Package: nonlinearTseries (via r-universe)

October 24, 2024

Type Package

Title Nonlinear Time Series Analysis

Version 0.3.1

Date 2024-9-23

Maintainer Constantino A. Garcia <`constantino.garciama@ceu.es`>

Description Functions for nonlinear time series analysis. This package permits the computation of the most-used nonlinear statistics/algorithms including generalized correlation dimension, information dimension, largest Lyapunov exponent, sample entropy and Recurrence Quantification Analysis (RQA), among others. Basic routines for surrogate data testing are also included. Part of this work was based on the book ``Nonlinear time series analysis'' by Holger Kantz and Thomas Schreiber (ISBN: 9780521529020).

License GPL-3

URL `https://github.com/constantino-garcia/nonlinearTseries`

BugReports `https://github.com/constantino-garcia/nonlinearTseries/issues`

Depends R (>= 3.3.0)

Imports Matrix, tseries, zoo, Rcpp (>= 0.12.12), graphics, stats, lifecycle

LinkingTo Rcpp, RcppArmadillo

Suggests rgl, plot3D, knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

Author Constantino A. Garcia [aut, cre], Gunther Sawitzki [ctb]

Repository CRAN

Date/Publication 2024-09-23 11:50:03 UTC

Contents

`buildTakens` *Build the Takens' vectors*

Description

This function builds the Takens' vectors from a given time series. The set of Takens' vectors is the result of embedding the time series in a m-dimensional space. That is, the n^{th} Takens' vector is defined as

$$T[n] = \{time.series[n], time.series[n + timeLag], ...time.series[n + m * timeLag]\}.$$

Taken's theorem states that we can then reconstruct an equivalent dynamical system to the original one (the dynamical system that generated the observed time series) by using the Takens' vectors.

Usage

```
buildTakens(time.series, embedding.dim, time.lag)
```

Arguments

`time.series`	The original time series.
`embedding.dim`	Integer denoting the dimension in which we shall embed the time.series.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.

Value

A matrix containing the Takens' vectors (one per row). The resulting matrix also contains information about the time lag and the embedding dimension used (as attributes).

Author(s)

Constantino A. Garcia and Gunther Sawitzki.

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
## Not run:
# Build the Takens vector for the Henon map using the x-coordinate time series
h = henon(n.sample=  3000,n.transient= 100, a = 1.4, b = 0.3,
start = c(0.73954883, 0.04772637), do.plot = FALSE)
takens = buildTakens(h$x,embedding.dim=2,time.lag=1)
# using the x-coordinate time series we are able to reconstruct
# the state space of the Henon map
plot(takens)
## End(Not run)
```

`cliffordMap` *Clifford map*

Description

Generates a 2-dimensional time series using the Clifford map.

Usage

```
cliffordMap(
  a = -1.4,
  b = 1.6,
  cc = 1,
  d = 0.7,
  start = runif(2),
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`a`	The *a* parameter. Default: -1.4
`b`	The *b* parameter. Default: 1.6
`cc`	The *c* parameter. Default: 1.0
`d`	The *d* parameter. Default: 0.7
`start`	a 2-dimensional vector indicating the starting values for the x and y Clifford coordinates. If the starting point is not specified, it is generated randomly.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated Clifford system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Clifford map is defined as follows:

$$x_{n+1} = sin(a \cdot y_n) + c \cdot cos(a \cdot x_n)$$

$$y_{n+1} = sin(b \cdot x_n) + d \cdot cos(b \cdot y_n)$$

The default selection for the a b c and d parameters is known to produce a deterministic chaotic time series.

Value

A list with two vectors named x and y containing the x-components and the y-components of the Clifford map, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

See Also

`henon`, `logisticMap`, `lorenz`,`rossler`, `ikedaMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
clifford.map=cliffordMap(n.sample = 1000, n.transient=10,do.plot=TRUE)
# accessing the x coordinate and plotting it
plot(ts(clifford.map$x))
## End(Not run)
```

`contourLines` *Obtain the contour lines of the space time plot.*

Description

Obtain the contour lines of the space time plot.

Usage

```
contourLines(x)
```

Arguments

`x`	A *spaceTimePlot* object.

Value

Returns a matrix representing the contour lines of the space time plot.

See Also

`spaceTimePlot`

`corrDim` *Correlation sum, correlation dimension and generalized correlation dimension (order q >1).*

Description

Functions for estimating the correlation sum and the correlation dimension of a dynamical system from 1-dimensional time series using Takens' vectors.

Usage

```
corrDim(
  time.series,
  min.embedding.dim = 2,
  max.embedding.dim = 5,
  time.lag = 1,
  min.radius,
  max.radius,
  corr.order = 2,
  n.points.radius = 5,
  theiler.window = 100,
  do.plot = TRUE,
  number.boxes = NULL,
  ...
)

## S3 method for class 'corrDim'
nlOrder(x)

## S3 method for class 'corrDim'
corrMatrix(x)

## S3 method for class 'corrDim'
radius(x)

## S3 method for class 'corrDim'
embeddingDims(x)

## S3 method for class 'corrDim'
plot(
```

```
  x,
  main = "Correlation Sum C(r)",
  xlab = NULL,
  ylab = "C(r)",
  type = "b",
  log = "xy",
  ylim = NULL,
  col = NULL,
  pch = NULL,
  localScalingExp = T,
  add.legend = T,
  cex.legend = 1,
  ...
)

## S3 method for class 'corrDim'
plotLocalScalingExp(
  x,
  main = "Correlation Dimension C(r)",
  xlab = NULL,
  ylab = "Local scaling exponents",
  type = "b",
  log = "x",
  ylim = NULL,
  col = NULL,
  pch = NULL,
  add.legend = T,
  ...
)

## S3 method for class 'corrDim'
estimate(
  x,
  regression.range = NULL,
  do.plot = FALSE,
  use.embeddings = NULL,
  col = NULL,
  pch = NULL,
  fit.col = NULL,
  fit.lty = 2,
  fit.lwd = 2,
  add.legend = T,
  lty = 1,
  lwd = 1,
  ...
)
```

Arguments

`time.series`	The original time series from which the correlation sum will be estimated.
`min.embedding.dim`	Integer denoting the minimum dimension in which we shall embed the time.series (see buildTakens).
`max.embedding.dim`	Integer denoting the maximum dimension in which we shall embed the time.series (see buildTakens).Thus, we shall estimate the correlation dimension between *min.embedding.dim* and *max.embedding.dim*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors (see buildTakens).
`min.radius`	Minimum distance used to compute the correlation sum C(r).
`max.radius`	Maximum distance used to compute the correlation sum C(r).
`corr.order`	Order of the generalized correlation Dimension q. It must be greater than 1 (corr.order>1). Default, corr.order=2.
`n.points.radius`	The number of different radius where we shall estimate. C(r). Thus, we will estimate C(r) in n.points.radius between min.radius and max.radius.
`theiler.window`	Integer denoting the Theiler window: Two Takens' vectors must be separated by more than theiler.window time steps in order to be considered neighbours. By using a Theiler window, we exclude temporally correlated vectors from our estimations.
`do.plot`	Logical value. If TRUE (default value), a plot of the correlation sum is shown.
`number.boxes`	Number of boxes that will be used in the box assisted algorithm (see neighbourSearch). If the user does not specify it, the function uses a proper number of boxes.
`...`	Additional plotting parameters.
`x`	A *corrDim* object.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`type`	Type of plot (see `?plot`).
`log`	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
`ylim`	Numeric vector of length 2, giving the y coordinates range.
`col`	Vector of colors for each of the dimensions of the plot.
`pch`	Vector of symbols for each of the dimensions of the plot.
`localScalingExp`	add a plot of the local scaling exponents of the correlation sum.
`add.legend`	add a legend to the plot?
`cex.legend`	Magnification value for the legend.

`regression.range`	Vector with 2 components denoting the range where the function will perform linear regression.
`use.embeddings`	A numeric vector specifying which embedding dimensions should the *estimate* function use to compute the correlation dimension.
`fit.col`	A vector of colors to plot the regression lines.
`fit.lty`	The type of line to plot the regression lines.
`fit.lwd`	The width of the line for the regression lines.
`lty`	The line type of the correlation sums.
`lwd`	The line width of the correlation sums.

Details

The correlation dimension is the most common measure of the fractal dimensionality of a geometrical object embedded in a phase space. In order to estimate the correlation dimension, the correlation sum is defined over the points from the phase space:

$$C(r) = \{(number\ of\ points\ (x_i, x_j)\ verifying\ that\ distance\ (x_i, x_j) < r\})/N^2$$

However, this estimator is biased when the pairs in the sum are not statistically independent. For example, Taken's vectors that are close in time, are usually close in the phase space due to the non-zero autocorrelation of the original time series. This is solved by using the so-called Theiler window: two Takens' vectors must be separated by, at least, the time steps specified with this window in order to be considered neighbours. By using a Theiler window, we exclude temporally correlated vectors from our estimations.

The correlation dimension is estimated using the slope obtained by performing a linear regression of $\log 10(C(r))\ Vs.\ \log 10(r)$. Since this dimension is supposed to be an invariant of the system, it should not depend on the dimension of the Taken's vectors used to estimate it. Thus, the user should plot $\log 10(C(r))\ Vs.\ \log 10(r)$ for several embedding dimensions when looking for the correlation dimension and, if for some range $\log 10(C(r))$ shows a similar linear behaviour in different embedding dimensions (i.e. parallel slopes), these slopes are an estimate of the correlation dimension. The *estimate* routine allows the user to get always an estimate of the correlation dimension, but the user must check that there is a linear region in the correlation sum over different dimensions. If such a region does not exist, the estimation should be discarded.

Note that the correlation sum C(r) may be interpreted as: $C(r) =< p(r) >$, that is: the mean probability of finding a neighbour in a ball of radius r surrounding a point in the phase space. Thus, it is possible to define a generalization of the correlation dimension by writing:

$$C_q(r) =< p(r)^{(q-1)} >$$

Note that the correlation sum

$$C(r) = C_2(r)$$

It is possible to determine generalized dimensions Dq using the slope obtained by performing a linear regression of $log10(Cq(r))\ Vs.\ (q-1)log10(r)$. The case q=1 leads to the information dimension, that is treated separately in this package (infDim). The considerations discussed for the correlation dimension estimate are also valid for these generalized dimensions.

Value

A *corrDim* object that consist of a list with four components named *radius*, *embedding.dims*, *order* and *corr.matrix*. *radius* is a vector containing the different radius where we have evaluated C(r). *embedding.dims* is a vector containing all the embedding dimensions in which we have estimated C(r). *order* stores the order of the generalized correlation dimension that has been used. Finally, *corr.matrix* stores all the correlation sums that have been computed. Each row stores the correlation sum for a concrete embedding dimension whereas each colum stores the correlation sum for a specific radius.

The *nlOrder* function returns the order of the correlation sum.

The *corrMatrix* function returns the correlations matrix storing the correlation sums that have been computed for all the embedding dimensions.

The *radius* function returns the radius on which the correlation sum function has been evaluated.

The *embeddingDims* function returns the embedding dimensions on which the correlation sum function has been evaluated.

The *plot* function plots the correlation sum. It is possible to plot the the correlation sum Vs the radius and also the local scaling exponents of the correlation sum Vs radius.

The *plotLocalScalingExp* function plots the local scaling exponents of the correlation sum.

The *estimate* function estimates the correlation dimension of the *corr.dim* object by averaging the slopes of the embedding dimensions specified in the *use.embeddings* parameter. The slopes are determined by performing a linear regression over the radius' range specified in *regression.range*.If *do.plot* is TRUE, a graphic of the regression over the data is shown.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
## Not run:
x=lorenz(sigma=10, rho = 28, beta =8/3, start = c(-10, -11, 47),
time =  seq(0, 70, by = 0.01), do.plot = FALSE)$x
cd=corrDim(time.series=x,min.embedding.dim=3,max.embedding.dim=6,
           time.lag=10,min.radius=1e-3,max.radius=50,
           n.points.radius=100,theiler.window=100,
           number.boxes=100,do.plot=F)

plot(cd,type="l")
plotLocalScalingExp(cd,cex=0.5,xlim=c(1e-1,5))
cd.est = estimate(cd,regression.range=c(0.2,2))
cat("expected: 2.05  --- estimate: ",cd.est,"\n")

## End(Not run)
```

