Package 'fractal'

July 2, 2014

Type Package

Title Fractal Time Series Modeling and Analysis

Version 2.0-0

Author William Constantine and Donald Percival (Applied Physics Laboratory, University of Washington)

Maintainer Don Percival <dbp@apl.washington.edu>

Depends R (>= 3.0.2), splus2R (>= 1.2-0), ifultools (>= 2.0-0)

Imports sapa(>= 2.0-0), wmtsa (>= 2.0-0), scatterplot3d

Suggests akima

Description Stochastic fractal and deterministic chaotic time series analysis.

License GPL-2

Collate fra_chaos.R fra_class.R fra_detm.R fra_dim.R fra_fdp.R fra_filt.R fra_hurst.R fra_kde.R fra_model.R fra_neig.R fra_stny.R fra_surr.R fra_pkg.R

Repository CRAN

NeedsCompilation no

Date/Publication 2014-01-03 18:16:41

R topics documented:

`beamchaos` *Chaotic beam data*

Description

A flexible thin steel beam was mounted vertically to a electromechanical shaker which provided a transverse sinusoidal excitation. The beam tip was placed near two rare earth magnets so as to provide nonlinear buckling forces. The beam was treated with a viscoelastic strip adhered to one side to provide a little damping. The addition of the damping treatment helps to form a more distinguishable fractal structure in phase space embeddings. A laser vibrometer was used to record the beam tip velocity and the analog signal streamed to a National Instruments data acquisition board. The data was sampled at 1000 Hz. The gain of excitation was adjusted until (seemingly) chaotic motion was observed.

References

William Constantine (1999), Ph.D. Dissertation: *Wavelet Techniques for Chaotic and Fractal Dynamics*, Mechanical Engineering Department, University of Washington.

See Also

ecgrr, eegduke, lorenz, pd5si.

Examples

```
plot(beamchaos)
```

---

chaoticInvariant *Class for chaotic invariant objects*

Description

Class constructor for chaoticInvariant.

S3 METHODS

eda.plot plots an extended data analysis plot, which graphically summarizes the process of obtaining a correlation dimension estimate. A time history, phase plane embeddding, correlation summation curves, and the slopes of correlation summation curves as a function of scale are plotted.

plot plots the correlation summation curves on a log-log scale. The following options may be used to adjust the plot components:

- **type** Character string denoting the type of data to be plotted. The "stat" option plots the correlation summation curves while the "dstat" option plots a 3-point estimate of the derivatives of the correlation summation curves. The "slope" option plots the estimated slope of the correlation summation curves as a function of embedding dimension. Default: "stat".
- **fit** Logical flag. If TRUE, a regression line is overlaid for each curve. Default: TRUE.
- **grid** Logical flag. If TRUE, a grid is overlaid on the plot. Default: TRUE.
- **legend** Logical flag. If TRUE, a legend of the estimated slopes as a function of embedding dimension is displayed. Default: TRUE.
- **...** Additional plot arguments (set internally by the par function).

print prints a qualitiative summary of the results.

See Also

infoDim, corrDim, lyapunov.

Examples

```
## create a faux object of class chaoticInvariant
faux.data <- list(matrix(rnorm(1024), ncol=2), matrix(1:512))
chaoticInvariant(faux.data,
    dimension   = 1:2,
    n.embed     = 10,
    n.reference = 50,
    n.neighbor  = 35,
    tlag        = 10,
    olag        = 15,
    resolution  = 2,
    series.name = "my series",
    series      = 1:10,
    ylab        = "log2(C2)",
    xlab        = "log2(scale)",
    metric      = Inf,
    invariant   = "correlation dimension")
```

`corrDim` *Correlation dimension*

Description

Estimates the correlation dimension by forming a delay embedding of a time series, calculating correlation summation curves (one per embedding dimension), and subsequently fitting the slopes of these curves on a log-log scale using a robust linear regression model. If the slopes converge at a given embedding dimension E, then E is the correct embedding dimension and the (convergent) slope value is an estimate of the correlation dimension for the data.

Usage

```
corrDim(x, dimension=5,
    tlag=timeLag(x, method="acfdecor"), olag=0, resolution=2)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series or a matrix containing an embedding with each column representing a different coordinate. If the latter, the `dimension` input is set to the number of columns and the `tlag` input is ignored.
`dimension`	the maximal embedding dimension. Default: `5`.
`olag`	the number of points along the trajectory of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low correlation dimension estimates. The orbital lag must be positive or zero. Default: `length(x)/10` or `500`, whichever is smaller.

`resolution` an integer representing the spatial resolution factor. A value of P increases the number of effective scales by a factor of P at a cost of raising the ℓ_∞ norm to the Pth power. For example, setting the resolution to 2 will double the number of scales while imposing and additional multiplication operation. The resolution must exceed unity. Default: `2`.

`tlag` the time delay between coordinates. Default: `timeLag(x, method="acfdecor")`, the decorrelation time of the autocorrelation function.

Details

To estimate the correlation dimension, correlation summation curves must be generated and subsequently fit with a robust linear regression model to obtain the slopes of these curves on a log-log plot. The dimension at which these slope estimates (appear to) converge reveals the proper embedding dimension for the data and the slope at this (and higher) embedding dimensions is an estimate of the correlation dimension. The function used to fit the correlation summation curves is `lmsreg` which fits a robust linear model to the data using the method of least median of squares regression. See the on-line help documentation for help on the `lmsreg` function: in R, `lmsreg` is found in the `MASS` package while in S-PLUS it is indigenous and appears in the `splus` database.

The correlation summation at scale ε for a given embedding dimension is defined as

$$C_2(\varepsilon) = \frac{2}{(N-\gamma)(N-\gamma-1)} \sum_{i=1}^{N} \sum_{j=i+\gamma+1}^{N} \Theta(\varepsilon - ||\mathbf{X_i} - \mathbf{X_j}||),$$

where $\Theta(\cdot)$ is the Heavyside function

$$\Theta(x) = \begin{cases} 0, & \text{if } x \leq 0; \\ 1, & \text{otherwise} \end{cases}$$

and $\mathbf{X_i}$ is the ith point of a collection of `N` points in the phase space. The parameter γ is the orbital lag.

The algorithm used to calculate the correlation summation is made computationally efficient by using:

1 The ℓ_∞ norm to calculate the distance between neighbors in the phase space as opposed to (say) the ℓ_2 norm which involves taking computationally intense square root and power of two operations. The ℓ_∞ norm of the distance between two points in the phase space is the absolute value of the maximal difference between any of the points' respective coordinates, i.e. if $\mathbf{X} = [z_1, z_2, z_3]^T$ then $||\mathbf{X}||_\infty \equiv \max_i |z_i|$.

2 Bitwise masking and shift operations to reveal the radix-2 exponent of the ℓ_∞ norm. This direct means of obtaining the exponent immediately yields the associated scale of the distance between neighbors in the phase space while avoiding costly log operations. The bitwise mask and shift factors are based on the IEEE standard 754 for binary floating-point arithmetic. Initial tests are performed in the code to verify that the current machine follows this standard.

3 a computationally efficient routine to calculate the resulting value of a float raised to a positive integer power. Specifically, the ℓ_∞ norm is raised to an integer power (`p`) to effectively increase the spatial resolution by a factor of `p`.

The correlation summation curves $C_2(E, \varepsilon)$ where E is the embedding dimension and ε is the scale, the correlation dimension curves $D_2(E, \varepsilon)$ can be calculated by

$$D_2(E, \varepsilon) = \frac{\ln C_2(E, 2\varepsilon) - \ln C_2(E, \varepsilon/2)}{\ln 2\varepsilon - \ln \varepsilon/2} = \frac{1}{2} \log_2 \frac{C_2(E, 2\varepsilon)}{C_2(E, \varepsilon/2)}.$$

This formulation is used to help suppress numerical instabilities that are present in other numerical derivative schemes such as a first order difference.

As a caveat to the user, the slope estimates of the correlation summation curves will typically display a fair amount of variability and the range of scales over which the slopes are approximately linear may be small. Inasmuch, the correlation dimension estimate should always be interpretted as a subjective summary statistic, even when the original times series is representative of a truly noise-free chaotic response.

Value

an object of class `chaoticInvariant`.

S3 METHODS

eda.plot plots an extended data analysis plot, which graphically summarizes the process of obtaining a correlation dimension estimate. A time history, phase plane embeddding, correlation summation curves, and the slopes of correlation summation curves as a function of scale are plotted.

plot plots the correlation summation curves on a log-log scale. The following options may be used to adjust the plot components:

- **type** Character string denoting the type of data to be plotted. The `"stat"` option plots the correlation summation curves while the `"dstat"` option plots a 3-point estimate of the derivatives of the correlation summation curves. The `"slope"` option plots the estimated slope of the correlation summation curves as a function of embedding dimension. Default: `"stat"`.
- **fit** Logical flag. If `TRUE`, a regression line is overlaid for each curve. Default: `TRUE`.
- **grid** Logical flag. If `TRUE`, a grid is overlaid on the plot. Default: `TRUE`.
- **legend** Logical flag. If `TRUE`, a legend of the estimated slopes as a function of embedding dimension is displayed. Default: `TRUE`.
- **...** Additional plot arguments (set internally by the `par` function).

print prints a qualitiative summary of the results.

References

Peter Grassberger and Itamar Procaccia (1983), Measuring the strangeness of strange attractors, *Physica D*, **9**, 189–208.

Holger Kantz and Thomas Schreiber (1997), *Nonlinear Time Series Analysis*, Cambridge University Press.

Peter Grassberger and Itamar Procaccia (1983), Characterization of strange attractors, *Physical Review Letters*, **50**(5), 346–349.

Rousseeuw, P. J. (1984). Least median of squares regression. *Journal of the American Statistical Association*, **79**, 871–88.

See Also

infoDim, embedSeries, timeLag, chaoticInvariant, lyapunov, poincareMap, spaceTime, findNeighbors, determinism.

Examples

```
## calculate the correlation dimension estimates
## for chaotic beam data using a delay
## embedding for dimensions 1 through 10, a
## orbital lag of 10, and a spatial resolution
## of 4.
beam.d2 <- corrDim(beamchaos, olag=10, dim=10, res=4)

## print a summary of the results
print(beam.d2)

## plot the correlation summation curves
plot(beam.d2, fit=FALSE, legend=FALSE)

## plot an extended data analysis plot
eda.plot(beam.d2)
```

---

determinism *Detecting determinism in a time series*

Description

Infers the existence of deterministic structure in a given time series. If fractal strucutre exists, this function is useful in helping the user decide whether a deterministic chaotic model or stochastic fractal time series model is appropriate for their data.

Usage

