Package 'psd'

July 2, 2014

Type Package

Title Adaptive, sine-multitaper power spectral density estimation

Version 0.4-1

Date 2014-04-15

Author Robert L. Parker and Andrew J. Barbour

Maintainer Andrew J. Barbour <andy.barbour@gmail.com>

Description Power spectral density estimates are produced through iterative refinement of the optimal number of sine-tapers at each frequency. The optimization procedure is based on the method of Riedel and Sidorenko (1995), which applies
smoothing that varies with frequency to minimize the sum of variance and bias at each point.

License GPL

URL http://abarbour.github.io/psd/ and
http://dx.doi.org/10.1016/j.cageo.2013.09.015 for the citation.

Depends R (>= 2.14.1), stats, utils, graphics, grDevices, fftw (>= 1.0-3)

Imports RColorBrewer, signal, zoo

Suggests bspec, ggplot2 (>= 0.9), knitr, multitaper, plyr, RSEIS,rbenchmark, reshape2

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-04-16 22:41:05

R topics documented:

`psd-package` *Adaptively estimate power spectral densities of an optimally tapered series.*

Description

Perform adaptive estimation of the power spectral density (PSD) using the sine multitapers in which the number of tapers (and hence the resolution and uncertainty) vary according to spectral shape. The main function to be used is `pspectrum`.

Details

In frequency ranges where the spectrum (S) is relatively flat, more tapers are taken and so a higher accuracy is attained at the expense of lower frequency resolution. The program makes a pilot estimate of the spectrum, then uses Riedel and Sidorenko's estimate of the MSE (minimum square error) value, which is based on an estimate of the second derivative of the PSD (S''). The process is repeated `niter` times with a default of `niter=5`. Further iteration may be necessary to reach convergence, or an acceptably low spectral variance. Although the term "acceptable" is rather subjective, one can usually detect an unconverged state by a rather jagged appearence of the spectrum; this is rather uncommon in our experience.

Adaptive estimation: The adaptive process used is as follows. A quadratic fit to the logarithm of the PSD within an adaptively determined frequency band is used to find an estimate of the local second derivative of the spectrum. This is used in an equation like R-S eq (13) for the MSE taper number, with the difference that a parabolic weighting is applied with increasing taper order. Because the FFTs of the tapered series can be found by resampling the FFT of the original time series (doubled in length and padded with zeros) only one FFT is required per series, no matter how many tapers are used. The spectra associated with the sine tapers are weighted before averaging with a parabolically varying weight. The expression for the optimal number of tapers given by R-S must be modified since it gives an unbounded result near points where S'' vanishes, which happens at many points in most spectra. This program restricts the rate of growth of the number of tapers so that a neighboring covering interval estimate is never completely contained in the next such interval.

Resolution and uncertainty: The sine multitaper adaptive process introduces a variable resolution and error in the frequency domain. See documentation for `spectral_properties` details on how these are computed.

Author(s)

Robert L. Parker and Andrew J. Barbour <andy.barbour@gmail.com>

References

Parker, R. L., *PSD*, Program documentation. *Maintained Software*, N.p. 11 Nov. 2011, Web. 17 Jan. 2013, <http://igppweb.ucsd.edu/%7Eparker/Software/#PSD>.

Percival, D. B., and A.T. Walden (1993), Spectral analysis for physical applications, *Cambridge University Press*

Prieto, G. A., R. L. Parker, D. J. Thomson, F. L. Vernon, and R. L. Graham (2007), Reducing the bias of multitaper spectrum estimates, *Geophysical Journal International*, **171**, 1269–1281, doi: 10.1111/j.1365-246X.2007.03592.x

Riedel, K. S., & Sidorenko, A. (1995), Minimum bias multiple taper spectral estimation, *Signal Processing, IEEE Transactions on*, **43**(1), 188–195.

Riedel, K. S. (1996), Adaptive smoothing of the log-spectrum with multiple tapering, *Signal Processing, IEEE Transactions on*, **44**(7), 1794–1800.

Walden, A. T., and E. J. McCoy, and D. B. Percival (1995), The effective bandwidth of a multitaper spectral estimator, *Biometrika*, **82**(1), 201–214.

See Also

`pspectrum`

`as.tapers` *Coerce an object into a* `'tapers'` *object.*

Description

In a tapered spectrum estimation algorithm, it is necessary to enforce rules on the number of tapers that may be applied.

Usage

```
as.tapers(x, min_taper = 1, max_taper = NULL, setspan = FALSE)

tapers(x, min_taper = 1, max_taper = NULL, setspan = FALSE)
```

Arguments

`x`	An object to set
`min_taper`	Set all values less than this to this.
`max_taper`	Set all values greater than this to this.
`setspan`	logical; should the tapers object be passed through `minspan` before it is return?

Details

Formal requirements enforced by this function are:

- Non-zero.
- Integer values.
- Fewer than the half-length of the spectrum.

For example, we cannot apply zero tapers (the result would be a raw periodogram) or one million tapers (that would be absurd, and violate orthogonality conditions for any series less than two million terms long!).

An object with S3 class `'tapers'` is created; this will have a minimum number of tapers in each position set by `min_taper`, and a maximum number of tapers in each position set by `max_taper`. If `minspan=TRUE`, the bounded taper is fed through `minspan` which will restrict the maximum tapers to less than or equal to the half-length of the spectrum.

Various classes can be coerced into a `'tapers'` object; those tested sofar include: scalar, vector, matrix, data.frame, and list.

Multiple objects are concatenated into a single vector dimension.

Enabling `setspan` will only override `max_taper` should it be larger than the half-width of the series.

Value

An object with class taper.

Note

No support (yet) for use of `min_taper,max_taper` as vectors, although this could be quite desirable.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

See Also

`is.tapers`

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Objects with class 'tapers'
##
is.tapers(as.tapers(1))
is.tapers(as.tapers(1:10))
is.tapers(as.tapers(matrix(1:10,ncol=1)))
as.tapers(list(x=1:10,y=1:30)) # note dimensions
as.tapers(x<-data.frame(x=1:10,y=10:19))
as.tapers(x, min_taper=3, max_taper=10)
# class 'character' is in-coercible; raise error
try(as.tapers(c("a","b")), silent=TRUE)
#RDEX#}
```

`hfsnm` *Noise levels found in PBO strainmeter data at seismic frequencies.*

Description

These values represent noise levels in high frequency data ($10^{-3} - 10$ Hz) from 2009, averaged over all stations in the Anza cluster of the Plate Boundary Observatory (PBO) borehole strainmeter network, and the UCSD-style longbase laser strainmeters.

Format

A dataframe with 141 observations on the following 4 variables:

`freq` Frequencies, in Hertz.

`P50` The 50th percentile (median) noise levels in decibels relative to $1\epsilon^2/$ Hz.

`P10` The 10th percentile noise levels also in decibels.

`meter.type` The strainmeter design type.

and 2 attributes:

`source.doi` The DOI number of the source publication.

`generator` The structure of a function which will refresh the values from the supplemental files of the original publication.

Details

`NA` values in the series highlight frequency bands where the noise levels are unreliable, due to a instrumental artifact.

Source

Barbour, A. J., and Agnew, D. C. (2011), Noise Levels on Plate Boundary Observatory Borehole Strainmeters in Southern California, *Bulletin of the Seismological Society of America*, **101**(5), 2453-2466, doi:10.1785/0120110062

See Also

`pspectrum`, `Tohoku`, `magnet`

Examples

```
data(hfsnm)
str(hfsnm)
FUN <- attr(hfsnm, "generator")
try(dat <- FUN(molten=FALSE)) # may fail without library-access to BSSA
try(all.equal(dat[,1:4], hfsnm[,1:4]))
```

---

magnet *A single line of Project MAGNET horizontal field intensity.*

Description

The Project MAGNET mission provided a wealth of airborne-magnetometer data spanning the globe (Coleman, 1992). This dataset represents a single track of horizontal field intensities (a very small subset of the full collection!).

Format

A dataframe with 2048 observations on the following 4 variables.

`km` Relative along-track distance, in kilometers. The first observation is at zero kilometers.

`raw` Raw intensities, in nanotesla.

`clean` Edited raw intensites, in nanotesla

`mdiff` The difference between `clean` and `raw` intensities, in nanotesla.

Details

Raw and Clean Sets: There are non-real data points in raw MAGNET series; these are instrumental artefacts, and can severely affect power spectral density (PSD) estimates. A clean series has been included so that a comparison of PSDs may be made.

Some command like `subset(magnet, abs(mdiff) > 0)` can be used to identify the rows where edits have been made.

Source

Project MAGNET page: http://www.ngdc.noaa.gov/geomag/proj_mag.shtml

References

Coleman, R. J. (1992), Project Magnet high-level vector survey data reduction. In *Types and Characteristics of Data for Geomagnetic Field Modeling*, **3153**, pp. 215-248.

See Also

pspectrum, Tohoku, hfsnm

Examples

