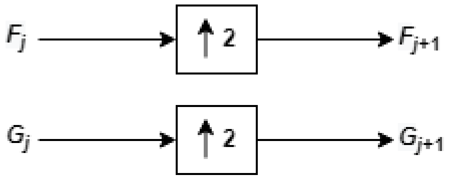
- 

— Convolve the rows of the entry with filter *X*.
-

— Convolve the columns of the entry with filter *X*.

Initialization

- $cA_0 = s$
- F_0 = `LoD`

- G_0 = `HiD`
- 

  where $\boxed{\uparrow 2}$ denotes upsample.

Note that $\mathtt{size}(cA_j) = \mathtt{size}(cD_j^{(h)}) = \mathtt{size}(cD_j^{(v)}) = \mathtt{size}(cD_j^{(d)}) = s$, where s equals the size of the analyzed image.

Truecolor Image Coefficient Arrays

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.

- If you perform a multilevel decomposition, the dimensions of `A`, `H`, `V`, and `D` are *m*-by-*n*-by-3-by-*k*, where *k* is the level of decomposition.
- If you perform a single-level decomposition, the dimensions of `A`, `H`, `V`, and `D` are *m*-by-*n*-by-1-by-3. Since MATLAB removes singleton last dimensions by default, the third dimension of the arrays is singleton.

Compatibility Considerations

Distinguish Single-Level Truecolor Image from Multilevel Indexed Image Decompositions

Behavior changed in R2017b

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.

- **Migrate from Previous Releases to R2017b**

  Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make `swt2` coefficient arrays from previous releases compatible with R2017b coefficient arrays. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Single Coefficient Array	Multiple Coefficient Arrays

Input: Truecolor image
• Single-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
• Multi-level: If `swc` is the output of `swt2` from a previous release, execute:
`swc1 = double(swc);`
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`
	`ca2 = reshape(ca1,[m,n,1,3]);`
	`chd2 = reshape(chd1,[m,n,1,3]);`
	`cvd2 = reshape(cvd1,[m,n,1,3]);`
	`cdd2 = reshape(cdd1,[m,n,1,3]);`
	• Multi-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from a previous release, execute:
	`ca1 = double(ca);`
	`chd1 = double(chd);`
	`cvd1 = double(cvd);`
	`cdd1 = double(cdd);`

- **Migrate from R2017b to Previous Releases**

  Depending on the original input data type and level of wavelet decomposition, you might have to take different steps to make R2017b `swt2` coefficient arrays compatible with the coefficient arrays from previous releases. The steps depend on whether you have a single coefficient array or separate approximation and detail coefficient arrays.

Single Coefficient Array	Multiple Coefficient Arrays
Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues	Input: Index image • Single-level: No compatibility issues • Multi-level: No compatibility issues

Input: Truecolor image
• Single-level: No compatibility issues
• Multi-level: No compatibility issues
	Input: Truecolor image
	• Single-level: If `ca`, `chd`, `cvd`, and `cdd` are outputs of `swt2` from R2017b, execute:
	`ca1 = single(squeeze(ca));`
	`chd1 = single(squeeze(chd));`
	`cvd1 = single(squeeze(cvd));`
	`cdd1 = single(squeeze(cdd));`
	• Multi-level: No compatibility issues

References

[1] Nason, G. P., and B. W. Silverman. “The Stationary Wavelet Transform and Some Statistical Applications.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:281–99. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_17.

[2] Coifman, R. R., and D. L. Donoho. “Translation-Invariant De-Noising.” In *Wavelets and Statistics*, edited by Anestis Antoniadis and Georges Oppenheim, 103:125–50. New York, NY: Springer New York, 1995. https://doi.org/10.1007/978-1-4612-2544-7_9.

[3] Pesquet, J.-C., H. Krim, and H. Carfantan. “Time-Invariant Orthonormal Wavelet Representations.” *IEEE Transactions on Signal Processing* 44, no. 8 (August 1996): 1964–70. https://doi.org/10.1109/78.533717.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input wavelet name must be constant.
- The input level of decomposition must be defined as a scalar during compilation.

See Also

`dwt2` | `iswt2` | `wavedec2`

Introduced before R2006a

symaux

Symlet wavelet filter computation

Syntax

```
w = symaux(n)
w = symaux( ___ ,sumw)
```

Description

The `symaux` function generates the scaling filter coefficients for the "least asymmetric" Daubechies wavelets.

`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.
- For `n` = 1, 2, and 3, the order `n` Symlet filters and order `n` Daubechies filters are identical. See “Extremal Phase” on page 1-1297.

`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.

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 coefficients is equal to $\sqrt{2}$ and the norm is equal to 1.

```
sqrt(2)-sum(w)
```

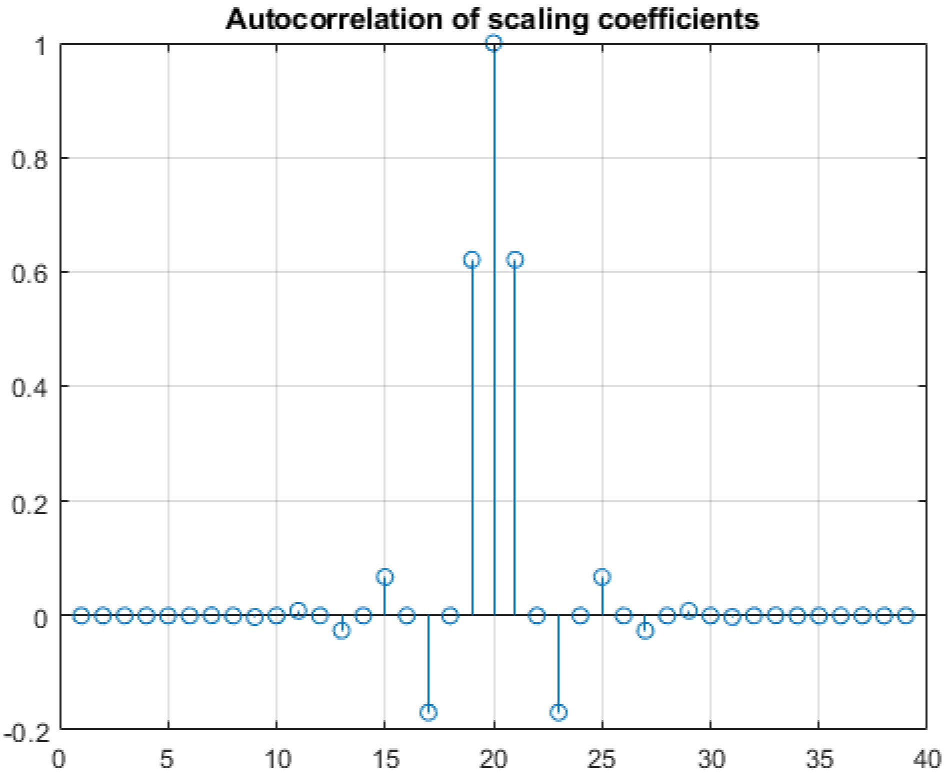
```
ans = 0
```

```
1-sum(w.^2)
```

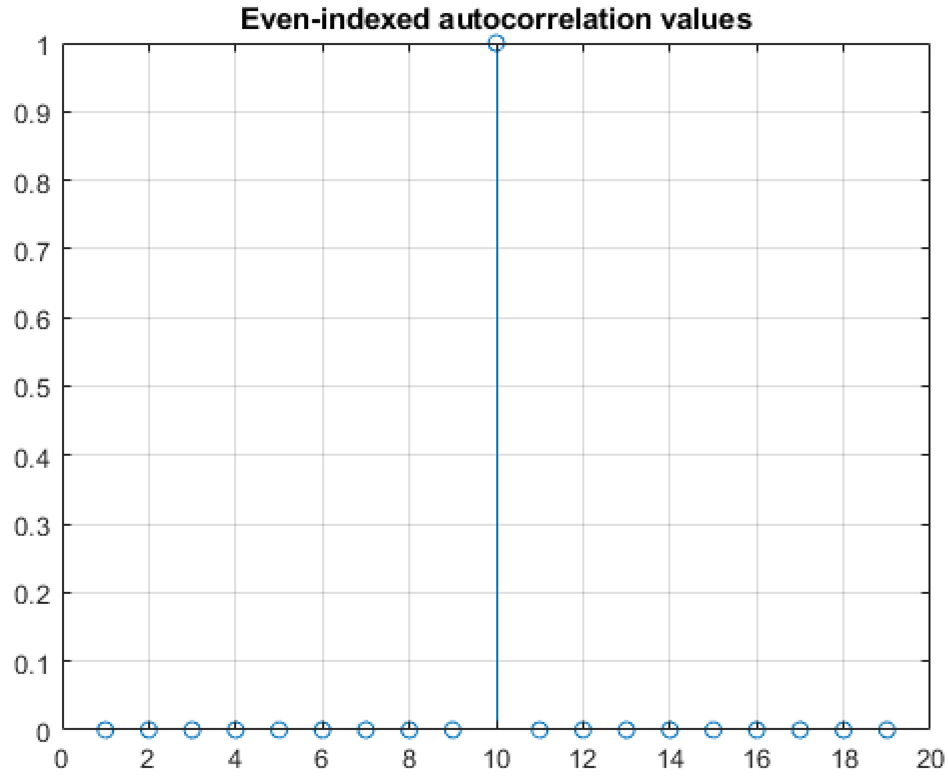
```
ans = 1.1324e-14
```

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

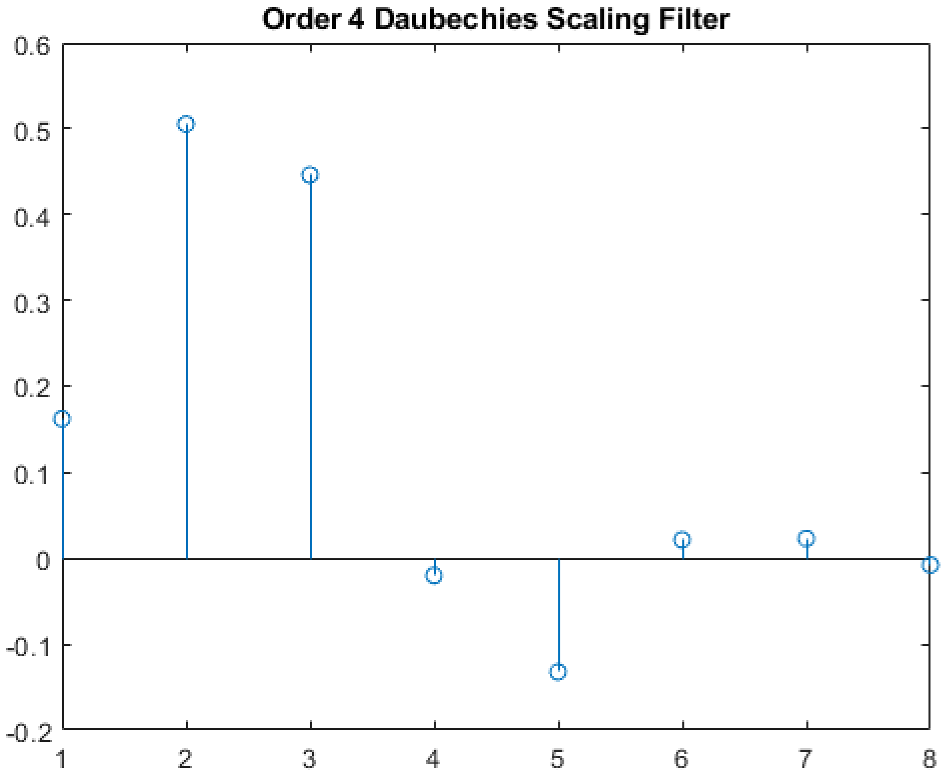
This example shows that symlet and Daubechies scaling filters of the same order are both solutions of the same polynomial equation.

Generate the order 4 Daubechies scaling filter and plot it.

```
wdb4 = dbaux(4)
```

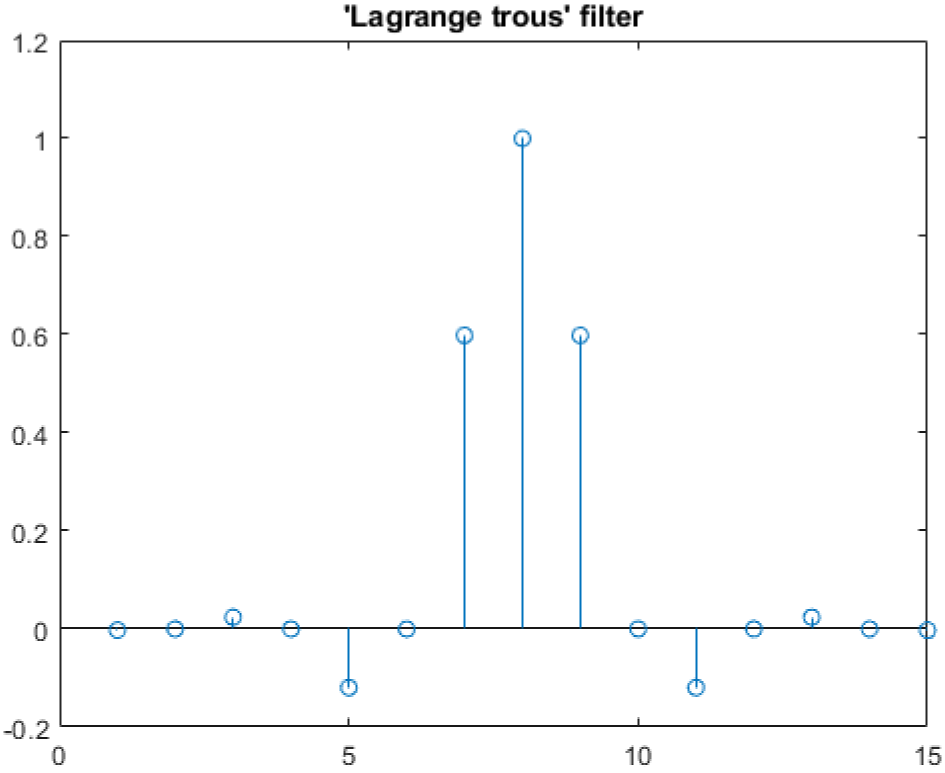
```
wdb4 = 1×8

    0.1629    0.5055    0.4461   -0.0198   -0.1323    0.0218    0.0233   -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')
```

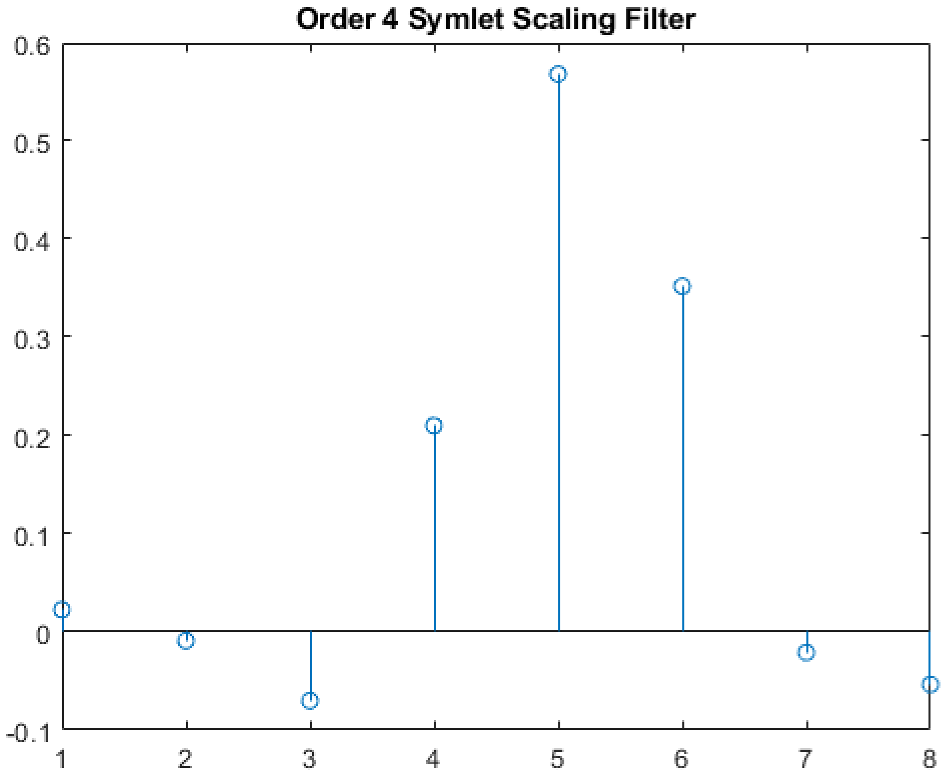


Generate wsym4, the order 4 symlet scaling filter and plot it. The Symlets are the "least asymmetric" Daubechies' wavelets obtained from another choice between the roots of P.

```
wsym4 = symaux(4)
```

```
wsym4 = 1×8

    0.0228   -0.0089   -0.0702    0.2106    0.5683    0.3519   -0.0210   -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 = 1.8677e-15
```

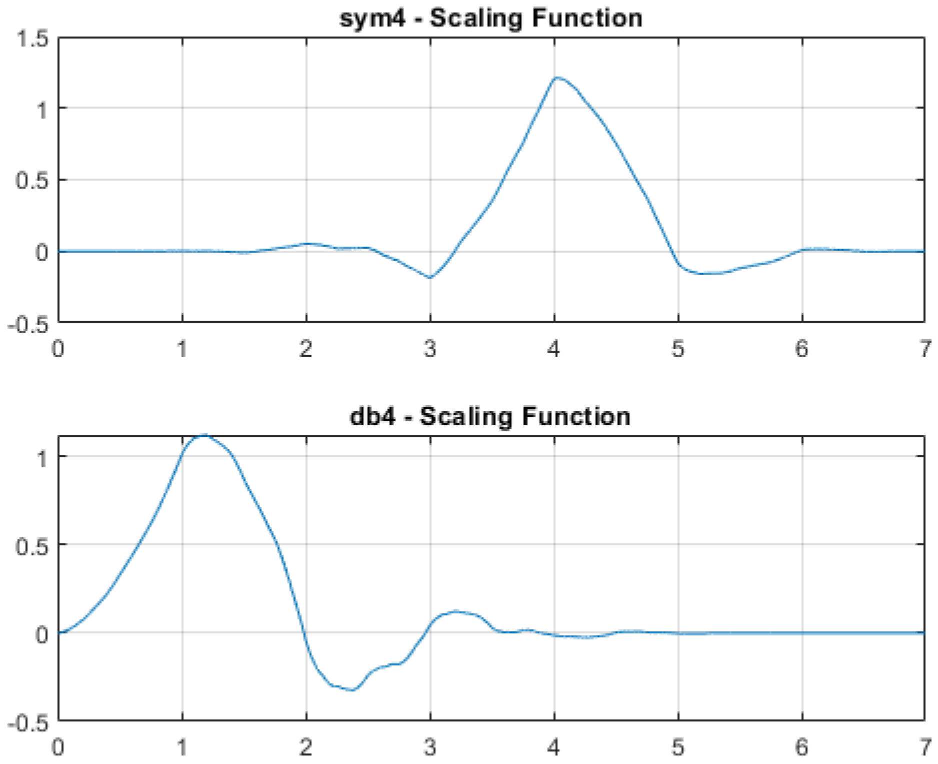
Least Asymmetric Wavelet and Phase

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.

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));
```

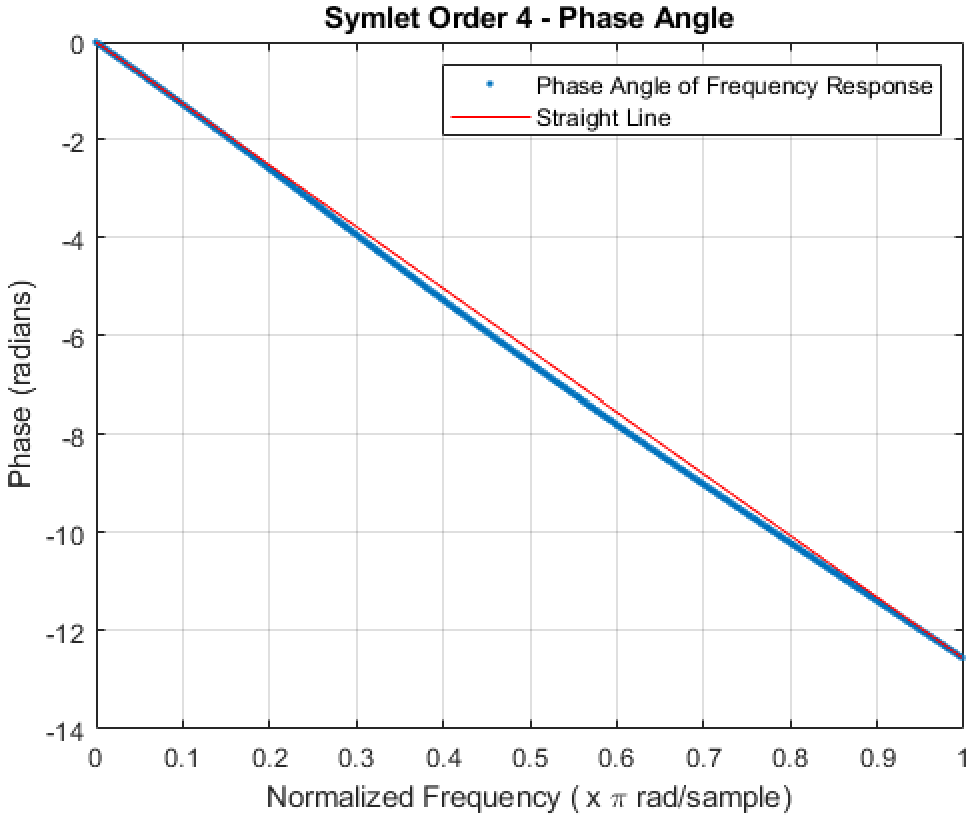
Compute the frequency response of the scaling synthesis filters.

```
[h_sym,w_sym] = freqz(scal_sym);
[h_db,w_db] = freqz(scal_db);
```

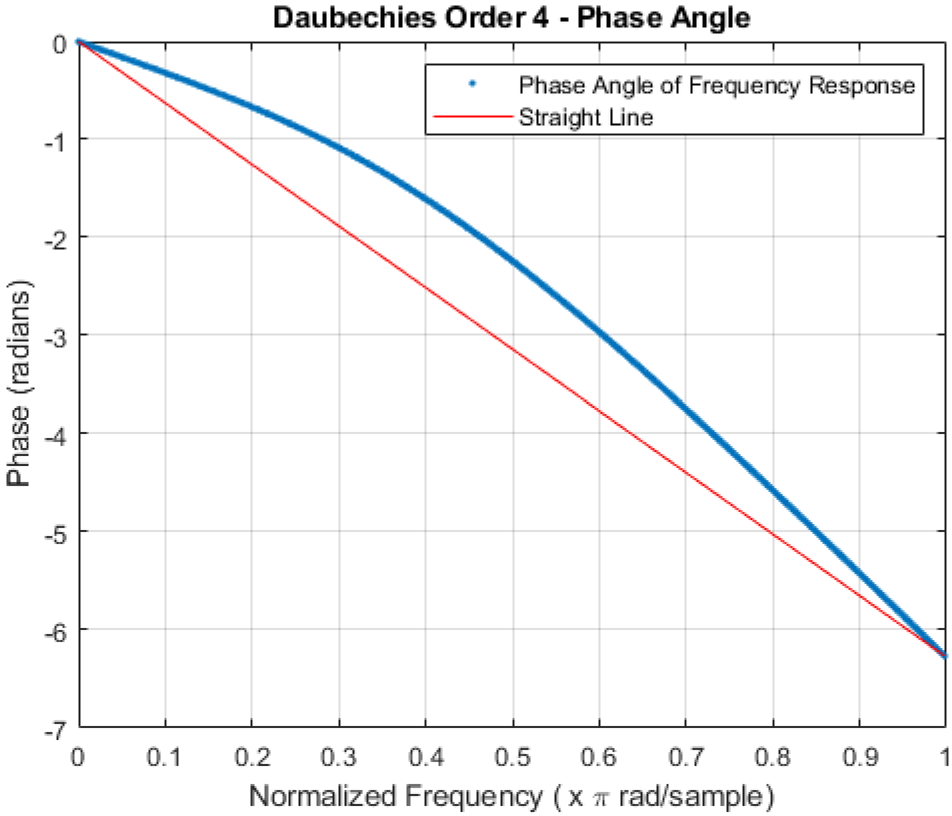
To avoid visual discontinuities, unwrap the phase angles of the frequency responses and plot them. Note how well the phase angle of the symlet filter approximates a straight line.

```
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')
```

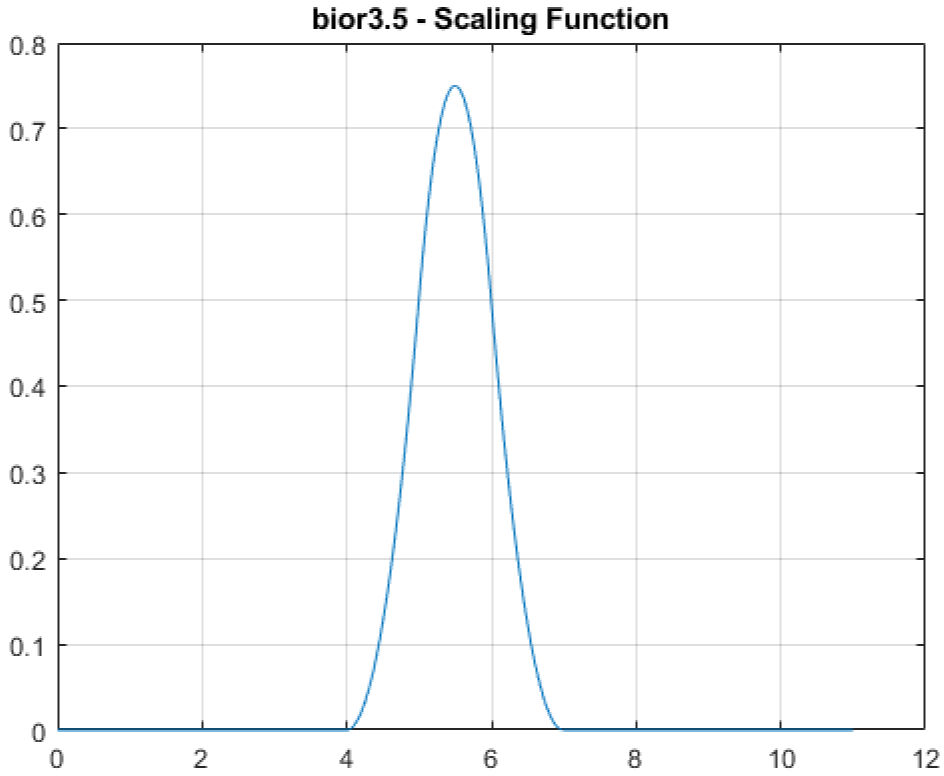


```
figure
plot(w_db/pi,h_db_u,'.')
hold on
plot(w_db([1 end])/pi,h_db_u([1 end]),'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('Phase Angle of Frequency Response','Straight Line')
title('Daubechies Order 4 - Phase Angle')
```

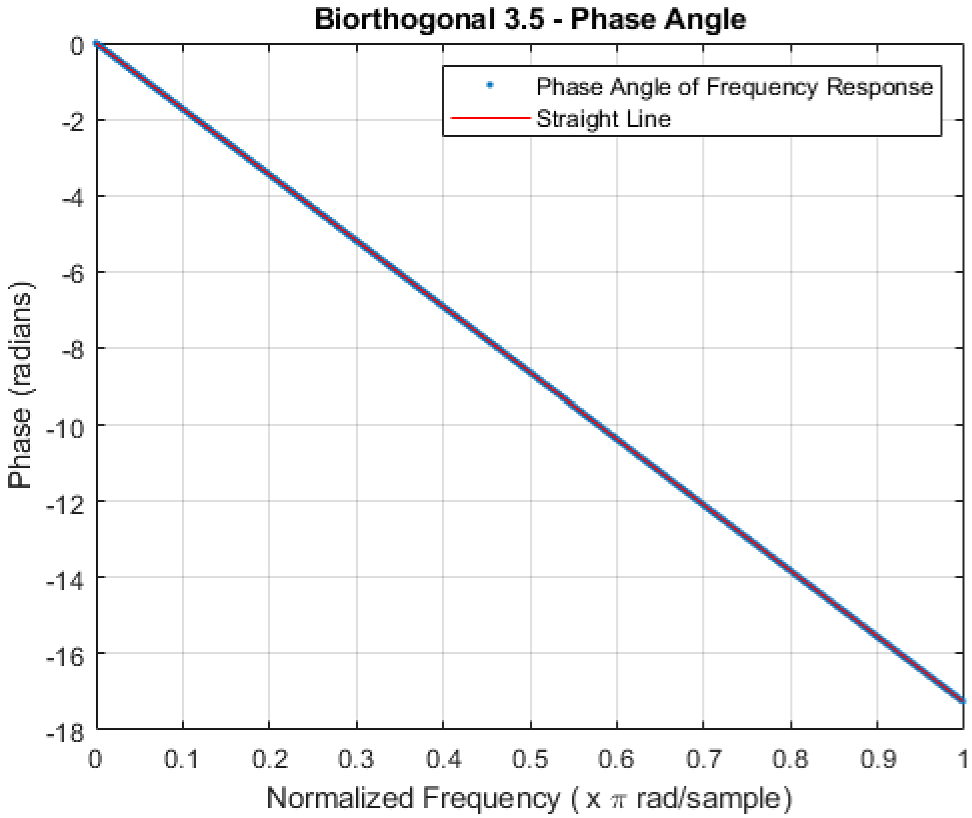


The sym4 and db4 wavelets are not symmetric, but the biorthogonal wavelet is. Plot the scaling function associated with the bior3.5 wavelet. Compute the frequency response of the synthesis scaling filter for the wavelet and verify that it has linear phase.

```
[~,~,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.

Generate the scaling filter coefficients for the db15 and sym15 wavelets. Both wavelets have support of width $2 \times 15 - 1 = 29$.

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

Next, generate the scaling filter coefficients for the coif5 wavelet. This wavelet also has support of width $6 \times 5 - 1 = 29$.

```
[~,~,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)
```

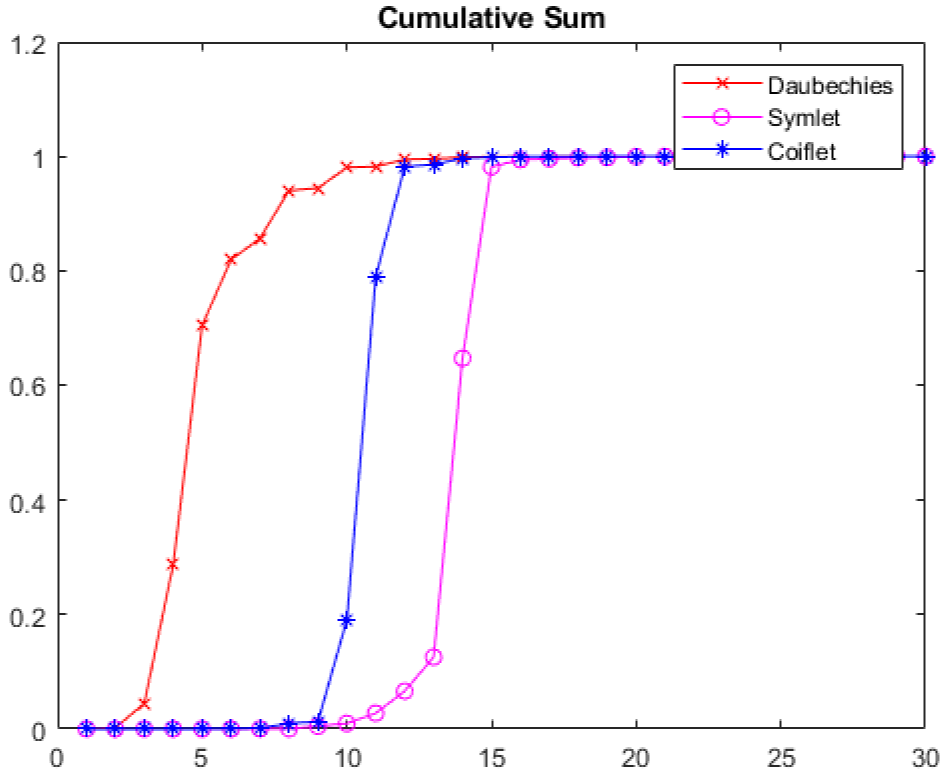
```
ans = 0
```

```
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 1-1285 for additional information.

More About

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.

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 $k = 1,\ldots,M$. Then for any other set of scaling coefficients $\{g_k\}$ resulting from a different choice of roots, the following inequality will hold for all $J = 1,\ldots,M$:

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

The $\{h_k\}$ are sometimes called a *minimal delay filter* [2].

The polynomial equation mentioned above depends on the desired number of vanishing moments N for the wavelet. To construct a wavelet basis involves choosing roots of the equation. In the case of least asymmetric wavelets and extremal phase wavelets for orders 1, 2, and 3, there are effectively no choices to make. For N = 1, 2, and 3, the `db`*N* and `sym`*N* filters are equal. The example "Symlet and Daubechies Scaling Filters" on page 1-1287 shows that two different scaling filters can satisfy the same polynomial equation. For additional information, see Daubechies [1].

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

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

See Also

`symwavf` | `wfilters` | `dbaux`

Introduced before R2006a

symwavf

Symlet wavelet filter

Syntax

```
f = symwavf(wname)
```

Description

`f = symwavf(wname)` returns the scaling filter associated with the Symlet wavelet specified by `wname`. `f` is a real-valued vector.

Examples

Scaling Filter Associated With the Symlet Wavelet

Specify the order 4 Symlet wavelet.

```
wname = 'sym4';
```

Compute the corresponding scaling filter.

```
f = symwavf(wname);
f'
```

```
ans = 8×1

    0.0228
   -0.0089
   -0.0702
    0.2106
    0.5683
    0.3519
   -0.0210
   -0.0536
```

Input Arguments

`wname` — Symlet wavelet
`'symN'`

Symlet wavelet with *N* vanishing moments, where *N* is a positive integer in the closed interval [1, 45].

See Also

`symaux` | `waveinfo`

Introduced before R2006a

thselect

Threshold selection for denoising

Syntax

```
THR = thselect(X,TPTR)
```

Description

`THR = thselect(X,TPTR)` returns the threshold value adapted to the 1-D signal `X` using the selection rule specified by `TPTR`. Available selection rules are:

- `'rigrsure'` — Adaptive threshold selection using the principle of Stein's Unbiased Risk Estimate (SURE).
- `'sqtwolog'` — Fixed-form threshold is `sqrt(2*log(length(X)))`.
- `'heursure'` — Heuristic variant of `'rigrsure'` and `'sqtwolog'`.
- `'minimaxi'` — Minimax thresholding.

Examples

Threshold Selection Rules

Generate a Gaussian white noise signal. For reproducibility, set the random seed to the default value.

```
rng default
x = randn(1,1000);
```

Find the threshold for each selection rule.

```
thrRig = thselect(x,'rigrsure');
disp(['SURE (''rigrsure'') threshold: ',num2str(thrRig)]);
```

```
SURE ('rigrsure') threshold: 2.0518
```

```
thrSqt = thselect(x,'sqtwolog');
disp(['Universal (''sqtwolog'') threshold: ',num2str(thrSqt)]);
```

```
Universal ('sqtwolog') threshold: 3.7169
```

```
thrHeu = thselect(x,'heursure');
disp(['Heuristic variant (''heursure'') threshold: ',num2str(thrHeu)]);
```

```
Heuristic variant ('heursure') threshold: 3.7169
```

```
thrMin = thselect(x,'minimaxi');
disp(['Minimax (''minimaxi'') threshold: ',num2str(thrMin)]);
```

```
Minimax ('minimaxi') threshold: 2.2163
```

Minimax and SURE threshold selection rules are more conservative and would be more convenient when small details of the signal lie near the noise range.

Input Arguments

X — Input data

real-valued vector

Input data, specified as a real-valued vector.

Data Types: `double`

TPTR — Threshold selection rule

`'rigrsure'` | `'heursure'` | `'sqtwolog'` | `'minimaxi'`

Threshold selection rule, specified:

- `'rigrsure'` — A threshold selection rule based on SURE (a quadratic loss function) for the soft threshold estimator. Starting with an estimate of risk for a particular threshold value, *t*, the algorithm minimizes the risks in *t* to yield a threshold value.
- `'heursure'` — A mixture of `'rigrsure'` and `'sqtwolog'`. If the signal-to-noise ratio is small, the SURE estimate is noisy. In that case, the fixed-form threshold is used.
- `'sqtwolog'` — A fixed-form (universal) threshold yielding minimax performance multiplied by a small factor proportional to `log(length(X))`.
- `'minimaxi'` — 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. The denoised signal can be assimilated to the estimator of the unknown regression function. Therefore, the minimax estimator realizes the minimum of the maximum mean square error obtained for the worst function in a given set.

Threshold selection rules are based on the underlying model $y = f(t) + e$, where e is an $N(0,1)$ white noise. Use level-dependent noise estimates for unscaled or nonwhite noise. (See `NoiseEstimate` parameter in `wdenoise` for more information.)

Output Arguments

THR — Threshold

positive real number

Threshold value adapted to `X`, returned as a positive real number.

References

[1] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[2] Donoho, D. L., and Johnstone, I. M. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*, Vol. 81, pp. 425–455, 1994.

[3] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*, Vol. 42, Number 3, pp. 613–627, 1995.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`wdenoise` | `wdenoise2`

Apps

Wavelet Signal Denoiser

Topics

“Denoise a Signal with the Wavelet Signal Denoiser”

Introduced before R2006a

Wavelet Time-Frequency Analyzer

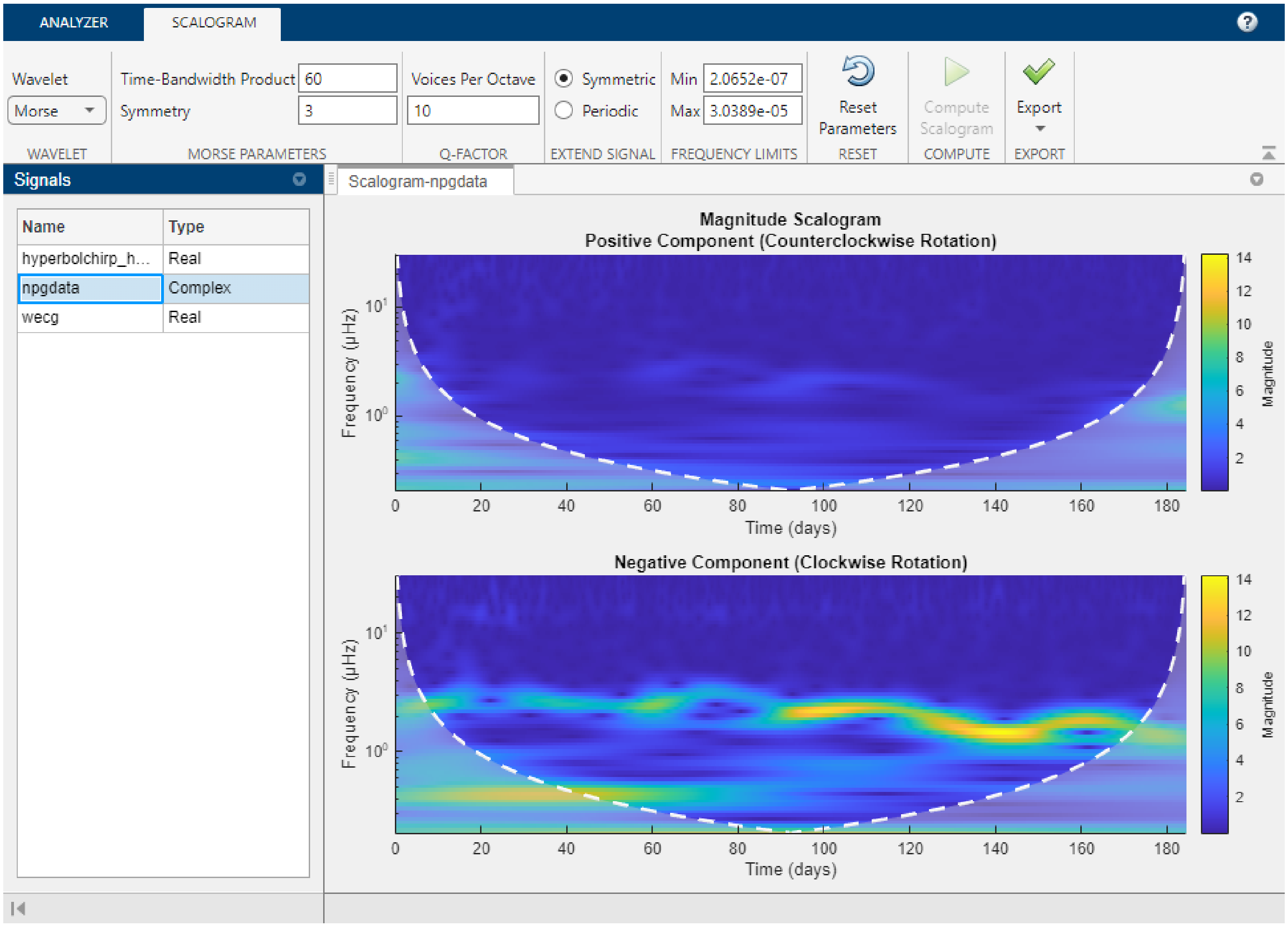
Visualize scalogram of signals

Description

The **Wavelet Time-Frequency Analyzer** app is an interactive tool for visualizing scalograms of real- and complex-valued 1-D signals. The scalogram is the absolute value of the continuous wavelet transform (CWT) plotted as a function of time and frequency. Frequency is plotted on a logarithmic scale. With the app, you can:

- Access all 1-D signals in your MATLAB workspace
- Import multiple signals simultaneously
- Adjust default parameters and visualize scalograms using `cwt`
- Select desired analytic wavelet
- Adjust analytic Morse wavelet symmetry and time-bandwidth parameters
- Export the CWT to your workspace
- Recreate the scalogram in your workspace by generating a MATLAB script
- Import multiple signals

For more information, see “Using Wavelet Time-Frequency Analyzer App”.



Wavelet: Morse

Time-Bandwidth Product: 60

Symmetry: 3

Voices Per Octave: 10

● Symmetric

○ Periodic

Min: 2.0652e-07

Max: 3.0389e-05

Name	Type
hyperbolchirp_h...	Real
npgdata	Complex
wecg	Real

Open the Wavelet Time-Frequency Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `waveletTimeFrequencyAnalyzer`.

Examples

Visualize Scalograms Using Default Settings

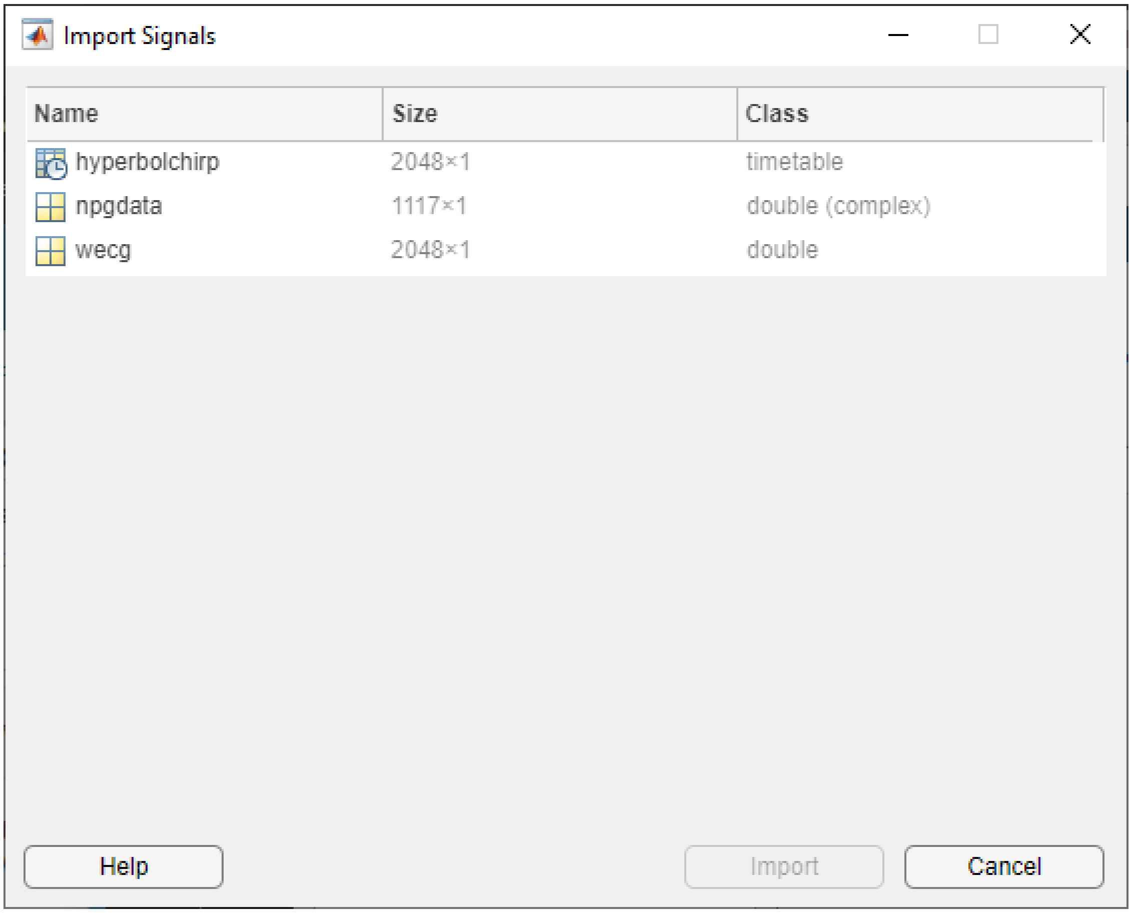
Load three 1-D signals into the MATLAB® workspace: an electrocardiogram signal, a hyperbolic chirp signal, and the NPG2006 dataset.

```
load wecg
load hyperbolchirp
load npg2006
```

Extract the complex-valued signal from the `npg2006` structure array.

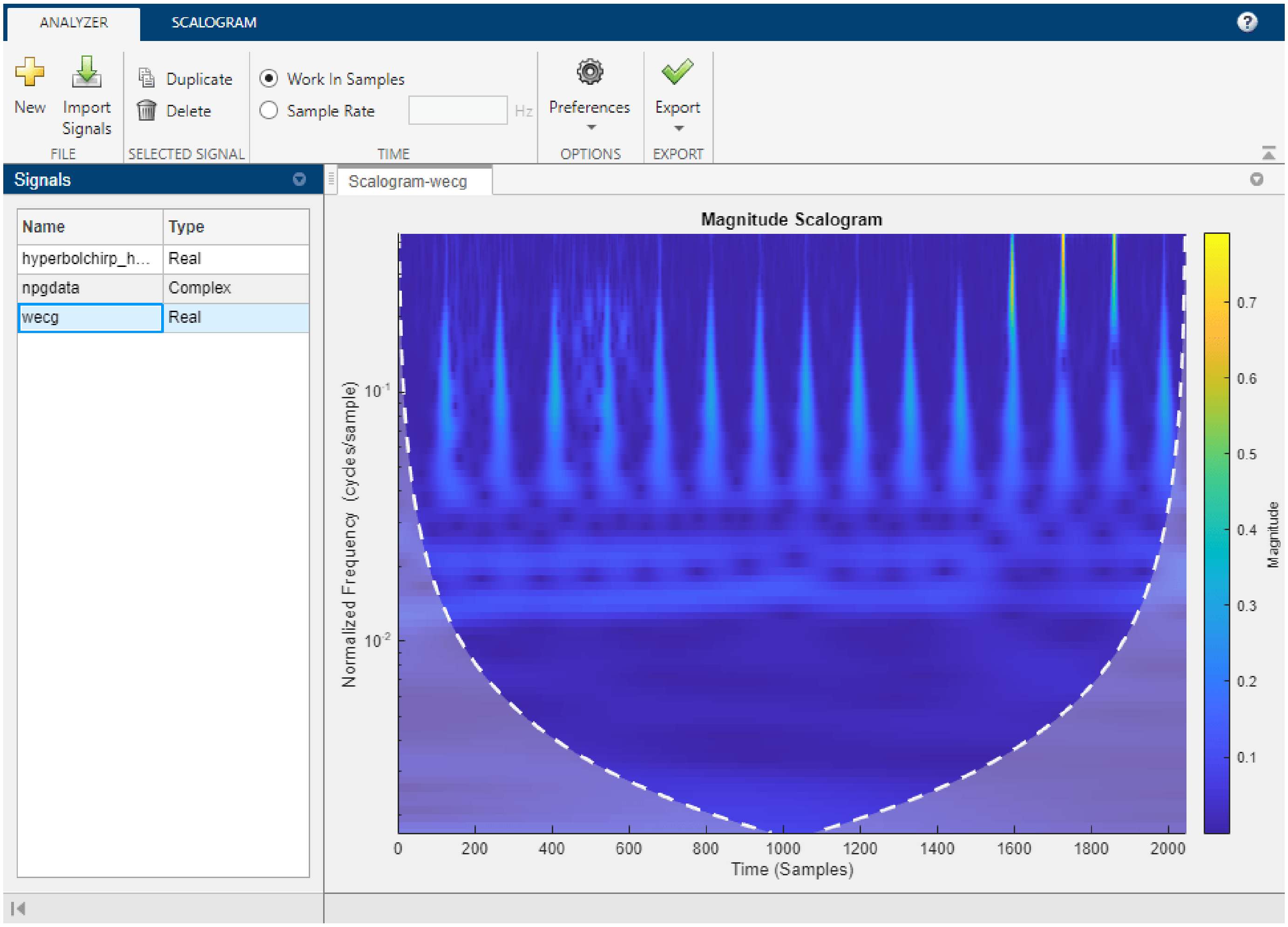
```
npgdata = npg2006.cx;
whos

  Name                   Size             Bytes  Class        Attributes

  hyperbolchirp       2048x1              33759  timetable              
  npg2006                1x1              37184  struct                 
  npgdata             1117x1              17872  double       complex   
  wecg                2048x1              16384  double                 
```

Open **Wavelet Time-Frequency Analyzer** and click **Import Signals**. A window appears listing all the workspace variables the app can process.



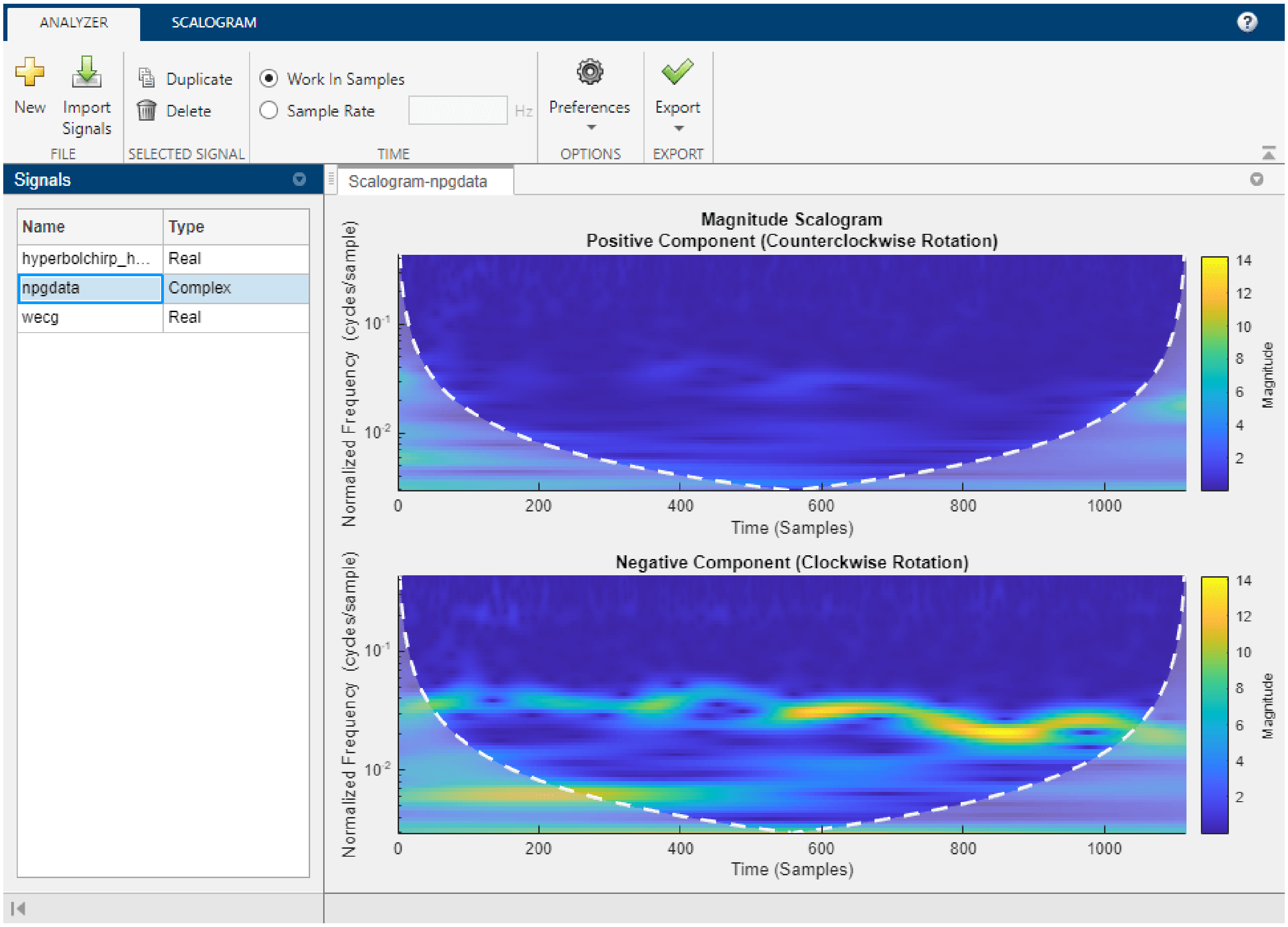
Name	Size	Class
hyperbolchirp	2048×1	timetable
npgdata	1117×1	double (complex)
wecg	2048×1	double

Select all the signals and click **Import**. After a brief, one-time initialization, the **Signals** pane is populated with the names of the imported signals, along with their types. In the case of `hyperbolchirp`, the name of the timetable variable containing the signal is appended to the name of the timetable: `hyperbolchirp_hchirp`. The app displays the scalogram of the highlighted signal. The scalogram is obtained using the `cwt` function with default settings. The cone of influence showing where edge effects become significant is also plotted. Gray regions outside the dashed white lines delineate regions where edge effects are significant. By default, frequencies are in cycles/sample.



Name	Type
hyperbolchirp_h…	Real
npgdata	Complex
wecg	Real

The ECG signal is real valued. In the **Signals** pane, select the complex-valued signal `npgdata`. The positive and negative components of the scalogram are displayed.

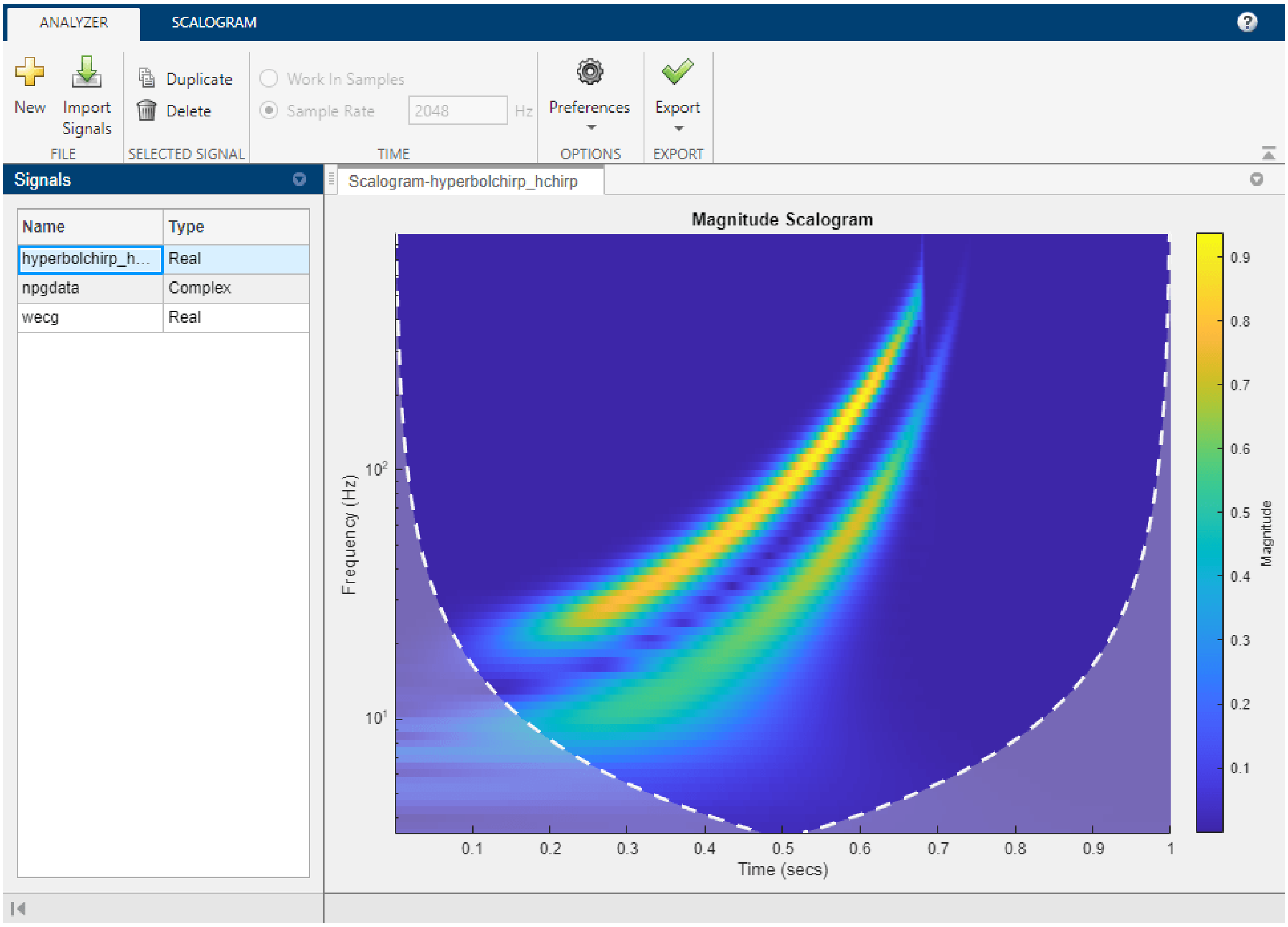


● Work In Samples

○ Sample Rate

Name	Type
hyperbolchirp_h...	Real
npgdata	Complex
wecg	Real

Select `hyperbolchirp_hchirp`. Because the timetable contains temporal information, the scalogram is plotted as a function of frequency in hertz and uses the row times of the timetable as the basis for the time axis. The disabled **Sample Rate** field displays the sampling rate as determined from the row times.



○ Work In Samples

● Sample Rate 2048 Hz

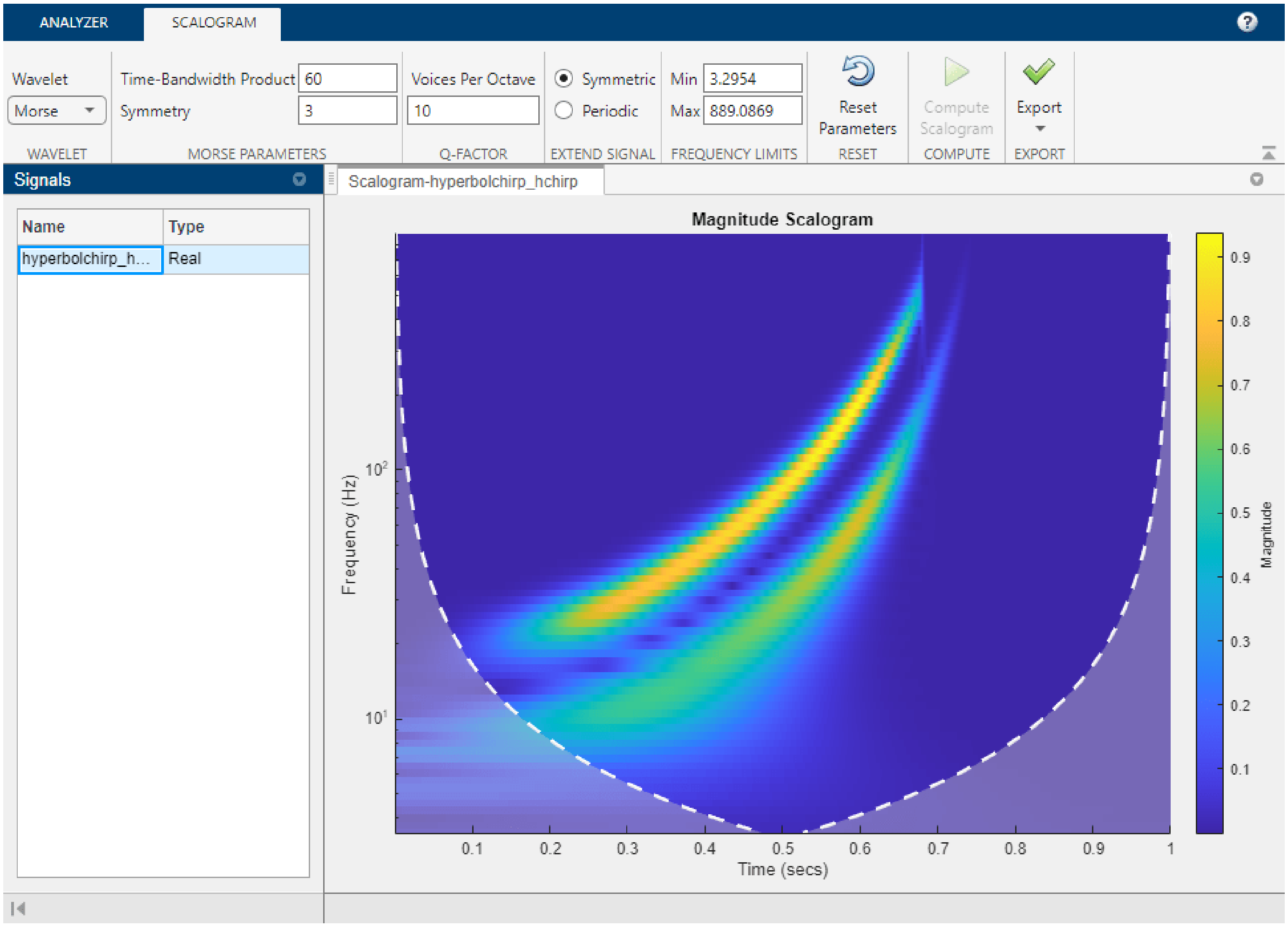
Name	Type
hyperbolchirp_h...	Real
npgdata	Complex
wecg	Real

Adjust Morse Wavelet Parameters

Load the hyperbolic chirp signal.

```
load hyperbolchirp
```

Open **Wavelet Time-Frequency Analyzer** and import the signal into the app. To access the parameter settings, click the **Scalogram** tab. By default, the app displays the scalogram obtained using the analytic Morse (3, 60) wavelet and the `cwt` function with default settings.



Wavelet: Morse

Time-Bandwidth Product: 60

Symmetry: 3

Voices Per Octave: 10

● Symmetric

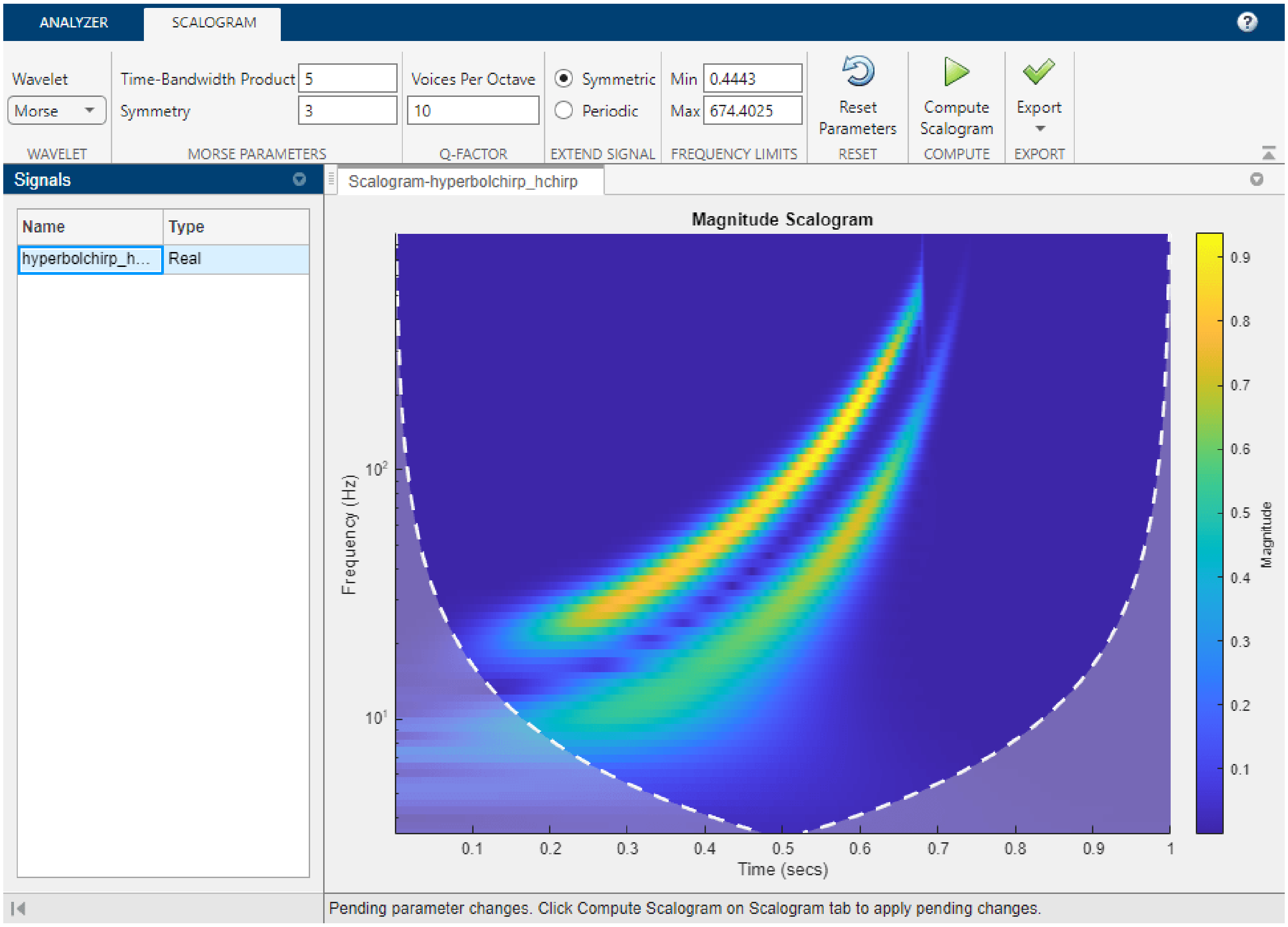
○ Periodic

Min: 3.2954

Max: 889.0869

Name	Type
hyperbolchirp_h...	Real

To visualize the scalogram using the (1,5) Morse wavelet, set **Time-Bandwidth Product** to 5. In the status bar, text appears stating there are pending changes. The **Compute Scalogram** button is now enabled. If you instead first set **Symmetry** to 1, the app would automatically change that value because a symmetry value of 1 violates the constraint that the ratio of **Time-Bandwidth Product** to **Symmetry** should not exceed 40. For more information, see “Tips” on page 1-1318.



Wavelet: Morse

Time-Bandwidth Product: 5

Symmetry: 3

Voices Per Octave: 10

● Symmetric
○ Periodic

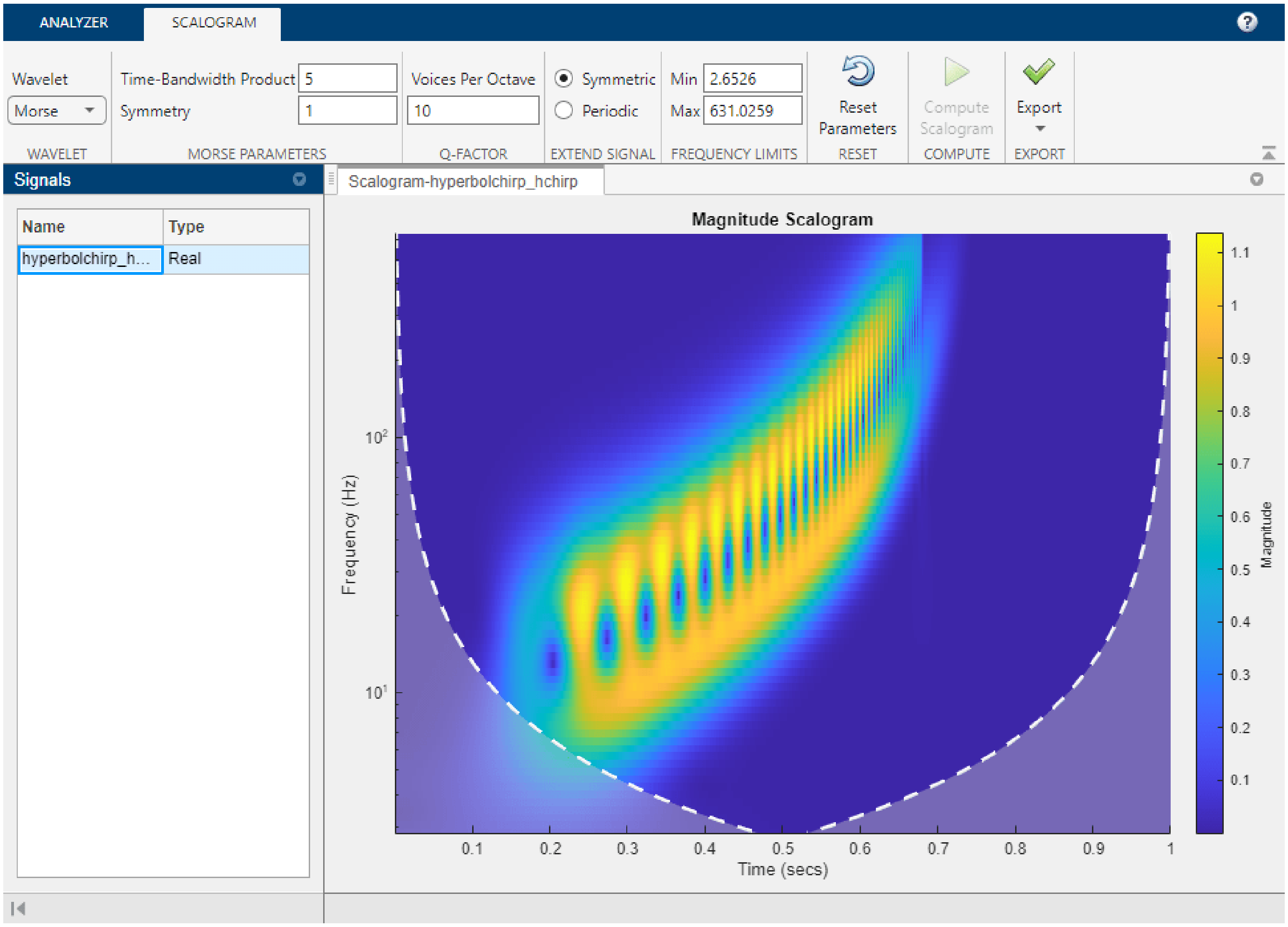
Min: 0.4443

Max: 674.4025

Name	Type
hyperbolchirp_h…	Real

Pending parameter changes. Click Compute Scalogram on Scalogram tab to apply pending changes.

Now set **Symmetry** to 1 and click **Compute Scalogram**. The app displays the scalogram obtained using the (1, 5) Morse wavelet.



Time-Bandwidth Product: 5
Symmetry: 1
Voices Per Octave: 10
● Symmetric
○ Periodic
Min: 2.6526
Max: 631.0259

Name	Type
hyperbolchirp_h...	Real

To visualize the scalogram using the (6, 50) Morse wavelet, first set **Time-Bandwidth Product** to 50 and **Symmetry** to 6, then click **Compute Scalogram**.

Adjust Scalogram Frequency Axis Scale

Import Signal

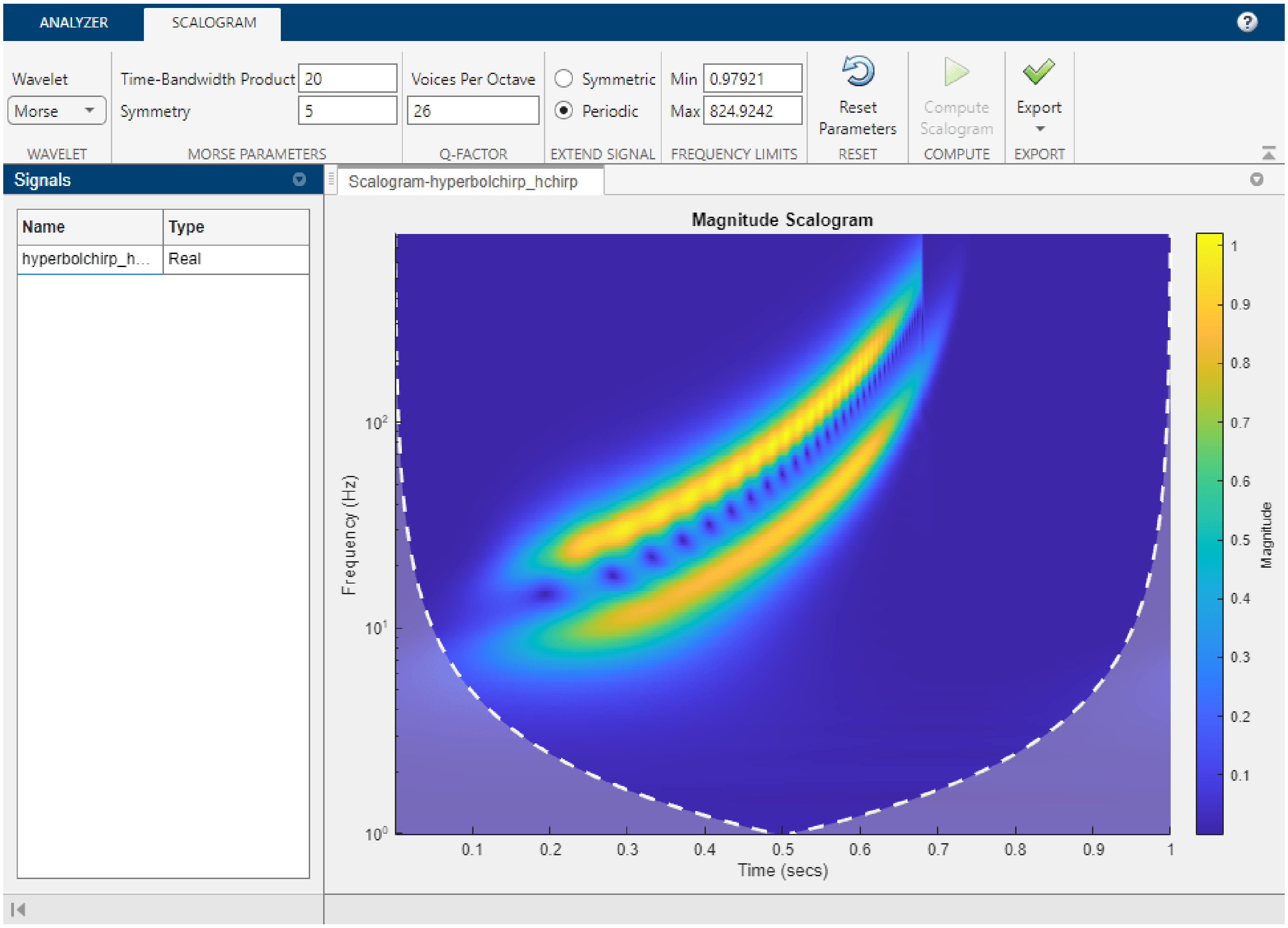
Load a hyperbolic chirp signal into your workspace.

```
load hyperbolchirp
```

Visualize Scalogram

Open **Wavelet Time-Frequency Analyzer** and import the signal. To access the parameter settings, click the **Scalogram** tab. By default, the app displays the scalogram obtained using the Morse (3,60) wavelet and the `cwt` function with default settings. Because the signal is a timetable, the scalogram is plotted as a function of frequency in hertz. The time axis is based on the row times of the timetable.

In the **Scalogram** tab, adjust the default settings. Specify the Morse (5,20) wavelet. Visualize the scalogram using 26 voices per octave and periodic boundary extension. The frequencies are plotted on a logarithmic scale.



Wavelet: Morse
Time-Bandwidth Product: 20
Symmetry: 5
Voices Per Octave: 26
○ Symmetric
● Periodic
Min: 0.97921
Max: 824.9242

Name	Type
hyperbolchirp_h...	Real

Generate Script

To adjust the frequency axis scale in the scalogram, first reproduce the wavelet analysis in your workspace.

Generate a script that recreates the scalogram in your workspace. From the **Export ▼** menu, select `Generate MATLAB Script`. An untitled script opens in your MATLAB® Editor. To include the boundary line of the cone of influence in the plot, add a third output argument, `coi`, to the `cwt` function call in the script. Save and execute the script. The variables `scalogram` and `frequency` contain the scalogram and frequency vector, respectively.

```
%Parameters
waveletParameters = [5,20];
voicesPerOctave = 26;
extendSignal = false;

%Compute cwt
%Run the function call below without output arguments to plot the results
```

```
[waveletTransform,frequency,coi] = cwt(hyperbolchirp,...
    WaveletParameters = waveletParameters,...
    VoicesPerOctave = voicesPerOctave,...
    ExtendSignal = extendSignal);
scalogram = abs(waveletTransform);
```

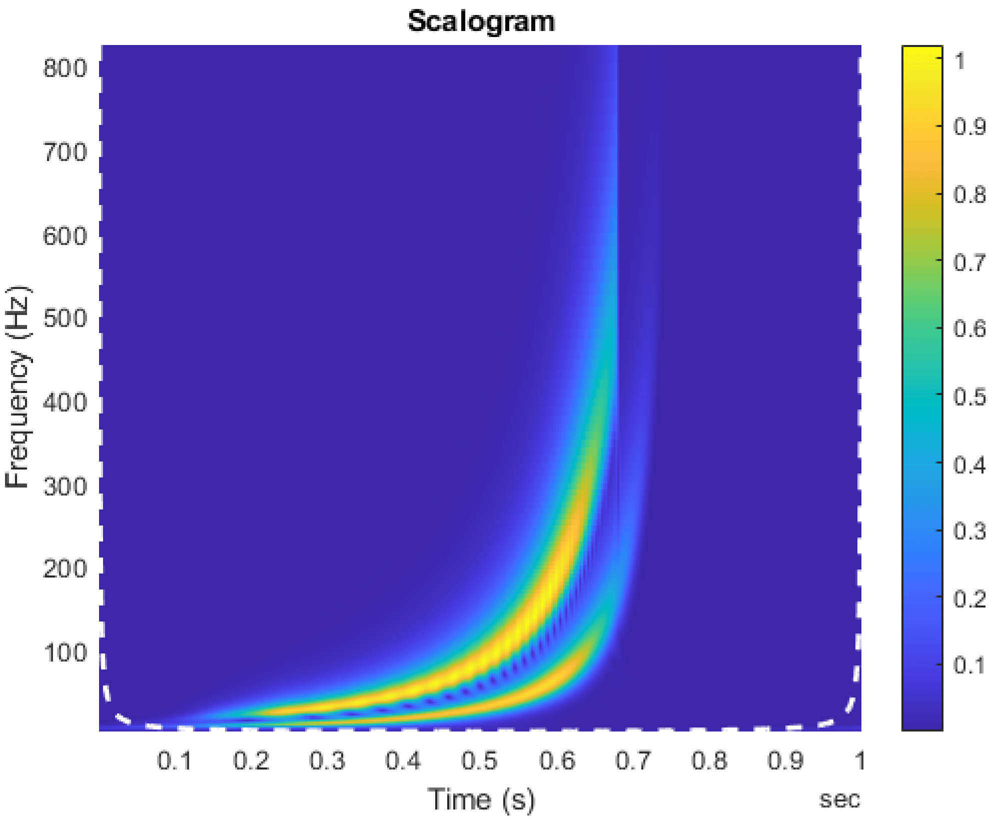
In order to plot the scalogram using the correct time axis, extract the vector of row times, `Time`, from the `hyperbolchirp` timetable.

```
dataTimes = hyperbolchirp.Time;
```

Adjust Frequency Axis — Linear Scale

Use the `pcolor` function to plot the scalogram. Include the cone of influence boundary. Frequency is plotted on a linear scale.

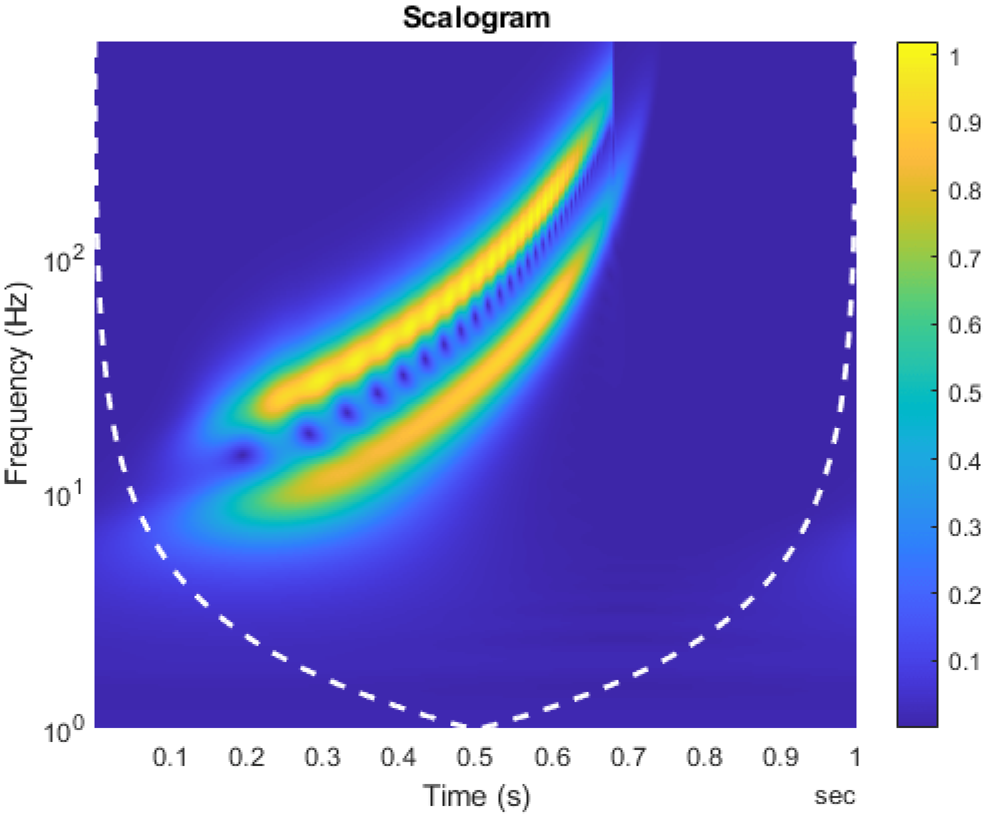
```
pcolor(dataTimes,frequency,scalogram)
shading flat
title("Scalogram")
xlabel("Time (s)")
ylabel("Frequency (Hz)")
hold on
plot(dataTimes,coi,"w--",LineWidth=2)
hold off
colorbar
```



Adjust Frequency Axis — Logarithmic Scale

To plot frequency on a logarithmic scale, get the handle to the current axes and set `YScale` to `"log"`.

```
AX = gca;
set(AX,YScale="log")
```

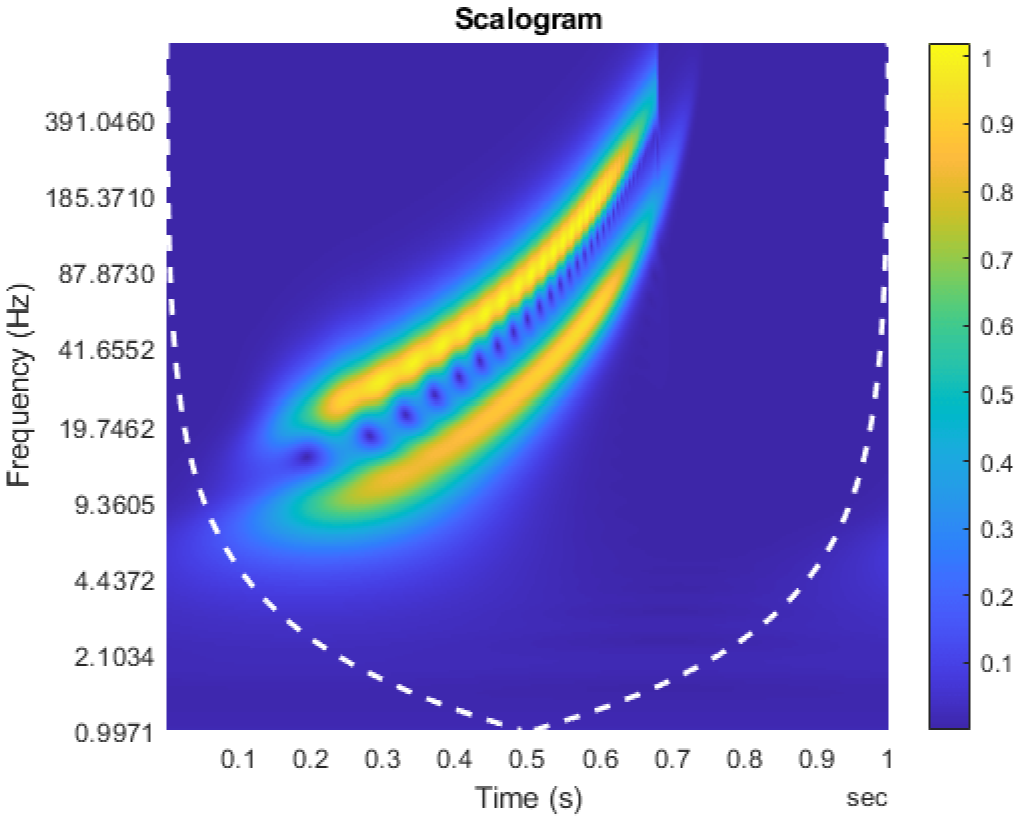


In MATLAB, logarithmic axes are in powers of 10 (decades). By default, MATLAB places frequency ticks at 1, 10, and 100 because they are the powers of 10 between the minimum and maximum frequencies. To add more frequency axis ticks, obtain the minimum and maximum frequencies in `frequency`. Create a logarithmically spaced set of frequencies between the minimum and maximum frequencies. Note that `cwt` returns frequencies ordered from high to low.

```
minf = frequency(end);
maxf = frequency(1);
numfreq = 10;
freq = logspace(log10(minf),log10(maxf),numfreq);
```

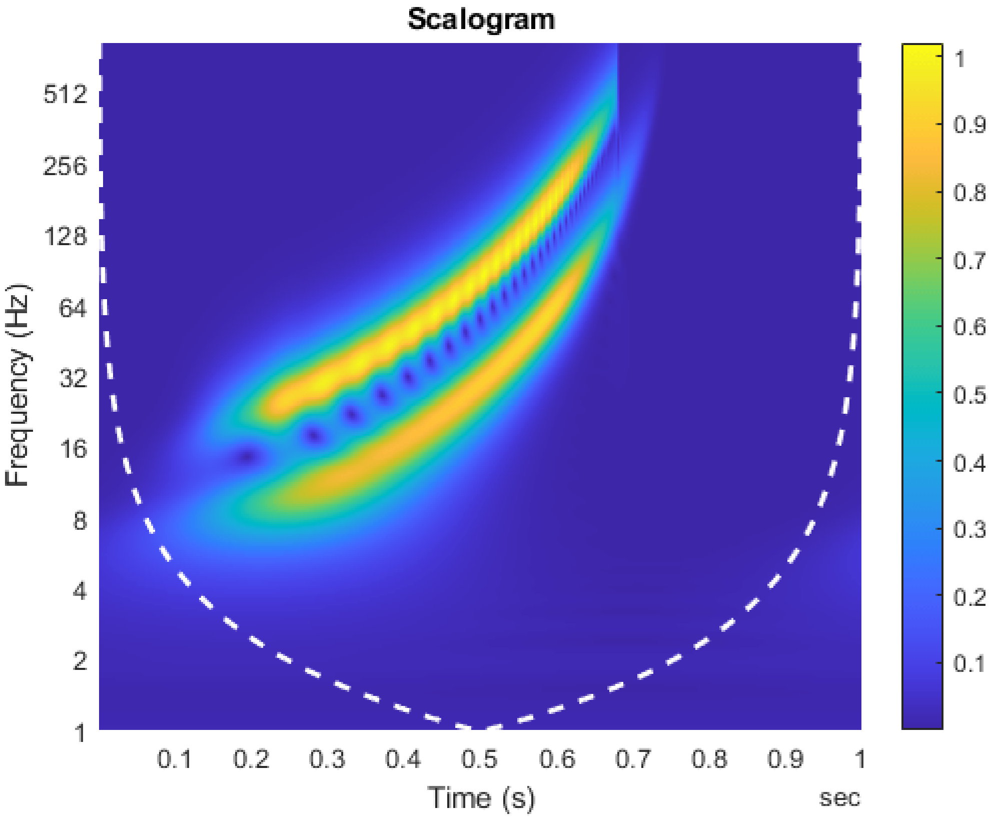
Replace the frequency axis ticks and labels with the new frequencies.

```
AX.YTickLabelMode = "auto";
AX.YTick = freq;
```



In the CWT, frequencies are computed in powers of two. Create the frequency ticks and tick labels in powers of two.

```
freq = 2.^(round(log2(minf)):round(log2(maxf)));
AX.YTickLabelMode = "auto";
AX.YTick = freq;
```



- “Using Wavelet Time-Frequency Analyzer App”

Parameters

`Wavelet` — Analytic wavelet

`Morse` (default) | `Morlet` | `bump`

Analytic wavelet used to compute the CWT. Valid options are `Morse`, `Morlet`, and `bump`, which specify the Morse, Morlet (Gabor), and bump wavelet, respectively.

`Time-Bandwidth Product` — Time-bandwidth product of the Morse wavelet

`60` (default) | scalar greater than or equal to the **Symmetry** value

Specify the time-bandwidth product of the Morse wavelet as a scalar greater than or equal to the **Symmetry** value. The ratio of the **Time-Bandwidth Product** value to the **Symmetry** value cannot exceed 40.

The values of **Time-Bandwidth Product** and **Symmetry** correspond to the `WaveletParameters` name-value argument of `cwt`.

Example: Setting **Time-Bandwidth Product** to 40 and **Symmetry** value to 5 is equivalent to setting the `WaveletParameters` name-value argument `cwt(…,WaveletParameters=[5 40],…)`.

`Symmetry` — Symmetry parameter of the Morse wavelet

`3` (default) | scalar greater than or equal to `1`

Specify the symmetry parameter of the Morse wavelet as a scalar greater than or equal to 1. The ratio of the **Time-Bandwidth Product** value to the **Symmetry** value cannot exceed 40.

The values of **Symmetry** and **Time-Bandwidth Product** correspond to the `WaveletParameters` name-value argument of `cwt`.

`Voices Per Octave` — Number of voices per octave

`10` (default) | integer between 1 and 48

Specify the number of voices per octave to use for the CWT as an integer from 1 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.

Programmatic Use

`waveletTimeFrequencyAnalyzer` opens the **Wavelet Time-Frequency Analyzer** app. Once the app initializes, import a signal for analysis by clicking **Import Signals**.

`waveletTimeFrequencyAnalyzer(sig)` opens the **Wavelet Time-Frequency Analyzer** app and imports, generates, and plots the scalogram of `sig` using `cwt` with default settings.

`sig` is a variable in the workspace. `sig` can be:

- A real- or complex-valued vector.
- A single-variable regularly sampled timetable.
- Single or double precision.

`sig` must have at least four samples.

By default, the app plots the scalogram as a function of frequency in cycles/sample and uses sample index as the basis of the time axis. If the signal is a timetable, then the app plots the scalogram as a function of frequency in hertz and uses the row times of the timetable as the basis for the time axis.

Limitations

- The MATLAB script you generate to create the scalogram in your workspace uses the name of the selected signal in the **Signals** pane. The script will throw an error if the variable does not exist in the MATLAB workspace. If an error occurs, either replace the variable name in the script with the name of the original signal or create the variable in your workspace.
- You can run only one instance of the **Wavelet Time-Frequency Analyzer** app in a MATLAB session.

Tips

- The Morse wavelet parameters, **Time-Bandwidth Product** and **Symmetry**, must satisfy three constraints:
  - **Symmetry**, or gamma, must be greater than or equal to 1.
  - **Time-Bandwidth Product** must be greater than or equal to **Symmetry**.

- The ratio of **Time-Bandwidth Product** to **Symmetry** cannot exceed 40.

To prevent attempts to visualize a scalogram using invalid settings, the app validates any parameter you change. If you enter a value that violates a constraint, the app automatically replaces it with a valid value. The new value might not be the desired value. To avoid unexpected results, you should ensure any value you enter always results in a valid setting. For more information, see the example “Adjust Morse Wavelet Parameters” on page 1-1309.

See Also

Apps
Signal Multiresolution Analyzer | **Signal Analyzer**

Functions
`cwt` | `cwtfilterbank` | `cwtfreqbounds`

Topics
“Using Wavelet Time-Frequency Analyzer App”
“Morse Wavelets”
“Practical Introduction to Continuous Wavelet Analysis”

Introduced in R2022a

timeSpectrum

Time-averaged wavelet spectrum

Syntax

```
tavgp = timeSpectrum(fb,x)
tavgp = timeSpectrum(fb,cfs)
[tavgp,f] = timeSpectrum( ___ )

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

timeSpectrum( ___ )
```

Description

`tavgp = timeSpectrum(fb,x)` returns the time-averaged wavelet power spectrum of the signal `x` using the continuous wavelet transform (CWT) filter bank `fb`. By default, `tavgp` is obtained by time-averaging the magnitude-squared scalogram over all times. The power of the time-averaged wavelet spectrum is normalized to equal the variance of `x`.

`tavgp = timeSpectrum(fb,cfs)` returns the time-averaged wavelet spectrum for the CWT coefficients `cfs`.

Note When using this syntax, the power of the time-averaged wavelet spectrum is normalized to equal the variance of the last signal processed by the filter bank object function `wt`.

`[tavgp,f] = timeSpectrum( ___ )` returns the wavelet center frequencies or center periods for the time-averaged wavelet spectrum. `f` is a column vector or duration array depending on whether the sampling frequency or sampling period is specified in the CWT filter bank, `fb`.

`[ ___ ] = timeSpectrum( ___ ,Name,Value)` specifies additional options using name-value pair arguments. These arguments can be added to any of the previous input syntaxes. For example, `'Normalization','none'` specifies no normalization of the time-averaged wavelet spectrum.

`timeSpectrum( ___ )` with no output arguments plots the time-averaged wavelet power spectrum in the current figure.

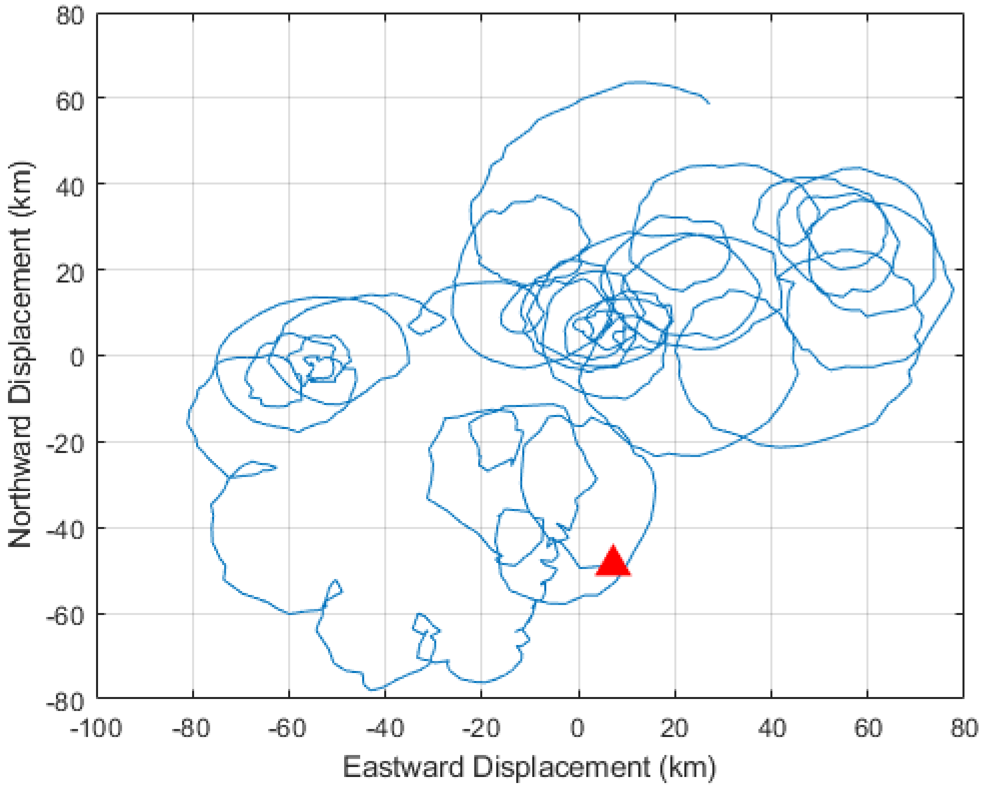
Examples

Time-Averaged Wavelet Spectrum of Oceanic Eddy Data

Load the NPG2006 dataset [1]. The data is the trajectory of a subsurface float trapped in an eddy. Plot the eastward and northward displacement. The triangle marks the initial position.

```
load npg2006
plot(npg2006.cx)
hold on
```

```
grid on
xlabel('Eastward Displacement (km)')
ylabel('Northward Displacement (km)')
plot(npg2006.cx(1),'^','markersize',11,'color','r',...
    'markerfacecolor',[1 0 0 ])
```



Create a CWT filter bank that can be applied to the data. Use the default Morse wavelet. The sampling period for the data is 4 hours.

```
fb = cwtfilterbank('SignalLength',length(npg2006.cx),'SamplingPeriod',hours(4));
```

Obtain the time-averaged wavelet power spectra and the center periods.

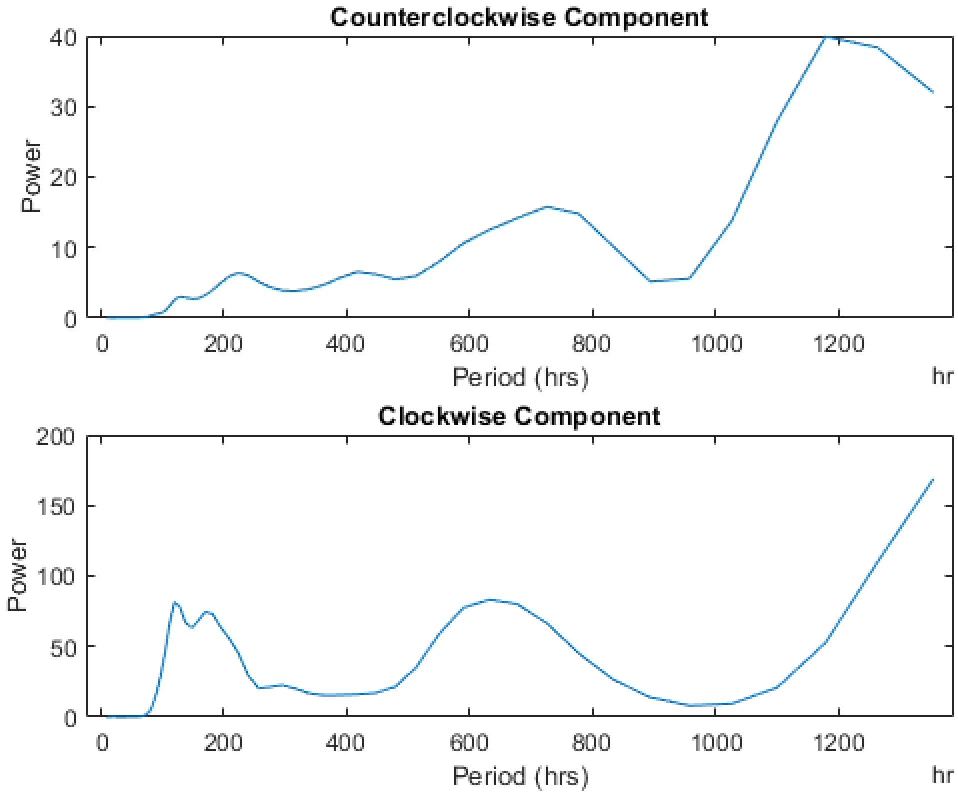
```
[tavgp,centerP] = timeSpectrum(fb,npg2006.cx);
size(tavgp)
```

```
ans = 1×3

    73     1     2
```

The first page is the time-averaged wavelet spectrum for the positive scales (analytic part or counterclockwise component), and the second page is the time-averaged wavelet spectrum for the negative scales (anti-analytic part or clockwise component). Plot both spectra.

```
subplot(2,1,1)
plot(centerP,tavgp(:,1,1))
title('Counterclockwise Component')
```

```
ylabel('Power')
xlabel('Period (hrs)')
subplot(2,1,2)
plot(centerP,tavgp(:,1,2))
title('Clockwise Component')
ylabel('Power')
xlabel('Period (hrs)')
```



If you omit the output arguments and execute `timeSpectrum(fb,npg2006.cx)` on the command line, the scalograms and time-averaged power spectra are plotted in the current figure. Note that the clockwise rotation of the float is captured in the clockwise rotary scalogram and the time-averaged spectrum.

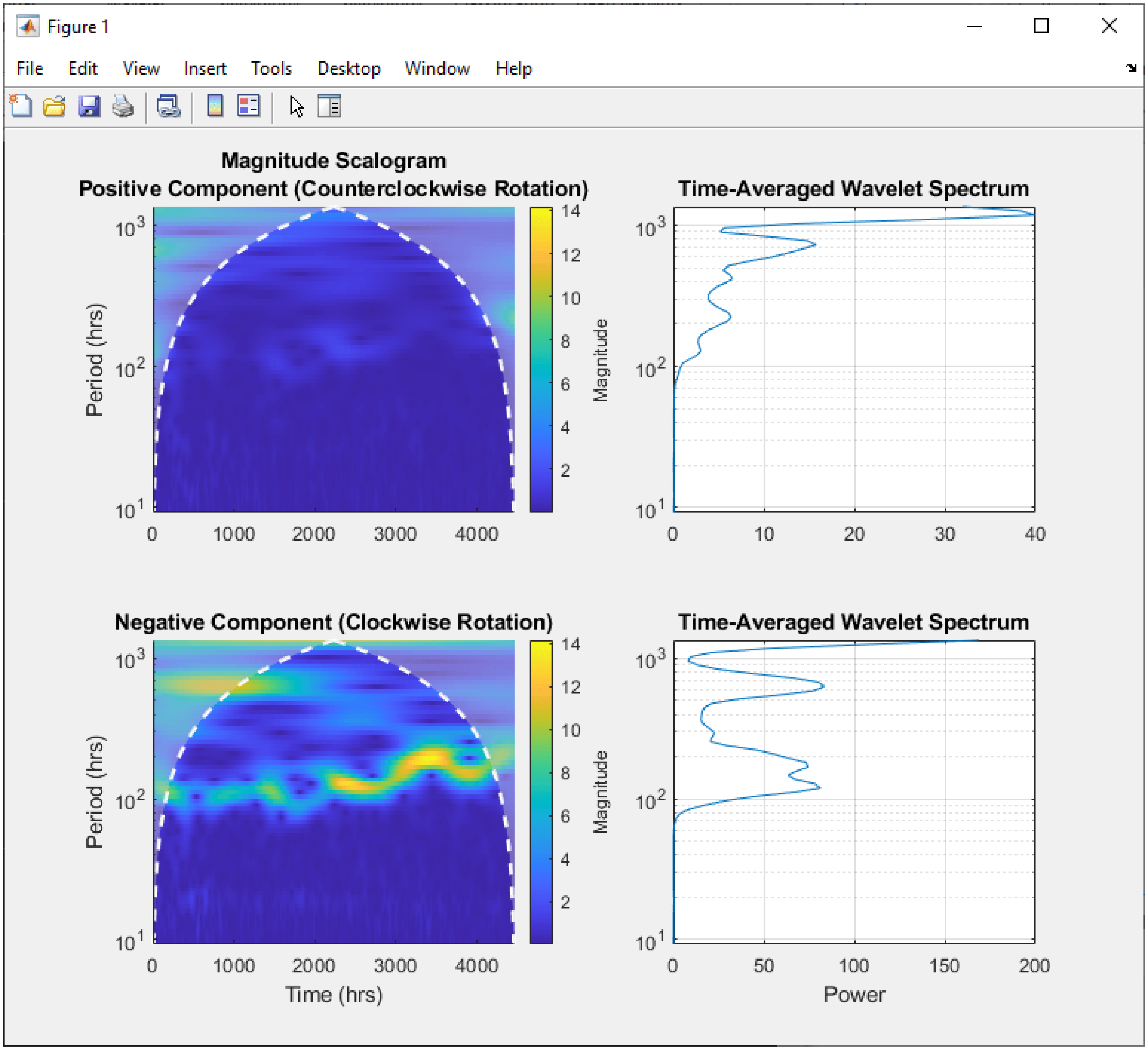
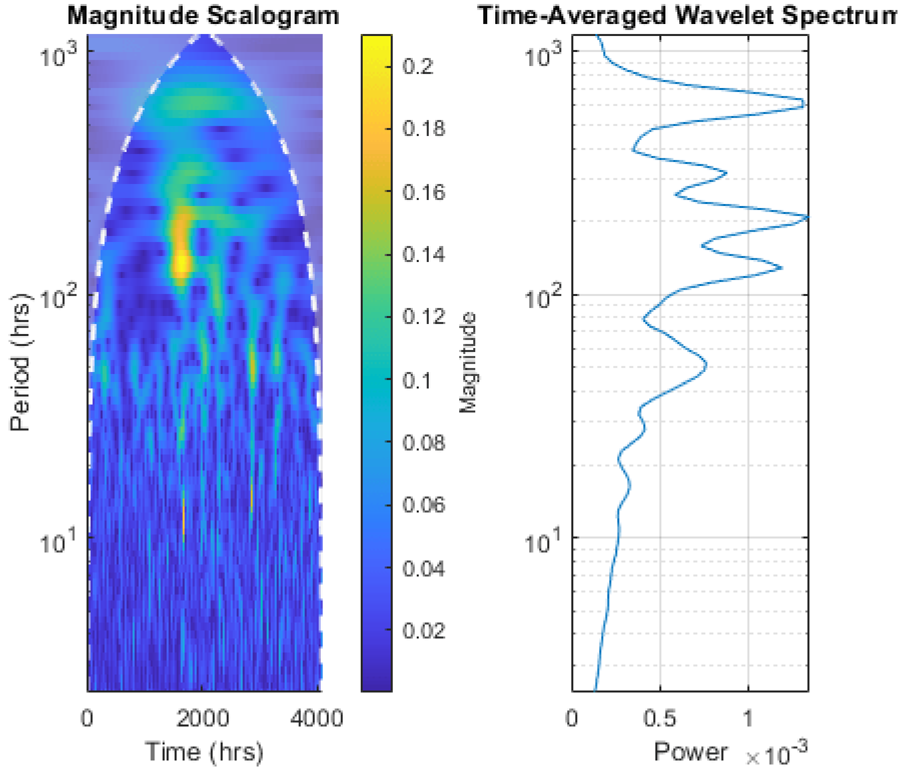


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude Scalogram
Positive Component (Counterclockwise Rotation)
Period (hrs)
10^3
10^2
10^1
0 1000 2000 3000 4000
Magnitude
14 12 10 8 6 4 2
Time-Averaged Wavelet Spectrum
0 10 20 30 40
Negative Component (Clockwise Rotation)
Time (hrs)
Time-Averaged Wavelet Spectrum
0 50 100 150 200
Power

Normalize Time-Averaged Wavelet Spectrum

Load a time series of solar magnetic field magnitudes recorded hourly over the south pole of the sun by the Ulysses spacecraft from 21:00 UT on December 4, 1993 to 12:00 UT on May 24, 1994. See [3] pp. 218–220 for a complete description of this data. Create a CWT filter bank that can be applied to the data. Plot the scalogram and the time-averaged wavelet spectrum.

```
load solarMFmagnitudes
fb = cwtfilterbank('SignalLength',length(sm),'SamplingPeriod',hours(1));
timeSpectrum(fb,sm)
```



Obtain the time-averaged wavelet spectrum of the signal using default values. By default, `timeSpectrum` normalizes the power of the time-averaged wavelet spectrum to equal the variance of the signal. Verify that the sum of the spectrum equals the variance of the signal.

```
tavg = timeSpectrum(fb,sm);
[var(sm) sum(tavg)]
```

```
ans = 1×2

    0.0448    0.0447
```

Obtain the time-averaged wavelet spectrum of the signal, but instead normalize the power as a probability density function. Verify that the sum is equal to 1.

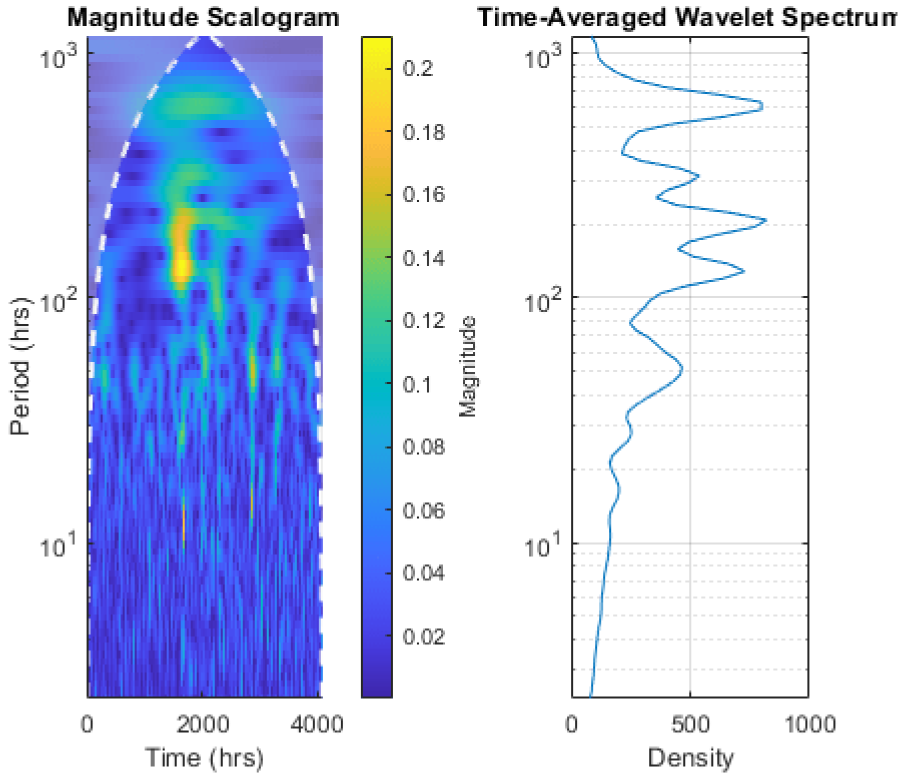
```
tavg = timeSpectrum(fb,sm,'Normalization','pdf');
sum(tavg)
```

```
ans = 1.0000
```

If you set `SpectrumType` to `'density'`, `timeSpectrum` normalizes the weighted integral of the wavelet spectrum according to the value of `Normalization`. The spectrum mimics a probability density function whose integral, numerically evaluated, equals the value specified by `Normalization`.

Plot the scalogram and the time-averaged wavelet spectrum with spectrum type `'density'` and `'pdf'` normalization.

```
figure
timeSpectrum(fb,sm,'SpectrumType','density','Normalization','pdf')
```



To confirm the integral of the spectrum equals 1, first obtain the time-averaged wavelet spectrum with `'density'` spectrum type and `'pdf'` normalization.

```
tavg = timeSpectrum(fb,sm,'SpectrumType','density','Normalization','pdf');
```

By default, the filter bank uses the analytic Morse (3,60) wavelet. Obtain the admissibility constant for the wavelet and numerically integrate the wavelet spectrum using the trapezoidal rule. Keep in mind that the CWT uses L1 normalization. Confirm that the integral equals 1.

```
ga = 3;
tbw = 60;

be = tbw/ga;
anorm = 2*exp(be/ga*(1+(log(ga)-log(be))));
cPsi = anorm^2/(2*ga).*(1/2)^(2*(be/ga)-1)*gamma(2*be/ga);

rawScales = scales(fb);
numInt = 2/cPsi*1/length(sm)*trapz(rawScales(:),tavg./rawScales(:))
```

```
numInt = 1.0000
```

Input Arguments

fb — Continuous wavelet transform filter bank
cwtfilterbank object

Continuous wavelet transform (CWT) filter bank, specified as a cwtfilterbank object.

x — Input data
vector

Input data, specified as a real- or complex-valued vector. The input data x must have at least four samples.

Data Types: single | double
Complex Number Support: Yes

cfs — CWT coefficients
matrix | 3-D array

CWT coefficients, specified as a 2-D matrix or as an *M*-by-*N*-by-2 array. cfs should be the output of the wt object function of the CWT filter bank fb.

Data Types: single | double
Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: timeSpectrum(fb,x,'TimeLimits',[100 500],'Normalization','none') returns the time-averaged wavelet spectrum averaged over the time limits specified in samples without normalizing the spectrum.

Normalization — Normalization
'var' (default) | 'pdf' | 'none'

Normalization of the time-averaged wavelet spectrum, specified as a comma-separated pair consisting of 'Normalization' and one of the following:

- 'var' — Normalize to equal the variance of the time series x. If you provide the cfs input, the timeSpectrum function uses the variance of the last time series processed by the filter bank object function wt.
- 'pdf' — Normalize to equal 1.
- 'none' — No normalization is applied.

SpectrumType — Type of wavelet spectrum
'power' (default) | 'density'

Type of wavelet spectrum to return, specified as a comma-separated pair consisting of 'SpectrumType' and either 'power' or 'density'. If specified as 'power', the averaged sum of the time-averaged wavelet spectrum over all times is normalized according to the value specified in

'`Normalization`'. If specified as `'density'`, the weighted integral of the wavelet spectrum over all times is normalized according to the value specified in '`Normalization`'.

Note With regards to the numerical implementation of the continuous wavelet transform, the integral over scale is performed using L1 normalization. With L1 normalization, if you have equal amplitude oscillatory components in your data at different scales, they will have equal magnitude in the CWT. Using L1 normalization provides a more accurate representation of the signal. For more information, see "L1 Norm for CWT" on page 1-167.

`TimeLimits` — Time limits

`[1 length(x)]` or `[1 size(cfs,2)]` (default) | two-element vector

Time limits over which to average the wavelet spectrum, specified in samples. `TimeLimits` is specified as a comma-separated pair consisting of `'TimeLimits'` and a two-element vector with nondecreasing elements. When you specify the input data as a signal, the elements are between 1 and the length of `x`. When you specify the input data as CWT coefficients, the elements are between 1 and `size(cfs,2)`.

Output Arguments

`tavgp` — Time-averaged wavelet power spectrum

real-valued vector | real-valued 3-D array

Time-averaged wavelet power spectrum, returned as a real-valued vector or real-valued 3-D array. If `x` is real-valued, `tavgp` is an *F*-by-1 vector, where *F* is the number of wavelet center frequencies or center periods in the CWT filter bank `fb`. If `x` is complex-valued, `tavgp` is an *F*-by-1-by-2 array, where the first page is the time-averaged wavelet spectrum for the positive scales (analytic part or counterclockwise component), and the second page is the time-averaged wavelet spectrum for the negative scales (anti-analytic part or clockwise component).

`f` — Center frequencies or center periods

column vector | duration array

Center frequencies or center periods for the time-averaged wavelet spectrum, returned as a column vector or duration array, respectively. If the sampling frequency is specified in `fb`, then the elements of `f` are the center frequencies ordered from high to low. If the sampling period is specified in `fb`, then the elements of `f` are the center periods.

References

[1] Lilly, J. M., and J.-C. Gascard. "Wavelet Ridge Diagnosis of Time-Varying Elliptical Signals with Application to an Oceanic Eddy." *Nonlinear Processes in Geophysics* 13, no. 5 (September 14, 2006): 467–83. https://doi.org/10.5194/npg-13-467-2006.

[2] Torrence, Christopher, and Gilbert P. Compo. "A Practical Guide to Wavelet Analysis." *Bulletin of the American Meteorological Society* 79, no. 1 (January 1, 1998): 61–78. https://doi.org/10.1175/1520-0477(1998)079<0061:APGTWA>2.0.CO;2.

[3] Percival, Donald B., and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge ; New York: Cambridge University Press, 2000.

[4] Lilly, J.M., and S.C. Olhede. “Higher-Order Properties of Analytic Wavelets.” *IEEE Transactions on Signal Processing* 57, no. 1 (January 2009): 146–60. https://doi.org/10.1109/TSP.2008.2007607.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`cwtfilterbank` | `scaleSpectrum`

Introduced in R2020b

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.

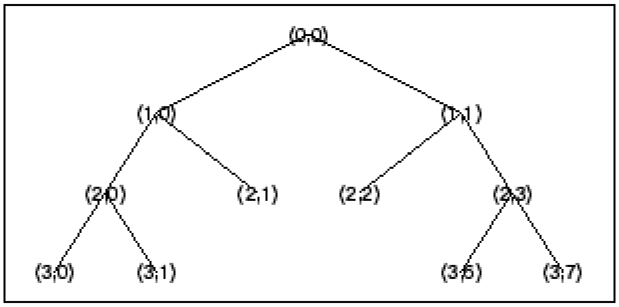
`N = tnodes(`*T*`,'deppos')` returns a matrix `N`, which contains the depths and positions of terminal nodes.

`N(i,1)` is the depth of the `i`-th terminal node. `N(i,2)` is the position of the `i`-th terminal node.

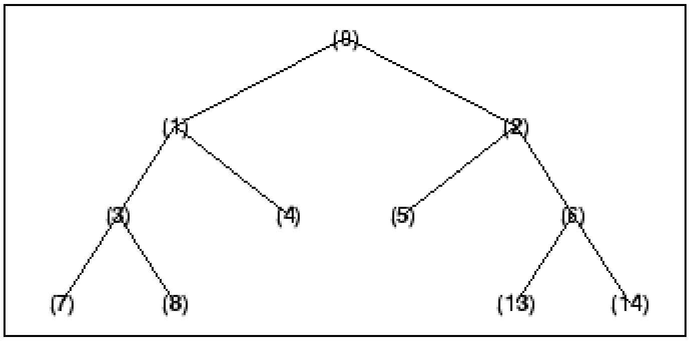
For `[N,K] = tnodes(`*T*`)` or `[N,K] = tnodes(`*T*`,'deppos'), M = N(K)` are the indices reordered as in tree *T*, from left to right.

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

tqwt

Tunable Q-factor wavelet transform

Syntax

```
wt = tqwt(x)
wt = tqwt(x,Name=Value)
[wt,info] = tqwt( ___ )
```

Description

`wt = tqwt(x)` returns the tunable Q-factor wavelet transform (TQWT) of `x`.

- The TQWT is computed to the maximum decomposition level with a quality factor of 1. For more information, see “TQWT Decomposition Levels” on page 1-1335.
- As implemented, the `tqwt` function uses a redundancy of 3. For more information, see “Redundancy” on page 1-1335.

`wt = tqwt(x,Name=Value)` specifies one or more additional name-value arguments. For example, `wt = tqwt(x,QualityFactor=2)` specifies a quality factor of 2.

`[wt,info] = tqwt( ___ )` returns the structure array, `info`, with information about the tunable Q-factor wavelet transform.

Examples

Tunable Q-factor Wavelet Transform of Multisignal

Load a multichannel EEG signal. The signal has 23 channels.

