Package ‘ForeCA’

March 30, 2016

Type Package
Title Forecastable Component Analysis
Version 0.2.4
Date 2016-03-01
Author Georg M. Goerg <im@gmge.org>
Maintainer Georg M. Goerg <im@gmge.org>
URL http://www.gmge.org
Description Implementation of Forecastable Component Analysis ('ForeCA'),
including main algorithms and auxiliary function (summary, plotting, etc.) to
apply 'ForeCA' to multivariate time series data. 'ForeCA' is a novel dimension
reduction (DR) technique for temporally dependent signals. Contrary to other
popular DR methods, such as 'PCA' or 'ICA', 'ForeCA' takes time dependency
explicitly into account and searches for the most "forecastable" signal.
The measure of forecastability is based on the Shannon entropy of the spectral
density of the transformed signal.
Depends R (>= 3.0.0), ifultools (>= 2.0.0)
License GPL-2
Imports MASS, sapa, graphics, reshape2, utils
Suggests astsa, mgcv, nlme (>= 3.1-64), testthat (>= 0.9.0), rSFA,
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2016-03-30 08:14:22

R topics documented:

ForeCA-package .......................................................... 2
common-arguments .................................................... 3
complete-controls ...................................................... 4
continuous_entropy ..................................................... 5
Description

Forecastable Component Analysis (ForeCA) is a novel dimension reduction technique for multivariate time series $X_t$. ForeCA finds a linear combination $y_t = X_t \mathbf{v}$ that is easy to forecast. The measure of forecastability $\Omega(y_t)$ (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 $X_t$.

Please consult the NEWS file for a list of changes to previous versions of this package.

Author(s)

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

References


Examples

```r
XX <- ts(diff(log(EuStockMarkets)))
Omega(XX)

plot(log10(lynx))
Omega(log10(lynx))
```
common-arguments

List of common arguments

Description

Common arguments used in several functions in this package.

Arguments

- **series**: a $T \times K$ array with $T$ observations from the $K$-dimensional time series $X_t$. Can be a matrix, data.frame, or a multivariate ts object.

- **U**: a $T \times K$ array with $T$ observations from the $K$-dimensional whitened (whiten) time series $U_t$. Can be a matrix, data.frame, or a multivariate ts object.

- **mvspectrum.output**: an object of class "mvspectrum" representing the multivariate spectrum of $X_t$ (not necessarily normalized).

- **f.U**: multivariate spectrum of class 'mvspectrum' with normalize = TRUE.

- **algorithm.control**: list; control settings for any iterative ForeCA algorithm. See complete_algorithm_control for details.

- **entropy.control**: list; control settings for entropy estimation. See complete_entropy_control for details.

- **spectrum.control**: list; control settings for spectrum estimation. See complete_spectrum_control for details.

- **entropy.method**: string; method to estimate the entropy from discrete probabilities $p_i$; here probabilities are the spectral density evaluated at the Fourier frequencies, $\hat{p}_i = \hat{f}(\omega_i)$.

- **spectrum.method**: string; method for spectrum estimation; see method argument in mvspectrum.

- **threshold**: numeric; 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 and an exponential family (with link = log). Only works for univariate spectrum. The smoothing parameter is chosen automatically using generalized cross-validation (see gam for details). Default: FALSE.

- **base**: logarithm base; entropy is measured in “nats” for base = exp(1); in “bits” if base = 2 (default).
**complete-controls**

**ComPLETES several control settings**

**Description**
ComPLETES algorithm, entropy, and spectrum control lists.

**Usage**
```r
complete_algorithm_control(algorithm.control = list(max.iter = 50, num.starts = 10, tol = 0.001, type = "EM"))
```
```r
complete_entropy_control(entropy.control = list(base = NULL, method = "MLE", prior.probs = NULL, prior.weight = 0.001, threshold = 0), num.outcomes)
```
```r
complete_spectrum_control(spectrum.control = list(kernel = NULL, method = c("wosa", "direct", "multitaper", "mvspec", "ar", "pgram"), smoothing = FALSE))
```

**Arguments**
- `algorithm.control` list; control parameters for any *iterative* ForeCA algorithm.
- `entropy.control` list; control settings for entropy estimation.
- `num.outcomes` positive integer; number of outcomes for the discrete probability distribution. Must be specified (no default value).
- `spectrum.control` list; control settings for spectrum estimation.

**Value**
A list with fully specified algorithm, entropy, or spectrum controls. Default values are only added if the input `{spectrum, entropy, algorithm}.control` list does not already set this value.

- `complete_algorithm_control` returns a list containing:
  - `max.iter` maximum number of iterations; default: 50.
  - `num.starts` number of random starts to avoid local optima; default: 10.
  - `tol` tolerance for when convergence is reached in any *iterative* ForeCA algorithm; default: 1e-03.
  - `type` string; type of algorithm. Default: 'EM'.

- `complete_entropy_control` returns a list with:
  - `base` logarithm base for the entropy.
  - `method` string; method to estimate entropy; default: "MLE".
continuous_entropy

prior.probs  prior distribution; default: uniform rep(1 / num.outcomes, num.outcomes).

prior.weight weight of the prior distribution; default: 1e-3.

threshold  non-negative float; set probabilities below threshold to zero; default: 0.

complete_spectrum_control returns a list containing:

kernel  R function; function to weigh each Fourier frequency λ; default: NULL (no re-weighting).

method  string; method to estimate the spectrum; default: 'wosa' if sapa is installed, 'mvspec' if only astsa is installed, and 'pgram' if neither is installed.

smoothing  logical; default: FALSE.

Available methods for spectrum estimation are (alphabetical order)

"ar"  autoregressive spectrum fit via spec.ar; only for univariate time series.
"direct"  raw periodogram using SDF.
"multitaper"  tapering the periodogram using SDF.
"mvspec"  smoothed estimate using mvspec; many tuning parameters are available – they can be passed as additional arguments (...) to mvspec.
"pgram"  uses mvpgm; is the same as the 'direct' method, but does not rely on the SDF package.
"wosa"  Welch overlapping segment averaging (WOSA) using SDF.

Setting smoothing = TRUE will smooth the estimated spectrum (again); this option is only available for univariate time series/spectra.