`corrMatrix` *Returns the correlation sums stored in the* corrDim *object*

Description

Returns the correlation sums stored in the *corrDim* object

Usage

```
corrMatrix(x)
```

Arguments

`x`	A *corrDim* object.

Value

The *corrMatrix* function returns the correlations matrix storing the correlation sums that have been computed for all the embedding dimensions.

See Also

`corrDim`

`dfa` *Detrended Fluctuation Analysis*

Description

Functions for performing Detrended Fluctuation Analysis (DFA), a widely used technique for detecting long range correlations in time series. These functions are able to estimate several scaling exponents from the time series being analyzed. These scaling exponents characterize short or long-term fluctuations, depending of the range used for regression (see details).

Usage

```
dfa(
  time.series,
  window.size.range = c(10, 300),
  npoints = 20,
  do.plot = TRUE,
  ...
)

## S3 method for class 'dfa'
windowSizes(x)
```

```
## S3 method for class 'dfa'
fluctuationFunction(x)

## S3 method for class 'dfa'
plot(
  x,
  main = "Detrended Fluctuation Analysis",
  xlab = "Window size: t",
  ylab = "Fluctuation function: F(t)",
  log = "xy",
  ...
)

## S3 method for class 'dfa'
estimate(
  x,
  regression.range = NULL,
  do.plot = FALSE,
  fit.col = 2,
  fit.lty = 1,
  fit.lwd = 1,
  add.legend = TRUE,
  ...
)
```

Arguments

`time.series`	The original time series to be analyzed.
`window.size.range`	Range of values for the windows size that will be used to estimate the fluctuation function. Default: c(10,300).
`npoints`	The number of different window sizes that will be used to estimate the Fluctuation function in each zone.
`do.plot`	logical value. If TRUE (default value), a plot of the Fluctuation function is shown.
`...`	Additional graphical parameters.
`x`	A *dfa* object.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`log`	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
`regression.range`	Vector with 2 components denoting the range where the function will perform linear regression.

`fit.col`	A colors to plot the regression line.
`fit.lty`	The type of line to plot the regression line.
`fit.lwd`	The width of the line for the regression line.
`add.legend`	add a legend with the resulting estmation to the plot?

Details

The Detrended Fluctuation Analysis (DFA) has become a widely used technique for detecting long range correlations in time series. The DFA procedure may be summarized as follows:

1. Integrate the time series to be analyzed. The time series resulting from the integration will be referred to as the profile.
2. Divide the profile into N non-overlapping segments.
3. Calculate the local trend for each of the segments using least-square regression. Compute the total error for each ofi the segments.
4. Compute the average of the total error over all segments and take its root square. By repeating the previous steps for several segment sizes (let's denote it by t), we obtain the so-called Fluctuation function $F(t)$.
5. If the data presents long-range power law correlations: $F(t) \sim t^{\alpha}$ and we may estimate using regression.
6. Usually, when plotting $\log(F(t))\ Vs\ log(t)$ we may distinguish two linear regions. By regressing them separately, we obtain two scaling exponents, α_1 (characterizing short-term fluctuations) and α_2 (characterizing long-term fluctuations).

Steps 1-4 are performed using the *dfa* function. In order to obtain a estimate of some scaling exponent, the user must use the *estimate* function specifying the regression range (window sizes used to detrend the series).

Value

A *dfa* object.

The *windowSizes* function returns the windows sizes used to detrend the time series.

The *fluctuationFunction* function returns the fluctuation function obtained in the DFA represented by the *dfa* object.

Author(s)

Constantino A. Garcia

References

Penzel, Thomas, et al. "Comparison of detrended fluctuation analysis and spectral analysis for heart rate variability in sleep and sleep apnea." Biomedical Engineering, IEEE Transactions on 50.10 (2003): 1143-1151.

Examples

```
## Not run:
 white.noise = rnorm(5000)
dfa.analysis = dfa(time.series = white.noise, npoints = 10,
                 window.size.range=c(10,1000), do.plot=FALSE)
white.estimation = estimate(dfa.analysis,do.plot=TRUE)
cat("Theorical: 0.5---Estimated: ",white.estimation ,"\n")


library(fArma)
fgn = as.numeric(fArma::fgnSim(n = 2000, H = 0.75))
dfa.analysis = dfa(time.series = fgn, npoints = 30,
                   window.size.range=c(10,1000),
                   do.plot=FALSE)
fgn.estimation = estimate(dfa.analysis, do.plot = TRUE,
                          fit.col="blue",fit.lwd=2,fit.lty=2,
                          main="Fitting DFA to fGn")
cat("Theorical: 0.75---Estimated: ",fgn.estimation ,"\n")


fbm = as.numeric(fArma::fbmSim(n = 2000, H = 0.25))
dfa.analysis = dfa(time.series = fbm, npoints = 50,
                   window.size.range=c(10,300),
                   do.plot=FALSE)
fbm.estimation = estimate(dfa.analysis,do.plot = TRUE,
                          add.legend=F, main="DFA of fBm")
cat("Theorical: 1.25 ---Estimated: ",fbm.estimation ,"\n")


## End(Not run)
```

`divergence` *Returns the rate of divergence of close trajectories needed for the maximum Lyapunov exponent estimation.*

Description

Returns the rate of divergence of close trajectories needed for the maximum Lyapunov exponent estimation.

Usage

```
divergence(x)
```

Arguments

`x`	A *maxLyapunov* object.

Value

A numeric matrix representing the time in which the divergence of close trajectories was computed. Each row represents an embedding dimension whereas that each column represents an specific moment of time.

See Also

`maxLyapunov`

`divTime`	*Returns the time in which the divergence of close trajectories was computed in order to estimate the maximum Lyapunov exponent.*

Description

Returns the time in which the divergence of close trajectories was computed in order to estimate the maximum Lyapunov exponent.

Usage

```
divTime(x)
```

Arguments

`x`	A *maxLyapunov* object.

Value

A numeric vector representing the time in which the divergence of close trajectories was computed.

See Also

`maxLyapunov`

`embeddingDims`	*Get the embedding dimensions used for compute a chaotic invariant.*

Description

Get the embedding dimensions used for compute a chaotic invariant.

Usage

```
embeddingDims(x)
```

Arguments

`x`	An object containing all the information needed for the estimate.

Value

A numeric vector with the embedding dimensions used for compute a chaotic invariant.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

`estimate` *Estimate several chaotic invariants using linear regression*

Description

Several chaotic invariants are estimated by using linear regression. This function provides a common interface for the estimate of all these parameters (see `corrDim`, `dfa` and `maxLyapunov` for examples).

Usage

```
estimate(x, regression.range, do.plot, ...)
```

Arguments

`x`	An object containing all the information needed for the estimate.
`regression.range`	Range of values on the x-axis on which the regression is performed.
`do.plot`	Logical value. If TRUE (default value), a plot of the regression is shown.
`...`	Additional parameters.

Value

An estimate of the proper chaotic invariant.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

`estimateEmbeddingDim` *Estimate the embedding dimension*

Description

This function determines the minimum embedding dimension from a scalar time series using the algorithm proposed by L. Cao (see references).

Usage

```
estimateEmbeddingDim(
  time.series,
  number.points = length(time.series),
  time.lag = 1,
  max.embedding.dim = 15,
  threshold = 0.95,
  max.relative.change = 0.1,
  do.plot = TRUE,
  main = "Computing the embedding dimension",
  xlab = "dimension (d)",
  ylab = "E1(d) & E2(d)",
  ylim = NULL,
  xlim = NULL,
  std.noise
)
```

Arguments

`time.series`	The original time series.
`number.points`	Number of points from the time series that will be used to estimate the embedding dimension. By default, all the points in the time series are used.
`time.lag`	Time lag used to build the Takens' vectors needed to estimate the embedding dimension (see buildTakens). Default: 1.
`max.embedding.dim`	Maximum possible embedding dimension for the time series. Default: 15.
`threshold`	Numerical value between 0 and 1. The embedding dimension is estimated using the E1(d) function. E1(d) stops changing when d is greater than or equal to embedding dimension, staying close to 1. This value establishes a threshold for considering that E1(d) is close to 1. Default: 0.95
`max.relative.change`	Maximum relative change in E1(d) with respect to E1(d-1) in order to consider that the E1 function has been stabilized and it will stop changing. Default: 0.01.
`do.plot`	Logical value. If TRUE (default value), a plot of E1(d) and E2(d) is shown.
`main`	Title for the plot.
`xlab`	Title for the x axis.

`ylab`	Title for the y axis.
`ylim`	numeric vectors of length 2, giving the y coordinates range.
`xlim`	numeric vectors of length 2, giving the x coordinates range.
`std.noise`	numeric value that permits to add a small amount of noise to the original series to avoid the appearance of false neighbours due to discretizacion errors. This also prevents the method to fail with periodic signals (in which neighbours at a distance of 0 appear). By default, a small amount of noise is always added. Use 0 not to add noise.