```
determinism(x, dimension=6, tlag=NULL,
    olag=1, scale.min=NULL, scale.max=NULL,
    resolution=NULL, method="ce",
    n.realization=10, attach.summary=TRUE,
    seed=0)
```

Arguments

x	a numeric vector or matrix containing uniformly-sampled real-valued time series.
attach.summary	a logical flag. If TRUE, a summary of the results is calculated and attached to returned object as an attribute named "summary". The summary statistics are calculated using the summary method. Default: TRUE.
dimension	an integer defining the maximum embedding dimension to use in analyzing the data. Default: 6.

`method` a character string representing the method to be used to generate surrogate data. Choices are:

`"aaft"` Theiler's Amplitude Adjusted Fourier Transform.

`"phase"` Theiler's phase randomization.

`"ce"` Davies and Harte's Circulant Embedding.

`"dh"` Davison and Hinkley's phase and amplitude randomization.

Default: `"ce"`.

`n.realization` an integer denoting the number of surrogate realizations to create and analyze for comparison to the ensmeble of E-statistics. Default: `10`.

`olag` the number of points along the trajectory of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low correlation dimension estimates. The orbital lag must be positive or zero. Default: `length(x)/10` or `500`, whichever is smaller.

`resolution` a numeric value representing the spacing between scales (Euclidean bin size). Default: `diff(range(x))/1000`.

`scale.max` a numeric value defining the maximum scale over which the results should be returned. Default: `diff(range(x)) * sqrt(dimension)`.

`scale.min` a numeric value defining the minimum scale over which the results should be returned. Default: `min(diff(sort(x)))/1000`.

`seed` a positive integer representing the initial seed value for generating surrogate realizations of the original input time series. These surrogates are used to collect an ensemble of determinism statistics (see DETAILS section for more information). If the specified `seed` value is positive, the seeds used for generating the surrogate ensemble will be calculated via `set.seed(seed);rsample(.Machine$integer.max, size=n.` This argument should only be used (by specifying a positive seed value) if the user wishes to replicate a particular set of results, such as those illustrated in the casebook examples. If `seed=0`, then the random seeds will be generated based on the current time. Default: `0` (generate the random seeds based on the current time).

`tlag` the time delay between coordinates. Default: the decorrelation time of the autocorrelation function.

Details

This function calculates the so-called delta-epsilon test for detecting deterministic structure in a time series by exploiting (possible) continuity of orbits comprising a phase space topology created by a time-delayed embedding of the original time series. This phase space continuity is non-existent for stochastic white noise processes. The delta-epsilon test works by

1 an ensemble of randomized realizations of the original time series, i.e., surrogate data is created.

2 an appropriate phase space statistic (called the E-statistic) is calculated for both the time-delayed embedding of the original time series and the ensemble of surrogates.

3 a comparison of the E-statistic for the original series and the ensemble of surrogate data is made. If there is a separation of the original E-statistic from that of the ensemble, it implies the existence of deterministic structure in the original time series. Conversely, an overlap of E-statstics implies that the original series cannot be discriminated from the ensemble of randomized surrogates and thus it is inferred that the original series is a realization of a random process.

The discriminating E-statistic is calculated as follows: Define

$$\delta_{j,k} = |z_j - z_k|$$

$$\epsilon_{j,k} = |z_{j+\kappa} - z_{k+\kappa}|$$

$$e(r) \equiv \overline{\epsilon_{j,k}} \qquad \text{for } j,k \text{ s.t. } r \leq \delta_{j,k} < r + \Delta r$$

where $\delta_{j,k}$ is the Euclidean distance (using an infinity-norm metric) between phase space points z_j and z_k, and $\epsilon_{j,k}$ is the corresponding separation distance between the points at a times κ points in the future along their respective orbits. These future points are referred to as *images* of the original pair. The variable κ is referred to as the orbital lag. The increment Δr is the width of a specificed Euclidean bin size. Given Δr, the distance $\delta_{j,k}$ is used solely to identify the proper bin in which to store the image distance $\epsilon_{j,k}$. The average of each bin forms the $e(r)$ statistic. Finally, the E-statistic is formed by calculating a cumulative summation over the the $e(r)$ statistic, i.e.,

$$E(r) \equiv \sum \overline{e(r)}.$$

If there exists a distinct separation of the E-statstics for the original time series and the ensemble of surrogate data, it implies that the signal is deterministic. The orbital lag κ should be chosen large enough to sufficiently decorrelate the points evaluated along a given orbit.

Value

an object of class `determinism`.

S3 METHODS

eda.plot plots a barplot of the determinism level (expressed as a percentage on [0,100]) based on the fraction of overlap between the E-statistics for the original series and that of the ensmeble of surrogates. The amount of non-overlap is calculated relative to both the first quartile and extreme values of the E-statistics for the surrogate ensemble.

plot plots the E-statistics at small scales of the original series overlaid with those of the ensmeble of surrogates (illustrated using boxplots over a subsampled set of the surrogate E-statistics).

print print a sumamry of the analysis.

summary produces a summary of the E-statistics for use in the `print`, and `plot`, and `eda.plot` methods.

References

Kaplan, D. (1994), *Exceptional Events as Evidence for Determinism*, Physica D, **73**, 38–48.

See Also

`embedSeries`, `timeLag`, `spaceTime`, `surrogate`.

Examples

```
## perform a determinism test for the beamchaos
## series. in order to do so, it is vitally
## important to provide the proper orbital lag,
## which can be estimated as the lag value
## associated with the first common maxima over
## all contours in a spaceTime plot.
plot(spaceTime(beamchaos))

## we esimate an appropriate olag of 30, and
## subsequently perform the deterrminism test
beam.det <- determinism(beamchaos, olag=30)
print(beam.det)
plot(beam.det)

eda.plot(beam.det)

## perform a similar analysis for a Gaussian white
## noise realization
rnorm.det <- determinism(rnorm(1024),olag=1)
print(rnorm.det)
plot(rnorm.det)

eda.plot(rnorm.det)
```

DFA *Detrended fluctuation analysis*

Description

Performs a detrended fluctuation analysis (DFA) and estimates the scaling exponent from the results. DFA is used to characterize long memory dependence in stochastic fractal time series.

Usage

```
DFA(x, detrend="poly1", sum.order=0, overlap=0,
    scale.max=trunc(length(x)/2), scale.min=NULL,
    scale.ratio=2, verbose=FALSE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
detrend	a character string denoting the type of detrending to use on each block of the time series. Supported types are:
	"poly}\emph{K}\code{" specifies a polynomial fit where *K* is an integer denoting the order of the polynomial. For example, if detrend="poly2", a second order polynomial of the form $x_t = b_0 + b_1 t + b_2 t^2$ will be used to fit the data in each block using least squares. The polynomial order must be positive or zero.

"bridge" specifies bridge detrending. A line connecting the endpoints of each block is subtracted.

"none" instructs the function to not detrend the data.

Default: "poly1".

overlap	the overlap of blocks in partitioning the time data expressed as a fraction in [0,1). A positive overlap will slow down the calculations slightly with the (possible) effect of generating less biased results. Default: 0.
scale.max	an integer denoting the maximum block size to use in partitioning the data. Default: trunc(length(x)/2).
scale.min	an integer denoting the minimum block size to use in partitioning the data. Default: for polynomial detrending the default value is 2*(K+1). For all other detrending techniques, the default value is 4 or length(x)/4, whichever is smaller.
scale.ratio	the ratio of successive scales. This argument is used as an input to the logScale function. Default: 2.
sum.order	an integer denoting the number of differences or cumulative summations to perform on the original data before performing a DFA. Differences are specified by negative integers and cumulative summations by positive integers. For example, to perform a second order difference, set sum.order=-2. Default: 0.
verbose	a logical value. If TRUE, the detrending model and processing progress information is displayed. Default: FALSE.

Details

The DFA algorithm is implemented as follows:

1 DFA is useful for characterizing long-memory correlations in stochastic fractal time series, i.e. sequences whose spectral density function $S(f)$ obeys a power law $S \sim |f|^{\alpha}$ at low frequencies where $0 \leq f \leq 1/2$ is the normalized frequency variable and $\alpha \leq -1$ is the long memory (scaling) exponent. If the scaling exponent for an original time series is $\alpha > -1$, then (possibly multiple) cumulative summations of the original time series must be performed to increase the scaling exponent (each cumulative summation decreases the exponent by 2). For example, a (single) cumulative summation must be performed on a white noise realization since its scaling exponent is zero. We also provide the user with the ability to perform (consecutive) first order differencing operations on the original time series prior to a DFA. Each differencing operation raises the scaling exponent by 2. Differencing a series is acceptable prior to DFA as long as the resulting scaling exponent is less than -1.

2 The series resulting from stage one is uniformly partitioned into blocks of a specified minimum size (scale.min), and each block is (optionally) detrended. The variance of the detrended sequence in each block is calculated and the collection of variances is averaged to form the scalar value F^2(scale.min) which summarizes the variability of the sequence at the current scale.

3 Stage two is repeated using successively larger blocks until the largest scale (scale.max) has been reached.

4 For long-memory processes, we expect to find a linear relation between $\log F(scale)$ and $\log scale$. The slope of the line which best fits a plot of $\log F(scale)$ versus $\log scale$ is defined as the *scaling exponent*.

Value

an object of class `fractalBlock`.

References

Peng C-K, Buldyrev SV, Havlin S, Simons M, Stanley HE, and Goldberger AL (1994), Mosaic organization of DNA nucleotides, *Physical Review E*, **49**, 1685–1689.

Peng C-K, Havlin S, Stanley HE, and Goldberger AL (1995), Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series, *Chaos*, **5**, 82–87.

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE (2000, June 13), PhysioBank, PhysioToolkit, and Physionet: Components of a New Research Resource for Complex Physiologic Signals, *Circulation*, **101**(23), e215-e220.

See Also

`logScale`, `fractalBlock`.

Examples

```
## calculate the scaling exponent for a random
## walk realization
DFA.walk <- DFA(rnorm(1024), detrend="poly1", sum.order=1)

## print the results
print(DFA.walk)

## plot a summary of the results
eda.plot(DFA.walk)
```

`dispersion` *Dispersion analysis*

Description

Dispersion analysis measures the standard deviation of aggregated means of a time series taken over logarithmically distributed scales. Dispersion analysis is designed for the analysis of fractional Gaussian noise and should not be used for analyzing fractional Brownian motion.

Usage

```
dispersion(x, front=FALSE)
```

Arguments

x	a numeric vector or signalSeries object containing a uniformly sampled real-valued time series.
front	a logical value. If TRUE, the aggregation is started from the beginning of the time series so that the first points will be included in the result. Otherwise, the aggregation is shifted to include the end of the series. Default: FALSE.

Value

a list containing the scale and dispersion analysis statistic vectors.

References

Bassingthwaighte, J. B., and G. M. Raymond. *Evaluation of the dispersional analysis method for fractal time series*, Annals Biomedical Engineering, **23**, 491–505, 1995.

See Also

DFA, RoverS.

Examples

```
set.seed(100)
z <- dispersion(rnorm(1024))
plot(log(z$scale),log(z$sd))
```

---

ecgrr *Electrocardiogram R-R Interval Data*

Description

These data are from the file rr1.txt in the 'RR interval time series modeling: A challenge from PhysioNet and Computers in Cardiology 2002' site of PhysioNet. sponsored by NIH's National Center for Research Resources.

The data are the RR intervals (beat-to-beat intervals measured between successive peaks of the QRS complex) for patients in normal sinus rhythm (record 16265 of the MIT-BIH database).

See Also

beamchaos, eegduke, lorenz, pd5si.

Examples

```
plot(ecgrr)
```

eda.plot *Generic function for generating extended data analysis plots*

Description

Data analysis plots are used to visually summarize the salient features of the output and typically involve a combination of plots in a single plot frame.

