Package ‘bspec’

April 30, 2015

Type Package
Title Bayesian Spectral Inference
Version 1.5
Date 2015-04-23
Author Christian Roever
Maintainer Christian Roever <christian.roever@med.uni-goettingen.de>
Description Bayesian inference on the (discrete) power spectrum of time series.
License GPL (>= 2)
NeedsCompilation no
Repository CRAN
Date/Publication 2015-04-30 09:06:49

R topics documented:

bspec-package .................................................. 2
acf.bspec .......................................................... 2
bspec ............................................................. 4
empiricalSpectrum .............................................. 7
expectation ...................................................... 9
likelihood ....................................................... 10
matchedfilter .................................................. 12
one.sided ....................................................... 15
ppsample ......................................................... 17
quantile.bspec ................................................ 18
sample.bspec .................................................. 19
snr ............................................................... 20
temper .......................................................... 21
temperature .................................................... 23
tukeywindow .................................................... 24
welchPSD ........................................................ 25

Index 28
**bspec-package**  
*Bayesian Spectral Inference*

**Description**

This package functions to derive posterior distributions of spectral parameters of a time series.

**Details**

<table>
<thead>
<tr>
<th>Package:</th>
<th>bspec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>1.5</td>
</tr>
<tr>
<td>Date:</td>
<td>2015-04-23</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;=2)</td>
</tr>
</tbody>
</table>

The main functionality is provided by the `bspec()` function.

**Author(s)**

Christian Roever <christian.roever@med.uni-goettingen.de>

**References**


---

**acf.bspec**  
*Posterior autocovariances*

**Description**

Deriving (posterior) autocovariances or autocorrelations from the spectrum’s posterior distribution.

**Usage**

```r
## S3 method for class 'bspec'
acf(x, spec = NULL,
    type = c("covariance", "correlation"),
    two.sided = x$two.sided, ...)
```
acf.bspec

Arguments

x          a bspec object.
spec       (optional) a numeric vector giving fixed values of the spectral parameters (e.g. derived by the `sample` function) for which the autocovariances then are deterministic.
type       a character string specifying the desired type of output.
two.sided  a logical flag indicating whether the spec values are to be interpreted as one-sided or two-sided.
...

Details

If `spec` is supplied, the autocovariance (or autocorrelation) function corresponding to that specific spectrum will be returned. As this is a completely deterministic relationship, the “stderr” slot of the result will be zero in this case.

If `spec` is not supplied, the (posterior) expected autocovariance is returned in the “acf” element, and its (posterior) standard deviation is returned in the “stderr” element. The posterior expectation of the autocovariance is only finite if all (!) posterior degrees-of-freedom parameters in the bspec object are > 2. The posterior variance (and with that the stderr element) is only finite if all these are > 4.

Autocorrelations are only returned if `spec` is supplied.

Value

A list of class `bspecACF` containing the following components:

1. lag           a numeric vector giving the lags corresponding to the (discrete) autocovariance / autocorrelation values.
2. acf           a numeric vector giving the values of the autocovariance / autocorrelation function corresponding to the above lags.
3. stderr        a numeric vector giving the standard errors (posterior standard deviations) of the above autocovariance values.
4. type          a character string giving the nature of the above acf element: either "covariance" or "correlation".
5. N             an integer giving the sample size of the original time series.
6. bspec         a character string giving the name of the bspec object the bspecACF object was generated from.

Note

(Posterior) expectation and standard deviation of the spectrum may in many cases not be finite (see above). Autocorrelations are only returned if `spec` is supplied.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>
References


See Also

bspec, expectation, sample.bspec, acf

Examples

lhspec1 <- bspec(lh)

# without any prior specifications,
# autocovariances are not finite:
print(acf(lhspec1))
str(acf(lhspec1))

# for given values of the spectral parameters,
# the autocovariances are fixed:
str(acf(lhspec1, spec=sample(lhspec1)))

# for all the prior degrees-of-freedom greater than one,
# the expected autocovariance is finite, its variance isn't:
lhspec2 <- bspec(lh, priordf=2, priorscale=0.6, intercept=FALSE)
print(acf(lhspec2))
str(acf(lhspec2))
plot(acf(lhspec2))

bspec

Computing the spectrum’s posterior distribution

Description

Derives the posterior distribution of the spectrum of one or several time series, based on data and prior specifications.