Details

The Cao's algorithm uses 2 functions in order to estimate the embedding dimension from a time series: the E1(d) and the E2(d) functions, where d denotes the dimension.

E1(d) stops changing when d is greater than or equal to the embedding dimension, staying close to 1. On the other hand, E2(d) is used to distinguish deterministic signals from stochastic signals. For deterministic signals, there exist some d such that E2(d)!=1. For stochastic signals, E2(d) is approximately 1 for all the values.

This function uses the Arya and Mount's C++ ANN library for nearest neighbour search (For more information on the ANN library please visit `http://www.cs.umd.edu/~mount/ANN/`). The R wrapper is a modified version of the RANN package code by Samuel E. Kemp and Gregory Jefferis.

If no suitable embedding dimension can be found within the provided range, the function will return NA.

Note

In the current version of the package, the automatic detection of stochastic signals has not been implemented yet.

Author(s)

Constantino A. Garcia

References

Cao, L. Practical method for determining the minimum embedding dimension of a scalar time series. Physica D: Nonlinear Phenomena, 110,1, pp. 43-50 (1997).

Arya S. and Mount D. M. (1993), Approximate nearest neighbor searching, Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA'93), 271-280.

Arya S., Mount D. M., Netanyahu N. S., Silverman R. and Wu A. Y (1998), An optimal algorithm for approximate nearest neighbor searching, Journal of the ACM, 45, 891-923.

Examples

```
## Not run:
h = henon(do.plot=FALSE)
dimension = estimateEmbeddingDim(h$x, time.lag=1, max.embedding.dim=6,
             threshold=0.9, do.plot=TRUE)
```

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

`FFTsurrogate` *Generate surrogate data using the Fourier transform*

Description

Generates surrogate samples from the original time series.

Usage

```
FFTsurrogate(time.series, n.samples = 1)
```

Arguments

`time.series`	The original time.series from which the surrogate data is generated.
`n.samples`	The number of surrogate data sets to generate,

Details

This function uses the phase randomization procedure for generating the surrogated data. This algorithm generates surrogate data with the same mean and autocorrelation function (and thus, the same power spectrum because of the Wiener-Khinchin theorem) as the original time series.

The phase randomization algorithm is often used when the null hypothesis being tested consist on the assumption that the time.series data comes from a stationary linear stochastic process with Gaussian inputs. The phase randomization preserves the Gaussian distribution.

Value

A matrix containing the generated surrogate data (one time series per row).

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
## Not run:
# generate 20 surrogate sets using as original time series
# an arma(1,1) simulation
time.series = arima.sim(list(order = c(1,0,1), ar = 0.6, ma = 0.5), n = 200)
surrogate = FFTsurrogate(time.series, 20)

## End(Not run)
```

`findAllNeighbours` *neighbour search*

Description

This function finds all the neighbours of all the vectors from Takens' vector array. The neighbours are found using a box assisted algorithm that creates a wrapped grid of a given number of boxes per dimension.

Usage

```
findAllNeighbours(takens, radius, number.boxes = NULL)
```

Arguments

`takens`	The matrix containing all the Takens' vectors (see buildTakens).
`radius`	Distance in which the algorithm will search for neighbours.
`number.boxes`	Integer denoting the number of boxes per dimension that will be used to construct a wrapped grid (see Schreiber). If the user does not specify a number of boxes, this function estimates a proper number.

Value

A list in which the n-th position contains another list with all the neighbours of the n-th Takens' vector. If the list is empty, that means that there is no neighbour of the n-th Takens' vector in the given radius.

Author(s)

Constantino A. Garcia

References

Schreiber, T. Efficient neighbor searching in nonlinear time series analysis. Int. J. Bifurcation and Chaos, 5, p. 349, (1995).

See Also

`neighbourSearch`.

Examples

```
## Not run:
# Find all the neighbours Takens' vectors build from the Henon time
# series. The size of the neighbourhood is set to 0.1.
h=henon(start = c(0.63954883, 0.04772637), do.plot = FALSE)
takens = buildTakens(h$x,embedding.dim=2,time.lag=1)
neighbours=findAllNeighbours(takens,0.1)
```

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

`fixedMass` *fixed mass*

Description

fixed mass

Usage

```
fixedMass(x)
```

Arguments

`x`	A *infDim* object.

Value

A numeric vector representing the fixed mass vector used in the information dimension algorithm represented by the *infDim* object.

See Also

`infDim`

`fluctuationFunction` *Returns the fluctuation function obtained in a DFA and represented by a* dfa *object.*

Description

Returns the fluctuation function obtained in a DFA and represented by a *dfa* object.

Usage

```
fluctuationFunction(x)
```

Arguments

`x`	A *dfa* object.

Value

The *fluctuationFunction* function returns the fluctuation function used obtained in the DFA.

See Also

`dfa`

`gaussMap` *Gauss map*

Description

Generates a 1-dimensional time series using the Gauss map

Usage

```
gaussMap(
  a = 4.9,
  b = -0.58,
  start = runif(1, min = -0.5, max = 0.5),
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`a`	The *a* parameter. Default: 4.9
`b`	The *b* parameter. Default: -0.58
`start`	A numeric value indicating the starting value for the time series. If the starting point is not specified, it is generated randomly.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated Gauss system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Gauss map is defined as follows:

$$x_{n+1} = exp(-a \cdot (x_n)^2) + b$$

The default selection for both *a* and *b* parameters is known to produce a deterministic chaotic time series.

Value

A vector containing the values of the time series that has been generated.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Chaos and nonlinear dynamics: an introduction for scientists and engineers, by Robert C. Hilborn, 2nd Ed., Oxford, Univ. Press, New York, 2004.

See Also

`henon`, `logisticMap`, `lorenz`,`rossler`, `ikedaMap`, `cliffordMap`, `sinaiMap`

`getContourLines` *Obtain the contour lines of the space time plot.*

Description

Obtain the contour lines of the space time plot.

Usage

```
getContourLines(x)
```

Arguments

`x`	A *spaceTimePlot* object.

See Also

`spaceTimePlot`

---

henon *Henon map*

Description

Generates a 2-dimensional time series using the Henon map.

Usage

```
henon(
  start = runif(min = -0.5, max = 0.5, n = 2),
  a = 1.4,
  b = 0.3,
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`start`	A 2-dimensional vector indicating the starting values for the x and y Henon coordinates. If the starting point is not specified, it is generated randomly.
`a`	The *a* parameter. Default: 1.4.
`b`	The *b* parameter. Default: 0.3.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated Henon system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Henon map is defined as follows:

$$x_n = 1 - a \cdot x_{n-1}^2 + y_{n-1}$$

$$y_n = b \cdot x_{n-1}$$

The default selection for both *a* and *b* parameters (*a*=1.4 and *b*=0.3) is known to produce a deterministic chaotic time series.

Value

A list with two vectors named *x* and *y* containing the x-components and the y-components of the Henon map, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering (Studies in Nonlinearity)

See Also

`logisticMap`, `lorenz`, `rossler`,`ikedaMap`, `cliffordMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
henon.map=henon(n.sample = 1000, n.transient=10,do.plot=TRUE,
                start=c(-0.006423277,-0.473545134))
# accessing the x coordinate and plotting it
plot(ts(henon.map$x))

## End(Not run)
```

`ikedaMap` *Ikeda map*

Description

Generates a time series using the Ikeda map

Usage

```
ikedaMap(
  a = 0.85,
  b = 0.9,
  cc = 7.7,
  k = 0.4,
  start = runif(2),
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`a`	The *a* parameter. Default: 0.85.
`b`	The *b* parameter. Default: 0.9.
`cc`	The *c* parameter. Default: 7.7.
`k`	The *k* parameter. Default: 0.4.
`start`	a 2-dimensional numeric vector indicating the starting value for the time series. If the starting point is not specified, it is generated randomly.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated ikeda system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Ikeda map is defined as follows:

$$z_{n+1} = a + b \cdot z_n \cdot exp(ik - \frac{ic}{(1 + |z_{n-1}|^2)})$$

The default selection for the a, b, c and k parameters is known to produce a deterministic chaotic time series.

Value

a list with 2 vectors named x and y the x-components and the y-components of the Ikeda map, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering (Studies in Nonlinearity)

See Also

`henon`, `logisticMap`, `lorenz`,`rossler`, `cliffordMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
ikeda.map=ikedaMap(n.sample = 1000, n.transient=10, do.plot=TRUE)

## End(Not run)
```

`infDim` *Information dimension*

Description

Functions for estimating the information dimension of a dynamical system from 1-dimensional time series using Takens' vectors

Usage

```
infDim(
  time.series,
  min.embedding.dim = 2,
  max.embedding.dim = min.embedding.dim,
  time.lag = 1,
  min.fixed.mass,
  max.fixed.mass,
  number.fixed.mass.points = 10,
  radius,
  increasing.radius.factor = sqrt(2),
  number.boxes = NULL,
  number.reference.vectors = 5000,
  theiler.window = 1,
  kMax = 1000,
  do.plot = TRUE,
  ...
)

## S3 method for class 'infDim'
fixedMass(x)

## S3 method for class 'infDim'
logRadius(x)

## S3 method for class 'infDim'
embeddingDims(x)

## S3 method for class 'infDim'
estimate(
  x,
  regression.range = NULL,
```

```
  do.plot = TRUE,
  use.embeddings = NULL,
  col = NULL,
  pch = NULL,
  fit.col = NULL,
  fit.lty = 2,
  fit.lwd = 2,
  add.legend = T,
  lty = 1,
  lwd = 1,
  ...
)

## S3 method for class 'infDim'
plot(
  x,
  main = "Information Dimension",
  xlab = "fixed mass (p)",
  ylab = "<log10(radius)>",
  type = "b",
  log = "x",
  ylim = NULL,
  col = NULL,
  pch = NULL,
  localScalingExp = T,
  add.legend = T,
  ...
)

## S3 method for class 'infDim'
plotLocalScalingExp(
  x,
  main = "Local scaling exponents d1(p)",
  xlab = "fixed mass p",
  ylab = "1/d1(p)",
  type = "b",
  log = "x",
  ylim = NULL,
  col = NULL,
  pch = NULL,
  add.legend = T,
  ...
)
```

Arguments

`time.series`	The original time series from which the information dimension will be estimated.

