Package ‘waveslim’

January 26, 2019

Version 1.7.5.1
Date 2014-12-21
Title Basic Wavelet Routines for One-, Two- And Three-Dimensional Signal Processing
Author Brandon Whitcher
Maintainer Brandon Whitcher <bwhitcher@gmail.com>
Depends R (>= 2.11.0), graphics, grDevices, stats, utils
Suggests fftw
Description Basic wavelet routines for time series (1D), image (2D) and array (3D) analysis. The code provided here is based on wavelet methodology developed in Percival and Walden (2000); Gencay, Selcuk and Whitcher (2001); the dual-tree complex wavelet transform (DTCWT) from Kingsbury (1999, 2001) as implemented by Selesnick; and Hilbert wavelet pairs (Selesnick 2001, 2002). All figures in chapters 4-7 of GSW (2001) are reproducible using this package and R code available at the book website(s) below.
License BSD_3_clause + file LICENSE
URL http://waveslim.blogspot.com,
       http://www2.imperial.ac.uk/~bwhitche/book
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-01-26 16:48:57 UTC

R topics documented:

Andel ......................................................... 3
ar1 ......................................................... 4
Band-pass variance ......................................... 4
barbara ...................................................... 5
basis ....................................................... 6
blocks ..................................................... 7
R topics documented:

- brick.wall ................................................................. 7
- convolve2D ............................................................... 8
- cpi ........................................................................... 9
- dau ........................................................................... 10
- denoise.2d ................................................................. 10
- doppler ................................................................. 12
- dpss.taper ................................................................. 12
- Dual-tree Filter Banks .................................................. 14
- dualfilt1 ................................................................. 16
- Dualtree ................................................................. 17
- Dualtree Complex ........................................................ 19
- dwpt ................................................................. 20
- dwpt.2d ................................................................. 22
- dwpt.boot ................................................................. 23
- dwpt.sim ................................................................. 24
- dwt ................................................................. 25
- dwt.2d ................................................................. 28
- dwt.3d ................................................................. 29
- exchange ............................................................... 30
- Farras ................................................................. 30
- fdp.mle ................................................................. 31
- find.adaptive.basis ..................................................... 32
- heavisine ............................................................... 33
- Hilbert ................................................................. 34
- hilbert.filter ........................................................... 35
- hosking.sim ............................................................ 36
- HWP Analysis ........................................................... 37
- ibm ................................................................. 38
- japan ................................................................. 39
- jumpsine ............................................................... 39
- kobe ................................................................. 40
- linchirp ............................................................... 40
- mexm ................................................................. 41
- modwt ................................................................. 41
- modwt.2d ............................................................... 43
- modwt.3d ............................................................... 45
- mra ................................................................. 46
- mra.2d ............................................................... 47
- mra.3d ............................................................... 49
- mult.loc ............................................................... 50
- my.acf ............................................................... 51
- nile ................................................................. 52
- ortho.basis ............................................................. 53
- per ................................................................. 54
- phase.shift ............................................................ 54
- phase.shift.hilbert ..................................................... 55
- plot.dwt.2d .............................................................. 56
- qmf ................................................................. 57
Description

The autocovariance and autocorrelation sequences from the time series model in Figures 8, 9, 10, and 11 of Andel (1986). They were obtained through numeric integration of the spectral density function.

Usage

data(acvs.andel18)
data(acvs.andel19)
data(acvs.andel110)
data(acvs.andel111)

Format

A data frame with 4096 rows and three columns: lag, autocovariance sequence, autocorrelation sequence.

References

**ar1**

*Simulated AR(1) Series*

**Description**
Simulated AR(1) series used in Gencay, Selcuk and Whitcher (2001).

**Usage**
```
data(ar1)
```

**Format**
A vector containing 200 observations.

**References**

---

**Band-pass variance**

*Bandpass Variance for Long-Memory Processes*

**Description**
Computes the band-pass variance for fractional difference (FD) or seasonal persistent (SP) processes using numeric integration of their spectral density function.

**Usage**
```
bandpass.fdp(a, b, d)
bandpass.spp(a, b, d, fG)
bandpass.spp2(a, b, d1, f1, d2, f2)
bandpass.var.spp(delta, fG, J, Basis, Length)
```

**Arguments**
- `a` : Left-hand boundary for the definite integral.
- `b` : Right-hand boundary for the definite integral.
- `d, delta, d1, d2` : Fractional difference parameter.
- `fG, f1, f2` : Gegenbauer frequency.
- `J` : Depth of the wavelet transform.
- `Basis` : Logical vector representing the adaptive basis.
- `Length` : Number of elements in `Basis`.
Details

See references.

Value

Band-pass variance for the FD or SP process between $a$ and $b$.

Author(s)

Brandon Whitcher

References


---

**Description**

The Barbara image comes from Allen Gersho’s lab at the University of California, Santa Barbara.

**Usage**

data(barbara)

**Format**

A $256 \times 256$ matrix.

**Source**

Internet.
Produce Boolean Vector from Wavelet Basis Names

Description

Produce a vector of zeros and ones from a vector of basis names.

Usage

\[
\text{basis}(x, \text{basis.names})
\]

Arguments

- \(x\): Output from the discrete wavelet package transform (DWPT).
- \(\text{basis.names}\): Vector of character strings that describe leaves on the DWPT basis tree. See the examples below for appropriate syntax.

Details

None.

Value

Vector of zeros and ones.

See Also

dwpt.

Examples

data(acvs.andel8)
## Not run:
x <- hosking.sim(1024, acvs.andel8[,2])
x.dwpt <- dwpt(x, "la8", 7)
## Select orthonormal basis from wavelet packet tree
x.basis <- basis(x.dwpt, c("w1.1","w2.1","w3.0","w4.3","w5.4","w6.10",
    "w7.22","w7.23"))
for(i in 1:length(x.dwpt))
x.dwpt[[i]] <- x.basis[i] * x.dwpt[[i]]
## Reconstruct original series using selected orthonormal basis
y <- idwpt(x.dwpt, x.basis)
par(mfrow=c(2,1), mar=c(5-1,4,4-1,2))
plot.ts(x, xlab="", ylab="", main="Original Series")
plot.ts(y, xlab="", ylab="", main="Reconstructed Series")
## End(Not run)
blocks

A Piecewise-Constant Function

Description

\[ \text{blocks}(x) = \sum_{j=1}^{11} \left(1 + \text{sign}(x - p_j)\right) h_j / 2 \]

Usage

data(blocks)

Format

A vector containing 512 observations.

Source

S+WAVELETS.

References


brick.wall

Replace Boundary Wavelet Coefficients with Missing Values

Description

Sets the first \( n \) wavelet coefficients to NA.

Usage

brick.wall(x, wf, method="modwt")
dwpt.brick.wall(x, wf, n.levels, method="modwpt")

Arguments

\( x \)  
DWT/MODWT/DWPT/MODWPT object

\( wf \)  
Character string: name of wavelet filter

\( \text{method} \)  
Either \text{dwt} or \text{modwt} for \text{brick.wall}, or either \text{dwpt} or \text{modwpt} for \text{dwpt.brick.wall}

\( \text{n.levels} \)  
depth of the wavelet transform
Details
The fact that observed time series are finite causes boundary issues. One way to get around this is to simply remove any wavelet coefficient computed involving the boundary. This is done here by replacing boundary wavelet coefficients with NA.

Value
Same object as x only with some missing values.

Author(s)
B. Whitcher

References


---

convolve2D

Fast Column-wise Convolution of a Matrix

Description
Use the Fast Fourier Transform to perform convolutions between a sequence and each column of a matrix.

Usage
convolve2D(x, y, conj = TRUE, type = c("circular", "open"))

Arguments
x \( M \times N \) matrix.
y numeric sequence of length \( N \).
conj logical; if TRUE, take the complex conjugate before back-transforming (default, and used for usual convolution).
type character; one of circular, open (beginning of word is ok). For circular, the two sequences are treated as circular, i.e., periodic. For open and filter, the sequences are padded with zeros (from left and right) first; filter returns the middle sub-vector of open, namely, the result of running a weighted mean of \( x \) with weights \( y \).
Details

This is a corrupted version of convolve made by replacing fft with mvfft in a few places. It would be nice to submit this to the R Developers for inclusion.

Author(s)

Brandon Whitcher

See Also

convolve

---

cpi |  U.S. Consumer Price Index
---

Description


Usage

data(cpi)

Format

A vector containing 624 observations.

Source

Unknown.

References

dau  

*Digital Photograph of Ingrid Daubechies*

**Description**

A digital photograph of Ingrid Daubechies taken at the 1993 AMS winter meetings in San Antonio, Texas. The photograph was taken by David Donoho with a Canon XapShot video still frame camera.

**Usage**

```r
data(dau)
```

**Format**

A $256 \times 256$ matrix.

**Source**

S+WAVELETS.

**References**


denoise.2d  

*Denoise an Image via the 2D Discrete Wavelet Transform*

**Description**

Perform simple de-noising of an image using the two-dimensional discrete wavelet transform.

**Usage**

```r
denoise.dwt.2d(x, wf = "la8", J = 4, method = "universal", H = 0.5, noise.dir = 3, rule = "hard")
denoise.modwt.2d(x, wf = "la8", J = 4, method = "universal", H = 0.5, rule = "hard")
```
Arguments

- `x`  
  input matrix (image)
- `wf`  
  name of the wavelet filter to use in the decomposition
- `J`  
  depth of the decomposition, must be a number less than or equal to $\log_2(\min\{M, N\})$
- `method`  
  character string describing the threshold applied, only "universal" and "long-memory" are currently implemented
- `H`  
  self-similarity or Hurst parameter to indicate spectral scaling, white noise is 0.5
- `noise.dir`  
  number of directions to estimate background noise standard deviation, the default is 3 which produces a unique estimate of the background noise for each spatial direction
- `rule`  
  either a "hard" or "soft" thresholding rule may be used

Details

See Thresholding.

Value

Image of the same dimension as the original but with high-frequency fluctuations removed.

Author(s)

B. Whitcher

References

See Thresholding for references concerning de-noising in one dimension.

See Also

Thresholding

Examples

```r
## Xbox image
data(xbox)
n <- NROW(xbox)
xbox.noise <- xbox + matrix(rnorm(n*n, sd=.15), n, n)
par(mfrow=c(2,2), cex=.8, pty="s")
image(xbox.noise, col=rainbow(128), main="Original Image")
image(denoise.dwt.2d(xbox.noise, wf="haar"), col=rainbow(128),
     zlim=range(xbox.noise), main="Denoised image")
image(xbox.noise - denoise.dwt.2d(xbox.noise, wf="haar"), col=rainbow(128),
     zlim=range(xbox.noise), main="Residual image")
```

```r
## Daubechies image
data(dau)
n <- NROW(dau)
dau.noise <- dau + matrix(rnorm(n*n, sd=10), n, n)
```
par(mfrow=c(2,2), cex=.8, pty="s")
image(dau.noise, col=rainbow(128), main="Original Image")
dau.denoise <- denoise.modwt.2d(dau.noise, wf="d4", rule="soft")
image(dau.denoise, col=rainbow(128), zlim=range(dau.noise),
     main="Denoised image")
image(dau.noise - dau.denoise, col=rainbow(128), main="Residual image")

---

doppler

*Sinusoid with Changing Amplitude and Frequency*

### Description

\[
doppler(x) = \sqrt{x(1 - x)} \sin \left( \frac{2.1\pi}{x + 0.05} \right)
\]

### Usage

```r
data(doppler)
```

### Format

A vector containing 512 observations.

### Source

S+WAVELETS.

### References


---

dpss.taper

*Calculating Thomson's Spectral Multitapers by Inverse Iteration*

### Description

The following function links the subroutines in "bell-p-w.o" to an R function in order to compute discrete prolate spheroidal sequences (dpss).