Usage

bspec(x, ...)
  ## Default S3 method:
bspec(x, priorscale=1, priordf=0, intercept=TRUE,
   two.sided=FALSE, ...)

bspec

Arguments

x

da time series object of the data to be analysed. May be a univariate (ts object) or multivariate (mts object) time series.

priorscale

either a numeric vector giving the scale parameters of the spectrum’s prior distribution; recycled if of length 1.
Or a function of frequency.

priordf

either a numeric vector giving the degrees-of-freedom parameters of the spectrum’s prior distribution; recycled if of length 1.
Or a function of frequency.

intercept

da logical flag indicating whether to include the ‘intercept’ (zero frequency) term.

two.sided

da logical flag indicating whether to refer to a one-sided or a two-sided spectrum. In particular affects the interpretation of the prior scale parameters, and sets the default for some methods applied to the resulting bspec object via its two.sided element.

... currently unused.

Details

Based on the assumptions of a zero mean and a finite spectrum, the posterior distribution of the (discrete) spectrum is derived. The data are modeled using the Maximum Entropy (Normal) distribution for the above constraints, and based on the prior information about the spectrum specified in terms of the (conjugate) scaled inverse $\chi^2$ distribution.

For more details, see the references.

Value

A list of class bspec containing the following elements:

freq

da numeric vector giving the (Fourier-) frequencies that the spectral parameters correspond to.

scale

da numeric vector giving the scale parameters of the posterior distributions of the spectral parameters corresponding to the above frequencies. These -internally-always correspond to the one-sided spectrum, regardless of the two.sided flag (see below).

df

da numeric vector giving the degrees-of-freedom parameters of the posterior distributions of the spectral parameters corresponding to the above frequencies.

priorscale

da numeric vector giving the prior scale parameters.

priordf

da numeric vector giving the prior degrees-of-freedom parameters.

datassq

da numeric vector giving the sum-of-squares contributed by the data.

datadf

da numeric vector giving the degrees-of-freedom contributed by the data.

N

the sample size of the original time series.

deltat

the sampling interval of the original time series.

deltaf

the frequency interval of the Fourier-transformed time series.
bspec

start

call

two.sided

the time of the first observation in the original time series.
an object of class call giving the function call that generated the bspec object.
a logical flag indicating whether the spectrum is to be interpreted as one-sided or two-sided.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

References


See Also

expectation, quantile.bspec, sample.bspec, ppsample, acf.bspec, spectrum

Examples

# determine spectrum's posterior distribution
# (for noninformative prior):
lspec <- bspec(lh)
print(lhspec)

# show some more details:
str(lhspec)

# plot 95 percent central intervals and medians:
plot(lhspec)

# draw and plot a sample from posterior distribution:
lines(lhspec$freq, sample(lhspec), type="b", pch=20)

# compare the default outputs of "bspec()" and "spectrum()":
bspec1 <- bspec(lh)
spectrum1 <- spectrum(lh, plot=FALSE)
plot(bspec1)
lines(spectrum1$freq, spectrum1$spec, col="blue")
# (note -among others- the factor 2 difference)

# match the outputs:
# Need to suppress tapering, padding and de-trending
empiricalSpectrum

Computes the "empirical" spectrum of a time series.

Description

Computes the "empirical power" of a time series via a discrete Fourier transform.

Usage

empiricalSpectrum(x, two.sided=FALSE)

Arguments

x
  a time series (ts object).

two.sided
  a logical flag indicating whether the output is supposed to correspond to the
  one-sided (default) or two-sided spectrum.

Details

Performs a Fourier transform, and then derives (based on the additional information on sampling
rate etc. provided via the time series' attributes) the spectral power as a function of frequency. The
result is simpler (in a way) than the `spectrum()` function’s output, see also the example below. What is returned is the real-valued frequency series

\[ \kappa_j \frac{\Delta t}{N} |\hat{x}(f_j)|^2 \]

where \( j = 0, ..., \frac{N}{2} + 1 \), and \( f_j = \frac{j}{N \Delta t} \) are the Fourier frequencies. \( \Delta t \) is the time series’ sampling interval and \( N \) is its length. \( \kappa_j \) is \( =1 \) for zero and Nyquist frequencies, and \( =2 \) otherwise, and denotes the number of (by definition) non-zero Fourier coefficients. In case `two.sided=TRUE`, the \( \kappa_j \) prefactor is omitted.

For actual spectral estimation purposes, the use of a windowing function (see e.g. the `tukeywindow()` function) is highly recommended.

**Value**

A list containing the following elements:

- `frequency` the Fourier frequencies.
- `power` the spectral power.
- `kappa` the number of (by definition) non-zero imaginary components of the Fourier series.
- `two.sided` a logical flag indicating one- or two-sidedness.

