Package 'ForeCA'

January 27, 2015

Type Package

Title ForeCA - Forecastable Component Analysis

Version 0.1

Date 2014-03-01

Author Georg M. Goerg <im@gmge.org>

Maintainer Georg M. Goerg <im@gmge.org>

URL http://www.gmge.org

Description Forecastable Component Analysis (ForeCA) is a novel dimension reduction (DR) technique for temporally dependent signals. Contrary to other popular DR methods, such as PCA or ICA, ForeCA explicitly searches for the most "forecastable" signal. The measure of forecastability is based on negative Shannon entropy of the spectral density of the transformed signal. This R package provides the main algorithms and auxiliary function(summary, plotting, etc) to apply ForeCA to multivariate data (time series).

Depends R (>= 2.15.0), ifultools (>= 2.0-0), splus2R (>= 1.2-0), nlme (>= 3.1-64)

License GPL-2

Imports R.utils, sapa, mgcv, astsa

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-03 00:52:23

R topics documented:

ForeCA-package *Implementation of Forecastable Component Analysis (ForeCA)*

Description

Forecastable Component Analysis (ForeCA) is a novel dimension reduction (DR) technique for multivariate time series. ForeCA finds a linar combination $y_t = \mathbf{w}'\mathbf{X}_t$ that is easy to forecast. The measure of forecastability $\Omega(x_t)$ (Omega) is based on the entropy of the spectral density $f_y(\lambda)$ of y_t: higher entropy means less forecastable, lower entropy is more forecastable.

The main function foreca runs ForeCA on a multivariate time series to find the most forecastable signals.

Even though the current version has most functionality of this R package, some function name conventions might change in future versions. Please consult the NEWS file for a list of changes.

Author(s)

Author and maintainer: Georg M. Goerg <im@gmge.org>

References

Goerg, G. M. (2013). "Forecastable Component Analysis". Journal of Machine Learning Research (JMLR) W&CP 28 (2): 64-72, 2013. Available at jmlr.org/proceedings/papers/v28/goerg13.html.

Examples

```
XX <- ts(diff(log(EuStockMarkets))[-c(1:1000),])
Omega(XX)

plot(log(lynx,10))
Omega(log(lynx,10), spectrum.method = "wosa")

## Not run:
ff <- foreca(XX, n.comp = 4, spectrum.method = "wosa")
plot(ff)
summary(ff)

## End(Not run)
```

`continuous_entropy` *Shannon entropy for a continuous pdf*

Description

Computes the Shannon entropy $\mathcal{H}(p)$ for a continuous probability density function (pdf) $p(x)$.

Usage

```
continuous_entropy(pdf, a = NULL, b = NULL, base = 2)
```

Arguments

`pdf`	R function (for example, `function(x) return(dunif(x, 0, 1))`) for the pdf $p(x)$ of a RV $X \sim p(x)$. Note that this function must integrate to 1 over the interval [`a`, `b`].
`base`	logarithm base; entropy is measured in "nats" for `base = e`; in "bits" if `base = 2` (default).
`a`	lower integration limit
`b`	upper integration limit

Details

The Shannon entropy of a continuous RV $X \sim p(x)$ is defined as

$$\mathcal{H}(p) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx$$

Contrary to the entropy of a discrete RV, continuous RVs can have negative entropy (see Examples).

Value

Entropy (real).

See Also

`discrete_entropy`

Examples

```
my_density = function(x){
 dunif(x, -1, 1)
}

## Not run:
# error since limits of integration were not specified
continuous_entropy(my_density)
```

```
# error since density does not integrate to '1' over [-0.5, 1]
continuous_entropy(my_density, -0.5, 1)

## End(Not run)
continuous_entropy(my_density, -1, 1) # = log(b - a)

# entropy of U(a,b) = log(b-a). Thus not necessarily positive anymore, e.g.
continuous_entropy(function(x) dunif(x, 0, 0.5), 0, 0.5) # log2(0.5)
```

`discrete_entropy` *Shannon entropy for discrete pmf*

Description

Computes the Shannon entropy $\mathcal{H}(p)$ of a discrete RV X taking values in $\{x_1, \ldots, x_n\}$ with probability mass function (pmf) $Prob(X = x_i) \in \{p_1, \ldots, p_n\}$.