`min.embedding.dim`	Integer denoting the minimum dimension in which we shall embed the time.series (see buildTakens).
`max.embedding.dim`	Integer denoting the maximum dimension in which we shall embed the time.series (see buildTakens).Thus, we shall estimate the information dimension between *min.embedding.dim* and *max.embedding.dim*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors (see `buildTakens`).
`min.fixed.mass`	Minimum percentage of the total points that the algorithm shall use for the estimation.
`max.fixed.mass`	Maximum percentage of the total points that the algorithm shall use for the estimation.
`number.fixed.mass.points`	The number of different *fixed mass* fractions between *min.fixed.mass* and *max.fixed.mass* that the algorithm will use for estimation.
`radius`	Initial radius for searching neighbour points in the phase space. Ideally, it should be small enough so that the fixed mass contained in this radius is slightly greater than the *min.fixed.mass*. However, whereas the radius is not too large (so that the performance decreases) the choice is not critical.
`increasing.radius.factor`	Numeric value. If no enough neighbours are found within *radius*, the radius is increased by a factor *increasing.radius.factor* until succesful. Default: sqrt(2) = 1.414214.
`number.boxes`	Number of boxes that will be used in the box assisted algorithm (see neighbourSearch).
`number.reference.vectors`	Number of reference points that the routine will try to use, saving computation time.
`theiler.window`	Integer denoting the Theiler window: Two Takens' vectors must be separated by more than theiler.window time steps in order to be considered neighbours. By using a Theiler window, we exclude temporally correlated vectors from our estimations.
`kMax`	Maximum number of neighbours used for achieving p with all the points from the time series (see Details).
`do.plot`	Logical value. If TRUE (default value), a plot of the correlation sum is shown.
`...`	Additional graphical parameters.
`x`	A *infDim* object.
`regression.range`	Vector with 2 components denoting the range where the function will perform linear regression.
`use.embeddings`	A numeric vector specifying which embedding dimensions should be used to compute the information dimension.
`col`	Vector of colors for each of the dimensions of the plot.

`pch`	Vector of symbols for each of the dimensions of the plot.
`fit.col`	A vector of colors to plot the regression lines.
`fit.lty`	The type of line to plot the regression lines.
`fit.lwd`	The width of the line for the regression lines.
`add.legend`	add a legend to the plot?
`lty`	The line type of the <log10(radius)> functions.
`lwd`	The line width of the <log10(radius)> functions.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`type`	Type of plot (see `?plot`).
`log`	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
`ylim`	Numeric vector of length 2, giving the y coordinates range.
`localScalingExp`	add a plot of the local information dimension scaling exponents?

Details

The information dimension is a particular case of the generalized correlation dimension when setting the order q = 1. It is possible to demonstrate that the information dimension D_1 may be defined as: $D_1 = lim_{r \to 0} < \log p(r) > / \log(r)$. Here, $p(r)$ is the probability of finding a neighbour in a neighbourhood of size r and <> is the mean value. Thus, the information dimension specifies how the average Shannon information scales with the radius r. The user should compute the information dimension for different embedding dimensions for checking if D_1 saturates.

In order to estimate D_1, the algorithm looks for the scaling behaviour of the the average radius that contains a given portion (a "fixed-mass") of the total points in the phase space. By performing a linear regression of $\log(p)\ Vs.\ \log(< r >)$ (being p the fixed-mass of the total points), an estimate of D_1 is obtained.

The algorithm also introduces a variation of p for achieving a better performance: for small values of p, all the points in the time series (N) are considered for obtaining $p = n/N$. Above a maximum number of neighbours $kMax$, the algorithm obtains p by decreasing the number of points considerd from the time series $M < N$. Thus $p = kMax/M$.

Even with these improvements, the calculations for the information dimension are heavier than those needed for the correlation dimension.

Value

A *infDim* object that consist of a list with two components: *log.radius* and *fixed.mass*. *log.radius* contains the average log10(radius) in which the *fixed.mass* can be found.

The *fixedMass* function returns the fixed mass vector used in the information dimension algorithm.

The *logRadius* function returns the average log(radius) computed on the information dimension algorithm.

The *embeddingDims* function returns the embeddings in which the information dimension was computed

The 'estimate' function estimates the information dimension of the 'infDim' object by by averaging the slopes of the embedding dimensions specified in the *use.embeddings* parameter. The slopes are determined by performing a linear regression over the fixed mass' range specified in 'regression.range'. If do.plot is TRUE, a graphic of the regression over the data is shown.

The 'plot' function plots the computations performed for the information dimension estimate: a graphic of <log10(radius)> Vs fixed mass. Additionally, the inverse of the local scaling exponents can be plotted.

The *plotLocalScalingExp* function plots the inverse of the local scaling exponentes of the information dimension (for reasons of numerical stability).

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

See Also

`corrDim`.

Examples

```
## Not run:
x=henon(n.sample=  3000,n.transient= 100, a = 1.4, b = 0.3,
        start =  c(0.8253681, 0.6955566), do.plot = FALSE)$x

leps = infDim(x,min.embedding.dim=2,max.embedding.dim = 5,
              time.lag=1, min.fixed.mass=0.04, max.fixed.mass=0.2,
              number.fixed.mass.points=100, radius =0.001,
              increasing.radius.factor = sqrt(2), number.boxes=100,
              number.reference.vectors=100, theiler.window = 10,
              kMax = 100,do.plot=FALSE)

plot(leps,type="l")
colors2=c("#999999", "#E69F00", "#56B4E9", "#009E73",
          "#F0E442", "#0072B2", "#D55E00")
id.estimation = estimate(leps,do.plot=TRUE,use.embeddings = 3:5,
                         fit.lwd=2,fit.col=1,
                         col=colors2)
cat("Henon---> expected: 1.24    predicted: ", id.estimation ,"\n")

## End(Not run)
```

`keenanTest` *Keenan's test*

Description

Keenan's test: test for nonlinearity against the null hypothesis that the time series follows some AR process.

Usage

```
keenanTest(time.series, ...)
```

Arguments

`time.series`	The original time.series.
`...`	Additional arguments for the `ar` function.

Value

A list containing the results of the test, including:

- test.stat: the F-squared test statistic
- df1 and df2: the degrees of freedom of the test statistic.
- p.value.
- order: order of the AR process used for testing.

References

Keenan, D. M. (1985), A Tukey nonadditivity-type test for time series Nonlinearity, Biometrika, 72, 39-44.

See Also

`nonlinearityTest`, `tsayTest`, `mcleodLiTest`,`thresholdTest`

`logisticMap` *Logistic map*

Description

Generates a time series using the logistic map.

Usage

```
logisticMap(
  r = 4,
  start = runif(n = 1, min = 0, max = 1),
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`r`	The *r* parameter. Default: 4
`start`	A numeric value indicating the starting value for the time series. If the starting point is not specified, it is generated randomly.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated logistic system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The logistic map is defined as follows:

$$x_n = r \cdot x_{n-1} \cdot (1 - x_{n-1})$$

The default selection for the *r* parameter is known to produce a deterministic chaotic time series.

Value

A vector containing the values of the time series that has been generated.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering (Studies in Nonlinearity)

See Also

`henon`, `lorenz`, `rossler`, `ikedaMap`,`cliffordMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
log.map=logisticMap(n.sample = 1000, n.transient=10,do.plot=TRUE)

## End(Not run)
```

`logRadius` *Obtain the the average log(radius) computed on the information dimension algorithm.*

Description

Obtain the the average log(radius) computed on the information dimension algorithm.

Usage

```
logRadius(x)
```

Arguments

`x`	A *infDim* object.

Value

A numeric vector representing the average log(radius) computed on the information dimension algorithm represented by the *infDim* object.

See Also

`infDim`

`lorenz` *Lorenz system*

Description

Generates a 3-dimensional time series using the Lorenz equations.

Usage

```
lorenz(
  sigma = 10,
  beta = 8/3,
  rho = 28,
  start = c(-13, -14, 47),
  time = seq(0, 50, by = 0.01),
  do.plot = deprecated()
)
```

Arguments

`sigma`	The σ parameter. Default: 10.
`beta`	The β parameter. Default: 8/3.
`rho`	The ρ parameter. Default: 28.
`start`	A 3-dimensional numeric vector indicating the starting point for the time series. Default: c(-13, -14, 47).
`time`	The temporal interval at which the system will be generated. Default: time=seq(0,50,by = 0.01).
`do.plot`	Logical value. If TRUE, a plot of the generated Lorenz system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Lorenz system is a system of ordinary differential equations defined as:

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = -\beta z + xy$$

The default selection for the system parameters ($\sigma = 10, \rho = 28, \beta = 8/3$) is known to produce a deterministic chaotic time series.

Value

A list with four vectors named *time*, x, y and z containing the time, the x-components, the y-components and the z-components of the Lorenz system, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering (Studies in Nonlinearity)

See Also

`henon`, `logisticMap`, `rossler`,`ikedaMap`, `cliffordMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
lor=lorenz(time=seq(0,30,by = 0.01))
# plotting the x-component
plot(lor$time,lor$x,type="l")

## End(Not run)
```

`maxLyapunov` *Maximum lyapunov exponent*

Description

Functions for estimating the maximal Lyapunov exponent of a dynamical system from 1-dimensional time series using Takens' vectors.

Usage

```
maxLyapunov(
  time.series,
  min.embedding.dim = 2,
  max.embedding.dim = min.embedding.dim,
  time.lag = 1,
  radius,
  theiler.window = 1,
  min.neighs = 5,
  min.ref.points = 500,
  max.time.steps = 10,
  number.boxes = NULL,
  sampling.period = 1,
  do.plot = TRUE,
```

```
  ...
)

## S3 method for class 'maxLyapunov'
divTime(x)

## S3 method for class 'maxLyapunov'
embeddingDims(x)

## S3 method for class 'maxLyapunov'
divergence(x)

## S3 method for class 'maxLyapunov'
plot(
  x,
  main = "Estimating maximal Lyapunov exponent",
  xlab = "time t",
  ylab = "S(t)",
  type = "p",
  col = NULL,
  pch = NULL,
  add.legend = T,
  ...
)