```
load Espiga3
size(Espiga3,2)
```

```
ans = 23
```

Obtain the tunable Q-factor wavelet transform of the multisignal to the maximum level using the default quality factor of 1.

```
wt = tqwt(Espiga3);
numel(wt)
```

```
ans = 12
```

For $1 \leq i \leq$ `numel(wt)`-1, the *i*th element of `wt` contains the wavelet transform coefficients for the *i*th subband. The last element of `wt` contains the lowpass subband coefficients. Confirm the number of columns of any element of `wt` is equal to the number of channels.

```
k = 7;
size(wt{k},2)
```

```
ans = 23
```

Reconstruct the multisignal and demonstrate perfect reconstruction.

```
xrec = itqwt(wt,size(Espiga3,1));
max(abs(xrec(:)-Espiga3(:)))
```

```
ans = 4.9738e-13
```

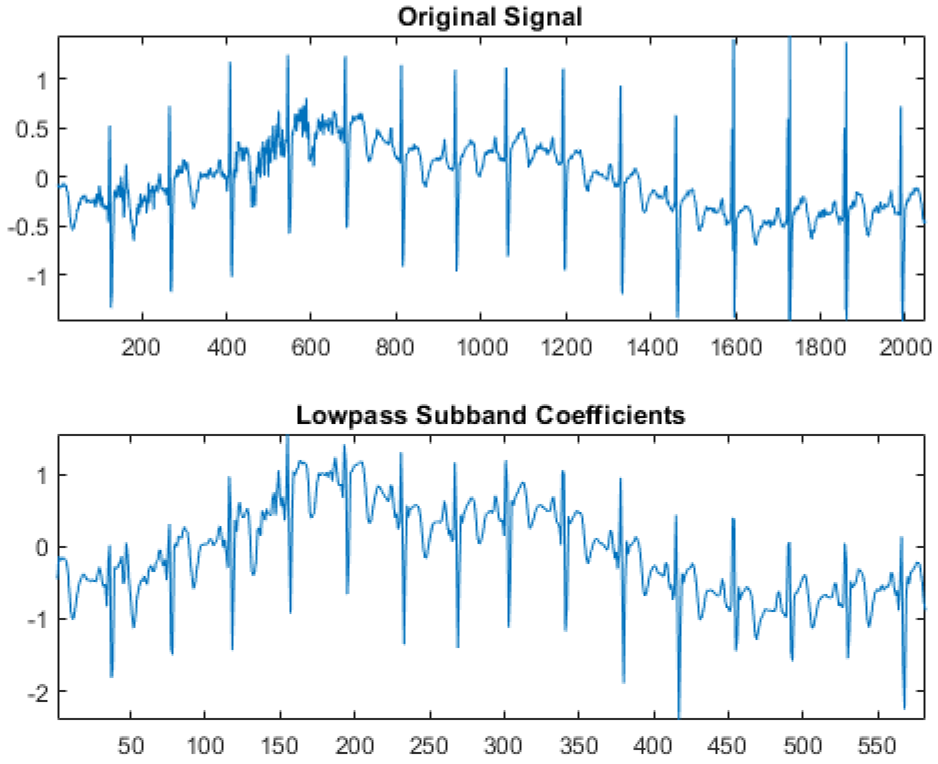
Inspect TQWT Information Structure

Load an ECG signal. Obtain the TQWT of the signal down to level 5 with a quality factor of 2. Also obtain information of the TQWT.

```
load wecg
lvl = 5;
qf = 2;
[wt,info] = tqwt(wecg,Level=lvl,QualityFactor=qf);
```

Plot the original signal and compare with the lowpass subband coefficients.

```
subplot(2,1,1)
plot(wecg)
title("Original Signal")
axis tight
subplot(2,1,2)
plot(wt{end})
title("Lowpass Subband Coefficients")
axis tight
```



Inspect the TQWT information structure. For each subband, confirm that the ratio of the center frequency to the approximate bandwidth equals the quality factor.

```
info
```

```
info = struct with fields:
    CenterFrequencies: [0.3333 0.2593 0.2016 0.1568 0.1220]
           Bandwidths: [0.1667 0.1296 0.1008 0.0784 0.0610]
                Level: 5
                Alpha: 0.7778
                 Beta: 0.6667
```

```
info.CenterFrequencies./info.Bandwidths
```

```
ans = 1×5

    2.0000    2.0000    2.0000    2.0000    2.0000
```

As implemented, the `tqwt` function uses the redundancy $r = 3$. Confirm the highpass and lowpass scaling factors, `info.Beta` and `info.Alpha` respectively, satisfy the relation $r = \frac{\beta}{1-\alpha}$.

```
info.Beta/(1-info.Alpha)
```

```
ans = 3
```

Input Arguments

x — Input signal

vector | matrix | 3-D array

Input signal, specified as a single- or double-precision vector, matrix, or 3-D array. If x is a matrix or 3-D array, the TQWT is computed along the columns of x. For 3-D arrays, tqwt interprets the first dimension as time, the second dimension as channels, and the third dimension as a batch.

The TQWT is defined for even-length signals. If the number of samples in x is odd, the last sample of x is repeated to obtain an even-length signal.

Data Types: single | double
Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: wt = tqwt(x,Level=3,QualityFactor=2)

Level — Decomposition level

positive integer

Decomposition level of the TQWT, specified as a positive integer between 1 and the maximum level. The maximum level depends on the signal length and quality factor. For more information, see “TQWT Decomposition Levels” on page 1-1335.

Example: wt = tqwt(x,Level=3) specifies a decomposition level of 3.

Data Types: single | double

QualityFactor — Quality factor

1 (default) | positive scalar

Quality factor, specified as a real-valued scalar greater than or equal to 1. The quality factor is the ratio of the center frequency to the bandwidth of the filters. If unspecified, the quality factor defaults to 1.

Example: wt = tqwt(x,QualityFactor=1.5) specifies a quality factor of 1.5.

Data Types: single | double

Output Arguments

wt — Tunable Q-factor wavelet transform

cell array

Tunable Q-factor wavelet transform, returned as a cell array. wt is a cell array with length equal to the maximum level of the TQWT plus one. The *i*th element of wt contains the TQWT coefficients for the *i*th subband. The subbands are ordered by decreasing center frequency. The final element of wt contains the lowpass subband coefficients. The wavelet coefficients in wt match x in data type and complexity.

- If `x` is a row vector, each element of `wt` is a column vector containing the TQWT coefficients.
- If `x` is a matrix or 3-D array, the column and page sizes of each element of `wt` match the column and page sizes of `x`.

Data Types: `single` | `double`

`info` — Transform information
structure array

Transform information, returned as a structure array. `info` has five fields:

- `CenterFrequencies` — The normalized center frequencies (cycles/sample) of the wavelet subbands in the TQWT of `x`. To convert the frequencies to hertz, multiply `CenterFrequencies` by the sample rate.
- `Bandwidths` — The approximate bandwidths of the wavelet subbands in normalized frequency (cycles/sample). To convert the bandwidths to hertz, multiply `Bandwidths` by the sample rate.
- `Level` — Level of the TQWT. Note that `info.Level` may be different from your specified level if you specify a level greater than the maximum supported level for your signal length and quality factor.
- `Beta` — Highpass scaling factor. The highpass scaling factor is computed from the quality factor as `2/(QualityFactor+1)`. Accordingly, `0 < Beta ≤ 1`.
- `Alpha` — Lowpass scaling factor. The lowpass scaling factor is computed from the highpass scaling factor as `1-Beta/3`. Accordingly, `2/3 ≤ Alpha < 1`.

Data Types: `struct`

More About

TQWT Decomposition Levels

The TQWT minimum and maximum decomposition levels depend on the signal length, N, and quality factor, Q. In the description that follows, the signal length, N, is one sample larger than the input length for odd-length signals.

The maximum decomposition level is

$$\left\lfloor \log\left(\frac{N}{4Q+4}\right)/\log\left(\frac{3Q+3}{3Q+1}\right)\right\rfloor,$$

where the $\lfloor\ \rfloor$ symbols denote the floor function.

The minimum level also depends on the signal length and quality factor. The logarithm of N, $\log(N)$, must satisfy the following inequality:

$$\log(N) \geq \log(4Q+4) - \log(3Q+1) + \log(3Q+3)\,.$$

If $\log(N) < \log(4Q+4) - \log(3Q+1) + \log(3Q+3)$, the maximum level is less than 1 and `tqwt` throws an error.

Redundancy

The TQWT algorithm depends on scaling in the frequency domain:

- lowpass scaling — frequency-domain scaling by α that preserves low-frequency content
- highpass scaling — frequency-domain scaling by β that preserves high-frequency content

The redundancy is defined to be

$$r = \frac{\beta}{1 - \alpha}.$$

For more information, see “Tunable Q-factor Wavelet Transform”.

References

[1] Selesnick, Ivan W. “Wavelet Transform With Tunable Q-Factor.” *IEEE Transactions on Signal Processing* 59, no. 8 (August 2011): 3560–75. https://doi.org/10.1109/TSP.2011.2143711.

Extended Capabilities

C/C++ Code Generation

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

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The signal `x` is the only supported input argument.
- The TQWT array `wt` is the only supported output argument.

See Also

Apps

Signal Multiresolution Analyzer

Functions

`itqwt` | `tqwtmra`

Topics

“Time-Frequency Gallery”
“Tunable Q-factor Wavelet Transform”

Introduced in R2021b

tqwtmra

Tunable Q-factor multiresolution analysis

Syntax

```
mra = tqwtmra(wt,n)
mra = tqwtmra(wt,n,QualityFactor=qf)
tqwtmra( ___ )
```

Description

`mra = tqwtmra(wt,n)` returns the tunable Q-factor wavelet multiresolution analysis (MRA) for the TQWT analysis, `wt`, obtained with the default quality factor of 1.

`mra = tqwtmra(wt,n,QualityFactor=qf)` uses the quality factor `qf` in obtaining the tunable Q-factor MRA. `qf` must match the value used in obtaining `wt` from `tqwt`.

`tqwtmra( ___ )` with no output arguments plots the tunable Q-factor wavelet MRA in a new figure. For complex-valued data, the real part is plotted in the first color in the MATLAB color order matrix and the imaginary part is plotted in the second color. This syntax does not support multidimensional MRAs.

Examples

Perform Tunable Q-factor Multiresolution Analysis

Load an ECG signal. Obtain the TQWT of the signal down to level 6 with a quality factor of 2.

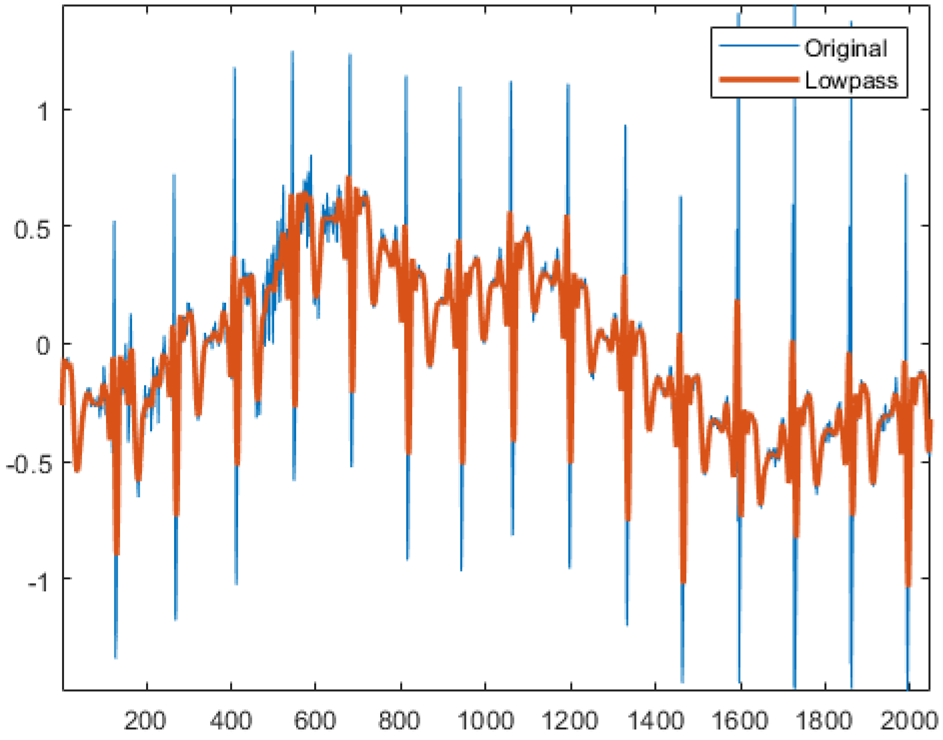
```
load wecg
wt = tqwt(wecg,QualityFactor=2,Level=6);
```

Obtain the tunable Q-factor MRA of the signal.

```
mra = tqwtmra(wt,length(wecg),QualityFactor=2);
```

Plot the original signal and the lowpass subband.

```
plot(wecg)
hold on
plot(mra(end,:),linewidth=2)
hold off
axis tight
legend(["Original","Lowpass"])
```



Confirm the sum along the rows of the MRA equals the original signal.

```
mraSum = sum(mra,1);
max(abs(mraSum(:)-wecg(:)))
```

```
ans = 7.2164e-16
```

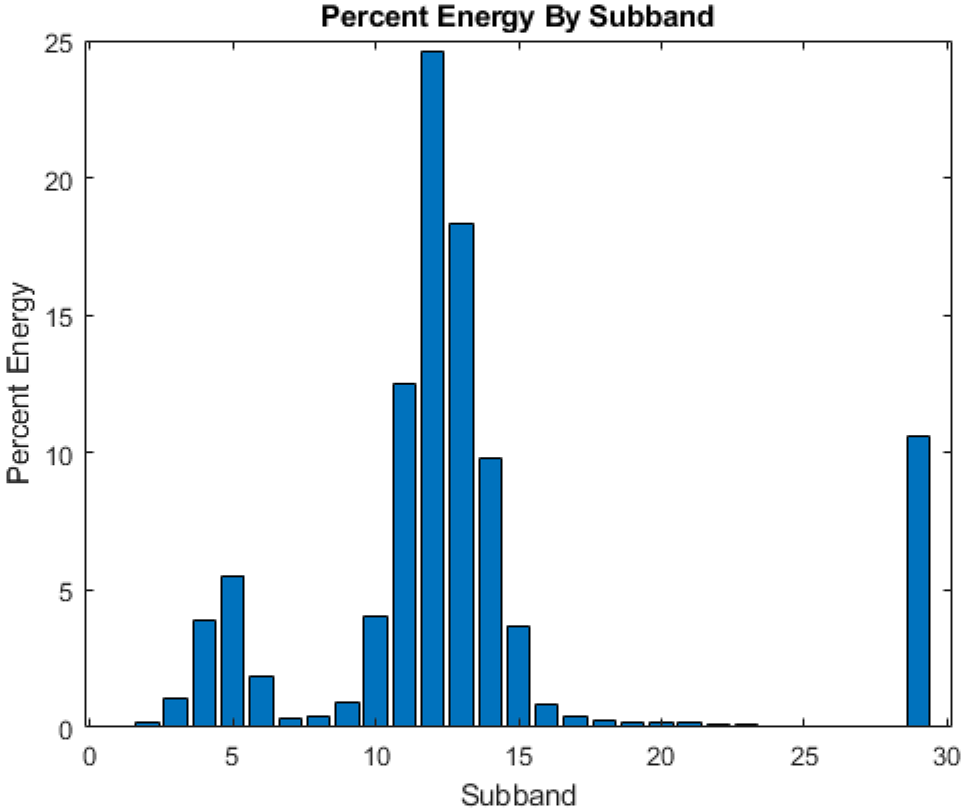
Identify Tunable Q-factor MRA Subbands by Energy

Load the Kobe earthquake data. Obtain the tunable Q-factor wavelet transform of the data using a quality factor of 3.

```
load kobe
qf = 3;
wt = tqwt(kobe,QualityFactor=qf);
```

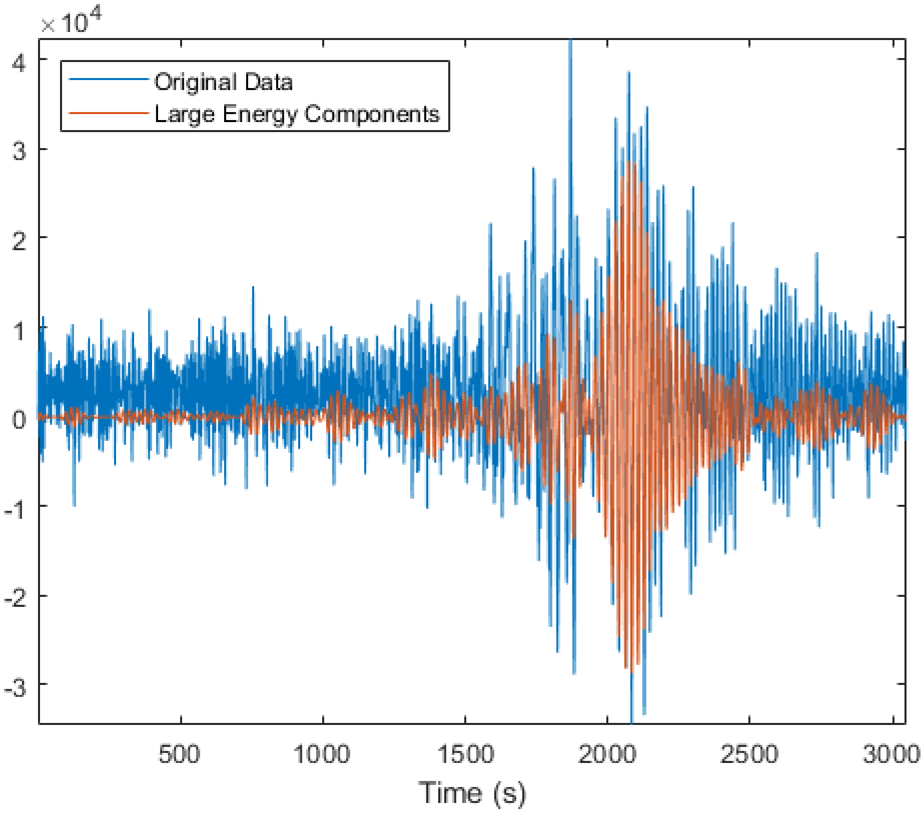
Identify the subbands that contain at least 15% of the total energy. Note that the last element of `wt` contains the lowpass subband coefficients.

```
EnergyBySubband = cellfun(@(x)norm(x,2)^2,wt)./norm(kobe,2)^2*100;
idx15 = EnergyBySubband >= 15;
bar(EnergyBySubband)
title("Percent Energy By Subband")
xlabel("Subband")
ylabel("Percent Energy")
```



Obtain a multiresolution analysis and sum those MRA components corresponding to previously identified subbands.

```
mra = tqwtmra(wt,numel(kobe),QualityFactor=qf);
ts = sum(mra(idx15,:));
plot([kobe ts'])
axis tight
legend("Original Data","Large Energy Components",...
        Location="NorthWest")
xlabel("Time (s)")
```



Input Arguments

wt — Tunable Q-factor wavelet transform
cell array

Tunable Q-factor wavelet transform, specified as a cell array. The elements of `wt` contain the wavelet subband and lowpass coefficients. `wt` is expected to be the output of `tqwt`.

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

n — Original signal length
positive integer

Original signal length in samples, specified as a positive integer. If the original signal length `n` is odd, `n` is extended to `n+1` to obtain the MRA and the final sample is removed before returning the MRA.

Data Types: `single` | `double`

qf — Quality factor
`1` (default) | positive scalar

Quality factor, specified as a real-valued scalar greater than or equal to 1. If unspecified, the quality factor defaults to 1.

Example: `wt = tqwtmra(qt,2024,QualityFactor=1.5)` specifies a quality factor of 1.5.

Data Types: `single` | `double`

Output Arguments

`mra` — Multiresolution analysis

array

Multiresolution analysis, returned as an array. `mra` is an *Ns*-by-*N*-by-*C*-by-*B* array where *Ns* denotes number of subbands in the tunable Q-factor wavelet transform ordered by decreasing center frequency, *N* is the number of signal samples in time, *C* is the number of channels, and *B* is the batch size.

Data Types: `single` | `double`

References

[1] Selesnick, Ivan W. “Wavelet Transform With Tunable Q-Factor.” *IEEE Transactions on Signal Processing* 59, no. 8 (August 2011): 3560–75. https://doi.org/10.1109/TSP.2011.2143711.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Plotting is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

- The TQWT array `wt` is the only supported input argument.

See Also

Apps

Signal Multiresolution Analyzer

Functions

`tqwt` | `itqwt`

Topics

“Time-Frequency Gallery”
“Tunable Q-factor Wavelet Transform”

Introduced in R2021b

treedpth

Tree depth

Syntax

```
D = treedpth(T)
```

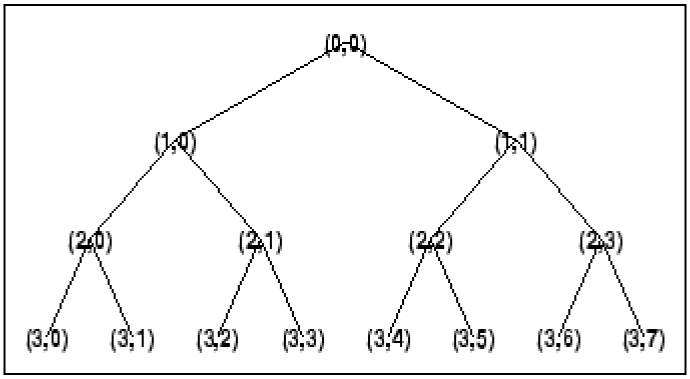
Description

`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)
```

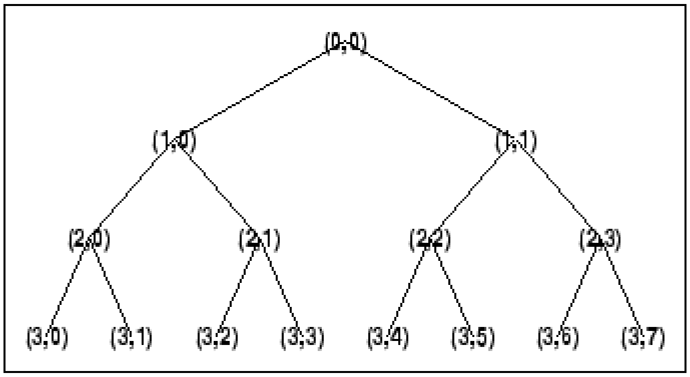
Description

`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

uminus

Unary minus for Laurent polynomial or Laurent matrix

Syntax

```
Q = uminus(P)
Q = -P
```

Description

`Q = uminus(P)` negates the Laurent polynomial or the Laurent matrix specified by `P`. If `P` is a Laurent matrix, `uminus` negates the matrix elements.

Note The `laurentPolynomial` and `laurentMatrix` objects have their own versions of `uminus`. The input data type determines which version is executed.

`Q = -P` is equivalent to `Q = uminus(P)`.

Examples

Unary Minus of Laurent Polynomial

Create a Laurent polynomial

```
a = laurentPolynomial(Coefficients=[-2 6 -7 -2 1],MaxOrder=3);
```

Confirm the sum of $a(z)$ and its unary minus is 0.

```
au = uminus(a);
a+au
```

```
ans = 
  laurentPolynomial with properties:

    Coefficients: 0
        MaxOrder: 0
```

Unary Negative of Laurent Matrix

Create the Laurent polynomials

- $a(z) = z + 1$
- $b(z) = z^2 + z + z^{-1}$
- $c(z) = z$

- $d(z) = z^2 + z^{-1}$

```
lpA = laurentPolynomial(Coefficients=[1 1],MaxOrder=1);
lpB = laurentPolynomial(Coefficients=[1 1 0 1],MaxOrder=2);
lpC = laurentPolynomial(Coefficients=[1],MaxOrder=1);
lpD = laurentPolynomial(Coefficients=[1 0 0 1],MaxOrder=2);
```

Create the matrix `lmat` = $\begin{bmatrix} a(z) & b(z) \\ c(z) & d(z) \end{bmatrix}$.

```
lmat = laurentMatrix(Elements={lpA,lpB;lpC,lpD});
```

Confirm the sum of `lmat` and its unary negative is 0.

```
lmatu = uminus(lmat);
xmat = lmat+lmatu;
dispMat(xmat)
```

```
| 0.00e+00   0.00e+00 |
|                     |
| 0.00e+00   0.00e+00 |
```

Input Arguments

P — Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Laurent polynomial or Laurent matrix, specified as a `laurentPolynomial` object or a `laurentMatrix` object, respectively.

Output Arguments

Q — Negated Laurent polynomial or Laurent matrix
`laurentPolynomial` object | `laurentMatrix` object

Negated Laurent polynomial or Laurent matrix, returned as a `laurentPolynomial` object or a `laurentMatrix` object. If `P` is a Laurent polynomial, the coefficients are negated.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`reflect`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

upcoef

Direct reconstruction from 1-D wavelet coefficients

Syntax

```
y = upcoef(o,x,wname)
y = upcoef(o,x,LoR,HiR)
y = upcoef(o,x,wname,n)
y = upcoef(o,x,LoR,HiR,n)
y = upcoef(o,x,wname,n,l)
y = upcoef(o,x,LoR,HiR,n,l)
```

Description

`upcoef` is a one-dimensional wavelet analysis function.

`y = upcoef(o,x,wname)` returns the 1-step reconstructed coefficients of type `o` of the vector `x` using the wavelet specified by `wname`.

`y = upcoef(o,x,LoR,HiR)` uses the specified lowpass and highpass reconstruction filters `LoR` and `HiR`, respectively.

`y = upcoef(o,x,wname,n)` returns the `n`-step reconstructed coefficients.

`y = upcoef(o,x,LoR,HiR,n)` uses the specified lowpass and highpass reconstruction filters `LoR` and `HiR`, respectively.

`y = upcoef(o,x,wname,n,l)` returns the length-`l` central portion of the `n`-step reconstruction.

`y = upcoef(o,x,LoR,HiR,n,l)` uses the specified lowpass and highpass reconstruction filters `LoR` and `HiR`, respectively.

Examples

Reconstruct Wavelet Coefficients

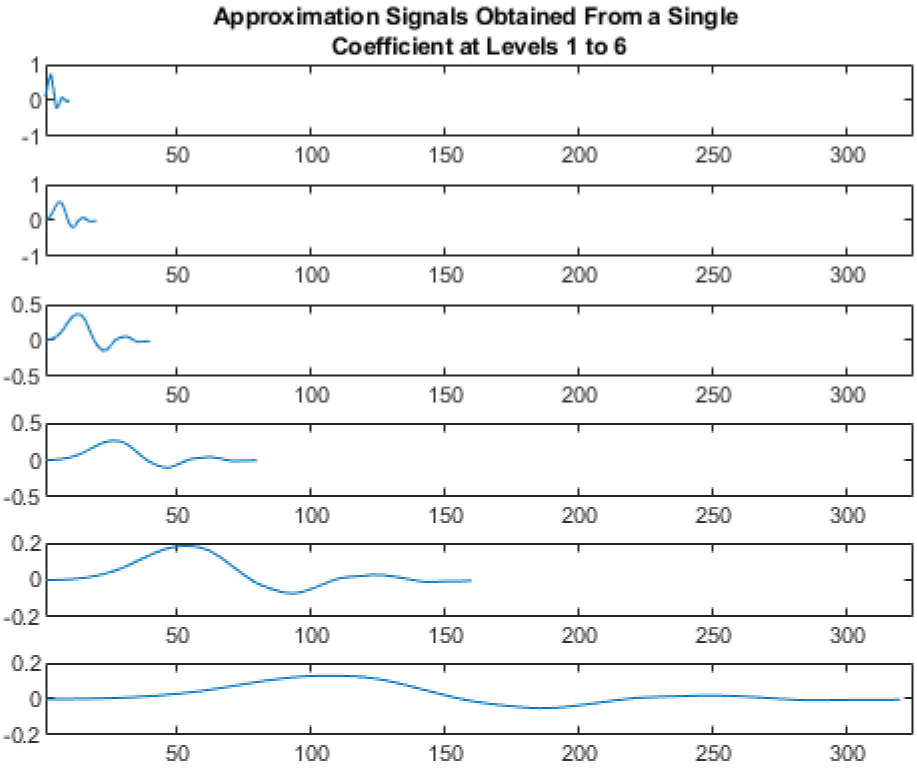
Save the current extension mode, then set the extension mode to zero-padding.

```
origMode = dwtmode('status','nodisp');
dwtmode('zpd','nodisp')
```

Reconstruct approximation signals, obtained from a single coefficient at levels 1 to 6. Use the `db6` wavelet.

```
cfs = 1;
essup = 10; % Essential support of the scaling filter db6.

for i=1:6
    % Reconstruct at the top level an approximation
    % which is equal to zero except at level i where only
```

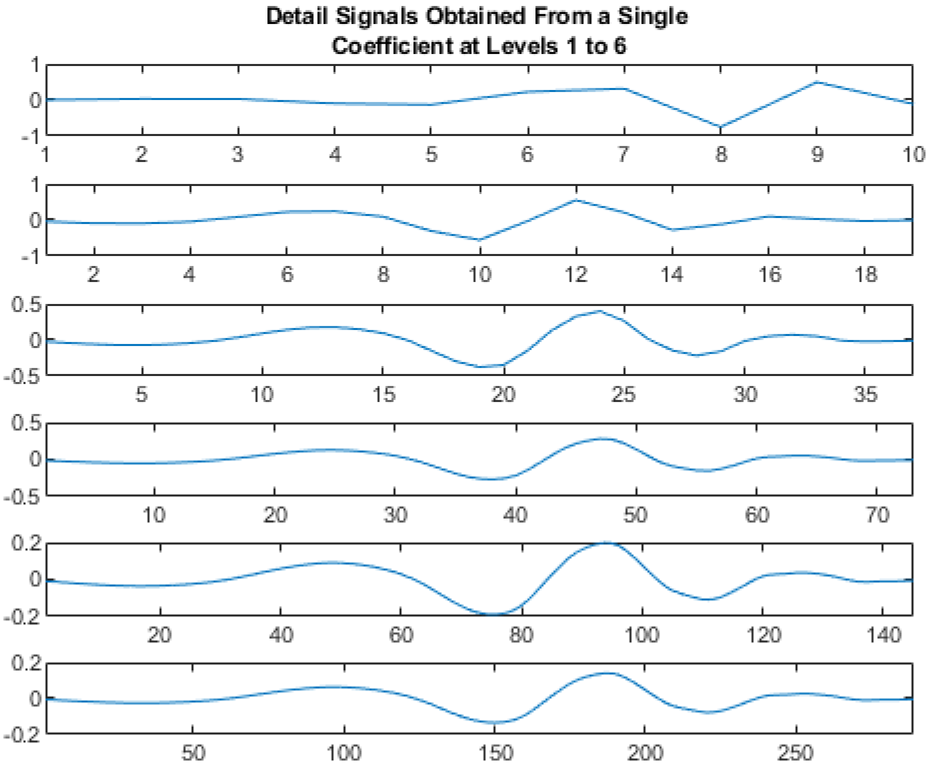
```
        % one coefficient is equal to 1.
        rec = upcoef("a",cfs,"db6",i);

        % essup is the essential support of the
        % reconstructed signal.
        % rec(j) is very small when j is ≥ essup.
        subplot(6,1,i)
        plot(rec(1:essup))
        xlim([1 325])
        if i<3
            ylim([-1 1])
        elseif i<5
            ylim([-0.5 0.5])
        else
            ylim([-0.2 0.2])
        end
        essup = essup*2;
    end
    subplot(6,1,1)
    title(["Approximation Signals Obtained From a Single " ...
        "Coefficient at Levels 1 to 6"])
```



The same can be done for details. Reconstruct details signals, obtained from a single coefficient at levels 1 to 6. Use the db6 wavelet.

```
cfs = [1];
mi = 12; ma = 30;   % Essential support of
                    % the wavelet filter db6.
```

```
rec = upcoef("d",cfs,"db6",1);
figure
subplot(6,1,1)
plot(rec(3:12))
axis tight
ylim([-1 1])
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)))
    axis tight
    if i<3
        ylim([-1 1])
    elseif i<5
        ylim([-0.5 0.5])
    else
        ylim([-0.2 0.2])
    end
end
subplot(6,1,1)
title(["Detail Signals Obtained From a Single " ...
    "Coefficient at Levels 1 to 6"])
```



Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

o — Type of reconstructed coefficients

"a" | "d"

Type of reconstructed coefficients, specified as "a" or "d", for approximation or details coefficients, respectively.

Data Types: string

x — Signal

vector

Signal, specified as a vector.

Data Types: double

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. The wavelet must be recognized by wavemngr. See wfilters for the wavelets available in each family.

Data Types: char | string

n — Number of reconstruction steps

1 (default) | positive integer

Number of reconstruction steps, specified as a positive integer.

Data Types: double

l — Length of central portion

0 (default) | nonnegative integer

Length of central portion of reconstruction to return, specified as a nonnegative integer. If l = 0, upcoef returns the entire reconstruction.

Data Types: double

LoR,HiR — Wavelet reconstruction filters

even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter. The lengths of LoR and HiR must be equal. See wfilters for additional information.

Data Types: double

Output Arguments

y — Reconstructed coefficients

vector

Reconstruction coefficients of `x`, returned as a vector.

Data Types: `double`

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 or string scalar specifying 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)
[NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R)
```

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.

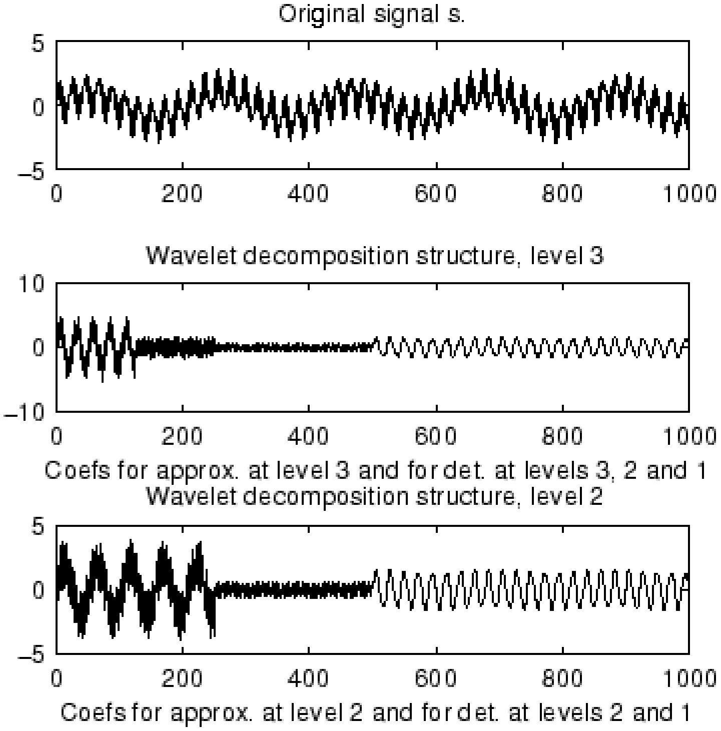
wname is a character vector or string scalar specifying the wavelet, C is the original wavelet decomposition vector, and L the corresponding bookkeeping vector (for detailed storage information, see wavedec).

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 or string scalar specifying the wavelet, `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

vertcat

Vertical concatenation of Laurent polynomials

Syntax

```
V = vertcat(P1,…,PN)
```

Description

`V = vertcat(P1,…,PN)` returns the vertical concatenation of the Laurent polynomials `P1,…,PN`.

Examples

Laurent Polynomial Concatenation

Create two Laurent polynomials:

- $a(z) = z - 1$
- $b(z) = -2z^3 + 6z^2 - 7z + 2$

```
a = laurentPolynomial(Coefficients=[1 -1],MaxOrder=1);
b = laurentPolynomial(Coefficients=[-2 6 -7 2],MaxOrder=3);
```

Obtain the vertical and horizontal concatenations of $a(z)$ and $b(z)$.

```
v = vertcat(a,b)
```

```
v=2×1 cell array
    {1x1 laurentPolynomial}
    {1x1 laurentPolynomial}
```

```
h = horzcat(a,b)
```

```
h=1×2 cell array
    {1x1 laurentPolynomial}    {1x1 laurentPolynomial}
```

Input Arguments

`P1,…,PN` — Input polynomials
`laurentPolynomial` objects

Input polynomials, specified as `laurentPolynomial` objects.

Example: `H = vertcat(P1,P2,P3)` returns the vertical concatenation of the three Laurent polynomials `P1`, `P2`, and `P3`.

Output Arguments

V — Vertical cell array
cell array

Vertical cell array of Laurent polynomials. V is an *N*-by-1 cell array, where *N* is the number of Laurent polynomials.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`horzcat`

Objects
`laurentMatrix` | `laurentPolynomial`

Introduced in R2021b

vmd

Variational mode decomposition

Syntax

```
imf = vmd(x)
[imf,residual] = vmd(x)
[imf,residual,info] = vmd(x)
```

```
[ ___ ] = vmd(x,Name,Value)
```

```
vmd( ___ )
```

Description

`imf = vmd(x)` returns the variational mode decomposition of `x`. Use `vmd` to decompose and simplify complicated signals into a finite number of intrinsic mode functions (IMFs) required to perform Hilbert spectral analysis.

`[imf,residual] = vmd(x)` also returns the residual signal `residual` corresponding to the variational mode decomposition of `x`.

`[imf,residual,info] = vmd(x)` returns additional information `info` on IMFs and the residual signal for diagnostic purposes.

`[ ___ ] = vmd(x,Name,Value)` performs the variational mode decomposition with additional options specified by one or more `Name,Value` pair arguments.

`vmd( ___ )` plots the original signal, IMFs, and the residual signal as subplots in the same figure.

Examples

Variational Mode Decomposition of Dial Tone Signal

Create a signal, sampled at 4 kHz, that resembles dialing all the keys of a digital telephone. Save the signal as a MATLAB® timetable.

```
fs = 4e3;
t = 0:1/fs:0.5-1/fs;

ver = [697 770 852 941];
hor = [1209 1336 1477];

tones = [];

for k = 1:length(ver)
    for l = 1:length(hor)
        tone = sum(sin(2*pi*[ver(k);hor(l)].*t))';
        tones = [tones;tone;zeros(size(tone))];
    end
```

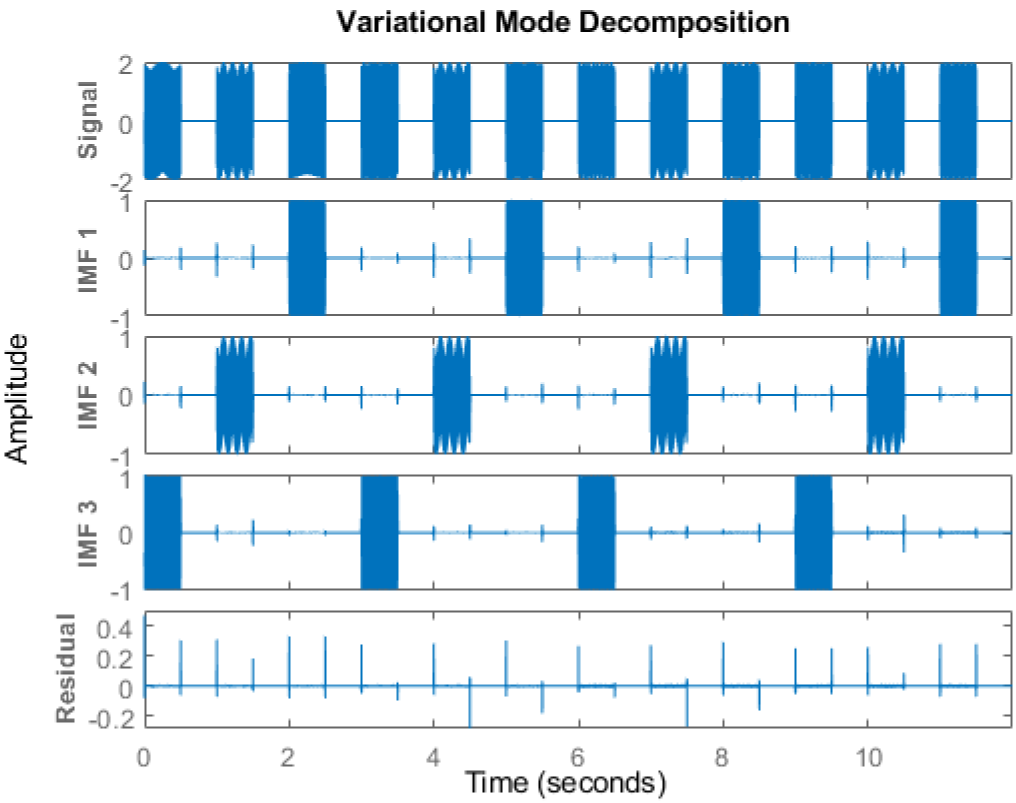
```
end
```

```
% To hear, type soundsc(tones,fs)
```

```
S = timetable(tones,'SampleRate',fs);
```

Plot the variational mode decomposition of the timetable.

```
vmd(S)
```



VMD of Multicomponent Signal

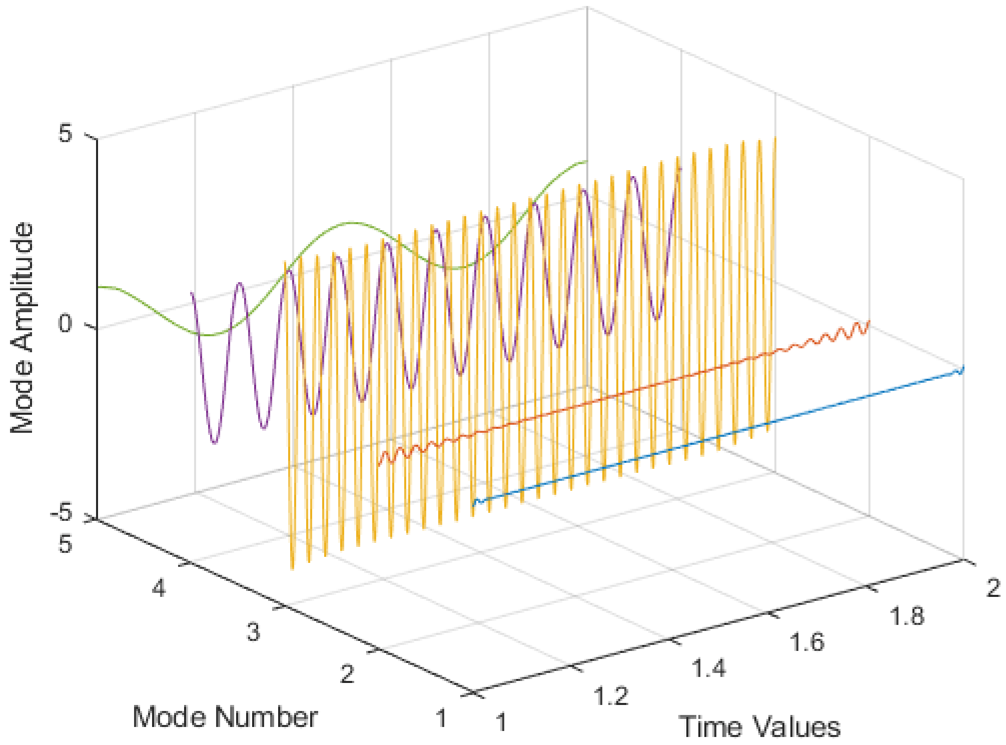
Generate a multicomponent signal consisting of three sinusoids of frequencies 2 Hz, 10 Hz, and 30 Hz. The sinusoids are sampled at 1 kHz for 2 seconds. Embed the signal in white Gaussian noise of variance 0.01^2.

```
fs = 1e3;
t = 1:1/fs:2-1/fs;
x = cos(2*pi*2*t) + 2*cos(2*pi*10*t) + 4*cos(2*pi*30*t) + 0.01*randn(1,length(t));
```

Compute the IMFs of the noisy signal and visualize them in a 3-D plot.

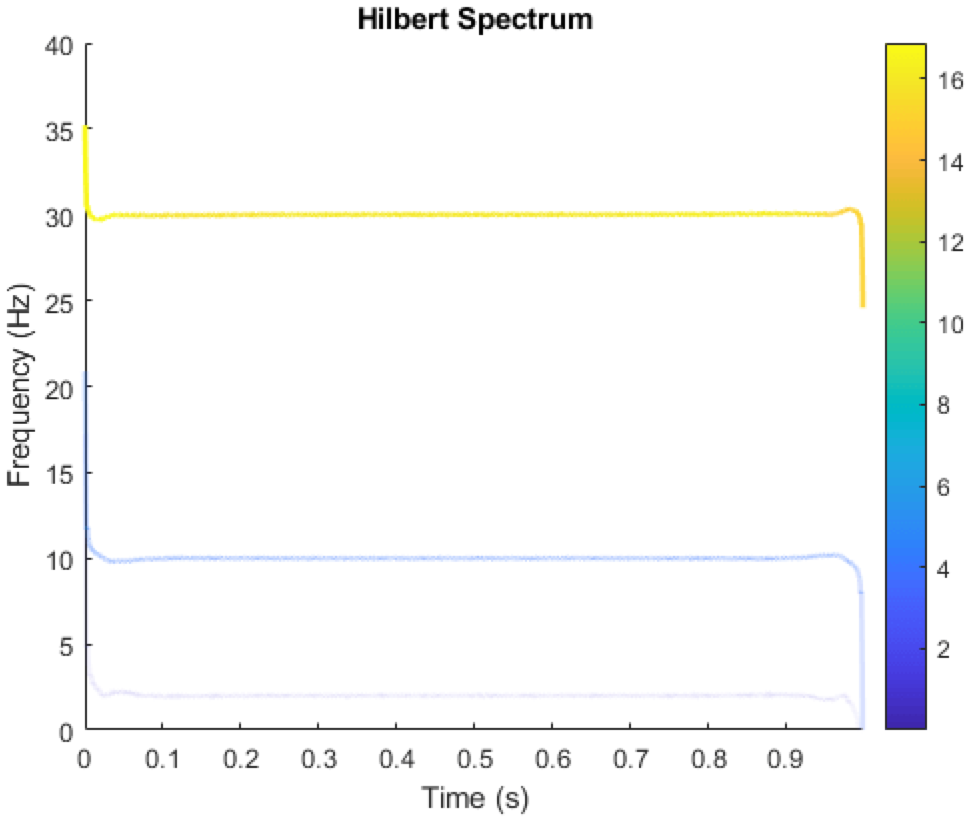
```
imf = vmd(x);
[p,q] = ndgrid(t,1:size(imf,2));
plot3(p,q,imf)
```

```
grid on
xlabel('Time Values')
ylabel('Mode Number')
zlabel('Mode Amplitude')
```



Use the computed IMFs to plot the Hilbert spectrum of the multicomponent signal. Restrict the frequency range to [0, 40] Hz.

```
hht(imf,fs,'FrequencyLimits',[0,40])
```



VMD of Piecewise Signal

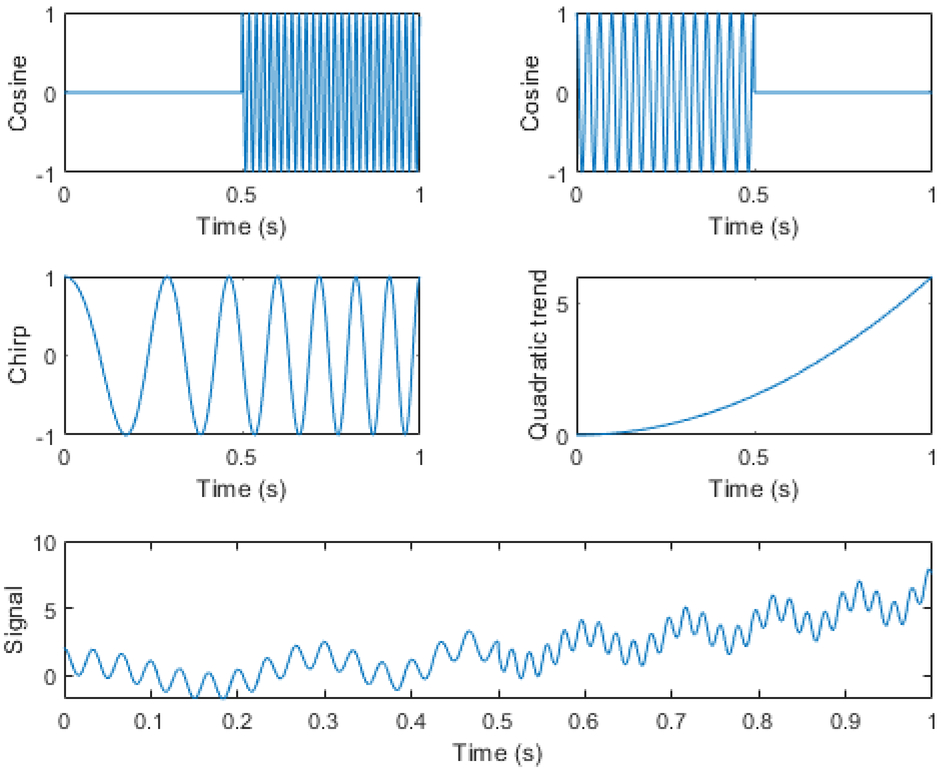
Generate a piecewise composite signal consisting of a quadratic trend, a chirp, and a cosine with a sharp transition between two constant frequencies at $t = 0.5$.

$$x(t) = 6t^2 + \cos\left(4\pi t + 10\pi t^2\right) + \begin{cases} \cos(60\pi t), & t \leq 0.5, \\ \cos(100\pi t - 10\pi), & t > 0.5. \end{cases}$$

The signal is sampled at 1 kHz for 1 second. Plot each individual component and the composite signal.

```
fs = 1e3;
t = 0:1/fs:1-1/fs;

x = 6*t.^2 + cos(4*pi*t+10*pi*t.^2) + ...
    [cos(60*pi*(t(t<=0.5))) cos(100*pi*(t(t>0.5)-10*pi))];

tiledlayout('flow')
nexttile
plot(t,[zeros(1,length(t)/2) cos(100*pi*(t(length(t)/2+1:end))-10*pi)])
xlabel('Time (s)')
ylabel('Cosine')

nexttile
```

```
plot(t,[cos(60*pi*(t(1:length(t)/2))) zeros(1,length(t)/2)])
xlabel('Time (s)')
ylabel('Cosine')

nexttile
plot(t,cos(4*pi*t+10*pi*t.^2))
xlabel('Time (s)')
ylabel('Chirp')


nexttile
plot(t,6*t.^2)
xlabel('Time (s)')
ylabel('Quadratic trend')

nexttile(5,[1 2])
plot(t,x)
xlabel('Time (s)')
ylabel('Signal')
```



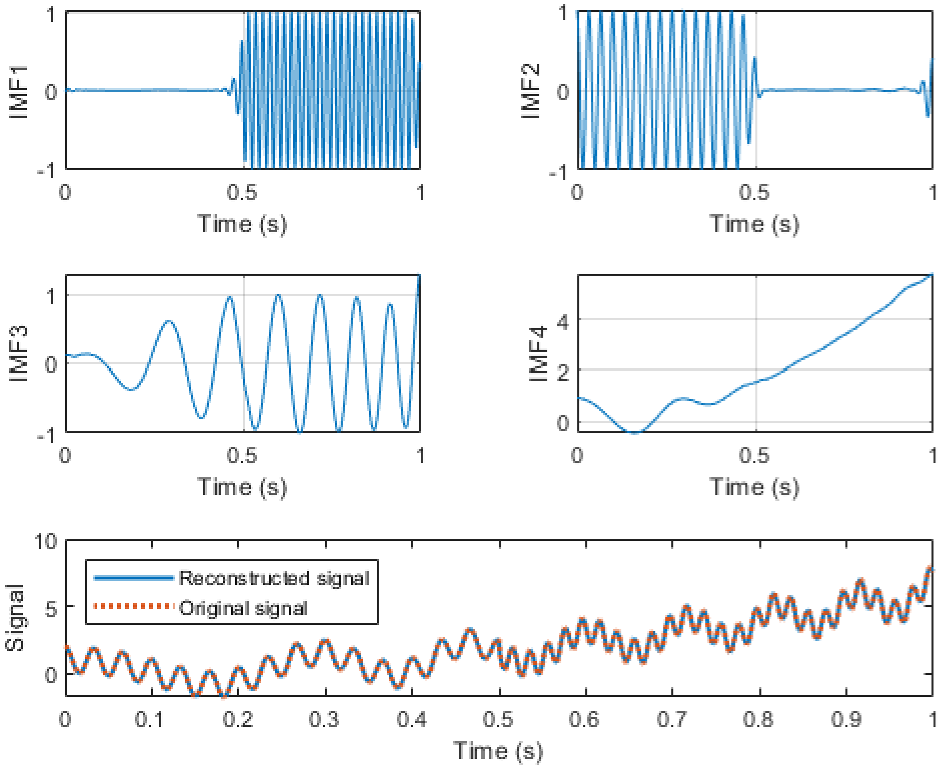
Perform variational mode decomposition to compute four intrinsic mode functions. The four distinct components of the signal are recovered.

```
[imf,res] = vmd(x,'NumIMFs',4);

tiledlayout('flow')

for i = 1:4
```

```
    nexttile
    plot(t,imf(:,i))
    txt = ['IMF',num2str(i)];
    ylabel(txt)
    xlabel('Time (s)')
    grid on
end
```

Reconstruct the signal by adding the mode functions and the residual. Plot and compare the original and reconstructed signals.

```
sig = sum(imf,2) + res;

nexttile(5,[1 2])
plot(t,sig,'LineWidth',1.5)

hold on

plot(t,x,':','LineWidth',2)
xlabel('Time (s)')
ylabel('Signal')
hold off
legend('Reconstructed signal','Original signal', ...
       'Location','northwest')
```



Calculate the norm of the difference between the original and the reconstructed signals.

```
norm(x-sig',Inf)
```

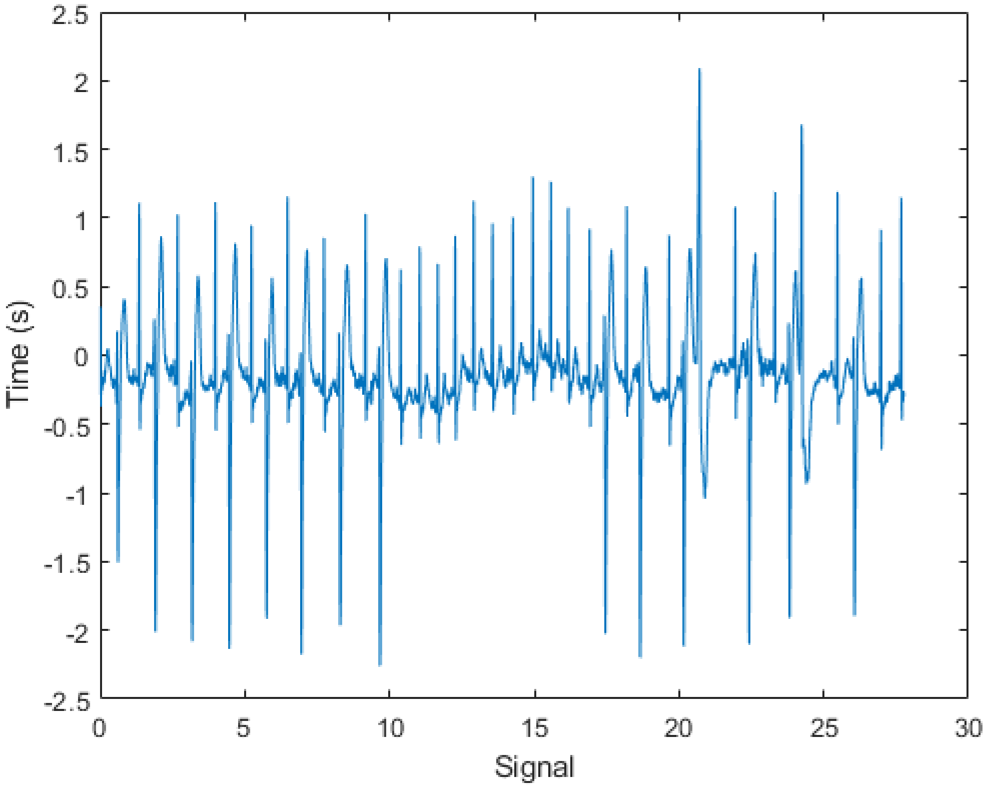
```
ans = 0
```

Noise Removal from ECG Signal Using VMD

The signals labeled in this example are from the MIT-BIH Arrhythmia Database [3] (Signal Processing Toolbox). The signal in the database was sampled at 360 Hz.

Load the MIT database signals corresponding to record 200 and plot the signal.

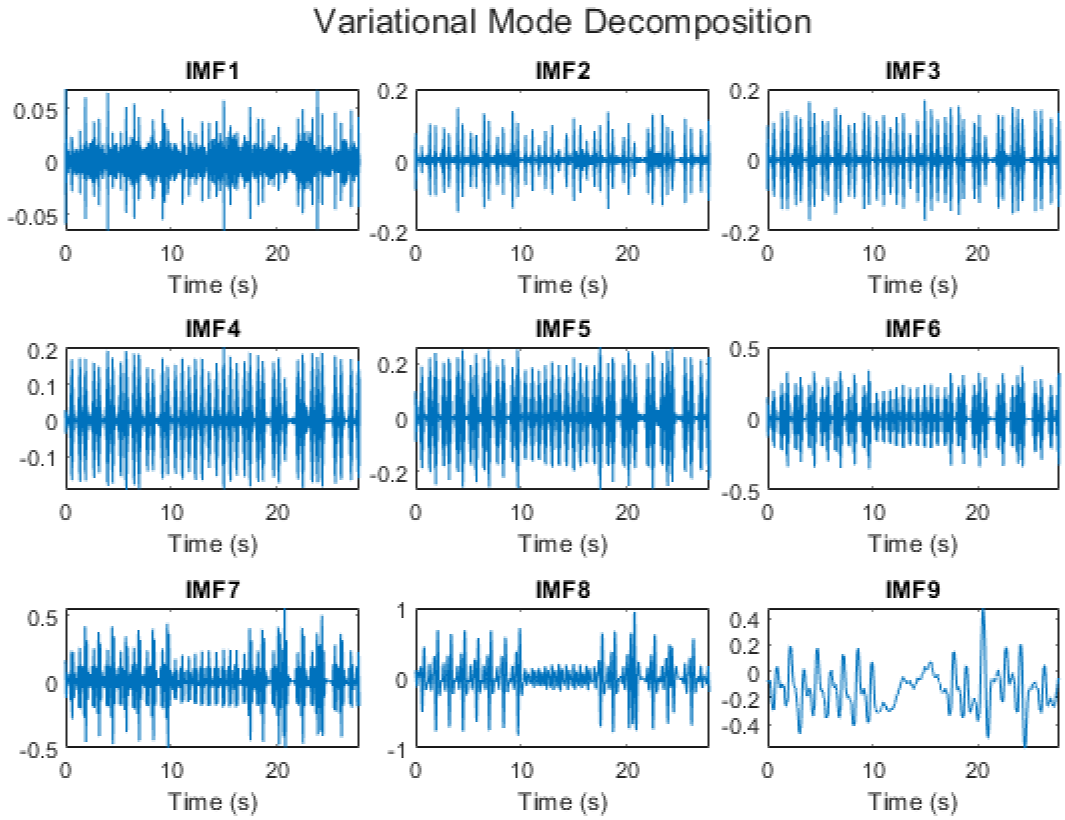
```
load mit200
Fs = 360;
plot(tm,ecgsig)
ylabel('Time (s)')
xlabel('Signal')
```



The ECG signal contains spikes driven by the rhythm of the heartbeat and an oscillating low frequency pattern. The distinct spokes of the ECG create important higher order harmonics.

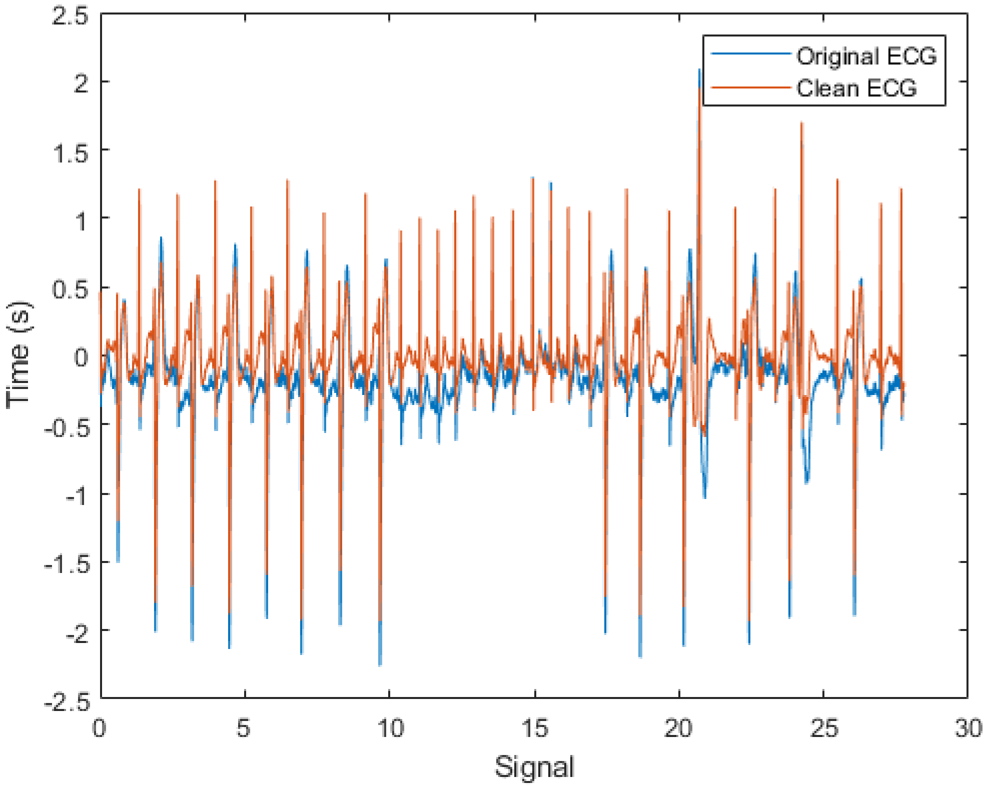
Calculate nine intrinsic mode functions of the windowed signal. Visualize the IMFs.

```
[imf,residual] = vmd(ecgsig,'NumIMF',9);
t = tiledlayout(3,3,'TileSpacing','compact','Padding','compact');
for n = 1:9
    ax(n) = nexttile(t);
    plot(tm,imf(:,n)')
    xlim([tm(1) tm(end)])
    txt = ['IMF',num2str(n)];
    title(txt)
    xlabel('Time (s)')
end
title(t,'Variational Mode Decomposition')
```



The first mode contains the most noise, and the second mode oscillates at the frequency of the heartbeat. Construct a clean ECG signal by summing all but the first and last VMD modes, thus discarding the low frequency baseline oscillation and most of the high frequency noise.

```
cleanECG = sum(imf(:,2:8),2);
figure
plot(tm,ecgsig,tm,cleanECG)
legend('Original ECG','Clean ECG')
ylabel('Time (s)')
xlabel('Signal')
```



Input Arguments

x — Uniformly sampled time-domain signal
vector | timetable

Uniformly sampled time-domain signal, specified as either a vector or a timetable. If x is a timetable, then it must contain increasing finite row times.. The timetable must contain only one numeric data vector with finite load values.

Note If a timetable has missing or duplicate time points, you can fix it using the tips in “Clean Timetable with Missing, Duplicate, or Nonuniform Times”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumIMF',10`

`AbsoluteTolerance` — Mode convergence absolute tolerance
`5e-6` (default) | positive real scalar

Mode convergence absolute tolerance, specified as the comma-separated pair consisting of 'AbsoluteTolerance' and a positive real scalar. AbsoluteTolerance is one of the stopping criteria for optimization, that is, optimization stops when the average squared absolute improvement toward convergence of IMFs, in two consecutive iterations, is less than AbsoluteTolerance.

RelativeTolerance — Mode convergence relative tolerance
AbsoluteTolerance*1e3 (default) | positive real scalar

Mode convergence relative tolerance, specified as the comma-separated pair consisting of 'RelativeTolerance' and a positive real scalar. RelativeTolerance is one of the stopping criteria for optimization, that is, optimization stops when the average relative improvement toward convergence of IMFs, in two consecutive iterations, is less than RelativeTolerance.

Note The optimization process stops when 'AbsoluteTolerance' and 'RelativeTolerance' are jointly satisfied.

MaxIterations — Maximum number of optimization iterations
500 (default) | positive scalar integer

Maximum number of optimization iterations, specified as the comma-separated pair consisting of 'MaxIterations' and a positive scalar integer. MaxIterations is one of the stopping criteria for optimization, that is, optimization stops when the number of iterations is greater than MaxIterations.

MaxIterations can be specified using only positive whole numbers.

NumIMF — Number of IMFs extracted
5 (default) | positive scalar integer

Number of IMFs extracted, specified as the comma-separated pair consisting of 'NumIMF' and a positive scalar integer.

CentralFrequencies — Initial central IMF frequencies
vector

Initial central IMF frequencies, specified as the comma-separated pair consisting of 'CentralFrequencies' and a vector of length NumIMFs. Vector values must be within [0, 0.5] cycles/sample, which indicates that the true frequencies are within [0, $f_s/2$], where f_s is the sample rate.

InitialIMFs — Initial IMFs
zero matrix (default) | real matrix

Initial IMFs, specified as the comma-separated pair consisting of 'InitialIMFs' and a real matrix. The rows correspond to time samples and columns correspond to modes.

PenaltyFactor — Penalty factor
1000 (default) | positive real scalar

Penalty factor, specified as the comma-separated pair consisting of 'PenaltyFactor' and a positive real scalar. This argument determines the reconstruction fidelity. Use smaller values of penalty factor to obtain stricter data fidelity.

InitialLM — Initial Lagrange multiplier

complex vector of zeros (default) | complex vector

Initial Lagrange multiplier, specified as the comma-separated pair consisting of 'InitialLM' and a complex vector. The range of the initial Lagrange multiplier in the frequency domain is [0, 0.5] cycles/sample. The multiplier enforces the reconstruction constraint. The length of the multiplier depends on the input size.

LMUpdateRate — Update rate for Lagrange multiplier

0.01 (default) | real scalar

Update rate for the Lagrange multiplier in each iteration, specified as the comma-separated pair consisting of 'LMUpdateRate' and a positive real scalar. A higher rate results in faster convergence, but increases the chance of the optimization process getting stuck in a local optimum.

InitializeMethod — Method to initialize central frequencies

'peaks' (default) | 'random' | 'grid'

Method to initialize the central frequencies, specified as the comma-separated pair consisting of 'InitializeMethod' and either 'peaks', 'random', or 'grid'.

Specify InitializeMethod as:

- 'peaks' to initialize the central frequencies as the peak locations of the signal in the frequency domain (default).
- 'random' to initialize the central frequencies as random numbers distributed uniformly in the interval [0,0.5] cycles/sample.
- 'grid' to initialize the central frequencies as a uniformly sampled grid in the interval [0,0.5] cycles/sample.

Display — Toggle information display in command window

false or 0 (default) | true or 1

Toggle progress display in the command window, specified as the comma-separated pair consisting of 'Display' and either 'true' (or 1) or 'false' (or 0). If you specify 'true', the function displays the average absolute and relative improvement of modes and central frequencies every 20 iterations, and show the final stopping information.

Specify Display as 1 to show the table or 0 to hide the table.

Output Arguments

imf — Intrinsic mode function

matrix | timetable

Intrinsic mode functions, returned as a matrix or timetable. Each imf is an amplitude and frequency modulated signal with positive and slowly varying envelopes. Each mode has an instantaneous frequency that is nondecreasing, varies slowly, and is concentrated around a central value. Use imf to apply Hilbert-Huang transform to perform spectral analysis on the signal.

imf is returned as:

- A matrix where each column is an imf, when x is a vector

- A timetable, when x is a single data column timetable

residual — Residual signal
column vector | single data column timetable

Residual signal, returned as a column vector or a single data column timetable. residual represents the portion of the original signal x not decomposed by vmd.

residual is returned as:

- A column vector, when x is a vector.
- A single data column timetable, when x is a single data column timetable.

info — Additional information for diagnostics
structure

Additional information for diagnostics, returned as a structure with these fields:

- ExitFlag – Termination flag. A value of 0 indicates the algorithm stopped when it reached the maximum number of iterations. A value of 1 indicates the algorithm stopped when it met the absolute and relative tolerances.
- CentralFrequencies – Central frequencies of the IMFs.
- NumIterations – Total number of iterations.
- AbsoluteImprovement – Average squared absolute improvement toward convergence of the IMFs between the final two iterations.
- RelativeImprovement – Average relative improvement toward convergence of the IMFs between the final two iterations.
- LagrangeMultiplier – Frequency-domain Lagrange multiplier at the last iteration.

More About

Intrinsic Mode Functions

The vmd function decomposes a signal $x(t)$ into a small number K of narrowband intrinsic mode functions (IMFs):

$$x(t) = \sum_{k=1}^{K} u_k(t)\,.$$

The IMFs have these characteristics:

1. Each mode u_k is an amplitude and frequency modulated signal of the form

   $$u_k(t) = A_k(t)\cos(\phi_k(t)),$$

   where $\phi_k(t)$ is the phase of the mode and $A_k(t)$ is its envelope.
2. The modes have positive and slowly varying envelopes.
3. Each mode has an instantaneous frequency $\phi'_k(t)$ that is nondecreasing, varies slowly, and is concentrated around a central value f_k.

The variational mode decomposition method simultaneously calculates all the mode waveforms and their central frequencies. The process consists of finding a set of $u_k(t)$ and $f_k(t)$ that minimize the constrained variational problem.

Optimization

To calculate u_k and f_k, the procedure finds an optimum of the augmented Lagrangian

$$L(u_k(t), f_k, \lambda(t)) = \underbrace{\alpha \sum_{k=1}^{K} \left\| \frac{d}{dt}\left[\left(\delta(t) + \frac{j}{\pi t}\right) * u_k(t)\right] e^{-j2\pi f_k t} \right\|_2^2}_{(i)} + \underbrace{\left\| x(t) - \sum_{k=1}^{K} u_k(t) \right\|_2^2}_{(ii)} + \underbrace{\left\langle \lambda(t), x(t) - \sum_{k=1}^{K} u_k(t) \right\rangle}_{(iii)},$$

where the inner product $\langle p(t), q(t) \rangle = \int_{-\infty}^{\infty} p^*(t)\, q(t)\, dt$ and the 2-norm $\|p(t)\|_2^2 = \langle p(t), p(t) \rangle$. The regularization term (i) includes these steps:

1. Use the Hilbert transform to calculate the analytic signal associated with each mode, where * denotes convolution. This results in each mode having a purely positive spectrum.
2. Demodulate the analytic signal to baseband by multiplying it with a complex exponential.
3. Estimate the bandwidth by calculating the squared 2-norm of the gradient of the demodulated analytic signal.

Terms (ii) and (iii) enforce the constraint $x(t) = \sum_{k=1}^{K} u_k(t)$ by imposing a quadratic penalty and incorporating a Lagrange multiplier. The `PenaltyFactor` α measures the relative importance of (i) compared to (ii) and (iii).

The algorithm solves the optimization problem using the alternating direction method of multipliers described in [1] (Signal Processing Toolbox).

Algorithms

The `vmd` function calculates the IMFs in the frequency domain, reconstructing $X(f) = \mathrm{DFT}\{x(t)\}$ in terms of $U_k(f) = \mathrm{DFT}\{u_k(t)\}$. To remove edge effects, the algorithm extends the signal by mirroring half its length on either side.

The Lagrange multiplier introduced in “Optimization” (Signal Processing Toolbox) has the Fourier transform $\Lambda(f)$. The length of the Lagrange multiplier vector is the length of the extended signal.

Unless otherwise specified in `'InitialIMFs'`, the IMFs are initialized at zero. Initialize `'CentralFrequencies'` using one of the methods specified in `'InitializeMethod'`. `vmd` iteratively updates the modes until one of these conditions is met:

- $\sum_k \|u_k^{n+1}(t) - u_k^n(t)\|_2^2 / \|u_k^n(t)\|_2^2 < \varepsilon_r$ and $\sum_k \|u_k^{n+1}(t) - u_k^n(t)\|_2^2 < \varepsilon_a$ are jointly satisfied, where ε_r and ε_a are specified using `'RelativeTolerance'` and `'AbsoluteTolerance'`, respectively.
- The algorithm exceeds the maximum number of iterations specified in `'MaxIterations'`.

For the $(n + 1)$ -th iteration, the algorithm performs these steps:

1. Iterate over the K modes of the signal (specified using `'NumIMF'`) to compute:

a The frequency-domain waveforms for each mode using

$$U_k^{n+1}(f) = \frac{X(f) - \sum_{i<k} U_k^{n+1}(f) - \sum_{i>k} U_k^n(f) + \frac{\Lambda^n}{2}(f)}{1 + 2\alpha\left\{2\pi(f - f_k^n)\right\}^2},$$

where $U_k^{n+1}(f)$ is the Fourier transform of the kth mode calculated in the $(n+1)$-th iteration.

b The kth central frequency f_k^{n+1} using

$$f_k^{n+1} = \frac{\int_0^\infty \left|U_k^{n+1}(f)\right|^2 f\,df}{\int_0^\infty \left|U_k^{n+1}(f)\right|^2 df} \approx \frac{\sum f\left|U_k^{n+1}(f)\right|^2}{\sum \left|U_k^{n+1}(f)\right|^2}.$$

2 Update the Lagrange multiplier using $\Lambda^{n+1}(f) = \Lambda^n(f) + \tau(X(f) - \sum_k U_k^{n+1}(f))$, where τ is the update rate of the Lagrange multiplier, specified using `'LMUpdateRate'`.

References

[1] Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." *Foundations and Trends® in Machine Learning*. Vol 3, Number 1, 2011, pp. 1–122.

[2] Dragomiretskiy, Konstantin, and Dominique Zosso. "Variational Mode Decomposition." *IEEE Transactions on Signal Processing*. Vol. 62, Number 3, 2014, pp. 531–534.

[3] Moody, George B., and Roger G. Mark. "The impact of the MIT-BIH Arrhythmia Database." *IEEE Engineering in Medicine and Biology Magazine*. Vol. 20, No. 3, May–June 2001, pp. 45–50.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Timetables are not supported for code generation.

See Also

Apps
Signal Multiresolution Analyzer

Functions
`hht` | `emd`

Topics
“Time-Frequency Gallery”

Introduced in R2020a

wave2lp

Laurent polynomials associated with wavelet

Syntax

```
[LoDz,HiDz,LoRz,HiRz] = wave2lp(wname)
[___,PRCond,AACond] = wave2lp(wname)
[___] = wave2lp(wname,PmaxHS)
[___] = wave2lp(wname,PmaxHS,AddPOW)
```

Description

`[LoDz,HiDz,LoRz,HiRz] = wave2lp(wname)` returns the four Laurent polynomials associated with the wavelet `wname`. The pairs (`LoRz`,`HiRz`) and (`LoDz`,`HiDz`) are associated with the synthesis and analysis filters, respectively.

`[___,PRCond,AACond] = wave2lp(wname)` also returns the perfect reconstruction condition `PRCond` and the anti-aliasing condition `AACond`.

`[___] = wave2lp(wname,PmaxHS)` sets the maximum order of `LoRz`.

`[___] = wave2lp(wname,PmaxHS,AddPOW)` sets the maximum order of the Laurent polynomial `HiRz`.

Examples

Laurent Polynomials Associated with Wavelet

Obtain the four Laurent polynomials associated with the orthogonal wavelet `db3`. Also obtain the perfect reconstruction and anti-aliasing conditions.

```
[LoDz,HiDz,LoRz,HiRz,PRC,AAC] = wave2lp("db3")
```

```
LoDz = 
  laurentPolynomial with properties:

    Coefficients: [0.0352 -0.0854 -0.1350 0.4599 0.8069 0.3327]
        MaxOrder: 5


HiDz = 
  laurentPolynomial with properties:

    Coefficients: [0.3327 -0.8069 0.4599 0.1350 -0.0854 -0.0352]
        MaxOrder: 1


LoRz = 
  laurentPolynomial with properties:

    Coefficients: [0.3327 0.8069 0.4599 -0.1350 -0.0854 0.0352]
```

```
        MaxOrder: 0


HiRz = 
  laurentPolynomial with properties:

    Coefficients: [-0.0352 -0.0854 0.1350 0.4599 -0.8069 0.3327]
        MaxOrder: 4


PRC = 
  laurentPolynomial with properties:

    Coefficients: 2.0000
        MaxOrder: 0


AAC = 
  laurentPolynomial with properties:

    Coefficients: 0
        MaxOrder: 0
```

Verify the perfect reconstruction condition.

```
eq(LoRz*LoDz + HiRz*HiDz,PRC)
```

```
ans = logical
   1
```

Verify the anti-aliasing condition. Use the helper function `helperMakeLaurentPoly` on page 1-0 to obtain $LoD(- z)$, where $LoD(z)$ is the Laurent polynomial `LoDz`. Use the helper function `helperMakeLaurentPoly` to obtain $HiD(- z)$, where $HiD(z)$ is the Laurent polynomial `HiDz`.

```
LoDzm = helperMakeLaurentPoly(LoDz);
HiDzm = helperMakeLaurentPoly(HiDz);
eq(LoRz*LoDzm + HiRz*HiDzm,AAC)
```

```
ans = logical
   1
```

Helper Functions

```
function polyout = helperMakeLaurentPoly(poly)
% This function is only intended to support this example.
% It may change or be removed in a future release.

polyout = poly;
cflen = length(polyout.Coefficients);
cmo = polyout.MaxOrder;
polyneg = (-1).^(mod(cmo,2)+(0:cflen-1));
polyout.Coefficients = polyout.Coefficients.*polyneg;
```

```
end
```

Input Arguments

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. `wname` must be one of the wavelets supported by `liftingScheme`. See the Wavelet property of `liftingScheme` for the list of wavelets.

Example: `[LoDz,HiDz,LoRz,HiRz] = wave2lp("db2")`

Data Types: `char` | `string`

PmaxHS — Maximum power

`0` (default) | integer

Maximum power of the Laurent polynomial `LoRz`, specified as an integer.

Example: If `[~,~,LoRz,HiRz] = wave2lp("db2",3)`, then the maximum power, or order, of the Laurent polynomial `LoRz` is 3.

Data Types: `double`

AddPOW — Integer

`0` (default) | integer

Integer to set the maximum order of the Laurent polynomial `HiRz`. `PmaxHiRz`, the maximum order of `HiRz`, is

```
    PmaxHiRz = PmaxHS+length(HiRz.Coefficients)-2+AddPow.
```

`AddPOW` must be an even integer to preserve the perfect reconstruction condition.

Data Types: `double`

Output Arguments

LoDz — Laurent polynomial

`laurentPolynomial` object

Laurent polynomial associated with the lowpass analysis filter, returned as a `laurentPolynomial` object.

HiDz — Laurent polynomial

`laurentPolynomial` object

Laurent polynomial associated with the highpass analysis filter, returned as a `laurentPolynomial` object.

LoRz — Laurent polynomial

`laurentPolynomial` object

Laurent polynomial associated with the lowpass synthesis filter, returned as a `laurentPolynomial` object.

HiRz — Laurent polynomial
laurentPolynomial object

Laurent polynomial associated with the highpass synthesis filter, returned as a laurentPolynomial object.

PRCond,AACond — Perfect reconstruction and anti-aliasing conditions
laurentPolynomial objects

Perfect reconstruction and anti-aliasing conditions, returned as laurentPolynomial objects. The perfect reconstruction condition PRCond and anti-aliasing condition AACond are:

- PRCond(*z*) = LoRz(*z*) LoDz(*z*) + HiRz(*z*) HiDz(*z*)
- AACond(*z*) = LoRz(*z*) LoDz(-*z*) + HiRz(*z*) HiDz(-*z*)

The pairs (LoRz, HiRz) and (LoDz, HiDz) are associated with perfect reconstructions filters if and only if:

- PRCond(*z*) = 2, and
- AACond(*z*) = 0

If PRCond(*z*) = 2 z^d, a delay is introduced in the reconstruction process.

Compatibility Considerations

wave2lp input syntax has changed
Behavior changed in R2021b

The wave2lp input syntax has changed.

- You can now set the maximum order of LoRz using PmaxHS.
- You can now set the maximum order of HiRz using AddPOW.

See Also

Functions
filters2lp | lp2filters

Objects
laurentMatrix | laurentPolynomial | liftingScheme

Introduced in R2021b

wavedec

1-D wavelet decomposition

Syntax

```
[c,l] = wavedec(x,n,wname)
[c,l] = wavedec(x,n,LoD,HiD)
```

Description

`[c,l] = wavedec(x,n,wname)` returns the wavelet decomposition of the 1-D signal `x` at level `n` using the wavelet `wname`. The output decomposition structure consists of the wavelet decomposition vector `c` and the bookkeeping vector `l`, which is used to parse `c`.

Note For `gpuArray` inputs, the supported modes are `'symh'` (`'sym'`) and `'per'`. If the input is a `gpuArray`, the discrete wavelet transform extension mode used by `wavedec` defaults to `'symh'` unless the current extension mode is `'per'`. See the example "Multilevel Discrete Wavelet Transform on a GPU" on page 1-1381.

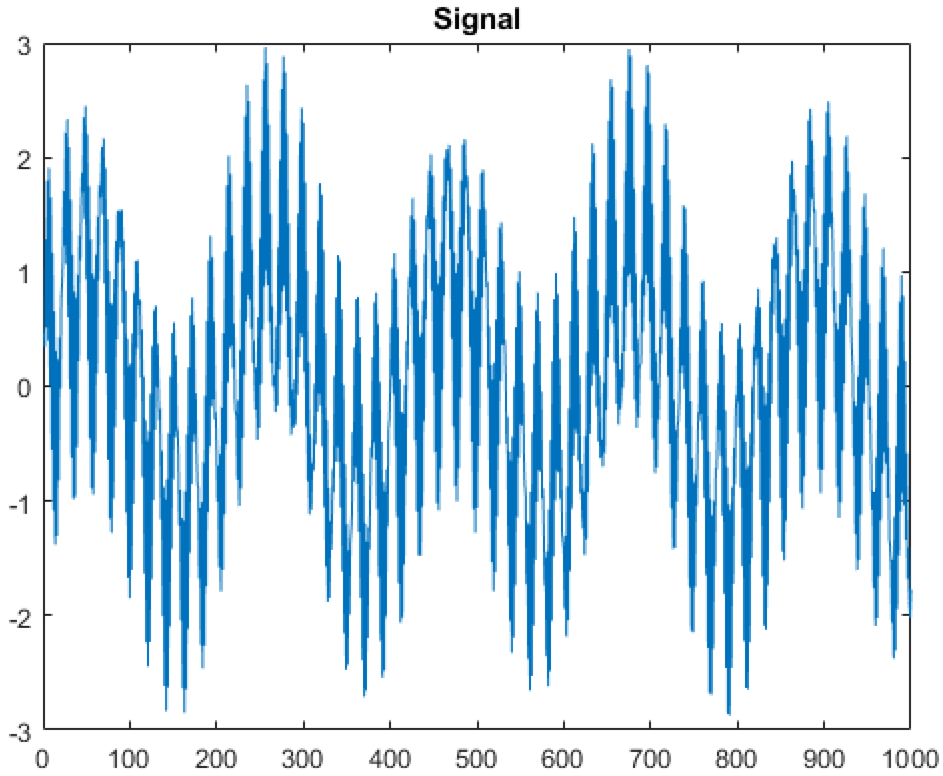
`[c,l] = wavedec(x,n,LoD,HiD)` returns the wavelet decomposition using the specified lowpass and highpass wavelet decomposition filters `LoD` and `HiD`, respectively.

Examples

Multilevel One-Dimensional Wavelet Analysis

Load and plot a one-dimensional signal.

```
load sumsin
plot(sumsin)
title('Signal')
```

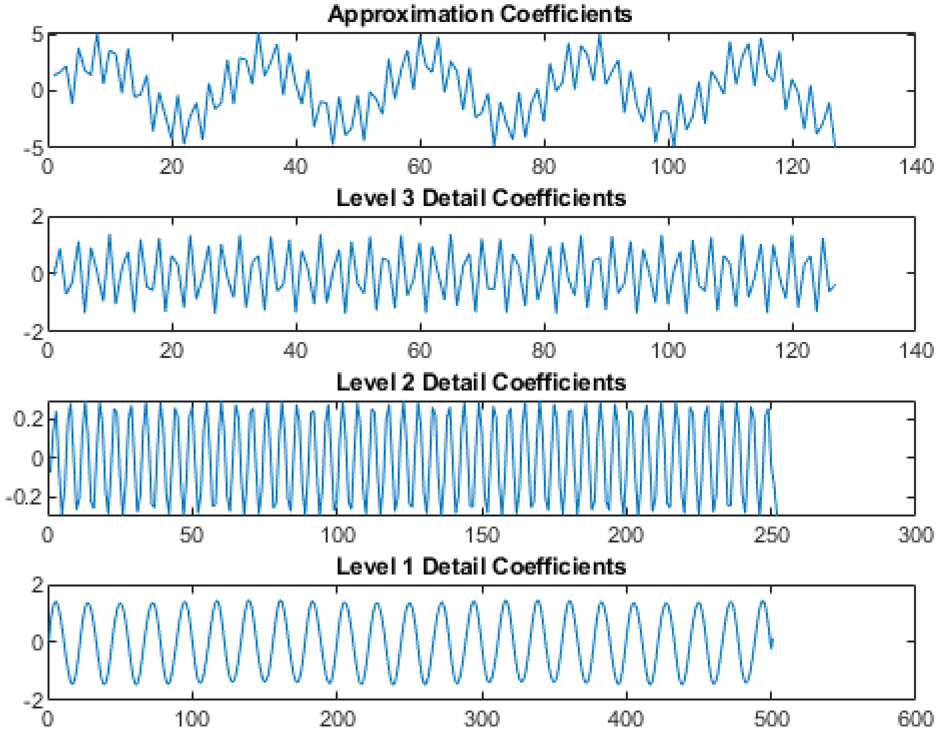


Perform a 3-level wavelet decomposition of the signal using the order 2 Daubechies wavelet. Extract the coarse scale approximation coefficients and the detail coefficients from the decomposition.

```
[c,l] = wavedec(sumsin,3,'db2');
approx = appcoef(c,l,'db2');
[cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);
```

Plot the coefficients.

```
subplot(4,1,1)
plot(approx)
title('Approximation Coefficients')
subplot(4,1,2)
plot(cd3)
title('Level 3 Detail Coefficients')
subplot(4,1,3)
plot(cd2)
title('Level 2 Detail Coefficients')
subplot(4,1,4)
plot(cd1)
title('Level 1 Detail Coefficients')
```



Multilevel Discrete Wavelet Transform on a GPU

Refer to “GPU Support by Release” (Parallel Computing Toolbox) to see what GPUs are supported.

Load the noisy Doppler signal. Put the signal on the GPU using `gpuArray`. Save the current extension mode.

```
load noisdopp
noisdoppg = gpuArray(noisdopp);
origMode = dwtmode('status','nodisp');
```

Use `dwtmode` to change the extension mode to zero-padding. Obtain the three-level DWT of the signal on the GPU using the `db4` wavelet.

```
dwtmode('zpd','nodisp')
[c,l] = wavedec(noisdoppg,3,'db4');
```

The current extension mode `zpd` is not supported for `gpuArray` input. Therefore, the DWT is instead performed using the `sym` extension mode. To confirm this, set the extension mode to `sym` and take DWT of `noisdoppg`, then compare with the previous result.

```
dwtmode('sym','nodisp')
[csym,lsym] = wavedec(noisdoppg,3,'db4');
[max(abs(c-csym)) max(abs(l-lsym))]
```

```
ans =

     0     0
```

Set the current extension mode to `per` and obtain the three-level DWT of `noisdopp`. The extension mode `per` is supported for `gpuArray` input. Confirm the result is different from the `sym` results.

```
dwtmode('per','nodisp')
[cper,lper] = wavedec(noisdoppg,3,'db4');
[length(csym) ; length(cper)]
```

```
ans = 2×1

        1044
        1024
```

```
[lsym ; lper]
```

```
ans = 2×5

         134          134          261          515         1024
         128          128          256          512         1024
```

Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

x — Input signal

vector

Input signal, specified as a vector.

Data Types: `single` | `double`

n — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer. `wavedec` does not enforce a maximum level restriction. Use `wmaxlev` to ensure that 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)))`.

Data Types: `single` | `double`

wname — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar.

Note `wavedec` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoD,HiD` — Wavelet decomposition filters

even-length real-valued vectors

Wavelet decomposition filters, specified as a pair of even-length real-valued vectors. `LoD` is the lowpass decomposition filter, and `HiD` is the highpass decomposition filter. The lengths of `LoD` and `HiD` must be equal. See `wfilters` for additional information.

Data Types: `single` | `double`

Output Arguments

`c` — Wavelet decomposition vector

vector

Wavelet decomposition vector, returned as a vector. The bookkeeping vector `l` is used to parse the coefficients in the wavelet decomposition vector by level.

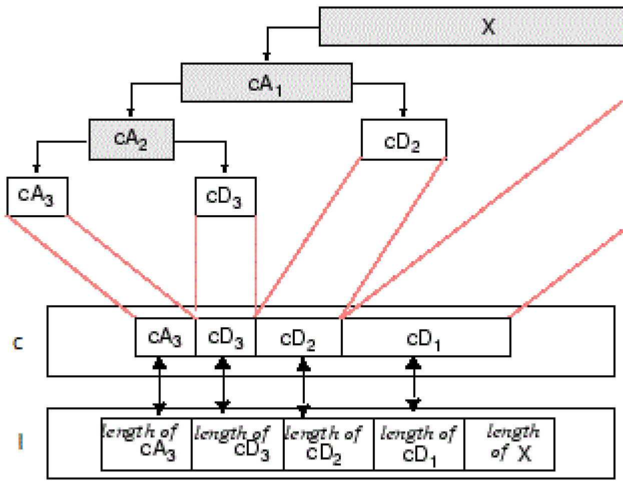
Data Types: `single` | `double`

`l` — Bookkeeping vector

vector of positive integers

Bookkeeping vector, returned as a vector of positive integers. The vector contains the number of coefficients by level and the length of the original signal.

The bookkeeping vector is used to parse the coefficients in the wavelet decomposition vector `c` by level. The decomposition vector and bookkeeping vector are organized as in this level-3 decomposition diagram.

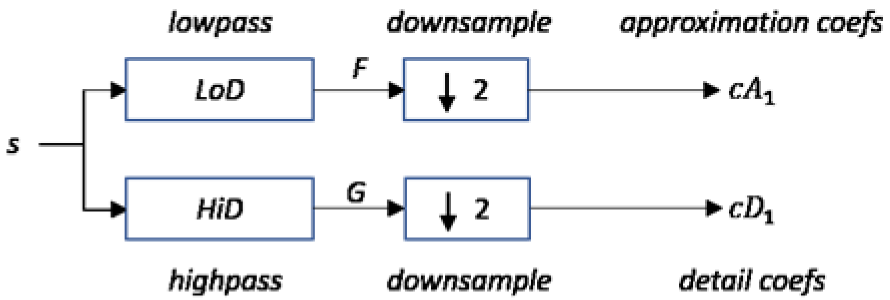


Data Types: `single` | `double`

Algorithms

Given a signal s of length N, the DWT consists of at most $\log_2 N$ steps. Starting from s, the first step produces two sets of coefficients: approximation coefficients cA_1 and detail coefficients cD_1.

Convolving *s* with the lowpass filter `LoD` and the highpass filter `HiD`, followed by dyadic decimation (downsampling), results in the approximation and detail coefficients respectively.



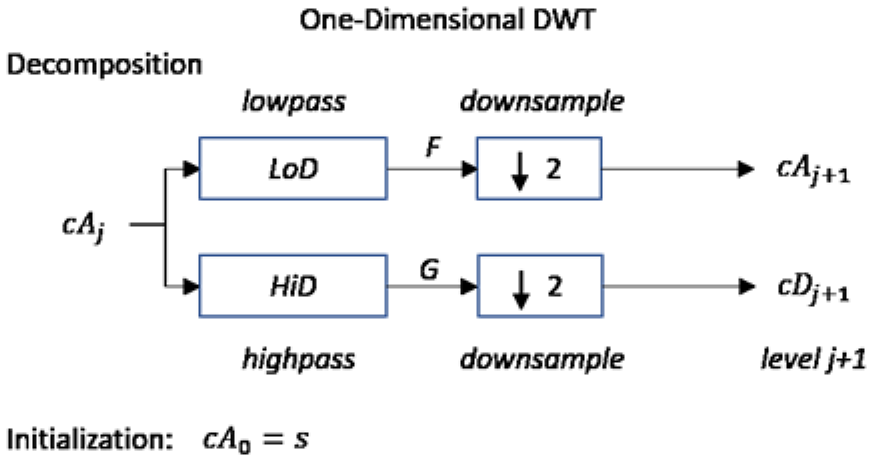
where

- $\boxed{X}$ — Convolve with filter *X*
- $\boxed{\downarrow 2}$ — Downsample (keep the even-indexed elements)

The length of each filter is equal to 2*n*. If *N* = length(*s*), the signals *F* and *G* are of length *N* + 2*n* −1 and the coefficients cA_1 and cD_1 are of length

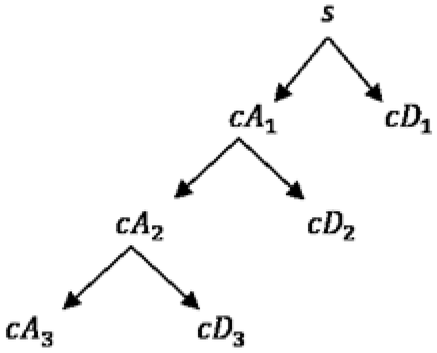
$$\text{floor}\left(\frac{N-1}{2}\right) + n.$$

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.



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

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.
- The level of decomposition, `n` must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`dwt` | `waveinfo` | `waverec` | `wfilters` | `wmaxlev` | `appcoef` | `detcoef` | `dwtfilterbank`

Introduced before R2006a

wavedec2

2-D wavelet decomposition

Syntax

```
[C,S] = wavedec2(X,N,wname)
[C,S] = wavedec2(X,N,LoD,HiD)
```

Description

`[C,S] = wavedec2(X,N,wname)` returns the wavelet decomposition of the matrix `X` at level `N` using the wavelet `wname`. The output decomposition structure consists of the wavelet decomposition vector `C` and the bookkeeping matrix `S`, which contains the number of coefficients by level and orientation.

Note For `gpuArray` inputs, the supported modes are `'symh'` (`'sym'`) and `'per'`. If the input is a `gpuArray`, the discrete wavelet transform extension mode used by `wavedec2` defaults to `'symh'` unless the current extension mode is `'per'`. See the example “Multilevel 2-D Discrete Wavelet Transform on a GPU” on page 1-1391.

`[C,S] = wavedec2(X,N,LoD,HiD)` returns the wavelet decomposition using the specified lowpass and highpass decomposition filters `LoD` and `HiD`, respectively. See `wfilters` for details.

Examples

Extract and Display Image Decomposition Level

This example shows how to extract and display images of wavelet decomposition level details.

Load an image. Perform a level 2 wavelet decomposition of the image using the `haar` wavelet.

```
load woman
[c,s]=wavedec2(X,2,'haar');
```

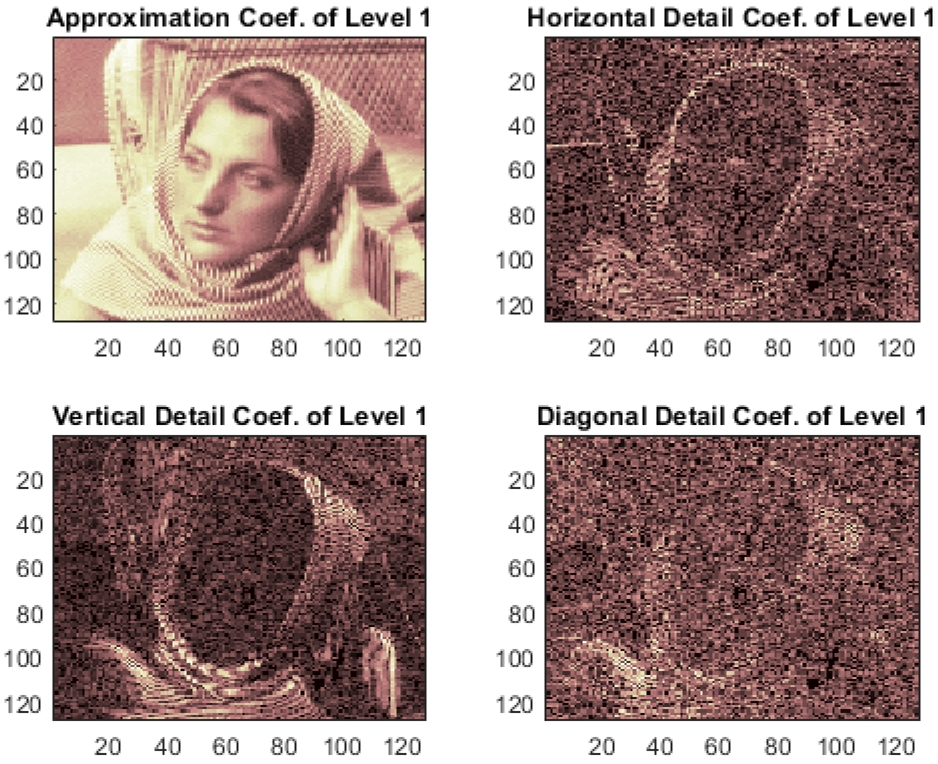
Extract the level 1 approximation and detail coefficients.

```
[H1,V1,D1] = detcoef2('all',c,s,1);
A1 = appcoef2(c,s,'haar',1);
```

Use `wcodemat` to rescale the coefficients based on their absolute values. Display the rescaled coefficients.

```
V1img = wcodemat(V1,255,'mat',1);
H1img = wcodemat(H1,255,'mat',1);
D1img = wcodemat(D1,255,'mat',1);
A1img = wcodemat(A1,255,'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')
```



Approximation Coef. of Level 1

Horizontal Detail Coef. of Level 1

Vertical Detail Coef. of Level 1

Diagonal Detail Coef. of Level 1

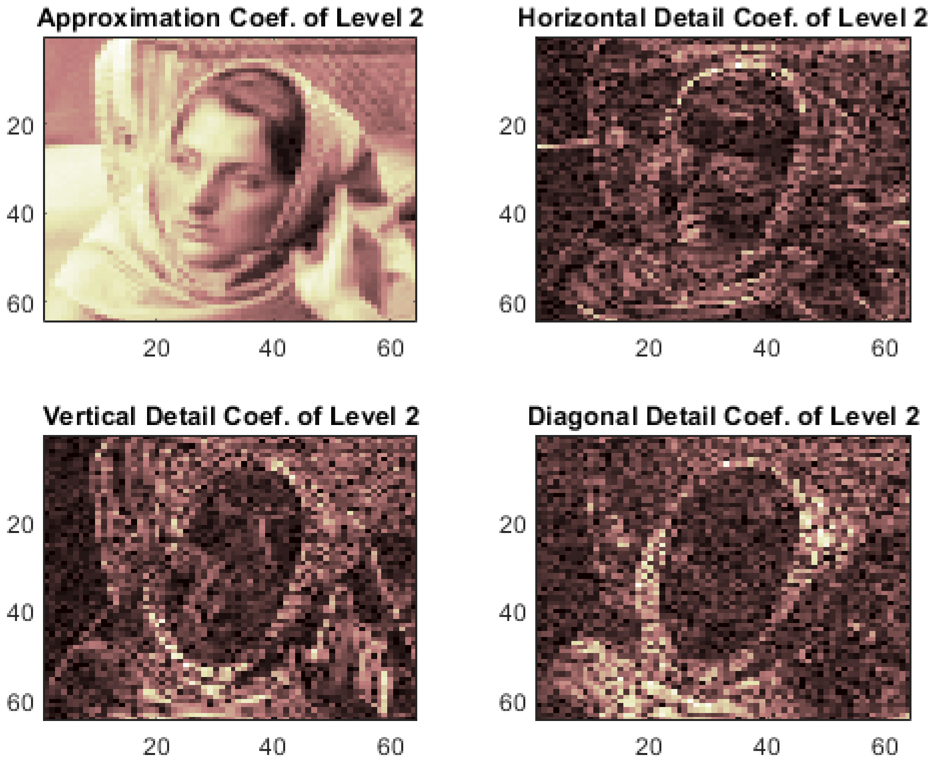
Extract the level 2 approximation and detail coefficients.

```
[H2,V2,D2] = detcoef2('all',c,s,2);
A2 = appcoef2(c,s,'haar',2);
```

Use wcodemat to rescale the coefficients based on their absolute values. Display the rescaled coefficients.

```
V2img = wcodemat(V2,255,'mat',1);
H2img = wcodemat(H2,255,'mat',1);
D2img = wcodemat(D2,255,'mat',1);
A2img = wcodemat(A2,255,'mat',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')
```



2-D Wavelet Decomposition Structure

This example shows the structure of `wavedec2` output matrices.

Load and display an image.

```
load woman
imagesc(X)
colormap(map)
```

Save the current discrete wavelet transform extension mode.

```
origMode = dwtmode('status','nodisplay');
```

Change to periodic boundary handling. The `dwtmode` function displays a message indicating that the DWT extension mode is changing.

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

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

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

Perform a level 3 decomposition of the image using the `db1` (Haar) wavelet.

```
[c,s] = wavedec2(X,3,'db1');
```

Return the number of elements in the image `X` and coefficient vector `c`. Confirm the number of elements in each are equal.

```
numel(X)
```

```
ans = 65536
```

```
numel(c)
```

```
ans = 65536
```

Display the bookkeeping matrix `s`. The first row displays the dimensions of the coarse scale approximation of the image. The last row displays the dimensions of the original image. The intermediate rows display the dimensions of the detail coefficients at the three levels of the decomposition, proceeding from coarse to fine scale.

```
s
```

```
s = 5×2

    32    32
    32    32
    64    64
   128   128
   256   256
```

Reset discrete wavelet transform extension mode to its original mode.

```
dwtmode(origMode,'nodisplay')
```

Multilevel 2-D Discrete Wavelet Transform on a GPU

Refer to “GPU Support by Release” (Parallel Computing Toolbox) to see what GPUs are supported.

Load an image. Put the image on the GPU using `gpuArray`. Save the current extension mode.

```
load mask
imgg = gpuArray(X);
origMode = dwtmode('status','nodisp');
```

Use `dwtmode` to change the extension mode to zero-padding. Obtain the three-level DWT of the image on the GPU using the `db4` wavelet.

```
dwtmode('zpd','nodisp')
[c,s] = wavedec2(imgg,3,'db4');
```

The current extension mode `zpd` is not supported for `gpuArray` input. Therefore, the DWT is instead performed using the `sym` extension mode. To confirm this, set the extension mode to `sym` and take DWT of `noisdoppg`, then compare with the previous result.

```
dwtmode('sym','nodisp')
[csym,ssym] = wavedec2(imgg,3,'db4');
[max(abs(c-csym)) max(abs(s-ssym))]
```

```
ans =

     0     0     0
```

Set the current extension mode to `per` and obtain the three-level DWT of `imgg`. The extension mode `per` is supported for `gpuArray` input. Confirm the result is different from the `sym` results.

```
dwtmode('per','nodisp')
[cper,sper] = wavedec2(imgg,3,'db4');
[length(csym) ; length(cper)]
```

```
ans = 2×1

       71542
       65536
```

```
ssym
```

```
ssym = 5×2

    38    38
    38    38
    69    69
   131   131
   256   256
```

```
sper
```

```
sper = 5×2

    32    32
    32    32
    64    64
   128   128
   256   256
```

Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

X — Input data

real-valued matrix | 3-D array of RGB triplets

Input data, specified as a real-valued *M*-by-*N* matrix representing an indexed image, or an *M*-by-*N*-by-3 array representing a truecolor image. For more information on truecolor images, see “RGB (Truecolor) Images”.

Data Types: `double`

N — Decomposition level

positive integer

Decomposition level, specified as a positive integer. `wavedec2` does not enforce a maximum level restriction. Use `wmaxlev` to determine the maximum decomposition level possible of the matrix `X` using the wavelet `wname`. The maximum level is the last level for which at least one coefficient is correct.

Data Types: `double`

`wname` — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar.

Note `wavedec2` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoD,HiD` — Wavelet decomposition filters

even-length real-valued vectors

Wavelet decomposition filters associated with an orthogonal or biorthogonal wavelet, specified as even-length real-valued vectors. `LoD` is the lowpass decomposition filter, and `HiD` is the highpass decomposition filter. See `wfilters` for details.

Data Types: `double`

Output Arguments

`C` — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector. The vector `C` contains the approximation and detail coefficients organized by level. The bookkeeping matrix `S` is used to parse `C`.

The vector `C` is organized as *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 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.

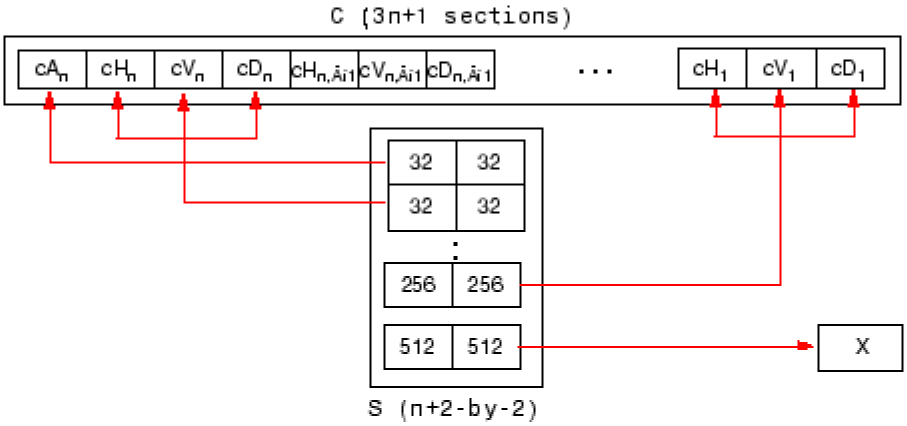
`S` — Bookkeeping matrix

integer-valued matrix

Bookkeeping matrix. The matrix `S` contains the dimensions of the wavelet coefficients by level and is used to parse the wavelet decomposition vector `C`.

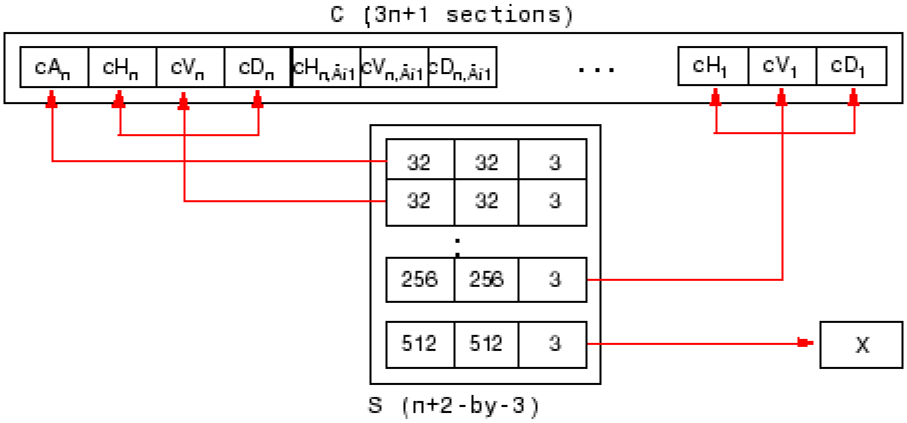
- `S(1,:)` = size of approximation coefficients(`N`).
- `S(i,:)` = size of detail coefficients(`N-i+2`) for `i` = 2, ...`N+1` and `S(N+2,:) = size(X)`.

The following diagram shows the relationship between `C` and `S` in the wavelet decomposition of a 512-by-512 matrix.



When X represents an indexed image, 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 shown.

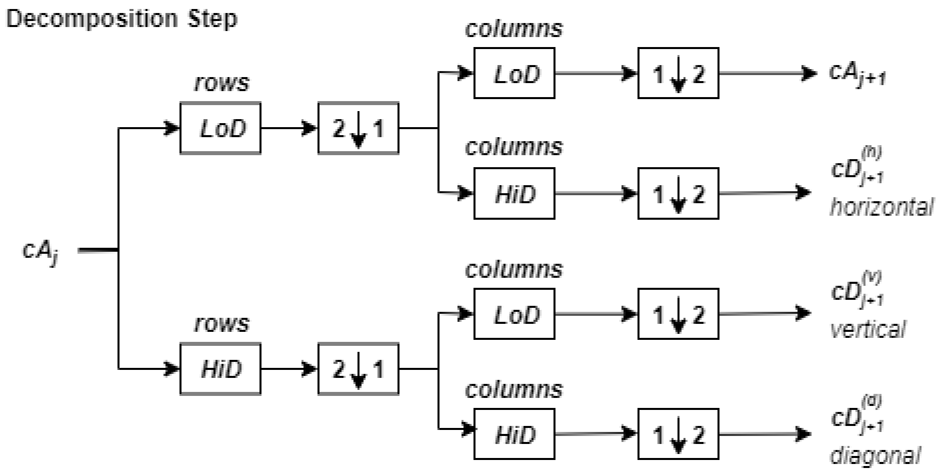


Algorithms

For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional vectors 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 chart describes the basic decomposition step for images:

Two-Dimensional DWT



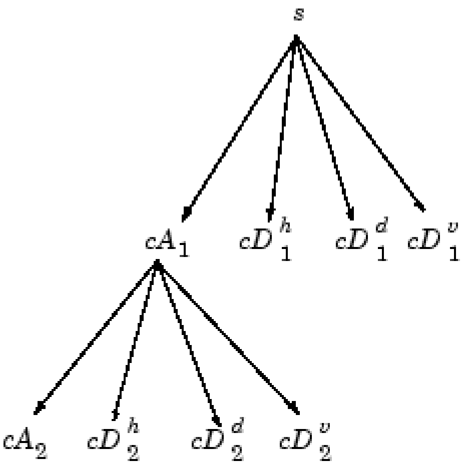
where

- $\boxed{2\downarrow 1}$ — Downsample columns: keep the even-indexed columns.
- $\boxed{1\downarrow 2}$ — Downsample rows: keep the even-indexed rows.
- $\overset{rows}{\boxed{X}}$ — Convolve with filter *X* the rows of the entry.
- $\overset{columns}{\boxed{X}}$ — Convolve with filter *X* the columns of the entry.

and

Initialization: $cA_0 = s$.

So, for $J = 2$, the two-dimensional wavelet tree has the form



References

[1] Daubechies, Ingrid. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics 61. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S.G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11, no. 7 (July 1989): 674–93. https://doi.org/10.1109/34.192463.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.
- The decomposition level, `N` must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`dwt2` | `waveinfo` | `waverec2` | `wfilters` | `wmaxlev`

Introduced before R2006a

wavedec3

3-D wavelet decomposition

Syntax

```
wdec = wavedec3(x,n,wname)
wdec = wavedec3(x,n,wname,'mode',extmode)
wdec = wavedec3(x,n,{LoD,HiD,LoR,HiR})
```

Description

`wdec = wavedec3(x,n,wname)` returns the wavelet decomposition of the 3-D array `x` at level `n`, using the wavelet specified by `wname`. `wavedec3` uses the default extension mode `'sym'`.

`wdec = wavedec3(x,n,wname,'mode',extmode)` uses the specified extension mode `extmode`.

`wdec = wavedec3(x,n,{LoD,HiD,LoR,HiR})` uses the specified decomposition and reconstruction filters `LoD,HiD` and `LoR,HiR`, respectively.

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');
```

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
```

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');
```

Input Arguments

x — Input data

3-D array

Input data, specified as a 3-D array.

Data Types: `double`

n — Decomposition level

positive integer

Decomposition level, specified as a positive integer. `wavedec3` does not enforce a maximum level restriction. See `wmaxlev`.

Data Types: `double`

wname — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar.

Note `wavedec3` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

extmode — Extension mode

`'zpd'` | `'sp0'` | `'spd'` | ...

Extension mode used when performing the wavelet decomposition, specified as one of the following:

mode	DWT Extension Mode
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1

mode	DWT Extension Mode
'sym' or 'symh'	Symmetric extension (half point): boundary value symmetric replication
'symw'	Symmetric extension (whole point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric extension (half point): boundary value antisymmetric replication
'asymw'	Antisymmetric extension (whole point): boundary value antisymmetric replication
'ppd', 'per'	Periodized extension If the signal length is odd and mode is 'per', an extra sample equal to the last value is added to the right and the extension is performed in 'ppd' mode. If the signal length is even, 'per' is equivalent to 'ppd'. This rule also applies to images.

The global variable managed by dwtmode specifies the default extension mode. See dwtmode for extension mode descriptions.

LoD,HiD — Wavelet decomposition filters
even-length real-valued vectors

Wavelet decomposition filters associated with an orthogonal or biorthogonal wavelet, specified as even-length real-valued vectors. LoD is the lowpass decomposition filter, and HiD is the highpass decomposition filter. See wfilters for details.

LoR,HiR — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters associated with an orthogonal or biorthogonal wavelet, specified as even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter. See wfilters for details.

Output Arguments

wdec — Wavelet output decomposition
structure

Wavelet output decomposition, returned as a structure with the following fields:

sizeINI — Input data size
vector

Input data size, returned as a 1-by-3 vector.

level — Level of the decomposition
integer

Level of the decomposition, returned as an integer.

mode — Name of the wavelet transform extension mode
character vector

Name of the wavelet transform extension mode, returned as a character vector.

`filters` — Wavelet filters

structure

Wavelet filters used for the decomposition, returned as a structure with the following fields:

- `LoD` — lowpass decomposition filter
- `HiD` — highpass decomposition filter
- `LoR` — lowpass decomposition filter
- `HiR` — highpass decomposition filter

dec — Decomposition coefficients

cell array

Decomposition coefficients, returned as an *N*-by-1 cell array, where *N* equals 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 sequence of letters gives 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

matrix

Successive sizes of the decomposition components, returned as an `n`+1-by-2 matrix.

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)
[psi,xval] = wavefun(wname,iter)

[ ___ ] = wavefun(wname,A,B)
[ ___ ] = wavefun(wname,0)
[ ___ ] = wavefun(wname,8,0)
[ ___ ] = wavefun(wname)
[ ___ ] = wavefun(wname,8)
```

Description

`[phi,psi,xval] = wavefun(wname,iter)` returns `psi` and `phi`, approximations of the wavelet and scaling functions, respectively, associated with the orthogonal wavelet `wname`, or the Meyer wavelet. The approximations are evaluated on the grid points `xval`. The positive integer `iter` specifies the number of iterations computed.

`[phi1,psi1,phi2,psi2,xval] = wavefun(wname,iter)` returns approximations of the wavelet and scaling functions associated with the biorthogonal wavelet `wname`. The wavelet and scaling function approximations `psi1` and `phi1`, respectively, are for decomposition. The wavelet and scaling function approximations `psi2` and `phi2`, respectively, are for reconstruction.

`[psi,xval] = wavefun(wname,iter)` returns the wavelet approximation `psi` for those wavelets that do not have an associated scaling function, such as Morlet, Mexican Hat, Gaussian derivatives wavelets, or complex wavelets.

`[ ___ ] = wavefun(wname,A,B)` plots the wavelet and scaling function approximations generated using `max(A,B)` iterations. The output arguments are optional.

`[ ___ ] = wavefun(wname,0)` is equivalent to `[ ___ ] = wavefun(wname,8,0)`.

`[ ___ ] = wavefun(wname)` is equivalent to `[ ___ ] = wavefun(wname,8)`.

Examples

Wavelet Approximations

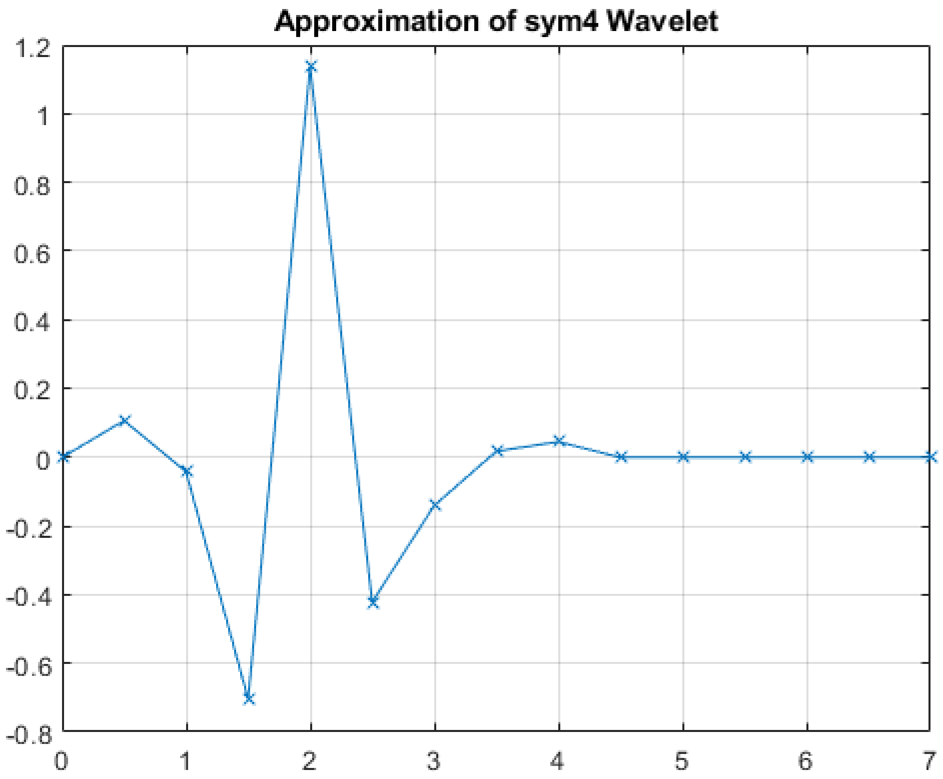
This example shows how the number of iterations affects the piecewise approximation of the specified wavelet.

Specify the number of iterations and the wavelet name.

