Package 'spectral.methods'

June 2, 2015

Title Singular Spectrum Analysis (SSA) Tools for Time Series Analysis

Version 0.7.2.133

Date 2015-05-20

Author Jannis v. Buttlar

Maintainer Jannis v. Buttlar `<jbuttlar@bgc-jena.mpg.de>`

Imports Rssa (>= 0.11), raster, nnet, abind, RNetCDF, ncdf.tools, foreach, JBTools, DistributionUtils, RColorBrewer

Description Contains some implementations of Singular Spectrum Analysis (SSA) for the gapfilling and spectral decomposition of time series. It contains the code used by Buttlar et. al. (2014), Nonlinear Processes in Geophysics. In addition, the iterative SSA gapfilling method of Kondrashov and Ghil (2006) is implemented. All SSA calculations are done via the truncated and fast SSA algorithm of Korobeynikov (2010) (package 'Rssa').

License GPL-2

LazyLoad yes

Depends R (>= 3.0.0)

Repository CRAN

Repository/R-Forge/Project jbtools

Repository/R-Forge/Revision 32

Repository/R-Forge/DateTimeStamp 2015-06-01 10:20:01

Date/Publication 2015-06-02 07:51:26

NeedsCompilation no

R topics documented:

`spectral.methods-package` *Singular Spectrum Analysis (SSA) Tools for Time Series Analysis*

Description

Contains some implementations of Singular Spectrum Analysis (SSA) for the gapfilling and spectral decomposition of time series. It contains the code used by Buttlar et. al. (2014), Nonlinear Processes in Geophysics. In addition, the iterative SSA gapfilling method of Kondrashov and Ghil (2006) is implemented. All SSA calculations are done via the truncated and fast SSA algorithm of Korobeynikov (2010) (package 'Rssa').

Details

Package:	spectral.methods
Title:	Singular Spectrum Analysis (SSA) Tools for Time Series Analysis
Version:	0.7.2.133
Date:	2015-05-20
Author:	Jannis v. Buttlar
Maintainer:	Jannis v. Buttlar <jbuttlar@bgc-jena.mpg.de>
Imports:	Rssa (>= 0.11), raster, nnet, abind, RNetCDF, ncdf.tools, foreach, JBTools, DistributionUtils, RColorBrewer
License:	GPL-2
LazyLoad:	yes
Depends:	R (>= 3.0.0)

Author(s)

Jannis v. Buttlar

`calcFrequency` *Estimate the frequency of a periodic signal*

Description

Function to estimate the "dominant" frequency of a periodic time series signal.

Usage

```
calcFrequency(series, plot.periodogram = FALSE)
```

Arguments

`series`	numeric vector: input vector (time series)
`plot.periodogram`	logical: whether to plot a periodogram

Details

This function uses Fourier decomposition to determine the 'major' frequency of a time series. Technically this is the frequency of the Fourier component with the highest variance. The function is used by filterTSeriesSSA to determine the frequencies of the individual SSA components.

Value

Frequency of the Fourier component with the highest variance [1/time steps]

Author(s)

Jannis v. Buttlar

See Also

`fft`,`filterTSeriesSSA`

`decomposeNcdf` *Spectrally decompose all time series in a netCDF datacube*

Description

Wrapper function to automatically decompose gridded time series inside a ncdf file and save the results to another ncdf file using SSA.

Usage

```
decomposeNcdf(file.name, borders.wl, calc.parallel = TRUE, center.series = TRUE,
    check.files = TRUE, debugging = FALSE, harmonics = c(), M = c(),
    max.cores = 16, n.comp = c(), pad.series = c(0, 0), print.status = TRUE,
    ratio.const = 0.05, repeat.extr = rep(1, times = length(borders.wl)),
    tresh.const = 1e-12, var.names = "auto", ...)
```

Arguments

`file.name`	character: name of the ncdf file to decompose. The file has to be in the current working directory!
`borders.wl`	list: borders of the different periodicity bands to extract. Units are sampling frequency of the series. In case of monthly data border.wl<- list(c(11, 13)) would extract the annual cycle (period = 12). For details, see the documentation of filterTSeriesSSA.
`calc.parallel`	logical: whether to use parallel computing. Needs package doMC process.
`center.series`	SSA calculation parameter: see the documentation of filterTSeriesSSA!
`check.files`	logical: whether to use checkNcdfFile to check ncdf files for consistency.
`debugging`	logical: if set to TRUE, debugging workspaces or dumpframes are saved at several stages in case of an error.
`harmonics`	SSA calculation parameter: Number of harmonics to be associated with each band. See the documentation of filterTSeriesSSA!
`M`	SSA calculation parameter. Window length for time series embedding (can be different for each element in borders.wl): see the documentation of filterTSeriesSSA.
`max.cores`	integer: maximum number of cores to use.
`n.comp`	SSA calculation parameter: see the documentation of filterTSeriesSSA!
`pad.series`	SSA calculation parameter: see the documentation of filterTSeriesSSA!
`print.status`	logical: whether to print status information during the process
`ratio.const`	numeric: max ratio of the time series that is allowed to be above tresh.const for the time series still to be not considered constant.
`repeat.extr`	SSA calculation parameter: see the documentation of filterTSeriesSSA!
`tresh.const`	numeric: value below which abs(values) are assumed to be constant and excluded from the decomposition
`var.names`	character string: name of the variable to fill. If set to 'auto' (default), the name is taken from the file as the variable with a different name than the dimensions. An error is produced here in cases where more than one such variables exist.
`...`	additional arguments transferred to filterTSeriesSSA.

Details

This is a wrapper function to automatically load, decompose and save a ncdf file using Singular Spectrum Analysis (SSA). It facilitates parallel computing and uses the filterTSeriesSSA() function.

Refer to the documentation of filterTSeriesSSA() for details of the calculations and the necessary parameters, especially for how to perform stepwise filtering.

NCDF file specifications

Due to (possible) limitations in file size the ncdf file can only contain one variable and the one dimensional coordinate variables. The file has to contain one time dimension called 'time'. This function will create a second ncdf file identical to the input file but with an additional dimension called 'spectral.bands' which contains the separated spectral bands. In general the data is internally split into individual time series along ALL dimensions other than time, e.g. a spatiotemporal data cube would be separated into individual time series along its longitude/latitude dimension . The individual series are decomposed and finally combined, transposed and saved in the new file.

The NCDF file may contain NaN values at grid locations where no data is available (e.g. ocean tiles) but individual time series from single "valid" grid points must not contain missing values. In other words, decomposition is only performed for series without missing values, results for non gap-free series will be missing_value the results file.

