Package ‘PaleoSpec’

March 12, 2021

Title Spectral tools for the ECUS group

Version 0.2.0

Description Spectral tools for the ECUS group

Depends R (¿= 3.6.1)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports multitaper,
zoo

Suggests testthat

R topics documented:

`AddConfInterval` *Add confidence intervals to a spectrum*

Description

Add confidence intervals to a spectrum

Usage

```
AddConfInterval(spec, MINVALUE = 1e-10, pval = 0.05)
```

Arguments

`spec`	spectrum list(spec,freq,dof)
`MINVALUE`	Minimum value to which the confidence interval is limited
`pval`	Interval from (pval/2 to 1-pval/2) is constructed

Value

spectrum as the input but including lim.1 and lim.2 as new list elements

Author(s)

Thomas Laepple

Examples

```
N.R=1000
N.T=100
save.spec<-matrix(NA,N.T/2,N.R)
for (i.R in 1:N.R) {
save.spec[,i.R]<-SpecMTM(ts(SimPowerlaw(1, N.T)))$spec
}

q.empirical<-apply(save.spec,1,quantile,c(0.025,0.975))
testspec<-SpecMTM(ts(SimPowerlaw(1, N.T)))
LPlot(AddConfInterval(testspec),ylim=c(0.05,10))
lines(testspec$freq,q.empirical[1,],col="red")
lines(testspec$freq,q.empirical[2,],col="red")
legend("bottomleft",lwd=2,col=c("black","red"),
c("one realization with chisq conf intervals","MC confidence intervals"))
```

`AnPowerlaw` *A PSD(freq) for a powerlaw with variance 1*

Description

A PSD(freq) for a powerlaw with variance 1

Usage

```
AnPowerlaw(beta, freq, return.scaling = FALSE)
```

Arguments

`beta`	slope of the powerlaw
`freq`	frequency vector

Value

vector containing the PSD

Author(s)

Thomas Laepple

`ApplyFilter` *Apply a filter to a timeseries*

Description

Apply a filter to a timeseries the timestep provided by ts is not used!!!! Thus for timeseries with a different spacing than 1, the filter has to be adapted Using endpoint constrains as describen in Mann et al., GRL 2003 no constraint (loss at both ends) (method=0) minimum norm constraint (method=1) minimum slope constraint (method=2) minimum roughness constraint (method=3) circular filtering (method=4)

Usage

```
ApplyFilter(data, filter, method = 0)
```

Arguments

`data`	Input timeseries (ts object)
`filter`	vector of filter weights
`method`	constraint method choice 0-4

Value

filtered timeseries (ts object)

Author(s)

Thomas Laepple

`ApproxNearest` *approximate a timeseries using the nearest neighbour*

Description

extends approx which always takes the right or left neighbour or the weighted mean between both if f¿0¡1

Usage

```
ApproxNearest(x, y, xout, rule = 1)
```

Arguments

`x`	numeric vector giving the coordinates of the points to be interpolate
`y`	corresponding y values
`xout`	set of numeric values specifying where interpolation is to take place.
`rule`	an integer (of length 1 or 2) describing how interpolation is to take place outside the interval see ?approx

Value

a list with components 'x' and 'y', containing length(xouth) coordinates which interpolate the given data points

Author(s)

Thomas Laepple

Examples

```
x<-1:10
y<-1:10
xout<-seq(from=0,to=11,by=0.01)
plot(x,y,type="b",pch=19,xlim=range(xout))
result<-ApproxNearest(x,y,xout)
lines(result,col="red")
```

`AvgToBin` *Bin averaging*

Description

Average a vector into bins.

Usage

```
AvgToBin(x, y, N = 2, breaks = pretty(x, N), right = TRUE, bFill = FALSE)
```

Arguments

`x`	vector of values on which the data in `y` is tabulated; e.g. depth or time points.
`y`	vector of observation values to be averaged into bins. Must have the same length as `x`.
`N`	desired number of breaks (ignored if `breaks` are supplied directly).
`breaks`	vector of break point positions to define the averagig bins; if omitted, break point positions are calculated from the range of `x` and the desired number of breaks given by `N`.
`right`	logical; indicate whether the bin intervals should be closed on the right and open on the left (`TRUE`, the default), or vice versa (`FALSE`).
`bFill`	logical; if `TRUE`, fill empty bins using linear interpolation from the neighbours to the center of the bin.