```
wname = 'sym4';
itr = 10;
```

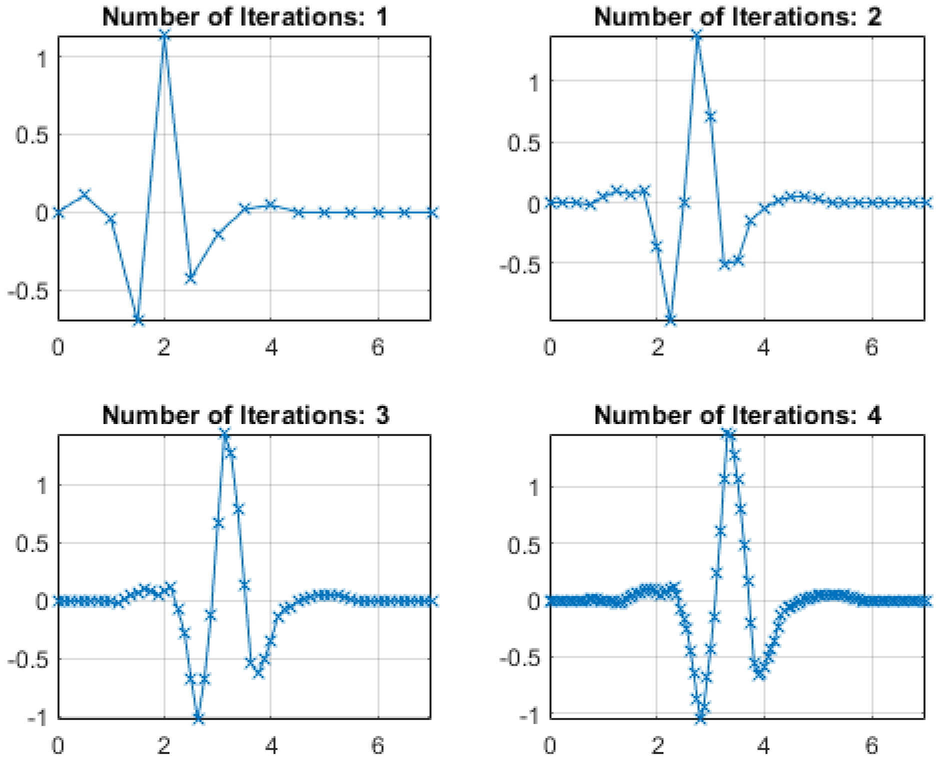
Plot the piecewise approximation of the wavelet generated after one iteration.

```
[~,psi,xval] = wavefun(wname,1);
plot(xval,psi,'x-')
grid on
title(['Approximation of ',wname,' Wavelet'])
```



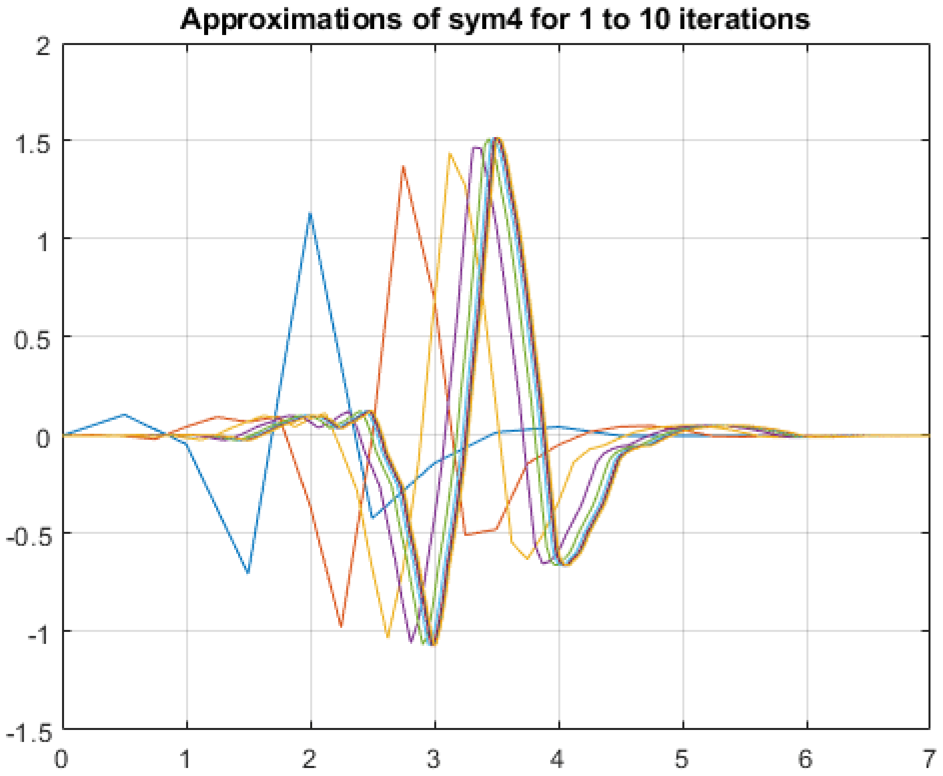
Vary the number of iterations from one through four and plot the approximations. Observe that as the number of iterations grows, so do the number of sample points.

```
figure
for k=1:4
    [~,psi,xval] = wavefun(wname,k);
    subplot(2,2,k)
    plot(xval,psi,'x-')
    axis tight
    grid on
    title(['Number of Iterations: ',num2str(k)])
end
```



Now vary the number of iterations from one to the number specified by `itr`.

```
figure
for k=1:itr
    [~,psi,xval] = wavefun(wname,k);
    plot(xval,psi)
    hold on
end
grid on
title(['Approximations of ',wname,' for 1 to ',num2str(itr),' iterations'])
```



Approximations of Biorthogonal Wavelets

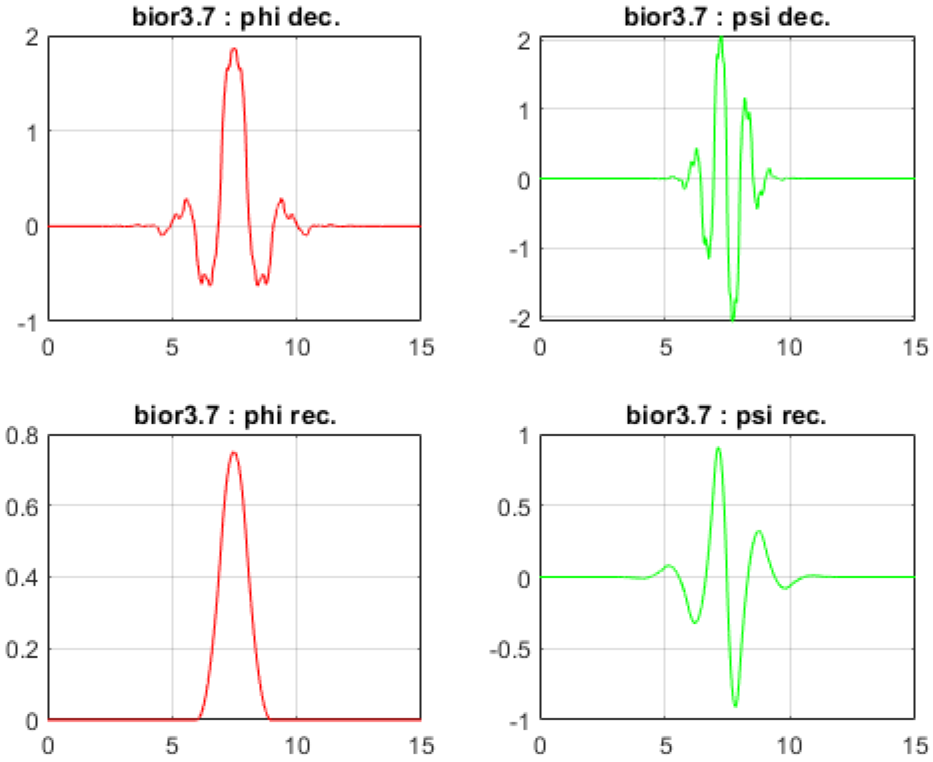
This example shows how to plot approximations of the scaling and wavelet functions associated with a biorthogonal wavelet.

Specify the name of a biorthogonal wavelet.

```
wname = 'bior3.7';
```

Plot approximations of the scaling and wavelet functions associated with the specified biorthogonal wavelet using the default number of iterations. Plot the approximations for both decomposition and reconstruction.

```
wavefun(wname,0);
```



Input Arguments

`wname` — Wavelet
character vector | string scalar

Wavelet, specified as a character vector or string scalar. See `waveinfo` for wavelets available.

`iter` — Number of iterations
8 (default) | positive integer

Number of iterations used to generate the wavelet and scaling function approximations, specified as a positive integer. Larger values of `iter` increase the refinement of the approximations.

`A,B` — Iteration
positive integers

Iteration, specified as a pair of positive integers. The number of iterations is equal to `max(A,B)`.

Output Arguments

`phi` — Scaling function approximation
real-valued vector

Scaling function approximation, returned as a vector.

psi — Wavelet approximation

real-valued vector | complex-valued vector

Wavelet approximation, returned as a vector. Depending on wname, psi can be a real- or complex-valued vector.

phi1,psi1 — Approximations of decomposition scaling and wavelet functions

real-valued vectors

Approximations of decomposition scaling and wavelet functions, respectively, associated with the biorthogonal wavelet wname, returned as real-valued vectors.

phi2,psi2 — Approximations of reconstruction scaling and wavelet functions

real-valued vectors

Approximations of reconstruction scaling and wavelet functions, respectively, associated with the biorthogonal wavelet wname, returned as real-valued vectors.

xval — Grid points

real-valued vector

Grid points where the wavelet and scaling function approximations are evaluated, returned as a real-valued vector.

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:

$$\|\phi - \eta\|_\infty \leq C.2^{-j\alpha}$$

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

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

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

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)`.

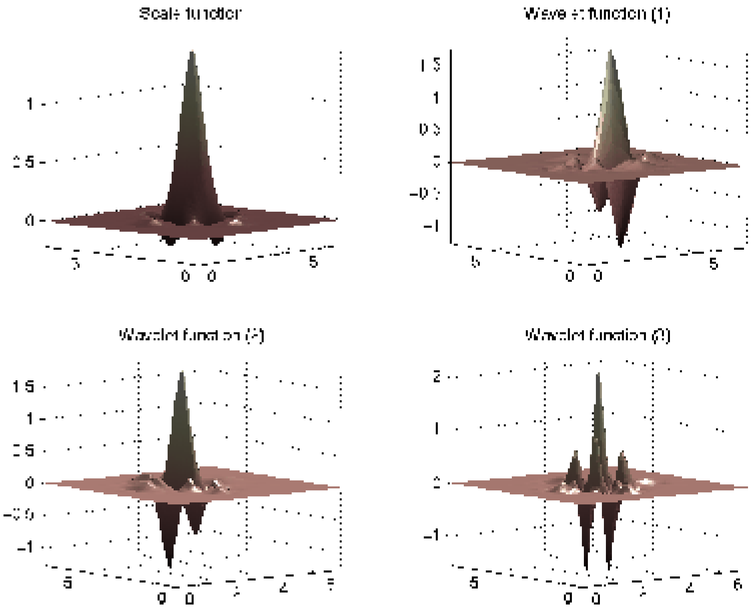
The output arguments are optional.

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. 202–213.

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
waveinfo(wname)
waveinfo('wsys')
```

Description

`waveinfo` provides information on all wavelets within the toolbox.

`waveinfo(wname)` provides information on the wavelet family associated with the wavelet short name `wname`.

`waveinfo('wsys')` provides information on wavelet packets.

Examples

Wavelet Family Information

Obtain information regarding the Daubechies wavelets.

```
waveinfo('db')
```

```
 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 a positive integer from 1 to 45.
    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.
```

Input Arguments

`wname` — Wavelet family short name

character vector | string scalar | `'haar'` | `'db'` | `'sym'` | `'coif'` | ...

Wavelet family short name, specified as a character vector or string scalar. The wavelet family short name can be for a user-defined wavelet (see `wavemngr` for more information) or one of the values listed here.

Wavelet Family Short Name	Wavelet Family Name
`'haar'`	Haar wavelet
`'db'`	Daubechies wavelets
`'sym'`	Symlets
`'coif'`	Coiflets
`'bior'`	Biorthogonal wavelets
`'fk'`	Fejér-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
`'shan'`	Shannon wavelets
`'fbsp'`	Frequency B-Spline wavelets
`'cmor'`	Complex Morlet wavelets

See Also

`wavemngr`

Introduced before R2006a

Wavelet Analyzer

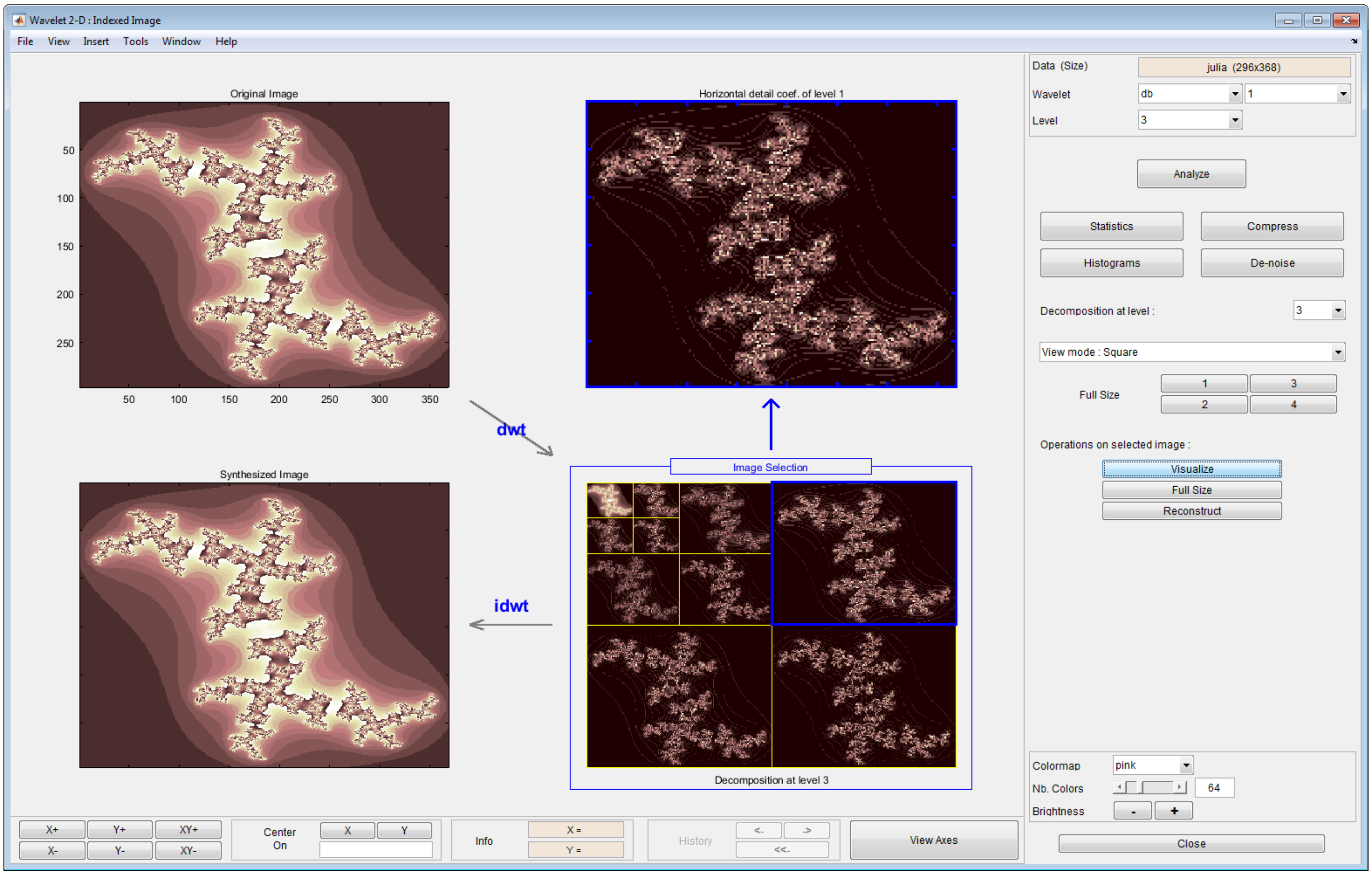
Analyze signals and images using wavelets

Note Some tools in **Wavelet Analyzer** will be removed in a future release. For a list of removed tools and recommended alternatives, see “Compatibility Considerations”.

Description

The **Wavelet Analyzer** app is an interactive tool for using wavelets to visualize and analyze signals and images. With the app, you can:

- Perform wavelet and wavelet packet analysis
- Denoise and compress signals and images
- Estimate density and regression
- Perform matching pursuit analysis
- Perform image fusion



Wavelet 2-D : Indexed Image

Data (Size): julia (296x368)

Wavelet: db 1

Level: 3

Decomposition at level : 3

View mode : Square

Colormap: pink

Nb. Colors: 64

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

- "Interactive 1-D Stationary Wavelet Transform Denoising"
- "Matching Pursuit Using Wavelet Analyzer App"

More About

Boundary Conditions

To change the way **Wavelet Analyzer** handles boundary conditions, use the `dwtmode` function.

Compatibility Considerations

Some tools in the Wavelet Analyzer app have been removed

The following tools in the **Wavelet Analyzer** app have been removed.

Tools	Replacement
Continuous Wavelet 1-D (Using FFT)	• To take the CWT of a single time series, use `cwt`. • To take the CWT of multiple time series, the recommended procedure is to precompute a CWT filter bank with `cwtfilterbank` and apply the filter bank to multiple time series. See "Using CWT Filter Bank on Multiple Time Series" on page 1-149. • To visualize the scalogram, use `cwt`. • To visualize wavelets in time and frequency, use `cwtfilterbank`.
New Wavelet for CWT	• To tune the generalized Morse wavelet to your needs, vary the time-bandwidth and symmetry parameters of `cwtfilterbank` or `cwt`. • To create a custom DWT filter bank, use `dwtfilterbank`. See "Add Quadrature Mirror and Biorthogonal Wavelet Filters".
Fractional Brownian Generation 1-D	To synthesize fractional Brownian motion, use `wfbm`.

Tools	Replacement
Wavelet Display, **Wavelet Packet Display**	• To visualize the analytic Morse, Morlet, and bump wavelets in time and frequency, use `cwtfilterbank`. • To visualize orthogonal and biorthogonal wavelets in time and frequency, use `dwtfilterbank`. • To visualize in time other wavelets such as the Meyer, Morlet, Gaussian, Mexican hat, and Shannon wavelets, use `wavefun`. • To display wavelet packets, use `wpfun`.
Signal Extension, **Image Extension**	To extend real-valued vectors or matrices, use `wextend`.

Additional tools in the Wavelet Analyzer app have been removed

The following tools in the **Wavelet Analyzer** app have been removed.

Tools	Replacement
Continuous Wavelet 1-D	To visualize the scalogram, use the new **Wavelet Time-Frequency Analyzer** app or the `cwt` function. With the app, you can select the wavelet to use as well as adjust Morse wavelet parameters. The app also supports single-variable regularly sampled timetables and real- or complex-valued single- or double-precision data.
Complex Continuous Wavelet 1-D	Use the new **Wavelet Time-Frequency Analyzer** app or the `cwt` function. With the app, you can also export the scalogram and generate a script to reproduce the wavelet analysis to your workspace.

See Also

Signal Multiresolution Analyzer | Wavelet Signal Denoiser | Wavelet Time-Frequency Analyzer

Topics

“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

Wavelet Families

Display the names of all available wavelet families.

```
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
Fejer-Korovkin            fk
===================================
```

Display the names of all available wavelets in each family.

```
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
===================================
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
===================================
```

Display all available wavelet families with their corresponding properties.

```
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 6 7 8
integer
cgauwavf
-5 5
------------------------
Family Name : Shannon
shan
5
1-1.5 1-1 1-0.5 1-0.1 2-3 **
string
shanwavf
-20 20
------------------------
Family Name : Frequency B-Spline
fbsp
5
1-1-1.5 1-1-1 1-1-0.5 2-1-1 2-1-0.5 2-1-0.1 **
string
fbspwavf
-20 20
------------------------
Family Name : Complex Morlet
cmor
5
1-1.5 1-1 1-0.5 1-1 1-0.5 1-0.1 **
string
cmorwavf
-8 8
------------------------
Family Name : Fejer-Korovkin
fk
1
4 6 8 14 18 22
integer
fejerkorovkin

------------------------
```

See Also

`wavemngr`

Introduced in R2008a

wavelets

CWT filter bank time-domain wavelets

Syntax

```
psi = wavelets(fb)
[psi,t] = wavelets(fb)
```

Description

`psi = wavelets(fb)` returns the time-domain wavelets `psi` for the continuous wavelet transform (CWT) filter bank `fb`. The time-domain wavelets are centered at the origin.

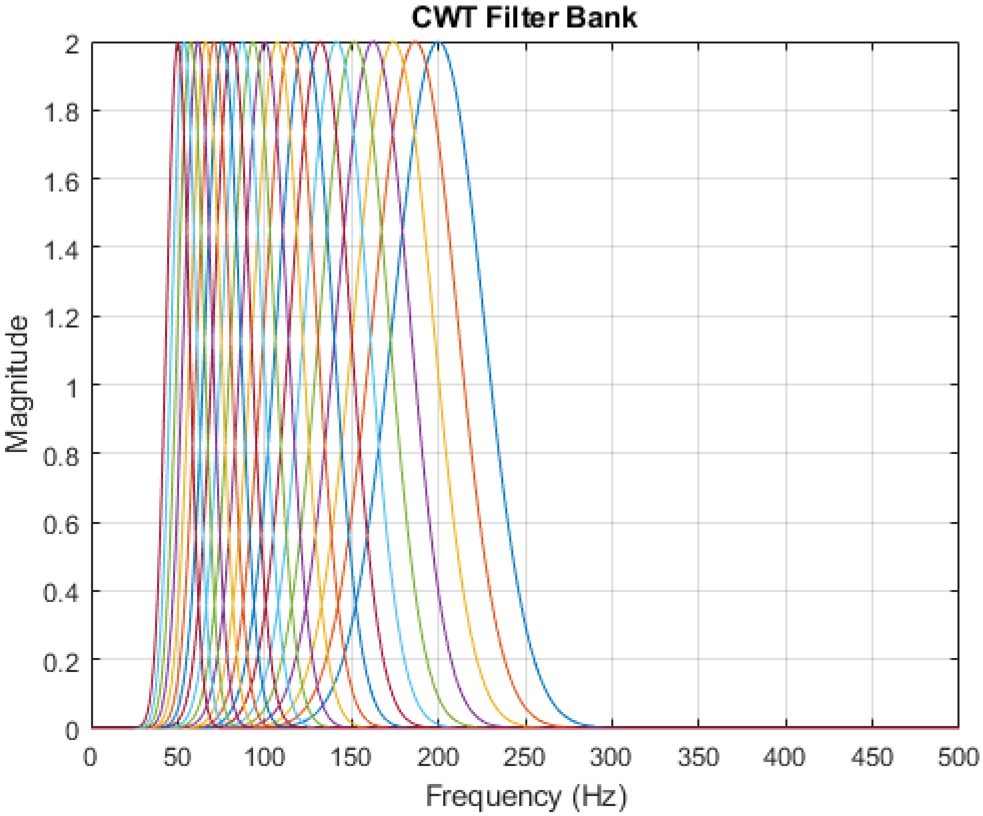
`[psi,t] = wavelets(fb)` returns the sampling instants `t` for the wavelets.

Examples

Filter Bank Time Domain Wavelets

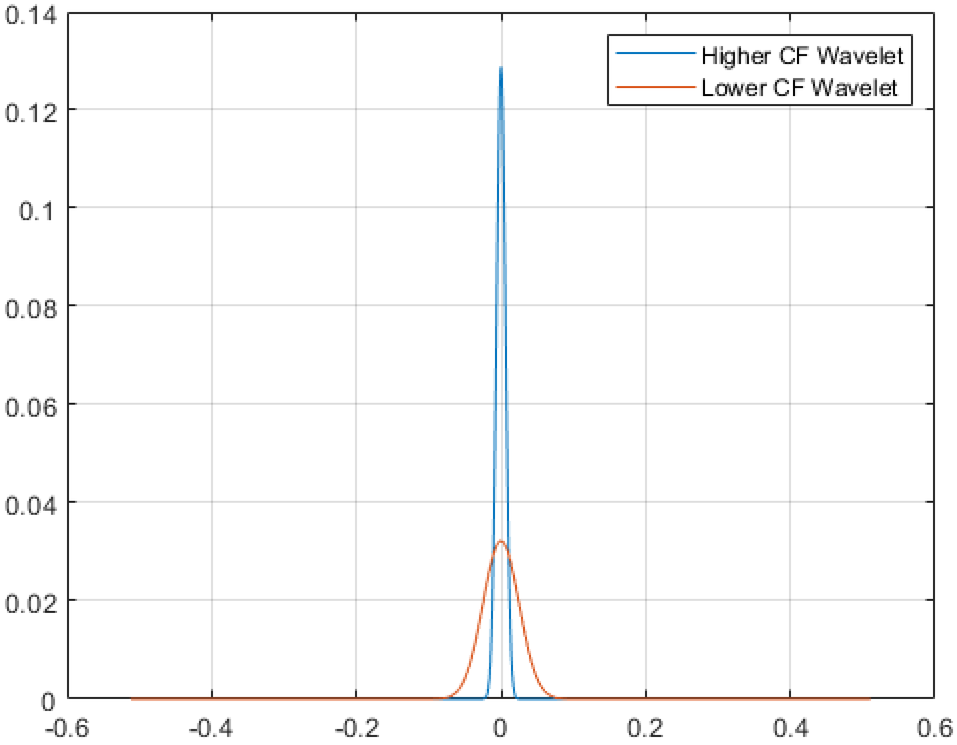
Create a continuous wavelet transform filter bank. Set the sampling frequency to 1000 Hz and the frequency limits to range from 50 Hz to 200 Hz. Plot the frequency response.

```
fb = cwtfilterbank('SamplingFrequency',1000,'FrequencyLimits',[50 200]);
freqz(fb)
```



Obtain the filter bank time-domain wavelets. Plot the magnitudes of the first and last wavelets contained in the output. The first wavelet corresponds to the wavelet filter with center frequency equal to 200 Hz, and the last wavelet corresponds to the wavelet filter with center frequency equal to 50 Hz.

```
[psi,t] = wavelets(fb);
figure
plot(t,abs(psi(1,:)))
hold on
plot(t,abs(psi(end,:)))
legend('Higher CF Wavelet','Lower CF Wavelet')
grid on
```



Input Arguments

`fb` — Continuous wavelet transform filter bank

`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

Output Arguments

`psi` — Time-domain wavelets

complex-valued matrix

Time-domain wavelets, returned as a *Ns*-by-*N* complex-valued matrix, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales) and *N* is the filter bank `SignalLength`. The wavelets are ordered in `psi` from the highest-frequency passband filter to the lowest-frequency passband filter.

`t` — Sampling instants

vector

Sampling instants of the time-domain wavelets, returned as a real-valued vector of length *N*, where *N* is the filter bank `SignalLength`. The data type of `t` is the same as the `SamplingPeriod`.

Extended Capabilities

C/C++ Code Generation

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

See Also

`cwtfilterbank` | `waveletsupport`

Introduced in R2018a

wavelets

DWT filter bank time-domain wavelets

Syntax

```
psi = wavelets(fb)
[psi,t] = wavelets(fb)
```

Description

`psi = wavelets(fb)` returns the time-domain and centered wavelets corresponding to the wavelet passband filters in the discrete wavelet transform (DWT) filter bank `fb`.

`[psi,t] = wavelets(fb)` returns the sampling instants `t`.

Examples

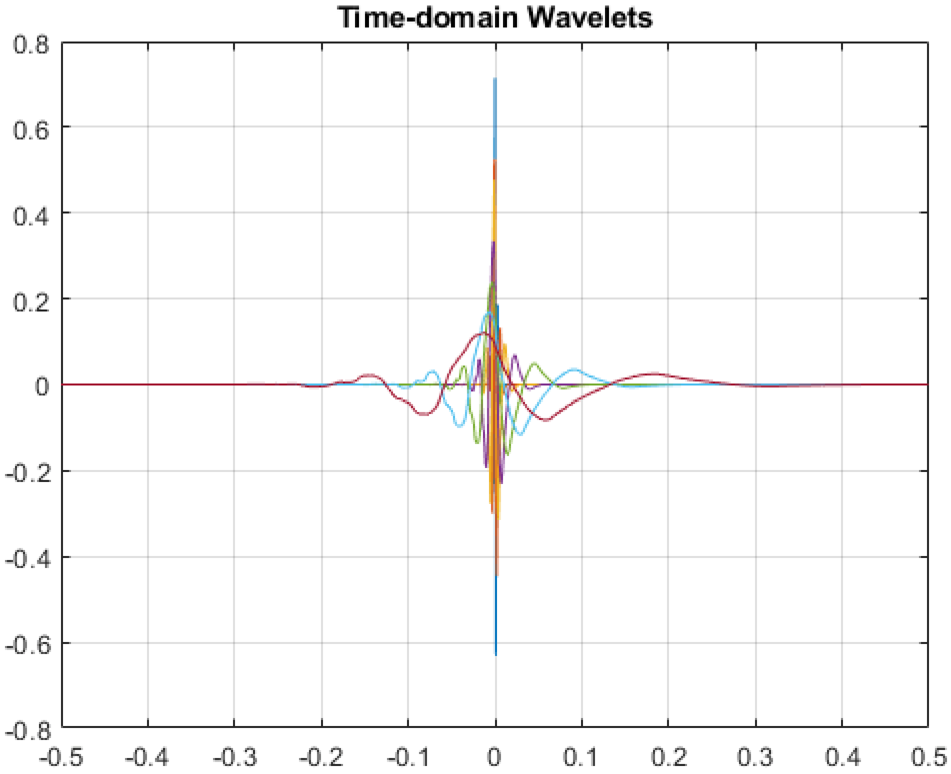
DWT Filter Bank Wavelets

Create a seven-level DWT filter bank with a signal length of 1000 samples, using the Daubechies `db2` wavelet and a sampling frequency of 1 kHz.

```
wv = "db4";
len = 1000;
lev = 7;
Fs = 1e3;
fb = dwtfilterbank('Wavelet',wv,'SignalLength',len,'Level',lev,'SamplingFrequency',Fs);
```

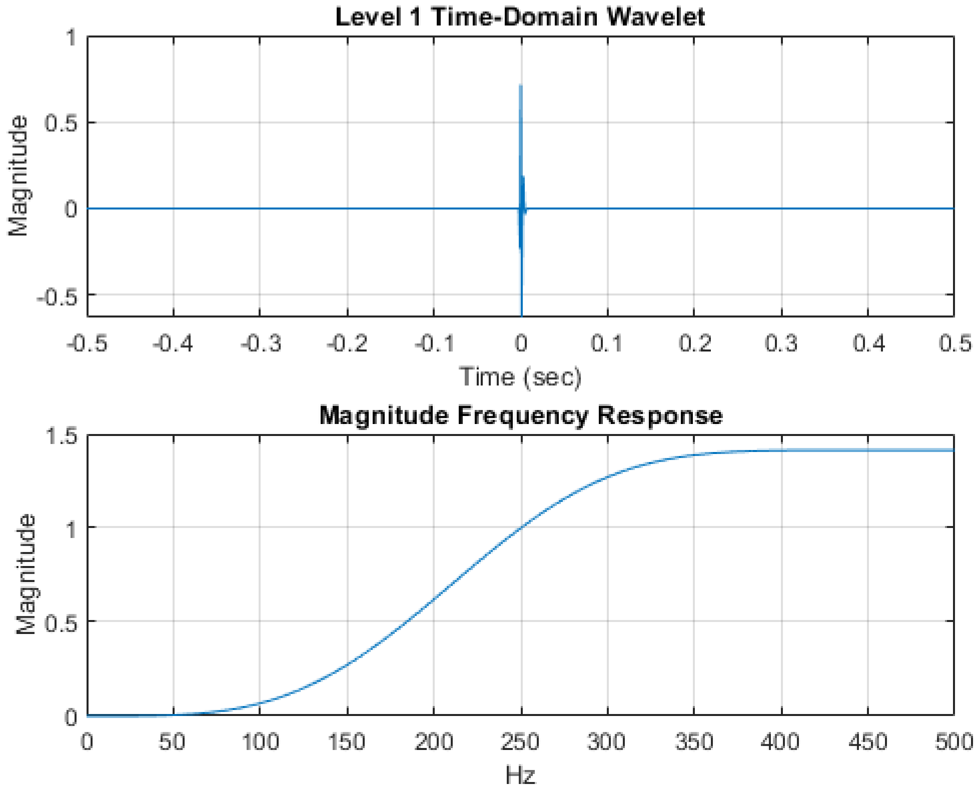
Plot the time-domain and centered wavelets corresponding to the wavelet bandpass filters.

```
[psi,t] = wavelets(fb);
plot(t,psi')
grid on
title('Time-domain Wavelets')
```



Plot the finest scale time-domain wavelet and the one-sided magnitude frequency response of the corresponding wavelet bandpass filter.

```
sc = 1;
[psidft,f] = freqz(fb);
subplot(2,1,1)
plot(t,psi(sc,:))
grid on
xlabel('Time (sec)')
ylabel('Magnitude')
title(['Level ',num2str(sc),' Time-Domain Wavelet'])
subplot(2,1,2)
plot(f(len/2:end),abs(psidft(sc,len/2:end)))
grid on
xlabel('Hz')
ylabel('Magnitude')
title('Magnitude Frequency Response')
```



Input Arguments

`fb` — Discrete wavelet transform filter bank

`dwtfilterbank` object

Discrete wavelet transform (DWT) filter bank, specified as a `dwtfilterbank` object.

Output Arguments

`psi` — Time-centered wavelets

real-valued matrix

Time-centered wavelets corresponding to the wavelet passband filters, returned as an *L*-by-*N* matrix, where *L* is the filter bank `Level` and *N* is the `SignalLength`. The wavelets are ordered in `psi` from the finest scale resolution to the coarsest scale resolution.

`t` — Sampling instants

real-valued vector

Sampling instants, returned as a real-valued vector `t` of length *N*, where *N* is the filter bank `SignalLength`. Sampling instants lie in the interval $[-\frac{1}{2}NDT,\ \frac{1}{2}NDT)$, where *DT* is the filter bank sampling period (reciprocal of the filter bank sampling frequency).

See Also

`dwtfilterbank` | `scalingfunctions` | `freqz`

Introduced in R2018a

waveletScattering

Wavelet time scattering

Description

Use the `waveletScattering` object to create a network for a wavelet time scattering decomposition using the Gabor (analytic Morlet) wavelet. The network uses wavelets and a lowpass scaling function to generate low-variance representations of real-valued time series data. Wavelet time scattering yields representations insensitive to translations in the input signal without sacrificing class discriminability. You can use the representations as inputs to a classifier. You can specify the duration of translation invariance and the number of wavelet filters per octave. The scattering network also supports time × channel × batch (T×C×B) inputs.

Creation

Syntax

```
sf = waveletScattering
sf = waveletScattering(Name,Value)
```

Description

`sf = waveletScattering` creates a wavelet time scattering network with two filter banks. The first filter bank has a quality factor of eight wavelets per octave. The second filter bank has a quality factor of one wavelet per octave. By default, `waveletScattering` assumes a signal input length of 1024 samples. The scale invariance length is 512 samples. By default, `waveletScattering` uses periodic boundary conditions.

`sf = waveletScattering(Name,Value)` creates a network for wavelet scattering, `sf`, with properties specified by one or more `Name,Value` pair arguments. Properties can be specified in any order as `Name1,Value1,...,NameN,ValueN`. Enclose each property name in quotes.

Note With the exception of `OversamplingFactor`, after creation you cannot change a property value of an existing scattering network. For example, if you have a network `sf` with a `SignalLength` of 2000, you must create a second network `sf2` for a signal with 2001 samples. You cannot assign a different `SignalLength` to `sf`.

Properties

`SignalLength` — Signal length
`1024` (default) | positive integer ≥ 16

Signal length in samples, specified as a positive integer ≥ 16. If the input to the scattering network is a row vector, `SignalLength` must match the number of columns in the input data. If the input to the scattering network is a column vector, matrix, or 3-D array, `SignalLength` must match the number of rows in the data.

Data Types: `double`

`SamplingFrequency` — Sampling frequency

`1` (default) | positive scalar

Sampling frequency in hertz, specified as a positive scalar. If unspecified, frequencies are in cycles/sample and the Nyquist frequency is ½.

Data Types: `double`

`InvarianceScale` — Scattering transform invariance scale

one-half of `SignalLength` (default) | positive scalar

Scattering transform invariance scale, specified as a positive scalar. `InvarianceScale` specifies the translation invariance of the scattering transform. If you do not specify `SamplingFrequency`, `InvarianceScale` is measured in samples. If you specify `SamplingFrequency`, `InvarianceScale` is measured in seconds. By default, `InvarianceScale` is one-half the `SignalLength` in samples.

`InvarianceScale` cannot exceed `SignalLength` in samples.

Example: `sf = waveletScattering('SignalLength',1000,'SamplingFrequency',200,'InvarianceScale',5)` has the largest possible `InvarianceScale`.

Data Types: `double`

`QualityFactors` — Scattering filter bank Q factors

`[8 1]` (default) | positive integer | vector of positive integers

Scattering filter bank Q factors, specified as a positive integer or a vector of positive integers. A filter bank Q factor is the number of wavelet filters per octave. Quality factors cannot exceed 32 and must be greater than or equal to 1.

If `QualityFactors` is specified as a vector, the elements of `QualityFactors` must be strictly decreasing.

Example: `sf = waveletScattering('QualityFactors',[8 2 1])` creates a wavelet scattering network with three filter banks.

Data Types: `double`

`Boundary` — Signal extension method

`'periodic'` (default) | `'reflection'`

Signal extension method to apply at the boundary:

- `'periodic'` — Extend signal periodically to length `2^ceil(log2(N))`, where `N` is the signal length.
- `'reflection'` — Extend signal by reflection to length `2^ceil(log2(2 N))`, where `N` is the signal length.

The signal is extended to match the length of the wavelet filters. The length of the filters are powers of two.

The signal extension method is for internal operations. Results are downsampled back onto the scale of the original signal before being returned.

Precision — Precision of scattering decomposition

'double' (default) | 'single'

Precision of scattering decomposition:

- 'double' — Double precision
- 'single' — Single precision

All calculations involving the wavelet scattering network are carried out in Precision. Wavelet scattering functions such as featureMatrix and filterbank return outputs such as filters in Precision. The precision of the output of the scatteringTransform function does not exceed the precision of sf.

Note

- If you construct a scattering network with double-precision filters and apply the network to single-precision data, the filters are cast internally to single-precision. Subsequent filtering is done with single precision until a new network is created regardless of input data type.
- Specifying Precision as 'single' at construction is especially useful in code generation because it eliminates the need to create an extra copy of the scattering filter banks.

OversamplingFactor — Oversampling factor

0 (default) | nonnegative integer | Inf

Oversampling factor, specified as a nonnegative integer or Inf. The factor specifies how much the scattering coefficients are oversampled with respect to the critically downsampled values. The factor is on a $\log_2$ scale. By default, OversamplingFactor is set to 0, which corresponds to critically downsampling the coefficients. You can use numCoefficients to determine the number of coefficients obtained for a scattering network. To obtain a fully undecimated scattering transform, set OversamplingFactor to Inf.

Setting OversamplingFactor to a value that would result in more coefficients than samples is equivalent to setting OversamplingFactor to Inf. Increasing the OversamplingFactor significantly increases the computational complexity and memory requirements of the scattering transform.

Example: If sf = waveletScattering('OversamplingFactor',2), the scattering transform returns 2^2 times as many coefficients for each scattering path with respect to the critically sampled number.

OptimizePath — Optimize scattering transform logical

false or 0 (default) | true or 1

Optimize scattering transform logical which determines whether the scattering transform reduces the number of scattering paths to compute based on a bandwidth consideration, specified as a numeric or logical 1 (true) or 0 (false).

If you specify OptimizePath as true, the scattering transform excludes scattering paths of order 2 and greater which do not satisfy the following criterion: The center frequency minus ½ the 3-dB bandwidth of the wavelet filter in the (i+1)th filter bank must overlap 0 (DC) plus ½ the 3-dB bandwidth of the wavelet filter in the ith filter bank. If this criterion is not satisfied, the higher-order path is excluded. Setting `OptimizePath` to true can significantly reduce the number of scattering paths and computational complexity of the scattering transform for most networks.

You can use `paths` to determine which and how many scattering paths are computed.

Object Functions

scatteringTransform	Wavelet 1-D scattering transform
featureMatrix	Scattering feature matrix
log	Natural logarithm of scattering transform
filterbank	Wavelet time scattering filter banks
littlewoodPaleySum	Littlewood-Paley sum
scattergram	Visualize scattering or scalogram coefficients
centerFrequencies	Wavelet scattering bandpass center frequencies
numorders	Number of scattering orders
numfilterbanks	Number of scattering filter banks
numCoefficients	Number of wavelet scattering coefficients
paths	Scattering network paths

Examples

Wavelet Time Scattering with Default Values

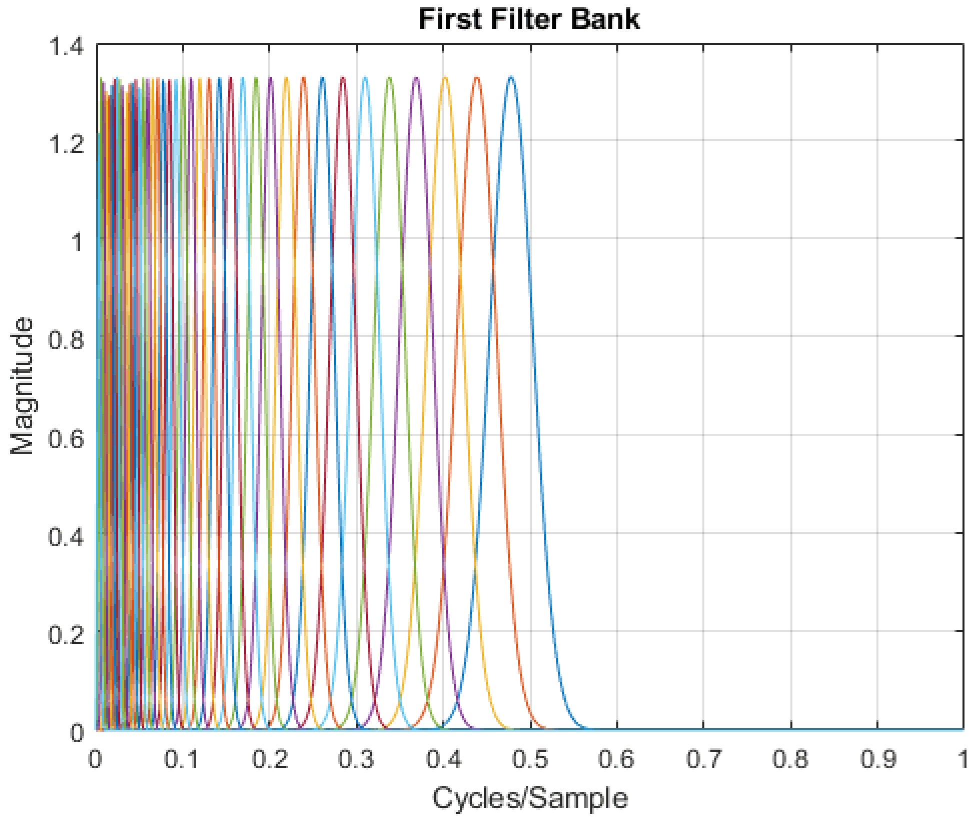
Create a wavelet time scattering network with default values.

```
sf = waveletScattering
```

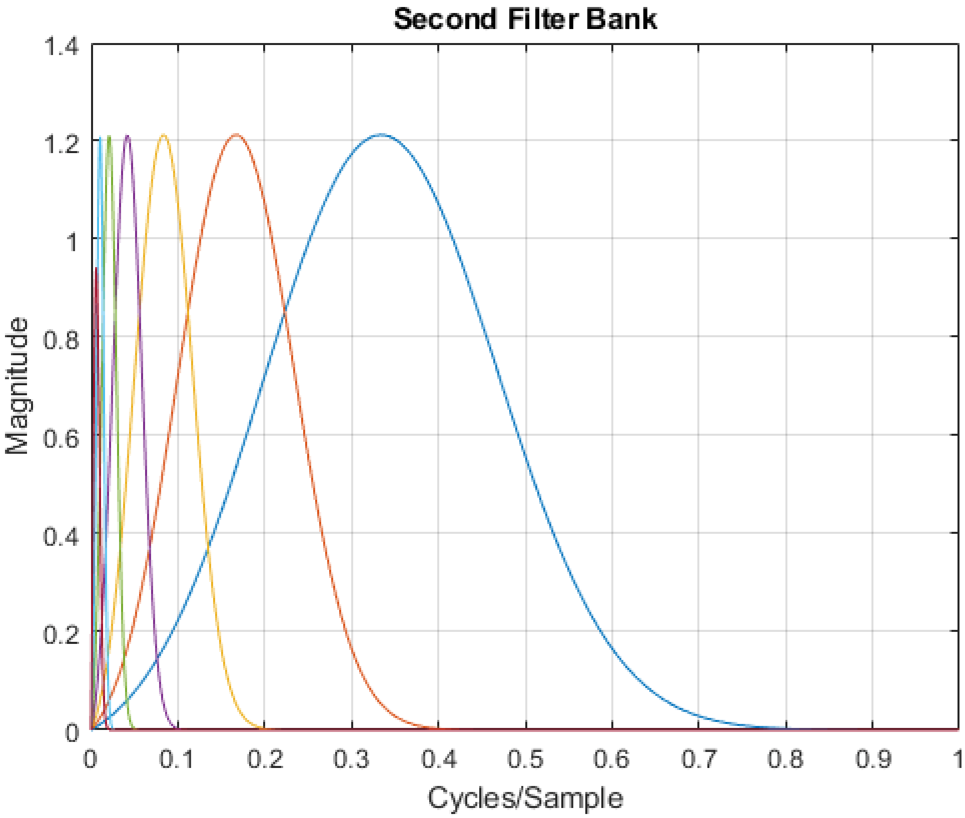
```
sf = 
  waveletScattering with properties:

          SignalLength: 1024
       InvarianceScale: 512
        QualityFactors: [8 1]
              Boundary: 'periodic'
     SamplingFrequency: 1
             Precision: 'double'
    OversamplingFactor: 0
          OptimizePath: 0
```

Plot the wavelet filters used in the first and second filter banks.

```
[filters,f] = filterbank(sf);
plot(f,filters{2}.psift)
title('First Filter Bank')
xlabel('Cycles/Sample')
ylabel('Magnitude')
grid on
```

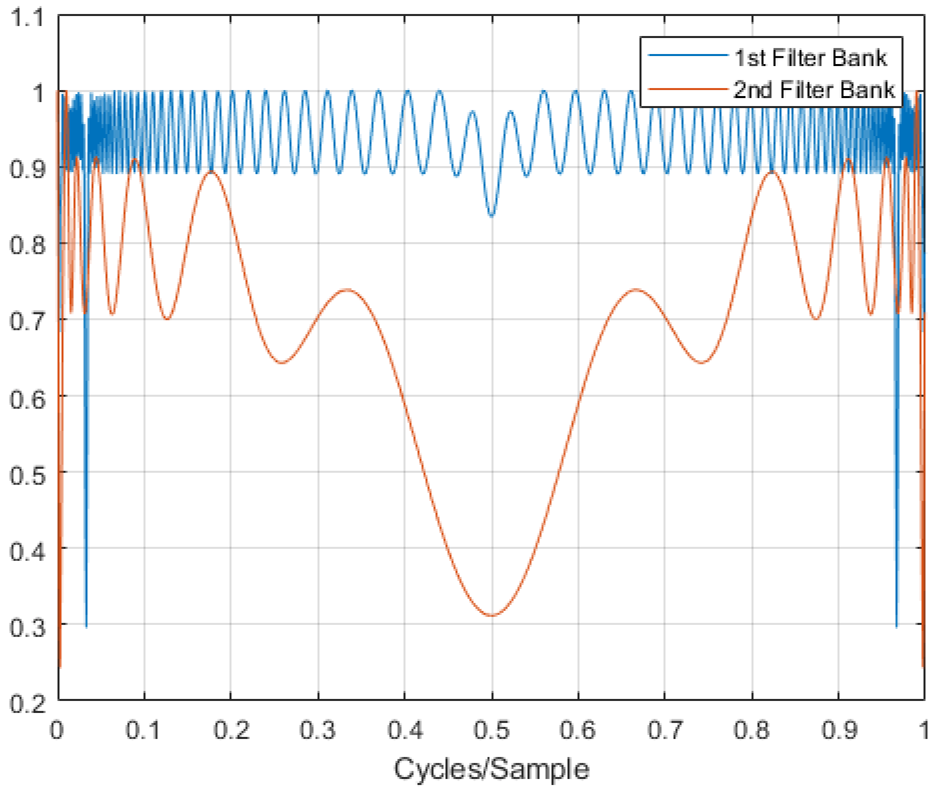


```
figure
plot(f,filters{3}.psift)
title('Second Filter Bank')
xlabel('Cycles/Sample')
ylabel('Magnitude')
grid on
```



Plot the Littlewood-Paley sums of the filter banks.

```
[lpsum,f] = littlewoodPaleySum(sf);
figure
plot(f,lpsum)
legend('1st Filter Bank','2nd Filter Bank')
xlabel('Cycles/Sample')
grid on
```



Apply Wavelet Time Scattering Network

This example shows how to create and apply a wavelet time scattering network with three filter banks to data.

Load in a data set. Create a scattering network with three filter banks that can be applied to the data.

```
load handel
disp(['Data Sampling Frequency: ',num2str(Fs),' Hz'])
```

```
Data Sampling Frequency: 8192 Hz
```

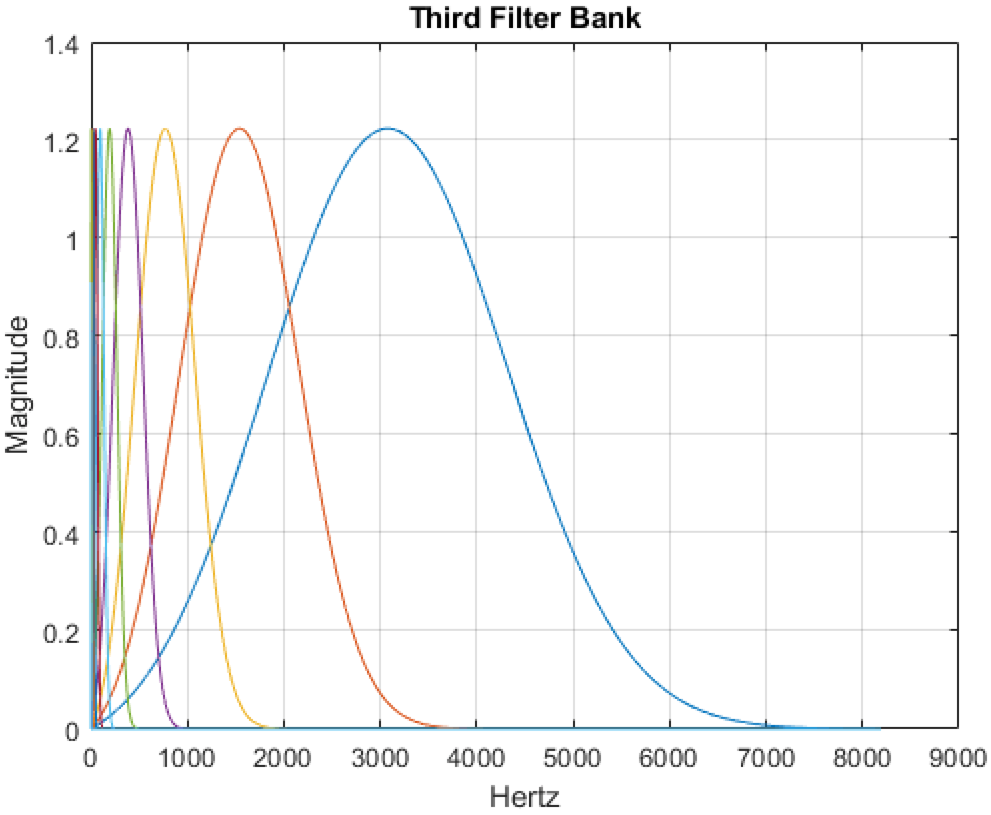
```
sf = waveletScattering('SignalLength',numel(y),...
    'SamplingFrequency',Fs,...
    'QualityFactors',[4 2 1])
```

```
sf = 
  waveletScattering with properties:

          SignalLength: 73113
       InvarianceScale: 4.4625
        QualityFactors: [4 2 1]
              Boundary: 'periodic'
     SamplingFrequency: 8192
             Precision: 'double'
```

```
    OversamplingFactor: 0
          OptimizePath: 0
```

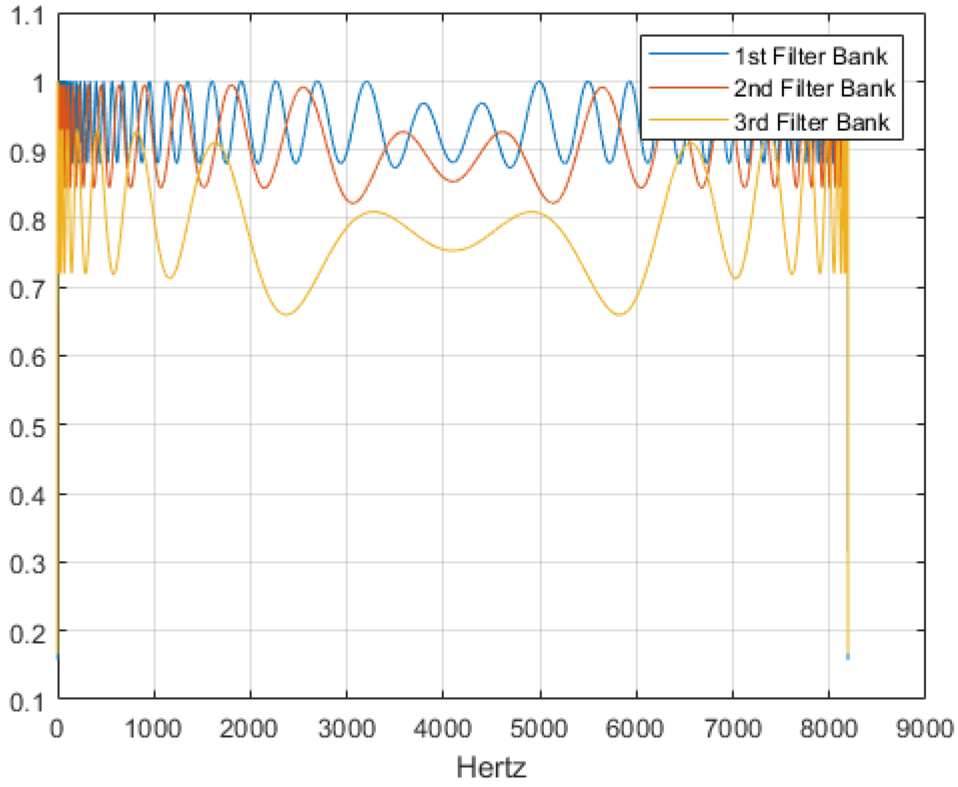
Inspect the network. Plot the wavelet filters used in the third filter bank.

```
[filters,f] = filterbank(sf);
plot(f,filters{4}.psift)
title('Third Filter Bank')
xlabel('Hertz')
ylabel('Magnitude')
grid on
```



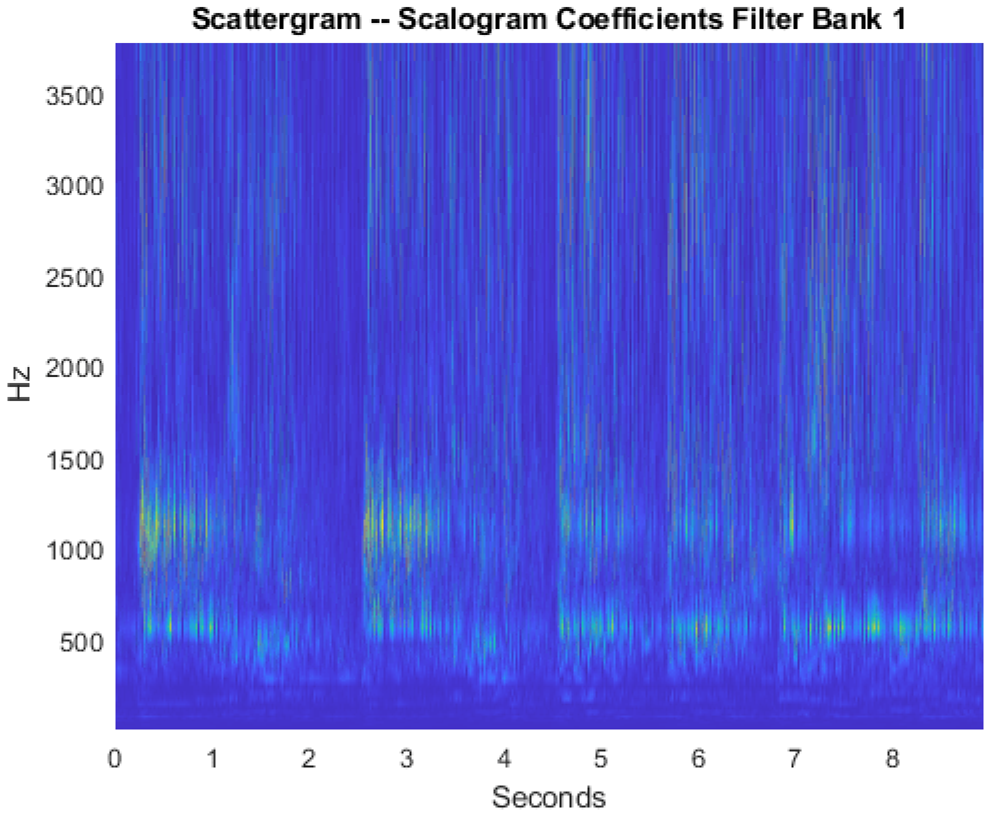
Plot the Littlewood-Paley sums of the three filter banks.

```
[lpsum,f] = littlewoodPaleySum(sf);
figure
plot(f,lpsum)
xlabel('Hertz')
grid on
legend('1st Filter Bank','2nd Filter Bank','3rd Filter Bank')
```



Calculate the wavelet 1-D scattering transform of the data for `sf`. Visualize the scattergram of the scalogram coefficients for the first filter bank.

```
[S,U] = scatteringTransform(sf,y);
figure
scattergram(sf,U,'FilterBank',1)
```



Compatibility Considerations

`waveletScattering` property `Decimate` has been removed
Errors starting in R2021a

The `waveletScattering` property `Decimate` has been removed. Use the property `OversamplingFactor` instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`Decimate` property	Errors	`OversamplingFactor`	• Replace all instances of `'Decimate',true` with `'OversamplingFactor',0`. • Replace all instances of `'Decimate',false` with `'OversamplingFactor',Inf`.

Highest wavelet center frequency is computed using geometric mean
Behavior changed in R2021a

Starting in R2021a, the highest wavelet center frequency is computed using the geometric mean. The method for determining how to space linearly those frequencies lower than the invariance scale has also changed. These changes improve the Littlewood-Paley sums of the resulting filter banks.

Center frequencies are logarithmically spaced from the highest frequency to the frequency that corresponds to the invariance scale. Starting in R2021a, depending on scattering network parameters such as the invariance scale, the *number* of filters you obtain may be different than in previous releases. The method for applying the filters to compute the scattering and scalogram coefficients has not changed.

References

[1] Andén, Joakim, and Stéphane Mallat. “Deep Scattering Spectrum.” *IEEE Transactions on Signal Processing* 62, no. 16 (August 2014): 4114–28. https://doi.org/10.1109/TSP.2014.2326991.

[2] Mallat, Stéphane. “Group Invariant Scattering.” *Communications on Pure and Applied Mathematics* 65, no. 10 (October 2012): 1331–98. https://doi.org/10.1002/cpa.21413.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- If specified, `'Precision'` and `'Boundary'` values must be `coder.Constant`.
- `QualityFactors` cannot be variable sized.
- `waveletScattering` is a handle class, so you must use an entry-point function.
- Filters are not cast when the network is compiled with single-precision data. The recommended workflow is to use the underlying data type of the signal when creating the network in the entry-point function.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Functions

`cwt` | `cwtfilterbank`

Topics

“Wavelet Scattering”
“Wavelet Scattering Invariance Scale and Oversampling”
“Music Genre Classification Using Wavelet Time Scattering”
“Wavelet Time Scattering for ECG Signal Classification”
“Wavelet Time Scattering Classification of Phonocardiogram Data”
“Wavelet Time Scattering with GPU Acceleration — Spoken Digit Recognition”

Introduced in R2018b

waveletScattering2

Wavelet image scattering

Description

Use the waveletScattering2 object to create a network for a wavelet image scattering decomposition using complex-valued 2-D Morlet wavelets.

Creation

Syntax

```
sf = waveletScattering2
sf = waveletScattering2(Name,Value)
```

Description

sf = waveletScattering2 creates a network for a wavelet image scattering decomposition with two complex-valued 2-D Morlet filter banks and isotropic scale invariance. Both filter banks have quality factors of one wavelet per octave. There are six rotations linearly spaced between 0 and π radians for each wavelet filter. By default, waveletScattering2 assumes an image input size of 128-by-128. The scale invariance is 64.

sf = waveletScattering2(Name,Value) creates a network for wavelet image scattering with properties specified by one or more Name,Value pair arguments. Properties can be specified in any order as Name1,Value1,...,NameN,ValueN. Enclose each property name in single quotes (' ') or double quotes (" ").

Note With the exceptions of OptimizePath and OversamplingFactor, you cannot change a property value of an existing scattering network. For example, if you create a network sf with ImageSize set to [256 256], you cannot assign a different ImageSize to sf.

Properties

ImageSize — Image size
[128 128] (default) | two-element integer-valued vector

Image size for wavelet image scattering network, specified as a two-element integer-valued vector [*numrows numcolumns*]. Images must be at least 10-by-10.

If your input is an RGB image, you do not have to specify the third dimension. waveletScattering2 only supports color images where the size of the third dimension is 3.

Example: sf = waveletScattering2('ImageSize',[100 200]) creates a network for 100-by-200 images and 100-by-200-by-3 color images.

InvarianceScale — Scattering transform invariance scale

64 (default) | positive scalar

Scattering transform invariance scale, specified as a positive scalar. InvarianceScale specifies the spatial support in the row and column dimensions of the scaling filter. InvarianceScale cannot exceed the minimum size of the row and column dimensions of the image.

By default, InvarianceScale is one-half the minimum of the row and column sizes of the image rounded to the nearest integer.

Example: `sf = waveletScattering2('ImageSize',[101 200])` creates a framework with InvarianceScale equal to 51.

NumRotations — Number of rotations per wavelet

[6 6] (default) | integer-valued vector

Number of rotations per wavelet per filter bank in the scattering network, specified as an integer-valued vector. Specify one integer less than or equal to 12 for each filter bank in the scattering network.

For each wavelet in each filter bank, there are NumRotations linearly spaced angles between 0 and π radians. The wavelet is rotated in a clockwise direction. The length of the vector specified in NumRotations must equal the length of the vector specified in QualityFactors.

Example: `sf = waveletScattering2('NumRotations',[7 5])` creates a network with seven rotations per wavelet in the first filter bank and five rotations per wavelet in the second filter bank.

Note The 2-D wavelet scattering network is constructed by rotating the 2-D Morlet wavelets in a clockwise direction. The opposite convention is used in the Image Processing Toolbox™. Creating a Gabor filter bank to apply to an image involves rotating the Gabor filter in a counter-clockwise direction. See “Slant Parameter” on page 1-502, and gabor in the Image Processing Toolbox.

QualityFactors — Scattering filter bank quality factors

[1 1] (default) | integer-valued vector

Scattering filter bank quality factors, specified as an integer-valued vector. The quality factor is the number of wavelet filters per octave. The number of wavelet filter banks in the scattering network is equal to the number of elements in QualityFactors. Valid quality factors are integers less than or equal to 4. If QualityFactors is specified as a vector, the elements of QualityFactors must be nonincreasing.

The length of the vector specified in QualityFactors must equal the length of the vector specified in NumRotations.

Example: `sf = waveletScattering2('QualityFactors',[2 1])`

Precision — Precision of scattering coefficients and filters

'single' (default) | 'double'

Precision of scattering coefficients and filters:

- 'single' — Single precision
- 'double' — Double precision

Note

- All calculations involving the wavelet scattering network are carried out in `Precision`.
- The precision of the output of the `scatteringTransform` function does not exceed the precision of the `waveletScattering2` object.

`OversamplingFactor` — Oversampling factor

`0` (default) | nonnegative integer | `Inf`

Oversampling factor, specified as a nonnegative integer or `Inf`. The factor specifies how much the image scattering coefficients are oversampled with respect to the critically downsampled values. The oversampling factor is on a $\log_2$ scale. For example, if `sf = waveletScattering2('OversamplingFactor',1)`, the scattering transform returns 2^1-by-2^1-by-*P* as many coefficients for each scattering path with respect to the critically sampled number. You can use `coefficientSize` to determine the number of coefficients obtained for a scattering network. By default, `OversamplingFactor` is set to `0`, which corresponds to critically downsampling the coefficients.

If you specify an oversampling factor that would result in an output image size larger than the input, the output size is truncated to the size of the input image. You can also specify the `OversamplingFactor` as `Inf`, which provides a fully undecimated scattering transform where each scattering path contains coefficient matrices equal in size to the input image.

Due to the computational complexity of the scattering transform, the recommended setting for the `OversamplingFactor` property is 0, 1, or 2. Values of 1 and 2 indicate a 2^1-by-2^1-by-*P* and a 2^2-by-2^2-by-*P* increase in the number of scattering coefficients per path, respectively.

Example: `sf.OversamplingFactor = 1` sets the `OversamplingFactor` property of an existing network to 1.

`OptimizePath` — Optimize scattering transform logical

`true` (default) | `false`

Optimize scattering transform logical, which determines whether the scattering transform reduces the number of scattering paths to compute based on a bandwidth consideration.

When `OptimizePath` is set to `true`, a scattering path is computed only if the bandwidth of the parent node overlaps significantly with the bandwidth of the child node. 'Significant' in this context is defined as follows: for a quality factor of 1, 1/2 the 3-dB bandwidth of the child node is subtracted from the child node's wavelet center frequency. If that value is less than the 3-dB bandwidth of the parent, the scattering path is computed. For quality factors greater than 1, significant overlap is defined to be an overlap between the center frequency of the child minus the child's 3-dB bandwidth. If that overlaps with the 3-dB bandwidth of the parent, the scattering path is computed.

You can use `paths` to determine which and how many scattering paths are computed. `OptimizePath` generally results in computational savings in the second and subsequent filter banks only when the quality factors are equal in each filter bank.

Example: `sf.OptimizePath = false` sets the `OptimizePath` property of an existing network to `false`.

Object Functions

scatteringTransform	Wavelet 2-D scattering transform
featureMatrix	Image scattering feature matrix
log	Natural logarithm of 2-D scattering transform
filterbank	Wavelet and scaling filters
littlewoodPaleySum	Littlewood-Paley sum
coefficientSize	Size of image scattering coefficients
numorders	Number of scattering orders
numfilterbanks	Number of scattering filter banks
paths	Scattering paths

Examples

Wavelet Image Scattering with Default Values

Create a wavelet image scattering network with default settings. The default image size is 128-by-128, and the default invariance scale is 64.

```
sf = waveletScattering2
```

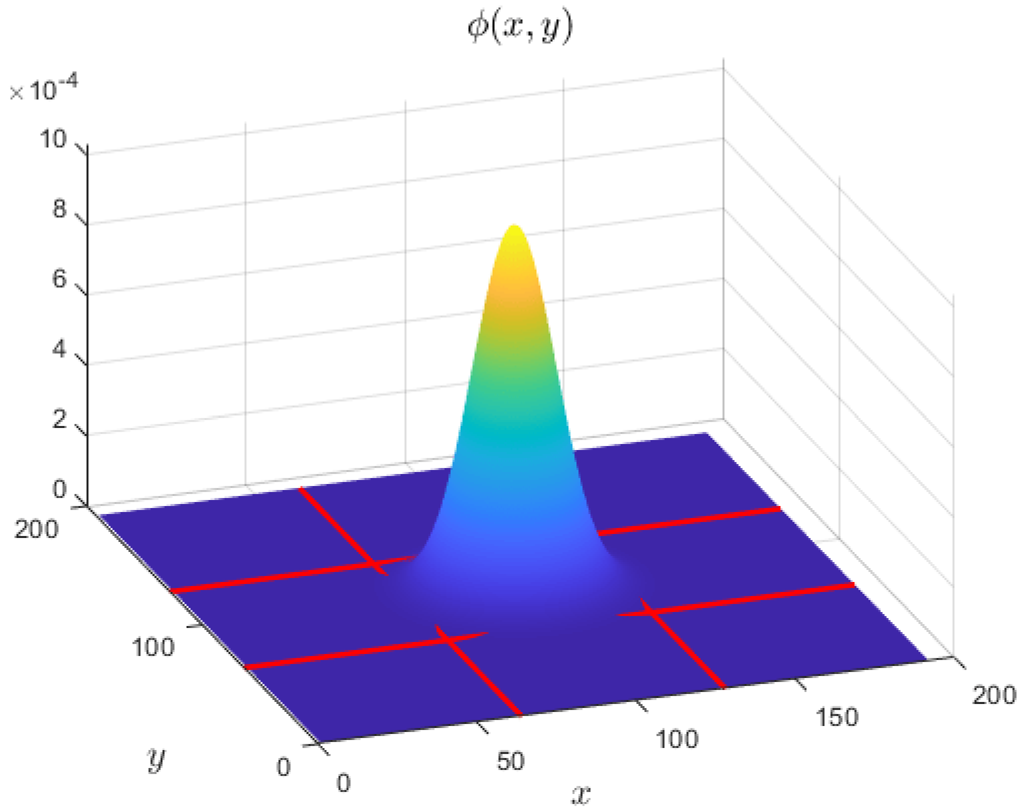
```
sf = 
  waveletScattering2 with properties:

             ImageSize: [128 128]
       InvarianceScale: 64
          NumRotations: [6 6]
        QualityFactors: [1 1]
             Precision: 'single'
    OversamplingFactor: 0
          OptimizePath: 1
```

Use the `filterbank` function to obtain the Fourier transform of the scaling function, the wavelet filters, and the center spatial frequencies of the wavelet filters.

```
[phif,psif,f] = filterbank(sf);
```

The invariance scale gives the width in the *x*- and *y*-directions of the 2-D Gaussian scaling function. To confirm the scaling function has the expected spatial width, first take the inverse Fourier transform of `phif`. Use the helper function `helperPlotPhiSurface` to plot the scaling function with the extent of the invariance scale in both *x* and *y* designated. The source code for `helperPlotPhiSurface` is provided in the appendix at the end of this example.

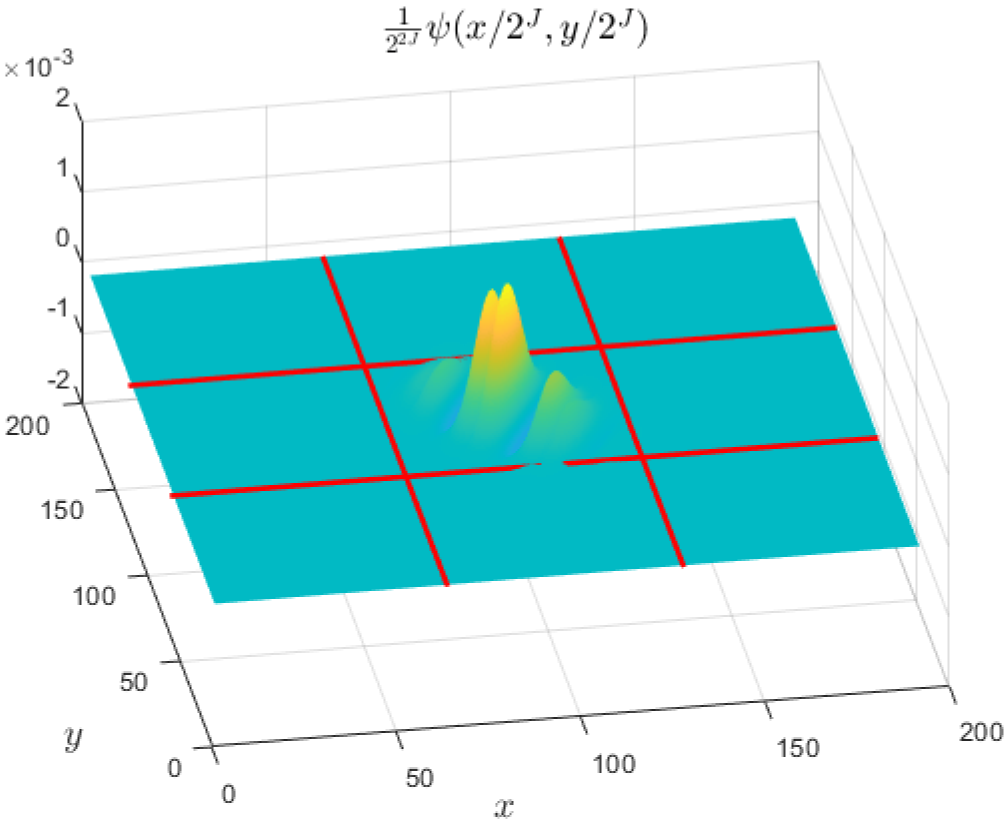
```
phi = ifftshift(ifft2(phif));
figure
helperPlotPhiSurface(sf,phi)
```



The scaling function is larger than 128-by-128 because it has been padded to avoid edge effects.

Extract the Fourier transform of the coarsest scale wavelet in the second filter bank and take its inverse Fourier transform. Use `helperPlotPsiSurface` to plot the real and imaginary parts of the wavelet and confirm the spatial extent of the coarsest scale wavelet does not exceed the invariance scale. Similar to the scaling function, the wavelet has been padded to avoid edge effects. The source code for `helperPlotPsiSurface` is provided in the appendix at the end of this example.

```
psiF = psif{2}(:,:,end);
psiL = ifftshift(ifft2(psiF));
figure
helperPlotPsiSurface(sf,psiL)
```



Appendix

The following helper functions are used in this example.

helperPlotPhiSurface

```
function helperPlotPhiSurface(scatFrame,data)
halfscale = scatFrame.InvarianceScale/2;
surf(data)
shading interp
view(-20,35)
Ysize = size(data,1);
Xsize = size(data,2);
Ycenter = Ysize/2;
Xcenter = Xsize/2;
hold on
plot([Xcenter-halfscale Xcenter-halfscale],[0 Ysize],'r','LineWidth',2);
plot([Xcenter+halfscale Xcenter+halfscale],[0 Ysize],'r','LineWidth',2);
plot([0 Xsize],[Ycenter-halfscale Ycenter-halfscale],'r','LineWidth',2);
plot([0 Xsize],[Ycenter+halfscale Ycenter+halfscale],'r','LineWidth',2);
title('$\phi(x,y)$','FontSize',14,'Interpreter','Latex');
xlabel('$x$','FontSize',14,'Interpreter','Latex')
ylabel('$y$','FontSize',14,'Interpreter','Latex')
end
```

helperPlotPsiSurface

```
function helperPlotPsiSurface(scatFrame,data)
halfscale = scatFrame.InvarianceScale/2;
Ysize = size(data,1);
Xsize = size(data,2);
Ycenter = Ysize/2;
Xcenter = Xsize/2;
surf(real(data))
shading interp
view(-5,13)
hold on
surf(imag(data))
shading interp
plot([Xcenter-halfscale Xcenter-halfscale],[0 Ysize],'r','LineWidth',2);
plot([Xcenter+halfscale Xcenter+halfscale],[0 Ysize],'r','LineWidth',2);
plot([0 Xsize],[Ycenter-halfscale Ycenter-halfscale],'r','LineWidth',2);
plot([0 Xsize],[Ycenter+halfscale Ycenter+halfscale],'r','LineWidth',2);
title('$\frac{1}{2^{2J}}\psi(x/2^J,y/2^J)$','FontSize',14,...
    'Interpreter','Latex');
xlabel('$x$','FontSize',14,'Interpreter','Latex')
ylabel('$y$','FontSize',14,'Interpreter','Latex')
view(-10,51)
end
```

References

[1] Bruna, J., and S. Mallat. "Invariant Scattering Convolution Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35, Number 8, 2013, pp. 1872–1886.

[2] Sifre, L., and S. Mallat. "Rigid-Motion Scattering for Texture Classification". arXiv preprint. 2014, pp. 1–19. https://arxiv.org/abs/1403.1687.

[3] Sifre, L., and S. Mallat. "Rotation, scaling and deformation invariant scattering for texture discrimination." *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp 1233–1240.

See Also

Objects
waveletScattering

Functions
cwtft2 | dddtree2 | wavedec2

Topics
“Wavelet Scattering”
“Wavelet Scattering Invariance Scale and Oversampling”
“Texture Classification with Wavelet Image Scattering”
“Digit Classification with Wavelet Scattering”

Introduced in R2019a

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:

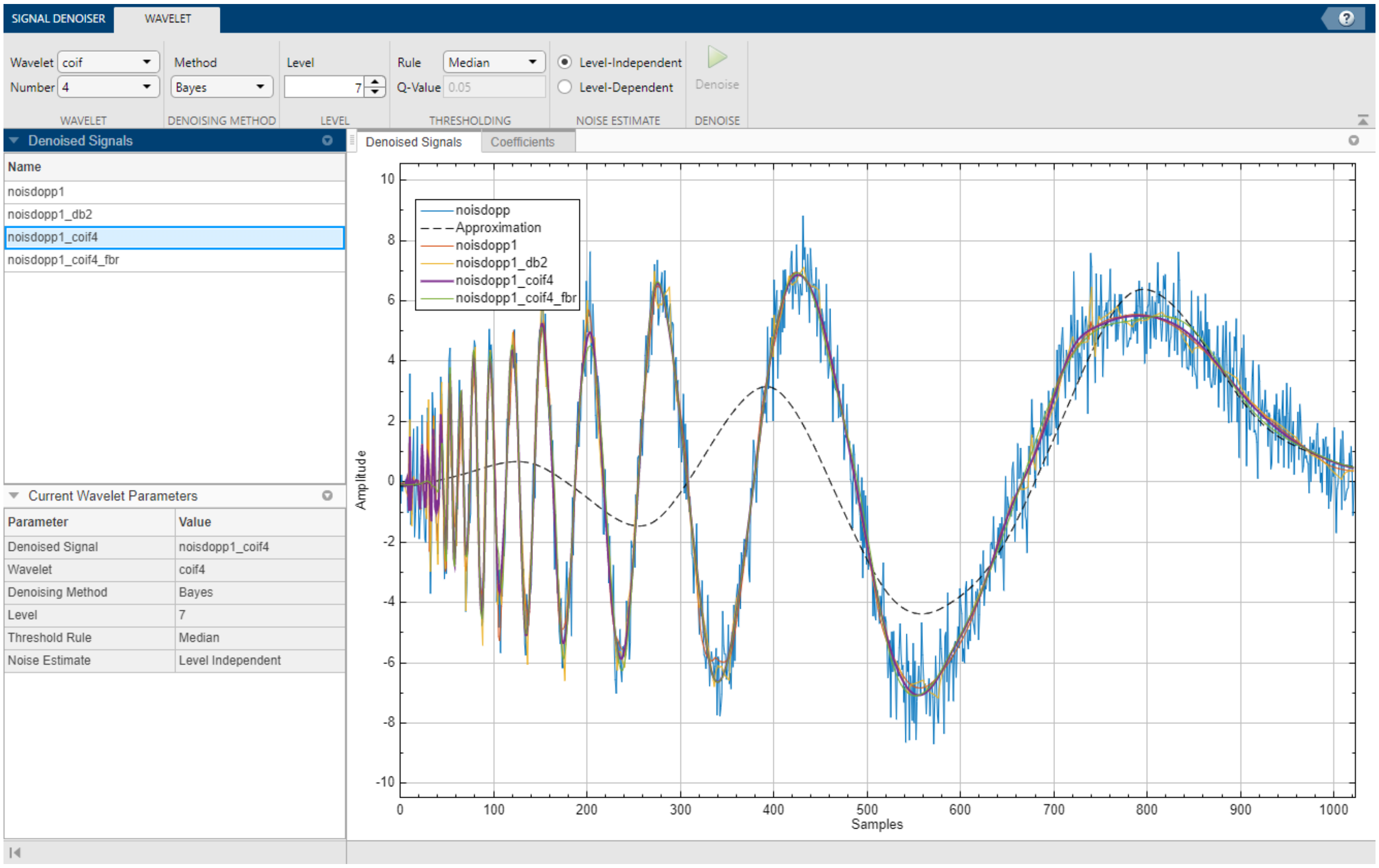
- Access all the signals in the MATLAB workspace.
- Easily adjust default parameters and apply different denoising techniques.
- Visualize and compare results.
- Export denoised signals to your workspace.
- Recreate the denoised signal in your workspace by generating a MATLAB script.

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 import 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.

For more information, see “Denoise a Signal with the Wavelet Signal Denoiser”.



Wavelet coif
Number 4
Method Bayes
Level 7
Rule Median
Q-Value 0.05
● Level-Independent
○ Level-Dependent

Denoised Signals

Name
noisdopp1
noisdopp1_db2
noisdopp1_coif4
noisdopp1_coif4_fbr

Current Wavelet Parameters

Parameter	Value
Denoised Signal	noisdopp1_coif4
Wavelet	coif4
Denoising Method	Bayes
Level	7
Threshold Rule	Median
Noise Estimate	Level Independent

Open the Wavelet Signal Denoiser App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `waveletSignalDenoiser`.

Examples

Denoise Signal Using Default Settings

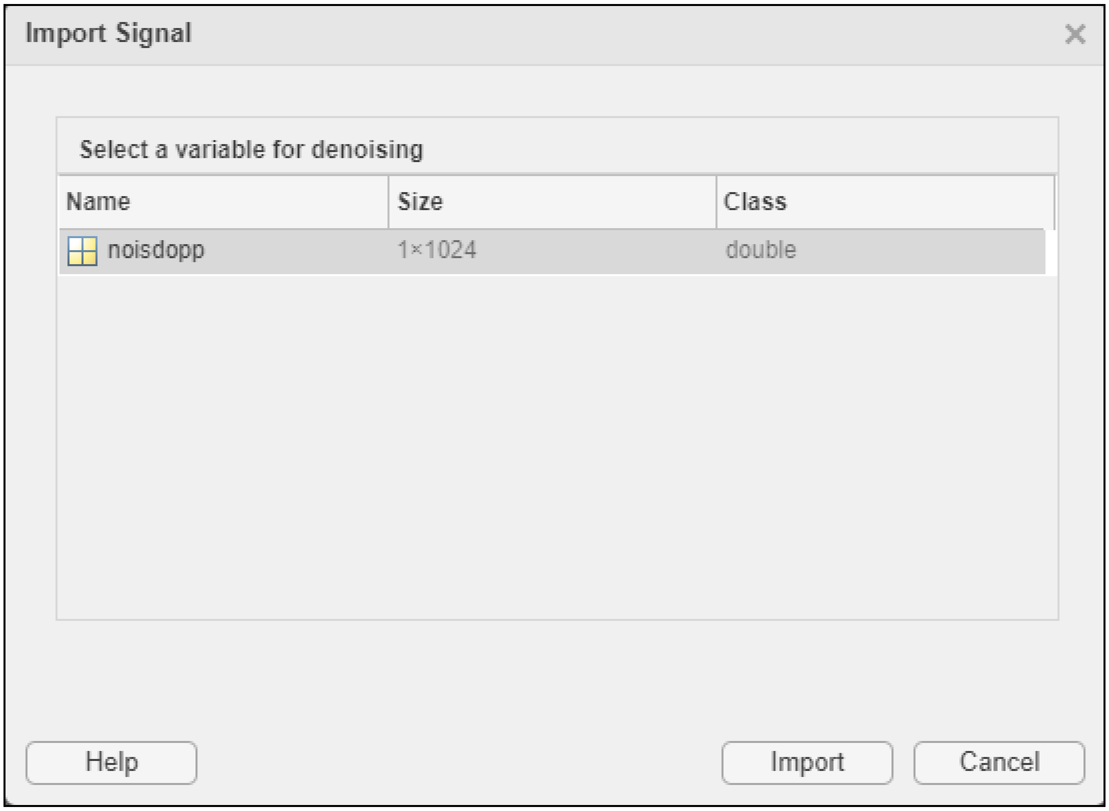
This example shows how to denoise a 1-D signal using the app default settings.

Load the noisy Doppler signal.

```
load noisdopp
```

Start the **Wavelet Signal Denoiser** app by choosing it from the **Apps** tab on the MATLAB® Toolstrip. You can also start the app by typing `waveletSignalDenoiser` at the MATLAB command prompt.

Load the noisy Doppler signal from the workspace into the app by clicking **Import** in the toolstrip. From the list of workspace variables that can be loaded into the app, select `noisdopp` and click **Import**.

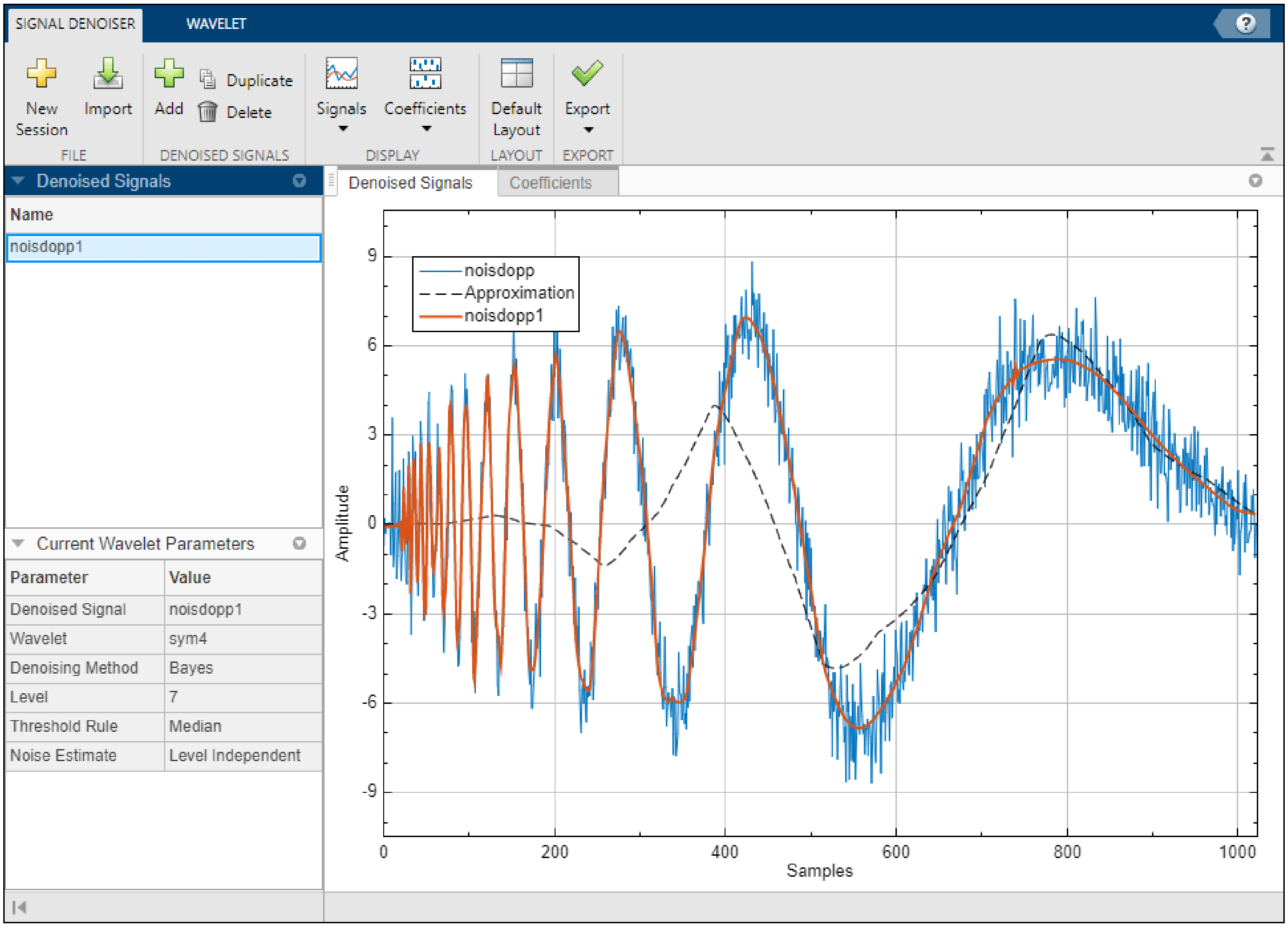


Import Signal

Select a variable for denoising

Name	Size	Class
noisdopp	1×1024	double

The app displays the original signal, `noisdopp`, the denoised signal, `noisdopp1`, and the coarse scale approximation, `Approximation`.



Parameter	Value
Denoised Signal	noisdopp1
Wavelet	sym4
Denoising Method	Bayes
Level	7
Threshold Rule	Median
Noise Estimate	Level Independent

To toggle what plots are visible, you can:

- Click **Signals ▼** in the toolstrip and use the drop-down menu to toggle the visibility of the original and approximation plots.
- Click individual signals in the plot legend.

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

For additional information, see `wdenoise`.

`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

For additional information, see `wdenoise`.

`Rule` — Thresholding rule

`Median` (default) | `Mean` | `Soft` | `Hard` | `James-Stein`

Thresholding rule to use. Valid options depend on the denoising method.

- 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`

For additional information, see `wdenoise`.

Programmatic Use

`waveletSignalDenoiser` opens the **Wavelet Signal Denoiser** app. Once the app initializes, import a signal to denoise by clicking **Import**.

`waveletSignalDenoiser(sig)` opens the **Wavelet Signal Denoiser** app, and imports and denoises `sig` using `wdenoise` with default settings. The app plots `sig`, the denoised signal, and its coarse scale approximation.

`sig` is a variable in the workspace.

- `sig` can be a 1-by-*N* or *N*-by-1 real-valued vector.
- `sig` is double precision.

Tips

To denoise more than one signal simultaneously, you can run multiple instances of the **Wavelet Signal Denoiser** app.

See Also

Apps
Signal Multiresolution Analyzer

Functions
`wdenoise` | `wdenoise2`

Topics
“Denoise a Signal with the Wavelet Signal Denoiser”

Introduced in R2017b

waveletsupport

CWT filter bank time supports

Syntax

```
spsi = waveletsupport(fb)
spsi = waveletsupport(fb,thresh)
```

Description

`spsi = waveletsupport(fb)` returns the wavelet time supports, defined as the time interval in which all of the wavelet's energy occurs. The default tolerance is 99.99% of the energy. The time supports are returned in the MATLAB table `spsi`. The wavelets are normalized to have unit energy.

`spsi = waveletsupport(fb,thresh)` specifies the threshold for the integrated energy. The time support of the wavelet is defined as the first instant the integrated energy exceeds `thresh` and the last instant the integrated energy is less than 1−`thresh`. If unspecified, `thresh` defaults to 10^{-4}.

Examples

Wavelet Filter Bank Time Supports

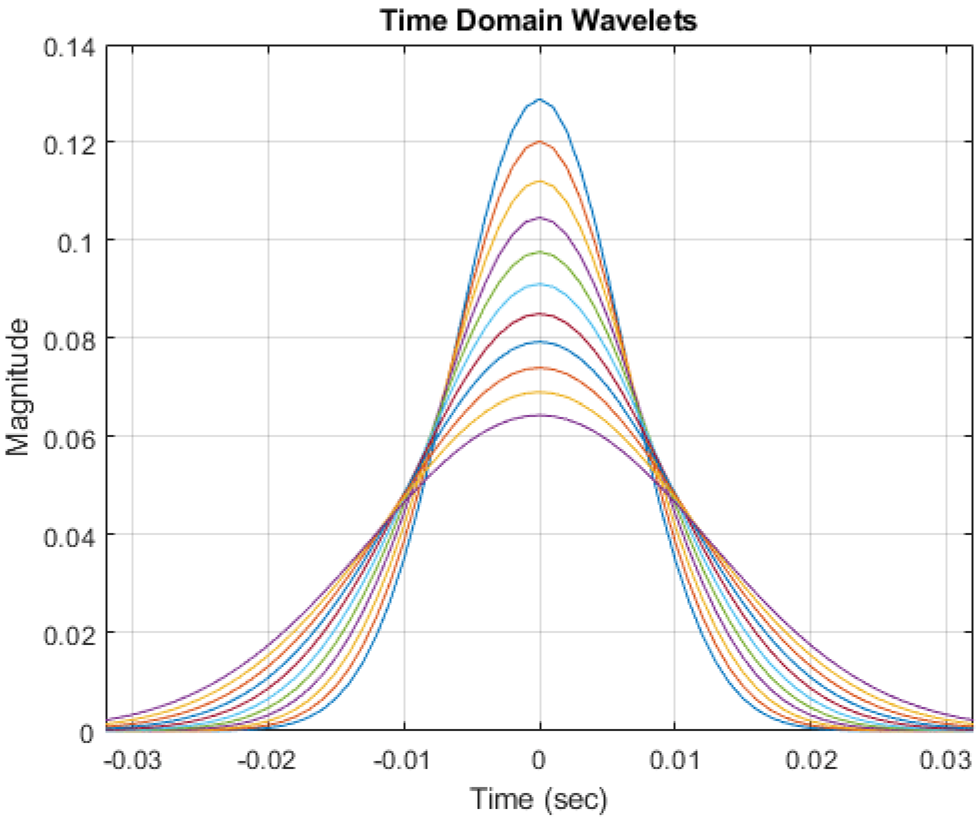
Create a continuous wavelet transform filter bank. Set the sampling frequency to 1000 Hz and the frequency limits to range from 100 Hz to 200 Hz. Obtain the time supports of the wavelets in the filter bank.

```
fb = cwtfilterbank('SamplingFrequency',1000,'FrequencyLimits',[100 200]);
spsi = waveletsupport(fb)
```

```
spsi=11×5 table
      CF      IsAnalytic    TimeSupport     Begin      End 
    ______    __________    ___________    ______    _____

       200    "Analytic"       0.032       -0.016    0.016
    186.61    "Analytic"       0.034       -0.017    0.017
    174.11    "Analytic"       0.038       -0.019    0.019
    162.45    "Analytic"        0.04        -0.02     0.02
    151.57    "Analytic"       0.042       -0.021    0.021
    141.42    "Analytic"       0.046       -0.023    0.023
    131.95    "Analytic"       0.048       -0.024    0.024
    123.11    "Analytic"       0.052       -0.026    0.026
    114.87    "Analytic"       0.056       -0.028    0.028
    107.18    "Analytic"        0.06        -0.03     0.03
       100    "Analytic"       0.064       -0.032    0.032
```

Obtain the time domain wavelets from the filter bank and plot their magnitudes. Use the table to set the minimum and maximum limits of the *x*-axis to the smallest `Begin` value and largest `End` value, respectively.

```
[psi,t] = wavelets(fb);
plot(t,abs(psi))
grid on
xlim([spsi.Begin(end) spsi.End(end)])
xlabel('Time (sec)')
ylabel('Magnitude')
title('Time Domain Wavelets')
```



Input Arguments

fb — Continuous wavelet transform filter bank

`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

`thresh` — Time support threshold

`10e–4` (default) | positive real number

Time support threshold for the wavelet, specified as a positive real number between 0 and 0.05. The time support of the wavelet is defined as the first instant the integrated energy of the wavelet exceeds `thresh` and the last instant the integrated energy is less than 1−`thresh`.

Data Types: `double`

Output Arguments

spsi — Wavelet time supports

table

Wavelet time supports, returned as an *Ns*-by-5 MATLAB table, where *Ns* is the number of wavelet bandpass frequencies (equal to the number of scales). The table has five variables:

CF — Wavelet center frequency

positive real number

Wavelet center frequency, returned as a positive real number.

Data Types: double

IsAnalytic — Wavelet designation

"Analytic" | "Nonanalytic"

Wavelet designation, returned as a string. Wavelets that do not decay to 5% of their peak value at the Nyquist frequency are not considered analytic. The time support information for those wavelets are returned as NaNs.

Data Types: string

TimeSupport — Wavelet time support

positive integer | NaN

Wavelet time support, returned in samples, seconds, or MATLAB durations. The units of TimeSupport depend on whether you specify SamplingFrequency or SamplingPeriod. If you specify a SamplingFrequency, the units are seconds. If you specify a SamplingPeriod, the units are the same as the SamplingPeriod. If no SamplingFrequency or SamplingPeriod is specified, the units are samples.

Data Types: double

Begin — Beginning of wavelet time support

integer

Beginning of wavelet time support, returned as an integer. Begin is defined as the first instant the wavelet integrated energy exceeds the default threshold, 10^{-4}. Begin has the same units as TimeSupport.

Data Types: double

End — End of wavelet time support

integer

End of wavelet time support, returned as an integer. End is defined as the last instant the wavelet integrated energy is less than $1 - 10^{-4}$. End has the same units as TimeSupport.

Data Types: double

Data Types: table

See Also

cwtfilterbank | wavelets

Introduced in R2018a

waveletsupport

DWT filter bank time supports

Syntax

```
spsi = waveletsupport(fb)
spsi = waveletsupport(fb,thresh)
[spsi,sphi] = waveletsupport(fb)
[spsi,sphi,tlow,thigh] = waveletsupport(fb)
```

Description

`spsi = waveletsupport(fb)` returns the wavelet time supports of the discrete wavelet transform (DWT) filter bank `fb`. The time supports are defined as the time interval in which all the wavelet energy occurs (> 99.99% of the energy for the default threshold).

`spsi = waveletsupport(fb,thresh)` specifies the threshold for the integrated energy. `thresh` is a positive real number in the interval 0 < `thresh` ≤ 0.05.

`[spsi,sphi] = waveletsupport(fb)` returns the scaling function time supports at all levels. `sphi` is a real-valued *L*-by-1 vector, where *L* is the number of levels in the DWT filter bank `fb`.

`[spsi,sphi,tlow,thigh] = waveletsupport(fb)` returns the instants the integrated energy in the wavelets and scaling functions exceed `thresh` in `tlow` and the last instant the integrated energy is less than 1 − `thresh` in `thigh`.

Examples

DWT Filter Bank Wavelet Time Supports

Find the time supports for a Haar wavelet filter bank.

```
fb = dwtfilterbank('Wavelet','haar','Level',8);
Spsi = waveletsupport(fb)
```

```
Spsi = 8×1

     2
     4
     8
    16
    32
    64
   128
   256
```

Input Arguments

fb — Discrete wavelet transform filter bank
dwtfilterbank object

Discrete wavelet transform (DWT) filter bank, specified as a dwtfilterbank object.

thresh — Threshold for the integrated energy
1e-6 (default) | positive scalar

Threshold for the integrated energy, specified as a positive scalar in the interval 0 < thresh ≤ 0.05. If unspecified, thresh defaults to 10^{-6}.

The percent energy contained in the time support is (1 − 2 × thresh) × 100. The time support of the wavelet is defined as the first instant the integrated energy exceeds thresh and the last instant it is less than 1-thresh. The wavelets are normalized to have unit energy for the computation.

Output Arguments

spsi — Wavelet time supports
real-valued column vector

Wavelet time supports, returned as a real-valued column vector of length *L*, where *L* is the number of levels in the DWT filter bank.

sphi — Scaling function time supports
real-valued column vector

Scaling function time supports, returned as a real-valued column vector of length *L*, where *L* is the number of levels in the DWT filter bank.

tlow — First instants
real-valued matrix

First instants the integrated energy in the wavelet and scaling functions exceed thresh, returned as real-valued *L*-by-2 matrix, where *L* is the number of levels in the filter bank. The first column of tlow contains the times for the wavelets. The second column of tlow contains the times for the scaling functions.

The difference between the first column of thigh and the first column of tlow plus one sampling period equals pspi. The difference between the second column of thigh and the second column of tlow plus one sampling period equals sphi.

thigh — Last instants
real-valued matrix

Last instants the integrated energy in the wavelet and scaling functions is less than 1−thresh, returned as real-valued *L*-by-2 matrix, where *L* is the number of levels in the filter bank. The first column of thigh contains the times for the wavelets. The second column of thigh contains the times for the scaling functions.

The difference between the first column of thigh and the first column of tlow plus one sampling period equals pspi. The difference between the second column of thigh and the second column of tlow plus one sampling period equals sphi.

See Also

`dwtfilterbank` | `scalingfunctions` | `wavelets`

Introduced in R2018a

wavemngr

Wavelet manager

Syntax

```
wavemngr('add',FN,FSN,WT,NUMS,FILE)
wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE)
wavemngr( ___ ,B)
```

```
wavemngr('del',WN)
```

```
wavemngr('restore')
wavemngr('restore',IN2)
```

```
out = wavemngr('read')
out = wavemngr('read',IN2)
out = wavemngr('read_asc')
```

Description

Use `wavemngr` to add, delete, restore, or read wavelets.

`wavemngr('add',FN,FSN,WT,NUMS,FILE)` adds a wavelet family to the toolbox. These parameters define the wavelet family:

- `FN` — Family name
- `FSN` — Family short name
- `WT` — Wavelet family type
- `NUMS` — Wavelet parameters
- `FILE` — Wavelet definition file

Note When you use `wavemngr` to add a wavelet family, three wavelet extension files are created in the current folder: the two ASCII files `wavelets.asc` and `wavelets.prv`, and the MAT-file `wavelets.inf`. If you add a new wavelet family, it is available in this folder only.

`wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE)` adds a wavelet family with parameter `NUMS` with input format type `TYPNUMS`.

`wavemngr( ___ ,B)` adds a wavelet family, where `B` specifies the effective support for the wavelets. The `B` input argument is valid only for wavelets of type `WT` = 3, 4, and 5. You can use this syntax with any of the previous syntaxes.

`wavemngr('del',WN)` deletes the wavelet family specified by `WN`.

`wavemngr('restore')` restores the previous `wavelets.asc` ASCII file

`wavemngr('restore',IN2)` restores the initial `wavelets.asc` ASCII file. Here `IN2` is a dummy argument.

`out = wavemngr('read')` returns all wavelet family names in a character array.

`out = wavemngr('read',IN2)` returns all wavelet names in a character array. Here `IN2` is a dummy argument.

`out = wavemngr('read_asc')` reads the `wavelets.asc` ASCII file and returns all wavelet 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

This example shows how to add new compactly supported orthogonal wavelets 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 a new family of orthogonal wavelets. You must define:

- Family Name: `Lemarie`
- Family Short Name: `lem`
- Type of wavelet: `1 (orth)`
- Wavelet numbers: `1 2 3 4 5`
- File driver: `lemwavf`

The source code for `lemwavf.m` is provided at the end of the example. The input argument of `lemwavf` is a character vector of the form `lem`*N*, where *N* = 1, 2, 3, 4, or 5.

```
wavemngr('add','Lemarie','lem',1,'1 2 3 4 5','lemwavf')
```

The ASCII file `wavelets.asc` is saved as `wavelets.prv`, then information defining the new family is added to `wavelets.asc`, 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. Confirm the new wavelet family is added.

```
wavemngr('read')
```

```
ans = 19x35 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             '
    'Lemarie           ->->lem            '
    '==================================='
```

Remove the added family. Regenerate the list of wavelet families.

```
wavemngr('del','Lemarie')
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             '
    '==================================='
```

Restore the previous ASCII file `wavelets.prv`, then build the MAT-file `wavelets.inf`. List the restored wavelets. Because `wavemngr` reads the ASCII file in the current working directory, the new family appears in the list.

```
wavemngr('restore')
wavemngr('read',1)

ans = 76x44 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->                                  '
'===================================         '
'Lemarie           ->->lem                     '
'------------------------------              '
'lem1->lem2->lem3->lem4->                        '
'lem5->                                       '
'===================================         '
```

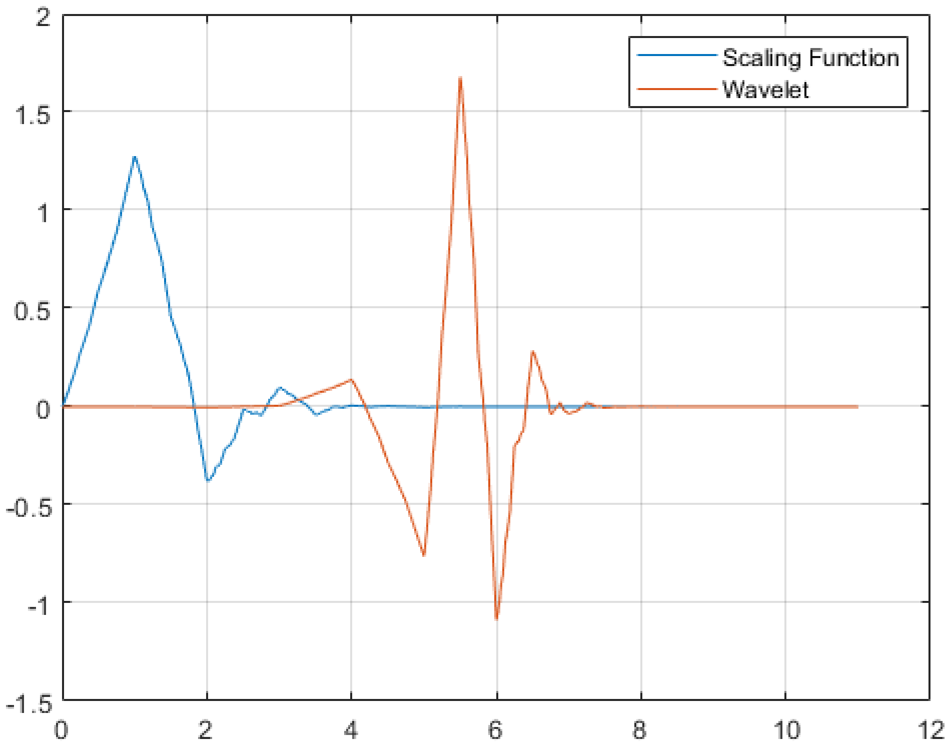
Restore the initial wavelets. Restore the initial ASCII file `wavelets.ini` and the initial MAT-file `wavelets.bin`. Regenerate the list of wavelet families. The list does not include the new family.

```
wavemngr('restore',0)
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             '
    '==================================='
```

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 4 5','lemwavf');
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('lem3');
[phi,psi,xval] = wavefun('lem3');
plot(xval,[phi;psi]);
legend('Scaling Function','Wavelet')
grid on
```



Remove the added family.

```
wavemngr('del','Lemarie')
```

lemwavf.m

```
function F = lemwavf(wname)
%LEMWAVF Lemarie wavelet filters.
%   F = LEMWAVF(W) returns the scaling filter associated with the Lemarie
%   wavelet specified by the character array, 'lemN'.
%   Possible values for N are 1, 2, 3, 4 or 5.
%
%   This function is only for use in the "Add Wavelet Families" example. It
%   may change or be removed in a future release.
%
%   Copyright 2019 The MathWorks, Inc.

TFlem = startsWith(wname,'lem');
if ~TFlem
    error('Wavelet short name is lem followed by filter number');
end
fnum = regexp(wname,'(\d+)','match','Once');

if isempty(fnum)
    error('Specify a filter number as 1,2,3,4,or 5');
end

if ~isempty(fnum)
```

```
    num = str2double(fnum);
end

tffilt = ismember(num,[1 2 3 4 5]);

if ~tffilt
    error('Filter number must be 1, 2, 3, 4, or 5');
end


switch num
    case 1
F = [...
   0.46069299844871  0.53391629051346  0.03930700681965  -0.03391629578182 ...
];

    case 2
F = [...
   0.31555164655258  0.59149765057882  0.20045477817080  -0.10034811856888 ...
  -0.01528128420694  0.00846362066021  -0.00072514051618  0.00038684732960 ...
        ];

    case 3
F = [...
   0.23108942231941  0.56838231367966  0.33173980738190  -0.09447000132310 ...
  -0.06203683305244  0.02661631105889  -0.00209952890579  0.00001769381066 ...
   0.00128429679795  -0.00053703458679  0.00002283826072 -0.00000928544107 ...
        ];

    case 4
F = [...
   0.17565337503255  0.52257484913870  0.42429244721660  -0.04601056550580 ...
  -0.11292720306517  0.03198741803409  0.00813124691980  -0.00743764392677 ...
   0.00548090619143 -0.00140066128481  -0.00054200083128   0.00025607264164 ...
  -0.00008795126642  0.00003025515674  -0.00000082014466  0.00000027569334 ...
        ];

    case 5
F = [...
   0.13807658847623  0.47310642622099  0.48217097800239  0.02112933622031 ...
  -0.15081998732499  0.01935767268926  0.02716532750995  -0.01588522540421 ...
   0.00671209165995  0.00120022744496  -0.00321203819186  0.00115266788547 ...
  -0.00018266213413  -0.00002953360842  0.00008433396295  -0.00002997969339 ...
   0.00000534552866  -0.00000159098026  0.00000003069431 -0.00000000895816 ...
        ];

end
```

Input Arguments

FN — Wavelet family name

character vector | string scalar

Wavelet family name, specified as a character vector or string scalar.

FSN — Wavelet family short name

character vector | string scalar

Wavelet family short name, specified as a character vector or string scalar. The number of characters in FSN must be less than or equal to 4.

WT — Wavelet family type

1 | 2 | 3 | 4 | 5

Wavelet family type, specified as one of the following:

- 1 – Orthogonal wavelets
- 2 – Biorthogonal wavelets
- 3 – Wavelet with a scaling function
- 4 – Wavelet without a scaling function
- 5 – Complex wavelet without a scaling function

NUMS — Wavelet parameters

'' | character vector | string scalar

Wavelet parameters, specified as:

- If the family consists of a single wavelet, NUMS is the empty string ''. For example, the mexh and morl families each contain a single wavelet.
- If the wavelet is member of a finite family of wavelets, NUMS contains a space-separated list of items representing wavelet parameters. For example, for the biorthogonal wavelet family bior, NUMS = '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'.
- If the wavelet is member of an infinite family of wavelets, NUMS contains a space-separated list of items representing wavelet parameters, terminated by the special sequence **. Two examples are listed in the following table.

Wavelet Family	NUMS
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 **'

TYPNUMS — Wavelet parameter input format

'integer' (default) | 'real' | 'charactervector'

Wavelet parameter input format, specified as:

- 'integer' — Use this option when the parameter is an integer. For example, the Daubechies wavelet family db uses an integer parameter.
- 'real' — Use this option when the parameter is real. For example, the biorthogonal wavelet family bior uses a real parameter.
- 'charactervector' — Use this option when the parameter is a character vector. For example, the Shannon wavelet family uses a character vector.

FILE — Wavelet definition file

character vector | string scalar

Wavelet definition file, specified as a character vector or string scalar. FILE is the name of a MAT-file or a code file name that defines the wavelet family.

If the wavelet family contains only one type 1 (orthogonal) or type 2 (biorthogonal) wavelet, you can define the wavelet in a MAT-file. The MAT-file contains the scaling filter coefficients. The filename must match the wavelet family short name.

- If you define an orthogonal wavelet in a MAT-file, the name of the variable containing the scaling filter coefficients must match the name of the wavelet family short name.
- If you define a biorthogonal wavelet in a MAT-file, the names of the variables containing the scaling filter coefficients must be `Df` and `Rf`.

For more information, see "Add Quadrature Mirror and Biorthogonal Wavelet Filters".

Example: If a family that contains a single orthogonal wavelet has the short name `wfsn`, the variable `wfsn` must contain the scaling filter coefficients. To create the necessary MAT-file, you would use the command `save wfsn wfsn`.

B — Effective support

two-element real-valued vector

Effective support for wavelets with family type `WT` equal to 3, 4, or 5, specified as a two-element real-valued vector. If `B = [lb ub]`, then `lb` specifies the lower bound, and `ub` specifies the upper bound.

Data Types: `double`

WN — Wavelet family

character vector | string scalar

Wavelet family, specified by the character vector or string scalar `WN`. The value of `WN` is either the wavelet family name or wavelet family short name.

Example: `wavemngr('del','Lemarie')`

Limitations

- `wavemngr` allows you to add a wavelet. You must verify that it is truly a wavelet. No check is performed to confirm the addition is a wavelet or to confirm the type of the new wavelet. You can use `dwtfilterbank` to verify if a wavelet is orthogonal or biorthogonal.

References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

See Also

"Add Quadrature Mirror and Biorthogonal Wavelet Filters"

Introduced before R2006a

wavenames

(To be removed) Wavelet names for LWT

Note `wavenames` will be removed in a future release. Use `liftingScheme`. For more information, see “Compatibility Considerations”.

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')`.

Compatibility Considerations

wavenames will be removed

Not recommended starting in R2021a

The `wavenames` function will be removed in a future release. See the `Wavelet` property of `liftingScheme` for a list of valid wavelets.

See Also

`liftingScheme`

Introduced before R2006a

waverec

1-D wavelet reconstruction

Syntax

```
x = waverec(c,l,wname)
x = waverec(c,l,LoR,HiR)
```

Description

`x = waverec(c,l,wname)` reconstructs the 1-D signal `x` based on the multilevel wavelet decomposition structure `[c,l]` and the wavelet specified by `wname`. See `wavedec`.

Note: `x = waverec(c,l,wname)` is equivalent to `x = appcoef(c,l,wname,0)`.

`x = waverec(c,l,LoR,HiR)` reconstructs the signal using the specified lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

Examples

Multilevel 1-D Wavelet Reconstruction

Load a signal. Perform a level 3 wavelet decomposition of the signal using the `db6` wavelet.

```
load leleccum
wv = 'db6';
[c,l] = wavedec(leleccum,3,wv);
```

Reconstruct the signal using the wavelet decomposition structure.

```
x = waverec(c,l,wv);
```

Check for perfect reconstruction.

```
err = norm(leleccum-x)
```

```
err = 1.0082e-09
```

Input Arguments

c — Wavelet decomposition

vector

Wavelet decomposition, specified as a vector. The vector contains the wavelet coefficients. The bookkeeping vector `l` contains the number of coefficients by level. See `wavedec`.

Data Types: `single` | `double`

l — Bookkeeping vector

vector

Bookkeeping vector, specified as a vector of positive integers. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition `c` by level. See `wavedec`.

Data Types: `single` | `double`

`wname` — Analyzing wavelet
character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar.

Note `waverec` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoR,HiR` — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. `LoR` is the lowpass reconstruction filter, and `HiR` is the highpass reconstruction filter. The lengths of `LoR` and `HiR` must be equal. See `wfilters` for additional information.

Data Types: `single` | `double`

Output Arguments

`x` — Reconstructed signal
vector

Reconstructed signal, returned as a vector.

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM Ed, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

[3] Meyer, Y. *Wavelets and Operators*. Translated by D. H. Salinger. Cambridge, UK: Cambridge University Press, 1995.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The input `wname` must be constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.
- For optimized GPU code generation, specify the bookkeeping vector `l` as a compile-time constant.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`appcoef` | `idwt` | `wavedec`

Introduced before R2006a

waverec2

2-D wavelet reconstruction

Syntax

```
x = waverec2(c,s,wname)
x = waverec2(c,s,LoR,HiR)
```

Description

`x = waverec2(c,s,wname)` performs a multilevel wavelet reconstruction of the matrix `x` based on the wavelet decomposition structure `[c,s]` and the wavelet specified by `wname`. See `wavedec2`.

`x = waverec2(c,s,wname)` is equivalent to `x = appcoef2(c,s,wname,0)`.

`x = waverec2(c,s,LoR,HiR)` reconstructs `x` using the specified lowpass and highpass wavelet reconstruction filters `LoR` and `HiR`, respectively.

Examples

2-D Wavelet Reconstruction

Load an image.

```
load woman
```

Perform a level 2 wavelet decomposition of the image using the `sym4` wavelet.

```
wv = 'sym4';
[c,s] = wavedec2(X,2,wv);
```

Reconstruct the image from the wavelet decomposition structure.