The function has only been exhaustively tested with ncdf files with two spatial dimensions (e.g. latitude and longitude) and the time dimension. Even though it was programmed to be more flexible, its functionality can not be guaranteed under circumstances with more and/or different dimensions. Input NCDF files should be compatible with the Climate Forcasting (CF) 1.5 ncdf conventions. Several crucial attributes and dimension units are checked and an error is caused if the convention regarding these aspects is not followed. Examples are the attributes scale_factor, add_offset _FillValue and the units for the time dimension

Parallel computing

If calc.parallel == TRUE, single time series are decomposed with parallel computing. This requires the package doMC (and its dependencies) to be installed on the computer. Parallelization with other packages is theoretically possible but not yet implemented. If multiple cores are not available, setting calc.parallel to FALSE will cause the process to be calculated sequential without these dependencies. The package foreach is needed in all cases.

Value

Nothing is returned but a ncdf file with the results is written in the working directory. TODO add mechanism to get constant values in datacube after calculation. TODO Try zero line crossings for frequency determination TODO Make method reproducible (seed etc) TODO Add way to handle non convergence prepare parallel back end save argument values of call check input open ncdf files set default parameters determine call settings for SSA prepare results file prepare parallel iteration parameters determine slices to process create 'iterator' define process during iteration perform calculation add missing value attribute save results add attributes with process information to ncdf files

Author(s)

Jannis v. Buttlar

See Also

`ssa`, `filterTSeriesSSA`, `gapfillNcdf`

Examples

```
## Example for the filtering of monthly data
filename   <- '<filename>.nc'
# Extract yearly cycle, intra annual part and high frequency residual in several steps
borders.wl <- list(a = c(10, 14)
                   , b = c(12, Inf)
                   , c = c(0, 12))
M        <- c(2*12, 4*12, 12)
#extract first four harmonics for yearly cycle
harmonics <- c(4, 0, 0)

# uncomment and run
# decomposeNcdf(file.name = filename, borders.wl = borders.wl, M = M, harmonics = harmonics)

# Extract yearly cycle, intra annual part and high frequency residual in one step
borders.wl <- list(c(0,10,14,Inf))
# use the same M for all bands
M          <- c(2*12)
# uncomment and run
#decomposeNcdf(file.name = filename, borders.wl = borders.wl, M = M)
```

---

filterTSeriesSSA *Decompose a vector (i.e. time series) into spectral bands*

Description

This function decomposes (or filters) a time series into a set of orthogonal (i.e. additive) components with variance on different and distinct timescales (i.e. within different bands). It uses the fast and optimized Singular Spectrum Analysis (SSA) method of Korobeynikov (2010).

Usage

```
filterTSeriesSSA(series, borders.wl, M = rep(floor(length(series)/3),
    times = n.steps), n.comp = rep(40, times = n.steps), harmonics = rep(0,
    times = n.steps), tolerance.harmonics = 0.05, var.thresh.ratio = 0.005,
    grouping = c("grouping.auto", "groupSSANearestNeighbour")[1],
    groupingMethod = "wcor", repeat.extr = rep(1, times = length(borders.wl)),
    recstr.type = "subtraction", pad.series = c(0, 0),
    SSA.methods = c("nutrlan", "propack", "eigen", "svd"),
    center.series = TRUE, call.freq = quote(calcFrequency(series.t)),
    n.steps = switch(class(borders.wl), list = length(borders.wl),
        dim(borders.wl)[2]), plot.spectra = FALSE, second.axis = TRUE,
    open.plot = TRUE, print.stat = TRUE, xlim = c(), debugging = FALSE,
    ...)
```

Arguments

`series`	numeric vector: Input time series (no gaps!)
`borders.wl`	list of numeric vectors: Borders of the different periodicity bands to extract. Units are the sampling frequency of the series (needs one vector per step (see details)).
`M`	integer (vector): Window length or embedding dimension (see details and ?ssa) (in ssa() this parameter is called L).
`n.comp`	integer (vector): Amount of SSA components to compute. See the help of ssa (package Rssa) for details.
`harmonics`	integer (vector): How many harmonics to include in each component (see details). No harmonics are used if set to 0 (default).
`tolerance.harmonics`	numeric fraction (0-1): Tolerance to use to determine the width of the bands the harmonics are looked for in. The actual width is calculated by multiplying the frequency of the "main" oscillation with this ratio. Use higher values for oscillations with few repetitions (and, hence, wider peaks in a spectrum) and lower ones with those with many repetitions (and thus sharper peaks).
`var.thresh.ratio`	numeric fraction(0-1): Variance threshold below which eigentriples are treated as "noise" and will not be included in the groups. The actual threshold is calculated by multiplying the total variance of the time series with this fraction.
`grouping`	character string: Method to use for grouping the individual SSA eigentriples. 'grouping.auto' uses the function of that name in package Rssa, 'groupSSANearestNeighbour' employs a rather crude scheme based on finding pairs (or larger groups) in an euclidian distance matrix of the reconstructions of all extracted SSA eigentriples. See ?grouping.auto or ?groupSSANearestNeighbour for details.
`groupingMethod`	
`repeat.extr`	integer value/vector: How often to repeat the extraction. If the respective value is > 1 than the result of the extraction is again subject to spectral decomposition/filtering for n times and only the (filtered) result is treated as the actual band (see details).
`recstr.type`	string: How to determine the high frequency residuals. If == 'subtraction', the high frequency signal is computed by subtracting all other signals from the original series (see details). For all other values only extracted eigentriples with high frequencies are grouped in this band.
`pad.series`	integer vector (length 2): Length of the part of the series to use for padding at the start (first value) and at the end of the series. Values of zero (default) cause no padding.
`SSA.methods`	character vector: Methods to use for the SSA computation. First the first method is tried, when convergence fails, the second is used and so on. See the help of ssa() in package Rssa for details on the methods. It is preferable to use more than one method as some methods (especially nutrlan) in some cases do not converge. The last two methods are relatively slow.

`center.series`	logical: Whether to center the series around zero prior to the computation. The (subtracted) mean will be added to the long term 'trend' component, e.g. to the step containing an Inf value in borders.wl. Not centering of the series may cause erroneous trend extraction in some cases!
`call.freq`	'quoted' function call : call to a function to compute the frequency of the 'major' oscillation present in some time series. This is used to compute the frequency of the (grouped) SSA eigentriples. See the help for 'calcFrequency' for details of the default mechanism.
`n.steps`	integer: Amount of steps in the process. This argument is only kept for backwards compatibility. Do not supply or change any values!
`plot.spectra`	logical: Whether to plot pseudo spectra for the different steps.
`second.axis`	logical: Whether to plot a second axis with period units
`open.plot`	logical: Whether to open a new plotting window for the plots. Set this to FALSE and open and set up a device prior to running the function to specify or change the device options.
`print.stat`	logical: whether to print status information during the calculations.
`xlim`	numeric vector: x-limits for the plotted spectra. If not supplied it goes from 1 / n....0,5.
`debugging`	logical: if TRUE, workspaces are saved that can be used for debugging non convergence cases that do not cause R errors but may indicate a possible error in the settings, data or code.
`...`	miscellaneous: further arguments passed to the plotting routines.