Usage

```
discrete_entropy(probs, base = 2, method = c("MLE", "smoothed"),
  threshold = 0, smoothing.weight = 1)
```

Arguments

`probs`	probabilities (empirical frequencies). Must add up to 1.
`base`	see `base` argument of `continuous_entropy`
`method`	method to estimate entropy; see Details below.
`threshold`	frequencies below `threshold` are set to 0; default `threshold = 0`.
`smoothing.weight`	how much weight does the uniform smoothing term get? default `1`.

Details

`discrete_entropy` uses a plug-in estimator (`method = ''MLE''`):

$$\widehat{H}(p) = -\sum_{i=1}^{n} \widehat{p}_i \log \widehat{p}_i.$$

The smoothed estimator (`method = 'smoothed'`) adds a uniform distribution $1/n$ to every probability (and then normalizes again):

$$\widehat{p}_i \leftarrow \widehat{p}_i + \alpha \cdot \frac{1}{n}, \quad i = 1, \ldots, n,$$

where α is the smoothing weight parameter.

Value

Entropy (a non-negative real value).

See Also

continuous_entropy

Examples

```
pmfs <- rexp(5)
pmfs <- sort(pmfs / sum(pmfs))

unif.distr <- rep(1/length(pmfs), length(pmfs))

matplot(cbind(pmfs, unif.distr), pch = 19, ylab = "P(X = k)", xlab = "k")
matlines(cbind(pmfs, unif.distr))
legend("topleft", c("non-uniform","uniform"), pch = 19, lty = 1:2, col = 1:2)

discrete_entropy(pmfs)
# uniform has largest entropy among all bounded distributions (here = log(5))
discrete_entropy(unif.distr)
# no uncertainty if one element occurs with probability 1
discrete_entropy(c(1,0,0,0,0))
```

---

foreca *Runs Forecastable Component Analysis*

Description

foreca performs Forecastable Component Analysis (ForeCA) on a K-dimensional time series with T observations.

Usage

```
foreca(series, algorithm.type = c("EM"), ...)
```

Arguments

series	multivariate time series of dimension $T \times K$
algorithm.type	specifies the algorithm that should be used to estimate forecastable components. Currently only the EM-like algorithm from the original Goerg (2013) paper is available (see References): algorithm.type = "EM" calls foreca.EM.
...	additional arguments passed to the available ForeCA algorithms.

Value

An object of class foreca: A list with similar output as princomp. Signals are ordered from most to least forecastable.

Examples

```
## Not run:
XX <- diff(log(EuStockMarkets[-c(1:50),])) * 100
plot(ts(XX))
ff <- foreca(XX[,1:3], n.comp = 3)

summary(ff)
plot(ff)

## End(Not run)
```

`foreca-utils` *Plot, summary, and print methods for class 'foreca'*

Description

Plot, summary, and print methods for class 'foreca'

`plot.foreca` shows a visual summary of the ForeCA results with biplots, screeplots, and white noise tests.

`plot.foreca.EM.opt_weightvector` shows how the EM-like algorithm arrived at the i-th optimal a weight-vector $\mathbf{w}_i^*$. It shows trace plots of the objective function (`foreca.EM.h`) and of the weight vector, and the transformed signal y_t^* along with its spectral density estimate $\widehat{f}_y(\omega_j)$.

`summary.foreca` computes summary statistics of the ForeCA results.

`biplot.foreca` shows a biplot of the ForeCA weightvectors (a wrapper around `biplot.princomp`).

Usage

```
## S3 method for class 'foreca'
plot(x, lag = 10, alpha = 0.05, ...)

## S3 method for class 'foreca.EM.opt_weightvector'
plot(x, main = "", cex.lab = 1.1, ...)

## S3 method for class 'foreca'
summary(object, lag = 10, alpha = 0.05, ...)

## S3 method for class 'foreca'
biplot(x, ...)
```

Arguments

`...`	additional arguments passed to `biplot.princomp`, `biplot.default`, `plot`, or `summary`.
`x`	an object of class `"foreca"` or `"foreca.EM.opt_weightvector"`
`main`	an overall title for the plot: see `title`.

cex.lab	size of the axes labels
object	an object of class "foreca"
alpha	significance level for testing white noise in Box.test; default: 5%.
lag	how many lags to test in Box.test; default 10 lags.