```
data(magnet)
summary(magnet)
```

`parabolic_weights` *Calculate parabolic weighting factors.*

Description

The resampled spectrum involves summing weighted tapers; this produces the weighting factors.

Usage

```
parabolic_weights(tapvec, tap.index = 1L)

## S3 method for class 'tapers'
parabolic_weights(tapvec, tap.index = 1L)

parabolic_weights_fast(ntap = 1L)

## Default S3 method:
parabolic_weights_fast(ntap = 1L)
```

Arguments

tapvec	'tapers' object; the number of tapers at each frequency
tap.index	integer; the index of tapvec from which to produce a sequence of weights for
ntap	integer; the number of tapers to provide weightings for.

Details

If one has a tapers object, specify the taper.index to produce a sequence of weights up to the value at that index; the user is likely to never need to use this function.

Weighting factors are calculated as follows:

$$W_N \equiv n_T^2 - \frac{3K_N^2}{2n_T(n_T - 1/4)(n_T + 1)}$$

where n_T is the total number of tapers, and K_N is the integer sequence $[0, n_T - 1]$

Value

A list with taper indices, and the weights W_N.

Author(s)

A.J. Barbour <andy.barbour@gmail.com> adapted original by R.L.Parker, and authored the optimized version.

See Also

psdcore, riedsid

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Show parabolic weighting factors
##

## some preliminaries
require(grDevices)
require(RColorBrewer)
#
maxx <- 1e3
xseq <- c(5,maxx,seq(from=1,to=2.8,by=0.2))
# plot palette
pal <- "Spectral"
npal <- switch(pal, RdYlBu=11, Spectral=11, Blues=9)
pal.col <- RColorBrewer::brewer.pal(npal, pal)
cols <- rev(grDevices::colorRampPalette(pal.col)(maxx))

## a roundabout way of bootstrapping y-axis limits:
```

```
WgtsU <- parabolic_weights_fast(xseq[1])
xseq <- xseq[-1]
DfU <- data.frame(matrix(unlist(WgtsU), ncol=2, byrow=FALSE))
WgtsL <- parabolic_weights_fast(xseq[1])
xseq <- xseq[-1]
DfL <- data.frame(matrix(unlist(WgtsL), ncol=2, byrow=FALSE))
# the limits:
ylims <- round(dB(c(min(DfL$X2), max(DfU$X2))), 1) + c(-2,5)

# function for plotting text
TFUN <- function(Df.){
  tx <- max(Df.$X1)
  ty <- mean(Df.$X2)
  text(log10(tx)+0.1, dB(ty), sprintf("%i", tx), col=cols[tx])
}
# function for weighting factors and plotting
WFUN<-function(x){
  message(x)
  Wgts <- parabolic_weights_fast(x)
  Df <- data.frame(matrix(unlist(Wgts), ncol=2, byrow=FALSE))
  lcol <- cols[x]
  lines(dB(X2) ~ log10(X1), Df, type="s", lwd=2, col=lcol)
  TFUN(Df)
}

## Plot parabolic weighting, in dB, colored by maximum num tapers
plot(dB(X2) ~ log10(X1), DfU, type="s", xlim=c(0, log10(maxx)+0.2),
     col=cols[5], lwd=2, ylim=ylims, yaxs="i",
     main="Multitaper weighting factors by maximum tapers applied",
     xlab="log10 taper sequence",
     ylab="dB")
TFUN(DfU)
invisible(lapply(round(10**xseq), FUN=WFUN))
WFUN(maxx)

##
#RDEX#}
```

`pilot_spec` *Calculate the pilot power spectral densities.*

Description

This PSD – the pilot spectrum – is used as the starting point for the adaptive estimation routine.

Usage

```
pilot_spec(x, x.frequency = 1, ntap = 7, remove.AR = 0, plot = FALSE,
  verbose = FALSE, ...)
```

```
## Default S3 method:
pilot_spec(x, x.frequency = 1, ntap = 7, remove.AR = 0,
  plot = FALSE, verbose = FALSE, ...)
```

Arguments

x	vector; the data series to find a pilot spectrum for
x.frequency	scalar; the sampling frequency (e.g. Hz) of the series
ntap	scalar; the number of tapers to apply during spectrum estimation
remove.AR	scalar; the max AR model to be removed from the data.
plot	logical; should a plot be created?
verbose	logical; should messages be given?
...	additional parameters passed to psdcore

Details

A fixed number of tapers is applied across all frequencies using psdcore, and subsequent taper-refinements are based on the spectral derivatives of this spectrum; hence, changes in the number of tapers can affect how many adaptive stages may be needed (though there are no formal convergence criteria to speak of).

The taper series of the returned spectrum is constrained using `as.tapers(..., minspan=TRUE)`.

The default behaviour (`remove.AR <= 0`) is to remove the standard linear model $[f(x) = \alpha x + \beta]$ from the data; however, the user can model the effect of an autoregressive process by specifiying `remove.AR`.

Value

An object with class 'spec', invisibly. It also assigns the object to `"pilot_psd"` in the working environment.

Removing an AR effect from the spectrum

If `remove.AR > 0` the argument is used as `AR.max` in prewhiten, from which an AR-response spectrum is calculated using the best fitting model.

If the value of `remove.AR` is too low the spectrum could become distorted, so use with care. *Note, however, that the value of* `remove.AR` *will be restricted to within the range* $[1, 100]$. If the AR order is much larger than this, it's unclear how prewhiten will perform and whether the AR model is appropriate.

Note that this function does not produce a parametric spectrum estimation; rather, it will return the amplitude response of the best-fitting AR model as `spec.ar` *would.* **Interpret these results with caution, as an AR response spectrum can be misleading.**

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

See Also

psdcore, prewhiten

Documentation for spec.ar.

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Pilot spectrum
##
data(magnet)
## simply calculate the pilot spectrum with a few tapers
plot(pilot_spec(xc <-  magnet$clean), log="dB",
     main="Pilot PSDs for MAGNET and its AR-innovations (red)")
## remove the effect of an AR model works exceptionally
## well for the Project MAGNET data:
plot(pilot_spec(xc, remove.AR=10), log="dB", add=TRUE, col="red")
##
#RDEX#}
```

---

prewhiten *Prewhiten a series.*

Description

Remove (optionally) mean, trend, and Auto Regressive (AR) model from the original series.

Usage

```
prewhiten(tser, AR.max = 0L, detrend = TRUE, demean = TRUE,
  impute = TRUE, plot = TRUE, verbose = TRUE, x.fsamp = 1,
  x.start = c(1, 1), ...)

## Default S3 method:
prewhiten(tser, AR.max = 0L, detrend = TRUE,
  demean = TRUE, impute = TRUE, plot = TRUE, verbose = TRUE,
  x.fsamp = 1, x.start = c(1, 1), ...)