Details

Purpose The function is based on "singular spectrum analysis" (SSA) (Golyandina et al. (2001)) based on the Rssa package (Korobeynikov (2010), see Golyandina et al. (2013) for a basic introduction).

Definition of the period borders (borders.wl): borders.wl contains the borders of the different periodicity bands to extract. Units are sampling frequency of the series. borders.wl has to be a list with one element per desired decomposition step (see also 'stepwise extraction'). In the case of a single band to be extracted, this vector has to consist of two values (e.g. c(<lower border>,<upper.border>)). In the case of the extraction of several bands per step, the upper border of each band is automatically the lower border of the next band. A value of c(0, 10, 100) would, hence, result in the extraction of two bands, one consisting of eigentriples with periods between 0 and 10 timesteps and one with periods between 10 and 100 timesteps. As a result the vector has to consist of one more value than desired groups. If the vector starts with 0 (and recstr.type = 'subtraction') all high frequency oscillations are combined this group. Use the value INF as the upper band for the long term trend (if desired) to savely include all low frequency parts here.

Window length (M): For optimal detection of spectral components the window length or embedding dimension M should be an integer multiple of the period of the oscillation that is to be extracted. See also ?ssa (here the same parameter is called L)

Padding (pad.series): For padding, the series should start and end exactly at the start and end of a major oscillation (e.g. a yearly cycle). Additionally the length to use for padding should be a integer multiple of this period length. The padding is solved internally by adding the indicated part of the series at the start and at the end of the series. This padded series is only used internally and only the part of the series with original data is returned in the results.

High frequency part (recstr.type): Two different ways to compute the high frequency part (e.g. the part with a frequency between 0 and the lowest border.wl value) are implemented. The standard way (if recstr.type == 'subtraction') is to sum all eigentriples with lower frequencies and subtract this sum from the original series to compute this as the high frequency residual. Otherwise (if (recstr.type != 'subtraction')) only the eigentriples with such high frequencies that were actually extracted are used to build this spectral band.

Stepwise extraction: In general the whole algorithm can be run stepwise. This means that first a certain spectral band is computed (with all possible harmonies etc. ...) and subtracted from the original series. Subsequently this process can be repeated with the residual as often as wanted. This allows for the adaptation of, for example, M, harmonics, ... to the particular oscillation to be extracted. Additionally it often this often leads to a clearer signal separation. To implement several steps, each of M, n.comp and harmonics needs to be a vector of the length of the amount of steps. For each step the corresponding element of this vector is used. borders.wl needs to contain one list entry per step which needs to be a vector containing all borders of the bands to extract in this step (see 'Definition of the period borders').

Repeated extraction (repeat.extr): Especially for the trend it may be advisable to repeat the filtering step for this particular band. In this case the result of the first filtering (e.g. the sum of all eigentriples within this band) is filtered again n times with the same period borders. Finally only the final filtered components are summed and treated as the actual spectral band. In many cases this is helpful to exclude high frequency parts from the trend or low frequency components. It is only possible to repeat the extraction for steps where single bands are extracted.

Visualize results (plot.spectra): In the case that diagnostic plots should be plotted (plot.spectra == TRUE) one (or more) pseudospectra are plotted. Each point in these represents one group of eigentriples determined. For each step a separate plot is produced. Colored regions represent the specified spectral bands. grey lines in the background represent a Fourier Spectrum of the original series.

Value

list with components

`dec.series`	numeric matrix: decomposed timeseries. Each row of this matrix contains one spectrally filtered component. There are as many rows as period borders (borders.wl) were defined.
`borders`	numeric matrix: The lower (first column) and upper (second column) period borders of each filtered component. The rows here correspond to the rows in "dec.series".
`settings`	list: Settings used to perform the calculation.

Author(s)

Jannis v. Buttlar

References

Golyandina, N.; Nekrutkin, V.; Nekrutkin, V. & Zhigljavsky, A. (2001), Analysis of time series structure: SSA and related techniques, CRC Press 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. (2013), 'Basic Singular Spectrum Analysis and forecasting with R', Computational Statistics & Data Analysis.

See Also

ssa,calcFrequency

Examples

```
#create series consisting of two oscillations and noise
series.ex <- sin(2 * pi * 1:1000 / 100) +  0.7 * sin(2 * pi * 1:1000 / 10)  +
  rnorm(n = 1000, sd = 0.4)

#prepare graphics
layout(matrix(c(1, 2, 3, 4, 5, 6, 7, 8), ncol = 2))
par(tcl = 0.2, mgp = c(2, 0, 0), mar = c(0, 4, 0, 0), oma = c(2, 0, 2, 0),
    ps = 10, cex = 1)
plot.new()

#perform decomposition
data.decomposed <- filterTSeriesSSA(series = series.ex,
    borders.wl = list(a = c(8, 12), b = c(80, 120)
        , c = c(0, 10, 100, Inf)),
    M = c(30, 200, 100),
    n.comp = c(10, 20, 20),
    harmonics = c(1, 0, 0),
    plot.spectra = TRUE, open.plot = FALSE)

#plot series and spectral parts
plot(series.ex)
plot(data.decomposed$dec.series[1, ], ylab = '')
plot(data.decomposed$dec.series[2, ], ylab = '')
plot(colSums(data.decomposed$dec.series[-c(1:2), ]), ylab = '')
mtext(side = 2, outer = TRUE, at = -(1 / 8) + ((4:1) * (1 / 4)),
    c('orig.series', '1.step', '2.step', '3.step'), las = 3, cex = 1.5, line = -1)
mtext(side = 3, outer = TRUE, at = -(1 / 4) + ((1:2) * (1 / 2)),
    c('pseudospectra', 'identified components'), las = 1, cex = 1.5, line = 1)
```

---

gapfillNcdf *Fill gaps in time series or spatial fields inside a netCDF file using SSA.*

Description

Wrapper function to automatically fill gaps in series or spatial fields inside a ncdf file and save the results to another ncdf file. This can be done via 1D, 2D or the spatio - temporal gap 3D filling algorithm of Buttlar et. al (2014).