### Usage

```r
dpss.taper(n, k, nw = 4, nmax = 2^(ceiling(log(n, 2))))
```
**Arguments**

- **n**: length of data taper(s)
- **k**: number of data tapers; 1, 2, 3, ... (do not use 0!)
- **nw**: product of length and half-bandwidth parameter (w)
- **nmax**: maximum possible taper length, necessary for FORTRAN code

**Details**

Spectral estimation using a set of orthogonal tapers is becoming widely used and appreciated in scientific research. It produces direct spectral estimates with more than 2 df at each Fourier frequency, resulting in spectral estimators with reduced variance. Computation of the orthogonal tapers from the basic defining equation is difficult, however, due to the instability of the calculations – the eigenproblem is very poorly conditioned. In this article the severe numerical instability problems are illustrated and then a technique for stable calculation of the tapers – namely, inverse iteration – is described. Each iteration involves the solution of a matrix equation. Because the matrix has Toeplitz form, the Levinson recursions are used to rapidly solve the matrix equation. FORTRAN code for this method is available through the Statlib archive. An alternative stable method is also briefly reviewed.

**Value**

- **v**: matrix of data tapers (cols = tapers)
- **eigen**: eigenvalue associated with each data taper
- **iter**: total number of iterations performed
- **n**: same as input
- **w**: half-bandwidth parameter
- **ifault**: 0 indicates success, see documentation for “bell-p-w” for information on non-zero values

**Author(s)**

B. Whitcher

**References**


**See Also**

*sine.taper.*
Dual-tree Filter Banks

Filter Banks for Dual-Tree Wavelet Transforms

Description

Analysis and synthesis filter banks used in dual-tree wavelet algorithms.

Usage

\[
\begin{align*}
&\text{afb}(x, \text{af}) \\
&\text{afb2D}(x, \text{af1}, \text{af2} = \text{NULL}) \\
&\text{afb2D.A}(x, \text{af}, d) \\
&\text{sfb}(\text{lo}, \text{hi}, \text{sf}) \\
&\text{sfb2D}(\text{lo}, \text{hi}, \text{sf1}, \text{sf2} = \text{NULL}) \\
&\text{sfb2D.A}(\text{lo}, \text{hi}, \text{sf}, d)
\end{align*}
\]

Arguments

- **x**: vector or matrix of observations
- **af**: analysis filters. First element of the list is the low-pass filter, second element is the high-pass filter.
- **af1, af2**: analysis filters for the first and second dimension of a 2D array.
- **sf**: synthesis filters. First element of the list is the low-pass filter, second element is the high-pass filter.
- **sf1, sf2**: synthesis filters for the first and second dimension of a 2D array.
- **d**: dimension of filtering (d = 1 or 2)
- **lo**: low-frequency coefficients
- **hi**: high-frequency coefficients

Details

The functions \text{afb2D.A} and \text{sfb2D.A} implement the convolutions, either for analysis or synthesis, in one dimension only. Thus, they are the workhorses of \text{afb2D} and \text{sfb2D}. The output for the analysis filter bank along one dimension (\text{afb2D.A}) is a list with two elements

- **lo**: low-pass subband
- **hi**: high-pass subband

where the dimension of analysis will be half its original length. The output for the synthesis filter bank along one dimension (\text{sfb2D.A}) will be the output array, where the dimension of synthesis will be twice its original length.
Value

In one dimension the output for the analysis filter bank (afb) is a list with two elements

- `lo` Low frequency output
- `hi` High frequency output

and the output for the synthesis filter bank (sfb) is the output signal.

In two dimensions the output for the analysis filter bank (afb2D) is a list with four elements

- `lo` low-pass subband
- `hi[[1]]` 'lohi‘ subband
- `hi[[2]]` 'hilo' subband
- `hi[[3]]` 'hihi' subband

and the output for the synthesis filter bank (sfb2D) is the output array.

Author(s)

Matlab: S. Cai, K. Li and I. Selesnick; R port: B. Whitcher

References

WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
http://taco.poly.edu/WaveletSoftware/

Examples

```r
## EXAMPLE: afb, sfb
af = farras()$af
sf = farras()$sf
x = rnorm(64)
x.afb = afb(x, af)
lo = x.sfb$lo
hi = x.sfb$hi
y = sfb(lo, hi, sf)
err = x - y
max(abs(err))

## EXAMPLE: afb2D, sfb2D
x = matrix(rnorm(32*64), 32, 64)
af = farras()$af
sf = farras()$sf
x.afb2D = afb2D(x, af)
lo = x.sfb2D$lo
hi = x.sfb2D$hi
y = sfb2D(lo, hi, sf, sf)
err = x - y
max(abs(err))

## Example: afb2D.A, sfb2D.A
```

```r
```
dualfilt1

Kingsbury’s Q-filters for the Dual-Tree Complex DWT

Description

Kingsbury’s Q-filters for the dual-tree complex DWT.

Usage

dualfilt1()

Arguments

None.

Details

These coefficients are rounded to 8 decimal places.

Value

<table>
<thead>
<tr>
<th>af</th>
<th>List ((i = 1, 2)) - analysis filters for tree (i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf</td>
<td>List ((i = 1, 2)) - synthesis filters for tree (i)</td>
</tr>
</tbody>
</table>

Note: \(af[[2]]\) is the reverse of \(af[[1]]\).

Author(s)

Matlab: S. Cai, K. Li and I. Selesnick; R port: B. Whitcher

References


WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
http://taco.poly.edu/WaveletSoftware/

See Also

dualtree
Dualtree Complex Discrete Wavelet Transform

Description

One- and two-dimensional dual-tree complex discrete wavelet transforms developed by Kingsbury and Selesnick et al.

Usage

dualtree(x, J, Faf, af)
idualtree(w, J, Fsfsf)
dualtree2D(x, J, Faf, af)
idualtree2D(w, J, Fsfsf)

Arguments

x \(N\)-point vector or \(M \times N\) matrix.
w DWT coefficients.
J number of stages.
Faf analysis filters for the first stage.
af analysis filters for the remaining stages.
Fsfsf synthesis filters for the last stage.
sf synthesis filters for the preceding stages.

Details

In one dimension \(N\) is divisible by \(2^J\) and \(N \geq 2^{J-1} \cdot \text{length}(af)\).
In two dimensions, these two conditions must hold for both \(M\) and \(N\).

Value

For the analysis of \(x\), the output is

\(w\) DWT coefficients. Each wavelet scale is a list containing the real and imaginary parts. The final scale \((J + 1)\) contains the low-pass filter coefficients.

For the synthesis of \(w\), the output is

\(y\) output signal

Author(s)

Matlab: S. Cai, K. Li and I. Selesnick; R port: B. Whitcher
References

WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY

http://taco.poly.edu/WaveletSoftware/

See Also

FSfarras, farras, convolve, cshift, afb, sfb.

Examples

```r
c C C example: dualtree
x = rnorm(512)
J = 4
Faf = FSfarras()$af
Fs = FSfarras()$sf
af = dualfilt1()$af
sf = dualfilt1()$sf
w = dualtree(x, J, Faf, af)
y = idualtree(w, J, Fs, sf)
err = x - y
max(abs(err))

c C C example: dualtree2D
x = matrix(rnorm(64*64), 64, 64)
J = 3
Faf = FSfarras()$af
Fs = FSfarras()$sf
af = dualfilt1()$af
sf = dualfilt1()$sf
w = dualtree2D(x, J, Faf, af)
y = idualtree2D(w, J, Fs, sf)
err = x - y
max(abs(err))

C C display 2D wavelets of dualtree2D.m

J <- 4
L <- 3 * 2^(J+1)
N <- L / 2^J
Faf <- FSfarras()$af
Fs <- FSfarras()$sf
af <- dualfilt1()$af
sf <- dualfilt1()$sf
x <- matrix(rep(0, 2*L, 3*L), L)
w[[J]][[1]][[1]][[1]][[1]][1][N/2, N/2+0:N] <- 1
w[[J]][[1]][[2]][[1]][[1]][2][N/2, N/2+1:N] <- 1
w[[J]][[1]][[3]][[1]][[1]][3][N/2, N/2+2:N] <- 1
w[[J]][[2]][[1]][[1]][[1]][2][N+2, N/2+0:N] <- 1
w[[J]][[2]][[2]][[1]][[1]][2][N+2, N/2+1:N] <- 1
w[[J]][[2]][[3]][[1]][[1]][2][N+2, N/2+2:N] <- 1
y <- idualtree2D(w, J, Fs, sf)
```

```
image(t(y), col=grey(0:64/64), axes=FALSE)

---

**Description**

Dual-tree complex 2D discrete wavelet transform (DWT).

**Usage**

cplxdual2D(x, J, Faf, af)
icplxdual2D(w, J, Fs, sf)

**Arguments**

- `x`: 2D array.
- `w`: wavelet coefficients.
- `J`: number of stages.
- `Faf`: first stage analysis filters for tree `i`.
- `af`: analysis filters for the remaining stages on tree `i`.
- `Fs`: last stage synthesis filters for tree `i`.
- `sf`: synthesis filters for the preceding stages.

**Value**

For the analysis of `x`, the output is

```
W = wavelet coefficients indexed by [[j][i][d1][d2]], where j = 1, ..., J (scale), i = 1 (real part) or i = 2 (imag part), d1 = 1, 2 and d2 = 1, 2, 3 (orientations).
```

For the synthesis of `w`, the output is

```
Y = output signal.
```

**Author(s)**

Matlab: S. Cai, K. Li and I. Selesnick; R port: B. Whitcher

**References**

WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
http://taco.poly.edu/WaveletSoftware/

**See Also**

`Fsfarras, farras, afb2D, sfb2D`
Examples

```r
## Not run:
## EXAMPLE: cplxdual2D
x = matrix(rnorm(32*32), 32, 32)
J = 5
Faf = FSfarras()$af
Fsf = FSfarras()$sf
af = dualfilt1()$af
sf = dualfilt1()$sf
w = cplxdual2D(x, J, Faf, af)
y = icplxdual2D(w, J, Fsf, sf)
err = x - y
max(abs(err))
## End(Not run)
```

### dwpt

**Inverse Discrete Wavelet Packet Transforms**

**Description**

All possible filtering combinations (low- and high-pass) are performed to decompose a vector or time series. The resulting coefficients are associated with a binary tree structure corresponding to a partitioning of the frequency axis.

**Usage**

