Package 'wmtsa'

August 29, 2016

Title Wavelet Methods for Time Series Analysis

Version 2.0-2

Depends R (>= 3.0.2)

Imports splus2R (>= 1.2-0), ifultools (>= 2.0-0), MASS, graphics, methods

Description Software to book Wavelet Methods for Time Series Analysis, Donald B. Percival and Andrew T. Walden, Cambridge University Press, 2000.

License GPL-2

ZipData no

Collate wav_boot.R wav_coef.R wav_dict.R wav_fdp.R wav_filt.R wav_gnrc.R wav_mrd.R wav_plot.R wav_shrk.R wav_sig.R wav_util.R wav_var.R wav_wtmm.R wav_xform.R wav_pkg.R

Repository CRAN

NeedsCompilation no

Author William Constantine [cre, aut], Donald Percival [aut]

Maintainer William Constantine <wlbconstan@gmail.com>

Date/Publication 2016-06-09 06:18:09

R topics documented:

atomclock *Cesium Beam Atomic Clock Data*

Description

This series represents the difference in time between a cesium beam atomic clock and an official time scale known as UTC(USNO) maintained by the US Naval Observatory, Washington DC. The UTC portion of the USNO series refers to coordinate universal time which is used as an international time standard. Negative values in the resulting (difference) series represents a lag in time relative to the UTC(USNO) standard. The differences in time were recorded at the same time for consecutive days in the 1970s resulting in a sampling interval of 1 day. The amplitude units are expressed in microseconds.

See Also

nile, ocean, ecg, fdp045.

Examples

```
plot(atomclock)
```

create.signalSeries *Converts various time series to an object of class*

Description

Converts numeric data to an object of class containing one dimensional data. The input data is assumed to be uniformly sampled.

Usage

```
create.signalSeries(x=NULL, position=list(from=1,by=1,units=character()),
    from=NULL, by=NULL, to=NULL, length.out=NULL,
   units=character(), title.data=character(), documentation=character(), na.rm=TRUE)
```

Arguments

by	a numeric containing the sampling rate at which the values in data should be extracted. This parameter must coordinate with the position arguments and can be used in combination with the by, to, or length.out arguments. This argument is not the same as the position\$by argument which denotes the sampling rate of the original data. Default: NULL.
documentation	a string used to describe the input data. Default: character().
from	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: NULL.

length.out	an integer containing the maximum number of values to extract from data. Because data is a finite length sequence, the actual number of values returned may be less than that specified by this argument depending upon the conditions imposed by the from and by arguments. The length.out argument should not be specified if both the from and to arguments are specified. Default: NULL.
na.rm	a logical flag used to indicate if NaN values should be removed from the input. Default: TRUE.
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
title.data	a string representing the name of the input data. Default: character().
to	a numeric containing the end point in data from which the values should be extracted. This parameter must coordinate with the position arguments and can be used in combination with the by and to arguments. The length.out argument should not be specified if both the from and to arguments are specified. Default: NULL.
units	a string denoting the units of the time series. Default: character().
x	a numeric vector, matrix or an object of class ts (uniform sampling assumed). If a matrix, the longest first row or column is extracted. Default: NULL (no data).

Value

an object of class signalSeries.

See Also

make.signal.

Examples

```
## convert an explicitly developed numeric vector
x <- 1:10
create.signalSeries(x)

## now impose hypothetical positions on the data
create.signalSeries(x, pos=list(from=0.3, by=0.1))

## extract the values from position 0.5 onward
create.signalSeries(x, pos=list(from=0.3, by=0.1), from=0.5)

## extract the values from position 0.5 onward,
## but keep only the first 3 values of the
## extraction
create.signalSeries(x, pos=list(from=0.3, by=0.1), from=0.5, length=3)

## extract the values from position 0.5 onward and
## skip every other point (sample the data at
## 0.2 position intervals)
create.signalSeries(x, pos=list(from=0.3, by=0.1), from=0.5, by=0.2)
```

```
## simply return the first 4 values, and supply a
## title and some documentation comments to the
## data
create.signalSeries(x, length=4, title="Faux Data", doc="An example")
```

`crystal.names` *Generic function for obtaining crystal names of wavelet transform objects*

Description

Returns the names assigned to crystals (vectors of wavelet transform coefficients).

Usage

```
crystal.names(x, ...)
```

Arguments

`x`	any object. Missing values (NAs) are allowed.
`...`	optional arguments to be passed directly to the inherited function without alteration and with the original names preserved.

Value

a vector of character strings.

See Also

wavMRD.

Examples

```
methods(crystal.names)
```

D.statistic *D-statistic*

Description

Given a numeric vector x, this function calculates the the maximum departure of x from an expected linear increase in cumulative energy based on a white noise model.

Usage

```
D.statistic(x)
```

Arguments

x	a numeric vector.

See Also

wavVar, wavVarTest, D.table, D.table.critical.

Examples

```
## compare the D-statistic for a white noise
## realization and a random walk. the random
## walk D-statistic will be relatively large in
## comparison to that of the white noise
## realization, signifying a stronger departure
## from an expected increase in cumulative
## energy.
D.statistic(rnorm(1024))
D.statistic(cumsum(rnorm(1024)))
```

D.table *Critical D-statistic table generation*

Description

The D-statistic denotes the maximum deviation of sequence from a hypothetical linear cumulative energy trend. The critical D-statistics define the distribution of D for a zero mean Gaussian white noise process. Comparing the sequence D-statistic to the corresponding critical values provides a means of quantitatively rejecting or accepting the linear cumulative energy hypothesis. The table is generated for an ensemble of distribution probabilities and sample sizes.

Usage

```
D.table(n.sample=c(127, 130), significance=c(0.1, 0.05, 0.01),
    lookup=TRUE, n.realization=10000, n.repetition=3,
    tolerance=1e-6)
```

Arguments

lookup	a logical flag for accessing precalculated critical D-statistics. The critical D-statistics are calculated for a variety of sample sizes and significances. If lookup is TRUE (recommended), this table is accessed. The table is stored as the matrix object D.table.critical. Missing table values are calculated using the input arguments: n.realization, n.repetition, and tolerance. Default: TRUE.
n.realization	an integer specifying the number of realizations to generate in a Monte Carlo simulation for calculating the D-statistic(s). This parameter is used either when lookup is FALSE, or when lookup is TRUE and the table is missing values corresponding to the specified significances. Default: 10000.
n.repetition	an integer specifying the number of Monte Carlo simulations to perform. This parameter coordinates with the n.realization parameter. Default: 3.
n.sample	a vector of integers denoting the sample sizes for which critical D-statistics are created. Default: c(127,130).
significance	a numeric vector of real values in the interval (0,1). The significance is the fraction of times that the linear cumulative energy hypothesis is incorrectly rejected. It is equal to the difference of the distribution probability (p) and unity. Default: c(0.1, 0.05, 0.01).
tolerance	a numeric real scalar that specifies the amplitude threshold to use in estimating critical D-statistic(s) via the Inclan-Tiao approximation. Setting this parameter to a higher value results in a lesser number of summation terms at the expense of obtaining a less accurate approximation. Default: 1e-6.

Details

A precalculated critical D-statistics object (D.table.critical) exists on the package workspace and was built for a variety of sample sizes and significances using 3 repetitions and 10000 realizations/repetition. This D.table function should be used in cases where specific D-statistics are missing from D.table.critical. Note: the results of the D.table value should not be returned to a variable named D.table.critical as it will override the precalculated table available in the package.

An Inclan-Tiao approximation of critical D-statistics is used for sample sizes n.sample ≥ 128 while a Monte Carlo technique is used for n.sample < 128. For the Monte Carlo technique, the D-statistic for a Gaussian white noise sequence of length n.sample is calculated. This process is repeated n.realization times, forming a distribution of the D-statistic. The critical values corresponding to the significances are calculated a total of n.repetition times, and averaged to form an approximation to the D-statistic(s).

Value

a matrix containing the critical D-statistics corresponding to the supplied sample sizes and significances.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

D.table.critical.

Examples

```
D.lookup <- D.table(significance=c(10,5,1)/100,
    n.realization=100, n.sample=125:130, lookup=FALSE)
```

D.table.critical *Critical D-Statistic Table*

Description

The critical D-statistics define the distribution of D for a zero mean Gaussian white noise process. Comparing a D-statistic to the corresponding critical values provides a means of quantitatively rejecting or accepting a linear cumulative energy hypothesis. The table is generated for an ensemble of distribution probabilities and sample sizes.

See Also

D.table, D.statistic.

Examples

```
D.table.critical[1:10,]
```

---

ecg *Electrocardiogram Data*

Description

This 2048 point ECG data series represents approximately 15 beats of a normal human cardiac rhythm. This sequence is sampled at 180 Hz and is in units of millivolts. This series was collected and supplied by Dr. G. Bardy M.D. and Dr. P. Reinhall of the University of Washington.

See Also

nile, atomclock, ocean, fdp045.

Examples

```
plot(ecg)
```

eda.plot *Generic function for generating extended data analysis plots*

Description

Data analysis plots are used to visually summarize the salient features of the output and typically involve a combination of plots in a single plot frame.

Usage

```
eda.plot(x, ...)
```

Arguments

x	any object. Missing values (NAs) are allowed.
...	optional arguments to be passed directly to the inherited function without alteration and with the original names preserved.

Note

An extended data analysis plot is shown.

See Also

wavMRD, wavTransform, wavStackPlot, crystal.names.

Examples

```
methods(eda.plot)
```

fdp045 *Fractional Difference Process Data*

Description

These data represent a single realization of a fractional difference process (FDP) with FDP exponent $\delta = 0.45$ and a process variance of unity.

See Also

nile, atomclock, ocean, ecg.

Examples

```
plot(fdp045)
```

`holderSpectrum` *The Holder spectrum of a time series*

Description

Using a tree, this function returns time localized exponent estimations for a given time series.

Usage

```
holderSpectrum(x, n.scale.min=3, fit=lmsreg)
```

Arguments

`x`	an object of class `wavCWTTree`.
`fit`	a linear regression function to use in fitting the resulting data. Default: `lmsreg`.
`n.scale.min`	the minimum number of scales (points) that a given *suitable* branch segment must have before being considered as an admissible candidate for exponent estimation. Default: 3.

Details

Many real-world time series contain sharp dicontinuities (cusps) which can be attributed to rapid changes in the observed system. These cusps are called *singularities* and their strength can be quantified via localized exponents as follows: Let $f(t)$ be a continuous real-valued function containing a singularity at time t_0. The exponent $h(t_0)$ is defined as the supremum (least upper bound) of all exponents h which satisfies the condition

$$|f(t) - P_n(t - t_0)| \leq C|t - t_0|^{h(t_0)},$$

where $P_n(t - t_0)$ is a polynomial of degree $n \leq h(t_0)$ and C is a constant. The collection of exponents for a given time series denotes the so-called spectrum. Mallat demonstrated that a cusp singularity at time t_0 can be estimated via the CWT by noting that the wavelet transform modulus maxima behave as $W_{a,t_0}(f) \propto |a|^{h(t_0)}$ as the scale $a \to 0$.

Thus, the strength of cusp singularities in a given time series can be quantified by

i Calculate the CWT of the time series.

ii Find the modulus maxima of the CWT (WTMM).

iii Link the WTMM into separate branches based (mainly) on their position in time to form a WTMM tree.

iv For each branch in the tree, perform an exponential fit of the WTMM over an admissible range of scale and as the scale approaches zero. The resulting *scaling exponent* is an estimate of the local exponent for the time series. The occurrence of the singularity in time is recorded as the location in time where the WTMM converges as the scale nears zero.

In practice, the above technique can be unstable when applied to observational data due to negative moment divergences and so-called *outliers* which correspond to the end points of sample singularities. One must also be very careful in selecting an appropriate scaling region of a tree branch before fitting the data. We accomplish this by first segmenting a given tree branch into regions which exhibit approximate linear behavior in the log(scale)-log(WTMM) space, and subsequently selecting the region corresponding to the smallest scales for exponent estimation. Furthermore, through the `n.scale.min` argument, the user can control the minimum number of scales (points) that must exist in the isolated scaling region before a exponent estimation is recorded.

Value

a list containing the estimated exponents, associated times and corresponding branch number.

References

S.G. Mallat, *A Wavelet Tour of Signal Processing (2nd Edition)*, Academic Press, Cambridge, 1999.

S.G. Mallat and W.L. Hwang, "Singularity detection and processing with wavelets", *IEEE Transactions on Information Theory*, **38**, 617–643 (1992).

S.G. Mallat and S. Zhong, "Complete signal representation with multiscale edges", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**, 710–732 (1992).

J.F. Muzy, E. Bacry, and A. Arneodo, "The multifractal formalism revisited with wavelets.", *International Journal of Bifurcation and Chaos*, **4**, 245–302 (1994).

See Also

`wavCWT`, `wavCWTFilters`, `wavCWTTree`.

Examples

```
## create series with a linear trend and two
## cusps: h(x = 1) = 0.5 and h(x = 15) = 0.3
cusps <- function(x) -0.2 * abs(x-1)^0.5 - 0.5* abs(x-15)^0.3 + 0.00346 * x + 1.34
x <- seq(-5, 20, length=1000)
y <- splus2R::signalSeries(cusps(x), x)

## calculate CWT using Mexican hat filter
W <- wavCWT(y, wavelet="gaussian2")

## calculate WTMM and extract first two branches
## in tree corresponding to the cusps
W.tree <- wavCWTTree(W)[1:2]

## plot the CWT tree overlaid with a scaled
## version of the time series to illustrate
## alignment of branches with cusps
yshift <- y@data - min(y@data)
yshift <- yshift / max(yshift) * 4 - 4.5
plot(W.tree, xlab="x")
lines(x, yshift, lwd=2)
text(6.5, -1, "f(x) = -0.2|x-1|^0.5 - 0.5|x-15|^0.3 + 0.00346x + 1.34", cex=0.8)
```

```
## estimate Holder exponents
holder <- holderSpectrum(W.tree)
print(holder)
```

`make.signal` *Test signal generation*

Description

Generates various test signals for wavelet transforms.

Usage

```
make.signal(name, n=1024, snr=Inf)
```

Arguments

`name`	a character string denoting the type of test signal to create. Supported values are: `"dirac"`, `"kronecker"`, `"heavisine"`, `"bumps"`, `"blocks"`, `"doppler"`, `"ramp"`, `"cusp"`, `"crease"`, `"sing"`, `"hisine"`, `"losine"`, `"linchirp"`, `"twochirp"`, `"quadchirp"`, `"mishmash1"`, `"mishmash2"`, `"mishmash3"`, `"levelshift"`, `"jumpsine"`, `"gauss"`, `"patches"`, `"linear"`, `"quadratic"`, and `"cubic"`.
`n`	an integer specifying the length out the output series. Default: `1024`.
`snr`	a numeric value representing the approximate signal to noise ratio of the output. Default: `Inf` (no noise).

Value

a vector of numeric values containing the resulting test series.

See Also

`wavDWT`, `wavMODWT`.