Usage

```
gapfillNcdf(amnt.artgaps = rep(list(rep(list(c(0.05, 0.05)),
    times = length(dimensions[[1]]))), times = length(dimensions)),
   amnt.iters = rep(list(rep(list(c(10, 10)), times = length(dimensions[[1]]))),
        times = length(dimensions)), amnt.iters.start = rep(list(rep(list(c(1,
        1)), times = length(dimensions[[1]]))), times = length(dimensions)),
    calc.parallel = TRUE, debugging = FALSE, debugging.SSA = FALSE,
    dimensions = list(list("time")), file.name, first.guess = "mean",
    force.all.dims = FALSE, gaps.cv = 0, keep.steps = TRUE, M,
    max.cores = 8, max.steps = 10, n.comp = lapply(amnt.iters,
        FUN = function(x) x[[1]][[1]][1] * 2), ocean.mask = c(),
    pad.series = rep(list(rep(list(c(0, 0)), times = length(dimensions[[1]]))),
      times = length(dimensions)), print.status = TRUE, process.cells = c("gappy",
        "all")[1], process.type = c("stepwise", "variances")[1],
    ratio.const = 0.05, ratio.test = 1, reproducible = FALSE,
    size.biggap = rep(list(rep(list(20), times = length(dimensions[[1]]))),
        times = length(dimensions)), tresh.const = 1e-12, tresh.converged = 0,
    tresh.fill = 0.1, var.names = "auto", ...)
```

Arguments

`amnt.artgaps`	list of numeric vectors: The relative ratio (length gaps/series length) of artificial gaps to include in the "innermost" SSA loop (e.g. the value used by the SSA run for each individual series/slice). These ratio is used to determine the iteration with the best prediction (c(ratio big gaps, ratio small gaps)) (see ?gapfillSSA for details)
`amnt.iters`	list of integer vectors: amount of iterations performed for the outer and inner loop (c(outer,inner)) (see ?gapfillSSA for details)
`amnt.iters.start`	list of integer vectors: index of the iteration to start with (outer, inner). If this value is >1, the reconstruction (!) part is started with this iteration. Currently it is only possible to set this to values >1 if amnt.artgaps and size.biggap == 0.
`calc.parallel`	logical: whether to use parallel computing. Needs the packages doMC and foreach to be installed.
`debugging`	logical: if set to TRUE, debugging workspaces or dumpframes are saved at several stages in case of an error.
`debugging.SSA`	
`dimensions`	list of string vectors: setting along which dimensions to perform SSA. These names have to be identical to the names of the dimensions in the ncdf file. Set this to 'time' to do only temporal gap filling or to (for example) c('latitude','longitude')

	to do 2 dimensional spatial gap filling. See the description for details on how to perform spatio-temporal gap filling.
`file.name`	character: name of the ncdf file to decompose. The file has to be in the current working directory!
`first.guess`	character string: if 'mean', standard SSA procedure is followed (using zero as the first guess). Otherwise this is the file name of a ncdf file with the same dimensions (with identical size!) as the file to fill which contains values used as a first guess (for the first step of the process!). This name needs to be exactly "<filename>_first_guess_step_1.nc".
`force.all.dims`	logical: if this is set to true, results from dimensions not chosen as the best guess are used to fill gaps that could not be filled by the best guess dimensions due to too gappy slices etc. .
`gaps.cv`	numeric: ratio (between 0 and 1) of artificial gaps to be included prior to the first cross validation loop that are used for cross validation.
`keep.steps`	logical: whether to keep the files with the results from the single steps
`M`	list of single integers: window length or embedding dimension in time steps. If not given, a default value of 0.5*length(time series) is computed.
`max.cores`	integer: maximum number of cores to use (if calc.parallel = TRUE).
`max.steps`	integer: maximum amount of steps in the variances scheme
`n.comp`	list of single integers: amount of eigentriples to extract (default (if no values are supplied) is 2*amnt.iters[1] (see ?gapfillSSA for details)
`ocean.mask`	logical matrix: contains TRUE for every ocean grid cell and FALSE for land cells. Ocean grid cells will be set to missing after spatial SSA and will be excluded from temporal SSA. The matrix needs to have dimensions identical in order and size to the spatial dimensions in the ncdf file. As an alternative to a R matrix, the name of a ncdf file can be supplied. It should only contain one non coordinate variable with 1 at ocean cells and 0 at land cells.
`pad.series`	list of integer vectors (of size 2): length of the extracts from series to use for padding. Only possible in the one dimensional case. See the documentation of gapfillSSA for details!
`print.status`	logical: whether to print status information during the process
`process.cells`	character string: which grid/series to process. 'gappy' means that only series grids with actual gaps will be processed, 'all' would result in also non gappy grids to be subjected to SSA. The first option results in faster computation times as reconstructions for non gappy grids/series are technically not needed for gap filling, whereas the second option provides a better understanding of the results trajectory to the final results.
`process.type`	
`ratio.const`	numeric: max ratio of the time series that is allowed to be above tresh.const for the time series still to be not considered constant.
`ratio.test`	numeric: ratio (0-1) of the data that should be used in the cross validation step. If set to 1, all data is used.

`reproducible`	logical: Whether a seed based on the characters of the file name should be set which forces all random steps, including the nutrlan SSA algorithm to be exactly reproducible.
`size.biggap`	list of single integers: length of the big artificial gaps (in time steps) (see ?gapfillSSA for details)
`tresh.const`	numeric: value below which abs(values) are assumed to be constant and excluded from the decomposition.
`tresh.converged`	numeric: ratio (0-1): determines the amount of SSA iterations that have to converge so that no error is produced.
`tresh.fill`	numeric fraction (0-1): This value determines the fraction of valid values below which series/grids will not be filled in this step and are filled with the first guess from the previous step (if any). For filling global maps while using a ocean.mask you need to set this value relative to the global grid size (and not only the land mask). Setting this value to zero would mean that also slices/series without any "real" values are tried to be filled with the "first guess" value of the last iteration alone. This can only be done if the cross validation scheme is switched off (e.g. by setting amnt.artgaps and size.biggap to zero.
`var.names`	character string: name of the variable to fill. If set to 'auto' (default), the name is taken from the file as the variable with a different name than the dimensions. An error is produced here in cases where more than one such variable exists.
`...`	further settings to be passed to gapfillSSA

Details

This is a wrapper function to automatically load, gapfill and save a ncdf file using SSA. Basically it automatically runs gapfillSSA (see also its documentation) for all time series or grids in a ncdf file automatically. Theoretically and conceptionally all methods could also be applied to simple datacubes (i.e. R arrays) and not only ncdf files. However, this has not yet been implemented. The values for several function arguments have to be supplied as rather complicated nested lists to facilitate the usage of different values per step (see 'stepwise calculation' for details)

dimensions: It is generally possible to perform one, two, or spatiotemporal 3D SSA (as in Buttlar et al (2014)) for gap filling. This is set by using the argument 'dimensions'. If only one string corresponding to a dimension name in the target ncdf file is supplied, only vectors in the direction of this dimension are extracted and filled. If two dimension names are supplied, matrices (i.e. spatial grids) along these dimension are extracted and 2D SSA is computed. To start the 3D spatio - temporal scheme (Buttlar et.al (2014)) which iterates between 1D temporal and 2D spatial SSA, set dimensions = list(list("time", c("longitude","latitude"))).