Details

This function averages the vector `y` into bins according to the positon of `x` within the breaks. You can either specify a desired number N of breaks which are used to calculate the actual breaks via `pretty(x,N)`, or directly specify the N + 1 break positions. For `right = TRUE` (the default) the averaging bins are defined via `x > breaks[i]` and `x <= breaks[i + 1]`, else they are defined via `x >= breaks[i]` and `x < breaks[i + 1]`. If `bFill = TRUE`, empty bins are filled using linear interpolation from the neighbours to the center of the bin.

Probably the binning could be considerably speeded up by using `?cut`.

Value

a list with four elements:

`breaks`: numeric vector of the used break point positions.

`centers`: numeric vector with the positions of the bin centers.

`avg`: numeric vector with the bin-averaged values.

`nobs`: numeric vector with the number of observations contributing to each bin average.

Author(s)

Thomas Laepple

`Bandpass` *calculate weights for a bandpass filter*

Description

Derive the (smoothed) least square bandpass based on Bloomfield 1976

Usage

```
Bandpass(omega.upper, omega.lower, n, sample = 1, convergence = T)
```

Arguments

`omega.upper`	upper cutoff frequency
`omega.lower`	lower cutoff frequency
`n`	length of the filter, has to be odd
`sample`	sampling rate of the timeseries on which the filter will be applied (1/deltat)
`convergence`	TRUE: smoothed least square lowpass; FALSE = unsmoothed
`omega.c`	cutoff frequency

Value

vector of filter weights

Author(s)

Thomas Laepple

`ClosestElement` *Get closest element of a vector*

Description

Get closest element of a vector

Usage

```
ClosestElement(xvector, x, type = "N")
```

Arguments

`xvector`	a vector of values
`x`	the value to find the closest match to
`type`	

`ColTransparent` *Modify a color to get brighter and tranparent for the confidence intervals*

Description

Modify a color to get brighter and tranparent for the confidence intervals

Usage

```
ColTransparent(color, alpha = 0.8, beta = 150)
```

Arguments

`color`	color value, e.g. "red"
`alpha`	(0..1) transparency value
`beta`	(0..255) to make it brighter, this value gets added on the RGB values

Value

modified color

Author(s)

Thomas Laepple

`ConfRatio` *Confidence Interval of ratios*

Description

Confidence Interval of ratios based on a ChiSquare Distribution

Usage

```
ConfRatio(varratio, df.1, df.2, pval = 0.1)
```

Arguments

`df.1`	degree of freedom of denominator
`df.2`	degree of freedom of numerator

Value

lower and upper confidence intervals

Author(s)

Thomas Laepple

`ConfVar` *Provide ChiSquared confidence intervals for ratios*

Description

Provide ChiSquared confidence intervals for ratios

Usage

```
ConfVar(varlist, pval = 0.05)
```

Arguments

`varlist`	list(var,dof)
`pval`	requested p-value

Value

Output: confidence intervals

Author(s)

Thomas Laepple

`FirstElement` *first element of a vector*

Description

first element of a vector

Usage

```
FirstElement(x)
```

Value

first element of X

Author(s)

Thomas Laepple

`fweights` *weights*

Description

weights

Usage

```
fweights(ftarget, f, df.log)
```

Value

weight vector

Author(s)

Thomas Laepple

`fweights.lin` *fweights.lin*

Description

fweights.lin

Usage

```
fweights.lin(ftarget, f, df.log)
```

Value

weight vector

Author(s)

Thomas Laepple

`GetTransferFunction` *Derives and plots the transfer function (given a filter)*

Description

Get the transfer function of a symetric filter, page 122 in Bloomfield 1976, page 135 in Bloomfield 2000

Usage

```
GetTransferFunction(
  g.u,
  resolution = 100,
  freq = NULL,
  bPlot = is.null(freq) == TRUE,
  add = FALSE,
  ...
)
```

Arguments

`g.u`	a filter, a numeric vector, values should sum to 1
`resolution`	the number of frequencies at which to evaluate the transfer function
`freq`	the specific frequency(s) at which to evaluate the transfer function if NULL, transfer function is evaluated at 1:resolution frequencies
`bPlot`	logical, plot the transfer function, defaults to FALSE if frequencies are specified, TRUE otherwise
`add`	logical, add to a previous plot
`...`	other arguments to pass to the plotting function

Value

list(freq, y) containing the transfer function

Author(s)