See Also

mvspectrum, discrete_entropy, continuous_entropy

Description

Computes the Shannon entropy \( H(p) \) for a continuous probability density function (pdf) \( p(x) \) using numerical integration.

Usage

continuous_entropy(pdf, lower, upper, base = 2)
Arguments

- `pdf` R function for the pdf \( p(x) \) of a RV \( X \sim p(x) \). This function must be non-negative and integrate to 1 over the interval \([\text{lower}, \text{upper}]\).
- `lower`, `upper` lower and upper integration limit. pdf must integrate to 1 on this interval.
- `base` logarithm base; entropy is measured in “nats” for base = \( \exp(1) \); in “bits” if base = 2 (default).

Details

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

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

Contrary to discrete RVs, continuous RVs can have negative entropy (see Examples).

Value

scalar; entropy value (real).

Since `continuous_entropy` uses numerical integration (\texttt{integrate}) convergence is not guaranteed (even if integral in definition of \( \mathcal{H}(p) \) exists). Issues a warning if \texttt{integrate} does not converge.

See Also

discrete_entropy

Examples

```r
# 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)

# Same, but for U(-1, 1)
my_density <- function(x){
  dunif(x, -1, 1)
}
calcul_s_entr(my_density, -1, 1) # = log(upper - lower)

# 'triangle' distribution
continuous_entropy(function(x) x, 0, sqrt(2))
```
discrete_entropy

**Shannon entropy for discrete pmf**

**Description**

Computes the Shannon entropy \( H(p) = -\sum_{i=1}^{n} p_i \log p_i \) of a discrete RV \( X \) taking values in \( \{x_1, \ldots, x_n\} \) with probability mass function (pmf) \( P(X = x_i) = p_i \) with \( p_i \geq 0 \) for all \( i \) and \( \sum_{i=1}^{n} p_i = 1 \).

**Usage**

```r
discrete_entropy(probs, base = 2, method = c("MLE"), threshold = 0,
                  prior.probs = rep(1/length(probs), length = length(probs)),
                  prior.weight = 0)
```

**Arguments**

- `probs` numeric; probabilities (empirical frequencies). Must be non-negative and add up to 1.
- `base` logarithm base; entropy is measured in “nats” for base = \( \exp(1) \); in “bits” if base = 2 (default).
- `method` string; method to estimate entropy; see Details below.
- `threshold` numeric; frequencies below threshold are set to 0; default threshold = 0, i.e., no thresholding. If `prior.weight > 0` then thresholding will be done *before* smoothing.
- `prior.probs` optional; only used if `prior.weight > 0`. Add a prior probability distribution to `probs`. By default it uses a uniform distribution putting equal probability on each outcome.
- `prior.weight` numeric; how much weight does the prior distribution get in a mixture model between data and prior distribution? Must be between 0 and 1. Default: 0 (no prior).

**Details**

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

\[
\hat{H}(p) = -\sum_{i=1}^{n} \hat{p}_i \log \hat{p}_i.
\]

If `prior.weight > 0`, then it mixes the observed proportions \( \hat{p}_i \) with a prior distribution

\[
\hat{p}_i \leftarrow (1 - \lambda) \cdot \hat{p}_i + \lambda \cdot \text{prior}_i, \quad i = 1, \ldots, n,
\]

where \( \lambda \in [0, 1] \) is the `prior.weight` parameter. By default the prior is a uniform distribution, i.e., \( \text{prior}_i = \frac{1}{n} \) for all \( i \).

Note that this plugin estimator is biased. See References for an overview of alternative methods.
Value

numeric; non-negative real value.

References


See Also

continuous_entropy

Examples

probs.tmp <- rexp(5)
probs.tmp <- sort(probs.tmp / sum(probs.tmp))

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

matplot(cbind(probs.tmp, unif.distr), pch = 19,
ylab = "P(X = k)", xlab = "k")
matlines(cbind(probs.tmp, unif.distr))
legend("topleft", c("non-uniform", "uniform"), pch = 19,
lty = 1:2, col = 1:2, box.lty = 0)

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

foreca(series, n.comp = 2, algorithm.control = list(type = "EM"), ...)

foreca.one_weightvector(U, f.U = NULL, spectrum.control = list(),
entropy.control = list(), algorithm.control = list(),
keep.all.optima = FALSE, dewhitening = NULL, ...)

foreca.multiple_weightvectors(U, spectrum.control = list(),
entropy.control = list(), algorithm.control = list(), n.comp = 2,
plot = FALSE, dewhitening = NULL, ...)

Arguments

series                        a $T \times K$ array with $T$ observations from the $K$-dimensional time series $X_t$. Can be a matrix, data.frame, or a multivariate ts object.
n.comp                        positive integer; number of components to be extracted. Default: 2.
algorithm.control             list; control settings for any iterative ForeCA algorithm. See complete_algorithm_control for details.
...                          additional arguments passed to available ForeCA algorithms.
U                             a $T \times K$ array with $T$ observations from the $K$-dimensional whitened (whiten) time series $U_t$. Can be a matrix, data.frame, or a multivariate ts object.
f.U                           multivariate spectrum of class 'mvspectrum' with normalize = TRUE.
spectrum.control             list; control settings for spectrum estimation. See complete_spectrum_control for details.
entropy.control              list; control settings for entropy estimation. See complete_entropy_control for details.
keep.all.optima              logical; if TRUE, it keeps the optimal solutions of each random start. Default: FALSE (only returns the best solution).
dewhitening                  optional; if provided (returned by whiten) then it uses the dewhiten transformation to obtain the original series $X_t$ and it uses that vector (normalized) as the initial weightvector which corresponds to the series $X_{t,i}$ with large $\Omega_i$.
plot                         logical; if TRUE a plot of the current optimal solution $w_i^*$ will be shown and updated for each iteration $i = 1, ..., n$.comp of any iterative algorithm. Default: FALSE.