Usage

```
eda.plot(x, ...)
```

Arguments

x	any object. Missing values (NAs) are allowed.
...	optional arguments to be passed directly to the inherited function without alteration and with the original names preserved.

Note

An extended data analysis plot is shown.

See Also

wavMRD, determinism, chaoticInvariant, embedSeries, fractalBlock, KDE, spaceTime, surrogate.

Examples

```
methods(eda.plot)
```

eegduke *Electroencephalogram Recordings of a Seizure*

Description

EEG of a patient undergoing ECT therapy for clinical depression at the ECT Lab at Duke. The data are fluctuations in electrical potential at a point on the patient's scalp during seizure, one of several recorded channels. They are measured in microvolts and represent measurements taken at time intervals of roughly $1/40$ of a second.

See Also

beamchaos, ecgrr, lorenz, pd5si.

Examples

```
plot(eegduke)
```

`embedSeries` *Creates a delay embedding of a single variable time series*

Description

Given the time series X_t, the embedding dimension E, and the time delay τ, the embedding coordinates are defined as $X_t, X_{t-\tau}, \ldots, X_{t-(E-1)\tau}$.

Usage

```
embedSeries(x, dimension=NULL, tlag=NULL, series.name=NULL)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series.
`dimension`	the maximal embedding dimension. Default: `2`.
`series.name`	a character string defining the name of the input time series. Default: `deparseText(substitute(x))`.
`tlag`	the time delay between coordinates. Default: the first zero crossing of the autocorrelation function of `x`.

Value

an object of class `embedSeries`.

S3 METHODS

[data access method. Usage: `x[1:3,1]`.

as.matrix convert embedding into matrix object. Usage: `as.matrix(x)`.

eda.plot creates an extended data analysis plot of the data summarizing many of its statistical features. Usage: `eda.plot(x)`.

plot plots the embedding. For embeddings higher than 3, a spin plot of the data is generated. Use the buttons on the spin control panel to conrtol the display. Available options to the plot function are:

dim The plot dimension. Must be less than or equal to the maximal embedding dimension (number of columns in the embedding matrix). Default: the maximal embedding dimension.

... Additional plot arguments (set internally by the `par` function).

Usage: `plot(x)`.

print prints a summary of the embedding. Available options are:

... Additional print arguments used by the standard `print` function.

Usage: `print(x)`.

See Also

`timeLag`, `FNN`, `corrDim`, `determinism`.

Examples

```
## embed the beamchaos series in 10 dimensions
## using a time lag of 15.
z <- embedSeries(beamchaos, tlag=15, dim=10)

## plot the attractor in the phase space
## Not run: plot(z)

## plot the embedding projected down to two
## dimensions
plot(z, dim=2)
```

`FDSimulate` *Simulation of an FD process with time varying model parameters*

Description

Creates a realization of a time-varying fractionally differenced (FD) process with a given vector of FD parameters and corresponding vector of innovations variances.

Usage

```
FDSimulate(delta, innovations.var=1, method="ce", seed=0)
```

Arguments

`delta`	a vector containing time-varying FD parameters.
`innovations.var`	a numeric vector or scalar containing (time-varying) FD innovations variances. If a scalar, the value is replicated appropriately. Otherwise, the length of this input should match the length of the `delta` vector. Default: `1`.
`method`	a character string defining the method to use in forming the FD realization. Choices are `"ce"` (circulent emebdding) and `"cholesky"`. Default: `"ce"`.
`seed`	a positive integer representing the initial seed value to use for the random number generator. If `seed=0`, the current time is used as a means of generating a (unique) seed value. Otherwise, the specified seed value is used. Default: `0`.

Value

a vector containing a (time-varying) FD process realization corresponding to the input FD model parameters.

S3 METHODS

plot plot the output object. Optional arguments include:

simulation Plot the simulated series. Default: TRUE.

delta Plot the FD parameter as a function of time. Default: TRUE.

innovations.var Plot the innovations variance as a function of time. Default: TRUE.

print print the output object.

References

D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, 2000.

D. B. Percival and W.L.B. Constantine, *Exact Simulations of Time-Varying Fractionally Differenced Processes*, submitted to Journal of Computational and Graphical Statistics, 2002.

See Also

FDWhittle, wavFDPBlock, wavFDPTime.

Examples

```
## create a time-varying FD parameter, linearly
## varying from white to pink noise, then jump
## to a red noise plateau
delta <- c(seq(0, 0.5, by=0.01), rep(1,100))

## set the innovations variance to unity
innovation <- rep(1, length(delta))

## simulate a time-varying FD process
z <- FDSimulate(delta=delta, innovation=innovation)
print(z)
plot(z)
```

FDWhittle *Estimate the Hurst coefficient by Whittle's method*

Description

Using an estimate of the spectral density function for an input time series, Whittle's method fits the parameters of a specified SDF model to the data by optimizing an appropriate functional. In this case, the SDF for a fractionally differenced (FD) process model is used and an estimate of (δ), the FD parameter, is returned.

Usage

```
FDWhittle(x, method="continuous", dc=FALSE, freq.max=0.5,
    delta.min=-1,delta.max=2.5, sdf.method="direct", ...)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
...	optional SDF estimation arguments passed directly to the SDF function. See help documentation for the SDF function for more information.
dc	a logical value. If FALSE, the DC component of the SDF (corresponding to the sample mean of the series) is not used in optimizing the Whittle functional. Default: FALSE.
delta.max	the maximum value for the FD parameter to use in the constrained optimization problem. Default: 2.5.
delta.min	the minimum value for the FD parameter to use in the constrained optimization problem. Default: -1.
freq.max	the largerst normalized frequency of the SDFs use in the analysis. Default: 0.25.
method	a character string indicating the method to be used in estimating the Hurst coefficient (H). Choices are: "continuous" Whittle's method using a continuous model approach to form the optimization functional. This functional is subsequently implemented via a discrete form of the SDF for an FD process. "discrete" Whittle's method using (directly) a discrete form of the SDF for an FD process. Default: "continuous".
sdf.method	a character string denoting the method to use in estimating the SDF. Choices are "direct", "lag window", "wosa" (Welch's Overlapped Segment Averaging), "multitaper". See help documentation for the SDF function for more information. Default: "direct".

Value

estimate of the FD parameter of the time series.

References

M. S. Taqqu and V. Teverovsky, On Estimating the Intensity of Long- Range Dependence in Finite and Infinite Variance Time Series (1998), in *A practical Guide to Heavy Tails: Statistical Techniques and Applications*, pp. 177–217, Birkhauser, Boston.

See Also

hurstSpec, FDSimulate.

Examples

```
set.seed(100)
walk <- cumsum(rnorm(1024))
FDWhittle(walk, method="discrete", sdf.method="multitaper")
FDWhittle(walk, method="continuous", sdf.method="multitaper")
```

findNeighbors *Nearest neighbor search in a multidimensional space*

Description

Finds a user specified number of nearest neighbors of a multivariate space defined by the coordinates of the input matrix. Alternatively, the user can specify a maximum distance over which to search for nearest neighbors.

Usage

```
findNeighbors(x, n.neighbor=NULL, metric=1, max.distance = 0.,
    olag=0, sort.distances=TRUE)
```

Arguments

x	an embedding matrix. Each column of the matrix respresents a single coordinate of the embedding and each row denotes the coordinates of a single point in the embedding.
max.distance	used an alternative to n.neighbor, use this parameter to specify the maximum distance to search relative to the current point in the phase space. The metric for the distance is specified separately by the optional metric input argument. This arguments must be positive and will only be used if n.neighbor is NULL, equal to zero, or less than zero. Default: 0.0.
metric	the metric used to define the distance between points in the embedding. Choices are limited to 1, 2, or Inf which represent an L_1, L_2, and L_∞ norm, respectively. Default: 1.
n.neighbor	the number of neighbors to find for each point in the embedding. If not NULL, this argument overrides the max.distance argument. Default: 2.
olag	an integer scalar representing the orbital lag, which defines the number of points along a trajectory (both forwards and backwards) that are to be excluded as nearest neighbor candidates. This argument helps to prevent temporally correlated data samples from being considered as neighbors to a given point in the embedding. This sitatuation can arise, for example, when a smooth trajectory has been highly oversampled in time. An orbital lag of 0 implies that the reference point itself may be considered a neighbor candidate. To exclude self-neighbors, set olag greater than zero. Default: 0.
sort.distances	a logical flag. If TRUE, the neighbors for a given point are sorted by distance from closest to farthest. Default: TRUE.

Details

An efficient recursive algorithm is used to find all nearest neighbors. First a quadtree is developed to form a recursive partitioning of the embedding matrix, returning row and column index vectors and a list of medians which may be used to sort the embedding matrix. The quadtree is then traversed as an efficient means to find nearest neighbors.

Value

a list containing the indices of the original points (corresponding to rows of the embedding matrix), the indices of the neighbors found, and the distance between them. The distance metric is based on that specified by the optional `metric` input argument.

References

Friedman, J., Bentley, J. L., and Finkel, R. A., "An algorithm for finding best matches in logarithmic expected time", *ACM Transactions on Mathematical Software* **3**, 209–226, 1977.

See Also

FNN, FNS.

Examples

```
## Calculate the 10 nearest neighbors for each
## point of 3-dimensional delayed coordinate
## embedding of the beamchaos data. Exclude
## self-neighbors from the output.
embedding <- embedSeries( beamchaos, dim = 3, tlag = 10 )
nn <- findNeighbors( embedding, n.neighbor=10, olag=1 )

## Using the same data, find only those neighbors
## within a distance 0.1 of the original points
## based on an L-infinity metric
nn.dist <- findNeighbors( embedding, max.distance=0.1,
metric=Inf, olag=1 )
```

FNN — *Estimation of the proper embedding dimension for a single-variable time series*

Description

Invokes the method of False Nearest Neighbors (FNN) to estimate the minimal embedding dimension of a multivariate data set.