stepwise calculation: The algorithm can be run step wise with different settings for each step where the results from each step can be used as 'first guesses' for the subsequent step. To do this, amnt.artgaps, size.biggap, amnt.iters, n.comp, M, tresh.fill and dimensions have to be supplied as lists. For each nth iteration step the values of the corresponding nth list element will be used. At each of these nth iteration steps, several repetitions with different dimensions are possible (as is the case with the 3D spatiotemporal scheme). To facilitate this, the individual list elements at each step have to be lists containing the different dimension names. As a consequence, all these arguments have to be nested lists. This is the case also if only one dimension is used (i.e. to do only temporal 1D SSA, dimensions = list(list('time')). One example for an application where supplying different settings for all these steps would be user defined spatio-temporal gap filling. This allows to clearly define which dimension (and M, gap amount etc) to use at each step of the process. On the contrary, the spatiotemporal gapfilling method applied by Buttlar et. al. (2014) uses identical settings for each (outer loop) iteration step and automatically determines which dimension to use. For this procedure the first list element of dimensions (and the other stepwise arguments) is recycled during each step. Hence, a list of only length one has to be supplied (dimensions = list(list(c('longitude','latitude'),'time'))) (see details for dimensions above).

NCDF file specifications: Due to limitations in the file size, the ncdf file has to contain one variable (and the dimensional coordinate variables) (for the time being). This function will create a second ncdf file identical to the input file but with an additional variable called 'flag.orig', which contains zero for gapfilled values and 1 for not filled values. The function has only been tested with ncdf files with two spatial dimensions (e.g. lat and long) and one time dimension. Even though it was programmed to be more flexible, its functionality can not be guaranteed under circumstances with more dimensions.

non fillable gid cells: Using the 3D method would result in a completely filled datacube. To prevent the filling of grid cells where no reasonable guess via the gapfilling may be achieved (i.e. ocean grid cells in the case of terrestrial data), a matrix indicating these grid cells can be supplied (see 'ocean.mask')

Value

Nothing is returned but a ncdf file with the results is written. TODO remove aperm steps TODO extract iloop convergence information for all loops TODO test inner loop convergence scheme for scenarios TODO indicate fraction of gaps for each time series TODO break down world into blocks TODO integrate onlytime into one dimensional variances scheme TODO facilitate one step filling process with global RMSE calculation TODO save convergence information in ncdf files TODO check for too gappy series at single dimension setting TODO create possibility for non convergence and indicate this in results TODO facilitate run without cross validation repetition TODO test stuff with different dimension orders in the file and in settings TODO substitute all length(processes)==2 tests with something more intuitive TODO put understandable documentation to if clauses TODO remove first guess stuff TODO incorporate non convergence information in final datacube TODO facilitate easy run of different settings (e.g. with different default settings) TODO switch off "force.all.dims" in case of non necessity TODO delete/modify MSSA stuff obsolete MSSA stuff start parallel processing workers insert gaps for cross validation determine different iteration control parameters prepare parallel iteration parameters determine call settings for SSA get first guess run calculation TODO try to stop foreach loop at first error message! test which dimension to be used for the next step TODO whole step can be excluded for "one step" processes determine first guess for next step use first guess from other dimensions in case of too gappy series exclude not to be filled slices (oceans etc) save first guess data TODO: add break criterium to get out of h loop check what happens if gapfillSSA stops further iterations due to limiting groups of eigentriples save process convergence information save results delete first guess files

Author(s)

Jannis v. Buttlar

References

v. Buttlar, J., Zscheischler, J., and Mahecha, M. D. (2014): An extended approach for spatiotemporal gapfilling: dealing with large and systematic gaps in geoscientific datasets, Nonlin. Processes Geophys., 21, 203-215, doi:10.5194/npg-21-203-2014

See Also

`ssa`, `gapfillSSA`, `decomposeNcdf`

Examples

```
    ## prerequisites: go to dir with ncdf file and specify file.name
    file.name        = 'scen_3_0.5_small.nc'
    max.cores        = 8
    calc.parallel    = TRUE

    ##example settings for traditional one dimensional "onlytime" setting and
    ## one step
    amnt.artgaps     <- list(list(c(0.05, 0.05)));
    amnt.iters       <- list(list(c(3, 10)));
    dimensions       <- list(list("time"));
    M                <- list(list(12));
    n.comp           <- list(list(6));
    size.biggap      <- list(list(5));
    var.name         <- "auto"
    process.type     <- "stepwise"
#    .gapfillNcdf(file.name = file.name, dimensions = dimensions, amnt.iters = amnt.iters,
#                 amnt.iters.start = amnt.iters.start, amnt.artgaps = amnt.artgaps,
#                 size.biggap = size.biggap, n.comp = n.comp, tresh.fill = tresh.fill,
#                 M = M, process.type = process.type)



    ##example settings for 3 steps, stepwise and mono dimensional
    dimensions       = list(list('time'), list('time'), list('time'))
    amnt.iters       = list(list(c(1,5)), list(c(2,5)), list(c(3,5)))
    amnt.iters.start = list(list(c(1,1)), list(c(2,1)), list(c(3,1)))
    amnt.artgaps     = list(list(c(0,0)), list(c(0,0)), list(c(0,0)))
    size.biggap      = list(list(0),      list(0),      list(0))
    n.comp           = list(list(6),      list(6),      list(6))
    M                = list(list(12),     list(12),     list(12))
    process.type     = 'stepwise'
#    gapfillNcdf(file.name = file.name, dimensions = dimensions, amnt.iters = amnt.iters,
#                 amnt.iters.start = amnt.iters.start, amnt.artgaps = amnt.artgaps,
#                 size.biggap = size.biggap, n.comp = n.comp, tresh.fill = tresh.fill,
#                 M = M, process.type = process.type)

   ##example settings for 4 steps, stepwise and alternating between temporal and spatial
    dimensions       = list(list('time'), list(c('longitude','latitude')),
      list('time'), list(c('longitude','latitude')))
    amnt.iters       = list(list(c(1,5)), list(c(1,5)), list(c(2,5)), list(c(2,5)))
    amnt.iters.start = list(list(c(1,1)), list(c(1,1)), list(c(2,1)), list(c(2,1)))
    amnt.artgaps     = list(list(c(0,0)), list(c(0,0)), list(c(0,0)), list(c(0,0)))
    size.biggap      = list(list(0),      list(0), list(0),      list(0))
    n.comp           = list(list(15),     list(15), list(15),     list(15))
    M                = list(list(23),     list(c(20,20)), list(23), list(c(20,20)))
    process.type     = 'stepwise'
#    gapfillNcdf(file.name = file.name, dimensions = dimensions,
#                amnt.iters = amnt.iters, amnt.iters.start = amnt.iters.start,
#               amnt.artgaps = amnt.artgaps, size.biggap = size.biggap, n.comp = n.comp,
#             tresh.fill = tresh.fill, M = M, process.type = process.type, max.cores = max.cores)

    ##example setting for process with alternating dimensions but variance criterium
    dimensions       = list(list('time', c('longitude','latitude')))
    n.comp           = list(list(5,      5))
    M                = list(list(10,     c(10, 10)))
    amnt.artgaps     = list(list(c(0,0), c(0,0)))
    size.biggap      = list(list(0,      0))
    process.type     = 'variances'
    max.steps        = 2
#    gapfillNcdf(file.name = file.name, dimensions = dimensions, n.comp = n.comp,
#                tresh.fill = tresh.fill, max.steps = max.steps, M = M,
#                process.type = process.type, amnt.artgaps = amnt.artgaps,
#                size.biggap = size.biggap, max.cores = max.cores)
```