```
xrec = waverec2(c,s,wv);
```

Check for perfect reconstruction.

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

```
ans = 2.0989e-10
```

Input Arguments

c — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector, specified as a real-valued vector. The vector `c` contains the approximation and detail coefficients organized by level. The bookkeeping matrix `s` is used to parse `c`. See `wavedec2`.

Data Types: `double`

`s` — Bookkeeping matrix
integer-valued matrix

Bookkeeping matrix, specified as an integer-valued matrix. The matrix `s` contains the dimensions of the wavelet coefficients by level and is used to parse the wavelet decomposition vector `c`. See `wavedec2`.

Data Types: `double`

`wname` — Analyzing wavelet
character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar.

Note `waverec2` supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See `wfilters` for a list of orthogonal and biorthogonal wavelets.

`LoR,HiR` — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. `LoR` is the lowpass reconstruction filter, and `HiR` is the highpass reconstruction filter. The lengths of `LoR` and `HiR` must be equal. See `wfilters` for additional information.

Data Types: `double`

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:

- The input `wname` must be constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input `wname` must be constant.
- For optimized GPU code generation, specify the bookkeeping matrix `s` as a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported. See `dwtmode`.

See Also

`appcoef2` | `idwt2` | `wavedec2`

Introduced before R2006a

waverec3

3-D wavelet reconstruction

Syntax

```
x = waverec3(wdec)
c = waverec3(wdec,type,n)
```

Description

`x = waverec3(wdec)` reconstructs the 3-D array `x` based on the multilevel wavelet decomposition structure `wdec`.

`c = waverec3(wdec,type,n)` reconstructs or extracts at level `n` the multilevel components specified by `type`. If `type` begins with `'c'` or `'C'`, `waverec3` extracts the specified components. Otherwise, `waverec3` reconstructs the components.

`x = waverec3(wdec,'a',0)` and `x = waverec3(wdec,'ca',0)` are equivalent to `x = waverec3(wdec)`, where `'a'` specifies the lowpass component. `x` is the reconstruction of the coefficients in `wdec` at level 0.

`c = waverec3(wdec,type)` is equivalent to `c = waverec3(wdec,type,wdec.level)`.

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
```

Input Arguments

wdec — Wavelet decomposition

structure

Wavelet decomposition, specified as a structure. The structure is the output of `wavedec3` and has the following fields:

sizeINI — Size

vector

Size of the 3-D array, specified as a 1-by-3 vector.

level — Level of the decomposition

integer

Level of the decomposition, specified as an integer.

mode — Name of the wavelet transform extension mode

character vector

Name of the wavelet transform extension mode, specified as a character vector.

filters — Wavelet filters

structure

Wavelet filters used for the decomposition, specified as a structure with the following fields:

- `LoD` — lowpass decomposition filter
- `HiD` — highpass decomposition filter

- `LoR` — lowpass decomposition filter
- `HiR` — highpass decomposition filter

`dec` — Decomposition coefficients

cell array

Decomposition coefficients, specified as an *N*-by-1 cell array, where *N* equals 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 sequence of letters gives 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

matrix

Successive sizes of the decomposition components, specified as a `wdec.level`+1-by-2 matrix.

`type` — Type of reconstruction or extraction

character vector | string scalar

Type of reconstruction or extraction, specified as a character vector or string scalar. For reconstruction, valid values of `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 lowpass filter and `'D'` (or `'H'`) is a highpass filter.
- The char `'d'` (or `'h'` or `'D'` or `'H'`) gives the sum of all the components different from the lowpass component.
- The char `'a'` (or `'l'` or `'A'` or `'L'`) gives the lowpass component (the approximation at level `n`).

To extract coefficients, the valid values for `type` are the same but prefixed by `'c'` or `'C'`.

`n` — Decomposition level

`wdec.level` (default) | integer

Decomposition level, specified as an integer.

Output Arguments

x — Reconstruction

3-D array

Reconstruction, returned as a 3-D array of size `sz(1)`-by-`sz(2)`-by-`sz(3)`, where `sz = wpdec.sizeINI`.

c — Extracted coefficients

3-D array

Extracted coefficients, returned as a 3-D array.

See Also

`idwt3` | `waveinfo` | `wavedec3`

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 Gaussian 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 denoising

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 denoising. 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 denoised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in denoising 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 denoised signal or image grows with ALPHA. Typically ALPHA = 2.

THR minimizes the penalized criterion given by the following:

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=$|c(t^*)|$.

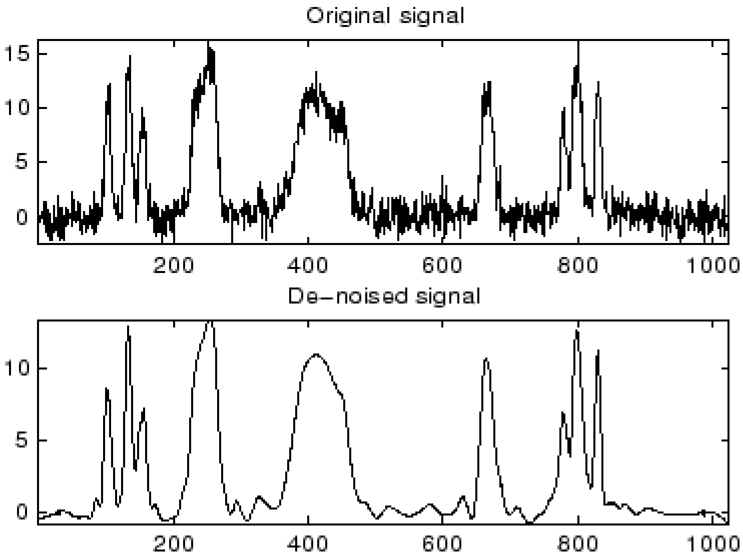
wbmpen(C,L,SIGMA,ALPHA,ARG) computes the global threshold and, in addition, plots three curves:

- 2*SIGMA^2*t*(ALPHA + log(n/t))
- sum(c(k)^2,k¬≤t)
- crit(t)

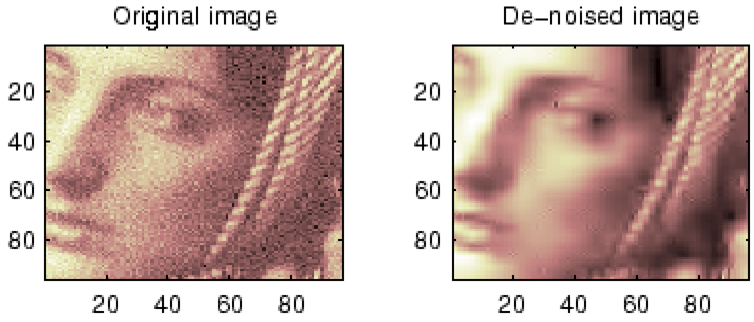
Examples

```
% Example 1: Signal denoising.
% 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 denoising, using the tuning parameter.
alpha = 2;
thr = wbmpen(c,l,sigma,alpha)
thr =

    2.7681

% Use wdencmp for denoising 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 denoised signals.
figure(1)
subplot(211), plot(x), title('Original signal')
subplot(212), plot(xd), title('De-noised signal')
```



```
% Example 2: Image denoising.
% 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 denoising.
alpha = 1.2;
thr = wbmpen(c,l,sigma,alpha)
thr =

   36.0621

% Use wdencmp for denoising 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 denoised 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

wdenoise | 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 as integers in the range [1,16].

y = wcodemat(x,nbcodes) rescales x as integers in the range [1,nbcodes].

y = wcodemat(x,nbcodes,opt) rescales x along the dimension specified by opt.

y = wcodemat(x,nbcodes,opt,absol) rescales 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.

Examples

Extended Pseudocolor Matrix Scaling

Load an image.

```
load woman
```

Obtain the range of the colormap.

```
NBCOL = size(map,1)
```

```
NBCOL = 255
```

Obtain the single-level discrete wavelet transform of the image using the Haar wavelet.

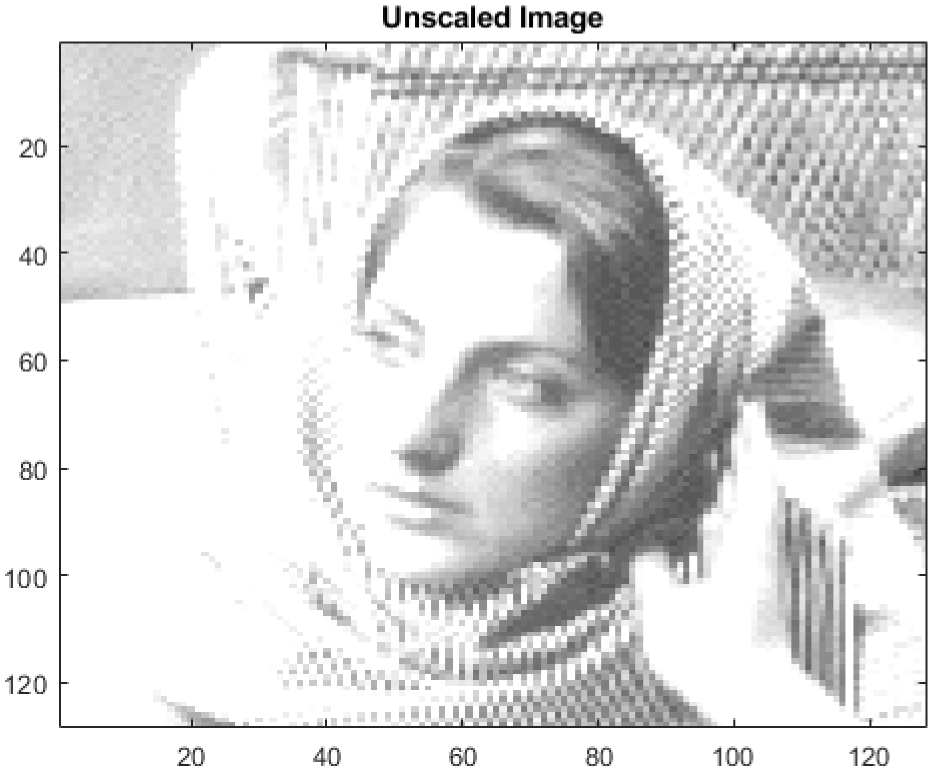
```
[cA1,cH1,cV1,cD1] = dwt2(X,"db1");
```

Scale the level-one approximation coefficients globally to the full range of the colormap.

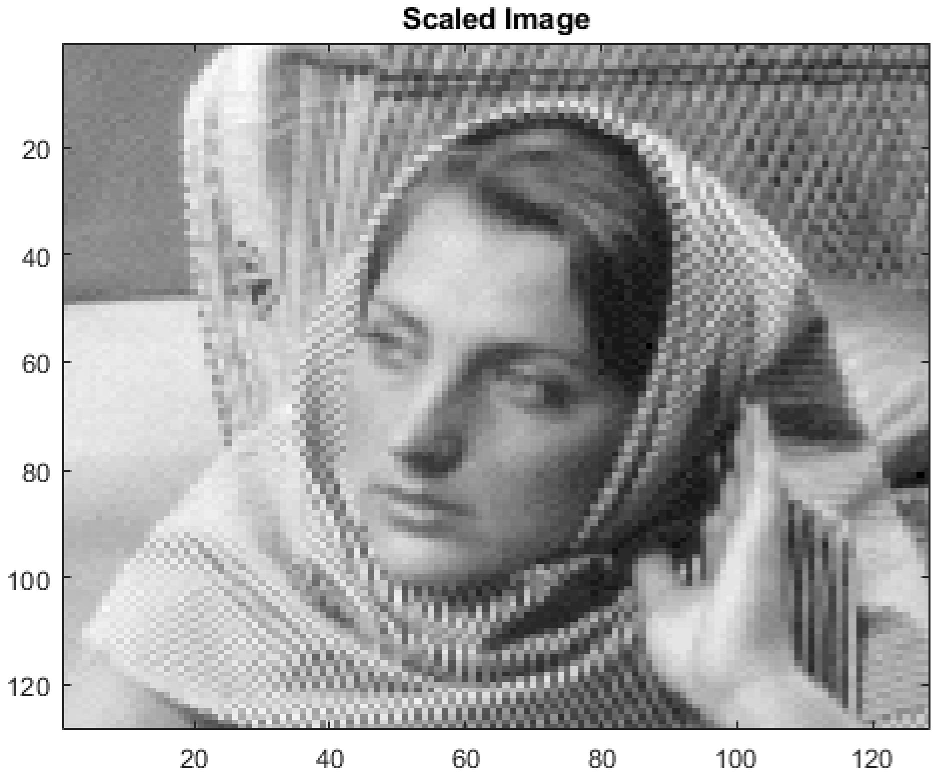
```
cA1scaled = wcodemat(cA1,NBCOL);
```

Display the approximation coefficients without scaling and with scaling.

```
image(cA1);
colormap(map)
title("Unscaled Image")
```

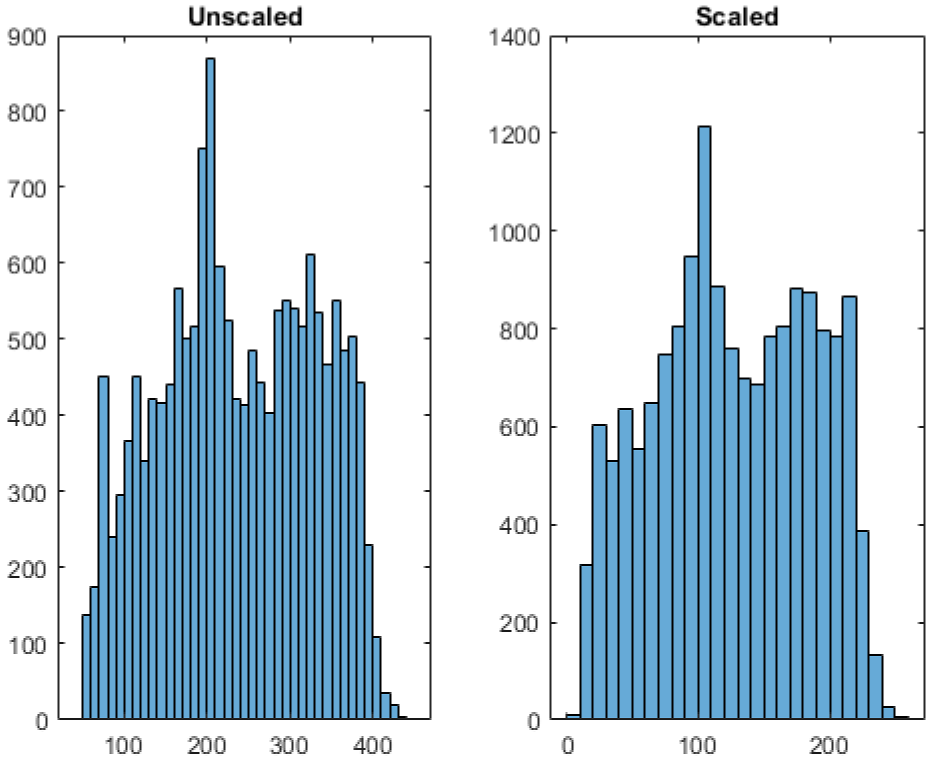


```
image(cA1scaled)
colormap(map)
title("Scaled Image")
```



Display histograms of the unscaled and scaled approximation coefficients.

```
subplot(1,2,1)
histogram(cA1)
title("Unscaled")
subplot(1,2,2)
histogram(cA1scaled)
title("Scaled")
```



Input Arguments

x — Input
matrix

Input, specified as a matrix.

Data Types: `double`

nbcodes — Upper bound
`16` (default) | positive integer

Upper bound of range to rescale `x` as integers, specified as a positive integer.

Data Types: `double`

opt — Dimension
`"mat"` (default) | `"m"` | `"column"` | `"c"` | `"row"` | `"r"`

Dimension along which to rescale the matrix, specified as one of the following:

- `"mat"` or `"m"` — rescale `x` globally
- `"column"` or `"c"` — rescale `x` column-wise
- `"row"` or `"r"` — rescale `x` row-wise

Data Types: `string`

`absol` — Rescale basis
`1` (default) | scalar

Rescale basis, specified as a scalar. If `absol` is nonzero, `wcodemat` rescales `x` based on the absolute values of the elements of `x`. If `absol` is equal to zero, `wcodemat` rescales `x` based on the signed values of the elements of `x`.

Data Types: `double`

See Also

`dwt2` | `wavedec2`

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:

- `'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');
```

More About

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:

$$\frac{\left|S(C_x^*(a,b)C_y(a,b))\right|^2}{S(\left|C_x(a,b)\right|^2)\cdot S(\left|C_y(a,b)\right|^2)}$$

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

“Compare Time-Frequency Content in Signals with Wavelet Coherence”
“Continuous Wavelet Analysis of Cusp Signal”
“Continuous and Discrete Wavelet Transforms”

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)
[ ___ ,coi,wtx,wty] = wcoherence( ___ )
[ ___ ] = wcoherence( ___ ,Name,Value)
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 the positive `duration` `ts` as the sampling interval. The duration `ts` is used to compute the scale-to-period conversion, `period`. The duration array `period` has the same format 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 sampling frequency `fs` 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 cone of influence is in Hz.

`[wcoh,wcs,period,coi] = wcoherence( ___ ,ts)` returns the cone of influence, `coi`, in cycles per unit time.

`[ ___ ,coi,wtx,wty] = wcoherence( ___ )` returns the continuous wavelet transforms (CWT) of `x` and `y` in `wtx`, `wty`, respectively. `wtx` and `wty` are used in the formation of the wavelet cross spectrum and coherence estimates.

`[ ___ ] = wcoherence( ___ ,Name,Value)` specifies additional options using one or more name-value pair arguments. This syntax may be used in any of the previous syntaxes.

`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);
```

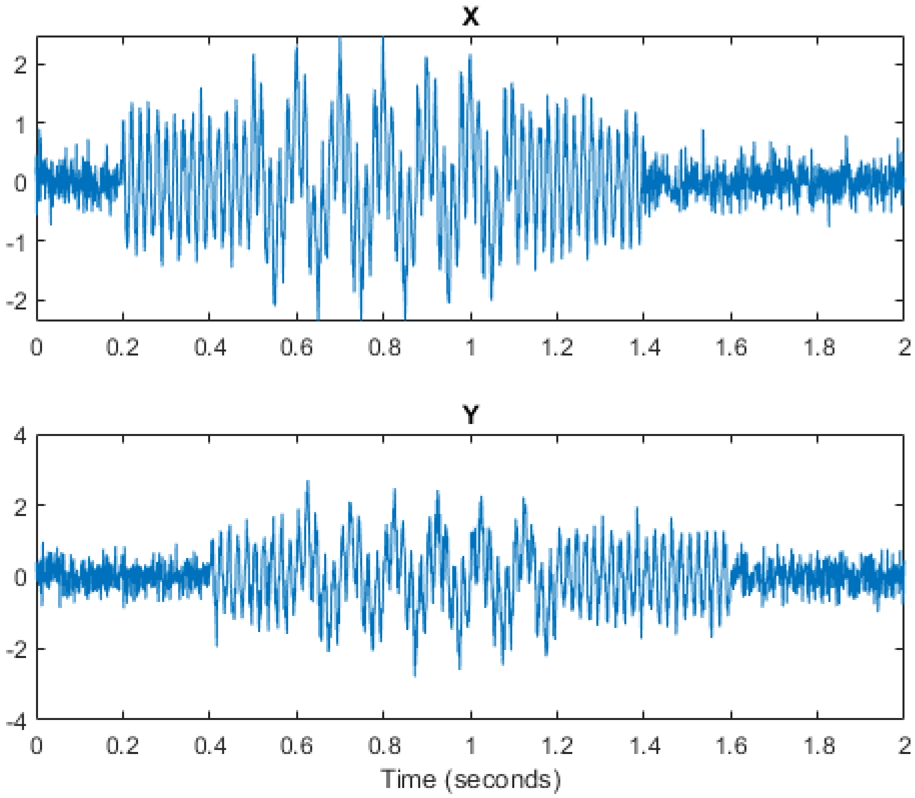
The default coherence computation uses the analytic Morlet wavelet, 12 voices per octave and smooths 12 scales. The default number of octaves is equal to floor(log2(numel(x)))-1, which in this case is 9.

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.

```
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));
subplot(2,1,1)
plot(t,x)
title('X')
subplot(2,1,2)
plot(t,y)
title('Y')
xlabel('Time (seconds)')
```

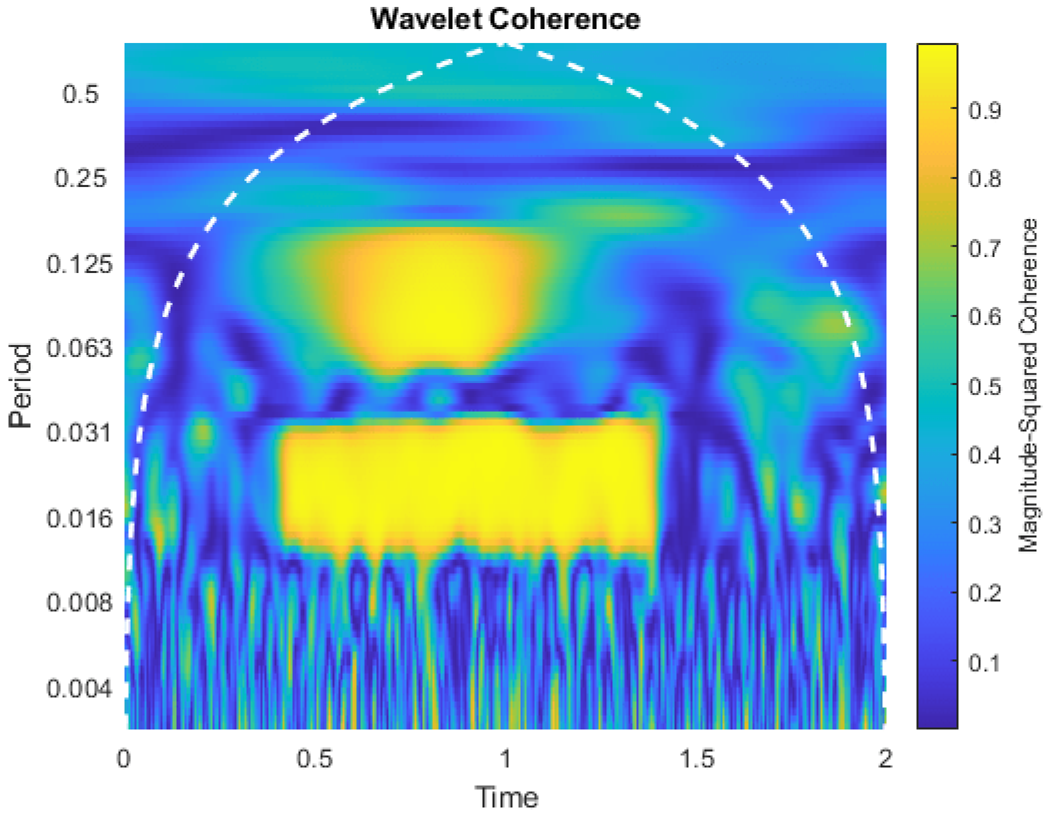


Obtain the coherence of the two signals.

```
[wcoh,~,period,coi] = wcoherence(x,y,seconds(0.001));
```

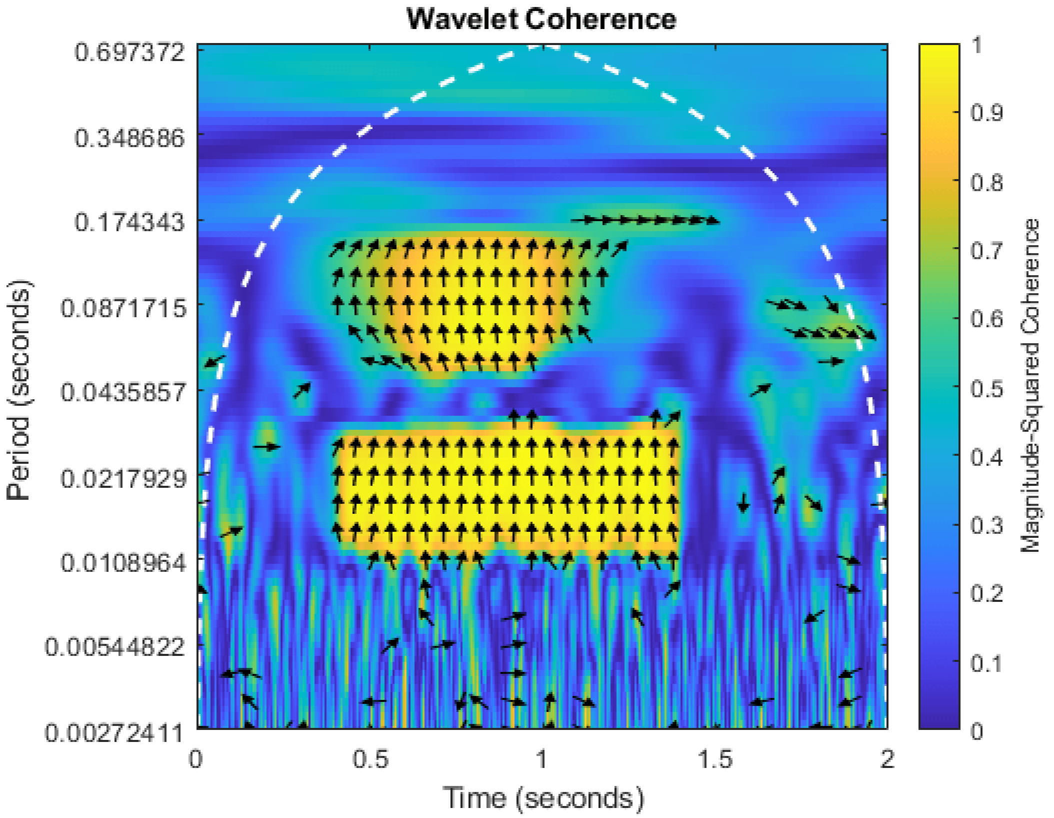
Use the `pcolor` command to plot the coherence and cone of influence.

```
figure
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

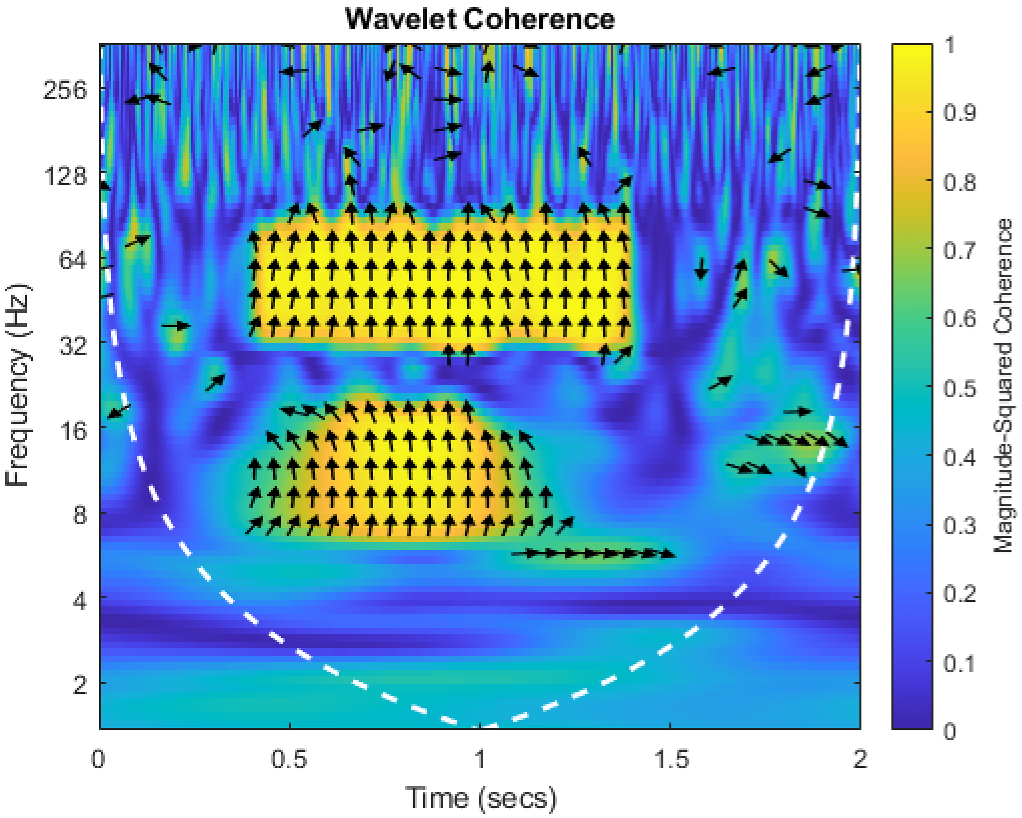
Obtain the wavelet coherence for two signals, specifying a sampling frequency of 1000 Hz. Both signals consist of two sine waves (10 Hz and 50 Hz) in white noise. The sine waves have different time supports.

Set the random number generator to its default settings for reproducibility and create the two signals.

```
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));
```

Obtain the wavelet coherence. The coherence plot is flipped with respect to the plot in the previous example, which specifies a sampling interval instead of a sampling frequency.

```
wcoherence(x,y,1000)
```



Obtain the scale-to-frequency conversion output in `f`.

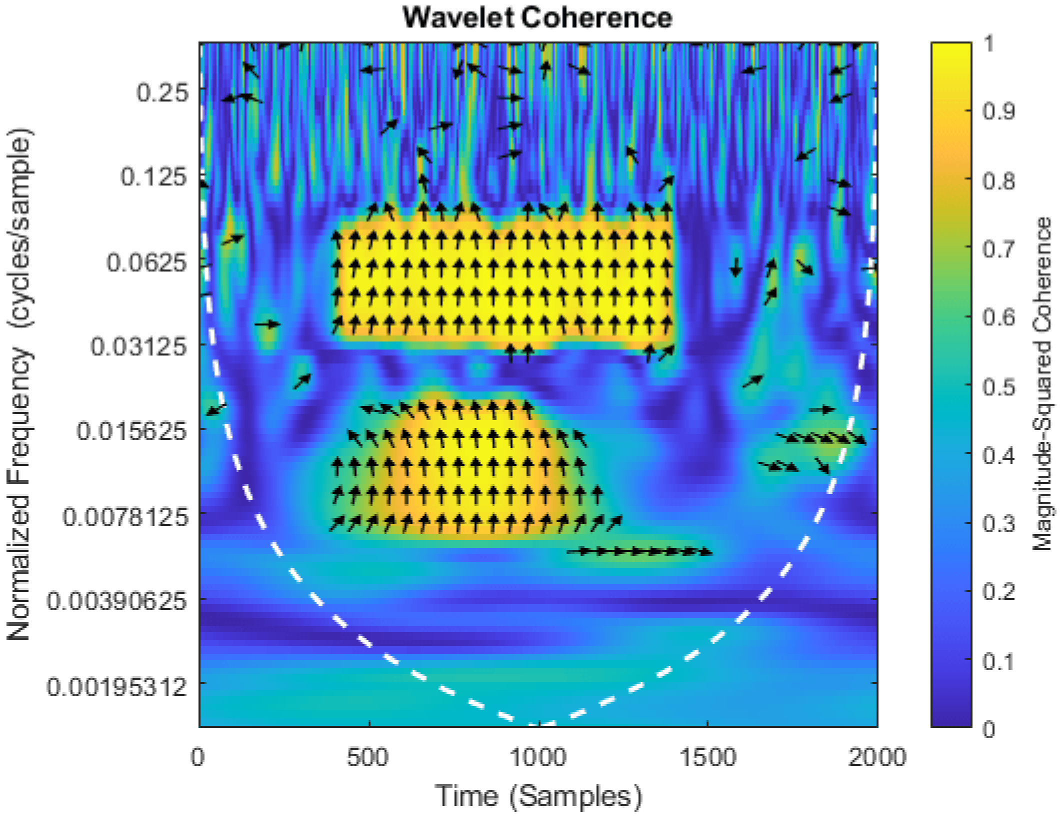
```
[wcoh,wcs,f] = wcoherence(x,y,1000);
```

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.

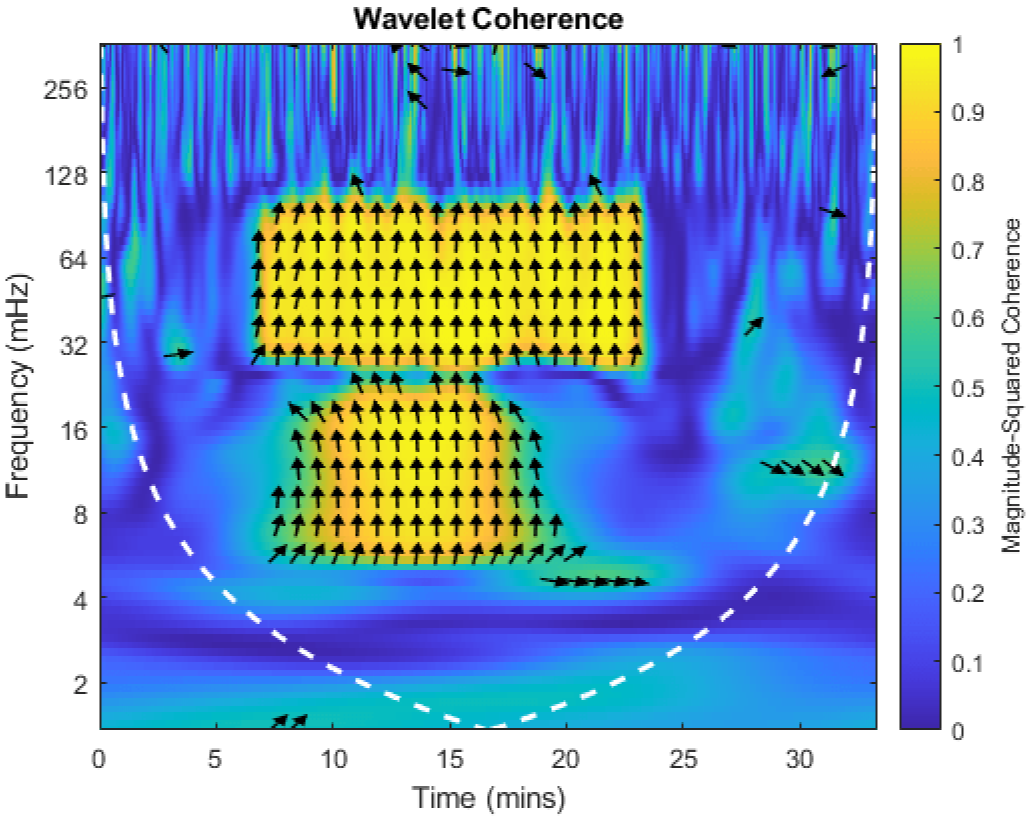
Set the random number generator to its default settings for reproducibility. Then, create the two signals and obtain the coherence.

```
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 18. The increased smoothing causes reduced low frequency resolution.

```
wcoherence(x,y,'NumScalesToSmooth',18)
```

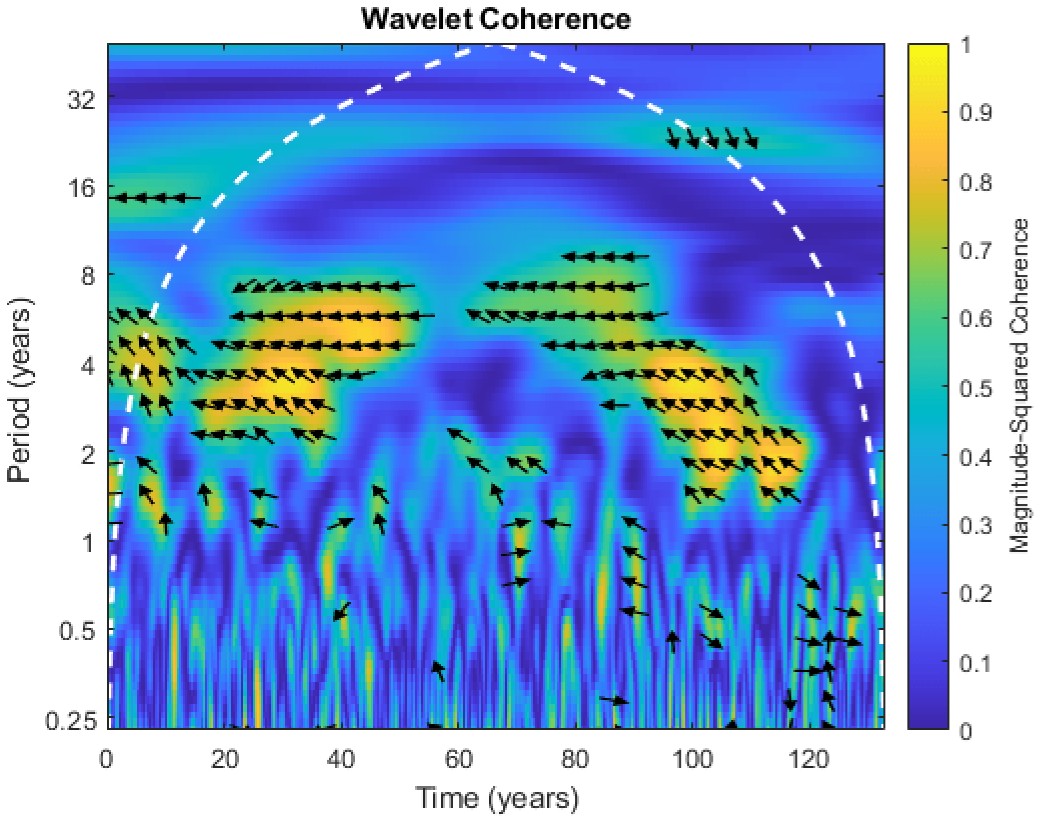


Effect of Phase Display Threshold on Wavelet Coherence of Weather Data

Compare the effects of using different phase display thresholds on the wavelet coherence.

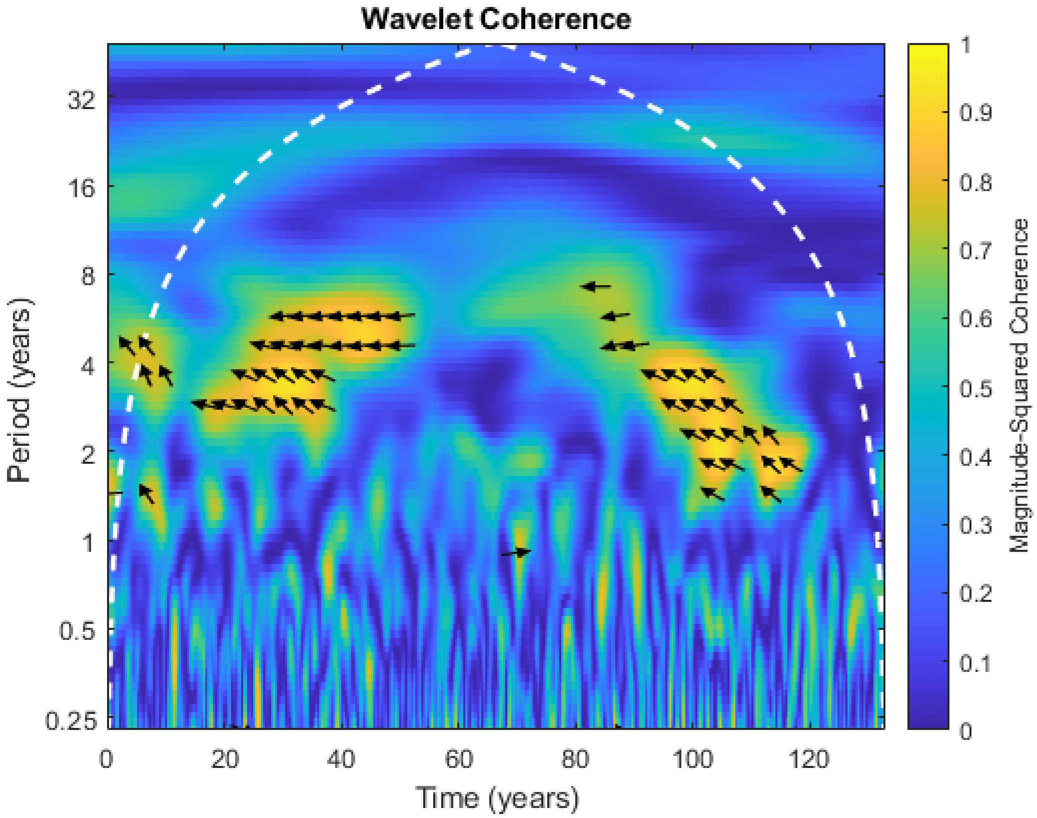
Plot the wavelet coherence between the El Nino time series and the All India Average Rainfall Index. The data are sampled monthly. Specify the sampling interval as 1/12 of a year to display the periods in years. Use the default phase display threshold of 0.5, which shows phase arrows only where the coherence 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 a sampling frequency `fs` and a sampling period `ts`.

`fs` — Sampling frequency

positive scalar | `[]`

Sampling frequency, specified as a positive scalar.

If you specify `fs` as empty, `wcoherence` uses normalized frequency in cycles/sample. The Nyquist frequency is ½.

You cannot specify both a sampling frequency `fs` and a sampling period `ts`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PhaseDisplayThreshold',0.7;` specifies the threshold for displaying phase vectors.

`FrequencyLimits` — Frequency limits

two-element scalar vector

Frequency limits to use in `wcoherence`, specified as a two-element vector with positive strictly increasing elements. The first element specifies the lowest peak passband frequency and must be greater than or equal to the product of the wavelet peak frequency in hertz and two time standard deviations divided by the signal length. The second element specifies the highest peak passband frequency and must be less than or equal to the Nyquist frequency. The base 2 logarithm of the ratio of the maximum frequency to the minimum frequency must be greater than or equal to 1/NV where NV is the number of voices per octave.

If you specify frequency limits outside the permissible range, `wcoherence` truncates the limits to the minimum and maximum valid values. Use `cwtfreqbounds` with the wavelet set to `'amor'` to determine frequency limits for different parameterizations of the wavelet coherence.

Example: `'FrequencyLimits',[0.1 0.3]`

`PeriodLimits` — Period limits

two-element duration array

Period limits to use in `wcoherence`, specified as a two-element duration array with strictly increasing positive elements. The first element must be greater than or equal to `2×ts` where `ts` is the sampling period. The base 2 logarithm of the ratio of the minimum period to the maximum period must be less than or equal to -1/NV where NV is the number of voices per octave. The maximum period cannot exceed the signal length divided by the product of two time standard deviations of the wavelet and the wavelet peak frequency.

If you specify period limits outside the permissible range, `wcoherence` truncates the limits to the minimum and maximum valid values. Use `cwtfreqbounds` with the wavelet set to `'amor'` to determine period limits for different parameterizations of the wavelet coherence.

Example: `'PeriodLimits',[seconds(0.2) seconds(1)]`

Data Types: `duration`

`VoicesPerOctave` — Number of voices per octave

12 (default) | even integer from 10 to 32

Number of voices per octave to use in the wavelet coherence, specified as an even integer from 10 to 32.

`NumScalesToSmooth` — Number of scales to smooth

positive integer

Number of scales to smooth in time and scale, specified as a positive integer less than or equal to one half *N*, where *N* is the number of scales in the wavelet transform. If unspecified, `NumScalesToSmooth` defaults to the minimum of `floor(N/2)` and `VoicesPerOctave`. The function uses a moving average filter to smooth across scale. 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.

The `'NumOctaves'` name-value pair is not recommended and will be removed in a future release. The recommended way to modify the frequency or period range of wavelet coherence is with the `'FrequencyLimits'` or `'PeriodLimits'` name-value pairs. You cannot specify both the `'NumOctaves'` and `'FrequencyLimits'` or `'PeriodLimits'` name-value pairs. See `cwtfreqbounds`.

`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.

`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 | 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.

For additional information, see “Boundary Effects and the Cone of Influence”.

`wtx` — Continuous wavelet transform of `x`

matrix

Continuous wavelet transform of `x`, returned as a matrix.

`wty` — Continuous wavelet transform of `y`

matrix

Continuous wavelet transform of `y`, returned as a matrix.

More About

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:

$$\frac{\left|S(C_x^*(a,b)C_y(a,b))\right|^2}{S(\left|C_x(a,b)\right|^2)\cdot S(\left|C_y(a,b)\right|^2)}$$

$C_x(a,b)$ 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.

Compatibility Considerations

`'NumOctaves'` name-value pair will be removed

Not recommended starting in R2020a

The `'NumOctaves'` name-value pair argument will be removed in a future release. Use either:

- Name-value pair argument `'FrequencyLimits'` to modify the frequency range of wavelet coherence.
- Name-value pair argument `'PeriodLimits'` to modify the period range of wavelet coherence.

See `cwtfreqbounds` for additional information.

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.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The following input arguments are not supported: `ts` (sampling interval), `PeriodLimits` name-value pair, and `PhaseDisplayThreshold` name-value pair.
- The `duration` data type is not supported.
- Plotting is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`cwt` | `cwtfreqbounds` | `cwtfilterbank`

Introduced in R2016a

wcompress

True compression of images using wavelets

Syntax

```
wcompress('c',x,cname,compmthd)
wcompress('c',fname,___)
wcompress('c',I,___)
wcompress(___,Name,Value)
[comprat,bpp] = wcompress('c',___)
```

```
xc = wcompress('u',cname)
xc = wcompress('u',cname,'plot')
xc = wcompress('u',cname,'step')
```

Description

The `wcompress` function performs either compression or uncompression of grayscale or truecolor images.

Compression

`wcompress('c',x,cname,compmthd)` compresses the image `x` using the compression method `compmthd` and saves the result in the file `cname`. The image `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.

You must have write permission in the current working directory or the function will change directory to `tempdir` and write the compressed image in that directory.

Note

- The Discrete Wavelet Transform uses the periodized extension mode.
- Data written to the 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,cname,compmthd,'legacy')
  ```

`wcompress('c',fname,___)` loads the image from the file `fname`.

`wcompress('c',I,___)` converts the indexed image `I{1}` to a truecolor image *Y* using the colormap `I{2}` and then compresses *Y*.

`wcompress(___,Name,Value)` specifies options related to display, data transform, and compression methods using one or more name-value pair arguments in addition to the input arguments in previous syntaxes. The name can be in uppercase or lowercase. For example, `'level',3,'CC','klt'` sets the level of the decomposition to 3 and the Color Conversion parameter if `x` is a truecolor image to the Karhunen-Loève transform.

`[comprat,bpp] = wcompress('c',___)` returns the compression ratio `comprat` and the bit-per-pixel ratio `bpp`.

Uncompression

`xc = wcompress('u',cname)` uncompresses the file `cname` which contains the compressed image, and the returns the image `xc`.

`xc = wcompress('u',cname,'plot')` plots the uncompressed image.

`xc = wcompress('u',cname,'step')` shows the step-by-step uncompression (only for Progressive Coefficients Significance Methods).

Examples

Image Compression Using Basic Parameters

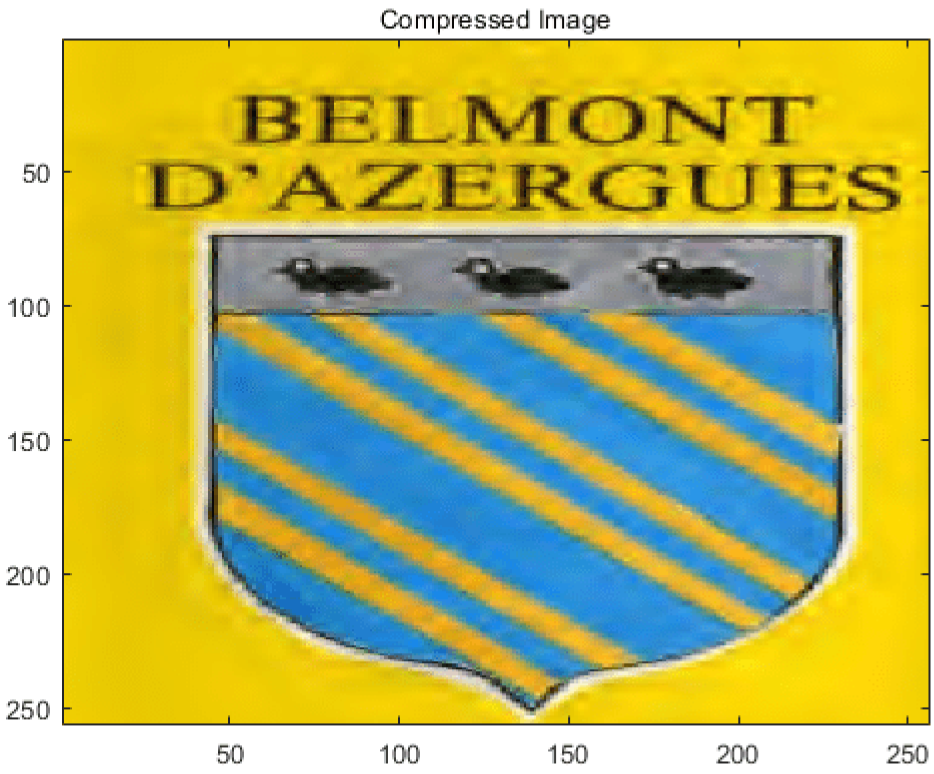
This example shows how to compress and uncompress the jpeg image arms.jpg.

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');
```



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
```

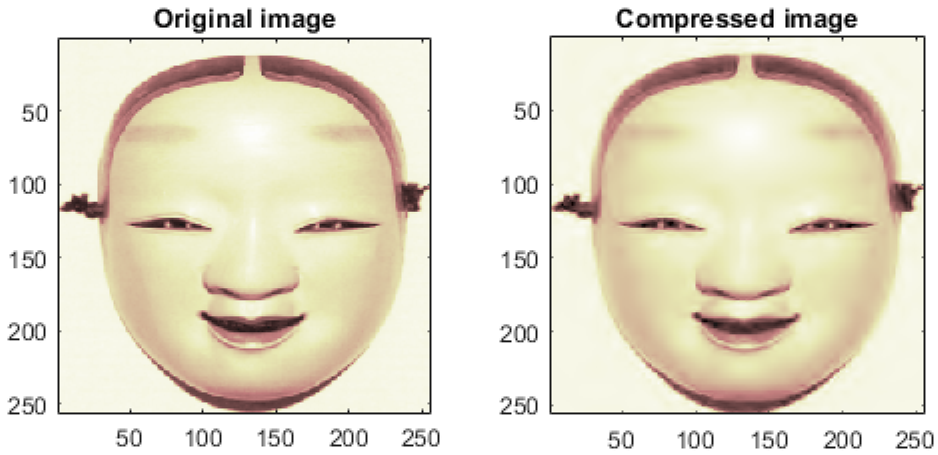
Load the stored image from the file, uncompress it, and delete the file.

```
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)
```

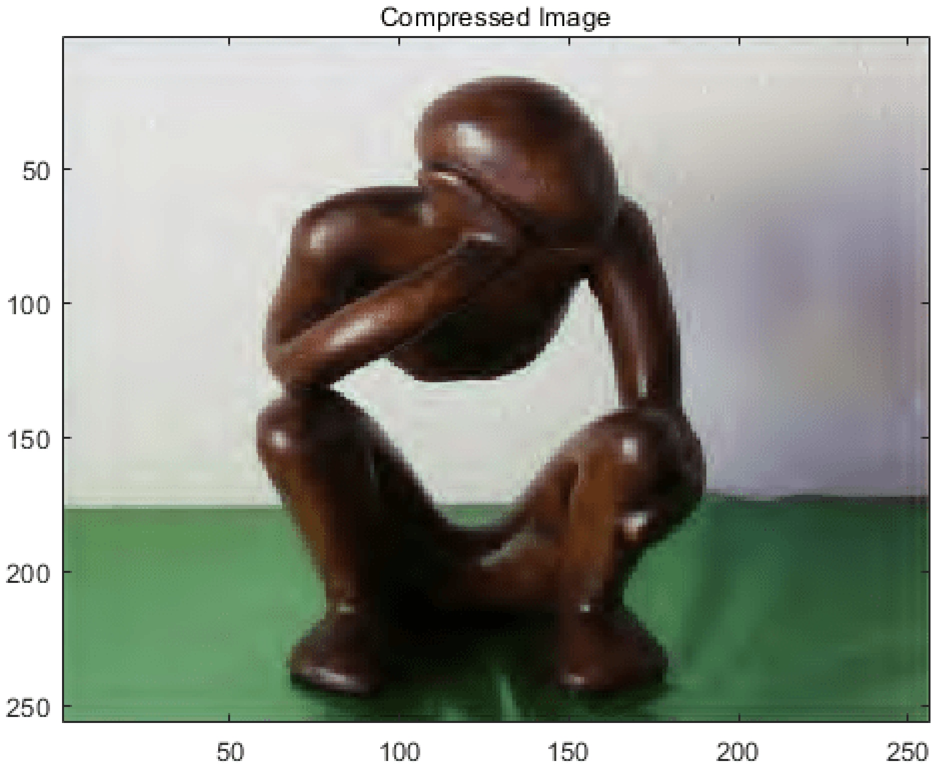
```
psnr = 32.8601
```

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 ('klt'). 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.

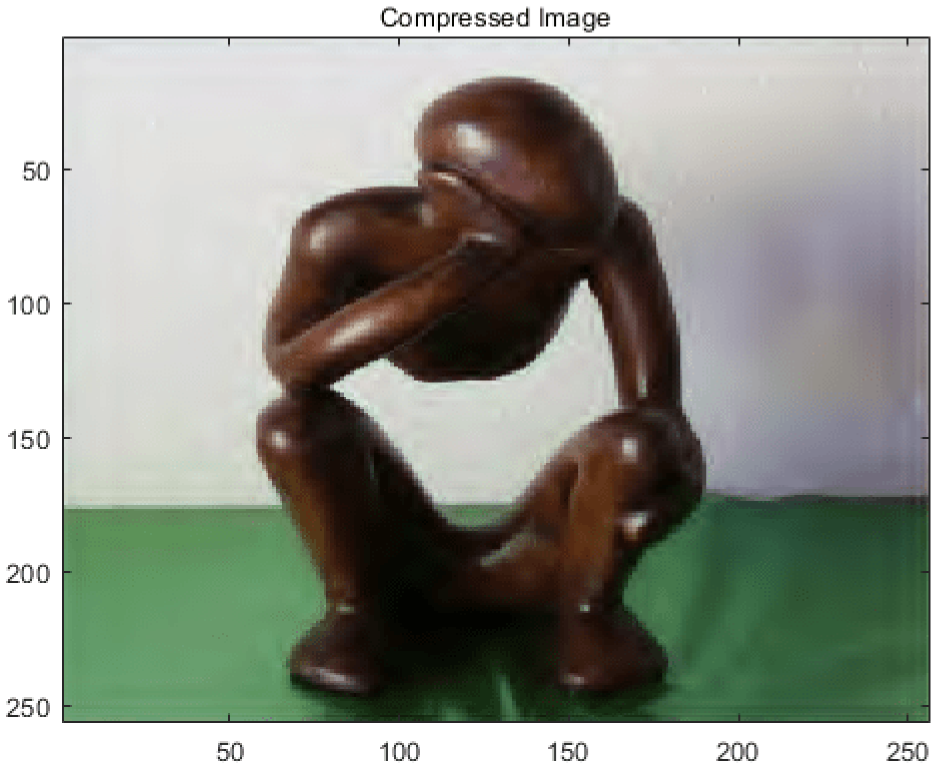
```
[cratio,bpp] = wcompress('c','woodstatue.jpg','woodstatue.wtc', ...
             'aswdr','cc','klt','maxloop',11,'plotpar','step');
cratio
bpp
```

```
cratio =

    3.0792

bpp =

    0.7390
```



Load the compressed image and step through the uncompression process.

```
wcompress('u','woodstatue.wtc','step');
```



Compression and Uncompression of a Truecolor Image

This example shows how to compress a truecolor 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
```

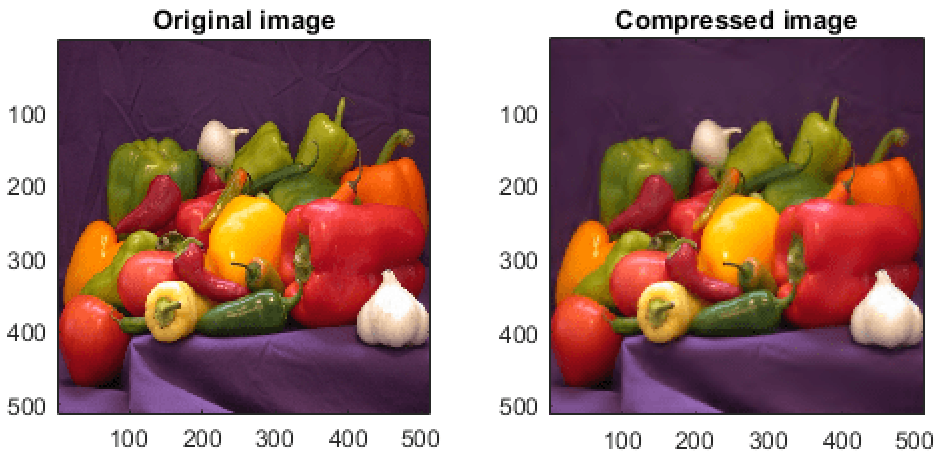
```
bpp = 0.3972
```

```
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
```

Input Arguments

x — Input image

2-D matrix | 3-D array

Input image to compress, specified as 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.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`cname` — Compressed image filename

character vector | string scalar

Compression

Compressed image filename, specified as a character vector or string scalar. The `wcompress` function writes the compressed image to the file `cname`.

Uncompression

Compressed image filename, specified as a character vector or string scalar. The `wcompress` function reads the compressed image from the file `cname` for uncompression.

`fname` — Image filename

character vector | string scalar

Image filename, specified as a character vector or string scalar. The file is a MATLAB Supported Format (MSF) file: MAT-file or other image files (see `imread`).

`compmthd` — Compression method

character vector | string scalar

Compression method, specified as a character vector or string scalar. The valid compression methods are divided into two categories.

- Progressive Coefficients Significance Methods (**PCSM**):

compmthd	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 additional information of these methods, see the references and especially [3] and [6].

- Coefficients Thresholding Methods (**CTM**):

compmthd	Compression Method Name
`'lvl_mmc'`	Subband thresholding of coefficients and Huffman encoding
`'gbl_mmc_f'`	Global thresholding of coefficients and fixed encoding
`'gbl_mmc_h'`	Global thresholding of coefficients and Huffman encoding

  For additional details on the `'lvl_mmc'` method, see [5]

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'IT','g'` sets the Image type transform to grayscale.

Data Transform Parameters

`wname` — Wavelet name

'bior4.4' (default) | character vector | string scalar

Wavelet name, specified as the comma-separated pair consisting of `'wname'` and a character vector or string scalar. See `waveletfamilies`.

`level` — Level of wavelet decomposition

positive integer

Level of wavelet decomposition, specified as the comma-separated pair consisting of `'level'` and a positive integer. The decomposition level `level` must be such that `1 ≤ level ≤ levmax`, where `levmax` is the maximum possible level (see `wmaxlev`).

The default level depends on the compression method `compmthd`.

- For PCSM methods,`level` is equal to `levmax`.
- For CTM methods, `level` is equal to `fix(levmax/2)`.

Data Types: `single` | `double`

`it` — Image type transform

`'n'` (default) | `'g'` | `'c'`

Image type transform, specified as the comma-separated pair consisting of `'it'` and one of the values listed:

- `'n'` — no transformation (default), image type (truecolor or grayscale) is automatically detected
- `'g'` — grayscale transformation type
- `'c'` — color transformation type (RGB `uint8`)

`cc` — Color conversion parameter

`'rgb'` or `'none'` (default) | `'yuv'` | `'klt'` | `'yiq'` | `'xyz'`

Color conversion parameter, specified as the comma-separated pair consisting of `'cc'` and one of the values listed:

- `'rgb'` or `'none'` — no conversion (default)
- `'yuv'` — YUV color space transform
- `'klt'` — Karhunen-Loève transform
- `'yiq'` — YIQ color space transform
- `'xyz'` — CIEXYZ color space transform

Progressive Coefficients Significance Methods (PCSM)

`maxloop` — Maximum number of steps

10 (default) | positive integer | `Inf`

Maximum number of steps for the compression algorithm, specified as the comma-separated pair consisting of `'maxloop'` and a positive integer or `Inf`.

Data Types: `single` | `double`

Coefficients Thresholding Methods (CTM)

`bpp` — Bit-per-pixel ratio

`0.25` (default) | scalar

Bit-per-pixel ratio, specified as the comma-separated pair consisting of `'bpp'` and a scalar. The ratio must be greater than 0 and less than or equal to 8 (grayscale) or 24 (truecolor).

If you specify the bit-per-pixel ratio, you cannot specify `comprat`.

Data Types: `single` | `double`

`comprat` — Compression ratio

`0.25` (default) | scalar

Compression ratio, specified as the comma-separated pair consisting of `'comprat'` and a scalar. The ratio must be greater than 0 and less than or equal to 100.

If you specify the compression ratio, you cannot specify `bpp`.

Data Types: `single` | `double`

`nbclas` — Number of classes for quantization

`75` (default) | positive integer

Number of classes for quantization, specified as the comma-separated pair consisting of `'nbclas'` and positive integer greater than or equal to 2 and less than or equal to 100.

`nbclas` is valid only for the global thresholding methods.

Data Types: `single` | `double`

`threshold` — Threshold value for compression

nonnegative integer

Threshold value for compression, specified as the comma-separated pair consisting of `'threshold'` and a nonnegative number.

`threshold` is valid only for the global thresholding methods.

If you specify `threshold`, you cannot specify `nbcfs`, `percfs`, `bpp`, or `comprat`.

Data Types: `single` | `double`

`nbcfs` — Number of preserved coefficients

nonnegative integer

Number of preserved coefficients in the wavelet decomposition, specified as the comma-separated pair consisting of `'nbcfs'` and nonnegative integer less than or equal to the total number of coefficients in the wavelet decomposition.

`nbcfs` is valid only for the global thresholding methods.

If you specify `nbcfs`, you cannot specify `threshold`, `percfs`, `bpp`, or `comprat`.

Data Types: `single` | `double`

`percfs` — Percentage of preserved coefficients
real number

Percentage of preserved coefficients in the wavelet decomposition, specified as the comma-separated pair consisting of `'percfs'` and real number greater than or equal to 0 and less than or equal to 100.

`percfs` is valid only for the global thresholding methods.

If you specify `percfs`, you cannot specify `threshold`, `nbcfs`, `bpp`, or `comprat`.

Data Types: `single` | `double`

Display Parameter

`plotpar` — Plot parameter
`'plot'` or 0 | `'step'` or 1

Plot parameter, specified as the comma-separated pair consisting of `'plotpar'` and one of the values listed:

- `'plot'` or 0 — plot only the compressed image
- `'step'` or 1 — display each step of the encoding process (only for PCSM methods)

Output Arguments

`comprat` — Compression ratio
scalar

Compression ratio, returned as a scalar.

`bpp` — Bit-per-pixel ratio
scalar

Bit-per-pixel ratio, returned as a scalar.

`xc` — Uncompressed image
2-D array | 3-D array

Uncompressed image, returned as a 2-D array containing either an indexed image or a 3-D array of `uint8` containing a truecolor image.

References

[1] Christophe, Emmanuel, Pierre Duhamel, and Corinne Mailhes. “Adaptation of Zerotrees Using Signed Binary Digit Representations for 3D Image Coding.” *EURASIP Journal on Image and Video Processing* 2007, no. 1 (2007): 054679. https://doi.org/10.1186/1687-5281-2007-054679.

[2] Misiti, Michel, Yves Misiti, Georges Oppenheim, and Jean-Michel Poggi, eds. *Wavelets and Their Applications*. London, UK: ISTE, 2007. https://doi.org/10.1002/9780470612491.

[3] Said, A., and W.A. Pearlman. “A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees.” *IEEE Transactions on Circuits and Systems for Video Technology* 6, no. 3 (June 1996): 243–50. https://doi.org/10.1109/76.499834.

[4] Shapiro, J.M. “Embedded Image Coding Using Zerotrees of Wavelet Coefficients.” *IEEE Transactions on Signal Processing* 41, no. 12 (December 1993): 3445–62. https://doi.org/10.1109/78.258085.

[5] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Rev. ed. Wellesley, Mass: Wellesley-Cambridge Press, 1997.

[6] Walker, James S. "Wavelet-Based Image Compression." Sub-chapter in *Transform and Data Compression. A Primer on Wavelets and Their Scientific Applications*. Vol. 29. Studies in Advanced Mathematics. CRC Press, 1999. https://doi.org/10.1201/9781420050011.

See Also

`imread` | `imwrite` | `wmaxlev` | `tempdir` | `path`

Topics

“Wavelet Compression for Images”

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 denoising 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 denoised 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 denoising.

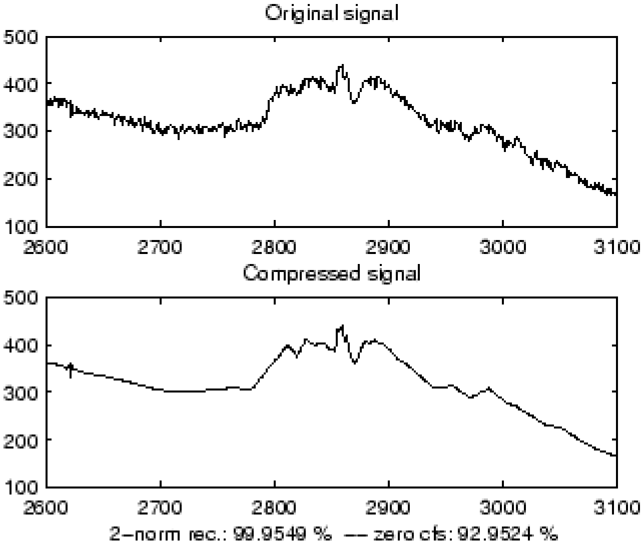
A default value for `M` is `M = L`(1), the number of the coarsest approximation coefficients, since the previous formula leads for i = j+1, to n_{j+1} = `M = L`(1). Recommended values for `M` are from `L`(1) to 2*`L`(1).

`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

wdenoise | 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.

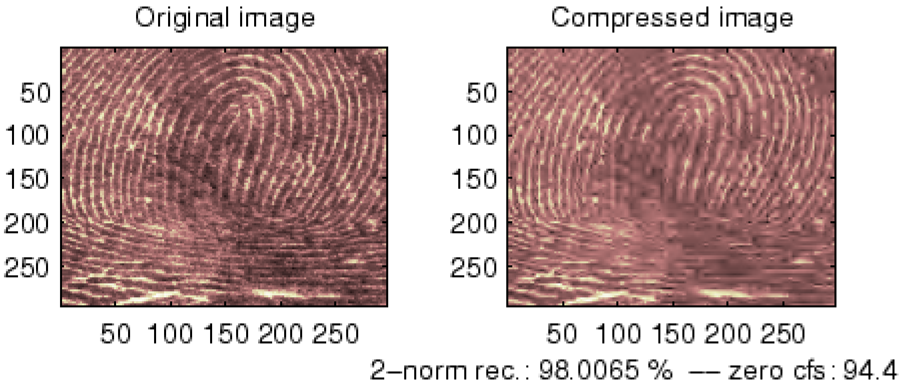
A default value for `M` is `M` = `prod(S(1,:))`, the length of the coarsest approximation coefficients, since the previous formula leads for i = j+1, to n_{j+1} = `M` = `prod(S(1,:))`.

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

Multisignal 1-D Decomposition Energy Distribution

Load the 23 channel EEG data `Espiga3` [1]. The channels are arranged column-wise. The data is sampled at 200 Hz.

```
load Espiga3
```

Perform a decomposition at level 2 using the `db2` wavelet.

```
dec = mdwtdec('c',Espiga3,2,'db2')
```

```
dec = struct with fields:
        dirDec: 'c'
         level: 2
         wname: 'db2'
    dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
      dwtShift: 0
      dataSize: [995 23]
            ca: [251x23 double]
            cd: {[499x23 double]  [251x23 double]}
```

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) in the second channel.

```
idx = 2;
e(idx)
```

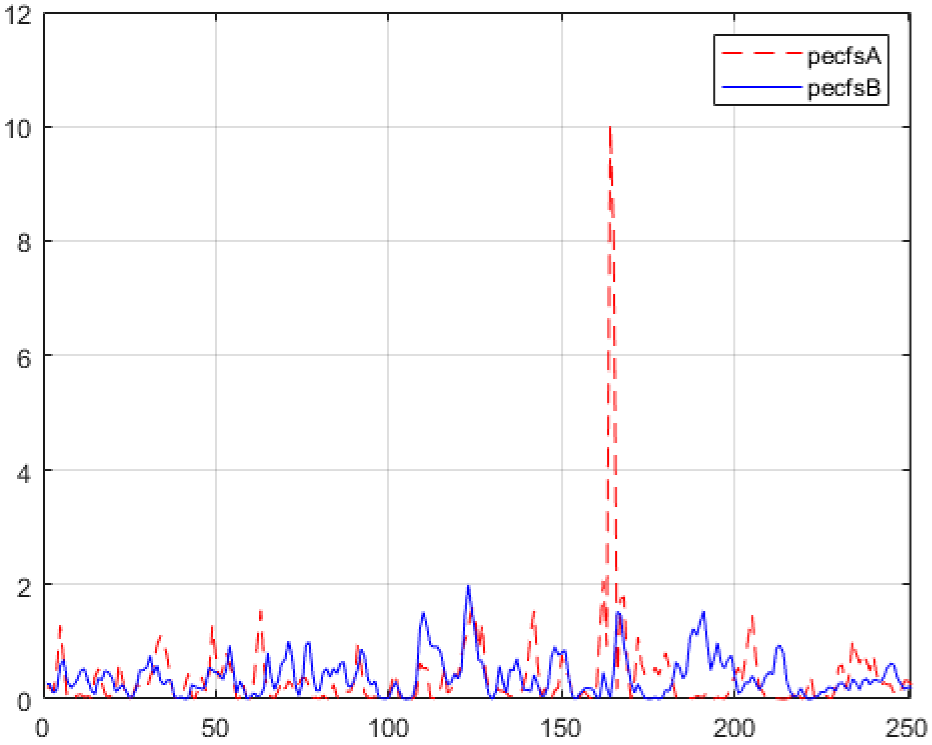
```
ans = 8.0761e+05
```

```
perA2D2D1 = pec(idx,:)
```

```
perA2D2D1 = 1×3

   99.0583    0.8535    0.0882
```

Compare the coefficient energy distribution for signal 1 and signal 10. Because most of the energy is in the approximation coefficients, zoom in the x-axis by the number of approximation coefficients.

```
sigA = 1;
sigB = 10;
pecfsA = pecfs(sigA,:);
pecfsB = pecfs(sigB,:);
plot(pecfsA,'r--')
hold on
plot(pecfsB,'b')
grid on
legend('pecfsA','pecfsB')
xlim([0 size(dec.ca,1)])
```



References

[1] Mesa, Hector. “Adapted Wavelets for Pattern Detection.” In *Progress in Pattern Recognition, Image Analysis and Applications*, edited by Alberto Sanfeliu and Manuel Lazo Cortés, 3773:933–44. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. https://doi.org/10.1007/11578079_96.

See Also

`mdwtdec` | `mdwtrec`

Introduced in R2012a

wden

Automatic 1-D denoising

Note wden is no longer recommended. Use wdenoise instead.

Syntax

```
XD = wden(X,TPTR,SORH,SCAL,N,wname)
XD = wden(C,L, ___ )
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

XD = wden(X,TPTR,SORH,SCAL,N,wname) returns a denoised version XD of the signal X. The function uses an N-level wavelet decomposition of X using the specified orthogonal or biorthogonal wavelet wname to obtain the wavelet coefficients. The thresholding selection rule TPTR is applied to the wavelet decomposition. SORH and SCAL define how the rule is applied.

XD = wden(C,L, ___) returns a denoised version XD of the signal X using the same options as in the previous syntax, but obtained directly from the wavelet decomposition structure [C,L] of X. [C,L] is the output of wavedec.

XD = wden(W,'modwtsqtwolog',SORH,'mln',N,wname) returns the denoised signal XD obtained by operating on the maximal overlap discrete wavelet transform (MODWT) matrix W, where W is the output of modwt. You must use the same orthogonal wavelet in both modwt and wden.

[XD,CXD] = wden(___) returns the denoised wavelet coefficients. For discrete wavelet transform (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 of CXD 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 denoised 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 the 'modwtsqtwolog' input argument.

Examples

Automatic 1-D Denoising Using Wavelets

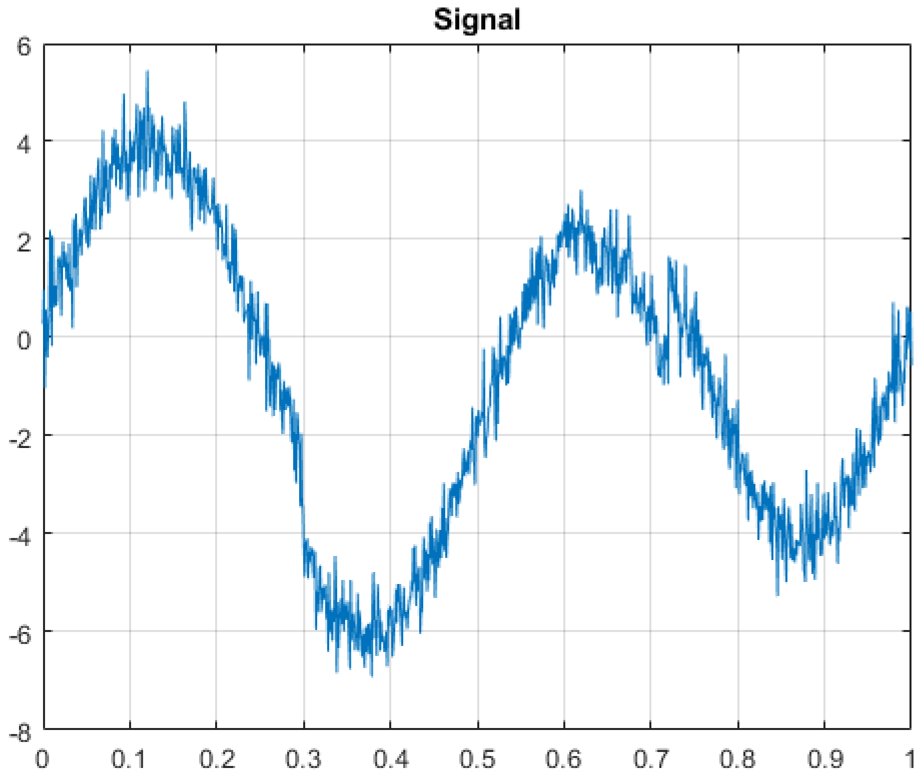
This example shows how to apply three different denoising techniques to a noisy signal. It compares the results with plots and the threshold values produced by each technique.

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')
grid on
```



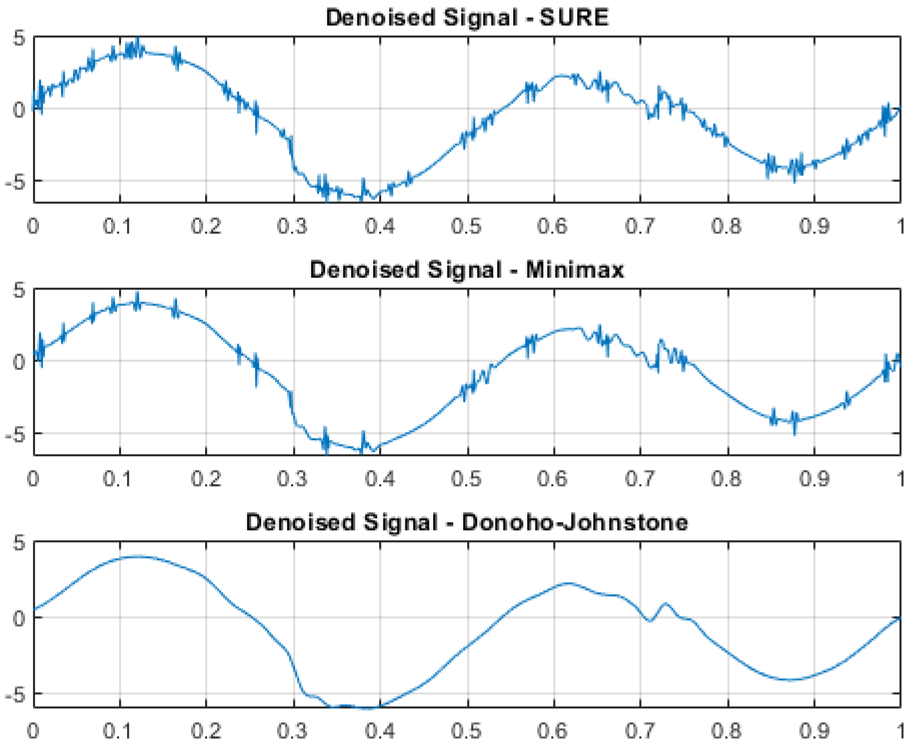
Using the sym8 wavelet, perform a level 5 wavelet decomposition of the signal and denoise it by applying three different threshold selection rules to the wavelet coefficients: SURE, minimax, and Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. In each case, apply 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')
grid on
subplot(3,1,2)
plot(t,dnsig2)
title('Denoised Signal - Minimax')
grid on
subplot(3,1,3)
plot(t,dnsig3)
title('Denoised Signal - Donoho-Johnstone')
grid on
```



Compare the thresholds applied at each detail level for the three denoising methods.

```
threshold_SURE
```

```
threshold_SURE = 1×5

    0.9592    0.6114    1.4734    0.7628    0.4360
```

```
threshold_Minimax
```

```
threshold_Minimax = 1×5

    1.1047    1.0375    1.3229    1.1245    1.0483
```

```
threshold_DJ
```

```
threshold_DJ = 1×5

    1.8466    1.7344    2.2114    1.8798    1.7524
```

Compare DWT and MODWT Denoising of a Sinusoid with Two Jumps

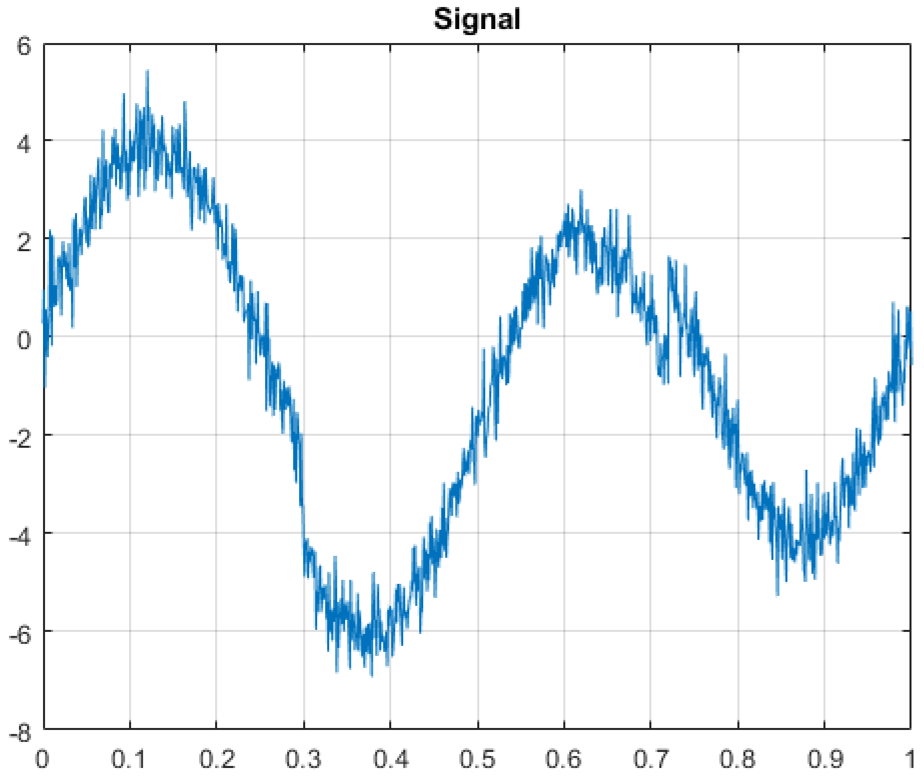
This example denoises a signal using the DWT and MODWT. It compares the results with plots and the threshold values produced by each technique.

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')
grid on
```

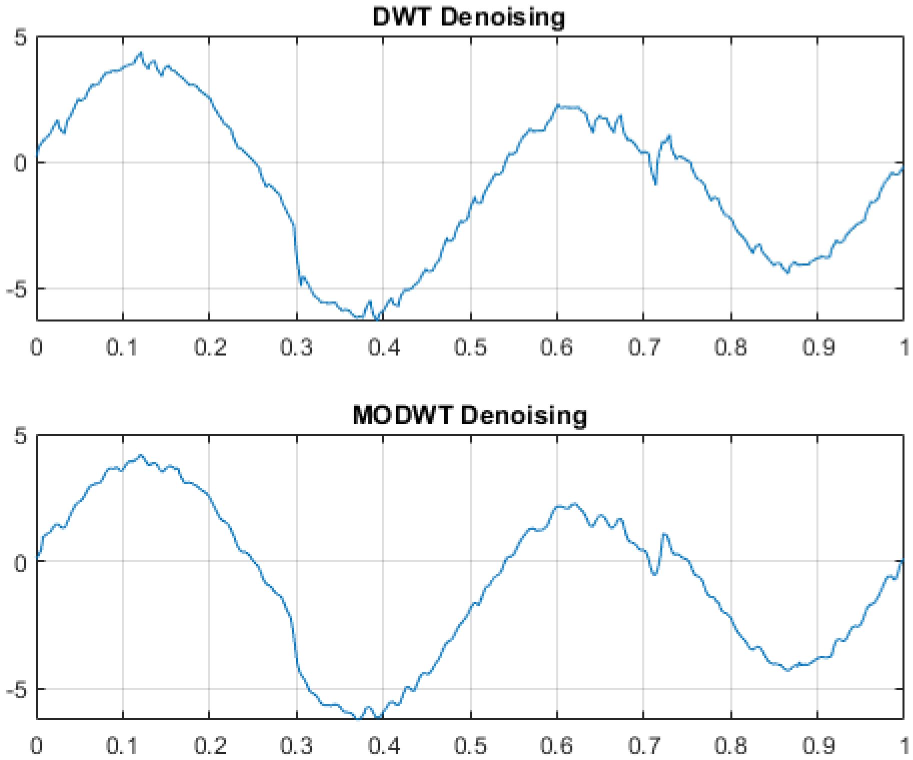


Using the db2 wavelet, perform a level 3 wavelet decomposition of the signal and denoise it using Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. Obtain denoised versions using DWT and MODWT, both with soft thresholding.

```
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)
grid on
title('DWT Denoising')
subplot(2,1,2)
plot(t,xdMODWT)
grid on
title('MODWT Denoising')
```



Compare the thresholds applied in each case.

```
threshold_DWT
```

```
threshold_DWT = 1×3

    1.7783    1.6876    2.0434
```

```
threshold_MODWT
```

```
threshold_MODWT = 1×3

    1.2760    0.6405    0.3787
```

Compare DWT and MODWT Denoising of a Blocky Signal

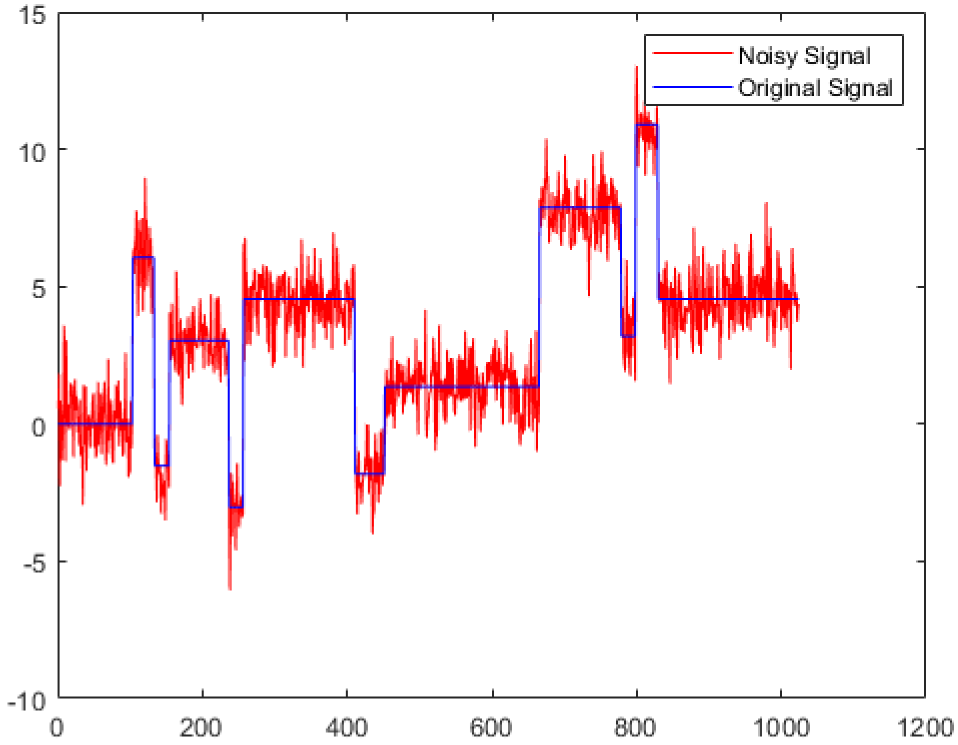
This example denoises a blocky signal using the Haar wavelet with DWT and MODWT denoising. It compares the results with plots and metrics for the original and denoised versions.

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 Signal','Original Signal')
```

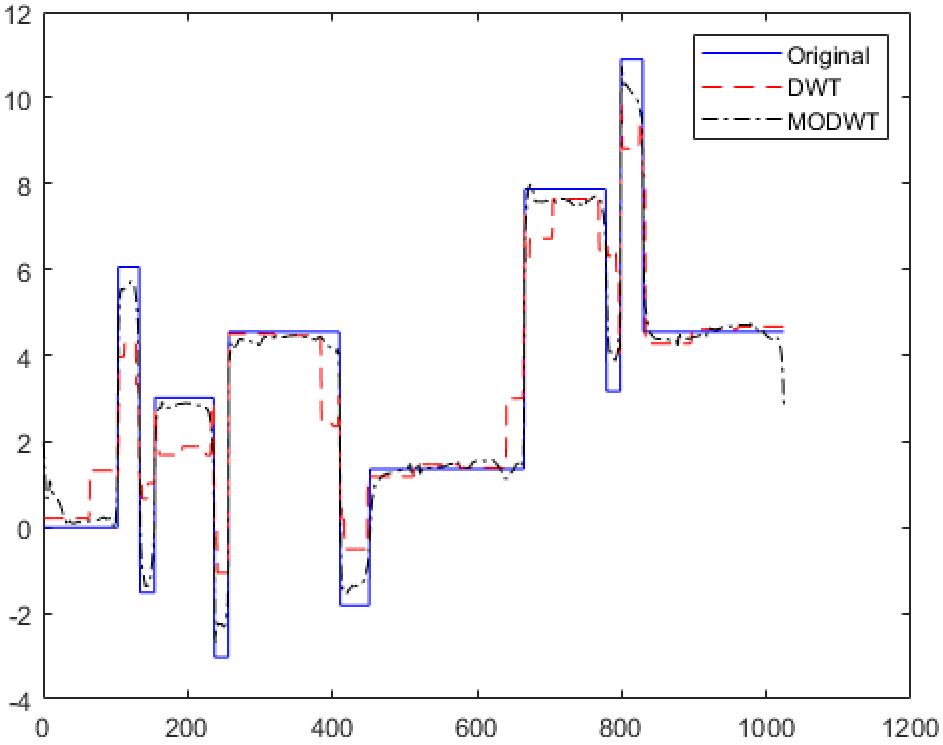


Using the Haar wavelet, perform a level 6 wavelet decomposition of the noisy signal and denoise it using Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. Obtain denoised versions using DWT and MODWT, both with soft thresholding.

```
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
```

Input Arguments

X — Input data

real-valued vector

Input data to denoise, specified as a real-valued vector.

Data Types: `double`

C — Wavelet expansion coefficients

real-valued vector

Wavelet expansion coefficients of the data to be denoised, specified as a real-valued vector. C is the output of wavedec.

Example: [C,L] = wavedec(randn(1,1024),3,'db4')

Data Types: double

L — Size of wavelet expansion coefficients

vector of positive integers

Size of wavelet expansion coefficients of the signal to be denoised, specified as a vector of positive integers. L is the output of wavedec.

Example: [C,L] = wavedec(randn(1,1024),3,'db4')

Data Types: double

W — Maximal overlap wavelet decomposition structure

real-valued matrix

Maximal overlap wavelet decomposition structure of the signal to denoise, specified as a real-valued matrix. W is the output of modwt. You must use the same orthogonal wavelet in both modwt and wden.

Data Types: double

TPTR — Threshold selection rule

character array

Threshold selection rule to apply to the wavelet decomposition structure of X:

- 'rigsure' — Use the principle of Stein's Unbiased Risk.
- 'heursure' — Use a heuristic variant of Stein's Unbiased Risk.
- 'sqtwolog — Use the universal threshold $\sqrt{2\ln(\text{length}(x))}$.
- 'minimaxi' — Use minimax thresholding. (See thselect for more information.)

SORH — Type of thresholding

's' | 'h'

Type of thresholding to perform:

- 's' — Soft thresholding
- 'h' — Hard thresholding

SCAL — Multiplicative threshold rescaling

'one' | 'sln' | 'mln'

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

N — Level of wavelet decomposition

positive integer

Level of wavelet decomposition, specified as a positive integer. Use wmaxlev to ensure that 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))).

wname — Name of wavelet

character array

Name of wavelet, specified as a character array, to use for denoising. For DWT denoising, the wavelet must be orthogonal or biorthogonal. For MODWT denoising, the wavelet must be orthogonal. 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 biorthogonal 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).

Output Arguments

XD — Denoised signal

real-valued vector

Denoised data, returned as a real-valued vector.

Data Types: double

CXD — Denoised wavelet coefficients

real-valued vector or matrix

Denoised wavelet coefficients, returned as a real-valued vector or matrix. 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.

Data Types: double

LXD — Size of denoised wavelet coefficients

vector of positive integers

Size of denoised wavelet coefficients by level for DWT denoising, returned as a vector of positive integers (see wavedec). The LXD output is not supported for MODWT denoising. [CXD,LXD] is the wavelet decomposition structure of the denoised signal XD.

Data Types: double

THR — Denoising thresholds
real-valued vector

Denoising thresholds by level, returned as a length `N` real-valued vector.

Data Types: `double`

Algorithms

The most general model for the noisy signal has 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 denoising objective is to suppress the noise part of the signal s and to recover f.

The denoising procedure has 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” and in the help of the `thselect` function. Note that:

- The detail coefficients vector is the superposition of the coefficients of f and the coefficients of e. The decomposition of e leads to detail coefficients that are standard Gaussian white noises.
- Minimax and SURE threshold selection rules are more conservative and 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. To deal with model deviations, the remaining parameter `scal` must be specified. It corresponds to threshold rescaling methods.

- The option `scal = 'one'` corresponds to the basic model.
- The option `scal = 'sln'` handles threshold rescaling using a single estimation of level noise based on the first-level coefficients.

  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. If level 1 coefficients contain f details, these details are concentrated in a few coefficients to avoid signal end effects, which are pure artifacts due to computations on the edges.
- The option `scal = 'mln'` handles threshold rescaling using a level-dependent estimation of the level noise.

  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.

References

[1] Antoniadis, A., and G. Oppenheim, eds. *Wavelets and Statistics*, 103. Lecture Notes in Statistics. New York: Springer Verlag, 1995.

[2] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[3] Donoho, D. L., and Johnstone, I. M. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*, Vol. 81, pp. 425–455, 1994.

[4] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*, Vol. 42, Number 3, pp. 613–627, 1995.

[5] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society*, *series B*. Vol. 57, Number 2, pp. 301–369, 1995.

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

Functions

`thselect` | `wavedec` | `wdencmp` | `wfilters` | `wthresh` | `wdenoise` | `wavemngr`

Apps

`Wavelet Signal Denoiser`

Introduced before R2006a

wdencmp

Denoising 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)
[ ___ ] = wdencmp('lvl',X,wname,N,THR,SORH)
[ ___ ] = wdencmp('lvl',C,L,wname,N,THR,SORH)
```

Description

[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('gbl',X,wname,N,THR,SORH,KEEPAPP) returns a denoised or compressed version XC of the input data X obtained by wavelet coefficients thresholding using the global positive threshold THR. X is a real-valued vector or matrix. [CXC,LXC] is the N-level wavelet decomposition structure of XC (see wavedec or wavedec2 for more information). PERFL2 and PERF0 are the L^2-norm recovery and compression scores in percentages, respectively. If KEEPAPP = 1, the approximation coefficients are kept. If KEEPAPP = 0, the approximation coefficients can be thresholded.

[___] = wdencmp('gbl',C,L,wname,N,THR,SORH,KEEPAPP) uses the wavelet decomposition structure [C,L] of the data to be denoised or compressed.

[___] = wdencmp('lvl',X,wname,N,THR,SORH) uses the level-dependent thresholds THR. The approximation coefficients are kept.

[___] = wdencmp('lvl',C,L,wname,N,THR,SORH) uses the wavelet decomposition structure [C,L].

Examples

Denoise 1-D Signal Using Default Global Threshold

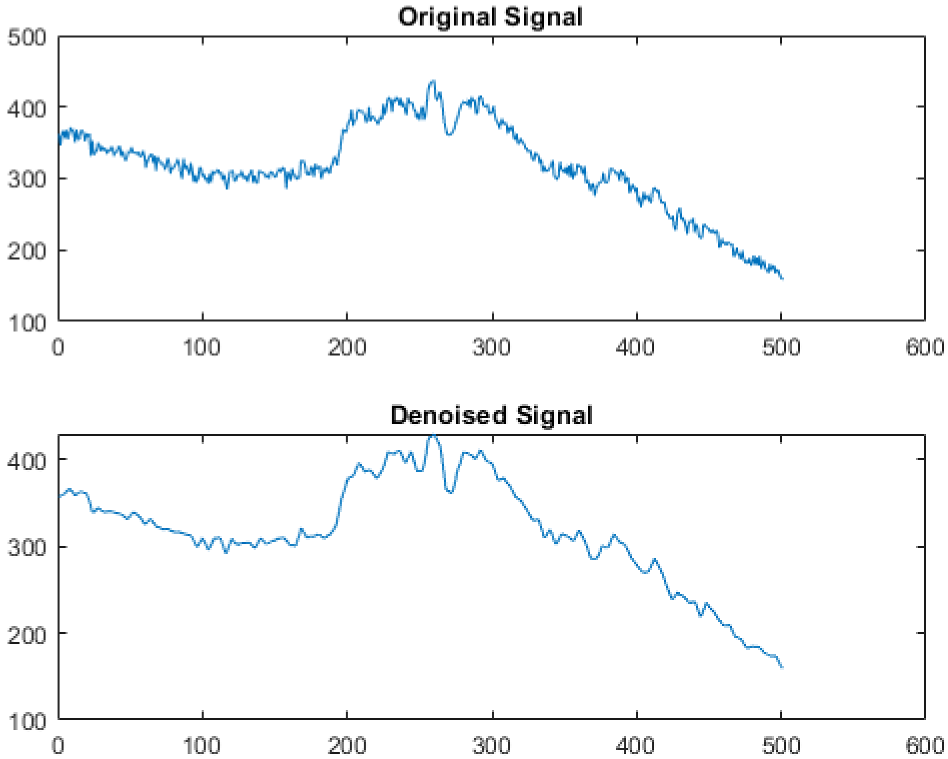
Denoise 1-D electricity consumption data using the Donoho-Johnstone global threshold.

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 Image Using Default Global Threshold

Denoise an image in additive white Gaussian noise using the Donoho-Johnstone universal threshold.

Load an image and add white Gaussian noise.

```
load sinsin
Y = X+18*randn(size(X));
```

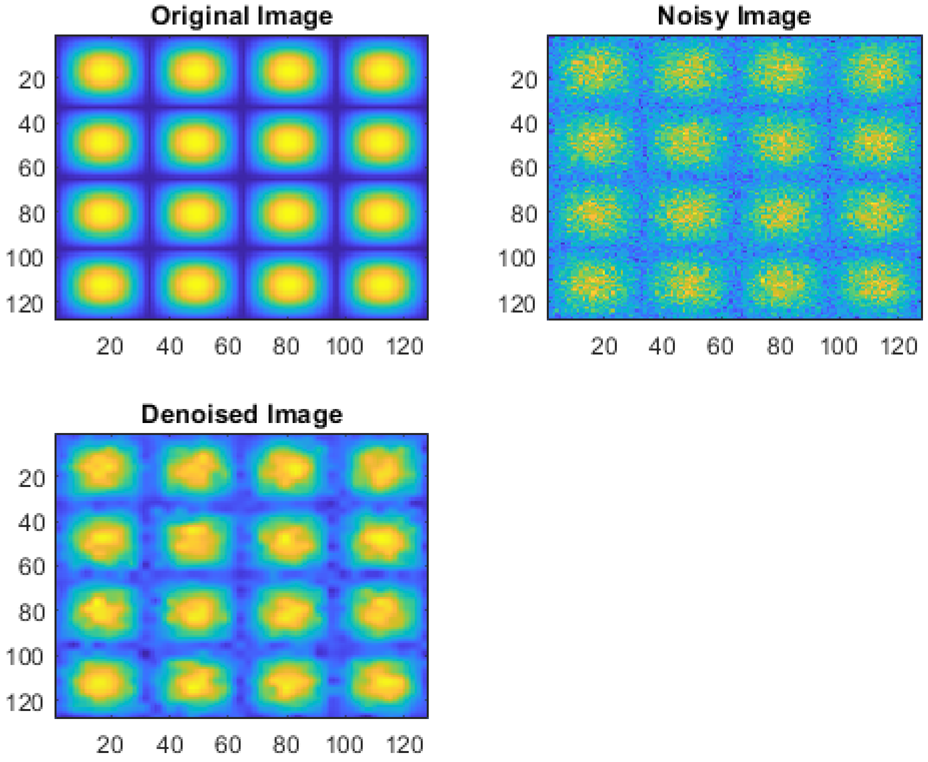
Use `ddencmp` to obtain the threshold.

```
[thr,sorh,keepapp] = ddencmp('den','wv',Y);
```

Denoise the image. Use the order 4 Symlet and a two-level wavelet decomposition. Plot the original image, the noisy image, and the denoised result.

```
xd = wdencmp('gbl',Y,'sym4',2,thr,sorh,keepapp);
subplot(2,2,1)
imagesc(X)
title('Original Image')
subplot(2,2,2)
imagesc(Y)
title('Noisy Image')
subplot(2,2,3)
```

```
imagesc(xd)
title('Denoised Image')
```



Input Arguments

X — Input data

real-valued vector | real-valued matrix

Input data to denoise or compress, specified by a real-valued vector or matrix.

Data Types: `double`

C — Wavelet expansion coefficients

real-valued vector

Wavelet expansion coefficients of the data to be compressed or denoised, specified as a real-valued vector. If the data is one-dimensional, `C` is the output of `wavedec`. If the data is two-dimensional, `C` is the output of `wavedec2`.

Example: `[C,L] = wavedec(randn(1,1024),3,'db4')`

Data Types: `double`

L — Size of wavelet expansion coefficients

vector of positive integers | matrix of positive integers

Size of wavelet expansion coefficients of the signal or image to be compressed or denoised, specified as a vector or matrix of positive integers.

For signals, `L` is the output of `wavedec`. For images, `L` is the output of `wavedec2`.

Example: `[C,L] = wavedec(randn(1,1024),3,'db4')`

Data Types: `double`

`wname` — Name of wavelet
character vector | string scalar

Name of wavelet, specified as a character vector or string scalar, to use for denoising or compression. See `wavemngr` for more information. `wdencmp` uses `wname` to generate the `N`-level wavelet decomposition of `X`.

`N` — Level of wavelet decomposition
positive integer

Level of wavelet decomposition, specified as a positive integer.

`THR` — Threshold
scalar | real-valued vector | real-valued matrix

Threshold to apply to the wavelet coefficients, specified as a scalar, real-valued vector, or real-valued matrix.

- For the case `'gbl'`, `THR` is a scalar.
- For the one-dimensional case and `'lvd'` option, `THR` is a length `N` real-valued vector containing the level-dependent thresholds.
- For the two-dimensional case and `'lvd'` option, `THR` is a 3-by-`N` matrix containing the level-dependent thresholds in the three orientations: horizontal, diagonal, and vertical.

Data Types: `double`

`SORH` — Type of thresholding
`'s'` | `'h'`

Type of thresholding to perform:

- `'s'` — Soft thresholding
- `'h'` — Hard thresholding

See `wthresh` for more information.

`KEEPAPP` — Threshold approximation setting
`0` | `1`

Threshold approximation setting, specified as either `0` or `1`. If `KEEPAPP = 1`, the approximation coefficients cannot be thresholded. If `KEEPAPP = 0`, the approximation coefficients can be thresholded.

Data Types: `double`

Output Arguments

XC — Denoised or compressed data

real-valued vector | real-valued matrix

Denoised or compressed data, returned as a real-valued vector or matrix. XC and X have the same dimensions.

CXC — Wavelet expansion coefficients

real-valued vector

Wavelet expansion coefficients of the denoised or compressed data XC, returned as a real-valued vector. LXC contains the number of coefficients by level.

LXC — Size of wavelet expansion coefficients

vector of positive integers | matrix of positive integers

Size of wavelet expansion coefficients of the denoised or compressed data XC, specified as a vector or matrix of positive integers. If the data is one-dimensional, LXC is a vector of positive integers (see wavedec for more information). If the data is two-dimensional, LXC is a matrix of positive integers (see wavedec2 for more information).

PERF0 — Compression score

scalar

Compression score, returned as a real number. PERF0 is the percentage of thresholded coefficients that are equal to 0.

PERFL2 — L^2 energy recovery

scalar

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

$$\frac{100\|XC\|^2}{\|X\|^2}$$

Algorithms

The denoising and compression procedures contain three steps:

1. Decomposition.
2. Thresholding.
3. Reconstruction.

The two procedures differ in Step 2. In compression, for each level in the wavelet decomposition, a threshold is selected and hard thresholding is applied to the detail coefficients.

References

[1] DeVore, R. A., B. Jawerth, and B. J. Lucier. “Image Compression Through Wavelet Transform Coding.” *IEEE Transactions on Information Theory*. Vol. 38, Number 2, 1992, pp. 719–746.

[2] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[3] Donoho, D. L., and I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*. Vol. 81, pp. 425–455, 1994.

[4] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society, series B*, Vol. 57, No. 2, pp. 301–369, 1995.

[5] Donoho, D. L., and I. M. Johnstone. “Ideal denoising in an orthonormal basis chosen from a library of bases.” *C. R. Acad. Sci. Paris, Ser. I*, Vol. 319, pp. 1317–1322, 1994.

[6] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*. Vol. 42, Number 3, pp. 613–627, 1995.

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

Functions

`ddencmp` | `wavedec` | `wavedec2` | `wbmpen` | `wcompress` | `wdcbm2` | `wpdencmp` | `wthresh` | `wdenoise`

Apps

`Wavelet Signal Denoiser`

Introduced before R2006a

wdenoise

Wavelet signal denoising

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(log`$_2$`N)` 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(log`$_2$`N)` where *N* is the number of samples in the data. If unspecified, `LEVEL` defaults to the minimum of `floor(log`$_2$`N)` and `wmaxlev(N,"sym4")`.

`XDEN = wdenoise( ___ ,Name,Value)` specifies one or more options using name-value pair arguments in addition to any of the input arguments in previous syntaxes. For example, `xden = wdenoise(x,3,"Wavelet","db2")` denoises `x` down to level 3 using the Daubechies `db2` wavelet.

`[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

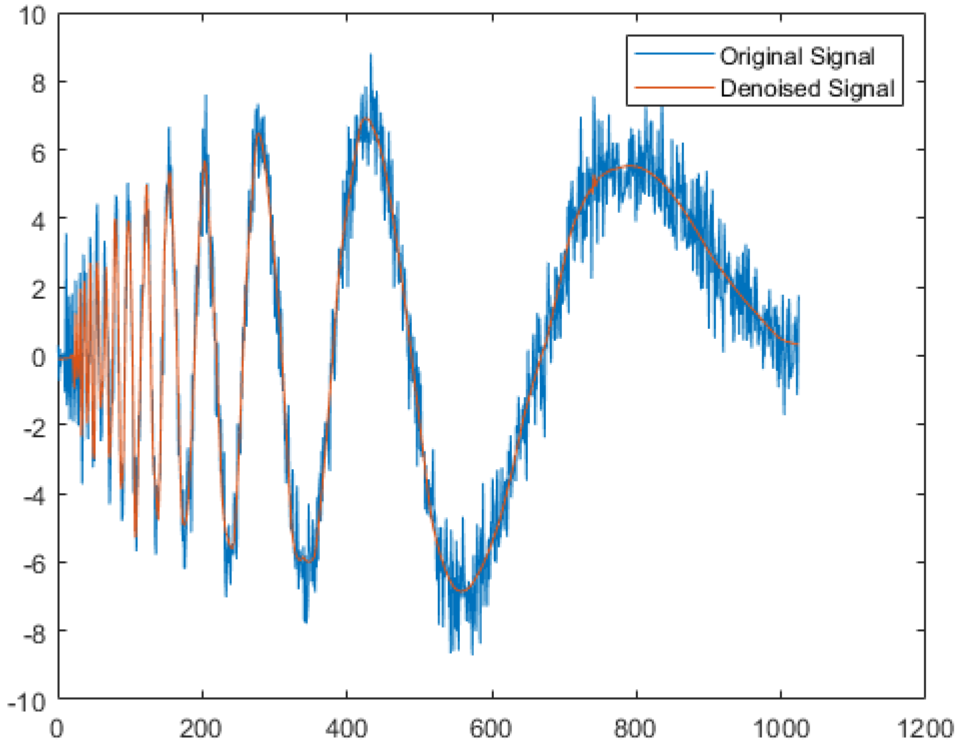
Denoise A Signal Using Default Values

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.

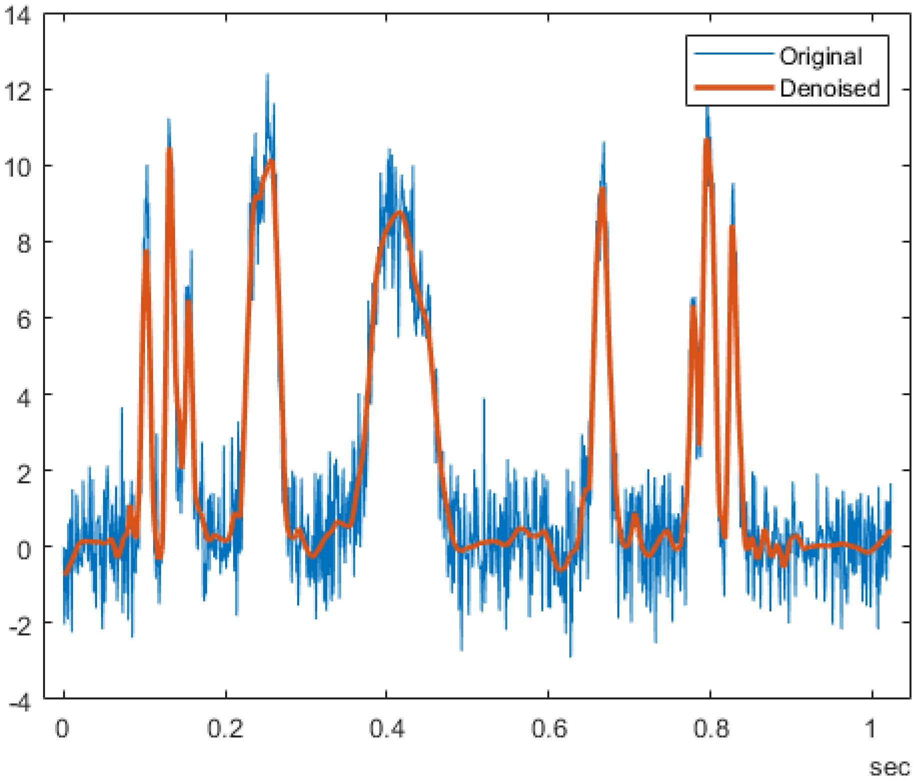
```
load wnoisydata
```

Denoise the data down to level 5 using block thresholding

```
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")
```

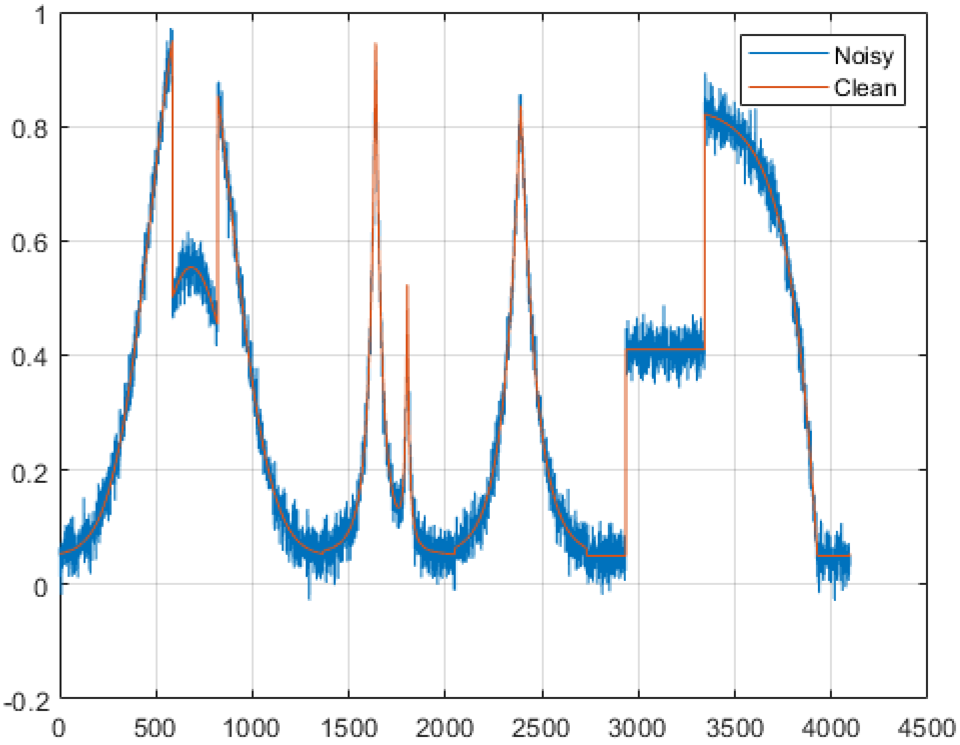


Compare Denoised Signals

Denoise a signal in different ways and compare results.

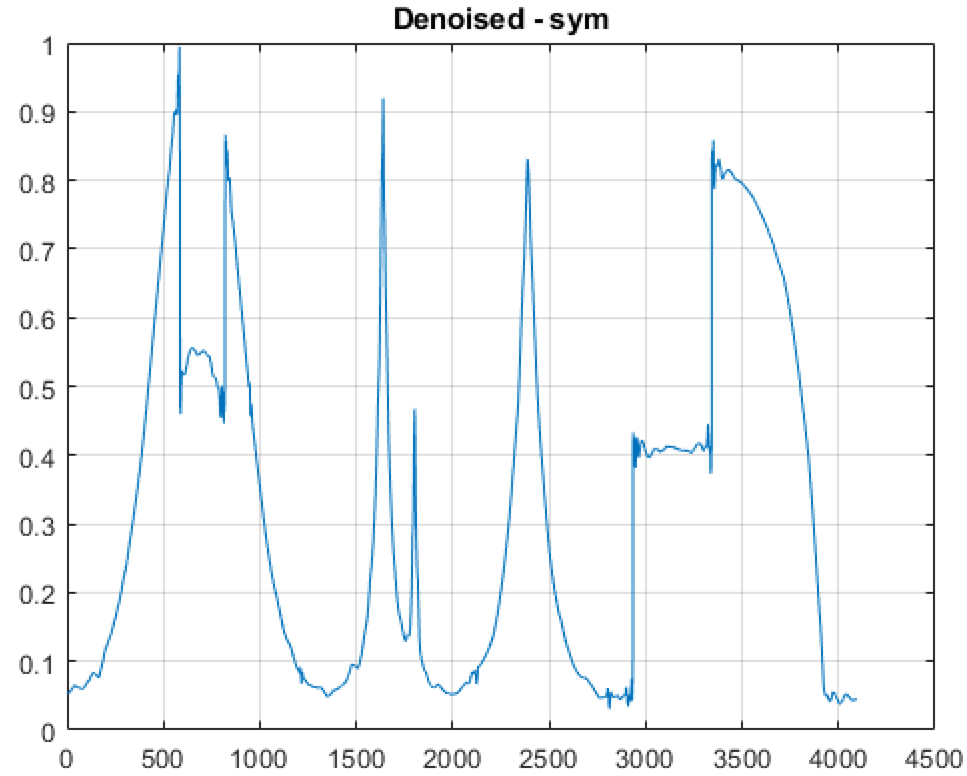
Load a datafile that contains clean and noisy versions of a signal. Plot the signals.

```
load fdata.mat
plot(fNoisy)
hold on
plot(fClean)
grid on
legend("Noisy","Clean")
hold off
```

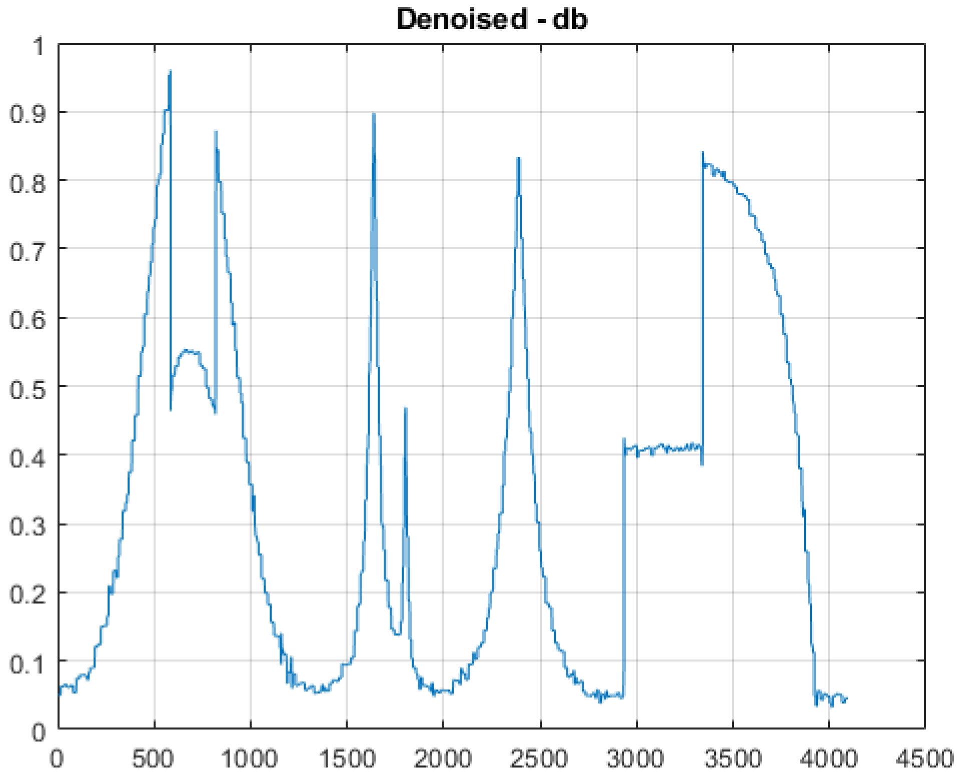


Denoise the signal using the `sym4` and `db1` wavelets, with a nine-level wavelet decomposition. Plot the results.

```
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 = 35.9623
```

```
snrdb = -20*log10(norm(abs(fClean-cleandb))/norm(fClean))
```

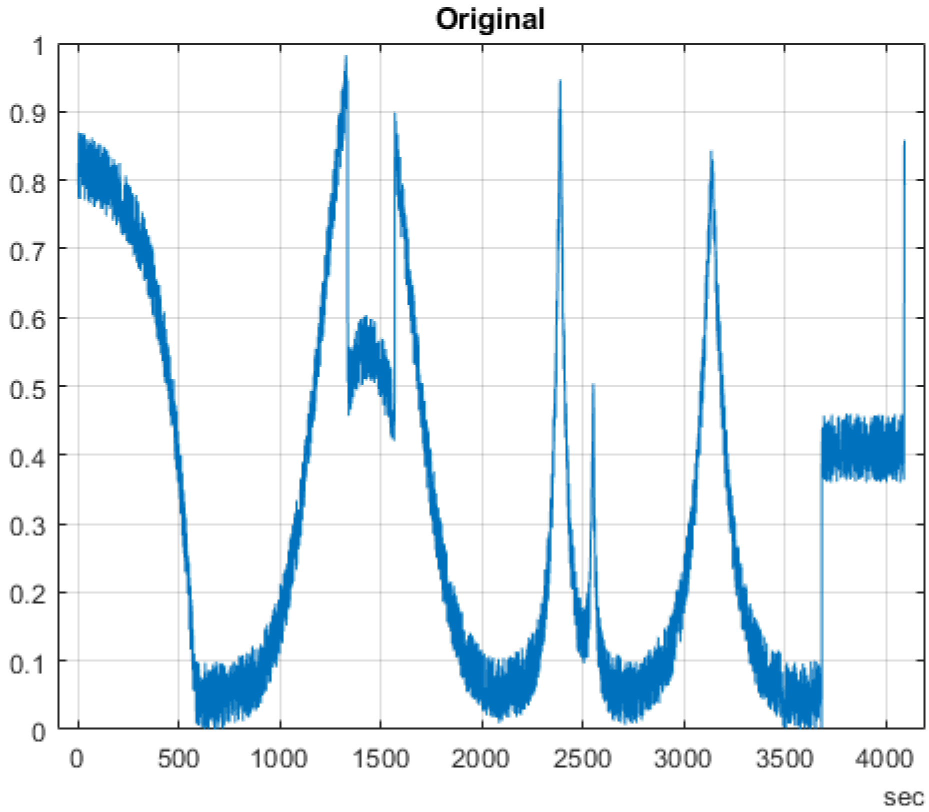
```
snrdb = 32.2672
```

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.

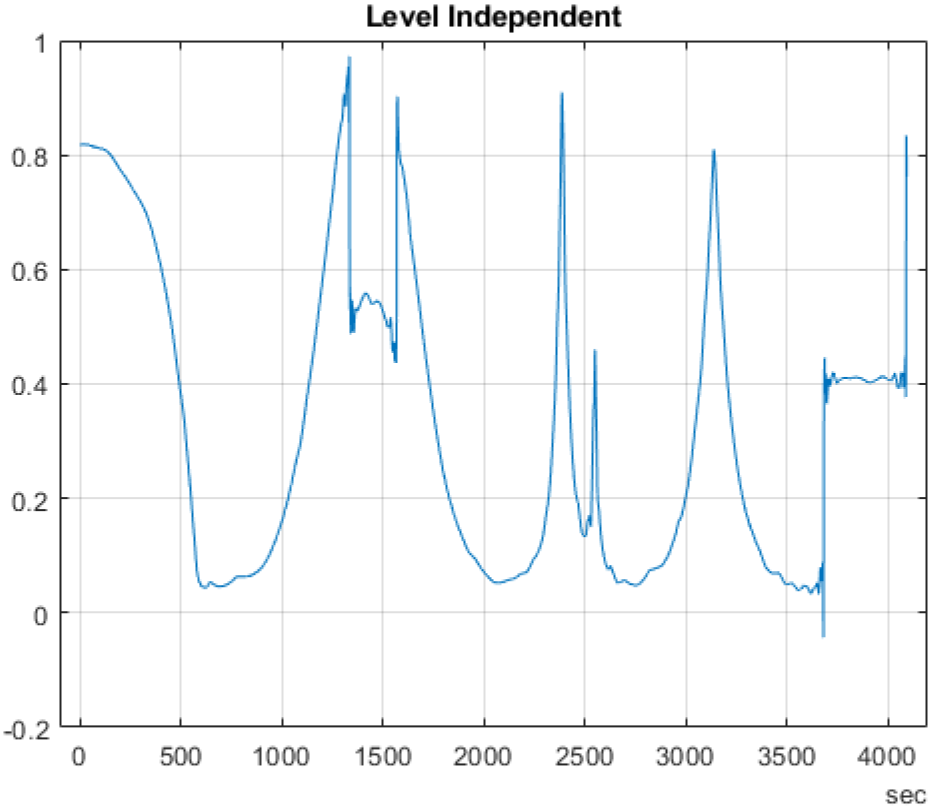
```
load fdataTS.mat
cleanTSld = wdenoise(fdataTS,9,NoiseEstimate="LevelDependent");
cleanTSli = wdenoise(fdataTS,9,NoiseEstimate="LevelIndependent");
```

Compare one of the noisy time series with its two denoised versions.

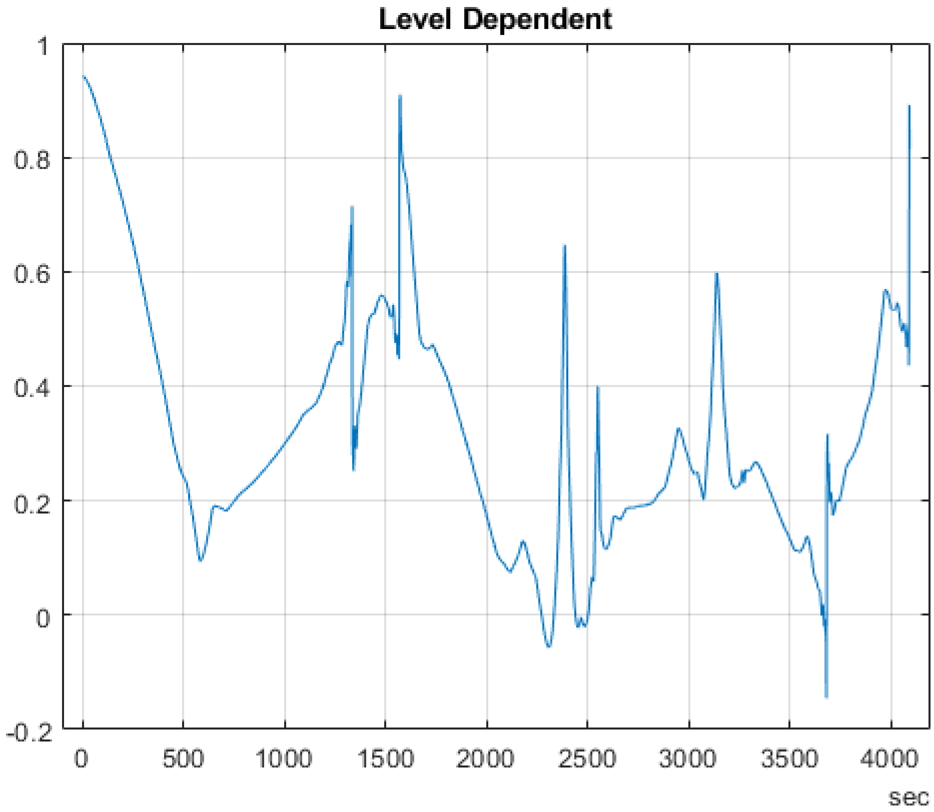
```
figure
plot(fdataTS.Time,fdataTS.fTS15)
title("Original")
grid on
```



```
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($\log_2$N) where *N* is the number of samples in the data.

