Package 'Rssa'

October 12, 2022

Type Package

Title A Collection of Methods for Singular Spectrum Analysis

Version 1.0.5

Depends R (>= 3.1), svd (>= 0.4), forecast

Imports lattice, methods

Suggests testthat (>= 0.7), RSpectra, PRIMME

SystemRequirements fftw (>=3.2)

Author Anton Korobeynikov, Alex Shlemov, Konstantin Usevich, Nina Golyandina

Maintainer Anton Korobeynikov <anton@korobeynikov.info>

Description Methods and tools for Singular Spectrum Analysis including decomposition, forecasting and gap-filling for univariate and multivariate time series.
General description of the methods with many examples can be found in the book Golyandina (2018, <doi:10.1007/978-3-662-57380-8>).
See 'citation(``Rssa'')' for details.

License GPL (>= 2)

URL https://github.com/asl/rssa

BugReports https://github.com/asl/rssa/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-08-22 22:10:02 UTC

R topics documented:

Rssa-package *A collection of methods for singular spectrum analysis*

Description

Singular Spectrum Analysis (SSA, in short) is a modern non-parametric method for the analysis of time series and digital images. This package provides a set of fast and reliable implementations of various routines to perform decomposition, reconstruction and forecasting. A comprehensive description of the methods and functions from Rssa can be found in Golyandina et al (2018). The companion web-site is https://ssa-with-r-book.github.io/.

Details

Typically the use of the package starts with the *decomposition* of the time series using `ssa`. After this a suitable *grouping* of the elementary time series is required. This can be done heuristically, for example, via looking at the plots of the decomposition (`plot`). Alternatively, one can examine the so-called w-correlation matrix (`wcor`). Automatic grouping can be performed by means of `grouping.auto`. In addition, Oblique SSA methods can be used to improve the series separability (`iossa`, `fossa`).

Next step includes the *reconstruction* of the time-series using the selected grouping (`reconstruct`). One ends with frequency estimation (`parestimate`), series forecasting (`forecast`, `rforecast`, `vforecast`) and (if any) gap filling (`gapfill`, `igapfill`).

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques*. Chapman and Hall/CRC. ISBN 1584881941f

Golyandina, N. and Stepanov, D. (2005): *SSA-based approaches to analysis and forecast of multi-dimensional time series*. In Proceedings of the 5th St.Petersburg Workshop on Simulation, June 26-July 2, 2005, St. Petersburg State University, St. Petersburg, 293–298. https://www.gistatgroup.com/gus/mssa2.pdf

Golyandina, N. and Usevich, K. (2009): *2D-extensions of singular spectrum analysis: algorithm and elements of theory*. In Matrix Methods: Theory, Algorithms, Applications. World Scientific Publishing, 450-474.

Korobeynikov, A. (2010): *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

Golyandina, N., Korobeynikov, A. (2012, 2014): *Basic Singular Spectrum Analysis and Forecasting with R*. Computational Statistics and Data Analysis, Vol. 71, Pp. 934-954. https://arxiv.org/abs/1206.6910

Golyandina, N., Zhigljavsky, A. (2013): *Singular Spectrum Analysis for time series*. Springer Briefs in Statistics. Springer.

Shlemov, A. and Golyandina, N. (2014): *Shaped extensions of singular spectrum analysis*. 21st International Symposium on Mathematical Theory of Networks and Systems, July 7-11, 2014. Groningen, The Netherlands. p.1813-1820. https://arxiv.org/abs/1507.05286

Golyandina, N., Korobeynikov, A., Shlemov, A. and Usevich, K. (2015): *Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package*. Journal of Statistical Software, Vol. 67, Issue 2. doi:10.18637/jss.v067.i02

See Also

ssa-input, ssa, decompose, reconstruct, wcor, plot, parestimate, rforecast, vforecast, forecast, iossa, fossa

Examples

```
s <- ssa(co2) # Perform the decomposition using the default window length
summary(s)        # Show various information about the decomposition
plot(s)           # Show the plot of the eigenvalues
r <- reconstruct(s, groups = list(Trend = c(1, 4),
                                  Seasonality = c(2:3, 5:6))) # Reconstruct into 2 series
plot(r, add.original = TRUE)  # Plot the reconstruction

# Simultaneous trend extraction using MSSA

s <- ssa(EuStockMarkets, kind = "mssa")
r <- reconstruct(s, groups = list(Trend = c(1,2)))
plot(r, plot.method = "xyplot", add.residuals = FALSE,
     superpose = TRUE, auto.key = list(columns = 2))
# Trend forecast
f <- rforecast(s, groups = list(Trend = c(1, 2)),
               len = 50, only.new = FALSE)
library(lattice)
xyplot(ts.union(Original = EuStockMarkets, "Recurrent Forecast" = f),
       superpose = TRUE, auto.key = list(columns = 2))
```

AustralianWine *Australian Wine Sales*

Description

Monthly Australian wine sales in thousands of litres from Jan 1980 till Jul 1995. By wine makers in bottles of less than or equal to 1 litre.

Usage

```
data(AustralianWine)
```

Format

A multivariate time series with 187 observations on 7 variables. The object is of class 'mts'.

Source

Hyndman, R.J. Time Series Data Library, http://data.is/TSDLdemo.

`Barbara` *Classical 'Barbara' image (color, wide)*

Description

Classical 'Barbara' image (wide version). 720 x 576 x 3 (color, RGB model), from 0 to 255.

Usage

```
data(Barbara)
```

Format

An integer array of dimension 3.

Source

http://www.hlevkin.com/hlevkin/06testimages.htm

`bforecast` *Perform bootstrap SSA forecasting of the series*

Description

Perform bootstrap SSA forecasting of the one-dimensional series.

Usage

```
## S3 method for class '1d.ssa'
bforecast(x, groups, len = 1, R = 100, level = 0.95,
          type = c("recurrent", "vector"),
          interval = c("confidence", "prediction"),
          only.new = TRUE,
          only.intervals = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
bforecast(x, groups, len = 1, R = 100, level = 0.95,
          type = c("recurrent", "vector"),
          interval = c("confidence", "prediction"),
          only.new = TRUE,
          only.intervals = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
```

Arguments

x	SSA object holding the decomposition
groups	list, the grouping of eigentriples to be used in the forecast
len	the desired length of the forecasted series
R	number of bootstrap replications
level	vector of confidence levels for bounds
type	the type of forecast method to be used during bootstrapping
interval	type of interval calculation
only.new	logical, if 'FALSE' then confidence bounds for the signal as well as prediction are reported
only.intervals	logical, if 'TRUE' then bootstrap method is used for confidence bounds only, otherwise — mean bootstrap forecast is returned as well
...	additional arguments passed to forecasting routines
drop	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)
drop.attributes	logical, if 'TRUE' then the attributes of the input series are not copied to the reconstructed ones
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object

Details

The routine uses the reconstruction residuals in order to calculate their empirical distribution (the residuals are assumed to be stationary). Empirical distribution of the residuals is used to perform bootstrap series simulation. Such bootsrapped series are then extended via selected forecast method. Finally, the distribution of forecasted values is used to calculate bootstrap estimate of series forecast and confidence bounds.

See Section 3.2.1.5 from Golyandina et al (2018) for details.

Value

List of matricies. Each matrix has 1 + 2*length(level) columns and 'len' rows. First column contains the forecasted values, remaining columns — low and upper bootstrap confidence bounds for average forecasted values.

The matrix itself, if length of groups is one and 'drop = TRUE'.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R.* Use R!. Springer, Berlin, Heidelberg.

See Also

Rssa for an overview of the package, as well as, rforecast, vforecast, forecast.

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Produce 24 forecasted values and confidence bounds of the series using
# the first 3 eigentriples as a base space for the forecast.

f <- bforecast(s, groups = list(1:3), len = 24, R = 50)
matplot(f, col = c("black", "red", "red"), type='l')
```

---

cadzow *Cadzow Iterations*

Description

Perform the finite rank approximation of the series via Cadzow iterations

Usage

```
## S3 method for class 'ssa'
cadzow(x, rank, correct = TRUE, tol = 1e-6, maxiter = 0,
         norm = function(x) max(abs(x)),
         trace = FALSE, ..., cache = TRUE)
```

Arguments

x	input SSA object
rank	desired rank of approximation
correct	logical, if 'TRUE' then additional correction as in Gillard et al (2013) is performed
tol	tolerance value used for convergence criteria
maxiter	number of iterations to perform, if zero then iterations are performed until the convergence
norm	distance function used for covergence criterion
trace	logical, indicates whether the convergence process should be traced
...	further arguments passed to reconstruct
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

Cadzow iterations aim to solve the problem of the approximation of the input series by a series of finite rank. The idea of the algorithm is quite simple: alternating projections of the trajectory matrix to Hankel and low-rank matrices are performed which hopefully converge to a Hankel low-rank matrix. See Algorithm 3.10 in Golyandina et al (2018).

Note that the results of one Cadzow iteration with no correction coincides with the result of reconstruction by the leading rank components.

Unfortunately, being simple, the method often yields the solution which is far away from the optimum.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Cadzow J. A. (1988) Signal enhancement a composite property mapping algorithm, IEEE Transactions on Acoustics, Speech, and Signal Processing, 36, 49-62.

Gillard, J. and Zhigljavsky, A. (2013) Stochastic optimization algorithms for Hankel structured low-rank approximation. Unpublished Manuscript. Cardiff School of Mathematics. Cardiff.

See Also

Rssa for an overview of the package, as well as, reconstruct

Examples

```
# Decompose co2 series with default parameters
s <- ssa(co2)
# Now make rank 3 approximation using the Cadzow iterations
F <- cadzow(s, rank = 3, tol = 1e-10)
library(lattice)
xyplot(cbind(Original = co2, Cadzow = F), superpose = TRUE)
# All but the first 3 eigenvalues are close to 0
plot(ssa(F))

# Compare with SSA reconstruction
F <- cadzow(s, rank = 3, maxiter = 1, correct = FALSE)
Fr <- reconstruct(s, groups = list(1:3))$F1
print(max(abs(F - Fr)))

# Cadzow with and without weights
set.seed(3)
N <- 60
L <- 30
K <- N - L + 1
alpha <- 0.1

sigma <- 0.1
signal <- cos(2*pi * seq_len(N) / 10)
x <- signal + rnorm(N, sd = sigma)

weights <- rep(alpha, K)
weights[seq(1, K, L)] <- 1
salpha <- ssa(x, L = L,
              column.oblique = "identity",
              row.oblique = weights)
calpha <- cadzow(salpha, rank = 2)

cz <- cadzow(ssa(x, L = L), rank = 2)

print(mean((cz - signal)^2))
print(mean((calpha - signal)^2))
```

`calc.v` *Calculate Factor Vector(s)*

Description

Generic function for the factor vector calculation given the SSA decomposition.

Usage

```
## S3 method for class 'ssa'
calc.v(x, idx, ...)
## S3 method for class 'cssa'
calc.v(x, idx, ...)
```

Arguments

`x`	SSA object holding the decomposition.
`idx`	indices of the factor vectors to compute.
`...`	additional arguments to 'calc.v'.

Details

Factor vector is a column of the factor matrix V, which is calculated as follows:

$$V = \Sigma^{-1} X^T U,$$

where X is a Hankel trajectory matrix, U is the matrix of eigenvectors and Sigma is a matrix of singular values.

Value

A numeric vector of suitable length (usually depends on SSA method and window length).

See Also

`Rssa` for an overview of the package, as well as, `ssa-object`, `ssa`, `decompose`,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Calculate the 5th factor vector
v <- calc.v(s, 5)
```

cleanup *Cleanup of all cached data from SSA objects*

Description

Function to copy SSA objects

Usage

```
cleanup(x)
```

Arguments

x	object to be cleaned

Details

For the sake of memory efficiency SSA objects hold references to the data, not the data itself. That is why they can hold huge amount of data and passing them by value is still cheap.

Also, SSA routines tend to save some intermediate information which can be used later inside SSA object. This includes (but not limited to) elementary series, etc.

cleanup call deletes all pre-cached stuff freeing memory necessary for calculations.

clone *Cloning of SSA objects*

Description

Function to copy SSA objects

Usage

```
## S3 method for class 'ssa'
clone(x, copy.storage = TRUE, copy.cache = TRUE, ...)
```

Arguments

x	object to be cloned
copy.storage	enable/disable copying of the internal storage
copy.cache	enable/disable copying of the set of pre-cached elementary series
...	additional arguments to clone

Details

For the sake of memory efficiency SSA objects hold references to the data, not the data itself. That is why they can hold huge amount of data and passing them by value is still cheap.

However, this means that one cannot safely copy the object using normal assignment operator, since freeing of references in one object would yield stale references in another. The `clone` method provides safe 'deep copy' of SSA objects.

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2);
# Perform 'normal copy' of SSA object
s1 <- s;
# Perform 'deep copy' of SSA object
s2 <- clone(s);
# Add some data to 's'
reconstruct(s);
# Now 's1' also contains this data, but 's2' - not
summary(s1);
summary(s2);
```

---

clplot *Ratio of complete lag vectors in dependence on window length*

Description

Function to plot the dependence of ratios of complete lagged vectors on window lengths.

Usage

```
clplot(x, ...)
```

Arguments

`x`	input series
`...`	further arguments passed to plotting functions

Details

The function plots the dependence of ratios of complete lagged vectors on window lengths. This information can be used for the choice of window length, since only complete lagged vectors are used for construction of the SVD expansion in SSA. See page 89 (Chapter 2) in Golyandina et al (2018).

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R.* Use R!. Springer, Berlin, Heidelberg.

See Also

Rssa for an overview of the package, as well as, igapfill, gapfill summarize.gaps,

---

decompose *Perform SSA Decomposition*

Description

Performs the SSA decomposition.

Usage

```
## S3 method for class 'ssa'
decompose(x, neig = NULL, ..., force.continue = FALSE)
## S3 method for class 'toeplitz.ssa'
decompose(x, neig = NULL, ..., force.continue = FALSE)
## S3 method for class 'cssa'
decompose(x, neig = NULL, ..., force.continue = FALSE)
```

Arguments

x	SSA object holding the decomposition.
neig	number of desired eigentriples or 'NULL' for default value (minimum from 50 and trajectory space dimension).
...	additional arguments passed to SVD routines.
force.continue	logical, if TRUE then continuation of the decomposition is explicitly requested

Details

This is the main function which does the decomposition of the SSA trajectory matrix. Depending on the SVD method selected in the ssa different SVD implementations are called. This might be the ordinary full SVD routines or fast methods which exploit the Hankel / Toeplitz / Hankel with Hankel blocks matrix structure and allow the calculation of first few eigentriples.

Some SVD methods support continuation of the decomposition: if the 'ssa' object already holds some decomposition and more eigentriples are requested, then the decomposition continues using the current values as a starting point reducing the computation time dramatically.

Value

The SSA object.

Note

Usually there is no need to call this function directly. Call to ssa does the decomposition in the end. Other functions do the decomposition when necessary.

See Also

Rssa for an overview of the package, as well as, svd, ssa.

Examples

```
# Decompose 'co2' series with default parameters and decomposition turned off.
s <- ssa(co2, force.decompose = FALSE, svd.method = "nutrlan")
# Perform the decomposition
decompose(s, neig = 50)
# Continue the decomposition
decompose(s, neig = 100)
```

---

eossa *ESPRIT-based O-SSA nested decomposition*

Description

Perform ESPRIT-based O-SSA (EOSSA) algorithm.

Usage

```
## S3 method for class 'ssa'
eossa(x, nested.groups, k = 2,
      subspace = c("column", "row"),
      dimensions = NULL,
      solve.method = c("ls", "tls"),
      beta = 8,
      ...)
```

Arguments

x	SSA object holding SSA decomposition
nested.groups	list or named list of numbers of eigentriples from full decomposition, describes elementary components for EOSSA nested redecomposition
k	the number of components in desired resultant decomposition
subspace	which subspace will be used for oblique matrix construction
dimensions	a vector of dimension indices to construct shift matrices along. 'NULL' means all dimensions
solve.method	approximate matrix equation solving method, 'ls' for least-squares, 'tls' for total-least-squares.
beta	In multidimensional (nD) case, coefficient(s) in convex linear combination of shifted matrices. The length of beta should be ndim - 1, where ndim is the number of independent dimensions. If only one value is passed, it is expanded to a geometric progression.
...	additional arguments passed to decompose routines

Details

EOSSA is an experimental signal separation method working in Nested Oblique SSA setting. As opposed to iossa, this method does not require initial approximate decomposition. Moreover, it can be used for initial decomposition construction for IOSSA.

EOSSA is motivated by parametric model of finite-dimensional signal, however it does not exploit this model directly and does not estimate the parameters. Therefore, it works for wider class of time series. According to the experiments, it works for series that could be locally approximated by a series of finite dimension, but at this moment there is no any theoretical results for this.

EOSSA constructs shift matrix estimation by the same way is in ESPRIT (see parestimate) method and uses its eigenspace to build separating scalar products (see iossa for more information about Oblique SSA decompositions). Consequently, the method ideally separates signals of finite dimension with absence of noise. With presence of noise it provides approximate results due to continuity. The method performs eigenvectors clustering inside (for now hclust is used), the number of components (argument k) should be passed.

Value

Object of 'ossa' class.

References

Shlemov A. (2017): *The method of signal separation using the eigenspaces of the shift matrices (in Russian)*, In Proceedings of the SPISOK-2017 conference, April 26–28, Saint Petersburg, Russia.

See Also

Rssa for an overview of the package, as well as, ssa-object, ESPRIT, iossa, fossa, owcor, iossa.result.

Examples