## S3 method for class 'ts'
prewhiten(tser, AR.max = 0L, detrend = TRUE, demean = TRUE,
  impute = TRUE, plot = TRUE, verbose = TRUE, x.fsamp = NA,
  x.start = NA, ...)
```

Arguments

tser	vector; An object to prewhiten.
AR.max	numeric; the maximum AR order to fit.
detrend	logical; Should a trend (and mean) be removed?
demean	logical; Should a mean value be removed?
impute	logical; Should NA values be imputed?
plot	logical; Should the results be plotted?
verbose	logical; Should messages be printed?
x.fsamp	sampling frequency (for non ts objects)
x.start	start time of observations (for non ts objects)
...	variables passed to prewhiten.ts (for non ts objects)

Details

The R-S multitapers do not exhibit the remarkable spectral-leakage suppression properties of the Thomson prolate tapers, so that in spectra with large dynamic range, power bleeds from the strong peaks into neighboring frequency bands of low amplitude – spectral leakage. Prewhitening can ameliorate the problem, at least for red spectra [see Chapter 9, Percival and Walden (1993)].

The value of the AR.max argument is made absolute, after which this function has essentially two modes of operation (detailed below):

AR.max **== 0** Remove (optionally) a mean and/or linear trend.

AR.max **> 0** Remove an autoregressive model

In the second case, the time series is filtered in the time domain with a finite-impluse-response filter of AR.max terms. The filter is found by solving the Yule-Walker equations for which it is assumed the series was generated by an autoregressive process, up to order AR.max.

Mean and trend (AR.max == 0):

Power spectral density estimates can become badly biased (especially at lower frequencies) if a signal of the form $f(x) = Ax + B$ is not removed from the series. If detrend=TRUE a model of this form is removed over the entire series using a linear least-squares estimator; in this case a mean value is removed regardless of the logical state of demean. To remove *only* a mean value, set detrend=FALSE and (obviously) demean=TRUE.

Auto Regressive (AR) innovations (AR.max > 0):

When an autoregressive model is removed from a non-stationary series, the residuals are known as 'innovations', and may be stationary (or very-nearly stationary). This function fits an AR model [order at least 1, but up to and including AR(AR.max)] to the series by solving the Yule-Walker equations; however, AIC is used to estimate the highest significant order, which means that higher-order components may not necessarily be fit. The resulting innovations can be used to better estimate the stationary component of the original signal, and possibly in an interactive editing method.

Note that the method used here–solving the Yule-Walker equations–is not a true maximum likelihood estimator; hence the AIC is calculated based on the variance estimate (no determinant).

From ?ar: *In* ar.yw *the variance matrix of the innovations is computed from the fitted coefficients and the autocovariance of* x.

A quick way to determine whether this may be needed for the series is to run acf on the series, and see if significant non-zero lag correlations are found. A warning is produced if the fit returns an AR(0) fit, indicating that AR prewhitening most likely inappropriate for the series, which is apparently stationary (or very nearly so). (The innovations could end up having *higher* variance than the input series in such a case.)

Note that AR.max *is restricted to the range* $[1, N-1]$ *where* N *is the series length.*

Value

A list with the model fits (lm and ar objects), the linear and AR prewhitened series (ts objects), and a logical flag indicating whether the I/O has been imputed. The names in the list are: "lmdfit", "ardfit", "prew_lm", "prew_ar", and "imputed"

Note that if AR.max=0 *the AR information will exist as* NULL.

NA values

NA values are allowed. If present, and impute=TRUE, the na.locf function in the package zoo is used twice (with and without fromLast so that lead and trailing NA values are also imputed). The function name is an acronym for "Last Observation Carried Forward", a very crude method of imputation.

Author(s)

A.J. Barbour <andy.barbour@gmail.com> and Robert L. Parker

See Also

psdcore, pspectrum

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Using prewhiten to improve spectral estimates
##
data(magnet)
dx <- 1
mts <- ts(magnet$clean, frequency=dx)
mts.slope <- mts + seq_along(mts)
# mean + trend
# Prewhiten by removing mean+trend, and
# AR model; fit truncates the series by
# a few terms, so zero pad
mts <- prewhiten(mts.slope,  AR.max=10, zero.pad="rear")
mts.p <- mts$prew_lm
mts.par <- mts$prew_ar
#
```

```
ntap <- 20
ylog <- "dB"
plot(PSD <- psdcore(mts.p, ntaper=ntap), log=ylog, lwd=2, ylim=c(-5,35))
# remove the effect of AR model
PSD.ar <- psdcore(mts.par, ntaper=ntap)
PSD.ar$spec <- PSD.ar$spec / mean(PSD.ar$spec)
PSD$spec <- PSD$spec / PSD.ar$spec
plot(PSD, log=ylog, add=TRUE, lwd=2, col="red")
plot(PSD.ar, log=ylog, add=TRUE, col="blue", lwd=2)
##
#RDEX#}
```

---

psd-environment *Various environment manipulation functions.*

Description

The computation of *adaptive* power spectral density estimates requires bookkeeping and non-destructive manipulation of variables. The functions here are mainly convenience wrappers designed to maintain variable separation from the `.GlobalEnv` environment so that no innocent variable is destroyed in the process of iteratively computing spectra. **The user should generally not be using the *setters* even though all functions exist in the namespace.**

`get_psd_env_pointer` is a convenience wrapper to get the environment pointer.

`get_psd_env_name` is a convenience wrapper to get the environment name.

`psd_envRefresh` will clear any variables in the enviroment and reset the initialization stamp.

`psd_envClear` clears the contents of the environment.

`psd_envStatus` returns a list of some information regarding the status of the environment.

`psd_envList` returns a listing of any assignments.

`psd_envGet` returns the value of `variable`.

`psd_envAssign` assigns `value` to `variable`, but does not return it.

`psd_envAssignGet` both assigns and returns a value.

`new_adapt_history` initializes a nested-list object to store the data from each iteration.

`update_adapt_history` updates the adaptive estimation history list.

Usage

```
get_psd_env_pointer()

get_psd_env_name()

psd_envRefresh(verbose = TRUE)

psd_envClear()
```

```
psd_envStatus()

psd_envList()

psd_envGet(variable)

psd_envAssign(variable, value)

psd_envAssignGet(variable, value)

new_adapt_history(adapt_stages)

get_adapt_history()

update_adapt_history(stage, ntap, PSD, freq = NULL)
```

Arguments

verbose	logical; should messages be given?
variable	character; the name of the variable to get or assign
value	character; the name of the variable to assign
adapt_stages	scalar; The number of adaptive iterations to save (excluding pilot spectrum).
stage	scalar; the current stage of the adaptive estimation procedure
ntap	vector; the tapers
PSD	vector; the power spectral densities
freq	vector; the frequencies

Value

`psd_envRefresh` returns (invisibly) the result of `psd_envStatus()`.

the object represented by `variable` in the `psd` environment.

Defaults and Initialization

One can use `get_psd_env_pointer()` and `get_psd_env_name()` to access the pointer and name of the environment, if needed.

`psd_envRefresh` should be used when a fresh environment is desired: typically only if, for example, `psdcore` is used rather than `pspectrum`.

Assigning and Retieving

`psd_envAssign` and `psd_envGet` perform the assignments and retrieval of objects in the environment. A convenience function, `psd_envAssignGet`, is included so that both assignment and retrieval may be performed at the same time. This ensures the assignment has succeeded, and the returned value is not from some other frame.

Getters and Setters

The functions here can be classified whether the get, or set variables in the environment; some do both. Others make no modifications to the environment.

Getter:

- `get_adapt_history`
- `get_psd_env_name`
- `get_psd_env_pointer`
- `psd_envGet`
- `psd_envList`
- `psd_envStatus`

Setter:

- `new_adapt_history`
- `psd_envAssign`

Getter and Setter:

- `psd_envAssignGet`
- `psd_envClear`
- `psd_envRefresh`
- `update_adapt_history`

Adaptive History

The list object for historical adapt-data may be accessed with `get_adapt_history`. The top names of the returned list are

`stg_kopt` Sequential taper vectors.

`stg_psd` Sequential power spectral density vectors.

`freq` The frequencies for each set of `stg_kopt` and `stg_psd`.

Note

`psd_envClear` does *not* remove the environment–simply the assignments within it.

See Also

`psd-utilities`, `pspectrum`

Examples

```
#RDEX#\dontrun{
require(psd)
##
## psd working environment
##
# Get some status information about the psd working environment
```

```
psd_envStatus()
#
# Get a list of all variables
psd_envList()
#
# Pull the variable "init" into .GlobalEnv
print(x <- psd_envGet("init"))
#
# Pull the adaptive history into .GlobalEnv
get_adapt_history()
#RDEX#}
```

---

psd-normalization *Normalization of power spectral density estimates.*

Description

Normalize power spectral densities from various estimators into single-sided spectra.