```r
dwpt(x, wf="la8", n.levels=4, boundary="periodic")
idwpt(y, y.basis)
modwpt(x, wf = "la8", n.levels = 4, boundary = "periodic")
```

**Arguments**

- `x`: a vector or time series containing the data to be decomposed. This must be a dyadic length vector (power of 2).
- `wf`: Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length $L = 8$ (Daubechies, 1992), least asymmetric family.
- `n.levels`: Specifies the depth of the decomposition. This must be a number less than or equal to log2(length(x)).
- `boundary`: Character string specifying the boundary condition. If boundary="periodic" the default, then the vector you decompose is assumed to be periodic on its defined interval, if boundary="reflection", the vector beyond its boundaries is assumed to be a symmetric reflection of itself.
- `y`: Object of S3 class `dwpt`.
- `y.basis`: Vector of character strings that describe leaves on the DWPT basis tree.
Details

The code implements the one-dimensional DWPT using the pyramid algorithm (Mallat, 1989).

Value

Basically, a list with the following components

- `w?`: Wavelet coefficient vectors. The first index is associated with the scale of the decomposition while the second is associated with the frequency partition within that level.
- `wavelet`: Name of the wavelet filter used.
- `boundary`: How the boundaries were handled.

Author(s)

B. Whitcher

References


See Also

dwt, modwpt, wave.filter.

Examples

data(mexm)
J <- 4
mexm.mra <- mra(log(mexm), "mb8", J, "modwt", "reflection")
mexm.nmean <- ts(
  apply(matrix(unlist(mexm.mra), ncol=J+1, byrow=FALSE)[,-(J+1)], 1, sum),
  start=1957, freq=12)
mexm.dwpt <- dwpt(mexm.nmean[-c(1:4)], "mb8", 7, "reflection")
(Inverse) Discrete Wavelet Packet Transforms in Two Dimensions

Description

All possible filtering combinations (low- and high-pass) are performed to decompose a matrix or image. The resulting coefficients are associated with a quad-tree structure corresponding to a partitioning of the two-dimensional frequency plane.

Usage

dwpt.2d(x, wf="la8", J=4, boundary="periodic")
idwpt.2d(y, y.basis)

Arguments

x a matrix or image containing the data be to decomposed. This object must be dyadic (power of 2) in length in each dimension.
wf Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length \( L = 8 \) (Daubechies, 1992), least asymmetric family.
J Specifies the depth of the decomposition. This must be a number less than or equal to \( \log(\text{length}(x), 2) \).
boundary Character string specifying the boundary condition. If boundary="periodic" the default, then the vector you decompose is assumed to be periodic on its defined interval, if boundary="reflection", the vector beyond its boundaries is assumed to be a symmetric reflection of itself.
y dwpt.2d object (list-based structure of matrices)
y.basis Boolean vector, the same length as y, where TRUE means the basis tensor should be used in the reconstruction.

Details

The code implements the two-dimensional DWPT using the pyramid algorithm of Mallat (1989).

Value

Basically, a list with the following components

w? N? W? N? Wavelet coefficient matrices (images). The first index is associated with the scale of the decomposition while the second is associated with the frequency partition within that level. The left and right strings, separated by the dash ‘-’, correspond to the first \((x)\) and second \((y)\) dimensions.
wavelet Name of the wavelet filter used.
boundary How the boundaries were handled.
Author(s)
B. Whitcher

References

See Also
dwt.2d, modwt.2d, wave.filter.

**Description**
An adaptive orthonormal basis is selected in order to perform the naive bootstrap within nodes of the wavelet packet tree. A bootstrap realization of the time series is produce by applying the inverse DWPT.

**Usage**
dwpt.boot(y, wf, J=log(length(y),2)-1, p=1e-04, frac=1)

**Arguments**
y Not necessarily dyadic length time series.
waf Name of the wavelet filter to use in the decomposition. See wave.filter for those wavelet filters available.
J Depth of the discrete wavelet packet transform.
p Level of significance for the white noise testing procedure.
frac Fraction of the time series that should be used in constructing the likelihood function.

**Details**
A subroutines is used to select an adaptive orthonormal basis for the piecewise-constant approximation to the underlying spectral density function (SDF). Once selected, sampling with replacement is performed within each wavelet packet coefficient vector and the new collection of wavelet packet coefficients are reconstructed into a bootstrap realization of the original time series.

**Value**
Time series of length $N$, where $N$ is the length of $y$. 
Author(s)

B. Whitcher

References


See Also

*dwpt.sim, spp.mle*

---

**Description**

A seasonal persistent process may be characterized by a spectral density function with an asymptote occurring at a particular frequency in \([0, \frac{1}{2})\). Its time domain representation was first noted in passing by Hosking (1981). Although an exact time-domain approach to simulation is possible, this function utilizes the discrete wavelet packet transform (DWPT).

**Usage**

dwpt.sim(N, wf, delta, fG, M=2, adaptive=TRUE, epsilon=0.05)

**Arguments**

- **N** Length of time series to be generated.
- **wf** Character string for the wavelet filter.
- **delta** Long-memory parameter for the seasonal persistent process.
- **fG** Gegenbauer frequency.
- **M** Actual length of simulated time series.
- **adaptive** Logical; if TRUE the orthonormal basis used in the DWPT is adapted to the ideal spectrum, otherwise the orthonormal basis is performed to a maximum depth.
- **epsilon** Threshold for adaptive basis selection.
Details

Two subroutines are used, the first selects an adaptive orthonormal basis for the true spectral density function (SDF) while the second computes the bandpass variances associated with the chosen orthonormal basis and SDF. Finally, when

\[ M > N \]

a uniform random variable is generated in order to select a random piece of the simulated time series. For more details see Whitcher (2001).

Value

Time series of length \( N \).

Author(s)

B. Whitcher

References


See Also

hosking.sim for an exact time-domain method and wave.filter for a list of available wavelet filters.

Examples

```r
## Generate monthly time series with annual oscillation
## library(ts) is required in order to access acf()
x <- dwpt.sim(256, "mb16", .4, 1/12, M=4, epsilon=.001)
par(mfrow=c(2,1))
plot(x, type="l", xlab="Time")
acf(x, lag.max=128, ylim=c(-.6,1))
data(acvs.andel8)
lines(acvs.andel8$lag[1:128], acvs.andel8$acf[1:128], col=2)
```

---

**dwt**

*Discrete Wavelet Transform (DWT)*

**Description**

This function performs a level \( J \) decomposition of the input vector or time series using the pyramid algorithm (Mallat 1989).
Usage

dwt(x, wf="la8", n.levels=4, boundary="periodic")
dwt.nondyadic(x)

Arguments

x  a vector or time series containing the data be to decomposed. This must be a dyadic length vector (power of 2).
wf  Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length $L = 8$ (Daubechies, 1992), least asymmetric family.
n.levels  Specifies the depth of the decomposition. This must be a number less than or equal to $\log_2(\text{length}(x))$.
boundary  Character string specifying the boundary condition. If boundary="periodic", the default, then the vector you decompose is assumed to be periodic on its defined interval, if boundary="reflection", the vector beyond its boundaries is assumed to be a symmetric reflection of itself.

Details

The code implements the one-dimensional DWT using the pyramid algorithm (Mallat, 1989). The actual transform is performed in C using pseudocode from Percival and Walden (2001). That means convolutions, not inner products, are used to apply the wavelet filters.

For a non-dyadic length vector or time series, dwt.nondyadic pads with zeros, performs the orthonormal DWT on this dyadic length series and then truncates the wavelet coefficient vectors appropriately.

Value

Basically, a list with the following components

- d?  Wavelet coefficient vectors.
- s?  Scaling coefficient vector.
- wavelet  Name of the wavelet filter used.
- boundary  How the boundaries were handled.

Author(s)

B. Whitcher

References


See Also

`modwt`, `mra`.

Examples

```r
data(ibm)
diff = diff(log(ibm))
## Haar
ibmr.haar <- dwt(diff, "haar")
names(ibmr.haar) <- c("w1", "w2", "w3", "w4", "v4")
## plot partial Haar DWT for IBM data
par(mfcol=c(6,1), pty="m", mar=c(5-2,4-2,2,2))
plot.ts(diff, axes=FALSE, ylab="", main="(a)"
for(i in 1:4)
  plot.ts(up.sample(ibmr.haar[[i]], 2^i), type="h", axes=FALSE, ylab="names(ibmr.haar)[i]"
plot.ts(up.sample(ibmr.haar$v4, 2^4), type="h", axes=FALSE, ylab="names(ibmr.haar)[5]"
axis(side=1, at=seq(0,368,by=23),
  labels=c(0,"","",46,"","",92,"","",138,"","",184,"","",230,"","",276,"","",322,"","",368))
## L(8)
ibmr.la8 <- dwt(diff, "la8")
names(ibmr.la8) <- c("w1", "w2", "w3", "w4", "v4")
## must shift LA(8) coefficients
ibmr.la8$w1 <- c(ibmr.la8$w1[-c(1:2)], ibmr.la8$w1[1:2])
ibmr.la8$w2 <- c(ibmr.la8$w2[-c(1:2)], ibmr.la8$w2[1:2])
for(i in names(ibmr.la8)[3:4])
  ibmr.la8[[i]] <- c(ibmr.la8[[i]][-c(1:3)], ibmr.la8[[i]][1:3])
ibmr.la8$v4 <- c(ibmr.la8$v4[-c(1:2)], ibmr.la8$v4[1:2])
## plot partial LA(8) DWT for IBM data
par(mfcol=c(6,1), pty="m", mar=c(5-2,4-2,2,2))
plot.ts(diff, axes=FALSE, ylab="", main="(b)"
for(i in 1:4)
  plot.ts(up.sample(ibmr.la8[[i]], 2^i), type="h", axes=FALSE, ylab="names(ibmr.la8)[i]"
plot.ts(up.sample(ibmr.la8$v4, 2^4), type="h", axes=FALSE, ylab="names(ibmr.la8)[5]"
axis(side=1, at=seq(0,368,by=23),
  labels=c(0,"","",46,"","",92,"","",138,"","",184,"","",230,"","",276,"","",322,"","",368))
```
Two-Dimensional Discrete Wavelet Transform

Description

Performs a separable two-dimensional discrete wavelet transform (DWT) on a matrix of dyadic dimensions.

Usage

dwt.2d(x, wf, J = 4, boundary = "periodic")
idwt.2d(y)

Arguments

- **x**: input matrix (image)
- **wf**: name of the wavelet filter to use in the decomposition
- **J**: depth of the decomposition, must be a number less than or equal to \( \log_2(\min\{M, N\}) \)
- **boundary**: only "periodic" is currently implemented
- **y**: an object of class dwt.2d

Details

See references.

Value

List structure containing the \( 3J + 1 \) sub-matrices from the decomposition.

Author(s)

B. Whitcher

References


See Also

modwt.2d.
**Examples**

```r
data(xbox)
xbox.dwt <- dwt.2d(xbox, "haar", 3)
par(mfrow=c(1,1), pty="s")
plot.dwt.2d(xbox.dwt)
par(mfrow=c(2,2), pty="s")
image(1:dim(xbox)[1], 1:dim(xbox)[2], xbox, xlab="", ylab="",
    main="Original Image")
image(1:dim(xbox)[1], 1:dim(xbox)[2], idwt.2d(xbox.dwt), xlab="", ylab="",
    main="Wavelet Reconstruction")
image(1:dim(xbox)[1], 1:dim(xbox)[2], xbox - idwt.2d(xbox.dwt),
    xlab="", ylab="", main="Difference")

# Daubechies image
data(dau)
par(mfrow=c(1,1), pty="s")
image(dau, col=rainbow(128))
sum(dau^2)
dau.dwt <- dwt.2d(dau, "d4", 3)
plot.dwt.2d(dau.dwt)
sum(plot.dwt.2d(dau.dwt, plot=FALSE)^2)
```

---

**dwt.3d**  
*Three Dimensional Separable Discrete Wavelet Transform*

**Description**

Three-dimensional separable discrete wavelet transform (DWT).

**Usage**

```r
dwt.3d(x, wf, J=4, boundary="periodic")
idwt.3d(y)
```

**Arguments**

- `x` input array
- `wf` name of the wavelet filter to use in the decomposition
- `J` depth of the decomposition, must be a number less than or equal to \(\log_2(\min\{X,Y,Z\})\)
- `boundary` only "periodic" is currently implemented
- `y` an object of class dwt.3d

**Author(s)**

B. Whitcher
Exchange Rates Between the Deutsche Mark, Japanese Yen and U.S. Dollar

**Description**

Monthly foreign exchange rates for the Deutsche Mark - U.S. Dollar (DEM-USD) and Japanese Yen - U.S. Dollar (JPY-USD) starting in 1970.

**Usage**

data(exchange)

**Format**

A bivariate time series containing 348 observations.

**Source**

Unknown.

**References**


Farras nearly symmetric filters

**Description**

Farras nearly symmetric filters for orthogonal 2-channel perfect reconstruction filter bank and Farras filters organized for the dual-tree complex DWT.

**Usage**

farras()
FSfarras()

**Arguments**

None.

**Value**

- **af** List \((i = 1, 2)\) - analysis filters for tree \(i\)
- **sf** List \((i = 1, 2)\) - synthesis filters for tree \(i\)
Author(s)
Matlab: S. Cai, K. Li and I. Selesnick; R port: Brandon Whitcher

References
WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
http://taco.poly.edu/WaveletSoftware/

See Also
afb, dualtree, dualfilt1.

fdp.mle Wavelet-based Maximum Likelihood Estimation for a Fractional Difference Process

Description
Parameter estimation for a fractional difference (long-memory, self-similar) process is performed via maximum likelihood on the wavelet coefficients.

Usage
fdp.mle(y, wf, J=log(length(y),2))

Arguments
y Dyadic length time series.
wf Name of the wavelet filter to use in the decomposition. See wave.filter for those wavelet filters available.
J Depth of the discrete wavelet transform.

Details
The variance-covariance matrix of the original time series is approximated by its wavelet-based equivalent. A Whittle-type likelihood is then constructed where the sums of squared wavelet coefficients are compared to bandpass filtered version of the true spectrum. Minimization occurs only for the fractional difference parameter $d$, while variance is estimated afterwards.

Value
List containing the maximum likelihood estimates (MLEs) of $d$ and $\sigma^2$, along with the value of the likelihood for those estimates.
find.adaptive.basis

**Author(s)**

B. Whitcher

**References**


**Examples**