```
# Separability of three finite-dimensional series, EOSSA vs Basic SSA
N <- 150
L <- 70
omega1 <- 0.065
omega2 <- 0.07
omega3 <- 0.02
sigma <- 0.5

F1.real <- 2*sin(2*pi*omega1*(1:N))
F2.real <- 4*sin(2*pi*omega2*(1:N))
F3.real <- sin(2*pi*omega3*(1:N))

noise <- rnorm(N, sd = sigma)
F <- F1.real + F2.real + F3.real + noise

ss <- ssa(F, L)
eoss <- eossa(ss, nested.groups = list(1:2, 3:4, 5:6), k = 3)

print(eoss)
```

```
plot(ss, type = "series", groups = list(1:2, 3:4, 5:6))
plot(eoss, type = "series", groups = eoss$iossa.groups)

plot(reconstruct(ss,
                 groups = list(1:2, 3:4, 5:6)),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

plot(reconstruct(eoss, groups = list(1:2, 3:4, 5:6)),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

plot(reconstruct(ss,
                 groups = list(Reconstructed = 1:6, F1 = 1:2, F2 = 3:4, F3 = 5:6)),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

plot(reconstruct(eoss,
                 groups = list(Reconstructed = 1:6, F1 = 1:2, F2 = 3:4, F3 = 5:6)),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

rec.ideal <- reconstruct(ss,
                         groups = list(Signal = 1:6, F1 = 1:2, F2 = 3:4, F3 = 5:6))
rec.ideal$Signal <- F1.real + F2.real + F3.real
rec.ideal$F1 <- F2.real
rec.ideal$F2 <- F1.real
rec.ideal$F3 <- F3.real

plot(rec.ideal,
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

# Real-life example (co2), EOSSA vs Basic SSA
sigma <- 0.05
ss <- ssa(co2)
plot(ss, type = "vector")
eoss <- eossa(ss, 1:6, k = 4)
eoss$iossa.groups

plot(eoss)
rec <- reconstruct(eoss, groups = eoss$iossa.groups)
plot(rec)

plot(reconstruct(ss,
                 groups = list(ET1 = 1,ET2 = 2,ET3 = 3,ET4 = 4,ET5 = 5,ET6 = 6)),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
     xlab = "")

plot(reconstruct(eoss,
                 groups = eoss$iossa.groups),
     add.residuals = TRUE, plot.method = "xyplot", main = "",
```

```
     xlab = "")

# Sine wave with phase shift, EOSSA vs Basic SSA
omega1 <- 0.06
omega2 <- 0.07
sigma <- 0.25

F1.real <- sin(2*pi*omega1*(1:N))
F2.real <- sin(2*pi*omega2*(1:N))
v <- c(F1.real,  F2.real)
v <- v + rnorm(v, sd = sigma)
# v <- c(F1.real,  F2.real)

ss <- ssa(v, L = 35)

eoss <- eossa(ss, 1:4, 2)
ioss <- iossa(ss, list(1:2, 3:4))

plot(reconstruct(eoss, groups = eoss$iossa.groups))

plot(reconstruct(eoss,
     groups = eoss$iossa.groups), plot.method = "xyplot", main = "",
     xlab = "")

plot(reconstruct(ss, groups = list(1:2, 3:4)),
     plot.method = "xyplot",
     main = "", xlab = "")
plot(reconstruct(ss, groups = list(1,2, 3,4)),
     plot.method = "xyplot",
     main = "", xlab = "")
```

`forecast` *Perform SSA forecasting of series*

Description

All-in-one function to perform SSA forecasting of one-dimensional series.

Usage

```
## S3 method for class '1d.ssa'
forecast(object,
         groups, h = 1,
         method = c("recurrent", "vector"),
         interval = c("none", "confidence", "prediction"),
         only.intervals = TRUE,
         ...,
         drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
```

```
forecast(object,
         groups, h = 1,
         method = c("recurrent", "vector"),
         interval = c("none", "confidence", "prediction"),
         only.intervals = TRUE,
         ...,
         drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class '1d.ssa'
predict(object,
        groups, len = 1,
        method = c("recurrent", "vector"),
        interval = c("none", "confidence", "prediction"),
        only.intervals = TRUE,
        ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
predict(object,
        groups, len = 1,
        method = c("recurrent", "vector"),
        interval = c("none", "confidence", "prediction"),
        only.intervals = TRUE,
        ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'mssa'
predict(object,
        groups, len = 1,
        method = c("recurrent", "vector"),
        direction = c("column", "row"),
        ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
```

Arguments

`object`	SSA object holding the decomposition
`groups`	list, the grouping of eigentriples to be used in the forecast
`h,len`	the desired length of the forecasted series
`method`	method of forecasting to be used
`interval`	type of interval calculation
`only.intervals`	logical, if 'TRUE' then bootstrap method is used for confidence bounds only, otherwise — mean bootstrap forecast is returned as well
`direction`	direction of forecast in multichannel SSA case, "column" stands for so-called L-forecast and "row" stands for K-forecast
`...`	further arguments passed for forecast routines (e.g. `level` argument to `bforecast`)
`drop`	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)

drop.attributes
: logical, if 'TRUE' then the forecast routines do not try to infer the time index arguments for the forecasted series.

cache
: logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

This function is a convenient wrapper over other forecast routines (see 'See Also') turning their value into object of type 'forecast' which can be used with the routines from **forecast** package.

Value

object of class 'forecast' for `forecast` function call, predicted series for `predict` call.

See Also

`Rssa` for an overview of the package, as well as, `rforecast`, `vforecast`, `bforecast`, `forecast (package)`

Examples

```
s <- ssa(co2)
# Calculate 24-point forecast using first 6 components as a base
f <- forecast(s, groups = list(1:6), method = "recurrent", bootstrap = TRUE, len = 24, R = 10)

# Plot the result including the last 24 points of the series
plot(f, include = 24, shadecols = "green", type = "l")
# Use of predict() for prediction
p <- predict(s, groups = list(1:6), method = "recurrent", len = 24)
# Simple plotting
plot(p, ylab = "Forecasteed Values")
```

---

fossa *Nested Filter-adjusted O-SSA decomposition*

Description

Perform nested decomposition by Filter-adjusted O-SSA (FOSSA).

Usage

```
  ## S3 method for class 'ssa'
fossa(x, nested.groups, filter = c(-1, 1), gamma = Inf, normalize = TRUE, ...)
```

Arguments

x	SSA object holding SSA decomposition
nested.groups	vector of numbers of eigentriples from full decomposition for nested decomposition. The argument is coerced to a vector, if necessary
filter	numeric vector or array of reversed impulse response (IR) coefficients for filter adjustment or list of such vectors or arrays
gamma	weight of filter adjustment. See 'Details' and 'References'
normalize	logical, whether to normalize left decomposition vectors before filtering
...	additional arguments passed to decompose routines

Details

See Golyandina N. and Shlemov A. (2015) and Section 2.5 in Golyanina et al (2018) for full details in the 1D case and p.250-252 from the same book for an example in the 2D case.

Briefly, FOSSA serves for decomposition of series components that are mixed due to equal contributions of their elementary components, e.g. of sinusoids with equal amplitudes or of complex-form trend and periodics. FOSSA performs a new decomposition of a part of the ssa-object, which is given by a set of eigentriples. Note that eigentriples that do not belong to the chosen set are not changed.

In particular, Filter-adjusted O-SSA performs a nested decomposition specified by a number of eigentriples via Oblique SSA with a specific inner product in the row space:

$$\langle x, y\rangle = (x, y) + \gamma^2(\Phi(x), \Phi(y)),$$

where $(\cdot, \cdot)$ denotes conventional inner product and 'Φ' is linear filtration which is specified by `filter` argument.

The default value of Φ corresponds to sequential differences, that is, to derivation. Such version of Filter-adjusted O-SSA is called 'DerivSSA'. See 'References' for more details.

`filter` **argument:** For 1D-SSA, Toeplitz-SSA and MSSA: Filter can be given by a vector or a list of vectors. Each vector corresponds to reversed IR for a filter, these filters are applied independently and their results are stacked such that the matrix $[X : \Phi_1(X) : \Phi_2(X)]$ is decomposed.

For 2D-SSA: the following variants are possible: (1) a list of vectors. Each vector corresponds to reversed IR for a filter. Each filter is applied to different dimensions, the first to columns, the second to rows, and the results are stacked. (2) single vector. Given vector corresponds to one-dimensional filter applied to both dimensions, the same as list of two equal vectors. (3) a list of matrices, where each matrix provides 2d filter coefficients and the results are stacked. (4) single matrix. Given matrix corresponds to two-dimensional filter applied once, the same as list of one matrix.

For nD-SSA: the same as for 2D-SSA, a list of vectors for filters by directions, single vector, a list of arrays (matroids) for nD filters or single array.

Normalization: Let us explain for the 1D case. Let X be the reconstructed matrix, corresponding to the selected eigentriples $\{(\sigma_i, U_i, V_i)\}$, $\Psi(X)$ is the matrix, where the filter is applied to each row of X.

Then `normalize = FALSE` (Algorithm 2.9 or 2.10 in Golyandina et al (2018)) corresponds to finding the basis in the column space of X by means of the SVD of $[X, \Psi(X)]$, while `normalize = TRUE` (by default, see Algorithm 2.11 in Golyandina et al (2018)) corresponds to finding the basis by the SVD of $[V, \Phi(V)]$, where the rows of matrix V are V_i. The value by default `TRUE` guaranties that the contributions of sine waves will be ordered by decreasing of frequencies, although can slightly worsen the weak separability

Value

Object of class 'ossa'. The field 'ossa.set' contains the vector of indices of elementary components used in Filter-adjusted O-SSA (that is, used in `nested.groups`).

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Golyandina N. and Shlemov A. (2015): *Variations of Singular Spectrum Analysis for separability improvement: non-orthogonal decompositions of time series*, Statistics and Its Interface. Vol.8, No 3, P.277-294. https://arxiv.org/abs/1308.4022

See Also

`Rssa` for an overview of the package, as well as, `iossa`.

Examples

```
# Separation of two mixed sine-waves with equal amplitudes
N <- 150
L <- 70
omega1 <- 1/15
omega2 <- 1/10

v <- sin(2*pi*omega1 * (1:N)) + sin(2*pi*omega2 * (1:N))
s <- ssa(v, L)
fs <- fossa(s, nested.groups = 1:4, gamma = 100)

# Rssa does most of the plots via lattice
ws <- plot(wcor(s, groups = 1:4))
wfs <- plot(wcor(fs, groups = 1:4))
plot(ws, split = c(1, 1, 2, 1), more = TRUE)
plot(wfs, split = c(2, 1, 2, 1), more = FALSE)

opar <- par(mfrow = c(2, 1))
plot(reconstruct(s, groups = list(1:2, 3:4)))
plot(reconstruct(fs, groups = list(1:2, 3:4)))
par(opar)

# Real-life example: Australian Wine Sales

data(AustralianWine)
s <- ssa(AustralianWine[1:120, "Fortified"], L = 60)
```

```
fs <- fossa(s, nested.groups = list(6:7, 8:9, 10:11), gamma = 10)

plot(reconstruct(fs, groups = list(6:7, 8:9, 10:11)))
plot(wcor(s, groups = 6:11))
plot(wcor(fs, groups = 6:11))

# Real life example: improving of strong separability
data(USUnemployment)
unempl.male <- USUnemployment[, "MALE"]
s <- ssa(unempl.male)
fs <- fossa(s, nested.groups = 1:13, gamma = 1000)

# Comparison of reconstructions
rec <- reconstruct(s, groups = list(c(1:4, 7:11), c(5:6, 12:13)))
frec <- reconstruct(fs, groups <- list(5:13, 1:4))
# Trends
matplot(data.frame(frec$F1, rec$F1, unempl.male), type= 'l',
        col=c("red","blue","black"), lty=c(1,1,2))
# Seasonalities
matplot(data.frame(frec$F2, rec$F2), type = 'l', col=c("red","blue"), lty=c(1,1))

# W-cor matrices before and after FOSSA
ws <- plot(wcor(s, groups = 1:30), grid = 14)
wfs <- plot(wcor(fs, groups = 1:30), grid = 14)
plot(ws, split = c(1, 1, 2, 1), more = TRUE)
plot(wfs, split = c(2, 1, 2, 1), more = FALSE)

# Eigenvectors before and after FOSSA
plot(s, type = "vectors", idx = 1:13)
plot(fs, type = "vectors", idx = 1:13)

# 2D plots of periodic eigenvectors before and after FOSSA
plot(s, type = "paired", idx = c(5, 12))
plot(fs, type = "paired", idx = c(1, 3))

# Compare FOSSA with and without normalize
N <- 150
L <- 70
omega1 <- 1/15
omega2 <- 1/10

v <- 3*sin(2*pi*omega1 * (1:N)) + 2*sin(2*pi*omega2 * (1:N))
s <- ssa(v, L)
fs <- fossa(s, nested.groups = 1:4, gamma = 100)
fs.norm <- fossa(s, nested.groups = 1:4, gamma = 100, normalize = TRUE)
opar <- par(mfrow = c(2, 1))
plot(reconstruct(fs, groups = list(1:2, 3:4)))
plot(reconstruct(fs.norm, groups = list(1:2, 3:4)))
par(opar)

# 2D example
data(Mars)
s <- ssa(Mars)
```

```
plot(s, "vectors", idx = 1:50)
plot(s, "series", idx = 1:50)
fs <- fossa(s, nested.groups = 1:50, gamma = Inf)
plot(fs, "vectors", idx = 1:14)
plot(fs, "series", groups = 1:13)

# Filters example, extracting horizontal and vertical stripes
data(Mars)
s <- ssa(Mars)
fs.hor <- fossa(s, nested.groups = 1:50, gamma = Inf,
                filter = list(c(-1, 1), c(1)))
plot(fs.hor, "vectors", idx = 1:14)
plot(fs.hor, "series", groups = 1:13)
fs.ver <- fossa(s, nested.groups = 1:50, gamma = Inf,
                filter = list(c(1), c(-1, 1)))
plot(fs.ver, "vectors", idx = 1:14)
plot(fs.ver, "series", groups = 1:13)
```

`frobenius.cor` *Calculate Frobenius correlations of the component matrices*

Description

Function calculates Frobenius correlations between grouped matrices from the SSA matrix decomposition

Usage

```
frobenius.cor(x, groups, ...)
```

Arguments

`x`	input SSA object, supposed to be of class 'ossa'
`groups`	list of numeric vectors, indices of elementary matrix components in the SSA matrix decomposition
`...`	further arguments passed to `decompose`

Details

Function computes matrix of Frobenius correlations between grouped matrices from the SSA matrix decomposition. For group $\mathcal{I} = \{i_1, \ldots, i_s\}$ the group matrix is defined as $\mathbf{X}_\mathcal{I} = \sum_{i \in \mathcal{I}} \sigma_i U_i V_i^{\mathrm{T}}$.

Frobenius correlation of two matrices is defined as follows:

$$\mathrm{fcor}(\mathbf{Z}, \mathbf{Y}) = \frac{\langle \mathbf{Z}, \mathbf{Y} \rangle_{\mathrm{F}}}{\|\mathbf{Z}\|_{\mathrm{F}} \cdot \|\mathbf{Y}\|_{\mathrm{F}}}.$$

Frobenius correlation is a measure of Frobenius orthogonality of the components. If grouped matrices are correlated then the w-correlations of the corresponding reconstructed series is not relevant measure of separability (and one should use owcor instead). Also, if the elementary matrices $\mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}}$ of the decomposition are not F-orthogonal, then σ_i do not reflect their true contributions into the matrix decomposition.

This function normally should be used only for object of class 'ossa'. Otherwise it always returns identical matrix (for disjoint groups).

Value

Object of type 'wcor.matrix'.

See Also

wcor, owcor, iossa.

Examples