Usage

```
FNN(x, dimension=5, tlag=NULL, rtol=10, atol=2, olag=1)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series.
`atol`	neighbor tolerance based on attractor size. If the Euclidean distance between two neighbor candidates is `Atol` times larger the estimated "size" of the attractor, then those neighbors are declared as false neighbors. Default: `2`.

`dimension` the maximal embedding dimension. Default: `5`.

`olag` orbital lag. The number of points along the trajectory (orbit) of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low minimal embedding dimension estimates. The orbital lag must be positive or zero. Default: `0`.

`rtol` false neighbor Euclidean distance tolerance. If the ratio of the Euclidean distances between neighbor candidates in successive embedding dimensions exceeds `Rtol`, then those neighbors are declared as false neighbors. For example, if `Rtol=5` neighbor candidates that are separated five times more so than in the previous embedding dimension are declared false neighbors. Default: `10`.

`tlag` the time delay between coordinates. Default: the decorrelation time of the autocorrelation function.

Value

an object of class `FNN`.

S3 METHODS

plot plots a summary of the results. Available options are:

xlab a character string defining the x-axis label. Default: `"Embedding Dimension"`.

ylab a character string defining the y-axis label. Default: `"FNN percentage"`.

`...` Additional plot arguments (set internally by the `par` function).

print prints a summary of the results. Available options are:

... Additional print arguments used by the standard `print` function.

References

M. B. Kennel, R. Brown, and H. D. I. Abarbanel (1992), Determining embedding dimension for phase-space reconstruction using a geometrical construction, *Physical Review A*, **45**(6), 3403–3411.

Fredkin, D. R., and Rice, J. A. (1995), Method of false nearest neighbors: A cautionary note, *Physical Review E*, **51**(4), 2950–2954.

See Also

`FNS`, `embedSeries`, `infoDim`, `corrDim`, `timeLag`, `determinism`.

Examples

```
## perform False Nearest Neighbors tests on
## chaotic beam data for embedding dimensions 1
## through 10, using a time delay embedding
## with a time lag of 10 and an orbital lag of
## 15
x <- FNN(beamchaos, tlag=10, olag=15 )
```

```
## print the results
print(x)

## plot the results
plot(x)
```

FNS *Estimation of the proper embedding dimension for a single-variable time series*

Description

Invokes the method of False Nearest Strands (FNS) to estimate the minimal embedding dimension of a multivariate data set.

Usage

```
FNS(x, dimension=5, tlag=NULL, atol=1,
    image.tol=1, olag=1)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
atol	fNS statitstic threshold. Default: 1.
dimension	the maximal embedding dimension. Default: 5.
image.tol	an integer defining the so-called iterate tolerance. Nearest neighbor pairs (i,J(i)) are separated in time by a point index span dindex = |i-J(i)|, where J(i) represents the index of the nearest neighbor to point i. If a point near i, say k points away also has a nearest neighbor such that |k - J(k)| = dindex +/- M, where M is the iterate tolerance, then the pair (k, J(k)) is added to the current strand. Typically, M=0 or M=1. If M=0, then the difference in index must be exactly the same for each pair included in the strand. If M=1, the index difference is allowed to be 1 point off from the reference pair. Default: 1.
olag	orbital lag. The number of points along the trajectory (orbit) of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low minimal embedding dimension estimates. The orbital lag must be positive or zero. Default: 1.
tlag	the time delay between coordinates. Default: the decorrelation time of the auto-correlation function.

Details

The statistic used for determining a false nearest strand (FNS) is based on a Euclidean tolerance supplied by the user (`atol`). Let $S(d)$ be the mean Euclidean distance in the projected $(d+1)$th coordinate between strand pairs found to be nearest neighbors in embedding dimension d. If $S(d)/A >$ atol, where A is the estimated attractor size, then the strand is considered to be a false strand. A is typically calculated to be the sample standard deviation of the original time series. The $S(d)$ statistic is a measure of the average additional Euclidean distance we gain by embedding the strand in the next dimension, and is used to assess when this extra distance has grown too large, indicating a false strand.

Value

an single-dimensional matrix containing the FNS percentage as a function of embedding dimension.

References

M. B. Kennel and Henry D.I. Abarbanel (2002), False neighbors and false strands: A reliable minimum embedding dimension algorithm, *Physical Review E*, **66**, 026209, 1–19.

M. B. Kennel, R. Brown, and H. D. I. Abarbanel (1992), Determining embedding dimension for phase-space reconstruction using a geometrical construction, *Physical Review A*, **45**(6), 3403–3411.

Fredkin, D. R., and Rice, J. A. (1995), Method of false nearest neighbors: A cautionary note, *Physical Review E*, **51**(4), 2950–2954.

See Also

`FNN`, `embedSeries`, `infoDim`, `corrDim`, `timeLag`, `determinism`.

Examples

```
## perform False Nearest Strands tests on chaotic
## beam data for embedding dimensions 1 through
## 10, using a time delay embedding with a time
## lag of 10 and an orbital lag of 15
x <- FNS(beamchaos, dim=10, tlag=10, olag=15)

## print the results
print(x)
```

`fractalBlock` *Class constructor for block-dependent estimators for stochastic fractal time series*

Description

Class constructor for `fractalBlock`.

Usage

```
fractalBlock(domain,estimator, exponent, exponent.name,
    scale, stat, stat.name, detrend, overlap,
    data.name, sum.order, series, logfit, sdf=NULL)
```

Arguments

domain	character string defining the domain in which the calculations took place, e.g., in the time or frequency domain.
estimator	character string briefly describing the estimator.
exponent	numerical value representing the scaling exponent.
exponent.name	character string defining the name of the scaling exponent.
scale	numeric vector containing the scales used in the analysis.
stat	numeric vector containing the statistic calculated in the analysis.
stat.name	character vector describing the name for the calculated statistic.
detrend	character string describing any series detrending used as a preprocessive measure. NULL values are allowed and signify no detrending.
overlap	numeric value on [0,1] defining the fraction of overlap used in adjacent blocks of data during the aggregation process.
data.name	character string defining the name of the input series.
sum.order	integer representing the sum order.
series	a numeric vector containing the input series.
logfit	a linear regression model (such as that output by lm, lmsreg, or ltsreg) containing the regression model of the log(scale) versus log(stat) data.
sdf	spectral density function. Default: NULL.

S3 METHODS

eda.plot extended data analyis plot of the data. Available options are:

- **cex** character expansion ala par. Default: 1.
- **col** line color index ala par. Default: 2.

plot plots a summary of the results. Available options are:

- **pch** plot character ala par. Default: 18.
- **col** color index ala par for a plot of the data. the first two elements are used to color the data and the regression line, respectively. Default: c(1,8).
- **lty** line types (ala par) for the data and regression line plot, respectively. Default: c(1,1).
- **grid** list of grid objects whose default values are grid=list(lty=2, col=16, density=3), key=TRUE, add=FALSE,
- **...** Additional plot arguments (set internally by the par function).

print prints the object. Available options are:

- **justify** text justification ala prettPrintList. Default: "left".
- **sep** header separator ala prettyPrintList. Default: ":".
- **n.digits** number of digits ala prettyPrintList. Default: 5.
- **...** Additional print arguments sent directly to the prettyPrintList function).

See Also

hurstBlock, DFA.

Examples

```
## construct a fractalBlock object
xvar <- 2^(1:10)
yvar <- 0.3^(1:10)
z <- fractalBlock(domain="time", estimator="estimator", exponent=pi, exponent.name="PI",
    scale=xvar, stat=yvar, stat.name="My Stat",
    detrend=NULL, overlap=0.2, data.name="My Series",
    sum.order=-1, series=rnorm(2^10),
    logfit=lm(y ~ x, data=data.frame(x=log(xvar), y=log(yvar))))

## print the result
print(z)

## plot the result
plot(z)
```

HDEst *Hurvich-Deo estimate of number of frequencies to use in a periodogram regression*

Description

Estimates the number of frequencies to use in a periodogram regression estimate of the Hurst parameter H of a long memory time series. Based on estimated spectrum of time series.

Usage

```
HDEst(NFT, sdf, A=0.3, delta=6/7)
```

Arguments

NFT	number of points used in the Fourier transform to generate the spectrum.
sdf	estimate of the spectrum of the time series (power, NOT dB).
A	parameter A of Hurvich-Deo model. Default: 0.3 (recommended).
delta	parameter delta of Hurvich-Deo model. Default: 6/7 (recommended).

Value

estimated optimum number of frequencies "m" to use in a periodogram regression estimate of the Hurst parameter H.

References

C.M. Hurvich and R.S. Deo (1999), Plug-in Selection of the Number of Frequencies in Regression Estimates of the Memory Parameter of a Long Memory Time Series, *J. Time Series Analysis*, **20**(3), 331–341.

See Also

hurstBlock.

Examples

```
S <- sapa::SDF(beamchaos)
HDEst(NFT=length(S),as.vector(S))
```

---

henon *Henon map*

Description

Calculates the Henon map states using the specifed parameter set. The Henon map is defined as

$$x_n = a - x_{n-1}^2 + by_{n-1}$$

$$y_n = x_{n-1}$$

A parameter set of $a = 1.4$ and $b = 0.3$ is known to produce a deterministic chaotic response.

Usage

```
henon(start=rnorm(2), a=1.4, b=0.3, n.sample=2000, n.transient=10)
```

Arguments

start	a two-element vector of numeric values denoting the starting values for the X and Y Henon coordinates, respectively.
a	the **a** parameter. Default: 1.4.
b	the **b** parameter. Default: 0.3.
n.sample	an integer denoting the number of iterates to create beyond that specified by n.transient. Default: 2000.
n.transient	an integer denoting the number of transient points. These transients are removed from the output. Default: 10.

Value

a list of vectors named x and y corresponding to the X- and Y states of the Henon map, respectively.

See Also

lorenz, lorenz.ode.

Examples

```
plot(henon(),pch=".",cex=0.1)
```

---

hurstACVF *Estimate the Hurst coefficient by regression of scaled asinh plot of ACVF vs log(lag)*

Description

Estimates long memory parameters beta (ACVF decay exponent), alpha (Equivalent PPL model spectral density exponent), and H (Equivalent Hurst parameter) by linear regression of scaled asinh of ACVF versus log(lag) over intermediate lag values.

Usage

```
hurstACVF(x, Ascale=1000, lag.min=1, lag.max=64)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
Ascale	scale factor for use in the scaled asinh plot. Default: 1000.
lag.max	maximum lag for use in linear regression. Default: 64.
lag.min	minimum lag for use in linear regression. Default: 1.

Details

Evaluates autocovariance function (ACVF) of input time series by call to S-Plus function acf. Constructs sequence asinh(Ascale * ACVF) / asinh(Ascale) and does linear regression (via S-Plus function "lsfit") of this sequence versus log(lag) from lag.min to lag.max. Draws a plot of the sequence and the fit line. Recommended usage: look at resulting plot. Is the intermediate range approximately linear? If plot is too flat, decrease Ascale. If it decreases to zero too quickly, increase Ascale. Values of Ascale from 10 to 10^6 have been found useful. If lag.min and lag.max do not bound the range where the sequence is approximately linear then change them and rerun the function to produce a better fit.

Value

a list with three components:

beta	decay exponent of autocovariance function
alpha	spectral density exponent of equivalent PPL model
H	Hurst exponent for equivalent ACVF decay rate

References

A. G. Gibbs and D. B. Percival, Forthcoming paper on the autocovariance of the PPL (pure power law) model. A section of the paper discusses the usefulness of scaled asinh plots.

See Also

hurstBlock.

Examples

```
hurstACVF(wmtsa::nile, Ascale=1000000, lag.min=3, lag.max=68)
```

---

hurstBlock *Hurst coefficient estimation in the time domain*

Description

Function to estimate the Hurst parameter H of a long memory time series by one of several methods as specified in input. These methods all work directly with the sample values of the time series (not the spectrum).

aggabs The series is partitioned into m groups. Within each group, the first absolute moment about the mean of the entire series is evaluated. A measure of the variability of this statistic between groups is calculated. The number of groups, m, is increased and the process is repeated. The observed variability changes with increasing m in a way related by theory to the Hurst parameter H of the input series. For the methods used here, a log-log plot of variability versus number of groups is, ideally, linear, with a slope related to H, so H can be determined by linear regression.

aggvar The series is partitioned into m groups. Within each group, the variance (relative to the mean of the entire series) is evaluated. A measure of the variability of this statistic between groups is calculated. The number of groups, m, is increased and the process is repeated. The observed variability changes with increasing m in a way related by theory to the Hurst parameter H of the input series. For the methods used here, a log-log plot of variability versus number of groups is, ideally, linear, with a slope related to H, so H can be determined by linear regression.

diffvar The series is partitioned into m groups. Within each group, the variance, relative to the mean of the entire series, is evaluated. The first difference of the variances is then evaluated. A measure of the variability of this statistic between groups is calculated. The number of groups, m, is increased and the process is repeated. The observed variability changes with increasing m in a way related by theory to the Hurst parameter H of the input series. For the methods used here, a log-log plot of variability versus number of groups is, ideally, linear, with a slope related to H, so H can be determined by linear regression.

higuchi The series is assumed to have the character of a noise, not a motion. The series is partitioned into m groups. The cumulative sums of the series are evaluated to convert the series from a noise to a motion. Absolute differences of the cumulative sums between groups are analyzed to estimate the fractal dimension of the path. The number of groups, m, is increased and

the process is repeated. The result changes with increasing m in a way related by Higuchi's theory to the Hurst parameter H of the input series. A log-log plot of the statistic versus number of groups is, ideally, linear, with a slope related to H, so H can be determined by linear regression.

Usage