Usage

```
normalize(Spec, Fsamp = 1, src = NULL, verbose = TRUE, ...)

## Default S3 method:
normalize(Spec, Fsamp = 1, src = NULL, verbose = TRUE,
  ...)

## S3 method for class 'list'
normalize(Spec, Fsamp = 1, src = NULL, verbose = TRUE, ...)

## S3 method for class 'spec'
normalize(Spec, Fsamp = 1, src = NULL, verbose = TRUE, ...)
```

Arguments

Spec	spectrum to normalize
Fsamp	sampling frequency
src	character string; the source of the spectrum estimator
verbose	logical; should messages be given?
...	(unused) additional parameters

Details

Normalizations commonly encountered for power spectra depend on it's assumed sidedness: whether the spectrum is either single- or double-sided. The normalizations performed here enforce single-sidedness, and correct as necessary.

Frequencies are assumed to be based on the Nyquist frequency (half the sampling rate). For example: If a series X has sampling frequency F_S, then the PSD frequencies will span $[0, F_S/2]$.

For amplitudes, improper normalization can can introduce errant factors of either 1/2 or F_S into the estimates, depending on the assumed sidedness. These factors can be accounted for with the `src` argument, which defaults to normalizing a double-sided spectrum.

Value

An object with its spectral values normalized accordingly.

Spectrum sidedness and the `src` argument

`"double.sided"` or `"spectrum"`:

These spectra assume frequency range of $[-F_S/2, F_S/2]$, and so are normalized by scaling by a factor of two upwards. Some estimators producing double-sided spectra:

- `stats::spectrum`
- `RSEIS::mtapspec`

`"single.sided"` or `"psd"`: As mentioned before, these spectra assume frequency range of $[0, F_S/2]$ and are scaled only by the inverse of the sampling rate. Some estimators producing single-sided spectra:

- psdcore

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

See Also

psdcore, spectral_properties

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Normalization
##
set.seed(1234)
# timeseries with sampling frequency **not** equal to 1:
X <- ts(rnorm(1e3), frequency=20)
# spec.pgram: double sided
pgram <- spectrum(X)
# psdcore: single sided
PSD <- psdcore(X)
# note the normalization differences:
plot(pgram, log="dB", ylim=c(-40,10))
plot(PSD, add=TRUE, col="red", log="dB")
# A crude representation of integrated spectrum:
#   should equal variance of white noise series (~= 1)
```

```
mean(pgram$spec)*max(pgram$freq)
mean(PSD$spec)*max(PSD$freq)
#
# normalize objects with class 'spec'
pgram <- normalize(pgram, src="spectrum")
PSD <- normalize(pgram, src="psd")
# replot them
plot(pgram, log="dB", ylim=c(-40,10))
plot(PSD, add=TRUE, col="red", log="dB")
#
# Again, integrated spectrum should be ~= 1:
mean(pgram$spec)*max(pgram$freq)
mean(PSD$spec)*max(PSD$freq)
#
#RDEX#}
```

---

psd-utilities *Various utility functions.*

Description

The various utility functions are:

vardiff returns the variance of the first (or second) difference of the series. varddiff is a convenience wrapper to return variance for the second difference.

dB returns an object converted to decibels.

char2envir converts a character string of an environment name to an evaluated name; whereas, envir2char converts an environment name to a character string.

vector_reshape reshapes a vector into another vector.

colvec returns the object as a vertically long vector; whereas rowvec returns the object as a horizontally long vector.

is.spec reports whether an object has class S3 class 'spec', as would one returned by, for example, spectrum.

is.tapers reports whether an object has S3 class 'tapers', as would one returned by, for example, as.tapers.

na_mat populates a matrix of specified dimensions with NA values.

zeros populate a column-wise matrix with zeros; whereas, ones populates a column-wise matrix with ones. *Note that* nrow *is enforced to be at least 1 for both functions.*

mod finds the modulo division of X and Y.

Usage

```
vardiff(Xd, double.diff = FALSE)

varddiff(Xd)
```

```
dB(Rat, invert = FALSE, pos.only = TRUE, is.power = FALSE)

char2envir(envchar)

envir2char(envir)

vector_reshape(x, vec.shape = c("horizontal", "vertical"))

## Default S3 method:
vector_reshape(x, vec.shape = c("horizontal", "vertical"))

colvec(x)

rowvec(x)

is.spec(Obj)

is.tapers(Obj)

na_mat(nrow, ncol = 1)

## Default S3 method:
na_mat(nrow, ncol = 1)

zeros(nrow)

## Default S3 method:
zeros(nrow)

ones(nrow)

## Default S3 method:
ones(nrow)

mod(X, Y)

## Default S3 method:
mod(X, Y)
```

Arguments

Xd	object to difference
double.diff	logical; should the double difference be used instead?
Rat	numeric; A ratio to convert to decibels (dB).
invert	logical; assumes Rat is already in decibels, so return ratio
pos.only	logical; if invert=FALSE, sets negative or zero values to NA
is.power	logical; should the factor of 2 be included in the decibel calculation?

envchar	An object with class 'character'.
envir	An object of class 'environment'.
x	An object to reshape (vector_reshape).
vec.shape	choice between horizontally-long or vertically-long vector.
Obj	An object to test for class inheritance.
nrow	integer; the number of rows to create.
ncol	integer; the number of columns to create (default 1).
X	numeric; the "numerator" of the modulo division
Y	numeric; the "denominator" of the modulo division

Details

Decibels are defined as $10 \log_{10} \frac{X_1}{X_2}$, unless `is.power=TRUE` in which $\mathrm{db}X^2 \equiv 20 \log_{10} X^2$

`colvec, rowvec` are simple wrapper functions to `vector_reshape`.

Modulo division has higher order-of-operations ranking than other arithmetic operations; hence, `x + 1 %% y` is equivalent to `x + (1 %% y)` which can produce confusing results. `mod` is simply a series of `trunc` commands which reduces the chance for unintentionally erroneous results.

Value

`char2envir` returns the result of evaluating the object: an environment object; `envir2char` returns the result of deparsing the environment name: a character string.

`vector_reshape` returns a "reshaped" vector, meaning it has had it's dimensions changes so that it has either one row (if `vec.shape=="horizontal"`), or one column (`"vertical"`).

`is.spec` and `is.tapers` both return logicals about whether or not the object does have class 'spec' or 'tapers', respectively

`na_mat` returns a matrix of dimensions `(nrow,ncol)` with `NA` values, the representation of which is set by `NA_real_`

For `zeros` or `ones` respectively, a matrix vector with `nrow` zeros or ones.

`mod` returns the result of a modulo division, which is equivalent to `(X) %% (Y)`.

Note

`char2envir` ensures the `envchar` object is a character, so that something is not unintentionally evaluated; `envir2char` simply deparses the object name.

The performance of `mod` has not been tested against the `%%` arithmetic method – it may or may not be slower for large numeric vectors.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

References

For mod: see Peter Dalgaard's explanation of the non-bug (#14771) I raised (instead I should've asked it on R-help): https://bugs.r-project.org/bugzilla3/show_bug.cgi?id=14771#c2

See Also

psd-package, as.tapers

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Various utilities
##
set.seed(1234)
X <- rnorm(1e2)
##
## Matrix and vector creation:
##
# NA matrix
nd <- 5
na_mat(nd)
na_mat(nd,nd-1)
# zeros
zeros(nd)
# and ones
ones(nd)
##
## Check for tapers object:
##
is.tapers(X)
is.tapers(as.tapers(X))
##
## Check for spec object:
##
PSD <- spectrum(X, plot=FALSE)
# return is class 'spec'
is.spec(PSD) # TRUE
# but the underlying structure is just a list
PSD <- unclass(PSD)
is.spec(PSD) # FALSE
##
## Environment character strings
##
print(envname <- get_psd_env_name())
print(envir <- char2envir(envname))
try(char2envir("some nonexistent environment"), silent=TRUE) # error
# and environment objects:
print(.GlobalEnv)
envir2char(.GlobalEnv)
envir2char(envir)
```

```
try(envir2char(some_nonexistent_environment), silent=TRUE) # error
##
## decibels
##
dB(1) # signal is equal <--> zero dB
sig <- 1e-10
all.equal(sig, dB(dB(sig), invert=TRUE))
pow <- sig**2
all.equal(pow, dB(dB(sig, is.power=TRUE), invert=TRUE, is.power=TRUE))
##
## Variance of difference series
##
vardiff(X)
varddiff(X)
all.equal(vardiff(X, TRUE), varddiff(X))
##
## modulo division
##
x <- 1:10
mc1a <- mod(1,2)
mc2a <- mod(1+x,2)
mc1b <- 1 %% 2
mc2b <- 1 + x %% 2
mc2c <- (1 + x) %% 2
all.equal(mc1a, mc1b) # TRUE
all.equal(mc2a, mc2b) # "Mean absolute difference: 2"
all.equal(mc2a, mc2c) # TRUE
##
#RDEX#}
```

---

psdcore *Multitaper power spectral density estimates of a series.*

Description

Compute power spectral density (PSD) estimates for the input series using sine multitapers.