```
# Separation of two mixed sine-waves with equal amplitudes
N <- 150
L <- 70
omega1 <- 1/5
omega2 <- 1/10

v <- sin(2*pi*omega1 * (1:N)) + sin(2*pi*omega2 * (1:N))
s <- ssa(v, L)
fs <- fossa(s, nested.groups = 1:4, gamma = 100)

# Decomposition is F-orthogonal
plot(frobenius.cor(fs, groups = 1:4), main = "F-correlation matrix")

plot(wcor(s, groups = 1:4))
plot(wcor(fs, groups = 1:4))


# Separate two non-separable sine series with different amplitudes

N <- 150
L <- 70

omega1 <- 0.07
omega2 <- 0.0675

F <- 2*sin(2*pi*omega1 * (1:N)) + 2*sin(2*pi*omega2 * (1:N))
s <- ssa(F, L)
ios <- iossa(s, nested.groups = list(1:2, 3:4),
             kappa = NULL, maxiter = 1000, tol = 1e-5)

plot(reconstruct(ios, groups = ios$iossa.groups))
summary(ios)

# Decomposition is really oblique
```

```
plot(frobenius.cor(ios, groups = 1:4), main = "F-correlation matrix")

plot(wcor(ios, groups = 1:4))
plot(owcor(ios, groups = list(1:2, 3:4)), main = "Oblique W-correlation matrix")




data(USUnemployment)
unempl.male <- USUnemployment[, "MALE"]

s <- ssa(unempl.male)
ios <- iossa(s, nested.groups = list(c(1:4, 7:11), c(5:6, 12:13)))
summary(ios)

# W-cor matrix before IOSSA and w-cor matrix after it
plot(wcor(s, groups = 1:30))
plot(wcor(ios, groups = 1:30))

# Confirmation of the indicated max value in the above warning
plot(frobenius.cor(ios, groups = 1:30), main = "F-correlation matrix")
```

---

gapfill *Perform SSA gapfilling via forecast*

Description

Perform SSA gapfilling of the series.

Usage

```
## S3 method for class '1d.ssa'
gapfill(x, groups, base = c("original", "reconstructed"),
        method = c("sequential", "simultaneous"),
        alpha = function(len) seq.int(0, 1, length.out = len), ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'mssa'
gapfill(x, groups, base = c("original", "reconstructed"),
        alpha = function(len) seq.int(0, 1, length.out = len), ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'cssa'
gapfill(x, groups, base = c("original", "reconstructed"),
        method = c("sequential", "simultaneous"),
        alpha = function(len) seq.int(0, 1, length.out = len), ...,
        drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
gapfill(x, groups, base = c("original", "reconstructed"),
        method = c("sequential", "simultaneous"),
```

```
alpha = function(len) seq.int(0, 1, length.out = len), ...,
drop = TRUE, drop.attributes = FALSE, cache = TRUE)
```

Arguments

x	Shaped SSA object holding the decomposition
groups	list, the grouping of eigentriples to be used in the forecast
base	series used as a 'seed' for gapfilling: original or reconstructed according to the value of groups argument
method	method used for gapfilling, "sequential" means to filling by a recurrent forecast from complete parts; "simultaneous" tries to build a projections onto the signal subspace. See 'References' for more info.
alpha	weight used for combining forecasts from left and right when method = "sequential"; 0.5 means that the forecasts are averaged, 0 (1) means that only forecast from the left (right correspondingly) is used, arbitrary function could be specified; by default linear weights are used.
...	additional arguments passed to reconstruct routines
drop	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)
drop.attributes	logical, if 'TRUE' then the attributes of the input series are not copied to the reconstructed ones.
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

The function fills in the missed entries in the series. Both methods described in Golyandina and Osipov (2007) are implemented:

- method = "sequential" performs forecast from complete chunks onto incomplete. For internal gaps forecast is performed from both sides of the gap and average is taken in order to reduce the forecast error. For gaps in the beginning or end of the series the method coincides with ordinary recurrent forecast;
- method = "simultaneous" performs gap filling via projections onto signal subspace. The method may fail if insufficient complete observations are provided.

Details of the used algorithms see in Golyandina et al (2018), Algorithms 3.8 and 3.9 respectively.

Value

List of objects with gaps filled in. Elements of the list have the same names as elements of groups. If group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in groups list.

Or, the forecasted object itself, if length of groups is one and 'drop = TRUE'.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

N. Golyandina, E. Osipov (2007): *The "Caterpillar"-SSA method for analysis of time series with missing values*. Journal of Statistical Planning and Inference, Vol. 137, No. 8, Pp 2642–2653 https://www.gistatgroup.com/cat/mvssa1en.pdf

See Also

Rssa for an overview of the package, as well as, rforecast, igapfill, clplot, summarize.gaps,

Examples

```
# Produce series with gaps
F <- co2; F[100:200] <- NA
# Perform shaped SSA
s <- ssa(F, L = 72)
# Fill in gaps using the trend and 2 periodicty components
g <- gapfill(s, groups = list(1:6))
# Compare the result
plot(g)
lines(co2, col = "red")
```

`grouping.auto` *Group Elementary Series*

Description

The ‘grouping.auto’ function performs the Grouping Step of SSA using different approaches.

Usage

```
grouping.auto(x, ..., grouping.method = c("pgram", "wcor"))
```

Arguments

`x`	SSA object
`grouping.method`	String specifying the method used to perform the grouping. Allowed methods are ‘"pgram"’ (the default) and ‘"wcor"’
`...`	Further arguments to specific methods

Details

‘grouping.auto’ is a wrapper function which calls the methods ‘grouping.auto.pgram’ and ‘grouping.auto.wcor’.

Value

List of integer vectors holding the indices of the elementary components forming each grouped objects.

See Also

grouping.auto.pgram, grouping.auto.wcor

---

grouping.auto.pgram *Group elementary series using periodogram*

Description

Group elementary components automatically using their frequency contributions

Usage

```
  ## S3 method for class '1d.ssa'
grouping.auto.pgram(x, groups,
             base = c("series", "eigen", "factor"),
             freq.bins = 2,
             threshold = 0,
             method = c("constant", "linear"),
             ...,
             drop = TRUE)
  ## S3 method for class 'grouping.auto.pgram'
plot(x, superpose, order, ...)
```

Arguments

x	SSA object
groups	indices of elementary components for grouping
base	input for periodogram: elementary reconstructed series, eigenvectors or factor vectors
freq.bins	single integer number > 1 (the number of intervals), vector of frequency breaks (of length >=2) or list of frequency ranges. For each range, if only one element provided it will be used as the upper bound and the lower bound will be zero
threshold	contribution threshold. If zero then dependent grouping approach will be used
method	method of periodogram interpolation
superpose	logical, whether to plot contributions for all intervals on one panel
order	logical, whether to reorder components by contribution
...	additional arguments passed to reconstruct and xyplot routines
drop	logical, whether to exclude empty groups from resulted list

Details

Elementary components are grouped using their frequency contribution (periodogram). Optionally (see argument 'base') periodogram of eigen or factor vectors may be used.

For each elementary component and for each frequency interval (which are specified by 'freq.bins' argument) relative (from 0 till 1) contribution is computed using one of two methods: 'constant' (periodogram is considered as a sequence of separate bars) or 'linear' (periodogram is linearly interpolated).

Two approaches of grouping is implemented:

'independent' or 'threshold' Each group includes components with frequency contribution in correspondent interval is greater than specified threshold; resulted groups can intersect. If 'threshold' is a vector, correspondent value of threshold will be using for each interval. See Algorithm 2.16 in Golyandina et al (2018).

'dependent' or 'splitting' Elementary components are separated to disjoint subsets; for each component interval with the highest contribution is selected. See Algorithm 2.17 in Golyandina et al (2018)

If 'freq.bins' is named, result groups will take the same names.

If drop = 'TRUE' (by default), empty groups will be excluded from result.

See Section 2.7 in Golyandina et al (2018) and the paper Alexandrov, Golyandina (2005) for the details of the algorithm.

Value

object of class 'grouping.auto.pgram' (list of groups with some additional info) for grouping method; 'trellis' object for plot method.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Alexandrov, Th., Golyandina, N. (2005): *Automatic extraction and forecast of time series cyclic components within the framework of SSA*. In Proceedings of the 5th St.Petersburg Workshop on Simulation, June 26 – July 2, 2005, St.Petersburg State University, St.Petersburg, Pp. 45–50 https://www.gistatgroup.com/gus/autossa2.pdf

See Also

Rssa for an overview of the package, as well as, reconstruct, rforecast, vforecast, parestimate

Examples

```
ss <- ssa(co2)
plot(ss, type = "vectors", idx = 1:12)
plot(ss, type = "vectors", vectors = "factor", idx = 1:12)
plot(ss, type = "series", groups = 1:12)

g1 <- grouping.auto(ss, base = "series", freq.bins = list(0.005), threshold = 0.95)
```

```
g2 <- grouping.auto(ss, base = "eigen", freq.bins = 2, threshold = 0)
g3 <- grouping.auto(ss, base = "factor", freq.bins = list(c(0.1), c(0.1, 0.2)),
                    threshold = 0, method = "linear")
g4 <- grouping.auto(ss, freq.bins = c(0.1, 0.2), threshold = 0)

g <- grouping.auto(ss, freq.bins = 8, threshold = 0)
plot(reconstruct(ss, groups = g))
plot(g)

g <- grouping.auto(ss, freq.bins = list(0.1, 0.2, 0.3, 0.4, 0.5), threshold = 0.95)
plot(reconstruct(ss, groups = g))
plot(g)
```

`grouping.auto.wcor` *Group Elementary Series Using W-correlation Matrix*

Description

Group elemenatry series automatically via the hierarchical clustering with w-correlation matrix as a proximity matrix

Usage

```
## S3 method for class 'ssa'
grouping.auto.wcor(x, groups, nclust = length(groups) / 2, ...)
```

Arguments

`x`	SSA object
`groups`	list of numeric vectors, indices of elementary components used for reconstruction
`nclust`	integer, desired number of output series
`...`	further arguments passed to `hclust`

Details

Standard `hclust` routine is used to perform the grouping of the elementary components. See Algorithm 2.15 in Golyandina et al (2018) for details.

Value

List of integer vectors holding the indices of the elementary components forming each grouped objects

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R.* Use R!. Springer, Berlin, Heidelberg.

See Also

hclust, wcor

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Form 3 series from the initial 6 ones:
lst <- grouping.auto(s, grouping.method = "wcor",
                     groups = 1:6, nclust=3)
# Automatic grouping:
print(lst)
plot(lst)
# Check separability
w <- wcor(s, groups = lst)
plot(w)
```

---

hbhmat *Hankel with Hankel block matrices operations.*

Description

A set of routines to operate on Hankel with Hankel block matrices stored in compact FFT-based form.

Usage

```
new.hbhmat(F, L = (N + 1) %/% 2,
           wmask = NULL, fmask = NULL, weights = NULL,
           circular = FALSE)
is.hbhmat(h)
hbhcols(h)
hbhrows(h)
hbhmatmul(hmat, v, transposed = FALSE)
```

Arguments

F	array to construct the trajectory matrix for.
L	the window length.
wmask, fmask, weights	special parameters for shaped SSA case (see ssa). wmask and fmask are logical matrices, window and factor masks respectively. weights is integer matrix which denotes hankel weights for array elements. If 'NULL', parameters for simple rectangular 2D SSA case are used.
circular	logical vector of one or two elements, describes field topology. 'TRUE' means circularity by a corresponding coordinate. If vector has only one element, this element will be used twice.

h, hmat	matrix to operate on.
transposed	logical, if 'TRUE' the multiplication is performed with the transposed matrix.
v	vector to multiply with.

Details

Fast Fourier Transform provides a very efficient matrix-vector multiplication routine for Hankel with Hankel blocks matrices. See the paper in 'References' for the details of the algorithm.

Author(s)

Konstantin Usevich

References

Korobeynikov, A. (2010) *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

---

hmat *Hankel matrices operations.*

Description

A set of routines to operate on Hankel matrices stored in compact FFT-based form.

Usage

```
new.hmat(F, L = (N + 1)%/%2, circular = FALSE, wmask = NULL,
         fmask = NULL, weights = NULL, fft.plan = NULL)
is.hmat(h)
hcols(h)
hrows(h)
hmatmul(hmat, v, transposed = FALSE)
hankel(X, L)
```

Arguments

F	series to construct the trajectory matrix for.
fft.plan	internal hint argument, should be NULL in most cases
wmask, fmask, weights	special parameters for shaped SSA case (see ssa). wmask and fmask are logical vectors, window and factor masks respectively. weights is integer vector which denotes hankel weights for array elements. If 'NULL', parameters for simple 1D SSA case are used.
circular	logical vector of one element, describes series topology. 'TRUE' means circularity by time.

L	the window length.
h, hmat	matrix to operate on.
transposed	logical, if 'TRUE' the multiplication is performed with the transposed matrix.
v	vector to multiply with.
X	series to construct the trajectory matrix for or matrix for hankelization

Details

Fast Fourier Transform provides a very efficient matrix-vector multiplication routine for Hankel matrices. See the paper in 'References' for the details of the algorithm.

References

Korobeynikov, A. (2010) *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

See Also

Rssa for an overview of the package, as well as, ssa, decompose,

Examples

```
# Construct the Hankel trajectory matrix for 'co2' series
h <- new.hmat(co2, L = 10)
# Print number of columns and rows
print(hrows(h))
print(hcols(h))
```

---

hmatr *Calculate the heterogeneity matrix.*

Description

Function calculates the heterogeneity matrix for the one-dimensional series.

Usage

```
hmatr(F, ...,
      B = N %/% 4, T = N %/% 4, L = B %/% 2,
      neig = 10)

## S3 method for class 'hmatr'
plot(x,
     col = rev(heat.colors(256)),
     main = "Heterogeneity Matrix", xlab = "", ylab = "", ...)
```

Arguments

F	the series to be checked for structural changes
...	further arguments passed to ssa routine for hmatr call or image for plot.hmatr call
B	integer, length of base series
T	integer, length of tested series
L	integer, window length for the decomposition of the base series
neig	integer, number of eigentriples to consider for calculating projections
x	'hmatr' object
col	color palette to use
main	plot title
xlab,ylab	labels for 'x' and 'y' axis

Details

The *heterogeneity matrix* (H-matrix) provides a consistent view on the structural discrepancy between different parts of the series. Denote by $F_{i,j}$ the subseries of F of the form: $F_{i,j} = (f_i, \dots, f_j)$. Fix two integers $B > L$ and $T \geq L$. Let these integers denote the lengths of *base* and *test* subseries, respectively. Introduce the H-matrix $G_{B,T}$ with the elements g_{ij} as follows:

$$g_{ij} = g(F_{i,i+B}, F_{j,j+T}),$$

for $i = 1, \dots, N - B + 1$ and $j = 1, \dots, N - T + 1$, that is we split the series F into subseries of lengths B and T and calculate the heterogeneity index between all possible pairs of the subseries.

The heterogeneity index $g(F^{(1)}, F^{(2)})$ between the series $F^{(1)}$ and $F^{(2)}$ can be calculated as follows: let $U_j^{(1)}$, $j = 1, \dots, L$ denote the eigenvectors of the SVD of the trajectory matrix of the series $F^{(1)}$. Fix I to be a subset of $\{1, \dots, L\}$ and denote $\mathcal{L}^{(1)} = \mathrm{span}\ (U_i,\ i \in I)$. Denote by $X_1^{(2)}, \dots, X_{K_2}^{(2)}$ ($K_2 = N_2 - L + 1$) the L-lagged vectors of the series $F^{(2)}$. Now define

$$g(F^{(1)}, F^{(2)}) = \frac{\sum_{j=1}^{K_2} \mathrm{dist}^2 \left(X_j^{(2)}, \mathcal{L}^{(1)}\right)}{\sum_{j=1}^{K_2} \left\|X_j^{(2)}\right\|^2},$$

where $\mathrm{dist}\,(X, \mathcal{L})$ denotes the Euclidean distance between the vector X and the subspace $\mathcal{L}$. One can easily see that $0 \leq g \leq 1$.

Value

object of type 'hmatr'

References

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

See Also

ssa

Examples

```
# Calculate H-matrix for co2 series
h <- hmatr(co2, L = 24)
# Plot the matrix
plot(h)
```

---

igapfill *Perform SSA gapfilling via iterative reconstruction*

Description

Perform iterative gapfilling of the series.