```
hurstBlock(x, method="aggAbs", scale.min=8, scale.max=NULL,
    scale.ratio=2, weight=function(x) rep(1,length(x)), fit=lm)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
fit	a function representing the linear regression scheme to use in fitting the resulting statistics (on a log-log scale). Supported functions are: lm, lmsreg, and ltsreg. See the on-line help documentation for each of these for more information: in R, these are found in the MASS package while in S-PLUS they are indigenous and found in the splus database. Default: lm.
method	a character string indicating the method to be used to estimate the Hurst coefficient (H). Choices are: "aggabs" Absolute Values of the Aggregated Series "aggVar" Aggregated Variance Method "diffvar" Differenced Variance Method "higuchi" Higuchi's Method Default: "aggabs".
scale.max	an integer denoting the maximum scale (block size) to use in partitioning the series. Default: length(x).
scale.min	an integer denoting the minimum scale (block size) to use in partitioning the series. Default: 8.
scale.ratio	ratio of successive scales to use in partitioning the data. For example, if scale.min=8 and scale.ratio=2, the first scale will be 8, the second scale 16, the third scale 32, and so on. Default: 2.
weight	a function with a single required variable (x) used to weight the resulting statistics (x) for each scale during linear regression. Currently, only supported when fit=lm. Default: function(x) rep(1,length(x)).

Value

an object of class fractalBlock.

References

T. Higuchi (1988), Approach to an irregular time series on the basis of the fractal theory, *Physica D*, **31**, 277–283.

M.S. Taqqu, V. Teverovsky, and W. Willinger, Estimators for Long- Range Dependence: an Empirical Study (1995), *Fractals*, **3**, pp. 785–798.

M. S. Taqqu and V. Teverovsky, On Estimating the Intensity of Long- Range Dependence in Finite and Infinite Variance Time Series (1998), in *A practical Guide to Heavy Tails: Statistical Techniques and Applications*, pp. 177–217, Birkhauser, Boston.

See Also

fractalBlock, hurstSpec, lm.

Examples

```
## create test series
set.seed(100)
x <- rnorm(1024)
walk <- cumsum(x)

## calculate the Hurst coefficient of a random
## walk series using various techniques
methods <- c("aggabs","aggvar","diffvar","higuchi")
z <- lapply(methods, function(method, walk){
    hurstBlock(ifelse1(method=="higuchi",diff(walk),walk), method=method)
},walk=walk )
names(z) <- methods

## plot results
old.plt <- splitplot(2,2,1)
for (i in 1:4){
    if (i > 1)
       splitplot(2,2,i)
    plot(z[[i]], key=FALSE)
    mtext(paste(attr(z[[i]],"stat.name"), round(as.numeric(z[[i]]),3), sep=", H="),
       line=0.5, adj=1)
}
par(old.plt)
```

---

hurstSpec *Hurst coefficient estimation via spectral regression*

Description

Function to estimate the Hurst parameter H of a time series by linear regression of the log(spectrum) versus log(frequency) with frequency points accumulated into boxes of equal width on a logarithmic scale and spectrum values averaged over each box.

standard Given an estimate of the SDF for the input time series, this function estimates the Hurst coefficient of the time series by performing a linear regression of log(SDF) versus log(frequency). The range of frequencies to be included in the regression is specified by the dc and freq.max input arguments.

smoothed Given an estimate of the SDF for the input time series, this function estimates the Hurst coefficient of the time series by performing a linear regression of log(SDF) versus log(frequency). The range of frequencies to be included in the regression is specified by the dc and freq.max input arguments. Frequencies are partitioned into blocks of equal width on a logarithmic scale and the SDF is averaged over each block. The number of blocks is controlled by the n.block argument.

robinson Estimates the Hurst coefficient by Robinson's SDF integration method. Given an estimate of the SDF for the input time series, this function estimates the Hurst coefficient of a time series by applying Robinson's integral method (typically) to the low- frequency end of the SDF. Use the freq.max argument to define the low-frequency cutoff.

Usage

```
hurstSpec(x, method="standard", freq.max=0.25, dc=FALSE, n.block=NULL,
    weight=function(x) rep(1,length(x)), fit=lm, sdf.method="direct", ...)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
...	optional SDF estimation arguments passed directly to the sdf function. See help documentation for the SDF function for more information.
dc	a logical value. If FALSE, the DC component of the spectrum (corresponding to the sample mean of the series) is not used in fitting the resulting statistics to estimate the Hurst coefficient. Default: FALSE.
fit	a function representing the linear regression scheme to use in fitting the resulting statistics (on a log-log scale). Supported functions are: lm, lmsreg, and ltsreg. See the on-line help documentation for each of these for more information: in R, these are found in the MASS package while in S-PLUS they are indigenous and found in the splus database. Only used when method="standard" or method="smoothed". Default: lm.
freq.max	the largerst normalized frequency to include in the regression scheme. Default: 0.25.
method	a character string indicating the method to be used in estimating the Hurst coefficient (H). Choices are: "standard" Regression of SDF estimate. "smoothed" Regression of block averages of the SDF estimate taken over dyadic partitions in frequency. "robinson" Robinson's SDF integration method. Default: "standard".
n.block	an integer denoting the number of logarithmic frequency divisions to use in partitioning the estimated SDF. This input argument is only used if method="smoothed". Default: as.integer(floor(logb(length(x),base=2))), which corresponds to the maximum number of decomposition levels possible for a discrete wavelet transformation of the input time seres.

`sdf.method` a character string denoting the method to use in estimating the SDF. Choices are `"direct"`, `"lag window"`, `"wosa"` (Welch's Overlapped Segment Averaging), `"multitaper"`. See help documentation for the `sdf` function for more information. Default: `"direct"`.

`weight` a function with a single required variable (`x`) used to weight the resulting statistics (`x`) for each scale during linear regression. Currently, only supported when `fit=lm` and is only used when `method="standard"` or `method="smoothed"`. Default: `function(x) rep(1,length(x))`.

Value

an object of class `fractalBlock`.

References

P.M. Robinson (1994), Semiparametric analysis of long-memory time series, *Annals of Statistics*, **22**, 515–539.

I. Lobato and P.M. Robinson (1996), Averaged periodogram estimation of long memory, *Journal of Econometrics*, **73**, 303–324.

J. Geweke and Susan Porter-Hudak (1983), The Estimation and Application of Long Memory Time Series Models, *Journal of Time Series Analysis*, **4**, 221–237.

Murad S. Taqqu, Vadim Teverovsky, and Walter Willinger (1995), Estimators for Long-Range Dependence: An Empirical Study, *Fractals*, **3**, 785–798.

See Also

`hurstBlock`, `fractalBlock`, `HDEst`, `lm`.

Examples

```
## create test series
set.seed(100)
x <- rnorm(1024)
walk <- cumsum(x)

## calculate the Hurst coefficient of a random
## walk series using various techniques. use a
## multitaper SDF
methods <- c("standard","smoothed")
z <- lapply(methods, function(method, walk){
    hurstSpec(walk, method=method, sdf.method="multitaper")
},walk=walk )
names(z) <- methods

## plot results
old.plt <- par("plt")
for (i in 1:2){
    splitplot(2,1,i)
    plot(z[[i]])
}
```

```
par(plt=old.plt)

## Robinson's method
hurstSpec(walk, method="robinson", sdf.method="multitaper")
```

---

infoDim *Information dimension*

Description

This function estimates the information dimension by forming a delay embedding of a time series, calculating related statistical curves (one per embedding dimension), and subsequently fitting the slopes of these curves on a log-log scale using a robust linear regression model. If the slopes converge at a given embedding dimension E, then E is the correct embedding dimension and the (convergent) slope value is an estimate of the information dimension for the data.

Usage

```
infoDim(x, dimension=5, tlag=NULL,
    olag=0, n.density=100, metric=Inf,
    max.neighbors=as.integer(min(c(round(length(x) / 3), 100))),
    n.reference=as.integer(round(length(x) / 20)))
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
dimension	the maximal embedding dimension. Default: 5.
max.neighbors	let $p = k/N$ for $0 < p \leq 1$ be the mass density where N is the number of points in the embedding and k is the number of neighbors found near an arbitrary reference point in the embedding. The max.neighbors parameter defines the maximum value for k, regardless of the required density. In the case where the number of neighbors k required to meet the density p exceeds max.neighbors, then k is limited to max.neighbors and (instead) N is decreased accordingly to $N' = \lfloor \text{max.neighbors}/p \rfloor$. It is important to note that only the database of neighbors (formed by an efficient kd-tree algorithm) is reduced to N' points while *all* N points in the embedding are considered as neighbor candidates for any given reference point. The point of all this is to reduce the computational burden. Setting max.neighbors to a larger value than the default will increase the computational burden but will lessen the error in estimating the average neighborhood radius of all reference points with a (specified) constant neighborhood density. Default: min(c(round(length(x) / 3), 100)).
metric	the metric used to define the distance between points in the embedding. Choices are limited to 1, 2, or Inf which represent an L_1, L_2, and L_∞ norm, respectively. Default: Inf.

`n.density` the number of points to create in developing the density vector. For a given reference point in the phase space, the density is defined by the relation $p = k/N$ where k is the number of neighbors in the phase space and N is the total number of points in the embedding. To obtain the informaiton dimension statistics, the density is varied logarithmically from $1/N$ to 1.0. Default: `100`.

`n.reference` the number of reference points to use in forming the information dimension statistic. This argument directly specifies the number of equi-dense neighborhoods to average in forming the average neighborhood radius statistic. As with the `max.neighbors` argument, increasing `n.reference` beyond the default will increase the computational burden at the benefit of obtaining (perhaps) less variable statistics. Default: `round(length(x) / 20)`.

`olag` the number of points along the trajectory of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low correlation dimension estimates. The orbital lag must be positive or zero. Default: `length(x)/10` or `500`, whichever is smaller.

`tlag` the time delay between coordinates. Default: the decorrelation time of the autocorrelation function.