**Author(s)**

Christian Roever, <christian.roever@med.uni-goettingen.de>

**References**


**See Also**

`fft`, `spectrum`, `tukeywindow`, `welchPSD`

**Examples**

```
# load example data:
data(lh)

# compute spectrum:
spec1 <- empiricalSpectrum(lh)
plot(spec1$frequency, spec1$power, log="y", type="b")

# plot "spectrum()" function's result in comparison:
spec2 <- spectrum(lh, plot=FALSE)
lines(spec2$freq, spec2$spec, col="red")
```
# make both spectra match:
spec3 <- empiricalSpectrum(lh, two.sided=TRUE)
spec4 <- spectrum(lh, plot=FALSE, taper=0, fast=FALSE, detrend=FALSE)
plot(spec3$frequency, spec3$power, log="y", type="b")
lines(spec4$freq, spec4$spec, col="green")

## expectation

### Expectations and variances of distributions

**Description**

Functions to compute (posterior) expectations or variances of the distributions specified as arguments.

**Usage**

```r
expectation(x, ...)
variance(x, ...)
## S3 method for class 'bspec'
expectation(x, two.sided=x$two.sided, ...)
## S3 method for class 'bspec'
variance(x, two.sided=x$two.sided, ...)
## S3 method for class 'bspecACF'
expectation(x, ...)
## S3 method for class 'bspecACF'
variance(x, ...)
```

**Arguments**

- `x` A `bspec` or `bspecACF` object.
- `two.sided` A logical flag to indicate whether to compute expectation / variance of one- or two-sided spectrum, if the argument `x` is a `bspec` object.
- `...` currently unused.

**Value**

A numeric vector giving the expectations/variances corresponding to the frequencies or lags of the argument.

**Author(s)**

Christian Roever, <christian.roever@med.uni-goettingen.de>

**References**

See Also

bspec, acf.bspec

Examples

# note the changing expectation
# with increasing prior/posterior degrees-of-freedom:
evaluation(bspec(lh))
evaluation(bspec(lh, priordf=1, priorscale=0.6))
evaluation(bspec(lh, priordf=2, priorscale=0.6))

# similar for variance:
variance(bspec(lh, priordf=2, priorscale=0.6))
variance(bspec(lh, priordf=3, priorscale=0.6))

# and again similar for autocovariances:
evaluation(acf(bspec(lh)))
evaluation(acf(bspec(lh, priordf=2, priorscale=0.6)))
variance(acf(bspec(lh)))
variance(acf(bspec(lh, priordf=4, priorscale=0.6)))

likelihood

Prior, likelihood and posterior

Description

Prior density, likelihood, posterior density, and marginal likelihood functions for the posterior distributions specified through a bspec object.

Usage

dprior(x, ...)
likelihood(x, ...)
marglikelihood(x, ...)
dposterior(x, ...)
## S3 method for class 'bspec'
dprior(x, theta, two.sided=x$two.sided, log=FALSE, ...)
## S3 method for class 'bspec'
likelihood(x, theta, two.sided=x$two.sided, log=FALSE, ...)
## S3 method for class 'bspec'
marglikelihood(x, log=FALSE, ...)
## S3 method for class 'bspec'
dposterior(x, theta, two.sided=x$two.sided, log=FALSE, ...)
Arguments

- **x**: A bspec object.
- **theta**: A numeric vector of parameter values, corresponding to the Fourier frequencies in the `x$freq` element.
- **two.sided**: A logical flag indicating whether the parameters `theta` correspond to the one-sided or two-sided spectrum.
- **log**: A logical flag indicating whether to return logarithmic density (or likelihood) values.
- ... currently unused.

Details

Prior and posterior are both scaled inverse $\chi^2$ distributions, and the likelihood is Normal.

Value

A numeric function value.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

References


See Also

bspec, quantile.bspec, expectation

Examples

```r
lhspec <- bspec(lh, priordf=1, priorscale=0.6)

# draw sample from posterior:
posteriorsample <- sample(lhspec)

# plot the sample:
plot(lhspec)
lines(lhspec$freq, posteriorsample, type="b", col="red")

# compute prior, likelihood, posterior:
print(c("prior" = dprior(lhspec, posteriorsample),
           "likelihood" = likelihood(lhspec, posteriorsample),
           "posterior" = dposterior(lhspec, posteriorsample),
           "marginal likelihood" = marglikelihood(lhspec)))
```
matchedfilter  

*Filter a noisy time series for a signal of given shape*

**Description**

Computes the maximized likelihood ratio (as a test- or detection-statistic) of "signal" vs. "noise only" hypotheses. The signal is modelled as a linear combination of orthogonal basis vectors of unknown amplitude and arrival time. The noise is modelled either as Gaussian or Student-t-distributed, and coloured.