Examples

```
nms <- c("blocks", "linchirp", "mishmash1", "bumps")
z <- lapply(nms, make.signal)
ifultools::stackPlot(x=seq(1024),y=z, ylab=nms)
```

nile *Yearly Nile river level minima*

Description

These data represent the measurements of the minimum yearly water level of the Nile over the years 622 to 1284 as recorded at the Roda gauge near Cairo.

See Also

atomclock, ocean, ecg, fdp045.

Examples

```
plot(nile)
```

ocean *Vertical Shear Ocean Data*

Description

This data is collected by an instrument dropped over the side of a ship in the ocean. As the instrument descends vertically in the water it collects information as a function of depth. The ocean data set is a record of the horizontal (shear) velocity of the water measured in 0.1 increments starting from a depth of 489.5 meters and ending at 899.0 meters. This series was collected and supplied by Mike Gregg, Applied Physics Laboratory, University of Washington.

See Also

nile, atomclock, ecg, fdp045.

Examples

```
plot(ocean)
```

oceansdf *SDF for ocean series*

Description

Parametric spectral density function for ocean series.

Usage

```
oceansdf(f)
```

Arguments

f	normalized frequencies over which the SDF function is evaluated.

Value

a numeric vector containing the values of ocean SDF function evaluated at the specified normalized frequencies.

See Also

ocean.

Examples

```
oceansdf(c(0.25,0.375))
```

reconstruct *Reconstruction (inverse transform) of various wavelet transforms*

Description

Inverts a discrete wavelet transform, mapping the data back into the time domain.

Usage

```
reconstruct(x, ...)
```

Arguments

x	an object of class wavTransform or wavMRD.
...	optional arguments passed directly to the reconstruction function.

Value

a numeric vector containing the result.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and, Applied Mathematics, 41, 909–96.

See Also

wavDWT, wavMODWT, wavMRD.

Examples

```
## create a random series, calculate a forward
## transform, then invert
x       <- rnorm(1024)
x.dwt   <- reconstruct(wavDWT(x))
x.modwt <- reconstruct(wavMODWT(x))
all(c(splus2R::vecnorm(x.dwt-x), splus2R::vecnorm(x.modwt-x)) < .Machine$single.eps)
```

---

wavBestBasis *DWPT Best basis selection*

Description

The discrete wavelet packet transform (DWPT) contains a multitude of disjoint dyadic decompositions representing an ensemble of different bases. Best basis selection is an attempt to isloate one such basis in an *optimal* way.

Usage

```
wavBestBasis(costs)
```

Arguments

costs	a numeric vector containing the costs for each crystal in a DWPT in $\mathcal{C}(W_{0,0}), \mathcal{C}(W_{1,0}), \mathcal{C}(W_{1,1}), \mathcal{C}(W_{2,0})$, order where $\mathcal{C}(\cdot)$ is the additive cost functional and $W_{j,n}$ is the DWPT crystal at level j and oscillation (local node) index n for $j = 1, \ldots, J$.

References

Ronald R. Coifman and Mladen Victor Wickerhauser, “Entropy-Based Algorithms for Best Basis Selection", *IEEE Transactions on Information Theory*, **38**(2), pp. 713–718, 1992.

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavDWPT.

Examples

```
## calculate the DWPT of the difference of the
## atomic clock series
W <- wavDWPT(diff(atomclock), n.level=6)

## define an entropy cost functional
"entropy" <- function(x){
    iz <- which(x==0)
    z <- -x^2 * log(x^2)
    if (length(iz))
       z[iz] <- 0
    sum(z)
}

## create the cost vector
C <- unlist(lapply(W$data, entropy))

## calculate the bets basis
z <- wavBestBasis(C)

## print the crystals of the best basis
paste("W(", z$level, ",", z$osc, ")", sep="")
```

---

wavBootstrap *Adaptive wavelet-based bootstrapping*

Description

Given a set of indices which represent the whitest transform available in a DWPT, this function randomizes the coefficients in each of the crystals comprising the transform (via random selection with replacement) followed by an inverse transform. The z is a bootstrapped version of the original time series.

Usage

```
wavBootstrap(x, white.indices=wavDWPTWhitest(x),
    n.realization=1, wavelet="s8", n.level=NULL)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series or an object of class wavTransform as output by the wavDWPT function.
n.level	the number of decomposition levels. This argument is used only if x is a time series. Default: floor(logb(length(x), base=2)) - 2.
n.realization	the number of realizations to generate. Default: 1.
wavelet	a character string denoting the filter type. See wavDaubechies for details. This argument is used only if x is a time series. Default: "s8".
white.indices	a list containing the level and osc vectors denoting the level and oscillation index, respectively, of the whitest transform. Default: wavDWPTWhitest(x).

Value

a list of numeric vectors containing the bootstrapped series. If `n.realization=1`, the the output is a numeric vector (not packed into a `list`).

References

D. B. Percival, S. Sardy and A. C. Davison, *Wavestrapping Time Series: Adaptive Wavelet-Based Bootstrapping*, in W. J. Fitzgerald, R. L. Smith, A. T. Walden and P. C. Young (Eds.), *Nonlinear and Nonstationary Signal Processing*, Cambridge, England: Cambridge University Press, 2001.

See Also

`wavDWPT`, `wavDWPTWhitest`.

Examples

```
## wavestrap the sunspots series
x <- as.numeric(sunspots)
z <- wavBootstrap(x, n.realization=1)

ifultools::stackPlot(x=seq(along=sunspots),
y=data.frame(x, z, abs(z)),
ylab=list(text=c("sunspots","wavestrap","|wavestrap|")))

title("Wavelet-based bootstrapping of sunspots series", cex=0.7)
```

---

wavBoundary *Wavelet transform boundary coefficent identification*

Description

A wavelet transform boundary coefficient is one subject to circular filter operations (or other boundary treatments). Conversely, the interior transform coefficients are those that are not affected by the imposed boundary treatment. The `wavBoundary` function separates the boundary coefficients from the interior wavelet transform coefficients.

Usage

```
wavBoundary(x)
```

Arguments

`x`	a DWT or MODWT transform object with class `wavTransform`.

Value

an object of class `wavBoundary`.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and, Applied Mathematics, 41, 909–96.

See Also

wavIndex, wavDWT, wavMODWT, wavShift.

Examples

```
## calculate the MODWT of the sunspots series
W <- wavMODWT(sunspots)

## identify the boundary coefficients
z <- wavBoundary(W)

## plot the results
plot(wavShift(z))

## obtain a summary
summary(z)
```

---

wavCWT *Continuous wavelet transform*

Description

The continuous wavelet transform (CWT) is a highly redundant transformation of a real-valued or complex-valued function $f(x)$, mapping it from the time domain to the so-called time-scale domain. Loosely, speaking the CWT coefficients are proportional to the variability of a function at a given time and scale.

The CWT is defined by a complex correlation of a scaled and time-shifted mother wavelet with a function $f(x)$. Let $\psi(x)$ be a real- or complex-valued function representing a *mother* wavelet, i.e. a function which meets the standard mathematical criteria for a wavelet and one that can be used to generate all other wavelets within the same family. Let $\psi^*(\cdot)$ be the complex conjugate of $\psi(\cdot)$. The CWT of $f(x)$ is defined as

$$W_f(a,b) \equiv \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(x)\psi^*\Big(\frac{x-b}{a}\Big)\, dx,$$

for $(a,b) \in \mathcal{R}$ and $a > 0$, where a is the scale of the wavelet and b is the shift of the wavelet in time. It can be shown that the above complex correlation maintains a duality with the Fourier transform defined by the relation

$$W_f(a,b) \equiv \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(x)\psi^*\Big(\frac{x-b}{a}\Big)\, dx \longleftrightarrow \sqrt{a}\, F(\omega)\, \Psi^*(a\omega)$$

where $F(\cdot)$ is the Fourier transform of $f(x)$ and ω is the frequency in radians. This function calculates the CWT in the Fourier domain followed by an inverse Fourier transform.

Usage

```
wavCWT(x, scale.range=deltat(x) * c(1, length(x)), n.scale=100,
    wavelet="gaussian2", shift=5, variance=1)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series. The time series may be of class class rts, ts, cts, or signalSeries, or be a numeric vector.
n.scale	the number of scales to evaluate over the scale.range. Default: 100.
scale.range	a two-element vector containing the range of scales over which to evaluate the CWT. The smallest specified scale must be greater than or equal to the sampling.interval of the time series. Default: deltat(x) * c(1, length(x)).
shift	numeric value representing the frequency shift to use for the Morlet wavelet filter. Default: 5.
variance	if the wavelet filter is of type "gaussian1" or "gaussian2" then this parameter represents the variance of the Gaussian PDF used to scale the corresponding filters. Default: 1.
wavelet	a character string denoting the wavelet filter to use in calculating the CWT. Choices are "haar", "gaussian1", "gaussian2", and "morlet", where gaussian1 and gaussian2 represent the first and second derivatives of a Gaussian PDF. Default: "gaussian2".

Value

an object of class wavCWT.

S3 METHODS

as.matrix returns the CWT coefficients as a complex matrix with rows and columns representing times and scales, respectively.

plot plots the CWT. By default, the modulus of the CWT coefficients are plotted in the time-scale plane. The plot method also supports the following optional arguments:

- **phase** Logical flag. If TRUE, the phase of the CWT is plotted. Default: FALSE.
- **xlab** A character string used for the label on the x-axis. Default: "time".
- **ylab** A character string used for the label on the y-axis. Default: "log2(units)" where units are the units of the time series if available. If units are not available, "log2(scale)" is used.
- **power.stretch** A numeric value used to scale the magnitude of the CWT coefficients for display purposes only. If power.stretch=0, then image is transformed with log(abs(x)+1). Otherwise, the image is transformed with (abs(x))^power.stretch. Default: 0.5.
- **type** A character string denoting the type of plot to produce. Choices are "image" and "persp" which plot the CWT as an image or as a meshed perspective plot, respectively. The perspective plot resamples the data to contain a maximum of grid.size rows and columns.

zoom A four-element vector containing the ranges to zoom into the data in c(xmin, xmax, ymin, ymax) format. Default: `NULL` (no zoom).

grid.size An integer specifying the maximum number of lines to use for each dimension in creating meshed perspective plots. Default: `100`.

new A logical flag. If `TRUE`, a new plot is forced with the `frame()` command.

print prints a qualitiative summary of the CWT and its components.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavCWTFilters.

Examples

```
## calculate the CWT of the sunspots series using
## a Mexican hat wavelet (gaussian2)
sunspots.cwt <- wavCWT(sunspots)

## print the result
print(sunspots.cwt)

## plot an image of the modulus of the CWT and the
## time series
plot(sunspots.cwt, series=TRUE)

## plot a coarse-scale wire-frame perspective of
## the CWT
plot(sunspots.cwt, type="persp")
```

---

wavCWTFilters *Frequency response of continuous wavelet transform filters*

Description

Returns the frequency response of a continuous wavelet filter. The choices for filters are limited to Haar, Gaussian, and Morlet families.

Usage

```
wavCWTFilters(wavelet="Gaussian2", frequency=seq(0, 2 * pi, length=1000),
    shift=3, variance=1, times=NULL)
```

Arguments

frequency	a numeric vector denoting the frequencies (in rad/sec) over which the frequency response function for the specified wavelet should be evaluated. Default: `seq(0, 2 * pi, length=1000)`
shift	the frequency shift ω_0 of the Morlet wavelet. Default: 3.
times	a numeric vector of values corresponding to times at which the specified filter should be evaluated. If not NULL, the impulse response of the specified filter is returned, otherwise the freequency response is returned. Default: NULL.
variance	the variance of a Gaussian PDF. Used only for the (derivatives of) Guassian filters. Default: 1.
wavelet	a character string denoting the wavelet filter. Choices are "haar", "gaussian1", "gaussian2", and "morlet", where gaussian1 and gaussian2 represent the first and second derivatives of a Gaussian PDF. Default: "gaussian2".

Value

the frequency response corresponding to the input frequencies.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavCWT.

Examples

```
## set the plot grid
old.plt <- ifultools::splitplot(2,2,1)

## create a frequency vector
w <- seq(-10,10,length=1000)

## calculate the frequency responses for various
## mother wavelets and plot the results

wavelets <- c("haar", "gaussian1", "gaussian2", "morlet")

for (i in seq(wavelets)){
    if (i > 1) ifultools::splitplot(2,2,i)
    filt <- wavelets[i]
    plot(w, abs(wavCWTFilters(wavelet=filt, frequency=w)),
       ylab="|Psi(w)|", xlab="frequency", type="l")
title(filt)
}

par(old.plt)
```

wavCWTPeaks *Peak detection in a time series via the CWT*

Description

Finds the local maxima in a time series via a CWT tree.

Usage

```
wavCWTPeaks(x, snr.min=3, scale.range=NULL, length.min=10,
    noise.span=NULL, noise.fun="quantile", noise.min=NULL)
```

Arguments

x	an object of class wavCWTTree.
length.min	the minimum number of points along a CWT tree branch and within the specified scale.range needed in order for that branches peak to be considered a peak candidate. Default: 10.
noise.fun	a character string defining the function to apply to the local noise estimates in order to sumarize and quantify the local noise level into a scalar value. See the **DETAILS** section for more information. Supported values are **"quantile"** quantile(x, probs=0.95) **"sd"** sd(x) **"mad"** mad(x, center=0) where x is a vector of smallest-scale CWT coefficients whose time indices are near that of the branch termination time. Default: "quantile".
noise.min	the minimum allowed estimated local noise level. Default: quantile(attr(x,"noise"), prob=0.05), where x is the input wavCWTTree object.
noise.span	the span in time surrounding each branche's temrination point to use in forming local noise estimates and (ultimately) peak SNR estimates. Default: NULL,max(0.01 * diff(range(tim where times and sampling.interval are attributes of the input wavCWTTree object.
scale.range	the range of CWT scales that a peak must fall into in order to be considered a peak candidate. Default: scale[range(which(branch.hist > quantile(branch.hist,prob=0.8))) where branch.hist is an attribute of the input wavCWTTree object. This default results in isolating the bulk of energetic CWT branches, but the user in encouraged to reduce the scale range in order to attenuate the computational burden of the peak detection scheme.
snr.min	the minimum allowed peak signal-to-noise ratio. Default: 3.

Details

The local maxima of the CWT are linked together to form so-called branches, where each branch represents one *ridge* of the CWT time-scale terrain. The collection of branches forms a tree, as output by the `wavCWTTree` function. The `wavCWTpeaks` function prunes the branches of the input CWT tree and records the termination time (i.e., the time associated with point of the branch that is closest to scale zero) as the time index associated with the local peak of the corresponding time series. Information regarding the collection of isolated peaks is returned as a `data.frame` object.

The tree branches are pruned in the following ways:

peak SNR an estimate of SNR at peak value is greater than or equal to the specified `snr.min`. A peak SNR estimate is formed as follows: For each branch of the input CWT tree, a subset of CWT coefficients is collected such that the CWT coefficients are both local to the branch termination time and correspond to the smallest analyzed CWT scale. The user specified `noise.span` argument is used to define the boundaries of each subset in time ala $[B-$`noise.span`, $B+$`noise.span`$]$, where B is the branch termination time. Each CWT subset is assumed to be representative of the local noise levels near the corresponding branch termination time and `noise.fun` is used to quantify (and summarize) each level resulting in a scalar z. The minimum value of z is specified by the user ala the `noise.min` argument. Finally, the ratio $|P|/|W|$ is used to form an estimate of the local signal-to-noise ration (SNR) for the corresponding branch, where P is the maximum CWT value along the branch in the CWT time-scale plane.

scale the scale corresponding to the peak is larger than the minimum of the specified `scale.range`.

branch length the length of the branch within the specified `scale.range` is greater than or equal to the specified minimum `length.min`.

endpoint the index of the terminating time of the branch is on the interval $(W, N - W)$, where N is the length of the series and W is integer equivalent of $1/4$ the length of the `noise.span` or 3, whichever is greater.