---

gapfillNcdfCoreprocess

helper function for gapfillNcdf

Description

######################## gapfill function for single core ####################

Usage

```
gapfillNcdfCoreprocess(args.call.SSA, datacube, datapts.n, dims.cycle.id,
    dims.cycle.length, dims.process.id, dims.process.length,
    file.name, first.guess, ind.process.cube, iter.gridind, iter.nr,
    iters.n, MSSA, print.status, reproducible)
```

Arguments

args.call.SSA

datacube

datapts.n

dims.cycle.id

dims.cycle.length

dims.process.id

`dims.process.length`

`file.name`

`first.guess`

`ind.process.cube`

`iter.gridind`

`iter.nr`

`iters.n`

`MSSA`

`print.status`

`reproducible`

Details

helper function for gapfillNcdf performs each individual series/grid extracion and handing it over to gapfillSSA.

Author(s)

Jannis v. Buttlar

`gapfillSSA` *Fill gaps in a vector (time-series) with SSA*

Description

gapfillSSA applies the iterative gap filling procedure proposed by Kondrashov and Ghil (2006) in a fast and optimized way developed by Korobeynikov (2010). Generally spoken, major periodic components of the time series are determined and interpolated into gap positions. An iterative cross validation scheme with artificial gaps is used to determine these periodic components.

Usage

```
gapfillSSA(amnt.artgaps = c(0.05, 0.05), DetBestIter = ".getBestIteration",
    debugging = FALSE, amnt.iters = c(10, 10), amnt.iters.start = c(1,
      1), fill.margins = FALSE, first.guess = c(), GroupEigTrpls = "grouping.auto",
    groupingMethod = "wcor", kind = c("auto", "1d-ssa", "2d-ssa")[1],
    M = floor(length(series)/3), matrix.best.iter = "perf.all.gaps",
    MeasPerf = "RMSE", n.comp = 2 * amnt.iters[1], open.plot = TRUE,
    plot.results = FALSE, plot.progress = FALSE, pad.series = c(0,
        0), print.stat = TRUE, remove.infinite = FALSE, scale.recstr = TRUE,
    series, seed = integer(), size.biggap = 20, SSA.methods = c("nutrlan",
        "propack", "eigen", "svd"), tresh.convergence = 0.01,
    tresh.min.length = 5, z.trans.series = TRUE)
```

Arguments

`amnt.artgaps`	numeric vector: The relative ratio (amount gaps/series length) of artificial gaps to include to determine the iteration with the best prediction (c(ratio big gaps, ratio small gaps)). If this is set to c(0,0), the cross validation step is excluded and the iteration is run until amnt.iters.
`DetBestIter`	function: Function to determine the best outer and inner iteration to use for reconstruction. If no function is given, the standard way is used. (see ?.getBestIteration)
`debugging`	logical: If set to TRUE, workspaces to be used for debugging are saved in case of (some) errors or warnings.
`amnt.iters`	integer vector: Amount of iterations performed for the outer and inner loop (c(outer,inner)).
`amnt.iters.start`	integer vector: Index of the iteration to start with c(outer, inner). If this value is > 1, the reconstruction (!) part is started with this iteration. Currently it is only possible to set this to values > 1 if amnt.artgaps != 0 as this would cause a cross validation loop.
`fill.margins`	logical: Whether to fill gaps at the outer margins of the series, i.e. to extrapolate before the first and after the last valid value. Doing this most probably produces unreliable results (i.e. a strong build up of amplitude).
`first.guess`	numeric vector/matrix: First guess for the gap values. The mean/zero is used if no value is supplied. Has to have the same dimensions and lengths as series.
`GroupEigTrpls`	character string: Name of the function used to group the eigentriples. This function needs to take a ssa object as its first input and other inputs as its ... argument. It has to return a list with the length of the desired amount of SSA groups. Each of its elements has to be a integer vector indicating which SSA eigentriple(s) belong(s) to this group. The function 'grouping.auto' uses the methods supplied by the Rssa package (See argument groupingMethod to set the corresponding argument for the method). Another possibility is 'groupSSANearestNeighbour' which uses a rather ad-hoc method of detecting the nearest (Euclidian) neighbour of each eigentriple. 2D SSA automatically uses the nearest neighbor method as grouping was not (yet) implemented for 2D SSA.
`groupingMethod`	
`kind`	character string: Whether to calculate one or two dimensional SSA (see the help of ssa()). Default is to determine this automatically by determining the dimensions of series.
`M`	integer: Window length or embedding dimension [time steps]. If not given, a default value of 0.33*length(timeseries) is computed. For 2d SSA a vector of length 2 has to be supplied. If only one number is given, this is taken for both dimensions. (see ?ssa, here the parameter is called L)
`matrix.best.iter`	character string: Which performance matrix to use (has to be one of recstr.perf.a, recstr.perf.s or recstr.perf.b (see ?.getBestIteration)).
`MeasPerf`	character string: Name of a function to determine the 'goodness of fit' between the reconstruction and the actual values in the artificial gaps. The respective

	function has to take two vectors as an input and return one single value. Set to the "Residual Mean Square Error" (RMSE) by default.
`n.comp`	integer: Amount of eigentriples to extract (default if no values are supplied is 2*amnt.iters[1]) (see ?ssa, here the parameter is called neig).
`open.plot`	logical: Whether to open a new layout of plots for the performance plots.
`plot.results`	logical: Whether to plot performance visualization for artificial gaps?
`plot.progress`	logical: whether to visualize the iterative estimation of the reconstruction process during the calculations.
`pad.series`	integer vector (length 2): Length of the part of the series to use for padding at the start (first value) and at the end of the series. Values of zero cause no padding. This feature has not yet been rigorously tested!
`print.stat`	logical: Whether to print status information during the calculations.
`remove.infinite`	logical: Whether to remove infinite values prior to the calculation.
`scale.recstr`	logical: whether to scale the reconstruction to sd = 1 at the end of each outer loop step.
`series`	numeric vector/matrix: equally spaced input time series or matrix with gaps (gap = NA)
`seed`	integer: Seed to be taken for the randomized determination of the positions of the artificial gaps and the nutrlan ssa algorithm. Per default, no seed is set.
`size.biggap`	integer: Length of the big artificial gaps (in time steps)
`SSA.methods`	character vector: Methods to use for the SSA computation. First the first method is tried, when convergence fails the second is used and so on. See the help of ssa() in package Rssa for details on the methods. The last two methods are relatively slow!
`tresh.convergence`	numeric value: Threshold below which the last three sums of squared differences between inner iteration loops must fall for the whole process to be considered to have converged.
`tresh.min.length`	integer: minimum length the series has to have to do computations.
`z.trans.series`	logical: whether to perform z-transformation of the series prior to the calculation.