Thomas Laepple

Examples

```
l <- 11
tf <- GetTransferFunction(rep(1/l, l), resolution = 1000)
tf <- GetTransferFunction(rep(1/l, l), freq = 1/c(100, 10, 2))
```

GetVarFromSpectra *Variance estimate by integrating a part of the spectrum*

Description

Variance estimate by integrating a part of the spectrum

Usage

```
GetVarFromSpectra(spec, f, dfreq = NULL, df.log = 0, bw = 3)
```

Arguments

spec	spectrum (list of spec,freq,dof) to be analysed
f	f[1],f[2]: frequency interval to be analysed
dfreq	frequency discretisation used in the temporary interpolation
df.log	if ¿ 0, smooth the spectra prior to integrating
bw	the bandwidth assumed for the confinterval calculation (from the multi-taper spectral estimate)

Value

list(var,dof) variance and corresponding dof

Author(s)

Thomas Laepple

Examples

```
x<-ts(rnorm(100))
spec<-SpecMTM(x)
var(x) #Sample variance of the timeseries
GetVarFromSpectra(spec,c(1/100,0.5))
GetVarFromSpectra(spec,c(0.25,0.5))
```

`Highpass` *calculate weights for a bandpass filter*

Description

Derive the (smoothed) least square highpass based on Bloomfield 1976

Usage

```
Highpass(omega.c, n = 9, sample = 1, convergence = T)
```

Arguments

`omega.c`	cutoff frequency
`n`	length of the filter, has to be odd
`sample`	sampling rate of the timeseries on which the filter will be applied (1/deltat)
`convergence`	TRUE: smoothed least square lowpass; FALSE = unsmoothed

Value

vector of filter weights

Author(s)

Thomas Laepple

`InverseFilter` *Construct the inverse filter in the time domain*

Description

Construct the inverse filter in the time domain

Usage

```
InverseFilter(filter.weights)
```

Arguments

`filter.weights`

Value

filter weights for the inverse filter

Author(s)

Thomas Laepple

`LastElement` *last element of a vector*

Description

last element of a vector

Usage

```
LastElement(x)
```

Arguments

`x`

Value

last element of X

Author(s)

Thomas Laepple

`LLines` *Add a spectrum to an existing log-log spectral plot.*

Description

This function adds a spectrum to an existing double-logarithmic plot and optionally adds a transparent confidence interval.

Usage

```
LLines(
  x,
  conf = TRUE,
  bPeriod = FALSE,
  col = "black",
  alpha = 0.3,
  removeFirst = 0,
  removeLast = 0,
  ...
)
```

Arguments

`x`	a spectral object resulting from a call to `SpecMTM`.
`conf`	if `TRUE` (the default) add a transparent confidence interval (suppressed if `x` contains no error limits).
`bPeriod`	if `TRUE` treat the x-axis values in units of period (inverse frequency). Defaults to `FALSE`.
`col`	color for the line plot and the confidence interval.
`alpha`	transparency level (between 0 and 1) for the confidence interval. Defaults to `0.3`.
`removeFirst`	omit `removeFirst` values on the low frequency side.
`removeLast`	omit `removeLast` values on the high frequency side.
`...`	further graphical parameters passed to `lines`.

Author(s)

Thomas Laepple

Examples

```
x <- ts(arima.sim(list(ar = 0.9), 1000))
spec <- SpecMTM(x)
LPlot(spec, col = "grey")
LLines(LogSmooth(spec), lwd = 2)
```

`LogSmooth` *Smoothes the spectrum using a log smoother*

Description

Smoothes the spectrum using a log smoother

Usage

```
LogSmooth(
  spectra,
  df.log = 0.05,
  removeFirst = 1e+06,
  removeLast = 0,
  bLog = FALSE
)
```

Arguments

`spectra`	spectra: list(spec,freq) spec[specIndex]: spectra density vector freq[specIndex]: frequency vector
`df.log`	width of the smoother in log units
`removeFirst`	elements to remove on the slow side (one element recommended because of the detrending
`removeLast`	elements to remove on the fast side
`bLog`	TRUE: average in the log space of the power, FALSE: arithmetic average

Value

smoothed spectrum

Author(s)