Details

The information dimension (D_1) is one of an infinite number of fractal dimensions of a chaotic system. For generalized fractal dimension estimations, correlation integral moments are determined as an average of the contents of neighbohoods in the phase space of equal radius eps. Using this approach. the information dimension for a given embedding dimension E is estimated via $D_1(E) =< \ln(p) > / \ln(\varepsilon)$ in the limit as ε approaches zero, where ε is the radius of an E-dimensional hypersphere, p is the density (also known as the mass fraction), and $< \ln(p) >$ is the average Shannon information needed to specify an arbitrary point in the phase space with accuracy ε.

Alternatively, the neighborhoods can be constructed with variable radii but with constant density. The scaling behavior of the average radii of these neighborhoods as a function of density is then used to estimate the fractal dimensions. In this function, we use this constant density approach to calculate the statistics for estimating the information dimension.

For single variable time series, the phase space is approximated with a delay embedding and $D_1(E)$ is thus estimated over statistics gathered for dimensions $1, \ldots, E$. For chaotic systems, these statistics will 'saturate' at a finite embedding dimension, revealing both the (estimated) information dimension and an appropriate embedding dimension for the system. A linear regression scheme should be to estimate the $D_1(E)$ using the statistics returned by this function.

Value

an object of class `chaoticInvariant`.

S3 METHODS

eda.plot plots an extended data analysis plot, which graphically summarizes the process of obtaining a information dimension estimate. A time history, phase plane embeddding, information

dimension curves, and the slopes of information dimension curves as a function of scale are plotted.

plot plots the information dimension curves on a log-log scale. The following options may be used to adjust the plot components:

- **type** Character string denoting the type of data to be plotted. The `"stat"` option plots the information dimension curves while the `"dstat"` option plots a 3-point estimate of the derivatives of the information dimension curves. The `"slope"` option plots the estimated slope of the information dimension curves as a function of embedding dimension. Default: `"stat"`.
- **fit** Logical flag. If `TRUE`, a regression line is overlaid for each curve. Default: `TRUE`.
- **grid** Logical flag. If `TRUE`, a grid is overlaid on the plot. Default: `TRUE`.
- **legend** Logical flag. If `TRUE`, a legend of the estimated slopes as a function of embedding dimension is displayed. Default: `TRUE`.
- **...** Additional plot arguments (set internally by the `par` function).

print prints a qualitiative summary of the results.

References

Peter Grassberger and Itamar Procaccia (1983), Measuring the strangeness of strange attractors, *Physica D*, **9**, 189–208.

Holger Kantz and Thomas Schreiber (1997), *Nonlinear Time Series Analysis*, Cambridge University Press.

See Also

`corrDim`, `embedSeries`, `timeLag`, `chaoticInvariant`, `lyapunov`, `poincareMap`, `spaceTime`, `findNeighbors`, `determinism`.

Examples

```
## calculate the information dimension estimates
## for chaotic beam data using a delay
## embedding for dimensions 1 through 10
beam.d1 <- infoDim(beamchaos, dim=10)

## print a summary of the results
print(beam.d1)

## plot the information dimension curves without
## regression lines
plot(beam.d1, fit=FALSE, legend=FALSE)

## plot an extended data analysis plot
eda.plot(beam.d1)
```

KDE *Nonparametric multidimensional probability density function estimation*

Description

Given a training matrix, this function estimates a multidimensional probability density function using the Epanechnikov kernel as a smoother. The density function is estimated at a specified and arbitrary set of points, i.e., at points not necessarily members of the training set.

Usage

```
KDE(x, at=NULL, n.grid=100)
```

Arguments

x	a matrix whose columns contain the coordinates for each dimension. Each row represents the location of a single point in a multidimensional embedding.
at	the locations of the points over which the KDE is to be calculated. Default: a multidimensional uniform grid of points spanning the training data space (defined by x).
n.grid	the number of divisions per dimension to using in forming the default grid when the at input is unspecified. Default: 100.

Details

The kernel bandwidth is constant (non-adaptive) and is determined by first computing the minimum variance of all dimensions (columns) of x. This minimum variance is then used in Scott's Rule to compute the final bandwidth.

This function is primarily used for estimating the mutual information of a time series and is included here for illustrative purposes.

Value

an object of class KDE.

S3 METHODS

eda.plot extended data analysis plot showing the original data along with a perspective and contour plot of the resulting KDE. In the case that the primary input x is a single variable (a time series), only the KDE is plotted.

plot plot the KDE or original (training) data. Options are:

- **style** a character string denoting the type of plot to produce. Choices are "original", "perspective", and "contour" for plotting the original training data, a perspective plot of the KDE, or a contour plot of the KDE over the specifed dimensions. In the case that the primary input x is a single variable (a time series), this parameter is automatically set to unity and a KDE is plotted. Default: "original".

dimensions a two-element integer vector denoting the dimensions/variables/columns to select from the training data and resulting multidimensional KDE for perspective and contour plotting. In the case that the primary input `x` is a single variable (a time series), this parameter is automatically set to unity and a KDE is plotted. Default: `1:2` for multivariate training data, 1 for univariate training data.

xlab character string defining the x-axis label. Default: `dimnames` of the specified `dimensions` of the training data. If missing, `"X"` is used. For univariate training data, the x-axis label is set to the name of the original time series.

ylab character string defining the y-axis label. Default: `dimnames` of the specified `dimensions` of the training data. If missing, `"Y"` is used. For univariate training data, the y-axis label is set to `"KDE"`.

zlab character string defining the z-axis label for perspective plots. Default: `"KDE"`.

grid a logical flag. If `TRUE`, a grid is plotted for the `"original"` style plot. Default: `"FALSE"`.

... Optional arguments to be passed directly to the specified plotting routine.

print a summary of the KDE object is printed.. Available options are:

justify text justification ala `prettPrintList`. Default: `"left"`.

sep header separator ala `prettyPrintList`. Default: `":"`.

... Additional print arguments sent directly to the `prettyPrintList` function).

See Also

`timeLag`.

Examples

```
## create a mixture of 2-D Gaussian distributed
## RVs with different means, standard
## deviations, point density, and orientation.
n.sample <- c(1000, 500, 300)
ind      <- rep(1:3, n.sample)
x <- rmvnorm(sum(n.sample),
    mean = rbind(c(-10,-20), c(10,0), c(0,0))[ ind, ],
    sd   = rbind(c(5,3), c(1,3) , c(0.3,1))[ ind, ],
    rho  = c(0.5, 1, -0.4)[ind])

## perform the KDE
z <- KDE(x)
print(z)

## plot a summary of the results
eda.plot(z)

## form KDE of beamchaos series
plot(KDE(beamchaos),type="l")
```

`lmACF` *ACF, PACF, and ACVF for various stochastic fractal time series models*

Description

Computes the autocovariance, autocorrelation or partial autocorrelation sequences for various stochastic fractal time series models.

Usage

```
lmACF(x, lag.max=32, type="correlation")
```

Arguments

`x`	an object of class `"lmModel"`. Use the `lmModel` function to create this input.
`lag.max`	the maximum number of lags at which to compute the autocovariance, the autocorrelation or the partial autocorrelation. Default: `32`.
`type`	a character string defining the output type based on the following options: **"covariance"** autocovariance sequence **"correlation"** autocorrelation sequence **"partial"** partial autocorrelation sequence Default: `"correlation"`.

Details

The autocovariance sequence is computed using Equation (2.10) of Beran (1994). The autocorrelation sequence is computed by dividing the autocovariance sequence by the variance of the process (i.e., the value of the autocovariance sequence at lag zero). The partial autocorrelation sequence is computed using the Levinson-Durbin recursions.

Value

an object of class `signalSeries` containing the result.

References

D. Percival and A. Walden (2000), *Wavelet Methods for Time Series Analysis*, Cambridge University Press, Chapter 7.

J. Beran (1994), *Statistics for Long-Memory Processes*, Chapman and Hall, Chapter 2.

D. Percival and A. Walden (1993), *Spectral Analysis for Physical Applications*, Cambridge University Press, 1993, Chapter 9.

See Also

`lmModel`, `lmSDF`, `lmSimulate`, `ACVStoPACS`.

Examples

```
models <- c("ppl","fdp","fgn")
lag <- 100
z <- lapply(models, function(x, models, lag)
    { lmACF(lmModel(x), lag=lag)@data},
    models=models, lag=lag)
names(z) <- paste(upperCase(models), "ACF")
stackPlot(seq(0,lag), z, xlab="lag")
title("Stochastic Fractal Model ACFs")
```

---

lmConfidence *Confidence intervals for unknown mean*

Description

Estimates confidence intervals for an unknown process mean of a time seeries well modeled by a stochastic fractal process.

Usage

```
lmConfidence(x, model, conf.level=0.95,
    parm.known=FALSE, n.rep=100000)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series or an object of class wavTransform.
model	an object of class "lmModel". Use the lmModel function to create this input.
conf.level	confidence interval probability on the interval (0,1). Default: 0.95.
n.rep	number of repititions in a Monte Carlo study. Default: 100000.
parm.known	a logical value. Default: FALSE.

Value

an two-element vector defining the low and high limits of the estimated confidence interval.

References

D. Percival and A. Walden (2000), *Wavelet Methods for Time Series Analysis*, Cambridge University Press, Chapter 7.

See Also

lmSimulate.

Examples

```
model <- lmModel("ppl",alpha=-0.9)
lmConfidence(lmSimulate(model), model)
```

lmConvert *Stochastic fractal exponent conversion*

Description

Estimates the unknown variance of a stochastic fractal process.

Usage

```
lmConvert(x, to="delta")
```

Arguments

x	an object of class "lmModel". Use the lmModel function to create this input.
to	a character string defining the target conversion for the given model exponent. Choices are: "alpha","delta","HG","HB","beta". Default: "delta".

Value

numeric value representing the converted exponent.

See Also

lmModel, lmConfidence.

Examples

```
model <- lmModel("ppl",alpha=-0.99)
lmConvert(model, to="delta")
```

lmModel *Constructor function for objects of class "lmModel"*

Description

Packs the parameters defining a specfied stochastic fractal time series model into a list an returns the result.

Usage