Usage

```
psdcore(X.d, X.frq = NULL, ntaper = as.tapers(1), ndecimate = 1L,
  preproc = TRUE, na.action = stats::na.fail, first.last = TRUE,
  plotpsd = FALSE, as.spec = TRUE, refresh = FALSE, verbose = FALSE,
  ...)

## Default S3 method:
psdcore(X.d, X.frq = NULL, ntaper = as.tapers(1),
  ndecimate = 1L, preproc = TRUE, na.action = stats::na.fail,
  first.last = TRUE, plotpsd = FALSE, as.spec = TRUE, refresh = FALSE,
  verbose = FALSE, ...)
```

Arguments

X.d	the series to estimate a spectrum for
X.frq	scalar; the sampling information (see section Sampling)
ntaper	scalar, or vector; the number of tapers
ndecimate	scalar; decimation factor
preproc	logical; should X.d have a linear trend removed?
na.action	the function to deal with NA values
first.last	the extrapolates to give the zeroth and Nyquist frequency estimates
plotpsd	logical; should the estimate be shown compared to the spec.pgram estimate
as.spec	logical; should the object returned be of class 'spec'?
refresh	logical; ensure a free environment prior to execution
verbose	logical; should messages be given?
...	(unused) Optional parameters

Details

Tapering: The parameter ntaper specifies the number of sine tapers to be used at each frequency: equal tapers at each frequency for a scalar; otherwise, use ntaper[j] sine tapers at frequency[j].

Truncation: The series, with length N, is necessarily truncated so that $1+N/2$ evenly spaced frequencies are returned. This truncation makes the series length "highly composite", which the discrete Fourier transform (DFT) is most efficient. The vignette "fftw" (accessed with vignette("fftw",package="psd")) shows how the performance of a DFT can be affected by series length.

Decimation: The parameter ndecimate determines the number of PSD estimates actually computed. This number is defined as a fraction of the truncated length, $(1 + N/2)/n_d$. Linear interpolation is used.

Sampling: If X.frq is NULL, the value is assumed to be 1, unless X.d is a 'ts' object. If X.frq > 0 it's assumed the value represents *frequency* (e.g. Hz). If X.frq < 0 it's assumed the value represents *interval* (e.g. seconds).

Value

An list object, invisibly. If as.spec=TRUE then an object with class spec; otherwise the list object will have information similar to a spec object, but with a few additional fields.

Warning

Decimation is not well tested as of this point.

The first.last parameter is a workaround for potential bug (under investigation), which causes the power at the zero and Nyquist frequencies to have anomalously low values. This argument enables using linear *extrapolation* to correct these values. **The feature will be deprecated if the supposed bug is both identified and fixed.**

Author(s)

A.J. Barbour <andy.barbour@gmail.com> adapted original by R.L.Parker.

See Also

pspectrum, riedsid

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Multitaper PSD estimation
##
set.seed(1234)
X <- rnorm(1e3)
#
# use the defaults, and appeal to plot.spec
plot(psdcore(X))
#
# use more tapers, compare to stats::spectrum, and clear
# env data from the previous calculation
psdcore(X, ntapers=10, plotpsd=TRUE, refresh=TRUE)
#
# change the sampling frequency to 20
psdcore(X, 20, 10, plotpsd=TRUE, refresh=TRUE)
#RDEX#}
```

---

pspectrum *Adaptive sine multitaper power spectral density estimation.*

Description

This is the primary function to be used in this package, and returns power spectral density estimates where the number of tapers at each frequency has been iteratively optimized (`niter` times).

Usage

```
pspectrum(x, x.frqsamp = 1, ntap.init = 7, niter = 3, AR = FALSE,
  Nyquist.normalize = TRUE, verbose = TRUE, no.history = FALSE,
  plot = FALSE, ...)

## Default S3 method:
pspectrum(x, x.frqsamp = 1, ntap.init = 7, niter = 3,
  AR = FALSE, Nyquist.normalize = TRUE, verbose = TRUE,
  no.history = FALSE, plot = FALSE, ...)

## S3 method for class 'spec'
```

```
pspectrum(x, x.frqsamp = 1, ntap.init = 7, niter = 3,
  AR = FALSE, Nyquist.normalize = TRUE, verbose = TRUE,
  no.history = FALSE, plot = FALSE, ...)
```

Arguments

x	vector; series to estimate PSD for.
x.frqsamp	scalar; the sampling rate (e.g. Hz) of the series x.
ntap.init	scalar; the number of sine tapers to use in the pilot spectrum estimation.
niter	scalar; the number of adaptive iterations to execute after the pilot spectrum.
AR	logical; should the effects of an AR model be removed from the pilot spectrum?
Nyquist.normalize	logical; should the units be returned on Hz, rather than Nyquist?
verbose	logical; Should messages be given?
no.history	logical; Should the adaptive history *not* be saved?
plot	logical; Should the results be plotted?
...	Optional parameters passed to riedsid

Details

See the **Adaptive estimation** section in the description of the psd-package for details regarding adaptive estimation.

Value

Object with class 'spec', invisibly. It also assigns the object to "final_psd" in the working environment.

Author(s)

A.J. Barbour <andy.barbour@gmail.com> adapted original by R.L. Parker.

See Also

psdcore, pilot_spec, riedsid, prewhiten

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Adaptive multitaper PSD estimation
## (portions extracted from overview vignette)
##
require(RColorBrewer)
##
## adaptive estimation for the Project MAGNET dataset
data(magnet)
```

```
# adaptive psd estimation (turn off diagnostic plot)
PSDr <- pspectrum(Xr <- magnet$raw, plot=FALSE)
PSDc <- pspectrum(Xc <- magnet$clean, plot=FALSE)
# plot them on the same scale
plot(PSDc, log="dB", main="Raw and Clean Project MAGNET power spectral density",
     lwd=3, ci.col=NA, ylim=c(0,32), yaxs="i")
plot(PSDr, log="dB", add=TRUE, lwd=3, lty=5)
text(c(0.25,0.34), c(11,24), c("Clean","Raw"), cex=1)

## Change sampling, and inspect the diagnostic plot
pspectrum(Xc, niter=1, x.frqsamp=10)

## Say we forgot to assign the results: we can recover from the environment with:
PSDc_recovered <- psd:::psd_envGet("final_psd")
plot(PSDc_recovered)

##
## Visualize adaptive history
##
## Previous adaptive estimation history
pspectrum(Xc, niter=6, plot=FALSE)
AH <- get_adapt_history()
Freqs <- (AH$freq)
Dat <- AH$stg_psd
numd <- length(Freqs)
numit <- length(Dat)
StgPsd <- dB(matrix(unlist(Dat), ncol=numit))
Dat <- AH$stg_kopt
StgTap <- matrix(unlist(Dat), ncol=numit)
rm(Dat, AH)

## plot psd history
seqcols <- 1:numit
itseq <- seqcols - 1
toadd <- matrix(rep(itseq, numd), ncol=numit, byrow=TRUE)
par(xpd=TRUE)
matplot(Freqs, StgPsd + (sc<-9)*toadd, type="l", lty=1, lwd=2, col="black",
            main="PSD estimation history", ylab="", xlab="Spatial frequency",
            yaxt="n", frame.plot=FALSE)
text(.52, 1.05*sc*itseq, itseq)
text(.49, 1.1*sc*numit, "Stage:")

## plot taper history "mountain range" silhouettes
par(xpd=TRUE)
Cols <- rev(rev(brewer.pal(9, "PuBuGn"))[seqcols])
invisible(lapply(rev(seqcols), FUN=function(mcol, niter=numit, Frq=Freqs, Dat=StgTap, cols=Cols){
  iter <- (niter+1)-mcol
  y <- Dat[,mcol]
  icol <- Cols[mcol]
  if (iter==1){
    plot(Frq, y, type="h", col=icol,
           main="Taper optimization history", ylab="", xlab="Spatial frequency",
           ylim=c(-50,650), frame.plot=FALSE)
```

```
  } else {
    lines(Frq, y, type="h", col=icol)
  }
  lines(Frq, y, type="l",  lwd=1.2)
  x <- (c(0,1)+iter-1)*.05+0.075
  y <- c(595,595,650,650,595)+10
  text(mean(x),max(y)+1.0*diff(range(y)), mcol-1)
  polygon(c(x,rev(x),x[1]),y,border="black",col=icol)
}
)) # end of invisible lapply
#RDEX#}
```