Thomas Laepple

Examples

```
x<-ts(arima.sim(list(ar = 0.9),1000))
spec<-SpecMTM(x)
LPlot(spec,col='grey')
LLines(LogSmooth(spec,df.log=0.01),lwd=2,col='green')
LLines(LogSmooth(spec,df.log=0.05),lwd=2,col='blue')
LLines(LogSmooth(spec,df.log=0.1),lwd=2,col='red')
legend('bottomleft', col=c('grey','green','blue','red'),
lwd=2,c('raw','smoothed 0.01',
 'smoothed 0.05', 'smoothed 0.1'), bty='n')
```

`Lowpass` *Calculate weights for lowpass filter*

Description

Derive the (smoothed) least square lowpass, given the cutoff frequency omega.c and the length of the filter n

based on Bloomfield 1976

Usage

```
Lowpass(omega.c, n = 9, sample = 1, convergence = TRUE)
```

Arguments

`omega.c`	cutoff frequency
`n`	length of the filter, has to be odd
`sample`	sampling rate of the timeseries on which the filter will be applied (1/deltat)
`convergence`	TRUE: smoothed least square lowpass; FALSE = unsmoothed

Value

vector of filter weights

Author(s)

Thomas Laepple

LPlot *Log-log spectral plot.*

Description

This function plots a spectrum on a double-logarithmic scale and optionally adds a transparent confidence interval.

Usage

```
LPlot(
  x,
  conf = TRUE,
  bPeriod = FALSE,
  bNoPlot = FALSE,
  axes = TRUE,
  col = "black",
  alpha = 0.3,
  removeFirst = 0,
  removeLast = 0,
  xlab = "f",
  ylab = "PSD",
  xlim = NULL,
  ylim = NULL,
  ...
)
```

Arguments

`x`	a spectral object resulting from a call to `SpecMTM`.
`conf`	if `TRUE` (the default) add a transparent confidence interval (suppressed if `x` contains no error limits).
`bPeriod`	if `TRUE` the x-axis is displayed in units of period (inverse frequency), increasing to the left. Defaults to `FALSE`.
`bNoPlot`	if `TRUE` only produce the plot frame (`type = "n"` behaviour of function `plot`). Defaults to `FALSE`.
`axes`	if `FALSE` the plotting of the x and y axes is suppressed. Defaults to `TRUE`.
`col`	color for the line plot and the confidence interval.
`alpha`	transparency level (between 0 and 1) for the confidence interval. Defaults to `0.3`.
`removeFirst`	omit `removeFirst` values on the low frequency side.
`removeLast`	omit `removeLast` values on the high frequency side.
`xlab`	character string for labelling the x-axis.
`ylab`	character string for labelling the y-axis.
`xlim`	range of x-axis values; if `NULL` (the default) it is calculated internally and automatically reversed for `bPeriod = TRUE`.
`ylim`	range of y-axis values; if `NULL` (the default) it is calculated internally.
`...`	further graphical parameters passed to `plot`.

Author(s)

Thomas Laepple

Examples

```
x <- ts(arima.sim(list(ar = 0.9), 1000))
spec <- SpecMTM(x)
LPlot(spec, col = "grey")
LLines(LogSmooth(spec), lwd = 2)
```

MakeEquidistant *Average an irregular timeseries to a regular timeseries*

Description

Make an irregular timeseries equidistant by interpolating to high resolution, lowpass filtering to the Nyquist frequency, and subsampling; e.g. as used in Huybers and Laepple, EPSL 2014

Usage

```
MakeEquidistant(
  t.x,
  t.y,
  dt = NULL,
  time.target = seq(from = t.x[1], to = t.x[length(t.x)], by = dt),
  dt.hres = NULL,
  bFilter = TRUE,
  k = 5,
  kf = 1.2,
  method.interpolation = "linear",
  method.filter = 2
)
```

Arguments

`t.x`	vector of timepoints
`t.y`	vector of corresponding values
`dt`	target timestep; can be omitted if time.target is supplied
`time.target`	time vector to which timeseries should be averaged/interpolated to by default the same range as t.x with a timestep dt
`dt.hres`	timestep of the intermediate high-resolution interpolation. Should be smaller than the smallest timestep
`bFilter`	(TRUE) low passs filter the data to avoid aliasing, (FALSE) just interpolate
`k`	scaling factor for the Length of the filter (increasing creates a sharper filter, thus less aliasing)
`kf`	scaling factor for the lowpass frequency; 1 = Nyquist, 1.2 = 1.2xNyquist is a tradeoff between reducing variance loss and keeping aliasing small

`method.interpolation`
: 'linear' or 'constant', see approx

`method.filter`
: To avoid loosing data at the ends of the dataset, endpoint constrains are used (see ApplyFilter) no constraint (loss at both ends) (method=0), only works if t.x covers more time than time.target minimum norm constraint (method=1) minimum slope constraint (method=2) minimum roughness constraint (method=3) circular filtering (method=4)

Value

ts object with the equidistant timeseries

Author(s)

Thomas Laepple

`MeanSpectrum` *average spectra with weighting*

Description

Calculate the weighted mean spectrum of all spectra by interpolating them to the highest resolution frequency grid and averaging them.