NOTE: For peak detection, the wavelet filters used to form the CWT must maintain an (approximate) zero phase property so that the CWT coefficients can be meaningfully aligned with the events of the original time series. Currently, only the so-called Mexican hat wavelet maintains this property due to the even-symmetry of the filter's impulse response. Therefore, only the Mexican hat wavelet ("gaussian2") is currently supported for CWT-based peak detection. See the `wavCWTFilters` and `wavCWT` function for more information.

Value

a list of `x` and `y` vectors identifying the peaks in the original time series. The pruning criteria (`snr.min`, `scale.range`, `length.min`, `noise.span`, `noise.fun`, `noise.min`) are attached are attached as attributes. In addition, a `peaks` attribute is attached and corresponds to a `data.frame` containing the following information for each peak:

`branch`	index of the associated branch in the CWT tree
`itime`	index location in time
`iscale`	index location in scale
`time`	location in time
`scale`	location in scale

extrema	CWT value
iendtime	index location of branch termination time, i.e., the index of the point in the time series corresponding to the current peak

References

Pan Du, Warren A. Kibbe, and Simon M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching", *Bioinformatics*, **22**, 2059–2065 (2006).

J.F. Muzy, E. Bacry, and A. Arneodo., "The multifractal formalism revisited with wavelets.", *International Journal of Bifurcation and Chaos*, **4**, 245–302 (1994).

See Also

wavCWTTree, wavCWT, wavCWTFilters.

Examples

```
## create linchirp series
linchirp <- make.signal("linchirp")

## calculate the CWT
W <- wavCWT(linchirp)

## form CWT tree
z <- wavCWTTree(W)

## estimate the peak locations using default
## scale.range
p <- wavCWTPeaks(z)

## plot an overlay of the original series and the
## peaks
x <- as(linchirp@positions,"numeric")
y <- linchirp@data
plot(x, y, type="l", xlab="time", ylab="linchirp")
points(p, pch=16, col="red", cex=1.2)
```

---

wavCWTTree *Tree map of continuous wavelet transform extrema*

Description

This function first finds the extrema locations (in time and in scale) of the continuous wavelet transform input. The set of extrema are then subdivided into sets of *branches*, where each branch represents a collection of extrema that correspond to the same ridge in the CWT time-scale plane. A coarse-to-fine scale strategy is used to identify the members of each branch as follows: (i) a single extremum at the coarsest scale is selected as the start of a given branch, (ii) the closest neighboring

extremum in time at the next finest scale is then added to the branch, (iii) step ii is repeated until the smallest scale is reached or an apparent break occurs in the branch across scale, and (iv) steps i-iii are repeated until all extrema have been accounted. A branch is not grown unless the nearest neighbor candidate at the next finest scale is close in time to the last recorded branch member, where "close" is defined as being less than the current scale of the neighbor candidate. This means that the window in time for admissible neighbor extrema candidates (at the next finest scale) shrinks proportionally with scale.

Usage

```
wavCWTTree(x, n.octave.min=1, tolerance=0.0, type="maxima")
```

Arguments

`x`	an object of class `wavCWT` (as produced by the `wavCWT` function).
`n.octave.min`	a pruning factor for excluding non-persistent branches. If a branch of connected extrema does not span this number of octaves, it is excluded from the tree. Default: 1.
`tolerance`	a tolerance vector used to find CWT extrema. This vector must be as long as there are scales in the CWT such that the j^{th} element defines the tolerance to use in finding modulus maxima at the j^{th} scale of the CWT. If not, the last value is replicated appropriately. Default: 0.
`type`	a character string denoting the type of extrema to seek in the CWT plane. Supported types are `"extrema"`, `"maxima"` and `"minima"`. Default: `"maxima"`.

Details

A point in the CWT $W(t,j)$ is defined as an extremum if $|W(t-1,j)| + tol < |W(t,j)|$ and $|W(t+1,j)| + tol < |W(t,j)|$ where `tol` is a (scale-dependent) tolerance specified by the user. The search algorithm is also adpated to identify plateaus in the data, and will select the the middle of the plateau as a maximum location when encountered. The data $|W(t,j)|$ is first scaled so that its maximum value is 1.0, so the tolerances should be adjusted accordingly. Since the CWT coefficients are (in effect) a result band-pass filtering operations, the large scale coefficients form a smoother curve than do the small scale coefficients. Thus, the tolerance vector allows the user to specify scale-dependent tolerances, helping to weed out undesirable local maxima. It is recommended that the tolerance be set proportional to the scale, e.g., `tolerance=C / sqrt(scale)` where C is a constant $0 < C < 1$. The user is also allowed to control the types of peaks to pursue in the CWT plane: extrema, maxima, or minima. The algorithm (described above) is adjusted accordingly.

The output object contains a list of sublists, each sublist corresponds to a single branch in the CWT tree and contains the named vectors:

itime index locations in time of CWT extrema

iscale index locations in scale of CWT extrema

time times associated with CWT extrema

scale scales associated with CWT extrema

extrema CWT extrema values

In addition, the returned object contains the following attributes:

iendtime integer vector of indices corresponding to the locations in time where the branches terminated as the scale appraoches zero.

endtime numeric vector containing branch termination times

time numeric vector of times corresponding to the original time series

scale numeric vector of scales used to form the CWT

extrema.mask a binary matrix (of the same dimension as the CWT matrix) containing a 1 where there exists a corresponding extremum value in the CWT plane

noise a numeric vector containing the first scale's CWT coefficients. Statistical analysis of these data are often used as a rough estimate of the (local) noise level(s) in the original time series.

branch.hist a numeric vector containing the sum across time of all extrema values. This data can be used to help automate the selection of scales of interest in the CWT plane.

Value

an object of class `wavCWTTree`. See **DETAILS** section for more information.

S3 METHODS

[extracts a subset of branches from the tree. For example, to extract branches 2 through 5, use the syntax `x[2:5]`. To extract branches which terminate near times 0.47, 0.3, and 1.4, use the syntax `x[time=c(0.47, 0.3, 1.4)]`. To extract all branches which terminate between times 1.2 and 1.5, use the syntax `x[range=c(1.2, 1.5)]`.

plot plots the WTMM tree. The plot method also supports the following optional arguments (assume that the variable `x` is an output of the `wavCWTTree` function):

- **fit** A logical flag. If `fit=TRUE`, a subset of branches (limited to four) are fit with various linear regression models on a log(|WTMM|) versus log(scale) basis. The models are specified by the optional `models` argument. This scheme illustrates the process by which exponents are estimated using the WTMM branches. For example, to see the regressions over chains 10 through 13, issue `plot(x[10:13], fit=TRUE)`. Default: `FALSE`.
- **models** A vector of character strings denoting the linear models to use in illustring the calculation of exponents. This argumetn is used only if `fit=TRUE`. Default: `c("lm", "lmsreg", "ltsreg")`.
- **labels** Logical flag. If `TRUE`, the branch number is placed at the head of each branch. Default: `TRUE`.
- **extrema** A logical flag. If `TRUE`, the locations of the (non-pruned and unbranched) extrema are marked in the time-scale plane. Default: `FALSE`.
- **pch** The marker used in plotting branch points via the `par` function. Default: `"o"`.

print prints a summary of the object.

summary sumamrizes the branches comprising the tree ala a data.frame object

References

J.F. Muzy, E. Bacry, and A. Arneodo., "The multifractal formalism revisited with wavelets.", *International Journal of Bifurcation and Chaos*, **4**, 245–302 (1994).

See Also

`wavCWT`, `wavCWTFilters`.

Examples

```
## create linchirp series
linchirp <- make.signal("linchirp")

## calculate the CWT
W <- wavCWT(linchirp)

## form CWT tree
W.tree <- wavCWTTree(W)

## print the object
print(W.tree)

## summarize the object
summary(W.tree)

## plot thea CWT image with a tree overlay
## (R-only)
plot(W)
if (is.R()) plot(W.tree, extrema=TRUE, add=TRUE)

## plot all CWT tree branches
plot(W.tree)

## plot a subset of CWT tree branches
plot(W.tree[5:10])

## plot an illustration of the Holder exponent
## estimation process. select branches between
## times 0.2 and 0.4 (only the first four found
## will be fitted)
plot(W.tree[range=c(0.2, 0.4)], fit=TRUE)
```

---

wavDaubechies *Daubechies wavelet and scaling filters*

Description

Ingrid Daubechies, a noted pioneer in wavelet theory, has established a number of wavelet filter types, each with different mathematical properties. This function calculates the wavelet and scaling coefficients for a given filter type. The wavelet coefficients, h_k for $k = 0, \ldots, L-1$ where L is the filter length, are related to the scaling coefficients through the quadrature mirror filter (QMF) relation

$$h_k = (-1)^{k-L} g_{L-1-k}.$$

Usage

```
wavDaubechies(wavelet="s8", normalized=TRUE)
```

Arguments

`normalized` a logical value. If `TRUE`, the filters are normalized by dividing each filter coefficient by the $\sqrt{2}$ (useful for maximum overlap wavelet transforms). If `FALSE`, no normalization is used. Default: `TRUE`.

`wavelet` a character string denoting the filter type. Supported types include:

EXTREMAL PHASE (daublet): "haar", "d2", "d4", "d6", "d8", "d10", "d12", "d14", "d16", "d18", "d20"

LEAST ASYMMETRIC (symmlet): "s2","s4", "s6", "s8", "s10", "s12", "s14", "s16", "s18", "s20"

BEST LOCALIZED: "l2","l4", "l6", "l14", "l18", "l20"

COIFLET: "c6", "c12", "c18", "c24", "c30"

Default: `"s8"`.

Details

Only relevant for Daubechies filter types. Inconsistent ordering of the coefficients in Daubechies' book was recognized and corrected by Percival (see references). The "correct" order is given here.

Value

an object of class `wavDaubechies`.

S3 METHODS

plot plot Daubechies filters.

Usage: plot(x, type="time")

x A `wavDaubechies` object.

type A character string denoting the type of plot to produce. Choices are `"time"`, `"gain"`, and `"phase"` for an impulse response, squared gain, and phase plot, respectively. Default: `"time"`.

print print Daubechies filters.

Usage: print(x, verbose=TRUE)

x A `wavDaubechies` object.

verbose A logical value. If `TRUE`, the filter coefficients are also printed. Default: `TRUE`.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and, Applied Mathematics, 41, 909–96.

See Also

`wavGain`, `wavDWT`, `wavMODWT`, `wavMODWPT`.

Examples

```
## obtain Daubechies least asymmetric 8-tap filter
## set
filters <- wavDaubechies("s8", normalize=TRUE)

## plot the impulse responses
plot(filters, type="time")

## plot the gain function
plot(filters, type="gain")
```

`wavDictionary` *Constructor function for objects of class wavDictionary*

Description

Packs input information regarding a discrete wavelet transform into a dictionary list.

Usage

```
wavDictionary(wavelet, dual, decimate, n.sample,
    attr.x, n.levels, boundary, conv, filters,
    fast, is.complex)
```

Arguments

`wavelet`	a character string denoting the type of wavelet used in the transform.
`dual`	a logical value. If `TRUE`, it signifies that a dual transform was performed.
`decimate`	a logical value. If `TRUE`, it signifies that a decimated transform was performed.
`n.sample`	an integer representing the number of samples in the original time series.
`attr.x`	a list of additional (arbitrary) attributes to append onto the output object.
`n.levels`	an integer denoting the number of decomposition levels.
`boundary`	a character string denoting the boundary extension type used in transform. Supported values are `"zero"`, `"periodic"`, `"reflection"`, and `"continue"`.
`conv`	a logical value. If `TRUE`, it signifies that a convolution style transform was performed (as opposed to correlation style).
`filters`	a list of vectors named `"scaling"` and `"wavelet"` containing the scaling and wavelet filter coefficients, respectively.
`fast`	a logical value. If `TRUE`, it signifies that a fast pyramidal scheme was used to develop the decimated transform as opposed to calculating the transform coefficients via an explicit matrix multiplication of the wavelet transform matrix and the original time series.
`is.complex`	a logical value. If `TRUE`, it signifies the transform was complex-valued.

Details

Used internally by the wavMODWT and wavDWT functions to package the transform contents into a dictionary list.

Value

an object of class wavDictionary.

S3 METHODS

print print the dictionary.

Usage: print(x)

See Also

wavDWT, wavMODWT.

Examples

```
## create a faux wavelet dictionary
wavelet <- "s8"
wavDictionary(wavelet=wavelet, dual=FALSE,
    decimate=FALSE, n.sample=1024,
    attr.x=NULL, n.levels=3,
    boundary="periodic", conv=TRUE,
    filters=wavDaubechies(wavelet),
    fast=TRUE, is.complex=FALSE)
```

---

wavDWPT *The discrete wavelet packet transform (DWPT)*

Description

Given j, n, t are the decomposition level, oscillation index, and time index, respectively, the DWPT is given by

$$W_{j,n,t} = \sum_{l=0}^{L-1} u_{n,l} W_{j-1,\lfloor n/2 \rfloor, 2t+1-l \mod N_{j-1}}, \qquad t = 0, \ldots, N_j - 1,$$

where $N_j \equiv N/2^j$ and $\lfloor \cdot \rfloor$ denotes the integer part. The variable L is the length of the filters defined by

$$u_{n,l} \equiv \begin{cases} g_l, & \text{if } n \bmod 4 = 0 \text{ or } 3; \\ h_l, & \text{if } n \bmod 4 = 1 \text{ or } 2, \end{cases}$$

The variables g and h represent the scaling filter and wavelet filter, respectively. Each filter is of length L. By definition, $W_{0,0,t} \equiv X_t$ where $\{X_t\}$ is the original time series.