**Usage**

```r
matchedfilter(data, signal, noisePSD, timerange = NA,
               reconstruct = TRUE, two.sided = FALSE)
```

```r
studenttfilter(data, signal, noisePSD, df = 10, timerange = NA,
                deltamax = 1e-06, itermax = 100,
                reconstruct = TRUE, two.sided = FALSE)
```

**Arguments**

- `data`  the data to be filtered, a time series (`ts`) object.
- `signal` the signal waveform to be filtered for. May be a vector, a matrix, a time series (`ts`) or a multivariate time series (`mts`) object.
- `noisePSD` the noise power spectral density. May be a vector of appropriate length (`length(data)/2+1`) or a function of frequency.
- `df` the number of degrees-of-freedom (`ν_j`) for each frequency bin. May be a vector of appropriate length (`length(data)/2+1`) or a function of frequency.
- `timerange` the range of times (with respect to the data argument’s time scale) to maximize the likelihood ratio over.
- `deltamax` the minimal difference in logarithmic likelihoods to be aimed for in the EM-iterations (termination criterion).
- `itermax` the maximum number of EM-iterations to be performed.
- `reconstruct` a logical flag indicating whether the output is supposed to include the best-fitting signal waveform.
- `two.sided` a logical flag indicating whether the `noisePSD` argument is to be interpreted as a one-sided or a two-sided spectrum.

**Details**

The time series data \(d(t)\) is modelled as a superposition of signal \(s(\beta, t_0, t)\) and noise \(n(t)\):

\[
d(t) = s(\beta, t - t_0) + n(t),
\]
where the signal is a linear combination of orthogonal (!) basis vectors $s_i(t)$, and whose 
time-of-arrival is given by the parameter $t_0$:

$$s(\beta, t - t_0) = \sum_{i=1}^{k} \beta_i s_i(t - t_0).$$

The noise is modelled as either Gaussian (matchedfilter()) or Student-t distributed (studenttfilter())
with given power spectral density and, for the latter model only, degrees-of-freedom parameters.

The filtering functions perform a likelihood maximization over the time-of-arrival $t_0$ and coefficients $\beta$. In the Gaussian model, the conditional likelihood, conditional on $t_0$, can be maximized analytically, while the maximization over $t_0$ is done numerically via a brute-force search. In the Student-t model, likelihood maximization is implemented using an EM-algorithm. The maximization over $t_0$ is restricted to the range specified via the timerange argument.

What is returned is the maximized (logarithmic) likelihood ratio of "signal" versus "noise-only" hypotheses (the result’s $\text{maxLLR}$ component), and the corresponding ML-estimates $\hat{t}_0$ and $\hat{\beta}$, as well as the ML-fitted signal ("$\text{reconstruction}$").

Value

A list containing the following elements:

- \text{maxLLR} the maximized likelihood ratio of signal vs. noise only hypotheses.
- \text{timerange} the range of times to maximize the likelihood ratio over (see the timerange
  input argument).
- \text{betaHat} the ML-estimated vector of coefficients.
- \text{tHat} the ML-estimated signal arrival time.
- \text{reconstruction} the ML-fitted signal (a time series (ts) object).
- \text{call} an object of class call giving the function call that generated the result.

elements only for the matchedfilter() function:

- \text{maxLLRseries} the time series of (conditionally) maximized likelihood ratio for each given time
  point (the profile likelihood).

elements only for the studenttfilter() function:

- \text{Eprogress} a matrix indicating the progress of the EM-fitting.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

References

Roever, C.. A Student-t based filter for robust signal detection. Physical Review D, 84(12):122004,
2011. See also arXiv preprint 1109.0442.

Roever, C., Meyer, R., Christensen, N. Modelling coloured residual noise in gravitational-wave
0804.3853.
Examples

```r
# sample size and sampling resolution:
deltaT <- 0.001
N <- 1000

# For the coloured noise, use some AR(1) process;
# AR noise process parameters:
sigmaAR <- 1.0
phiAR <- 0.9

# generate non-white noise
# (autoregressive AR(1) low-frequency noise):
noiseSample <- rnorm(10*N)
for (i in 2:length(noiseSample))
  noiseSample[i] <- phiAR*noiseSample[i-1] + noiseSample[i]
noiseSample <- ts(noiseSample, delta=deltaT)

# estimate the noise spectrum:
PSDestimate <- welchPSD(noiseSample, seglength=1,
                           windowingPsdCorrection=FALSE)

# show noise and noise PSD:
par(mfrow=c(2,1))
  plot(noiseSample, main="noise sample")
  plot(PSDestimate$freq, PSDestimate$pow, log="y", type="l",
       main="noise PSD", xlab="frequency", ylab="power")
par(mfrow=c(1,1))

# generate actual data:
noise <- rnorm(N)
for (i in 2:length(noise))
  noise[i] <- phiAR*noise[i-1] + noise[i]
noise <- ts(noise, start=0, delta=deltaT)

# the "sine-Gaussian" signal to be injected into the noise:
t0 <- 0.6
phase <- 1.0
signal <- exp(-(time(noise)-t0)^2/(2*0.01^2)) * sin(2*pi*150*(time(noise)-t0)+phase)
plot(signal)

t <- seq(-0.1, 0.1, by=deltaT)
# the signal's orthogonal (sine- and cosine-) basis waveforms:
signalSine <- exp(-t^2/(2*0.01^2)) * sin(2*pi*150*t)
signalCosine <- exp(-t^2/(2*0.01^2)) * sin(2*pi*150*t+pi/2)
signalBasis <- ts(cbind("sine"=signalSine, "cosine"=signalCosine),
                  start=-0.1, delta=deltaT)
plot(signalBasis[,1], col="red", main="the signal basis")
lines(signalBasis[,2], col="green")
# (the sine- and cosine- components allow to span
# signals of arbitrary phase)
# Note that "signalBasis" may be shorter than "data",
# but must be of the same time resolution.
```