```
lmModel(model, variance.=1.0, delta=0.45,
    alpha=-0.9, HG=0.95, HB=0.95,
    innovations.var=NULL, Cs=NULL,
    bterms=10, dterms=10, M=100)
```

Arguments

`model` a character string defining the model type. Choices are

`"ppl"` Pure power law (PPL) process. A process $\{X_t\}$ is a PPL process if its SDF is given by

$$S_X(f) = C_S|f|^\alpha, \quad |f| \le 1/2,$$

where $C_S > 0$. The innovations variance for this process is given by $C_s e^{-\alpha(\log(2)+1)}$ (this is the variance of the best linear predictor of the process given its infinite past).

`"fdp"` Fractionally differenced (FD) process. A process $\{X_t\}$ is a FD process if its SDF is given by

$$S_X(f) = \frac{\sigma_\varepsilon^2}{[4\sin^2(\pi f)]^\delta}, \quad |f| \le 1/2$$

where σ_ε^2 is the innovations variance, and δ is the FD parameter. Thus, an FD model is completely defined by the innovations variance and FD parameter.

`"fgn"` Fractional Gaussian noise (FGN) process. An FGN process $\{X_t\}$ is a *stationary* Gaussian process if its ACVF is given by

$$s_{X,\tau} = \frac{\sigma_X^2}{2}\left(|\tau+1|^{2H_G} - 2|\tau|^{2H_G} + |\tau-1|^{2H_G}\right),$$

where $\sigma_X^2 > 0$ is the variance of the process, while $0 < H_G < 1$ is the so-called Hurst coefficient. The coefficient H_G is sometimes called the self-similarity parameter for a FGN process and is usually designated in the literature as simply H.

`"dfbm"` Discrete Fractional Brownian Motion. i.e., regularly-spaced samples from a FBM process that is defined over the entire real axis.

`Cs` pure power law constant. If supplied, this argument is used to compute `variance` and `innovations.var`. If not supplied and `innovations.var` is supplied, then `Cs` and `variance` are determined from the `innovations.var`. Default: `NULL`.

`HB` the Hurst coefficient for a DFBM process. Default: `0.95`.

`HG` the Hurst coefficient for an FGN process. Default: `0.95`.

`M` sets the number of terms used in the Euler-Maclaurin summation for calculating the SDF of an FGN process and DFBM process. The default value should be adequate for all values of the Hurst coefficient. Default: `100`.

`alpha` power law exponent for a PPL model. Default: `-0.9`.

`bterms` an integer used to control the number of primary terms cumulatively summed in computing an ACVS for a PPL process. Default: `10`.

`delta` the FD parameter. Default: `0.45`.

`dterms` an integer used to control the number of secondary terms cumulatively summed in computing an ACVS for a PPL process. Default: `10`.

`innovations.var`
: innovations variance for an FD or PPL model. If supplied, this argument is used to compute `variance` and, for a PPL model, `Cs`. If not supplied and `Cs` is supplied for a PPL model, then `Cs` determines `innovations.var`. If not supplied and `Cs` is also not supplied for a PPL model or if not supplied for an FD model, then `variance` determines `innovations.var`. Default: `NULL`.

`variance.`
: the process variance with a default of unity. If `cs` or `innovations.var` is specified, this parameter is set in agreement with those. If the process is nonstationary but has stationary differences, i.e., incrementally stationary, then the process variance is taken to be the variance of the stationary process that is formed by appropriately differencing the nonstationary process.

Value

an object of class `lmModel` containing a list of model parameters.

References

D. Percival and A. Walden (2000), *Wavelet Methods for Time Series Analysis*, Cambridge University Press, Chapter 7.

J. Beran (1994), *Statistics for Long-Memory Processes*, Chapman and Hall, Chapter 2.

D. Percival and A. Walden (1993), *Spectral Analysis for Physical Applications*, Cambridge University Press, 1993, Chapter 9.

See Also

`lmACF`, `lmSDF`, `lmSimulate`, `lmConvert`, `lmConfidence`, `FDWhittle`.

Examples

```
lmModel("ppl", alpha=-2.0)
lmModel("fdp", delta=0.45, innov=1.3)
lmModel("fgn", HG=0.98)
lmModel("dfbm", HB=0.35)
```

`lmSDF` *SDF for various stochastic fractal time series models*

Description

Compute a discretized version of a single-sided parametric spectral density function (SDF) for various stochastic fractal time series models.

Usage

```
lmSDF(x, sampling.interval=1, n.freq=NULL,
    n.sample=NULL, with.Nyquist=NULL)
```

Arguments

x	an object of class "lmModel". Use the lmModel function to create this input.
n.freq	the number of frequencies at which the SDF is computed (this argument should not be supplied if n.sample is supplied). If n.sample is non-NULL supplied but n.freq is NULL, the actual grid of frequencies is determined by the argument with.Nyquist. Default: if neither n.sample nor n.freq is specified, n.freq defaults to 32.
n.sample	length of a time series. If non-NULL, the spectral resolution is set to 1/(n.sample * sampling.interva Default: NULL (n.freq is used to set the specral resolution instead).
sampling.interval	the sampling interval for the process. The SDF is computed for frequencies on the interval [0, Nyquist] where Nyquist is 1/(2*sampling.interval). The value of sampling.interval must be a positive number. Default: 1.
with.Nyquist	a logical flag. If TRUE, the grid of frequencies over which the SDF is evaluated ranges from 1/2*n.freq*sampling.interval up to the Nyquist frequency; otherwise, the range is from 1/(2*n.freq + 1)*sampling.interval to just below the Nyquist frequency. The intent of this argument is to mimic the grid of Fourier frequencies for time series with an even or odd sample size by setting with.Nyquist to, respectively, TRUE or FALSE. This argument is only really intended to be used if n.sample is not supplied, but n.freq is. Default: TRUE.

Details

The SDF is computed as described in Section 7.6 of Percival and Walden (2000), after a possible change of variable to take into account the sampling interval (the discussion in the reference assumes a unit sampling interval).

Value

an object of class signalSeries containing the SDF.

References

D. Percival and A. Walden (2000), *Wavelet Methods for Time Series Analysis*, Cambridge University Press, Chapter 7.

J. Beran (1994), *Statistics for Long-Memory Processes*, Chapman and Hall, Chapter 2.

D. Percival and A. Walden (1993), *Spectral Analysis for Physical Applications*, Cambridge University Press, 1993, Chapter 9.

See Also

lmModel, lmACF, lmSimulate.

Examples

```
old.plt <- par("plt")
models <- c("ppl","fdp","fgn","dfbm")
for (i in seq(along=models)){
    splitplot(2,2,i)
    plot(lmSDF(lmModel(models[i])),
    reference.grid=FALSE, log.axes="xy")
}
par(plt=old.plt)
```

`lmSimulate` *Stochastic fractal time series simulation*

Description

Create a simulation of a stochastic fractal time series according to a specfied model.

Usage

```
lmSimulate(x, sampling.interval=1,
    mean=0, n.sample=128, generate.Sj=FALSE,
    Sj=NULL, rn=NULL)
```

Arguments

`x`	an object of class `"lmModel"`. Use the `lmModel` function to create this input.
`Sj`	a numeric vector of Davies-Harte frequency domain weights used to create the simulation(s). These weights are calculated if not supplied. Default: `NULL` (not supplied).
`generate.Sj`	a logical value. If `TRUE`, the Davies-Harte frequency domain weights (Sj) are returned instead of a simulated series. See the references for details. Default: `FALSE`.
`mean`	the mean value of of the resulting simulation. Default: `0.0`.
`n.sample`	length of a time series. Default: `128`.
`rn`	a vector of random normal deviates used to generate uncorrelated random variables for the Davies-Harte simulator. Default: `rnorm(2 * length(Sj) - 2)`.
`sampling.interval`	the sampling interval for the process. The SDF is computed for frequencies on the interval [0, Nyquist] where Nyquist is `1/(2*sampling.interval)`. The value of `sampling.interval` must be a positive number. Default: `1`.

Details

Simulates a stochastic fractal time series via the Davies-Harte technique, which randomizes spectral weights and inverts the result back to the time domain. See the references for more details.

Value

an object of class `signalSeries` containing the simulated series.

References

D. Percival and A. Walden (2000), *Wavelet Methods for Time Series Analysis*, Cambridge University Press, Chapter 7.

J. Beran (1994), *Statistics for Long-Memory Processes*, Chapman and Hall, Chapter 2.

D. Percival and A. Walden (1993), *Spectral Analysis for Physical Applications*, Cambridge University Press, 1993, Chapter 9.

Davies,R.B.and Harte,D.S.(1987). Tests for the Hurst effect, *Biometrika*, **74**, 95–102.

See Also

`lmModel`, `lmACF`, `lmSDF`, `lmConfidence`, `FDSimulate`.

Examples

```
old.plt <- par("plt")
models <- c("ppl","fdp","fgn","dfbm")
for (i in seq(along=models)){
    splitplot(2,2,i)
    plot(lmSimulate(lmModel(models[i])),
    reference.grid=FALSE)
}
par(plt=old.plt)
```

---

localProjection *Time series denoising via a local projection filtering technique*

Description

Given a time series, $X[t]$, this function performs one iteration of the local projection filtering algorithm as described in Kantz and Schreiber [1]. This noise reduction algorithm is summarized in the following steps:

1. A time lag embedding of dimension **dimension** is formed using $X[t]$, where **dimension** is typically at least twice the dimension at which the underlying dynamics of $X[t]$ become deterministic. At each point in the embedding a neighborhood is determined by a given radius and a given minimum number of required neighbors.
2. Center-of-mass vectors are computed for each embedding point neighborhood and corresponding covariance matrices are computed with respect to the center-of-maxx vectors.
3. The eigenvectors corresponding to the **noise.dimension** smallest eigenvalues are assumed to form a (local) basis for the noise subspace and the projection of the embedding vector onto these "noise" eigenvectors is subtracted from the original embedding vector.
4. The components of the corrected embedding vectors are averaged to compute the overall correction for each point in the original time series.

Usage

```
localProjection(x, dimension=3, tlag=timeLag(x), n.neighbor=dimension + 1,
    max.distance=2*stdev(x), metric=Inf, noise.dimension=1, corr.curve=TRUE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
corr.curve	boolean argument. If true, the center-of-mass vectors will be corrected for curvature effects. Default: TRUE.
dimension	the dimension of the time lag embedding created from the given time series. This value should be at least twice the dimension in which the underlying dynamics of the time series become deterministic. Default: 3.
max.distance	the neighbor search for each embedding point finds all neighbors within **max.distance**. Default: 2*stdev(x).
metric	the metric used when searching for neighbors. Supported values are 1 (1-norm), 2 (2-norm), and Inf (infinity norm). Default: Inf.
n.neighbor	the minimum number of neighbors acceptable to define a neighborhood for each embedding point. If the neighbor search using input **max.distance** does not produce at least this number of neighbors a new search is performed which finds exactly **n.neighbor** neighbors. Default: dimension + 1.
noise.dimension	the assumed dimension of the (local) noise subspace. This should be equal to the embedding dimension, **dimension**, minus the dimension where the underlying dynamics of the time series become deterministic. Default: 1.
tlag	the time lag used when creating the time lag embedding. Default: 1.

Value

the resulting denoised time series, a vector the same length as the original time series.

References

Holger Kantz and Thomas Schreiber (1997), *Nonlinear Time Series Analysis*, Cambridge University Press.

See Also

embedSeries, medianFilter, timeLag, FNN.

Examples

```
x      <- beamchaos@data
x      <- x - mean(x)
sigma  <- stdev(x)
xnoise <- x + rnorm(length(x)) * sigma / 3
xclean <- localProjection(xnoise, dimension=7, noise.dimension=5,
    max.distance=3*sigma, n.neighbor=100)
```

```
y <- data.frame(xclean, xnoise, x)
stackPlot(x=positions(beamchaos)[], y=y,
    ylab=c("denoised","noisy","original"),
    ylim=range(y))
```

---

lorenz *Chaotic response of the Lorenz system*

Description

The Lorenz system is defined by the third order set of ordinary differential equations:

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

$$\dot{y} = rx - y - xz$$

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

.

If the parameter set is $\sigma = 10$, $r = 28$, $b = 8/3$, then the system response is chaotic. The Lorenz is one the hallmark examples used in illustrating nonlinear deterministic chaotic motion.

See Also

beamchaos, ecgrr, eegduke, pd5si.

Examples

```
plot(lorenz[,1], lorenz[,3], pch=".", col="blue")
```

---

lorenz.ode *Lorenz system ODEs*

Description

Ordinary differential equations fro the Lorenz system. See the help documentation for the `lorenz` data object for more information. A parameter space defined by `sigma=10`, `r=28`, and `b=8/3` is known to produce chaotic motion.