Usage

```
## S3 method for class '1d.ssa'
igapfill(x, groups, fill = NULL, tol = 1e-6, maxiter = 0,
          norm = function(x) sqrt(max(x^2)),
          base = c("original", "reconstructed"), ..., trace = FALSE,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'cssa'
igapfill(x, groups, fill = NULL, tol = 1e-6, maxiter = 0,
          norm = function(x) sqrt(max(x^2)),
          base = c("original", "reconstructed"), ..., trace = FALSE,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
igapfill(x, groups, fill = NULL, tol = 1e-6, maxiter = 0,
          norm = function(x) sqrt(max(x^2)),
          base = c("original", "reconstructed"), ..., trace = FALSE,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'nd.ssa'
igapfill(x, groups, fill = NULL, tol = 1e-6, maxiter = 0,
          norm = function(x) sqrt(max(x^2)),
          base = c("original", "reconstructed"), ..., trace = FALSE,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
```

Arguments

x	Shaped SSA object holding the decomposition
groups	list, the grouping of eigentriples to be used in the forecast
fill	initial values for missed entries, recycled if necessary; if missed, then average of the series will be used

tol	tolerance for reconstruction iterations
maxiter	upper bound for the number of iterations
norm	distance function used for covergence criterion
base	series used as a 'seed' for gapfilling: original or reconstructed according to the value of groups argument
...	additional arguments passed to reconstruct routines
trace	logical, indicates whether the convergence process should be traced
drop	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)
drop.attributes	logical, if 'TRUE' then the attributes of the input series are not copied to the reconstructed ones.
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

Iterative gapfilling starts from filling missed entries with initial values, then the missed values are imputed from the successive reconstructions. This process continues until convergence up to a stationary point (e.g. filling / reconstruction does not change missed values at all).

Details of the used algorithm see in Golyandina et al (2018), Algorithms 3.7.

Value

List of objects with gaps filled in. Elements of the list have the same names as elements of groups. If group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in groups list.

Or, the forecasted object itself, if length of groups is one and 'drop = TRUE'.

Note

The method is very sensitive to the initial value of missed entries ('fill' argument). If the series are not stationary (e.g. contains some trend) than the method may be prohibitely slow, or even fail to converge or produce bogus results.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Kondrashov, D. & Ghil, M. (2006) *Spatio-temporal filling of missing points in geophysical data sets*. Nonlinear Processes In Geophysics, Vol. 13(2), pp. 151-159.

See Also

Rssa for an overview of the package, as well as, gapfill, clplot, summarize.gaps,

Examples

```
# Produce series with gaps
F <- co2; F[100:200] <- NA
# Perform shaped SSA
s <- ssa(F, L = 72)
# Fill in gaps using the trend and 2 periodicty components
# Due to trend, provide a linear filler to speedup the process
fill <- F; fill[100:200] <- F[99] + (1:101)/101*(F[201] - F[99])
g <- igapfill(s, groups = list(1:6), fill = fill, maxit = 50)
# Compare the result
plot(g)
lines(co2, col = "red")
```

`iossa` *Iterative O-SSA nested decomposition*

Description

Perform Iterative O-SSA (IOSSA) algorithm.

Usage

```
## S3 method for class 'ssa'
iossa(x, nested.groups, ..., tol = 1e-5, kappa = 2,
      maxiter = 100,
      norm = function(x) sqrt(mean(x^2)),
      trace = FALSE,
      kappa.balance = 0.5)
```

Arguments

`x`	SSA object holding SSA decomposition
`nested.groups`	list or named list of numbers of eigentriples from full decomposition, describes initial grouping for IOSSA iterations
`tol`	tolerance for IOSSA iterations
`kappa`	'kappa' parameter for sigma-correction (see 'Details' and 'References') procedure. If 'NULL', sigma-correction will not be performed
`maxiter`	upper bound for the number of iterations
`norm`	function, calculates a norm of a vector; this norm is applied to the difference between the reconstructed series at sequential iterations and is used for convergence detection
`trace`	logical, indicates whether the convergence process should be traced
`kappa.balance`	sharing proportion of sigma-correction multiplier between column and row inner products
`...`	additional arguments passed to `decompose` routines

Details

See Golyandina N. and Shlemov A. (2015) and Section 2.4 in Golyanina et al (2018) for full details in the 1D case and p.250-252 from the same book for an example in the 2D case.

Briefly, Iterative Oblique SSA (IOSSA) is an iterative (EM-like) method for improving separability in SSA. In particular, it serves for separation of mixed components, which are not orthogonal, e.g., of sinusoids with close frequencies or for trend separation for short series. IOSSA performs a new decomposition of a part of the ssa-object, which is given by a set of eigentriples. Note that eigentriples that do not belong to the chosen set are not changed.

Oblique SSA can make many series orthogonal by the choice of inner product. Iterative O-SSA find the separating inner products by iterations that are hopefully converges to a stationary point. See References for more details.

Sigma-correction procedure does the renormalization of new inner products. This prevents the mixing of the components during the next iteration. Such approach makes the whole procedure more stable and can solve the problem of lack of strong separability.

Details of the used algorithms can be found in Golyandina et al (2018), Algorithms 2.7 and 2.8.

Value

Object of 'ossa' class. In addition to usual 'ssa' class fields, it also contains the following fields:

iossa.result object of 'iossa.result' class, a list which contains algorithm parameters, condition numbers, separability measures, the number of iterations and convergence status (see `iossa.result`)

iossa.groups list of groups within the nested decomposition; numbers of components correspond to their numbers in the full decomposition

iossa.groups.all list, describes cumulative grouping after after sequential Iterative O-SSA decompositions in the case of non-intersecting `nested.groups`. Otherwise, `iossa.groups.all` coincides with `iossa.groups`

ossa.set vector of the indices of elementary components used in Iterative O-SSA (that is, used in `nested.groups`)

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Golyandina N. and Shlemov A. (2015): *Variations of Singular Spectrum Analysis for separability improvement: non-orthogonal decompositions of time series*, Statistics and Its Interface. Vol.8, No 3, P.277-294. https://arxiv.org/abs/1308.4022

See Also

`Rssa` for an overview of the package, as well as, `ssa-object`, `fossa`, `owcor`, `iossa.result`.

Examples

```
# Separate three non-separable sine series with different amplitudes
N <- 150
L <- 70
```

```
omega1 <- 0.05
omega2 <- 0.06
omega3 <- 0.07

F <- 4*sin(2*pi*omega1 * (1:N)) + 2*sin(2*pi*omega2 * (1:N)) + sin(2*pi*omega3 * (1:N))
s <- ssa(F, L)
ios <- iossa(s, nested.groups = list(1:2, 3:4, 5:6), kappa = NULL, maxiter = 100, tol = 1e-3)

plot(reconstruct(ios, groups = ios$iossa.groups))
summary(ios)


# Separate two non-separable sines with equal amplitudes
N <- 200
L <- 100
omega1 <- 0.07
omega2 <- 0.06

F <- sin(2*pi*omega1 * (1:N)) + sin(2*pi*omega2 * (1:N))
s <- ssa(F, L)

# Apply FOSSA and then IOSSA
fs <- fossa(s, nested.groups = 1:4)
ios <- iossa(fs, nested.groups = list(1:2, 3:4), maxiter = 100)
summary(ios)

opar <- par(mfrow = c(3, 1))
plot(reconstruct(s, groups = list(1:2, 3:4)))
plot(reconstruct(fs, groups = list(1:2, 3:4)))
plot(reconstruct(ios, groups = ios$iossa.groups))
par(opar)

wo <- plot(wcor(ios, groups = 1:4))
gwo <- plot(owcor(ios, groups = 1:4))
plot(wo, split = c(1, 1, 2, 1), more = TRUE)
plot(gwo, split = c(2, 1, 2, 1), more = FALSE)



data(USUnemployment)
unempl.male <- USUnemployment[, "MALE"]

s <- ssa(unempl.male)
ios <- iossa(s, nested.groups = list(c(1:4, 7:11), c(5:6, 12:13)))
summary(ios)

# Comparison of reconstructions
rec <- reconstruct(s, groups = list(c(1:4, 7:11), c(5:6, 12:13)))
iorec <- reconstruct(ios, groups <- ios$iossa.groups)
# Trends
matplot(data.frame(iorec$F1, rec$F1, unempl.male), type='l',
        col=c("red","blue","black"), lty=c(1,1,2))
```

```
# Seasonalities
matplot(data.frame(iorec$F2, rec$F2), type='l', col=c("red","blue"),lty=c(1,1))

# W-cor matrix before IOSSA and w-cor matrix after it
ws <- plot(wcor(s, groups = 1:30), grid = 14)
wios <- plot(wcor(ios, groups = 1:30), grid = 14)
plot(ws, split = c(1, 1, 2, 1), more = TRUE)
plot(wios, split = c(2, 1, 2, 1), more = FALSE)

# Eigenvectors before and after Iterative O-SSA
plot(s, type = "vectors", idx = 1:13)
plot(ios, type = "vectors", idx = 1:13)

# 2D plots of periodic eigenvectors before and after Iterative O-SSA
plot(s, type = "paired", idx = c(5, 12))
plot(ios, type = "paired", idx = c(10, 12), plot.contrib = FALSE)

data(AustralianWine)
Fortified <- AustralianWine[, "Fortified"]
s <- ssa(window(Fortified, start = 1982 + 5/12, end = 1986 + 5/12), L = 18)
ios <- iossa(s, nested.groups = list(trend = 1, 2:7),
             kappa = NULL,
             maxIter = 1)
fs <- fossa(s, nested.groups = 1:7, gamma = 1000)

rec.ssa <- reconstruct(s, groups = list(trend = 1, 2:7))
rec.iossa <- reconstruct(ios, groups = ios$iossa.groups);
rec.fossa <- reconstruct(fs, groups = list(trend = 7, 1:6))

Fort <- cbind(`Basic SSA trend` = rec.ssa$trend,
              `Iterative O-SSA trend` = rec.iossa$trend,
              `DerivSSA trend` = rec.fossa$trend,
              `Full series` = Fortified)

library(lattice)
xyplot(Fort, superpose = TRUE, col = c("red", "blue", "green4", "black"))



# Shaped 2D I. O-SSA separates finite rank fields exactly
mx1 <- outer(1:50, 1:50,
             function(i, j) exp(i/25 - j/20))
mx2 <- outer(1:50, 1:50,
             function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7))

mask <- matrix(TRUE, 50, 50)
mask[23:25, 23:27] <- FALSE
mask[1:2, 1] <- FALSE
mask[50:49, 1] <- FALSE
mask[1:2, 50] <- FALSE

mx1[!mask] <- mx2[!mask] <- NA
```

```
s <- ssa(mx1 + mx2, kind = "2d-ssa", L = c(10, 10))
plot(reconstruct(s, groups = list(1, 2:5)))

ios <- iossa(s, nested.groups = list(1, 2:5), kappa = NULL)
plot(reconstruct(ios, groups = ios$iossa.groups))


# I. O-SSA for MSSA
N.A <- 150
N.B <- 120
L <- 40

omega1 <- 0.05
omega2 <- 0.055

tt.A <- 1:N.A
tt.B <- 1:N.B
F1 <- list(A = 2 * sin(2*pi * omega1 * tt.A), B = cos(2*pi * omega1 * tt.B))
F2 <- list(A = 1 * sin(2*pi * omega2 * tt.A), B = cos(2*pi * omega2 * tt.B))

F <- list(A = F1$A + F2$A, B = F1$B + F2$B)

s <- ssa(F, kind = "mssa")
plot(reconstruct(s, groups = list(1:2, 3:4)), plot.method = "xyplot")

ios <- iossa(s, nested.groups = list(1:2, 3:4), kappa = NULL)
plot(reconstruct(ios, groups = ios$iossa.groups), plot.method = "xyplot")
```

`iossa.result` *Summary of Iterative O-SSA results*

Description

Various routines to print Iterative Oblique SSA results

Usage

```
  ## S3 method for class 'iossa.result'
print(x, digits = max(3, getOption("digits") - 3), ...)
  ## S3 method for class 'iossa.result'
summary(object, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x, object`	object of class 'iossa.result' or 'ossa'
`digits`	integer, used for number formatting
`...`	further arguments passed to method

Details

An object of class 'iossa.result' is a list with the following fields:

converged logical, whether algorithm has been converged

iter the number of OSSA iterations

cond numeric vector with two elements, condition numbers of the final column and row inner products

initial.tau numeric vector, proportions of high rank components contribution for each of initial series (denotes how well the series is approximated by a series of finite rank)

tau numeric vector, proportions of high rank components contribution for each of final series

initial.wcor W-correlation matrix of the initial nested decomposition

wcor W-correlations matrix of the final nested decomposition

owcor oblique W-correlation matrix (see owcor) of the final nested decomposition

initial.rec list of initial series (reconstructed initial nested decomposition)

kappa, maxiter, tol Iterative O-SSA procedure parameters

References

Golyandina N. and Shlemov A. (2015): *Variations of Singular Spectrum Analysis for separability improvement: non-orthogonal decompositions of time series*, Statistics and Its Interface. Vol.8, No 3, P.277-294. https://arxiv.org/abs/1308.4022

See Also

Rssa for an overview of the package, as well as, iossa, owcor, summary.ssa.

Examples

```
# Separate three non-separable sines with different amplitudes
N <- 150
L <- 70

omega1 <- 0.05
omega2 <- 0.06
omega3 <- 0.07

F <- 4*sin(2*pi*omega1 * (1:N)) + 2*sin(2*pi*omega2 * (1:N)) + sin(2*pi*omega3 * (1:N))
s <- ssa(F, L)
ios <- iossa(s, nested.groups = list(1:2, 3:4, 5:6), kappa = NULL, maxiter = 100, tol = 1e-3)

print(ios)
print(ios$iossa.result)
```

`lrr` *Calculate the min-norm Linear Recurrence Relation*

Description

Calculates the min-norm Linear Recurrence Relation given the one-dimensional 'ssa' object.

Usage

```
## S3 method for class '1d.ssa'
lrr(x, groups, reverse = FALSE, ..., drop = TRUE)
## S3 method for class 'toeplitz.ssa'
lrr(x, groups, reverse = FALSE, ..., drop = TRUE)
## Default S3 method:
lrr(x, eps = sqrt(.Machine$double.eps),
        reverse = FALSE, ..., orthonormalize = TRUE)
## S3 method for class 'lrr'
roots(x, ..., method = c("companion", "polyroot"))
## S3 method for class 'lrr'
plot(x, ..., raw = FALSE)
```

Arguments

`x`	SSA object holding the decomposition or matrix containing the basis vectors in columns for `lrr` call or 'lrr' object itself for other function calls
`groups`	list, the grouping of eigentriples used to derive the LRR
`reverse`	logical, if 'TRUE', then LRR is assumed to go back
`...`	further arguments to be passed to `decompose` or `plot` call, if necessary
`drop`	logical, if 'TRUE' then the result is coerced to lrr object itself, when possible (length of 'groups' is one)
`eps`	Tolerance for verticality checking
`method`	methods used for calculation of the polynomial roots: via eigenvalues of companion matrix or R's standard `polyroot` routine
`raw`	logical, if 'TRUE' then `plot` routine will not add any additional plot components (e.g. unit circle)
`orthonormalize`	logical, if 'FALSE' then the basis is assumed orthonormal. Otherwise, orthonormalization is performed

Details

Produces the min-norm linear recurrence relation from the series. The default implementation works as follows.

Denote by U_i the columns of matrix x. Denote by $\tilde{U}_i$ the same vector U_i but without the last coordinate. Denote the last coordinate of U_i by π_i. The returned value is

$$\mathcal{R} = \frac{1}{1-\nu^2} \sum_{i=1}^{d} \pi_i \tilde{U}_i,$$

where

$$\nu^2 = \pi_1^2 + \ldots + \pi_d^2.$$

For `lrr.ssa` case the matrix U used is the matrix of basis vector corresponding to the selected elementary series.

For `reverse = 'TRUE'` everything is the same, besides the last coordinate substituted for the first coordinate.

Details of the used algorithm see in Golyandina et al (2018), Algorithms 3.1 and 3.2.

Value

Named list of object of class 'lrr' for `lrr` function call, where elements have the same names as elements of `groups` (if group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in `groups` list). Or the object itself if 'drop = TRUE' and groups has length one.

Vector with the roots of the of the characteristic polynomial of the LRR for `roots` function call. Roots are ordered by moduli decreasing.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R.* Use R!. Springer, Berlin, Heidelberg.

See Also

`Rssa` for an overview of the package, as well as, `ssa`, `parestimate`,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2, L = 24)
# Calculate the LRR out of first 3 eigentriples
l <- lrr(s, groups = list(1:3))
# Calculate the roots of the LRR
r <- roots(l)
# Moduli of the roots
Mod(r)
# Periods of three roots with maximal moduli
2*pi/Arg(r)[1:3]
# Plot the roots
plot(l)
```

Mars *Webcam image of Mars*

Description

Image of Mars obtained by a webcam. 258 x 275, grayscale, from 0 to 255.