Details

Artificial Gaps: The amount of artificial gaps to be included is determined as follows: amnt.artgaps determines the total size of the artificial gaps to be included. The number (0-1) determines the number a relative ratio of the total amount of available datapoints. To switch off the inclusion of either small or biggaps, set respective ratio to 0. In general the ratios determine a maximum amount of gaps. size.biggap sets the size of the biggaps. Subsequently the number of biggaps to be included is determined by calculating the maximum possible amount of gaps of this size to reach the amount of biggaps set by amnt.artgaps[1]. The amount of small gaps is then set according to the ratio of amnt.artgaps[1]/amnt.artgaps[2].

Iteration performance measure: The DetBestIter function should take any of the RMSE matrices (small/big/all gaps) as an input and return i.best with best inner loops for each outer loop and h.best as the outer loop until which should be iterated. Use the default function as a reference.

Visualize results: If plot.per == TRUE an image plot is produced visualizing the RMSE between the artificial gaps and the reconstruction for each iteration. A red dot indicates the iteration chosen for the final reconstruction.

Padding: For padding the series should start and end exactly at the start and end of a major oscillation (e.g. a yearly cycle and the length to use for padding should be a integer multiple of this length. The padding is solved internally by adding the indicated part of the series at the start and at the end of the series. This padded series is only used internally and only the part of the series with original data is returned in the results. Padding is not (yet) possible for two dimensional SSA.

Multidimensional SSA: 1d or 2d SSA is possible. If a vector is given, one dimensional SSA is computed. In case of a matrix as input, two dimensional SSA is performed. For the two dimensional case two embedding should be given (one in the direction of each dimension). If 'big gaps' are set to be used for the cross validation, quadratic blocks of gaps with the size 'size.biggap'*'size.biggap' are inserted.

Value

list with components

`error.occoured`	logical: whether a non caught error occoured in one of the SSA calculations.
`filled.series`	numeric vector/matrix: filled series with the same length as series but without gaps. Gaps at the margins of the series can not be filled and will occur in filled.series (and reconstr).
`i.best`	integer matrix: inner loop iteration for each outer loop step in which the process has finally converged (depending on the threshold determined by tresh.convergence). If the RMSE between two inner loop iterations has been monotonously sinking (and hence, the differences between SSA iterations can be expected to be rather small), this is set to amnt.iters[2]. If not, the process most likely has been building up itself, this is set to 0. In both cases iloop.converged is set FALSE.
`iloop.converged`	logical matrix: Whether each outer loop iteration has converged (see also i.best).
`iter.chosen`	integer vector: iterations finally chosen for the reconstruction.
`perf.all.gaps`	numeric matrix: performance (RMSE) for the filling of all artificial gaps.
`perf.small.gaps`	numeric matrix: performance (RMSE) for the filling of the small artificial gaps.
`perf.big.gaps`	numeric matrix: performance (RMSE) for the filling of the big artificial gaps.
`process.converged`	logical: Whether the whole process has converged. For simplicity reasons, this only detects whether the last outer loop of the final filling process has converged.
`reconstr`	numeric vector/matrix: filtered series or reconstruction finally used to fill gaps.
`recstr.diffsum`	numeric matrix: RMSE between two consecutive inner loop iterations. This value is checked to be below tresh.convergence to determine whether the process has converged.
`settings`	list: settings used to perform the calculation.

Author(s)

Jannis v. Buttlar

References

Kondrashov, D. & Ghil, M. (2006), Spatio-temporal filling of missing points in geophysical data sets, Nonlinear Processes In Geophysics,S 2006, Vol. 13(2), pp. 151-159 Korobeynikov, A. (2010), Computation- and space-efficient implementation of SSA. Statistics and Its Interface, Vol. 3, No. 3, Pp. 257-268

See Also

`ssa`

Examples

```
## create series with gaps
series.ex <- sin(2 * pi * 1:1000 / 100) +  0.7 * sin(2 * pi * 1:1000 / 10) +
  rnorm(n = 1000, sd = 0.4)
series.ex[sample(c(1:1000), 30)] <- NA
series.ex[c(seq(from = sample(c(1:1000), 1), length.out = 20),
            seq(from = sample(c(1:1000), 1), length.out = 20))]<-NA
indices.gaps <- is.na(series.ex)

## prepare graphics
layout(matrix(c(1:5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7), ncol = 5, byrow = TRUE),
       widths = c(1, 1, 1, 0.1, 0.1))
par(mar = c(2, 0, 0, 0.2), oma = c(0, 3, 2, 0.2), tcl = 0.2, mgp = c(0, 0, 100),
    las = 1)

## perform gap filling
data.filled <- gapfillSSA(series = series.ex, plot.results = TRUE, open.plot = FALSE)

## plot series and filled series
plot(series.ex, xlab = '', pch = 16)
plot(data.filled$filled.series, col = indices.gaps+1, xlab = '', pch = 16)
points(data.filled$reconstr, type = 'l', col = 'blue')
mtext(side = 1, 'Index', line = 2)
legend(x = 'topright', merge = TRUE, pch = c(16, 16, NA), lty = c(NA, NA, 1),
       col = c('black', 'red', 'blue'),
       legend = c('original values', 'gap filled values', 'reconstruction'))
```

groupSSANearestNeighbour

Group SSA eigentriples by finding nearest euclidian neighbours

Description

This function finds groups in SSA eigentriples by reconstructing individual eigentriples back to their original (time) domain and by determining the nearest euclidian neighbour for each eigentriple. Groups with more than two members are constructed by identifying groups with a very similar Euclidian distance.

Usage

```
groupSSANearestNeighbour(x, ...)
```

Arguments

`x`	object of class ssa (e.g. the results from a call to ssa)
`...`	other objects that can be passed to the function but which are not used. This is only implemented to make the function identical in its call to the grouping.auto function.

Value

list: list indicating the grouping of the SSA eigentriples

Author(s)

Jannis v. Buttlar

See Also

`ssa`, `grouping.auto`

`plotAdditionalAxis` *Add a second plot axis with transformed label values*

Description

This function adds a second axis additional labels to a plot. It uses the axis values of the opposite side and mathematically transforms these into the values added. This can be used for example to indicate the period and frequency of a periodic signal.