Value

An object of class foreca, which is similar to the output from princomp, with the following components (amongst others):

- center: sample mean $\hat{\mu}_X$ of each series,
- whitening: whitening matrix of size $K \times K$ from whiten: $U_t = (X_t - \hat{\mu}_X) \cdot \text{whitening}$; note that $X_t$ is centered prior to the whitening transformation,
- weightvectors: orthonormal matrix of size $K \times n.\text{comp}$, which converts whitened data to $n.\text{comp}$ forecastable components (ForeCs) $\mathbf{F}_t = \mathbf{U}_t \cdot \text{weightvectors}$.
- loadings: combination of whitening $\times$ weightvectors to obtain the final loadings for the original data: $\mathbf{F}_t = (\mathbf{X}_t - \bar{\mathbf{X}}) \cdot \text{whitening} \cdot \text{weightvectors}$; again, it centers $\mathbf{X}_t$ first.
- loadings.normalized: normalized loadings (unit norm). Note though that if you use these normalized loadings the resulting signals do not have variance 1 anymore.
- scores: $n.\text{comp}$ forecastable components $\mathbf{F}_t$. They have mean 0, variance 1, and are uncorrelated.
- Omega: forecastability score of each ForeC of $\mathbf{F}_t$.

ForeCs are ordered from most to least forecastable (according to Omega).

**Warning**

Estimating Omega directly from the ForeCs $\mathbf{F}_t$ can be different to the reported $\hat{\Omega}$ Omega estimates from foreca. Here is why:

In theory $f_y(\lambda)$ of a linear combination $y_t = \mathbf{X}_t w$ can be analytically computed from the multivariate spectrum $f_X(\lambda)$ by the quadratic form $f_y(\lambda) = w' f_X(\lambda) w$ for all $\lambda$ (see spectrum_of_linear_combination).

In practice, however, this identity does not hold always exactly since (often data-driven) control settings for spectrum estimation are not identical for the high-dimensional, noisy $\mathbf{X}_t$ and the combined univariate time series $y_t$ (which is usually more smooth, less variable). Thus estimating $\hat{f}_y$ directly from $y_t$ can give slightly different estimates to computing it as $w' \hat{f}_X w$. Consequently also $\hat{\Omega}$ estimates can be different.

In general, these differences are small and have no relevant implications for estimating ForeCs. However, especially for rare occasions, the obtained ForeCs can have smaller Omega than the maximum Omega of the original series. In such a case users should not re-estimate $\Omega$ from the resulting ForeCs $\mathbf{F}_t$, but access them via $\hat{\Omega}$ provided by ‘foreca’ output (the univariate estimates are stored in $\hat{\Omega}_\text{univ}$).

**References**


**Examples**

```r
XX <- diff(log(EuStockMarkets[c(100:200),])) * 100
plot(ts(XX))

# Not run:
ff <- foreca(XX[,1:4], n.comp = 2, plot = TRUE)
ff
summary(ff)
plot(ff)

# End(Not run)
```
### Description

A collection of S3 methods for estimated ForeCA results (class "foreca").

- `summary.foreca` computes summary statistics.
- `print.foreca` prints a human-readable summary in the console.
- `biplot.foreca` shows a biplot of the ForeCA loadings (wrapper around `biplot.princomp`).
- `plot.foreca` shows biplots, screeplots, and white noise tests.

### Usage

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

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

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

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

- lag: integer; how many lags to test in `Box.test`; default: 10.
- alpha: significance level for testing white noise in `Box.test`; default: 0.05.
- ... additional arguments passed to `biplot.princomp`, `biplot.default`, `plot`, or `summary`.
- x, object: an object of class "foreca".

Examples

# see examples in 'foreca'

Description

foreca.EM.Em_one_weightvector relies on several auxiliary functions:
foreca.EM.Em_E_step computes the spectral density of $y_t = U_t w$ given the weightvector $w$ and the normalized spectrum estimate $f_0$. A wrapper around `spectrum_of_linear_combination`.
foreca.EM.Em_M_step computes the minimizing eigenvector ($\rightarrow \hat{w}_{i+1}$) of the weighted covariance matrix, where the weights equal the negative logarithm of the spectral density at the current $\hat{w}_i$.
foreca.EM.Em.E_and_M_step is a wrapper around foreca.EM.Em_E_step followed by foreca.EM.Em_M_step.
foreca.EM.Em.h evaluates (an upper bound of) the entropy of the spectral density as a function of $w_i$ (or $w_{i+1}$). This is the objective function that should be minimized.

Usage

foreca.EM.Em_E_step(f.U, weightvector)

foreca.EM.Em_M_step(f.U, f.current, minimize = TRUE, entropy.control = list())

foreca.EM.Em.E_and_M_step(weightvector, f.U, minimize = TRUE, entropy.control = list())

foreca.EM.Em.h(weightvector.new, f.U, weightvector.current = weightvector.new, f.current = NULL, entropy.control = list(), return.negative = FALSE)

Arguments

- f.U: multivariate spectrum of class 'mvspectrum' with normalize = TRUE.
- weightvector: numeric; weights w for $y_t = U_t w$. Must have unit norm in $l^2$.
- f.current: numeric; spectral density estimate of $y_t = U_t w$ for the current estimate $\hat{w}_i$ (required for foreca.EM.Em_M_step; optional for foreca.Em.h).
minimize logical; if TRUE (default) it returns the eigenvector corresponding to the smallest eigenvalue; otherwise to the largest eigenvalue.

entropy.control list; control settings for entropy estimation. See complete_entropy_control for details.

weightvector.new weightvector \( \hat{w}_{i+1} \) of the new iteration \((i+1)\).

weightvector.current weightvector \( \hat{w}_i \) of the current iteration \((i)\).

return.negative logical; if TRUE it returns the negative spectral entropy. This is useful when maximizing forecastibility which is equivalent (up to an additive constant) to maximizing negative entropy. Default: FALSE.

Value

foreca.EM.E_step returns the normalized univariate spectral density (normalized such that its sum equals 0.5).

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

- matrix: weighted covariance matrix, where the weights are the negative log of the spectral density. If density is estimated by discrete probabilities, then this matrix is positive semi-definite, since \(-\log(p) \geq 0\) for \(p \in [0, 1]\). See weightvector.entropy.wcov.
- vector: minimizing (or maximizing if minimize = FALSE) eigenvector of matrix,
- value: corresponding eigenvalue.

Contrary to foreca.EM.M_step, foreca.EM.E_and_M_step only returns the optimal weightvector as a numeric.

foreca.EM.h returns non-negative real value (see References for details):

- entropy, if weightvector.new = weightvector.current,
- an upper bound of that entropy for weightvector.new, otherwise.