```r
## Figure 5.5 in Gencay, Selcuk and Whitcher (2001)
fdp.sdf <- function(freq, d, sigma2=1)
    sigma2 / ((2*sin(pi * freq))^2)*d
dB <- function(x) 10 * log10(x)
per <- function(z) {
    n <- length(z)
    (Mod(fft(z))**2/(2*pi*n))[1:(n %% 2 + 1)]
}
data(ibm)
ibmreturns <- diff(log(ibm))
ibm.volatility <- abs(ibmreturns)
ibm.vol.mle <- fdp.mle(ibm.volatility, "d4", 4)
freq <- 0:184/368
ibm.vol.per <- 2 * pi * per(ibm.volatility)
ibm.vol.resid <- ibm.vol.per/ fdp.sdf(freq, ibm.vol.mle$parameters[1])
par(mfrow=c(1,1), las=0, pty="m")
plot(freq, dB(ibm.vol.per), type="l", xlab="frequency", ylab="Spectrum")
lines(freq, dB(fdp.sdf(freq, ibm.vol.mle$parameters[1],
    ibm.vol.mle$parameters[2]/2)), col=2)
```

**find.adaptive.basis**

Determine an Orthonormal Basis for the Discrete Wavelet Packet Transform

**Description**

Subroutine for use in simulating seasonal persistent processes using the discrete wavelet packet transform.

**Usage**

```r
find.adaptive.basis(wf, J, fG, eps)
```
heavisine

Arguments

- **wf**: Character string; name of the wavelet filter.
- **J**: Depth of the discrete wavelet packet transform.
- **fG**: Gegenbauer frequency.
- **eps**: Threshold for the squared gain function.

Details

The squared gain functions for a Daubechies (extremal phase or least asymmetric) wavelet family are used in a filter cascade to compute the value of the squared gain function for the wavelet packet filter at the Gegenbauer frequency. This is done for all nodes of the wavelet packet table.

The idea behind this subroutine is to approximate the relationship between the discrete wavelet transform and long-memory processes, where the squared gain function is zero at frequency zero for all levels of the DWT.

Value

Boolean vector describing the orthonormal basis for the DWPT.

Author(s)

B. Whitcher

See Also

Used in `dwpt.sim`.

Description

heavisine(x) = 4\sin(4\pi x) − \text{sign}(x − 0.3) − \text{sign}(0.72 − x)

Usage

data(heavisine)

Format

A vector containing 512 observations.

Source

S+WAVELETS.
References


---

Hilbert

**Discrete Hilbert Wavelet Transforms**

**Description**

The discrete Hilbert wavelet transforms (DHWTs) for seasonal and time-varying time series analysis. Transforms include the usual orthogonal (decimated), maximal-overlap (non-decimated) and maximal-overlap packet transforms.

**Usage**

```r
# dwt.hilbert(x, wf, n.levels=4, boundary="periodic", ...)
# dwt.hilbert.nondyadic(x, ...)
# idwt.hilbert(y)
# modwt.hilbert(x, wf, n.levels=4, boundary="periodic", ...)
# imodwt.hilbert(y)
# modwpt.hilbert(x, wf, n.levels=4, boundary="periodic")
```

**Arguments**

- `x`: Real-valued time series or vector of observations.
- `wf`: Hilbert wavelet pair
- `n.levels`: Number of levels (depth) of the wavelet transform.
- `boundary`: Boundary treatment, currently only periodic and reflection.
- `y`: Hilbert wavelet transform object (list).
- `...`: Additional parameters to be passed on.

**Author(s)**

B. Whitcher

**References**


**See Also**

`hilbert.filter`
**Description**

Converts name of Hilbert wavelet pair to filter coefficients.

**Usage**

hilbert.filter(name)

**Arguments**

name  
Character string of Hilbert wavelet pair, see acceptable names below (e.g., "k313").

**Details**

Simple switch statement selects the appropriate HWP. There are two parameters that define a Hilbert wavelet pair using the notation of Selesnick (2001,2002), \(K\) and \(L\). Currently, the only implemented combinations \((K,L)\) are (3,3), (3,5), (4,2) and (4,4).

**Value**

List containing the following items:

- \(L\)  
  length of the wavelet filter
- \(h0, g0\)  
  low-pass filter coefficients
- \(h1, g1\)  
  high-pass filter coefficients

**Author(s)**

B. Whitcher

**References**


**See Also**

wave.filter
Examples

hilbert.filter("k313")
hilbert.filter("k315")
hilbert.filter("k412")
hilbert.filter("k414")

Description

Uses exact time-domain method from Hosking (1984) to generate a simulated time series from a specified autocovariance sequence.

Usage

hosking.sim(n, acvs)

Arguments

n Length of series.
acvs Autocovariance sequence of series with which to generate, must be of length at least n.

Value

Length n time series from true autocovariance sequence acvs.

Author(s)

Brandon Whitcher

References


Examples

dB <- function(x) 10 * log10(x)
der <- function (z) {
  n <- length(z)
  (Mod(fft(z))^2/(2 * pi * n))[1:(n/2 + 1)]
}
spp.sdf <- function(freq, delta, omega)
  abs(2 * (cos(2*pi*freq) - cos(2*pi*omega)))^(-2*delta)
HWP Analysis

Time-varying and Seasonal Analysis Using Hilbert Wavelet Pairs

Description

Performs time-varying or seasonal coherence and phase analysis between two time series using the maximal-overlap discrete Hilbert wavelet transform (MODHWT).

Usage

```r
modhwt.coh(x, y, f.length = 0)
modhwt.phase(x, y, f.length = 0)
modhwt.coh.seasonal(x, y, S = 10, season = 365)
modhwt.phase.seasonal(x, y, season = 365)
```

Arguments

- `x`: MODHWT object.
- `y`: MODHWT object.
- `f.length`: Length of the rectangular filter.
- `S`: Number of "seasons".
- `season`: Length of the "season".

Details

The idea of seasonally-varying spectral analysis (SVSA, Madden 1986) is generalized using the MODWT and Hilbert wavelet pairs. For the seasonal case, S seasons are used to produce a consistent estimate of the coherence and phase. For the non-seasonal case, a simple rectangular (moving-average) filter is applied to the MODHWT coefficients in order to produce consistent estimates.

Value

Time-varying or seasonal coherence and phase between two time series. The coherence estimates are between zero and one, while the phase estimates are between $-\pi$ and $\pi$. 

### Example

```r
data(acvs.andel8)
n <- 1024
## Not run:
z <- hosking.sim(n, acvs.andel8[,2])
per.z <- 2 * pi * per(z)
par(mfrow=c(2,1), las=1)
plot.ts(z, ylab="", main="Realization of a Seasonal Long-Memory Process")
plot(0:(n/2)/n, db(per.z), type="l", xlab="Frequency", ylab="dB", main="Periodogram")
lines(0:(n/2)/n, db(spp.sdf(0:(n/2)/n, .4, 1/12)), col=2)
## End(Not run)
```
Author(s)

B. Whitcher

References


See Also

hilbert.filter

---

**ibm**                      

*Daily IBM Stock Prices*

---

Description


Usage

```
data(ibm)
```

Format

A vector containing 369 observations.

Source


References

[http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/]
japan

Japanese Gross National Product

Description

Usage
data(japan)

Format
A vector containing 169 observations.

Source
Unknown.

References

jumpsine

Sine with Jumps at 0.625 and 0.875

Description

\[ jumpsine(x) = 10 \left( \sin(4\pi x) + I_{[0.625<x\leq0.875]} \right) \]

Usage
data(jumpsine)

Format
A vector containing 512 observations.

Source
S+WAVELETS.
References


---

**kobe**

1995 Kobe Earthquake Data

**Description**

Seismograph (vertical acceleration, nm/sq.sec) of the Kobe earthquake, recorded at Tasmania University, Hobart, Australia on 16 January 1995 beginning at 20:56:51 (GMTRUE) and continuing for 51 minutes at 1 second intervals.

**Usage**

data(kobe)

**Format**

A vector containing 3048 observations.

**Source**

Data management centre, Washington University.

**References**


---

**linchirp**

Linear Chirp

**Description**

\[ \text{linchirp}(x) = \sin(0.125\pi nx^2) \]

**Usage**

data(linchirp)

**Format**

A vector containing 512 observations.
mexm

Source

S+WAVELETS.

References


mexm  
*Mexican Money Supply*

Description

Percentage changes in monthly Mexican money supply.

Usage

data(mexm)

Format

A vector containing 516 observations.

Source

Unknown.

References


modwt

*(Inverse) Maximal Overlap Discrete Wavelet Transform*

Description

This function performs a level \( J \) decomposition of the input vector using the non-decimated discrete wavelet transform. The inverse transform performs the reconstruction of a vector or time series from its maximal overlap discrete wavelet transform.

Usage

```r
modwt(x, wf = "la8", n.levels = 4, boundary = "periodic")
imodwt(y)
```
Arguments

x    a vector or time series containing the data be to decomposed. There is no restriction on its length.
y    Object of class "modwt".
wf   Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length \( L = 8 \) (Daubechies, 1992), least asymmetric family.
n.levels Specifies the depth of the decomposition. This must be a number less than or equal to \( \log_2(\text{length}(x)) \).
boundary Character string specifying the boundary condition. If boundary="periodic" the default is TRUE, then the vector you decompose is assumed to be periodic on its defined interval, if boundary="reflection", the vector beyond its boundaries is assumed to be a symmetric reflection of itself.

Details

The code implements the one-dimensional non-decimated DWT using the pyramid algorithm. The actual transform is performed in C using pseudocode from Percival and Walden (2001). That means convolutions, not inner products, are used to apply the wavelet filters.

The MODWT goes by several names in the statistical and engineering literature, such as, the “stationary DWT”, “translation-invariant DWT”, and “time-invariant DWT”.

The inverse MODWT implements the one-dimensional inverse transform using the pyramid algorithm (Mallat, 1989).

Value

Object of class "modwt", basically, a list with the following components

d? Wavelet coefficient vectors.
s? Scaling coefficient vector.
wavelet Name of the wavelet filter used.
boundary How the boundaries were handled.

Author(s)

B. Whitcher

References


modwt.2d  Two-Dimensional Maximal Overlap Discrete Wavelet Transform

Description

Performs a separable two-dimensional maximal overlap discrete wavelet transform (MODWT) on a matrix of arbitrary dimensions.

Usage

modwt.2d(x, wf, J = 4, boundary = "periodic")
imodwt.2d(y)

Arguments

- x: input matrix
- wf: name of the wavelet filter to use in the decomposition
- J: depth of the decomposition

See Also
dwt, idwt, mra.

Examples