Usage

```
data(Mars)
```

Format

A double matrix with integer values.

Source

Thierry, P. Tutorial for IRIS 5.59 (an astronomical images processing software), http://www.astrosurf.com/buil/iris/tutorial8/doc23_us.htm. Last updated: 20.12.2005

MotorVehicle *Total U.S. Domestic and Foreign Car Sales*

Description

Monthly series containing total domestic and foreign car sales in the USA in thousands, from 1967 till 2010.

Usage

```
data(MotorVehicle)
```

Format

A time series of length 541.

Source

U.S. Bureau of Economic Analysis. Table 7.2.5S. Auto and Truck Unit Sales Production Inventories Expenditures and Price, 2010.

owcor *Calculate generalized (oblique) W-correlation matrix*

Description

Function calculates oblique W-correlation matrix for the series.

Usage

```
owcor(x, groups, ..., cache = TRUE)
```

Arguments

x	the input object of 'ossa' class
groups	list of numeric vectors, indices of elementary components used for reconstruction. The elementary components must belong to the current OSSA component set
...	further arguments passed to `reconstruct` routine
cache	logical, if 'TRUE' then intermediate results will be cached in 'ssa' object.

Details

Matrix of oblique weighted correlations will be computed. For two series, oblique W-covariation is defined as follows:

$$\mathrm{owcov}(F_1, F_2) = \langle L^\dagger X_1 (R^\dagger)^{\mathrm{T}}, L^\dagger X_2 (R^\dagger)^{\mathrm{T}} \rangle_{\mathrm{F}},$$

where X_1, X_2 denotes the trajectory matrices of series F_1, F_2 correspondingly, $L = [U_{b_1} : ... : U_{b_r}], R = [V_{b_1} : ... V_{b_r}]$, where
$\{b_1, \ldots, b_r$
$\}$ is current OSSA component set (see description of 'ossa.set' field of 'ossa' object), '$\langle \cdot, \cdot \rangle_{\mathrm{F}}$' denotes Frobenius matrix inner product and '†' denotes Moore-Penrose pseudo-inverse matrix.

And oblique W-correlation is defined the following way:

$$\mathrm{owcor}(F_1, F_2) = \frac{\mathrm{owcov}(F_1, F_2)}{\sqrt{\mathrm{owcov}(F_1, F_1) \cdot \mathrm{owcov}(\mathrm{F}_2, \mathrm{F}_2)}}$$

Oblique W-correlation is an OSSA analogue of W-correlation, that is, a measure of series separability. If I-OSSA procedure separates series exactly, their oblique W-correlation will be equal to zero.

Value

Object of class 'wcor.matrix'

References

Golyandina N. and Shlemov A. (2015): *Variations of Singular Spectrum Analysis for separability improvement: non-orthogonal decompositions of time series*, Statistics and Its Interface. Vol.8, No 3, P.277-294. https://arxiv.org/abs/1308.4022

See Also

Rssa for an overview of the package, as well as, wcor, iossa, fossa.

Examples

```
# Separate two non-separable sines
N <- 150
L <- 70

omega1 <- 0.06
omega2 <- 0.065

F <- 4*sin(2*pi*omega1 * (1:N)) + sin(2*pi*omega2 * (1:N))
s <- ssa(F, L)
ios <- iossa(s, nested.groups = list(1:2, 3:4), kappa = NULL, maxIter = 200, tol = 1e-8)

p.wcor <- plot(wcor(ios, groups = list(1:2, 3:4)))
p.owcor <- plot(owcor(ios, groups = list(1:2, 3:4)), main = "OW-correlation matrix")
print(p.wcor, split = c(1, 1, 2, 1), more = TRUE)
print(p.owcor, split = c(2, 1, 2, 1))
```

---

parestimate *Estimate periods from (set of) eigenvectors*

Description

Function to estimate the parameters (frequencies and rates) given a set of SSA eigenvectors.

Usage

```
## S3 method for class '1d.ssa'
parestimate(x, groups, method = c("esprit", "pairs"),
            subspace = c("column", "row"),
            normalize.roots = NULL,
            dimensions = NULL,
            solve.method = c("ls", "tls"),
            ...,
            drop = TRUE)
## S3 method for class 'toeplitz.ssa'
parestimate(x, groups, method = c("esprit", "pairs"),
            subspace = c("column", "row"),
            normalize.roots = NULL,
```

```
            dimensions = NULL,
            solve.method = c("ls", "tls"),
            ...,
            drop = TRUE)
## S3 method for class 'mssa'
parestimate(x, groups, method = c("esprit", "pairs"),
            subspace = c("column", "row"),
            normalize.roots = NULL,
            dimensions = NULL,
            solve.method = c("ls", "tls"),
            ...,
            drop = TRUE)
## S3 method for class 'cssa'
parestimate(x, groups, method = c("esprit", "pairs"),
            subspace = c("column", "row"),
            normalize.roots = NULL,
            dimensions = NULL,
            solve.method = c("ls", "tls"),
            ...,
            drop = TRUE)
## S3 method for class 'nd.ssa'
parestimate(x, groups,
            method = c("esprit"),
            subspace = c("column", "row"),
            normalize.roots = NULL,
            dimensions = NULL,
            solve.method = c("ls", "tls"),
            pairing.method = c("diag", "memp"),
            beta = 8,
            ...,
            drop = TRUE)
```

Arguments

x	SSA object
groups	list of indices of eigenvectors to estimate from
...	further arguments passed to 'decompose' routine, if necessary
drop	logical, if 'TRUE' then the result is coerced to lowest dimension, when possible (length of groups is one)
dimensions	a vector of dimension indices to perform ESPRIT along. 'NULL' means all dimensions.
method	For 1D-SSA, Toeplitz SSA, and MSSA: parameter estimation method, 'esprit' for 1D-ESPRIT (Algorithm 3.3 in Golyandina et al (2018)), 'pairs' for rough estimation based on pair of eigenvectors (Algorithm 3.4 in Golyandina et al (2018)). For nD-SSA: parameter estimation method. For now only 'esprit' is supported (Algorithm 5.6 in Golyandina et al (2018)).
solve.method	approximate matrix equation solving method, 'ls' for least-squares, 'tls' for total-least-squares.

`pairing.method` method for esprit roots pairing, 'diag' for '2D-ESPRIT diagonalization', 'memp' for "MEMP with an improved pairing step'

`subspace` which subspace will be used for parameter estimation

`normalize.roots`
logical vector or 'NULL', force signal roots to lie on unit circle. 'NULL' means automatic selection: normalize iff circular topology OR Toeplitz SSA used

`beta` In nD-ESPRIT, coefficient(s) in convex linear combination of shifted matrices. The length of `beta` should be `ndim - 1`, where `ndim` is the number of independent dimensions. If only one value is passed, it is expanded to a geometric progression.

Details

See Sections 3.1 and 5.3 in Golyandina et al (2018) for full details.

Briefly, the time series is assumed to satisfy the model

$$x_n = \sum_k C_k \mu_k^n$$

for complex μ_k or, alternatively,

$$x_n = \sum_k A_k \rho_k^n \sin(2\pi\omega_k n + \phi_k).$$

The return value are the estimated moduli and arguments of complex μ_k, more precisely, ρ_k ('moduli') and $T_k = 1/\omega_k$ ('periods').

For images, the model

$$x_{ij} = \sum_k C_k \lambda_k^i \mu_k^j$$

is considered.

Also 'print' and 'plot' methods are implemented for classes 'fdimpars.1d' and 'fdimpars.nd'.

Value

For 1D-SSA (and Toeplitz), a list of objects of S3-class 'fdimpars.1d'. Each object is a list with 5 components:

roots complex roots of minimal LRR characteristic polynomial

periods periods of dumped sinusoids

frequencies frequencies of dumped sinusoids

moduli moduli of roots

rates rates of exponential trend (`rates == log(moduli)`)

For 'method' = 'pairs' all moduli are set equal to 1 and all rates equal to 0.

For nD-SSA, a list of objects of S3-class 'fdimpars.nd'. Each object is named list of `n` 'fdimpars.1d' objects, each for corresponding spatial coordinate.

In all cases elements of the list have the same names as elements of `groups`. If group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in `groups` list.

If 'drop = TRUE' and length of 'groups' is one, then corresponding list of estimated parameters is returned.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Roy, R., Kailath, T., (1989): *ESPRIT: estimation of signal parameters via rotational invariance techniques*. IEEE Trans. Acoust. 37, 984–995.

Rouquette, S., Najim, M. (2001): *Estimation of frequencies and damping factors by two- dimensional esprit type methods*. IEEE Transactions on Signal Processing 49(1), 237–245.

Wang, Y., Chan, J-W., Liu, Zh. (2005): *Comments on "estimation of frequencies and damping factors by two-dimensional esprit type methods"*. IEEE Transactions on Signal Processing 53(8), 3348–3349.

Shlemov A, Golyandina N (2014) Shaped extensions of Singular Spectrum Analysis. In: 21st international symposium on mathematical theory of networks and systems, July 7–11, 2014. Groningen, The Netherlands, pp 1813–1820.

See Also

Rssa for an overview of the package, as well as, ssa, lrr,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2, neig = 20)
# Estimate the periods from 2nd and 3rd eigenvectors using 'pairs' method
print(parestimate(s, groups = list(c(2, 3)), method = "pairs"))
# Estimate the peroids from 2nd, 3rd, 5th and 6th eigenvectors using ESPRIT
pe <- parestimate(s, groups = list(c(2, 3, 5, 6)), method = "esprit")
print(pe)
plot(pe)


# Artificial image for 2D SSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with default parameters
s <- ssa(mx, kind = "2d-ssa")
# Estimate parameters
pe <- parestimate(s, groups = list(1:5))
print(pe)
plot(pe, col = c("green", "red", "blue"))

# Real example: Mars photo
data(Mars)
# Decompose only Mars image (without background)
s <- ssa(Mars, mask = Mars != 0, wmask = circle(50), kind = "2d-ssa")
# Reconstruct and plot texture pattern
plot(reconstruct(s, groups = list(c(13,14, 17, 18))))
# Estimate pattern parameters
pe <- parestimate(s, groups = list(c(13,14, 17, 18)))
print(pe)
```

```
plot(pe, col = c("green", "red", "blue", "black"))
```

---

plot *Plot SSA object*

Description

This function plots various sorts of figures related to the SSA method.

Usage

```
## S3 method for class 'ssa'
plot(x,
     type = c("values", "vectors", "paired", "series", "wcor"),
     ...,
     vectors = c("eigen", "factor"),
     plot.contrib = TRUE,
     numvalues = nsigma(x),
     numvectors = min(nsigma(x), 10),
     idx = 1:numvectors,
     idy,
     groups)
```

Arguments

x	SSA object holding the decomposition
type	Type of the plot (see 'Details' for more information)
...	Arguments to be passed to methods, such as graphical parameters
vectors	For type = 'vectors', choose the vectors to plot
plot.contrib	logical. If 'TRUE' (the default), the contribution of the component to the total variance is plotted. For 'ossa' class, Frobenius orthogonality checking of elementary matrices is performed. If not all matrices are orthogonal, corresponding warning is risen
numvalues	Number of eigenvalues to plot (for type = 'values')
numvectors	Total number of eigenvectors to plot (for type = 'vectors')
idx	Indices of eigenvectors to plot (for type = 'vectors')
idy	Second set of indices of eigenvectors to plot (for type = 'paired')
groups	Grouping used for the decomposition (see reconstruct)

Details

This function is the single entry to various plots of SSA objects. Right now this includes:

values plot the graph of the component norms.

vectors plot the eigenvectors.

paired plot the pairs of eigenvectors (useful for the detection of periodic components).

series plot the reconstructed series.

wcor plot the W-correlation matrix for the reconstructed objects.

Additional (non-standard) graphical parameters which can be transfered via ...:

plot.type lattice plot type. This argument will be transfered as `type` argument to function `panel.xyplot`.

ref logical. Whether to plot zero-level lines in series-plot, eigenvectors-plot and paired-plot. Zero-level isolines will be plotted for 2d-eigenvectors-plot.

symmetric logical. Whether to use symmetric scales in series-plot, eigenvectors-plot and paired-plot.

useRaster logical. For 2d-eigenvector-plot and wcor-plot, indicating whether raster representations should be used. 'TRUE' by default.

col color vector for colorscale (for 2d- and wcor-plots), given by two or more colors, the first color corresponds to the minimal value, while the last one corresponds to the maximal value (will be interpolated by `colorRamp`)

zlim for 2d-plot, range of displayed values

at for 2d-eigenvectors-plot, a numeric vector giving breakpoints along the range of `z`, a list of such vectors or a character string. If a list is given, corresponding list element (with recycling) will be used for each plot panel. For character strings, values 'free' and 'same' are allowed: 'free' means special breakpoints' vectors (will be evaluated automatically, see description of `cuts` argument in 'Details') for each component. 'same' means one breakpoints' vector for all component (will be evaluated automatically too)

cuts for 2d-reconstruction-plot, the number of levels the range of `z` would be divided into.

fill.color color or 'NULL'. Defines background color for shaped 2d-eigenvectors plot. If 'NULL', standard white background will be used.

See Also

`ssa-object`, `ssa` `plot.reconstruction`,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Plot the eigenvalues
plot(s, type = "values")
# Plot W-cor matrix for first 10 reconstructed components
plot(s, type = "wcor", groups = 1:10)
# Plot the paired plot for first 6 eigenvectors
```

```
plot(s, type = "paired", idx = 1:6)
# Plot eigenvectors for first 6 components
plot(s, type = "vectors", idx = 1:6)
# Plot the first 4 reconstructed components
plot(s, type = "series", groups = list(1:4))
# Plot the eigenvalues by points only
plot(s, type = "values", plot.type = "p")

# Artificial image for 2dSSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with default parameters
s <- ssa(mx, kind = "2d-ssa")
# Plot the eigenvalues
plot(s, type = "values")
# Plot eigenvectors for first 6 components
plot(s, type = "vectors", idx = 1:6,
     ref = TRUE, at = "same", cuts = 50,
     plot.contrib = TRUE, symmetric = TRUE)
# Plot factor vectors for first 6 components
plot(s, type = "vectors", vectors = "factor", idx = 1:6,
     ref = TRUE, at = "same", cuts = 50,
     plot.contrib = TRUE, symmetric = TRUE)
# Plot wcor for first 12 components
plot(s, type = "wcor", groups = 1:12, grid = c(2, 6))

# 3D-SSA example (2D-MSSA)
data(Barbara)
ss <- ssa(Barbara, L = c(50, 50, 1))
plot(ss, type = "values")
plot(ss, type = "vectors", idx = 1:12, slice = list(k = 1),
     cuts = 50, plot.contrib = TRUE)
plot(ss, type = "vectors", idx = 1:12, slice = list(k = 1, i = 1))
plot(ss, type = "vectors", vectors = "factor", idx = 1:12, slice = list(k = 3),
     cuts = 50, plot.contrib = FALSE)
plot(ss, type = "series", groups = 1:12, slice = list(k = 1))
plot(ss, type = "series", groups = 1:12, slice = list(k = 1, i = 1))
plot(ss, plot.method = "xyplot", type = "series", groups = 1:12, slice = list(k = 1, i = 1))
```

`plot.reconstruction` *Plot the results of SSA reconstruction*

Description

Plot the result of SSA Reconstruction step

Usage

```
## S3 method for class '1d.ssa.reconstruction'
plot(x, ...,
     type = c("raw", "cumsum"),
     plot.method = c("native", "matplot", "xyplot"),
     base.series = NULL,
     add.original = TRUE,
     add.residuals = TRUE)
## S3 method for class 'toeplitz.ssa.reconstruction'
plot(x, ...,
     type = c("raw", "cumsum"),
     plot.method = c("native", "matplot", "xyplot"),
     base.series = NULL,
     add.original = TRUE,
     add.residuals = TRUE)
## S3 method for class 'mssa.reconstruction'
plot(x,
     slice = list(),
     ...,
     type = c("raw", "cumsum"),
     plot.method = c("native", "matplot", "xyplot"),
     na.pad = c("left", "right"),
     base.series = NULL,
     add.original = TRUE,
     add.residuals = TRUE)
## S3 method for class '2d.ssa.reconstruction'
plot(x, ...,
     type = c("raw", "cumsum"),
     base.series = NULL,
     add.original = TRUE,
     add.residuals = TRUE,
     add.ranges,
     col = grey(c(0, 1)),
     zlim,
     at)
## S3 method for class 'nd.ssa.reconstruction'
plot(x, slice, ...)
```