- If unspecified, LEVEL defaults to the minimum of floor($\log_2 N$) and wmaxlev(N,"sym4").
- For James-Stein block thresholding, "BlockJS", there must be floor($\log_2$N) coefficients at the coarsest resolution level, LEVEL.

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `xden = wdenoise(x,4,Wavelet="db6")` denoises `x` down to level 4 using the Daubechies `db6` wavelet.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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 biorthogonal 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 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 $\sqrt{2\ln(\mathrm{length}(x))}$.

  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.

- `"LevelIndependent"` — Estimate the variance of the noise based on the finest-scale (highest-resolution) wavelet coefficients.
- `"LevelDependent"` — Estimate the variance of the noise based on the wavelet coefficients at each resolution level.

Specifying `NoiseEstimate` with the `"BlockJS"` denoising method has no effect. The block James-Stein estimator always uses a `"LevelIndependent"` noise estimate.

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 most general model for the noisy signal has 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 denoising objective is to suppress the noise part of the signal s and to recover f.

The denoising procedure has 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” and in the help of the `thselect` function.

References

[1] Abramovich, F., Y. Benjamini, D. L. Donoho, and I. M. Johnstone. “Adapting to Unknown Sparsity by Controlling the False Discovery Rate.” *Annals of Statistics*, Vol. 34, Number 2, pp. 584–653, 2006.

[2] Antoniadis, A., and G. Oppenheim, eds. *Wavelets and Statistics*. Lecture Notes in Statistics. New York: Springer Verlag, 1995.

[3] Cai, T. T. “On Block Thresholding in Wavelet Regression: Adaptivity, Block size, and Threshold Level.” *Statistica Sinica*, Vol. 12, pp. 1241–1273, 2002.

[4] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[5] Donoho, D. L., I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*, Vol. 81, pp. 425–455, 1994.

[6] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*, Vol. 42, Number 3, pp. 613–627, 1995.

[7] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society*, *series B*, Vol. 57, No. 2, pp. 301–369, 1995.

[8] Johnstone, I. M., and B. W. Silverman. “Needles and Straw in Haystacks: Empirical Bayes Estimates of Possibly Sparse Sequences.” *Annals of Statistics*, Vol. 32, Number 4, pp. 1594–1649, 2004.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Timetable input data is not supported.
- The value of the `"Wavelet"` name-value pair argument must be constant.
- The input `LEVEL` must be defined as a scalar during compilation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Timetable input data is not supported.
- The value of the `"Wavelet"` name-value pair argument must be constant.
- For optimized GPU code generation, specify `LEVEL` as a compile-time constant.
- The `"Bayes"`, `"UniversalThreshold"`, `"Minimax"`, and `"SURE"` denoising methods support optimized GPU code generation.

See Also

Functions

`waveinfo` | `wavemngr` | `wdenoise2`

Apps

Wavelet Signal Denoiser

Topics

"Denoise a Signal with the Wavelet Signal Denoiser"
"Denoise Signal Using Generated C Code"

Introduced in R2017b

wdenoise2

Wavelet image denoising

Syntax

```
IMDEN = wdenoise2(IM)
IMDEN = wdenoise2(IM,LEVEL)
[IMDEN,DENOISEDCFS] = wdenoise2( ___ )

[IMDEN,DENOISEDCFS,ORIGCFS] = wdenoise2( ___ )
[IMDEN,DENOISEDCFS,ORIGCFS,S] = wdenoise2( ___ )
[IMDEN,DENOISEDCFS,ORIGCFS,S,SHIFTS] = wdenoise2( ___ )

[ ___ ] = wdenoise2( ___ ,Name,Value)

wdenoise2( ___ )
```

Description

`IMDEN = wdenoise2(IM)` denoises the grayscale or RGB image `IM` using an empirical Bayesian method. The `bior4.4` wavelet is used with a posterior median threshold rule. Denoising is down to the minimum of `floor(log2([M N]))` and `wmaxlev([M N],'bior4.4')`, where `M` and `N` are the row and column sizes of the image. `IMDEN` is the denoised version of `IM`.

For RGB images, by default, `wdenoise2` projects the image onto its principal component analysis (PCA) color space before denoising. To denoise an RGB image in the original color space, use the `ColorSpace` name-value pair.

`IMDEN = wdenoise2(IM,LEVEL)` denoises the image `IM` down to resolution level `LEVEL`. `LEVEL` is a positive integer less than or equal to `floor(log2(min([M N])))`, where `M` and `N` are the row and column sizes of the image.

`[IMDEN,DENOISEDCFS] = wdenoise2( ___ )` returns the scaling and denoised wavelet coefficients in `DENOISEDCFS` using any of the preceding syntaxes.

`[IMDEN,DENOISEDCFS,ORIGCFS] = wdenoise2( ___ )` returns the scaling and wavelet coefficients of the input image in `ORIGCFS` using any of the preceding syntaxes.

`[IMDEN,DENOISEDCFS,ORIGCFS,S] = wdenoise2( ___ )` returns the sizes of the approximation coefficients at the coarsest scale along with the sizes of the wavelet coefficients at all scales. `S` is a matrix with the same structure as the `S` output of `wavedec2`.

`[IMDEN,DENOISEDCFS,ORIGCFS,S,SHIFTS] = wdenoise2( ___ )` returns the shifts along the row and column dimensions for cycle spinning. `SHIFTS` is 2-by-`(numshifts+1)`^{2} matrix where each column of `SHIFTS` contains the shifts along the row and column dimension used in cycle spinning and `numshifts` is the value of `CycleSpinning`.

`[ ___ ] = wdenoise2( ___ ,Name,Value)` returns the denoised image with additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

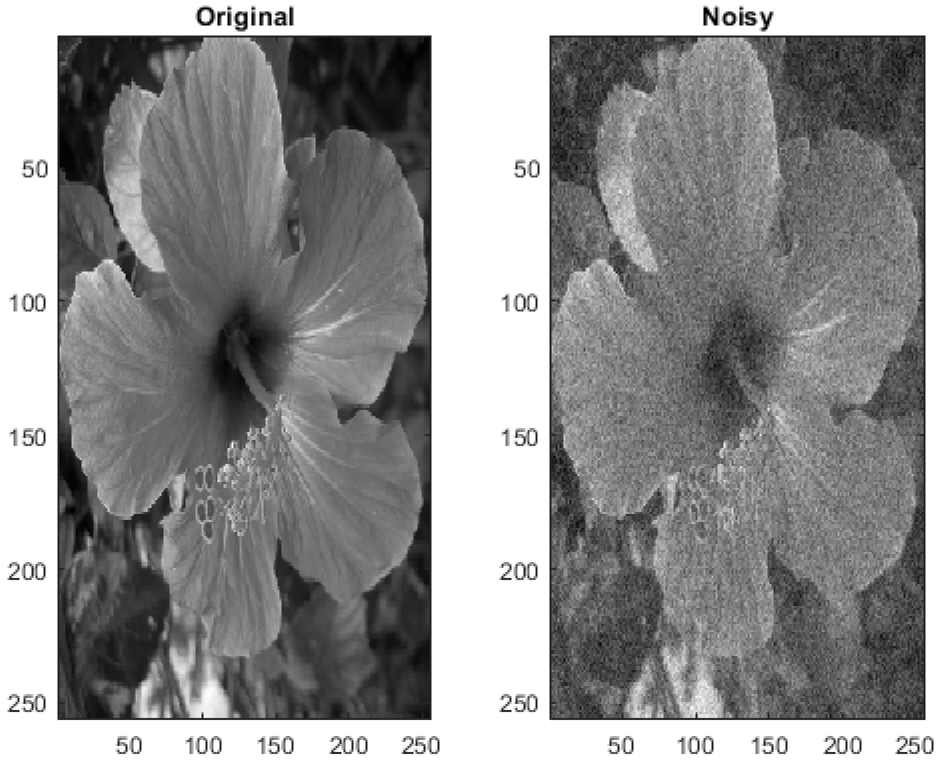
`wdenoise2( ___ )` with no output arguments plots the original image along with the denoised image in the current figure.

Examples

Denoise Grayscale Image Using Default Settings

Load the structure `flower`. The structure contains a grayscale image of a flower, and a noisy version of that image. Display the original and noisy images.

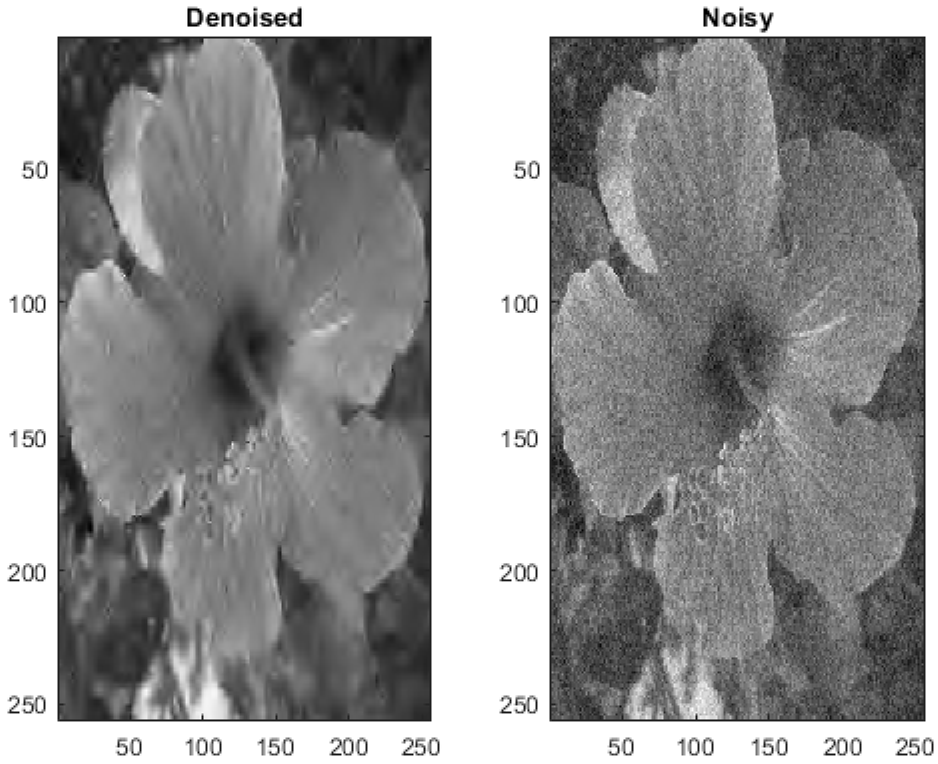
```
load flower
subplot(1,2,1)
imagesc(flower.Orig)
title('Original')
subplot(1,2,2)
imagesc(flower.Noisy)
title('Noisy')
colormap gray
```



Denoise the noisy image using the default `wdenoise2` settings. Compare with the original image.

```
imden = wdenoise2(flower.Noisy);
subplot(1,2,1)
imagesc(imden)
title('Denoised')
```

```
subplot(1,2,2)
imagesc(flower.Noisy)
title('Noisy')
colormap gray
```



Note the improvement in SNR before and after denoising.

```
beforeSNR = ...
    20*log10(norm(flower.Orig(:))/norm(flower.Orig(:)-flower.Noisy(:)))
```

```
beforeSNR = 14.1300
```

```
afterSNR = ...
    20*log10(norm(flower.Orig(:))/norm(flower.Orig(:)-imden(:)))
```

```
afterSNR = 20.1388
```

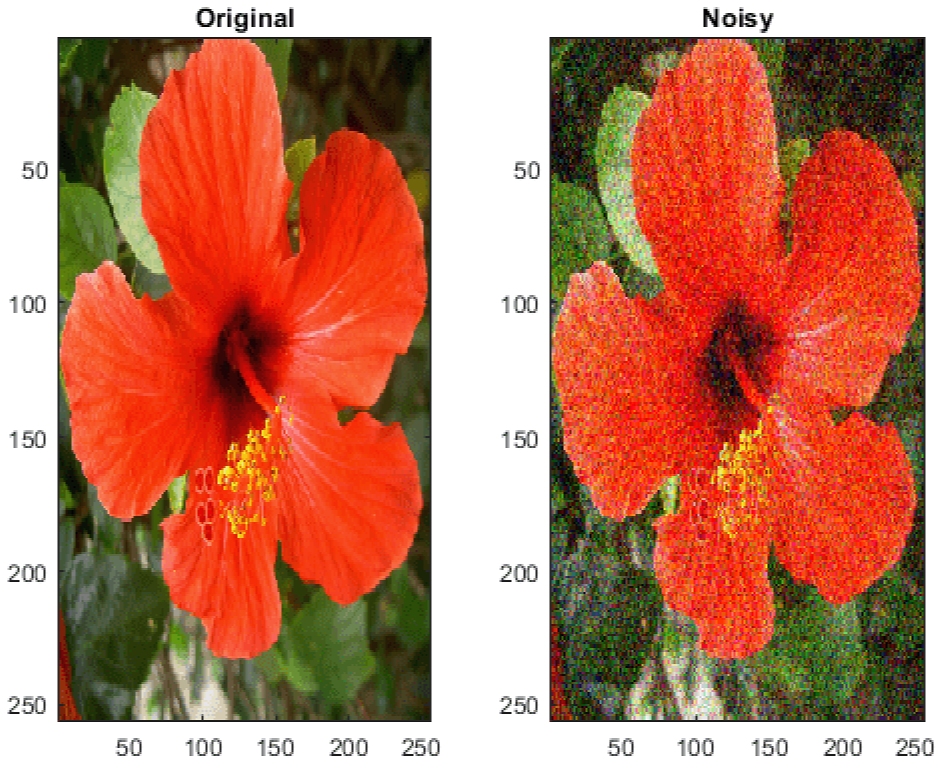
Denoise Color Image Using Cycle Spinning

This example shows how to denoise a color image using cycle spinning.

Load the structure `colorflower`. The structure contains the RGB image of a flower, and a noisy version of that image. Display the original and noisy images.

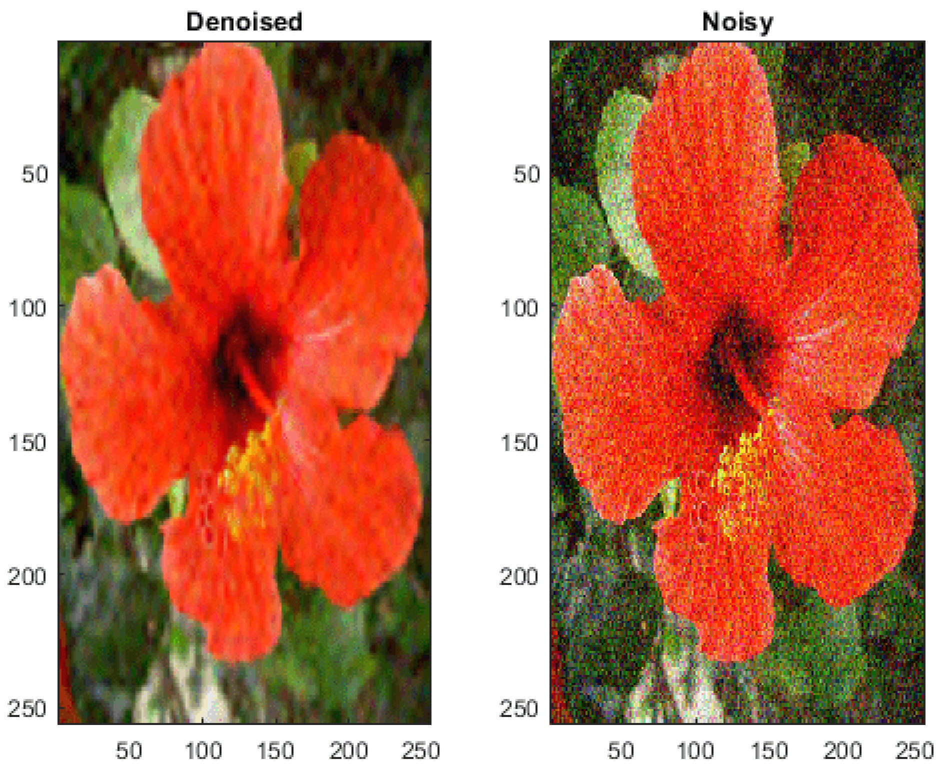
```
load colorflower
subplot(1,2,1)
```

```
imagesc(colorflower.Orig)
title('Original')
subplot(1,2,2)
imagesc(colorflower.Noisy)
title('Noisy')
```



Denoise the image down to level 2 using the default Bayesian method and cycle spinning with$(1 + 1)^2$ shifts. Display the noisy and denoised images.

```
imden = wdenoise2(colorflower.Noisy,2,'CycleSpinning',1);
figure
subplot(1,2,1)
imagesc(imden)
title('Denoised')
subplot(1,2,2)
imagesc(colorflower.Noisy)
title('Noisy')
```



Compute the SNR before and after denoising.

```
beforeSNR = ...
    20*log10(norm(colorflower.Orig(:))/norm(colorflower.Orig(:)-colorflower.Noisy(:)))
```

```
beforeSNR = 11.2217
```

```
afterSNR = ...
    20*log10(norm(colorflower.Orig(:))/norm(colorflower.Orig(:)-imden(:)))
```

```
afterSNR = 19.8813
```

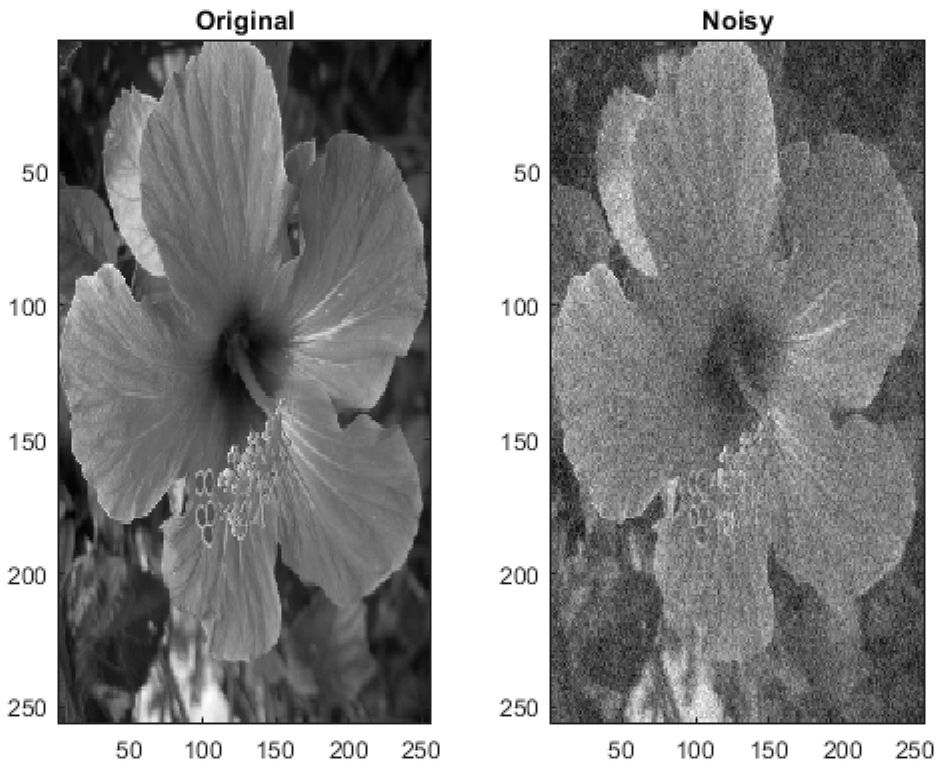
Denoise Image Using Specific Subband

This example shows how to denoise an image using a specific subband to estimate the variance of the noise.

Load the structure `flower`. The structure contains the grayscale image of a flower, and a noisy version of that image. Display the original and noisy images.

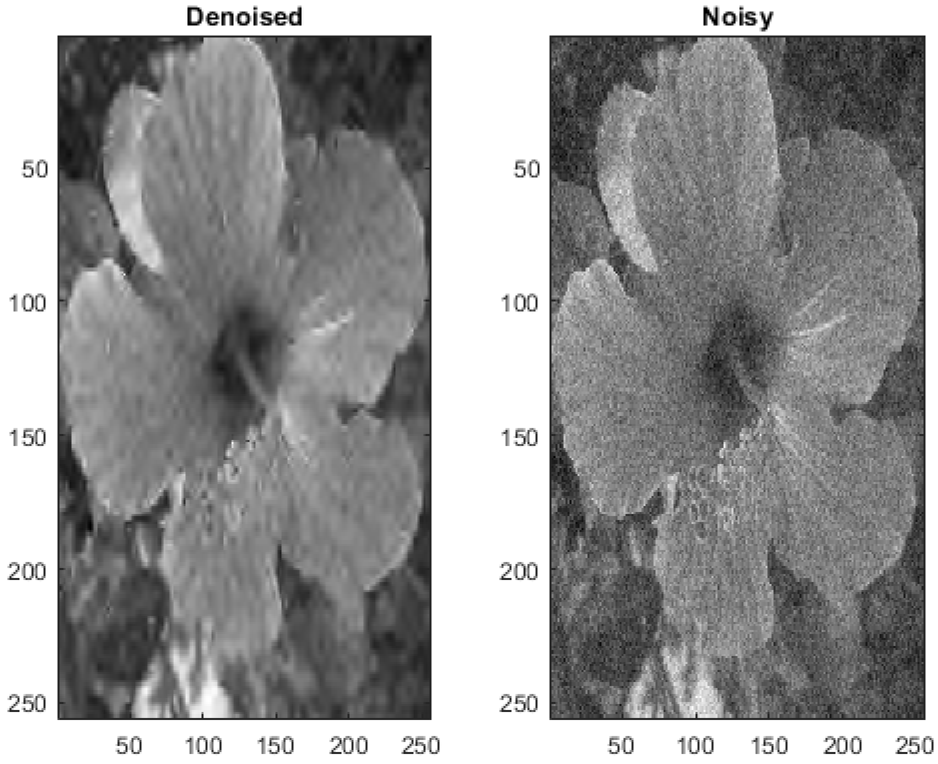
```
load flower
subplot(1,2,1)
imagesc(flower.Orig)
title('Original')
subplot(1,2,2)
```

```
imagesc(flower.Noisy)
title('Noisy')
colormap gray
```



Denoise the image down to level 2 using the False Discovery Rate method with a Q-value of 0.01. Denoise only based on the diagonal wavelet coefficients. Display the denoised and noisy images.

```
imden = wdenoise2(flower.Noisy,2,...
    'DenoisingMethod',{'FDR',0.01},...
    'NoiseDirection',"d");
figure
subplot(1,2,1)
imagesc(imden)
title('Denoised')
subplot(1,2,2)
imagesc(flower.Noisy)
title('Noisy')
colormap gray
```



Compute the SNR before and after denoising.

```
beforeSNR = ...
    20*log10(norm(flower.Orig(:))/norm(flower.Orig(:)-flower.Noisy(:)))
```

```
beforeSNR = 14.1300
```

```
afterSNR = ...
    20*log10(norm(flower.Orig(:))/norm(flower.Orig(:)-imden(:)))
```

```
afterSNR = 19.9164
```

Input Arguments

IM — Input image

real-valued 2-D matrix | real-valued 3-D array

Input image to denoise, specified as a real-valued 2-D matrix or real-valued 3-D array. If `IM` is 3-D, `IM` is assumed to be a color image in the RGB color space and the third dimension of `IM` must be 3. For RGB images, `wdenoise2` by default projects the image onto its PCA color space before denoising. To denoise an RGB image in the original color space, use the `ColorSpace` name-value pair.

LEVEL — Wavelet decomposition level

positive integer

Wavelet decomposition level used for denoising, specified as a positive integer. `LEVEL` is a positive integer less than or equal to `floor(log2(min([M N])))`, where `M` and `N` are the row and column sizes of the image. If unspecified, `LEVEL` defaults to `min([floor(log2(min([M N]))),wmaxlev([M N],wname)])`, where *wname* is the wavelet used (`'bior4.4'` by default).

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NoiseEstimate','LevelDependent','Wavelet','sym6'`

`Wavelet` — Name of wavelet

`'bior4.4'` (default) | character vector | string scalar

Name of wavelet, specified as a character vector or string scalar, 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 biorthogonal 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) | `'FDR'` | `'Minimax'` | `'SURE'` | `'UniversalThreshold'`

Denoising method to use to determine the denoising thresholds for the image `IM`.

- `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 [7].
- `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].

  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`.

- Minimax - Minimax Estimation

  This method uses a fixed threshold chosen to yield minimax performance for mean squared 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 $\sqrt{2\ln(\text{length}(x))}$.

  This method uses a fixed-form threshold yielding minimax performance multiplied by a small factor proportional to log(length(X)).

ThresholdRule — Threshold rule
'Hard' | 'Soft' | 'Mean' | 'Median'

Threshold rule 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:

- '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 image. 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.

There are three wavelet subbands: horizontal, vertical, and diagonal. The value of 'NoiseDirection' specifies which subbands to use in estimating the variance.

NoiseDirection — Wavelet subbands
["h","v","d"] (default) | string vector | scalar string

Wavelet subbands to use to estimate the variance of the noise, specified as a string vector or scalar string. Valid entries are "h", "v", or "d", for the horizontal, vertical, and diagonal subbands, respectively.

Example: 'NoiseDirection',["h" "v"] specifies the horizontal and vertical subbands.

CycleSpinning — Number of circular shifts
0 (default) | nonnegative integer

Number of circular shifts in both the row and column directions to use for denoising IM with cycle spinning. In cycle spinning, circular shifts of the image along the row and column dimensions are denoised, shifted back, and averaged together to provide the final result.

Generally, SNR improvements are observed with cycle spinning up to 3-4 shifts and asymptote after that. Because of the asymptotic effect on SNR and the fact that (CycleSpinning+1)^2 images are being denoised, it is recommended to start with CycleSpinning equal to 0. Then gradually increase the number of shifts to determine if there is any improvement in SNR to justify the computational expense.

For example, specifying 'CycleSpinning',1 results in four copies of IM being denoised:

- The original image (unshifted)
- IM circularly shifted a single-element along the row dimension
- IM circularly shifted a single-element along the column dimension
- IM circularly shifted a single-element along both the row and column dimensions

The four denoised copies of IM are denoised, reconstructed, shifted back to their original positions, and averaged together. The value of CycleSpinning represents the maximum shift along both the row and column dimensions. For RGB images, there are no shifts applied along the color space dimension.

ColorSpace — Color space

'PCA' (default) | 'Original'

Color space used for denoising an RGB image. Valid options are 'PCA' and 'Original'.

- 'PCA': The RGB image is first projected onto its PCA color space, denoised in the PCA color space, and returned to the original color space after denoising.
- 'Original': Denoising is done in the same color space as the input image.

ColorSpace is valid only for RGB images.

Output Arguments

IMDEN — Denoised image

real-valued matrix

Denoised image, returned as a matrix. The dimensions of IM and IMDEN are equal.

DENOISEDCFS — Scaling and denoised wavelet coefficients

real-valued matrix

Scaling and denoised wavelet coefficients of the denoised image, returned as a real-valued matrix. DENOISEDCFS is a (numshifts+1)^2-by-N matrix where N is the number of wavelet coefficients in the decomposition of IM and numshifts is the value of 'CycleSpinning'. Each row of DENOISEDCFS contains the denoised wavelet coefficients for one of (numshifts+1)^2 shifted versions of IM. For RGB images, DENOISEDCFS are the denoised coefficients in the specified color space.

The i^{th} row of DENOISEDCFS contains the denoised wavelet coefficients of the image circularly shifted by the amount returned in the i^{th} column of SHIFTS. For example, if the second column of SHIFTS is [1 ; 1], the second row of DENOISEDCFS contains the denoised coefficients of the image circularly shifted by a single element in the row direction and a single element in the column direction.

ORIGCFS — Scaling and wavelet coefficients

real-valued matrix

Scaling and wavelet coefficients of the input image, returned as a real-valued 2-D matrix. ORIGCFS is a $(\texttt{numshifts+1})^2$-by-N matrix where N is the number of wavelet coefficients in the decomposition of IM and numshifts is the value of the 'CycleSpinning'. Each row of ORIGCFS contains the wavelet coefficients for one of $(\texttt{numshifts+1})^2$ shifted versions of IM. For RGB images, ORIGCFS are the original coefficients in the specified color space.

The i^{th} row of ORIGCFS contains the wavelet coefficients of the image circularly shifted by the amount returned in the i^{th} column of SHIFTS. For example, if the second column of SHIFTS is [1 ; 1], the second row of ORIGCFS contains the coefficients of the image circularly shifted by a single element in the row direction and a single element in the column direction.

S — Bookkeeping matrix

integer-valued matrix

Bookkeeping matrix. The matrix S contains the dimensions of the approximation coefficients at the coarsest scale, the sizes of the wavelet coefficients at all scales, and the size of the original input image. S is a matrix with the same structure as the S output of wavedec2.

SHIFTS — Image shifts

integer-valued matrix

Image shifts used in cycle spinning, returned as an integer-valued matrix. SHIFTS is 2-by-$(\texttt{numshifts+1})^2$ matrix where each column of SHIFTS contains the shifts along the row and column dimension used in cycle spinning.

References

[1] Abramovich, F., Y. Benjamini, D. L. Donoho, and I. M. Johnstone. “Adapting to Unknown Sparsity by Controlling the False Discovery Rate.” *Annals of Statistics*, Vol. 34, Number 2, pp. 584–653, 2006.

[2] Antoniadis, A., and G. Oppenheim, eds. *Wavelets and Statistics*. Lecture Notes in Statistics. New York: Springer Verlag, 1995.

[3] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[4] Donoho, D. L., I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*, Vol. 81, pp. 425–455, 1994.

[5] Donoho, D. L. “De-noising by Soft-Thresholding.” *IEEE Transactions on Information Theory*, Vol. 42, Number 3, pp. 613–627, 1995.

[6] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society, series B*, Vol. 57, No. 2, pp. 301–369, 1995.

[7] Johnstone, I. M., and B. W. Silverman. “Needles and Straw in Haystacks: Empirical Bayes Estimates of Possibly Sparse Sequences.” *Annals of Statistics*, Vol. 32, Number 4, pp. 1594–1649, 2004.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- The value of the `Wavelet` name-value pair argument must be constant.
- The input `LEVEL` must be defined as a scalar during compilation.
- When `ColorSpace` is set to `'PCA'`, eigenvectors are used to denoise the RGB image. `wdenoise2` uses the `eig` function to calculate the eigenvectors. The eigenvectors calculated by the generated code might be different in C and C++ code than in MATLAB. As a result, the signs of the detail coefficients returned by the generated denoising code might be different than in MATLAB. The lowpass coefficients are not affected. For more information, see `eig`.
- Plotting is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The value of the `Wavelet` name-value pair argument must be constant.
- For optimized GPU code generation, specify `LEVEL` as a compile-time constant.
- The `'Bayes'`, `'UniversalThreshold'`, `'Minimax'`, and `'SURE'` denoising methods support optimized GPU code generation.
- When `ColorSpace` is set to `'PCA'`, eigenvectors are used to denoise the RGB image. `wdenoise2` uses the `eig` function to calculate the eigenvectors. The eigenvectors calculated by the generated code might be different in C and C++ code than in MATLAB. As a result, the signs of the detail coefficients returned by the generated denoising code might be different than in MATLAB. The lowpass coefficients are not affected. For more information, see `eig`.
- Plotting is not supported.

See Also

`wdenoise` | `wavedec2`

Introduced in R2019a

wenergy

Energy for 1-D wavelet or wavelet packet decomposition

Syntax

```
[Ea,Ed] = wenergy(c,l)
E = wenergy(wpt)
```

Description

`[Ea,Ed] = wenergy(c,l)` returns, for a 1-D wavelet decomposition, `Ea`, the percentage of energy corresponding to the approximation, and `Ed`, the percentages of energies corresponding to the details. `c` and `l` are outputs of `wavedec`.

`E = wenergy(wpt)` returns the percentages of energy corresponding to the terminal nodes of the wavelet packet tree `wpt` (see `wptree`, `wpdec`, and `wpdec2`). In this case, `wenergy` is a method of the `wptree` object `wpt`, which overloads the previous `wenergy` function.

Examples

Energy of Wavelet Decompositions

Load a 1-D signal.

```
load noisbump
```

Obtain the 4-level wavelet decomposition of the signal using the `sym4` wavelet.

```
wv = "sym4";
[c,l] = wavedec(noisbump,4,wv);
```

Obtain the percentages of energy in the approximation and details coefficients.

```
[Ea,Ed] = wenergy(c,l)
```

```
Ea = 88.2860
```

```
Ed = 1×4

    2.1560    1.2286    1.4664    6.8630
```

Obtain the wavelet packet tree corresponding to the 3-level wavelet packet decomposition of the signal using the `sym4` wavelet.

```
t = wpdec(noisbump,3,wv);
```

Obtain the percentages of energy in the terminal nodes.

```
e = wenergy(t)
```

```
e = 1×8

   95.0329    1.4664    0.6100    0.6408    0.5935    0.5445    0.5154    0.5965
```

Input Arguments

c — Wavelet decomposition

vector

Wavelet decomposition, specified as a vector. The vector contains the wavelet coefficients. The bookkeeping vector `l` contains the number of coefficients by level. See `wavedec`.

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

l — Bookkeeping vector

vector

Bookkeeping vector, specified as a vector of positive integers. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition `c` by level. See `wavedec`.

Data Types: `single` | `double`

wpt — Wavelet packet tree

`wptree` object

Wavelet packet tree, specified as a `wptree` object. See `wptree`, `wpdec`, and `wpdec2`.

Output Arguments

Ea — Percentage of energy corresponding to approximation

positive scalar

Percentage of energy corresponding to approximation coefficients, returned as a positive scalar.

Data Types: `single` | `double`

Ed — Percentage of energy corresponding to details

vector

Percentage of energy corresponding to details coefficients, returned as a vector.

Data Types: `single` | `double`

E — Percentage of energy corresponding to terminal nodes

vector

Percentage of energy corresponding to the terminal nodes, returned as a vector.

Data Types: `single` | `double`

See Also

`wptree` | `wpdec` | `wpdec2`

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)
E = wentropy(X,T,P)
```

Description

`E = wentropy(X,T)` returns the entropy specified by `T` of the vector or matrix `X`.

`E = wentropy(X,T,P)` returns the entropy where `P` is a parameter depending on `T`.

`E = wentropy(X,T,0)` is equivalent to `E = wentropy(X,T)`.

Examples

Signal Entropy

This example shows the different values of entropy of a random signal.

For purposes of reproducibility, reset the random seed and generate a random signal.

```
rng default
x = randn(1,200);
```

Compute the Shannon entropy of `x`.

```
e = wentropy(x,'shannon')
```

```
e = -224.5551
```

Compute the log energy entropy of `x`.

```
e = wentropy(x,'log energy')
```

```
e = -229.5183
```

Compute the threshold entropy of `x` with the threshold entropy equal to 0.2.

```
e = wentropy(x,'threshold',0.2)
```

```
e = 168
```

Compute the Sure entropy of `x` with the threshold equal to 3.

```
e = wentropy(x,'sure',3)
```

```
e = 35.7962
```

Compute the norm entropy of `x` with power equal to 1.1

```
e = wentropy(x,'norm',1.1)
```

```
e = 173.6578
```

You can use your own entropy function `ABC` with `wentropy`. Your function must be defined in a `.m` file, and the first line must be of the form:

```
function e = ABC(x)
```

where `x` is a vector and `e` is a real number. The new entropy can be used by typing

```
e = wentropy(x,'user','ABC')
```

or more directly

```
e = wentropy(x,'ABC')
```

The function file `myEntropy.m` returns the normalized Shannon entropy of a signal. Compute the normalized Shannon entropy of `x`.

```
w = wentropy(x,'myEntropy')
```

```
w = -1.1228
```

Input Arguments

X — Input data

real-valued vector or matrix

Input data, specified as a real-valued vector or matrix.

T — Entropy type

`'shannon'` | `'log energy'` | `'threshold'` | `'sure'` | `'norm'` | `'user'` | `'`*`FunName`*`'`

Entropy type, specified as one of the following:

Entropy Type (T)	Threshold 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 (T)	Threshold Parameter (P)	Comments
'*FunName*'	No constraints on P	FunName is any character vector other than the previous entropy types listed. FunName contains the file name of your own entropy function, with x as input and P as an additional parameter to your entropy function.

T and the threshold parameter P together define the entropy criterion.

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.

P — Threshold parameter

real number | character vector | string scalar

Threshold parameter, specified by a real number, character vector, or string scalar. P and the entropy type T together define the entropy criterion.

Output Arguments

E — Entropy

real number

Entropy of X, returned as a real number.

More About

Entropy

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.

$E5(s) = n - \#\left\{i \text{ such that } \left|s_i\right| \le p\right\} + \sum_i \min(s_i^2, p^2)$, where n is the length of the signal.

For more information, see “Wavelet Packets for Compression and Denoising”.

References

[1] Coifman, R. R., and M. V. Wickerhauser. "Entropy-based Algorithms for best basis selection." *IEEE Transactions on Information Theory*. Vol. 38, Number 2, March 1992, pp. 713–718.

[2] Donoho, D. L., and I. M. Johnstone. "Ideal denoising in an orthonormal basis chosen from a library of bases." *Comptes Rendus Acad. Sci. Paris, Ser. I*. Vol. 319, 1994, pp. 1317–1322.

See Also

`wpdec` | `wpdec2` | `wpdencmp`

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 = 1×3

     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 = 1×7

     0     0     1     2     3     0     0
```

```
xextzpd2 = wextend('1D','zpd',x,len,'b')
```

```
xextzpd2 = 1×7

     0     0     1     2     3     0     0
```

Perform a half-point symmetric extension.

```
xextsym = wextend('1D','sym',x,len)
```

```
xextsym = 1×7

     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 = 1×8

     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 = 2×3

     1     2     3
     4     5     6
```

Perform a zero-pad extension of the array.

```
Xextzpd = wextend(2,'zpd',X,len)
```

```
Xextzpd = 6×7

     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 = 6×7

     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 = 1×10

    -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 = 1×10

    -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 = 1×10

     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 = 1×10

    -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 = 1×10

  -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 = 1×10

   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 = 1×10

  -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 = 1×10

  -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. For more information, see `dwtmode`.

`MODE`	Description
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1
`'sym'` or `'symh'`	Symmetric padding (half point): boundary value symmetric replication
`'symw'`	Symmetric padding (whole point): boundary value symmetric replication
`'asym'` or `'asymh'`	Antisymmetric padding (half point): boundary value antisymmetric replication
`'asymw'`	Antisymmetric padding (whole point): boundary value antisymmetric replication
`'ppd'`	Periodized extension (1)
`'per'`	Periodized extension (2) If the signal length is odd, `wextend` appends on the right a copy of 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 one of the following occurs.

- The conversion to double causes a loss of precision.
- 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 — Length of extension
nonnegative integer | two-element vector of nonnegative integers

Length of extension, specified as a nonnegative integer or two-element vector of nonnegative 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.

An extension of length 0 is equivalent to the null extension.

Example: wextend('2D','sym',[1 2 3 4;5 6 7 8],[2 0]) extends only two rows up and two rows down.

LOC — Location of extension
'l' | 'u' | 'r' | 'd' | 'b' | 'n' | two-character array

Location of extension, specified as one or a pair of the following:

- 'l' — Extension left
- 'u' — Extension up
- 'r' — Extension right
- 'd' — Extension down
- 'b' — Extension on both sides
- 'n' — Null extension

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', 'b', or 'n' 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.

TYPE	LOC
2, '2', '2d' or '2D'	[LOCROW,LOCCOL], where LOCROW and LOCCOL are 1-D extension locations or 'n' (none). 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', 'b', or 'n' 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.
'ac' or 'addcol'	'l', 'u', 'r', 'd', 'b', or 'n' 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.

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 1-1584 and “Extend int8 Data Beyond Range Limits” on page 1-1586.

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`.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `'sym'` and `'per'` extension modes are supported.
- The only syntax supported is `YEXT = wextend(TYPE,MODE,X,LEN)`.
  - The `LOC` input argument is not supported.
  - For one-dimensional extensions, the default location `'b'` is used. For two-dimensional extensions, the default location `'bb'` is used.
- Only extensions in one dimension are supported.
  - The `LEN` input argument must have length equal to one.
  - For one-dimensional extensions, the only supported extension methods are: 1, `'1'`, `'1d'`, and `'1D'`.
  - For two-dimensional extensions, the only supported extension methods are: `'addrow'`, and `'addcol'`.

See Also

`dwtmode`

Introduced before R2006a

wfbm

Fractional Brownian motion synthesis

Syntax

```
fBm = wfbm(H,L)
fBm = wfbm(H,L,ns,w)

fBm = wfbm(H,L,'plot')
```

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 [1]. By default, `wfbm` uses six reconstruction steps and the orthogonal `db10` wavelet.

`fBm = wfbm(H,L,ns,w)` returns the signal using `ns` reconstruction steps and the sufficiently regular orthogonal wavelet `w`.

`fBm = wfbm(H,L,w,ns)` is equivalent to `fBm = wfbm(H,L,ns,w)`.

`fBm = wfbm(H,L,'plot')` generates and plots the `fBm` signal. The following syntaxes also generate and plot the signal.

- `fBm = wfbm(H,L,'plot',w)`
- `fBm = wfbm(H,L,'plot',ns)`
- `fBm = wfbm(H,L,'plot',w,ns)`
- `fBm = wfbm(H,L,'plot',ns,w)`

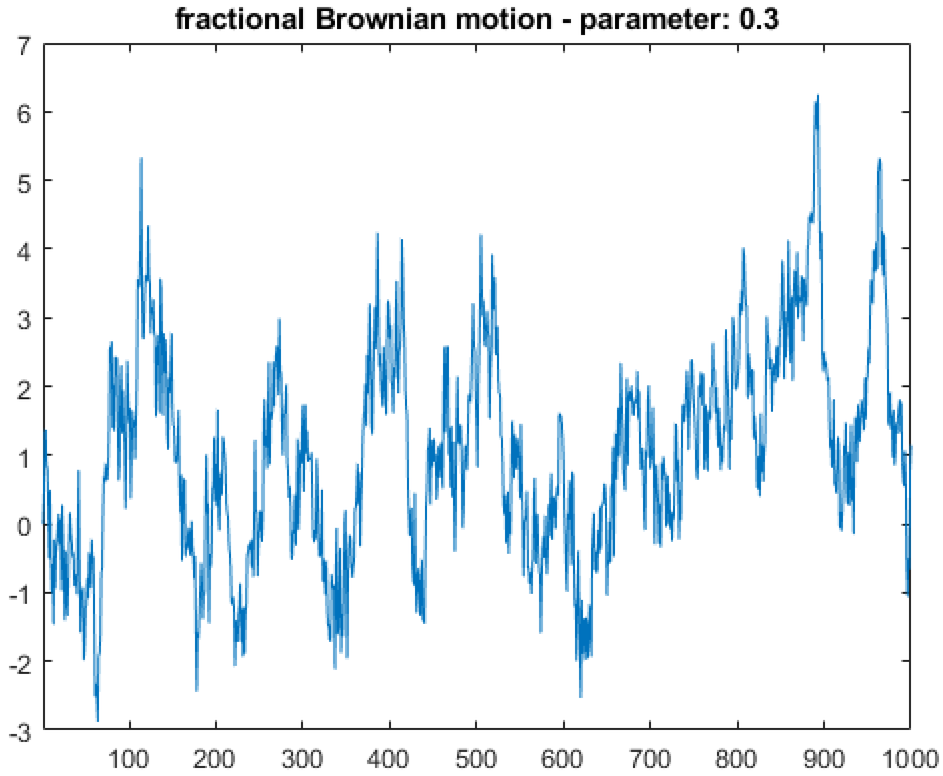
Examples

Generate Fractional Brownian Motion Signals

According to the value of the Hurst parameter `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` function, which generates a sample path of this process.

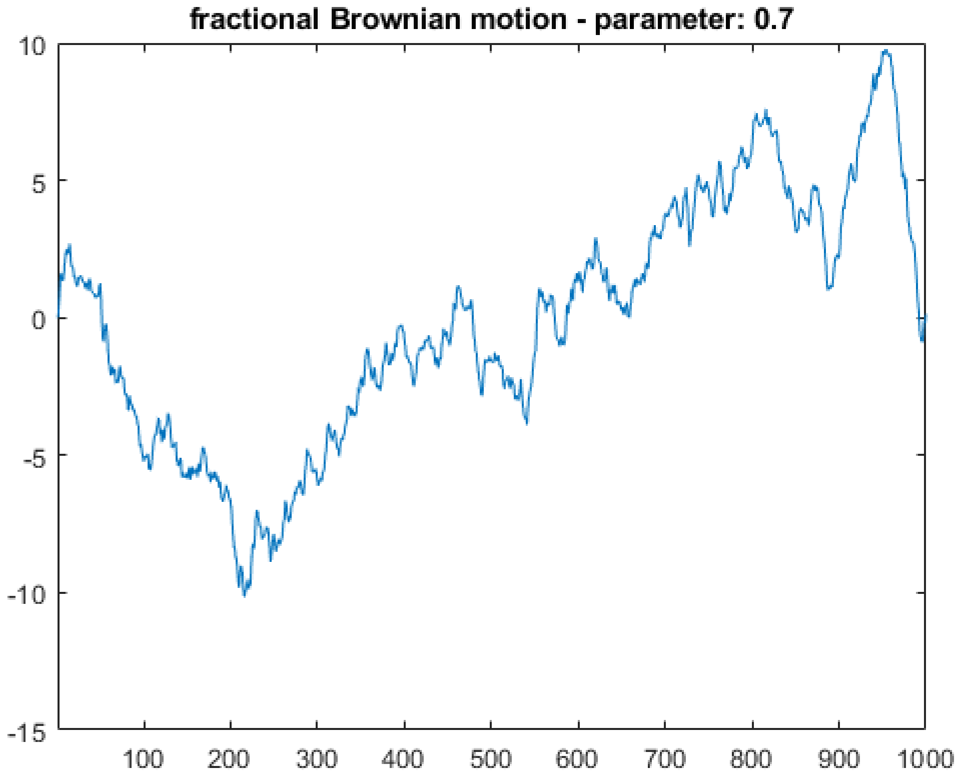
For purposes of reproducibility, set the random seed to the default value. Generate a fractional Brownian motion signal of length 1000 with the Hurst parameter of 0.3. Plot the signal.

```
rng default
h = 0.3;
l = 1000;
fBm03 = wfbm(h,l,'plot');
```



Now generate a fractional Brownian motion signal of length 1000 with the Hurst parameter of 0.7. The signal clearly exhibits a stronger low-frequency component and has, locally, less irregular behavior than `fBm03`.

```
h = 0.7;
l = 1000;
fBm07 = wfbm(h,l,'plot');
```



Input Arguments

H — Hurst parameter

positive scalar

Hurst parameter, specified as a positive scalar strictly less than 1.

Example: `fBm = wfbm(0.4,1000)` generates a fractional Brownian motion of length `L = 1000` with Hurst parameter `H = 0.4`.

Data Types: `double`

L — Signal length

positive integer

Signal length, specified as a positive integer strictly greater than 100.

Example: `fBm = wfbm(0.1,500)` generates a fractional Brownian motion of length `L = 500` with Hurst parameter `H = 0.1`.

Data Types: `double`

ns — Number of reconstruction steps

positive integer

Number of reconstruction steps, specified as a positive integer greater than 1.

Data Types: `double`

`w` — Orthogonal wavelet
character vector | string scalar

Orthogonal wavelet recognized by `wavemngr`, specified as a character vector or string scalar.

Output Arguments

`fBm` — Fractional Brownian motion signal
vector

Fractional Brownian motion signal, returned as a vector of length `L`.

More About

Fractional Brownian Motion

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.

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 [1].

Nevertheless, the samples generated following this original scheme exhibit too many high-frequency components. To circumvent this undesirable behavior Bardet et al. [2] propose downsampling the obtained sample by a factor of 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 [2].

References

[1] Abry, Patrice, and Fabrice Sellan. “The Wavelet-Based Synthesis for Fractional Brownian Motion Proposed by F. Sellan and Y. Meyer: Remarks and Fast Implementation.” *Applied and*

Computational Harmonic Analysis 3, no. 4 (October 1996): 377–83. https://doi.org/10.1006/acha.1996.0030.

[2] Bardet, Jean-Marc, Gabriel Lang, Georges Oppenheim, Anne Philippe, Stilian Stoev, and Murad S. Taqqu. “Generators of Long-Range Dependent Processes: A Survey.” In *Theory and Applications of Long-Range Dependence*, edited by Paul Doukhan, Georges Oppenheim, and Murad S. Taqqu, 579–623. Boston: Birkhauser, 2003.

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.

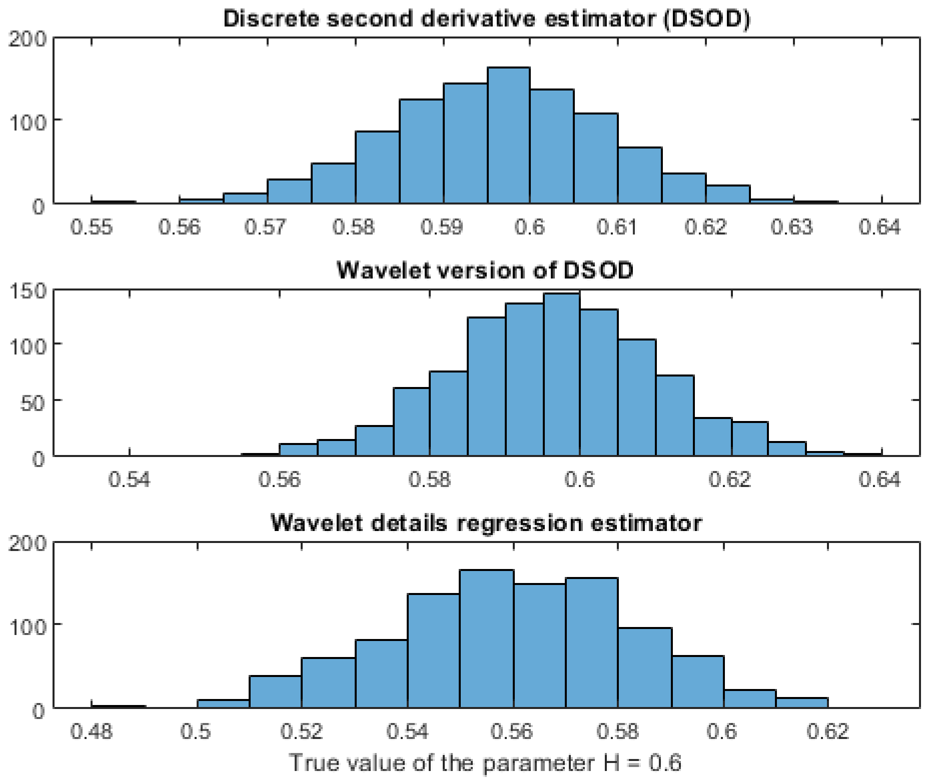
```
rng default;
H = 0.6;
len = 10000;
```

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

```
[LoD,HiD,LoR,HiR] = wfilters(wname)
[F1,F2] = wfilters(wname,type)
```

Description

`[LoD,HiD,LoR,HiR] = wfilters(wname)` returns the four lowpass and highpass, decomposition and reconstruction filters associated with the orthogonal or biorthogonal wavelet `wname`.

`[F1,F2] = wfilters(wname,type)` returns the pair of `type` filters associated with the orthogonal or biorthogonal wavelet `wname`. For example, `wfilters('db6','h')` returns the pair of highpass filters `HiD` and `HiR` associated with the `db6` wavelet.

Examples

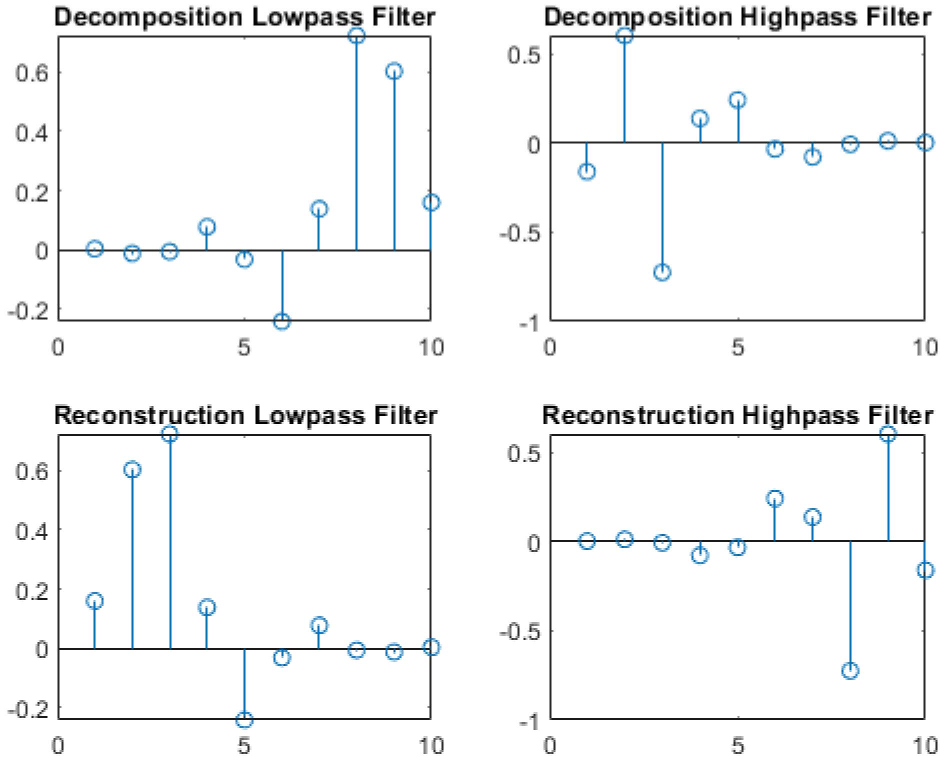
Compute Four Filters

Set the wavelet name.

```
wname = 'db5';
```

Compute the four filters associated with wavelet name specified by `wname` and plot the results.

```
[LoD,HiD,LoR,HiR] = wfilters(wname);
subplot(2,2,1)
stem(LoD)
title('Decomposition Lowpass Filter')
subplot(2,2,2)
stem(HiD)
title('Decomposition Highpass Filter')
subplot(2,2,3)
stem(LoR)
title('Reconstruction Lowpass Filter')
subplot(2,2,4)
stem(HiR)
title('Reconstruction Highpass Filter')
xlabel(['The four filters for ',wname])
```



The four filters for db5

Input Arguments

`wname` — Name of orthogonal or biorthogonal wavelet

`'haar'` | `'db1'` | `'db2'` | `'coif1'` | `'coif2'` | ...

Name of orthogonal or biorthogonal wavelet, specified as one of the values listed here.

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'`

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'`

Wavelet Families	Wavelets
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'

type — Type of filter pair
'd' | 'r' | 'l' | 'h'

Type of filter pair to return, specified as one of the values listed here.

type	Description
'd'	Decomposition filters (LoD and HiD)
'r'	Reconstruction filters (LoR and HiR)
'l'	Lowpass filters (LoD and LoR)
'h'	Highpass filters (HiD and HiR)

Output Arguments

LoD — Decomposition lowpass filter
real-valued vector

Decomposition lowpass filter, returned as a real-valued vector, associated with the wavelet wname.

HiD — Decomposition highpass filter
real-valued vector

Decomposition highpass filter, returned as a real-valued vector, associated with the wavelet wname.

LoR — Reconstruction lowpass filter
real-valued vector

Reconstruction lowpass filter, returned as a real-valued vector, associated with the wavelet wname.

HiR — Reconstruction highpass filter
real-valued vector

Reconstruction highpass filter, returned as a real-valued vector, associated with the wavelet wname.

F1, F2 — Filter pair
real-valued vectors

Filter pair of requested type, returned, specified as one of the pairs of filters listed here.

type	Description	Filter Pair
'd'	Decomposition filters	LoD and HiD
'r'	Reconstruction filters	LoR and HiR
'l'	Lowpass filters	LoD and LoR
'h'	Highpass filters	HiD and HiR

References

[1] Daubechies, I. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[2] Mallat, S. G. “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 11, Issue 7, July 1989, pp. 674–693.

See Also

`biorfilt` | `orthfilt` | `waveinfo` | `wavemngr`

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( ___ ,'plot')
```

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.

`xfus = wfusimg(x1,x2,wname,level,afusmeth,dfusmeth)` returns the fused image `xfus` obtained by fusion of the two original images `x1` and `x2`.

`[xfus,txfus,tx1,tx2] = wfusimg(x1,x2,wname,level,afusmeth,dfusmeth)` also returns three wavelet decomposition tree objects associated with `xfus`, `x1`, and `x2`, respectively.

`[ ___ ] = wfusimg( ___ ,'plot')` plots the objects `txfus`, `tx1`, and `tx2`. This syntax can be used with any of the previous syntaxes.

Examples

Fuse Two Images

This example shows how to fuse two images to create a new image.

Load the mask and bust images.

```
load mask
x1 = X;
load bust
x2 = X;
```

Merge the two images from level 5 wavelet decompositions using the `db2` wavelet. Perform the fusion by taking the mean for both approximations and details.

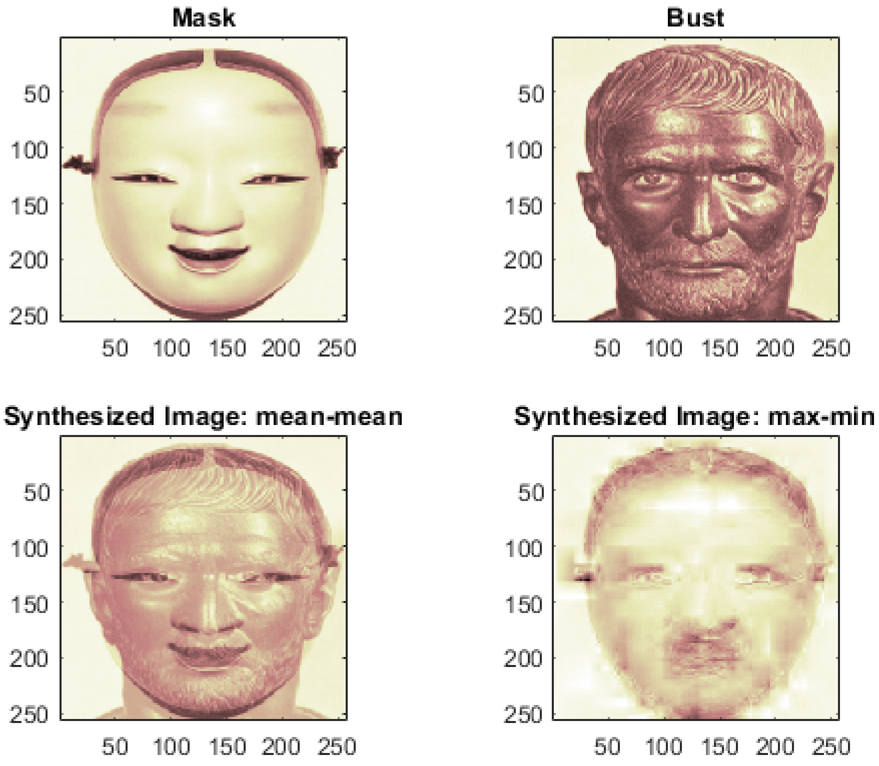
```
wv = 'db2';
lv = 5;
xfusmean = wfusimg(x1,x2,wv,lv,'mean','mean');
```

Merge the two images again, but this time perform the fusion by taking the maximum of the approximations and the minimum for the details.

```
xfusmaxmin = wfusimg(x1,x2,wv,lv,'max','min');
```

Plot the original and fused images.

```
subplot(2,2,1)
image(x1)
axis square
title('Mask')
subplot(2,2,2)
image(x2)
axis square
title('Bust')
subplot(2,2,3)
image(xfusmean)
axis square
title('Synthesized Image: mean-mean')
subplot(2,2,4)
image(xfusmaxmin)
axis square
title('Synthesized Image: max-min')
colormap(map)
```



Restore Image From Two Fuzzy Versions

This example shows how to restore an image from two fuzzy versions of an original image.

Load two fuzzy versions of an original image.

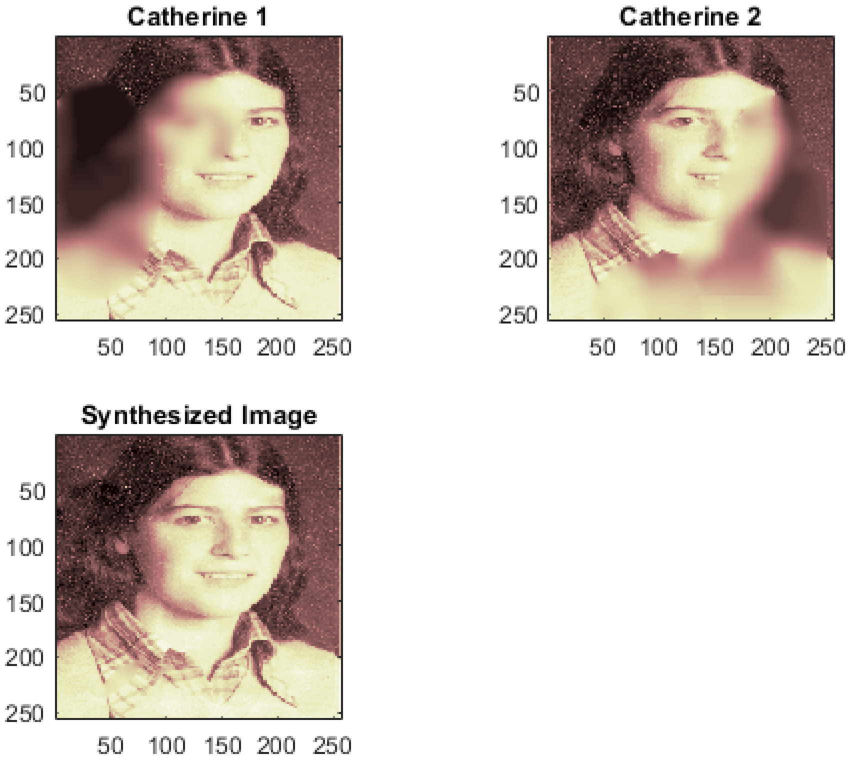
```
load cathe_1
x1 = X;
load cathe_2
x2 = X;
```

Merge the two images from level 5 wavelet decompositions using the `smy4` wavelet. Perform the fusion by taking the maximum of the absolute value of the coefficients for both approximations and details.

```
wv = 'sym4';
lv = 5;
xfus = wfusimg(x1,x2,wv,lv,'max','max');
```

Plot the original and fused images.

```
subplot(2,2,1)
image(x1)
axis square
title('Catherine 1')
subplot(2,2,2)
image(x2)
axis square
title('Catherine 2')
subplot(2,2,3)
image(xfus)
axis square
title('Synthesized Image')
colormap(map)
```



Fuse Two Images With User-Defined Fusion Method

This example shows how to fuse two images using a user-defined fusion method.

Load two images of the same size.

```
load mask
a = X;
load bust
b = X;
```

Define the fusion method and call the fusion function `helperUserFusion`. The source code for `helperUserFusion` is listed in the appendix.

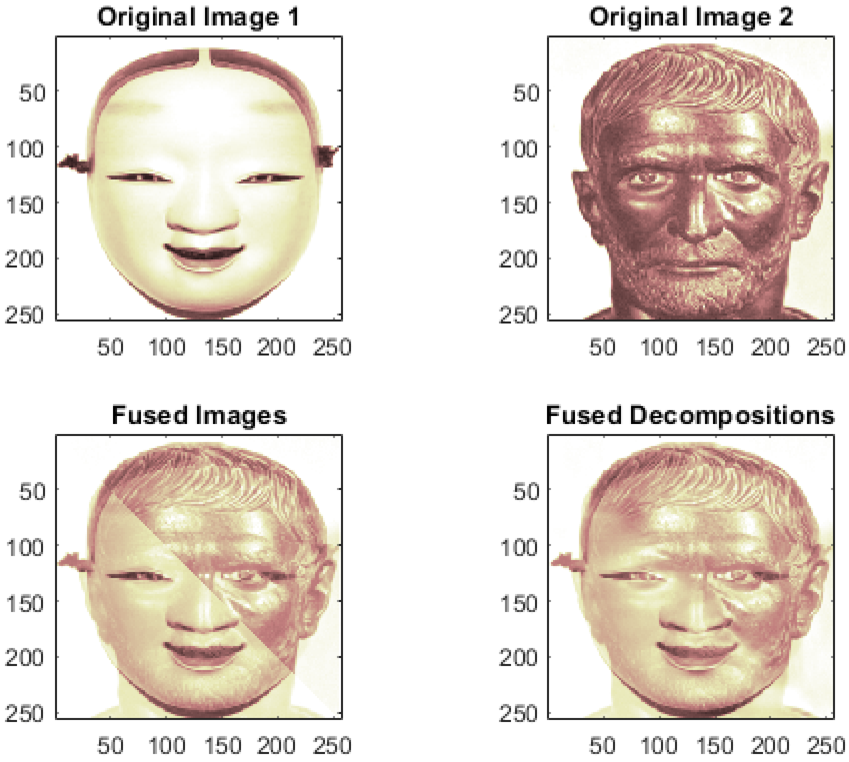
```
fus_method = struct('name','userDEF','param','helperUserFusion');
```

Merge the images twice with the user-defined method. First use `wfusmat`, which fuses the images themselves and not their wavelet decompositions. Then use `wfusimg`, which fuses the wavelet decompositions.

```
c = wfusmat(a,b,fus_method);
d = wfusimg(a,b,'db4',5,fus_method,fus_method);
```

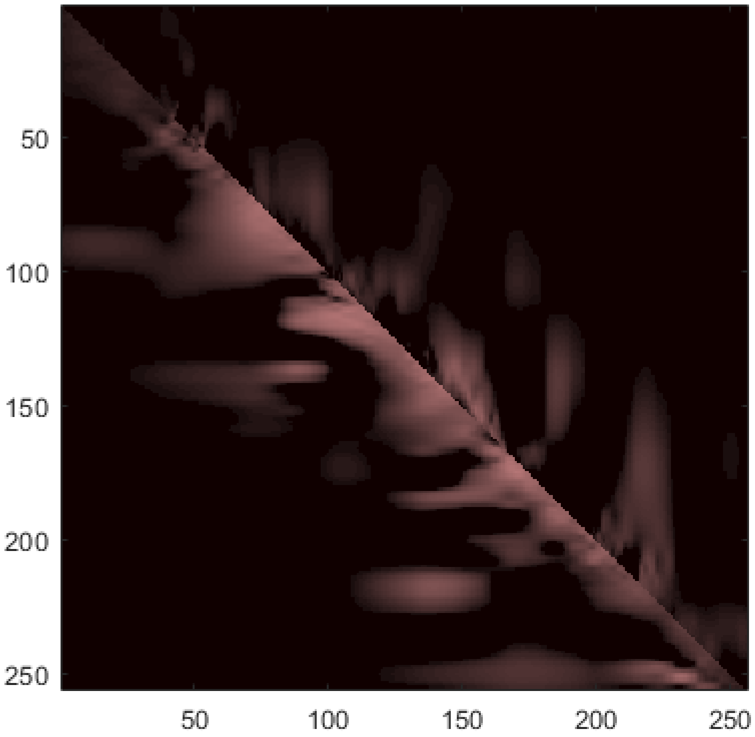
Plot the original and fused images.

```
subplot(2,2,1)
image(a)
title('Original Image 1')
axis square
subplot(2,2,2)
image(b)
title('Original Image 2')
axis square
subplot(2,2,3)
image(c)
title('Fused Images')
axis square
subplot(2,2,4)
image(d)
title('Fused Decompositions')
axis square
colormap(pink(220))
```



Visualize the differences between the merged images.

```
figure
image(c-d)
axis square
colormap(pink(220))
```



Appendix

helperUserFusion

If you want to try a different user-defined fusion method, edit the file helpUserFusion.m, which is located in the same folder as this example.

```
function c = helperUserFusion(A,B)
% This function is in support of the wavelet fusion examples only. It may
% change or be removed in a future release.

% create an upper triangular logical array the same size as A.
d = logical(triu(ones(size(A))));
% set a threshold
t = 0.3;

c = A;
% set the upper triangular portion of the output to a blend of A and B
c(d) = t*A(d)+(1-t)*B(d);
% set the lower triangular portion of the output to a different blend of A
% and B
```

```
c(~d) = t*B(~d)+(1-t)*A(~d);
end
```

Input Arguments

x1,x2 — Images to merge

real-valued 2-D matrix | real-valued 3-D array

Images to merge, specified as real-valued 2-D matrices or real-valued 3-D arrays. If specified as 3-D arrays, x1 and x2 are assumed to be color images in the RGB color space and the third dimension of the arrays must be 3.

The images x1 and x2 must be the same size. To resize the images, use wextend or imresize.

wname — Wavelet

character vector | string scalar

Wavelet used to create the wavelet decomposition, specified as a character vector or string scalar. The wavelet must be orthogonal or biorthogonal and recognized by wfilters.

level — Wavelet decomposition level

positive integer

Wavelet decomposition level, specified as a positive integer.

afusmeth,dfusmeth — Fusion methods for approximations and details

'max' | 'min' | 'mean' | 'img1' | 'img2' | 'rand' | structure array

Fusion methods for approximations and details, respectively, each specified either as a structure array or as one of the values listed here. The approximation and details are merged element-wise.

afusmeth	Description
'max'	Maximum
'min'	Minimum
'mean'	Mean
'img1'	First element
'img2'	Second element
'rand'	Random element

When specified as a structure array, the structure has the form struct('name',nameMETH,'param',paramMETH) where nameMETH can be one of the values listed here.

nameMETH	Description
'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`.

Example: `afusmeth = struct('name','linear','param',0.3)`

Data Types: `double` | `struct`

Output Arguments

`xfus` — Fused image
real-valued 2-D matrix | real-valued 3-D array

Fused image, returned as a real-valued 2-D matrix or a real-valued 3-D array. The fused image `xfus` has the same size as `x1` and `x2`.

`txfus,tx1,tx2` — Wavelet decomposition trees
`wdectree` object

Wavelet decomposition trees associated with `xfus`, `x1`, and `x2`, respectively, returned as `wdectree` objects.

Example: `plot(txfus)` plots the object in a GUI tool that you can use to inspect the tree.

References

[1] de Zeeuw, P. M. "Wavelet and image fusion." CWI, Amsterdam, March 1998. `https://groups.google.com/d/msg/comp.soft-sys.matlab/AjqIENmx1Z4/5g7QDFrZvWMJ`

[2] Li, H., B. S. Manjunath, and S. K. Mitra. "Multisensor Image Fusion Using the Wavelet Transform." *Graphical Models and Image Processing*. Volume 57, Issue 3, May 1995, pp. 235–245.

[3] Misiti, M., Y. Misiti, G. Oppenheim, and J.-M. Poggi. *Les ondelettes et leurs applications*. France: Hermes Science/Lavoisier, 2003.

See Also

`wfusmat` | `wextend`

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 (column-wise fusion)
  - `'RL_fusion'`: Right-left fusion (column-wise fusion)
  - `'UserDEF'`: User-defined fusion, `paramMETH` is a character vector or string scalar `'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,S)
Y = wkeep(X,S,[firstr,firstc])
```

Description

`Y = wkeep(X,L,opt)` extracts the vector `Y` from the vector `X`. The length of `Y` is `L`.

If `opt` is `'c'`, `'l'`, or `'r'`, `Y` is the central, left, or right part, respectively, of `X`.

The syntax `Y = wkeep(X,L)` is equivalent to `Y = wkeep(X,L,'c')`.

`Y = wkeep(X,L,first)` extracts the vector `X(first:first+L-1)`.

`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 the matrix `X`, of size `S` and starting from `X(firstr,firstc)`.

Examples

Extract from Vector and Matrix

Create a vector.

```
x = 1:10;
```

Extract a vector of length 6 from the central part of `x`. Confirm both possible syntaxes return the same vector.

```
y = wkeep(x,6,'c')
```

```
y = 1×6

     3     4     5     6     7     8
```

```
y = wkeep(x,6)
```

```
y = 1×6

     3     4     5     6     7     8
```

Extract a vector of length 7 from the central part of `x`.

```
y = wkeep(x,7,'c')
```

```
y = 1×7

     2     3     4     5     6     7     8
```

Extract two vectors of length 6, one from the left part of x, and the other from the right part of x.

```
y = wkeep(x,6,'l')
```

```
y = 1×6

     1     2     3     4     5     6
```

```
y = wkeep(x,6,'r')
```

```
y = 1×6

     5     6     7     8     9    10
```

Create a 5-by-5 matrix.

```
x = magic(5)
```

```
x = 5×5

    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
```

Extract from the center of x a 3-by-2 matrix.

```
y = wkeep(x,[3 2])
```

```
y = 3×2

     5     7
     6    13
    12    19
```

Extract from x the 2-by-4 submatrix starting at x(3,1).

```
y = wkeep(x,[2 4],[3 1])
```

```
y = 2×4

     4     6    13    20
    10    12    19    21
```

Input Arguments

X — Input

vector | matrix

Input, specified as a vector or matrix.

Data Types: `single` | `double`

L — Length of vector to extract

integer | `Inf`

Length of vector to extract from the input vector `X`, specified as an integer or `Inf`. If `L` is specified as `Inf`, `wkeep` returns the input vector `X`.

Data Types: `single` | `double`

opt — Location of extraction

`'c'` | `'l'` | `'r'`

Location of extraction from the input vector `X`, specified as:

- `'c'` — central part of the vector
- `'l'` — left part of the vector
- `'r'` — right part of the vector

Example: `wkeep(1:10,4,'r')` returns the extraction `[7 8 9 10]`.

first — Starting index

positive integer

Starting index of the input vector `X`, specified as a positive integer. The first element in the extraction is `X(first)`.

Data Types: `single` | `double`

S — Dimensions of submatrix

two-element vector

Dimensions of submatrix to extract from the input matrix `X`, specified as a two-element vector. Each element of `S` is a positive integer or `Inf`.

Example: If `X` is a 27-by-5 matrix, `wkeep(X,[Inf 3])` extracts the 27-by-3 submatrix from the central part of `X`.

Data Types: `single` | `double`

firstr,firstc — Starting row, column indices

two positive integers

Starting row, column indices of the input matrix `X`, specified as two positive integers. The value of the extraction `Y(1,1)` is `X(firstr,firstc)`.

Example: `wkeep(X,[3 2],[1 4])` extracts a 3-by-2 submatrix from the matrix `X` starting from `X(1,4)`.

Data Types: `single` | `double`

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`wextend`

Introduced before R2006a

wmaxlev

Maximum wavelet decomposition level

Syntax

```
L = wmaxlev(S,wname)
```

Description

`L = wmaxlev(S,wname)` returns the maximum level `L` possible for a wavelet decomposition of a signal or image of size `S` using the wavelet specified by `wname` (see `wfilters` for more information). The maximum level is the last level for which at least one coefficient is correct.

`wmaxlev` returns the maximum allowed level decomposition, but in a general, a smaller value is taken.

Examples

Maximum Levels of Decomposition for a Signal and Image

Return the maximum level of decomposition of a 1-D signal with 1024 samples using the Haar wavelet.

```
s = 1024;
wv = 'haar';
l = wmaxlev(s,wv)
```

```
l = 10
```

Return the maximum level using the `db7` wavelet.

```
wv = 'db7';
l = wmaxlev(s,wv)
```

```
l = 6
```

Return the maximum level of decomposition for a 2-D signal of dimension 512-by-128 using the Haar wavelet.

```
s = [512 128];
wv = 'haar';
l = wmaxlev(s,wv)
```

```
l = 7
```

Observe the maximum level is the same when taking the minimum of the two dimensions.

```
l = wmaxlev(min(s),wv)
```

```
l = 7
```

Return the maximum level using the `db7` wavelet.

```
wv = 'db7';
l = wmaxlev(s,wv)
```

```
l = 3
```

Input Arguments

S — Size of signal or image

positive integer | two-element vector of positive integers

Size of signal or image, specified as a positive integer for a signal, or two-element vector of positive integers for an image.

Data Types: `double`

`wname` — Wavelet

character vector | string scalar

Wavelet used to determine maximum level of wavelet decomposition. The wavelet is from one of the following wavelet families: Daubechies, Coiflets, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

See Also

`wavedec` | `wavedec2` | `wpdec` | `wpdec2`

Introduced before R2006a

wmpalg

(Not recommended) Matching pursuit

Note The `wmpalg` function no longer supports plotting and is no longer recommended. See "Compatibility Considerations".

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.

`[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` arguments.

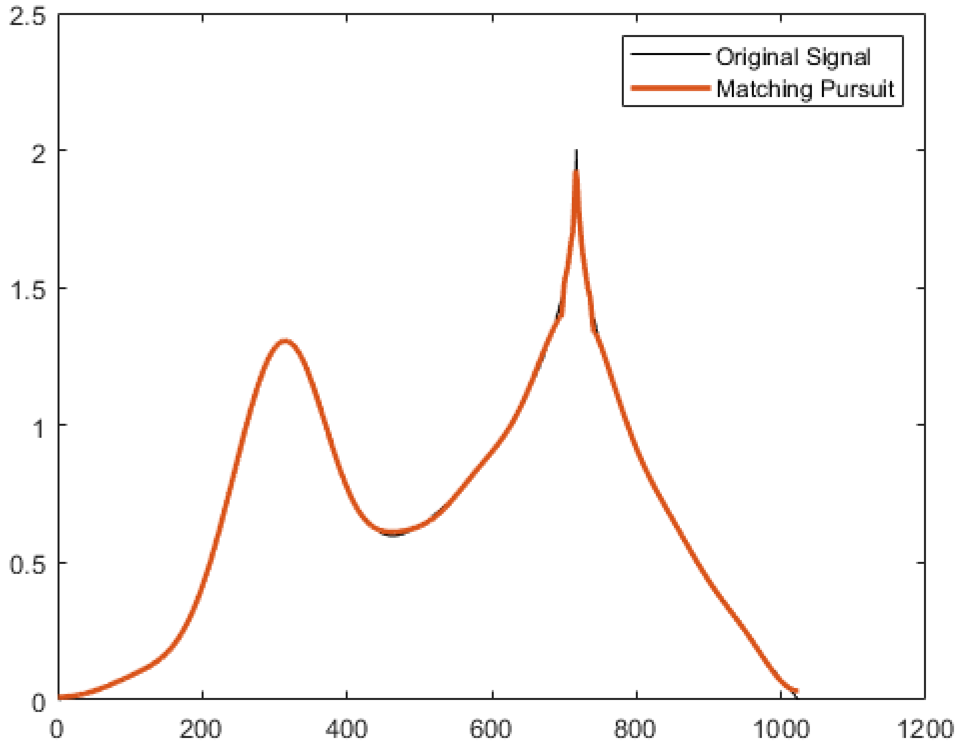
Examples

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
yfit = wmpalg('OMP',cuspamax,'lstcpt',{{'wpsym4',2},'dct'});
plot(cuspamax,'k')
hold on
plot(yfit,'linewidth',2)
hold off
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.

Specify 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;
[yfit,r,coeff,iopt,qual] = wmpalg('OMP',cuspamax,...
    'lstcpt',{{'wpsym4',2},'dct'});
```

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
[yfit,r,coeff,iopt,qual] = wmpalg('OMP',cuspamax,...
    'lstcpt',{{'wpsym4',1},{'wpsym4',2},'dct'},...
    'itermax',50);
```

Change Optimality Factor for Weak Orthogonal Matching Pursuit

This example shows how to allow for a suboptimal choice in the update of the orthogonal matching pursuit.

Load a signal.

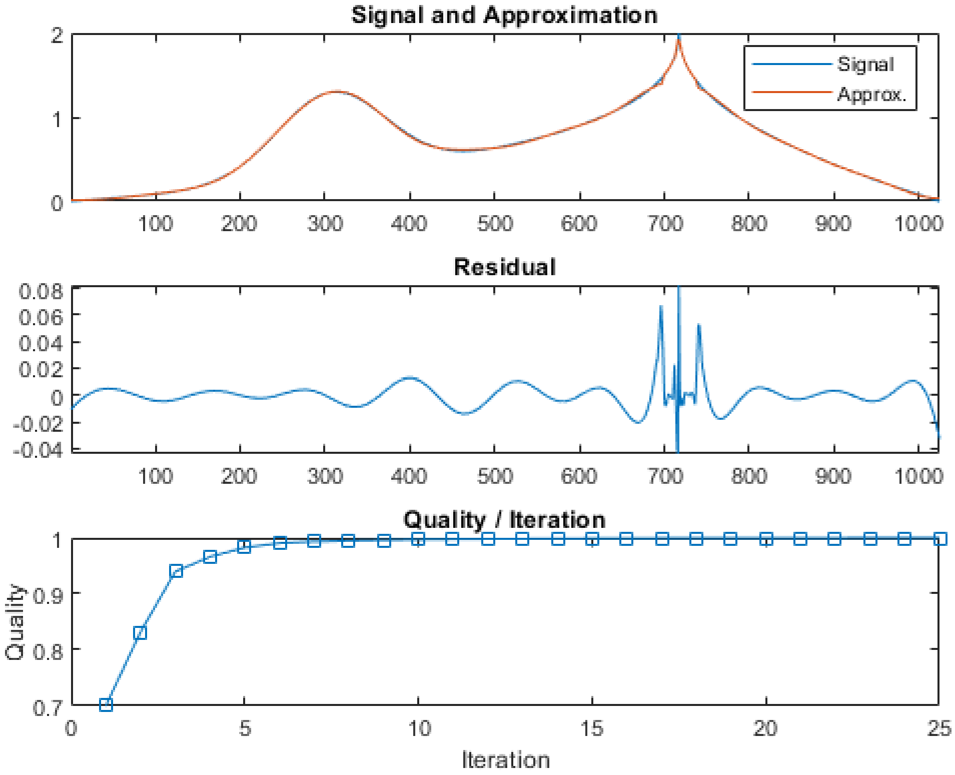
```
load cuspamax
```

Approximate the signal using weak orthogonal matching pursuit. Relax the requirement to be 0.8 times the optimal assignment.

```
[yfit,r,coeff,iopt,qual] = wmpalg('WMP',cuspamax,...
    'lstcpt',{{'wpsym4',1},{'wpsym4',2},'dct'},...
    'wmpcfs',0.8);
```

Plot the signal, approximation, residual, and the proportion of retained signal energy for each iteration in the matching pursuit result.

```
subplot(3,1,1)
plot(cuspamax)
hold on
plot(yfit)
hold off
legend('Signal','Approx.')
title('Signal and Approximation')
axis tight
subplot(3,1,2)
plot(r)
title('Residual')
axis tight
subplot(3,1,3)
plot(qual,'s-')
title('Quality / Iteration')
ylabel('Quality')
xlabel('Iteration')
```

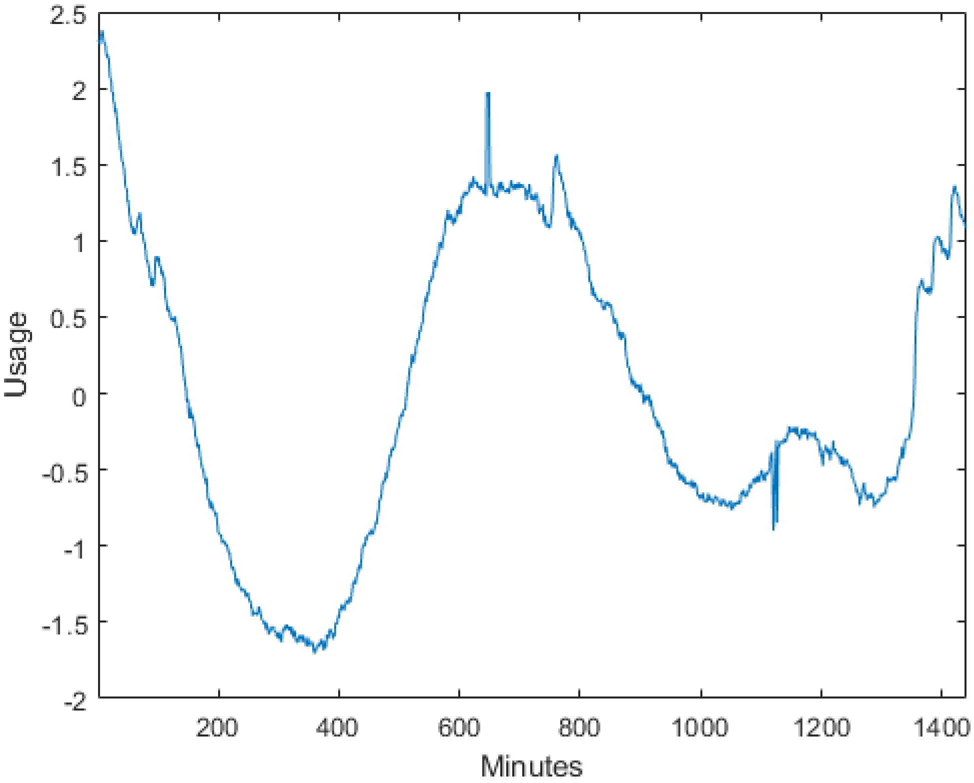


Matching Pursuit of Electricity Consumption Data

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])
```

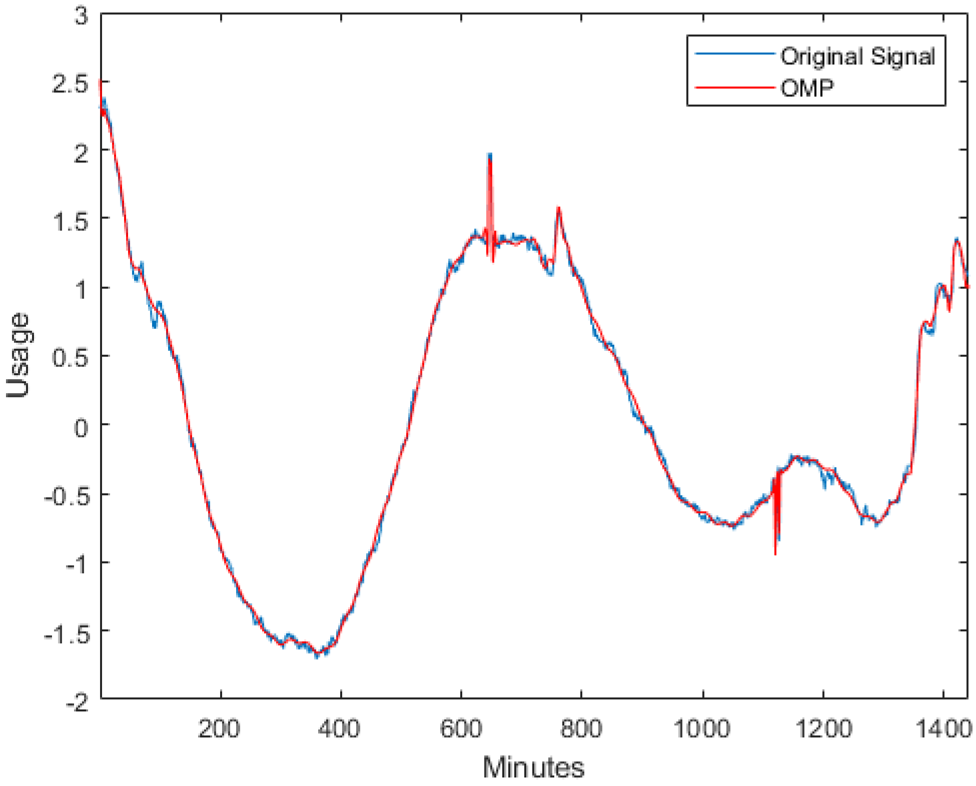


Specify a dictionary for matching pursuit consisting of the Daubechies' extremal-phase wavelet with 2 vanishing moments at level 2, the Daubechies' least-asymmetric wavelet with 4 vanishing moments at levels 1 and 4, the discrete cosine transform-II basis, and the sine basis.

```
dictionary = {{'db4',2},'dct','sin',{'sym4',1},{'sym4',4}};
```

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,...
    'lstcpt',dictionary,'itermax',35);
plot(y)
hold on
plot(yfit,'r')
xlabel('Minutes')
ylabel('Usage');
legend('Original Signal','OMP','Location','NorthEast')
set(gca,'xlim',[1 1440])
```



Input Arguments

MPALG — Matching pursuit algorithm

'BMP' (default) | 'OMP' | 'WMP'

Matching pursuit algorithm, specified as one of the following:

- 'BMP' — Basic matching pursuit
- 'OMP' — Orthogonal matching pursuit
- 'WMP' — Weak orthogonal matching pursuit

See "Matching Pursuit Algorithms".

MPDICT — Matching pursuit dictionary

matrix

Matching pursuit dictionary, specified as a matrix. MPDICT is a N-by-P matrix, where N is equal to the length of the input signal, Y. In matching pursuit, MPDICT is commonly a frame, or overcomplete set of vectors. You may use the name-value argument 'lstcpt' to specify a dictionary instead of using MPDICT.

Data Types: double

Y — Signal

vector

Signal for matching pursuit, specified as a vector. The row dimension of MPDICT must match the length of Y.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: yfit = wmpalg('OMP',y,lstcpt={'dct'}) specifies the DCT-II dictionary.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: yfit = wmpalg('OMP',y,'lstcpt',{'dct'})

itermax — Maximum number of iterations

25 (default) | positive integer

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'.

Data Types: double

lstcpt — Valid subdictionaries

cell array

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:

$$\phi_k(n) = \begin{cases} \frac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\frac{2}{N}}\cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right) & k = 1, 2, \ldots, N-1. \end{cases}$$

- '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:

  $$\phi_k(n) = \delta(n - k) \quad k = 0, 1, \ldots N$$

Data Types: `double`

`maxerr` — Maximum relative error
cell array

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.

Data Types: `double`

`wmpcfs` — Optimality factor
`0.6` (default) | scalar

Optimality factor for weak orthogonal matching pursuit. The optimality factor is a real number in the interval (0,1]. This name-value argument is only valid when `MPALG` is `'WMP'`.

Data Types: `double`

Output Arguments

`YFIT` — Adaptive greedy approximation
vector

Adaptive greedy approximation of the input signal, `Y`, in the dictionary

Data Types: `double`

`R` — Residual
vector

Residual after matching pursuit terminates

Data Types: `double`

`COEFF` — Expansion coefficients
vector

Expansion coefficients in the dictionary. The selected dictionary atoms weighted by the expansion coefficients yield the approximated signal, `YFIT`.

Data Types: `double`

`IOPT` — Column indices
vector

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`.

Data Types: `double`

`QUAL` — Proportion of retained signal energy
vector

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.

Data Types: `double`

`X` — Normalized matching pursuit dictionary
matrix

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.

Data Types: `double`

Compatibility Considerations

`wmpalg` no longer supports plotting
Errors starting in R2022a

The `wmpalg` function no longer supports the name-value arguments `stepplot` and `typeplot`. Remove all instances from your code. Instead, use MATLAB plotting commands. See the example "Change Optimality Factor for Weak Orthogonal Matching Pursuit" on page 1-1622.

`wmpalg` is no longer recommended
Not recommended starting in R2022a

The `wmpalg` function is no longer recommended. Use `sensingDictionary` with `matchingPursuit` and `basisPursuit`.

References

[1] Cai, T. Tony, and Lie Wang. “Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise.” *IEEE Transactions on Information Theory* 57, no. 7 (July 2011): 4680–88. https://doi.org/10.1109/TIT.2011.2146090.

[2] Donoho, D.L., M. Elad, and V.N. Temlyakov. “Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise.” *IEEE Transactions on Information Theory* 52, no. 1 (January 2006): 6–18. https://doi.org/10.1109/TIT.2005.860430.

[3] Mallat, S.G. and Zhifeng Zhang. “Matching Pursuits with Time-Frequency Dictionaries.” *IEEE Transactions on Signal Processing* 41, no. 12 (December 1993): 3397–3415. https://doi.org/10.1109/78.258082.

[4] Tropp, J.A. “Greed Is Good: Algorithmic Results for Sparse Approximation.” *IEEE Transactions on Information Theory* 50, no. 10 (October 2004): 2231–42. https://doi.org/10.1109/TIT.2004.834793.

See Also

`sensingDictionary` | `matchingPursuit` | `basisPursuit`

Topics

“Matching Pursuit”
“Matching Pursuit Algorithms”

Introduced in R2012a

wmpdictionary

(To be removed) Dictionary for matching pursuit

Note `wmpdictionary` will be removed in a future release. Use `sensingDictionary` instead. For more information, see “Compatibility Considerations”.

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.

Examples

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'});
```

Input Arguments

N — Input signal length

positive integer

Length of your input signal, specified as a positive integer. The dictionary atoms are constructed to have `N` elements. `N` equals the row dimension of the dictionary, `MPDICT`.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MPDICT = wmpdictionary(100,lstcpt={'dct','RnIdent'})`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `MPDICT = wmpdictionary(100,'lstcpt',{'dct','RnIdent'})`

addbeg — Prepended subdictionary

matrix

Prepended subdictionary, specified as 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.

Data Types: `double`

addend — Appended subdictionary

matrix

Appended subdictionary, specified as 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 appended to the default dictionary. The column vectors in the appended subdictionary do not have to be unit-norm.

Data Types: `double`

`lstcpt` — Valid subdictionaries

cell array

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:

  $$\phi_k(n) = \begin{cases} \frac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\frac{2}{N}}\cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right) & k = 1, 2, \ldots, N - 1. \end{cases}$$
- `'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:

  $$\phi_k(n) = \delta(n - k) \quad k = 0, 1, \ldots N$$

Data Types: `double`

Output Arguments

`MPDICT` — Matching pursuit dictionary

matrix

Matching pursuit dictionary, returned as a matrix. `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

vector

Number of vectors in subdictionaries, returned as a vector. `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` — Dictionary description

cell array

Dictionary description, returned as a cell array. `LST` is a 1-by-*L* cell array, where *L* 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` — Number of elements for each subdictionary
cell array

Number of elements for each subdictionary, returned as a cell array. `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.

More About

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-preserving formulation that guarantees convergence. See “Matching Pursuit Algorithms” for more details.

Compatibility Considerations

`wmpdictionary` will be removed

Not recommended starting in R2022a

The `wmpdictionary` function will be removed in a future release. Use `sensingDictionary` instead.

Functionality	Result	Use Instead	Compatibility Considerations

`MPDICT = wmpdictionary(N)`	Still runs	Execute these steps:
		1 Create a `sensingDictionary` using pre-built support for wavelet and DCT frames:
		`A1 = sensingDictionary('Size',N,...`
		`'Type',{'dwt','dct'},...`
		`'Name',{'sym4'},...`
		`'Level',[5]);`
		2 Create a custom `sensingDictionary`:
		`T = linspace(0,1,N)';`
		`K = 1:ceil(N/2);`
		`T1 = repmat(T,1,numel(K));`
		`K1 = repmat(K,numel(T),1);`
		`Amat = sin(2*pi*(K1.*T1));`
		`A2 = sensingDictionary('CustomDictionary',Amat);`
		3 Concatenate the results:
		`MPDICT = [A1 A2];`
			• `sensingDictionary` provides pre-built support for a variety of frames, including Fourier, Gaussian and Bernoulli random distributions, and Walsh code.
			• `sensingDictionary` does not currently support wavelet packet bases.

Functionality	Result	Use Instead	Compatibility Considerations

`MPDICT = wmpdictionary(N,'lstcpt',dtypes)`, where `dtypes` is a cell array of cell arrays with valid subdictionaries	Still runs	Each cell array in `dtypes` describes one subdictionary. To specify a subdictionary in `sensingDictionary`, use the `Type`, `Name`, and `Level` name-value arguments.
		For example:
		• Replace
		`mpdict = wmpdictionary(100,'lstcpt',{'dct','RnIden`
		with
		`D = sensingDictionary('Size',100,'Type',{'dct','ey`
		• Replace
		`mpdict = wmpdictionary(100,...`
		`'lstcpt',{{'db4',3},{'dct'}});`
		with
		`x = sensingDictionary('Size',100,...`
		`'Type',{'dwt','dct'},...`
		`'Name',{'db4'},...`
		`'Level',[3 0])`
			For the wavelet option, `sensingDictionary` and `wmpdictionary` behave differently.
			• `wmpdictionary` returns the wavelets at all levels and the scaling functions at the final level.
			• `sensingDictionary` returns the wavelets at only the final level.
			For example,
			`mpdict = wmpdictionary(100,'lstcpt',{{'db1',2}});`
			returns the scaling functions for level 2, the wavelets for level 2, and the wavelets for level 1, whereas
			`A = sensingDictionary('Size',100,'Type',{'dwt'},'Name',{'db1'},'Level',2);`
			only returns the wavelets at level 2.

Functionality	Result	Use Instead	Compatibility Considerations
`[~,NBVECT] = wmpdictionary(N)` or `[~,NBVECT] = wmpdictionary(N,'lstcpt')`	Still runs	`NBVECT` is the number of vectors in each subdictionary. The number of vectors in a subdictionary of a `sensingDictionary` object depends on the associated basis type. • For a non-random basis type, the number of vectors is `N`. • For a random basis type, the number of vectors is the column size you specified when you created the `sensingDictionary` object. • For a custom `sensingDictionary`, the number of vectors is the column size you specified when you created the `sensingDictionary` object.	You can also use the `subdict` method of `sensingDictionary` to extract the vectors.

References

[1] Cai, T. Tony, and Lie Wang. “Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise.” *IEEE Transactions on Information Theory* 57, no. 7 (July 2011): 4680–88. https://doi.org/10.1109/TIT.2011.2146090.

[2] Donoho, D.L., M. Elad, and V.N. Temlyakov. “Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise.” *IEEE Transactions on Information Theory* 52, no. 1 (January 2006): 6–18. https://doi.org/10.1109/TIT.2005.860430.

[3] Mallat, S.G. and Zhifeng Zhang. “Matching Pursuits with Time-Frequency Dictionaries.” *IEEE Transactions on Signal Processing* 41, no. 12 (December 1993): 3397–3415. https://doi.org/10.1109/78.258082.

[4] Tropp, J.A. “Greed Is Good: Algorithmic Results for Sparse Approximation.” *IEEE Transactions on Information Theory* 50, no. 10 (October 2004): 2231–42. https://doi.org/10.1109/TIT.2004.834793.

See Also

`sensingDictionary` | `matchingPursuit` | `basisPursuit`

Topics

"Matching Pursuit"
"Matching Pursuit Algorithms"

Introduced in R2012a

wmspca

Multiscale principal component analysis

Syntax

```
[xsim,qual,npc_out,decsim,pca_params] = wmspca(x,level,wname,npc_in)
[ ___ ] = wmspca(x,level,wname,'mode',extmode,npc_in)
[ ___ ] = wmspca(dec,npc_in)
```

Description

`[xsim,qual,npc_out,decsim,pca_params] = wmspca(x,level,wname,npc_in)` returns a simplified version `xsim` of the input matrix `x` obtained from the wavelet-based multiscale principal component analysis (PCA). The wavelet decomposition is performed using the decomposition level `level` and the wavelet `wname`.

`[ ___ ] = wmspca(x,level,wname,'mode',extmode,npc_in)` uses the specified discrete wavelet transform (DWT) extension mode `extmode`.

`[ ___ ] = wmspca(dec,npc_in)` uses the wavelet decomposition structure `dec`. `dec` is expected to be the output of `mdwtdec`.

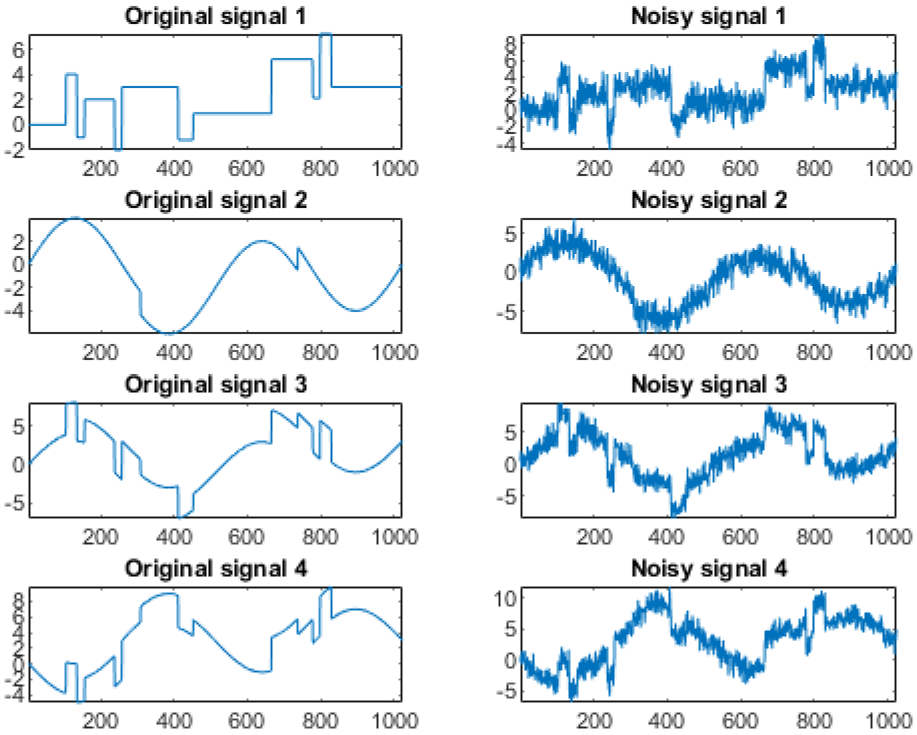
Examples

Wavelet Principal Component Analysis of Noisy Multivariate Signal

Use wavelet multiscale principal component analysis to denoise a multivariate signal.

Load the dataset consisting of four signals of length 1024. Plot the original signals and the signals with additive noise.

```
load ex4mwden
for i = 0:3
    subplot(4,2,2*i+1)
    plot(x_orig(:,i+1))
    axis tight
    title(['Original signal ',num2str(i+1)])
    subplot(4,2,2*i+2)
    plot(x(:,i+1))
    axis tight
    title(['Noisy signal ',num2str(i+1)])
end
```

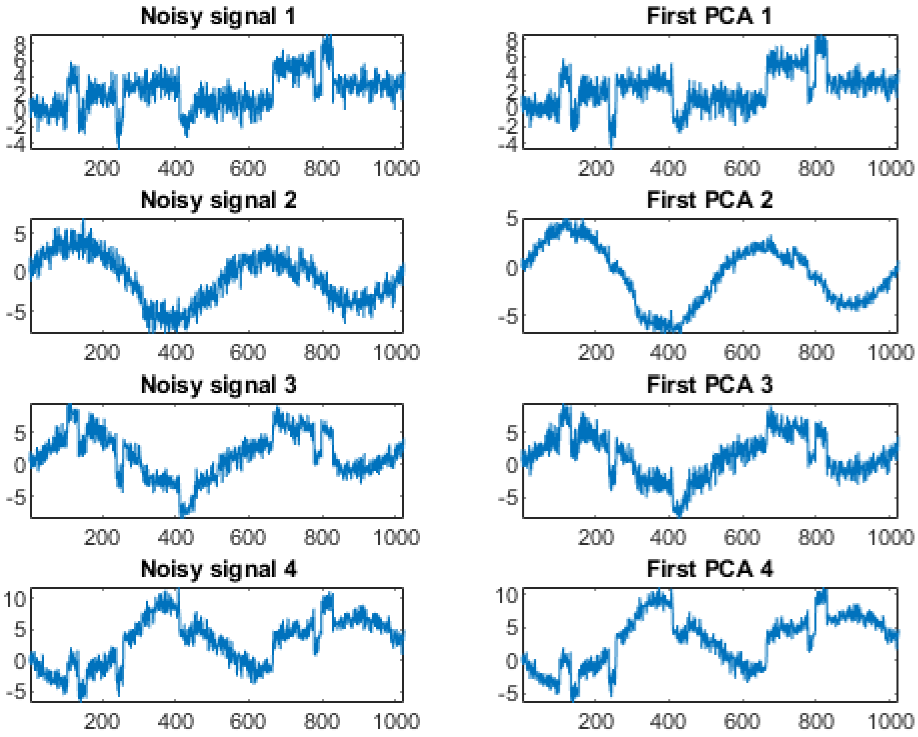


Perform the first multiscale wavelet PCA using the Daubechies least-asymmetric wavelet with four vanishing moments, `sym4`. Obtain the multiresolution decomposition down to level 5. Use the heuristic rule to decide how many principal components to retain.

```
level = 5;
wname = 'sym4';
npc = 'heur';
[x_sim,qual,npcA] = wmspca(x,level,wname,npc);
```

Plot the result and examine the quality of the approximation.

```
for i = 0:3
    subplot(4,2,2*i+1)
    plot(x(:,i+1))
    axis tight
    title(['Noisy signal ',num2str(i+1)])
    subplot(4,2,2*i+2)
    plot(x_sim(:,i+1))
    axis tight
    title(['First PCA ',num2str(i+1)])
end
```



```
qual
```

```
qual = 1×4

   97.4372   94.5520   97.7362   99.5219
```

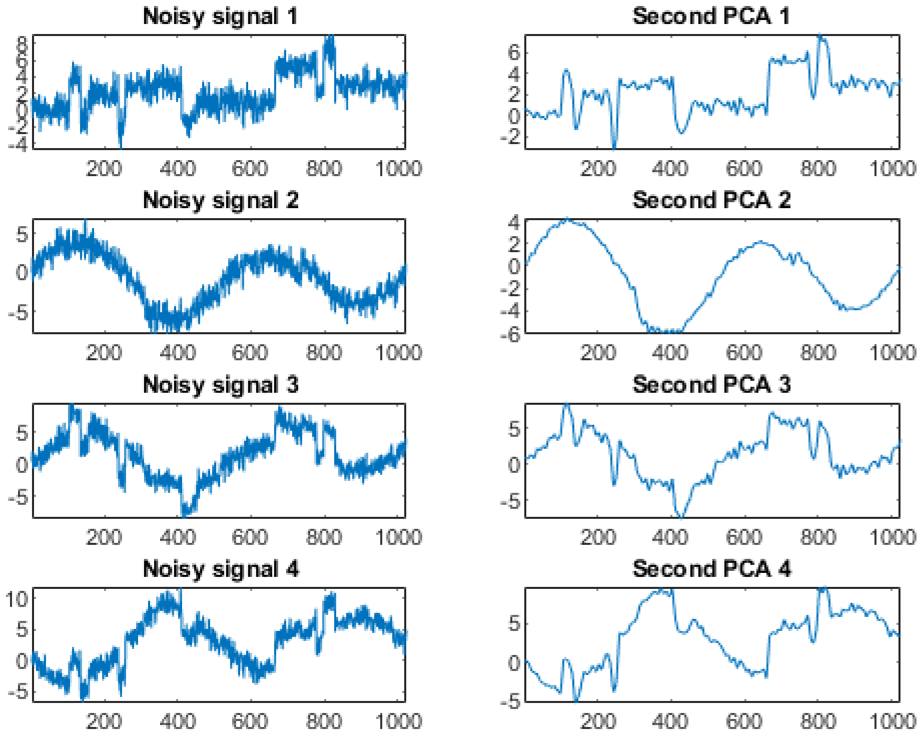
The quality results are all close to 100%. The `npc` vector gives the number of principal components retained at each level.

Suppress the noise by removing the principal components at levels 1-3. Perform the multiscale PCA again.

```
npcA(1:3) = zeros(1,3);
[x_sim,qual,npcB] = wmspca(x,level,wname,npcA);
```

Plot the result.

```
for i = 0:3
    subplot(4,2,2*i+1)
    plot(x(:,i+1))
    axis tight
    title(['Noisy signal ',num2str(i+1)])
    subplot(4,2,2*i+2)
    plot(x_sim(:,i+1))
    axis tight
    title(['Second PCA ',num2str(i+1)])
end
```



Input Arguments

x — Multisignal

real-valued matrix

Multisignal, specified as a real-valued matrix. The matrix `x` contains *P* signals of length *N* stored column-wise (*N* > *P*).

Data Types: `double`

`level` — Level of decomposition

positive integer

Level of decomposition, specified as a positive integer. `wmspca` does not enforce a maximum level restriction. Use `wmaxlev` to ensure that the wavelet coefficients are free from boundary effects. If boundary effects are not a concern, a good rule is to set `level` less than or equal to `fix(log2(length(N)))`, where *N* is the signal length.

Data Types: `double`

`wname` — Analyzing wavelet

character vector | string scalar

Analyzing wavelet, specified as a character vector or string scalar. 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 biorthogonal 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).

npc_in — Principal components parameter

vector | "kais" | "heur" | "nodet"

Principal components parameter, specified as a vector, character vector, or string scalar.

- If `npc_in` is a vector, then it must be of length `level+2`. The vector `npc_in` contains the number of retained principal components for each PCA performed:
  - `npc_in(d)` is the number of retained noncentered principal components for details at level `d`, for 1 ≤ `d` ≤ `level`.
  - `npc_in(level+1)` is the number of retained non-centered principal components for approximations at level `level`.
  - `npc_in(level+2)` is the number of retained principal components for final PCA after wavelet reconstruction.

  `npc_in` must be such that 0 ≤ `npc_in(d)` ≤ *P*, where *P* is the number of signals, for 1 ≤ `d` ≤ `level`+2.
- If `npc_in` is `"kais"`, then the number of retained principal components is selected automatically using Kaiser's rule. Kaiser's rule keeps the components associated with eigenvalues exceeding the mean of all eigenvalues.
- If `npc_in` is `"heur"`, then the number of retained principal components is selected automatically using the heuristic rule. The heuristic rule keeps the components associated with eigenvalues greater than 0.05 times the sum of all eigenvalues.
- If `npc_in` is `"nodet"`, then the details are "killed" and all the approximations are retained.

Data Types: `double` | `string` | `char`

extmode — Extension mode

'zpd' | 'sp0' | 'spd' | ...

Extension mode used when performing the wavelet decomposition, specified as:

mode	DWT Extension Mode
`'zpd'`	Zero extension
`'sp0'`	Smooth extension of order 0
`'spd'` (or `'sp1'`)	Smooth extension of order 1
`'sym'` or `'symh'`	Symmetric extension (half point): boundary value symmetric replication

mode	DWT Extension Mode
'symw'	Symmetric extension (whole point): boundary value symmetric replication
'asym' or 'asymh'	Antisymmetric extension (half point): boundary value antisymmetric replication
'asymw'	Antisymmetric extension (whole point): boundary value antisymmetric replication
'ppd', 'per'	Periodized extension If the signal length is odd and mode is 'per', an extra sample equal to the last value is added to the right and the extension is performed in 'ppd' mode. If the signal length is even, 'per' is equivalent to 'ppd'. This rule also applies to images.

The global variable managed by dwtmode specifies the default extension mode. Use dwtmode to determine the extension modes.

dec — Wavelet decomposition structure

structure

Wavelet decomposition structure of a multisignal, specified as a structure. dec is expected to be the output of mdwtdec. The multisignal input of mdwtdec is a matrix *A*, where the signals are arranged column-wise. If *A* is *N*-by-*P*, then *N* must be greater than *P*.

Data Types: double

Output Arguments

xsim — Simplified multivariate multisignal

matrix

Simplified multivariate multisignal, returned as a matrix. The dimensions of xsim equal the dimensions of x.

Data Types: double

qual — Quality of column reconstructions

vector

Quality of column reconstructions, returned as a vector of length *P*, where *P* is equal to size(x,2). qual contains the quality of column reconstructions given by the relative mean square errors in percent.

Data Types: double

npc_out — Number of retained principal components

vector

Number of retained principal components, returned as a vector. If npc_in is a vector, then npc_out equals npc_in.

Data Types: double

`decsim` — Wavelet decomposition
structure

Wavelet decomposition of the simplified multisignal `xsim`, returned as a structure with the following fields:

- `dirDec` — `'c'` (column), indicator of decomposition direction
- `level` — Level of wavelet decomposition
- `wname` — Wavelet name
- `dwtFilters` — Structure with four fields:
  - `LoD` — Lowpass decomposition filter
  - `HiD` — Highpass decomposition filter
  - `LoR` — Lowpass reconstruction filter
  - `HiR` — Highpass reconstruction filter
- `dwtEXTM` — DWT extension mode
- `dwtShift` — DWT shift parameter (0 or 1)
- `dataSize` — Size of `x`
- `ca` — Approximation coefficients at level `level`
- `cd` — Cell array of detail coefficients, from level 1 to level `level`

`ca` and `cd{k}`, for *k* from 1 to `level`, are matrices, where the coefficients are stored as columns.

`pca_params` — PCA parameters
structure array

PCA parameters, returned as a structure array of length `level+2`, where:

- `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_out`

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

[1] Bakshi, Bhavik R. “Multiscale PCA with Application to Multivariate Statistical Process Monitoring.” *AIChE Journal* 44, no. 7 (July 1998): 1596–1610. https://doi.org/10.1002/aic.690440712.

See Also

`wmulden`

Topics

“Multiscale Principal Components Analysis”

Introduced in R2006b

wmulden

Wavelet multivariate denoising

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 denoised version X_DEN of the input matrix X. The strategy combines univariate wavelet denoising 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 column-wise 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.

Denoising Parameters: TPTR and SORH

The default values for the denoising 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 denoised 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 denoising method (TPTR).
- DEN_Params.thrTYPE is a character variable containing the type of the thresholding (SORH).

Special Cases

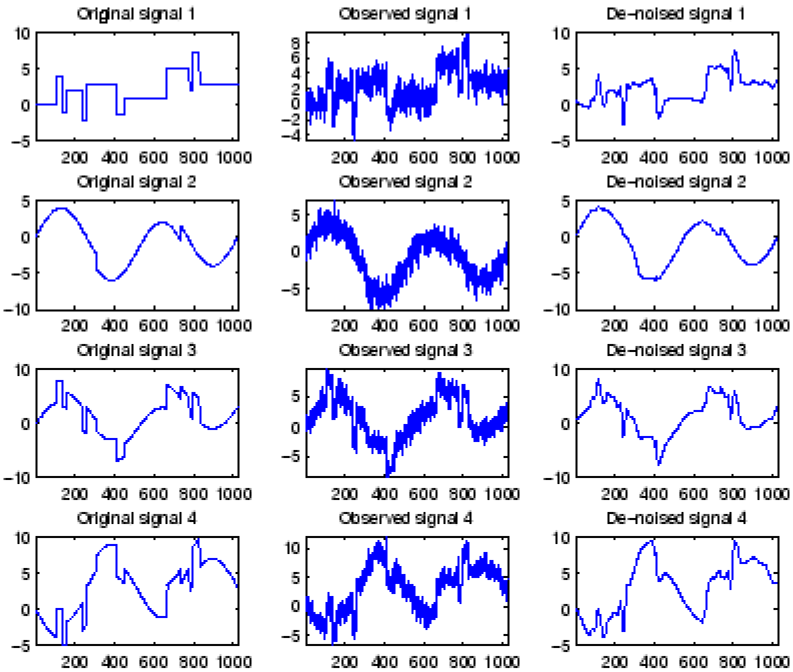
[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN) returns the wavelet decomposition DEC and the Principal Components Estimates PCA_Params.

[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 denoising 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 denoising.
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
                                   npc_fin, tptr, sorh);

% Display the original and denoised 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(['Denoised 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 denoised.