"matchedfilter"
# compute the signal's signal-to-noise ration (SNR):
signalSnr <- snr(signal, psd=PSDestimate$pow)

# scale signal to SNR = 6:
rho <- 6
data <- noise + signal * (rho/signalSnr)
data <- data * tukeywindow(length(data))
# Note that the data has (and should have!)
# the same resolution, size, and windowing applied
# as in the PSD estimation step.

# compute filters:
f1 <- matchedfilter(data, signalBasis, PSDestimate$power)
f2 <- studenttfilter(data, signalBasis, PSDestimate$power)

# illustrate the results:
par(mfrow=c(3,1))
plot(data, ylab="", main="data")
lines(signal* (rho/signalSnr), col="green")
legend(0, max(data), c("noise + signal","signal only"),
  lty="solid", col=c("black","green"), bg="white")

plot(signal * (rho/signalSnr), xlim=c(0.55, 0.65), ylab="", main="original & recovered signals")
lines(f1$reconstruction, col="red")
lines(f2$reconstruction, col="blue")
abline(v=c(f1$Hat, f2$Hat), col=c("red", "blue"), lty="dashed")
legend(0.55, max(signal*(rho/signalSnr)),
  c("injected signal","best-fitting signal (Gaussian model)",
  "best-fitting signal (Student-t model)"),
  lty="solid", col=c("black","red","blue"), bg="white")

plot(f1$maxLLRseries, type="n", ylim=c(0, f1$maxLLR),
  main="profile likelihood (Gaussian model)",
  ylab="maximized (log-) likelihood ratio")
lines(f1$maxLLRseries, col="grey")
lines(window(f1$maxLLRseries, start=f1$timerange[1], end=f1$timerange[2])
  abline(v=f1$timerange, lty="dotted")
lines(c(f1$Hat, f1$Hat, -1), c(0, f1$maxLLR, f1$maxLLR), col="red", lty="dashed")
par(mfrow=c(1,1))

---

(one.sided) Conversion between one- and two-sided spectra

Description

Functions to convert between one- and two-sided bspec objects.
Usage

```r
one.sided(x, ...) two.sided(x, ...)  
```

## S3 method for class 'bspec'
```r
one.sided(x, ...)  
```

## S3 method for class 'bspec'
```r
two.sided(x, ...)  
```

Arguments

- `x`: a `bspec` object.
- `...`: currently unused.

Details

The conversion only means that the `$two.sided` element of the returned `bspec` object is set correspondingly, as internally always the same (one-sided) spectrum is used.

Value

A `bspec` object (see the help for the `bspec` function).

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

See Also

`bspec`

Examples

```r
lhspec <- bspec(lh)

# compare distributions visually:
par(mfrow=c(2,1))
plot(lhspec)
plot(two.sided(lhspec))
par(mfrow=c(1,1))

# ...and numerically:
print(cbind("frequency"=lhspec$freq,
            "median-1sided"=quantile(lhspec,0.5),
            "median-2sided"=quantile(two.sided(lhspec),0.5)))
```
**ppsample**

**Posterior predictive sampling**

**Description**

Draws a sample from the posterior predictive distribution specified by the supplied bspec object.

**Usage**

```r
ppsample(x, ...)  # S3 method for class 'bspec'
ppsample(x, start=x$start, ...)  # S3 method for class 'bspec'
```

**Arguments**

- `x`: a bspec object specifying the posterior distribution from which to sample.
- `start`: the start time of the resulting time series.
- `...`: currently unused.

**Value**

A time series (ts) object of the same kind (with respect to sampling rate and sample size) as the data the posterior distribution is based on.