Usage

```
plotAdditionalAxis(side = 1, trans.fun, label = c(), ...)
```

Arguments

`side`	integer: which axis to use as a basis for the second one.
`trans.fun`	function: the transfer function to use between the two axis values.
`label`	character: labels of the axis.
`...`	further arguments to pass to the axis call.

Value

Nothing is returned.

Author(s)

Jannis v. Buttlar

See Also

`axis`

`plotDecomposition` *Plot the results of a SSA decomposition*

Description

This function visualizes the results from a call to filterTSeriesSSA

Usage

```
plotDecomposition(decomp.results, time.vector = c(), n.magnif = 3)
```

Arguments

`decomp.results`	list: object to plot: results from a call to filterTSeriesSSA
`time.vector`	R Date/time object: optional time vector used to label x axes.
`n.magnif`	amount of positions where to plot the different magnifications. This results in the amount of columns in the plot

Value

Nothing is returned. do plots

Author(s)

Jannis v. Buttlar

`plotDecompSeries` *Visualize/plot an overview of a SSA decomposed ncdf file.*

Description

This function plots an visualisation of a SSA decomposed ncdf file.

Usage

```
plotDecompSeries(file.orig, file.decomp = sub("[.]nc", "_specdecomp.nc",
    file.orig), file.plot = "", ...)
```

Arguments

`file.orig`	object to plot: file name or file.con object linking to the original ncdf file
`file.decomp`	object to plot: file name or file.con object linking to the decomposed ncdf file.
`file.plot`	character string: name of the file containing the plots. If not given, a plot window is opened.
`...`	

Value

nothing is returned.

Author(s)

Jannis v. Buttlar

`plotGapfillCube` *Plot an overview of a the results of a SSA gapfilling (from a ncdf file).*

Description

This function plots some overview statistics of the results of a gapfilling (i.e. the results of a call to gapfillNcdf) .

Usage

```
plotGapfillCube(file.orig, file.filled = sub("[.]nc", "_gapfill.nc",
    file.orig), file.prefill = "", data.orig = c(), data.filled = c(),
    data.prefill = c(), n.series = 16, lwd = 2, max.cores = 1,
    ...)
```

Arguments

`file.orig`	object to plot: file name or file.con object linking to the original (unfilled) ncdf file
`file.filled`	object to plot: file name or file.con object linking to the filled ncdf file
`file.prefill`	object to plot: file name or file.con object linking to the prefilled ncdf file
`data.orig`	array: data (see file.orig). Can be supplied to prevent the reloading of huge datacubes.
`data.filled`	see data.orig
`data.prefill`	see data.orig
`n.series`	integer: how many example series to plot
`lwd`	graphical parameter, see ?par
`max.cores`	integer: amount of cores to use for parallelized computations.
`...`	

Value

some overview statistics of the different datacubes.

Author(s)

Jannis v. Buttlar

`plotGapfillSeries` *Plot an overview of a the results of a SSA gapfilling (from a ncdf file)*

Description

This function plots some overview statistics of the results of a gapfilling run in a netCDF file, i.e. the results of a call to gapfillNcdf().

Usage

```
plotGapfillSeries(file.orig, file.filled = sub("[.]nc", "_gapfill.nc",
    file.orig), data.orig = c(), data.filled = c(), ...)
```

Arguments

`file.orig`	object to plot: file name or file.con object linking to a ncdf file
`file.filled`	character string: name of the filled file.
`data.orig`	array: Unfilled data. Can be supplied to prevent loading the data from a ncdf file again. This is read from 'file.filled' if no value is given.
`data.filled`	array: Filled data. Can be supplied to prevent loading the data from a ncdf file again. This is read from 'file.filled' if no value is given.
`...`	

Author(s)

Jannis v. Buttlar

See Also

gapfillSSA, gapfillNcdf

`plotPseudospectrum` *Plot and calculate the pseudospectrum of spectrally decomposed SSA eigentriples*

Description

This function plots the pseudospectrum of the results from a SSA run, e.g. it plots the variance of the individual eigentriples vs. their frequencies. It can also be used to compute the frequency, variance and period of all SSA eigentriples.

Usage

```
plotPseudospectrum(ssa.object, calc.raw.SSA = TRUE, plot.spectrum = TRUE,
    plot.fourier = TRUE, series.orig = c(), pch = 16, col = "red",
    show.harmonies = TRUE, label.points = TRUE, label.tresh = 5e-04,
    call.freq = quote(DetermineFreq(series.t)), print.stat = TRUE,
    ...)
```

Arguments

`ssa.object`	SSA object: the results of a run of ssa().
`calc.raw.SSA`	logical: Whether to additionally compute the whole spectrum for all un-grouped eigentriples (my slow the process in case of long time series.
`plot.spectrum`	logical: whether to plot the pseudospectrum.
`plot.fourier`	logical: Whether to plot the Fourier spectrum in the background.
`series.orig`	numeric vector: original, non decomposed time series (used to calculate Fourier spectrum). If not supplied, an object with the saved in ssa.object is searched for in all active environments.
`pch`	integer: graphic parameter passed to plot() (?par)
`col`	character: graphic parameter passed to plot() (?par)
`show.harmonies`	logical: whether to mark the positions of the harmonies of the oscillation with the highest variance.
`label.points`	logical: whether to label the points with period values
`label.tresh`	numeric: threshold used to label points
`call.freq`	function to use to determine the frequencies of the ssa eigentriples.
`print.stat`	logical: whether to print status information during the calculations.
`...`	

Value

list with values

`paired`	matrix: frequency, period and variance of the paired reconstruction
`raw`	matrix: frequency, period and variance of the unpaired reconstruction

Author(s)

Jannis v. Buttlar

See Also

`ssa`

`rbindMod` *helper function for gapfillNcdf*

Description

############### combine data from foreach iteration #######################

Usage

```
rbindMod(...)
```

Arguments

`...`

Details

helper function for gapfillNcdf that combines the foreach output in a convenient way.

Author(s)

Jannis v. Buttlar

`readNcdfSpectral` *Read the results of a spectral decomposition (from a netCDF file)*

Description

readNcdfSpectral reads spectrally decomposed ncdf data (i.e. the output of a call to decomposeNcdf).

Usage

```
readNcdfSpectral(fileName, varName, rangeBandsGet)
```

Arguments

`fileName`	character string: name of the netCDF file
`varName`	character string: name of the variable to extract.
`rangeBandsGet`	vector: Vector defining the bands to extract. Can be either logical with one TRUE/FALSE per band in the file or a numeric vector of length two with the lower and the upper spectral border.

Value

matrix: the spectral bands defined.

Author(s)

Jannis v. Buttlar

Index