Arguments

x	SSA object holding the decomposition
slice	for ‘mssa’: list with elements named ’series’ and ’components’; for ‘nd.ssa’: list with elements named ’i’, ’j’, ’k’ or ’x’, ’y’, ’z’, ’t’ or ’d1’, ’d2’, ... or ‘1‘, ‘2‘, ...; works like ’[’-operator, allows one to select which components from the reconstruction of multivariate time series or which subarray from reconstruction of multidimentional array to draw.
type	Type of the plot (see ’Details’ for more information)

`...`	Arguments to be passed to methods, such as graphical parameters
`plot.method`	Plotting method to use: either ordinary all-in-one via matplot or xyplot, or native plotting method of the input time series
`na.pad`	select how to pad the series of unequal length with NA's
`base.series`	another SSA reconstruction object, the series of which should be considered as an original. Useful for plotting the results of sequential SSA
`add.original`	logical, if 'TRUE' then the original series are added to the plot
`add.residuals`	logical, if 'TRUE' then the residuals are added to the plot
`col`	color vector for colorscale, given by two or more colors, the first color corresponds to the minimal value, while the last one corresponds to the maximal value (will be interpolated by `colorRamp`)
`zlim`	for 2d-plot, range of displayed values
`at`	for 2d-eigenvectors-plot, a numeric vector giving breakpoints along the range of `z`, a list of such vectors or a character string. If a list is given, corresponding list element (with recycling) will be used for each plot panel. For character strings, values 'free' and 'same' are allowed: 'free' means special breakpoints' vectors (will be evaluated automatically, see description of `cuts` argument in 'Details') for each component. 'same' means one breakpoints' vector for all component (will be evaluated automatically too)
`add.ranges`	logical, if 'TRUE', the range of the components values will be printed in panels captions

Details

Additional (non-standard) graphical parameters applicable to 2D SSA plots can be transfered via `...`:

cuts the number of levels the range of image would be divided into.

ref logical, whether to plot zero-level isolines

symmetric logical, whether to use symmetric image range scale

useRaster logical, indicates whether raster representations should be used. 'TRUE' by default.

fill.uncovered single number, matrix, one of the following strings: 'mean', 'original', 'void' or a list of such objects. For shaped 2d-reconstruction-plot this argument defines filling method for uncovered by window array elements on components and residuals plots. If number, all uncovered elements will be replaced by it. If matrix, all uncovered elements will be replaced by corresponding matrix elements. If 'mean', they will be replaced by mean value of current component. If 'original', they will be replaced by corresponding elements of original array. 'void' (by default) means no filling. If list is given, corresponding list element (with recycling) will be used for each plot panel.

fill.color color or 'NULL'. Defines background color for shaped 2d-reconstruction plot. If 'NULL', standard white background will be used.

See Also

`ssa-object`, `ssa reconstruct`, `plot`,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
r <- reconstruct(s, groups = list(c(1, 4), c(2, 3), c(5, 6)))
# Plot full 'co2' reconstruction into trend, periodic components and noise
plot(r)

# Artificial image for 2dSSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with default parameters
s <- ssa(mx, kind = "2d-ssa")
# Reconstruct
r <- reconstruct(s, groups = list(1, 2:5))
# Plot components, original image and residuals
plot(r)
# Plot cumulative sum of components only
plot(r, type = "cumsum", add.residuals = FALSE, add.original = FALSE)

# Real example: Mars photo
data(Mars)
# Decompose only Mars image (without backgroud)
s <- ssa(Mars, mask = Mars != 0, wmask = circle(50), kind = "2d-ssa")
# Reconstruct and plot trend
plot(reconstruct(s, 1), fill.uncovered = "original")
# Reconstruct and plot texture pattern
plot(reconstruct(s, groups = list(c(13, 14, 17, 18))))

# Decompose 'EuStockMarkets' series with default parameters
s <- ssa(EuStockMarkets, kind = "mssa")
r <- reconstruct(s, groups = list(Trend = 1:2))
# Plot original series, trend and residuals superimposed
plot(r, plot.method = "xyplot", superpose = TRUE,
     auto.key = list(columns = 3),
     col = c("blue", "green", "red", "violet"),
     lty = c(rep(1, 4), rep(2, 4), rep(3, 4)))
# Plot the series separately
plot(r, plot.method = "xyplot", add.residuals = FALSE,
     screens = list(colnames(EuStockMarkets)),
     col = c("blue", "green", "red", "violet"),
     lty = c(rep(1, 4), rep(2, 4), rep(3, 4)))

# 3D-SSA example (2D-MSSA)
data(Barbara)
ss <- ssa(Barbara, L = c(50, 50, 1))
plot(reconstruct(ss, groups = 1), slice = list(k = 1))
```

precache *Calculates and caches elementary components inside SSA object*

Description

Calculates all the elementary series and saves inside SSA object. After this the grouping procedure can be performed much faster.

Usage

```
precache(x, n, ...)
```

Arguments

x	SSA object
n	integer, number of series to calculate and save
...	further arguments passed to the reconstruction routines

Note

In most cases it is not necessary to call this routine directly. By default functions from the package collect all elementary series they encounter during the calculations.

See Also

reconstruct

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
summary(s)
# Precache the stuff
precache(s)
summary(s)
```

reconstruct *Perform a series reconstruction*

Description

Reconstruct the data given the SSA decomposition and the desired grouping of the elementary components.

Usage

```
## S3 method for class 'ssa'
reconstruct(x, groups, ..., drop.attributes = FALSE, cache = TRUE)
```

Arguments

x	SSA object
groups	list of numeric vectors, indices of elementary components used for reconstruction, the entries of the list can be named, see 'Value' for more information
...	further arguments passed to routines (e.g. to decompose routine if the continuation is desired).
drop.attributes	logical, if 'TRUE' then the attributes of the input objects are not copied to the reconstructed ones.
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

Reconstruction is performed in a common form for different types of input objects. See Section 1.1.2.6 in Golyandina et al (2018) for the explanation. Formal algorithms are described in this book in Algorithm 2.2 for 1D-SSA, Algorithm 4.3 for MSSA, Algorithm 5.2 for 2D-SSA and Algorithm 5.6 for Shaped 2D-SSA.

Fast implementation of reconstruction with the help of FFT is described in Korobeynikov (2010) for the 1D case and in Section 6.2 (Rank-one quasi-hankelization) of Golyandina et al (2015) for the general case.

Value

List of reconstructed objects. Elements of the list have the same names as elements of groups. If the group is unnamed, then corresponding component will obtain name 'Fn', where 'n' is its index in groups list.

Note

By default (argument drop.attributes) the routine tries to preserve all the attributes of the input object. This way, for example, the reconstruction result of 'ts' object is the 'ts' object with the same time scale.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Korobeynikov, A. (2010): *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

Golyandina, N., Korobeynikov, A., Shlemov, A. and Usevich, K. (2015): *Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package*. Journal of Statistical Software, Vol. 67, Issue 2. doi:10.18637/jss.v067.i02

See Also

Rssa for an overview of the package, as well as, ssa-input, ssa, plot.reconstruction,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Reconstruct the series, grouping elementary series.
r <- reconstruct(s, groups = list(Trend = c(1, 4), Season1 = c(2,3), Season2 = c(5, 6)))
plot(r)
# 'groups' argument might contain duplicate entries as well
r <- reconstruct(s, groups = list(1, 1:4, 1:6))
plot(r)


# Real example: Mars photo
data(Mars)
# Decompose only Mars image (without backgroud)
s <- ssa(Mars, mask = Mars != 0, wmask = circle(50), kind = "2d-ssa")
# Reconstruct and plot trend
plot(reconstruct(s, 1), fill.uncovered = "original")
# Reconstruct and plot texture pattern
plot(reconstruct(s, groups = list(c(13, 14, 17, 18))))

# Decompose 'EuStockMarkets' series with default parameters
s <- ssa(EuStockMarkets, kind = "mssa")
r <- reconstruct(s, groups = list(Trend = 1:2))
# Plot original series, trend and residuals superimposed
plot(r, plot.method = "xyplot", superpose = TRUE,
     auto.key = list(columns = 3),
     col = c("blue", "green", "red", "violet"),
     lty = c(rep(1, 4), rep(2, 4), rep(3, 4)))
```

---

residuals *Obtain the residuals from SSA reconstruction*

Description

Obtain the residuals from SSA reconstruction

Usage

```
## S3 method for class 'ssa'
residuals(object, groups, ..., cache = TRUE)
## S3 method for class 'ssa.reconstruction'
residuals(object, ...)
```

Arguments

object	input object
groups	list of numeric vectors, indices of elementary components used for reconstruction, the entries of the list can be named.
...	further arguments passed to reconstruct routine
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

This function calculates the residuals either from SSA object corresponding to reconstruction using groups arguments, or just extracts the residuals from reconstruction object.

Value

residuals object

See Also

Rssa for an overview of the package, as well as, reconstruct.

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Reconstruct the series, grouping elementary series.
r <- reconstruct(s, groups = list(c(1, 4), c(2,3), c(5, 6)))
print(residuals(r))

# If there are several groups, then the residuals are calculated as
# residuals for the model corresponding to the combined model.
r <- reconstruct(s, groups = list(c(6, 7), c(6,7), c(8, 9)))
r1 <- reconstruct(s, groups = list(6:9))
max(abs(residuals(r) - residuals(r1))) # 0
max(abs(co2 - (r1$F1 + residuals(r1)))) # 0
```

---

rforecast *Perform recurrent SSA forecasting of the series*

Description

Perform recurrent SSA forecasting of the series.

Usage

```
## S3 method for class '1d.ssa'
rforecast(x, groups, len = 1, base = c("reconstructed", "original"),
          only.new = TRUE, reverse = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'toeplitz.ssa'
rforecast(x, groups, len = 1, base = c("reconstructed", "original"),
          only.new = TRUE, reverse = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'mssa'
rforecast(x, groups, len = 1, base = c("reconstructed", "original"),
          direction = c("row", "column"), only.new = TRUE, ..., drop = TRUE,
          drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'cssa'
rforecast(x, groups, len = 1, base = c("reconstructed", "original"),
          only.new = TRUE, reverse = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
## S3 method for class 'pssa.1d.ssa'
rforecast(x, groups, len = 1, base = c("reconstructed", "original"),
          only.new = TRUE, reverse = FALSE, ...,
          drop = TRUE, drop.attributes = FALSE, cache = TRUE)
```

Arguments

x	SSA object holding the decomposition
groups	list, the grouping of eigentriples to be used in the forecast
len	integer, the desired length of the forecasted series
base	series used as a 'seed' of forecast: original or reconstructed according to the value of groups argument
direction	direction of forecast in multichannel SSA case, "column" stands for so-called L-forecast and "row" stands for K-forecast
only.new	logical, if 'TRUE' then only forecasted values are returned, whole series otherwise
reverse	logical, direction of forecast in 1D SSA case, 'FALSE' (default) means that the forecast moves forward in the time and 'TRUE' means the opposite
...	additional arguments passed to reconstruct routines
drop	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)
drop.attributes	logical, if 'TRUE' then the attributes of the input series are not copied to the reconstructed ones.
cache	logical, if 'TRUE' then intermediate results will be cached in the SSA object.

Details

The routines applies the recurrent SSA forecasting algorithm to produce the new series which is expected to 'continue' the current series on the basis of a given decomposition. The algorithm sequentialy projects the incomplete embedding vectors (from either the original or the reconstructed series) onto the subspace spanned by the selected eigentriples of the decomposition to derive the missed (last) values of the such vectors. Then the filled value In such a way the forecasting elements are produced on one-by-one basis.

It is shown in Golyandina et al (2001) that this approach corresponds to application of a linear recurrence formula (the same formula as described in `lrr`) to initial data taken from either the original or the reconstructed series.

In particular, the m-th step of the forecast is calculated by means of linear recurrence relation (see `lrr`) as $y_{n+m} = \sum_{k=1}^{L-1} a_k y_{n+m-k}$ where the starting points $y_{n-(L-2)}, \ldots, y_n$ are taken from the reconstructed time series (`base="reconstructed"`) or from the initial (`base="initial"`) time series.

For multichannel SSA the column forecast is obtained via applying the LRR to each series separately. The row forecast is more complicated and is based on a multivariate LRR. Forecast uses the formulae from Golyandina and Stepanov (2005) and Golyandina et.al (2015).

For details of 1D-SSA recurrent forecasting, see Section 3.2.1.2 and Algorithm 3.5 in Golyandina et al (2018). For details of MSSA recurrent forecasting, see Section 4.3.1.2 and Algorithm 4.4 (column forecasting).

Value

List of forecasted objects. Elements of the list have the same names as elements of `groups`. If group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in `groups` list.

Or, the forecasted object itself, if length of groups is one and 'drop = TRUE'.

References

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

Golyandina, N., Korobeynikov, A., Shlemov, A. and Usevich, K. (2015): *Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package*. Journal of Statistical Software, Vol. 67, Issue 2. doi:10.18637/jss.v067.i02

Golyandina, N. and Stepanov, D. (2005): *SSA-based approaches to analysis and forecast of multi-dimensional time series*. In Proceedings of the 5th St.Petersburg Workshop on Simulation, June 26-July 2, 2005, St. Petersburg State University, St. Petersburg, 293–298. https://www.gistatgroup.com/gus/mssa2.pdf

See Also

`Rssa` for an overview of the package, as well as, `forecast`, `vforecast`, `bforecast`.

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
```

```
# Produce 24 forecasted values of the series using different sets of eigentriples
# as a base space for the forecast.
rfor <- rforecast(s, groups = list(c(1,4), 1:4), len = 24, only.new=FALSE)
matplot(data.frame(c(co2, rep(NA, 24)), rfor), type = "l")

# Forecast `co2' trend by SSA with projections
s <- ssa(co2, column.projector = 2, row.projector = 2)
len <- 100
rfor <- rforecast(s, groups = list(trend = seq_len(nspecial(s))), len = len, only.new = FALSE)
matplot(data.frame(c(co2, rep(NA, len)), rfor), type = "l")

# Forecast finite rank series with polynomial component by SSA with projections
v <- 5000 * sin(2*pi / 13 * (1:100)) +  (1:100)^2 + 10000
s <- ssa(v, row.projector = 2, column.projector = 2)
plot(rforecast(s, groups = list(all = 1:6), len = 100, only.new = FALSE), type = "l")
```

---

ssa *Create a new SSA object*

Description

Set up the SSA object and perform the decomposition, if necessary.

Usage