Usage

```
wavDWPT(x, wavelet="s8", n.levels=ilogb(length(x), base=2),
    position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character())
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
documentation	a character string used to describe the input data. Default: character().
n.levels	the number of decomposition levels. Default: as.integer(floor(logb(length(x),base=2))).
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
title.data	a character string representing the name of the input data. Default: character().
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".

Value

an object of class wavTransform.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavBestBasis, wavPacketBasis, reconstruct, wavDWT, wavMODWT, wavMODWPT, wavDaubechies, wavMaxLevel.

Examples

```
## calculate the DWPT of sunspots series out to 3
## levels using Daubechies least asymmetric
## 8-tap filter set
z <- wavDWPT(sunspots, wavelet="s8", n.levels=3)

## plot the transform
plot(z)

## summarize the transform
summary(z)
```

`wavDWPTWhitest` *Seeks the whitest transform of a discrete wavelet packet transform (DWPT)*

Description

This function seeks the whitest orthonormal transform of a DWPT. The goal is to segment the normalized frequency interval $f \in [0, 1/2]$ into subintervals such that, within each subinterval, the variability of the (corresponding) spectral density function (SDF) is minimized, i.e., each segment of the SDF is as flat as possible. Given an N-point uniformly sampled time series $\mathbf{X}$, and denoting $\mathbf{W}_{j,n}$ as the DWPT crystal at level j and (sequency ordered) oscillation index n, this optimization is achieved as follows:

1 Perform a level $J - 2$ partial DWPT of $\mathbf{X}$ where $J = \lfloor \log_2(\mathbf{X}) \rfloor$. By definition, $\mathbf{W}_{0,0} \equiv \mathbf{X}$. Begin step 2 with $j = n = 0$.

2 Perform a white noise test on the current (parent) crystal: $\mathbf{W}_{j,n}$. If it passes (or the current crystal is in the last decomposition level) retain the crystal. Otherwise, discard the current parent crystal and perform the white noise test on its children: $\mathbf{W}_{j+1,2n}$ and $\mathbf{W}_{j+1,2n+1}$.

3 Repeat step 2 as many times as necessary until a suitable transform is found.

Usage

```
wavDWPTWhitest(x, significance=0.05, test="port2", wavelet="s8", n.level=NULL)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series or an object of class `wavTransform`.
`n.level`	the number of decomposition levels. This argument is used only if `x` is a time series. Default: `floor(logb(length(x), base=2)) - 2`.
`significance`	a numeric value on the interval (0,1) which qualitatively signifies the fraction of times that the white noise hypothesis is incorrectly rejected. The `significance` is used to calculate comparative chi-square distribution p x 100 percentage points where $p = 1 -$ significance (the chi-square degrees of freedom are estimated automatically within the specified white noise test). Default: `0.05`.
`test`	a character string denoting the white noise test to use. Options are `"port1"`, `"port2"`, `"port3"` and `"cumper"` respresenting the Portmanteau I, II, III and cumulative periodogram tests, respectively. See the reference(s) for more details. Default: `"port2"`.
`wavelet`	a character string denoting the filter type. See `wavDaubechies` for details. This argument is used only if `x` is a time series. Default: `"s8"`.

Value

a `list` containing the `level` and `osc` vectors denoting the level and oscillation index, respectively, of the whitest transform.

References

D. B. Percival, S. Sardy and A. C. Davison, *Wavestrapping Time Series: Adaptive Wavelet-Based Bootstrapping*, in W. J. Fitzgerald, R. L. Smith, A. T. Walden and P. C. Young (Eds.), *Nonlinear and Nonstationary Signal Processing*, Cambridge, England: Cambridge University Press, 2001.

See Also

wavDWPT, wavBootstrap.

Examples

```
## calculate the DWPT of the sunspots series
W <- wavDWPT(as.numeric(sunspots), wavelet="s8", n.levels=9)

## find the whitest transform based on the
## Portmanteau I white noise test
z <- wavDWPTWhitest(W, test="port1")

print(z)
```

---

wavDWT *The discrete wavelet transform (DWT)*

Description

The discrete wavelet transform using convolution style filtering and periodic extension.

Let j, t be the decomposition level, and time index, respectively, and $s_{0,t} = X_{t=0}^{N-1}$ where X_t is a real-valued uniformly-sampled time series. The j^{th} level DWT wavelet coefficients ($d_{j,t}$) and scaling coefficients ($s_{j,t}$) are defined as $d_{j,t} \equiv \sum_{l=0}^{L-1} h_l s_{j-1,2t+1-l \bmod N_{j-1}}, \quad t = 0, \ldots, N_j - 1$ and $s_{j,t} \equiv \sum_{l=0}^{L-1} g_l s_{j-1,2t+1-l \bmod N_{j-1}}, \quad t = 0, \ldots, N_j - 1$ for $j = 1, \ldots, J$ where $\{h_l\}$ and $\{g_l\}$ are the j^{th} level wavelet and scaling filter, respectively, and $N_j \equiv N/2^j$. The DWT is a collection of all wavelet coefficients and the scaling coefficients at the last level: $\mathbf{d_1}, \mathbf{d_2}, \ldots, \mathbf{d_J}, \mathbf{s_J}$ where $\mathbf{d_j}$ and $\mathbf{s_j}$ denote a collection of wavelet and scaling coefficients, respectively, at level j.

Usage

```
wavDWT(x, n.levels=ilogb(length(x), base=2),
   wavelet="s8", position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character(), keep.series=FALSE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
documentation	a character string used to describe the input data. Default: character().
keep.series	a logical value. If TRUE, the original series is preserved in the output object. Default: FALSE.

n.levels	the number of decomposition levels. Default: as.integer(floor(logb(length(x),base=2))).
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
title.data	a character string representing the name of the input data. Default: character().
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".

Details

This DWT imposes an ad hoc storage sytem for odd length scaling coefficient crystals: if the length of a scaling coefficient crystal is odd, the last coefficient is "stored" in the *extra* crystal. During reconstruction, any extra scaling coefficients are returned to their proper location. Such as system imposes no spurious energy in the transform coefficients at the cost of a little bookkeeping.

Value

an object of class wavTransform.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

reconstruct, wavDaubechies, wavMODWT, wavMODWPT, wavMRD, wavDictionary, wavIndex, wavTitle, wavBoundary, wavShrink.

Examples

```
## calculate the DWT of linear chirp
linchirp <- make.signal("linchirp", n=1024)
result   <- wavDWT(linchirp, wavelet="s8", n.levels=5, keep.series=TRUE)

## plot the transform shifted for approximate zero
## phase alignment
plot(wavShift(result))

## plot summary
eda.plot(result)

## summarize the transform
summary(result)
```

wavEDOF *Equivalent degrees of freedom (EDOF) estimates for a chi-squared distribution*

Description

Let X be a collection of M uncorrelated zero mean Gaussian random variables (RVs). The sum of the squares of the RVs in X will obey a scaled chi-square distribution with M degrees of freedom (DOF). If, however, the original Gaussian RVs are (partially) correlated, we can approximate the distribution of the sum of the squares of (correlated Gaussian) RVs using a scaled chi-square distribution with the DOF adjusted for the correlation in the RVs. These adjusted DOF estimates are known as the *equivalent degrees of freedom* (EDOF). In the context of unbiased wavelet variance analysis, the EDOF can be used to estimate confidence intervals that are guaranteed to have non-negative bounds.

This program calculates three estimates of the EDOF for each level of a discrete wavelet transform. The three modes are described as follows for the MODWT of an an input sequence $\{X_t\}_{t=0}^{N-1}$:

EDOF 1 Large sample approximation that requires an SDF estimation via wavelet coefficients.

$$\hat{\eta}_1 = \frac{M_j(\hat{s}_{j,0})^2}{\hat{A}_j},$$

where $\hat{s}_{j,\tau}$ is the autocovariance sequence defined by

$$\hat{s}_{j,\tau} \equiv \frac{1}{M_j} \sum_{t=0}^{M_j-1} \tilde{d}_{j,t}^{(int)} \tilde{d}_{j,t+|\tau|}^{(int)} \;\; 0 \leq |\tau| \leq M_j - 1,$$

and $\tilde{d}_{j,t}^{(int)}$ are the M_j j^{th} level interior MODWT wavelet coefficients and $\hat{A}_j$ is defined as

$$\hat{A}_j \equiv \frac{(\hat{s}_{j,0})}{2} + \sum_{\tau=1}^{M_j-1} (\hat{s}_{j,\tau})^2.$$

EDOF 2 Large sample approximation where the SDF is known a priori.

$$\hat{\eta}_2 = \frac{2\left(\sum_{k=1}^{\lfloor (M_j-1)/2 \rfloor} C_j(f_k)\right)^2}{\sum_{k=1}^{\lfloor (M_j-1)/2 \rfloor} C_j^2(f_k)},$$

where $f_k \equiv k/M_j$ and $C_j \equiv \widetilde{\mathcal{H}}_j^{(D)}(f) S_X(f)$ is the product of Daubechies wavelet filter squared gain function and the spectral density function of X_t.

EDOF 3 Large sample approximation using a band-pass approximation for the SDF.

$$\hat{\eta}_3 = \max\{M_j/2^j, 1\}$$

.

See references for more details.

Usage

```
wavEDOF(x, wavelet="s8", levels=NULL, sdf=NULL, sdfargs=NULL,
    sampling.interval=1, n.fft=1024)
```

Arguments

x	an object of class wavTransform or a vector containing a uniformly-sampled real-valued time series.
levels	a vector containing the decomposition levels. Default: when x is of class wavTransform then levels is set to 1:n.level, otherwise levels is set to 1:J, where J is the maximum wavelet transform level in which there exists at least one interior wavelet coefficient.
n.fft	a positive integer (greater than one) defining the number of frequencies to use in evaluating the SDF for EDOF 2 calculations. The frequencies are uniformly distributed over the interval [0, Nyquist] ala f=[0, 1/P , 2/P, 3/P, ..., (n.freq-1)/P] where P=2*(n.freq-1)/sampling.interval. Only used when the input SDF is not NULL. Default: 1024.
sampling.interval	sampling interval of the time series. Default: 1.
sdf	a spectral density function of the process corresponding to the input time series. This input must be a function whose first argument is f (representing frequency). At a minimum, the SDF must be defined over frequencies [0, Nyquist] where Nyquist=1/(2*sampling.interval). Additional arguments that are needed to calculate the SDF should be passed via the sdfargs parameter. This argument is used only for calculating mode 2 EDOF. If the EDOF mode 2 estimates are not desired, specify this this argument as NULL and the EDOF mode 2 and corresponding confidence intervals will not be calculated. See the mutilsSDF function for more details. Default: NULL.
sdfargs	a list of arguments passed directly to the SDF function ala do.call. Default: NULL (no additional arguments).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Only used if input x is a time series. Default: "s8".

Value

a list containing the EDOF estimates for modes 1, 2 and 3 as well as the block-dependent unbiased wavelet variance estimates.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavVar, wavVarConfidence, mutilsSDF.

Examples

```
## initialize variables
n.level <- 9
wavelet <- "d6"
N       <- 4096
phi     <- 0.9

## define input SDF
S <- function(f, phi)  1/(1 + phi^2 - 2*phi*cos(2*pi*f))
sdfarg <- list(phi=phi)

## create series and MODWT
set.seed(100)
x <- rnorm(N)
W <- wavMODWT(x, wavelet=wavelet, n.level=n.level)

## calculate EDOF using the wavTransform object
z1 <- wavEDOF(W, sdf=S, sdfarg=sdfarg)
print(z1)

## calculate EDOF using original time series
z2 <- wavEDOF(x, wavelet=wavelet, levels=seq(n.level), sdf=S, sdfarg=sdfarg)
print(z2)

## compare the two approaches
print(all.equal(z1,z2))
```

---

wavFDP *Class constructor for block- and time-dependent wavelet-based FD model parameter estimators*

Description

Class constructor for block- and time-dependent wavelet-based FD model parameter estimators.

Usage

```
wavFDP(estimator, delta, variance.delta,
    innovations.variance, delta.range, dictionary, levels,
    edof.mode, boundary, series, sdf.method, type)
```

Arguments

`estimator`	character string briefly describing the estimator.
`delta`	numeric value/vector denoting the estimated FD model parameter.
`innovations.variance`	numeric value/vector denoting the estimated FD innovations variance.
`variance.delta`	numeric value/vector defining the variance of delta.

delta.range	two element numeric vector defining the range of delta.
dictionary	wavelet transform dictionary used in the analysis.
levels	vector of integers denoting the wavelet decomposition levels used in the analysis.
edof.mode	an integer on [1,3] defining the equivalent degrees of freedom mode used in the analysis.
boundary	a list containing named objects mode and description, containing a logical value and a character string, respectively. The mode object should be be TRUE if a boundary treatment was used, and description should contain a description of the boundary treatment.
series	a signSeries object containing the input series.
sdf.method	a character string defining the SDF method used in the analysis, e.g., "Integration lookup table".
type	a character string defining the type of estimator, e.g., ""instantaneous"" or "block".

S3 METHODS

eda.plot extended data analyis plot of the data. Available options are:

- **mean.delta** mean value of delta, plotted as a horizontal reference line for instantaneous delat estimations. Default: NULL (no reference line).
- **xlab** character string defining the x-axis label. Default: "Time".
- **ylab** character string defining the y-axis label. Default: biased/unbiased and estimator condition.
- **title.str** character string defining the main title of the plot. Default: NULL (no title).
- **type** character string defining type of plot ala par function. Default: "l" (solid line).

plot plots a summary of the results. Available options are:

- **mean.delta** mean value of delta, plotted as a horizontal reference line for instantaneous delat estimations. Default: NULL (no reference line).
- **xlab** character string defining the x-axis label. Default: "Time".
- **ylab** character string defining the y-axis label. Default: biased/unbiased and estimator condition.
- **title.str** character string defining the main title of the plot. Default: NULL (no title).
- **type** character string defining type of plot ala par function. Default: "l" (solid line).
- **show.key** a logical value. If TRUE, a key of the plot is shown. Default: TRUE.
- **conf.color** color index ala par for the confidence intervals. Default: 16.

print prints the object. Available options are:

- **digits** number of digits to use in displaying numeric values. Default: 5.

print.summary print a summary of the results.

summary create a summary of the results.

See Also

wavFDPBlock, wavFDPTime.

Examples

```
## create a faux dictionary
dictionary <- wavDictionary(wavelet="s8",
    dual=FALSE, decimate=FALSE, n.sample=512,
    attr.x=NULL, n.levels=5,
    boundary="periodic", conv=TRUE,
    filters=wavDaubechies("s8"),
    fast=TRUE, is.complex=FALSE)

## construct a faux wavFDP object
z <- wavFDP(estimator="wlse",
    delta=0.45,
    variance.delta=1.0,
    innovations.variance=1.0,
    delta.range=c(-10.0,10.0),
    dictionary=dictionary,
    levels=c(1,3:4),
    edof.mode=2,
    boundary=list(mode=TRUE,description="unbiased"),
    series=create.signalSeries(fdp045),
    sdf.method="Integration lookup table",
    type="block")

## print the result
print(z)
```

---

wavFDPBand *Mid-octave spectral density function (SDF) estimation*

Description

The wavelet and scaling filters used for wavelet decompositions are nominally associated with approximate bandpass filters. Specifically, at decomposition level j, the wavelet transform coefficients correspond approximately to the normalized frequency range of $[1/2^{j+1}, 1/2^j]$. The square of the wavelet coefficients are used to form the so-called wavelet variance (or wavelet spectrum) which is seen as a regularization of the SDF. Under an assumed FD process, this function estimates the mid-octave SDF values. The estimates are calculated assuming that the wavelet transform filters form perfect (rectangular) passbands. Decomposition levels 1 and 2 are calculated using a second order Taylor series expansion about the mid-octave frequencies while, for levels greater than 2, a small angle approximation ($sin(\pi f) \approx \pi f$) is used to develop a closed form solution which is a function of FD model parameters as well as the mid-octave frequencies.

Usage