Spectra can have different resolution and span a different freq range.

Usage

```
MeanSpectrum(specList, iRemoveLowest = 1, weights = rep(1, length(specList)))
```

Arguments

`iRemoveLowest`	number of lowest frequencies to remove (e.g. to remove detrending bias)
`weights`	vector of weights (same length as elements in speclist)
`speclist`	list of spectra

Value

list(spec,nRecords) spec=average spectrum, nRecords = number of records contributing to each spectral estimate

Author(s)

Thomas Laepple

`MonthlyFromDaily` *Bin daily values to monthly values*

Description

Assumes months of equal length and a 365 day long year

Usage

```
MonthlyFromDaily(ts.daily)
```

Arguments

`ts.daily`	vector of 365 values

Value

vector of 12 values

Author(s)

Thomas Laepple

`NaFillTs` *NaFill*

Description

NaFill

Usage

```
NaFillTs(x)
```

Arguments

`x`	

Value

filled x

Author(s)

Thomas Laepple

`PS.VarUntilF` *Variance of a powerlaw process if integrated from until frequency f*

Description

Integral of PSD=f^(-beta) from f1=1/N to f2=f this equals the variance of a lowpass filtered powerlaw process WARNING: The result is not normalized

Usage

```
PS.VarUntilF(f, beta, N)
```

Arguments

`f`	frequency until which to integrate
`beta`	powerlaw slope
`N`	length of the timeseries

Value

non-normalized variance

Author(s)

Thomas Laepple

Examples

```
beta <- 1
signal <- ts(SimPowerlaw(beta,100000))
spec <- SpecMTM(signal)
v1 <- GetVarFromSpectra(spec,f=c(1/length(signal),0.5))
v2 <- GetVarFromSpectra(spec,f=c(1/length(signal),0.01))
PS.VarUntilF(0.01,beta,length(signal))/PS.VarUntilF(0.5,beta,length(signal))
v2$var/v1$var
```

`PSP.CorAfterRollmean` *Numerical correlation of random timeseries with different filtering (running mean)*

Description

Numerical correlation of random timeseries with different filtering (running mean)

Usage

```
PSP.CorAfterRollmean(N, betaSignal, betaNoise, R)
```

Arguments

`N`	Number of points of the timeseries
`betaSignal`	powerlaw slope of the signal
`betaNoise`	powerlaw slope of the noise
`R`	expected correlation of the whole timeseries

Value

correlation of unfiltered, 10point mean and 50 point mean

Author(s)

Thomas Laepple

Examples

```
temp <- replicate(1000,PSP.CorAfterRollmean(1000,1,0,0.5))
rowMeans(temp)
PSP.CorUntilF(c(0.5,0.5/10,0.5/50),1,0,1000,0.5)
```

`PSP.CorUntilF` *lowpass filtered expected correlation of powerlaw signal pair*

Description

Correlation of two timeseries with powerlaw signal and powerlaw noise evaluated until f this equals the correlation of linearly coupled lowpass filtered powerlaw process

Usage

```
PSP.CorUntilF(f, betaSignal, betaNoise, N, r)
```

Arguments

`f`	frequency until which to integrate
`betaSignal`	powerlaw slope of the signal
`betaNoise`	powerlaw slope of the noise
`N`	Number of points per timeseries
`r`	expected correlation of the unfiltered

Value

expected correlation of the timeseries

Author(s)

Thomas Laepple

Examples

```
temp <- replicate(1000,PSP.CorAfterRollmean(1000,1,0,0.5))
rowMeans(temp)
PSP.CorUntilF(f=c(0.5,0.5/10,0.5/50),betaSignal=1,betaNoise=0,N=1000,r=0.5)
```

`SimFromEmpiricalSpec` *Simulate a random timeseries consistent with an arbitrary numerical power spectrum*