```r
## Figure 4.23 in Gencay, Selcuk and Whitcher (2001)
data(ibm)
ibm.runs <- diff(log(ibm))
# Haar
ibm.haar <- modwt(ibm.runs, "haar")
names(ibm.haar) <- c("w1", "w2", "w3", "w4", "v4")
# LA8
ibm.la8 <- modwt(ibm.runs, "la8")
names(ibm.la8) <- c("w1", "w2", "w3", "w4", "v4")
# shift the MODWT vectors
ibm.la8 <- phase.shift(ibm.la8, "la8")
## plot partial MODWT for IBM data
par(mfcol=c(2,1), pty="m", mar=c(5-2,4,4-2,2))
plot.ts(ibm.runs, axes=FALSE, ylab="", main="(a)"
for(i in 1:5)
  plot.ts(ibm.haar[[i]], axes=FALSE, ylab=names(ibm.haar)[i])
axis(side=1, at=seq(0,368,by=23),
  labels=c(0,"",46,"",92,"",138,"",184,"",230,"",276,"",322,"",368))
par(mfcol=c(2,1), pty="m", mar=c(5-2,4,4-2,2))
plot.ts(ibm.runs, axes=FALSE, ylab="", main="(b)"
for(i in 1:5)
  plot.ts(ibm.la8[[i]], axes=FALSE, ylab=names(ibm.la8)[i])
axis(side=1, at=seq(0,368,by=23),
  labels=c(0,"",46,"",92,"",138,"",184,"",230,"",276,"",322,"",368))
```
boundary

only "periodic" is currently implemented

y

an object of class dwt.2d

Details

See references.

Value

List structure containing the $3J + 1$ sub-matrices from the decomposition.

Author(s)

B. Whitcher

References


See Also

dwt.2d, shift.2d.

Examples

```r
## Xbox image
data(xbox)
xbox.modwt <- modwt.2d(xbox, "haar", 2)
## Level 1 decomposition
par(mfrow=c(2,2), pty="s")
image(xbox.modwt$LH1, col=rainbow(128), axes=FALSE, main="LH1")
image(xbox.modwt$HH1, col=rainbow(128), axes=FALSE, main="HH1")
frame()
image(xbox.modwt$HL1, col=rainbow(128), axes=FALSE, main="HL1")
## Level 2 decomposition
par(mfrow=c(2,2), pty="s")
image(xbox.modwt$LH2, col=rainbow(128), axes=FALSE, main="LH2")
image(xbox.modwt$HH2, col=rainbow(128), axes=FALSE, main="HH2")
image(xbox.modwt$LL2, col=rainbow(128), axes=FALSE, main="LL2")
image(xbox.modwt$HL2, col=rainbow(128), axes=FALSE, main="HL2")
sum((xbox - imodwt.2d(xbox.modwt))^2)

data(dau)
par(mfrow=c(1,1), pty="s")
image(dau, col=rainbow(128), axes=FALSE, main="Ingrid Daubechies")
sum(dau^2)
dau.modwt <- modwt.2d(dau, "d4", 2)
## Level 1 decomposition
```
modwt.3d

Three Dimensional Separable Maximal Overlap Discrete Wavelet Transform

Description

Three-dimensional separable maximal overlap discrete wavelet transform (MODWT).

Usage

modwt.3d(x, wf, J = 4, boundary = "periodic")

imodwt.3d(y)

Arguments

x          input array
wf         name of the wavelet filter to use in the decomposition
J          depth of the decomposition
boundary   only "periodic" is currently implemented
y          an object of class modwt.3d

Author(s)

B. Whitcher


**Description**

This function performs a level $J$ additive decomposition of the input vector or time series using the pyramid algorithm (Mallat 1989).

**Usage**

```r
mra(x, wf = "la8", J = 4, method = "modwt", boundary = "periodic")
```

**Arguments**

- **x**: A vector or time series containing the data to be decomposed. This must be a dyadic length vector (power of 2) for `method="dwt"`.
- **wf**: Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length $L = 8$ least asymmetric family.
- **J**: Specifies the depth of the decomposition. This must be a number less than or equal to $\log(\text{length}(x), 2)$.
- **method**: Either "dwt" or "modwt".
- **boundary**: Character string specifying the boundary condition. If `boundary="periodic"` the default, then the vector you decompose is assumed to be periodic on its defined interval, if `boundary="reflection"`, the vector beyond its boundaries is assumed to be a symmetric reflection of itself.

**Details**

This code implements a one-dimensional multiresolution analysis introduced by Mallat (1989). Either the DWT or MODWT may be used to compute the multiresolution analysis, which is an additive decomposition of the original time series.

**Value**

Basically, a list with the following components

- **D**: Wavelet detail vectors.
- **S**: Wavelet smooth vector.
- **wavelet**: Name of the wavelet filter used.
- **boundary**: How the boundaries were handled.

**Author(s)**

B. Whitcher
References


See Also
dwt, modwt.

Examples

## Easy check to see if it works...
x <- rnorm(32)
x.mra <- mra(x)
sum(x - apply(matrix(unlist(x.mra), nrow=32), 1, sum))^2

## Figure 4.19 in Gencay, Selcuk and Whitcher (2001)
data(ibm)
ibm.outputs <- diff(log(ibm))
ibm.volatility <- abs(ibm.outputs)

## Haar
ibmv.haar <- mra(ibm.volatility, "haar", 4, "dwt")
names(ibmv.haar) <- c("d1", "d2", "d3", "d4", "s4")

## La8
ibmv.la8 <- mra(ibm.volatility, "la8", 4, "dwt")
names(ibmv.la8) <- c("d1", "d2", "d3", "d4", "s4")

## plot multiresolution analysis of IBM data
par(mfcol=c(6,1), pty="m", mar=c(5-2,4,4-2,2))
plot.ts(ibm.volatility, axes=FALSE, ylab="", main="(a)"
for(i in 1:5)
  plot.ts(ibmv.haar[[i]], axes=FALSE, ylab=names(ibmv.haar)[i])
  axis(side=1, at=seq(0,368,by=23),
       labels=c(0,"",46,"",92,"",138,"",184,"",230,"",276,"",322,"",368))
par(mfcol=c(6,1), pty="m", mar=c(5-2,4,4-2,2))
plot.ts(ibm.volatility, axes=FALSE, ylab="", main="(b)"
for(i in 1:5)
  plot.ts(ibmv.la8[[i]], axes=FALSE, ylab=names(ibmv.la8)[i])
  axis(side=1, at=seq(0,368,by=23),
       labels=c(0,"",46,"",92,"",138,"",184,"",230,"",276,"",322,"",368))
Description

This function performs a level $J$ additive decomposition of the input matrix or image using the pyramid algorithm (Mallat 1989).

Usage

\[
\text{mra.2d}(x, \text{wf} = "la8", J = 4, \text{method} = "modwt", \text{boundary} = "periodic")
\]

Arguments

- **x**: A matrix or image containing the data to be decomposed. This must be have dyadic length in both dimensions (but not necessarily the same) for `method=dwt`
- **wf**: Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length \( L = 8 \) least asymmetric family.
- **J**: Specifies the depth of the decomposition. This must be a number less than or equal to \( \log(\text{length}(x), 2) \).
- **method**: Either "dwt" or "modwt".
- **boundary**: Character string specifying the boundary condition. If `boundary="periodic"` the default, then the matrix you decompose is assumed to be periodic on its defined interval, if `boundary="reflection"`, the matrix beyond its boundaries is assumed to be a symmetric reflection of itself.

Details

This code implements a two-dimensional multiresolution analysis by performing the one-dimensional pyramid algorithm (Mallat 1989) on the rows and columns of the input matrix. Either the DWT or MODWT may be used to compute the multiresolution analysis, which is an additive decomposition of the original matrix (image).

Value

Basically, a list with the following components

- **LH**: Wavelet detail image in the horizontal direction.
- **HL**: Wavelet detail image in the vertical direction.
- **HH**: Wavelet detail image in the diagonal direction.
- **LL.J**: Wavelet smooth image at the coarsest resolution.
- **J**: Depth of the wavelet transform.
- **wavelet**: Name of the wavelet filter used.
- **boundary**: How the boundaries were handled.

Author(s)

B. Whitcher
mra.3d

References


See Also

`dwt.2d, modwt.2d`

Examples

```r
## Easy check to see if it works...
## --------------------------------

x <- matrix(rnorm(32*32), 32, 32)
# MODWT
x.mra <- mra.2d(x, method="modwt")
x.mra.sum <- x.mra[[1]]
for(j in 2:length(x.mra))
  x.mra.sum <- x.mra.sum + x.mra[[j]]
sum((x - x.mra.sum)^2)

# DWT
x.mra <- mra.2d(x, method="dwt")
x.mra.sum <- x.mra[[1]]
for(j in 2:length(x.mra))
  x.mra.sum <- x.mra.sum + x.mra[[j]]
sum((x - x.mra.sum)^2)
```

-------

mra.3d

*Three Dimensional Multiresolution Analysis*

Description

This function performs a level $J$ additive decomposition of the input array using the pyramid algorithm (Mallat 1989).

Usage

```r
mra.3d(x, wf, J=4, method="modwt", boundary="periodic")
```

Arguments

- `x` A three-dimensional array containing the data to be decomposed. This must be have dyadic length in all three dimensions (but not necessarily the same) for method="dwt".
Name of the wavelet filter to use in the decomposition. By default this is set to "la8", the Daubechies orthonormal compactly supported wavelet of length $L = 8$ least asymmetric family.

Specifies the depth of the decomposition. This must be a number less than or equal to $\log(\text{length}(x), 2)$.

Either "dwt" or "modwt".

Character string specifying the boundary condition. If boundary="periodic" the default and only method implemented, then the matrix you decompose is assumed to be periodic on its defined interval.

This code implements a three-dimensional multiresolution analysis by performing the one-dimensional pyramid algorithm (Mallat 1989) on each dimension of the input array. Either the DWT or MODWT may be used to compute the multiresolution analysis, which is an additive decomposition of the original array.

List structure containing the filter triplets associated with the multiresolution analysis.

B. Whitcher


See Also

dwt.3d, modwt.3d

This is the major subroutine for testing.hov, providing the workhorse algorithm to recursively test and locate multiple variance changes in so-called long memory processes.