```
wavFDPBand(delta=1/4, method="bandpass", scaling=TRUE,
    levels=1:5, n.sample=n.sample <- 2^(max(levels)+1))
```

Arguments

`delta`	the fractional difference parameter. If the scaling band estimates are desired (prompted by setting `n.sample` > 0), then `delta` must be less than 0.5 since the formulae for calculating the scaling band estimates implicitly assume stationarity. Default: `0.4`.
`levels`	a vector containing the decomposition levels. If `n.sample` ≤ 0, then the levels may be given in any order and levels may be skipped. If, however, `n.sample` > 0, then `levels` must contain the values $1, 2, 3, \ldots, J$ where J is the maximum wavelet transform decomposition level. Default: `1:5`.
`method`	a character string denoting the method to be used for estimating the average spectral density values at the center frequency (on a log scale) of each DWT octave. The choices are `"integration"` Numerical integration of the standard FDP spectral density function. `"bandpass"` A small angle approximation to the standard FDP spectral density functions for decomposition levels $j \geq 3$ in combination with a Taylor series approximation for levels $j = 1, 2$. Default: `"bandpass"`.
`n.sample`	the number of samples in the time series. Although no time series is actually passed to the `wavFDPBand` function, the `n.sample` argument is used in estimating the mid-octave SDF value over the band of frequencies which are nominally associated with the scaling filter in a wavelet transform. If `n.sample` > 0, this function will append the estimate of the average SDF value over the scaling band to the wavelet octave estimates. If `n.sample` ≤ 0, only the wavelet octave estimates are returned. Default: `1024`.
`scaling`	a logical flag. If `TRUE`, the mid-octave value of the FDP SDF octave corresponding to the scaling coefficients is also returned. Default: `TRUE`.

Details

Estimates are made for the scaling filter band based upon an implicit assumption that the FD process is stationary ($\delta < 1/2$).

Value

a vector containing the mid-octave SDF estimates for an FD process.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000, 343–54.

See Also

`wavFDPBlock`, `wavFDPTime`, `wavVar`, `wavFDPSDF`.

Examples

```
## calculate the mid-octave SDF values for an FD
## process over various wavelet bands
wavFDPBand(levels=c(1, 3, 5:7), delta=0.45, scaling=FALSE)
```

`wavFDPBlock` *Block-dependent estimation of fractionally differenced (FD) model parameters*

Description

A discrete wavelet transform of the input series is used to calculate block-dependent estimates of the FD parameter, the variance of the FD parameter and the innovations variance. Both a maximum likelihood estimation (MLE) and weighted least squares estimation (WLSE) scheme are supported. If an MLE scheme is chosen, then the DWT is used for its ability to de-correlate long-memory processes. If a WLSE scheme is chosen, then the MODWT is used for its known statistical wavelet variance properties.

Usage

```
wavFDPBlock(x, wavelet="s8", levels=NULL, sdf=NULL,
    boundary=NULL, edof.mode=1,
    estimator="wlse", delta.range=c(-10.0,10.0),
    position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character(), keep.series=FALSE)
```

Arguments

`x` a vector containing a uniformly-sampled real-valued time series.

`boundary` a character string representing the different methods by which boundary wavelet coefficients and scaling coefficients are handled in calculating the FD model parameters. The options for this argument are dependent upon the `estimator` argument.

For the **MLE** case, the `boundary` options are:

`"stationary"` Under a stationary FD process model, boundary wavelet and scaling coefficients are used in estimating the FD model parameters.

`"nonstationary"` A stationary-nonstationary FD model assumes that the governing process may fall into the nonstationary regime and, accordingly, the boundary wavelet coefficients and scaling coefficients are excluded in estimating the FD model parameters.

For the **WLSE** case, the `boundary` options are:

`"biased"` Boundary wavelet coefficients are included in the estimate.

`"unbiased"` Boundary wavelet coefficients are excluded in the estimate.

The scaling coefficients are (always) excluded in weighted least squares estimates of FD model parameters. Default: `"unbiased"`.

`delta.range` a two-element vector containing the search range for the FD parameter. Typically, the range $[-10, 10]$ is suitable for all physical systems. Default: `c(-10 10)`.

`documentation` a character string used to describe the input `data`. Default: `character()`.

`edof.mode` the mode by which the equivalent degrees of freedom are calculated. This argument is limited to 1,2, or 3 and is used only for the WLSE scheme. See `wavEDOF` for details. Default: `1`.

`estimator` a character string denoting the estimation method. Use `"wlse"` for a weighted least squares estimate and `"mle"` for a maximum likelihood estimate. Default: `"wlse"`.

`keep.series` a logical value. If `TRUE`, the original series is preserved in the output object. Default: `FALSE`.

`levels` a vector containing the decomposition levels. The levels may be given in any order but must be positive. Default: `1:J` where J is the maximum wavelet decomposition level at which there exists at least one interior wavelet coefficient.

`position` a `list` containing the arguments `from`, `by` and `to` which describe the position(s) of the input `data`. All position arguments need not be specified as missing members will be filled in by their default values. Default: `list(from=1, by=1, units=character())`.

`sdf` a vector containing a discretized approximation of the process spectral density function (SDF). The coefficients of this argument should correspond exactly with the normalized Fourier frequencies $f = [0, 1/P, 2/P, 3/P, \ldots, (M-1)/P]$, where $P = 2 * (M-1)$ and M is the number of points in the SDF vector. For example, if the sdf vector contains five elements, the corresponding frequencies will be $f = [0, 1/8, 1/4, 3/8, 1/2]$. This argument is used only for the WLSE scheme when calculating EDOF mode 2 estimates. Default: `NULL` (EDOF mode 2 not used).

`title.data` a character string representing the name of the input `data`. Default: `character()`.

`units` a string denoting the units of the time series. Default: `character()` (no units).

`wavelet` a character string denoting the filter type. See `wavDaubechies` for details. Default: `"s8"`.

Details

When `estimator="mle"` and `boundary="stationary"`, the `levels` vector is forced to take on values $1, 2, \ldots, J$ where J is the maximum number of levels in a full DWT. This is done because (in this case) the scaling coefficient and all wavelet coefficients are used to form the FD model parameter estimates.

In using the WLSE scheme it is recommended that only the unbiased estimator be used since the confidence intervals for the biased estimator have not been sufficiently studied.

Value

an object of class `wavFDP`.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000, 340–92.

W. Constantine, D. B. Percival and P. G. Reinhall, *Inertial Range Determination for Aerothermal Turbulence Using Fractionally Differenced Processes and Wavelets*, Physical Review E, 2001, 64(036301), 12 pages.

See Also

wavEDOF, wavFDP, wavFDPTime, wavFDPBand, wavFDPSDF.

Examples

```
## perform a block-averaged MLE of FD parameters
## for an FD(0.45, 1) realization over levels 1
## through 6 using a stationary-nonstationary
## FD model and Daubechies least asymmetric
## 8-tap filters
wavFDPBlock(fdp045, levels=1:6, wavelet="s8", est="mle", boundary="nonstationary")
```

---

wavFDPSDF *Spectral density function for a fractionally differenced process*

Description

Returns the spectral density function (SDF) for a fractionally differenced (FD) process. Given a unit sampling rate, the SDF for an FD proces is

$$\frac{\sigma_\varepsilon^2}{|2\sin(\pi f)|^{2\delta}},$$

where σ_ε^2 is the innovations variance, δ is the FD parameter, and f is the normalized frequency for $|f| < 1/2$.

Usage

```
wavFDPSDF(f, delta=0.45, variance=1, response=NULL)
```

Arguments

f	a numeric value representing normalized frequency where the sampling interval is unity.
delta	the FD parameter. Default: 0.45.
response	a list containing the objects frequency and sqrgain which represent, respectively, a numeric normalized frequency vector corresponding to a wavelet squared gain response at a particular wavelet decomposition level. This argument typically will not be set by the user. Rather, it is used internally by FD process maximum likelihood estimators. Default: NULL.
variance	the FD innovations variance. Default: 1.

Value

the SDF values corresponding to the FD model parameters.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000, 340–92.

See Also

wavFDPBand, wavFDPBlock, wavFDPTime.

Examples

```
## create a normalized frequency vector
f <- seq(from=1e-2, to=1/2, length=100)

## calculate the FDP SDF for delta=0.45 and unit
## innovations variance
S <- wavFDPSDF(f, delta=0.45, variance=1)

## plot the results
plot(f, S,log="xy", xlab="Frequency", ylab="SDF of FDP(0.45, 1)")
```

---

wavFDPTime *Instantaneous estimation of fractionally differenced model parameters*

Description

The MODWT is used to calculate instantaneous estimates of the FD parameter, the variance of the FD parameter and the innovations variance. The user can select between maximum likelihood and least squares estimators. Localized estimates may also be formed by using multiple chi-squared degrees of freedom in estimating the FD model parameters.

Usage

```
wavFDPTime(x, wavelet="s8", levels=NULL,
    biased=FALSE, estimator="mle",
    dof.order=0, delta.range=c(-10.0,10.0),
    position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character(), keep.series=FALSE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
biased	a logical flag used to choose between denoting biased or unbiased estimates. Biased estimates are those which use all available levels in calculating the FD model parameters. Unbiased estimates are calculated with only those wavelet coefficients not subject to circular filter operations, i.e. only the interior wavelet coefficients are used in calculating unbiased estimates. Default: TRUE.
delta.range	a two-element vector containing the search range for the FD parameter. Typically, the range $[-10, 10]$ is suitable for all physical systems. Default: c(-10, 10).
documentation	a character string used to describe the input x. Default: character().
dof.order	the degree of freedom (DOF) order. The number of chi-square DOFs used in estimating the FD parameters is equal to $2 \times$ dof.order $+ 1$ where necessarily dof.order > 0. As the order increases, the estimates will become smoother but less localized in time. Default: 0.
estimator	a character string denoting the estimation method. Use "lse" for least squares estimates and "mle" for maximum likelihood estimates. Default: "lse".
keep.series	a logical value. If TRUE, the original series is preserved in the output object. Default: FALSE.
levels	a vector containing the decomposition levels. The levels may be given in any order but must be positive. Default: 1:J where J is the maximum wavelet decomposition level at which there exists at least one interior wavelet coefficient.
position	a list containing the arguments from, by and to which describe the position(s) of the input x. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
title.data	a character string representing the name of the input x. Default: character().
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".

Value

an object of class wavFDP.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000, 340–92.

W. Constantine, D. B. Percival and P. G. Reinhall, *Inertial Range Determination for Aerothermal Turbulence Using Fractionally Differenced Processes and Wavelets*, Physical Review E, 2001, 64(036301), 12 pages.

See Also

wavFDP, wavFDPBlock, wavFDPBand, wavFDPSDF.

Examples

```
## perform a unbiased instantaneous LSE of FD
## parameters for an FD(0.45, 1) realization
## over levels 1 through 6 using Daubechies
## least asymmetric 8-tap filters. Use a zeroth
## order DOF (equivalent to 1 chi-square DOF)
z <- wavFDPTime(fdp045, levels=1:6, wavelet="s8", est="lse", biased=FALSE)

## display the results
print(z)

## plot the results
plot(z)

## plot the results with the confidence intervals
## centered about the mean (known) value of the
## the FD parameter
plot(z, mean.delta=0.45)
```

`wavGain` *The gain functions for Daubechies wavelet and scaling filters*

Description

Given $\{g\}$ and $\{h\}$ are the impulse responses for the scaling and wavelet filters, respectively, and $G_1(f)$ and $H_1(f)$ are their corresponding gain functions, then the gain functions for decomposition level $j > 1$ are calculated using the recursive algorithm:

$$H_j(f) = H_1(2^{j-1}f)G_{j-1}(f),$$

$$G_j(f) = G_1(2^{j-1}f)G_{j-1}(f).$$

Usage

```
wavGain(wavelet="s8", n.levels=5, n.fft=1024, normalize=TRUE)
```

Arguments

`n.fft`	the number of Fourier coefficients to use in approximating the gain functions. Default: `1024`.
`n.levels`	the number of decomposition levels. Default: `5`.
`normalize`	a boolean value. If `TRUE`, the filters are normalized by dividing each filter coefficient by the $\sqrt{2}$ (used for maximal overlap wavelet transforms). If `FALSE`, no normalization is used. Default: `TRUE`.
`wavelet`	a character string denoting the filter type. See `wavDaubechies` for details. Default: `"s8"`.

Value

an object of class wavGain.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and, Applied Mathematics, 41, 909–96.

See Also

wavDaubechies.

Examples

```
## approximate the gain functions for the
## normalized Daubechies least asymmetric
## 20-tap filters for levels 1,...,5 using a
## 1024 Fourier frequencies
result <- wavGain(wavelet="s20", n.levels=5,
    norm=TRUE)

## plot the results
plot(result)
```

---

wavIndex *Boundary and interior wavelet coefficient identification*

Description

The boundary wavelet and scaling coefficients are those subject to circular filtering operations. This function returns the range of indices which span the interior (or nonboundary) wavelet and scaling coefficients. If approximate zero phase filters are used in the wavelet trsnsform input then the shift factors needed to bring the coefficients to (approximate) zero phase are also returned.

Usage

```
wavIndex(x)
```

Arguments

x	an object of class wavTransform or wavBoundary. The transform type must be a DWT or MODWT.

Value

a list the indices locating the interior and boundary coefficients as well as the the zero phase shift factors need for each level of the transform.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavDWT, wavMODWT.

Examples

```
## calculate the coefficient indices for a MODWT
## of a simple time series
wavIndex(wavMODWT(1:8, wavelet="s8"))
```

---

wavMaxLevel *Maximum decomposition level*

Description

Interior wavelet coefficients are those not subject to circular filter operations. This function calculates the maximum level for a wavelet transform for which there exists at least one interior wavelet coefficient.

Usage

```
wavMaxLevel(n.taps=8, n.sample=1024, xform="modwt")
```

Arguments

n.sample	the number of points in the original time series. Default: 1024.
n.taps	the length of the wavelet filter. Default: 8.
xform	a character string denoting the transform type. Supported types are "dwt", "dwpt", "modwt", and "modwpt". Default: "modwt".

Value

an integer denoting the maximum decomposition level which contains more than one interior wavelet coefficient.

See Also

wavDWT, wavMODWT, wavDWPT.

Examples

```
wavMaxLevel(n.taps=8, n.sample=1024, xform="modwt")
```

wavMODWPT *The maximal overlap discrete wavelet packet transform (MODWPT)*

Description

Given j, n, t are the decomposition level, oscillation index, and time index, respectively, the MODWPT is given by

$$\widetilde{W}_{j,n,t} \equiv \sum_{l=0}^{L-1} \widetilde{u}_{n,l} \widetilde{W}_{j-1,\lfloor n/2 \rfloor, t-2^{j-1}\ l \bmod N}$$

The variable L is the length of the filters defined by

$$\widetilde{u}_{n,l} \equiv \begin{cases} \widetilde{g}_l/\sqrt{2}, & \text{if } n \bmod 4 = 0 \text{ or } 3; \\ \widetilde{h}_l/\sqrt{2}, & \text{if } n \bmod 4 = 1 \text{ or } 2, \end{cases}$$

where g and h are the scaling filter and wavelet filter, respectively. By definition, $\widetilde{W}_{0,0,t} \equiv X_t$ where $\{X_t\}$ is the original time series.