Examples

```
# see examples in 'foreca.EM'
```

---

foreca.EM *EM-like algorithm to estimate optimal ForeCA transformation*

Description

foreca.EM estimates the optimal transformation of multivariate time series to obtain forecastable signals using an EM-like algorithm.

Usage

```
foreca.EM(series, spectrum.method = c("multitaper", "direct", "lag window",
  "wosa", "mvspec"), n.comp = 2, kernel = NULL, plot = TRUE,
  threshold = 0, control = list(tol = 1e-06, max.iter = 200, num.starts =
  10), smoothing = FALSE, ...)

foreca.EM.opt_weightvector(series, f.U = NULL,
  spectrum.method = c("multitaper", "direct", "lag window", "wosa", "mvspec"),
  entropy.method = "MLE", control = list(tol = 1e-06, max.iter = 100,
  num.starts = 10), kernel = NULL, threshold = 0, smoothing = FALSE, ...)
```

Arguments

n.comp	number of components to be extracted
plot	logical; if TRUE a plot of the current optimal solution $\mathbf{w}_i^*$ will be shown and updated in each iteration of the EM-like algorithm.
series	a $T \times K$ array containing a multivariate time series. Can be a matrix, data.frame, or a multivariate ts object.
f.U	multivariate spectrum of class mvspectrum with normalize = TRUE
spectrum.method	string; method to estimate the spectrum; see mvspectrum.
entropy.method	string; method to estimate the entropy; see discrete_entropy.
control	a list with control parameters for the iterative EM algorithm: i) tol sets the tolerance level for convergence; ii) max.iter sets the maximum number of iterations; and iii) num.starts determines how many random starts should be done to avoid local minima (default:5)

kernel	an R function `kernel = function(lambda) ...` that weighs the Fourier frequencies; if NULL (default) all frequencies get equal weight.
threshold	set estimate of spectral density below threshold to 0 (and renormalize so it adds up to one)
smoothing	logical; if TRUE the spectral density estimate will be smoothed using gam. See mvspectrum for further details.
...	other arguments passed to mvspectrum

Value

A list with similar output as princomp. Signals are ordered from most to least forecastable.

Examples

```
## Not run:
XX <- diff(log(EuStockMarkets[-c(1:50),])) * 100
plot(ts(XX))
foreca <- foreca.EM(XX[,1:3], n.comp = 3)

summary(foreca)
plot(foreca)

## End(Not run)
## Not run:
XX <- diff(log(EuStockMarkets)) * 100
one.weight <- foreca.EM.opt_weightvector(XX, smoothing = FALSE)

plot(one.weight)

## End(Not run)
```

---

foreca.EM-aux *ForeCA EM auxiliary functions*

Description

The EM-like algorithm relies on several auxiliary functions:

`foreca.EM.E_step` computes the spectral density of $y_t = \mathbf{w}'\mathbf{X}_t$ given the weightvector $\mathbf{w}$.

`foreca.EM.M_step` computes the minimizing eigenvector ($\rightarrow \widehat{\mathbf{w}}_{i+1}$) of the weighted covariance matrix, where the weights equal the negative logarithm of the spectral density at the current $\widehat{\mathbf{w}}_i$.

`foreca.EM.h` evaluates the entropy of the spectral density as a function of $\mathbf{w}$. This is the objective funcion that should be minimized.