% 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 denoising procedure is a generalization of the one-dimensional strategy. It combines univariate wavelet denoising 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

Functions
`wmspca` | `wdenoise`

Apps
`Wavelet Signal Denoiser`

Introduced in R2006b

wnoise

Noisy wavelet test data

Syntax

```
x = wnoise(fun,n)
[x,xn] = wnoise(fun,n,sqrtsnr)
[x,xn] = wnoise( ___ ,init)
```

Description

`x = wnoise(fun,n)` returns values `x` of the test signal `fun` evaluated at 2^n linearly spaced points from 0 to 1.

`[x,xn] = wnoise(fun,n,sqrtsnr)` returns `x` rescaled such that the standard deviation of x is `sqrtsnr`. `xn` is `x` corrupted by additive Gaussian white noise $N(0,1)$ and has a signal-to-noise ratio (SNR) of `sqrtsnr`^{2}.

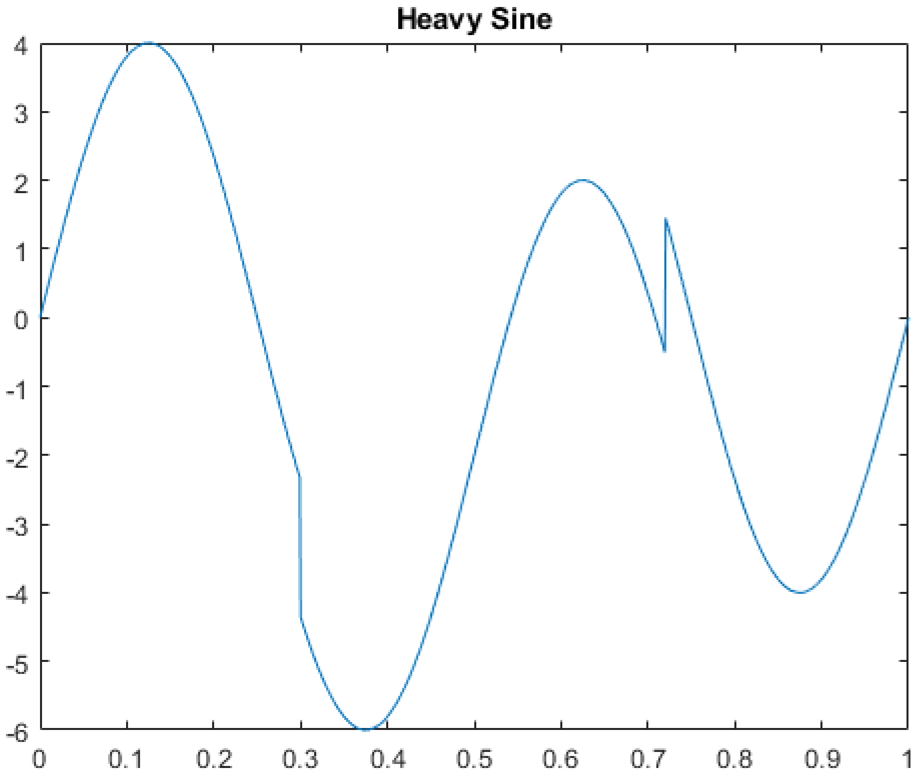
`[x,xn] = wnoise( ___ ,init)` sets the generator seed to `init` before generating additive Gaussian white noise $N(0,1)$.

Examples

Plot Wavelet Test Signals

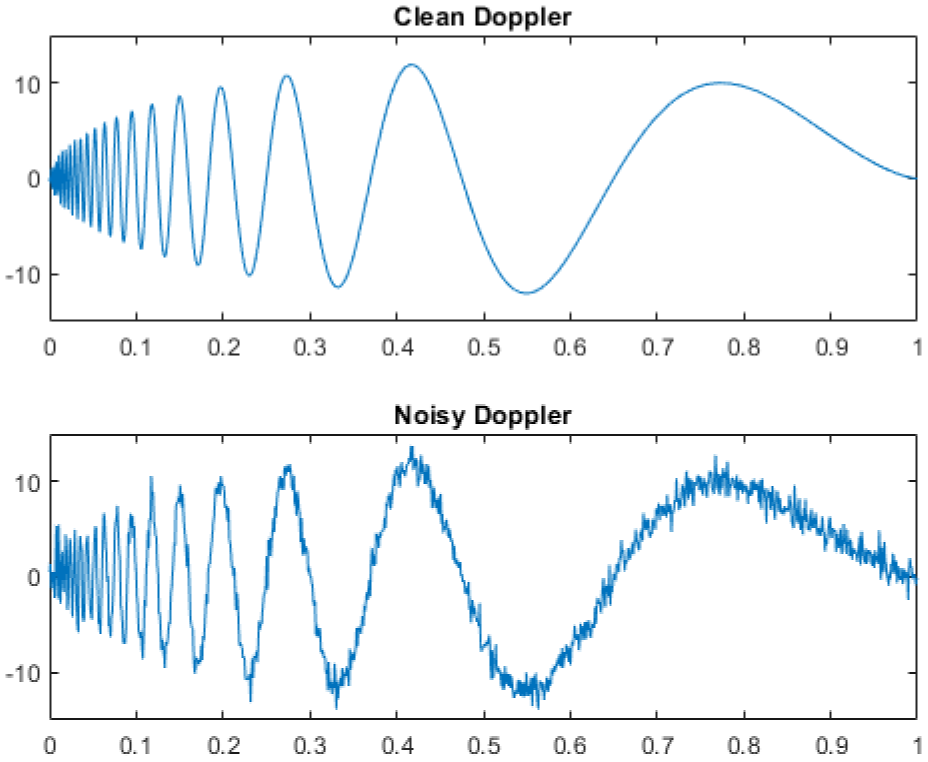
There are six test signals. Generate and plot 2^{10} samples of the third test signal, `heavy sine`.

```
loc = linspace(0,1,2^10);
x = wnoise(3,10);
plot(loc,x)
title('Heavy Sine')
```



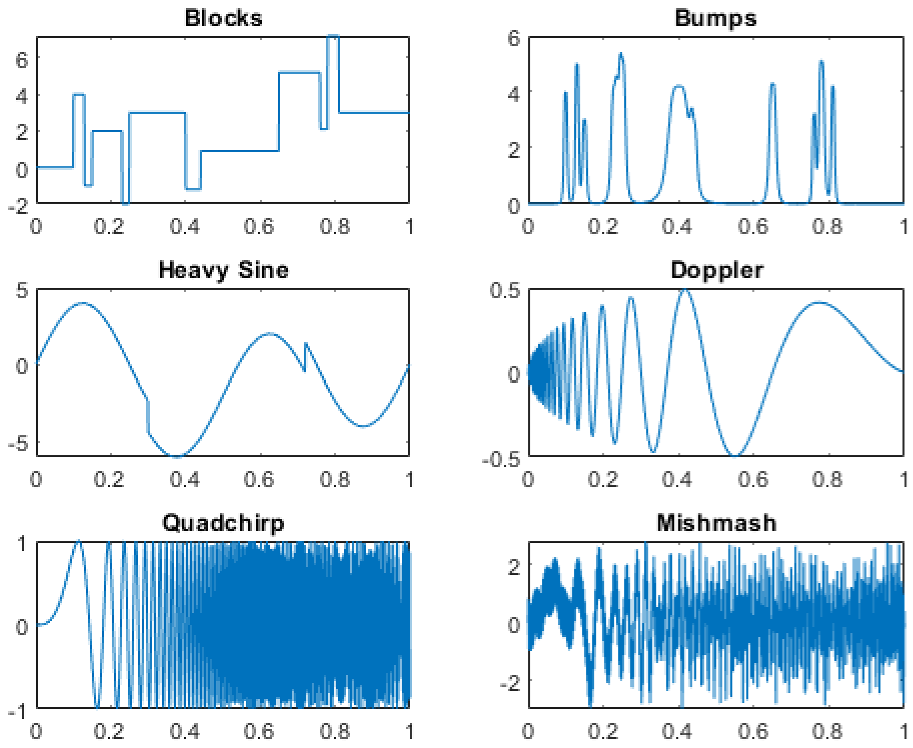
Generate and plot 2^{10} samples of the `doppler` test signal and a noisy version of `doppler` with a square root of the signal-to-noise ratio equal to 7.

```
[x,noisyx] = wnoise('doppler',10,7);
subplot(2,1,1)
plot(loc,x)
title('Clean Doppler')
ylim([-15 15])
subplot(2,1,2)
plot(loc,noisyx)
title('Noisy Doppler')
ylim([-15 15])
```



Plot all the test functions.

```
testFunctions = {'Blocks','Bumps','Heavy Sine','Doppler','Quadchirp','Mishmash'};
for i=1:6
    x = wnoise(lower(testFunctions{i}),10);
    subplot(3,2,i)
    plot(loc,x)
    title(testFunctions{i})
end
```



Input Arguments

`fun` — Wavelet test function

positive integer | character array

Wavelet test function, specified as one of the values listed here. The six test functions are due to Donoho and Johnstone [1], [2].

- 1 or `'blocks'`
- 2 or `'bumps'`
- 3 or `'heavy sine'`
- 4 or `'doppler'`
- 5 or `'quadchirp'`
- 6 or `'mishmash'`

`n` — Exponent

positive integer

Exponent used to determine the number of linearly spaced points from 0 to 1 to evaluate the test function, specified as a positive integer. The number of linearly spaced points is 2^n.

`sqrtsnr` — Square root of SNR

positive real number

Square root of SNR, specified by a positive real number. The test values x are rescaled such that the standard deviation of x is sqrtsnr. xn is equal to x corrupted by additive Gaussian white noise *N*(0,1) and has an SNR of sqrtsnr^2.

init — Seed
nonnegative integer

Seed used to initialize the random number generator, specified as a nonnegative integer. init is used to generate additive Gaussian white noise.

Example: [a,b] = wnoise(4,10,7,2055415866); returns a noisy version of the fourth test signal using the seed init = 2055415866.

Output Arguments

x — Test signal
real-valued vector

Test signal, returned as a real-valued vector of length 2^n. x are the values of the test function specified by fun evaluated at the 2^n evenly spaced points from 0 to 1. If sqrtsnr is set, the standard deviation of x is sqrtsnr.

xn — Noisy test signal
real-valued vector

Noisy test signal, returned as a real-valued vector of length 2^n. xn is x corrupted by additive Gaussian white noise *N*(0,1) and has an SNR of sqrtsnr^2.

References

[1] Donoho, D. L., and I. M. Johnstone. “Ideal spatial adaptation by wavelet shrinkage.” *Biometrika*. Vol. 81, Issue 3, 1994, pp. 425–455.

[2] Donoho, D. L., and I. M. Johnstone. “Adapting to unknown smoothness via wavelet shrinkage.” *Journal of the American Statistical Association*. Vol. 90, 1995, pp. 1200–1224.

See Also

wdenoise | wden

Introduced before R2006a

wnoisest

Estimate noise of 1-D wavelet coefficients

Syntax

```
stdc = wnoisest(c,l,s)
stdc = wnoisest(c)
```

Description

`stdc = wnoisest(c,l,s)` returns estimates of the detail coefficients' standard deviation for levels specified in `s`. `[c,l]` is a multilevel wavelet decomposition structure and is the output of `wavedec`.

The estimator used is Median Absolute Deviation / 0.6745, well suited for zero-mean Gaussian white noise in the denoising one-dimensional model (see `thselect` for more information).

`stdc = wnoisest(c)` returns estimates of the standard deviations of `c`, where `c` is a one-dimensional cell array or a numeric array.

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 = 1×2

    0.9650    1.0279
```

In spite of the outliers, `wnoisest` provides a robust estimate of the standard deviation.

Input Arguments

`c` — Input

vector | matrix | cell array

Input, specified as a vector, matrix, or 1-D cell array.

- When used in the syntax `stdc = wnoisest(c,l,s)`, `c` is the wavelet decomposition output of `wavedec`: `[c,l] = wavedec(x,`*`N`*`,`*`wname`*`)`. The bookkeeping vector `l` contains the number of coefficients by level.
- When used in the syntax `stdc = wnoisest(c)`, `c` is either a numeric matrix or 1-D cell array.

Data Types: `double`

`l` — Bookkeeping vector

vector

Bookkeeping vector, specified as a vector of positive integers. `l` is the output of `wavedec`: `[c,l] = wavedec(x,`*`N`*`,`*`wname`*`)`. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition `c` by level.

Data Types: `double`

`s` — Detail coefficient levels

vector

Detail coefficient levels, specified as a vector of positive integers less than or equal to *N*, where *N* is the level of the wavelet decomposition used to obtain [`c`,`l`]. Specifically, *`N`*` = length(l)-2`.

Data Types: `double`

Output Arguments

`stdc` — Standard deviation estimates

vector | cell array

Standard deviation estimates, returned as a vector or cell array.

- If `c` is the output of `wavedec`, `stdc` are estimates of the detail coefficients' standard deviation for the levels specified in `s`.
- If `c` is a one-dimensional cell array, `stdc{`*`k`*`}` is an estimate of the standard deviation of `c{`*`k`*`}`, where *`k`*` = 1,…,length(c)`.
- If `c` is a numeric array, `stdc(`*`k`*`)` is an estimate of the standard deviation of `c(`*`k`*`,:)`, where *`k`*` = 1,…,size(c,1)`.

References

[1] Donoho, David L, and Iain M Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika* 81, no. 3 (September 1, 1994): 425–55. https://doi.org/10.1093/biomet/81.3.425.

[2] Donoho, David L., and Iain M. Johnstone. “Adapting to Unknown Smoothness via Wavelet Shrinkage.” *Journal of the American Statistical Association* 90, no. 432 (December 1995): 1200–1224. https://doi.org/10.1080/01621459.1995.10476626.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`thselect` | `wavedec` | `wdenoise`

Apps

`Wavelet Signal Denoiser`

Introduced before R2006a

wp2wtree

Extract wavelet tree from wavelet packet tree

Syntax

```
T = wp2wtree(T)
```

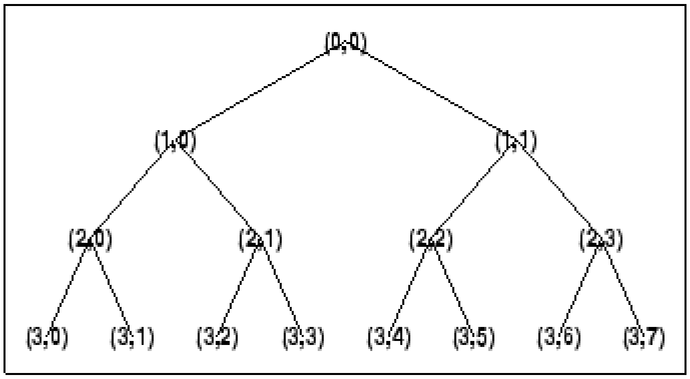
Description

`wp2wtree` is a one- or two-dimensional wavelet packet analysis function.

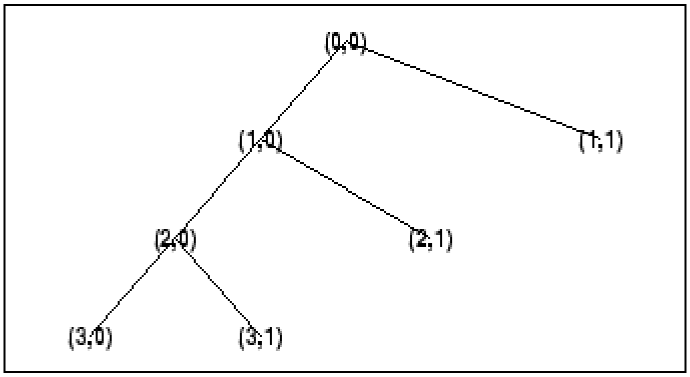
`T = wp2wtree(`*T*`)` computes the modified wavelet packet tree *T* corresponding to the wavelet decomposition tree.

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 denoising

Syntax

```
THR = wpbmpen(T,SIGMA,ALPHA)
wpbmpen(T,SIGMA,ALPHA,ARG)
```

Description

THR = wpbmpen(T,SIGMA,ALPHA) returns a global threshold THR for denoising. THR is obtained by a wavelet packet coefficients selection rule using a penalization method provided by Birgé-Massart.

T is a wavelet packet tree corresponding to the wavelet packet decomposition of the signal or image to be denoised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in the denoising 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 denoised 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))
```

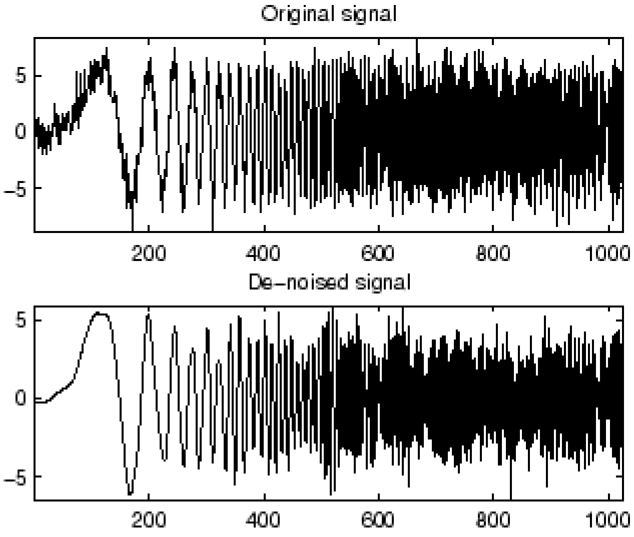
where c(k) are the wavelet packet coefficients sorted in decreasing order of their absolute value and n is the number of coefficients, then THR=$|c(t^*)|$.

wpbmpen(T,SIGMA,ALPHA,ARG) computes the global threshold and, in addition, plots three curves:

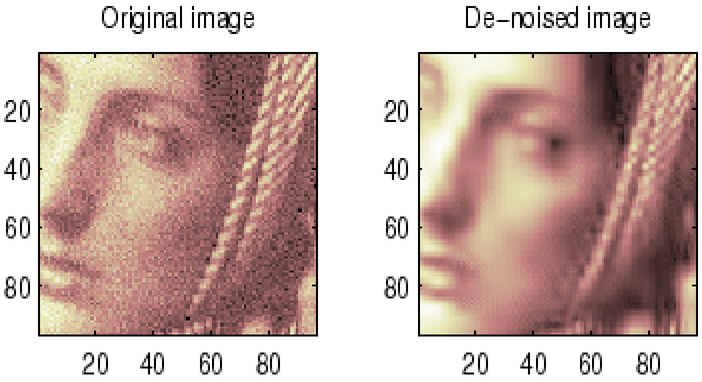
- 2*SIGMA^2*t*(ALPHA + log(n/t))
- sum(c(k)^2,k£t)
- crit(t)

Examples

```
% Example 1: Signal denoising.
% 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 denoising, using the recommended parameter.
alpha = 2;
thr = wpbmpen(tree,sigma,alpha)

thr =

    4.5740


% Use wpdencmp for denoising 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 denoised signals.
figure(1)
subplot(211), plot(x),
title('Original signal')
subplot(212), plot(xd)
title('De-noised signal')
```



```
% Example 2: Image denoising.
% 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 denoising.
alpha = 1.1;
thr = wpbmpen(tree,sigma,alpha)

thr =

   38.5125


% Use wpdencmp for denoising 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 denoised 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

wdenoise | wbmpen | wden | wdencmp | wpdencmp

Introduced before R2006a

wpcoef

Wavelet packet coefficients

Syntax

```
x = wpcoef(wpt,n)
x = wpcoef(wpt)
```

Description

`wpcoef` is a one- or two-dimensional wavelet packet analysis function.

`x = wpcoef(wpt,n)` returns the coefficients associated with the node `n` of the wavelet packet tree `wpt`. If node `n` does not exist, `x = []`.

`x = wpcoef(wpt)` is equivalent to `x = wpcoef(wpt,0)`.

Examples

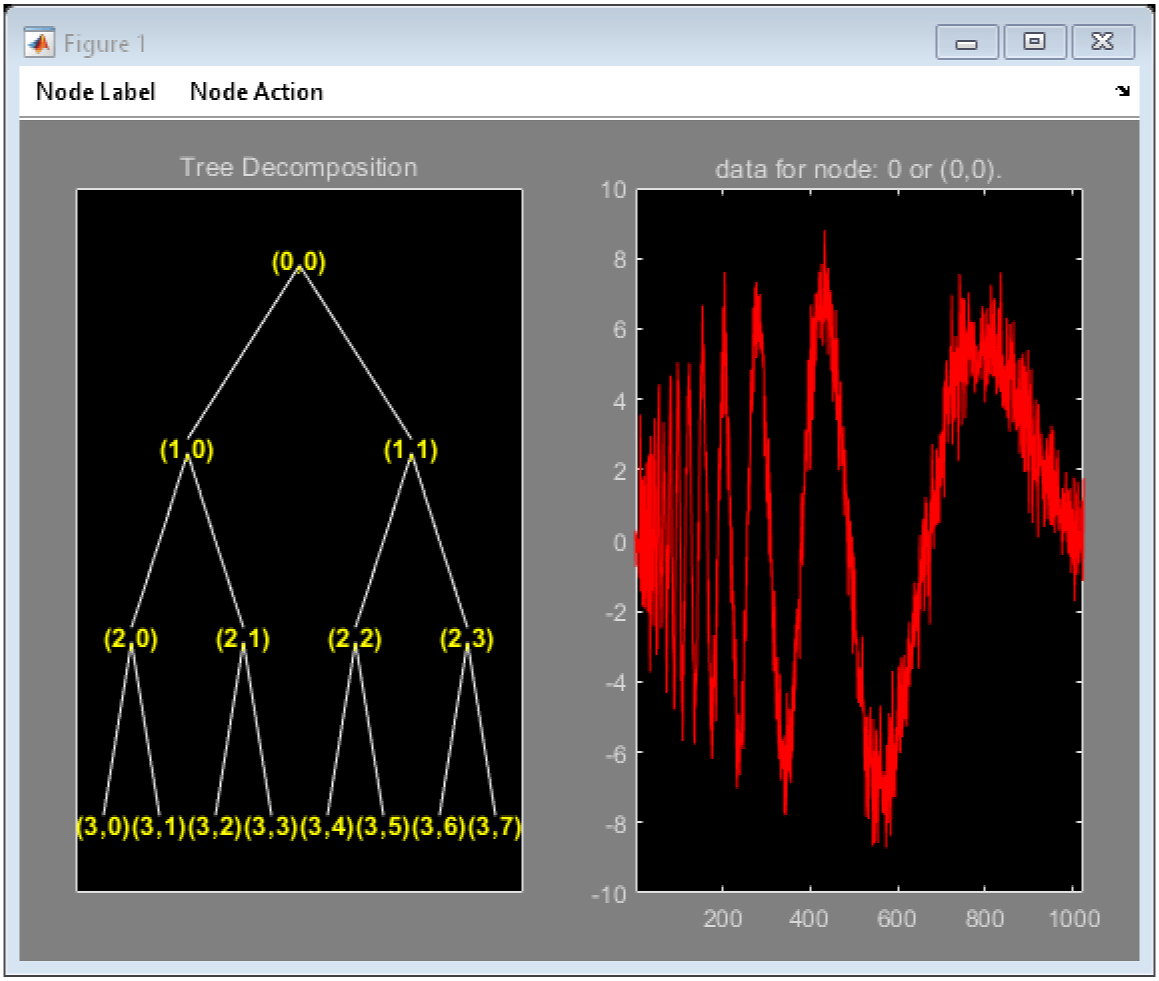
Obtain Wavelet Packet Coefficients

Load a 1-D signal. Save the current extension mode.

```
load noisdopp
x = noisdopp;
origMode = dwtmode('status','nodisp');
```

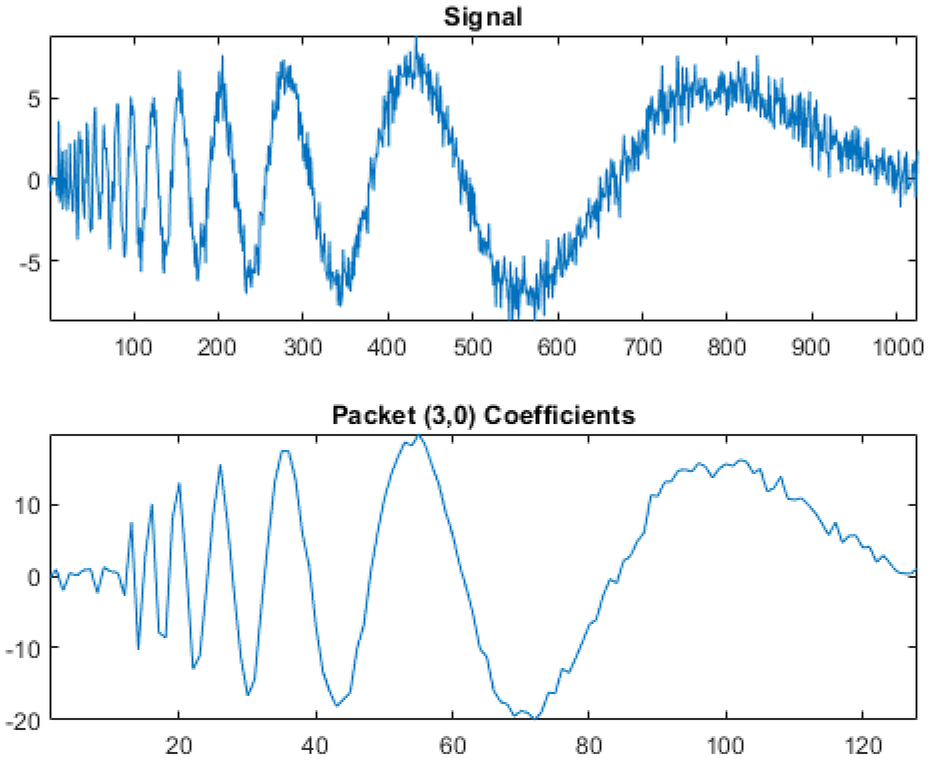
Use `dwtmode` to change the extension mode to zero-padding. Obtain the wavelet packet tree object corresponding to the 3-level wavelet packet decomposition of the signal using the `db1` wavelet. Plot the tree.

```
dwtmode('zpd','nodisp')
wpt = wpdec(x,3,"db1");
plot(wpt)
```



Obtain the coefficients at the node (3,0). Plot the signal and coefficients.

```
cfs = wpcoef(wpt,[3 0]);
subplot(2,1,1)
plot(x)
title('Signal')
axis tight
subplot(2,1,2)
plot(cfs)
title('Packet (3,0) Coefficients')
axis tight
```

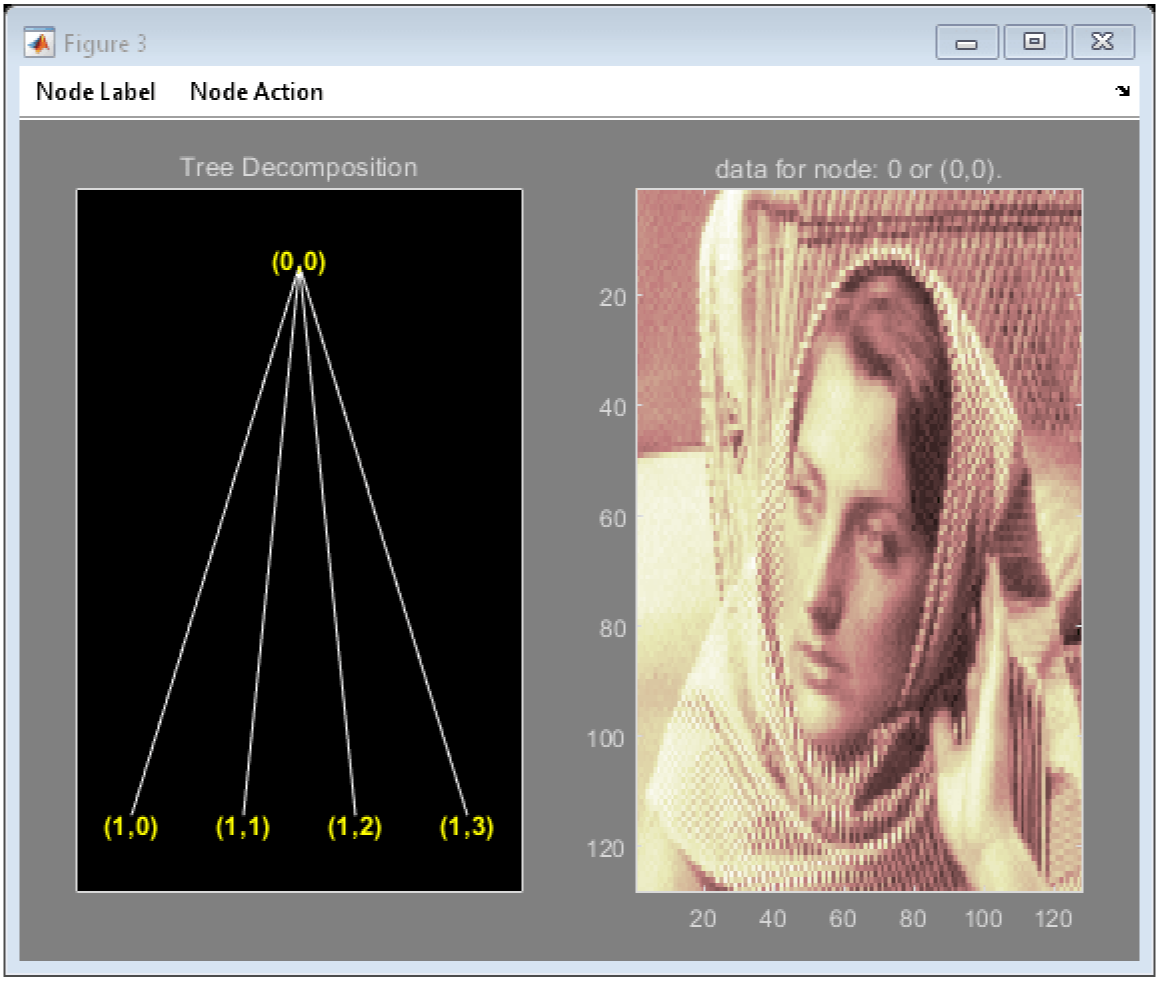


Load an image. Obtain the wavelet packet tree that corresponds to a one-level wavelet packet decomposition of the image using the `sym4` wavelet.

```
load woman2
t = wpdec2(X,1,'sym4');
```

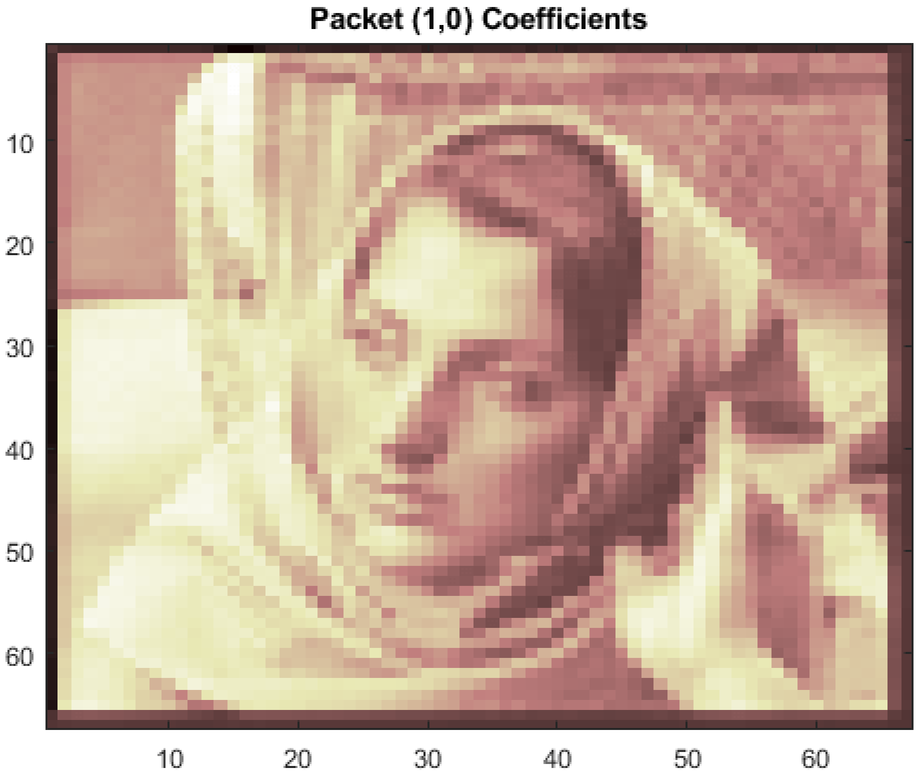
Plot the tree.

```
plot(t)
```



Obtain the coefficients at the node (1,0). Plot the coefficients.

```
cfs = wpcoef(t,[1 0]);
figure
imagesc(cfs)
title('Packet (1,0) Coefficients')
colormap(pink)
```



Restore the extension mode to the original setting.

```
dwtmode(origMode,'nodisp')
```

Input Arguments

wpt — Wavelet packet tree

wptree object

Wavelet packet tree, specified as a wptree object.

n — Node

0 (default) | nonnegative integer | 1-by-2 vector

Node in a wavelet packet tree, specified as a nonnegative integer, or pair of nonnegative integers. See depo2ind and ind2depo.

Example: If wpt = wpdec(1:256,2,"sym4"), then wpcoef(wpt,3) and wpcoef(wpt,[2 0]) specify the same node.

Data Types: double

Output Arguments

x — Node coefficients

vector | matrix

Node coefficients, returned as a vector or matrix.

Data Types: `double`

See Also

`wpcoef` | `wpdec` | `wpdec2` | `wprcoef` | `plot`

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

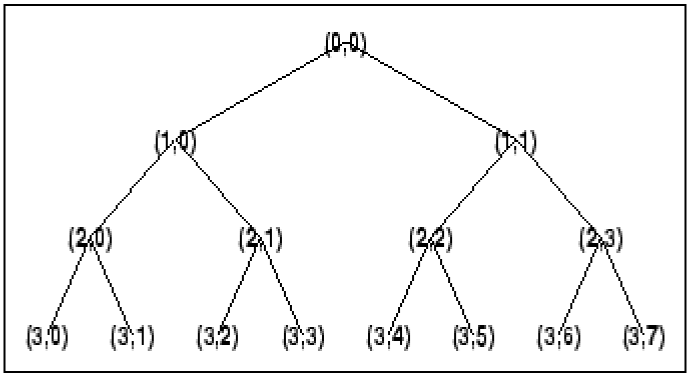
`wpcutree` is a one- or two-dimensional wavelet packet analysis function.

`T = wpcutree(T,L)` cuts the tree `T` at level `L`.

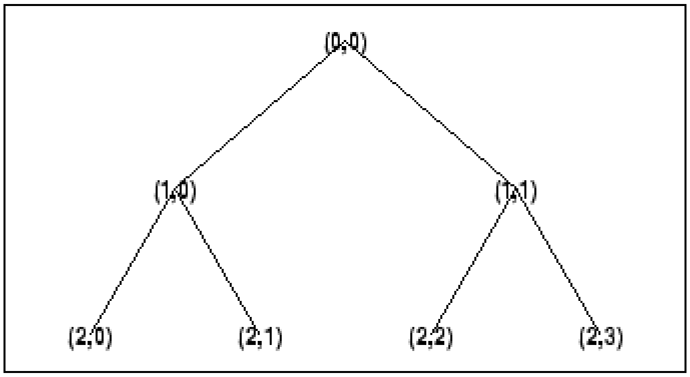
`[T,RN] = wpcutree(T,L)` returns the same arguments as above and, in addition, the vector `RN` contains the indices of the reconstructed nodes.

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

```
tobj = wpdec(x,n,wname)
tobj = wpdec(x,n,wname,etype,p)
```

Description

`tobj = wpdec(x,n,wname)` returns a wavelet packet tree object `tobj` corresponding to the wavelet packet decomposition of the vector `x` at level `n`, using Shannon entropy and the wavelet specified by `wname` (see `wfilters` for more information).

`tobj = wpdec(x,n,wname,etype,p)` uses the entropy type specified by `etype`. `p` is an optional parameter depending on the value of `etype`. See `wentropy` for more information.

Note `tobj = wpdec(x,n,wname)` is equivalent to `tobj = wpdec(x,n,wname,'shannon')`.

Examples

Visualize Wavelet Packet Tree

Load a signal.

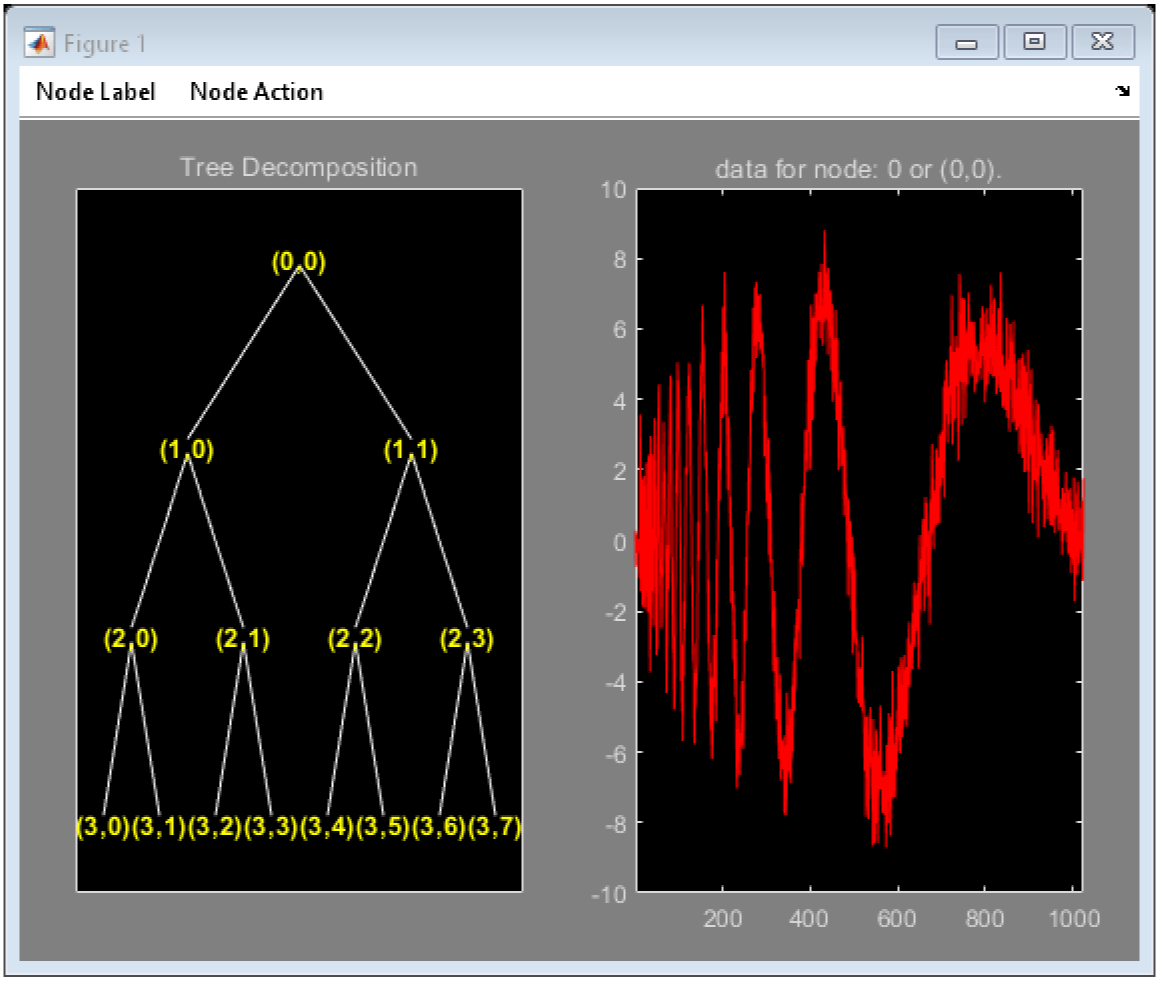
```
load noisdopp
```

Decompose the signal at level 3 with `db1` wavelet packets using Shannon entropy.

```
wpt = wpdec(noisdopp,3,'db1','shannon');
```

Plot the wavelet packet tree.

```
plot(wpt)
```



Input Arguments

x — Input data
real-valued numeric vector

Input data, specified as a real-valued numeric vector.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

n — Decomposition level
positive integer

Decomposition level, specified as a positive integer.

Data Types: single | double

wname — Wavelet
character vector | string scalar

Wavelet used in the wavelet packet decomposition, specified as a character vector or string scalar. The wavelet is from one of the following wavelet families: Daubechies, Symlets, Fejér-Korovkin, Discrete Meyer, Biorthogonal, and Reverse Biorthogonal. See `wfilters` for the wavelets available in each family.

etype — Entropy type
'shannon' | 'log energy' | 'threshold' | 'sure' | 'norm' | 'user' | '*FunName*'

Entropy type, specified as one of the following:

Entropy Type (T)	Threshold 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`.
'*FunName*'	No constraints on `P`	`FunName` is any character vector other than the previous entropy types listed. `FunName` contains the file name of your own entropy function, with `x` as input and `P` as an additional parameter to your entropy function.

`etype` and the threshold parameter `p` together define the entropy criterion. See `wentropy` for more information.

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 does the same as the `'user'` option and in addition gives the possibility to pass a parameter to your own entropy function.

p — Threshold parameter
real number | character vector | string scalar

Threshold parameter, specified by a real number, character vector, or string scalar. `p` and the entropy type `etype` together define the entropy criterion.

More About

Wavelet Packet Decomposition

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.

Tips

- To obtain the wavelet packet transform of a 1-D multisignal, use `dwpt`.

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

[1] Coifman, R.R., and M.V. Wickerhauser. “Entropy-Based Algorithms for Best Basis Selection.” *IEEE Transactions on Information Theory* 38, no. 2 (March 1992): 713–18. https://doi.org/10.1109/18.119732.

[2] Meyer, Yves. *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition, 1994. (English translation: *Wavelets: Algorithms and Applications*, SIAM).

[3] Wickerhauser, M.V. "INRIA lectures on wavelet packet algorithms." *Proceedings ondelettes et paquets d'ondes*, 17–21 June 1991, Rocquencourt, France, pp. 31–99.

[4] Wickerhauser, Mladen Victor. *Adapted Wavelet Analysis from Theory to Software*. Wellesley, MA: A.K. Peters, 1994.

See Also

`wavedec` | `waveinfo` | `wenergy` | `wprec` | `dwpt` | `idwpt`

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,wname,'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 the specified wavelet wname (see wfilters for more information).

T = wpdec2(X,N,wname) is equivalent to T = wpdec2(X,N,wname,'shannon').

E is a character vector or string scalar containing the type of entropy and P is an optional parameter depending on the value of T (see wentropy for more information).

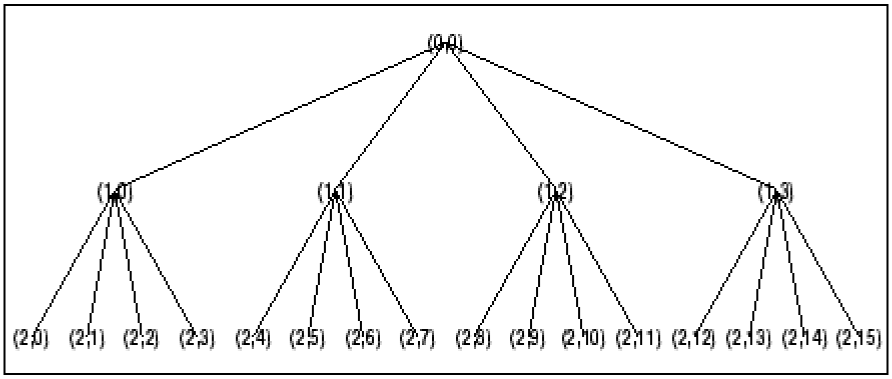
Entropy Type Name (E)	Parameter (P)	Comments
'shannon'		P is not used.
'log energy'		P is not used.
'threshold'	0 ≤ P	P is the threshold.
'sure'	0 ≤ P	P is the threshold.
'norm'	1 ≤ P	P is the power.
'user'	Character vector or string scalar	P is a character vector or string scalar containing the file name of your own entropy function, with a single input X.
FunName	No constraints on P	FunName is any other character vector or string scalar 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 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` | `dwpt` | `idwpt`

Topics

"Build Wavelet Tree Objects"
"Examples Using Wavelet Packet Tree Objects"
"Objects in the Wavelet Toolbox Software"

Introduced before R2006a

wpdencmp

Denoising or compression using wavelet packets

Syntax

```
[xd,treed,perf0,perfl2] = wpdencmp(x,sorh,n,wname,crit,par,keepapp)
[ ___ ] = wpdencmp(tree,sorh,crit,par,keepapp)
```

Description

`wpdencmp` performs a denoising or compression process of a signal or image using wavelet packets. The ideas and procedures for denoising and compression using either wavelet or wavelet packet decompositions are the same. See `wdenoise` or `wdencmp` for more information.

`[xd,treed,perf0,perfl2] = wpdencmp(x,sorh,n,wname,crit,par,keepapp)` returns a denoised or compressed version `xd` of the input data `x` obtained by wavelet packet coefficient thresholding. `wpdencmp` also returns the wavelet packet best tree decomposition `treed` of `xd` (see `besttree` for more information), and the L^2 energy recovery and compression scores in percentages as `perfl2` and `perf0`, respectively.

`[ ___ ] = wpdencmp(tree,sorh,crit,par,keepapp)` uses the wavelet packet decomposition `tree` of the data to be denoised or compressed.

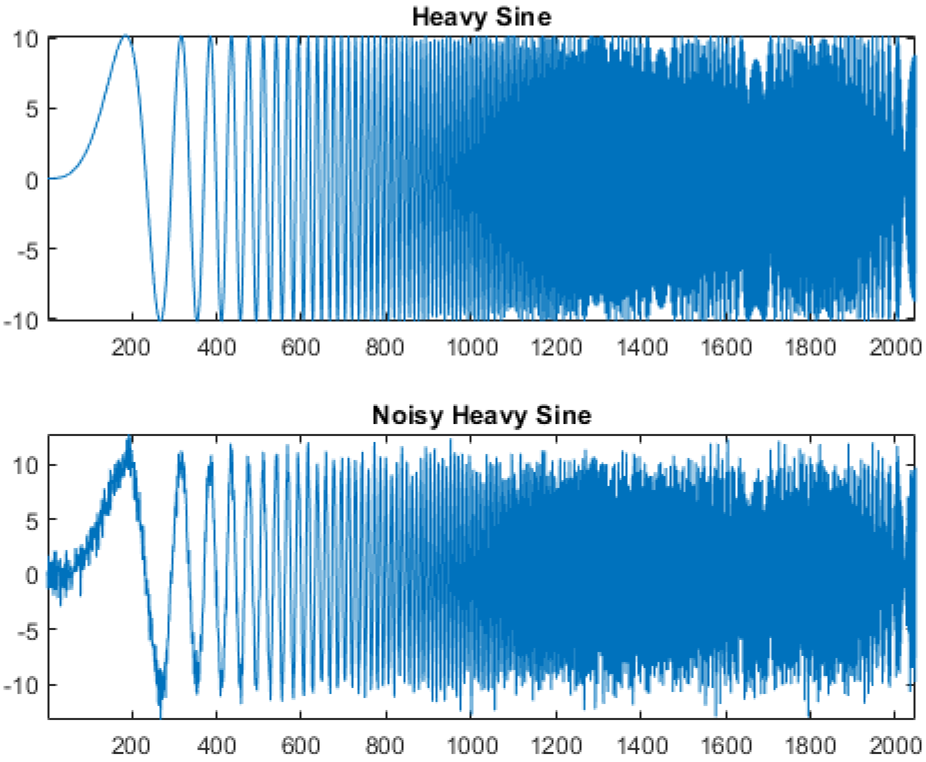
Examples

1-D Denoising Using Wavelet Packets

This example shows how to denoise using wavelet packets.

Use `wnoise` to generate the heavy sine signal and a noisy version.

```
init = 1000;
[xref,x] = wnoise(5,11,7,init);
figure
subplot(2,1,1)
plot(xref)
axis tight
title('Heavy Sine')
subplot(2,1,2)
plot(x)
axis tight
title('Noisy Heavy Sine')
```



Heavy Sine

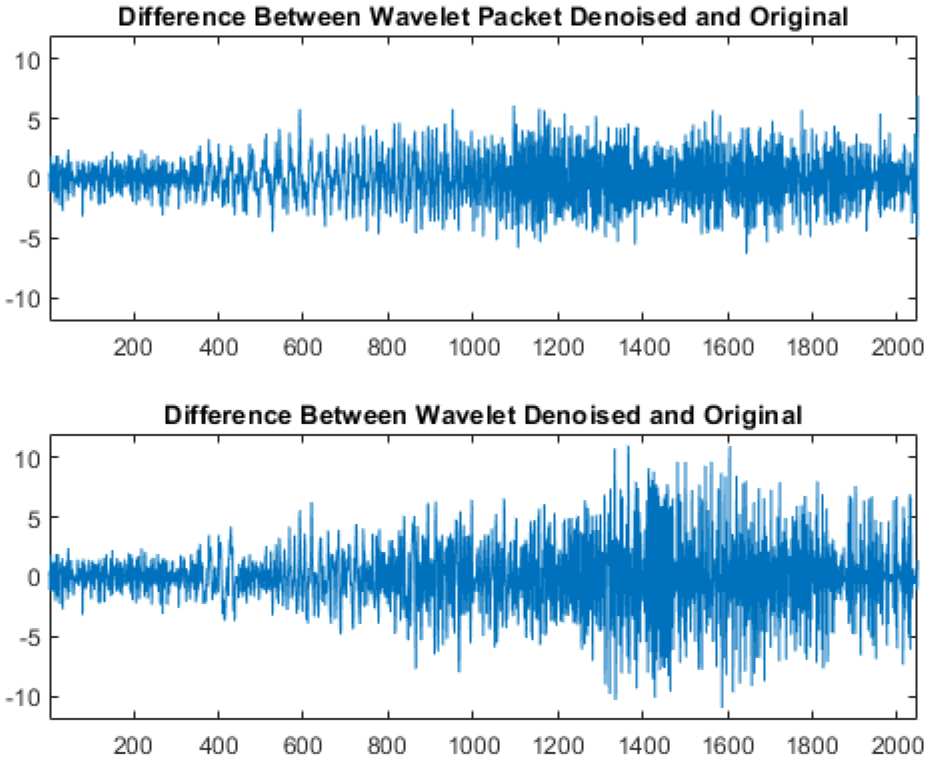
Noisy Heavy Sine

Denoise the noisy signal using a four-level wavelet packet decomposition. Use the order 4 Daubechies least asymmetric wavelet.

```
n = length(x);
thr = sqrt(2*log(n*log(n)/log(2)));
xwpd = wpdencmp(x,'s',4,'sym4','sure',thr,1);
```

Compare with a wavelet-based denoising result. Use `wdenoise` with comparable input arguments. Plot the differences between the two denoised signals and original signal.

```
xwd = wdenoise(x,4,'Wavelet','sym4','DenoisingMethod','UniversalThreshold','ThresholdRule','Hard
figure
subplot(2,1,1)
plot(x-xwpd)
axis tight
ylim([-12 12])
title('Difference Between Wavelet Packet Denoised and Original')
subplot(2,1,2)
plot(x-xwd)
axis tight
ylim([-12 12])
title('Difference Between Wavelet Denoised and Original')
```

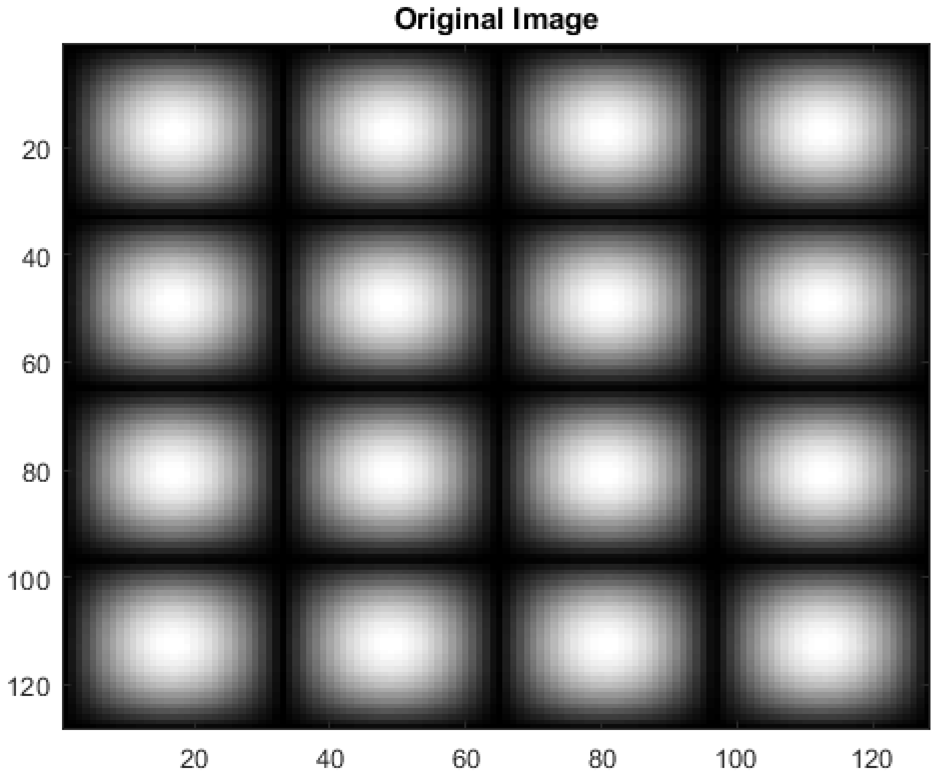


2-D Denoising Using Wavelet Packets

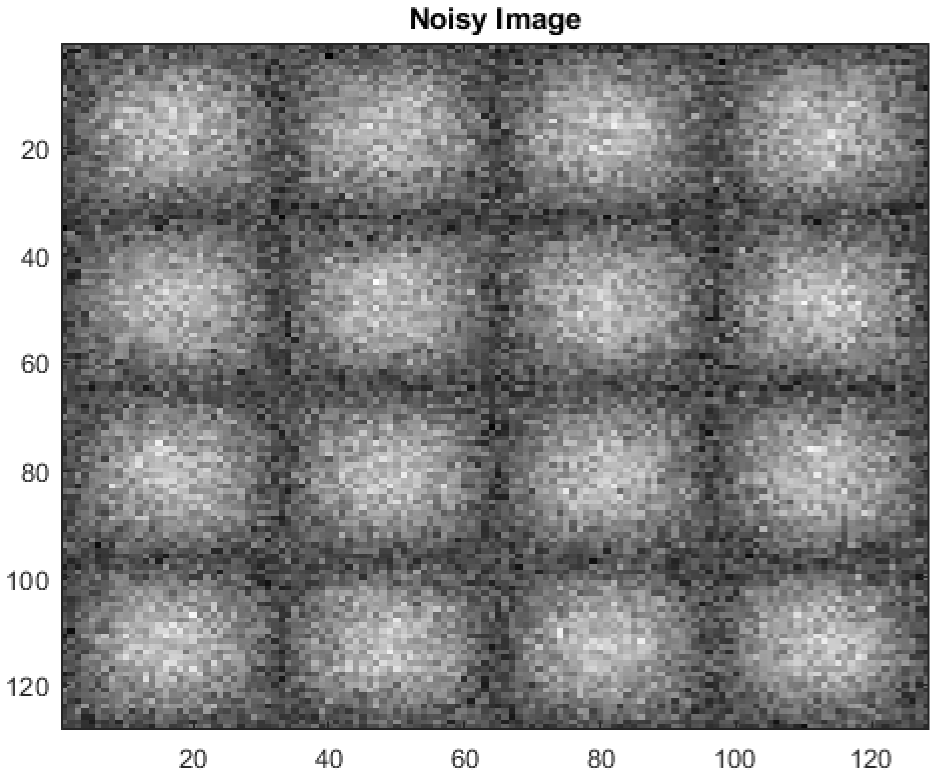
This example shows how to denoise an image using wavelet packets.

Load an image and generate a noisy copy. For reproducibility set the random seed.

```
rng default
load sinsin
x = X/18 + randn(size(X));
imagesc(X)
colormap(gray)
title('Original Image')
```

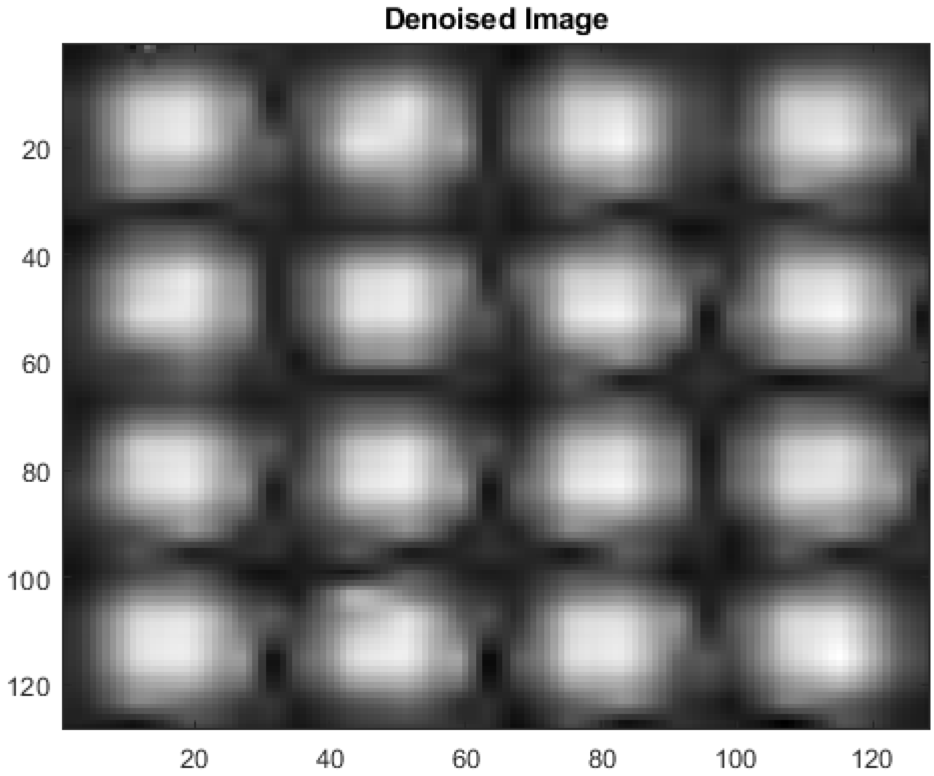


```
figure
imagesc(x)
colormap(gray)
title('Noisy Image')
```



Denoise the noisy image using wavelet packet decomposition. Use ddencmp to determine denoising parameters. Do a three-level decomposition with the order 4 Daubechies least asymmetric wavelet.

```
[thr,sorh,keepapp,crit] = ddencmp('den','wp',x);
xd = wpdencmp(x,sorh,3,'sym4',crit,thr,keepapp);
figure
imagesc(xd)
colormap(gray)
title('Denoised Image')
```



1-D Compression Using Wavelet Packets

This example shows how to compress a 1-D signal using wavelet packets.

Load a signal. Use `ddencmp` to determine compression values for that signal.

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

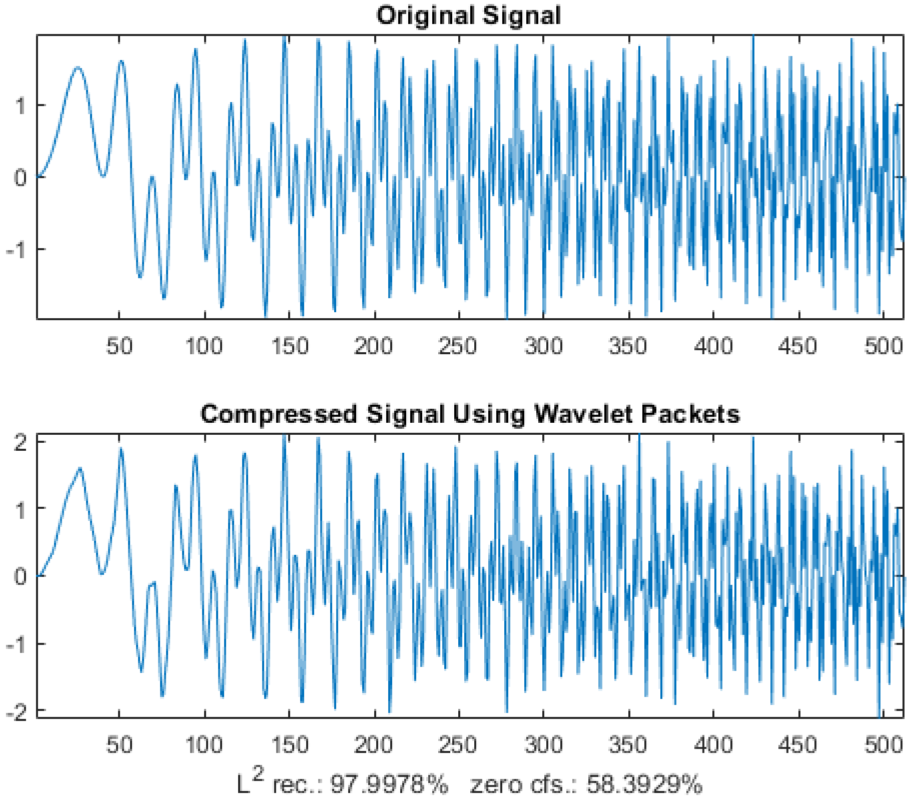
```
thr = 0.5193

sorh = 
'h'

keepapp = 1

crit = 
'threshold'
```

Compress the signal using global thresholding with threshold best basis. Use the order 4 Daubechies least asymmetric wavelet and do a three-level wavelet packet decomposition.

```
[xc,wpt,perf0,perfl2] = wpdencmp(x,sorh,3,'sym4',crit,thr,keepapp);
```

Compare the original signal with the compressed version.

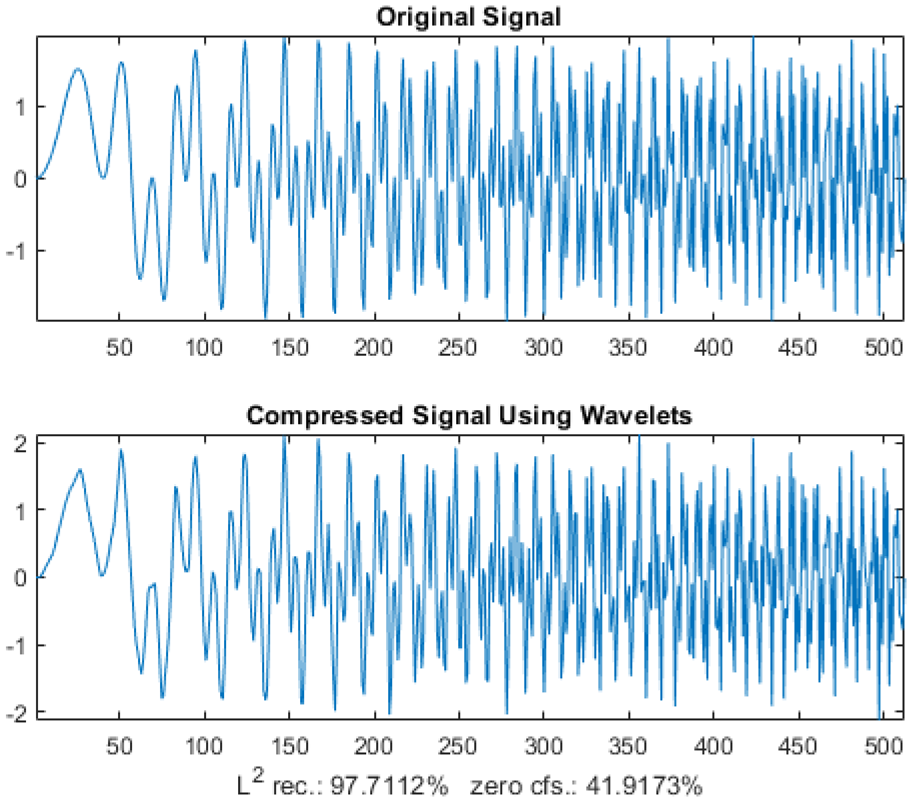
```
subplot(2,1,1)
plot(x)
title('Original Signal')
axis tight
subplot(2,1,2)
plot(xc)
xlabel(['L^2 rec.: ',num2str(perfl2),'%   zero cfs.: ',num2str(perf0),'%'])
title('Compressed Signal Using Wavelet Packets')
axis tight
```


Original Signal

Compressed Signal Using Wavelet Packets

Compress the signal again, but this do a three-level wavelet decomposition. Keep all the other parameters the same.

```
[thr,sorh,keepapp] = ddencmp('cmp','wv',x);
[xcwv,~,~,perf0wv,perfl2wv] = wdencmp('gbl',x,'sym4',3,thr,sorh,keepapp);
figure
subplot(2,1,1)
plot(x)
title('Original Signal')
axis tight
subplot(2,1,2)
plot(xc)
xlabel(['L^2 rec.: ',num2str(perfl2wv),'%   zero cfs.: ',num2str(perf0wv),'%'])
title('Compressed Signal Using Wavelets')
axis tight
```



A larger fraction of coefficients are set equal to 0 when compressing using a wavelet packet decomposition.

Input Arguments

x — Input data

real-valued vector or matrix

Input data to denoise or compress, specified by a real-valued vector or matrix.

Data Types: `double`

tree — Wavelet packet decomposition

wavelet packet decomposition

Wavelet packet decomposition of the data to be denoised or compressed, specified as a wavelet packet tree. See `wpdec` and `wpdec2` for more information.

sorh — Type of thresholding

`'s'` | `'h'`

Type of thresholding to perform:

- `'s'` — Soft thresholding

- 'h' — Hard thresholding

See wthresh for more information.

n — Wavelet packet decomposition level
positive integer

Wavelet packet decomposition level, specified as a positive integer.

wname — Name of wavelet
character vector | string scalar

Name of wavelet, specified as a character vector or string scalar, to use for denoising. See wavemngr for more information.

crit — Entropy type
'shannon' | 'log energy' | 'threshold' | 'sure' | 'norm' | 'user' | ...

Entropy type, specified as one of the following:

Entropy Type (crit)	Threshold Parameter (par)	Comments
'shannon'		par is not used.
'log energy'		par is not used.
'threshold'	0 ≤ par	par is the threshold.
'sure'	0 ≤ par	par is the threshold.
'norm'	1 ≤ par	par is the power.
'user'	Character vector	par is a character vector containing the file name of your own entropy function, with a single input x.
'*FunName*'	No constraints on par	FunName is any character vector other than the previous entropy types listed. FunName contains the file name of your own entropy function, with x as input and par as an additional parameter to your entropy function.

crit and threshold parameter par together define the entropy criterion used to determine the best decomposition. See wentropy for more information.

If crit = 'nobest', no optimization is done, and the current decomposition is thresholded.

par — Threshold parameter
real number | character vector | string scalar

Threshold parameter, specified by a real number, character vector, or string scalar. par and the entropy type crit together define the entropy criterion used to determine the best decomposition. See wentropy for more information.

Data Types: double

keepapp — Threshold approximation setting
0 | 1

Threshold approximation setting, specified as either 0 or 1. If keepapp = 1, the approximation coefficients cannot be thresholded. If keepapp = 0, the approximation coefficients can be thresholded.

Data Types: double

Output Arguments

xd — Denoised or compressed data

real-valued vector or matrix

Denoised or compressed data, returned as a real-valued vector or matrix. xd and x have the same dimensions.

treed — Wavelet packet best tree decomposition

wavelet packet tree

Wavelet packet best tree decomposition of xd, returned as a wavelet packet tree.

perf0 — Compression score

real number

Compression score, returned as a real number. perf0 is the percentage of thresholded coefficients that are equal to 0.

perfl2 — L^2 energy recovery

real number

L^2 energy recovery, returned as a real number. perfl2 is equal to $100 \times \left(\frac{\text{vector-norm of wavelet packet coefficients of } xd}{\text{vector-norm of wavelet packet coefficients of } x}\right)^2$. If x is a one-dimensional signal and wname an orthogonal wavelet, perfl2 simplifies to $\frac{100\|xd\|^2}{\|x\|^2}$.

References

[1] Antoniadis, A., and G. Oppenheim, eds. *Wavelets and Statistics*. Lecture Notes in Statistics. New York: Springer Verlag, 1995.

[2] Coifman, R. R., and M. V. Wickerhauser. “Entropy-Based Algorithms for Best Basis Selection.” *IEEE Transactions on Information Theory*. Vol. 38, Number 2, 1992, pp. 713–718.

[3] DeVore, R. A., B. Jawerth, and B. J. Lucier. “Image Compression Through Wavelet Transform Coding.” *IEEE Transactions on Information Theory*. Vol. 38, Number 2, 1992, pp. 719–746.

[4] Donoho, D. L. “Progress in Wavelet Analysis and WVD: A Ten Minute Tour.” *Progress in Wavelet Analysis and Applications* (Y. Meyer, and S. Roques, eds.). Gif-sur-Yvette: Editions Frontières, 1993.

[5] Donoho, D. L., and I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage.” *Biometrika*. Vol. 81, 1994, pp. 425–455.

[6] Donoho, D. L., I. M. Johnstone, G. Kerkyacharian, and D. Picard. “Wavelet Shrinkage: Asymptopia?” *Journal of the Royal Statistical Society, series B*. Vol. 57, Number 2, 1995, pp. 301–369.

See Also

Functions

`wdenoise` | `besttree` | `ddencmp` | `wdencmp` | `wenergy` | `wpbmpen` | `wpdec` | `wpdec2` | `wthresh` | `wden` | `wentropy`

Apps

Wavelet Signal Denoiser

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

$$(W_{2n+1}(x) = W_n(2x) - W_n(2x-1))$$

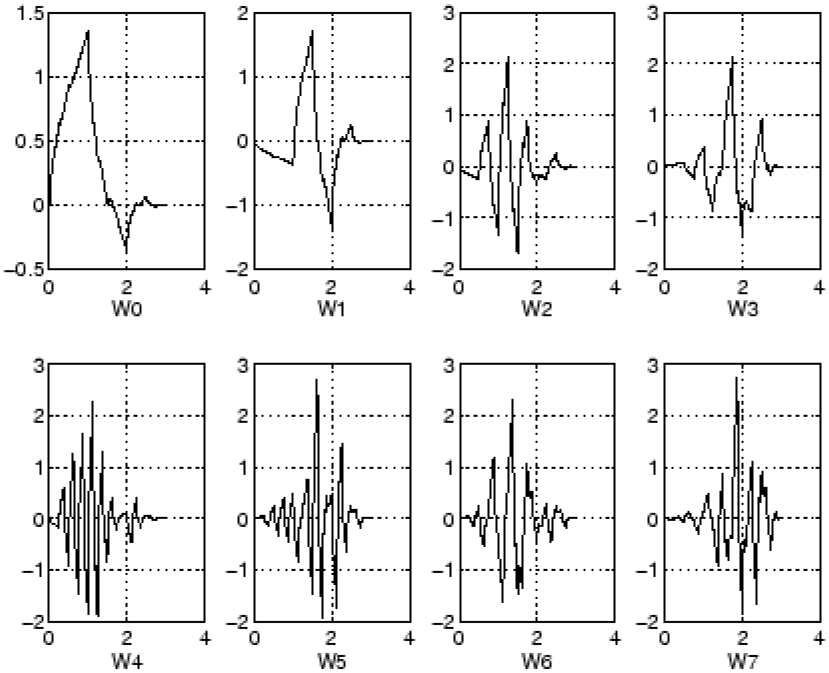
$W_0(x) = \phi(x)$ is the haar scaling function and $W_1(x) = \psi(x)$ is the haar wavelet, both supported in [0,1].

Then we can obtain $W_{2\ n}$ by adding two 1/2-scaled versions of W_n with distinct supports [0,1/2] and [1/2,1], and obtain W_{2n+1} by subtracting the same versions of W_n.

Starting from more regular original wavelets, using a similar construction, we obtain smoothed versions of this system of *W*-functions, all with support in the interval [0, 2*N*-1].

Examples

```
% Compute the db2 Wn functions for n = 0 to 7, generating
% the db2 wavelet packets.
[wp,x] = wpfun('db2',7);

% Using some plotting commands,
% the following figure is generated.
```



References

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

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and applications*, SIAM).

Wickerhauser, M.V. (1991), “INRIA lectures on wavelet packet algorithms,” *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

See Also

`wavefun` | `waveinfo`

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`.

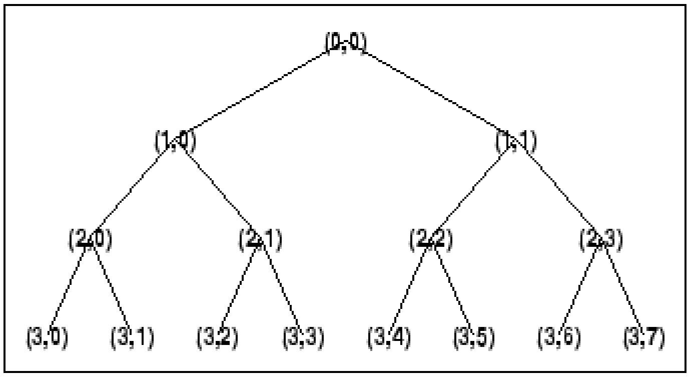
`[T,X] = wpjoin(T,N)` also returns the coefficients of the node.

`T = wpjoin(T)` is equivalent to `T = wpjoin(T,0)`.

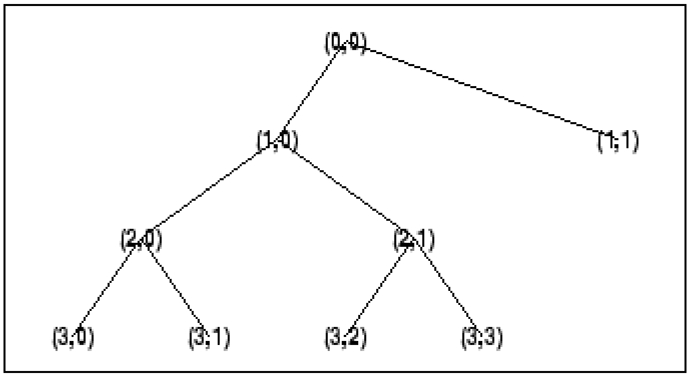
`[T,X] = wpjoin(T)` is equivalent to `[T,X] = wpjoin(T,0)`.

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)
X = wprcoef(T,N)
```

Description

`X = wprcoef(T)` reconstructs coefficients of the node `0` of the wavelet packet tree `T`.

`wprcoef` is a one- or two-dimensional wavelet packet analysis function.

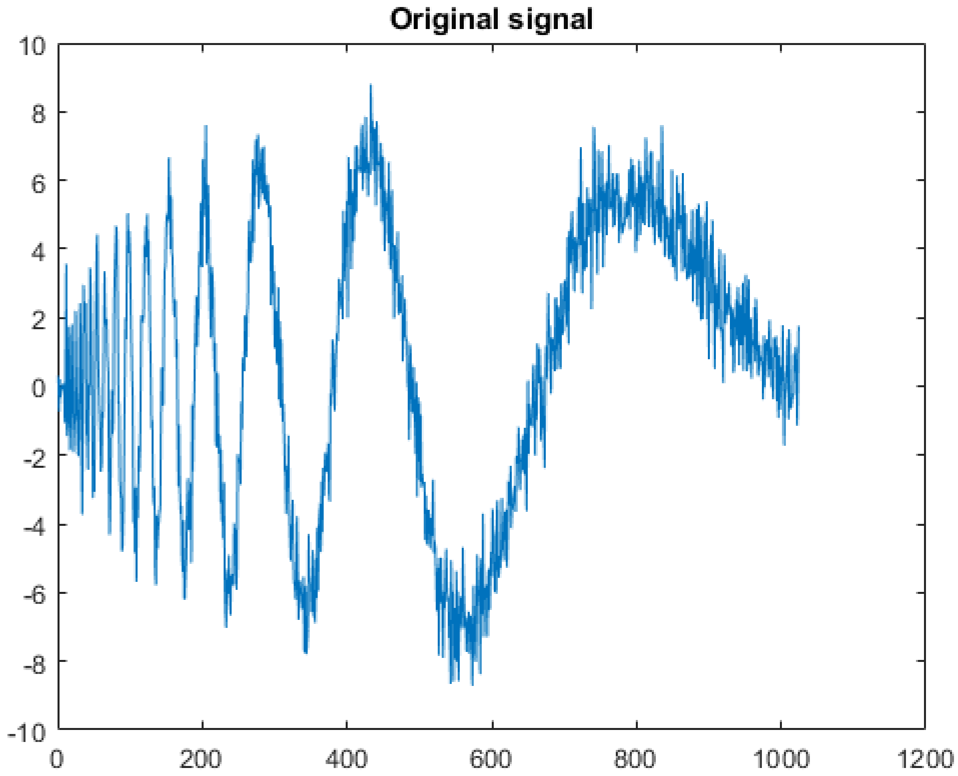
`X = wprcoef(T,N)` reconstructs coefficients of the node `N` of the wavelet packet tree `T`.

Examples

Reconstruct Wavelet Packet Coefficients

Load and plot original signal. The function uses zero padding as an extension mode for dealing with the problem of border distortion in signal or image analysis. For more information, see `dwtmode`.

```
load noisdopp;
s = noisdopp;
plot(s);
title('Original signal');
```

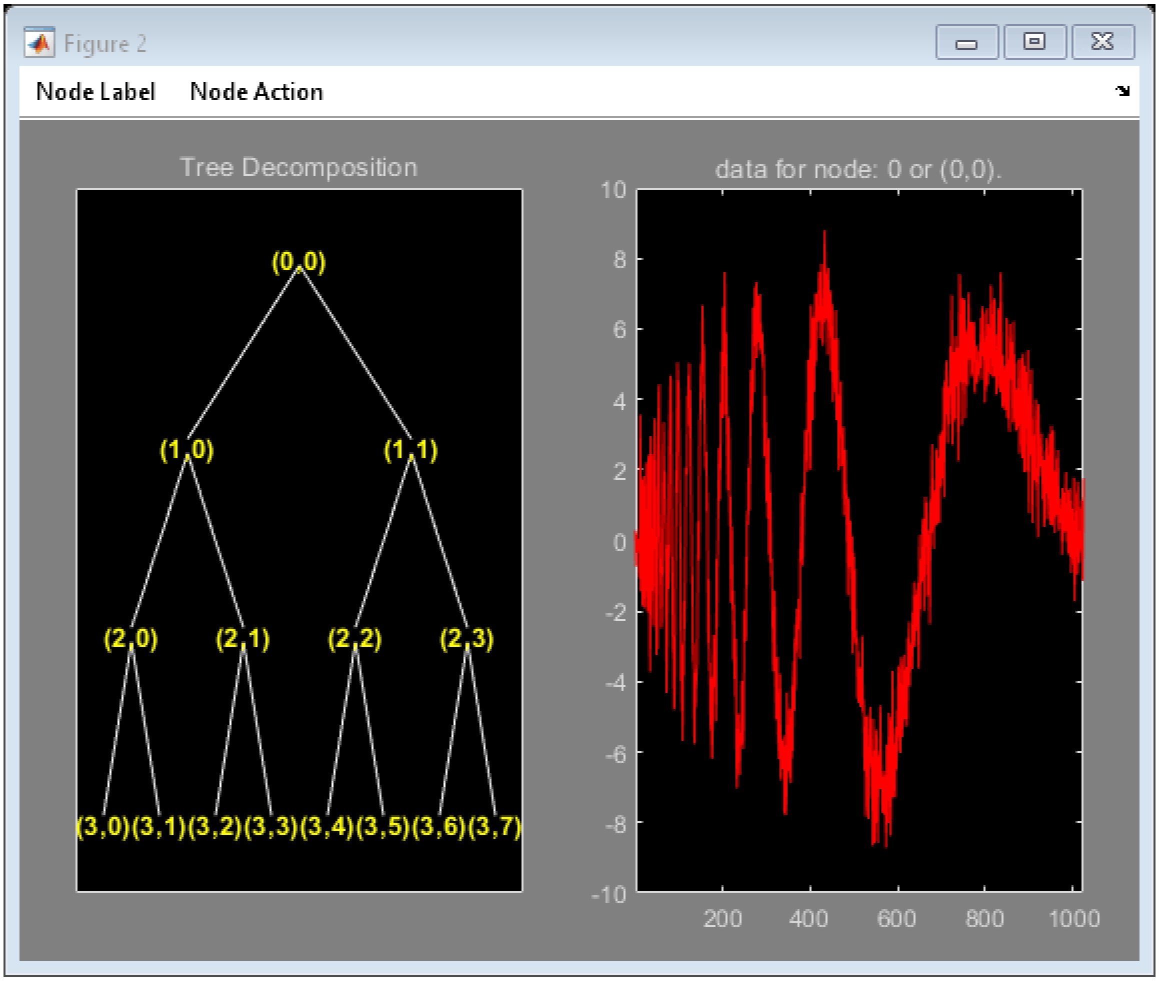


Decompose the original signals at depth 3 with `db1` wavelet packets using Shannon entropy.

```
T = wpdec(s,3,'db1','shannon');
```

Plot the wavelet packet tree.

```
plot(T)
```

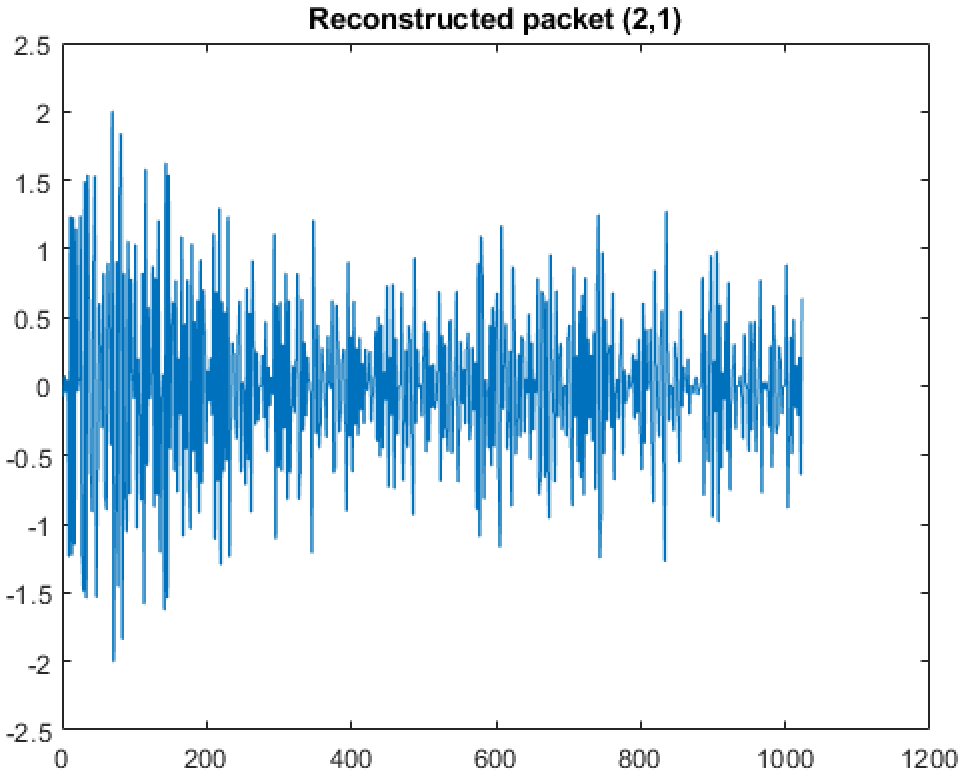


Reconstruct the packet at node (2,1).

```
X = wprcoef(T,[2 1]);
```

Plot the reconstructed packet.

```
plot(X);
title('Reconstructed packet (2,1)');
```



Input Arguments

T — Wavelet packet tree
wptree object

Wavelet packet tree, specified as a wptree object.

N — Node in wavelet packet tree
0 (default) | nonnegative integer | 1-by-2 vector

Node in the wavelet packet tree T, specified as a nonnegative integer or as a pair of nonnegative integers.

Output Arguments

X — Reconstructed coefficients
row vector

Reconstructed coefficients of the wavelet packet, returned as a row vector.

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(tobj)
```

Description

`x = wprec(tobj)` returns the reconstructed vector `x` corresponding to the wavelet packet tree object `tobj`.

Examples

Reconstruct Signal from Wavelet Packet Tree Object

Load a signal.

```
load noisdopp
x = noisdopp;
```

Decompose the signal at level 3 with `sym4` wavelet packets using log energy entropy.

```
wpt = wpdec(x,3,'sym4','log energy');
```

Reconstruct the signal from the wavelet packet tree object.

```
xrec = wprec(wpt);
```

Compare the original signal with the reconstruction.

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

```
ans = 8.2778e-12
```

Input Arguments

`tobj` — Wavelet packet tree
wavelet packet tree object

Wavelet packet tree, specified as a wavelet packet tree object. The `wprec` function assumes that you obtained `tobj` using `wpdec`.

See Also

`wpdec` | `wpdec2` | `wpjoin` | `wprec2` | `wpsplt` | `dwpt` | `idwpt`

Topics
"Build Wavelet Tree Objects"

“Examples Using Wavelet Packet Tree Objects”

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` | `dwpt` | `idwpt`

Introduced before R2006a

wpspectrum

Wavelet packet spectrum

Syntax

```
[spec,times,freq] = wpspectrum(wpt,fs)
[___] = wpspectrum(wpt,fs,'plot')
[___,tinfo] = 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. `times` is a vector of times and `freq` is a vector of frequencies.

`[___] = wpspectrum(wpt,fs,'plot')` displays the wavelet packet spectrum.

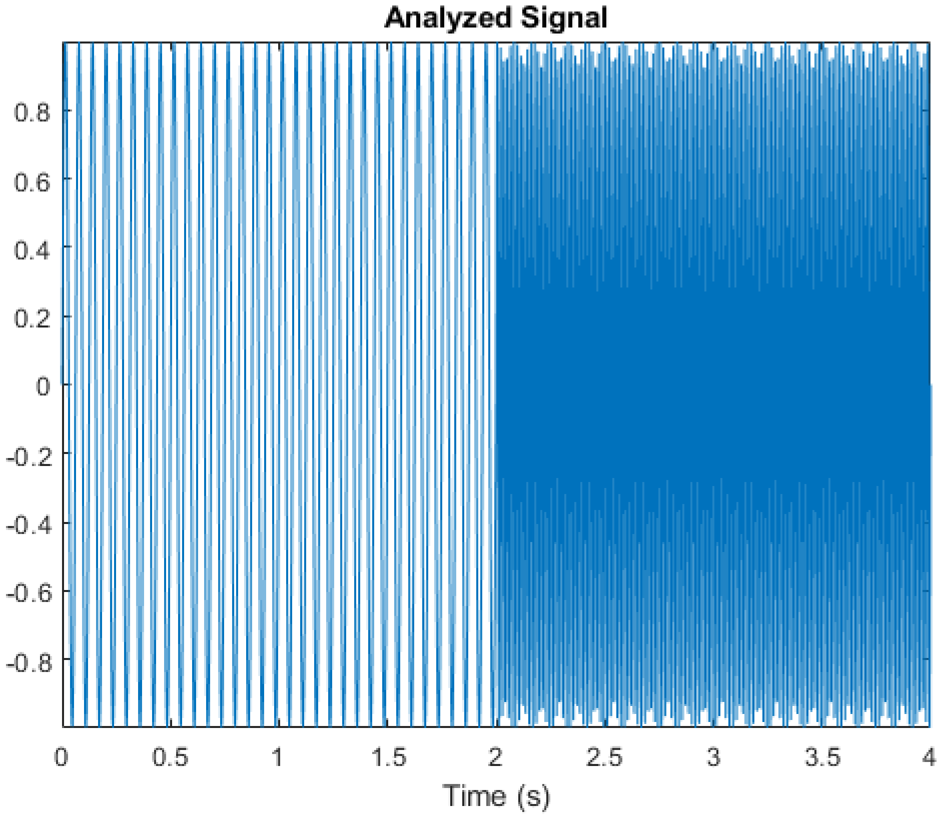
`[___,tinfo] = wpspectrum(___)` returns the terminal nodes of the wavelet packet tree in frequency order.

Examples

Wavelet Packet Spectrum for Sinusoids

Create a signal consisting of two sinusoids with disjoint support. The sinusoids have frequencies of 16 Hz and 64 Hz. Sample the signal at 500 Hz for 4 seconds.

```
fs = 500;
frA = 16;
frB = 64;
t = 0:1/fs:4;
sig = sin(frA*2*pi*t).*(t<2) + sin(frB*2*pi*t).*(t>=2);
plot(t,sig)
axis tight
title('Analyzed Signal')
xlabel('Time (s)')
```

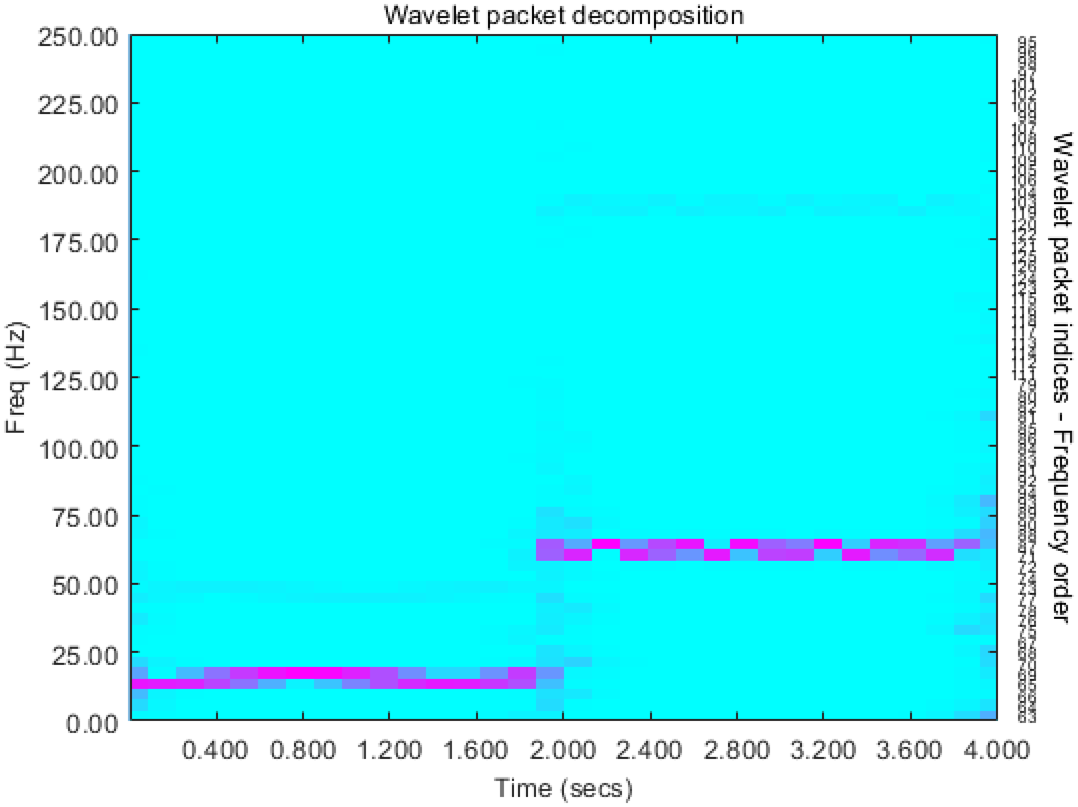


Obtain the wavelet packet tree object corresponding to the level 6 wavelet packet decomposition of the signal using the `sym6` wavelet.

```
level = 6;
wname = 'sym6';
wpt = wpdec(sig,level,wname);
```

Obtain and plot the wavelet packet spectrum.

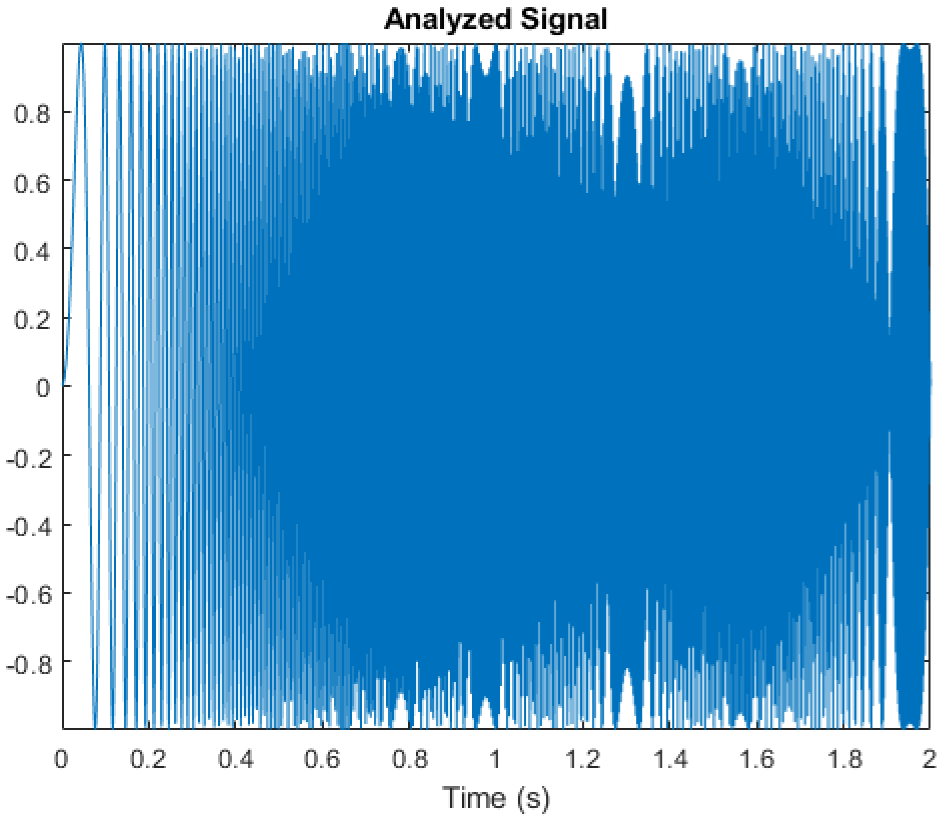
```
[S,T,F] = wpspectrum(wpt,fs,'plot');
```



Wavelet Packet Spectrum of Chirp Signal

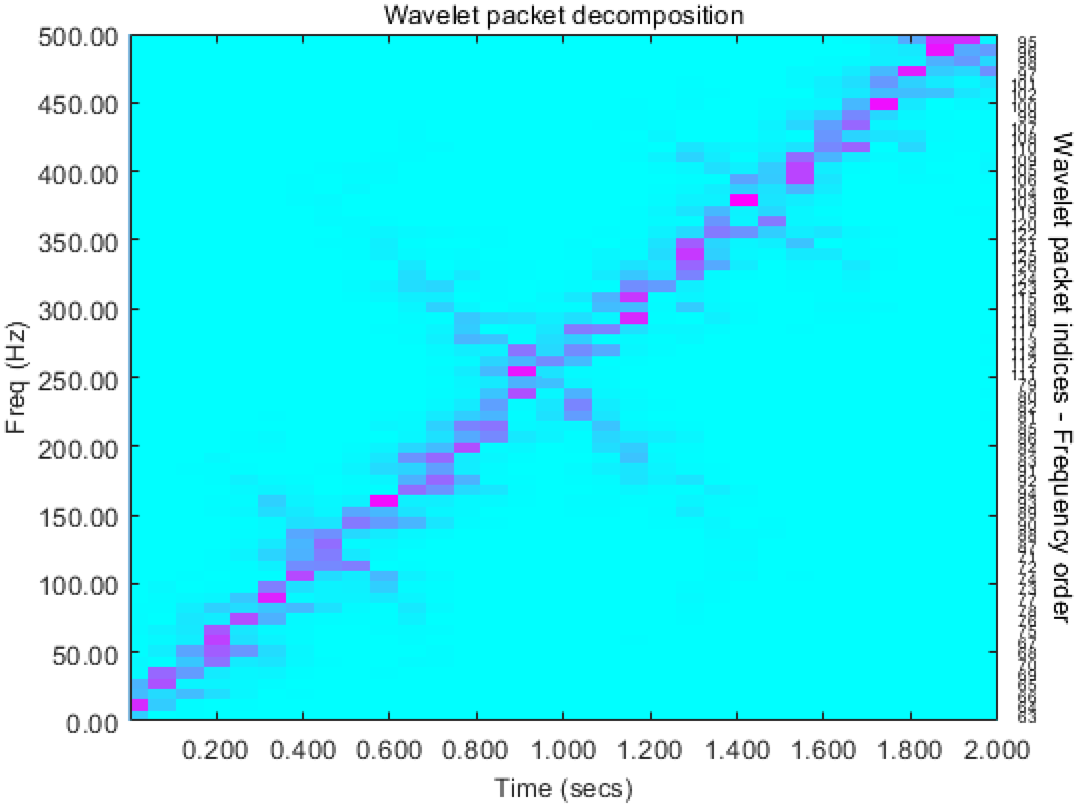
Generate a chirp signal sampled at 1000 Hz for 2 seconds.

```
fs = 1000;
t = 0:1/fs:2;
sig = sin(256*pi*t.^2);
plot(t,sig)
axis tight
title('Analyzed Signal')
xlabel('Time (s)')
```



Obtain the wavelet packet tree object corresponding to the level 6 wavelet packet decomposition of the signal using the `sym8` wavelet. Plot the wavelet packet spectrum.

```
level = 6;
wpt = wpdec(sig,level,'sym8');
[S,T,F] = wpspectrum(wpt,fs,'plot');
```



Input Arguments

`wpt` — Binary wavelet packet tree

`wptree` object

Binary wavelet packet tree, specified as a wavelet packet tree object.

`fs` — Sampling frequency

1 (default) | positive scalar

Sampling frequency in hertz, specified as a positive scalar.

Data Types: `double`

Output Arguments

`spec` — Wavelet packet spectrum estimates

matrix

Wavelet packet spectrum estimates, returned as a matrix. `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. N is equal to the length of node 0 in the wavelet packet tree object.

The frequency spacing between the rows of `spec` is `fs`/2^{J+1}.

Data Types: `double`

`times` — Times
vector

Times, returned as a 1-by-*N* vector, where *N* is the length of the time series. The time spacing between elements is 1/`fs`.

Data Types: `double`

`freq` — Frequencies
vector

Frequencies, returned as a 1-by-2^J vector, where *J* is the level of the wavelet packet transform. The frequency spacing in `freq` is `fs`/2^{J+1}.

Data Types: `double`

`tinfo` — Terminal nodes
vector

Terminal nodes of the wavelet packet tree object in frequency order.

Data Types: `double`

More About

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

[1] Wickerhauser, M.V. *Lectures on Wavelet Packet Algorithms*, Technical Report, Washington University, Department of Mathematics, 1992.

See Also

`otnodes` | `wpdec` | `dwpt` | `modwpt`

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,

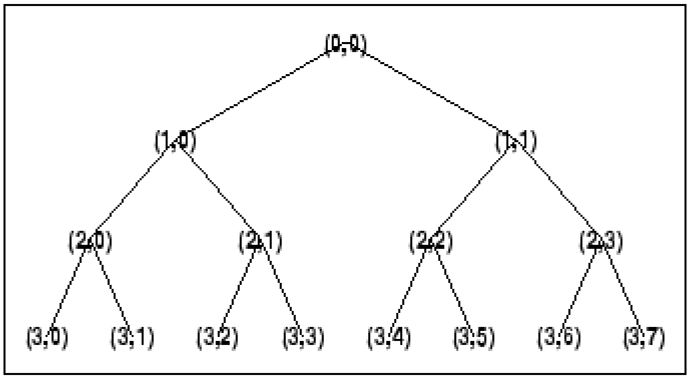
`[T,cA,cD] = wpsplt(T,N)` with `cA` = approximation and `cD` = detail of node `N`.

For a two-dimensional decomposition,

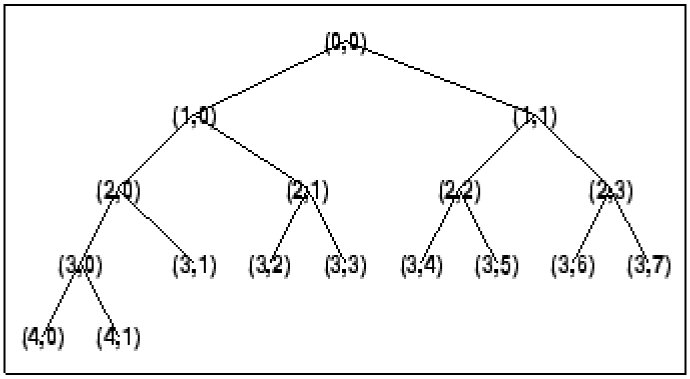
`[T,cA,cH,cV,cD] = wpsplt(T,N)` with `cA` = approximation and `cH,cV,c` = horizontal, vertical, and diagonal details of node `N`.

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 or string scalar 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

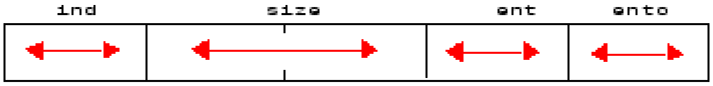
Fields from the DTREE parent object:

'allNI'	All nodes information

'allNI' is an array of size nbnode by 5, which contains

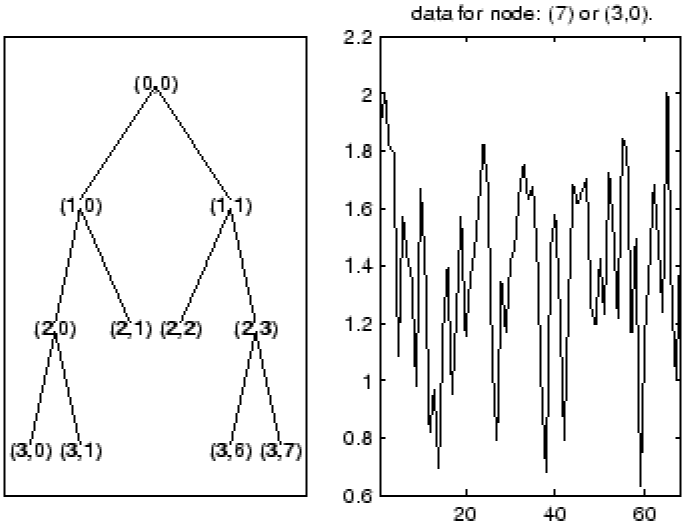
ind	Index
size	Size of data
ent	Entropy
ento	Optimal entropy

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*.

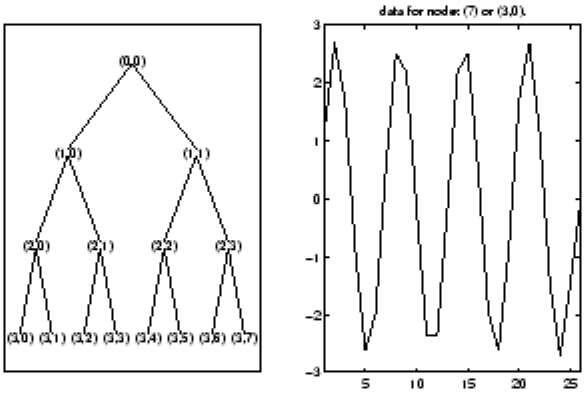
T is a wavelet packet tree and `CMODE` is an integer, which represents the color mode. The color modes are listed in the table below.

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

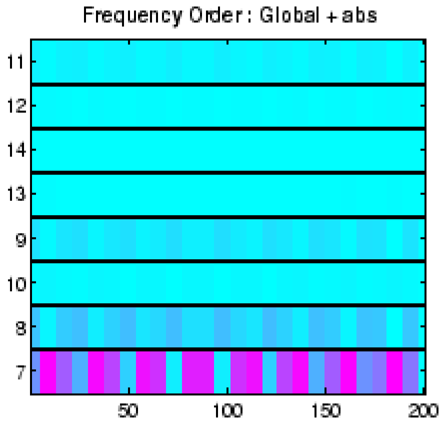
`wpviewcf(T,CMODE,NBCOL)` uses `NBCOL` colors.

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)
x = wrcoef(type,c,l,LoR,HiR)
x = wrcoef(___,n)
```

Description

`x = wrcoef(type,c,l,wname)` reconstructs the coefficients vector of type `type` based on the wavelet decomposition structure `[c,l]` of a 1-D signal (see `wavedec` for more information) using the wavelet specified by `wname`. The coefficients at the maximum decomposition level are reconstructed. The length of `x` is equal to the length of the original 1-D signal.

`x = wrcoef(type,c,l,LoR,HiR)` uses the reconstruction filters `LoR` and `HiR`.

`x = wrcoef(___,n)` reconstructs the coefficients at level `n` using any of the previous syntaxes.

Examples

Reconstruct Wavelet Coefficients

Load a 1-D signal.

```
load sumsin
s = sumsin;
```

Perform a level 5 wavelet decomposition of the signal using the `sym4` wavelet.

```
[c,l] = wavedec(s,5,'sym4');
```

Reconstruct the approximation coefficients at level 5 from the wavelet decomposition structure `[c,l]`.

```
a5 = wrcoef('a',c,l,'sym4');
```

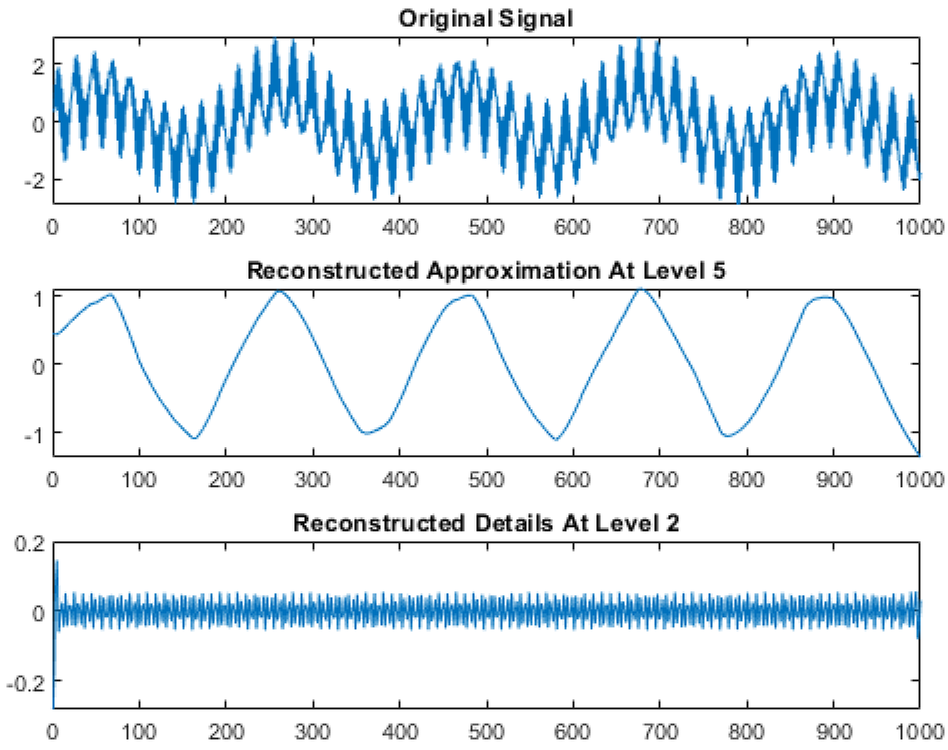
Reconstruct the detail coefficients at level 2.

```
d2 = wrcoef('d',c,l,'sym4',2);
```

Plot the original signal and reconstructed coefficients.

```
subplot(3,1,1)
plot(s)
title('Original Signal')
subplot(3,1,2)
plot(a5)
title('Reconstructed Approximation At Level 5')
subplot(3,1,3)
```

```
plot(d2)
title('Reconstructed Details At Level 2')
```



Input Arguments

`type` — Coefficients to reconstruct

`'a'` | `'d'`

Coefficients to reconstruct, specified as `'a'` or `'d'`, for approximation or detail coefficients, respectively.

`c` — Wavelet decomposition

real-valued vector

Wavelet decomposition of a 1-D signal, specified as a real-valued vector. The vector contains the wavelet coefficients. The bookkeeping vector `l` contains the coefficients by level. See `wavedec`.

Data Types: `double` | `single`

`l` — Bookkeeping vector

vector of positive integers

Bookkeeping vector, specified as a vector of positive integers. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition `c` by level. See `wavedec`.

Data Types: `double` | `single`

wname — Analyzing wavelet
character vector | string scalar

Analyzing wavelet used to create the wavelet decomposition structure `[c,l]`, specified as a character vector or string scalar. `wrcoef` supports only orthogonal or biorthogonal wavelets. See `wfilters`.

LoR,HiR — Wavelet reconstruction filters
even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. `LoR` is the lowpass reconstruction filter, and `HiR` is the highpass reconstruction filter. The lengths of `LoR` and `HiR` must be equal. See `wfilters` for additional information.

n — Coefficients level
`length(l)-2` | nonnegative integer

Coefficients level, specified as a nonnegative integer. When `type` is `'a'`, `n` is allowed to be 0. Otherwise, `n` is a strictly positive integer such that `n ≤ length(l)-2`. The default value of `n` is `length(l)-2`.

See Also

`appcoef` | `detcoef` | `wavedec`

Introduced before R2006a

wrcoef2

Reconstruct single branch from 2-D wavelet coefficients

Syntax

```
x = wrcoef2(type,c,s,wname)
x = wrcoef2(type,c,s,LoR,HiR)
x = wrcoef2( ___ ,n)
```

Description

wrcoef2 is a two-dimensional wavelet analysis function. wrcoef2 reconstructs the coefficients of an image.

x = wrcoef2(type,c,s,wname) returns the matrix of reconstructed coefficients of type type based on the wavelet decomposition structure [c,s] of an image (see wavedec2 for more information) using the wavelet specified by wname. The coefficients at the maximum decomposition level are reconstructed. The size of x is equal to the size of the original image.

x = wrcoef2(type,c,s,LoR,HiR) uses the lowpass and highpass reconstruction filters LoR and HiR, respectively.

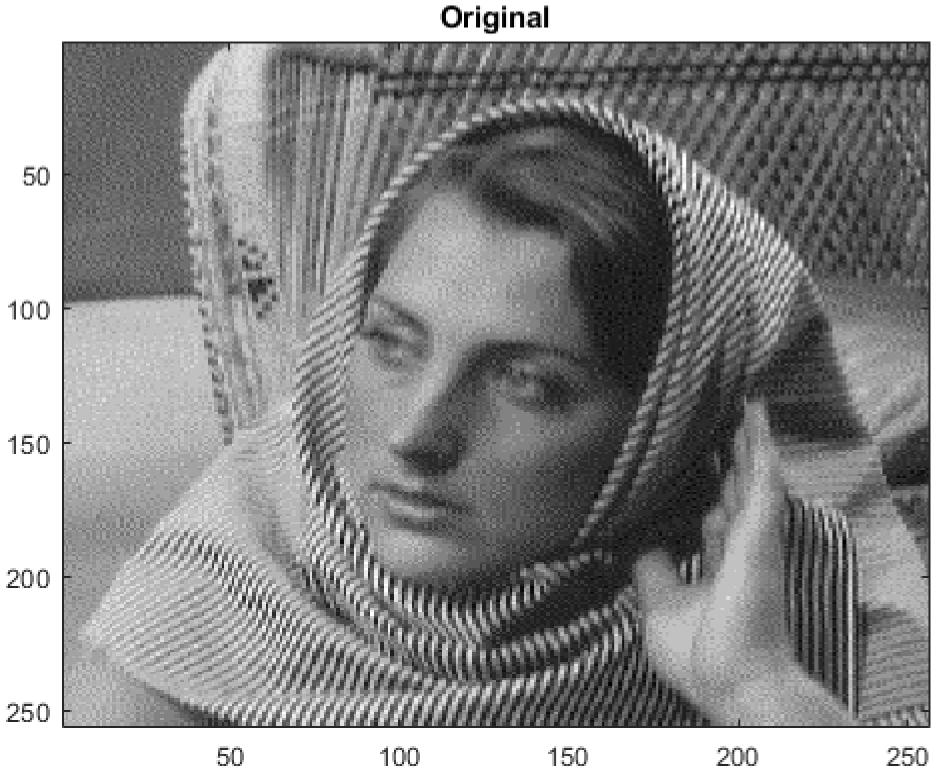
x = wrcoef2(___ ,n) reconstructs the coefficients at level n using any of the previous syntaxes.

Examples

Reconstruct 2-D Wavelet Coefficients

Save the current extension mode. Load an image.

```
origMode = dwtmode("status","nodisp");
load woman
imagesc(X)
title("Original")
colormap gray
```

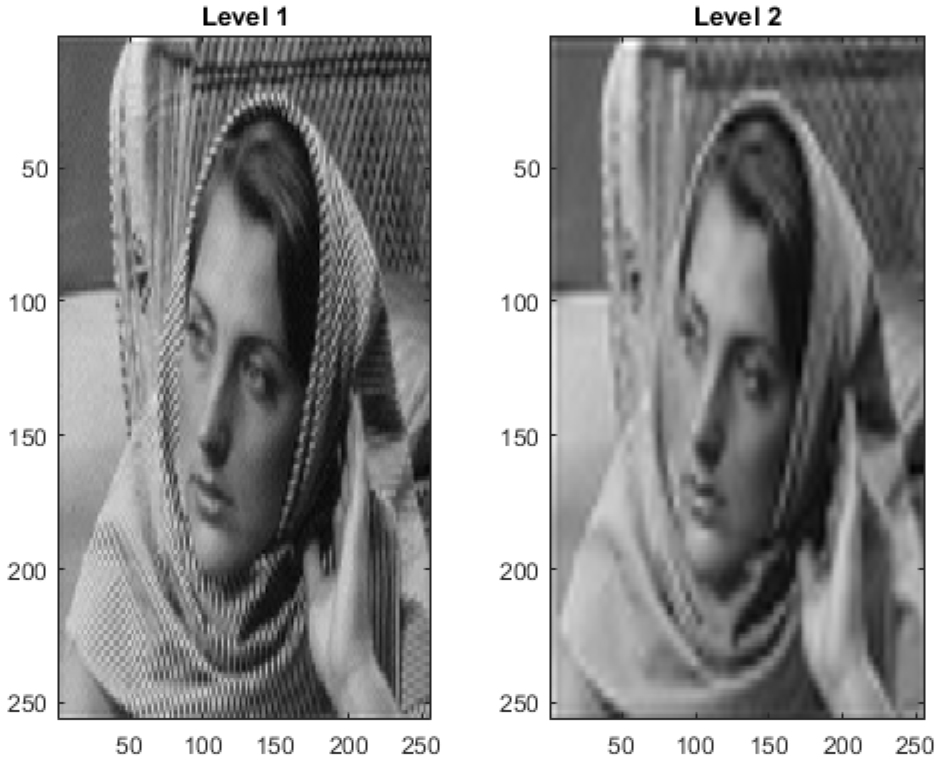


Use `dwtmode` to change the extension mode to zero-padding. Obtain the 2-level wavelet decomposition of the image using the `sym5` wavelet.

```
dwtmode("zpd","nodisp")
[c,s] = wavedec2(X,2,"sym5");
```

Reconstruct the approximation coefficients at levels 1 and 2. Display the results.

```
a1 = wrcoef2("a",c,s,"sym5",1);
a2 = wrcoef2("a",c,s,"sym5",2);
subplot(1,2,1)
imagesc(a1)
title("Level 1")
subplot(1,2,2)
imagesc(a2)
title("Level 2")
colormap gray
```



Reconstruct the horizontal, vertical, and diagonal detail coefficients at level 2.

```
h2 = wrcoef2("h",c,s,"sym5",2);
v2 = wrcoef2("v",c,s,"sym5",2);
d2 = wrcoef2("d",c,s,"sym5",2);
```

Confirm all the reconstructions are the same size as the original image.

```
sX = size(X);
sa1 = size(a1);
sa2 = size(a2);
sh2 = size(h2);
sv2 = size(v2);
sd2 = size(d2);
[sX;sa1;sa2;sh2;sv2;sd2]
```

```
ans = 6×2

   256   256
   256   256
   256   256
   256   256
   256   256
   256   256
```

Restore the extension mode to the original setting.

```
dwtmode(origMode,"nodisp")
```

Input Arguments

type — Coefficients to reconstruct

"a" | "h" | "v" | "d"

Coefficients to reconstruct, specified as follows:

- "a" — Approximation coefficients
- "h" — Horizontal detail coefficients
- "v" — Vertical detail coefficients
- "d" — Diagonal detail coefficients

Data Types: string | char

c — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector, specified as a real-valued vector. The vector c contains the approximation and detail coefficients organized by level. The bookkeeping matrix s is used to parse c. See wavedec2.

Data Types: double

s — Bookkeeping matrix

integer-valued matrix

Bookkeeping matrix, specified as an integer-valued matrix. The matrix s contains the dimensions of the wavelet coefficients by level and is used to parse the wavelet decomposition vector c. See wavedec2.

Data Types: double

wname — Wavelet

character vector | string scalar

Wavelet, specified as a character vector or string scalar. wrcoef2 supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets. See wfilters for a list of orthogonal and biorthogonal wavelets.

LoR,HiR — Wavelet reconstruction filters

even-length real-valued vectors

Wavelet reconstruction filters, specified as a pair of even-length real-valued vectors. LoR is the lowpass reconstruction filter, and HiR is the highpass reconstruction filter. The lengths of LoR and HiR must be equal. See wfilters for additional information.

Data Types: double

n — Coefficients level

size(s,1)-2 (default) | integer

Coefficients level, specified as an integer.

- When `type` is `"a"`, `n` must be an integer such that 0 ≤ `n` ≤ `size(s,1)-2`.
- When `type` is `"h"`, `"v"`, or `"d"`, `n` must be an integer such that 1 ≤ `n` ≤ `size(s,1)-2`.

Data Types: `double`

Output Arguments

x — Reconstructed coefficients

matrix

Reconstructed coefficients, returned as a matrix. The size of `x` is equal to the size of the original image

Data Types: `double`

See Also

`appcoef2` | `detcoef2` | `wavedec2`

Introduced before R2006a

wrev

Flip vector

Syntax

```
y = wrev(x)
```

Description

`y = wrev(x)` reverses the vector `x`.

Examples

Flip Vector

Create a vector.

```
v = [1 2 3 4 5];
```

Flip the vector.

```
wrev(v)
```

```
ans = 1×5

     5     4     3     2     1
```

Flip the transpose of the vector.