`riedsid` *Constrained, optimal tapers using the Riedel & Sidorenko–Parker method.*

Description

Estimates the optimal number of tapers at each frequency of given PSD, using a modified Riedel-Sidorenko MSE recipe (RS-RLP).

Usage

```
riedsid(PSD, ntaper, tapseq = NULL, Deriv.method = c("local_qls", "spg"),
  Local.loss = c("Optim", "Less", "More"), constrained = TRUE,
  c.method = NULL, verbose = TRUE, ...)

## S3 method for class 'spec'
riedsid(PSD, ...)

## Default S3 method:
riedsid(PSD, ntaper, tapseq = NULL,
  Deriv.method = c("local_qls", "spg"), Local.loss = c("Optim", "Less",
  "More"), constrained = TRUE, c.method = NULL, verbose = TRUE, ...)
```

Arguments

`PSD`	vector or class 'spec'; the spectral values used to optimize taper numbers
`ntaper`	scalar or vector; number of tapers to apply optimization
`tapseq`	vector; representing positions or frequencies (same length as `PSD`)
`Deriv.method`	character string; choice of gradient estimation method
`Local.loss`	string; sets how sensitive the spectral derivatives are
`constrained`	logical; should the taper constraints be applied to the optimum tapers?
`c.method`	string; constraint method to use if `constrained=TRUE`
`verbose`	logical; should messages be printed?
`...`	optional argments passed to `constrain_tapers`

Details

The optimization is as follows. First, weighted derivatives of the input PSD are computed. Using those derivates the optimal number of tapers is found through the RS-RLP formulation. Constraints are then placed on the practicable number of tapers.

Taper constraints: The parameter `c.method` provides an option to change the method of taper constraints. A description of each may be found in the documentation for `constrain_tapers`.

Once can use `constrained=FALSE` to turn off all taper constraints; this could lead to strange behavior though.

Spectral derivatives: The parameter `Deriv.method` determines which method is used to estimate derivatives.

- `"local_qls"` (**default**) uses quadratic weighting and local least-squares estimation; then, `Local.loss` can alter slightly the weighting to make the derivatives more or less succeptible to changes in spectral values. Can be slower than `"spg"`.
- `"spg"` uses `splineGrad`; then, additional arguments may be passed to control the smoothness of the derivatives (e.g `spar` in `smooth.spline`).

Value

Object with class 'tapers'.

Warning

The `"spg"` can become numerically unstable, and it's not clean when it will be preferred to the `"local_qls"` method other than for efficiency's sake.

Author(s)

A.J. Barbour <andy.barbour@gmail.com> adapted original by R.L. Parker.

See Also

`constrain_tapers`, `psdcore`, `smooth.spline`

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Riedel-Sidorenko--Parker taper optimization
##
set.seed(1234)
# some params
nd <- 512 # num data
ntap <- 10 # num tapers
nrm <- 40 # sharpness of the peaks rel 2*variance
#
# create a pseudo spectrum
# with broad peaks
```

```
riex <- rnorm(nd) + nrm*abs(cos(pi*(x<-0:(nd-1))/180) + 1.2)
riex <- riex + 8*nrm*dcauchy(x, nd/3)
riex <- riex + 5*nrm*dnorm(x, nd/2)
# flat regions
riex[riex<25] <- 25
ried <- dB(riex, invert=TRUE)
#
# optimize tapers
rtap <- riedsid(riex, ntaper=ntap)
#
# plot
op <- par(no.readonly = TRUE)
par(mfrow=c(2,1), mar=rep(1.3,4), mai=rep(0.6,4))
# ... the mock spectrum
plot(riex, type="h", xaxs="i", ylim=c(0,200))
# ... the optimal tapers
plot(rtap, log="y")
# original tapers:
lines(as.tapers(rep.int(ntap,nd)), col="blue")
par(op)
#RDEX#}
```

`spec-methods` *Generic methods for objects with class* `'spec'`.

Description

Generic methods for objects with class `'spec'`.

Usage

```
## S3 method for class 'spec'
as.data.frame(x, ...)

## S3 method for class 'spec'
data.frame(x, ...)
```

Arguments

`x`	spec object
`...`	optional arguments

Details

Objects with class `'spec'` are simply list objects. `as.data.frame` converts the list into a `'data.frame'` with individual columns for the frequency, PSD, and taper vectors; all other information will be retained as an attribute. `data.frame` is an alias.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Objects with class 'spec'
##
set.seed(1234)
#
x <- spectrum(xn<-rnorm(10), plot=FALSE)
xdf <-as.data.frame(x)
str(xdf)
is.tapers(xdf$taper)
#
# tapers class is retained
#
x <- psdcore(xn)
xdf <- as.data.frame(x)
str(xdf)
is.tapers(xdf$taper)
#RDEX#}
```

---

spectral_properties *Calculate spectral properties.*

Description

Various spectral properties may be computed from the vector of tapers, and if necessary the sampling frequency.

Usage

```
spectral_properties(tapvec, f.samp = 1, n.freq = NULL, p = 0.95,
  db.ci = FALSE, ...)

## S3 method for class 'spec'
spectral_properties(tapvec, ...)

## S3 method for class 'tapers'
spectral_properties(tapvec, f.samp = 1, n.freq = NULL,
  p = 0.95, db.ci = FALSE, ...)
```

Arguments

tapvec	object with class tapers or spec
f.samp	scalar; the sampling frequency (e.g. Hz) of the series the tapers are for
n.freq	scalar; the number of frequencies of the original spectrum (if NULL the length of the tapers object is assumed to be the number)
p	numeric; the coverage probability, bound within $[0, 1)$
db.ci	logical; should the uncertainty confidence intervals be returned as decibels?
...	additional arguments (unused)

Value

A list with the following properties (and names):

- taper: The original taper vector.
- stderr.chi .upper, .lower, .median: results returned from spec_confint.
- resolution: The effective spectral resolution.
- dof: The number of degrees of freedom.
- bw: The effective bandwidth of the spectrum.

Parameter Details

Uncertainty: See spec_confint for details.

Resolution: The frequency resolution depends on the number of tapers (K), and is found from

$$\frac{K \cdot f_N}{N_f}$$

where f_N is the Nyquist frequency and N_f is the number of frequencies estimated.