Usage

```
foreca.EM.E_step(f.U, weightvector)

foreca.EM.M_step(f.U, f.current, base = NULL, minimize = TRUE,
  Sigma.X = NULL)

foreca.EM.h(weightvector.new, f.U, weightvector.current = weightvector.new,
  f.current = NULL, base = NULL)
```

Arguments

`f.U`	an object of class `"mvspectrum"` with `normalized = TRUE`
`weightvector`	a weight vector for $y_t = \mathbf{w}'\mathbf{X}_t$. Must have unit norm in L^2.
`base`	logarithm base; if `NULL` it sets it automatically to T, such that the resulting discrete entropy estimate is bounded above by 1 (it is always bounded below by 0).
`minimize`	logical; if `TRUE` it returns the eigenvector corresponding to the smallest eigenvalue; otherwise to the largest eigenvalue.
`Sigma.X`	optional; covariance matrix of $\mathbf{X}$
`weightvector.new`	weightvector $\widehat{\mathbf{w}}_{i+1}$ of the new iteration (i+1)
`f.current`	spectral density of $y_t = \mathbf{w}'\mathbf{X}_t$ for the current estimate $\widehat{\mathbf{w}}_i$ (required for `foreca.EM.M_step`; optional for `foreca.EM.h`).
`weightvector.current`	weightvector $\widehat{\mathbf{w}}_i$ of the current iteration (i)

Value

`foreca.EM.E_step` returns a vector containing the normalized spectral density (normalized such that its `mean` equals 0.5 - since `f.U` only contains positive frequencies and is symmetric).

`foreca.EM.M_step` returns a list with three elements:

`matrix`**:** is the weighted covariance matrix (positive semi-definite), where the weights are the negative log of the spectral density,

`vector`**:** minimizing (or maximizing if `minimize = FALSE`) eigenvector of `matrix`,

`value`**:** corresponding eigenvalue.

`foreca.EM.h` returns (see References for details):

- the negative entropy (non-negative real) if `weightvector.new = weightvector.current`
- an upper bound of that entropy for `weightvector.new` if `weightvector.new != weightvector.current`

Examples

```
XX <- diff(log(EuStockMarkets)) * 100
ff <- mvspectrum(XX, 'wosa', normalize = TRUE)

ww0 <- matrix(rnorm(ncol(XX)))
ww0 <- ww0 / norm(ww0, 'F')

f.ww0 <- foreca.EM.E_step(ff, ww0)
onestep <- foreca.EM.M_step(ff, f.ww0)
image(onestep$matrix)
## Not run:
require(LICORS)
# if you have 'LICORS' package installed, better use this:
image2(onestep$matrix)

## End(Not run)
ww1 <- onestep$vector
f.ww1 <- foreca.EM.E_step(ff, ww1)

layout(matrix(1:2, ncol = 2))
#par(mar = c(4, 4, 1, 1), mfcol= c(1,2))
matplot(seq(0, pi, length = length(f.ww0)), cbind(f.ww0, f.ww1),
        type = "l", lwd =2, xlab = "omega_j", ylab = "f(omega_j)")
plot(f.ww0, f.ww1, pch = ".", cex = 3, xlab = "iteration 0",
     ylab = "iteration 1", main = "Spectral density")
abline(0, 1, col = 'blue', lty = 2)
foreca.EM.h(ww0, ff)       # iteration 0
foreca.EM.h(ww1, ff, ww0)  # min eigenvalue inequality
foreca.EM.h(ww1, ff)       # KL divergence inequality
onestep$value

Omega(spectrum.estimate = f.ww0) / 100 + foreca.EM.h(ww0, ff)
Omega(spectrum.estimate = f.ww1) / 100 + foreca.EM.h(ww1, ff)
```

`initialize_weightvector` *Initialize the weightvector for iterative ForeCA algorithms*

Description

`initialize_weightvector` returns a unit norm (in L^2) vector $\mathbf{w}_0 \in R^k$ that can be used as the initial starting point for any iterative ForeCA algorithm, e.g., the EM-like algorithm in `foreca.EM.opt_weightvector`. Several quickly computable heuristics to get a good starting value are available. See `method` argument and Details below.

Usage

```
initialize_weightvector(U, f.U, method = c("SFA", "SFA.slow", "SFA.fast",
  "cauchy", "unif", "norm", "max"), seed = NULL, lag = 1)
```

Arguments

f.U	an object of class "mvspectrum" with normalized = TRUE
U	uncorrelated multivariate time series of dimension $T \times K$.
method	string indicating the method to estimate the initial weightvector; default "SFA"; see Details.
seed	seed to use for random initialization; if NULL, it sets a random seed (and it will be returned for reproducibility).
lag	integer; lag for the difference operator; default lag=1.