```
wrev(v')
```

```
ans = 5×1

     5
     4
     3
     2
     1
```

Input Arguments

x — Input
vector

Input, specified as a vector.

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

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

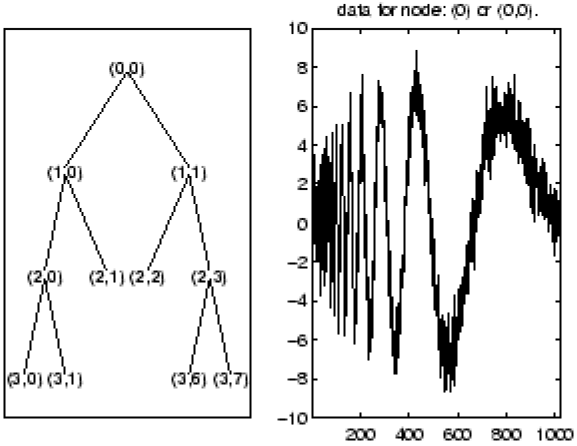
`T = write(T,'cfs',NODE,COEFS)` writes coefficients for the terminal node `NODE`.

`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.

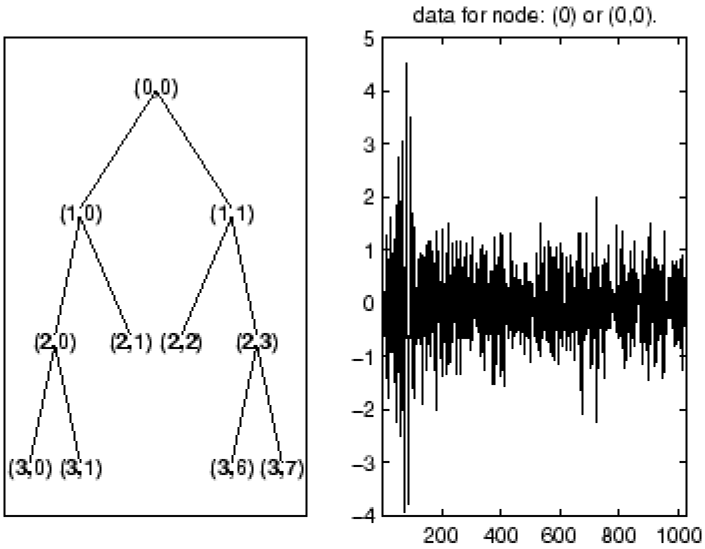
Examples

```
% 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.

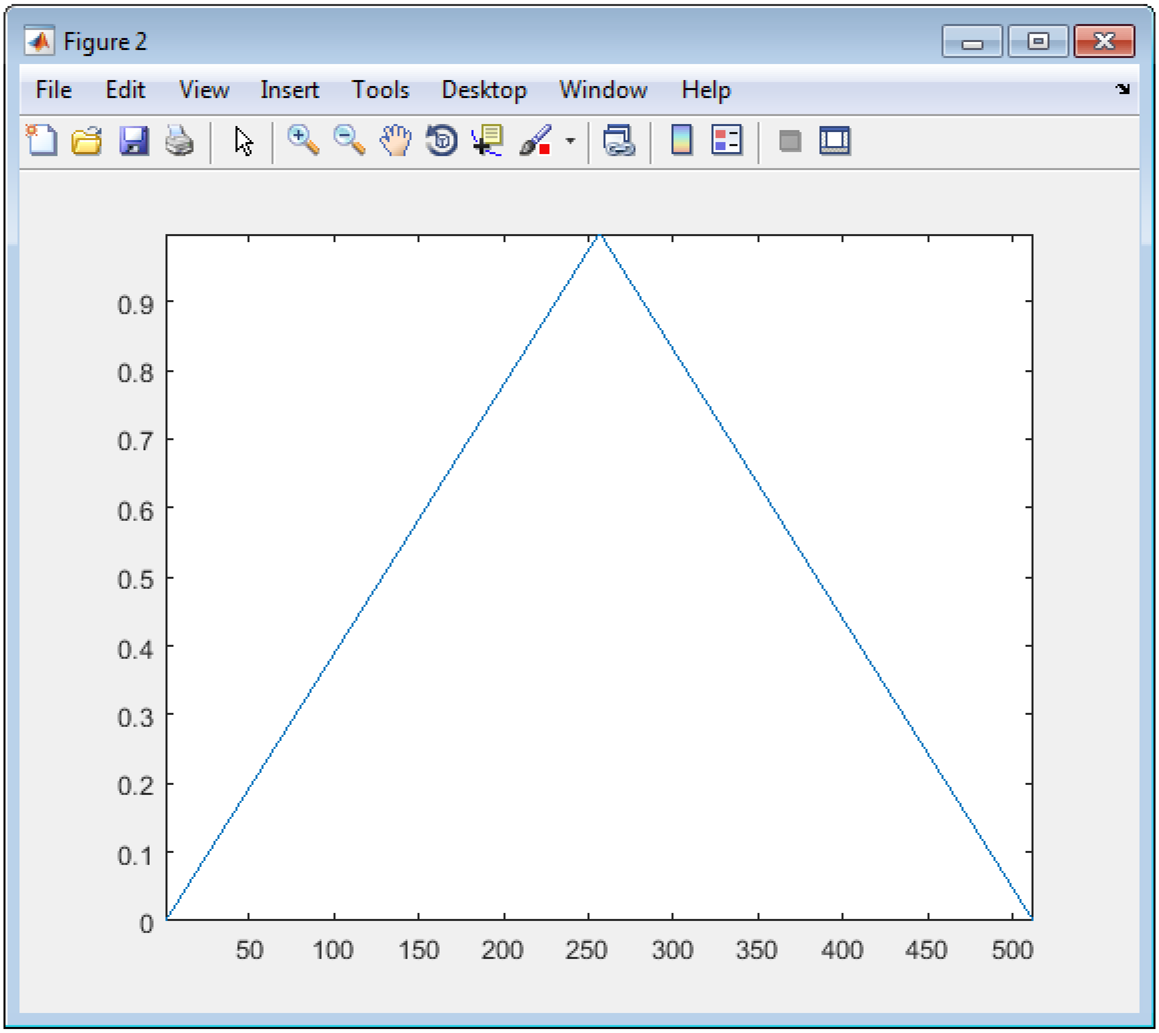
If power > 0, coefficients are first normalized

```
coefs(k,:) = coefs(k,:)/(scales(k)^power)
```

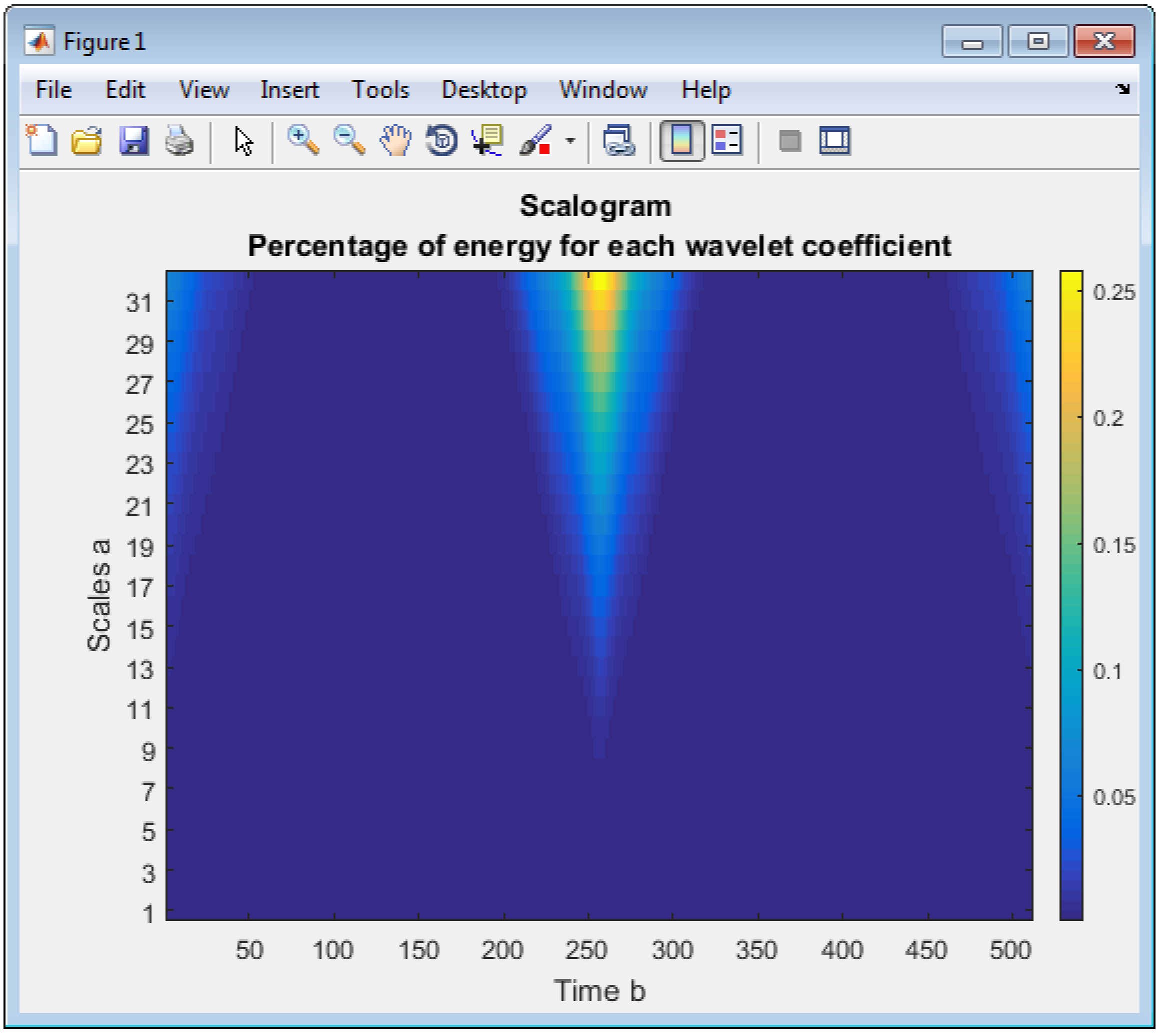
and then the scalogram is computed as explained above.

Examples

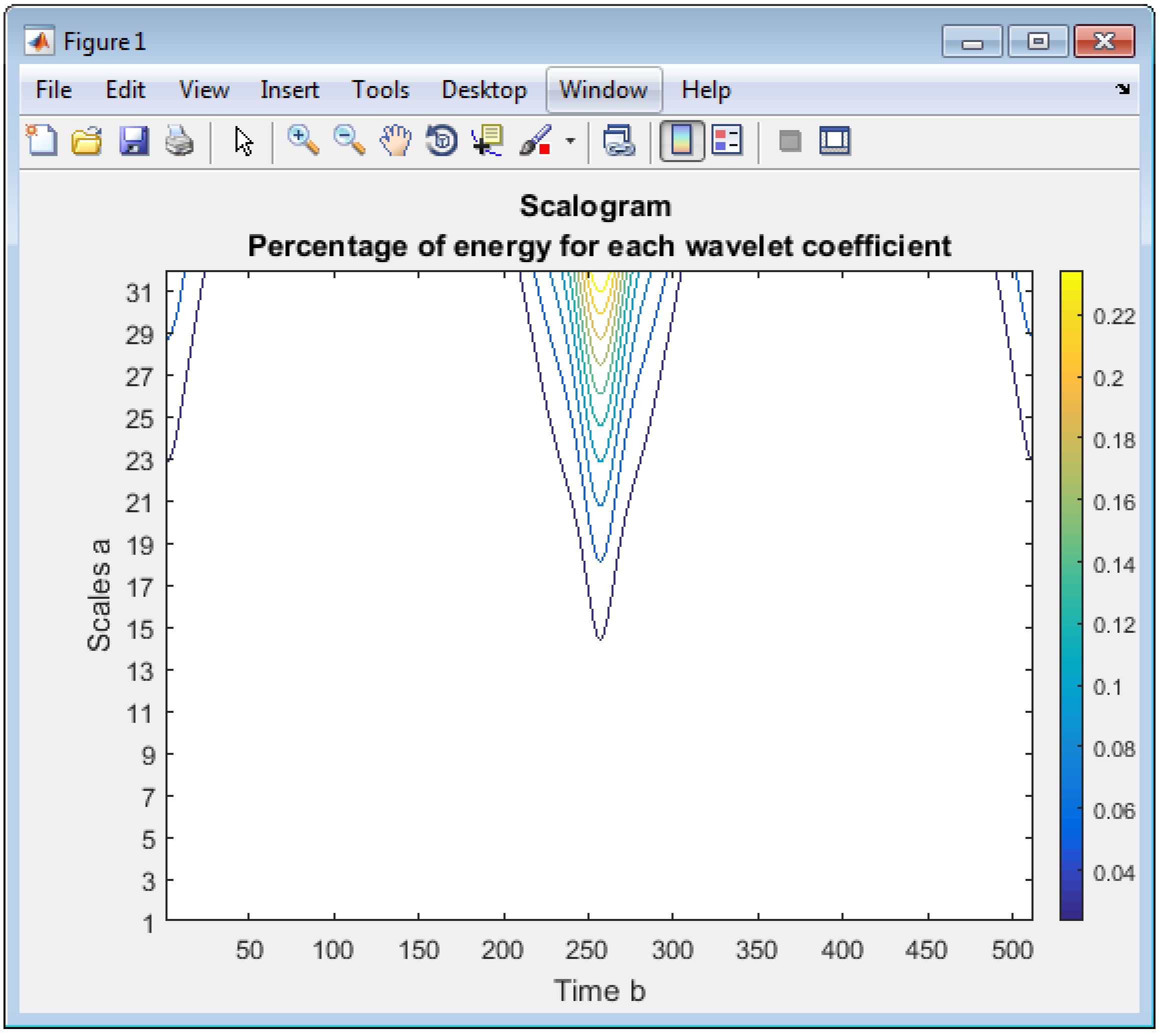
```
% Compute signal s
t = linspace(-1,1,512);
s = 1-abs(t);

% 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 synchrosqueezed 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

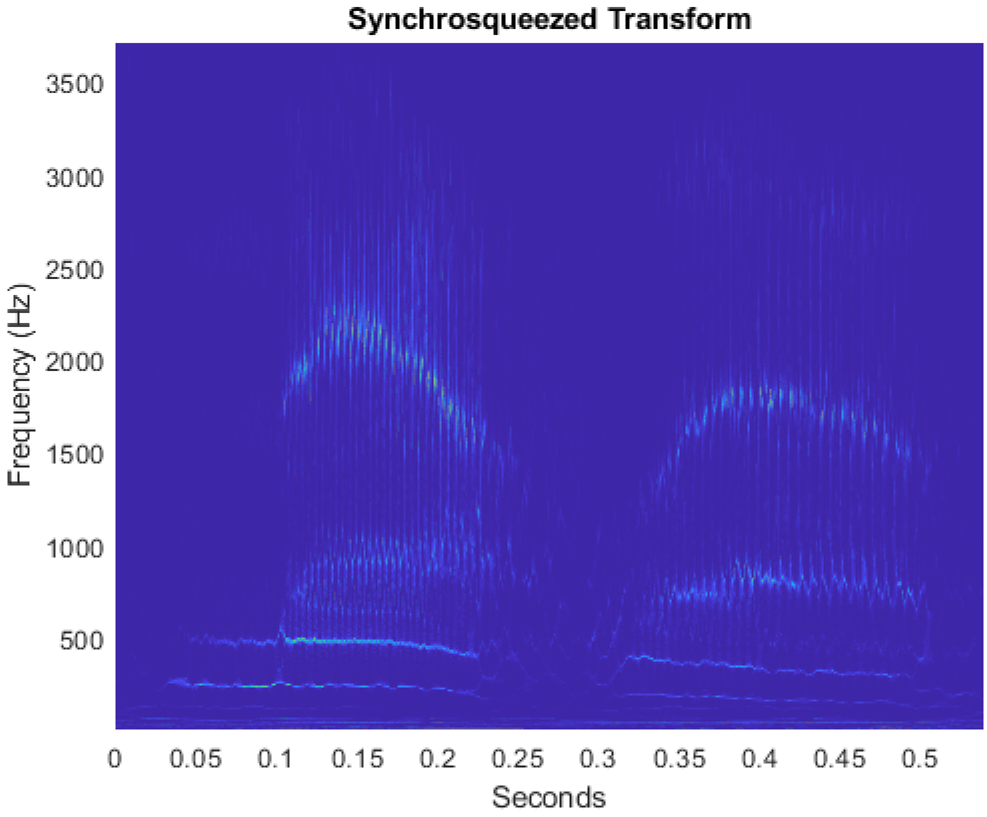
Obtain the wavelet synchrosqueezed transform of a speech signal and compare the original and reconstructed signals.

Load the speech signal and obtain its synchrosqueezed transform.

```
load mtlb
soundsc(mtlb,Fs)
dt = 1/Fs;
t = 0:dt:numel(mtlb)*dt-dt;
[sst,f] = wsst(mtlb,Fs);
```

Plot the synchrosqueezed transform.

```
pcolor(t,f,abs(sst))
shading interp
xlabel('Seconds')
ylabel('Frequency (Hz)')
title('Synchrosqueezed Transform')
```



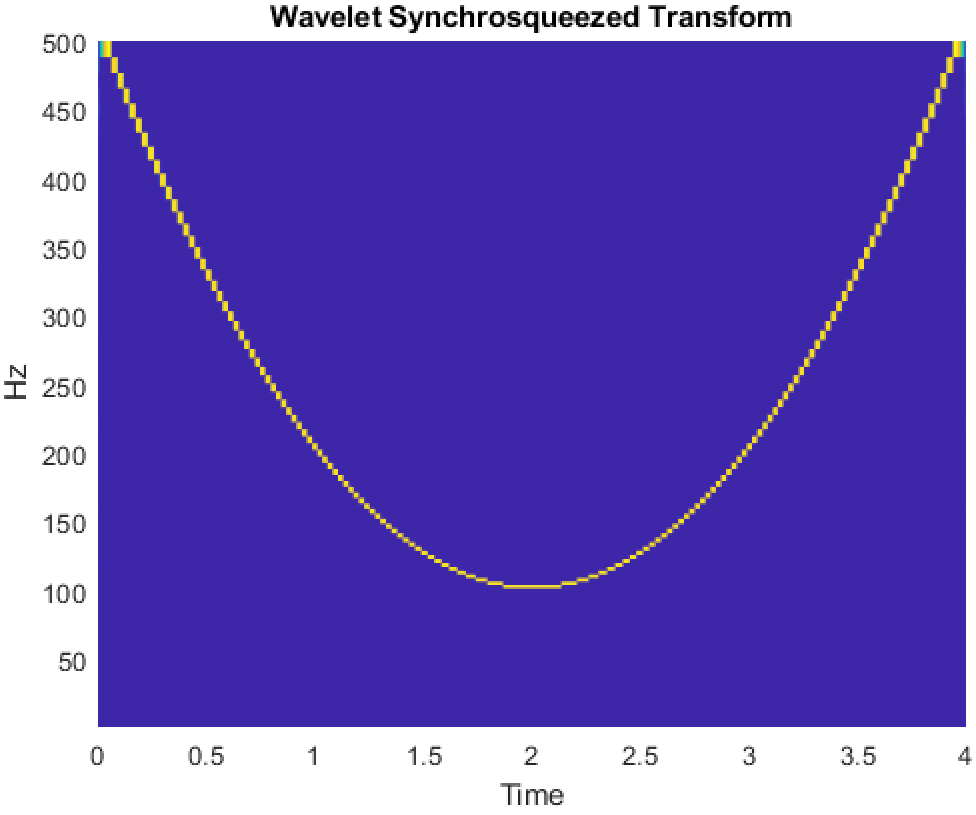
Obtain the inverse synchrosqueezed transform and play the reconstructed speech signal.

```
xrec = iwsst(sst);
soundsc(xrec,Fs)
```

Synchrosqueezed Transform of Quadratic Chirp

Obtain and plot the wavelet synchrosqueezed transform of a quadratic chirp. The chirp is sampled at 1000 Hz.

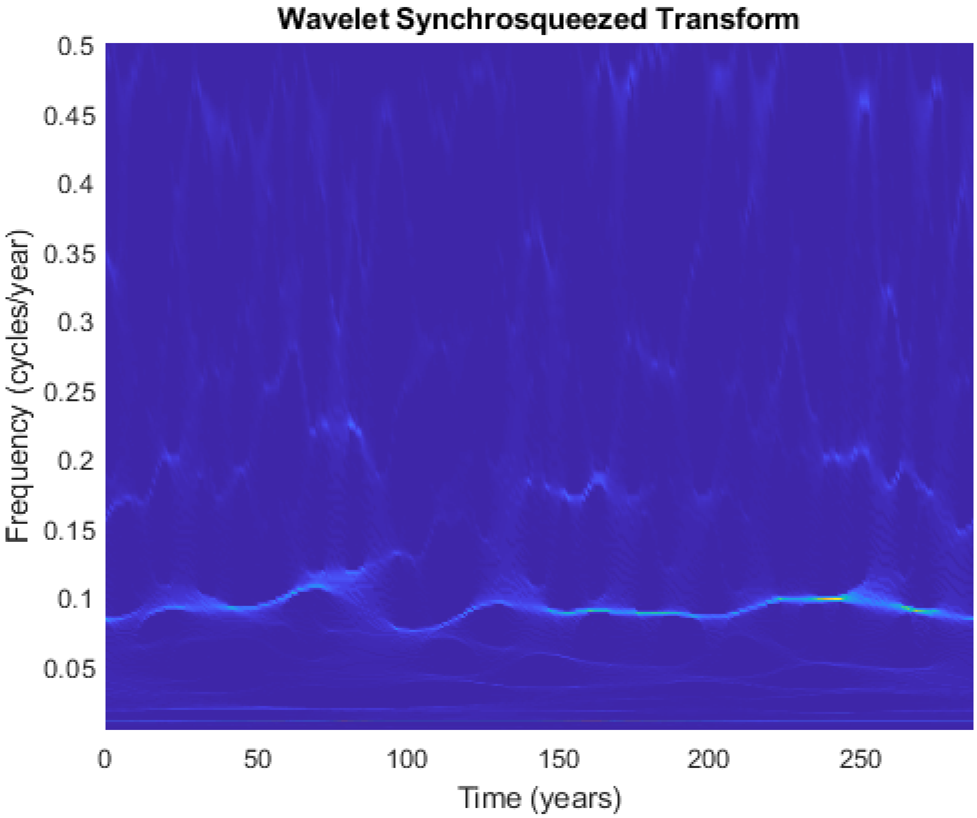
```
load quadchirp;
[sst,f] = wsst(quadchirp,1000);
hp = pcolor(tquad,f,abs(sst));
hp.EdgeColor = 'none';
title('Wavelet Synchrosqueezed Transform');
xlabel('Time'); ylabel('Hz');
```



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.

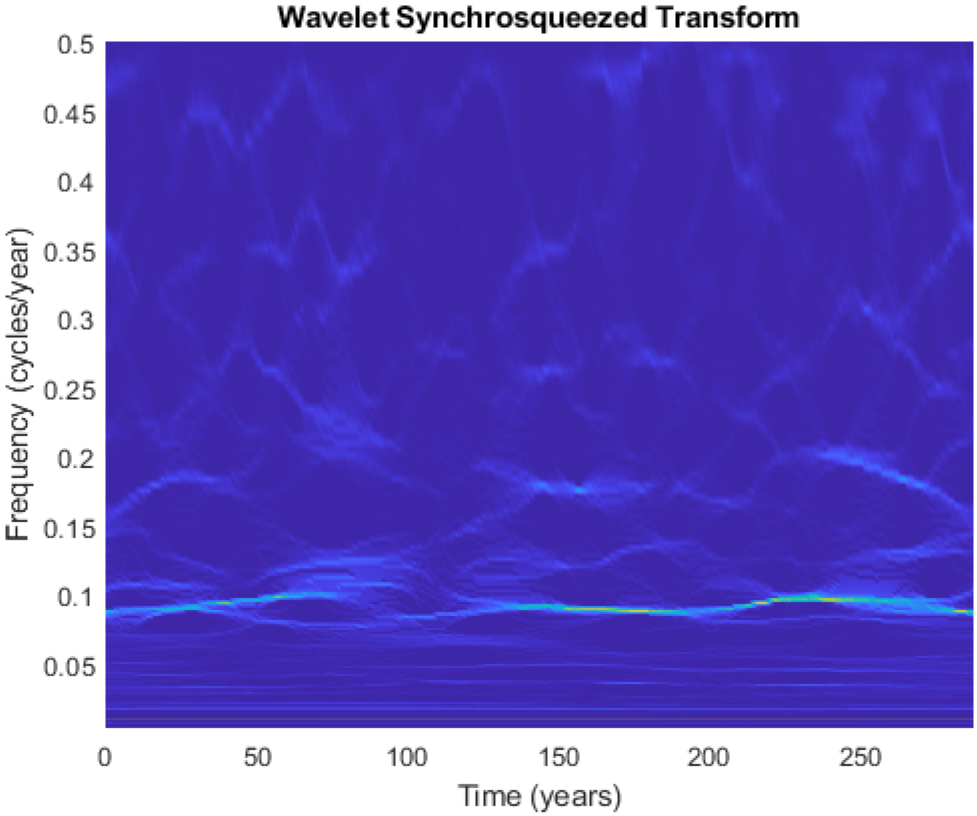
```
load sunspot
wsst(sunspot(:,2),years(1))
```



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
wsst(sunspot(:,2),years(1),'bump')
```



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 one of the following:

- `'amor'` — Analytic Morlet wavelet
- `'bump'` — Bump wavelet

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'VoicesPerOctave',26`

`VoicesPerOctave` — Number of voices per octave
`32` (default) | even 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 even 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] Daubechies, I., J. Lu, and H.-T. Wu. "Synchrosqueezed wavelet transforms: an empirical mode decomposition-like tool." *Applied and Computational Harmonic Analysis*. Vol. 30, Number 2, 2011, 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, Number 5, 2013, pp. 1079–1094.

See Also

`iwsst` | `wsstridge` | `years` | `days` | `hours` | `minutes` | `seconds` | `duration`

Topics

“Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”
“Wavelet Synchrosqueezing”
“Time-Frequency Gallery”

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 output 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.

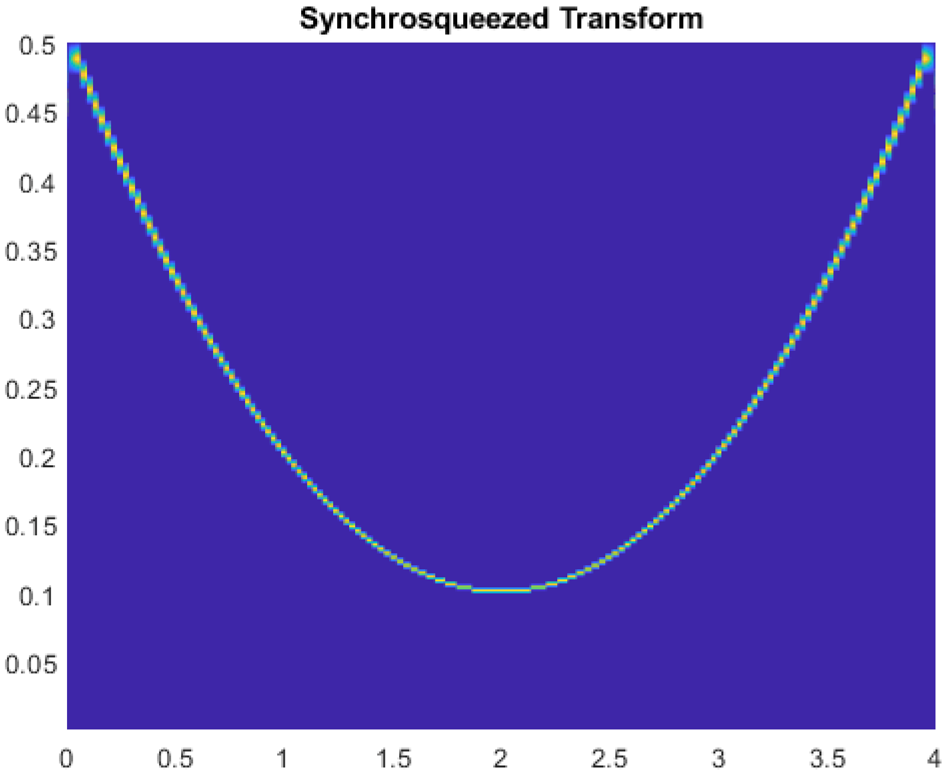
```
load quadchirp;
[sst,f] = wsst(quadchirp);
```

Extract the maximum time-frequency ridge.

```
[fridge,iridge] = wsstridge(sst);
```

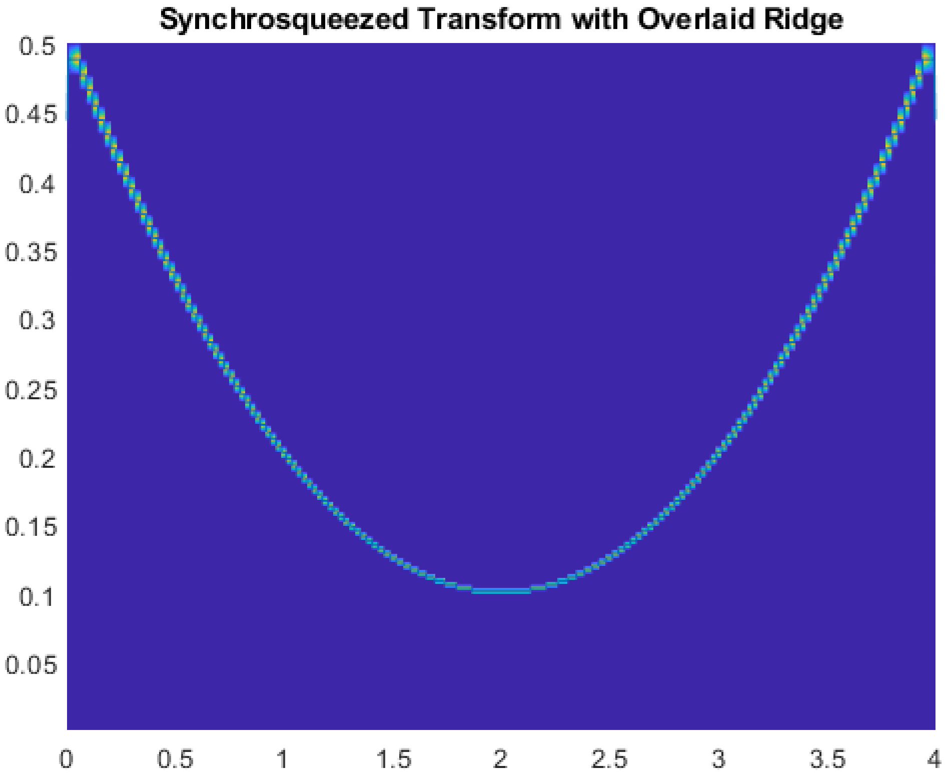
Plot the synchrosqueezed transform.

```
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')
```

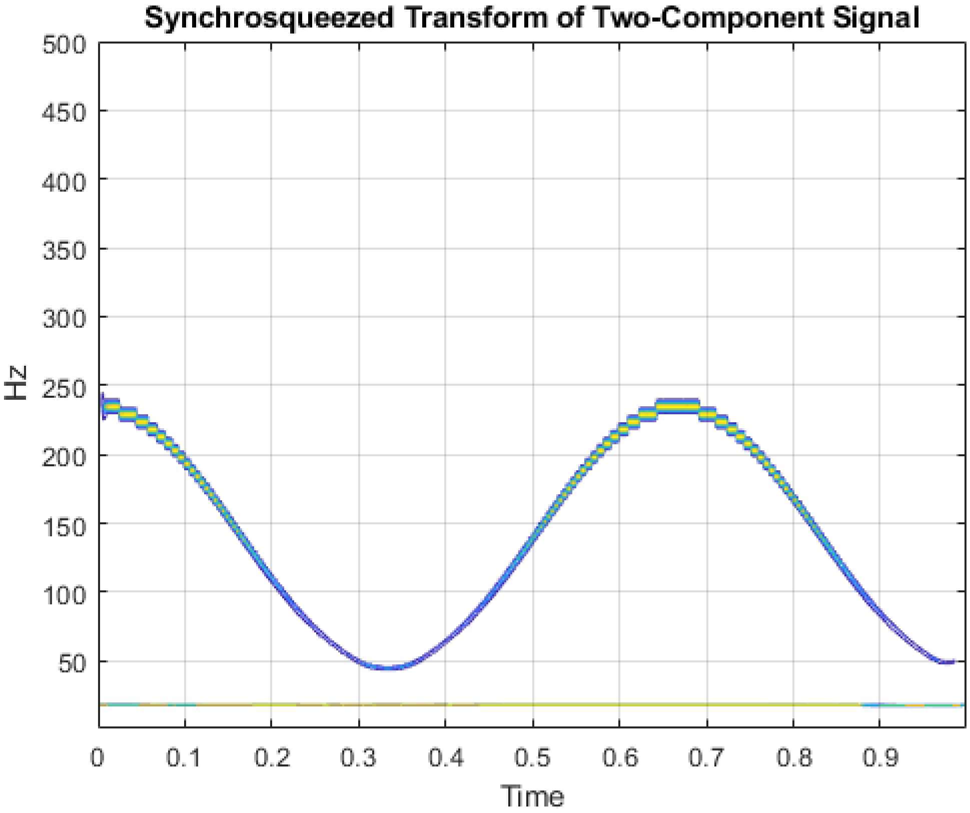


Extract Time-Frequency Ridge from Multicomponent Signal

Extract the two highest energy modes from a multicomponent signal.

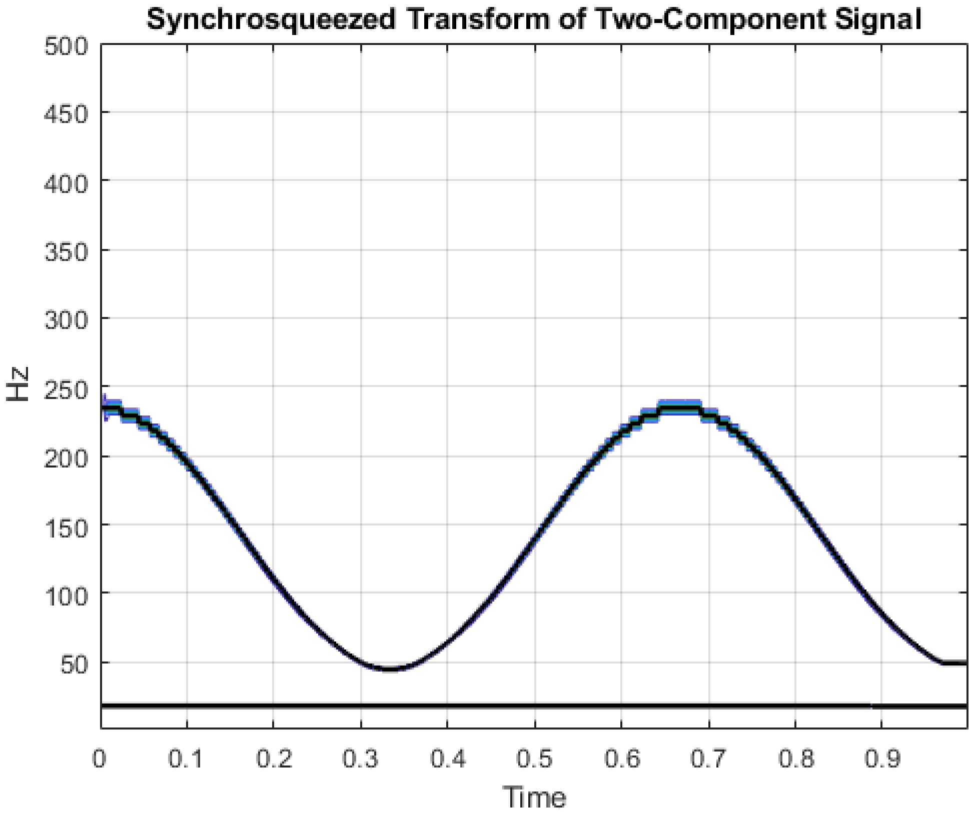
Obtain and plot the wavelet synchrosqueezed transform.

```
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);
```



Input Arguments

`sst` — Synchrosqueezed transform
matrix

Synchrosqueezed transform, specified as a matrix. `sst` is a time-frequency matrix and is the output of `wsst`.

`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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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 number 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

The function uses a penalized forward-backward greedy algorithm to extract the maximum-energy ridges from a time-frequency matrix. The algorithm finds the maximum time-frequency ridge by minimizing –ln *A* at each time point, where *A* is the absolute value of the matrix. Minimizing –ln *A* is equivalent to maximizing the value of *A*. The algorithm optionally constrains jumps in frequency with a penalty that is proportional to the distance between frequency bins.

The following example illustrates the time-frequency ridge algorithm using a penalty that is two times the distance between frequency bins. Specifically, the distance between the elements `(j,k)` and `(m,n)` is defined as `(j-m)`^{2}. The time-frequency matrix has three frequency bins and three time steps. The matrix columns correspond to time steps, and the matrix rows correspond to frequency bins. The values in the second row represent a sine wave.

1. Suppose you have the matrix:

   ```
   1   4   4
   2   2   2
   5   5   4
   ```

2. Update the value for the (1,2) element as follows.

   a. Leave the values 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 second time point. The bin value is 4. 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

      1 + 2 × 0 =  1
      2 + 2 × 1 =  4
      5 + 2 × 4 = 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 as follows.

   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. For example, when updating the (2,2) element, which has bin value 2, applying the penalty to the column yields

   ```
   original value + penalty × distance

   1 + 2 × 1 =  3
   2 + 2 × 0 =  2
   5 + 2 × 1 =  7
   ```

   Add the minimum value, 2, to the current bin value. The updated value for (2,2) becomes 4. After updating the (3,2) element, the matrix is

$$\begin{array}{lll} 1 & 5_{(1)} & 4 \\ 2 & 4_{(2)} & 2 \\ 5 & 9_{(2)} & 4 \end{array}$$

Only the second column has been updated. The subscripts indicate the index of the bin in the previous column from which a value came.

4. Repeat Step 2 for the third column. But now the penalty is applied to the updated second column. For example, when updating the (1,3) element, the penalty is

   ```
   5 + 2 × 0 =  5
   4 + 2 × 1 =  6
   9 + 2 × 4 = 17
   ```

   The minimum value, 5, which is in the first bin, is added to the (1,3) bin value. After updating all the values in the third column, the final matrix is

   $$\begin{array}{lll} 1 & 5_{(1)} & 9_{(1)} \\ 2 & 4_{(2)} & 6_{(2)} \\ 5 & 9_{(2)} & 10_{(2)} \end{array}$$

5. 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.

If you are extracting multiple ridges, the algorithm removes the first ridge from the time-frequency matrix and repeats the process.

References

[1] Daubechies, I., J. Lu, and H.-T. Wu. "Synchrosqueezed wavelet transforms: an empirical mode decomposition-like tool." *Applied and Computational Harmonic Analysis*. Vol. 30, Number 2, 2011, 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, Number 4, 2013, pp. 1079–1094.

See Also

`wsst` | `iwsst`

Topics

“Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”
“Wavelet Synchrosqueezing”

Introduced in R2016a

wt

Continuous wavelet transform with filter bank

Syntax

```
cfs = wt(fb,x)
[cfs,f] = wt(fb,x)
[cfs,f,coi] = wt(fb,x)
[cfs,f,coi,scalcfs] = wt(fb,x)
[cfs,p] = wt(fb,x)
[cfs,p,coi] = wt(fb,x)
[cfs,p,coi,scalcfs] = wt(fb,x)
```

Description

`cfs = wt(fb,x)` returns the continuous wavelet transform (CWT) coefficients of the signal `x`, using `fb`, a CWT filter bank. `x` is a real- or complex-valued vector. `x` must have at least 4 samples. If `x` is real-valued, `cfs` is a 2-D matrix, where each row corresponds to one scale. The column size of `cfs` is equal to the length of `x`. If `x` is complex-valued, `cfs` is a 3-D array, 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).

`[cfs,f] = wt(fb,x)` returns the frequencies `f` corresponding to the scales (rows) of `cfs` if the `SamplingPeriod` property is not specified in the CWT filter bank `fb`. If you do not specify a sampling frequency, `f` is in cycles/sample.

`[cfs,f,coi] = wt(fb,x)` returns the cone of influence `coi` for the CWT. `coi` is in the same units as `f`. If the input `x` is complex, the `coi` applies to both pages of `cfs`.

`[cfs,f,coi,scalcfs] = wt(fb,x)` returns the scaling coefficients `scalcfs` for the wavelet transform.

`[cfs,p] = wt(fb,x)` returns the periods `p` corresponding to the scales (rows) of `cfs` if you specify a sampling period in the CWT filter bank. `p` has the same units and format as the duration scalar sampling period.

`[cfs,p,coi] = wt(fb,x)` returns the cone of influence `coi` in periods for the CWT. `coi` is an array of durations with the same format property as the sampling period. If the input `x` is complex, the `coi` applies to both pages of `cfs`.

`[cfs,p,coi,scalcfs] = wt(fb,x)` returns the scaling coefficients `scalcfs` for the wavelet transform.

Examples

Continuous Wavelet Transform Using Filter Bank

Load the noisy Doppler signal. Create a CWT filter bank that can be applied to the signal.

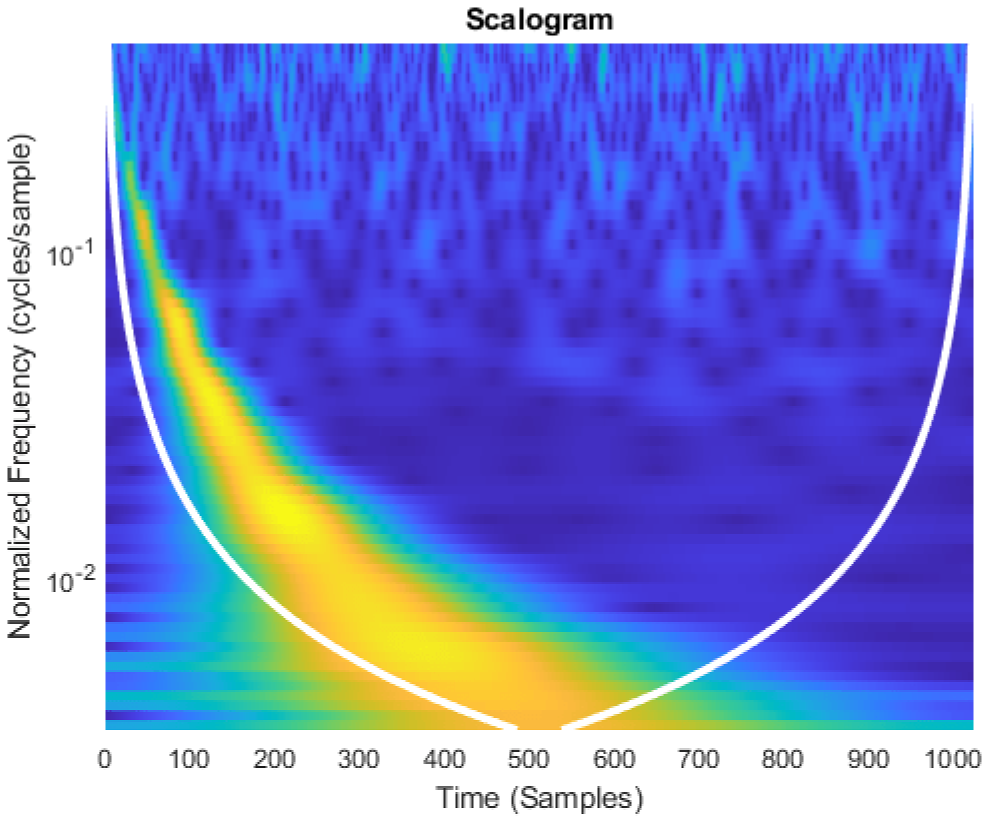
```
load noisdopp
fb = cwtfilterbank('SignalLength',numel(noisdopp));
```

Use the filter bank to obtain the continuous wavelet transform of the signal.

```
[cfs,f,coi] = wt(fb,noisdopp);
```

Plot the CWT scalogram, including the cone of influence.

```
t = 0:numel(noisdopp)-1;
pcolor(t,f,abs(cfs))
shading flat
set(gca,'YScale','log')
hold on
plot(t,coi,'w-','LineWidth',3)
xlabel('Time (Samples)')
ylabel('Normalized Frequency (cycles/sample)')
title('Scalogram')
```

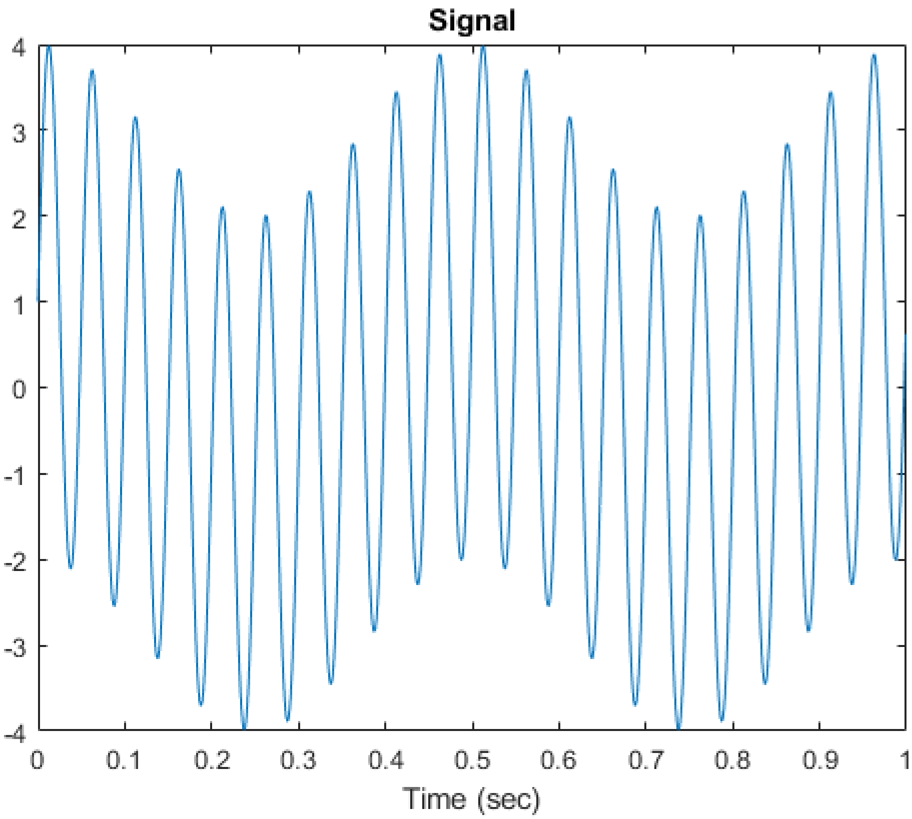


Inverse Continuous Wavelet Transform Using Scaling Coefficients

Create and plot a signal sampled at 1000 Hz. Create a CWT filter bank that can be used on the signal. Since the signal is periodic, set the boundary extension property of the filter bank to `'periodic'`.

```
Fs = 1000;
t = 0:1/Fs:1-1/Fs;
```

```
sig = 3*sin(2*pi*20*t) + cos(2*pi*2*t);
fb = cwtfilterbank('SignalLength',length(sig),...
    'SamplingFrequency',Fs,...
    'Boundary','periodic');
plot(t,sig)
xlabel('Time (sec)')
title('Signal')
```

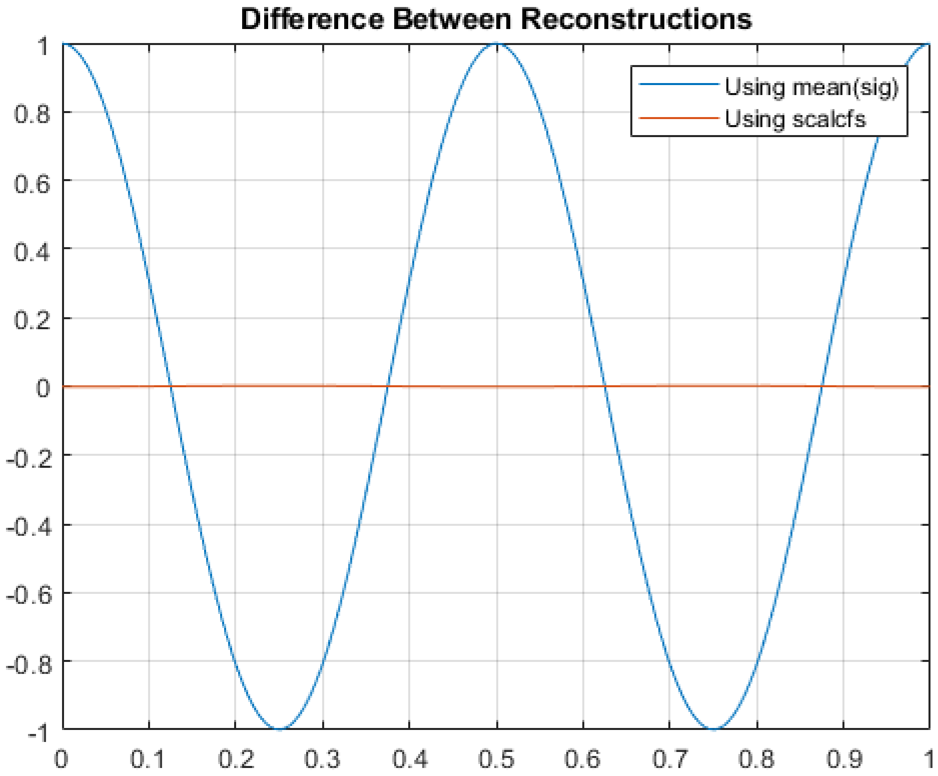


Take the CWT of the signal. Return the wavelet and scaling coefficients.

```
[cfs,~,~,scalcfs] = wt(fb,sig);
```

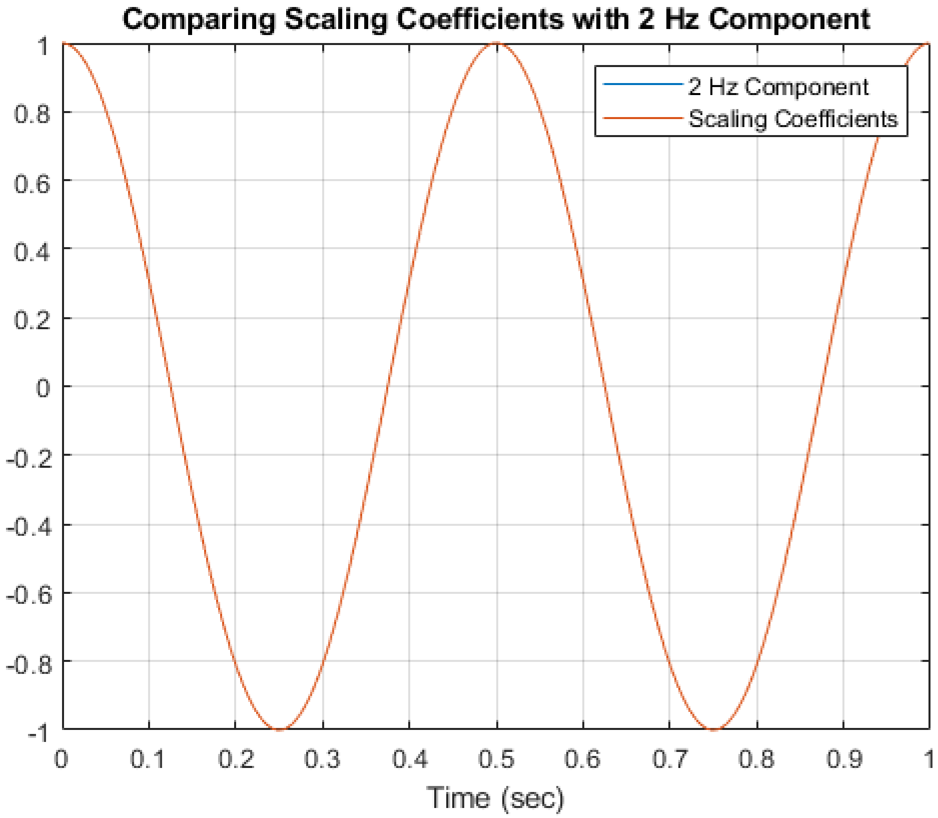
Reconstruct the signal two ways. First use the mean of the signal, then use the scaling coefficients. Plot the difference between the original signal and both reconstructions.

```
xrec0 = icwt(cfs,'SignalMean',mean(sig));
xrec1 = icwt(cfs,'ScalingCoefficients',scalcfs);
plot(t,sig-xrec0)
hold on
plot(t,sig-xrec1)
grid on
legend('Using mean(sig)','Using scalcfs')
title('Difference Between Reconstructions')
```



The scaling coefficients results in a significantly more accurate reconstruction. To investigate the source of the dramatic improvement, create a second signal consisting of the 2 Hz component of the original signal. Compare the scaling coefficients with the 2 Hz signal. The scaling coefficients and 2 Hz signal are virtually identical. Using the scaling coefficients helps with the reconstruction because the 2 Hz component is not representable by a wavelet with this sampling frequency and length.

```
figure
sig2hz = cos(2*pi*2*t);
plot(t,sig2hz)
hold on
plot(t,scalcfs)
grid on
title('Comparing Scaling Coefficients with 2 Hz Component')
xlabel('Time (sec)')
legend('2 Hz Component', 'Scaling Coefficients')
```



Using CWT Filter Bank on Multiple Time Series

This example shows how using a CWT filter bank can improve computational efficiency when taking the CWT of multiple time series.

Create a 100-by-1024 matrix `x`. Create a CWT filter bank appropriate for signals with 1024 samples.

```
x = randn(100,1024);
fb = cwtfilterbank;
```

Use `cwt` with default settings to obtain the CWT of a signal with 1024 samples. Create a 3-D array that can contain the CWT coefficients of 100 signals, each of which has 1024 samples.

```
cfs = cwt(x(1,:));
res = zeros(100,size(cfs,1),size(cfs,2));
```

Use the `cwt` function and take the CWT of each row of the matrix `x`. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = cwt(x(k,:));
end
toc
```

```
Elapsed time is 0.928160 seconds.
```

Now use the `wt` object function of the filter bank to take the CWT of each row of `x`. Display the elapsed time.

```
tic
for k=1:100
    res(k,:,:) = wt(fb,x(k,:));
end
toc
```

```
Elapsed time is 0.393524 seconds.
```

Input Arguments

`fb` — Continuous wavelet transform filter bank

`cwtfilterbank` object

Continuous wavelet transform (CWT) filter bank, specified as a `cwtfilterbank` object.

`x` — Input signal

real- or complex-valued vector | `gpuArray`

Input signal, specified as a real- or complex-valued vector. `x` must have at least four samples.

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

Output Arguments

`cfs` — Continuous wavelet transform

matrix | 3-D array

Continuous wavelet transform, returned as a matrix or 3-D array of complex values. If `x` is real-valued, `cfs` is a 2-D matrix, where each row corresponds to one scale. The column size of `cfs` is equal to the length of `x`. If `x` is complex-valued, `cfs` is a 3-D array, 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).

Data Types: `double` | `single`

`f` — Frequencies

vector

Frequencies, returned as a vector, corresponding to the scales (rows) of `cfs` if the `'SamplingPeriod'` is not specified in `fb`. If you specify a sampling frequency, `f` is in hertz. If you do not specify a frequency, `f` is in cycles/sample.

Data Types: `double`

`p` — Periods

array

Periods, returned as an array of durations, corresponding to the scales (rows) of `cfs` if `fb` has a specified sampling period. `p` has the same units and format as the duration scalar sampling period.

Data Types: `duration`

coi — Cone of influence
array of real numbers | array of durations

Cone of influence for the CWT, returned as either an array of real numbers or an array of durations. The cone of influence indicates where edge effects occur in the CWT. If you specify a sampling frequency, `coi` is an array of real numbers in the same units as `f`. If you specify a sampling period, `coi` is an array of durations with the same format property as the sampling period. Due to the edge effects, give less credence to areas that are outside or overlap the cone of influence.

For additional information, see “Boundary Effects and the Cone of Influence”.

Data Types: `double` | `duration`

scalcfs — Scaling coefficients
real- or complex-valued vector

Scaling coefficients for the wavelet transform, returned as a vector with the same length as `x`. If `x` is real-valued, `scalcfs` is real valued. If `x` is complex-valued, `scalcfs` is complex-valued.

Data Types: `double`

Tips

- The first time you use a filter bank to take the CWT of a signal, the wavelet filters are constructed to have the same datatype as the signal. A warning message is generated when you apply the same filter bank to a signal with a different datatype. Changing datatypes comes with the cost of redesigning or changing the precision of the filter bank. For optimal performance, use a consistent datatype.
- When performing multiple CWTs, for example inside a for-loop, the recommended workflow is to first create a `cwtfilterbank` object and then use the `wt` object function. This workflow minimizes overhead and maximizes performance. See “Using CWT Filter Bank on Multiple Time Series” on page 1-1754.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps
Wavelet Time-Frequency Analyzer

Functions
`cwt` | `cwtfilterbank` | `icwt`

Topics
"Using Wavelet Time-Frequency Analyzer App"
"Boundary Effects and the Cone of Influence"

Introduced in R2018a

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.

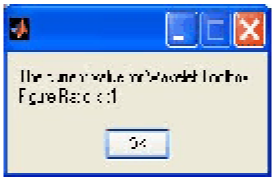
`wtbxmngr(OPTION)` sets a toolbox option. Available options are

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.

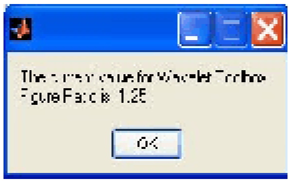
`V = wtbxmngr('version')` saves the current version of the toolbox to variable `V`.

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
```



The current value for Wavelet Toolbox Figure Ratio is 1



The current value for Wavelet Toolbox Figure Ratio is 1.25

Introduced before R2006a

wthcoef

1-D wavelet coefficient thresholding

Syntax

```
nc = wthcoef("a",c,l)
nc = wthcoef("d",c,l,n)
nc = wthcoef("d",c,l,n,p)
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("a",c,l)` returns coefficients obtained from the multilevel wavelet decomposition structure [`c`,`l`] by setting the approximation coefficients to zero . For information about the decomposition structure, see `wavedec`.

`nc = wthcoef("d",c,l,n)` returns coefficients obtained from [`c`,`l`] by setting all the coefficients at detail levels specified in `n` to zero.

`nc = wthcoef("d",c,l,n,p)` returns coefficients obtained from [`c`,`l`] by rate compression defined in vectors `n` and `p`. `n` specifies the detail levels to be compressed and `p` the corresponding percentages of lower coefficients to set to zero. `n` and `p` must be of the same length.

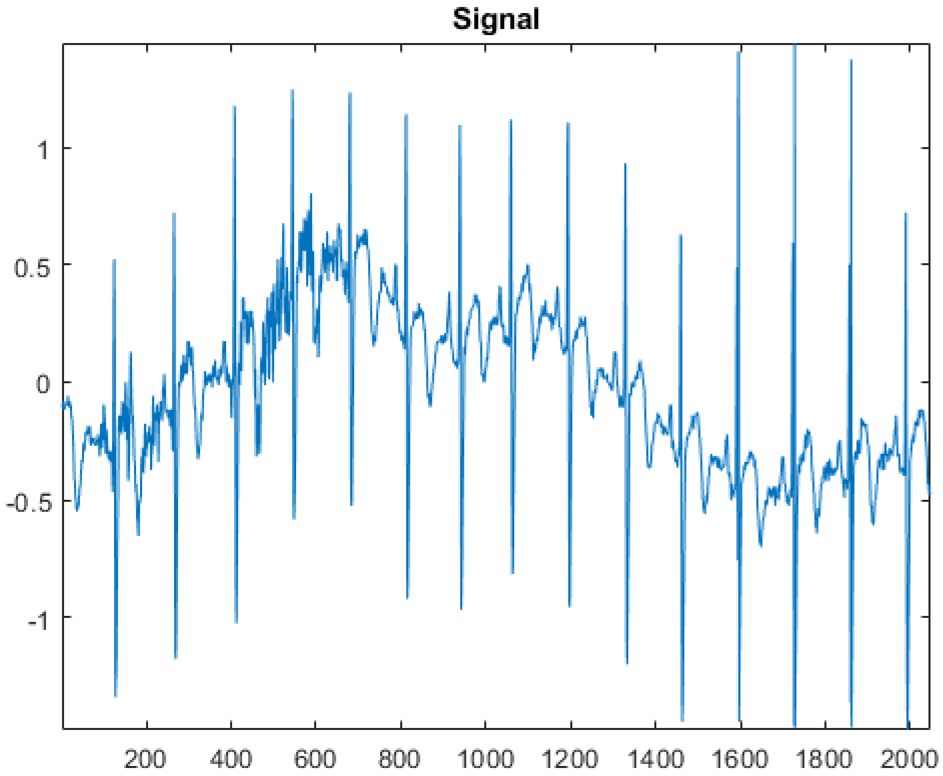
`nc = wthcoef("t",c,l,n,t,sorh)` returns coefficients obtained from [`c`,`l`] by thresholding specified in `thr`. `n` specifies the detail levels to be thresholded and `t` the corresponding thresholds. `n` and `t` must be of the same length.

Examples

Modify Approximation Coefficients

Load and plot a 1-D signal.

```
load wecg
plot(wecg)
title("Signal")
axis tight
```

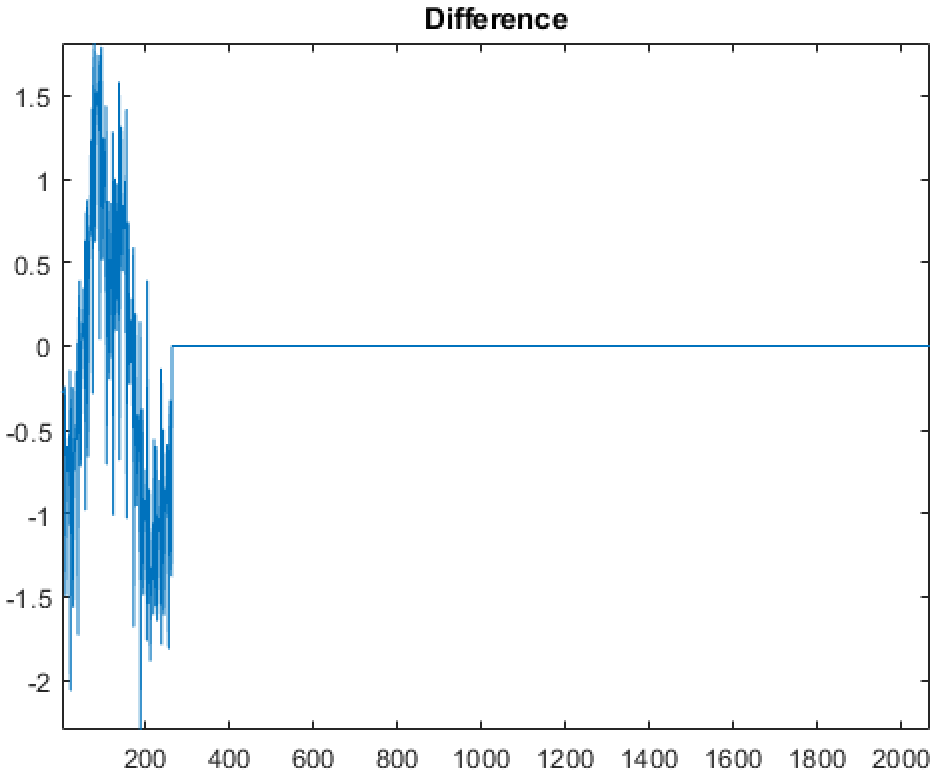


Obtain a level 3 wavelet decomposition of the signal using the Daubechies `db4` wavelet.

```
wv = "db4";
[c,l] = wavedec(wecg,3,wv);
```

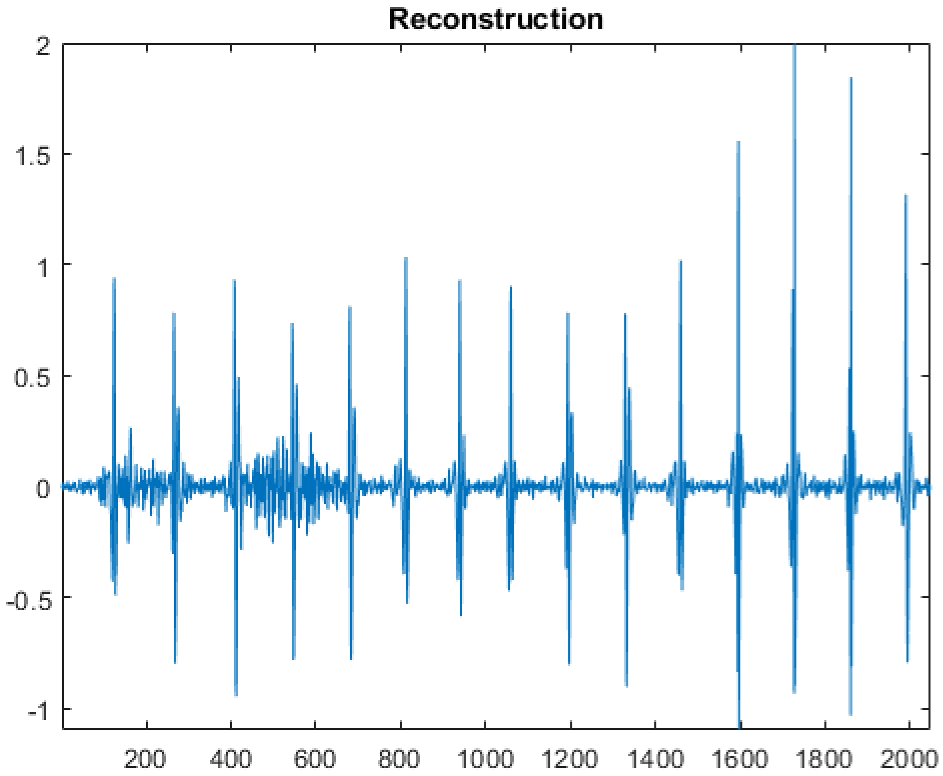
Use `wthcoef` to modify the wavelet decomposition `c`. Set the approximation coefficients to zero and plot the difference between the original and modified wavelet decompositions.

```
nc = wthcoef("a",c,l);
plot(c-nc)
title("Difference")
axis tight
```



Reconstruct a signal using the modified wavelet decomposition.

```
xrec = waverec(nc,l,wv);
plot(xrec)
title("Reconstruction")
axis tight
```



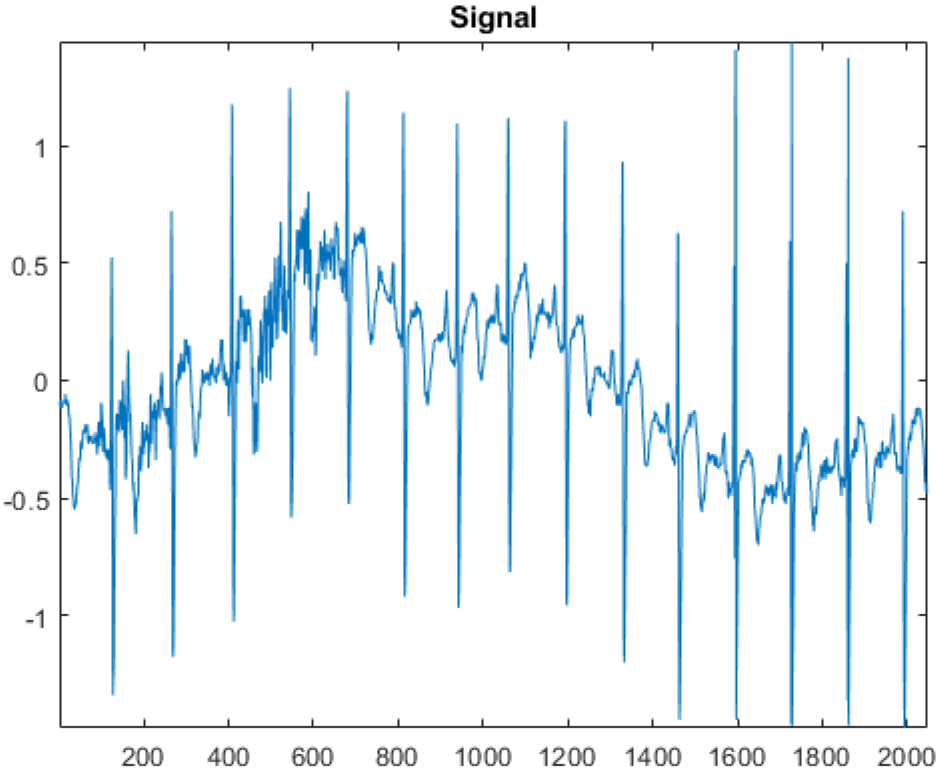
Threshold Wavelet Coefficients

Save the current extension mode. Change the extension mode to periodized extension.

```
origmode = dwtmode("status","nodisplay");
dwtmode("per","nodisplay")
```

Load and plot a 1-D signal. The signal has 2048 samples.

```
load wecg
plot(wecg)
title("Signal")
axis tight
```



Obtain a level 2 wavelet decomposition of the signal using the Haar wavelet. Display the bookkeeping vector. Confirm there are 1024 detail coefficients at level 1, and 512 detail coefficients at level 2.

```
wv = "haar";
[c,l] = wavedec(wecg,2,wv);
l
```

```
l = 4×1

         512
         512
        1024
        2048
```

Use `wthcoef` to threshold the level 1 and level 2 detail coefficients in the wavelet decomposition `c`. Set 75% of the level 1 detail coefficients to zero, and set 50% of the level 2 detail coefficients to zero.

```
nc = wthcoef("d",c,l,[1 2],[75 50]);
```

Confirm there are 256 nonzero wavelet coefficients at levels 1 and 2 in the modified wavelet decomposition `nc`.

```
[level1,level2] = detcoef(nc,l,[1 2]);
[nnz(level1) 1024*0.25]
```

```
ans = 1×2
```

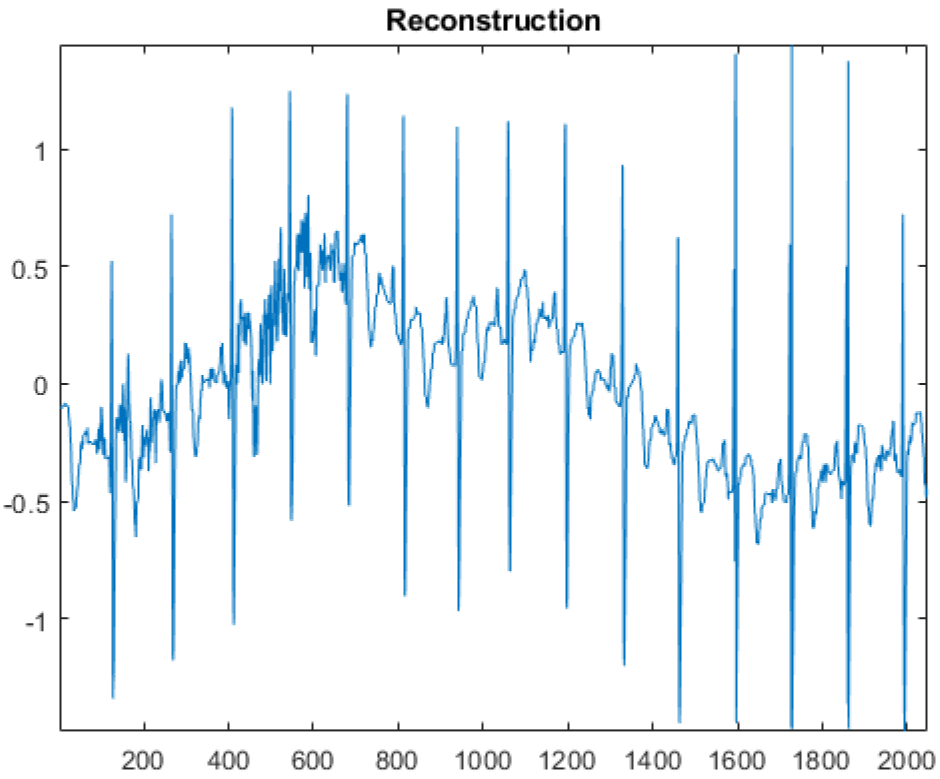
```
   256   256
```

```
[nnz(level2) 512*0.5]
```

```
ans = 1×2

   256   256
```

Reconstruct a signal using the modified wavelet decomposition.

```
xrec = waverec(nc,l,wv);
plot(xrec)
title("Reconstruction")
axis tight
```



Restore the original extension mode.

```
dwtmode(origmode,"nodisplay")
```

Input Arguments

c — Wavelet decomposition

vector

Wavelet decomposition, specified as a vector. The vector contains the wavelet coefficients. The bookkeeping vector `l` contains the number of coefficients by level. `c` is the output of `wavedec`.

Data Types: `single` | `double`

`l` — Bookkeeping vector

vector

Bookkeeping vector, specified as a vector of positive integers. The bookkeeping vector is used to parse the coefficients in the wavelet decomposition `c` by level. `l` is the output of `wavedec`.

Data Types: `single` | `double`

`n` — Detail levels

vector

Detail levels, specified as a vector of positive integers less than or equal to `N`, where `N` is the level of the wavelet decomposition used to obtain `[c,l]`. Specifically, `N = length(l)-2`.

Data Types: `double`

`p` — Percentages

vector

Percentages of coefficients to set to zero, specified as a vector of positive integers less than or equal to 100. `p` and `n` must be the same length.

Data Types: `double`

`t` — Thresholds

vector

Thresholds to apply to detail coefficients, specified as a real-valued vector. `t` and `n` must be the same length.

Data Types: `double`

`sorh` — Type of thresholding

`"s"` | `"h"`

Type of thresholding to perform:

- `"s"` — Soft thresholding
- `"h"` — Hard thresholding

Output Arguments

`nc` — Modified wavelet decomposition

vector

Modified wavelet decomposition, returned as a vector. `nc` and `c` have equal length.

Extended Capabilities

C/C++ Code Generation

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

See Also

wavedec | wdenoise | wthresh

Introduced before R2006a

wthcoef2

2-D wavelet coefficient thresholding

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

`NC = wthcoef2('type',C,S,N,T,SORH)` returns the horizontal, vertical, or diagonal coefficients obtained from the wavelet decomposition structure `[C,S]` by soft or hard thresholding defined in vectors `N` and `T`.

`wthcoef2` is a two-dimensional denoising and compression oriented function.

`NC = wthcoef2('type',C,S,N)` returns the horizontal, vertical, or diagonal 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 or hard thresholding defined in vectors `N` and `T`.

`[NC,S]` is the modified wavelet decomposition structure.

Examples

Calculate Coefficients Obtained From Wavelet Decomposition Structure

Load the image data.

```
load mask
```

Perform a level 2 wavelet decomposition of the image using the `haar` wavelet.

```
[C,S]=wavedec2(X,2,'haar');
```

Calculate the vertical coefficients obtained from the wavelet decomposition structure by soft thresholding defined in thresholding vectors `[1 2]` and `[2 4]`.

```
NC = wthcoef2('v',C,S,[1 2],[2 4],'s')
```

```
NC = 1×65536
10^3 ×

    0.9280    0.9265    0.9295    0.9258    0.9305    0.9245    0.9340    0.9235    0.9268    0.9
```

Input Arguments

'type' — Type of coefficients

'h' | 'v' | 'd'

Type of coefficients obtained from the wavelet decomposition structure, specified as one of the following:

- 'h'— Horizontal coefficients
- 'v'— Vertical coefficients
- 'd'— Diagonal coefficients

For more information, see wthresh.

C — Wavelet decomposition vector

real-valued vector

Wavelet decomposition vector. The vector C contains the approximation and detail coefficients organized by level. The function uses the bookkeeping matrix S to parse C.

The vector C is organized as *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 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.

For more information, see wavedec2.

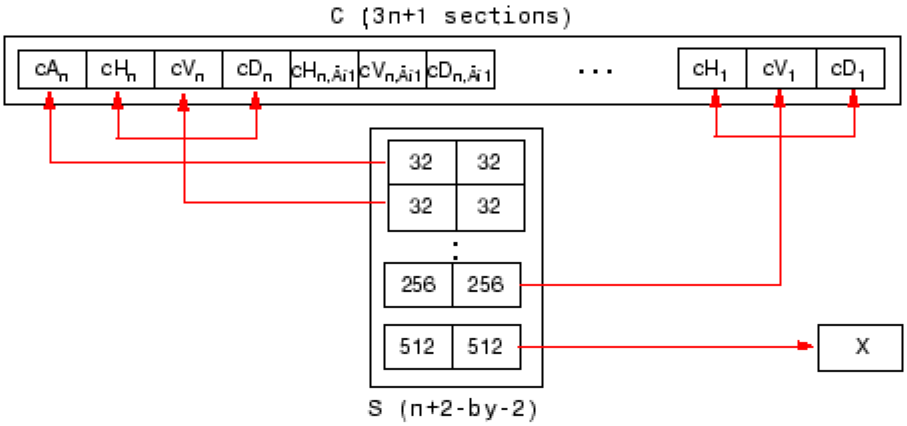
S — Bookkeeping matrix

integer-valued matrix

Bookkeeping matrix. The matrix S contains the dimensions of the wavelet coefficients by level and the function uses it to parse the wavelet decomposition vector C.

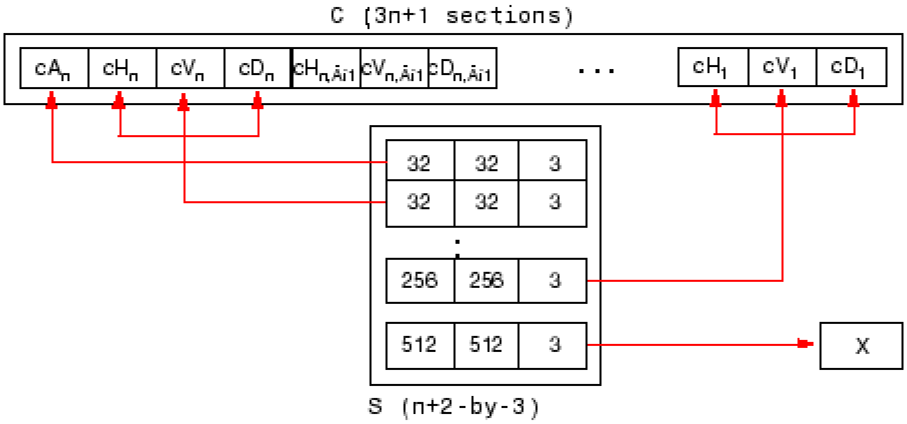
- S(1,:) = size of approximation coefficients(N).
- S(i,:) = size of detail coefficients(N-i+2) for i = 2, ...N+1 and S(N+2,:) = size(X).

The following diagram shows the relationship between C and S in the wavelet decomposition of a 512-by-512 matrix.



When X represents an indexed image, 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 the analyzed image.

For a truecolor image, the decomposition vector C and the corresponding bookkeeping matrix S can be represented as shown.



For more information, see `wavedec2`.

N — Threshold vector

`1 ≤ N(i) ≤ size(S,1)-2`

Threshold vector, specified by a size `1 ≤ N(i) ≤ size(S,1)-2`. N contains the detail levels to be thresholded and T the corresponding thresholds.

T — Threshold vector

nonnegative vectors

Threshold vector, specified as a nonnegative vector. N and T must be of the same length. N contains the detail levels to be thresholded and T the corresponding thresholds.

SORH — Soft or hard threshold

`'s'` | `'h'`

Soft or hard threshold, specified as `'s'` or `'h'`.

For more information, see `wthresh`.

Output Arguments

`NC` — Wavelet coefficient threshold

real-valued vector

Wavelet coefficient threshold, returned as a real-valued vector.

Extended Capabilities

C/C++ Code Generation

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

See Also

`wthcoef` | `wavedec2` | `wthresh`

Introduced before R2006a

wthresh

Soft or hard thresholding

Syntax

```
Y = wthresh(X,sorh,T)
```

Description

`Y = wthresh(X,sorh,T)` returns the soft or hard thresholding, indicated by `sorh`, of the vector or matrix X. T is the threshold value.

Examples

Hard and Soft Thresholding

Generate a signal and set a threshold.

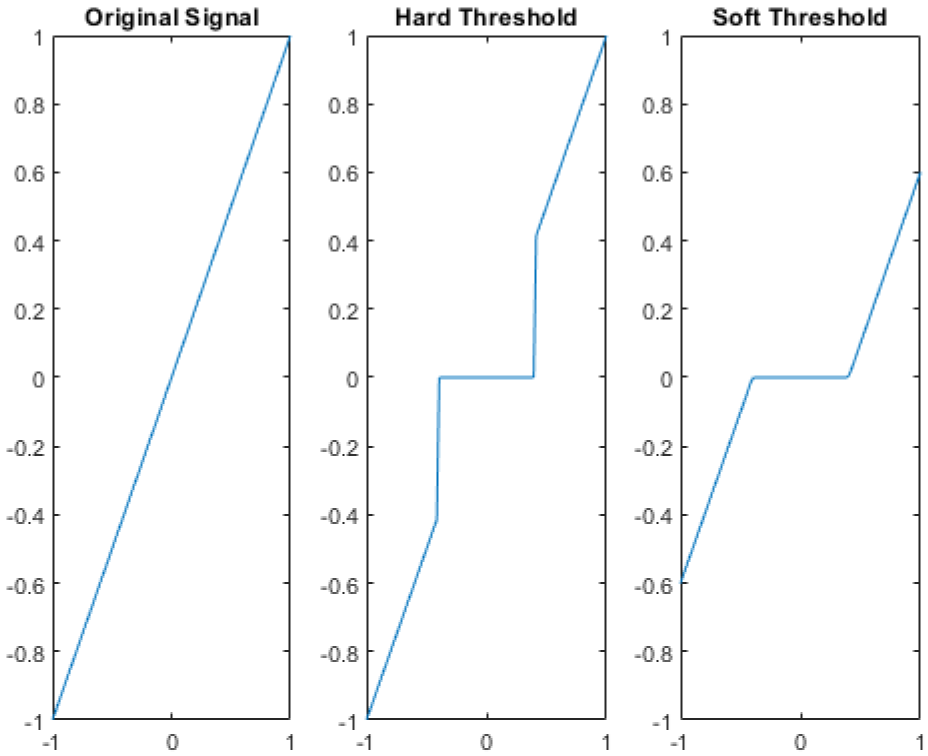
```
y = linspace(-1,1,100);
thr = 0.4;
```

Perform hard and soft thresholding.

```
ythard = wthresh(y,'h',thr);
ytsoft = wthresh(y,'s',thr);
```

Plot the results and compare with the original signal.

```
subplot(1,3,1)
plot(y,y)
ylim([-1 1])
title('Original Signal')
subplot(1,3,2)
plot(y,ythard)
ylim([-1 1])
title('Hard Threshold')
subplot(1,3,3)
plot(y,ytsoft)
ylim([-1 1])
title('Soft Threshold')
```



Input Arguments

X — Input data

real-valued vector or matrix

Input data to threshold, specified as a real-valued vector or matrix.

Data Types: `double`

`sorh` — Type of thresholding

`'s'` | `'h'`

Type of thresholding to perform:

- `'s'` — Soft thresholding
- `'h'` — Hard thresholding

T — Threshold value

positive real number

Threshold value, specified as a positive real number.

Output Arguments

Y — Thresholded data

real-valued vector or matrix

Thresholded data, returned as a real-valued vector or matrix. Y has the same dimensions as X.

Algorithms

If sorh is 's', Y is the soft thresholding of X: $Y = \text{sign}(X) \cdot (|X| - T)_{+}$ where

$$(x)_{+} = \begin{cases} x & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Soft thresholding is wavelet shrinkage.

If sorh is 'h', Y is the hard thresholding of X: $Y = X \cdot \mathbf{1}_{(|X| \ > \ T)}$ where

$$\mathbf{1}_{(|X| \ > \ T)} = \begin{cases} 1 & \text{if} \quad |X| > T \\ 0 & \text{otherwise} \end{cases}$$

Hard thresholding is cruder than soft thresholding.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

wdenoise | wden | wdencmp | wpdencmp

Apps

Wavelet Signal Denoiser

Introduced before R2006a

wthrmngr

Threshold settings manager

Syntax

```
thr = wthrmngr(opt,method,C,L)
thr = wthrmngr(opt,method,C,L,alpha)
thr = wthrmngr(opt,method,C,L,scale)

thr = wthrmngr(opt,method,swtdec,alpha)
thr = wthrmngr(opt,method,swtdec,scale)

thr = wthrmngr(opt,method,wpt)
thr = wthrmngr(opt,'rem_n0',X)
```

Description

`wthrmngr` returns a global threshold or level-dependent thresholds for wavelet-based denoising and compression. The function derives thresholds from the wavelet coefficients in a wavelet decomposition.

The thresholds are used by Wavelet Toolbox denoising and compression tools in the **Wavelet Analyzer** app.

`thr = wthrmngr(opt,method,C,L)` returns the threshold for the `[C,L]` wavelet decomposition of the signal or image to compress or denoise. For signals, `[C,L]` is the output of `wavedec`. For images, `[C,L]` is the output of `wavedec2`.

`thr = wthrmngr(opt,method,C,L,alpha)` returns the `[C,L]` wavelet decomposition threshold using the sparsity parameter `alpha`. For signals, `[C,L]` is the output of `wavedec`. For images, `[C,L]` is the output of `wavedec2`.

To learn more about `alpha,` see `wdcbm` or `wdcbm2` for compression, and `wbmpen` for denoising.

`thr = wthrmngr(opt,method,C,L,scale)` returns the `[C,L]` wavelet decomposition threshold using the type of multiplicative threshold rescaling specified in `scale`. For signals, `[C,L]` is the output of `wavedec`. For images, `[C,L]` is the output of `wavedec2`.

The `'rigrsure'`, `'heursure'`, and `'minimaxi'` denoising methods are only applicable to signals.

To learn more about multiplicative threshold rescaling, see `wden`.

`thr = wthrmngr(opt,method,swtdec,alpha)` returns the level-dependent threshold for the stationary wavelet decomposition, `swtdec`, of the signal or image to denoise. `alpha` specifies the sparsity parameter (see `wbmpen`). For signals, `swtdec` is the output of `swt`. For images, `swtdec` is the output of `swt2`.

Thresholds are derived from a subset of the coefficients in the stationary wavelet decomposition. For more information, see “Coefficient Selection” on page 1-1796.

`thr = wthrmngr(opt,method,swtdec,scale)` returns the level-dependent threshold for the stationary wavelet decomposition using the type of multiplicative threshold rescaling specified in `scale`. For signals, `swtdec` is the output of `swt`. For images, `swtdec` is the output of `swt2`.

Thresholds are derived from a subset of the coefficients in the stationary wavelet decomposition. For more information, see “Coefficient Selection” on page 1-1796.

The `'rigrsure'`, `'heursure'`, and `'minimaxi'` denoising methods apply only to signals.

To learn more about multiplicative threshold rescaling, see `wden`.

`thr = wthrmngr(opt,method,wpt)` returns the global threshold for the wavelet packet decomposition, `wpt`, of the signal or image to compress or denoise.

`thr = wthrmngr(opt,'rem_n0',X)` returns the global threshold to compress the signal or image, `X`, using the specified wavelet option and method `'rem_n0'`.

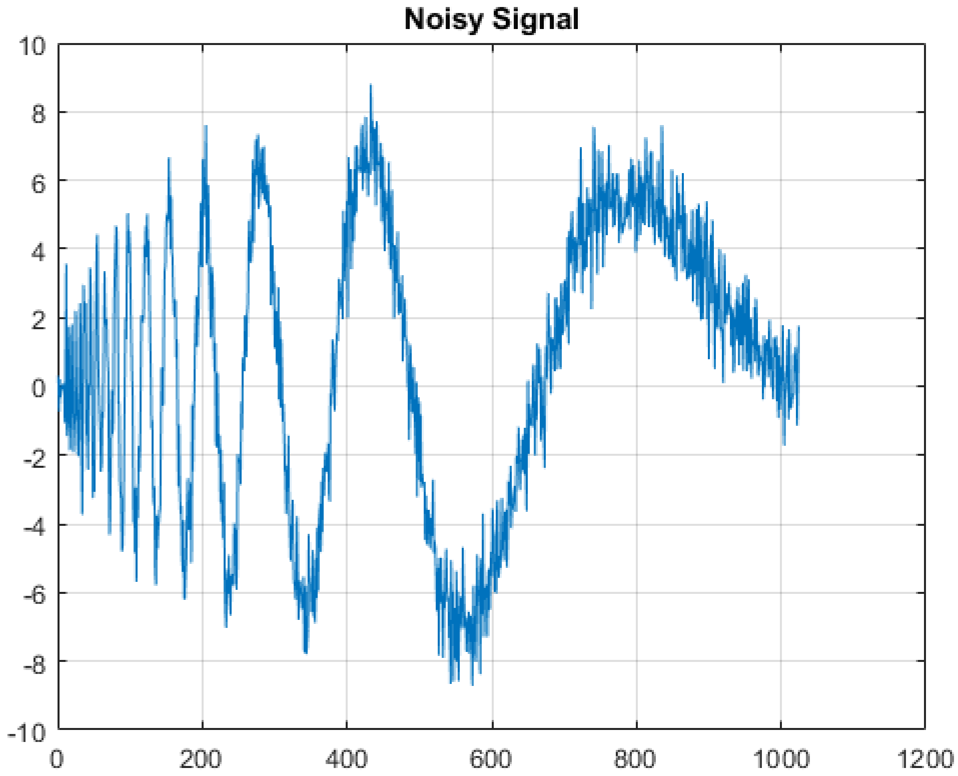
If `opt` is `'dw1dcompGBL'` or `'dw2dcompGBL'`, thresholds are based on the finest-scale wavelet coefficients obtained using the Haar wavelet. If `opt` is `'wp1dcompGBL'` or `'wp2dcompGBL'`, thresholds are based on the finest-scale wavelet packet coefficients obtained using the Haar wavelet.

Examples

Global Threshold — Discrete Wavelet Decomposition

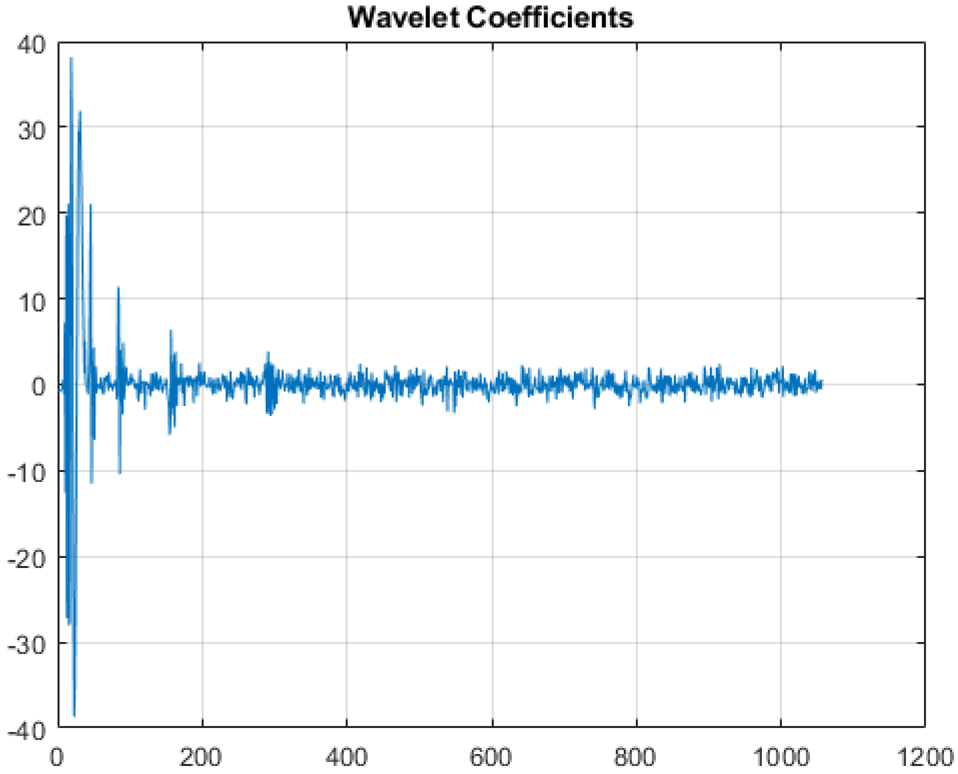
Load and plot a noisy signal.

```
load noisdopp
plot(noisdopp)
grid on
title('Noisy Signal')
```



Generate a level 5 wavelet decomposition of the noisy signal using the order 4 Daubechies wavelet. Plot the coefficients.

```
[c,l] = wavedec(noisdopp,5,'db4');
plot(c)
grid on
title('Wavelet Coefficients')
```

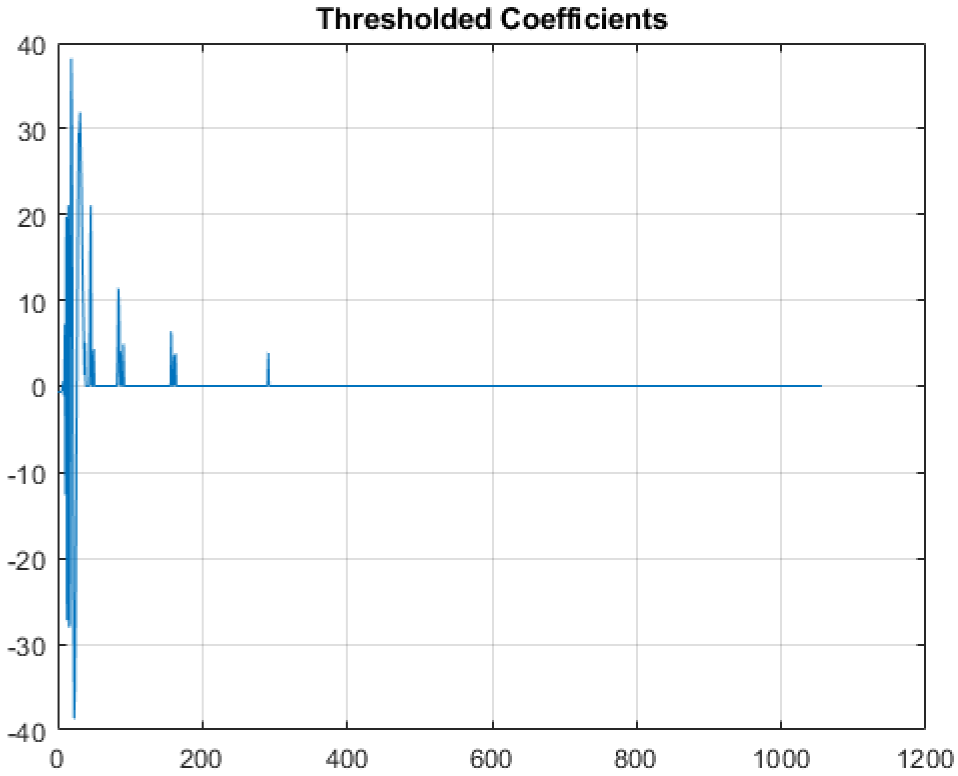


Determine a global threshold for compressing the signal.

```
thr = wthrmngr('dw1dcompGBL','bal_sn',c,l);
```

The index of the first wavelet detail coefficient in `c` is `l(1)+1`. Apply the threshold to all the detail coefficients. Plot the thresholded coefficients. Observe that most of the coefficients have been set to 0.

```
c(l(1)+1:end) = c(l(1)+1:end).*(c(l(1)+1:end)>thr);
plot(c)
grid on
title('Thresholded Coefficients')
```



Reconstruct the signal from the thresholded coefficients. Plot the reconstruction.

```
xrec = waverec(c,l,'db4');
plot(xrec)
grid on
title('Compressed Signal')
```

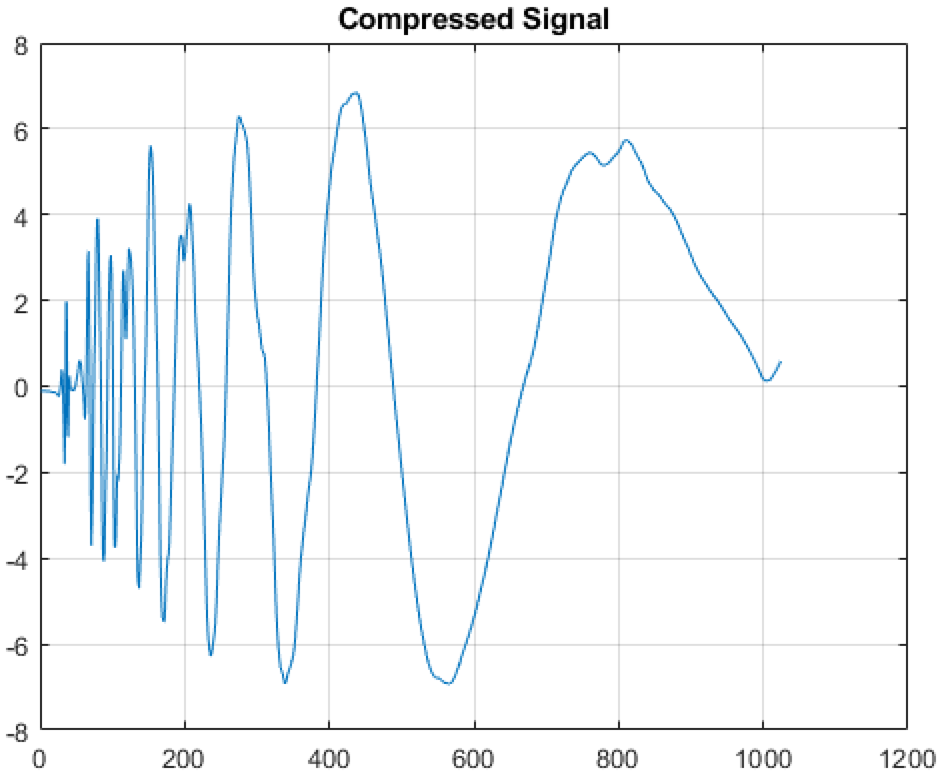


Image Compression — Birgé-Massart Thresholds

Compress an image using the Birgé-Massart strategy.

Load an image and add white Gaussian noise. For purposes of reproducibility, set the random seed to the default value.

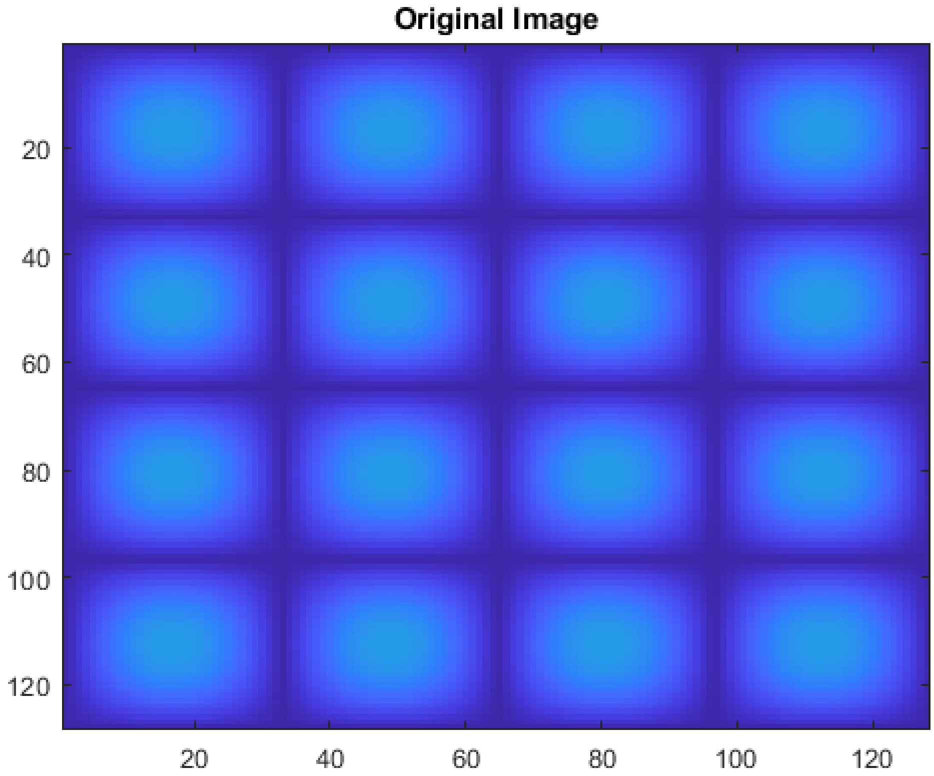
```
rng default
load sinsin
x = X+18*randn(size(X));
```

Obtain the 2-D discrete wavelet transform down to level 3 using the Daubechies least-asymmetric wavelet with 4 vanishing moments. Obtain the compression thresholds using the Birgé-Massart strategy with sparsity parameter, `alpha`, equal to 2.

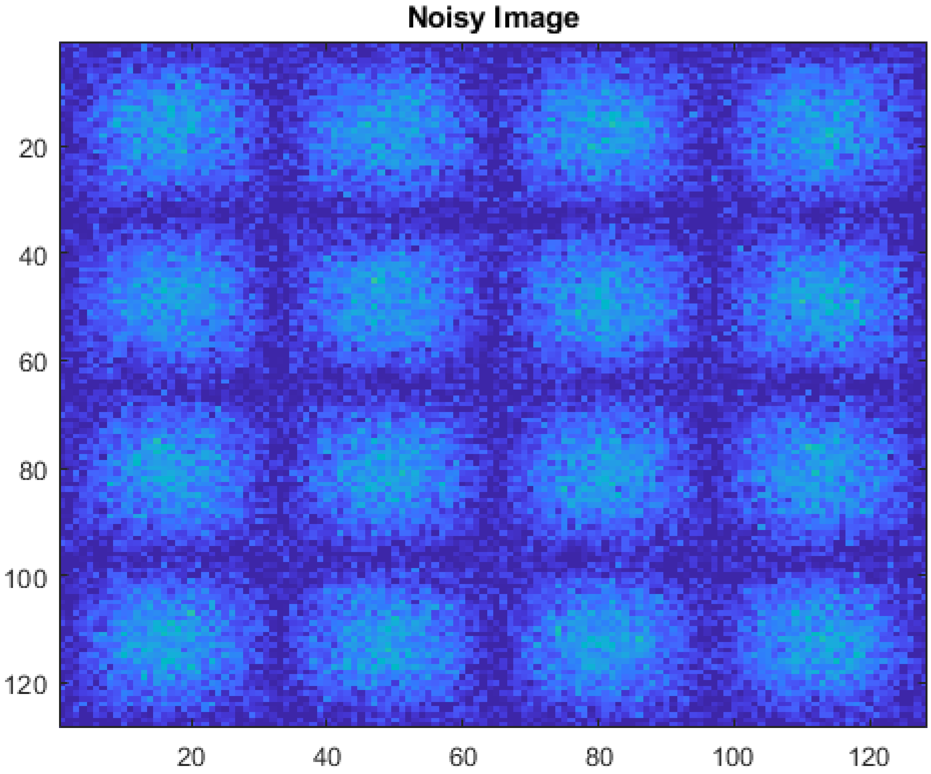
```
[C,L] = wavedec2(x,3,'sym4');
alpha = 2;
THR = wthrmngr('dw2dcompLVL','scarcehi',C,L,alpha);
```

Compress the image and display the result.

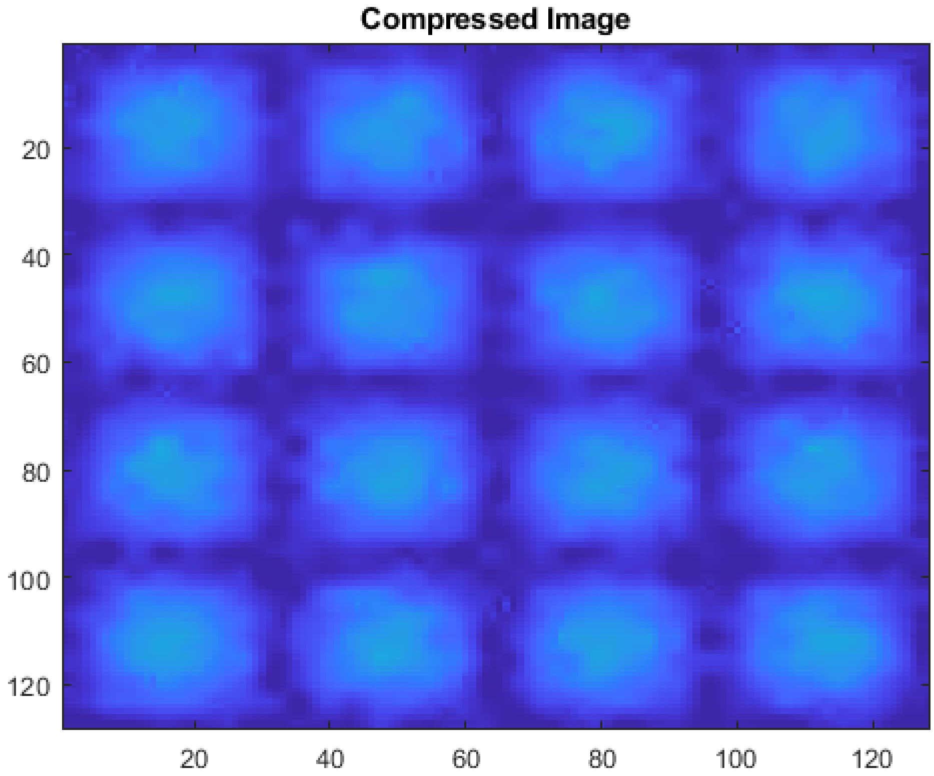
```
xd = wdencmp('lvd',x,'sym4',3,THR,'s');
image(X)
title('Original Image')
```



```
figure
image(x)
title('Noisy Image')
```



```
figure
image(xd)
title('Compressed Image')
```



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 by using the Haar wavelet.

```
load noisbloc
L = 5;
swc = swt(noisbloc,L,'haar');
```

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 every element equal to the universal threshold for the corresponding scale.

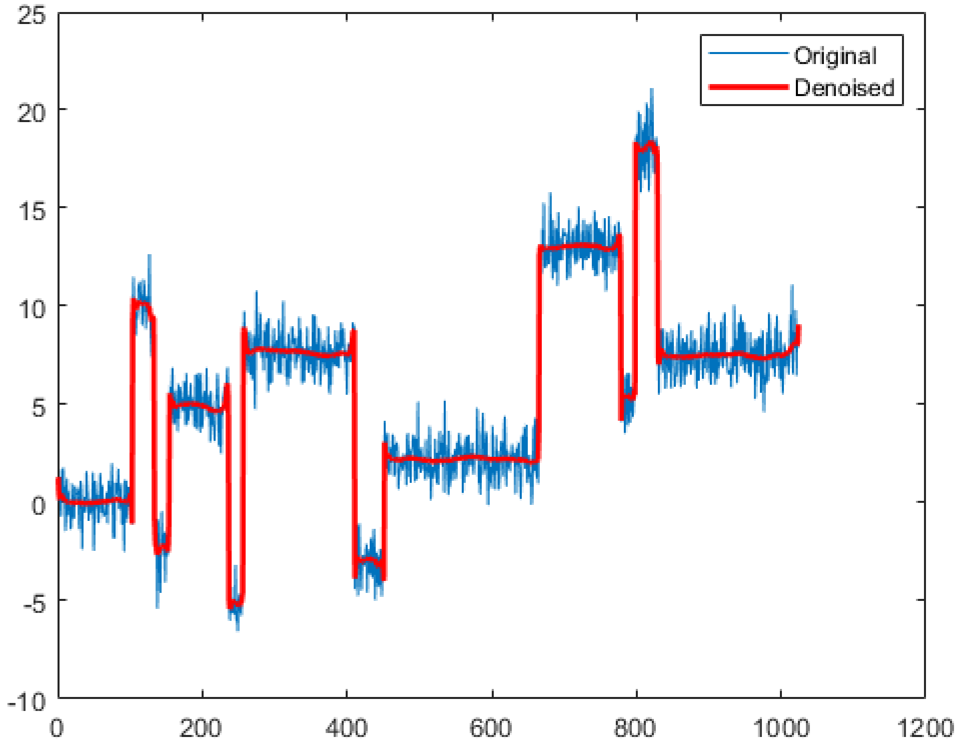
```
swcnew = swc;
ThreshML = wthrmngr('sw1ddenoLVL','sqtwolog',swc,'mln');
```

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,'haar');
plot(noisbloc)
hold on
plot(noisbloc_denoised,'r','linewidth',2)
legend('Original','Denoised')
```

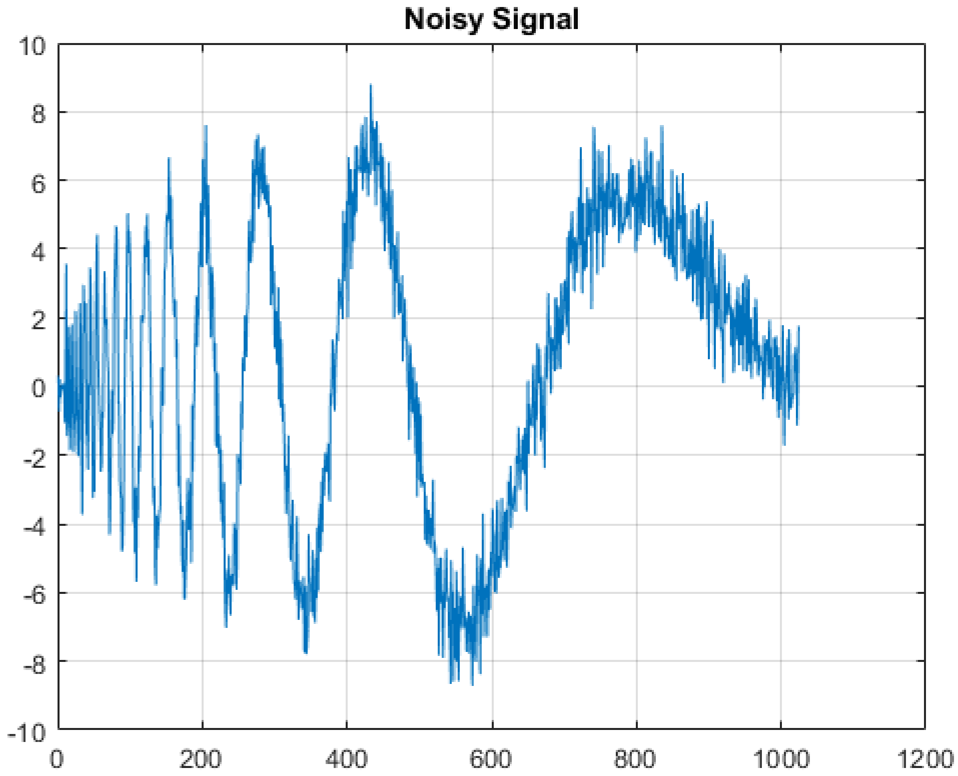


Global Threshold — Wavelet Packet Decomposition

Denoise a noisy signal by applying a global threshold to a wavelet packet decomposition structure.

Load and plot a noisy signal.

```
load noisdopp
plot(noisdopp)
grid on
title('Noisy Signal')
```



Generate a level 3 wavelet packet decomposition of the noisy signal using the order 4 Daubechies wavelet.

```
T = wpdec(noisdopp,3,'db4');
```

Determine a global threshold for denoising the signal.

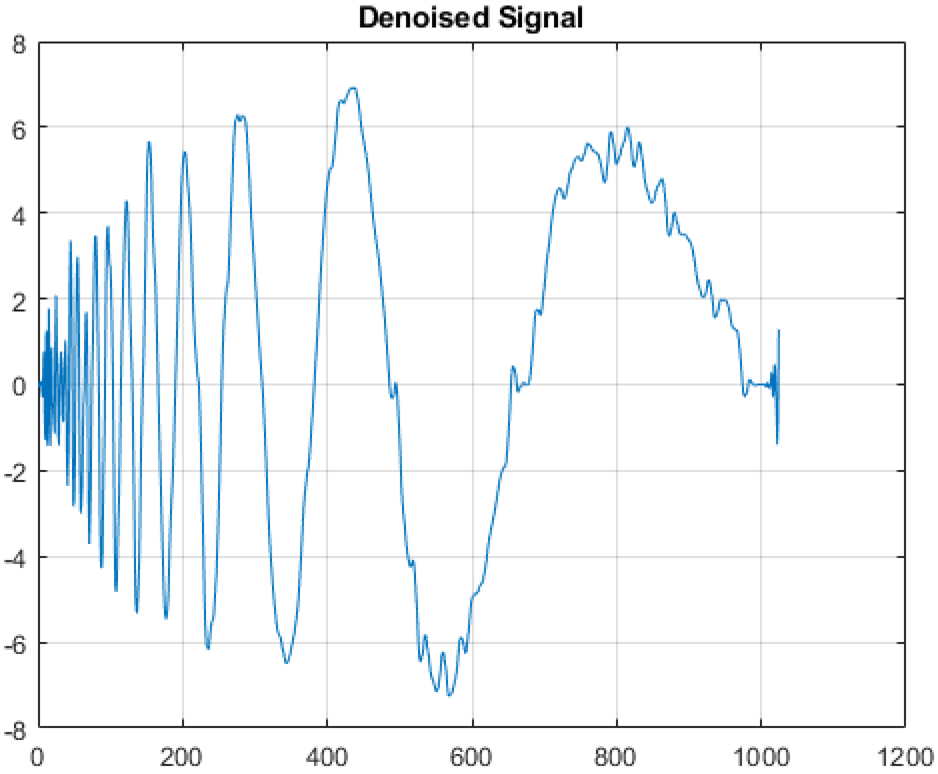
```
thr = wthrmngr('wp1ddenoGBL','sqtwologuwn',T);
```

Obtain the leaves from the wavelet packet decomposition tree T and apply the threshold to the leaves. Use hard thresholding.

```
T1 = T;
sorh = 'h';
cfs = read(T,'data');
cfs = wthresh(cfs,sorh,thr);
T1 = write(T1,'data',cfs);
```

Reconstruct the denoised signal from the thresholded coefficients. Plot the reconstruction.

```
xrec = wprec(T1);
plot(xrec)
grid on
title('Denoised Signal')
```



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 by using the Haar wavelet.

```
load noisbloc
L = 5;
swc = swt(noisbloc,L,'haar');
```

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.