Usage

```
wavMODWPT(x, wavelet="s8", n.levels=ilogb(length(x), base=2),
    position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character())
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
documentation	a character string used to describe the input data. Default: character().
n.levels	the number of decomposition levels. Default: as.integer(floor(logb(length(x),base=2))).
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
title.data	a character string representing the name of the input data. Default: character().
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".

Value

an object of class wavTransform.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

reconstruct, wavMRD, wavMODWT, wavDWT, wavDWPT, wavDaubechies, wavShift, wavZeroPhase.

Examples

```
## calculate the MODWPT of sunspots series out to
## 3 levels using Daubechies least asymmetric
## 8-tap filter set
z <- wavMODWPT(sunspots, wavelet="s8", n.levels=3)

## plot the transform
plot(z)

## summarize the transform
summary(z)
```

---

wavMODWT *The maximal overlap discrete wavelet transform (MODWT)*

Description

Let j, t be the decomposition level, and time index, respectively, and $s_{0,t} = X_{t=0}^{N-1}$ where X_t is a real-valued uniformly-sampled time series. The j^{th} level MODWT wavelet coefficients $\tilde{d}_{j,t}$ and scaling coefficients $\tilde{s}_{j,t}$ are defined as $\tilde{d}_{j,t} \equiv \sum_{l=0}^{L-1} \widetilde{h}_l \tilde{s}_{j-1,t-2^{j-1}\ l \bmod N,}$ and $\tilde{s}_{j,t} \equiv \sum_{l=0}^{L-1} \widetilde{g}_l \tilde{s}_{j-1,t-2^{j-1}\ l \bmod N}$ The variable L is the length of both the scaling filter (g) and wavelet filter (h). The $\tilde{d}_{j,t}$ and $\tilde{s}_{j,t}$ are the wavelet and scaling coefficients, respectively, at decomposition level j and time index t. The MODWT is a collection of all wavelet coefficients and the scaling coefficients at the last level: $\tilde{\mathbf{d}}_1, \tilde{\mathbf{d}}_2, \ldots, \tilde{\mathbf{d}}_\mathbf{J}, \tilde{\mathbf{s}}_\mathbf{J}$ where $\tilde{\mathbf{d}}_\mathbf{j}$ and $\tilde{\mathbf{s}}_\mathbf{j}$ denote a collection of wavelet and scaling coefficients, respectively, at level j.

Usage

```
wavMODWT(x, wavelet="s8", n.levels=ilogb(length(x), base=2),
    position=list(from=1,by=1,units=character()), units=character(),
    title.data=character(), documentation=character(), keep.series=FALSE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
documentation	a character string used to describe the input data. Default: character().
keep.series	a logical value. If TRUE, the original series is preserved in the output object. Default: FALSE.
n.levels	the number of decomposition levels. Default: as.integer(floor(logb(length(x),base=2))).
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).

title.data	a character string representing the name of the input data. Default: character().
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".

Details

The MODWT is a non-decimated form of the discrete wavelet transform (DWT) having many advantages over the DWT including the ability to handle arbitrary length sequences and shift invariance (while the wavDWT function can handle arbitrary length sequences, it does so by means of an ad hoc storage sytem for odd length scaling coefficient crystals. The MODWT needs no such scheme and is more robust in this respect). The cost of the MODWT is in its redundancy. For an N point input sequence, there are N wavelet coefficients per scale. However, the number of multiplication operations is $O(N \log_2(N))$ which is the same as the fast Fourier transform, and is acceptably fast for most situations.

Value

an object of class wavTransform.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

reconstruct, wavDaubechies, wavDWT, wavMODWPT, wavDictionary, wavIndex, wavTitle, wavBoundary.

Examples

```
## calculate the MODWT of linear chirp
linchirp <- make.signal("linchirp", n=1024)
result   <- wavMODWT(linchirp, wavelet="s8", n.levels=5, keep.series=TRUE)

## plot the transform shifted for approximate zero
## phase alignment
plot(wavShift(result))

## plot summary
eda.plot(result)

## summarize the transform
summary(result)
```

wavMRD *Calculate the detail sequences for wavelet transform crystals*

Description

Let $\mathbf{W}_{j,n}$ be a discrete wavelet packet crystal where j is the decomposition level and n is the oscillation index. The detail sequence $\mathcal{D}_{j,n}$ is formed (essentially) by reconstructing the transform after zeroing out all other crystals except $\mathbf{W}_{j,n}$. The wavMRD function calculates the details for a DWT and MODWT in an optimized way.

Usage

```
wavMRD(x, level=NULL, osc=NULL)
```

Arguments

x	an object of class wavTransform.
level	an integer (vector) containing the decomposition level(s) corresponding to the crystal(s) to be decomposed. Default: If the input is of class wavTransform, then the default is to return the details at all levels of the transform, i.e., a full multiresolution decomposition.
osc	an integer (vector) containing the oscillation indices corresponding to the crystal(s) to be decomposed. Default: the default values are coordinated with that of the level argument.

Value

an object of class WaveletMRD.

S3 METHODS

[single level data access.

Usage: x["D2"] or x["S4"]

Access a subset of wavelet transform details/smooth.

[<- single level data replacement method.

Usage: x["D2"] <- 1:4

Replace an entire wavelet transform details/smooth with explicitly defined coefficients.

[[double level data access.

Usage: x[["D2"]] or x[[2]]

Returns a vector of wavelet transform detail/smooth coefficients corresponding to the specified crystal.

as.matrix transforms the list of wavelet transform details/smooth coefficients into a single-column matrix whose row names identify the transform coefficient, e.g., D4(3) is the third coefficient of the D4 detail.

Usage: as.matrix(x)

boxplot plots a boxplot for each wavelet transform detail/smooth.

Usage: boxplot(x)

crystal.names return the crystal names for each wavelet transform detail/smooth.

Usage: crystal.names(x)

plot plot a stack plot of the discrete wavelet transform details/smooth. Usage: plot(x, n.top=15, vgap=.05, col=1, show.sum=TRUE, add=FALSE, ...)

x A `wavMRD` object.

n.top An integer defining the (maximum) number of top-most energetic crystals to plot. Default: `15`.

sort.energy A logical value. If `TRUE`, the crystals are sorted in the display from the most energetic (top) to the least energetic (bottom) of the specified `n.top` crystals. Default: `FALSE`.

vgap A numeric scalar defining the vertical gap between plots expressed as a fraction of the maximum value of the details/smooth that are plotted. Default: `0.05`.

col An integer or vecto rof integers deining the color index to apply to each detail/smooth line plot. Default: `1`.

show.sum A logical value. If `TRUE`, a plot of the sum of all details/smooth is also plotted. Default: `TRUE`.

add A logical value. If `TRUE`, the plot is added to the current plot layout without a frame ejection. Default: `FALSE`.

... Additional arguments to be sent to the plot routine.

print print the wavelet transform details/smooth object. Usage: print(x)

reconstruct reconstruct/synthesize/invert the wavelet transform details/smooth. Usage: reconstruct(x)

If the transform coefficients were not modified, the original time series will be returned (+/- some numerical noise).

summary provide a statistical summary of the wavelet transform details/smooth object. Usage: z <- summary(x); print(z)

wavStackPlot stack plot of the wavelet transform details/smooth. Usage: wavStackPlot(x)

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

`wavMRDSum`, `reconstruct`, `wavDWT`, `wavMODWT`.

Examples

```
## calculate various wavelet transforms of the
## first difference of a linear chirp sequence
x <- make.signal("linchirp", n=1024)
x.dwt <- wavDWT(x, n.levels = 3)
x.modwt <- wavMODWT(x, n.levels = 3)
```

```
## calculate the wavelet details for all crystals
## of the DWT and MODWT
wavMRD(x.dwt)
wavMRD(x.modwt)

## plot the wavelet details for levels 1 and 3 of
## the MODWT
plot(wavMRD(x.modwt, level = c(1,3)))

## plot the wavelet details for all levels of the
## MODWT of a linear chirp.
plot(wavMRD(x.modwt))
```

---

wavMRDSum *Partial summation of a multiresolution decomposition*

Description

Forms a multiresolution decomposition (MRD) by taking a specified discrete wavelet transform of the input series and subsequently inverting each level of the transform back to the "time" domain. The resulting components of the MRD form an octave-band decomposition of the original series, and can be summed together to reconstruct the original series. Summing only a subset of these components can be viewed as a denoising operation if the "noisy" components are excluded from the summation.

Usage

```
wavMRDSum(x, wavelet="s8",
    levels=1, xform="modwt", reflect=TRUE,
    keep.smooth=TRUE, keep.details=TRUE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
keep.details	a logical value. If TRUE, the details corresponding to the specified levels are included in the partial summation over the MRD components. The user also has the choice to exclude the smooth in the summation via the keep.smooth option, but one of keep.details and keep.smooth must be TRUE. Default: TRUE.
keep.smooth	a logical value. If TRUE, the smooth at the last decomposition level is added to the partial summation over specified details. The smooth typically contains low-frequency trends present in a series, so removing the smooth (keep.smooth=FALSE) will result in removing the trend in such cases. The user also has the choice to exclude the details in the summation via the keep.details option, but one of keep.details and keep.smooth must be TRUE. Default: TRUE.

`levels` an integer vector of integers denoting the MRD detail(s) to sum over in forming a denoised approximation to the orginal series (the summation is performed across scale and nto across time). All values must be positive integers, and cannot exceed `floor(logb(length(x),2))` if `reflect=FALSE` and, if `reflect=TRUE`, cannot exceed `floor(logb((length(x)-1)/(L-1) + 1, b=2))` where L is the length of the wavelet filter. Use the `keep.smooth` option to also include the last level's smooth in the summation. Default: `1`.

`reflect` a logical value. If `TRUE`, the last $L_J = (2^{\text{n.level}} - 1)(L-1) + 1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level `n.level`, where L is the length of the wavelet (or scaling) filter. A similar operation is performed at the beginning of the series. After synthesis and (partial) summation of the resulting details and smooth, the middle N points of the result are returned, where N is the length of the original time series. Default: `TRUE`.

`wavelet` a character string denoting the filter type. See `wavDaubechies` for details. Default: `"s8"`.

`xform` a character string denoting the wavelet transform type. Choices are `"dwt"` and `"modwt"` for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*`n.level`. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: `"modwt"`.

Details

Performs a level J decimated or undecimated discrete wavelet transform on the input series and inverts the transform at each level separately to produce details $D_1, \ldots, D_J$ and smooth S_J. The decomposition is additive such that the original series X may be reconstructed ala $X = S_J + \sum_{j=1}^{J} D_j$. As the effective wavelet filters at level j are nominally associated with approximate band pass filters, the details D_j correspond approximately to normalized frequencies on the interval $[1/2^{j+1}, 1/2^j]$, while the content of the smooth S_J corresponds approximately to normalized frequencies $[0, 1/2^{J+1}]$. The collection of details and smooth form a multiresolution decomposition (MRD).

With the intent of removing unwanted noise events, a summation over a subset of MRD components may be calculated yielding a smooth approximation to the original series. For example, summing all MRD components beyond D_1 is tantamount to a low-pass filtering of the original series (whether or not this is a relevant and sufficient noise removal technique is left to the discretion of the practitioner). This function allows the user to specify the decomposition levels they wish to sum over in order to form a multiresolution approximation. The inclusion of the last level's smooth in the summation is controlled by the optional `keep.smooth` argument.

The user may also select either a decimated wavelet transform (DWT) or an undecimated wavelet

transform (MODWT). However, we recommend that the user stick with the MODWT for the following reasons:

Translation invariance Unlike the DWT, the MODWT is translation invariant, meaning that a (circular) shift of the input series will result in a corresponding (circular) shift of the transform coefficients.

Smoothness The MODWT coefficients are a result of *cycle-spinning*, where averages are taken over all unique DWTs resulting from various circular shifts of the original series. The resulting MODWT MRD is relatively more smooth than the corresponding DWT MRD.

Zero phase aligment Unlike the DWT MRD, the MODWT MRD produces components that are associated with **exactly** zero phase filter operations such that events (such as peaks) in the details and smooth line up exactly with those of the original series.

Computational speed The DWT is faster than the MODWT, but the MODWT is still quite fast, requiring multiplication and summation operations on the same order as the popular Fast Fourier Transform.

Value

a vector containing the denoised series.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

T.W. Randolph and Y. Yasui, *Multiscale Processing of Mass Spectrometry Data*, *Biometrics*, **62**:589–97, 2006.

See Also

wavDaubechies, wavDWT, wavMODWT, wavMRD.

Examples

```
## create a MODWT-based MRD of the sunspots series
## and sum over decomposition levels 4 to 6
x <- as.vector(sunspots)
z1 <- wavMRDSum(x, levels=4:6)
ifultools::stackPlot(x=as.vector(time(sunspots)),
    y=list(sunspots=x, "D4+D5+D6"=z1),
    ylim=range(x,z1))
```

`wavPacketBasis` *Extract wavelet packet basis from a DWPT*

Description

Returns the DWPT crystals (in a list) corresponding to the basis specified by the indices vector. The indices are mapped as follows:

0 original series

1:2 $\{W_{1,0}, W_{1,1}\}$, i.e., all level 1 crystals

3:6 $\{W_{2,0}, \ldots, W_{2,3}\}$, i.e., all level 2 crystals

and so on. If the indices do not form a basis, an error is issued.

Usage

```
wavPacketBasis(x, indices=0)
```

Arguments

`x`	an object of class `wavTransform` associated with the output of the `wavDWPT` function.
`indices`	an integer vector. Each integer denotes a particular crystal of the DWPT to extract. The set of crystals shoudl form a basis, i.e., the collective frequency ranges associated with the set of crystals should span normalized frequencies [0, 1/2]. The indices for each DWPT level and the corresponding (ideal) normalized frequency ranges are listed in the table below:

0 Frequency range: [0, 1/2], associated with crystal $W_{0,0}$ (the original series).

1,2 Frequency range: [0,1/4],[1/4, 1/2], associated with crystals $W_{1,0}$, $W_{1,1}$, respectively.

3,4,5,6 Frequency range: [0,1/8],[1/8, 1/4],[1/4,3/8],[3/8, 1/2], associated with crystals $W_{2,0}$,$W_{2,1}$,$W_{2,2}$,$W_{2,3}$, respectively.

and so forth.

See Also

`wavDWPT`, `wavBestBasis`.

Examples

```
## calculate a 3-level DWPT of the sunspots series
W <- wavDWPT(sunspots, n.level=3)

## extract the level 1 basis
W12 <- wavPacketBasis(W, 1:2)
```

```
## obtain the names of the crystals that were
## extracted: "w1.0" "w1.1"
names(W12$data)

## extract basis corresponding to crystal set:
## "w2.0" "w2.1" "w1.1". This set comprises a
## split-level basis
Wsplit <- wavPacketBasis(W, c(3,4,2))
names(Wsplit$data)
```

`wavPacketIndices` *Wavelet packet node indices*

Description

Converts flattened wavelet packet node indices to corresponding *level* and *oscillation* indices.

Usage