Details

The method argument specifies the heuristics that is used to get a good starting vector $\mathbf{w}_0$. This vector has length k and unit norm in L^2:

max uses the vector with all 0s, but a 1 at the position of the maximum forecastable series in U.

SFA.slow uses the first eigenvector of SFA (slowest signal).

SFA.fast uses the last eigenvector of SFA (fastest signal).

SFA checks both slow and fast, and chooses the one with higher forecastability.

cauchy random starts using rcauchy(k)

unif random starts using runif(k, -1, 1)

norm random starts using rnorm(k, 0, 1)

Slow Feature Analysis (SFA) finds *slow* signals (see References below), and can be quickly (and analytically) computed solving a generalized eigen-value problem. For ForeCA it is important to know that SFA is equivalent to finding the signal with largest lag 1 autocorrelation. This is not necessarily the most forecastable, but a good start.

The disadvantage of SFA for forecasting is that e.g., white noise (WN) is ranked higher than an AR(1) with negative autocorrelation coefficient $\rho_1 < 0$. While it is true that WN is slower, it is not more forecastable. Thus we are also interested in the fastest signal, i.e., the last eigenvector. The so obtained fastest signal corresponds to minimizing the lag 1 auto-correlation (possibly $\rho_1 < 0$).

Value

A vector of length k with unit norm (in L^2).

References

Laurenz Wiskott and Terrence J. Sejnowski (2002). "Slow Feature Analysis: Unsupervised Learning of Invariances", Neural Computation 14:4, 715-770.

`mvspectrum2wcov` *Compute (weighted) covariance matrix from frequency spectrum*

Description

Estimates the (weighted) covariance matrix from the frequency spectrum (see Details).

Usage

```
mvspectrum2wcov(mvspectrum.output, weights = 1)
```

Arguments

`mvspectrum.output`	output of class `"mvspectrum"`
`weights`	a sequence of weights for each frequency. By default uses weights that average out to 1.

Details

The covariance matrix of a multivariate time series equals the average of the power spectrum

$$\Sigma_X = \int_{-\pi}^{\pi} S_X(\lambda) d\lambda.$$

A generalized covariance matrix can be obtained using a weighted average

$$\tilde{\Sigma}_X = \int_{-\pi}^{\pi} K(\lambda) S_X(\lambda) d\lambda,$$

where $K(\lambda)$ is a kernel symmetric around 0 which integrates to 1. This allows one to remove or amplify specific frequencies in the covariance matrix estimate.

For ForeCA `mvspectrum2wcov` is especially important as we use

$$K(\lambda) = -\log f_y(\lambda),$$

as the *weights* (their average is not 1!).

Value

An $n \times n$ matrix.

See Also

`mvspectrum`

Examples

```
# use SDF first and then SDF2mvspectrum
set.seed(1)
nn <- 20
YY <- cbind(rnorm(nn), arima.sim(n = nn, list(ar = 0.9)), rnorm(nn))
XX <- YY %*% matrix(rnorm(9), ncol = 3)
XX <- scale(XX, scale = FALSE, center = TRUE)

# sample estimate of covariance matrix
Sigma.hat <- cov(XX)
dimnames(Sigma.hat) <- NULL

# using the frequency spectrum
SS <- mvspectrum(XX, "wosa")
Sigma.hat.freq <- mvspectrum2wcov(SS)

plot(c(Sigma.hat/Sigma.hat.freq))
abline(h = 1)

image(Sigma.hat)
image(Sigma.hat.freq)
image(Sigma.hat / Sigma.hat.freq)
```

`Omega` *Estimate forecastability of a time series*

Description

A plug-in estimator for the forecastability $\Omega(x_t)$ of a univariate time series x_t.