## S3 method for class 'maxLyapunov'
estimate(
  x,
  regression.range = NULL,
  do.plot = FALSE,
  use.embeddings = NULL,
  main = "Estimating maximal Lyapunov exponent",
  xlab = "time t",
  ylab = "S(t)",
  type = "p",
  col = NULL,
  pch = NULL,
  ylim = NULL,
  fit.col = NULL,
  fit.lty = 2,
  fit.lwd = 2,
  add.legend = T,
  ...
)
```

Arguments

`time.series`	The original time series from which the maximal Lyapunov exponent will be estimated.

`min.embedding.dim`	Integer denoting the minimum dimension in which we shall embed the time.series (see buildTakens).
`max.embedding.dim`	Integer denoting the maximum dimension in which we shall embed the time.series (see buildTakens).Thus, we shall estimate the Lyapunov exponent between *min.embedding.dim* and *max.embedding.dim*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors (see buildTakens).
`radius`	Maximum distance in which will look for nearby trajectories.
`theiler.window`	Integer denoting the Theiler window: Two Takens' vectors must be separated by more than *theiler.window* time steps in order to be considered neighbours. By using a Theiler window, we exclude temporally correlated vectors from our estimations.
`min.neighs`	Minimum number of neighbours that a Takens' vector must have to be considered a reference point.
`min.ref.points`	Number of reference points that the routine will try to use. The routine stops when it finds *min.ref.points* reference points, saving computation time.
`max.time.steps`	Integer denoting the number of time steps marking the end of the linear region.
`number.boxes`	Number of boxes that will be used in the box assisted algorithm (see neighbourSearch).
`sampling.period`	Sampling period of the time series. When dealing with a discrete system, the *sampling.period* should be set to 1.
`do.plot`	Logical value. If TRUE (default value), a plot of $S(t)$ Vs t is shown.
`...`	Additional plotting parameters.
`x`	A *maxLyapunov* object.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`type`	Type of plot (see `?plot`).
`col`	Vector of colors for each of the dimensions of the plot.
`pch`	Vector of symbols for each of the dimensions of the plot.
`add.legend`	add a legend to the plot?
`regression.range`	Vector with 2 components denoting the range where the function will perform linear regression.
`use.embeddings`	A numeric vector specifying which embedding dimensions should the *estimate* function use to compute the Lyapunov exponent.
`ylim`	Numeric vector of length 2, giving the y coordinates range.
`fit.col`	A vector of colors to plot the regression lines.
`fit.lty`	The type of line to plot the regression lines.
`fit.lwd`	The width of the line for the regression lines.

Details

It is a well-known fact that close trajectories diverge exponentially fast in a chaotic system. The averaged exponent that determines the divergence rate is called the Lyapunov exponent (usually denoted with λ). If $\delta(0)$ is the distance between two Takens' vectors in the embedding.dim-dimensional space, we expect that the distance after a time t between the two trajectories arising from this two vectors fulfills:

$$\delta(n) \sim \delta(0) \cdot exp(\lambda \cdot t)$$

The lyapunov exponent is estimated using the slope obtained by performing a linear regression of $S(t) = \lambda \cdot t \sim log(\delta(t)/\delta(0))$ on t. $S(t)$ will be estimated by averaging the divergence of several reference points.

The user should plot $S(t) Vs t$ when looking for the maximal lyapunov exponent and, if for some temporal range $S(t)$ shows a linear behaviour, its slope is an estimate of the maximal Lyapunov exponent per unit of time. The estimate routine allows the user to get always an estimate of the maximal Lyapunov exponent, but the user must check that there is a linear region in the $S(t) Vs t$. If such a region does not exist, the estimation should be discarded. The computations should be performed for several embedding dimensions in order to check that the Lyapunov exponent does not depend on the embedding dimension.

Value

A list with three components named $time$ and $s.function$. $time$ is a vector containing the temporal interval where the system evolves. It ranges from 0 to $max.time.steps \cdot sampling.period$. $s.function$ is a matrix containing the values of the $S(t)$ for each t in the time vector(the columns) and each embedding dimension (the rows).

The *divTime* function returns the time in which the divergence of close trajectories was computed.

The *embeddingDims* function returns the embeddings in which the divergence of close trajectories was computed

The *divergence* function returns the rate of divergence of close trajectories needed for the maximum Lyapunov exponent estimation.

In order to obtain an estimation of the Lyapunov exponent the user can use the *estimate* function. The $estimate$ function allows the user to obtain an estimation of the maximal Lyapunov exponent by averaging the slopes of the embedding dimensions specified in the *use.embeddings* parameter. The slopes are determined by performing a linear regression over the radius' range specified in *regression.range*

Author(s)

Constantino A. Garcia

References

Eckmann, Jean-Pierre and Kamphorst, S Oliffson and Ruelle, David and Ciliberto, S and others. Liapunov exponents from time series. Physical Review A, 34-6, 4971–4979, (1986).

Rosenstein, Michael T and Collins, James J and De Luca, Carlo J.A practical method for calculating largest Lyapunov exponents from small data sets. Physica D: Nonlinear Phenomena, 65-1, 117–134, (1993).

Examples

```
## Not run:
## Henon System
h=henon(n.sample=  5000,n.transient= 100, a = 1.4, b = 0.3,
        start = c(0.63954883, 0.04772637), do.plot = FALSE)
my.ts=h$x
ml=maxLyapunov(time.series=my.ts,
               min.embedding.dim=2,
               max.embedding.dim=5,
               time.lag=1,
               radius=0.001,theiler.window=4,
               min.neighs=2,min.ref.points=500,
               max.time.steps=40,do.plot=FALSE)
plot(ml)
ml.estimation = estimate(ml,regression.range = c(0,15),
                         use.embeddings=4:5,
                         do.plot = TRUE)
# The max Lyapunov exponent of the Henon system is 0.41
cat("expected: ",0.41," calculated: ",ml.estimation,"\n")

## Rossler system
r=rossler(a=0.15,b=0.2,w=10,start=c(0,0,0), time=seq(0,1000,0.1),
          do.plot=FALSE)
my.ts=r$x
use.cols = c("#999999","#E69F00","#56B4E9")
ml=maxLyapunov(time.series=my.ts,min.embedding.dim=5,max.embedding.dim = 7,
               time.lag=12,radius=0.1,theiler.window=50,
               min.neighs=5,min.ref.points=length(r),
               max.time.steps=300,number.boxes=NULL,
               sampling.period=0.1,do.plot=TRUE,
               col=use.cols)
#  The max Lyapunov exponent of the Rossler system is 0.09
ml.est=estimate(ml,col=use.cols,do.plot=T,
                fit.lty=1,
                fit.lwd=5)
cat("expected: ",0.090," calculated: ",ml.est,"\n")

## End(Not run)
```

`mcleodLiTest` *McLeod-Li test*

Description

McLeod-Li test for conditional heteroscedascity (ARCH).

Usage

```
mcleodLiTest(time.series, lag.max)
```

Arguments

`time.series`	The original time.series.
`lag.max`	Maximum number of lags for which to test for conditional heteroscedascity.

Value

A list containing the *p.values* for each of the Ljung-Box tests computed using lags ranging from 1 to *lag.max*.

References

Tsay, Ruey S., and Rong Chen. Nonlinear time series analysis. Vol. 891. John Wiley & Sons, 2018.

See Also

`nonlinearityTest`, `keenanTest`, `tsayTest`, `thresholdTest`

`mutualInformation` *Average Mutual Information (AMI)*

Description

Functions for estimating the Average Mutual Information (AMI) of a time series.

Usage

```
mutualInformation(
  time.series,
  lag.max = NULL,
  n.partitions = NULL,
  units = c("Nats", "Bits", "Bans"),
  do.plot = TRUE,
  ...
)

## S3 method for class 'mutualInf'
plot(
  x,
  main = "Average Mutual Information (AMI)",
  xlab = "Time lag",
  ylab = NULL,
  type = "h",
  ...
)

## S3 method for class 'mutualInf'
as.numeric(x, ...)
```

```
## S3 method for class 'mutualInf'
x[i]

## S3 method for class 'mutualInf'
x[[i]]
```

Arguments

`time.series`	The observed time series.
`lag.max`	Largest lag at which to calculate the AMI.
`n.partitions`	Number of bins used to compute the probability distribution of the time series.
`units`	The units for the mutual information. Allowed units are "Nats", "Bits" or "Bans" (somethings called Hartleys). Default is "Nats".
`do.plot`	Logical value. If TRUE, the AMI is plotted
`...`	Further arguments for the plotting function.
`x`	A *mutualInf* object.
`main`	Title for the plot.
`xlab`	Title for the x axis.
`ylab`	Title for the y axis.
`type`	Type of plot to be drawn.
`i`	Indices specifying elements to extract.

Details

The Average Mutual Information (AMI) measures how much one random variable tells us about another. In the context of time series analysis, AMI helps to quantify the amount of knowledge gained about the value of $x(t+\tau)$ when observing $x(t)$.

To measure the AMI iof a time series, we create a histogram of the data using bins. Let p_i the probability that the signal has a value inside the ith bin, and let $p_{ij}(\tau)$ be the probability that $x(t)$ is in bin i ans $x(t+\tau)$ is in bin j. Then, AMI for time delay τ is defined as

$$AMI(\tau) = \sum_{i,j} p_{ij} log(\frac{p_{ij}}{p_i p_j})$$

Depending on the base of the logarithm used to define AMI, the AMI is measured in bits (base 2, also called shannons), nats (base e) or bans (base 10, also called hartleys).

Value

A *mutualInf* object that consist of a list containing all the relevant information of the AMI computation: *time.lag*, *mutual.information*, *units* and *n.partitions*.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press) H. Abarbanel: Analysis of observed chaotic data (Springer, 1996).

See Also

`timeLag`

Examples

```
## Not run:
sx = sinaiMap(a=0.3,n.sample=5000,start=c(0.23489,0.8923),do.plot=FALSE)$x
mutinf = mutualInformation(sx, n.partitions = 20, units = "Bits")
## End(Not run)
```

`neighbourSearch` *neighbour search*

Description

This function finds all the neighbours of a given Takens' vector. The neighbours are found using a box assisted algorithm that creates a wrapped grid with a given number of boxes per dimension.

Usage

```
neighbourSearch(takens, positionTakens, radius, number.boxes = NULL)
```

Arguments

`takens`	The matrix containing all the Takens' vectors (see buildTakens).
`positionTakens`	Integer denoting the Takens' vector whose neighbours will be searched.
`radius`	Distance in which the algorithm will search for neighbours.
`number.boxes`	Integer denoting the number of boxes per dimension that will be used to construct a wrapped grid (see Schreiber). If the user does not specify a number of boxes, this function estimates a proper number.

Value

A containing all the neighbours of the *positionTakens-th* Takens' vector. If the list is empty, that means that there is no neighbour of the *positionTakens-th* Takens' vector in the given radius.

Author(s)

Constantino A. Garcia

References

Schreiber, T. Efficient neighbor searching in nonlinear time series analysis. Int. J. Bifurcation and Chaos, 5, p. 349, (1995).

See Also

`findAllNeighbours`.

`nlOrder` *Get the order of the nonlinear chaotic invariant.*

Description

Get the order of the nonlinear chaotic invariant.

Usage

```
nlOrder(x)
```

Arguments

`x`	An object containing all the information needed for the estimate of the chaotic invariant.

Value

A numeric vector with the radius of the neighborhoods used for the computations of a chaotic invariant.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

See Also

`corrDim`, `sampleEntropy`

`nonlinearityTest` *Nonlinearity test*

Description

Nonlinearity test

Usage