```
ssa(x,
    L = (N + 1) %/% 2,
    neig = NULL,
    mask = NULL,
    wmask = NULL,
    column.projector = "none",
    row.projector = "none",
    column.oblique = "identity",
    row.oblique = "identity",
    ...,
    kind = c("1d-ssa", "2d-ssa", "nd-ssa", "toeplitz-ssa", "mssa", "cssa"),
    circular = FALSE,
   svd.method = c("auto", "nutrlan", "propack", "svd", "eigen", "rspectra", "primme"),
    force.decompose = TRUE)
```

Arguments

x	object to be decomposed. See ssa-input for more information
L	integer, window length. Fixed to half of the series length by default. Should be vector of length 2 for 2d SSA
neig	integer, number of desired eigentriples. If 'NULL', then sane default value will be used, see 'Details'

`mask` for shaped 2d SSA case only. Logical matrix with same dimension as `x`. Specifies form of decomposed array. If 'NULL', then all non-NA elements will be used

`wmask` for shaped 2d SSA case only. Logical matrix which specifies window form. See 'Details' for more information about the window shape selection

`...` further arguments passed to `decompose`

`kind` SSA method. This includes ordinary 1d SSA, 2d SSA, Toeplitz variant of 1d SSA, multichannel variant of SSA and complex SSA

`circular` logical vector of one or two elements, describes series topology for 1d SSA and Toeplitz SSA or field topology for 2d SSA. 'TRUE' means series circularity for 1d case or circularity by a corresponding coordinate for 2d case. See Shlemov and Golyandina (2014) for more information

`svd.method` singular value decomposition method. See 'Details' for more info

`column.projector, row.projector` column and row signal subspaces projectors for SSA with projection. See 'Details' for information about methods of projectors specification

`column.oblique, row.oblique` column and row matrix weights for Weighted Oblique SSA. See 'Details' for information about how to use this feature

`force.decompose` logical, if 'TRUE' then the decomposition is performed before return.

Details

This is the main entry point to the package. This routine constructs the SSA object filling all necessary internal structures and performing the decomposition if necessary. For the comprehensive description of SSA modifications and their algorithms see Golyandina et al (2018).

Variants of SSA: The following implementations of the SSA method are supported (corresponds to different values of `kind` argument):

1d-ssa Basic 1d SSA as described in Chapter 1 of Golyandina et al (2001). This is also known as Broomhead-King variant of SSA or BK-SSA, see Broomhead and King (1986).

toeplitz-ssa Toeplitz variant of 1d SSA. See Section 1.7.2 in Golyandina et al (2001). This is also known as Vautard-Ghil variant of SSA or VG-SSA for analysis of stationary time series, see Vautard and Ghil (1989).

mssa Multichannel SSA for simultaneous decomposition of several time series (possible of unequal length). See Golyandina and Stepanov (2005).

cssa Complex variant of 1d SSA.

2d-ssa 2d SSA for decomposition of images and arrays. See Golyandina and Usevich (2009) and Golyandina et.al (2015) for more information.

nd-ssa Multidimensional SSA decomposition for arrays (tensors).

Window shape selection (for shaped 2d SSA): Window shape may be specified by argument `wmask`. If `wmask` is 'NULL', then standard rectangular window (specified by `L`) will be used.

Also in `wmask` one may use following functions:

circle(R) circular mask of radius `R`

triangle(side) mask in form of isosceles right-angled triangle with cathetus `side`. Right angle lay on topleft corner of container square matrix

These functions are not exported, they defined only for `wmask` expression. If one has objects with the same names and wants to use them rather than these functions, one should use special wrapper function `I()` (see 'Examples').

Projectors specification for SSA with projection: Projectors are specified by means of `column.projector` and `row.projector` arguments (see Golyandina and Shlemov (2017)). Each may be a matrix of orthonormal (otherwise QR orthonormalization process will be perfomed) basis of projection subspace, or single integer, which will be interpreted as dimension of orthogonal polynomial basis (note that the dimension equals to degree plus 1, e.g. quadratic basis has dimension 3), or one of following character strings (or unique prefix): 'none', 'constant' (or 'centering'), 'linear', 'quadratic' or 'qubic' for orthonormal bases of the corresponding functions.

Here is the the list of the most used options

both projectors are 'none' corresponds to ordinary 1D SSA,

column.projector='centering' corresponds to 1D SSA with centering,

column.projector='centering' and row.projector='centering' corresponds to 1D SSA with double centering.

SSA with centering and double centering may improve the separation of linear trend (see Golyandina et.al (2001) for more information).

Weighted Oblique SSA: Corresponding matrix norm weights may be specified for ordinary 1D SSA case by means of `column.oblique` and `row.oblique` arguments. These arguments should be either 'identical' or positive numeric vectors of length `L` and `N - L + 1` for `column.oblique` and `row.oblique` respectively.

Weighted Oblique SSA inside `Cadzow` iterations may improve finite-rank estimation of signal (see e.g. Cadzow(alpha) iterations in Zvonarev and Golyandina (2017) for more information).

SVD methods: The main step of the SSA method is the singular decomposition of the so-called series trajectory matrix. Package provides several implementations of this procedure (corresponds to different values of `svd.method`) argument:

auto Automatic method selection depending on the series length, window length, SSA kind and number of eigenvalues requested.

nutrlan Thick-restart Lanczos eigensolver which operates on cross-product matrix. This methods exploits the Hankel structure of the trajectory matrix efficiently and is really fast. The method allows the truncated SVD (only specifid amount of eigentriples to be computed) and the continuation of the decomposition. See Korobeynikov (2010) for more information.

propack SVD via implicitly restarted Lanczos bidiagonalization with partial reothogonalization. This methods exploits the Hankel structure of the trajectory matrix efficiently and is really fast. This is the 'proper' SVD implementation (the matrix of factor vectors are calculated), thus the memory requirements of the methods are higher than for nu-TRLAN. Usually the method is slightly faster that nu-TRLAN and more numerically stable. The method allows the truncated SVD (only specifid amount of eigentriples to be computed). See Korobeynikov (2010) for more information.

svd Full SVD as provided by LAPACK DGESDD routine. Neither continuation of the decomposition nor the truncated SVD is supported. The method does not assume anything special about the trajectory matrix and thus is slow.

eigen Full SVD via eigendecompsition of the cross-product matrix. In many cases faster than previous method, but still really slow for more or less non-trivial matrix sizes.

rspectra SVD via `svds` function from Rspectra package (if installed)

primme SVD via `svds` function from PRIMME package (if installed)

Usually the `ssa` function tries to provide the best SVD implementation for given series length and the window size. In particular, for small series and window sizes it is better to use generic black-box routines (as provided by 'svd' and 'eigen' methods). For long series special-purpose routines are to be used.

Value

Object of class 'ssa'. The precise layout of the object is mostly meant opaque and subject to change in different version of the package. See `ssa-object` for details.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Broomhead, D.S., and King, G.P. (1986a): *Extracting qualitative dynamics from experimental data*, Physica D, 20, 217–236.

Vautard, R., and Ghil, M. (1989): *Singular spectrum analysis in nonlinear dynamics, with applications to paleoclimatic time series*, Physica D, 35, 395–424.

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

Golyandina, N. and Stepanov, D. (2005): *SSA-based approaches to analysis and forecast of multidimensional time series*. In Proceedings of the 5th St.Petersburg Workshop on Simulation, June 26-July 2, 2005, St. Petersburg State University, St. Petersburg, 293–298. https://www.gistatgroup.com/gus/mssa2.pdf

Golyandina, N. and Usevich, K. (2009): *2D-extensions of singular spectrum analysis: algorithm and elements of theory.* In Matrix Methods: Theory, Algorithms, Applications. World Scientific Publishing, 450-474.

Korobeynikov, A. (2010): *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

Golyandina, N., Korobeynikov, A. (2012, 2014): *Basic Singular Spectrum Analysis and Forecasting with R*. Computational Statistics and Data Analysis, Vol. 71, Pp. 934-954. https://arxiv.org/abs/1206.6910

Golyandina, N., Zhigljavsky, A. (2013): *Singular Spectrum Analysis for time series*. Springer Briefs in Statistics. Springer.

Shlemov, A. and Golyandina, N. (2014): *Shaped extensions of singular spectrum analysis*. 21st International Symposium on Mathematical Theory of Networks and Systems, July 7-11, 2014. Groningen, The Netherlands. p.1813-1820. https://arxiv.org/abs/1507.05286

Golyandina, N., Korobeynikov, A., Shlemov, A. and Usevich, K. (2015): *Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package*. Journal of Statistical Software, Vol. 67, Issue 2. doi:10.18637/jss.v067.i02

Golyandina, N. and Shlemov, A. (2017): *Semi-nonparametric singular spectrum analysis with projection*. Statistics and its Interface, Vol. 10, Issue 1, p. 47-57. https://arxiv.org/abs/1401.4980

Zvonarev, N. and Golyandina, N. (2017): *Iterative algorithms for weighted and unweighted finite-rank time-series approximations*. Statistics and its Interface, Vol. 10, Issue 1, p. 5-18. https://arxiv.org/abs/1507.02751

See Also

svd, ssa-object, ssa-input, ssa.capabilities, decompose, reconstruct, plot, forecast,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Show the summary
summary(s)
# Reconstruct the series, with suitable grouping
r <- reconstruct(s, groups = list(c(1, 4), c(2, 3), c(5, 6)))

plot(r)


# Decompose 'EuStockMarkets' series with default parameters
s <- ssa(EuStockMarkets, kind = "mssa")
r <- reconstruct(s, groups = list(Trend = 1:2))
# Plot original series, trend and residuals superimposed

plot(r, plot.method = "xyplot", superpose = TRUE,
     auto.key = list(columns = 3),
     col = c("blue", "green", "red", "violet"),
     lty = c(rep(1, 4), rep(2, 4), rep(3, 4)))


# Artificial image for 2dSSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with circular window
s <- ssa(mx, kind = "2d-ssa", wmask = circle(5), neig = 10)
# Reconstruct
r <- reconstruct(s, groups = list(1, 2:5))

# Plot components, original image and residuals
plot(r)


# Real example: Mars photo
data(Mars)
# Decompose only Mars image (without backgroud)
s <- ssa(Mars, mask = Mars != 0, wmask = circle(50), kind = "2d-ssa")
```

```
# Plot eigenarrays
plot(s, type = "vectors", idx = 1:25)
# Plot factor arrays
plot(s, type = "vectors", vectors = "factor", idx = 1:25)
# Reconstruct and plot trend
plot(reconstruct(s, 1), fill.uncovered = "original")
# Reconstruct and plot texture pattern
plot(reconstruct(s, groups = list(c(13,14, 17, 18))))

# I()-wrapper demo
circle <- 50
s <- ssa(Mars, wmask = circle(R = I(circle)))

# CSSA-based trend extraction
s <- ssa(EuStockMarkets[, 1] + 1.0i*EuStockMarkets[, 2], kind = "cssa")
r <- reconstruct(s, groups = list(Trend = 1:2))
plot(r)

# `co2' decomposition with double projection to linear functions
s <- ssa(co2, column.projector = "centering", row.projector = "centering")
plot(reconstruct(s, groups = list(trend = seq_len(nspecial(s)))))


# Artificial 2d example with double projection
ii <- matrix(1:100, 100, 100); jj <- t(ii)
x <- ii + 2 * jj
s <- ssa(x, column.projector = "centering", row.projector = "centering")
plot(s)
plot(reconstruct(s, groups = list(trend = seq_len(nspecial(s)))))

# 3D-SSA example (2D-MSSA)
data(Barbara)
Barbara.noised <- Barbara

# Corrupt image by regular noise
noise <- outer(seq_len(dim(Barbara)[1]),
               seq_len(dim(Barbara)[2]),
               function(i, j) sin(2*pi * (i/13 + j/23)))
Barbara.noised[,, 1] <- Barbara.noised[,, 1] + 10 * noise
Barbara.noised[,, 2] <- Barbara.noised[,, 2] + 30 * noise
Barbara.noised[,, 3] <- Barbara.noised[,, 3] + 5 * noise

# Normalize image for plotting
Barbara.noised <- (Barbara.noised - min(Barbara.noised)) / diff(range(Barbara.noised))

plot(c(0, 1), c(0, 1), type = "n", xlab = "", ylab = "")
rasterImage(Barbara.noised, 0, 0, 1, 1, interpolate = FALSE)


# Multichannel 2D-SSA
ss <- ssa(Barbara.noised, L = c(50, 50, 1))
plot(ss, type = "series", groups = 1:18, slice = list(k = 1))
plot(ss, type = "vectors", idx = 1:12, slice = list(k = 1))
```

```
plot(ss, type = "vectors", vectors = "factor", idx = 1:12, slice = list(k = 3))
# Denoise image
Barbara.rec <- residuals(ss, groups = 5:6)
plot(c(0, 1), c(0, 1), type = "n", xlab = "", ylab = "")
rasterImage(Barbara.rec, 0, 0, 1, 1, interpolate = FALSE)
```

---

ssa-input *Input Data Formats Used by SSA Routines*

Description

The inputs of SSA can be quite different depending on the kind of SSA used. However, there is a common of all the variants of SSA and all the routines. The package tries hard to preserve the specifics of input object as much as possible. This means, that all the attributes, etc. are copied back to the reconstructed objects. This way, the result of the SSA decomposition of a 'ts' object is a 'ts' object as well.

For forecasting, it is not possible in general to preserve the attributes of the input objects. However, `Rssa` knows about some common time series classes (e.g. 'ts') and tries to infer the time scales for forecasted objects as well.

The input formats are as follows:

1d SSA and Toeplitz SSA: Input is assumed to be a simple vector, or vector-like object (e.g. univariare 'ts' or 'zooreg' object). Everything else is coerced to vector.

2d SSA: Input assumed to be a matrix. If there are any `NA`'s then the shaped variant of 2d SSA will be used. All non-`NA` elements will be used as a mask.

nd SSA: Input assumed to be an array of arbitrary dimension. If there are any `NA`'s then the shaped variant will be used.

MSSA: While the representation of a one dimensional time series in R is pretty obvious, there are multiple possible ways of defining the multivariate time series. Let us outline some common choices.

- Matrix with separate series in the columns. Optionally, additional time structure like in 'mts' objects, can be embedded.
- Matrix-like (e.g. a 'data.frame') object with series in the columns. In particular, 'data.frame' would be a result of reading the series from the file via 'read.table' function.
- List of separate time series objects (e.g. a 'list' of 'ts' or 'zoo' objects).

Also, the time scales of the individual time series can be normalized via head or tail padding with `NA` (for example, as a result of the `ts.union` call), or specified via time series attributes. Or, everything can be mixed all together.

The `ssa` routine with 'kind = mssa' allows one to provide any of the outlined multivariate series formats. As usual, all the attributes, names of the series, NA padding, etc. is carefully preserved.

CSSA: Complex vectors are assumed at the input.

See Also

ssa

Examples

```
s <- ssa(co2) # Perform the decomposition using the default window length
r <- reconstruct(s, groups = list(Trend = c(1, 4),
                                  Seasonality = c(2, 3))) # Reconstruct into 2 series
class(r$Trend) # Result is 'ts' object

# Simultaneous trend extraction using MSSA
s <- ssa(EuStockMarkets, kind = "mssa")
r <- reconstruct(s, groups = list(Trend = c(1,2)))
class(r$Trend) # Result is 'mts' object

# Trend forecast
f <- rforecast(s, groups = list(Trend = c(1, 2)), len = 50, only.new = FALSE)
class(f) # For 'ts' objects the time scales are inferred automatically

# Artificial image for 2dSSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with circular window
s <- ssa(mx, kind = "2d-ssa", wmask = circle(5), neig = 10)
# Reconstruct
r <- reconstruct(s, groups = list(1, 2:5))
# Plot components, original image and residuals
plot(r)

# 3D-SSA example (2D-MSSA)
data(Barbara)

ss <- ssa(Barbara, L = c(50, 50, 1))
plot(ss)
```

---

ssa-object *Properties of SSA object*

Description

Functions to access various fields of SSA object, query for number of singular values, eigenvectors, factor vectors and 'special' decomposition triples (now, ProjectionSSA triples) in the SSA object and other miscellaneous info. See Chapter 1 in Golyandina et al (2018) for explanation.

Usage

```
nsigma(x)
nu(x)
nv(x)
## S3 method for class 'ssa'
nspecial(x)
## S3 method for class 'ssa'
summary(object, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'ssa'
x$name
```

Arguments

x	SSA object to query
object	an object for which a summary is desired
digits	integer, used for number formatting
...	additional arguments affecting the summary produced
name	field of SSA object to extract. See 'Details' for list of the fields

Details

The internals of SSA object is inheritely opaque, because depending on the selected SVD method and other conditions it might contains different fields.

However, it is possible to extract some fields out of it using the operator $, in particular, the following values of argument name are supported:

sigma Vector of singular values

U The matrix of eigenvectors

V The matrix of factor vectors. Might not exist depending on the selected SVD method

If SSA with projections is being performed, then the eigentriples are ordered as follows: at first, row projection triples, then column projection triples and then SVD-triples. Non-SVD triples (like projection triples) are called 'special triples'. The number of special triples can be obtained by calling nspecial method. Also, one can use the following fields of the SSA object:

nPR the number of row projection triples, may be NULL

nPL the number of column projection triples, may be NULL

Value

an 'integer' of length 1 for nu, nv, nsigma, nspecial routines, matrix or vector for $ operator.

About decompositions

The result of Decomposition step of SSA and its modifications can be written down in the following form:

$$(*) \quad \mathbf{X} = \sum_i \mathbf{X}_i, \qquad \mathbf{X}_i = \sigma_i U_i V_i^{\mathrm{T}},$$

where $\mathbf{X}$ is the trajectory matrix, $U_i \in R^L$, $V_i \in R^K$, σ_i are non-negative numbers. Also, we assume that $\|U_i\| = 1$, $\|V_i\| = 1$.

The Singular Value Decomposition is a particular case of $(*)$ and corresponds to orthonormal systems of $\{U_i\}$ and $\{V_i\}$. We call (σ_i, U_i, V_i) eigentriple, σ_i are singular values, U_i are left singular values or eigenvectors, V_i are right singular vectors or factor vectors, by analogy with the SVD.

For the most of SSA decompositions, U_i belongs to the column space of $\mathbf{X}$, while V_i belongs to the row space of $\mathbf{X}$. Therefore, let us consider such decompositions called *consistent*.

Note that $(*)$ is a decomposition of $\mathbf{X}$ into a sum of rank-one matrices. If the systems $\{U_i\}$ and $\{V_i\}$ are linearly-independent, then the decomposition $(*)$ is minimal (has minimal possible number of addends).