```
wavPacketIndices(x, check.basis=TRUE)
```

Arguments

`x`	a vector of flattened wavelet packet node indices.
`check.basis`	a logical value. If `TRUE`, the set of specified indices is checked to ensure that the corresponding wavelet packet nodes form a legitimate basis by ensuring that (i) the union of all frequency ranges corresponding to the packet crystals span the normalized frequencies [0,1/2] and (ii) the normalized frequency ranges for all nodes do not overlap. Default: `TRUE`.

Details

In general, wavelet packet crystals can be arranged in the so-called *natural order* ala $W_{0,0}, W_{1,0}, W_{1,1}, W_{2,0}, W_{2,1}, W_{2,2}, W_{2,}$ where J is the number of decomposition levels and $N_J = 2^J - 1$. By definition, $W_{0,0}$ is the original time series. A given crystal is identified in the $W_{j,n}$ form above or by a flattened index. For example, the DWPT crystal in level 2 at oscillation index 1 is identified either by j,n=2,1 or by its flattened index 4 (with zero based indexing, 4 represents the fifth crystal of the wavelet packet transform in natural order). This function converts such flattened wavelet packet indices to the $W_{j,n}$ form.

Value

a list of `flat`, `level`, and `osc` vectors containing the flattened, decomposition level, and oscillation indices, respectively, corresponding to the input.

See Also

`wavDWPT`.

Examples

```
## specify a basis formed by the flattened indices
## corresponding to the wavelet packet nodes
## W(2,0:1) and W(3,4:7), but submit them in
## arbitrary order
wavPacketIndices(c(14,11,12,13,4,3), check.basis=TRUE)
```

`wavShift` *Shifts wavelet transform coefficients for approximate zero phase alignment*

Description

If Daubechies symmlet or coiflet filters are used in forming a DWT or MODWT (ala `wavDWT` or `wavMODWT`, respectively), then the transform coefficients can be circularly rotated so that they are approximately aligned (in time) with events of the original time series. An appropriate shift of the coefficients (generated by approximate linear phase filter operations) is approximately equivalent to using zero phase filters in the wavelet transform.

Usage

```
wavShift(x)
```

Arguments

`x`	an object of class `wavTransform` or `wavBoundary`.

Details

Only relevant for transforms calculated using Daubechies coiflet and symmlet filters. A second application of `wavShift` to the same input object will result in the original input object, i.e. without any imposed shift in the transform coefficients.

Value

an object of the same class as the input with the transform coefficients adjusted to approximate zero phase filtering operations.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

I. Daubechies, *Orthonormal Bases of Compactly Supported Wavelets*, Communications on Pure and, Applied Mathematics, 41, 909–96.

See Also

`wavZeroPhase`, `wavDWT`, `wavMODWT`.

Examples

```
## plot the zero phase shifted MODWT of a linear
## chirp sequence
linchirp <- make.signal("linchirp", n=1024)
plot(wavShift(wavMODWT(linchirp, wavelet="s8",
    n.levels=4, keep.series=TRUE)))
```

---

wavShrink *Nonlinear denoising via wavelet shrinkage*

Description

Performs a decimated or undecimated discrete wavelet transform on the input series and "shrinks" (decreases the amplitude towards zero) the wavelet coefficients based on a calculated noise threshold and specified shrinkage function. The resulting shrunken set of wavelet transform coefficients is inverted in a synthesis operation, resulting in a denoised version of the original series.

Usage

```
wavShrink(x, wavelet="s8",
    n.level=ilogb(length(x), base=2),
    shrink.fun="hard", thresh.fun="universal", threshold=NULL,
    thresh.scale=1, xform="modwt", noise.variance=-1.0,
    reflect=TRUE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
n.level	the number of decomposition levels, limited to floor(logb(length(x),2)). Default: floor(logb(length(x),2)).
noise.variance	a numeric scalar representing (an estimate of) the additive Gaussian white noise variance. If unknown, setting this value to 0.0 (or less) will prompt the function to automatically estimate the noise variance based on the median absolute deviation (MAD) of the scale one wavelet coefficients. Default: -1.
reflect	a logical value. If TRUE, the last $L_J = (2^{\text{n.level}} - 1)(L-1) + 1$ coefficients of the series are reflected (reversed and appended to the end of the series) in order to attenuate the adverse effect of circular filter operations on wavelet transform coefficients for series whose endpoint levels are (highly) mismatched. The variable L_J represents the effective filter length at decomposition level n.level, where L is the length of the wavelet (or scaling) filter. After waveshrinking and reconstructing, the first N points of the result are returned, where N is the length of the original time series. Default: TRUE.
shrink.fun	a character string denoting the shrinkage function. Choices are "hard", "soft", and "mid". Default: "hard".
thresh.fun	a character string denoting the threshold function to use in calculating the waveshrink thresholds.

character string Choices are `"universal"`, `"minimax"`, and `"adaptive"`.

numeric values Either a single threshold value or a vector of values containing `n.levels` thresholds (one threshold per decomposition level).

Note: if `xform == "modwt"`, then only the `"universal"` threshold function is (currently) supported. Default: `"universal"`.

`thresh.scale` a positive valued numeric scalar which is used to amplify or attenuate the threshold values at each decomposition level. The use of this argument signifies a departure from a model driven estimate of the thresholds and can be used to tweak the levels to obtain a smoother or rougher result. Default: 1.

`threshold` explicit setting of the wavelet shrinkage thresholds, one for each level of the decomposition. If a single threshold is given, it is replicated appropriately and (if the chosen transform is additionally a MODWT then) these thresholds are normalized by dividing the threshold at level j by $2^(j-1)/2)$. If the number of thresholds is equal to the number of decomposition levels, the thresholds are unaltered prior to use. Default: `NULL` (thresholds are calculated based on the model defined by the `thresh.fun` and `thresh.scale` input arguments).

`wavelet` a character string denoting the filter type. See `wavDaubechies` for details. Default: `"s8"`.

`xform` a character string denoting the wavelet transform type. Choices are `"dwt"` and `"modwt"` for the discrete wavelet transform (DWT) and maximal overlap DWT (MODWT), respectively. The DWT is a decimated transform where (at each level) the number of transform coefficients is halved. Given N is the length of the original time series, the total number of DWT transform coefficients is N. The MODWT is a non-decimated transform where the number of coefficients at each level is N and the total number of transform coefficients is N*`n.level`. Unlike the DWT, the MODWT is shift-invariant and is seen as a weighted average of all possible non-redundant shifts of the DWT. See the references for details. Default: `"modwt"`.

Details

Assume that an appropriate model for our time series is $\mathbf{X} = \mathbf{D} + \epsilon$ where $\mathbf{D}$ represents an unknown deterministic signal of interest and ϵ is some undesired stochastic noise that is independent and identically distributed and has a process mean of zero. Waveshrink seeks to eliminate the noise component ϵ of $\mathbf{X}$ in hopes of obtaining (a close approximation to) $\mathbf{D}$. The basic algorithm works as follows:

1 Calculate the DWT of X.

2 Shrink (reduce towards zero) the wavelet coefficients based on a selected thresholding scheme.

3 Invert the DWT.

This function support different shrinkage methods and threshold estimation schemes. Let W represent an arbitrary DWT coefficient and $W^{(\mathrm{t})}$ the correpsonding thresholded coefficient using a threshold of δ. The supported shrinkage methods are

hard thresholding

$$W^{(\mathrm{t})} = \begin{cases} 0, & \text{if } |W| \leq \delta; \\ W, & \text{otherwise} \end{cases}$$

soft thresholding

$$W^{(\mathrm{t})} = \mathrm{sign}(W)\ f(|W| - \delta)$$

where

$$\mathrm{sign}(W) \equiv \begin{cases} +1, & \text{if } W > 0; \\ 0, & \text{if } W = 0; \\ -1, & \text{if } W < 0. \end{cases}$$

and

$$f(x) \equiv \begin{cases} x, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0. \end{cases}$$

mid thresholding

$$W^{(\mathrm{t})} = \mathrm{sign}(W)\ g(|W| - \delta)$$

where

$$g(|W| - \delta) \equiv \begin{cases} 2f(|W| - \delta), & \text{if } |W| < 2\delta; \\ |W|, & \text{otherwise.} \end{cases}$$

Hard thresholding reduces to zero all coefficients that do not exceed the threshold. *Soft* thresholding pushes toward zero any coefficient whose magnitude exceeds the threshold, and zeros the coefficient otherwise. *Mid* thresholding represents a compromise between hard and soft thresholding such that coefficients whose magnitude exceeds twice the threshold are not adjusted, those between the threshold and twice the trhreshold are shrunk, and those below the threshodl are zeroed.

The threshold is selected based on a model of the noise. The supported techniques for estimating the noise threshold are

universal $\sqrt{2\sigma_{\varepsilon}^2 \log(N)}$ where is the number of samples in the time series. As the noise variance is typically unknown, it is estimated based on the median absolute deviation of the absolute value of the scale one wavelet coefficients (and scaled by dividing the result by 0.6745 so that if the series were Gaussian white noise, the correct variance would be returned). The universal threshold is defined so that if the original time series was solely comprised of Gaussian noise, then all the wavelet coefficients would be (correctly) set to zero using a hard thresholding scheme. Inasmuch, the universal threshold results in highly smoothed output.

minimax These thresholds are used with soft and hard thresholding, and are precomputed based on a minimization of a theoretical upperbound on the asymptotic risk. The minimax thresholds are always smaller than the universal threshold for a given sample size, thus resulting in relatively less smoothing.

adaptive These are scale-adaptive thresholds, based on the minimization of Stein's Unbiased Risk Estimator for each level of the DWT. This method is only available with soft shrinkage. As a caveat, this threshold can produce poor results if the data is too sparse (see the references for details).

Finally, the user has the choice of using either a decimated (standard) form of the discrete wavelet transform (DWT) or an undecimated version of the DWT (known as the Maximal Overlap DWT (MODWT)). Unlike the DWT, the MODWT is a (circular) shift-invariant transform so that a circular shift in the original time series produces an equivalent shift of the MODWT coefficients. In addition, the MODWT can be interpreted as a *cycle-spun* version of the DWT, which is achieved by averaging over all non-redundant DWTs of shifted versions of the original series. The z is a smoother version of the DWT at the cost of an increase in computational complexity (for an N-point series, the DWT requires $O(N)$ multiplications while the MODWT requires $O(N \log_2 N$ multiplications.

Value

vector containing the denoised series.

References

Donoho, D. and Johnstone, I. *Ideal Spatial Adaptation by Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

Donoho, D. and Johnstone, I. *Adapting to Unknown Smoothness via Wavelet Shrinkage*. Technical report, Department of Statistics, Stanford University, 1992.

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavDaubechies, wavDWT, wavMODWT.

Examples

```
## MODWT waveshrinking using various thresh.scale
## values on sunspots series
x <- as.vector(sunspots)
tt <- as.numeric(time(sunspots))
thresh <- seq(0.5,2,length=4)
ws <- lapply(thresh, function(k,x)
    wavShrink(x, wavelet="s8",
       shrink.fun="hard", thresh.fun="universal",
       thresh.scale=k, xform="modwt"), x=x)

ifultools::stackPlot(x=tt, y=data.frame(x, ws),
    ylab=c("sunspots",thresh),
    xlab="Time")

## DWT waveshrinking using various threshold
## functions
threshfuns <- c("universal", "minimax", "adaptive")
ws <- lapply(threshfuns, function(k,x)
    wavShrink(x, wavelet="s8",
       thresh.fun=k, xform="dwt"), x=x)

ifultools::stackPlot(x=tt, y=data.frame(x, ws),
ylab=c("original", threshfuns),
xlab="Normalized Time")
```

---

wavSortCrystals *Sort wavelet transform crystal names*

Description

Sorts the crystal names for a discrete wavelet transform by level then by corresponding frequency content, from low to high.

Usage

```
wavSortCrystals(x, reverse=FALSE)
```

Arguments

x	a vector of character strings containing the names of the wavelet transform crystals.
reverse	a logical value. If TRUE, the order of the sorted names is reversed. Default: FALSE.

Value

a vector of character strings containing the sorted crystal names.

See Also

wavDWT, wavMODWT.

Examples

```
W <- wavMODWT(1:100)
wavSortCrystals(names(W$data), reverse=TRUE)
```

---

wavStackPlot *Generic function for generating stacked plots*

Description

Stackplots stack each plot on top of one another without having to manually divide the ploting region via par("mfrow") or other.

Usage

```
wavStackPlot(x, ...)
```

Arguments

x	any object. Missing values (NAs) are allowed.
...	optional arguments to be passed directly to the inherited function without alteration and with the original names preserved.

Note

An extended data analysis plot is shown.

See Also

wavMRD, wavTransform, eda.plot, crystal.names.

Examples

```
methods(wavStackPlot)
```

---

wavStackPlot.default *Plot wavelet transform crystals*

Description

Forms a stack plot of wavelet transform crystals.

Usage

```
## Default S3 method:
wavStackPlot(x, x.axis=TRUE, y.axis=TRUE, type="l", plot=TRUE,
    bars=FALSE, vgap=.05, grid=FALSE, times=time(x[[1]]),
    grid.lty="dashed", same.scale=NULL,
    zerocenter=FALSE, zeroline=FALSE, col=rep(1,n),
    complex.math="mod", cex.main=0.7, cex.axis=0.7, ...)
```

Arguments

x	a named list of wavelet transform crystals.
...	additional arguments sent directly to various internal plot functions.
bars	a logical value. If TRUE, a vertical bars are produced. Default: FALSE.
cex.axis	axis label character expansion factor ala par function. Default: 0.7.
cex.main	main title character expansion factor ala par function. Default: 0.7.
col	color indices for each crystal ala par function. Default: rep(1,n).
complex.math	math function to perform on each crystal prior to display. Default: "mod".
grid	a logical value. If TRUE, grid lines are produced. Default: FALSE.
grid.lty	grid line type ala par function. Default: "dashed".
plot	a logical value. If TRUE, a plot is produced. Default: TRUE.
same.scale	a logical value. If TRUE, the crystal coefficients are scaled. Default: TRUE.
times	a numeric vector of x-axis times to be displayed. Default: time(x[[1]]).
type	type of plot ala par function. Default: "l".
vgap	vertical gap factor used to separate stacked crystals. Default: 0.05.

x.axis	a logical value defining the visibility of the x-axis. Default: TRUE.
y.axis	a logical value defining the visibility of the y-axis. Default: TRUE.
zerocenter	a logical value. If TRUE, crystals are centered about zero. Default: FALSE.
zeroline	a logical value. If TRUE, a y=0 line is drawn for reference. Default: FALSE.

Details

Used by various wavelet functions to plot wavelet transform crystals. In general, the user should not call this function directly and, rather, shoudl rely on the plot methods of wavTransform objects.

See Also

wavTransform.

Examples

```
wavStackPlot.default(wavMODWT(sunspots)$data)
```

---

wavTitle *Extract the name of the data used to generate objects of various wavelet classes*

Description

Wavelet functions store the original name of the data used to create the output in various locations within the output object. This function provides a means by which the user can directly access data name.

Usage

```
wavTitle(x, default="x")
```

Arguments

x	an object of class mra, mrd, decompose, wavTransform or signalSeries.
default	a default chacter string to use if no valid time series name is found. Default: "x".