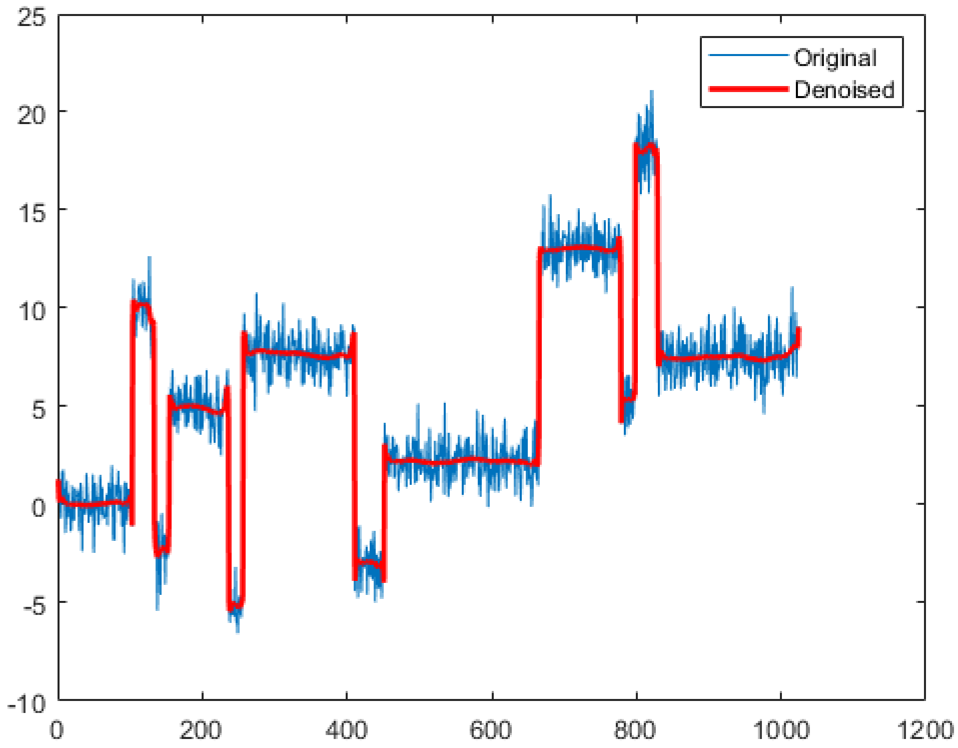
```
swcnew = swc;
ThreshSL = mean(wthrmngr('sw1ddenoLVL','sqtwolog',swc,'sln'));
```

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,'haar');
plot(noisbloc)
hold on
plot(noisbloc_denoised,'r','linewidth',2)
legend('Original','Denoised')
```



Input Arguments

opt — Type and dimension of compression or denoising
'dw1dcompGBL' | 'dw1dcompLVL' | 'dw1ddenoLVL' | 'sw1ddenoLVL' | 'dw2dcompGBL' | 'dw2dcompLVL' | ...

Type and dimension of compression or denoising, specified as one of the values listed in the tables that follow. `wthrmngr` returns thresholds appropriate for the option you specify.

With a discrete wavelet or wavelet packet decomposition of the data, you can compress or denoise that data. With a stationary wavelet decomposition of the data, you can only denoise the data.

For an explanation of which coefficients are used to determine the thresholds, see “Coefficient Selection” on page 1-1796.

1-D Discrete Wavelet Decomposition Options

In these options, X is the signal, the wavelet coefficients are in the vector C, and the lengths of the coefficient vectors are in L. The argument alpha is the sparsity parameter, and scale defines the multiplicative threshold rescaling.

For additional information regarding the wavelet decomposition, see wavedec. To learn more about alpha and scale, see wdcbm and wden respectively.

opt	Description	Valid Syntaxes
'dw1dcompGBL'	1-D compression using a global threshold	• thr = wthrmngr('dw1dcompGBL','rem_n0',X) • thr = wthrmngr('dw1dcompGBL','bal_sn',C,L)
'dw1dcompLVL'	1-D compression using level-dependent thresholds	• thr = wthrmngr('dw1dcompLVL','scarcehi',C,L,alpha), where 2.5 < alpha < 10 • thr = wthrmngr('dw1dcompLVL','scarceme',C,L,alpha), where 1.5 < alpha < 2.5 • thr = wthrmngr('dw1dcompLVL','scarcelo',C,L,alpha), where 1 < alpha < 2
'dw1ddenoLVL'	1-D denoising using level-dependent thresholds	• thr = wthrmngr('dw1ddenoLVL','sqtwolog',C,L,scale) • thr = wthrmngr('dw1ddenoLVL','rigrsure',C,L,scale) • thr = wthrmngr('dw1ddenoLVL','heursure',C,L,scale) • thr = wthrmngr('dw1ddenoLVL','minimaxi',C,L,scale) • thr = wthrmngr('dw1ddenoLVL','penalhi',C,L,alpha), where 2.5 < alpha < 10 • thr = wthrmngr('dw1ddenoLVL','penalme',C,L,alpha), where 1.5 < alpha < 2.5 • thr = wthrmngr('dw1ddenoLVL','penallo',C,L,alpha), where 1 < alpha < 2

2-D Discrete Wavelet Decomposition Options

In these options, `X` is the data, the wavelet coefficients are in the vector `C`, and the size of the coefficient matrices are in `L`. The argument `alpha` is the sparsity parameter, and `scale` defines the multiplicative threshold rescaling.

For additional information regarding the wavelet decomposition, see `wavedec2`. To learn more about `alpha` and `scale`, see `wdcbm2` and `wden` respectively.

opt	Description	Valid Syntaxes
`'dw2dcompGBL'`	2-D compression using a global threshold	• `thr = wthrmngr('dw2dcompGBL','rem_n0',X)` • `thr = wthrmngr('dw2dcompGBL','bal_sn',C,L)` • `thr = wthrmngr('dw2dcompGBL','sqrtbal_sn',C,L)`
`'dw2dcompLVL'`	2-D compression using level-dependent thresholds	• `thr = wthrmngr('dw2dcompLVL','scarcehi',C,L,alpha)`, where 2.5 < `alpha` < 10 • `thr = wthrmngr('dw2dcompLVL','scarceme',C,L,alpha)`, where 1.5 < `alpha` < 2.5 • `thr = wthrmngr('dw2dcompLVL','scarcelo',C,L,alpha)`, where 1 < `alpha` < 2
`'dw2ddenoLVL'`	2-D denoising using level-dependent thresholds	• `thr = wthrmngr('dw2ddenoLVL','sqrtbal_sn',C,L)` • `thr = wthrmngr('dw2ddenoLVL','penalhi',C,L,alpha)`, where 2.5 < `alpha` < 10 • `thr = wthrmngr('dw2ddenoLVL','penalme,C,L,alpha)`, where 1.5 < `alpha` < 2.5 • `thr = wthrmngr('dw2ddenoLVL','penallo,C,L,alpha)`, where 1 < `alpha` < 2 • `thr = wthrmngr('dw2ddenoLVL','sqtwolog',C,L,scale)`

1-D Wavelet Packet Decomposition Options

In these options, `X` is the signal and `wpt` is the wavelet packet decomposition structure of the signal.

For additional information regarding the wavelet packet decomposition, see `wpdec`.

opt	Description	Valid Syntaxes
'wp1dcompGBL'	1-D compression using a global threshold	• `thr = wthrmngr('wp1dcompGBL','rem_n0',X)` • `thr = wthrmngr('wp1dcompGBL','bal_sn',wpt)`
'wp1ddenoGBL'	1-D denoising using a global threshold	• `thr = wthrmngr('wp1ddenoGBL','sqtwologuwn',wpt)` • `thr = wthrmngr('wp1ddenoGBL','sqtwologswn',wpt)` • `thr = wthrmngr('wp1ddenoGBL','bal_sn',wpt)` • `thr = wthrmngr('wp1ddenoGBL','penalhi',wpt)` The `wpbmpen` function is used with the tuning parameter ALPHA = 6.25. • `thr = wthrmngr('wp1ddenoGBL','penalme',wpt)` The `wpbmpen` function is used with the tuning parameter ALPHA = 2. • `thr = wthrmngr('wp1ddenoGBL','penallo',wpt)` The `wpbmpen` function is used with the tuning parameter ALPHA = 1.5.

2-D Wavelet Packet Decomposition Options

In these options, `X` is the data and `wpt` is the wavelet packet decomposition structure of the data.

For additional information regarding the wavelet packet decomposition, see `wpdec2`.

opt	Description	Valid Syntaxes
'wp2dcompGBL'	2-D compression using a global threshold	• `thr = wthrmngr('wp2dcompGBL','rem_n0',X)` • `thr = wthrmngr('wp2dcompGBL','bal_sn',wpt)` • `thr = wthrmngr('wp2dcompGBL','sqrtbal_sn',wpt)`

opt	Description	Valid Syntaxes
`'wp2ddenoGBL'`	2-D denoising using a global threshold	• `thr = wthrmngr('wp2ddenoGBL','sqtwologuwn',wpt)` • `thr = wthrmngr('wp2ddenoGBL','sqtwologswn',wpt)` • `thr = wthrmngr('wp2ddenoGBL','sqrtbal_sn',wpt)` • `thr = wthrmngr('wp2ddenoGBL','penalhi',wpt)` The `wpbmpen` function is used with the tuning parameter `ALPHA` = 6.25. • `thr = wthrmngr('wp2ddenoGBL','penalme',wpt)` The `wpbmpen` function is used with the tuning parameter `ALPHA` = 2. • `thr = wthrmngr('wp2ddenoGBL','penallo',wpt)` The `wpbmpen` function is used with the tuning parameter `ALPHA` = 1.5.

1-D Stationary Wavelet Decomposition Options

Denoising using level-dependent thresholds is the only option available for a 1-D stationary wavelet decomposition, `swtdec`. In this option, `alpha` is a sparsity parameter and `scale` defines the multiplicative threshold rescaling.

For more information regarding the stationary wavelet decomposition, see `swt`. To learn more about `alpha` and `scale`, see `wbmpen` and `wden` respectively.

opt	Valid Syntaxes
'sw1ddenoLVL'	• thr = wthrmngr('sw1ddenoLVL','sqtwolog',swtdec,scale) • thr = wthrmngr('sw1ddenoLVL','rigrsure',swtdec,scale) • thr = wthrmngr('sw1ddenoLVL','heursure',swtdec,scale) • thr = wthrmngr('sw1ddenoLVL','minimaxi',swtdec,scale) • thr = wthrmngr('sw1ddenoLVL','penalhi',swtdec,alpha), where 2.5 < alpha < 10 • thr = wthrmngr('sw1ddenoLVL','penalme',swtdec,alpha), where 1.5 < alpha < 2.6 • thr = wthrmngr('sw1ddenoLVL','penallo',swtdec,alpha), where 1 < alpha < 2

Thresholds are based on a subset of the coefficients in the stationary wavelet decomposition. See “Coefficient Selection” on page 1-1796 for additional information.

2-D Stationary Wavelet Decomposition Options

Denoising using level-dependent thresholds is the only option available for a 2-D stationary wavelet decomposition, swtdec. In this option, alpha is a sparsity parameter and scale defines the multiplicative threshold rescaling.

For more information regarding the stationary wavelet decomposition, see swt2. To learn more about alpha and scale, see wbmpen and wden respectively.

opt	Valid Syntaxes
'sw2ddenoLVL'	• thr = wthrmngr('sw2ddenoLVL','sqrtbal_sn',swtdec) • thr = wthrmngr('sw2ddenoLVL','penalhi',swtdec,alpha) where 2.5 < alpha < 10 • thr = wthrmngr('sw2ddenoLVL','penalme',swtdec,alpha) where 1.5 < alpha < 2.5 • thr = wthrmngr('sw2ddenoLVL','penallo',swtdec,alpha) where 1 < alpha < 2 • thr = wthrmngr('sw2ddenoLVL','sqtwolog',swtdec,scale)

Thresholds are based on a subset of the coefficients in the stationary wavelet decomposition. See “Coefficient Selection” on page 1-1796 for additional information.

method — Thresholding method

'scarcehi' | 'scarceme' | 'scarcelo' | 'sqtwolog' | 'sqtwologuwn' | 'sqtwologswn' | ...

Thresholding method, specified as one of the values listed here.

method	Description
'scarcehi'	Uses Birgé-Massart strategy on page 1-1796 for determining thresholds.
'scarceme'	Uses Birgé-Massart strategy for determining thresholds.
'scarcelo'	Uses Birgé-Massart strategy for determining thresholds.
'sqtwolog'	Uses fixed-form universal threshold. See 'sqtwolog' option in wden.
'sqtwologuwn'	Uses fixed-form universal threshold. See 'sqtwolog' option in wden when used with 'sln' option.
'sqtwologswn'	Uses fixed-form universal threshold. See 'sqtwolog' option in wden when used with 'mln' option.
'rigsure'	Uses soft threshold estimator rule based on Stein's Unbiased Estimate of Risk. See 'SURE' option in wdenoise.
'heursure'	Uses mixture of 'rigsure' and 'sqtwolog'. See 'heursure' option in wden.
'minimaxi'	Uses a fixed threshold chosen which yields minimax performance. See 'Minimax' option in wdenoise.
'penalhi'	Used to define Birgé-Massart strategy on page 1-1796 for determining thresholds.
'penalme'	Used to define Birgé-Massart strategy for determining thresholds.
'penallo'	Used to define Birgé-Massart strategy for determining thresholds.
'rem_n0'	Returns a threshold close to 0. A typical THR value is median(abs(coefficients)).
'bal_sn'	Returns a threshold such that the percentages of retained energy and number of zeros are the same.
'sqrtbal_sn'	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.

Data Types: char

X — Input data

real-valued vector | real-valued matrix

Input data, specified as a real-valued vector or real-valued matrix.

Data Types: double

C — Wavelet expansion coefficients

real-valued vector

Wavelet expansion coefficients of the data to be compressed or denoised, specified as a real-valued vector. If the data is one-dimensional, C is the output of wavedec. If the data is two-dimensional, C is the output of wavedec2.

Example: `[C,L] = wavedec(randn(1,1024),3,'db4')`

Data Types: `double`

`L` — Size of wavelet expansion coefficients

vector of positive integers | matrix of positive integers

Size of wavelet expansion coefficients of the signal or image to be compressed or denoised, specified as a vector or matrix of positive integers.

For signals, `L` is the output of `wavedec`. For images, `L` is the output of `wavedec2`.

Example: `[C,L] = wavedec(randn(1,1024),3,'db4')`

Data Types: `double`

`alpha` — Sparsity parameter

positive scalar

Sparsity parameter used for compressing or denoising data, specified as a positive scalar greater than 1 and less than 10. See `wdcbm`, `wdcbm2`, and `wbmpen` for additional information.

Data Types: `double`

`scale` — Multiplicative threshold rescaling

`'one'` | `'sln'` | `'mln'`

Multiplicative threshold rescaling, specified as one of the following:

- `'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

For more information, see `wden`.

`swtdec` — Stationary wavelet decomposition structure

real-valued matrix

Stationary wavelet decomposition structure of data to be compressed or denoised, specified as a real-valued matrix. If the data is one-dimensional, `swtdec` is the output of `swt`. If the data is two-dimensional, `swtdec` is the output of `swt2`.

Example: `swtdec = swt2(randn(256),3,'db1')`

Data Types: `double`

`wpt` — Wavelet packet decomposition structure

wavelet packet object structure

Wavelet packet decomposition structure of the data to be compressed or denoised. If the data is one-dimensional, `wpt` is the output of `wpdec`. If the data is two-dimensional, `wpt` is the output of `wpdec2`.

Example: `wpt = wpdec(randn(1,1024),5,'db1')`

Output Arguments

thr — Threshold
real-valued scalar | real-valued vector | real-valued matrix

Threshold, returned as a real-valued scalar for global thresholds, or a real-valued vector or matrix for level-dependent thresholds.

Data Types: double

Tips

- To denoise 1-D signals, consider using the **Wavelet Signal Denoiser**. The app visualizes and denoises real-valued 1-D signals using default parameters. You can also compare results. In addition, you can also recreate the denoised signal in your workspace by generating a MATLAB script, which uses the wdenoise function.

Algorithms

Coefficient Selection

A critically sampled wavelet or wavelet packet decomposition involves decimating coefficients by a factor of 2 at each stage of the decomposition. Decimation does not occur in the nondecimated stationary wavelet decomposition.

wthrmngr derives denoising and compression thresholds from the wavelet coefficients. For a critically sampled wavelet or wavelet packet decomposition, the option and method determine whether all wavelet coefficients or only the finest scale coefficients are used.

For the stationary wavelet decomposition, wthrmngr always uses a subset of the wavelet coefficients. When computing the denoising thresholds of an N-level stationary wavelet decomposition, the algorithm first subsamples the wavelet coefficients at level k by a factor of 2^k, for k = 1,…,N. The algorithm uses this subset of coefficients to determine the thresholds. Most of the coefficients in the stationary wavelet decomposition are not considered.

Birgé-Massart Strategy

The Birgé-Massart strategy for determining thresholds depends on several different parameters. You specify the wavelet decomposition and a thresholding method. You can also specify a sparsity parameter, alpha, or a specific multiplicative threshold rescaling, scale. Based on your inputs, wthrmngr derives the necessary Birgé-Massart parameters. The parameters depend on the dimension of the signal, and the total number, N, of coefficients at the coarsest scale of wavelet decomposition.

If the thresholding method is 'scarcehi', 'scarceme', or 'scarcelo', the wthrmngr executes either wdcbm or wdcbm2. If the thresholding method is 'penalhi', 'penalme', or 'penallo', then wthrmngr executes either wbmpen or wpbmpen.

Thresholding Method	Description
'scarcehi'	• If the signal is 1-D, then wdcbm is used with input argument M = N. • If the signal is 2-D, then wdcbm2 is used with M = 4*N.

Thresholding Method	Description
'scarceme'	• If the signal is 1-D, then wdcbm is used with input argument M = 3*N/2. • If the signal is 2-D, then wdcbm2 is used with input argument with M = 16*N/3.
'scarcelo'	• If the signal is 1-D, then wdcbm is used with input argument M = 2 N. • If the signal is 2-D, then wdcbm2 is used with input argument M = 32*N/3.
'penalhi'	• If the input is a wavelet decomposition, then wbmpen is used with ALPHA = 5*(3*alpha+1)/8. • If the input is a wavelet packet decomposition, then wpbmpen is used ALPHA = 6.25.
'penalme'	• If the input is a wavelet decomposition, then wbmpen is used with ALPHA = (alpha+5)/8. • If the input is a wavelet packet decomposition, then wpbmpen is used ALPHA = 2.
'penallo'	• If the input is a wavelet decomposition, then wbmpen is used with ALPHA = (alpha+3)/4. • If the input is a wavelet packet decomposition, then wpbmpen is used ALPHA = 1.5.

References

[1] Birgé, L., and P. Massart. “From Model Selection to Adaptive Estimation.” *Festschrift for Lucien Le Cam: Research Papers in Probability and Statistics* (E. Torgersen, D. Pollard, and G. Yang, eds.). New York: Springer-Verlag, 1997, pp. 55–88.

See Also

Apps
Wavelet Signal Denoiser

Functions
wdenoise | wbmpen | wdcbm2 | wdcbm | wpbmpen

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

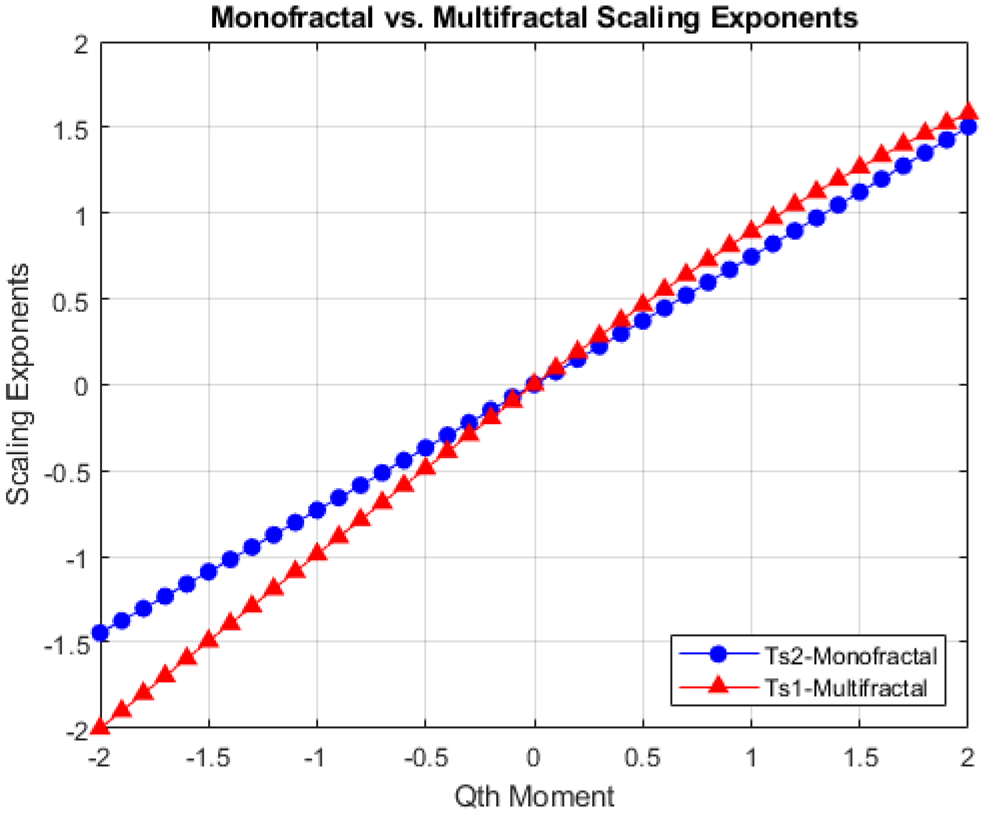
Confirm that for a monofractal signal, the scaling exponents are a linear function of the moments. For multifractal signals, the exponents are a nonlinear function of the moments.

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 exponent 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
```

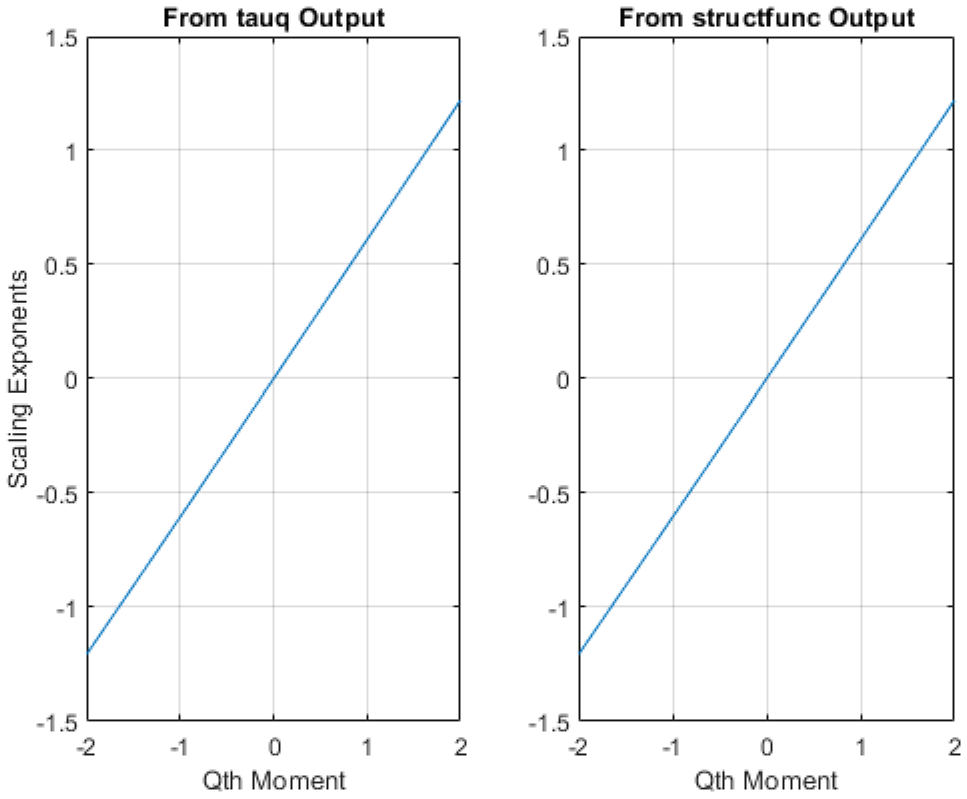
```
hexp = 0.6072
```

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.

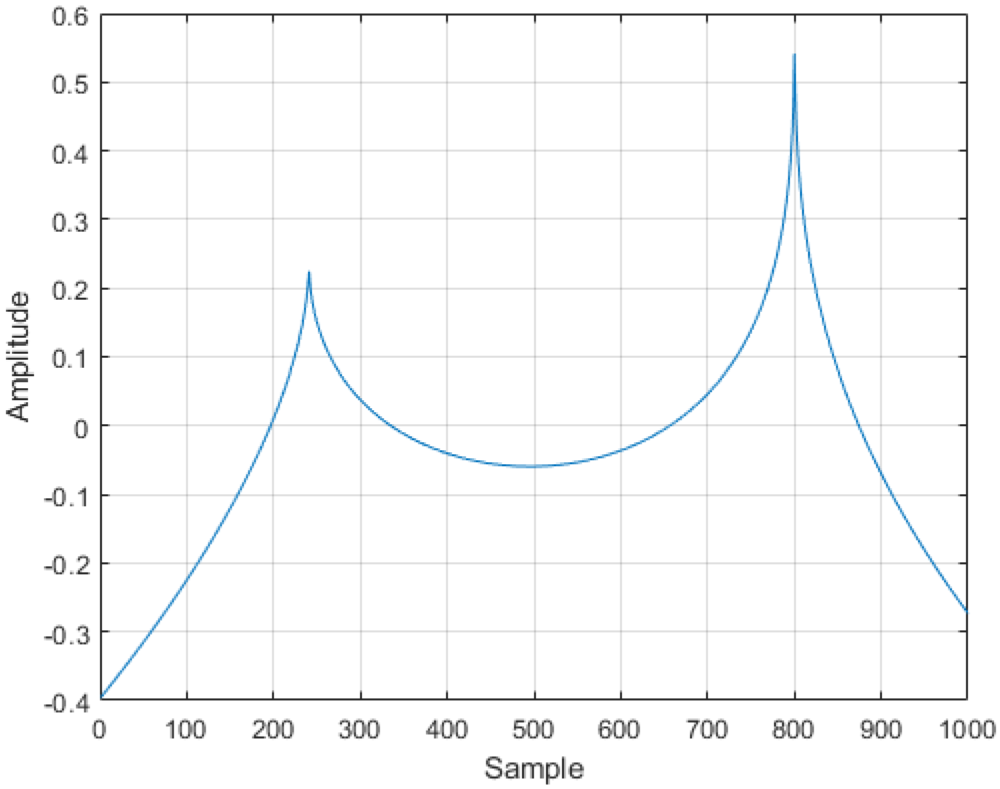
Local Holder Exponents for Cusp Signal and Delta Functions

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')
```

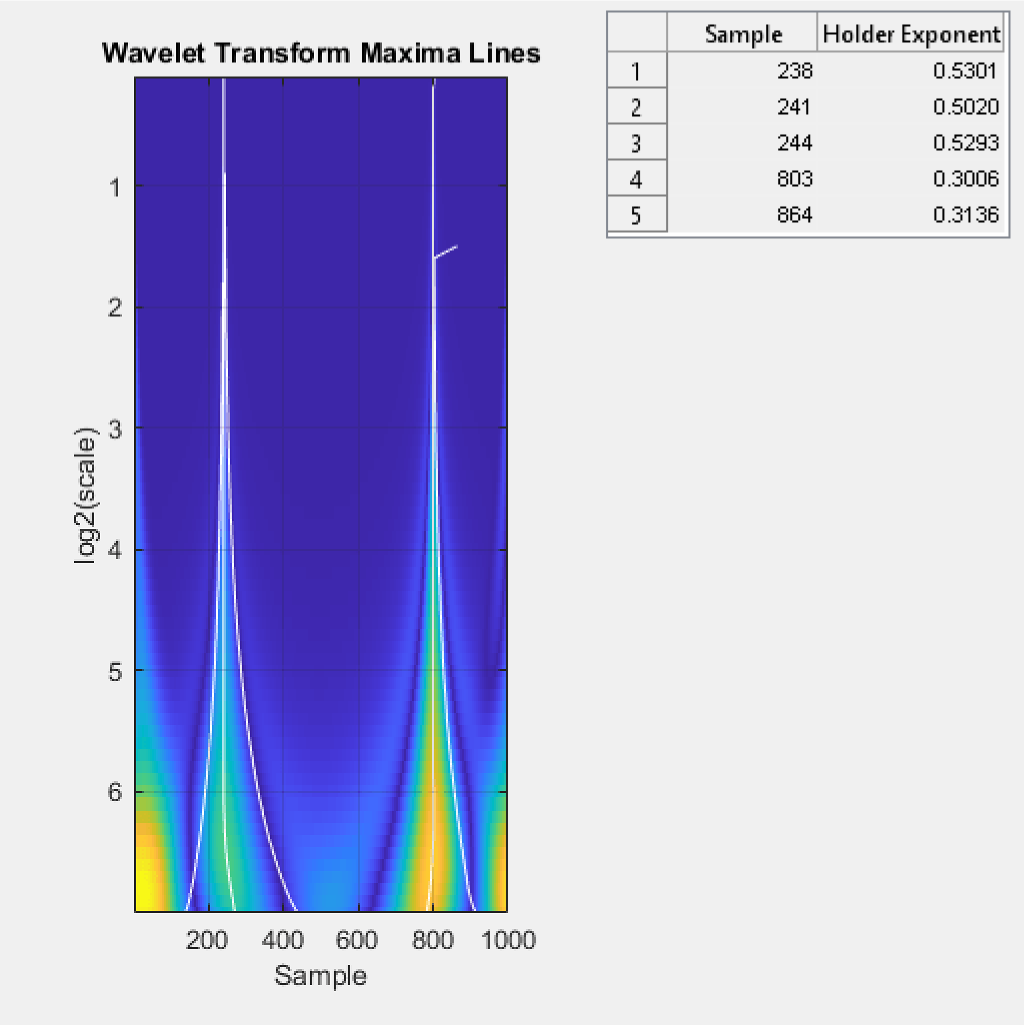


The equation for this cusp signal specifies a Holder exponent of 0.5 at sample 241 and a Holder exponent of 0.3 at sample 803.

```
-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');
```



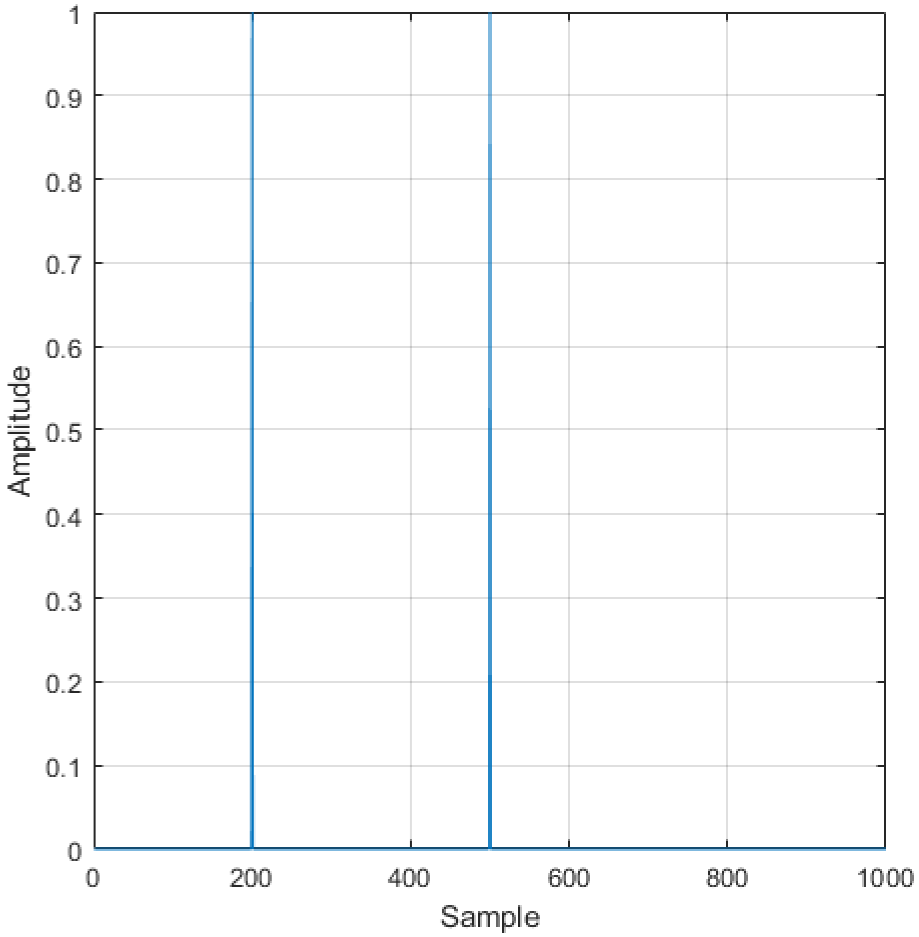
	Sample	Holder Exponent
1	238	0.5301
2	241	0.5020
3	244	0.5293
4	803	0.3006
5	864	0.3136

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

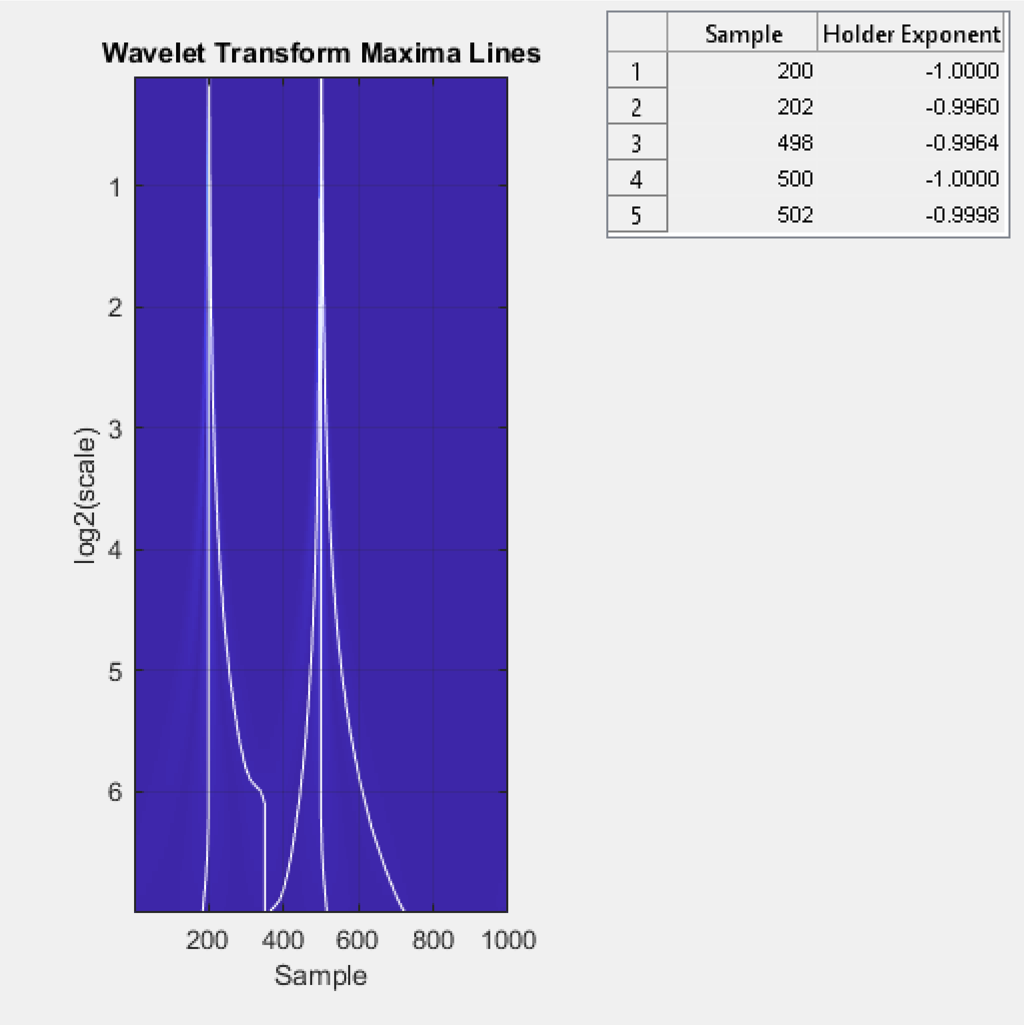
Create and plot two delta functions.

```
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 exponent of -1.

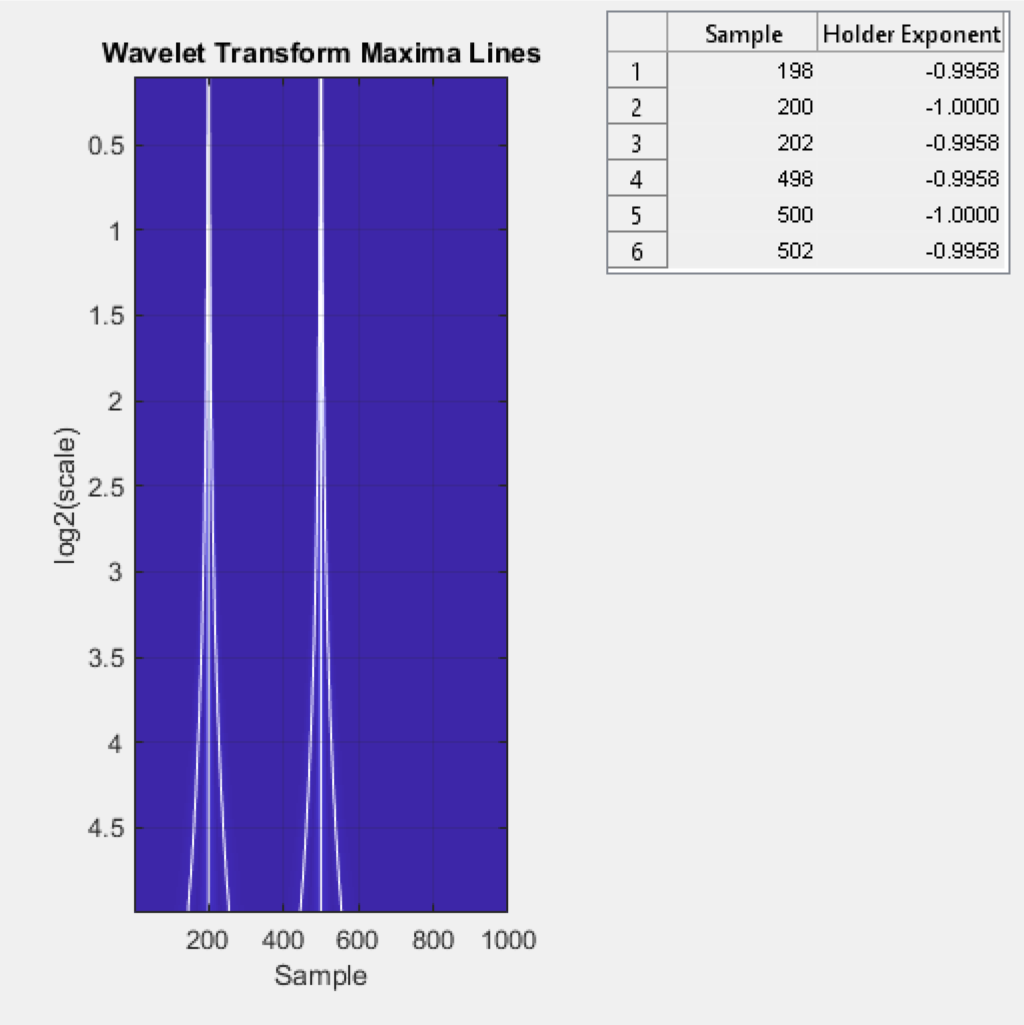
```
wtmm(x,'ScalingExponent','local');
```



	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 pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

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

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

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

$$|Wf(s_0, x)| \le |Wf(s_0, x_0)|$$

. 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

```
[chgpts,kopt,est] = wvarchg(Y)
[ ___ ] = wvarchg(Y,K)
[ ___ ] = wvarchg(Y,K,D)
```

Description

`[chgpts,kopt,est] = wvarchg(Y)` computes estimated variation change points for the signal `Y` for six change points, where the minimum delay between two change points is 10.

`[ ___ ] = wvarchg(Y,K)` computes estimated variation change points for `j` change points, where `j = 0, 1, 2, …, K`, and the minimum delay between two change points is 10.

`[ ___ ] = wvarchg(Y,K,D)` computes estimated variation change points where the minimum delay between two change points is `D`.

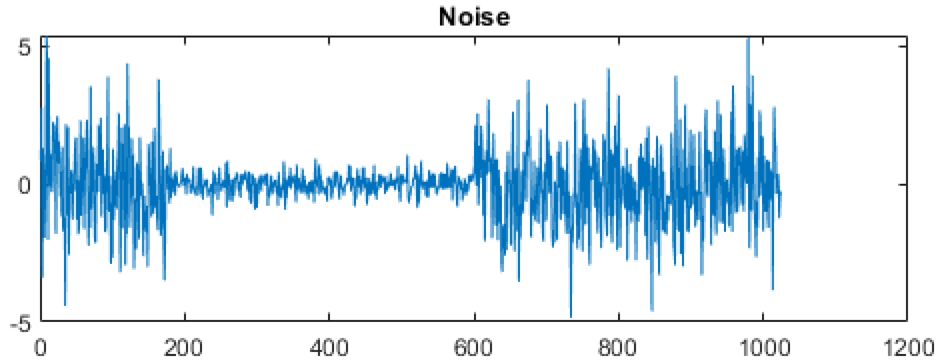
- `wvarchg(Y,6,10)` is equivalent to `wvarchg(Y)`.
- `wvarchg(Y,K,10)` is equivalent to `wvarchg(Y,K)`.

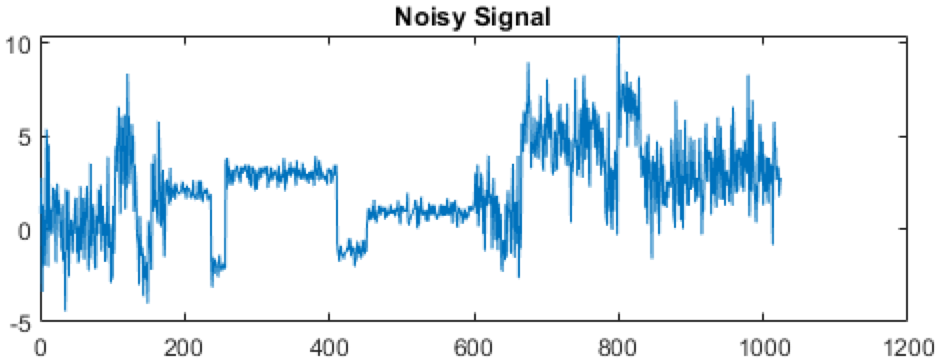
Examples

Detect Variance Change Points

For reproducibility, set the random seed to the default value. Load the `blocks` wavelet test signal. Add white noise with two variance change points located at indices 180 and 600. Plot the noise and the noisy signal.

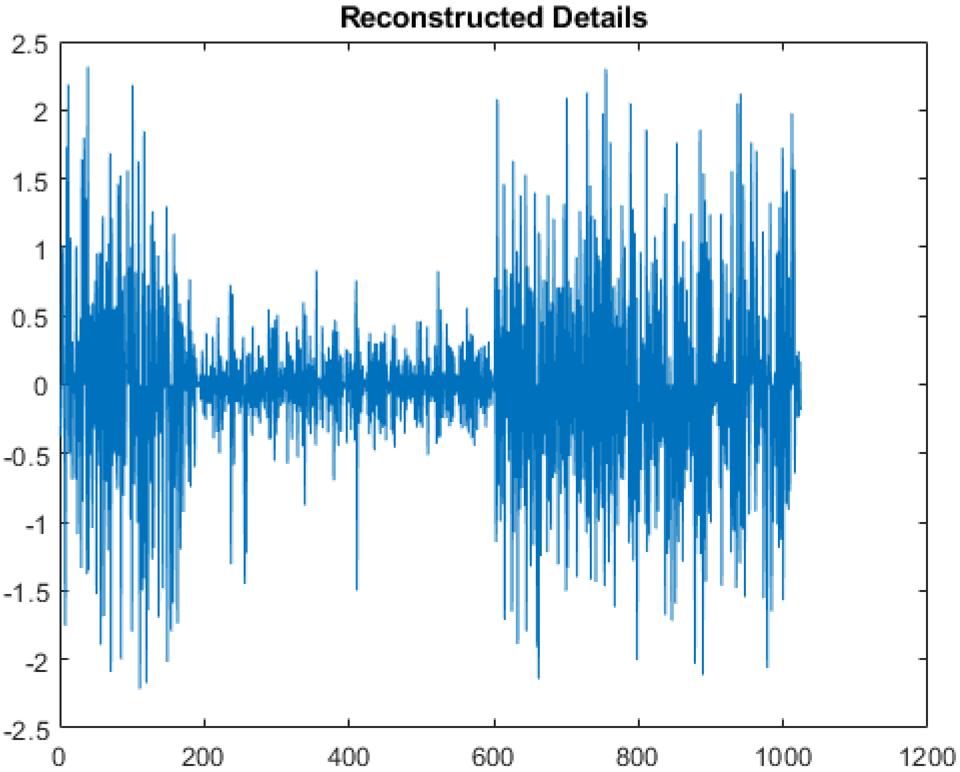
```
rng default
x = wnoise(1,10);
cp1 = 180;
cp2 = 600;
bb = 1.5*randn(1,length(x));
seg1 = bb(1:cp1);
seg2 = bb(cp1+1:cp2)/4;
seg3 = bb(cp2+1:end);
wn = [seg1 seg2 seg3];
x = x+wn;
subplot(2,1,1)
plot(wn)
title('Noise')
subplot(2,1,2)
plot(x)
title('Noisy Signal')
```





Use the db3 wavelet and do a level-1 wavelet decomposition of the signal. Reconstruct the detail coefficients. Replace the top 2% of values with the mean value of the wavelet coefficients to remove most of the signal. Plot the values.

```
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);
figure
plot(det)
title('Reconstructed Details')
```



Estimate the variance change points using the wavelet coefficients.

```
[pts_Opt,kopt,t_est] = wvarchg(det,5);
fprintf('The estimated change points are %d and %d.',pts_Opt)
```

```
The estimated change points are 181 and 601.
```

Input Arguments

Y — Input signal

real-valued vector

Input signal, specified as a real-valued vector. The input signal Y should have zero mean.

Data Types: double

K — Number of change points

6 (default) | positive integer

Number of change points, specified as an integer. K satisfies the inequalities 1 < K ≪ length(Y).

Data Types: double

D — Minimum delay

10 (default) | positive integer

Number of change points, specified as an integer. D satisfies the inequalities 1 ≤ D ≪ length(Y).

Data Types: `double`

Output Arguments

`chgpts` — Estimated variance change points

vector

Estimated variance change points, returned as a vector. `chgpts` is the empty vector `[]` when no change points are found.

`kopt` — Proposed number of change points

nonnegative integer

Proposed number of change points, returned as a nonnegative integer in the interval [0, `k`].

`est` — Instants of the variation change points

real-valued matrix

Instants of the variation change points, returned as a real-valued matrix. For `1 ≤ k ≤ K`, `est(k+1,1:k)` contains the `k` instants of the variance change points. If `kopt > 0`, then `chgpts = est(kopt+1,1:kopt)`, else `chgpts = []`.

References

[1] Lavielle, M. "Detection of multiple changes in a sequence of dependent variables." *Stochastic Processes and their Applications*. Vol. 83, Number 1, 1999, pp. 79–102.

See Also

`cmddenoise`

Topics

“Scale-Localized Volatility and Correlation”

Introduced before R2006a

wvd

Wigner-Ville distribution and smoothed pseudo Wigner-Ville distribution

Syntax

```
d = wvd(x)
d = wvd(x,fs)
d = wvd(x,ts)

d = wvd( ___ ,'smoothedPseudo')
d = wvd( ___ ,'smoothedPseudo',twin,fwin)
d = wvd( ___ ,'smoothedPseudo',Name,Value)

d = wvd( ___ ,'MinThreshold',thresh)

[d,f,t] = wvd( ___ )

wvd( ___ )
```

Description

`d = wvd(x)` returns the Wigner-Ville distribution of `x`.

`d = wvd(x,fs)` returns the Wigner-Ville distribution when `x` is sampled at a rate `fs`.

`d = wvd(x,ts)` returns the Wigner-Ville distribution when `x` is sampled with a time interval `ts` between samples.

`d = wvd( ___ ,'smoothedPseudo')` returns the smoothed pseudo Wigner-Ville distribution of `x`. The function uses the length of the input signal to choose the lengths of the windows used for time and frequency smoothing. This syntax can include any combination of input arguments from previous syntaxes.

`d = wvd( ___ ,'smoothedPseudo',twin,fwin)` specifies the time window, `twin`, and the frequency window, `fwin`, used for smoothing. To use the default window for either time or frequency smoothing, specify the corresponding argument as empty, `[]`.

`d = wvd( ___ ,'smoothedPseudo',Name,Value)` specifies additional options for the smoothed pseudo Wigner-Ville distribution using name-value pair arguments. You can specify `twin` and `fwin` in this syntax, or you can omit them.

`d = wvd( ___ ,'MinThreshold',thresh)` sets to zero those elements of `d` whose amplitude is less than `thresh`. This syntax applies to both the Wigner-Ville distribution and the smoothed pseudo Wigner-Ville distribution.

`[d,f,t] = wvd( ___ )` also returns a vector of frequencies, `f`, and a vector of times, `t`, at which `d` is computed.

`wvd( ___ )` with no output arguments plots the Wigner-Ville or smoothed pseudo Wigner-Ville distribution in the current figure.

Examples

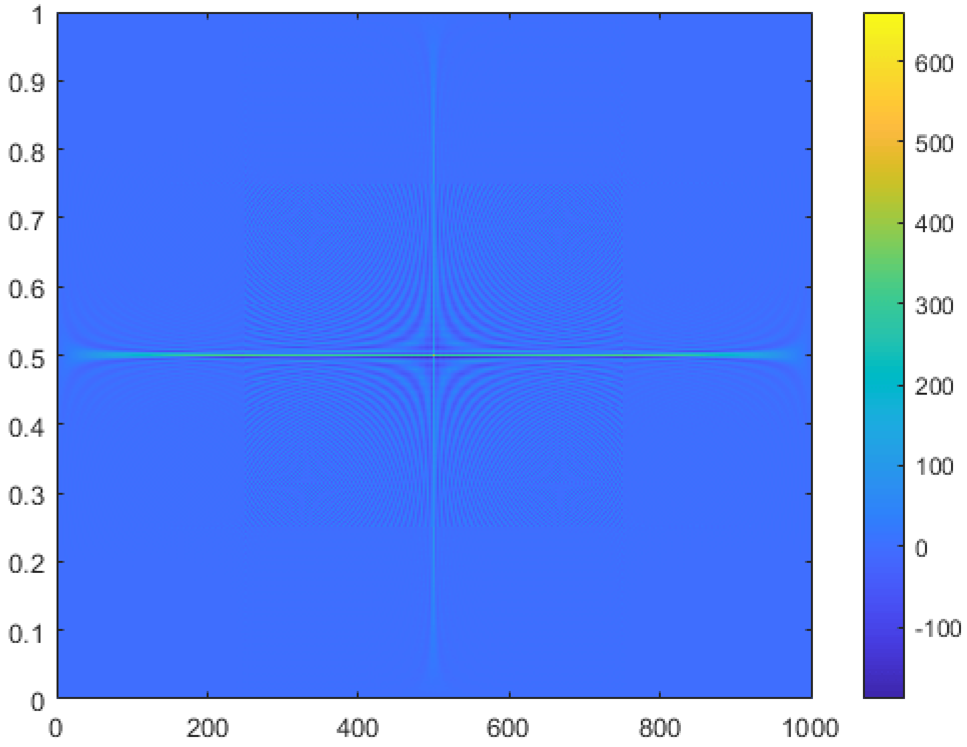
Wigner-Ville Distribution of Impulse and Tone

Generate a 1000-sample impulse and a 1000-sample tone with normalized frequency $\pi/2$. Compute the Wigner-Ville distribution of the sum of the two signals.

```
x = zeros(1001,1);
x(500) = 10;

y = sin(pi*(0:1000)/2)';

[d,f,t] = wvd(x+y);
```

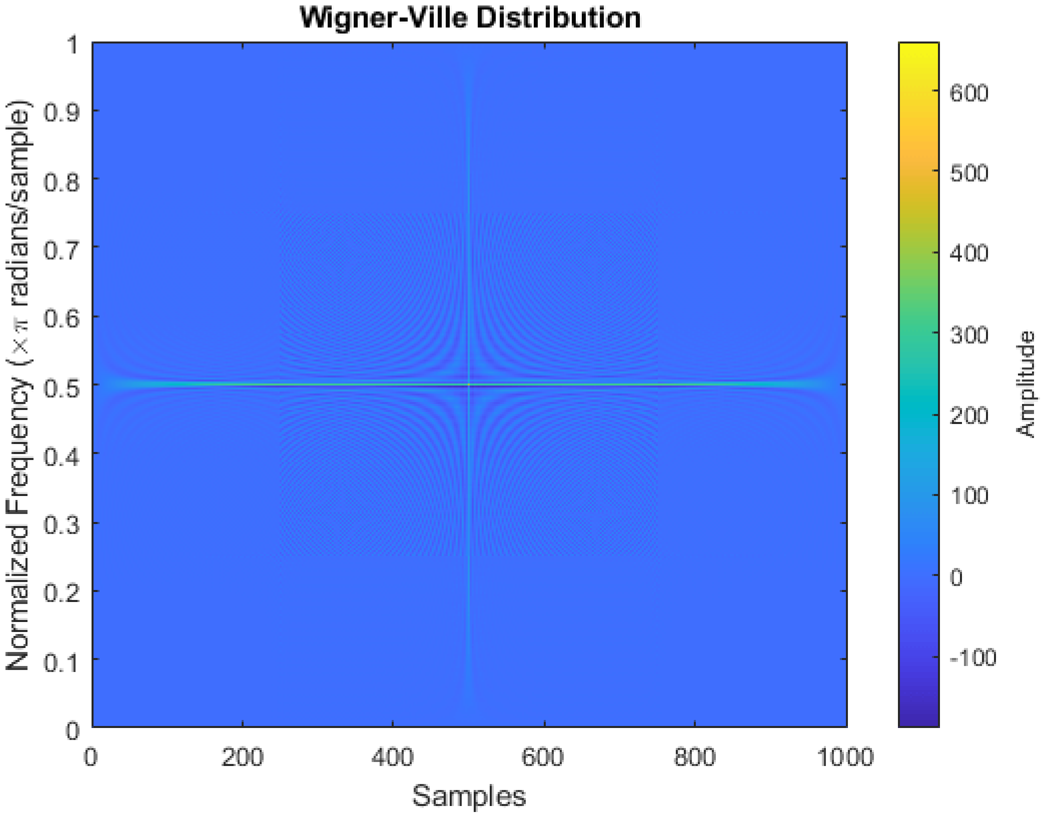
Plot the Wigner-Ville distribution.

```
imagesc(t,f,d)
axis xy
colorbar
```



Reproduce the result by calling `wvd` with no output arguments.

```
wvd(x+y)
```



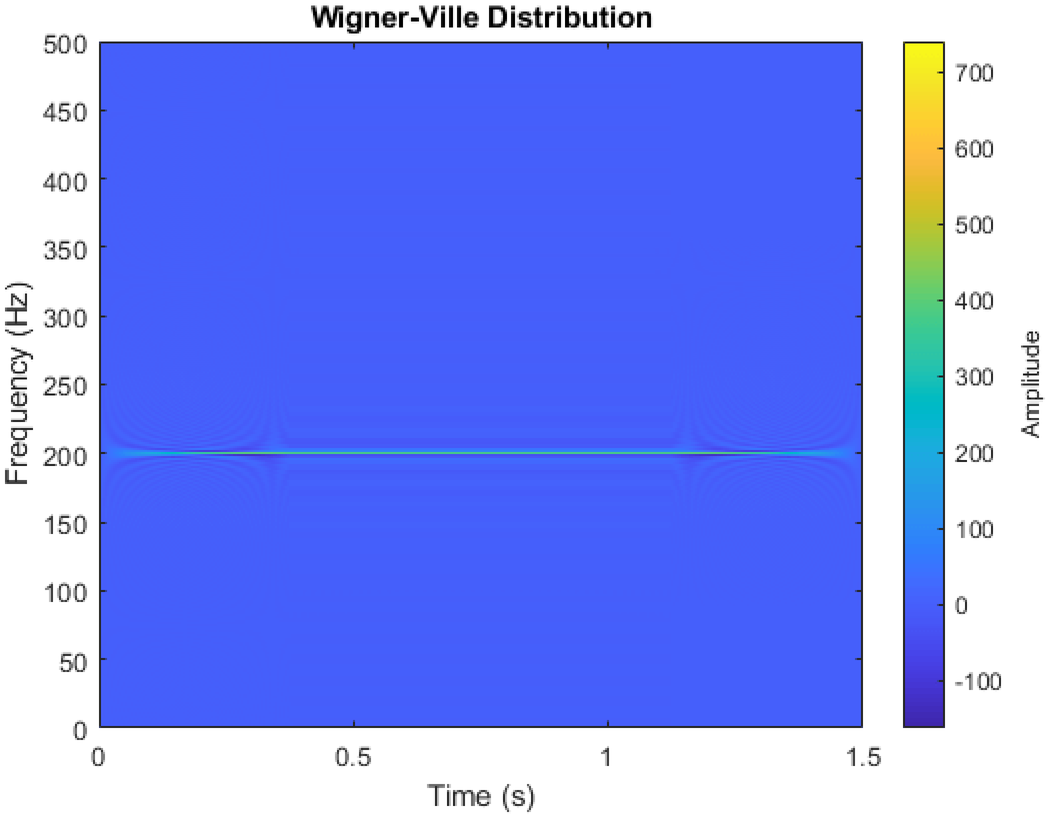
Wigner-Ville Distribution of Sinusoids

Generate a signal consisting of a 200 Hz sinusoid sampled at 1 kHz for 1.5 seconds.

```
fs = 1000;
t = (0:1/fs:1.5)';
x = cos(2*pi*t*200);
```

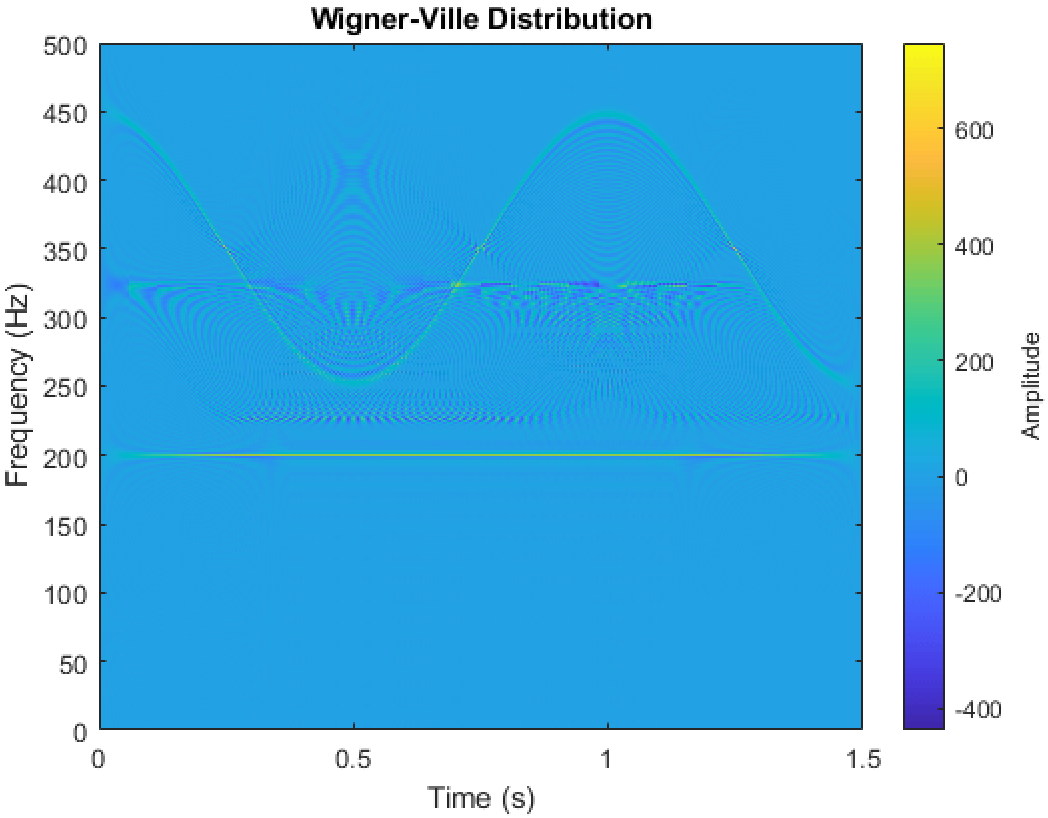
Compute the Wigner-Ville distribution of the signal.

```
wvd(x,fs)
```



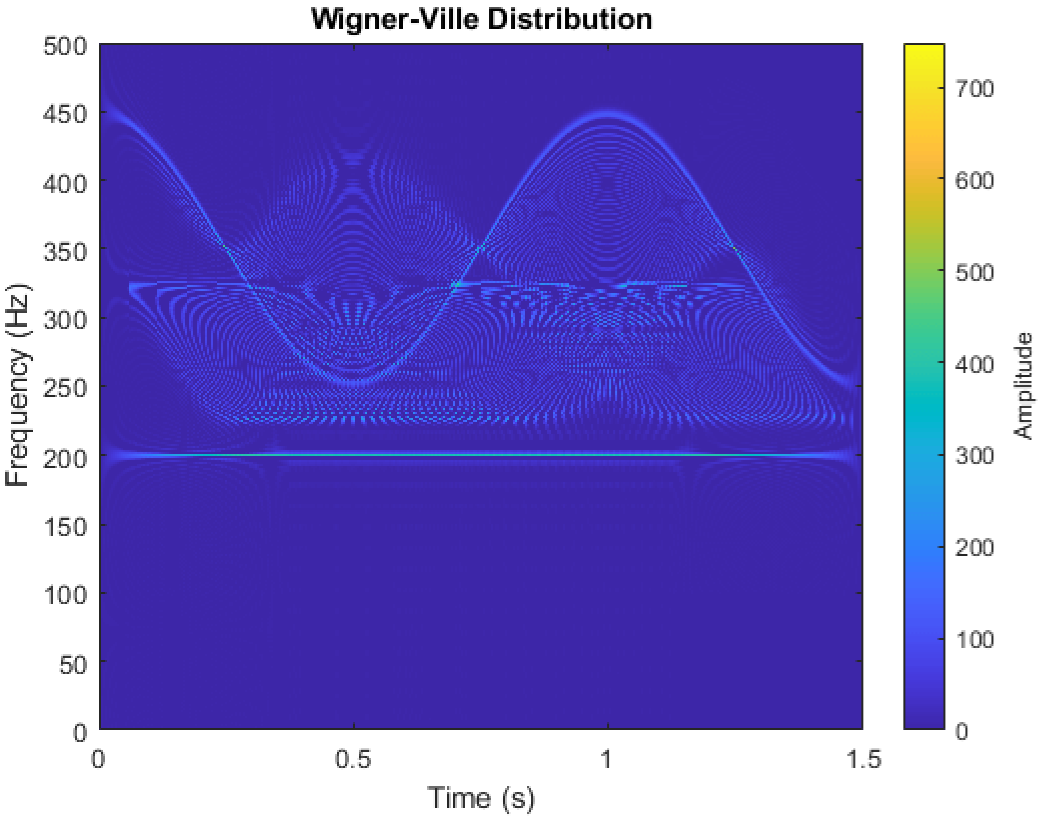
Add to the signal a chirp whose frequency varies sinusoidally between 250 Hz and 450 Hz. Convert the signal to a MATLAB® timetable. Compute the Wigner-Ville distribution.

```
x = x + vco(cos(2*pi*t),[250 450],fs);
xt = timetable(seconds(t),x);

wvd(xt)
```



Set to zero the distribution elements with amplitude less than 0.

```
wvd(xt,'MinThreshold',0)
```



Wigner-Ville Distribution of Chirps

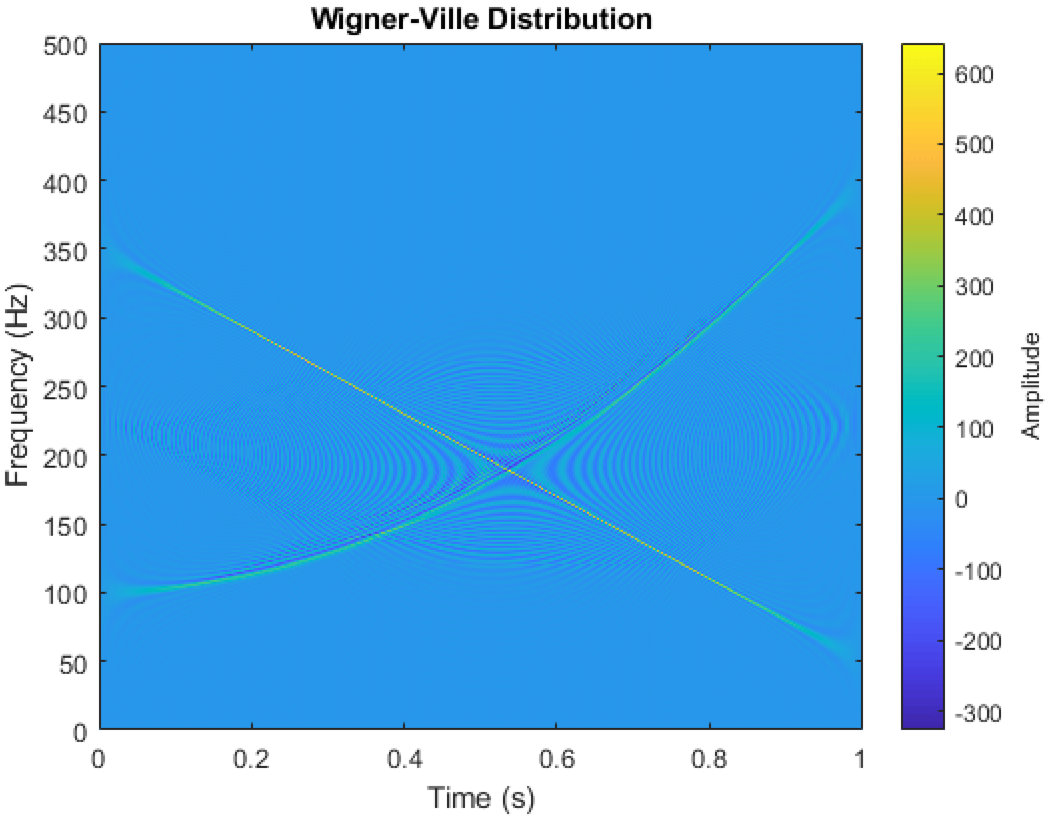
Generate a signal sampled at 1 kHz for 1 second. One component of the signal is a chirp that increases in frequency quadratically from 100 Hz to 400 Hz during the measurement. The other component of the signal is a chirp that decreases in frequency linearly from 350 Hz to 50 Hz in the same lapse.

Store the signal in a timetable.

```
fs = 1000;
t = 0:1/fs:1;

x = chirp(t,100,1,400,'quadratic') + chirp(t,350,1,50);
```

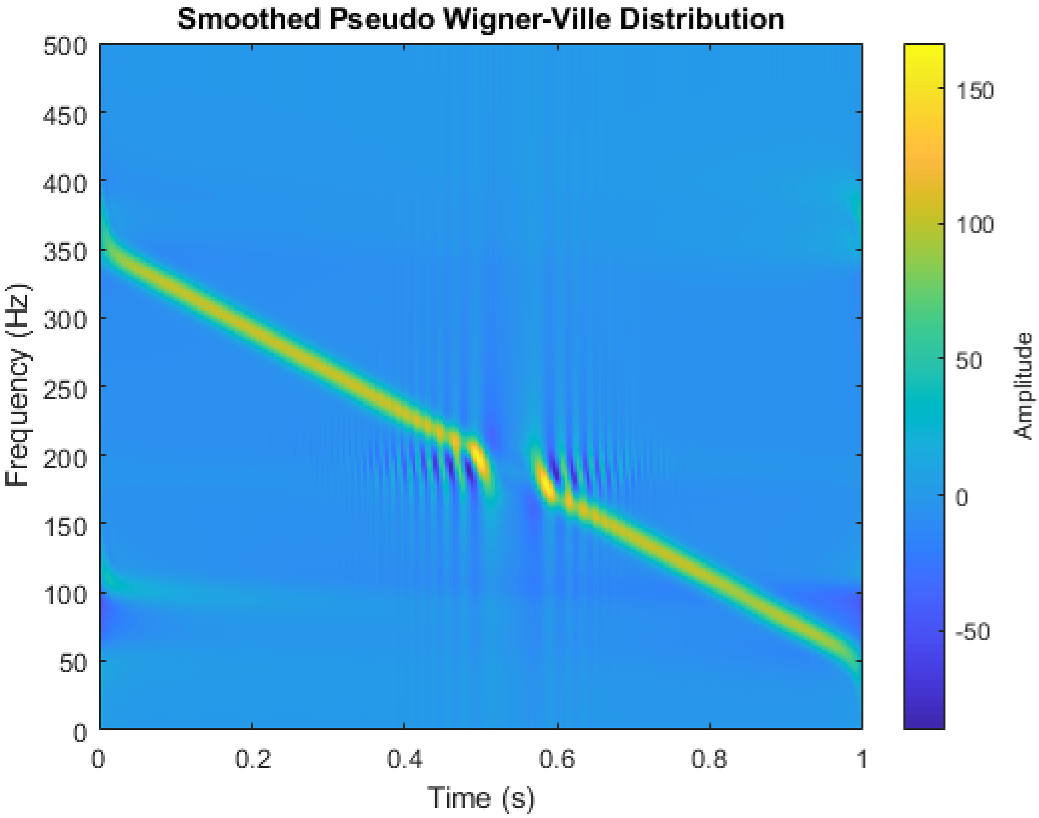
Compute the Wigner-Ville distribution of the signal.

```
wvd(x,fs)
```



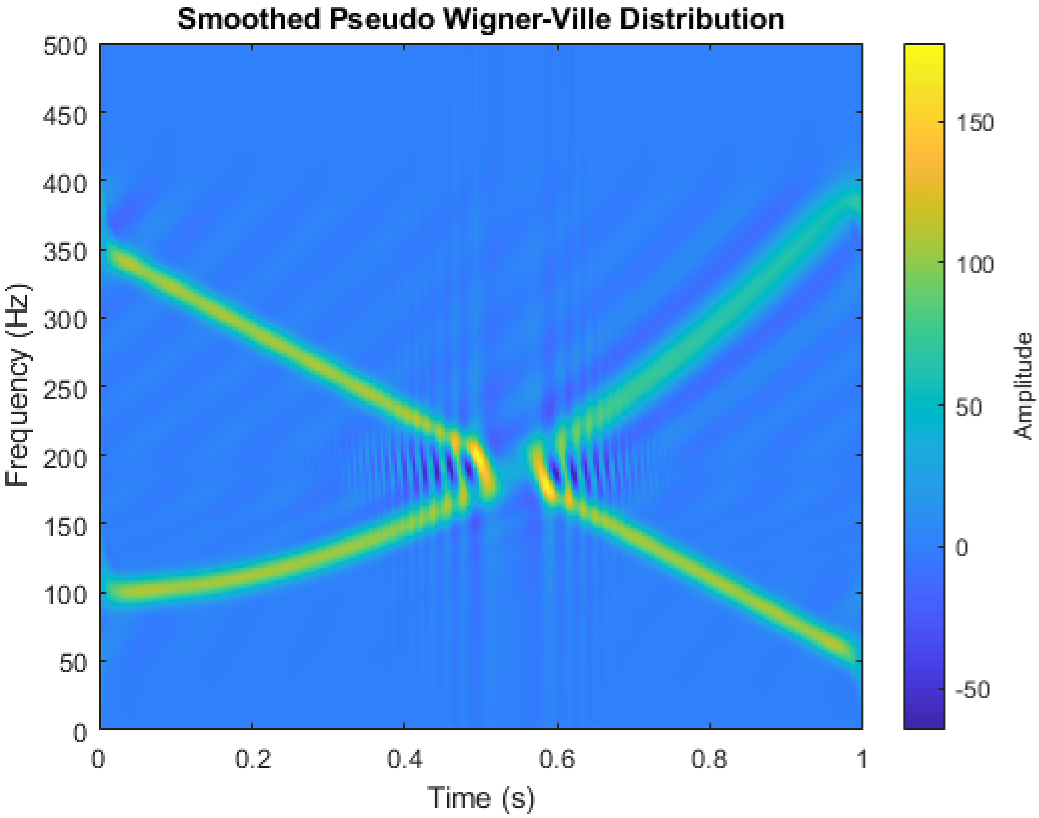
Compute the smoothed pseudo Wigner-Ville distribution of the signal. Specify 501 frequency points and 502 time points.

```
wvd(x,fs,'smoothedPseudo','NumFrequencyPoints',501,'NumTimePoints',502)
```



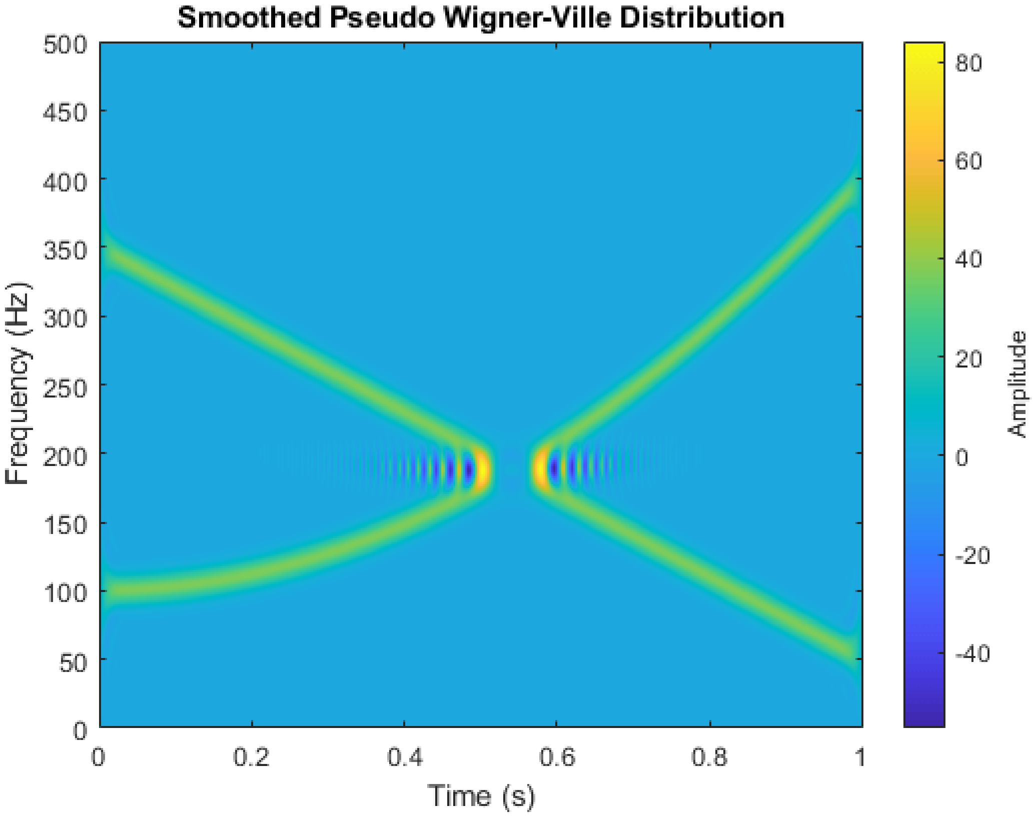
Increase the number of time points so the quadratic chirp becomes visible.

```
wvd(x,fs,'smoothedPseudo','NumFrequencyPoints',501,'NumTimePoints',522)
```



Increase the frequency points and time points to get a sharper image.

```
wvd(x,fs,'smoothedPseudo','NumFrequencyPoints',1000,'NumTimePoints',1502)
```



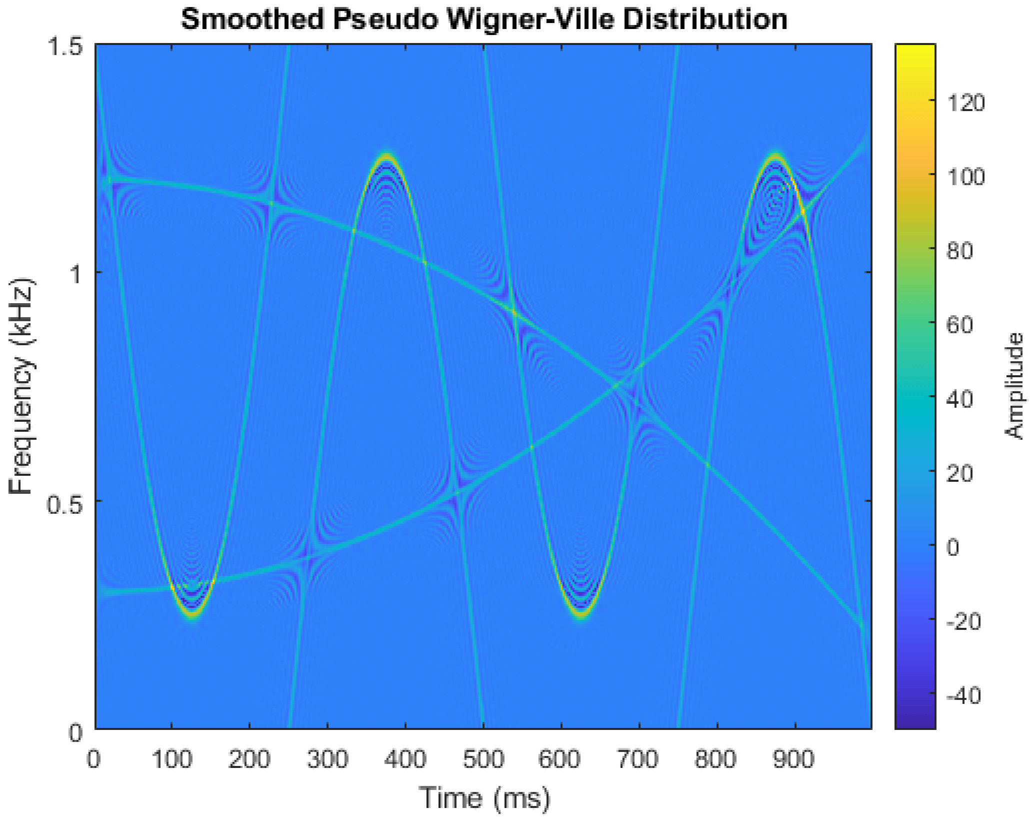
Smoothed Pseudo Wigner-Ville Distribution of Complex Signal

Generate a two-component signal sampled at 3 kHz for 1 second. The first component is a quadratic chirp whose frequency increases from 300 Hz to 1300 Hz during the measurement. The second component is a chirp with sinusoidally varying frequency content. The signal is embedded in white Gaussian noise. Express the time between consecutive samples as a `duration` scalar.

```
fs = 3000;
t = 0:1/fs:1-1/fs;
dt = seconds(t(2)-t(1));

x1 = chirp(t,300,t(end),1300,'quadratic');
x2 = exp(2j*pi*100*cos(2*pi*2*t));

x = x1 + x2 + randn(size(t))/10;
```

Compute and plot the smoothed pseudo Wigner Ville of the signal. Window the distribution in time using a 601-sample Hamming window and in frequency using a 305-sample rectangular window. Use 600 frequency points for the display. Set to zero those components of the distribution with amplitude less than −50.

```
wvd(x,dt,'smoothedPseudo',hamming(601),rectwin(305), ...
    'NumFrequencyPoints',600,'MinThreshold',-50)
```



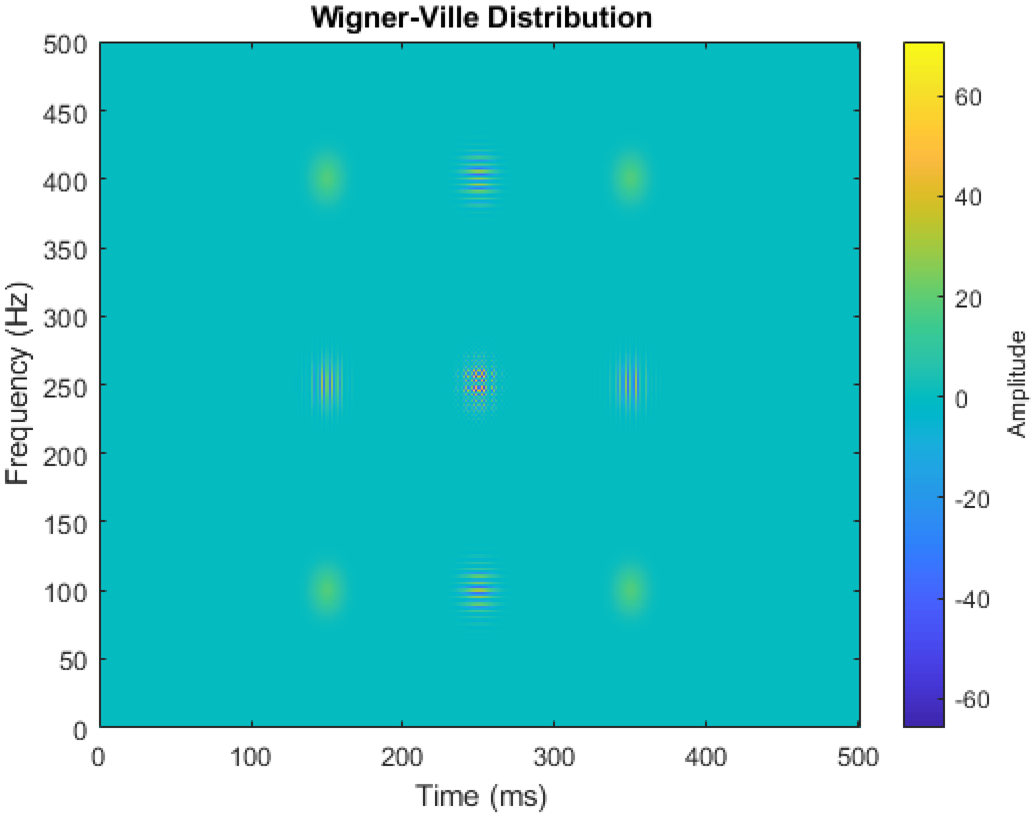
Interference Terms

Generate a signal composed of four Gaussian atoms. Each atom consists of a sinusoid modulated by a Gaussian. The sinusoids have frequencies of 100 Hz and 400 Hz. The Gaussians are centered at 150 milliseconds and 350 milliseconds and have a variance of 0.01^2. All atoms have unit amplitude. The signal is sampled at 1 kHz for half a second.

```
fs = 1000;
t = (0:1/fs:0.5)';

f1 = 100;
f2 = 400;

mu1 = 0.15;
mu2 = 0.35;

gaussFun = @(A,x,mu,f) exp(-(x-mu).^2/(2*0.01^2)).*sin(2*pi*f.*x)*A';

s = gaussFun([1 1 1 1],t,[mu1 mu1 mu2 mu2],[f1 f2 f1 f2]);
```

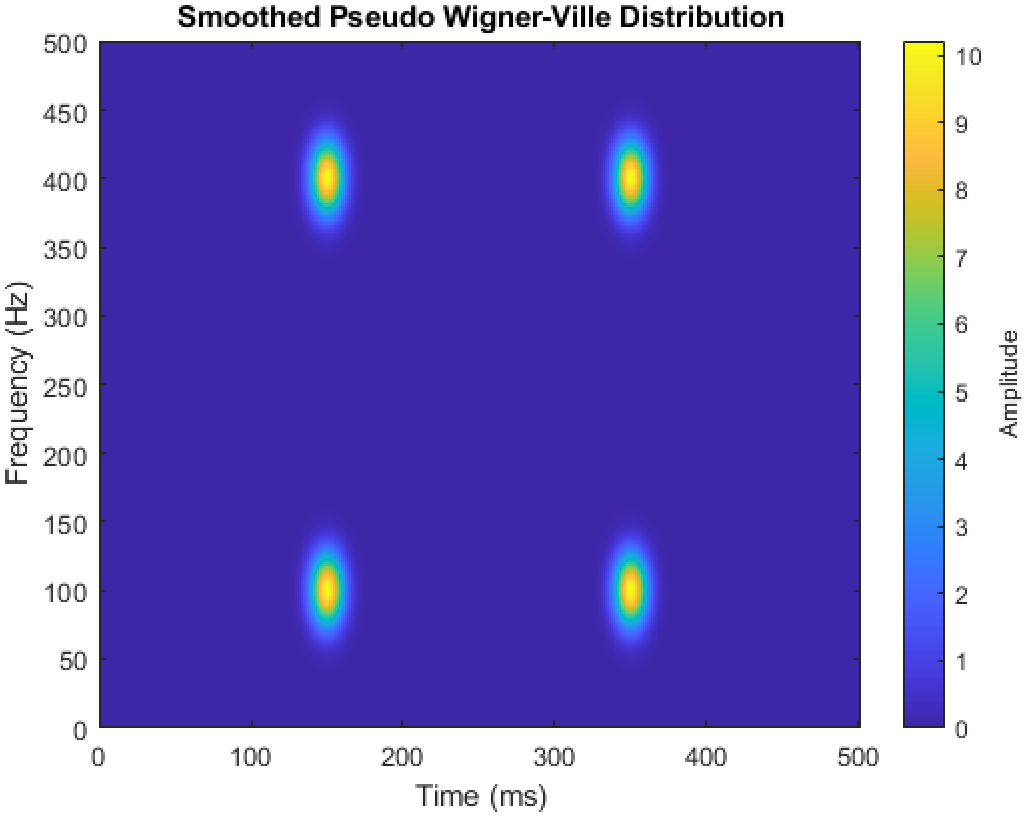
Compute and display the Wigner-Ville distribution of the signal. Interference terms, which can have negative values, appear halfway between each pair of auto-terms.

```
wvd(s,fs)
```



Compute and display the smoothed pseudo Wigner-Ville distribution of the signal. Smoothing in time and frequency attenuates the interference terms.

```
wvd(s,fs,'SmoothedPseudo')
```



Input Arguments

x — Input signal

vector | timetable

Input signal, specified as a vector or a MATLAB timetable containing a single vector variable.

- If x is a timetable, then it must contain increasing finite row times.
- If a timetable has missing or duplicate time points, you can fix it using the tips in “Clean Timetable with Missing, Duplicate, or Nonuniform Times”.

If the input signal has odd length, the function appends a zero to make the length even.

Example: `cos(pi/8*(0:159))'+randn(160,1)/10` specifies a sinusoid embedded in white noise.

Example: `timetable(seconds(0:5)',rand(6,1))` specifies a random variable sampled at 1 Hz for 5 seconds.

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

fs — Sample rate

`2*pi` (default) | positive numeric scalar

Sample rate, specified as a positive numeric scalar.

ts — Sample time
duration scalar

Sample time, specified as a duration scalar.

twin, fwin — Time and frequency windows
vectors of odd length

Time and frequency windows used for smoothing, specified as vectors of odd length. By default, wvd uses Kaiser windows with shape factor $\beta = 20$.

- The default length of twin is the smallest odd integer greater than or equal to round(length(x)/10).
- The default length of fwin is the smallest odd integer greater than or equal to nf/4, where nf is specified using NumFrequencyPoints.

Each window must have a length smaller than or equal to 2*ceil(length(x)/2).

Example: kaiser(65,0.5) specifies a 65-sample Kaiser window with a shape factor of 0.5.

thresh — Minimum nonzero value
-Inf (default) | real scalar

Minimum nonzero value, specified as a real scalar. The function sets to zero those elements of d whose amplitudes are less than thresh.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: 'NumFrequencyPoints',201,'NumTimePoints',300 computes the Wigner-Ville distribution at 201 frequency points and 300 time points.

NumFrequencyPoints — Number of frequency points
2*ceil(length(x)/2) (default) | integer

Number of frequency points, specified as the comma-separated pair consisting of 'NumFrequencyPoints' and an integer. This argument controls the degree of oversampling in frequency. The number of frequency points must be at least (length(fwin)+1)/2 and cannot be greater than the default.

NumTimePoints — Number of time points
4*ceil(length(x)/2) (default) | even integer

Number of time points, specified as the comma-separated pair consisting of 'NumTimePoints' and an even integer. This argument controls the degree of oversampling in time [3] (Signal Processing Toolbox). The number of time points must be at least 2*length(twin) and cannot be greater than the default.

Tip If the input signal is large, reduce the number of time points to lower the memory requirements and speed up the computation.

Output Arguments

d — Wigner-Ville distribution

matrix

Wigner-Ville distribution, returned as a matrix. Time increases across the columns of d, and frequency increases down the rows. The matrix is of size $N_\mathrm{f} \times N_\mathrm{t}$, where N_f is the length of f and N_t is the length of t.

f — Frequencies

vector

Frequencies, returned as a vector.

- If the input has time information, then f contains frequencies expressed in Hz.
- If the input does not have time information, then f contains normalized frequencies expressed in rad/sample.

t — Time instants

vector

Time instants, returned as a vector.

- If the input has time information, then t contains time values expressed in seconds.
- If the input does not have time information, then t contains sample numbers.

More About

Wigner-Ville Distribution

The Wigner-Ville distribution provides a high-resolution time-frequency representation of a signal. The distribution has applications in signal visualization, detection, and estimation.

For a continuous signal $x(t)$, the Wigner-Ville distribution is defined as

$$\mathrm{WVD}_x(t,f) = \int_{-\infty}^{\infty} x\left(t+\frac{\tau}{2}\right)x^*\left(t-\frac{\tau}{2}\right)e^{-j2\pi f\tau}\,d\tau\,.$$

For a discrete signal with N samples, the distribution becomes

$$\mathrm{WVD}_x(n,k) = \sum_{m\,=\,-N}^{N} x(n+m/2)\,x^*(n-m/2)\,e^{-j2\pi km/N}\,.$$

For odd values of m, the definition requires evaluation of the signal at half-integer sample values. It therefore requires interpolation, which makes it necessary to zero-pad the discrete Fourier transform to avoid aliasing.

The Wigner-Ville distribution contains interference terms that often complicate its interpretation. To sharpen the distribution, one can filter the definition with lowpass windows. The smoothed pseudo Wigner-Ville distribution uses independent windows to smooth in time and frequency:

$$\mathrm{SPWVD}_x^{g,H}(t,f) = \int_{-\infty}^{\infty} g(t)\,H(f)\,x\left(t+\frac{\tau}{2}\right)x^*\left(t-\frac{\tau}{2}\right)e^{-j2\pi f\tau}\,d\tau\,.$$

References

[1] Cohen, Leon. *Time-Frequency Analysis: Theory and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[2] Mallat, Stéphane. *A Wavelet Tour of Signal Processing*. Second Edition. San Diego, CA: Academic Press, 1999.

[3] O'Toole, John M., and Boualem Boashash. "Fast and Memory-Efficient algorithms for Computing Quadratic Time-Frequency Distributions." *Applied and Computational Harmonic Analysis*. Vol. 35, Number 2, 2013, pp. 350–358.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Arguments specified using name-value pairs must be compile-time constants.
- Timetables are not supported for code generation.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Functions

`xwvd`

Topics

“Time-Frequency Gallery”

Introduced in R2018b

xwvd

Cross Wigner-Ville distribution and cross smoothed pseudo Wigner-Ville distribution

Syntax

```
d = xwvd(x,y)
d = xwvd(x,y,fs)
d = xwvd(x,y,ts)
```

```
d = xwvd( ___ ,'smoothedPseudo')
d = xwvd( ___ ,'smoothedPseudo',twin,fwin)
d = xwvd( ___ ,'smoothedPseudo','NumFrequencyPoints',nf)
```

```
d = xwvd( ___ ,'MinThreshold',thresh)
```

```
[d,f,t] = xwvd( ___ )
```

```
xwvd( ___ )
```

Description

`d = xwvd(x,y)` returns the cross Wigner-Ville distribution of `x` and `y`.

`d = xwvd(x,y,fs)` returns the cross Wigner-Ville distribution when `x` and `y` are sampled at a rate `fs`.

`d = xwvd(x,y,ts)` returns the cross Wigner-Ville distribution when `x` and `y` are sampled with a time interval `ts` between samples.

`d = xwvd( ___ ,'smoothedPseudo')` returns the cross smoothed pseudo Wigner-Ville distribution of `x` and `y`. The function uses the length of the input signals to choose the lengths of the windows used for time and frequency smoothing. This syntax can include any combination of input arguments from previous syntaxes.

`d = xwvd( ___ ,'smoothedPseudo',twin,fwin)` specifies the time window, `twin`, and the frequency window, `fwin`, used for smoothing. To use the default window for either time or frequency smoothing, specify the corresponding argument as empty, `[]`.

`d = xwvd( ___ ,'smoothedPseudo','NumFrequencyPoints',nf)` computes the cross smoothed pseudo Wigner-Ville distribution using `nf` frequency points. You can specify `twin` and `fwin` in this syntax, or you can omit them.

`d = xwvd( ___ ,'MinThreshold',thresh)` sets to zero those elements of `d` whose amplitude is less than `thresh`. This syntax applies to both the cross Wigner-Ville distribution and the cross smoothed pseudo Wigner-Ville distribution.

`[d,f,t] = xwvd( ___ )` also returns a vector of frequencies, `f`, and a vector of times, `t`, at which `d` is computed.

`xwvd( ___ )` with no output arguments plots the real part of the cross Wigner-Ville or cross smoothed pseudo Wigner-Ville distribution in the current figure.

Examples

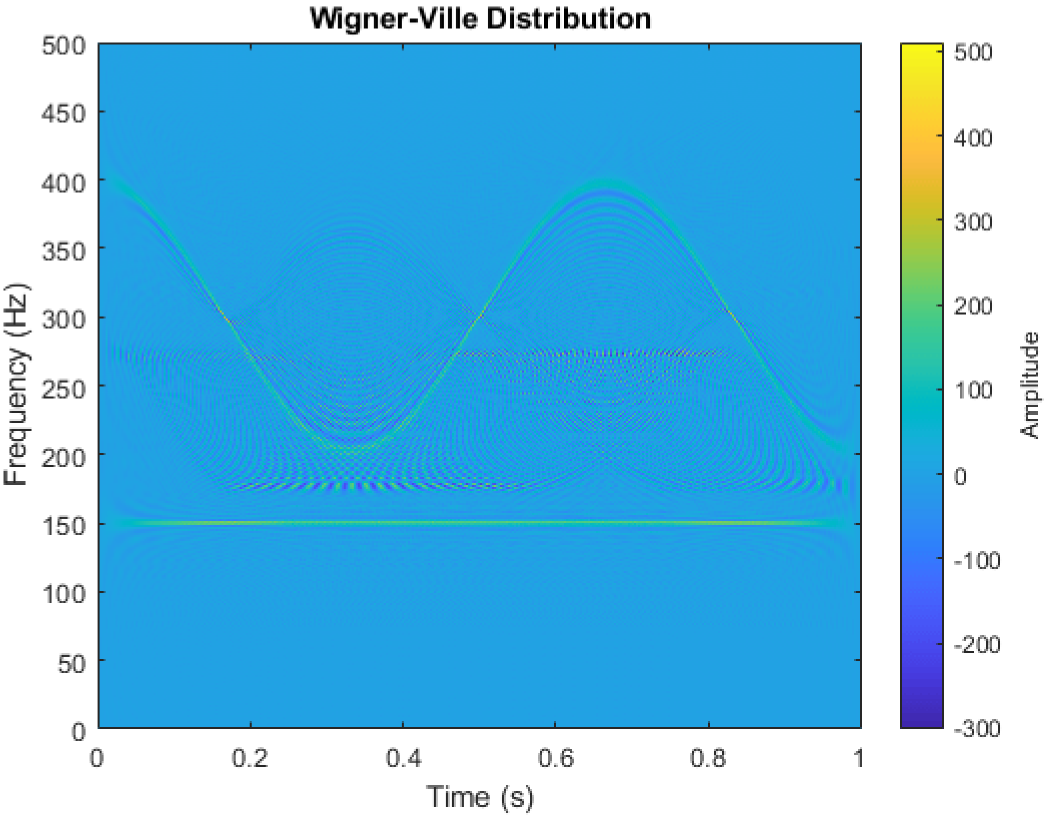
Cross Wigner-Ville Distribution of Signals

Generate two signals sampled at 1 kHz for 1 second and embedded in white noise. One signal is a sinusoid of frequency 150 Hz. The other signal is a chirp whose frequency varies sinusoidally between 200 Hz and 400 Hz. The noise has a variance of 0.1^2.

```
fs = 1000;
t = (0:1/fs:1)';

x = cos(2*pi*t*150) + 0.1*randn(size(t));
y = vco(cos(3*pi*t),[200 400],fs) + 0.1*randn(size(t));
```

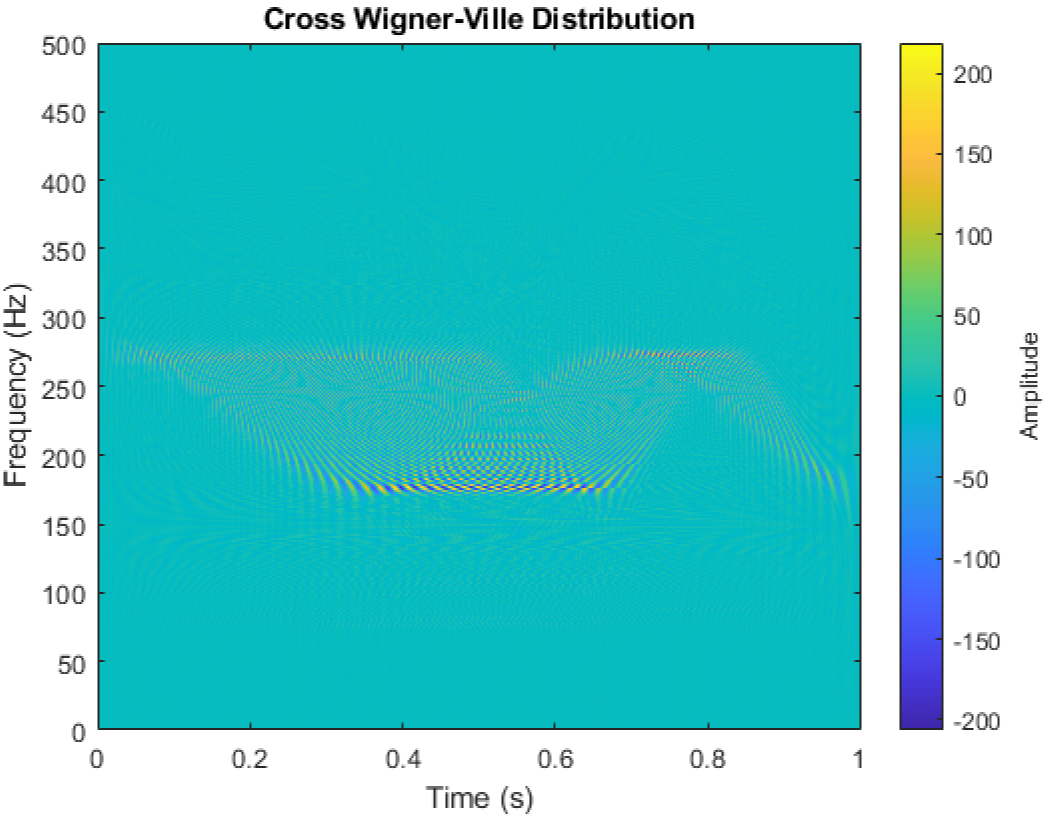
Compute the Wigner-Ville distribution of the sum of the signals.

```
wvd(x+y,fs)
```



Compute and plot the cross Wigner-Ville distribution of the signals. The cross-distribution corresponds to the cross-terms of the Wigner-Ville distribution.

```
xwvd(x,y,fs)
```



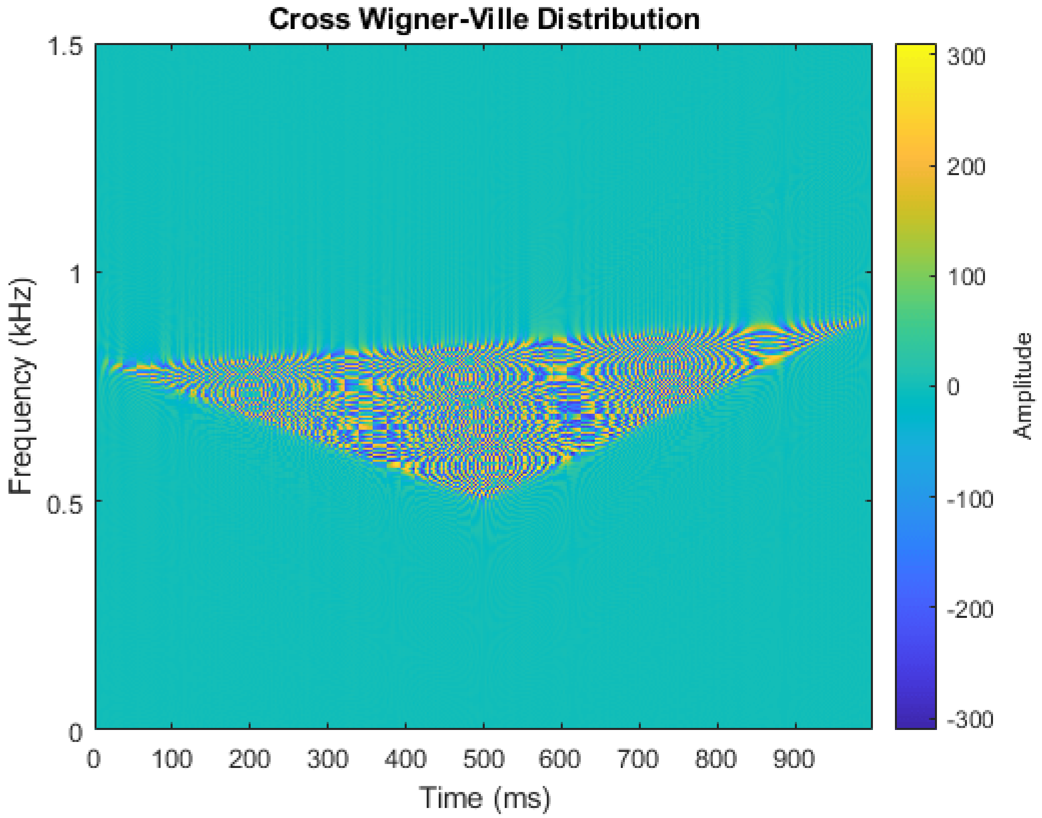
Cross Wigner-Ville Distribution of Chirps

Generate a two-channel signal that consists of two chirps. The signal is sampled at 3 kHz for one second. The first chirp has an initial frequency of 400 Hz and reaches 800 Hz at the end of the sampling. The second chirp starts at 500 Hz and reaches 1000 Hz at the end. The second chirp has twice the amplitude of the first chirp.

```
fs = 3000;
t = (0:1/fs:1-1/fs)';

x1 = chirp(t,1400,t(end),800);
x2 = 2*chirp(t,200,t(end),1000);
```

Store the signal as a timetable. Compute and plot the cross Wigner-Ville distribution of the two channels.

```
xt = timetable(seconds(t),x1,x2);

xwvd(xt(:,1),xt(:,2))
```



Use Cross Wigner-Ville Distribution to Estimate Instantaneous Frequency

Compute the instantaneous frequency of a signal by using a known reference signal and the cross Wigner-Ville distribution.

Create a reference signal consisting of a Gaussian atom sampled at 1 kHz for 1 second. A Gaussian atom is a sinusoid modulated by a Gaussian. Specify a sinusoid frequency of 50 Hz. The Gaussian is centered at 64 milliseconds and has a variance of 0.01^2.

```
fs = 1e3;
t = (0:1/fs:1-1/fs)';

mu = 0.064;
sigma = 0.01;
fsin = 50;

xr = exp(-(t-mu).^2/(2*sigma^2)).*sin(2*pi*fsin*t);
```

Create the "unknown" signal to analyze, consisting of a chirp. The signal starts suddenly at 0.4 second and ends suddenly half a second later. In that lapse, the frequency of the chirp decreases linearly from 400 Hz to 100 Hz.

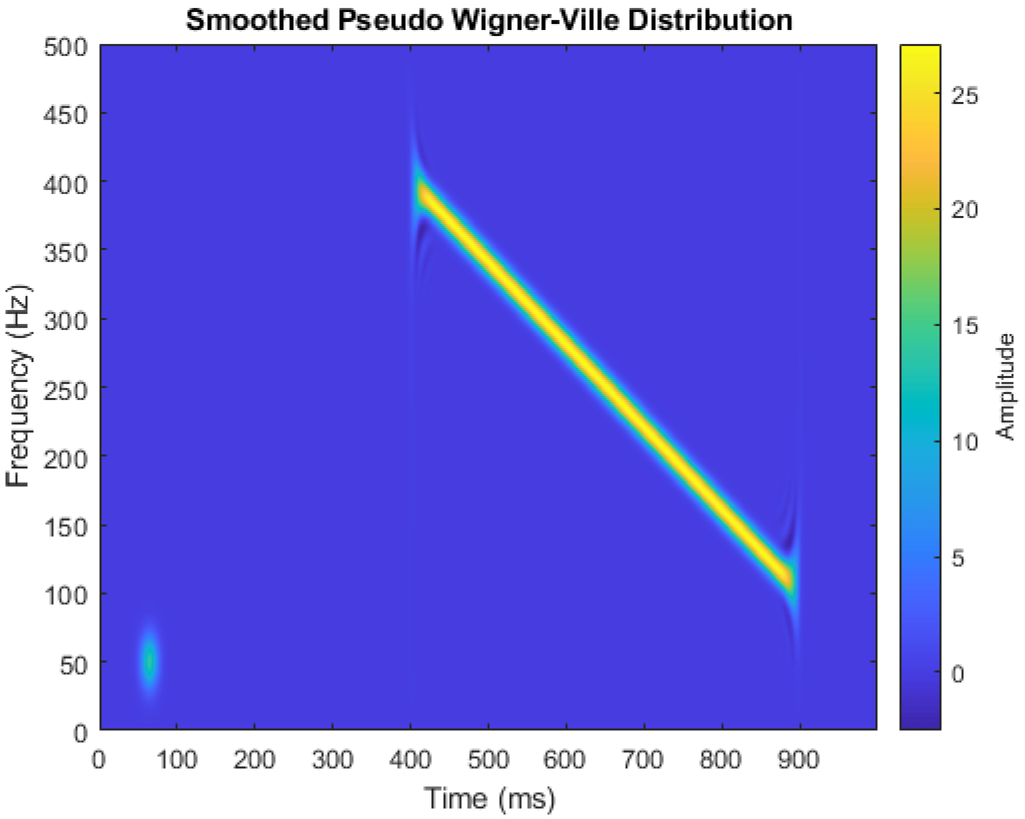
```
f0 = 400;
f1 = 100;
```

```
xa = zeros(size(t));
xa(t>0.4 & t<=0.9) = chirp((0:1/fs:0.5-1/fs)',f0,0.5,f1);
```

Create a two-component signal consisting of the sum of the unknown and reference signals. The smoothed pseudo Wigner-Ville distribution of the result provides an "ideal" time-frequency representation.

Compute and display the smoothed pseudo Wigner-Ville distribution.

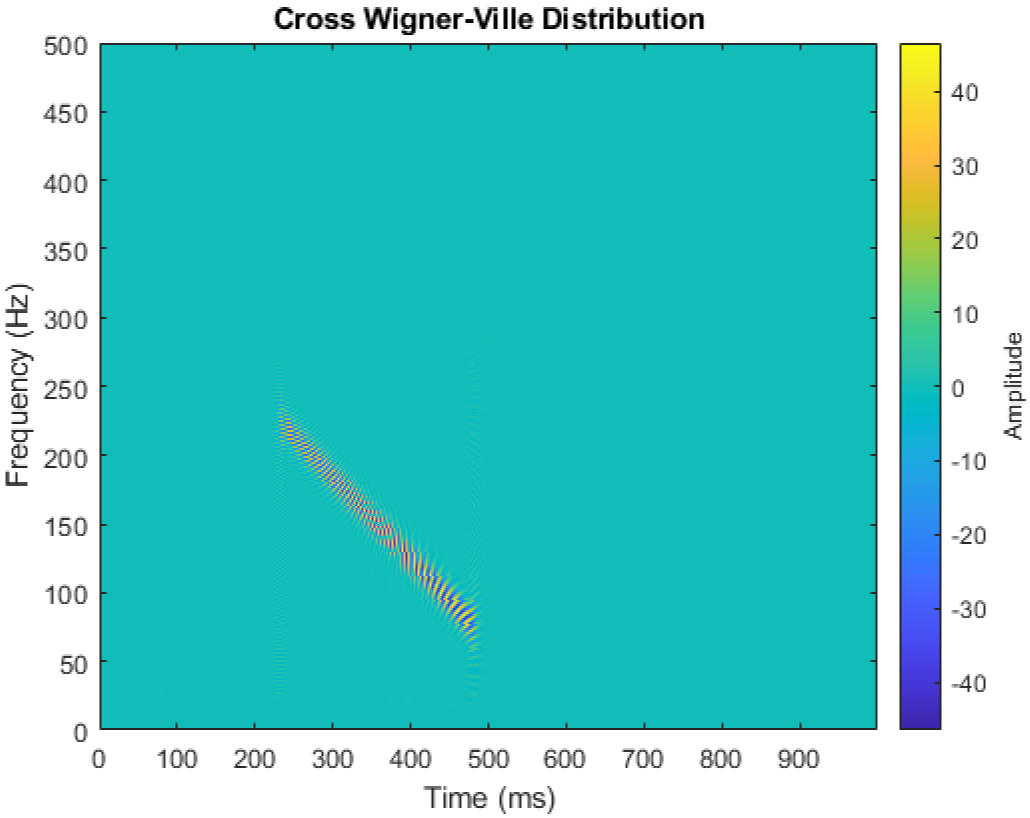
```
w = wvd(xa+xr,fs,'smoothedPseudo');

wvd(xa+xr,fs,'smoothedPseudo')
```



Compute the cross Wigner-Ville distribution of the unknown and reference signals. Take the absolute value of the distribution and set to zero the elements with amplitude less than 10. The cross Wigner-Ville distribution is equal to the cross-terms of the two-component signal.

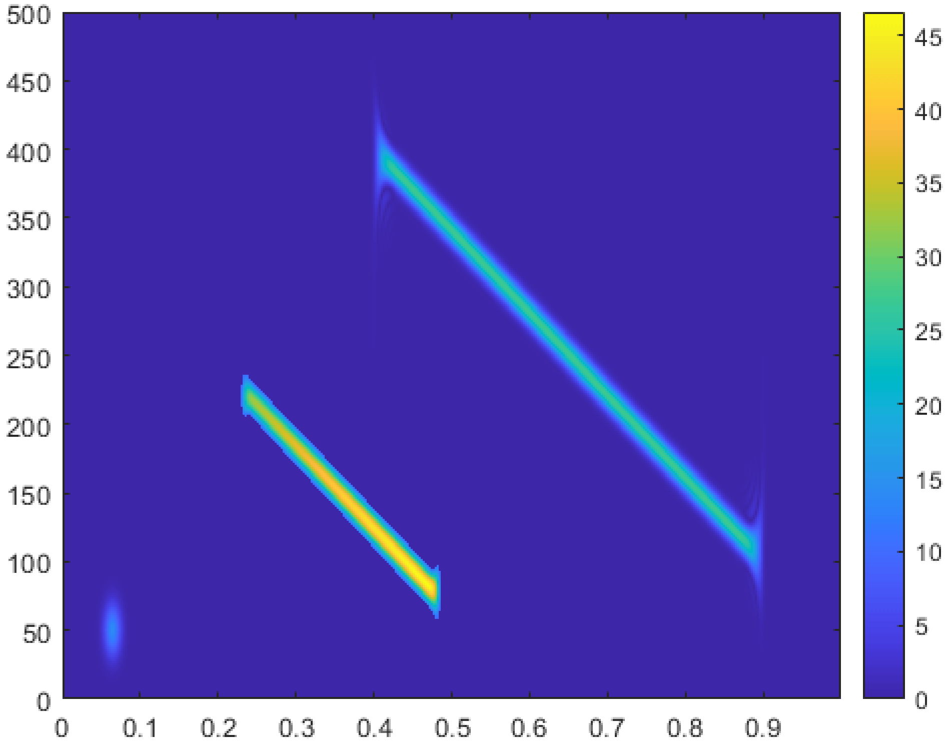
Plot the real part of the cross Wigner-Ville distribution.

```
[c,fc,tc] = xwvd(xa,xr,fs);
c = abs(c);
c(c<10) = 0;

xwvd(xa,xr,fs)
```



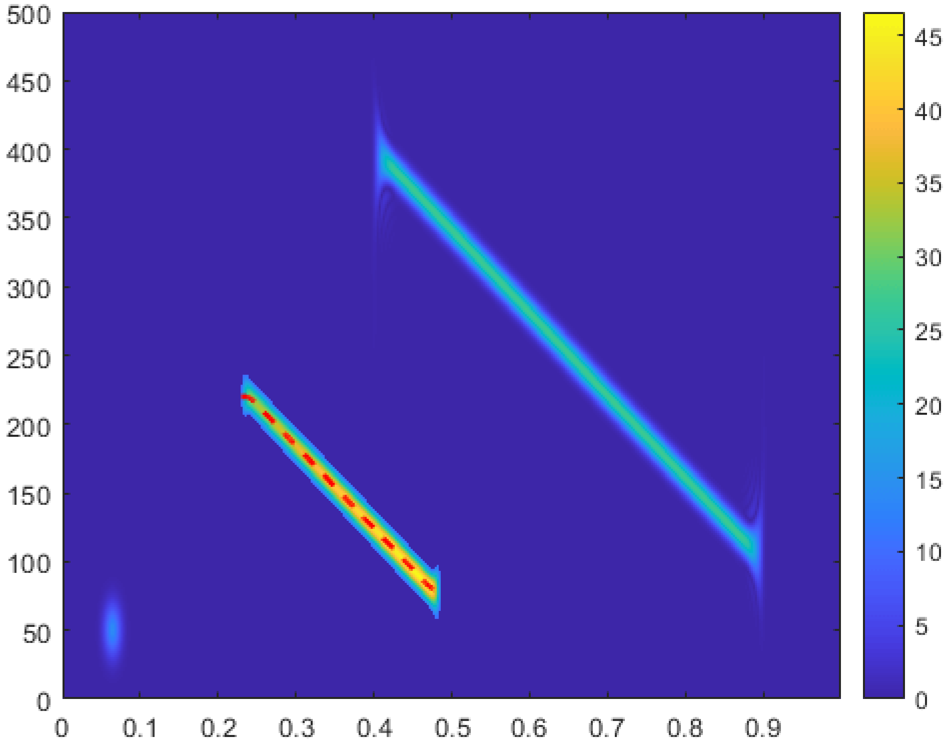
Enhance the Wigner-Ville cross-terms by adding the ideal time-frequency representation to the cross Wigner-Ville distribution. The cross-terms of the Wigner-Ville distribution occur halfway between the reference signal and the unknown signal.

```
d = w + c;

d = abs(real(d));

imagesc(tc,fc,d)
axis xy
colorbar
```



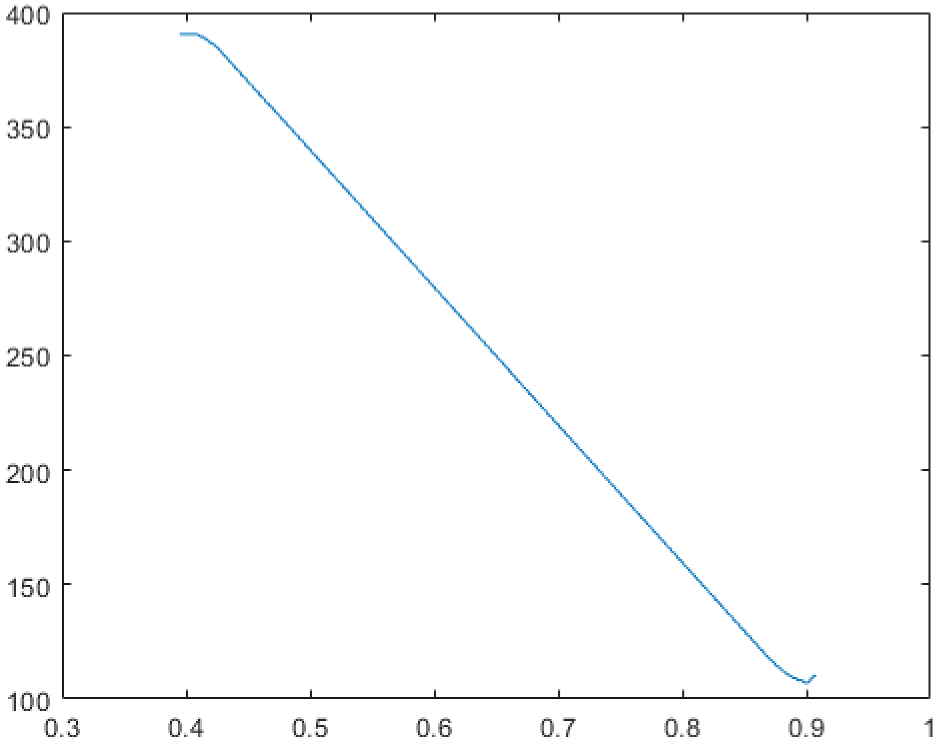
Identify and plot the high-energy ridge corresponding to the cross-terms. To isolate the ridge, find the time values where the cross-distribution has nonzero energy.

```
ff = tfridge(c,fc);

tv = sum(c)>0;

ff = ff(tv);
tc = tc(tv);

hold on
plot(tc,ff,'r--','linewidth',2)
hold off
```



Reconstruct the instantaneous frequency of the unknown signal by using the ridge and the reference function. Plot the instantaneous frequency as a function of time.

```
tEst = 2*tc - mu;
fEst = 2*ff - fsin;

plot(tEst,fEst)
```



Input Arguments

x, y — Input signals
vectors | timetables

Input signals, specified as vectors or MATLAB timetables each containing a single vector variable. `x` and `y` must both be vectors or both be timetables and must have the same length.

- If `x` and `y` are timetables, then they must contain increasing finite row times.
- If a timetable has missing or duplicate time points, you can fix it using the tips in "Clean Timetable with Missing, Duplicate, or Nonuniform Times".

If the input signals have odd length, the function appends a zero to make the length even.

Example: `cos(pi/8*(0:159))'+randn(160,1)/10` specifies a sinusoid embedded in white noise.

Example: `timetable(seconds(0:5)',rand(6,1))` specifies a random variable sampled at 1 Hz for 4 seconds.

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

`fs` — Sample rate
`2*pi` (default) | positive numeric scalar

Sample rate, specified as a positive numeric scalar.

ts — Sample time
duration scalar

Sample time, specified as a duration scalar.

twin, fwin — Time and frequency windows
vectors of odd length

Time and frequency windows used for smoothing, specified as vectors of odd length. By default, xwvd uses Kaiser windows with shape factor $\beta = 20$.

- The default length of twin is the smallest odd integer greater than or equal to round(length(x)/10).
- The default length of fwin is the smallest odd integer greater than or equal to nf/4.

Each window must have a length smaller than or equal to 2*ceil(length(x)/2).

Example: kaiser(65,0.5) specifies a 65-sample Kaiser window with a shape factor of 0.5.

nf — Number of frequency points
2*ceil(length(x)/2) (default) | integer

Number of frequency points, specified as an integer. This argument controls the degree of oversampling in frequency. The number of frequency points must be at least (length(fwin)+1)/2 and cannot be greater than the default.

thresh — Minimum nonzero value
-Inf (default) | real scalar

Minimum nonzero value, specified as a real scalar. The function sets to zero those elements of d whose amplitudes are less than thresh.

Output Arguments

d — Cross Wigner-Ville distribution
matrix

Cross Wigner-Ville distribution, returned as a matrix. Time increases across the columns of d, and frequency increases down the rows. The matrix is of size $N_\mathrm{f} \times N_\mathrm{t}$, where N_f is the length of f and N_t is the length of t.

f — Frequencies
vector

Frequencies, returned as a vector.

- If the input has time information, then f contains frequencies expressed in Hz.
- If the input does not have time information, then f contains normalized frequencies expressed in rad/sample.

t — Time instants
vector

Time instants, returned as a vector.

- If the input has time information, then `t` contains time values expressed in seconds.
- If the input does not have time information, then `t` contains sample numbers.

The number of time points is fixed as `4*ceil(length(x)/2)`.

More About

Cross Wigner-Ville Distribution

For continuous signals $x(t)$ and $y(t)$, the *cross Wigner-Ville distribution* is defined as

$$\mathrm{XWVD}_{x,y}(t,f) = \int_{-\infty}^{\infty} x\left(t+\frac{\tau}{2}\right)y^*\left(t-\frac{\tau}{2}\right)e^{-j2\pi f\tau}\,d\tau\,.$$

For a discrete signal with N samples, the distribution becomes

$$\mathrm{XWVD}_{x,y}(n,k) = \sum_{m=-N}^{N} x(n+m/2)\,y^*(n-m/2)\,e^{-j2\pi km/N}\,.$$

For odd values of m, the definition requires evaluation of the signal at half-integer sample values. It therefore requires interpolation, which makes it necessary to zero-pad the discrete Fourier transform to avoid aliasing.

The cross Wigner-Ville distribution contains interference terms that often complicate its interpretation. To sharpen the distribution, one can filter the definition with lowpass windows. The cross smoothed pseudo Wigner-Ville distribution uses independent windows to smooth in time and frequency:

$$\mathrm{XSPWVD}_{x,y}^{g,H}(t,f) = \int_{-\infty}^{\infty} g(t)\,H(f)\,x\left(t+\frac{\tau}{2}\right)y^*\left(t-\frac{\tau}{2}\right)e^{-j2\pi f\tau}\,d\tau\,.$$

References

[1] Cohen, Leon. *Time-Frequency Analysis: Theory and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[2] Mallat, Stéphane. *A Wavelet Tour of Signal Processing*. Second Edition. San Diego, CA: Academic Press, 1999.

[3] Malnar, Damir, Victor Sucic, and Boualem Boashash. "A cross-terms geometry based method for components instantaneous frequency estimation using the cross Wigner-Ville distribution." In *11th International Conference on Information Sciences, Signal Processing and their Applications (ISSPA)*, pp. 1217–1222. Montréal: IEEE, 2012.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Arguments specified using name-value pairs must be compile-time constants.

- Timetables are not supported for code generation.

See Also

Functions
`xspectrogram` | `wvd`

Topics
“Time-Frequency Gallery”

Introduced in R2018b