If at least one of the systems is not linear independent, the decomposition $(*)$ is not minimal. If both $\{U_i\}$ and $\{V_i\}$ are orthonormal, then the decomposition $(*)$ is called bi-orthogonal. If $\{U_i\}$ is orthonormal, the decomposition is called left-orthogonal; If $\{V_i\}$ is orthonormal, the decomposition is called right-orthogonal.

Let r be rank of $\mathbf{X}$. Minimal decomposition has exactly r addends. Introduce the Frobenius-inner product as $\langle \mathbf{Z}, \mathbf{Y} \rangle_{\mathrm{F}} = \sum_{i,j} z_{i,j} \cdot y_{i,j}$. Thus, we can say about F-orthogonality and F-orthogonal decompositions if $\mathbf{X}_i$ are F-orthogonal. For F-orthogonality, left or right orthogonality is sufficient.

Generally, $\|\mathbf{X}\|^2$ can be not equal to $\sum_i \|\mathbf{X}_i\|^2$. For F-orthogonal decompositions, $\|\mathbf{X}\|^2 = \sum_i \|\mathbf{X}_i\|^2$.

The contribution of k-th matrix component is defined as $\|\mathbf{X}_k\|^2 / \|\mathbf{X}\|^2 = \sigma_k^2 / (\sum_i \sigma_i^2)$.

For F-orthogonal decompositions, the sum of component contributions is equal to 1. Otherwise, this sum can considerably differ from 1 (e.g., the sum of component contributions can be 90% or 146%).

Remark. If the system $\{U_i\}$ (or $\{V_i\}$) has vectors that do not belong to the column (or row) spaces, then the decomposition can be not minimal even if $\{U_i\}$ (or $\{V_i\}$) are linearly independent, since these projections on the column (or row) space can be dependent.

Decompositions for different SSA modifications

Basic SSA the SVD, consistent, minimal, bi-orthogonal and therefore F-orthogonal decomposition. Implemented in `ssa` with `kind='1d-ssa'`

FOSSA consistent, minimal F-orthogonal decomposition. Implemented in `fossa`

IOSSA consistent, minimal oblique decomposition. Implemented in `iossa`

SSA with projections non-consistent if at least one basis vector used for the projection does not belong to the column (row) trajectory space, F-orthogonal decomposition. The components, which are obtained by projections, are located at the beginning of the decomposition and have numbers $1, \ldots, n_{\mathrm{special}}$. Implemented in `ssa` with `kind='1d-ssa'` and non-`NULL` `row.projector` or `column.projector` arguments

Toeplitz SSA generally, non-consistent, non-minimal F-orthogonal decomposition. Implemented in `ssa` with `kind='toeplitz-ssa'`

Note

For `nsigma`, `nu`, `nv`, `$` routines, the values returned solely depend on used singular value decomposition method and parameters of this method (e.g. 'neig' argument for 'propack' and 'nutrlan' SVD methods).

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

See Also

Rssa for an overview of the package, as well as, ssa, calc.v, iossa, fossa,

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2, neig = 20)
# Show the number of eigentriples saved in the 's'
print(nsigma(s))
# Show the summary
summary(s)
# Show the set of singular values
print(s$sigma)
# Show the first eigenvector
print(s$U[, 1])

# Decompose `co2' series with polynomial projections
s <- ssa(co2, row.projector = 1, column.projector = 2)
print(nspecial(s))
print(c(s$nPL, s$nPR))
# Reconstruct a polynomial trend
plot(reconstruct(s, groups = list(trend = seq_len(nspecial(s)))))
```

---

ssa.capabilities *SSA methods and capabilities check*

Description

Not all SSA algorithms and methods could be applied to SSA objects of any kind (e.g. gapfilling requires shaped SSA object, one cannot forecast for 3D-SSA and so on). This function allows one to determine a set of methods allowed to be applied to a particular SSA object

Usage

```
ssa.capabilities(x)
```

Arguments

x	SSA object holding the decomposition

Value

Logical vector, indicating which methods are allowed

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Since this is 1d SSA object, everything should be supported except
# gapfilling
print(ssa.capabilities(s))
```

---

summarize.gaps *Summarize Gaps in a Series*

Description

Provide a summary about the gaps in a series given desired window length, namely whether the gap is internal or not, whether it is sparce or dense, etc.

Usage

```
## S3 method for class '1d.ssa'
summarize.gaps(x, L = NULL)
## S3 method for class 'toeplitz.ssa'
summarize.gaps(x, L = NULL)
## S3 method for class 'cssa'
summarize.gaps(x, L = NULL)
## Default S3 method:
summarize.gaps(x, L)
```

Arguments

x	SSA object
L	vector of window lengths, if missing or NULL, then all viable window lengths are considered

Value

Object of type 'ssa.gaps': a list with entries which correspond to every window length. For each window length, entry is a list of gaps with their descriptions.

See Also

Rssa for an overview of the package, as well as, gapfill, igapfill, clplot,

Examples

```
# Produce series with gaps
F <- co2; F[c(12, 100:200, 250)] <- NA
# Summarize the gaps
s <- ssa(F, L = 72)
g <- summarize.gaps(s, L = c(36, 72, 144))
# Print the results
print(g)
# Plot the proportion of complete lag-vectors
plot(g)
```

`tmat` *Toeplitz matrices operations.*

Description

A set of routines to operate on Toeplitz matrices stored in compact FFT-based form.

Usage

```
new.tmat(F, L = (N + 1) %/% 2, circular = FALSE, fft.plan = NULL)
is.tmat(t)
tcols(t)
trows(t)
tmatmul(tmat, v, transposed = FALSE)
```

Arguments

`F`	series to construct the Toeplitz version of L x L autocovariance matrix.
`fft.plan`	internal hint argument, should be NULL in most cases
`L`	the window length.
`circular`	logical vector of one element, describes series topology. 'TRUE' means series circularity
`t, tmat`	matrix to operate on.
`transposed`	logical, if 'TRUE' the multiplication is performed with the transposed matrix.
`v`	vector to multiply with.

Details

Fast Fourier Transform provides a very efficient matrix-vector multiplication routine for Toeplitz matrices. See the paper in 'References' for the details of the algorithm.

References

Korobeynikov, A. (2010) *Computation- and space-efficient implementation of SSA*. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

See Also

Rssa for an overview of the package, as well as, ssa,

Examples

```
# Construct the Toeplitz version of the autocovariance matrix for 'co2' series
h <- new.tmat(co2, L = 10)
# Print the number of columns and rows
print(trows(h)); print(tcols(h))
```

USUnemployment *U.S. unemployment figures*

Description

Monthly U.S. male (16-19 years and from 20 years) and female (16-19 years and from 20 years) unemployment figures in thousands from 1948 till 1981.

Usage

```
data(USUnemployment)
```

Format

A multivariate time series with 408 observations on 4 variables. The object is of class 'mts'.

Source

Andrews D. F. and Herzberg H. M. (1985): *Data: A Collection of Problems from Many Fields for the Student and Research Worker*, Springer Series in Statistics.

---

vforecast *Perform vector SSA forecasting of the series*

Description

Perform vector SSA forecasting of the series.

Usage

```
## S3 method for class '1d.ssa'
vforecast(x, groups, len = 1, only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
## S3 method for class 'toeplitz.ssa'
vforecast(x, groups, len = 1, only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
## S3 method for class 'toeplitz.ssa'
vforecast(x, groups, len = 1, only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
## S3 method for class 'mssa'
vforecast(x, groups, len = 1,
          direction = c("row", "column"),
          only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
## S3 method for class 'cssa'
vforecast(x, groups, len = 1, only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
## S3 method for class 'pssa.1d.ssa'
vforecast(x, groups, len = 1, only.new = TRUE, ...,
          drop = TRUE, drop.attributes = FALSE)
```

Arguments

x	SSA object holding the decomposition
groups	list, the grouping of eigentriples to be used in the forecast
len	integer, the desired length of the forecasted series
direction	direction of forecast in multichannel SSA case, "column" stands for so-called L-forecast and "row" stands for K-forecast
only.new	logical, if 'TRUE' then only forecasted values are returned, whole series otherwise
...	additional arguments passed to decompose routines
drop	logical, if 'TRUE' then the result is coerced to series itself, when possible (length of 'groups' is one)
drop.attributes	logical, if 'TRUE' then the attributes of the input series are not copied to the reconstructed ones.

Details

The routines applies the vector SSA forecasting algorithm to produce the new series which is expected to 'continue' the current series on the basis of a given decomposition. Vector forecast differs from recurrent forecast in such a way that it continues the set of vectors in the subspace spanning the chosen eigenvectors (the same formula as described in lrr is used for constructing of the last components of the new vectors) and then derives the series out of this extended set of vectors.

For multichannel SSA, forecast can be constructed in two versions, row and column ones; it uses the formulae from Golyandina et al (2015).

For details of 1D-SSA recurrent forecasting, see Section 3.2.1.3 and Algorithm 3.6 in Golyandina et al (2018). For details of MSSA recurrent forecasting, see Section 4.3.1.3 and Algorithm 4.5 (column forecasting).

Value

List of forecasted objects. Elements of the list have the same names as elements of `groups`. If group is unnamed, corresponding component gets name 'Fn', where 'n' is its index in `groups` list.

Or, the forecasted object itself, if length of groups is one and 'drop = TRUE'.

References

Golyandina N., Korobeynikov A., Zhigljavsky A. (2018): *Singular Spectrum Analysis with R*. Use R!. Springer, Berlin, Heidelberg.

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

Golyandina, N. and Stepanov, D. (2005): *SSA-based approaches to analysis and forecast of multi-dimensional time series*. In Proceedings of the 5th St.Petersburg Workshop on Simulation, June 26-July 2, 2005, St. Petersburg State University, St. Petersburg, 293–298. https://www.gistatgroup.com/gus/mssa2.pdf

Golyandina, N., Korobeynikov, A., Shlemov, A. and Usevich, K. (2015): *Multivariate and 2D Extensions of Singular Spectrum Analysis with the Rssa Package*. Journal of Statistical Software, Vol. 67, Issue 2. doi:10.18637/jss.v067.i02

See Also

`Rssa` for an overview of the package, as well as, `rforecast`, `bforecast`, `forecast`.

Examples

```
# Decompose 'co2' series with default parameters
s <- ssa(co2)
# Produce 24 forecasted values of the series using different sets of eigentriples
# as a base space for the forecast.
vfor <- vforecast(s, groups = list(c(1,4), 1:4), len = 24, only.new=FALSE)
matplot(data.frame(c(co2, rep(NA, 24)), vfor), type="l")

# Forecast `co2' trend by SSA with projections
s <- ssa(co2, column.projector = 2, row.projector = 2)
len <- 100
vfor <- vforecast(s, groups = list(trend = seq_len(nspecial(s))), len = len, only.new = FALSE)
matplot(data.frame(c(co2, rep(NA, len)), vfor), type = "l")

# Forecast finite rank series with polynomial component by SSA with projections
v <- 5000 * sin(2*pi / 13 * (1:100)) +  (1:100)^2 + 10000
s <- ssa(v, row.projector = 2, column.projector = 2)
plot(vforecast(s, groups = list(all = 1:6), len = 100, only.new = FALSE), type = "l")
```

wcor *Calculate the W-correlation matrix*

Description

Function calculates the W-correlation matrix for the series.

Usage

```
## S3 method for class 'ssa'
wcor(x, groups, Fs, ..., cache = TRUE)
## S3 method for class 'ossa'
wcor(x, groups, Fs, ..., cache = TRUE)
## Default S3 method:
wcor(x, L = (N + 1) %/% 2, ..., weights = NULL)
## S3 method for class 'wcor.matrix'
plot(x,
     grid = c(),
     ...,
     col = grey(c(1, 0)),
     cuts = 20,
     zlim = range(abs(x), 0, 1),
     at)
```

Arguments

x	the input object. This might be ssa object for *ssa* method, or just a matrix with elementary series in columns for *default* implementation.
L	window length.
weights	additional weights
groups	list of numeric vectors, indices of elementary components used for reconstruction.
Fs	list of series (e.g. 'ssa.reconstruction' object) for W-cor computation. If missing, reconstructed series from the input 'ssa' object x will be used.
...	further arguments passed to reconstruct routine for wcor or to plot for plot.wcor.matrix
cache	logical, if 'TRUE' then intermediate results will be cached in 'ssa' object.
grid	numeric vector, indices of matrix blocks (groups) which will be separated by grid line. Lines will be drawn on the left of and under noted blocks. Also this argument can be list of two numeric vectors with names 'x' and 'y', for control vertical and horizontal grid lines separately.
col	color vector for colorscale, given by two or more colors, the first color corresponds to the minimal value, while the last one corresponds to the maximal value (will be interpolated by colorRamp)
cuts	integer, the number of levels the range of W-cor values will be divided into.

zlim	range of displayed W-cor values.
at	A numeric vector giving breakpoints along the range of the image. if missing, will be evaluated automatically (see description of the cuts argument).

Details

W-correlation matrix is a standard way of checking for weak separability between the elementary components. In particular, the strongly correlated elementary components should be placed into the same group. The function calculates such a matrix either directly from 'ssa' object or from the matrix of elementary series.

For plotting additional (non-standard) graphical parameters which can be passed via ...:

useRaster logical, indicates whether raster plot should be used. 'FALSE' by default

For class 'ossa', checking of Frobenius orthogonality is performed. If there are reconstructed matrices, which are not F-orthogonal (it is a usual case for Oblique SSA), the warning about possible irrelevancy will be shown, since then weighted correlations do not indicate weak separability properly. In such a case, the use of owcor is preferred.

Value

Object of type 'wcor.matrix'.

References

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

See Also

reconstruct owcor.

Examples

```
# Decompose co2 series with default parameters
s <- ssa(co2)
# Calculate the w-correlation matrix between first 20 series
# for a guess for grouping
w <- wcor(s, groups = 1:20)
plot(w, grid = c(2,4, 5,7))
# Calculate the w-correlation matrix for the chosen groups
# to check separability
w <- wcor(s, groups = list(c(1,4), c(2,3), c(5,6)))


# Artificial image for 2D SSA
mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Decompose 'mx' with default parameters
s <- ssa(mx, kind = "2d-ssa")
```

```
# Plot wcor for first 12 components
plot(wcor(s, groups = 1:12), grid = c(2, 6))

# Real example: Mars photo
data(Mars)
# Decompose only Mars image (without backgroud)
s <- ssa(Mars, mask = Mars != 0, wmask = circle(50), kind = "2d-ssa")
# Plot wcor for the first 25 components
plot(wcor(s, groups = 1:25), grid = c(13, 15, 17,19))
```

`wnorm` *Calculate Weighted Norm of series*

Description

Function calculates the W-norm for input objects or for objects stored in input ssa obect.

Usage

```
## S3 method for class '1d.ssa'
wnorm(x, ...)
## S3 method for class 'nd.ssa'
wnorm(x, ...)
## S3 method for class 'toeplitz.ssa'
wnorm(x, ...)
## S3 method for class 'mssa'
wnorm(x, ...)
## Default S3 method:
wnorm(x, L = (N + 1) %/% 2, ...)
## S3 method for class 'complex'
wnorm(x, L = (N + 1) %/% 2, ...)
```

Arguments

`x`	the input object. This might be ssa object for *ssa* method, or just a series.
`L`	window length.
`...`	arguments to be passed to methods.

Details

`L`-weighted norm of series is Frobenius norm of its `L`-trajectory matrix. So, if `x` is vector (series), the result of `wnorm(x, L)` is equal to `sqrt(sum(hankel(x, L)^2)`, but in fact is calculated much more efficiently. For 1d SSA and Toeplitz SSA `wnorm(x)` calculates weighted norm for stored original input series and stored window length.

`L`-weighted norm of 2d array is Frobenius norm of its `L[1] * L[2]`-trajectory hankel-block-hankel matrix. For 2d SSA this method calculates weighted norm for stored original input array and stored 2d-window lengths.

References

Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2001): *Analysis of Time Series Structure: SSA and related techniques.* Chapman and Hall/CRC. ISBN 1584881941

See Also

ssa-input, hankel, wcor

Examples

```
wnorm(co2, 20)
# Construct ssa-object for 'co2' with default parameters but don't decompose
ss <- ssa(co2, force.decompose = FALSE)
wnorm(ss)

# Artificial image for 2D SSA

mx <- outer(1:50, 1:50,
            function(i, j) sin(2*pi * i/17) * cos(2*pi * j/7) + exp(i/25 - j/20)) +
      rnorm(50^2, sd = 0.1)
# Construct ssa-object for 'mx' with default parameters but don't decompose
s <- ssa(mx, kind = "2d-ssa", force.decompose = FALSE)
wnorm(s)
```

Index