**Author(s)**

Christian Roever, <christian.roever@med.uni-goettingen.de>

**See Also**

bspec, sample.bspec

**Examples**

```r
par(mfrow=c(2,1))
plot(lh, main="lh data")
plot(ppsample(bspec(lh)), main="posterior predictive sample")
par(mfrow=c(1,1))
```
Quantiles of the posterior spectrum

Description

Function to compute quantiles of the spectrum’s posterior distribution specified through the supplied bspec object argument.

Usage

```r
## S3 method for class 'bspec'
quantile(x, probs = c(0.025, 0.5, 0.975),
         two.sided = x$two.sided, ...)
```

Arguments

- `x` a bspec object.
- `probs` a numerical vector of probabilities.
- `two.sided` a logical flag indicating whether quantiles are supposed to correspond to the one-sided or two-sided spectrum.
- `...` currently unused.

Details

The posterior distribution is a product of independent scaled inverse $\chi^2$ distributions.

Value

A matrix with columns corresponding to elements of `probs`, and rows corresponding to the Fourier frequencies `x$freq`. If `probs` is of length 1, a vector is returned instead.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

See Also

- `bspec`
- `quantile`

Examples

```r
lhspec <- bspec(lh)

# posterior medians:
print(cbind("frequency"=lhspec$freq,
             "median"=quantile(lhspec, 0.5)))
```
Description
Function to generate samples from the spectrum’s posterior distribution specified through the supplied bspec object argument.

Usage
```r
## S3 method for class 'bspec'
sample(x, size = 1, two.sided = x$two.sided, ...)
```

Arguments
- `x`: a bspec object.
- `size`: the sample size.
- `two.sided`: a logical flag indicating whether the drawn samples are supposed to correspond to the one-sided or two-sided spectrum.
- `...`: currently unused.

Details
The posterior distribution is a product of independent scaled inverse $\chi^2$ distributions.

Value
A (numerical) vector of samples from the posterior distribution of the spectral parameters, of the same length as and corresponding to the $\text{freq}$ element of the supplied bspec object.

Author(s)
Christian Roever, <christian.roever@med.uni-goettingen.de>

See Also
bspec, ppsample

Examples
```r
# determine spectrum's posterior distribution:
lspec <- bspec(lh)

# plot 95 percent central intervals and medians:
plot(lhspec)

# draw and plot two samples from posterior distribution:
lines(lhspec$freq, sample(lhspec), type="b", pch=20, col="red")
lines(lhspec$freq, sample(lhspec), type="b", pch=20, col="green")
```
**`snr`**

*Compute the signal-to-noise ratio (SNR) of a signal*

---

**Description**

Compute the SNR for a given signal and noise power spectral density.

**Usage**

```r
snr(x, psd, two.sided = FALSE)
```

**Arguments**

- `x` the signal waveform, a time series (ts) object.
- `psd` the noise power spectral density. May be a vector of appropriate length (`length(x)/2+1`) or a function of frequency.
- `two.sided` a logical flag indicating whether the `psd` argument is to be interpreted as a one-sided or a two-sided spectrum.

**Details**

For a signal \( s(t) \), the complex-valued discrete Fourier transform \( \hat{s}(f) \) is computed along the Fourier frequencies \( f_j = \frac{j}{N\Delta t}, j = 0, \ldots, N/2 + 1 \), where \( N \) is the sample size, and \( \Delta t \) is the sampling interval. The SNR, as a measure of "signal strength" relative to the noise, then is given by

\[
\varrho = \sqrt{\frac{N/2+1}{\sum_{j=0}^{N/2+1} \frac{|s(f_j)|^2}{N/4\Delta t S_1(f_j)}}},
\]

where \( S_1(f) \) is the noise’s one-sided power spectral density. For more on its interpretation, see e.g. Sec. II.C.4 in the reference below.

**Value**

The SNR \( \varrho \).

**Author(s)**

Christian Roever, `<christian.roever@med.uni-goettingen.de>`

**References**


**See Also**

`matchedfilter`, `studenttfilter`
Examples

# sample size and sampling resolution:
N <- 1000
deltaT <- 0.001

# For the coloured noise, use some AR(1) process;
# AR noise process parameters:
sigmaAR <- 1.0
phiAR <- 0.9

# generate non-white noise
# (autoregressive AR(1) low-frequency noise):
noiseSample <- rnorm(10*N)
for (i in 2:length(noiseSample))
  noiseSample[i] <- phiAR*noiseSample[i-1] + noiseSample[i]
noiseSample <- ts(noiseSample, deltat=deltaT)

# estimate the noise spectrum:
PSDestimate <- welchPSD(noiseSample, seglength=1,
                           windowingPsdCorrection=FALSE)

# generate a (sine-Gaussian) signal:
t0 <- 0.6
phase <- 1.0
t <- ts((0:(N-1))*deltaT, deltat=deltaT, start=0)
signal <- exp(-(t-t0)^2/(2*0.01^2)) * sin(2*pi*150*(t-t0)+phase)
plot(signal)

# compute the signal's SNR:
snr(signal, psd=PSDestimate$power)

---

temper

Tempering of (posterior) distributions

Description

Setting the tempering parameter of ('tempered') bspec objects.