Description

Adapted from SimPowerlaw

Usage

```
SimFromEmpiricalSpec(spec, N)
```

Arguments

`spec`	Numerical power spectrum consisting of a list with components $freq and $spec
`N`	length of timeseries to be generated

Value

vector containing the timeseries

Author(s)

Thomas Laepple and Andrew Dolman

See Also

Other SimPowerlaw SimPLS SimFromEmpiricalSpectrum: `SimPowerlaw()`

Examples

```
# Create a piecewise spectrum

## helper function to generate continuous piecewise spectrum

PiecewiseLinear <- function(x, val.at.min.x, breaks, slopes){

breaks <- c(-Inf, breaks, Inf)
slp.vec <- slopes[findInterval(x, breaks)]
d.x <- diff(x)
d.y <- c(d.x * tail(slp.vec, -1))

y <- cumsum(c(val.at.min.x, d.y))

data.frame(x, y)

}
slps <- c(-1, -0.5, -1)
brks <- c(1e-03, 1e-02)
emp.spec <- PiecewiseLinear(log(seq(1/1e05, 1/2, 1/1e05)), 0, log(brks), slps)
emp.spec <- exp(emp.spec)
names(emp.spec) <- c("freq", "spec")
```

```
plot(emp.spec, type = "l", log = "xy")

# Sample consistent with spectrum
ts1 <- ts(SimFromEmpiricalSpec(emp.spec, 50000))

# re-estimate power spectrum
spec1 <- SpecMTM(ts1)
LPlot(spec1)
lines(emp.spec, col = "Red")
abline(v = brks, col = "Green")
```

SimPLS *Simulate a random timeseries with a powerlaw spectrum*

Description

This function creates a power-law series. It has the problem that it effectively produces (fractional) Brownian bridges, that is, the end is close to the beginning (cyclic signal), rather than true fBm or fGn series.

If alpha¿0, then the EXPECTED PSD is equal to alpha*fˆ(-beta).

If alpha¡0, then the timeseries is normalized such that it has EXPECTED variance abs(alpha), and the EXPECTED PSD is proportional to fˆ(-beta).

Usage

```
SimPLS(N, beta, alpha = -1)
```

Arguments

N	length of timeseries to be generated
beta	Slope of the powerlaw. beta = 1 produces timeseries with -1 slope when plotted on log-log power ˜ frequency axes
alpha	the constant. If alpha ¿ 0 this is the parameter alpha * fˆ(-beta). If alpha ¡ 0, the variance of the returned timeseries is scaled so that its expected value is abs(alpha)

Value

a vector containing the timeseries

Author(s)

Torben Kunz, Andrew Dolman

Examples

```
# With a beta = 1 and alpha = 0.1
set.seed(202010312)
ts1 <- ts(SimPLS(N = 1000, beta = 1, alpha = 0.1))
plot(ts1)
sp1 <- SpecMTM(ts1)
LPlot(sp1)
```

```
abline(log10(0.1), -1, col = "Red")

# beta = 0.5, alpha = 0.4
ts2 <- ts(SimPLS(1000, beta = 0.5, alpha = 0.4))
plot(ts2)
sp2 <- SpecMTM(ts2)
LPlot(sp2)
abline(log10(0.4), -0.5, col = "Red")

# beta = 1, alpha = -2
ts3 <- ts(SimPLS(1000, 1, alpha = -2))
plot(ts3)
var(ts3)

# the EXPECTED variance is -2, for a given random timeseries the actual value will differ
rep.var <- replicate(100, {
 var(SimPLS(1000, 1, -2))
})

hist(rep.var)
abline(v  = 2, col = "Red")
mean(rep.var)
```

`SimPowerlaw` *Simulate a random timeseries with a powerlaw spectrum*

Description

Simulate a random timeseries with a powerlaw spectrum

Usage

```
SimPowerlaw(beta, N)
```

Arguments

`beta`	slope
`N`	length of timeseries to be generated

Details

Method: FFT white noise, rescale, FFT back, the result is scaled to variance 1

Value

vector containing the timeseries

Author(s)

Thomas Laepple

See Also

Other SimPowerlaw SimPLS SimFromEmpiricalSpectrum: `SimFromEmpiricalSpec()`

`SimPowerlawPiecewise` *Simulate a timeseries with length N which has a spectra consisting of two powerlaws*

Description

Simulate a timeseries with length N which has a spectra consisting of two powerlaws