Value

a character string vector containing the result.

See Also

wavDWT, wavMODWT.

Examples

```
wavTitle(wavDWT(1:8))
```

wavTransform *Constructor function for objects of class wavTransform*

Description

Packs input information regarding a discrete wavelet transform into a list.

Usage

```
wavTransform(data, series, n.levels, dictionary, shifted, xform)
```

Arguments

data	a list of vectors containing discrete wavelet transform coefficients.
series	a numeric vector or signalSeries object representing the input series.
n.levels	an integer denoting the number of decomposition levels.
dictionary	an object of class wavDictionary representing the wavelet dictionary of the transform.
shifted	a logical value. If TRUE, it signifies that the transform coefficients have already been shifted for approximate zero phase alignment.
xform	a character string denoting the type of wavelet transform that has been performed. Typical values are "modwt" or "dwt".

Details

Used internally by the wavMODWT and wavDWT functions to package the transform contents into a list.

Value

an object of class wavTransform.

S3 METHODS

[single level data access.

Usage: x["d2"] or x[2]

Access a subset of wavelet transform crystals.

[<- single level data replacement method.

Usage: x["d2"] <- 1:4

Replace an entire crystal with explicitly defined coefficients.

[[double level data access.

Usage: x[["d2"]] or x[[2]]

Returns a vector of transform coefficients corresponding to the specified crystal.

as.matrix transforms the list of wavelet transform coefficients into a single-column matrix whose row names identify the transform coefficient, e.g., d4(3) is the third coefficient of the d4 crystal (fourth level wavelet coefficients).

Usage: as.matrix(x)

boxplot plots a boxplot for each crystal in the discrete wavelet transform.

Usage: boxplot(x)

eda.plot extended data analysis plot. A 2x2 grid of plots containing a stack plot, boxplot, and two energy related plots are shown.

Usage: eda.plot(x)

plot plot a discrete wavelet transform. Usage: plot(x, type="h", plot.bar=TRUE, plot.pie=TRUE, add=FALSE, cex.main=ifelse1(is.R(),1,0.7), ...)

x A `wavTransform` object.

type A character denoting the type of line plot to produce in a stack plot of the wavelet transform coefficients (see primary plot() function for details). If type is the character string `"energy"` an energy plot (bar or pie chart) is produced. Default: `"h"`.

plot.bar A logical value. If `TRUE` and `type="energy"`, a bar plot of crystal energy is plotted. Default: `TRUE`.

plot.pie A logical value. If `TRUE` and `type="energy"`, a pie chart of crystal energy is plotted. Default: `FALSE`.

add A logical value. If `TRUE`, the plot is added to the current plot layout without a frame ejection. Default: `FALSE`.

... Additional arguments to be sent to the stackplot routine.

print print the wavelet transform object. Usage: print(x)

reconstruct reconstruct/synthesize/invert the wavelet transform. Usage: reconstruct(x)

If the transform coefficients were not modified, the original time series will be returned (+/- some numerical noise).

summary provide a statistical summary of the wavelet transform object. Usage: z <- summary(x); print(z)

wavStackPlot stack plot of the wavelet transform. Usage: wavStackPlot(x)

See Also

`wavDWT`, `wavMODWT`, `wavDWPT`, `wavMODWPT`, `wavBoundary`, `wavSortCrystals`, `wavPacketIndices`, `wavShrink`.

Examples

```
## calculate a MODWT of the sunpots series and
## verify the class
W <- wavMODWT(sunspots)
print(class(W))

## summarize the object
summary(W)

## reconstruct the MODWT of the sunspots series
```

```
## and compare to the original
sunup <- reconstruct(W)
splus2R::vecnorm(sunup - sunspots)
```

---

wavVar *Discrete wavelet variance estimation*

Description

The discrete wavelet variance is a useful alternative to the spectral density function (SDF) and is seen as an octave-band regularization of the SDF. The wavelet variance decomposes the variance of certain stochastic processes on a scale-by-scale basis, and thus, is very appealing to the analyst studying physical phenomena which fluctuate both within and across a wide range of scale.

By definition, the wavelet variance involves an averaged energy summation of MODWT wavelet coefficients. While DWT wavelet coefficients can also be used, the statistical properties are inferior to those of the MODWT wavelet variance. See the references for more details.

The MODWT Wavelet Variance

Let N be the the number of samples in a time series $\{X_t\}$, L be the length of the wavelet filter, $L_j \equiv (2^j - 1)(L - 1) + 1$ be the equivalent filter width at level \$j\$ in a MODWT, and $\tau_j \equiv 2^{j-1}$ be the scale of the data at level j for $j = 1, \ldots, J$. Then the unbiased wavelet variance is defined as

$$\hat{\nu}_X^2(\tau_j) \equiv \frac{1}{M_j} \sum_{t=L_j-1}^{N-1} \tilde{d}_{j,t}^2,$$

where $\tilde{d}_{j,t}$ are the MODWT coefficients at level j and time t, and $M_j \equiv N - L_j + 1$. The unbiased wavelet variance estimator avoids so-called boundary coefficients which are those coefficients subject to circular filter operations in a discrete wavelet transform. The biased estimator is defined as

$$\widetilde{\nu}_X^2(\tau_j) \equiv \frac{1}{N} \sum_{t=0}^{N-1} \tilde{d}_{j,t}^2,$$

and includes the effects of the boundary coefficients.

The DWT Wavelet Variance

The DWT can also be used to calculate wavelet variance estimates, but is not preferred over the MODWT due to its poor statistical properties. Let $N_j \equiv \lfloor N/2^j \rfloor$ be the number of DWT wavelet coefficients at level j, and $L'_j \equiv \lceil (L-2)(1-2^{-j}) \rceil$ be the number of DWT boundary coefficients at level j (assuming $N_j > L'j$). Then the DWT version of the unbiased wavelet variance is defined as

$$\hat{\hat{\nu}}_X^2(\tau_j) \equiv \frac{1}{N - 2^j L'_j} \sum_{t=L'_j-1}^{N_j-1} d_{j,t}^2,$$

where $d_{j,t}$ are the DWT coefficients at level j and time t. Similarly, the DWT version of the biased wavelet variance is defined as

$$\widetilde{\widetilde{\nu}}_X^2(\tau_j) \equiv \frac{1}{N} \sum_{t=0}^{N_j-1} d_{j,t}^2.$$

Usage

```
wavVar(x, xform="modwt", wavelet="s8", n.levels=NULL,
    position=list(from=1,by=1,units=character()), units=character(),
    documentation=character(), sdf=NULL, sdfargs=NULL,
    sampling.interval=1, n.fft=1024)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
documentation	a character string used to describe the input data. Default: character().
n.fft	a positive integer (greater than one) defining the number of frequencies to use in evaluating the SDF. The frequencies are uniformly distributed over the interval [0, Nyquist] ala f=[0, 1/P , 2/P, 3/P, ..., (n.freq-1)/P] where P=2*(n.freq-1)/sampling.interval. Only used when the input SDF is not NULL. Default: 1024.
n.levels	the number of decomposition levels. Default: the maximum level at which there exists at least one interior wavelet coefficient.
position	a list containing the arguments from, by and to which describe the position(s) of the input data. All position arguments need not be specified as missing members will be filled in by their default values. Default: list(from=1, by=1, units=character()).
sampling.interval	sampling interval of the time series. Default: 1.
sdf	a spectral density function of the process corresponding to the input time series. This input must be a function whose first argument is f (representing frequency). At a minimum, the SDF must be defined over frequencies [0, Nyquist] where Nyquist=1/(2*sampling.interval). Additional arguments that are needed to calculate the SDF should be passed via the sdfargs parameter. This argument is used only for calculating mode 2 EDOF. If the EDOF mode 2 estimates are not desired, specify this this argument as NULL and the EDOF mode 2 and corresponding confidence intervals will not be calculated. See the mutilsSDF function for more details. Default: NULL.
sdfargs	a list of arguments passed directly to the SDF function ala do.call. Default: NULL (no additional arguments).
units	a string denoting the units of the time series. Default: character() (no units).
wavelet	a character string denoting the filter type. See wavDaubechies for details. Default: "s8".
xform	a character string denoting the type of wavelet transform: "modwt" or "dwt". Default: "modwt".

Value

an object of class wavVar.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavVarTest, wavEDOF, wavVarConfidence, mutilsSDF.

Examples

```
## create sequence
x <- make.signal("doppler")

## perform a time independent wavelet variance
## analysis
vmod <- wavVar(x)

## plot the result
plot(vmod, pch=15, title="Wavelet Variance of Doppler")

## calculate wavelet variance estimaates for the
## ocean series and calculate EDOF mode 2
## estimates and corresponding 95 percent
## confidence intervals
## Not run:
vocean <- wavVar(ocean, sdf=oceansdf, wavelet="d6")

## summarize the results
plot(vocean, edof=1:3)

summary(vocean)

## End(Not run)
```

---

wavVarConfidence *Wavelet variance confidence intervals*

Description

This function calculates wavelet variance confidence intervals for the unbiased and block averaged discrete wavelet variance estimates. Given $\hat{\nu}_X^2(\tau_j)$ are the time independent unbiased wavelet variance estimates at scales $\tau_j \equiv 2^{j-1}$ where j are the decomposition levels, the approximate $100(1-2p)\%$ confidence interval is given by

$$\left[\frac{n\hat{\nu}_X^2(\tau_j)}{Q_n(1-p)}, \frac{n\hat{\nu}_X^2(\tau_j)}{Q_n(p)}\right]$$

where $Q_n(p)$ is the $p \times 100$ percentage point for a chi-squared distribution with n degrees of freedom distribution.

Usage

```
wavVarConfidence(wvar, edof, probability=0.95)
```

Arguments

wvar	a vector containing the block-averaged unbiased wavelet variance estimates.
edof	a vector containing the equivalent degrees of freedom estimates. See wavEDOF for details.
probability	the probability desired for the confidence intervals. Supported probabilities are 0.005, .025, .05, .95, .975, and .995. Default: 0.95.

Value

a list of the low and high confidence interval limits for levels $1, \ldots, J$.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavVar, wavEDOF.

Examples

```
## first calculate the EDOF for the ocean series
edof <- wavEDOF(ocean)

## calculate the 95% confidence intervals for EDOF
## mode 1
wavVarConfidence(edof$variance.unbiased, edof$EDOF1)
```

---

wavVarTest *Homogeneity test for discrete wavelet transform crystals*

Description

Tests for homogeneity of variance for each scale of a discrete wavelet transform (DWT) decomposition. Based on the assumption that the DWT decorrelates colored noise processes, the interior wavelet coefficients in a given scale ($\mathbf{d}_j^{int}$) can be regarded as a zero mean Gaussian white noise process. For a homogeneous distribution of $\mathbf{d}_j^{int}$, there is an expected linear increase in the cumulative energy as a function of time. The so called D-statistic denotes the maximum deviation of the $\mathbf{d}_j^{int}$ from a hypothetical linear cumulative energy trend. This D-statistic is then compared to a table of critical D-statistics that defines the distribution of D for various sample sizes. Comparing the D-statistic of $\mathbf{d}_j^{int}$ to the corresponding critical values provides a means of quantitatively rejecting or accepting the linear cumulative energy hypothesis. This function performs this test for an ensemble of distribution probabilities.

Usage

```
wavVarTest(x, wavelet="s8", n.levels=NULL,
    significance=c(0.1,0.05,0.01), lookup=TRUE, n.realization=10000,
    n.repetition=3, tolerance=1e-6)
```

Arguments

x	an object of class wavTransform as output by the wavDWT function, a corresponding wavBoundary object, or a numeric vector. In the latter case, wavDWT parameters can be passed to specify the type of wavelet to use and the number of decomposition levels to perform.
lookup	a logical flag for accessing precalculated critical D-statistics. The critical D-statistics are calculated for a variety of sample sizes and significances. If lookup is TRUE, this table is accessed. The table is stored as the matrix object D.table.critical and is loaded with the package. Missing table values are calculated using the input arguments: n.realization, n.repetition and tolerance. Default: TRUE.
n.levels	the number of decomposition levels. Valid only for input not of class wavTransform or wavBoundary. Default: the maximum decomposition level that contains at least one interior wavelet coefficient.
n.realization	an integer specifying the number of realizations to generate in a Monte Carlo simulation for calculating the D-statistic(s). This parameter is used either when lookup is FALSE, or when lookup is TRUE and the table is missing values corresponding to the specified significances. Default: 10000.
n.repetition	an integer specifying the number of Monte Carlo simulations to perform. This parameter is coordinated with the n.realization parameter. Default: 3.
significance	a numeric vector of real values on the interval (0,1). Qualitatively the significance is the fraction of times that the linear cumulative energy hypothesis is incorrectly rejected. It is equal to the difference of the distribution probability (p) and unity. Default: c(0.1, 0.05, 0.01).
tolerance	a numeric real scalar that specifies the amplitude threshold to use in estimating critical D-statistic(s) via the Inclan-Tiao approximation. Setting this parameter to a higher value results in a lesser number of summation terms at the expense of obtaining a less accurate approximation. Default: 1e-6.
wavelet	a character string denoting the filter type. Valid only for input not of class wavTransform or wavBoundary. Default: "s8".

Details

An Inclan-Tiao approximation of critical D-statistics is used for sample sizes $N \geq 128$ while a Monte Carlo technique is used for $N < 128$. For the Monte Carlo technique, the D-statistic for a Gaussian white noise sequence of length N is calculated. This process is repeated n.realization times, forming a distribution of the D-statistic. The critical values corresponding to the significances are calculated a total of n.repetition times, and averaged to form an approximation to the D-statistic(s). Because the Monte Carlo study can be both computationally and memory intensive, it is highly recommended that lookup be set to TRUE, its default value.

Value

an object of class `wavVarTest`.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

`wavVar`, `wavDWT`, `D.table`.

Examples

```
## perform a homogeneity of variance test for a
## DWT decomposition of a long memory process
## realization
homogeneity <- wavVarTest(fdp045)
```

`wavZeroPhase` *Zero phase shift factors for Daubechies symmlet and coiflet filters*

Description

Daubechies coiflet and symmlet filters are approximate linear phase filters. Consequently, the wavelet and scaling coefficients of the DWT and MODWT can be circularly shifted for approximate zero phase alignment with the original time series. This function calculates the circular shift factors needed to bring the wavelet and scaling coefficients to approximate zero phase.

Usage

```
wavZeroPhase(wavelet="s8", levels=1:3)
```

Arguments

`levels`	an integer vector containing the decomposition levels. Default: `1:3`.
`wavelet`	a character string denoting the filter type. See `wavDaubechies` for details. Default: `"s8"`.

Details

Only relevant for DWT or MODWT definitions as given in the above reference and is valid only for Daubechies symmlet and coiflet filters.

Value

a list containing the shifts for each crystal of a DWTor MODWT for the specified decomposition `levels`. A negative shift factor implies an advance (circular shift to the left) of the wavelet transform crystals.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

See Also

wavDaubechies, wavDWT, wavMODWT, wavShift.

Examples

```
## calculate the zero phase shift factors for
## Daubechies coiflet 12-tap filters for levels
## 2 and 4.
wavZeroPhase(wavelet="c12", levels=c(2,4))
```

Index