See Also

weightvector.entropy.wcov

Examples

```r
XX <- diff(log(EuStockMarkets)) * 100
UU <- whiten(XX)$U
ff <- mvspectrum(UU, 'wosa', normalize = TRUE)

ww0 <- initialize_weightvector(num.series = ncol(XX), method = 'rnorm')

f.ww0 <- foreca.EM.E_step(ff, ww0)
plot(f.ww0, type = "l")

one.step <- foreca.EM.M_step(ff, f.ww0,
```
foreca.EM.one_weightvector

EM-like algorithm to estimate optimal ForeCA transformation

Description

foreca.EM.one_weightvector finds the optimal weightvector \( \mathbf{w}^* \) that gives the most forecastable signal \( y_i^* = \mathbf{U}_i \mathbf{w}^* \) using an EM-like algorithm (see References).
### Usage

```r
foreca.EM.one_weightvector(U, f.U = NULL, spectrum.control = list(),
                            entropy.control = list(), algorithm.control = list(),
                            init.weightvector = initialize_weightvector(num.series = ncol(U), method =
                            "rnorm"), ...)
```

### Arguments

- **U**: a $T \times K$ array with $T$ observations from the $K$-dimensional whitened (whiten) time series $U_t$. Can be a matrix, data.frame, or a multivariate ts object.
- **f.U**: multivariate spectrum of class 'mvspectrum’ with normalize = TRUE.
- **spectrum.control**: list; control settings for spectrum estimation. See `complete_spectrum_control` for details.
- **entropy.control**: list; control settings for entropy estimation. See `complete_entropy_control` for details.
- **algorithm.control**: list; control settings for any iterative ForeCA algorithm. See `complete_algorithm_control` for details.
- **init.weightvector**: numeric; starting point $w_0$ for several iterative algorithms. By default it uses a (normalized) random vector from a standard Normal distribution (see `initialize_weightvector`).
- **...**: other arguments passed to `mvspectrum`

### Value

A list with useful quantities like the optimal weighvector, the corresponding signal, and its forecastability.

### References


### See Also

- `foreca.one_weightvector`, `foreca.EM-aux`

### Examples

```r
## Not run:
XX <- diff(log(EuStockMarkets)[100:200,]) * 100
one.weight <- foreca.EM.one_weightvector(whiten(XX)$U,
                              spectrum.control =
                              list(method = "wosa"))

## End(Not run)
```
Description

S3 methods for the one weightvector optimization in ForeCA (class "foreca.one_weightvector").

summary.foreca.one_weightvector computes summary statistics.

plot.foreca.one_weightvector shows the results of an (iterative) algorithm that obtained the i-th optimal a weightvector \( w_i^* \). It shows trace plots of the objective function and the weightvector, and a time series plot of the transformed signal \( y_t^* \) along with its spectral density estimate \( \hat{f}_{y}(\omega_j) \).

Usage

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

```r
## S3 method for class 'foreca.one_weightvector'
plot(x, main = "", cex.lab = 1.1, ...)
```

Arguments

- `lag` integer; how many lags to test in `Box.test`; default: 10.
- `alpha` significance level for testing white noise in `Box.test`; default: 0.05.
- `...` additional arguments passed to `plot`, or `summary`.
- `x`, `object` an object of class "foreca.one_weightvector".
- `main` an overall title for the plot: see `title`.
- `cex.lab` size of the axes labels.

Examples

```r
# see examples in 'foreca.one_weightvector'
```
**initialize_weightvector**

*Initialize weightvector for iterative ForeCA algorithms*

**Description**

initialize_weightvector returns a unit norm (in $\ell^2$) vector $w_0 \in \mathbb{R}^K$ that can be used as the starting point for any iterative ForeCA algorithm, e.g., foreca.EM.one_weightvector. Several quickly computable heuristics are available via the method argument.

**Usage**

```r
initialize_weightvector(U = NULL, f.U = NULL, num.series = ncol(U),
method = c("rnorm", "max", "SFA", "PCA", "rcauchy", "runif", "SFA.slow",
"SFA.fast", "PCA.large", "PCA.small"), seed = sample(1e+06, 1), ...)
```

**Arguments**

- **U** a $T \times K$ array with $T$ observations from the $K$-dimensional whitened (whiten) time series $U_i$. Can be a matrix, data.frame, or a multivariate ts object.
- **f.U** multivariate spectrum of class 'mvspectrum' with normalize = TRUE.
- **num.series** positive integer; number of time series $K$ (determines the length of the weightvector). If num.series = 1 it simply returns a $1 \times 1$ array equal to 1.
- **method** string; which heuristics should be used to generate a good starting $w_0$? Default: "rnorm"; see Details.
- **seed** non-negative integer; seed for random initialization which will be returned for reproducibility. By default it sets a random seed.
- **...** additional arguments

**Details**

The method argument specifies the heuristics that is used to get a good starting vector $w_0$:

- "max" vector with all 0s, but a 1 at the position of the maximum forecastable series in U.
- "rcauchy" random start using rcauchy(k).
- "rnorm" random start using rnorm(k, 0, 1).
- "runif" random start using runif(k, -1, 1).
- "PCA.large" first eigenvector of PCA (largest variance signal).
- "PCA.small" last eigenvector of PCA (smallest variance signal).
- "PCA" checks both small and large, and chooses the one with higher forecastability as computed by Omega..
- "SFA.fast" last eigenvector of SFA (fastest signal).
- "SFA.slow" first eigenvector of SFA (slowest signal).
• "SFA" checks both slow and fast, and chooses the one with higher forecastability as computed by Omega.

Each vector has length K and is automatically normalized to have unit norm in $\ell^2$.

For the 'SFA*' methods see sfa. Note that maximizing (or minimizing) the lag 1 auto-correlation does not necessarily yield the most forecastable signal, but it’s a good start.

Value

numeric; a vector of length $K$ with unit norm in $\ell^2$.