Usage

```
SimPowerlawPiecewise(beta1, beta2, N, deltat = 1, breakpoint = 1/50)
```

Arguments

`beta1`	slope for frequencies lower than breakpoint
`beta2`	slope for frequencies higher than breakpoint
`N`	Number of points to simulate
`deltat`	timestep of the timeseries
`breakpoint`	frequency of the breakpoint

Value

N random numbers drawn according to the piecewise powerlaw PSD

Author(s)

Thomas Laepple

`SimulatePowerlawSignalPair`
Create a pair of random signals with powerlaw signal and powerlaw noise

---

Description

The timeseries have an expected variance of 1 and an expected correlation of r

Usage

```
SimulatePowerlawSignalPair(n, beta.signal, beta.noise, r)
```

Arguments

`r`	expected correlation between both vectors
`N`	Number of points per timeseries
`betaSignal`	powerlaw slope of the signal
`betaNoise`	powerlaw slope of the noise

Value

list containing both vectors y1 and y2

Author(s)

Thomas Laepple

Examples

```
mean(replicate(1000,{test <- SimulatePowerlawSignalPair(200,1,1,0.5);cor(test$y1,test$y2)}))
```

SlopeFit *Fit a power-law to the spectrum*

Description

Fit a power-law to the spectrum

Usage

```
SlopeFit(
  spec,
  freq.start = NULL,
  freq.end = NULL,
  bDebug = TRUE,
  breaks = NULL,
  indexRemove = NULL,
  df.log = 0.05,
  i.fStart = 4
)
```

Arguments

freq.start	vector containing the start frequencies of the fitting interval(s)
freq.end	vector containing the end frequencies of the fitting interval(s)
bDebug	(TRUE) plot diagnostics
breaks	vector of breakpoints to which the spectra is binned (optional)
indexRemove	bins that are removed (e.g. containing the annual and semiannual cycle)
df.log	resolution of the bins (if breaks are not provided)
i.fStart	index of first (lowest) frequency to be used

Value

list(slope=slope,slopesd=slopesd,spec=saveSpec,freq=binFreq,intercept=intercept)

Author(s)

Thomas Laepple

`smoothlin.cutEnd` *smoothlin.cutEnd*

Description

smoothlin.cutEnd

Usage

```
smoothlin.cutEnd(x, f, df.log, dof = 1)
```

Value

smoothed x

Author(s)

Thomas Laepple

`smoothlog` *smoothlog*

Description

smoothlog

Usage

```
smoothlog(x, f, df.log)
```

Value

smoothed x

Author(s)

Thomas Laepple

smoothlog.cutEnd — *smoothlog.cutEnd*

Description

smoothlog.cutEnd

Usage

```
smoothlog.cutEnd(x, f, df.log, dof = 1)
```

Value

smoothed x

Author(s)

Thomas Laepple

SpecInterpolate — *Interpolates the spectrum spec to the specRef frequency resolution*

Description

Interpolates the spectrum spec to the specRef frequency resolution

Usage

```
SpecInterpolate(freqRef, spec)
```

Arguments

`freqRef`	frequency vector of the target resolution
`spec`	list(spec,freq,dof)

Value

one spectum as list(spec,freq,dof) (spec on the specRef resolution)

Author(s)

Thomas Laepple

`SpecMean` *Mean spectrum from of a list of spectra*

Description

Mean spectrum from of a list of spectra

Usage

```
SpecMean(speclist, weight = FALSE, dof = TRUE)
```

Arguments

`speclist`	list of spectra each containing list(spec, freq, dof)
`weight`	weight by the uncertainty

Details

Returns the mean of the spectra Inputs: speclist[[]], each containing a list(spec,freq,dof) spec[specIndex]: spectra density vector freq[specIndex]: frequency vector dof[specIndex]: DOF ... or a single value df weight: weight by the uncertainty

Weighting by 1/sigma^2; sigma^2 = variance is proportional to 1/DOF

This is true if we assume a constant spectral density... if not, higher spectral densities have higher uncertainty

Value

the mean spectra

`SpecMTM` *MTM spectral estimator*

Description

calls `spec.mtm` from library multitaper

Usage

```
SpecMTM(
  timeSeries,
  k = 3,
  nw = 2,
  nFFT = "default",
  centre = c("Slepian"),
  dpssIN = NULL,
  returnZeroFreq = FALSE,
  Ftest = FALSE,
  jackknife = FALSE,
  jkCIProb = 0.95,
```

```
    maxAdaptiveIterations = 100,
    plot = FALSE,
    na.action = na.fail,
    returnInternals = FALSE,
    detrend = TRUE,
    bPad = FALSE,
    ...
)
```

Arguments

`timeSeries` A time series of equally spaced data, this can be created by the ts() function where deltat is specified.