Usage

```
Omega(series, spectrum.method = c("direct", "multitaper", "lag window",
  "wosa", "mvspec", "ar"), entropy.method = c("MLE"),
  spectrum.estimate = NULL, threshold = 0, smoothing = FALSE, ...)
```

Arguments

`series`	a univariate time series; if multivariate `Omega` works component-wise (i.e. same as using `apply(series, 2, Omega)`).
`spectrum.method`	method to estimate the spectrum; see `method` argument in `mvspectrum`.
`entropy.method`	method to estimate the entropy; see `method` argument in `discrete_entropy`.
`spectrum.estimate`	optionally one can directly pass an estimate of the spectrum (rather than estimating it from `series` within `Omega`).
`threshold`	threshold for the entropy estimator; see `discrete_entropy` for details.
`smoothing`	indicator; see `mvspectrum`.
`...`	optional arguments passed to `spectral_entropy`

Details

The *forecastability* of a stationary process x_t is defined as (see References)

$$\Omega(x_t) = 1 - \frac{-\int_{-\pi}^{\pi} f_x(\lambda) \log f_x(\lambda) d\lambda}{\log 2\pi} \in [0, 1]$$

where $f_x(\lambda)$ is the spectral density of x_t. In particular $\int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1$.

For white noise $\varepsilon_t \sim WN(0, \sigma^2)$ forecastability $\Omega(\varepsilon_t) = 0$; for a sum of sinusoids it equals 100 %.

However, empirically it reaches 100% only if the estimated spectrum has exactly one peak at some ω_j and $\widehat{f}(\omega_k) = 0$ for all $k \neq j$.

In practice, we have T Fourier frequencies which represent a discrete probability distribution. Hence entropy of $f_x(\lambda)$ must be normalized by $\log T$, not by $\log 2\pi$.

Value

A real-value between 0 and 100 (%). 0 means not forecastable (white noise); 100 means perfectly forecastable (a sinusoid).

References

Goerg, G. M. (2013). "Forecastable Component Analysis". Journal of Machine Learning Research (JMLR) W&CP 28 (2): 64-72, 2013. Available at jmlr.org/proceedings/papers/v28/goerg13.html.

See Also

spectral_entropy, discrete_entropy, continuous_entropy

Examples

```
set.seed(1)
nn <- 100
eps <- rnorm(nn)
Omega(eps) # default is direct estimation; no smoothing
Omega(eps, spectrum.method = "wosa") # smoothing makes it closer to 0

xx <- sin(seq_len(nn) * pi / 10)
Omega(xx) # direct (no smoothing of spectrum)
# direct (no smoothing of spectrum) plus thresholding to single out important frequency
Omega(xx, threshold = 1/40)
Omega(xx, spectrum.method = "wosa") # smoothing
Omega(xx, spectrum.method = "wosa", threshold = 1/20) # smoothing and treshold
Omega(xx, spectrum.method = "multitaper") # multitaper smoothing

set.seed(1)
# an AR(1) with phi = 0.5
yy <- arima.sim(n = nn, model = list(ar=0.5))
Omega(yy, spectrum.method = "wosa")
Omega(yy, spectrum.method = "multitaper")
```

```
# an AR(1) with phi = 0.9 is more forecastable
yy <- arima.sim(n = nn, model = list(ar=0.9))
Omega(yy, spectrum.method = "wosa")
Omega(yy, spectrum.method = "multitaper")
```

---

quadratic_form *Computes quadratic form x' A x*

Description

Computes the quadratic form $\mathbf{x}'\mathbf{A}\mathbf{x}$ for an $n \times n$ matrix $\mathbf{A}$ and an n-dimensional vector $\mathbf{x}$, i.e., a wrapper for `t(x) %*% A %*% x`.

Works with real and complex valued matrices/vectors.

Usage

```
quadratic_form(mat, vec)
```

Arguments

`mat`	$n \times n$ matrix (real or complex)
`vec`	$n \times 1$ vector (real or complex)

Value

A real/complex value ($\mathbf{x}'\mathbf{A}\mathbf{x}$).

Examples

```
set.seed(1)
AA <- matrix(1:4, ncol = 2)
bb <- matrix(rnorm(2))
t(bb) %*% AA %*% bb
quadratic_form(AA, bb)
```

SDF2mvspectrum *Estimates spectrum of multivariate time series*

Description

SDF2mvspectrum converts SDF output to a 3D array with number of frequencies in the first dimension and the spectral density matrix in the other two (see Details below).

mvspec2mvspectrum converts output from the mvspec function to the mvspectrum output.