Examples

```r
XX <- diff(log(EuStockMarkets))
## Not run:
initialize_weightvector(U = XX, method = "SFA")

## End(Not run)
initialize_weightvector(num.series = ncol(XX), method = "rnorm")
```

---

**mvspectrum**

Estimates spectrum of multivariate time series

Description

The spectrum of a multivariate time series is a matrix-valued function of the frequency $\lambda \in [-\pi, \pi]$, which is symmetric/Hermitian around $\lambda = 0$.

mvspectrum estimates it and returns a 3D array of dimension `num.freqs x K x K`. Since the spectrum is symmetric/Hermitian around $\lambda = 0$ it is sufficient to store only positive frequencies. In the implementation in this package we thus usually consider only positive frequencies (omitting 0); `num.freqs` refers to the number of positive frequencies only.

normalize_mvspectrum normalizes the spectrum such that it adds up to 0.5 over all positive frequencies (by symmetry it will add up to 1 over the whole range – thus the name `normalize`).

For a $K$-dimensional time series it adds up to a Hermitian $K \times K$ matrix with 0.5 in the diagonal and imaginary elements (real parts equal to 0) in the off-diagonal. Since it is Hermitian the mvspectrum will add up to the identity matrix over the whole range of frequencies, since the off-diagonal elements are purely imaginary (real part equals 0) and thus add up to 0.

check_mvspectrum_normalized checks if the spectrum is normalized (see normalize_mvspectrum for the requirements).

mvpgram computes the multivariate periodogram estimate using bare-bone multivariate fft (mvfft). Please use mvspectrum(..., method = 'pgram') instead of mvpgram directly.

This function is merely included to have one method that does not require the astsa nor the sapa R packages. However, it is strongly encouraged to install either one of them to get (much) better estimates. See Details.

get_spectrum_from_mvspectrum extracts the spectrum of one time series from an "mvspectrum" object by taking the i-th diagonal entry for each frequency.
spectrum_of_linear_combination computes the spectrum of the linear combination $y_t = X_t \beta$ of $K$ time series $X_t$. This can be efficiently computed by the quadratic form

$$f_y(\lambda) = \beta' f_X(\lambda) \beta \geq 0,$$

for each $\lambda$. This holds for any $\beta$ (even $\beta = 0$ - not only for $||\beta||_2 = 1$. For $\beta = e_i$ (the $i$-th basis vector) this is equivalent to get_spectrum_from_mvspectrum(..., which = i).

Usage

mvspectrum(series, method = c("pgram", "multitaper", "direct", "wosa", "mvspec", "ar"), normalize = FALSE, smoothing = FALSE, ...)

normalize_mvspectrum(mvspectrum.output)

check_mvspectrum_normalized(f.U, check.attribute.only = TRUE)

mvpgram(series)

gt_spectrum_from_mvspectrum(mvspectrum.output, which = seq_len(dim(mvspectrum.output)[2]))

spectrum_of_linear_combination(mvspectrum.output, beta)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>series</td>
<td>a $T \times K$ array with $T$ observations from the $K$-dimensional time series $X_t$. Can be a matrix, data.frame, or a multivariate ts object.</td>
</tr>
<tr>
<td>method</td>
<td>string; method for spectrum estimation; see method argument in SDF (in the sapa package); use &quot;mvspec&quot; to use mvspec (astsa package); or use &quot;pgram&quot; to use spec.pgram.</td>
</tr>
<tr>
<td>normalize</td>
<td>logical; if TRUE the spectrum will be normalized (see Value below for details).</td>
</tr>
<tr>
<td>smoothing</td>
<td>logical; if TRUE the spectrum will be smoothed with a nonparametric estimate using gam and an exponential family (with link = log). Only works for univariate spectrum. The smoothing parameter is chosen automatically using generalized cross-validation (see gam for details). Default: FALSE.</td>
</tr>
<tr>
<td>mvspectrum.output</td>
<td>an object of class &quot;mvspectrum&quot; representing the multivariate spectrum of $X_t$ (not necessarily normalized).</td>
</tr>
<tr>
<td>f.U</td>
<td>multivariate spectrum of class 'mvspectrum' with normalize = TRUE.</td>
</tr>
<tr>
<td>check.attribute.only</td>
<td>logical; if TRUE it checks the attribute only. This is much faster (it just needs to look up one attribute value), but it might not surface silent bugs. For sake of performance the package uses the attribute version by default. However, for testing/debugging the full computational version can be used.</td>
</tr>
<tr>
<td>which</td>
<td>integer(s); the spectrum of which series should be extracted. By default, it returns all univariate spectra as a matrix (frequencies in rows).</td>
</tr>
<tr>
<td>beta</td>
<td>numeric; vector $\beta$ that defines the linear combination.</td>
</tr>
<tr>
<td>...</td>
<td>additional arguments passed to SDF or mvspec (e.g., taper)</td>
</tr>
</tbody>
</table>
Details

For an orthonormal time series $U_t$ the raw periodogram adds up to $I_K$ over all (negative and positive) frequencies. Since we only consider positive frequencies, the normalized multivariate spectrum should add up to $0.5 \cdot I_K$ plus a Hermitian imaginary matrix (which will add up to zero when combined with its symmetric counterpart.) As we often use non-parametric smoothing for less variance, the spectrum estimates do not satisfy this identity exactly. `normalize_mvspectrum` thus adjust the estimates so they satisfy it again exactly.

`mvpgram` has no options for improving spectrum estimation whatsoever. It thus yields very noisy (in fact, inconsistent) estimates of the multivariate spectrum $f_X(\lambda)$. If you want to obtain better estimates then please use other methods in `mvspectrum` (this is highly recommended to obtain more reasonable/stable estimates).

Value

`mvspectrum` returns a 3D array of dimension `num.freqs × K × K`, where

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

Note that it also has an attribute “normalized” which is `FALSE` if `normalize = FALSE`; otherwise `TRUE`. See `normalize_mvspectrum` for details.

`normalize_mvspectrum` returns a normalized spectrum over positive frequencies, which:

- **univariate**: adds up to 0.5,
- **multivariate**: adds up to Hermitian $K \times K$ matrix with 0.5 in the diagonal and purely imaginary elements in the off-diagonal.

`check_mvspectrum_normalized` throws an error if spectrum is not normalized correctly.

`get_spectrum_from_mvspectrum` returns either a matrix of all univariate spectra, or one single column (if `which` is specified.)

`spectrum_of_linear_combination` returns a vector with length equal to the number of rows of `mvspectrumNoutput`.