Usage

temper(x, ...)
  ## S3 method for class 'bspec'
temper(x, temperature = 2, likelihood.only = TRUE, ...)

Arguments

x                a bspec object.
temperature      a (positive) 'temperature' value.
likelihood.only

a logical flag indicating whether to apply the tempering to the ‘complete’ posterior density, or to the likelihood only (default).

... currently unused.

Details

In the context of Markov chain Monte Carlo (MCMC) applications it is often desirable to apply tempering to the distribution of interest, as it is supposed to make the distribution more easily tractable. Examples where tempering is utilised are simulated annealing, parallel tempering or evolutionary MCMC algorithms. In the context of Bayesian inference, tempering may be done by specifying a ‘temperature’ \( T \) and then manipulating the original posterior distribution \( p(\theta|y) \) by applying an exponent \( \frac{1}{T} \) either to the complete posterior distribution:

\[
p_T(\theta) \propto p(\theta|y)^{\frac{1}{T}} = (p(y|\theta)p(\theta))^{\frac{1}{T}}
\]

or to the likelihood part only:

\[
p_T(\theta) \propto p(y|\theta)^{\frac{1}{T}}p(\theta).
\]

In this context, where the posterior distribution is a product of scaled inverse \( \chi^2 \) distributions, the tempered distributions in both cases turn out to be again of the same family, just with different parameters. For more details see also the references.

Value

An object of class \texttt{bspec} (see the help for the \texttt{bspec} function), but with an additional temperature element.

Note

Tempering with the \texttt{likelihood.only} flag set to \texttt{FALSE} only works as long as the temperature is less than \( \min((x^2 df + 2)/2) \).

Author(s)

Christian Roever, \texttt{christian.roever@med.uni-goettingen.de}

References


See Also

\texttt{temperature,bspec}
Examples

```r
lhspec <- bspec(lh, priorscale=0.6, priordf=1)

# details of the regular posterior distribution:
str(lhspec)

# details of the tempered distribution
# (note the differing scale and degrees-of-freedom):
str(temper(lhspec, 1.23))
```

---

**temperature**

*Querying the tempering parameter*

---

**Description**

Retrieving the “temperature” parameter of (‘tempered’) bspec objects

**Usage**

```r
temperature(x, ...)
```

## S3 method for class 'bspec'
temperature(x, ...)

**Arguments**

- `x` a bspec object.
- `...` currently unused.

**Value**

The (numeric) value of the temperature element of the supplied bspec object, if present, and 1 otherwise.

**Author(s)**

Christian Roever, <christian.roever@med.uni-goettingen.de>

**References**


**See Also**

- `temper`, `bspec`
Examples

```r
lhspec1 <- bspec(lh)
lhspec2 <- temper(lhspec1, 1.23)

print(lhspec2$temperature)
print(lhspec1$temperature)

print(temperature(lhspec2))
print(temperature(lhspec1))
```

**Description**

Several windowing functions for spectral or Fourier analysis of time series data are provided.

**Usage**

```r
tukeywindow(N, r = 0.1)
squarewindow(N)
hannwindow(N)
welchwindow(N)
trianglewindow(N)
hammingwindow(N, alpha=0.543478261)
cosinowindow(N, alpha=1)
kaiserwindow(N, alpha=3)
```

**Arguments**

- `N`: the length of the time series to be windowed
- `r`: the Tukey window's parameter, denoting the its "non-flat" fraction.
- `alpha`: additional parameter for Hamming-, cosine-, and Kaiser-windows.

**Details**

Windowing of time series data, i.e., multiplication with a tapering function, is often useful in spectral or Fourier analysis in order to reduce "leakage" effects due to the discrete and finite sampling. These functions provide windowing coefficients for a given sample size `N`.

**Value**

A vector (of length `N`) of windowing coefficients.