`k` a positive integer, the number of tapers, often 2*nw.

`nw` a positive double precision number, the time-bandwidth parameter.

`nFFT` This function pads the data before computing the fft. nFFT indicates the total length of the data after padding.

`centre` The time series is centred using one of three methods: expansion onto discrete prolate spheroidal sequences ('Slepian'), arithmetic mean ('arithMean'), trimmed mean ('trimMean'), or not at all ('none').

`dpssIN` Allows the user to enter a dpss object which has already been created. This can save computation time when Slepians with the same bandwidth parameter and same number of tapers are used repeatedly.

`returnZeroFreq` Boolean variable indicating if the zeroth frequency (DC component) should be returned for all applicable arrays.

`Ftest` Boolean variable indicating if the Ftest result should be computed and returned.

`jackknife` Boolean variable indicating if jackknifed confidence intervals should be computed and returned.

`jkCIProb` Decimal value indicating the jackknife probability for calculating jackknife confidence intervals. The default returns a 95% confidence interval.

`maxAdaptiveIterations`
Maximum number of iterations in the adaptive multitaper calculation. Generally convergence is quick, and should require less than 100 iterations.

`plot` Boolean variable indicating if the spectrum should be plotted.

`na.action` Action to take if NAs exist in the data, the default is to fail.

`returnInternals`
Return the weighted eigencoefficients, complex mean values, and so on. These are necessary for extensions to the multitaper, including magnitude-squared coherence (function mtm.coh in this package). Note: The internal ($mtm) variables eigenCoefs and eigenCoefWt correspond to the multitaper eigencoefficients. The eigencoefficients correspond to equation (3.4) and weights, eigenCoefWt, correspond to sqrt(—d_k(f)—^2) from equation (5.4) in Thomson's 1982 paper. This is because the square root values contained in eigenCoefWt are commonly used in additional calculations (example: eigenCoefWt * eigenCoefs). The values returned in mtm$cmv correspond to the the estimate of the coefficients hat(mu)(f) in equation (13.5) in Thomson (1982), or to the estimate of hat(C)_1 at frequency 1 in equation (499) form Percival and Walden (1993)

`detrend`	logical, detrend timeseries before estimating the spectrum
`bPad`	if FALSE (the default) nFFT is set to the length of the timeseries
`...`	Additional parameters, such as xaxs="i" which are passed to the plotting function. Not all parameters are supported.

Value

spectra object list(freq, spec, dof)

Author(s)

Thomas Laepple

Examples

```
x <- ts(arima.sim(list(ar = 0.9), 1000))
spec <- SpecMTM(x)
LPlot(spec, col='grey')
LLines(LogSmooth(spec), lwd=2)
```

`SubsampleTimeseriesBlock` *Subsample (downsample) timeseries using block averaging#'*

Description

Resample a equidistant timeseries (e.g. model result) at the 'timepoints' using block averaging. The blocks are divided at 1/2 time between the requested output points. For the first (and last) timepoint, the interval starting mean(diff(timepoints)) before (ending after) are used. Example usage is to downsample a model timeseries to mimick an integrating proxy (e.g. water isotopes that are measured by melting pieces of ice).

Usage

```
SubsampleTimeseriesBlock(ts, timepoints)
```

Arguments

`ts`	ts object or vector containing the equidistant timeseries
`timepoints`	vector with the points in time

Value

values at timepoints

Author(s)

Thomas Laepple

Examples

```
input <- ts(SimPowerlaw(0.5, 1000))
timepoints <- seq(from = 50, to = 950, by = 50)
result <- SubsampleTimeseriesBlock(input, timepoints)
plot(input, main = "Comparison of block avg. vs. simple interpolation",
  ylab = "unitless")
points(timepoints, result, pch = 19, col = "red", lwd = 3)
points(approx(time(input), c(input), timepoints), col = "green",
  pch = 10, lwd = 3)
legend("bottom", col = c("black", "red", "green"), lwd = 2, bty = "n",
  c("High-resolution timeseries (input)", " Block Avg", "interpolated values"))
```

Index