References

See References in `spectrum`, `SDF`, `mvspec`.

Examples

```r
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

ss <- mvspectrum(XX[, 1], smoothing = TRUE)
```
# Not run:
mvspectrum(XX, normalize = TRUE)

## End(Not run)

## S3 methods for multivariate spectrum estimation.

## Description

S3 methods for multivariate spectrum estimation.

`plot.mvspectrum` plots all univariate spectra. Analogous to `spectrum` when `plot = TRUE`. 
mvspectrum2wcov

Usage

```r
## S3 method for class 'mvspectrum'
plot(x, log = TRUE, ...)
```

Arguments

- `x` an object of class "foreca.one_weightvector".
- `log` logical; if TRUE (default), it plots the spectra on log-scale.
- `...` additional arguments passed to `matplot`.

See Also

`get_spectrum_from_mvspectrum`

Examples

```r
# see examples in 'mvspectrum'

SS <- mvspectrum(diff(log(EuStockMarkets)) * 100,
                  spectrum.control = list(method = "multitaper"))
plot(SS, log = FALSE)
```

mvspectrum2wcov

*Compute (weighted) covariance matrix from frequency spectrum*

Description

`mvspectrum2wcov` computes a (weighted) covariance matrix estimate from the frequency spectrum (see Details).

`weightvector2entropy_wcov` computes the weighted covariance matrix using the negative entropy of the univariate spectrum (given the weightvector) as kernel weights. This matrix is the objective matrix for many `foreca.*` algorithms.

Usage

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

weightvector2entropy_wcov(weightvector = NULL, f.U, f.current = NULL,
                          entropy.control = list())
```
Arguments

- **mvspectrum.output**
  - an object of class "mvspectrum" representing the multivariate spectrum of \( X_t \) (not necessarily normalized).

- **kernel.weights**
  - numeric; weights for each frequency. By default uses weights that average out to 1.

- **weightvector**
  - numeric; weights \( w \) for \( y_t = U_t w \). Must have unit norm in \( \ell^2 \).

- **f.U**
  - multivariate spectrum of class 'mvspectrum' with normalize = TRUE.

- **f.current**
  - numeric; spectral density estimate of \( y_t = U_t w \) for the current estimate \( \hat{w}_i \) (required for foreca.M_Step; optional for foreca.M.h).

- **entropy.control**
  - list; control settings for entropy estimation. See `complete_entropy_control` for details.

Details

The covariance matrix of a multivariate time series satisfies the identity

\[
\Sigma_X \equiv \int_{-\pi}^{\pi} S_X(\lambda) d\lambda.
\]

A generalized covariance matrix estimate can thus be obtained using a weighted average

\[
\hat{\Sigma}_X = \int_{-\pi}^{\pi} K(\lambda) S_X(\lambda) d\lambda,
\]

where \( K(\lambda) \) is a kernel symmetric around 0 which averages out to 1 over the interval \([-\pi, \pi]\), i.e.,

\[
\frac{1}{2\pi} \int_{-\pi}^{\pi} K(\lambda) d\lambda = 1.
\]

This allows one to remove or amplify specific frequencies in the covariance matrix estimation.

For ForeCA `mvspectrum2wcov` is especially important as we use

\[
K(\lambda) = -\log f_y(\lambda),
\]

as the weights (their average is not 1!). This particular kernel weight is implemented as a wrapper in `weightvector2entropy_wcov`.

Value

A symmetric \( n \times n \) matrix.

If **kernel.weights** \( \geq 0 \), then it is positive semi-definite; otherwise, it is symmetric but not necessarily positive semi-definite.

See Also

- `mvspectrum`
Examples

```r
nn <- 50
YY <- cbind(rnorm(nn), arima.sim(n = nn, list(ar = 0.9)), rnorm(nn))
XX <- YY %*% matrix(rnorm(9), ncol = 3)  # random mix
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 <- mvsppectrum(XX, "wosa")
Sigma.hat.freq <- mvsppectrum2wcov(SS)

layout(matrix(1:4, ncol = 2))
par(mar = c(2, 2, 1, 1))
plot(c(Sigma.hat/Sigma.hat.freq))
abline(h = 1)

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

# examples for entropy wcov
XX <- diff(log(EuStockMarkets)) * 100
UU <- whiten(XX)
ff <- mvsppectrum(UU, 'wosa', normalize = TRUE)

ww0 <- initialize_weightvector(num.series = ncol(XX), method = 'rnorm')

weightvector2entropy_wcov(ww0, ff,
                          entropy.control =
                          list(prior.weight = 0.1))
```

---

**Omega**

*Estimate forecastability of a time series*

**Description**

An estimator for the forecastability $\Omega(x_t)$ of a univariate time series $x_t$. Currently it uses a discrete plug-in estimator given the empirical spectrum (periodogram).

**Usage**

```
Omega(series = NULL, spectrum.control = list(), entropy.control = list(),
      mvsppectrum.output = NULL)
```
**Omega**

**Arguments**

- **series**
  - a univariate time series; if it is multivariate, then Omega works component-wise (i.e., same as apply(series, 2, Omega)).

- **spectrum.control**
  - list; control settings for spectrum estimation. See complete_spectrum_control for details.

- **entropy.control**
  - list; control settings for entropy estimation. See complete_entropy_control for details.

- **mv spectrum.output**
  - an object of class "mv spectrum" representing the multivariate spectrum of \( X_t \) (not necessarily normalized).

**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 normalized spectral density of \( x_t \). In particular \( \int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1 \).

For white noise \( \varepsilon_t \) 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 \( \omega_j \) and \( \hat{f}(\omega_k) = 0 \) for all \( k \neq j \).

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

Also we can use several smoothing techniques to obtain a less variance estimate of \( f_x(\lambda) \).

**Value**

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

**References**


**See Also**

spectral_entropy, discrete_entropy, continuous_entropy
Examples

```r
nn <- 100
eps <- rnorm(nn)  # white noise has Omega() = 0 in theory
Omega(eps, spectrum.control = list(method = "direct"))
# smoothing makes it closer to 0
Omega(eps, spectrum.control = list(method = "wosa"))

xx <- sin(seq_len(nn) * pi / 10)
Omega(xx, spectrum.control = list(method = "direct"))
Omega(xx, entropy.control = list(threshold = 1/40))
Omega(xx, spectrum.control = list(method = "wosa"),
     entropy.control = list(threshold = 1/20))

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

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

---

### quadratic_form

*Computes quadratic form $x'Ax$*

**Description**

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

`fill_symmetric` and `quadratic_form` work with real and complex valued matrices/vectors.