```
nonlinearityTest(time.series, verbose = TRUE)
```

Arguments

`time.series`	The original time.series from which the surrogate data is generated.
`verbose`	Logical value. If TRUE, a summary of each of the tests is shown.

Details

This function runs a set of nonlinearity tests implemented by this and other R packages, including:

- Teraesvirta's neural metwork test for nonlinearity (`terasvirta.test`).
- White neural metwork test for nonlinearity (`white.test`).
- Keenan's one-degree test for nonlinearity (`keenanTest`).
- Perform the McLeod-Li test for conditional heteroscedascity (ARCH). (`mcleodLiTest`).
- Perform the Tsay's test for quadratic nonlinearity in a time series. (`tsayTest`).
- Perform the Likelihood ratio test for threshold nonlinearity. (`thresholdTest`).

Value

A list containing the results of each of the tests.

See Also

`keenanTest`, `tsayTest`, `mcleodLiTest`,`thresholdTest`

`nonLinearNoiseReduction` *Nonlinear noise reduction*

Description

Function for denoising a given time series using nonlinear analysis techniques.

Usage

```
nonLinearNoiseReduction(time.series, embedding.dim, radius)
```

Arguments

`time.series`	The original time series to denoise.
`embedding.dim`	Integer denoting the dimension in which we shall embed the *time.series*.
`radius`	The radius used to looking for neighbours in the phase space (see details).

Details

This function takes a given time series and denoises it. The denoising is achieved by averaging each Takens' vector in an m-dimensional space with his neighbours (time lag=1). Each neighbourhood is specified with balls of a given radius (max norm is used).

Value

A vector containing the denoised time series.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

`nonLinearPrediction` *Nonlinear time series prediction*

Description

Function for predicting futures values of a given time series using previous values and nonlinear analysis techniques.

Usage

```
nonLinearPrediction(
  time.series,
  embedding.dim,
  time.lag,
  prediction.step,
  radius,
  radius.increment
)
```

Arguments

`time.series`	Previous values of the time series that the algorithm will use to make the prediction.
`embedding.dim`	Integer denoting the dimension in which we shall embed the *time.series*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.
`prediction.step`	Integer denoting the number of time steps ahead for the forecasting.
`radius`	The radius used to looking for neighbours in the phase space (see details).
`radius.increment`	The increment used when no neighbours are found (see details).

Details

Using *time.series* measurements, an embedding in *embedding.dim*-dimensional phase space with time lag *time.lag* is used to predict the value following the given time series after *prediction.steps* sample steps. This is done by finding all the neighbours of the last Takens' vector in a radius of size *radius* (the max norm is used). If no neighbours are found within a distance radius, the neighbourhood size is increased until succesful using *radius.increment*(*radius* = *radius* + *radius.increment*).

Value

The predicted value *prediction.step* time steps ahead.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
## Not run:
h=henon(n.sample=5000,start=c(0.324,-0.8233))
predic=nonLinearPrediction(time.series=h$x[10:2000],embedding.dim=2,
                           time.lag=1,
                           prediction.step=3,radius=0.03,
                           radius.increment=0.03/2)
cat("real value: ",h$x[2003],"Vs Forecast:",predic)

## End(Not run)
```

`plotLocalScalingExp` *Plot local scaling exponents*

Description

Plots the local scaling exponents of the correlation sum or the average Shannon information (when computing information dimension).

Usage

```
plotLocalScalingExp(x, ...)
```

Arguments

`x`	An object containing all the information needed for the estimate of the chaotic invariant.
`...`	Additional graphical parameters.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

`poincareMap` *Poincare map*

Description

Computes the Poincare map of the reconstructed trajectories in the phase-space.

The Poincare map is a classical dynamical system technique that replaces the n-th dimensional trajectory in the phase space with an (n-1)-th order discrete-time called the Poincare map. The points of the Poincare map are the intersection of the trajectories in the phase-space with a certain Hyper-plane.

Usage

```
poincareMap(
  time.series = NULL,
  embedding.dim = 2,
  time.lag = 1,
  takens = NULL,
  normal.hiperplane.vector = NULL,
  hiperplane.point
)
```

Arguments

`time.series`	The original time series from which the phase-space reconstruction is done.
`embedding.dim`	Integer denoting the dimension in which we shall embed the *time.series*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.
`takens`	Instead of specifying the *time.series*, the *embedding.dim* and the *time.lag*, the user may specify directly the Takens' vectors.
`normal.hiperplane.vector`	The normal vector of the hyperplane that will be used to compute the Poincare map. If the vector is not specifyed the program choses the vector (0,0,...,1).
`hiperplane.point`	A point on the hyperplane (an hyperplane is defined with a point and a normal vector).

Details

This function computes the Poincare map taking the Takens' vectors as the continuous trajectory in the phase space. The *takens* param has been included so that the user may specify the real phase-space instead of using the phase-space reconstruction (see examples).

Value

Since there are three different Poincare maps, an R list is returned storing all the information related which all of these maps:

- The positive Poincare map is formed by all the intersections with the hyperplane in positive direction (defined by the normal vector). The *pm.pos* returns the points of the map whereas that *pm.pos.time* returns the number of time steps since the beginning where the intersections occurred. Note that these time steps probably won't be integers since the algorithm uses an interpolation procedure for calculating the intersection with the hyperplane.
- Similarly we define a negative Poincare map (*pm.neg* and *pm.neg.time*).
- Finally, we may define a two-side Poincare map that stores all the intersections (no matter the direction of the intersection) (*pm* and *pm.time*).

Author(s)

Constantino A. Garcia

References

Parker, T. S., L. O. Chua, and T. S. Parker (1989). Practical numerical algorithms for chaotic systems. Springer New York

Examples

```
## Not run:
r=rossler(a = 0.2, b = 0.2, w = 5.7, start=c(-2, -10, 0.2),
time=seq(0,300,by = 0.01), do.plot=FALSE)
takens=cbind(r$x,r$y,r$z)
# calculate poincare sections
pm=poincareMap(takens = takens,normal.hiperplane.vector = c(0,1,0),
 hiperplane.point=c(0,0,0) )
if (requireNamespace("rgl", quietly = TRUE)) {
   rgl::plot3d(takens,size=0.7)
   rgl::points3d(pm$pm,col="red")
}

## End(Not run)
```

`radius` *Get the radius of the neighborhoods used for the computations of a chaotic invariant.*

Description

Get the radius of the neighborhoods used for the computations of a chaotic invariant.

Usage

```
radius(x)
```

Arguments

`x`	An object containing all the information needed for the estimate of the chaotic invariant.

Value

A numeric vector with the radius of the neighborhoods used for the computations of a chaotic invariant.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

`recurrencePlot` *Recurrence Plot*

Description

Plot the recurrence matrix.

Usage

```
recurrencePlot(
  takens = NULL,
  time.series,
  embedding.dim,
  time.lag,
  radius,
  ...
)
```

Arguments

`takens`	Instead of specifying the *time.series*, the *embedding.dim* and the *time.lag*, the user may specify directly the Takens' vectors.
`time.series`	The original time series from which the phase-space reconstruction is performed.
`embedding.dim`	Integer denoting the dimension in which we shall embed the *time.series*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.
`radius`	Maximum distance between two phase-space points to be considered a recurrence.
`...`	Additional plotting parameters.

Details

WARNING: This function is computationally very expensive. Use with caution.

Author(s)

Constantino A. Garcia

References

Zbilut, J. P. and C. L. Webber. Recurrence quantification analysis. Wiley Encyclopedia of Biomedical Engineering (2006).

`rossler` *Rossler system*

Description

Generates a 3-dimensional time series using the Rossler equations.

Usage

```
rossler(
  a = 0.2,
  b = 0.2,
  w = 5.7,
  start = c(-2, -10, 0.2),
  time = seq(0, 50, by = 0.01),
  do.plot = deprecated()
)
```

Arguments

`a`	The *a* parameter. Default:0.2.
`b`	The *b* parameter. Default: 0.2.
`w`	The *w* parameter. Default: 5.7.
`start`	A 3-dimensional numeric vector indicating the starting point for the time series. Default: c(-2, -10, 0.2).
`time`	The temporal interval at which the system will be generated. Default: time=seq(0,50,by = 0.01).
`do.plot`	Logical value. If TRUE, a plot of the generated Lorenz system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Rossler system is a system of ordinary differential equations defined as:

$$\dot{x} = -(y + z)$$

$$\dot{y} = x + a \cdot y$$

$$\dot{z} = b + z * (x - w)$$

The default selection for the system parameters (a = 0.2, b = 0.2, w = 5.7) is known to produce a deterministic chaotic time series.

Value

A list with four vectors named *time*, x, y and z containing the time, the x-components, the y-components and the z-components of the Rossler system, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering (Studies in Nonlinearity)

See Also

`henon`, `logisticMap`, `rossler`,`ikedaMap`, `cliffordMap`, `sinaiMap`, `gaussMap`

Examples

```
## Not run:
r.ts = rossler(time=seq(0,30,by = 0.01))

## End(Not run)
```

`rqa` *Recurrence Quantification Analysis (RQA)*

Description

The Recurrence Quantification Analysis (RQA) is an advanced technique for the nonlinear analysis that allows to quantify the number and duration of the recurrences in the phase space.

Usage

```
rqa(
  takens = NULL,
  time.series = NULL,
  embedding.dim = 2,
  time.lag = 1,
  radius,
  lmin = 2,
  vmin = 2,
  distanceToBorder = 2,
  save.RM = TRUE,
  do.plot = FALSE,
  ...
)
```

Arguments

`takens`	Instead of specifying the *time.series*, the *embedding.dim* and the *time.lag*, the user may specify directly the Takens' vectors.
`time.series`	The original time series from which the phase-space reconstruction is performed.
`embedding.dim`	Integer denoting the dimension in which we shall embed the *time.series*.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.
`radius`	Maximum distance between two phase-space points to be considered a recurrence.
`lmin`	Minimal length of a diagonal line to be considered in the RQA. Default *lmin* = 2.
`vmin`	Minimal length of a vertical line to be considered in the RQA. Default *vmin* = 2.
`distanceToBorder`	In order to avoid border effects, the *distanceToBorder* points near the border of the recurrence matrix are ignored when computing the RQA parameters. Default, *distanceToBorder* = 2.
`save.RM`	Logical value. If TRUE, the recurrence matrix is stored as a sparse matrix. Note that computing the recurrences in matrix form can be computationally expensive.