```
mult.loc(dwt.list, modwt.list, wf, level, min.coef, debug)
```
**Arguments**

- **dwt.list** List of wavelet vector coefficients from the `dwt`.
- **modwt.list** List of wavelet vector coefficients from the `modwt`.
- **wf** Name of the wavelet filter to use in the decomposition.
- **level** Specifies the depth of the decomposition.
- **min.coef** Minimum number of wavelet coefficients for testing purposes.
- **debug** Boolean variable: if set to TRUE, actions taken by the algorithm are printed to the screen.

**Details**

For details see Section 9.6 of Percival and Walden (2000) or Section 7.3 in Gencay, Selcuk and Whitcher (2001).

**Value**

Matrix.

**Author(s)**

B. Whitcher

**References**


**See Also**

- `rotcumvar`, `testing.hov`.

---

**my.acf**

*Autocovariance Functions via the Discrete Fourier Transform*

**Description**

Computes the autocovariance function (ACF) for a time series or the cross-covariance function (CCF) between two time series.

**Usage**

- `my.acf(x)`
- `my.ccf(a, b)`
Arguments

\( x, a, b \)  

Arguments

time series

Details

The series is zero padded to twice its length before the discrete Fourier transform is applied. Only the values corresponding to nonnegative lags are provided (for the ACF).

Value

The autocovariance function for all nonnegative lags or the cross-covariance function for all lags.

Author(s)

B. Whitcher

Examples

data(ibm)
ibm.log <- log(ibm)
diff.log <- diff(ibm.log)

plot(1:length(diff.log) - 1, my.acf(diff.log), type="h",
     xlab="lag", ylab="ACVS", main="Autocovariance Sequence for IBM Returns")

Description

Yearly minimal water levels of the Nile river for the years 622 to 1281, measured at the Roda gauge near Cairo (Tousson, 1925, p. 366-385). The data are listed in chronological sequence by row.

The original Nile river data supplied by Beran only contained only 500 observations (622 to 1121). However, the book claimed to have 660 observations (622 to 1281). The remaining observations from the book were added, by hand, but the series still only contained 653 observations (622 to 1264).

Note, now the data consists of 663 observations (spanning the years 622-1284) as in original source (Toussoun, 1925).

Usage

data(nile)

Format

A length 663 vector.
**Source**


**References**


---

**ortho.basis**

**Derive Orthonormal Basis from Wavelet Packet Tree**

**Description**

An orthonormal basis for the discrete wavelet transform may be characterized via a disjoint partitioning of the frequency axis that covers \([0, \frac{1}{2})\). This subroutine produces an orthonormal basis from a full wavelet packet tree.

**Usage**

`ortho.basis(xtree)`

**Arguments**

- `xtree` is a vector whose entries are associated with a wavelet packet tree.

**Details**

A wavelet packet tree is a binary tree of Boolean variables. Parent nodes are removed if any of their children exist.

**Value**

Boolean vector describing the orthonormal basis for the DWPT.

**Author(s)**

B. Whitcher

**Examples**

```r
data(japan)
J <- 4
wf <- "mb8"
japan.mra <- mra(log(japan), wf, J, boundary="reflection")
japan.nmean <-
  ts(apply(matrix(unlist(japan.mra[-(J+1)]), ncol=J, byrow=FALSE), 1, sum),
    start=1955, freq=4)
japan.nmean2 <- ts(japan.nmean[42:169], start=1965.25, freq=4)
```
phase.shift

plot(japan.nomean2, type="l")
japan.dwpt <- dwpt(japan.nomean2, wf, 6)
japan.basis <-
   ortho.basis(portmanteau.test(japan.dwpt, p=0.01, type="other"))
# Not implemented yet
# par(mfrow=c(1,1))
# plot.basis(japan.basis)

per

Periodogram

Description
Computation of the periodogram via the Fast Fourier Transform (FFT).

Usage
per(z)

Arguments
z time series

Author(s)
Author: Jan Beran; modified: Martin Maechler, Date: Sep 1995.

phase.shift

Phase Shift Wavelet Coefficients

Description
Wavelet coefficients are circularly shifted by the amount of phase shift induced by the wavelet transform.

Usage
phase.shift(z, wf, inv = FALSE)
phase.shift.packet(z, wf, inv = FALSE)

Arguments
z DWT object
wf character string; wavelet filter used in DWT
inv Boolean variable; if inv=TRUE then the inverse phase shift is applied
Details
The center-of-energy argument of Hess-Nielsen and Wickerhauser (1996) is used to provide a flexible way to circularly shift wavelet coefficients regardless of the wavelet filter used. The results are not identical to those used by Percival and Walden (2000), but are more flexible.

Value
DWT (DWPT) object with coefficients circularly shifted.

Author(s)
B. Whitcher

References

Phase Shift for Hilbert Wavelet Coefficients

Description
Wavelet coefficients are circularly shifted by the amount of phase shift induced by the discrete Hilbert wavelet transform.

Usage
\begin{verbatim}
phase.shift.hilbert(x, wf)
phase.shift.hilbert.packet(x, wf)
\end{verbatim}

Arguments
\begin{itemize}
  \item \texttt{x}  
  \hspace{1em} Discrete Hilbert wavelet transform (DHWT) object.
  \item \texttt{wf}  
  \hspace{1em} character string; Hilbert wavelet pair used in DHWT
\end{itemize}

Details
The "center-of-energy" argument of Hess-Nielsen and Wickerhauser (1996) is used to provide a flexible way to circularly shift wavelet coefficients regardless of the wavelet filter used.

Value
DHWT (DHWPT) object with coefficients circularly shifted.
**Author(s)**

B. Whitcher

**References**


**See Also**

phaseNshift

plotNdwtNRd

---

**plot.dwt.2d**  
*Plot Two-dimensional Discrete Wavelet Transform*

**Description**

Organizes the wavelet coefficients from a 2D DWT into a single matrix and plots it. The coarser resolutions are nested within the lower-lefthand corner of the image.

**Usage**

```r
## S3 method for class 'dwt.2d'
plot(x, cex.axis = 1, plot = TRUE, ...)
```

**Arguments**

- `x`: input matrix (image)
- `cex.axis`: par plotting parameter that controls the size of the axis text
- `plot`: if `plot = FALSE` then the matrix of wavelet coefficients is returned, the default is `plot = TRUE`
- `...`: additional graphical parameters if necessary

**Details**

The wavelet coefficients from the DWT object (a list) are reorganized into a single matrix of the same dimension as the original image and the result is plotted.

**Value**

Image plot.

**Author(s)**

B. Whitcher

**See Also**

dwt.2d.
Description

Computes the quadrature mirror filter from a given filter.

Usage

qmf(g, low2high=TRUE)

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Filter coefficients.</th>
</tr>
</thead>
<tbody>
<tr>
<td>low2high</td>
<td>Logical, default is TRUE which means a low-pass filter is input and a high-pass filter is output. Setting low2high=F performs the inverse.</td>
</tr>
</tbody>
</table>

Details

None.

Value

Quadrature mirror filter.

Author(s)

B. Whitcher

References

Any basic signal processing text.

See Also

wave.filter.

Examples

```r
## Haar wavelet filter
g <- wave.filter("haar")$lpf
qmf(g)
```
rotcumvar  

**Rotated Cumulative Variance**

**Description**

Provides the normalized cumulative sums of squares from a sequence of coefficients with the diagonal line removed.

**Usage**

\[
\text{rotcumvar}(x)
\]

**Arguments**

- \(x\): vector of coefficients to be cumulatively summed (missing values excluded)

**Details**

The rotated cumulative variance, when plotted, provides a qualitative way to study the time dependence of the variance of a series. If the variance is stationary over time, then only small deviations from zero should be present. If on the other hand the variance is non-stationary, then large departures may exist. Formal hypothesis testing may be performed based on boundary crossings of Brownian bridge processes.

**Value**

Vector of coefficients that are the cumulative sum of squared input coefficients.

**Author(s)**

B. Whitcher

**References**


Description

Miscellaneous functions for dual-tree wavelet software.

Usage

cshift(x, m)
cshift2D(x, m)
pm(a, b)

Arguments

x          N-point vector
m          amount of shift
a, b       input parameters

Value

y          vector x will be shifted by m samples to the left or matrix x will be shifted by m samples down.

u          \((a + b)/sqrt(2)\)

v          \((a - b)/sqrt(2)\)

Author(s)

Matlab: S. Cai, K. Li and I. Selesnick; R port: B. Whitcher

References

WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
http://taco.poly.edu/WaveletSoftware/
Circularly Shift Matrices from a 2D MODWT

Description

Usage
shift.2d(z, inverse=FALSE)

Arguments
z Two-dimensional MODWT object
inverse Boolean value on whether to perform the forward or inverse operation.

Details
The "center of energy" technique of Wickerhauser and Hess-Nielsen (1996) is employed to find circular shifts for the wavelet sub-matrices such that the coefficients are aligned with the original series. This corresponds to applying a (near) linear-phase filtering operation.

Value
Two-dimensional MODWT object with circularly shifted coefficients.

Author(s)
Brandon Whitcher

References


See Also
phase.shift, modwt.2d.
Examples

```r
n <- 512
G1 <- G2 <- dnorm(seq(-n/4, n/4, length=n))
G <- 100 * zapsmall(outer(G1, G2))
G <- modwt.2d(G, wf="la8", J=6)
k <- 50
xr <- yr <- trunc(n/2) + (-k:k)
par(mfrow=c(3,3), mar=c(1,1,2,1), pty="s")
for (j in names(G)[1:9]) {
  image(G[[j]][xr, yr], col=rainbow(64), axes=FALSE, main=j)
}
Gs <- shift.2d(G)
for (j in names(G)[1:9]) {
  image(Gs[[j]][xr, yr], col=rainbow(64), axes=FALSE, main=j)
}
```

Description

Computes sinusoidal data tapers directly from equations.

Usage

```r
sine.taper(n, k)
```

Arguments

- `n` length of data taper(s)
- `k` number of data tapers

Details

See reference.

Value

A vector or matrix of data tapers (cols = tapers).

Author(s)

B. Whitcher

References

Spectral Density Functions

**See Also**

* dpss.taper.

---

**Spectral Density Functions**

* **Spectral Density Functions for Long-Memory Processes**

**Description**

Draws the spectral density functions (SDFs) for standard long-memory processes including fractional difference (FD), seasonal persistent (SP), and seasonal fractional difference (SFD) processes.

**Usage**

```r
fdp.sdf(freq, d, sigma2 = 1)  
spp.sdf(freq, d, fG, sigma2 = 1)  
spp2.sdf(freq, d1, f1, d2, f2, sigma2 = 1)  
sfd.sdf(freq, s, d, sigma2 = 1)
```

**Arguments**

- `freq` vector of frequencies, normally from 0 to 0.5
- `d, d1, d2` fractional difference parameter
- `fG, f1, f2` Gegenbauer frequency
- `s` seasonal parameter
- `sigma2` innovations variance

**Value**

The power spectrum from an FD, SP or SFD process.

**Author(s)**

Brandon Whitcher

**See Also**

* fdp.mle, spp.mle.
Computes wavelet cross-covariance or cross-correlation between two time series.

Usage

spin.covariance(x, y, lag.max = NA)
spin.correlation(x, y, lag.max = NA)

Arguments

x first time series
y second time series, same length as x
lag.max maximum lag to compute cross-covariance (correlation)

Details

See references.

Value

List structure holding the wavelet cross-covariances (correlations) according to scale.

Author(s)

B. Whitcher
References


See Also

*wave.covariance*, *wave.correlation*.

Examples

```r
## Figure 7.9 from Gencay, Selcuk and Whitcher (2001)

data(exchange)
returns <- diff(log(exchange))
returns <- ts(returns, start=1970, freq=12)
wf <- "d4"
demusd.modwt <- modwt(returns[,"DEM.USD"], wf, 8)
demusd.modwt.bw <- brick.wall(demusd.modwt, wf)
jpyusd.modwt <- modwt(returns[,"JPY.USD"], wf, 8)
jpyusd.modwt.bw <- brick.wall(jpyusd.modwt, wf)
n <- dim(returns)[1]
J <- 6
lmax <- 36
returns.cross.cor <- NULL
for(i in 1:J) {
  blah <- spin.correlation(demusd.modwt.bw[[i]], jpyusd.modwt.bw[[i]], lmax)
  returns.cross.cor <- cbind(returns.cross.cor, blah)
}
returns.cross.cor <- ts(as.matrix(returns.cross.cor), start=-36, freq=1)
dimnames(returns.cross.cor) <- list(NULL, paste("level", 1:J))
lags <- length(-lmax:lmax)
lower.ci <- tanh(atanh(returns.cross.cor) - qnorm(0.975) / sqrt(matrix(trunc(n/2*(1:j)), nrow=lags, ncol=J, byrow=TRUE) - 3))
upper.ci <- tanh(atanh(returns.cross.cor) + qnorm(0.975) / sqrt(matrix(trunc(n/2*(1:j)), nrow=lags, ncol=J, byrow=TRUE) - 3))
par(mfrow=c(3,2), las=1, pty="m", mar=c(5,4,4,2)+.1)
for(i in 1:J) {
  plot(returns.cross.cor[,i], ylim=c(-1,1), xlab="n", xlab="Lag (months)",
       ylab="", main=dimnames(returns.cross.cor)[[2]][[i]])
  axis(side=1, at=seq(-36, 36, by=12))
  lines(lower.ci[,i], lty=1, col=2)
  lines(upper.ci[,i], lty=1, col=2)
  abline(h=0, v=0)
}
```
**spp.mle**

---

### Wavelet-based Maximum Likelihood Estimation for Seasonal Persistent Processes

**Description**

Parameter estimation for a seasonal persistent (seasonal long-memory) process is performed via maximum likelihood on the wavelet coefficients.

**Usage**