`fill_hermitian` fills up the lower triangular part (NA) of an upper triangular matrix to its Hermitian (symmetric if real matrix) version, such that it satisfies $A = \bar{A}'$, where $\bar{z}$ is the complex conjugate of $z$. If the matrix is real-valued this makes it simply symmetric.

Note that the input matrix must have a **real-valued** diagonal and NAs in the lower triangular part.

**Usage**

```r
quadratic_form(mat, vec)

fill_hermitian(mat)
```

**Arguments**

- `mat` numeric; $n \times n$ matrix (real or complex).
- `vec` numeric; $n \times 1$ vector (real or complex).
Value
A real/complex value \( x' Ax \).

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

```
AA <- matrix(1:16, ncol = 4)
AA[lower.tri(AA)] <- NA
AA

fill_hermitian(AA)
```

Description

`sfa` performs Slow Feature Analysis (SFA) on a \( K \)-dimensional time series with \( T \) observations.

**Important:** This implementation of SFA is just the most basic version; it is merely included here for convenience in `initialize_weightvector`. If you want to use SFA in R please use the `rSFA` package, which has many more advanced and efficient implementations of SFA. `sfa()` here corresponds to `sfa1` in the `rSFA` package.

Usage
```
sfa(series, ...)
```

Arguments

- `series` a \( T \times K \) array with \( T \) observations from the \( K \)-dimensional time series \( X_t \). Can be a matrix, data.frame, or a multivariate ts object.
- `...` additional arguments

Details

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.

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 $\rho_1 < 0$).

Note though that maximizing (or minimizing) the lag 1 auto-correlation does not necessarily yield the most forecastable signal (as measured by $\Omega$), but it is a good start.

Value

An object of class sfa which inherits methods from princomp. Signals are ordered from slowest to fastest.

References


See Also

initialize_weightvector

Examples

XX <- diff(log(EuStockMarkets[-c(1:100),])) * 100
plot(ts(XX))
ss <- sfa(XX[,1:4])

summary(ss)
plot(ss)
plot(ts(ss$scores))
apply(ss$scores, 2, function(x) acf(x, plot = FALSE)$acf[2])
biplot(ss)

---

**spectral_entropy**

Estimates spectral entropy of a time series

Description

Estimates *spectral entropy* from a univariate (or multivariate) normalized spectral density.

Usage

```r
spectral_entropy(series = NULL, spectrum.control = list(),
                 entropy.control = list(), mvspectrum.output = NULL, ...)
```
Arguments

series  
univariate time series of length \( T \). In the rare case that users want to call this for a multivariate time series, note that the estimated spectrum is in general not normalized for the computation. Only if the original data is whitened, then it is normalized.

spectrum.control  
list; control settings for spectrum estimation. See `complete_spectrum_control` for details.

entropy.control  
list; control settings for entropy estimation. See `complete_entropy_control` for details.

mvspectrum.output  
optional; one can directly provide an estimate of the spectrum of `series`. Usually the output of `mvspectrum`.

...  
additional arguments passed to `mvspectrum`.

Details

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

\[
H_s(x_t) = - \int_{-\pi}^{\pi} f_x(\lambda) \log f_x(\lambda) d\lambda,
\]

where the density is normalized such that \( \int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1 \). An estimate of \( f(\lambda) \) can be obtained by the (smoothed) periodogram (see `mvspectrum`); thus using discrete, and not continuous entropy.