Spectra of a multivariate time series are matrix-valued functions of the frequency λ. mvspectrum estimates these spectra and puts them in a 3D array of dimension $num.freqs \times K \times K$.

Usage

```
SDF2mvspectrum(sdf.output)

mvspec2mvspectrum(mvspec.output)

mvspectrum(series, method = c("multitaper", "direct", "lag window", "wosa",
  "mvspec", "ar"), normalize = FALSE, smoothing = FALSE, Sigma.X = NULL,
  ...)

normalize_mvspectrum(mvspectrum.output, Sigma.X = NULL)
```

Arguments

sdf.output	an object of class "SDF" from SDF
mvspec.output	output from mvspec
series	univariate or multivariate time series
method	method for spectrum estimation; see method argument in SDF or use 'mvspec' for estimation via mvspec from the **astsa** package.
normalize	logical; if TRUE the spectrum will be normalized such that it adds up to 0.5 (for univariate spectra) or to $0.5 \times I_n$ (for multivariate spectra from n time series).
smoothing	logical; if TRUE the spectrum gets additionally smoothed using a nonparametric smoother from gam with an automatically chosen (cross-validation) smoothing parameter.
...	additional arguments passed to SDF or mvspec (e.g., taper)
mvspectrum.output	an object of class "mvspectrum"
Sigma.X	optional; covariance matrix of series

Details

The SDF function only returns the upper diagonal (including diagonal), since spectrum matrices are Hermitian. For fast vectorized computations, however, the full matrices are required. Thus SDF2mvspectrum converts SDF output to a $3D$ array with number of frequencies in the first dimension and the spectral density matrix in the latter two.

SDF2mvspectrum is typically not called by the user, but by mvspectrum.

The mvspec function returns the multivariate spectrum in a $3D$ array with frequencies in the first dimension, whereas mvspectrum in the last. mvspec2mvspectrum simply reshapes the former to the latter array.

Value

mvspec2mvspectrum returns the same object as mvspectrum.

mvspectrum returns a 3D array of dimension $num.freqs \times K \times K$, where

- num.freqs is the number of frequencies
- K is the number of series (columns in series).

normalize_mvspectrum returns a normalized spectrum:

univariate: output adds up to 0.5.

multivariate: output adds up to $0.5\, I_n$, where I_n is the $n \times n$ identity matrix.

References

See References in spectrum, SDF, mvspec.

Examples

```
# use SDF first and then SDF2mvspectrum
set.seed(1)
XX <- cbind(rnorm(100), arima.sim(n = 100, list(ar= 0.9)))
ss3d <- mvspectrum(XX)
dim(ss3d)

ss3d[2,,] # at omega_1; in general complex-valued, but Hermitian
identical(ss3d[2,,], Conj(t(ss3d[2,,]))) # is Hermitian
xx <- rnorm(1000)
var(xx)
mean(mvspectrum(xx, normalize = FALSE, method = "direct")) * 2

mean(mvspectrum(xx, normalize = FALSE, method = "wosa")) * 2
mean(mvspectrum(xx, normalize = TRUE, method = "wosa")) * 2
```

spectral_entropy *Estimates spectral entropy of a time series*

Description

Estimates *spectral entropy* from a univariate (or multivariate) time series.

Usage

```
spectral_entropy(series, spectrum.method = c("multitaper", "direct",
  "lag window", "wosa", "mvspec", "ar"), spectrum.estimate = NULL,
  base = NULL, entropy.method = c("MLE"), threshold = 0,
  smoothing = FALSE, ...)
```

Arguments

series	univariate time series of length T (multivariate also supported but generally not used).
spectrum.method	method for spectrum estimation; see method argument in mvspectrum
base	base of the logarithm in the entropy estimate; default base = NULL. In this case it is set internally to base = T, such that the entropy is bounded above by 1.
entropy.method	method to estimate the entropy from discrete probabilities p_i; here 'probabilities' are the spectral density evaluated at the Fourier frequencies, $\widehat{p}_i = \widehat{f}(\omega_i)$.
spectrum.estimate	optional; one can directly provide an estimate of the spectrum
threshold	values of spectral density below threshold are set to 0; default threshold = 0.
smoothing	logical; if TRUE the spectrum will be smoothed with a nonparametric estimate using gam with an automatically chosen (cross-validation) smoothing parameter.
...	optional additional arguments passed to mvspectrum