```r
spp.mle(y, wf, J=log(length(y),2)-1, p=0.01, frac=1)
spp2.mle(y, wf, J=log(length(y),2)-1, p=0.01, dyadic=TRUE, frac=1)
```

**Arguments**

- `y`: Not necessarily dyadic length time series.
- `wf`: Name of the wavelet filter to use in the decomposition. See `wave.filter` for those wavelet filters available.
- `J`: Depth of the discrete wavelet packet transform.
- `p`: Level of significance for the white noise testing procedure.
- `dyadic`: Logical parameter indicating whether or not the original time series is dyadic in length.
- `frac`: Fraction of the time series that should be used in constructing the likelihood function.

**Details**

The variance-covariance matrix of the original time series is approximated by its wavelet-based equivalent. A Whittle-type likelihood is then constructed where the sums of squared wavelet coefficients are compared to bandpass filtered version of the true spectral density function. Minimization occurs for the fractional difference parameter $d$ and the Gegenbauer frequency $f_G$, while the innovations variance is subsequently estimated.

**Value**

List containing the maximum likelihood estimates (MLEs) of $\delta$, $f_G$ and $\sigma^2$, along with the value of the likelihood for those estimates.

**Author(s)**

B. Whitcher

**References**

## Variance of a Seasonal Persistent Process

### Description

Computes the variance of a seasonal persistent (SP) process using a hypergeometric series expansion.

### Usage

```r
spp.var(d, fG, sigma2 = 1)
Hypergeometric(a, b, c, z)
```

### Arguments

- `d`: Fractional difference parameter.
- `fG`: Gegenbauer frequency.
- `sigma2`: Innovations variance.
- `a,b,c,z`: Parameters for the hypergeometric series.

### Details

See Lapsa (1997). The subroutine to compute a hypergeometric series was taken from *Numerical Recipes in C*.

### Value

The variance of an SP process.

### Author(s)

B. Whitcher

### References


squared.gain

Squered Gain Function of a Filter

Description
Produces the modulus squared of the Fourier transform for a given filtering sequence.

Usage
squared.gain(wf.name, filter.seq = "L", n = 512)

Arguments
- wf.name: Character string of wavelet filter.
- filter.seq: Character string of filter sequence. H means high-pass filtering and L means low-pass filtering. Sequence is read from right to left.
- n: Length of zero-padded filter. Frequency resolution will be n/2+1.

Details
Uses cascade subroutine to compute the squared gain function from a given filtering sequence.

Value
Squared gain function.

Author(s)
B. Whitcher

See Also
wave.filter, wavelet.filter.

Examples
par(mfrow=c(2,2))
f.seq <- "H"
plot(0:256/512, squared.gain("d4", f.seq), type="l", ylim=c(0,2),
     xlab="frequency", ylab="L = 4", main="Level 1")
lines(0:256/512, squared.gain("fk4", f.seq), col=2)
lines(0:256/512, squared.gain("mb4", f.seq), col=3)
abline(v=c(1,2)/4, lty=2)
legend(-.02, 2, c("Daubechies", "Fejer-Korovkin", "Minimum-Bandwidth"),
       lty=1, col="1:3", bty="n", cex=1)
f.seq <- "HL"
plot(0:256/512, squared.gain("d4", f.seq), type="l", ylim=c(0,4),
     xlab="frequency", ylab="", main="Level 2")
lines(0:256/512, squared.gain("fk4", f.seq), col=2)
lines(0:256/512, squared.gain("mb4", f.seq), col=3)
abline(v=c(1.2)/8, lty=2)
f.seq <- "H"
plot(0:256/512, squared.gain("d8", f.seq), type="l", ylim=c(0,2),
  xlab="frequency", ylab="L = 8", main="")
lines(0:256/512, squared.gain("fk8", f.seq), col=2)
lines(0:256/512, squared.gain("mb8", f.seq), col=3)
abline(v=c(1.2)/4, lty=2)
f.seq <- "HL"
plot(0:256/512, squared.gain("d8", f.seq), type="l", ylim=c(0,4),
  xlab="frequency", ylab="", main="")
lines(0:256/512, squared.gain("fk8", f.seq), col=2)
lines(0:256/512, squared.gain("mb8", f.seq), col=3)
abline(v=c(1.2)/8, lty=2)

---

**stackPlot**

*Stack Plot*

**Description**

Stack plot of an object. This function attempts to mimic a function called stack.plot in S+WAVELETS. It is mostly a hacked version of plot.ts in R.

**Usage**

```
stackPlot(x, plot.type = c("multiple", "single"), panel = lines,
  log = "", col = par("col"), bg = NA, pch = par("pch"), cex = par("cex"),
  lty = par("lty"), lwd = par("lwd"), ann = par("ann"), xlab = "Time",
  main = NULL, oma = c(6, 0, 5, 0), layout = NULL,
  same.scale = 1:dim(x)[2], ...)```

**Arguments**

- `x`: ts object
- `layout`: Doublet defining the dimension of the panel. If not specified, the dimensions are chosen automatically.
- `same.scale`: Vector the same length as the number of series to be plotted. If not specified, all panels will have unique axes.
- `plot.type, panel, log, col, bg, pch, cex, lty, lwd, ann, xlab, main, oma, ...
  See plot.ts`

**Details**

Produces a set of plots, one for each element (column) of `x`.

**Author(s)**

Brandon Whitcher
testing.hov

Testing for Homogeneity of Variance

Description

A recursive algorithm for detecting and locating multiple variance change points in a sequence of random variables with long-range dependence.

Usage

testing.hov(x, wf, J, min.coef=128, debug=FALSE)

Arguments

x Sequence of observations from a (long memory) time series.
wf Name of the wavelet filter to use in the decomposition.
J Specifies the depth of the decomposition. This must be a number less than or equal to \( \log(\text{length}(x), 2) \).
min.coef Minimum number of wavelet coefficients for testing purposes. Empirical results suggest that 128 is a reasonable number in order to apply asymptotic critical values.
debug Boolean variable: if set to TRUE, actions taken by the algorithm are printed to the screen.

Details

For details see Section 9.6 of Percival and Walden (2000) or Section 7.3 in Gencay, Selcuk and Whitcher (2001).

Value

Matrix whose columns include (1) the level of the wavelet transform where the variance change occurs, (2) the value of the test statistic, (3) the DWT coefficient where the change point is located, (4) the MODWT coefficient where the change point is located. Note, there is currently no checking that the MODWT is contained within the associated support of the DWT coefficient. This could lead to incorrect estimates of the location of the variance change.

Author(s)

B. Whitcher

References

Thresholding

Wavelet Shrinkage via Thresholding

See Also
dwt, modwt, rotcumvar, mult.loc.

Thresholding

Description
Perform wavelet shrinkage using data-analytic, hybrid SURE, manual, SURE, or universal thresholding.

Usage
da.thresh(wc, alpha = .05, max.level = 4, verbose = FALSE, return.thresh = FALSE)
hybrid.thresh(wc, max.level = 4, verbose = FALSE, seed = 0)
manual.thresh(wc, max.level = 4, value, hard = TRUE)
sure.thresh(wc, max.level = 4, hard = TRUE)
universal.thresh(wc, max.level = 4, hard = TRUE)
universal.thresh.modwt(wc, max.level = 4, hard = TRUE)

Arguments
wc wavelet coefficients
alpha level of the hypothesis tests
max.level maximum level of coefficients to be affected by threshold
verbose if verbose=TRUE then information is printed to the screen
value threshold value (only utilized in manual.thresh)
hard Boolean value, if hard=F then soft thresholding is used
seed sets random seed (only utilized in hybrid.thresh)
return.thresh if return.thresh=TRUE then the vector of threshold values is returned, otherwise the surviving wavelet coefficients are returned

Details
An extensive amount of literature has been written on wavelet shrinkage. The functions here represent the most basic approaches to the problem of nonparametric function estimation. See the references for further information.

Value
The default output is a list structure, the same length as was input, containing only those wavelet coefficients surviving the threshold.

Author(s)
B. Whitcher (some code taken from R. Todd Ogden)
References


tourism

<table>
<thead>
<tr>
<th>tourism</th>
<th>U.S. Tourism</th>
</tr>
</thead>
</table>

Description


Usage

data(tourism)

Format

A vector containing 160 observations.

Source

Unknown.

References


unemploy

<table>
<thead>
<tr>
<th>unemploy</th>
<th>U.S. Unemployment</th>
</tr>
</thead>
</table>

Description


Usage

data(unemploy)
Format

A vector containing 624 observations.

Source

Unknown.

References


Description

Upsamples a given vector.

Usage

\texttt{up.sample(x, f, y = NA)}

Arguments

- \texttt{x} vector of observations
- \texttt{f} frequency of upsampling; e.g., 2, 4, etc.
- \texttt{y} value to upsample with; e.g., NA, 0, etc.

Value

A vector twice its length.

Author(s)

B. Whitcher

References

Any basic signal processing text.
Description

Converts name of wavelet filter to filter coefficients.

Usage

\texttt{wave.filter(name)}

Arguments

name \hspace{1cm} \text{Character string of wavelet filter.}

Details

Simple switch statement selects the appropriate filter.

Value

List containing the following items:

- \texttt{L} \hspace{1cm} \text{Length of the wavelet filter.}
- \texttt{hpf} \hspace{1cm} \text{High-pass filter coefficients.}
- \texttt{lpf} \hspace{1cm} \text{Low-pass filter coefficients.}

Author(s)

B. Whitcher

References


See Also

\texttt{wavelet.filter, squared.gain}. 
Description

Produces an estimate of the multiscale variance, covariance or correlation along with approximate confidence intervals.

Usage

```r
wave.variance(x, type="eta3", p=0.025)
wave.covariance(x, y)
wave.correlation(x, y, N, p=0.975)
```

Arguments

- `x`: first time series
- `y`: second time series
- `type`: character string describing confidence interval calculation; valid methods are `gaussian`, `eta1`, `eta2`, `eta3`, `nongaussian`
- `p`: (one minus the) two-sided p-value for the confidence interval
- `N`: length of time series

Details

The time-independent wavelet variance is basically the average of the squared wavelet coefficients across each scale. As shown in Percival (1995), the wavelet variance is a scale-by-scale decomposition of the variance for a stationary process, and certain non-stationary processes.

Value

Matrix with as many rows as levels in the wavelet transform object. The first column provides the point estimate for the wavelet variance, covariance, or correlation followed by the lower and upper bounds from the confidence interval.