`do.plot`	Logical. If TRUE, the recurrence plot is shown. However, plotting the recurrence matrix is computationally expensive. Use with caution.
`...`	Additional plotting parameters.

Value

A *rqa* object that consist of a list with the most important RQA parameters:

- *recurrence.matrix*: A sparse symmetric matrix containing the recurrences of the phase space.
- *REC*: Recurrence. Percentage of recurrence points in a Recurrence Plot.
- *DET*: Determinism. Percentage of recurrence points that form diagonal lines.
- *LAM*: Percentage of recurrent points that form vertical lines.
- *RATIO*: Ratio between *DET* and *RR*.
- *Lmax*: Length of the longest diagonal line.
- *Lmean*: Mean length of the diagonal lines. The main diagonal is not taken into account.
- *DIV*: Inverse of *Lmax*.
- *Vmax*: Longest vertical line.
- *Vmean*: Average length of the vertical lines. This parameter is also referred to as the Trapping time.
- *ENTR*: Shannon entropy of the diagonal line lengths distribution
- *TREND*: Trend of the number of recurrent points depending on the distance to the main diagonal
- *diagonalHistogram*: Histogram of the length of the diagonals.
- *recurrenceRate*: Number of recurrent points depending on the distance to the main diagonal.

Author(s)

Constantino A. Garcia and Gunther Sawitzki

References

Zbilut, J. P. and C. L. Webber. Recurrence quantification analysis. Wiley Encyclopedia of Biomedical Engineering (2006).

Examples

```
## Not run:
rossler.ts =  rossler(time=seq(0, 10, by = 0.01),do.plot=FALSE)$x
rqa.analysis=rqa(time.series = rossler.ts, embedding.dim=2, time.lag=1,
               radius=1.2,lmin=2,do.plot=FALSE,distanceToBorder=2)
plot(rqa.analysis)

## End(Not run)
```

`sampleEntropy` *Sample Entropy (also known as Kolgomorov-Sinai Entropy)*

Description

The Sample Entropy measures the complexity of a time series. Large values of the Sample Entropy indicate high complexity whereas that smaller values characterize more regular signals.

Usage

```
sampleEntropy(corrDim.object, do.plot = TRUE, ...)

## S3 method for class 'sampleEntropy'
sampleEntropyFunction(x)

## S3 method for class 'sampleEntropy'
nlOrder(x)

## S3 method for class 'sampleEntropy'
radius(x)

## S3 method for class 'sampleEntropy'
embeddingDims(x)

## S3 method for class 'sampleEntropy'
plot(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  type = "l",
  col = NULL,
  pch = NULL,
  ylim = NULL,
  log = "x",
  add.legend = T,
  ...
)

## S3 method for class 'sampleEntropy'
estimate(
  x,
  regression.range = NULL,
  do.plot = TRUE,
  use.embeddings = NULL,
  fit.col = NULL,
  fit.lty = 2,
```

```
  fit.lwd = 2,
  add.legend = T,
  ...
)
```

Arguments

`corrDim.object`	A *corrDim* object from which the Sample Entropy of the time series characterized by *corrDim* shall be estimated.
`do.plot`	do.plot Logical value. If TRUE (default value), a plot of the sample entropy is shown.
`...`	Additional plotting arguments.
`x`	A *sampleEntropy* object.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`type`	Type of plot (see `?plot`).
`col`	Vector of colors for each of the dimensions of the plot.
`pch`	Vector of symbols for each of the dimensions of the plot
`ylim`	Numeric vector of length 2, giving the y coordinates range..
`log`	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
`add.legend`	add a legend to the plot?
`regression.range`	Vector with 2 components denoting the range where the function will perform linear regression.
`use.embeddings`	A numeric vector specifying which embedding dimensions should the *estimate* function use to compute the sample entropy.
`fit.col`	A vector of colors to plot the regression lines.
`fit.lty`	The type of line to plot the regression lines.
`fit.lwd`	The width of the line for the regression lines.

Details

The sample entropy is computed using:

$$h_q(m,r) = log(C_q(m,r)/C_q(m+1,r))$$

where *m* is the embedding dimension and *r* is the radius of the neighbourhood. When computing the correlation dimensions we use the linear regions from the correlation sums in order to do the estimates. Similarly, the sample entropy $h_q(m,r)$ should not change for both various *m* and *r*.

For each embedding dimension the sample entropy is estimated by averaging

$$h_q(m,r) = log(C_q(m,r)/C_q(m+1,r))$$

over the range specified by *regression range* in the *estimate* function.

Value

A *sampleEntropy* object that contains a list storing the sample entropy (*sample.entropy*), the embedding dimensions (*embedding.dims*) and radius (*radius*) for which the sample entropy has been computed, and the order of the sample entropy (*entr.order*). The sample entropy is stored as a matrix in which each row contains the computations for a given embedding dimension and each column stores the computations for a given radius.

The *sampleEntropyFunction* returns the sample entropy function $h_q(m, r)$ used for the computations. The sample entropy function is represented by a matrix. Each row represents a given embedding dimension whereas that each column representes a different radius.

The *nlOrder* function returns the order of the sample entropy.

The *radius* function returns the radius on which the sample entropy function has been evaluated.

The *embeddingDims* function returns the embedding dimensions on which the sample entropy function has been evaluated.

The *plot* function shows the graphics for the sample entropy.

The *estimate* function returns a vector storing the sample entropy estimate for each embedding dimension.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
## Not run:
x=henon(n.sample = 15000, n.transient = 100, a = 1.4, b = 0.3,
        start = c(0.78,0.8165), do.plot = FALSE)$x

cd=corrDim(time.series=x,
           min.embedding.dim=2,max.embedding.dim=9,
           corr.order=2,time.lag=1,
           min.radius=0.05,max.radius=1,
           n.points.radius=100,
           theiler.window=20,
           do.plot=TRUE)

use.col = c("#999999", "#E69F00", "#56B4E9", "#009E73",
            "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
se=sampleEntropy(cd,do.plot=TRUE,col=use.col,
                 type="l",xlim=c(0.1,1),
                 add.legend=T)
se.est = estimate(se,
                  regression.range = c(0.4,0.6),
                  use.embeddings = 6:9,col=use.col,type="b")
print(se.est)
```

```
cat("Expected K2 = ",0.325," Estimated = ",mean(se.est),"\n")

## End(Not run)
```

`sampleEntropyFunction` *Returns the sample entropy function $h\_q(m, r)$ used for the computations of the sample entropy.*

Description

Returns the sample entropy function $h_q(m, r)$ used for the computations of the sample entropy.

Usage

```
sampleEntropyFunction(x)
```

Arguments

`x`	A *sampleEntropy* object.

Value

A numeric matrix representing the sample entropy function $h_q(m, r)$ obtained in #' the sample entropy computations represented by the *sampleEntropy* object.

See Also

`sampleEntropy`

`sinaiMap` *Sinai map*

Description

Generates a 2-dimensional time series using the Sinai map.

Usage

```
sinaiMap(
  a = 0.1,
  start = runif(2),
  n.sample = 5000,
  n.transient = 500,
  do.plot = deprecated()
)
```

Arguments

`a`	The *a* parameter. Default: 0.1
`start`	A 2-dimensional vector indicating the starting values for the x and y Sinai coordinates. If the starting point is not specified, it is generated randomly.
`n.sample`	Length of the generated time series. Default: 5000 samples.
`n.transient`	Number of transient samples that will be discarded. Default: 500 samples.
`do.plot`	Logical value. If TRUE, a plot of the generated Sinai system is shown. Before version 0.2.11, default value was TRUE; versions 0.2.11 and later use FALSE as default.

Details

The Sinai map is defined as follows:

$$x_{n+1} = (x_n + y_n + a \cdot cos(2 \cdot pi \cdot y_n)) mod 1$$

$$y_{n+1} = (x_n + 2 \cdot y_n) mod 1$$

The default selection for the *a* parameter is known to produce a deterministic chaotic time series.

Value

A list with two vectors named x and y containing the x-components and the y-components of the Sinai map, respectively.

Note

Some initial values may lead to an unstable system that will tend to infinity.

Author(s)

Constantino A. Garcia

References

Mcsharry, P. E. and P. R. Ruffino (2003). Asymptotic angular stability in nonlinear systems: rotation numbers and winding numbers. Dynamical Systems 18(3), 191-200.

See Also

`henon`, `logisticMap`, `lorenz`,`rossler`, `ikedaMap`, `cliffordMap`, `gaussMap`

Examples

```
## Not run:
sinai.map = sinaiMap(n.sample = 1000, n.transient=10,do.plot=TRUE)
# accessing the x coordinate and plotting it
plot(ts(sinai.map$x))

## End(Not run)
```

`spaceTimePlot` *Space Time plot*

Description

The space time separation is a broadly-used method of detecting non-stationarity and temporal correlations in the time series being analyzed. The space time separation plot is also used to select a proper Theiler window by selecting a temporal separation enough to saturate the contour lines.

Usage

```
spaceTimePlot(
  takens = NULL,
  time.series = NULL,
  embedding.dim = 2,
  time.lag = 1,
  max.radius = NULL,
  time.step = 1,
  number.time.steps = NULL,
  numberPercentages = 10,
  do.plot = TRUE,
  ...
)

## S3 method for class 'spaceTimePlot'
contourLines(x)

## S3 method for class 'spaceTimePlot'
getContourLines(x)

## S3 method for class 'spaceTimePlot'
plot(
  x,
  main = "Space time separation plot",
  xlab = NULL,
  ylab = NULL,
  type = "l",
  ylim = NULL,
  col = NULL,
  pch = NULL,
  add.legend = TRUE,
  ...
)
```

Arguments

`takens`	Instead of specifying the *time.series*, the *embedding.dim* and the *time.lag*, the user may specify directly the Takens' vectors.

`time.series`	The original time series being analyzed.
`embedding.dim`	Integer denoting the dimension in which we shall embed the time series.
`time.lag`	Integer denoting the number of time steps that will be use to construct the Takens' vectors.
`max.radius`	Maximum neighbourhood radius in which the algorithm will look for finding neighbours. This parameter may be used to avoid heavy computations. If the user does not specify a radius, the algorithm estimates it.
`time.step`	Integer denoting the number of discrete steps between two calculations of the space time plot.
`number.time.steps`	Integer denoting the number of temporal jumps in steps of *time.step* in which we want to compute the space time plot.
`numberPercentages`	Number of contour lines to be computed. Each contour line represent a concrete percentage of points (see Details).
`do.plot`	Logical. If TRUE, the time space plot is shown.
`...`	Additional plotting parameters.
`x`	A *spaceTimePlot* object.
`main`	A title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`type`	Type of plot (see `?plot`).
`ylim`	Numeric vector of length 2, giving the y coordinates range.
`col`	Vector of colors for each of the percentages of the plot.
`pch`	Vector of symbols for each of the percentages of the plot.
`add.legend`	add a legend to the plot?

Details

Each contour line of the space time plot indicate the distance you have to go (y-axis) to find a given fraction of neighbour pairs, depending on their temporal separation (x-axis).

WARNING: The parameter *number.time.steps* should be used with caution since this method performs heavy computations.

Value

A *timeSpacePlot* object that consist, essentially, of a matrix storing the values for each contour line. Each row stores the value for a given percentage of points. Each column stores the value of the radius you have to go to find a given fraction of neighbour pairs (the rows), depending on their temporal separation (the colums). This matrix can be accessed by using the *contourlines* method.

The *contourLines* function returns the contour lines of the space time plot.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

Examples