**Author(s)**

Christian Roever, <christian.roever@med.uni-goettingen.de>
welchPSD

References


See Also

welchPSD, empiricalSpectrum

Examples

# illustrate the different windows' shapes:
N <- 100
matplot(1:N,
  cbind(cosinewindow(N),
        hammingwindow(N),
        hannwindow(N),
        kaiserwindow(N),
        squarewindow(N),
        trianglewindow(N),
        tukeywindow(N, r=0.5),
        welchwindow(N)),
  type="l", lty="solid", col=1:8)
legend(N, 0.99, legend=c("cosine", "hamming", "hann", "kaiser",
                          "square", "triangle", "tukey", "welch"),
        col=1:8, lty="solid", xjust=1, yjust=1, bg="white")

# show their effect on PSD estimation:
data(sunspots)

spec1 <- welchPSD(sunspots, seglength=10, windowfun=squarewindow)
plot(spec1$frequency, spec1$power, log="y", type="l")

spec2 <- welchPSD(sunspots, seglength=10, windowfun=tukeywindow, r=0.25)
lines(spec2$frequency, spec2$power, log="y", type="l", col="red")

spec3 <- welchPSD(sunspots, seglength=10, windowfun=trianglewindow)
lines(spec3$frequency, spec3$power, log="y", type="l", col="green")

welchPSD

Power spectral density estimation using Welch's method.

Description

Estimates a time series' power spectral density using *Welch's method*, i.e., by subdividing the data into segments, computing spectra for each, and averaging over the results.
Usage

welchPSD(x, seglength, two.sided = FALSE, windowfun = tukeywindow,
method = c("mean", "median"), windowingPsdCorrection = TRUE, ...)

Arguments

x             a time series (ts object).
seglength     the length of the subsegments to be used (in units of time relative to x).
two.sided     a logical flag indicating whether the result is supposed to be one-sided (default) or two-sided.
windowfun     The windowing function to be used.
method        The "averaging" method to be used – either "mean" or "median".
windowingPsdCorrection
              a logical flag indicating whether an overall correction for the windowing is supposed to be applied to the result – this essentially specifies whether the result is supposed to reflect the PSD of the windowed or of the "un-windowed" time series.
...           other parameters passed to the windowing function.

Details

The time series will be divided into overlapping sub-segments, each segment is windowed and its "empirical" spectrum is computed. The average of these spectra is the resulting PSD estimate. For robustness, the median may also be used instead of the mean.

Value

A list containing the following elements:

frequency     the Fourier frequencies.
power         the estimated spectral power.
kappa         the number of (by definition) non-zero imaginary components of the Fourier series.
two.sided     a logical flag indicating one- or two-sidedness.
segments      the number of (overlapping) segments used.

Author(s)

Christian Roever, <christian.roever@med.uni-goettingen.de>

References

See Also

empiricalSpectrum, tukeywindow, spectrum

Examples

# load example data:
data(sunspots)
# compute and plot the "plain" spectrum:spec1 <- empiricalSpectrum(sunspots)plot(spec1$frequency, spec1$power, log="y", type="l")

# plot Welch spectrum using segments of length 10 years:spec2 <- welchPSD(sunspots, seglength=10)lines(spec2$frequency, spec2$power, col="red")

# use 20-year segments and a flatter Tukey window:spec3 <- welchPSD(sunspots, seglength=20, r=0.2)lines(spec3$frequency, spec3$power, col="blue")
Index

*Topic models
  bspec-package, 2
*Topic package
  bspec-package, 2
*Topic robust
  matchedfilter, 12
*Topic ts
  acf, 4
  acf (acf.bspec), 2
  acf.bspec, 2, 6, 10
  bspec, 2, 4, 4, 9–11, 16–19, 22, 23
  bspec-package, 2
  bspecACF, 9
  cosinewindow (tukeywindow), 24
  dposterior (likelihood), 10
  dprior (likelihood), 10
  empiricalSpectrum, 7, 25, 27
  expectation, 4, 6, 9, 11
  fft, 8
  hammingwindow (tukeywindow), 24
  hannwindow (tukeywindow), 24
  is.bspec, 2
  is.bspecACF (acf.bspec), 2
  kaiserwindow (tukeywindow), 24
  likelihood, 10
  marglikelihood (likelihood), 10
  matchedfilter, 12
  mts, 5, 12
  one.sided, 15
  plot.bspec (bspec), 4
  plot.bspecACF (acf.bspec), 2
  ppsample, 6, 17, 19
  print.bspec (bspec), 4
  print.bspecACF (acf.bspec), 2
  quantile, 18
  quantile.bspec, 6, 11, 18
  sample, 3
  sample (sample.bspec), 19
  sample.bspec, 4, 6, 17, 19
  snr, 20
  spectrum, 6, 8, 27
  squarewindow (tukeywindow), 24
  studenttfilter, 20
  studenttfilter (matchedfilter), 12
  temper, 21
  temperature, 22, 23
  trianglewindow (tukeywindow), 24
  ts, 5, 7, 12, 13, 20, 26
  tukeywindow, 8, 24, 27
  two.sided (one.sided), 15
  variance (expectation), 9
welchPSD, 8, 25, 25
welchwindow (tukeywindow), 24