Value

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

References


See Also

Omega, `discrete_entropy`

Examples

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

phi.v <- seq(-.95, .95, by = .1)
```
whiten <- c("wosa", "multitaper", "direct", "pgram")
SE <- matrix(NA, ncol = length(kMethods), nrow = length(phi.v))
for (ii in seq_along(phi.v)) {
  xx.tmp <- arima.sim(n = 200, list(ar = phi.v[ii]))
  for (mm in seq_along(kMethods)) {
    SE[ii, mm] <- spectral_entropy(xx.tmp, spectrum.control = list(method = kMethods[mm]))
  }
}
matplot(phi.v, SE, type = "l", col = seq_along(kMethods))
legend("bottom", kMethods, lty = seq_along(kMethods), col = seq_along(kMethods))

# 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.control = list(method = "multitaper"))
}
matplot(phi.v, SE.arma, type = "l", col = 1:2, xlab = "parameter (phi or theta)",
ylab = "Spectral entropy")
abline(v = 0, col = "blue", lty = 3)
legend("bottom", c("AR(1)", "MA(1)"), lty = 1:2, col = 1:2)

whiten

whitens multivariate data

Description

whiten transforms a multivariate K-dimensional signal X with mean $\mu_X$ and covariance matrix $\Sigma_X$ to a whitened signal U with mean 0 and $\Sigma_U = I_K$. Thus it centers the signal and makes it contemporaneously uncorrelated. See Details.

check_whitened checks if data has been whitened; i.e., if it has zero mean, unit variance, and is uncorrelated.

sqrt_matrix computes the square root B of a square matrix A. The matrix B satisfies $BB = A$.

Usage

whiten(data)

check_whitened(data, check.attribute.only = TRUE)

sqrt_matrix(mat, return.sqrt.only = TRUE, symmetric = FALSE)
whiten

Arguments

- **data** \( n \times K \) array representing \( n \) observations of \( K \) variables.
- **check.attribute.only** logical; if TRUE it checks the attribute only. This is much faster (it just needs to look up one attribute value), but it might not surface silent bugs. For sake of performance the package uses the attribute version by default. However, for testing/debugging the full computational version can be used.
- **mat** a square \( K \times K \) matrix.
- **return.sqrt.only** logical; if TRUE (default) it returns only the square root matrix; if FALSE it returns other auxiliary results (eigenvectors and eigenvalues, and inverse of the square root matrix).
- **symmetric** logical; if TRUE the eigen-solver assumes that the matrix is symmetric (which makes it much faster). This is in particular useful for a covariance matrix (which is used in whiten). Default: FALSE.

Details

whiten uses zero component analysis (ZCA) (aka zero-phase whitening filters) to whiten the data; i.e., it uses the inverse square root of the covariance matrix of \( X \) (see sqrt_matrix) as the whitening transformation. This means that on top of PCA, the uncorrelated principal components are back-transformed to the original space using the transpose of the eigenvectors. The advantage is that this makes them comparable to the original \( X \). See References for details.

The square root of a quadratic \( n \times n \) matrix \( A \) can be computed by using the eigen-decomposition of \( A \)

\[
A = V \Lambda V',
\]

where \( \Lambda \) is an \( n \times n \) matrix with the eigenvalues \( \lambda_1, \ldots, \lambda_n \) in the diagonal. The square root is simply \( B = V \Lambda^{1/2} V' \) where \( \Lambda^{1/2} = diag(\lambda_1^{1/2}, \ldots, \lambda_n^{1/2}) \).

Similarly, the inverse square root is defined as \( A^{-1/2} = V \Lambda^{-1/2} V', \) where \( \Lambda^{-1/2} = diag(\lambda_1^{-1/2}, \ldots, \lambda_n^{-1/2}) \) (provided that \( \lambda_i \neq 0 \)).

Value

whiten returns a list with the whitened data, the transformation, and other useful quantities.

check.whitened throws an error if the input is not whitened, and returns (invisibly) the data with an attribute 'whitened' equal to TRUE. This allows to simply update data to have the attribute and thus only check it once on the actual data (slow) but then use the attribute lookup (fast).

sqrt_matrix returns an \( n \times n \) matrix. If \( A \) is not semi-positive definite it returns a complex-valued \( B \) (since square root of negative eigenvalues are complex).

If return.sqrt.only = FALSE then it returns a list with:

- **values** eigenvalues of \( A \),
- **vectors** eigenvectors of \( A \),
- **sqrt** square root matrix \( B \),
- **sqrt.inverse** inverse of \( B \).
whiten

References


Examples

XX <- matrix(rnorm(100), ncol = 2)  # matrix(runif(4), ncol = 2)
cov(XX)
UU <- whiten(XX)$U
cov(UU)
Index

*Topic bplot
  foreca-utils, 11
  foreca.one_weightvector-utils, 16
  mvspectrum-utils, 21

*Topic iteration
  foreca, 8
  foreca.EM.one_weightvector, 14

*Topic manip
  foreca-utils, 11
  foreca.EM-aux, 12
  foreca.EM-one_weightvector, 14
  foreca-one_weightvector-utils, 16
  initialize_weightvector, 17
  mvspectrum, 18
  mvspectrum-utils, 21
  whiten, 30

*Topic math
  continuous_entropy, 5
  discrete_entropy, 7
  foreca.EM-aux, 12
  Omega, 24
  quadratic_form, 26
  spectral_entropy, 28
  whiten, 30

*Topic optimize
  foreca.EM-one_weightvector, 14

*Topic package
  ForeCA-package, 2

*Topic ts
  mvspectrum, 18
  mvspectrum2w cov, 22
  spectral_entropy, 28

*Topic univar
  continuous_entropy, 5
  discrete_entropy, 7
  Omega, 24
  quadratic_form, 26
  spectral_entropy, 28

*Topic utils
  complete-utilities, 4
  biplot.default, 12
  biplot.foreca(foreca-utils), 11
  biplot.princomp, 11, 12
  Box.test, 12, 16
  check_mvspectrum_normalized
    (mvspectrum), 18
  check_whitened (whiten), 30
  common-arguments, 3
  complete-controllers, 4
  complete_algorithm_control, 3, 9, 15
  complete_algorithm_control
    (complete-controllers), 4
  complete_entropy_control, 3, 9, 13, 15, 23, 25, 29
  complete_entropy_control
    (complete-controllers), 4
  complete_spectrum_control, 3, 9, 15, 25, 29
  complete_spectrum_control
    (complete-controllers), 4
  continuous_entropy, 5, 5, 8, 25
  discrete_entropy, 5, 6, 7, 25, 29
  fill_hermitian (quadratic_form), 26
  ForeCA(ForeCA-package), 2
  foreca, 2, 8
  ForeCA-package, 2
  foreca-utils, 11
  foreca.EM-aux, 12
  foreca.EM.E_and_M_step (foreca.EM-aux), 12
  foreca.EM.E_step (foreca.EM-aux), 12
  foreca.EM.h (foreca.EM-aux), 12
  foreca.EM.M_step (foreca.EM-aux), 12
  foreca.EM-one_weightvector, 14, 17
  foreca.one_weightvector, 15
foreca.one_weightvector-utils, 16

gam, 3, 19
get_spectrum_from_mvspectrum, 22
get_spectrum_from_mvspectrum
  (mvspectrum), 18
initialize_weightvector, 15, 17, 27, 28
matplot, 22
mvfft, 18
mvpgram, 5
mvpgram (mvspectrum), 18
mvspec, 5, 19, 20
mvspectrum, 3, 5, 15, 18, 20, 23, 29
mvspectrum-utils, 21
mvspectrum2wcov, 22
normalize_mvspectrum, 18
normalize_mvspectrum (mvspectrum), 18
Omega, 2, 9, 10, 17, 18, 24, 25, 28, 29
plot, 12, 16
plot.foreca (foreca-utils), 11
plot.foreca.one_weightvector
  (foreca.one_weightvector-utils), 16
plot.mvspectrum (mvspectrum-utils), 21
princomp, 9, 28
print.foreca (foreca-utils), 11
quadratic_form, 26
SDF, 5, 19, 20
sfa, 18, 27
spec.ar, 5
spec.pgram, 19
spectral_entropy, 25, 28
spectrum, 20, 21
spectrum_of_linear_combination, 10, 12
spectrum_of_linear_combination
  (mvspectrum), 18
sqrt_matrix, 31
sqrt_matrix (whiten), 30
summary, 12, 16
summary.foreca (foreca-utils), 11
summary.foreca.one_weightvector
  (foreca.one_weightvector-utils), 16

weightvector2entropy_wcov, 13
weightvector2entropy_wcov
  (mvspectrum2wcov), 22
whiten, 3, 9, 15, 17, 30, 31