Degrees of Freedom: There are two degrees of freedom for each taper K:

$$\nu = 2K$$

Bandwidth: The bandwidth of a multitaper estimate depends on the number of tapers. Following Walden et al (1995) the effective bandwidth is $\approx 2W$ where

$$W = \frac{K+1}{2N}$$

and N is the number of terms in the series, which makes $N \cdot W$ the approximate time-bandwidth product.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

References

Prieto, G. A., R. L. Parker, D. J. Thomson, F. L. Vernon, and R. L. Graham (2007), Reducing the bias of multitaper spectrum estimates, *Geophysical Journal International*, **171**, 1269–1281, doi: 10.1111/j.1365-246X.2007.03592.x

See Also

spec_confint, psd-package

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Spectral properties from the number of tapers used
## (portions extracted from overview vignette)
##

##
## Theoretical uncertainties from Chi^2 distribution
##
sp <- spectral_properties(as.tapers(1:50), p=0.95, db.ci=TRUE)
par(las=1)
plot(stderr.chi.upper ~ taper, sp, type="s",
       ylim=c(-10,20), yaxs="i", xaxs="i",
       xlab=expression("number of tapers ("* nu/2 *")"), ylab="dB",
       main="Spectral uncertainties")
mtext("(additive factor)", line=.3)
lines(stderr.chi.lower ~ taper, sp, type="s")
lines(stderr.chi.median ~ taper, sp, type="s", lwd=2)
lines(stderr.chi.approx ~ taper, sp, type="s", col="red",lwd=2)
# to reach 3 db width confidence interval at p=.95
abline(v=33, lty=3)
legend("topright",
        c(expression("Based on "* chi^2 *"(p,"*nu*") and (1-p,"*nu*")"),
          expression(""* chi^2 *"(p=0.5,"*nu*")"),
          "approximation"),
lwd=c(1,3,3), col=c("black","black","red"), bg="white")

##
## An example using the Project MAGNET dataset
##
data(magnet)
tapinit <- 15 # tapers
dt <- 1 # 1/km

# remove mean/trend (not really necessary but good practice; also, done internally)
ats <- prewhiten(ts(magnet$clean, deltat=dt), plot=FALSE)$prew_lm

# normal and adaptive multitaper spectra
Pspec <- psdcore(ats, dt, tapinit)
Aspec <- pspectrum(ats, dt, tapinit, niter=3, plot=FALSE)
```

```
# calculate spectral properties
spp <- spectral_properties(Pspec$taper, db.ci=TRUE)
spa <- spectral_properties(Aspec$taper, db.ci=TRUE)

# function to create polygon data, and create them
create_poly <- function(x, y, dy){
  xx <- c(x, rev(x))
  yy <- c(y+dy, rev(y-dy))
  return(data.frame(xx=xx, yy=yy))
}
pspp <- create_poly(Pspec$freq, dB(Pspec$spec), spp$stderr.chi.approx)
psppu <- create_poly(Pspec$freq, dB(Pspec$spec), spp$stderr.chi.upper)
pspa <- create_poly(Aspec$freq, dB(Aspec$spec), spa$stderr.chi.approx)
pspau <- create_poly(Aspec$freq, dB(Aspec$spec), spa$stderr.chi.upper)

##
## Project MAGNET uncertainties
##
plot(c(0,0.5),c(-8,35),col="white",
       main="Project MAGNET Spectral Uncertainty (p > 0.95)",
       ylab="", xlab="spatial frequency, 1/km", yaxt="n", frame.plot=FALSE)
lines(c(2,1,1,2)*0.01,c(5,5,8.01,8.01)-8)
text(.05, -1.4, "3.01 dB")
polygon(psppu$xx, (psppu$yy), col="light grey", border="black", lwd=0.5)
polygon(pspp$xx, (pspp$yy), col="dark grey", border=NA)
text(0.15, 6, "With adaptive\ntaper refinement", cex=1.2)
polygon(pspau$xx, (pspau$yy)-10, col="light grey", border="black", lwd=0.5)
polygon(pspa$xx, (pspa$yy)-10, col="dark grey", border=NA)
text(0.35, 22, "Uniform tapering", cex=1.2)

##
## Project MAGNET resolution
##
frq <- Aspec$freq
relp <- dB(1/spa$resolution)
par(las=1)
plot(frq, relp,
     col="light grey",
     ylim=dB(c(1,5)),
     type="h", xaxs="i", yaxs="i",
     ylab="dB", xlab="frequency, 1/km",
     main="Project MAGNET Spectral Resolution and Uncertainty")
lines(frq, relp)
lines(frq, spp$stderr.chi.upper+relp, lwd=1.5, lty=3)
lines(frq, spa$stderr.chi.upper+relp, lwd=3, lty=2)
abline(h=dB(sqrt(vardiff(Aspec$spec))), lwd=1.5, lty=2, col="red")

##
#RDEX#}
```

spec_confint *Multitaper PSD confidence intervals.*

Description

Multitaper PSD confidence intervals.

Usage

```
spec_confint(dof, p = 0.95, as.db = FALSE)

## S3 method for class 'spec'
spec_confint(dof, p = 0.95, as.db = FALSE)

## S3 method for class 'tapers'
spec_confint(dof, p = 0.95, as.db = FALSE)

## Default S3 method:
spec_confint(dof, p = 0.95, as.db = FALSE)
```

Arguments

dof	numeric; the degrees of freedom ν
p	numeric; the coverage probability p, bound within $[0, 1)$
as.db	logical; should the values be returned as decibels?

Details

The errors are estimated from the number of degrees of freedom ν by evaluating the $\chi^2_{p,\nu}(\nu, \nu)$ distribution for an optional coverage probability p (defaulting to $p = 0.95$). Additionally, the $p = 0.5$ values and an approximation from $1/\sqrt{\nu - 1}$ are returned.

A more sophisticated (and complicated) approach would be to estimate via jack-knifing (Prieto et al 2007), but this is not yet made available.

Additive uncertainties δS are returned, such that the spectrum with confidence interval is $S \pm \delta S$.

Value

A data.frame with the following properties (and names):

- lower: Based on upper tail probabilities (p)
- upper: Based on lower tail probabilities ($1 - p$)
- median: Based on lower tail probabilities ($p = 0.5$)
- approx: Approximation based on $1/\sqrt{(\nu - 1)}$.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>, modified from the `spec.ci` function inside `stats::plot.spec`.

See Also

spectral_properties, psd-package, `plot.spec`, dB

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Confidence intervals from taper numbers
##
sp <- spectral_properties(as.tapers(1:50), p=0.95, db.ci=TRUE)
par(las=1)
plot(stderr.chi.upper ~ taper, sp, type="s",
       ylim=c(-10,20), yaxs="i", xaxs="i",
       xlab=expression("number of tapers ("* nu/2 *")"), ylab="dB",
       main="Spectral uncertainties")
mtext("(additive factor)", line=.3)
lines(stderr.chi.lower ~ taper, sp, type="s")
lines(stderr.chi.median ~ taper, sp, type="s", lwd=2)
lines(stderr.chi.approx ~ taper, sp, type="s", col="red",lwd=2)
# to reach 3 db width confidence interval at p=.95
abline(v=33, lty=3)
legend("topright",
        c(expression("Based on "* chi^2 *"(p,"*nu*") and (1-p,"*nu*")"),
          expression(""* chi^2 *"(p=0.5,"*nu*")"),
          "approximation"),
lwd=c(1,3,3), col=c("black","black","red"), bg="white")
##
#RDEX#}
```

`splineGrad` *Numerical derivatives of a series based on a weighted, smooth spline representation.*

Description

`splineGrad` computes the numerical derivatives of a spline representation of the input series; differentiation of spline curves is numerically efficient.

With smoothing, the numerical instability for "noisy" data can be drastically reduced, since spline curves are inherently (at least) twice differentiable. See the **Examples** for an illustration of this.

Usage

```
splineGrad(dseq, dsig, plot.derivs = FALSE, ...)

## Default S3 method:
splineGrad(dseq, dsig, plot.derivs = FALSE, ...)
```

Arguments

dseq	numeric; a vector of positions for dsig.
dsig	numeric; a vector of values (which will have a spline fit to them).
plot.derivs	logical; should the derivatives be plotted?
...	additional arguments passed to smooth.spline.

Value

A matrix with columns representing $x, f(x), f'(x), f''(x)$.

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

See Also

smooth.spline

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Spline gradient
##
set.seed(1234)
x <- seq(0,5*pi,by=pi/64)
y <- cos(x) #**2
splineGrad(x, y, TRUE)
y <- y + rnorm(length(y), sd=.1)
# unfortunately, the presence of
# noise will affect numerical derivatives
splineGrad(x, y, TRUE)
# so change the smoothing used in smooth.spline
splineGrad(x, y, TRUE, spar=0.2)
splineGrad(x, y, TRUE, spar=0.6)
splineGrad(x, y, TRUE, spar=1.0)
##
#RDEX#}
```

tapers-constraints *Taper constraint methods.*

Description

In the Riedel-Sidorenko recipe, the number of optimal tapers at each frequency is strongly dependent on the first and second derivatives of the spectrum. It is crucial to enforce constraints on the number of actual tapers applied; this is because the derivatives of "noisy" series can be bogus.

`minspan` sets the maximum span a tapers object may have, which is necessary because it would be nonsense to have more tapers than the length of the series.

`constrain_tapers` refines the number of tapers; the method by which it does this is chosen with the `constraint.method` parameter. See **Constraint methods** section for descriptions of each method. Below is a summary of the function associated with each `constraint.method`:

- `'simple.slope'` uses `ctap_simple`
- `'loess.smooth'` uses `ctap_loess`
- `'none'` returns unbounded tapers.

Usage

```
minspan(tapvec, ...)