Details

The *spectral entropy* equals the Shannon entropy of the spectral density $f_x(\lambda)$ of a stationary process x_t:

$$H_s(x_t) = -\int_{-\pi}^{\pi} f_x(\lambda) \log f_x(\lambda) d\lambda \geq 0,$$

where the density is normalized such that $\int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1$.

An estimate can be obtained using the periodogram (from mvspectrum); thus using discrete, and not continuous entropy.

Value

A non-negative real value for the spectral entropy $H_s(x_t)$.

References

Jerry D. Gibson and Jaewoo Jung (2006). "The Interpretation of Spectral Entropy Based Upon Rate Distortion Functions". IEEE International Symposium on Information Theory, pp. 277-281.

L. L. Campbell, "Minimum coefficient rate for stationary random processes", Information and Control, vol. 3, no. 4, pp. 360 - 371, 1960.

See Also

Omega, discrete_entropy

Examples

```
set.seed(1)
eps <- rnorm(100)
spectral_entropy(eps)

phi.v <- seq(-0.95, 0.95, by = 0.1)
SE <- matrix(NA, ncol = 3, nrow = length(phi.v))


for (ii in seq_along(phi.v)) {
 xx.temp <- arima.sim(n = 1000, list(ar = phi.v[ii]))
 SE[ii, 1] <- spectral_entropy(xx.temp, spectrum.method = "direct")
 SE[ii, 2] <- spectral_entropy(xx.temp, spectrum.method = "multitaper")
 SE[ii, 3] <- spectral_entropy(xx.temp, spectrum.method = "wosa")
}

matplot(phi.v, SE, type = "l", col = 1:3)
legend("bottom", c("direct", "multitaper", "wosa"), lty = 1:3, col = 1:3)


# AR vs MA
SE.arma <- matrix(NA, ncol = 2, nrow = length(phi.v))
SE.arma[, 1] <- SE[, 2]

for (ii in seq_along(phi.v)){
 yy.temp <- arima.sim(n = 1000, list(ma = phi.v[ii]))
 SE.arma[ii, 2] <- spectral_entropy(yy.temp, spectrum.method = "multitaper")
}

matplot(phi.v, SE.arma, type = "l", col = 1:2, xlab = "parameter")
abline(v = 0)
legend("bottom", c("AR(1)", "MA(1)"), lty = 1:2, col = 1:2)
```

---

whiten *whitens multivariate data*

Description

This function transforms a multivariate signal $\mathbf{X}$ with mean $\boldsymbol{\mu}_X$ and covariance matrix Σ_X to a *whitened* signal $\mathbf{U}$ with mean $\mathbf{0}$ and $\Sigma_U = I_n$. Thus it makes the signal contemporaneously uncorrelated.

Same as projecting it on the principal components (PCs) and then scaling all PCs to unit variance.

Usage

```
whiten(data)
```

Arguments

`data`	$T \times n$ array

Details

The whitening is achieved by pre-multiplying $\mathbf{X}$ with the inverse of the *square root* of Σ_X using the eigen-decomposition

$$\Sigma_X = V'\Lambda V,$$

where Λ is an $n \times n$ matrix with the eigenvalues $\lambda_1, \ldots, \lambda_n$ in the diagonal. The *inverse square root* is defined as $\Sigma_X^{-1/2} = V'\Lambda^{-1/2}$, where $\Lambda^{-1/2} = diag(\lambda_1^{-1/2}, \ldots, \lambda_n^{-1/2})$.

Value

A list with the whitened data, the transformation, and other useful quantities

Examples

```
set.seed(1)
XX <- matrix(rnorm(100), ncol = 2)
cov(XX)
UU <- whiten(XX)$U
cov(UU)
```

Index