Author(s)

B. Whitcher

References


**Examples**

```r
## Figure 7.3 from Gencay, Selcuk and Whitcher (2001)
data(ar1)
ar1.modwt <- modwt(ar1, "haar", 6)
ar1.modwt.bw <- brick.wall(ar1.modwt, "haar")
ar1.modwt.var2 <- wave.variance(ar1.modwt.bw, type="gaussian")
ar1.modwt.var <- wave.variance(ar1.modwt.bw, type="nongaussian")
par(mfrow=c(1,1), las=1, mar=c(5,4,4,2)+.1)
matplot(2^((0:5)), ar1.modwt.var2[-7:1], type="b", log="xy",
        xaxt="n", ylim=c(.025, 6), pch="*LU", lty=1, col=c(1,4,4),
        xlab="Wavelet Scale", ylab="")
matlines(2^((0:5)), as.matrix(ar1.modwt.var)[-7:2:3], type="b",
        pch="LU", lty=1, col=3)
axis(side=1, at=2^((0:5)))
legend(1, 6, c("Wavelet variance", "Gaussian CI", "Non-Gaussian CI"),
       lty=1, col=c(1,4,3), bty="n")

## Figure 7.8 from Gencay, Selcuk and Whitcher (2001)
data(exchange)
returns <- diff(log(as.matrix(exchange)))
returns <- ts(returns, start=1970, freq=12)
wf <- "d4"
J <- 6
demusd.modwt <- modwt(returns[,"DEM.USD"], wf, J)
demusd.modwt.bw <- brick.wall(demusd.modwt, wf)
jpyusd.modwt <- modwt(returns[,"JPY.USD"], wf, J)
jpyusd.modwt.bw <- brick.wall(jpyusd.modwt, wf)
returns.modwt.cov <- wave.covariance(demusd.modwt.bw, jpyusd.modwt.bw)
par(mfrow=c(1,1), las=0, mar=c(5,4,4,2)+.1)
matplot(2^((0:(J-1))), returns.modwt.cov[(-(J+1):J),], type="b", log="x",
        pch="*LU", xaxt="n", lty=1, col=c(1,4,4), xlab="Wavelet Scale",
        ylab="Wavelet Covariance")
axis(side=1, at=2^((0:7)))
abline(h=0)

returns.modwt.cor <- wave.correlation(demusd.modwt.bw, jpyusd.modwt.bw,
                                       N = dim(returns)[1])
par(mfrow=c(1,1), las=0, mar=c(5,4,4,2)+.1)
matplot(2^((0:(J-1))), returns.modwt.cor[(-(J+1):J),], type="b", log="x",
        pch="*LU", xaxt="n", lty=1, col=c(1,4,4), xlab="Wavelet Scale",
        ylab="Wavelet Correlation")
axis(side=1, at=2^((0:7)))
abline(h=0)
```
Description

Create a wavelet filter at arbitrary scale.

Usage

wavelet.filter(wf.name, filter.seq = "L", n = 512)

Arguments

- wf.name: Character string of wavelet filter.
- filter.seq: Character string of filter sequence. H means high-pass filtering and L means low-pass filtering. Sequence is read from right to left.
- n: Length of zero-padded filter. Frequency resolution will be n/2+1.

Details

Uses cascade subroutine to compute higher-order wavelet coefficient vector from a given filtering sequence.

Value

Vector of wavelet coefficients.

Author(s)

B. Whitcher

References


See Also

- squared.gain
- wave.filter
Examples

```r
## Figure 4.14 in Gencay, Selcuk and Whitcher (2001)
par(mfrow=c(3,1), mar=c(5-2,4,4-1,2))
f.seq <- "HLLLLL"
plot(c(rep(0,33), wavelet.filter("mb4", f.seq), rep(0,33)), type="l",
     xlab="", ylab="", main="D(4) in black, MB(4) in red")
lines(c(rep(0,33), wavelet.filter("d4", f.seq), rep(0,33)), col=2)
plot(c(rep(0,35), -wavelet.filter("mb8", f.seq), rep(0,35)), type="l",
     xlab="", ylab="", main="D(8) in black, MB(8) in red")
lines(c(rep(0,35), wavelet.filter("d8", f.seq), rep(0,35)), col=2)
plot(c(rep(0,39), wavelet.filter("mb16", f.seq), rep(0,39)), type="l",
     xlab="", ylab="", main="D(16) in black, MB(16) in red")
lines(c(rep(0,39), wavelet.filter("d16", f.seq), rep(0,39)), col=2)
```

Description

A wavelet packet tree, from the discrete wavelet packet transform (DWPT), is tested node-by-node for white noise. This is the first step in selecting an orthonormal basis for the DWPT.

Usage

```r
cpgram.test(y, p = 0.05, taper = 0.1)
css.test(y)
entropy.test(y)
portmanteau.test(y, p = 0.05, type = "Box-Pierce")
```

Arguments

- `y`: wavelet packet tree (from the DWPT)
- `p`: significance level
- `taper`: weight of cosine bell taper (cpgram.test only)
- `type`: "Box-Pierce" and other recognized (portmanteau.test only)

Details

Top-down recursive testing of the wavelet packet tree is

Value

Boolean vector of the same length as the number of nodes in the wavelet packet tree.

Author(s)

B. Whitcher
References


See Also

*ortho.basis.*

Examples

data(mexm)
J <- 6
wf <- "la8"
mexm.dwpt <- dwpt(mexm[-(1:4)], wf, J)
## Not implemented yet
## plot.dwpt(x.dwpt, J)
mexm.dwpt.bw <- dwpt.brick.wall(mexm.dwpt, wf, 6, method="dwpt")
mexm.tree <- ortho.basis(portmanteau.test(mexm.dwpt.bw, p=0.025))
## Not implemented yet
## plot.basis(mexm.tree)

---

*xbox*  
*Image with Box and X*

Description

\[ xbox(i, j) = I_{[i=n/4, 3n/4, j; n/4 \leq j \leq 3n/4]} + I_{[n/4 \leq i \leq 3n/4; j=n/4, 3n/4, i]} \]

Usage

data(xbox)

Format

A 128 × 128 matrix.

Source

S+WAVELETS.

References

Index

+Topic datasets
   Andel, 3
   ar1, 4
   barbara, 5
   blocks, 7
   cpi, 9
   dau, 10
   doppler, 12
   exchange, 30
   heavisine, 33
   ibm, 38
   japan, 39
   jumpsine, 39
   kobe, 40
   linchirp, 40
   mexm, 41
   mile, 52
   tourism, 71
   unemploy, 71
   xbox, 78
+Topic hplot
   stackPlot, 68
+Topic ts
   Band-pass variance, 4
   basis, 6
   brick.wall, 7
   convolve2D, 8
   denoise.2d, 10
   dpss.taper, 12
   Dual-tree Filter Banks, 14
   dualfilt1, 16
   Dualtree, 17
   Dualtree Complex, 19
   dwpt, 20
   dwpt.2d, 22
   dwpt.boot, 23
   dwpt.sim, 24
   dwt, 25
   dwt.2d, 28
   dwt.3d, 29
   Farras, 30
   fdp.mle, 31
   find.adaptive.basis, 32
   Hilbert, 34
   hilbert.filter, 35
   hosking.sim, 36
   HWP Analysis, 37
   modwt, 41
   modwt.2d, 43
   modwt.3d, 45
   mra, 46
   mra.2d, 47
   mra.3d, 49
   mult.loc, 50
   my.acf, 51
   ortho.basis, 53
   per, 54
   phase.shift, 54
   phase.shift.hilbert, 55
   plot.dwt.2d, 56
   qmf, 57
   rotcumvar, 58
   Selesnick, 59
   shift.2d, 60
   sine.taper, 61
   Spectral Density Functions, 62
   spin.covariance, 63
   spp.mle, 65
   spp.var, 66
   squared.gain, 67
   testing.hov, 69
   Thresholding, 70
   up.sample, 72
   wave.filter, 73
   wave.variance, 74
   wavelet.filter, 75
   wpt.test, 77
   acvs.andel10 (Andel), 3
acvsandel11 (Andel), 3
acvsandel8 (Andel), 3
acvsandel9 (Andel), 3
afb, 18, 31
afb (Dual-tree Filter Banks), 14
afb2D, 19
afb2D (Dual-tree Filter Banks), 14
Andel, 3
AntonB (dualfilt1), 16
ar1, 4

Band-pass variance, 4
bandpass.fdp (Band-pass variance), 4
bandpass.spp (Band-pass variance), 4
bandpass.spp2 (Band-pass variance), 4
bandpass.var.spp (Band-pass variance), 4
barbara, 5
basis, 6
bishrink (Thresholding), 70
blocks, 7
brick.wall, 7

convolve, 9, 18
convolve2D, 8
cpgram.test (wpt.test), 77
cri, 9
cplxdual2D (Dualtree Complex), 19
cshift, 18
cshift (Selesnick), 59
cshift2D (Selesnick), 59
css.test (wpt.test), 77

da.thresh (Thresholding), 70
dau, 10
denoise.2d, 10
denoise.dwt.2d (denoise.2d), 10
denoise.modwt.2d (denoise.2d), 10
doppler, 12
dpss.taper, 12, 62
Dual-tree Filter Banks, 14
dualfilt1, 16, 31
Dualtree, 17
dualtree, 16, 31
dualtree (Dualtree), 17
Dualtree Complex, 19
dualtree2D (Dualtree), 17
dwpt, 6, 7, 20
dwpt.2d, 22
dwpt.boot, 23
dwpt.brick.wall (brick.wall), 7
dwpt.sim, 24, 24, 33
dwt, 7, 21, 25, 43, 47, 70
dwt.2d, 23, 28, 44, 49, 56
dwt.3d, 29, 50
dwt.hilbert (Hilbert), 34
entropy.test (wpt.test), 77
exchange, 30

Farras, 30
farras, 18, 19
farras (Farras), 30
fdp.mle, 31, 62, 66
fdp.sdf (Spectral Density Functions), 62
find.adaptive.basis, 32
FSfarras, 18, 19
FSfarras (Farras), 30

heavisine, 33
Hilbert, 34
hilbert.filter, 34, 35, 38
hosking.sim, 25, 36
HWP Analysis, 37
hybrid.thresh (Thresholding), 70
Hypergeometric (spp.var), 66

ibm, 38
icplxdual2D (Dualtree Complex), 19
idualtree (Dualtree), 17
idualtree2D (Dualtree), 17
idwpt (dwt), 20
idwpt.2d (dwt.2d), 22
idwpt.3d, 43
idwpt (dwt), 25
idwpt.2d (dwt.2d), 28
idwpt.3d (dwt.3d), 29
idwpt.hilbert (Hilbert), 34
imodwt (modwt), 41
imodwt.2d (modwt.2d), 43
imodwt.3d (modwt.3d), 45
imodwt.hilbert (Hilbert), 34

japan, 39
jumpsine, 39

kobe, 40

linchirp, 40
INDEX

manual.thresh (Thresholding), 70
mexm, 41
modhw.t.coh (HWP Analysis), 37
modhw.t.phase (HWP Analysis), 37
modwpt, 7, 21
modwpt (dwpt), 20
modwpt.hilbert (Hilbert), 34
modwt, 7, 27, 41, 47, 70
modwt.2d, 23, 28, 43, 49, 60
modwt.3d, 45, 50
modwt.hilbert (Hilbert), 34
mra, 27, 43, 46
mra.2d, 47
mra.3d, 49
mult.loc, 50, 70
my.acf, 51
my.ccf (my.acf), 51

nile, 52

ortho.basis, 53, 78

per, 54
phase.shift, 54, 56, 60
phase.shift.hilbert, 55
plot.dwt.2d, 56
pm (Selesnick), 59
portmanteau.test (wpt.test), 77

qmf, 57

rotcumvar, 51, 58, 70

Selesnick, 59
sfb, 18
sfb (Dual-tree Filter Banks), 14
sfb2D, 19
sfb2D (Dual-tree Filter Banks), 14
sfd.sdf (Spectral Density Functions), 62
shift.2d, 44, 60
sine.taper, 13, 61
soft (Thresholding), 70
Spectral Density Functions, 62
spin.correlation (spin.covariance), 63
spin.covariance, 63
spp.mle, 24, 62, 65
spp.sdf (Spectral Density Functions), 62
spp.var, 66
spp2.mle (spp.mle), 65

spp2.sdf (Spectral Density Functions), 62
squared.gain, 67, 73, 76
stackPlot, 68
sure.thresh (Thresholding), 70
testing.hov, 50, 51, 69
Thresholding, 11, 70
tourism, 71

unemploy, 71
universal.thresh (Thresholding), 70
up.sample, 72

wave.correlation, 64
wave.correlation (wave.variance), 74
wave.covariance, 64
wave.covariance (wave.variance), 74
wave.filter, 21, 23, 25, 31, 35, 57, 63, 67, 73, 76
wave.variance, 74
wavelet.filter, 67, 73, 75
wpt.test, 77

xbox, 78