## S3 method for class 'tapers'
minspan(tapvec, ...)

constrain_tapers(tapvec, tapseq = NULL,
  constraint.method = c("simple.slope", "loess.smooth", "none"),
  verbose = TRUE, ...)

## S3 method for class 'tapers'
constrain_tapers(tapvec, tapseq = NULL,
  constraint.method = c("simple.slope", "loess.smooth", "none"),
  verbose = TRUE, ...)

ctap_simple(tapvec, tapseq = NA, maxslope = 1, ...)

## S3 method for class 'tapers'
ctap_simple(tapvec, tapseq = NA, maxslope = 1, ...)

ctap_loess(tapvec, tapseq = NULL, loess.span = 0.3, loess.degree = 1,
  verbose = TRUE, ...)

## S3 method for class 'tapers'
ctap_loess(tapvec, tapseq = NULL, loess.span = 0.3,
  loess.degree = 1, verbose = TRUE, ...)
```

```
ctap_markov()

## S3 method for class 'tapers'
ctap_markov()

ctap_friedman()

## S3 method for class 'tapers'
ctap_friedman()
```

Arguments

`tapvec`	`'tapers'` object; the number of tapers at each frequency
`tapseq`	vector; positions or frequencies – necessary for smoother methods
`constraint.method`	character; method to use for constraints on tapers numbers
`verbose`	logical; should warnings and messages be given?
`maxslope`	integer; constrain based on this maximum first difference
`loess.span`	scalar; the span used in `loess`
`loess.degree`	scalar; the polynomial degree
`...`	optional arguments (unused)

Details

`minspan` bounds the number of tapers to within the minimum of either the maximum number of tapers found in the object, or the half-length of the series.

Value

An object with class `'tapers'`.

Details of Constraint Methods

via first differencing (the default): `ctap_simple` is the preferred constraint method. The algortihm uses first-differencing to modify the number of tapers in the previous position. Effectively, the constraint is based on a causal, 1st-order Finite Impulse-response Filter (FIR) which makes the method sensitive to rapid changes in the number of tapers; naturally, smoother spectra tend to produce less fluctuation in taper numbers, which makes this well suited for adaptive processing.

This produces, generally, the most stable results, meaning repeatedly running the constraint will not change values other than on the first execution; the same cannot be said for the other methods.

In pure-R this algorithm can be very slow; however, here we have included it as dynamically loaded c-code so it it reasonably fast.

via LOESS smoothing: `ctap_loess` uses `loess` to smooth the taper vector; is can be very slow thanks to quadratic scaling.

Warning

ctap_loess results tend to be strongly dependent on the tuning parameters given to loess (for obvious reasons); hence, some effort should be given to understand their effect, and/or re-tuning them if needed.

Author(s)

A.J. Barbour <andy.barbour@gmail.com> and R.L.Parker. AJB adapted some of RLP's original code, and wrote the main function in ctap_simple for dynamic loading C-code.

See Also

splineGrad, riedsid

Examples

```
#RDEX#\dontrun{
require(psd)
##
## Taper constraint procedures
##
data(magnet)
X <- magnet$clean
##
## spectrum, then riedsid
kopt <- riedsid(PSD <- psdcore(X, ntaper=10, refresh=TRUE))
kopt.loess  <- riedsid(PSD, c.method="loess.smooth")
#
plot(kopt, log="y", ylim =c(.1, 3e2))
lines(kopt.loess, log="y", col="green")
##
##
## To compare all the methods at once:
demo("ctap")
##
#RDEX#}
```

---

tapers-methods *Generic methods for objects with class 'tapers'.*

Description

Generic methods for objects with class 'tapers'.

Usage

```
## S3 method for class 'tapers'
as.data.frame(x, ...)

## S3 method for class 'tapers'
data.frame(x, ...)

## S3 method for class 'tapers'
print(x, ...)

## S3 method for class 'tapers'
summary(object, ...)

## S3 method for class 'summary.tapers'
print(x, ...)

## S3 method for class 'tapers'
lines(x, lwd = 1.8, col = "red", ...)

## S3 method for class 'tapers'
points(x, pch = "_", cex = 1, ...)

## S3 method for class 'tapers'
plot(x, xi = NULL, color.pal = c("Blues", "Spectral"),
  ylim = NULL, hv.lines = FALSE, ...)
```

Arguments

x	tapers object
xi	optional vector for indices of x
object	tapers object
lwd	line width (default is 1.8)
col	color of line (default is "red")
pch	point character (default is "_")
cex	point size (default is 1)
color.pal	color palette to use (choices are: "Blues","Spectral")
ylim	optional limits for y-axis
hv.lines	logical; should horizontal and vertival reference lines be plotted?
...	optional arguments

Value

plot returns a list with names: line.colors (hex values)

Author(s)

A.J. Barbour <andy.barbour@gmail.com>

See Also

as.tapers, constrain_tapers, par

Examples

```
##
tap <- as.tapers(c(1:49,50:0)+rnorm(1e2))
print(tap)
print(summary(tap))
plot(tap)
# no arithmetic methods
tap <- as.tapers(tap/2)
lines(tap)
```

Tohoku *Observations of teleseismic strains from the 2011 Tohoku earthquake.*

Description

The $M_w 9$ Tohoku earthquake happend on March 11, 2011. The seismic waves were recorded at stations across the globe, including by strainmeters in the Plate Boundary Observatory (PBO) borehole strainmeters.

Format

A dataframe with 16000 observations on the following 15 variables.

`Dts` The original datetime string, in UTC.

`areal` Areal strains

`areal.tide` Tidal correction to the areal strains.

`areal.baro` Barometric correction to the areal strains.

`gamma1` Engineering differential extensional strain: γ_1

`gamma1.tide` Tidal correction for the γ_1 strains.

`gamma1.baro` Barometric pressure correction to the γ_1 strains.

`gamma2` Engineering shear strain: γ_2.

`gamma2.tide` Tidal correction for the γ_2 strains.

`gamma2.baro` Barometric pressure correction to the γ_2 strains.

`pressure.atm` Atmospheric pressure.

`pressure.pore` Pore-fluid pressure.

`Dt` The `Dts` information converted to POSIX datetime.

`Origin.secs` The number of seconds relative to the earthquake-origin time.

`epoch` Classification based on predicted P-wave arrival: preseismic or seismic.

and 2 attributes:

`units` A list of strings regarding the units of various physical quantities given here.

`iasp` A list of source and station characteristics, including the the origin time, predicted traveltimes for P and S waves, and the geodetic information used in the traveltime calculation.

Details

These data are for station B084, which is located approximately 8500 km away from the epicenter. Because this distance is large, the seismic waves didn't arrive at this station for more than 700 seconds after the origin time. So there is a record of pre-seismic noise included, the timeseries extends 6784 seconds prior to the origin time, and 9215 seconds after.

The data are classified with the `"epoch"` variable, which separates the series into pre-seismic and seismic data; this is defined relative to the predicted P-wave arrival time from a traveltime model.

The original dataset contained `NA` values, which were imputed using `zoo::na.locf`, which fills `NA` with the last previous observation.

Source

PBO High Frequency archive:

http://borehole.unavco.org/bsm/earthquakes/NeartheEastCoastofHonshuJapan_20110311

References

USGS summary page:

http://earthquake.usgs.gov/earthquakes/eqinthenews/2011/usc0001xgp/

See Also

`pspectrum`, `hfsnm`, `magnet`

`TauP.R` for an R-implementation of the traveltime calculations:

http://cran.r-project.org/web/packages/TauP.R/

Examples

```
data(Tohoku)
str(Tohoku)
```

Index