```
 ## Not run:
 tak = buildTakens(sin(2*pi*0.005*(0:5000)),2,1)
 stp.test = spaceTimePlot(takens=tak,number.time.steps=400,do.plot=TRUE)

## End(Not run)
```

`surrogateTest` *Surrogate data testing*

Description

Surrogate data testing

Usage

```
surrogateTest(
  time.series,
  significance = 0.05,
  one.sided = FALSE,
  alternative = c("smaller", "larger"),
  K = 1,
  FUN,
  verbose = TRUE,
  do.plot = TRUE,
  xlab = "Values of the statistic",
  ylab = "",
  main = "Surrogate data testing",
  ...
)
```

Arguments

`time.series`	The original time.series from which the surrogate data is generated.
`significance`	Significance of the test
`one.sided`	Logical value. If *TRUE*, the routine runs a one-side test. If *FALSE*, a two-side test is applied (default).

`alternative`	Specifies the concrete type of one-side test that should be performed: If the the user wants to test if the statistic from the original data is smaller (*alternative="smaller"*) or larger (*alternative="larger"*) than the expected value under the null hypothesis.
`K`	Integer controlling the number of surrogates to be generated (see details).
`FUN`	The function that computes the discriminating statistic that shall be used for testing.
`verbose`	Logical value. If TRUE, a brief summary of the test is shown.
`do.plot`	Logical value. If TRUE, a graphical representation of the statistic value for both surrogates and original data is shown.
`xlab`	a title for the x axis.
`ylab`	a title for the y axis.
`main`	an overall title for the plot.
`...`	Additional arguments for the FUN function.

Details

This function tests the null hypothesis (H0) stating that the series is a gaussian linear process. The test is performed by generating several surrogate data according to H0 and comparing the values of a discriminating statistic between both original data and the surrogate data. If the value of the statistic is significantly different for the original series than for the surrogate set, the null hypothesis is rejected and nonlinearity assumed.

To test with a significance level of α if the statistic from the original data is smaller than the expected value under the null hypothesis (a one-side test), $K/\alpha - 1$ surrogates are generated. The null hypothesis is then rejected if the statistic from the data has one of the K smallest values. For a two-sided test, $2K/\alpha - 1$ surrogates are generated. The null hypothesis is rejected if the statistic from the data gives one of the K smallest or largest values.

The surrogate data is generated by using a phase randomization procedure.

Value

A list containing the values of the statistic for the surrogates (*surrogates.statistics* field) and the value for the original time series (*data.statistic*)

Author(s)

Constantino A. Garcia

References

SCHREIBER, Thomas; SCHMITZ, Andreas. Surrogate time series. Physica D: Nonlinear Phenomena, 2000, vol. 142, no 3, p. 346-382.

Examples

```
## Not run:
 lx = lorenz(do.plot=F)$x
 sdt = surrogateTest(time.series = lx,significance = 0.05,
                     K=5, one.sided = FALSE, FUN=timeAsymmetry)

## End(Not run)
```

`thresholdTest` *Threshold nonlinearity test*

Description

Computes the likelihood ratio test for threshold nonlinearity with H0 being an AR process and H1 a TAR model.

Usage

```
thresholdTest(
  time.series,
  p,
  d = 1,
  lower.percent = 0.25,
  upper.percent = 0.75
)
```

Arguments

`time.series`	The original time.series.
`p`	The order of the AR process.
`d`	Delay used for the threshold value in the TAR process.
`lower.percent`	The threshold value is searched over an interval defined by *lower.percent* and *upper.percent* of the time series values. This defines the lower percent.
`upper.percent`	The threshold value is searched over an interval defined by *lower.percent* and *upper.percent* of the time series values. This defines the upper percent.

Value

A list containing

- p.value: p-value of the test
- test.statistic: Likelihood ratio test statistic.
- percentiles: Since the search for the threshold parameter may occur in a narrower interval than the one specified by the user, the effective lower and upper percents are returned here.

Note

Adapted from the *tlrt* function of the *TSA* package.

Author(s)

Kung-Sik Chan

References

Chan, K.S. (1990). Percentage points of likelihood ratio tests for threshold autoregression. Journal of Royal Statistical Society, B 53, 3, 691-696.

See Also

`nonlinearityTest`, `keenanTest`, `tsayTest`, `mcleodLiTest`

`timeAsymmetry` *Time Reversibility statistic*

Description

Time Reversibility statistic

Usage

```
timeAsymmetry(time.series)
```

Arguments

`time.series`	The time series used to compute the statistic.

Details

The time simmetry statistic measures the asymmetry of a time series under time reversal by calculating:

$$E[s_n \cdot s_{n+1}^2] - E[s_n^2 \cdot s_{n+1}]$$

. Since linear stochastic series are symmetric under time reversal, this statistic may be used for testing the assertion that the data was generated from a stationary linear stochastic process or not.

Value

The time simmetry statistic.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press, second edition, section 7.1.3)

`timeAsymmetry2` *Time Reversibility statistic*

Description

Time Reversibility statistic

Usage

```
timeAsymmetry2(time.series, tau)
```

Arguments

`time.series`	The time series used to compute the statistic
`tau`	Time delay used to compute the third order statistic.

Details

The time simmetry statistic measures the asymmetry of a time series under time reversal by implementing the third order statistic:

$$E[s_n - s_{n-\tau})^3]$$

. Since linear stochastic series are symmetric under time reversal, this statistic may be used for testing the assertion that the data was generated from a stationary linear stochastic process or not.

Value

The time simmetry statistic for the delays specified with *tau*.

Author(s)

Constantino A. Garcia

See Also

`timeAsymmetry`

`timeLag` *Estimate an appropiate time lag for the Takens' vectors*

Description

Given a time series (time.series), an embedding dimension (m) and a time lag (timeLag), the n^{th} Takens' vector is defined as

$$T[n] = \{time.series[n], time.series[n + timeLag], ...time.series[n + m * timeLag]\}.$$

This function estimates an appropiate time lag by using the autocorrelation function or the average mutual information .

Usage

```
timeLag(
  time.series,
  technique = c("acf", "ami"),
 selection.method = c("first.e.decay", "first.zero", "first.minimum", "first.value"),
  value = 1/exp(1),
  lag.max = NULL,
  do.plot = TRUE,
  main = NULL,
  ...
)
```

Arguments

`time.series`	The original time series.
`technique`	The technique that we shall use to estimate the time lag (see the Details section). Allowed values are *"acf"* and *"ami"*.
`selection.method`	Method used for selecting a concrete time lag. Available methods are *"first.zero"*, *"first.e.decay"* (default), *"first.minimum"* and *"first.value"*.
`value`	Numeric value indicating the value that the autocorrelation/AMI function must cross in order to select the time lag. It is used only with the "first.value" selection method.
`lag.max`	Maximum lag at which to calculate the acf/AMI.
`do.plot`	Logical value. If TRUE (default value), a plot of the autocorrelation/AMI function is shown.
`main`	A title for the plot.
`...`	Additional parameters for the *acf* or the *mutualInformation*.

Details

A basic criteria for estimating a proper time lag is based on the following reasoning: if the time lag used to build the Takens' vectors is too small, the coordinates will be too highly temporally correlated and the embedding will tend to cluster around the diagonal in the phase space. If the time lag is chosen too large, the resulting coordinates may be almost uncorrelated and the resulting embedding will be very complicated. Thus, the autocorrelation function can be used for estimating an appropiate time lag of a time series. However, it must be noted that the autocorrelation is a linear statistic, and thus it does not take into account nonlinear dynamical correlations. To take into account nonlinear correlations the average mutual information (AMI) can be used. Independently of the technique used to compute the correlation, the time lag can be selected in a variety of ways:

- Select the time lag where the autocorrelation/AMI function decays to 0 (*first.zero* selection method). This method is not appropriate for the AMI function, since it only takes positive values.
- Select the time lag where the autocorrelation/AMI function decays to 1/e of its value at zero (*first.e.decay* selection method).
- Select the time lag where the autocorrelation/AMI function reaches its first minimum (*first.minimum* selection method).
- Select the time lag where the autocorrelation/AMI function decays to the value specified by the user (*first.value* selection method and *value* parameter).

Value

The estimated time lag.

Note

If the autocorrelation/AMI function does not cross the specifiged value, an error is thrown. This may be solved by increasing the *lag.max* or selecting a higher value to which the autocorrelation/AMI function may decay.

Author(s)

Constantino A. Garcia

References

H. Kantz and T. Schreiber: Nonlinear Time series Analysis (Cambridge university press)

See Also

`mutualInformation`

Examples

```
## Not run:
sx = sinaiMap(a=0.3,n.sample=5000,start=c(0.23489,0.8923),do.plot=FALSE)$x
timeLag(sx, technique="ami",
        n.partitions = 20, units = "Bits")
```

```
timeLag(sx, technique="acf")
## End(Not run)
```

`tsayTest` *Tsay's test*

Description

Tsay's test: test for nonlinearity against the null hypothesis that the time series follows some AR process. This is a generalization of Keenan's test.

Usage

```
tsayTest(time.series, order)
```

Arguments

`time.series`	The original time.series.
`order`	Order used for the AR model.

Value

A list containing the results of the test, including:

- test.stat: the F-squared test statistic
- p.value.
- order: order of the AR process used for testing.

References

Tsay, R. S. (1986), Nonlinearity test for time series, Biometrika, 73, 461-466.

See Also

`nonlinearityTest`, `keenanTest`, `mcleodLiTest`,`thresholdTest`

`windowSizes` *Returns the window sizes used for DFA in a* dfa *object.*

Description

Returns the window sizes used for DFA in a *dfa* object.

Usage

```
windowSizes(x)
```

Arguments

`x`	A *dfa* object.

Value

The *windowSizes* function returns the windows sizes used to detrend the time series in the DFA.

See Also

`dfa`

Index