Usage

```
lorenz.ode(x, sigma=10, r=28, b=8/3)
```

Arguments

x	a three-element numeric vector representing the current X, Y, and Z states, respectively, of the Lorenz system.
b	the b parameter. Default: 8/3.
r	the r parameter. Default: 28.
sigma	the sigma parameter. Default: 10.

Value

a vector of three values representing the Lorenz states X, Y, and Z, respectively, evaluated with the specified parameter regime.

See Also

lorenz, henon.

Examples

```
lorenz.ode(c(0.3,-0.1,1.0))
```

---

lyapunov *Local-Global Lyapunov Spectrum Estimation*

Description

Estimates the local Lyapunov exponents over a range of user supplied scales and dimensions. The local Lyapunov spectrum is calculated as follows:

1 A delayed embedding of the input time series is formed.

2 For each global reference point (specified by an intger index in the reference matrix) a local Lyapunov spectrum is calculated, one exponent for each dimension from 1 to local.dimension and for each (integer) scale specified by the scale vector. As the scales grow larger, the Lyapunov exponent estimates tend toward asymptotic values corresponding to the global Lyapunov exponents. The details of how each local spectrum is estimated is given below.

3 The local spectra are then averaged over each global reference point to stabilize the results.

Each local spectrum is obtained by estimating the eigenvalues of the so-called Oseledec matrix, which is formed through a matrix product of successive local Jacobians with the transpose of the Jacobians. The number of Jacobians in the product is equivalent to the scale. Each Jacobian is formed by fitting a local neighborhood of points (relative to a some reference point) with a multidimensional polynomial of order polynomial.order. The number of neighbors found for each reference point in the embedding is chosen to be twice the polynomial order for numerical stability. To further stabilize the results, a local Lyapunov spectrum is formed for each local reference point.

Usage

```
lyapunov(x, tlag=NULL, dimension=5, local.dimension=3,
    reference=NULL, n.reference=NULL, olag=2,
    sampling.interval=NULL, polynomial.order=3, metric=Inf, scale=NULL)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
dimension	an integer representing the embedding dimension. Default: 5.
local.dimension	an integer representing the dimension (number of) local Lyapunov exponents to estimate. This value must be less than or equal to the embedding dimension. Default: 3.
metric	the metric used to define the distance between points in the embedding. Choices are limited to 1, 2, or Inf which represent an L_1, L_2, and L_∞ norm, respectively. Default: Inf.
n.reference	the number of neighbors to use in in developing the kd-tree (used as a quick means of finding nearest neighbors in the phase space). These neighbors are collected relative to the reference points. This value must be greater than 10. Default: min(as.integer(round(length(x)/20)), 100).
olag	the number of points along the trajectory of the current point that must be exceeded in order for another point in the phase space to be considered a neighbor candidate. This argument is used to help attenuate temporal correlation in the the embedding which can lead to spuriously low correlation dimension estimates. The orbital lag must be positive or zero. Default: length(x)/10 or 500, whichever is smaller.
polynomial.order	the order of the polynomial to use in fitting data around reference points in the phase space. This poloynomial fit will be used to form the Jacobians which are in turn used to calcualte the Lypaunov exponents. Default: 3.
reference	a vector of integers representing the indices of global reference points to use in estimating the local Lyapunov spectrum. A local spectrum is estimated around each global reference point, and all the local spectra are then averaged to stabilize the results. These global reference points should be chosen such that they are far apart in time. Default: Five indices uniformly distributed on the interval [1,M], where M = Ne - max(scale) - n.reference - 2 and Ne is the number of embedding points.
sampling.interval	a numeric value representing the interval between samples in the input time series. Default: deltat(x).
scale	a vector of integers defining the scales over which the local Lyapunov exponents are to be estimated. As this scale increases, one expects the local Lyapunov exponent estimates to converge towards the global estimates. All scales must be greater than one. Default: as.integer(2^(seq(min(floor(logb(scale.max,2)) - 2 , 10)) - 1)) where scale.max = Ne - 2 - n.reference.
tlag	the time delay between coordinates. Default: the decorrelation time of the autocorrelation function.

Value

an object of class FNN.

S3 METHODS

plot plots a summary of the results. Available options are:

... Additional plot arguments (set internally by the par function).

print prints a summary of the results. Available options are:

... Additional print arguments used by the standard print function.

summary summarizes the results.

References

P. Bryant, R. Brown, and H.D.I. Abarbanel (1990), Lyapunov exponents from observed time series, *Physical Review Letters*, **65**(13), 1523–1526.

H.D.I. Abarbanel, R. Brown, J.J. Sidorowich, and L. Tsimring (1993), The analysis of observed chaotic data in physical systems, *Reviews of Modern Physics*, **65**(4), 1331–1392.

See Also

embedSeries, infoDim, corrDim, timeLag, FNN.

Examples

```
## Calculate the local Lyapunov spectrum for the
## beamchaos series
z <- lyapunov(beamchaos)

## print the results
print(z)

## summarize the results
summary(z)

## plot the results
plot(z)
```

---

medianFilter *Median filtering of a time series*

Description

Given a filter order P, the output of applying a median filter to a time series $X[t]$ for $t = 0, \ldots, N-1$ is

P odd: $Y[k] = \text{median}(X[k-(P-1)/2, ..., k+(P-1)/2])$

P even: $Y[k] = \text{median}(X[k-P/2, ..., k+P/2-1])$

for $k = 0, \ldots, N-1$. Thus, median filtering replaces the k^{th} value of the time series with the median of the time series over an P-point window centered about point k. In the case where a portion of the window exceeds the boundaries of the time series, the values outside the boundaries are ignored in the median value calculation.

Usage

```
medianFilter(x, order=2)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series.
`order`	the median filter order. This argument defines the size of the windows over which the median values are calculated. The filter order must be positive and less than twice the length of the time series. Default: `2`.

Value

a vector containing the result and of the same length as the original time series.

See Also

`localProjection`.

Examples

```
x      <- beamchaos@data
x      <- x - mean(x)
sigma  <- stdev(x)
xnoise <- x + rnorm(length(x)) * sigma / 3
xclean <- medianFilter(xnoise, order=10)
y <- data.frame(xclean, xnoise, x)

stackPlot(x=positions(beamchaos)[], y=y,
    ylab=c("denoised","noisy","original"),
    ylim=range(y))
```

pd5si *Gait stride intervals for a patient with Parkinson's Disease*

Description

These data are from the file pd5-si.txt in the 'gait in aging and disease' database of PhysioBank, which is a large collection of physiologic signals maintained as a public service at http://www.physionet.org/physiobank by NIH's National Center for Research Resources. The following quote from this Web site describes how the stride intervals were obtained. *Subjects walked continuously on level ground around an obstacle-free path. The stride interval was measured using ultra-thin, force sensitive resistors placed inside the shoe. The analog force signal was sampled at 300 Hz with a 12 bit A/D converter, using an ambulatory, ankle-worn microcomputer that also recorded the data. Subsequently, the time between foot-strikes was automatically computed. The method for determining the stride interval is a modification of a previously validated method that has been shown to agree with force-platform measures, a 'gold' standard.*

Data were collected from the healthy subjects as they walked in a roughly circular path for 15 minutes, and from the subjects with Parkinson's disease as they walked for 6 minutes up and down a long hallway.

See Also

beamchaos, ecgrr, eegduke, lorenz.

Examples

```
plot(pd5si)
```

poincareMap *Create a Poincare map*

Description

Create a map using the extrema of a scalar time series.

Usage

```
poincareMap(x, extrema="min", denoise=FALSE)
```

Arguments

x	a vector holding a scalar time series.
denoise	a logical value. If TRUE, the data is first denoised via waveshrink prior to analysis. Default: FALSE.
extrema	the type of extrema desired. May be "min" for minima, "max" for maxima, or "all" for both maxima and minima. Default: "min".

Details

This function finds the extrema of a scalar time series to form a map. The time series is assumed to be a uniform sampling of $s(t)$, where $s(t)$ is a (possibly noisy) measurement from a deterministic non-linear system. It is known that $s'(t)$, $s''(t)$, ... are legitimate coordinate vectors in the phase space. Hence the hyperplane given by $s'(t) = 0$ may be used as a Poincare surface of section. The intersections with this plane are exactly the extrema of the time series. The time series minima (or maxima) are the interesections in a given direction and form a map that may be used to estimate invariants, e.g., correlation dimension and Lyapunov exponents, of the underlying non-linear system.

The algorithm used to create a Poincare map is as follows.

1 The first and second derivatives of the resulting series are approximated via the continuous wavelet transform (CWT) using the first derivative of a Gaussian as a mother wavelet filter (see references for details).

2 The locations of the local extrema are then estimated using the standard first and second derivative tests on the CWT coefficients at a single and appropriate scale (an appropriate scale is one that is large enough to smooth out noisy components but not so large as to the oversmooth the data).

3 The extrema locations are then fit with a quadratic interpolation scheme to estimate the amplitude of the extrema using the original time series.

Value

a list where the first element (`location`) is a vector containing the temporal locations of the extrema values, with respect to sample numbers $1, \ldots, N$, where N is the length of the original time series. The second element (`amplitude`) is a vector containing the extrema amplitudes.

References

Holger Kantz and Thomas Schreiber, *Nonlinear Time Series Analysis*, Cambridge University Press, 1997.

See Also

`embedSeries`, `corrDim`, `infoDim`.

Examples

```
## Using the third coordinate (\eqn{z} state) of a
## chaotic Lorenz system, form a discrete map
## using the series maxima. Embed the resulting
## extrema in a 2-dimensional delay embedding
## (with delay=1 for a map). The resulting plot
## reveals a tent map structure common to
## Poincare sections of chaotic flows.
z <- poincareMap(lorenz[,3], extrema="max")
z <- embedSeries(z$amplitude, tlag=1, dimension=2)
plot(z, pch=1, cex=1)
```

RoverS *Estimate the Hurst coefficient by rescaled range (R/S) method*

Description

The series is partitioned into m groups. The R/S statistic is computed as described in the references, the number of groups is increased, and the calculation is repeated. A log-log plot of R/S versus number of groups is, ideally, linear, with a slope related to H, so H can be determined by linear regression.

Usage

```
RoverS(x, n.block.min=2, scale.ratio=2, scale.min=8)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
n.block.min	minimum number of blocks in partitioning the data. Must be at least 2. Default: 2.
scale.min	minimum number of data values allowed in a block This may be restricted so the statistic evaluated within each group is from a reasonable sample. Default: 8.
scale.ratio	ratio of successive scales to use in partitioning the data. For example, if scale.min=8 and scale.ratio=2, the first scale will be 8, the second scale 16, the third scale 32, and so on. Default: 2.

Value

estimated Hurst parameter H of the time series.

References

B.B. Mandelbrot and J.R. Wallis (1969), *Water Resources Research*, **5**, 228–267.

See summary in M.S. Taqqu and V. Teverovsky (1998), On Estimating the Intensity of Long-Range Dependence in Finite and Infinite Variance Time Series, in *A practical Guide to Heavy Tails: Statistical Techniques and Applications*, 177–217, Birkhauser, Boston.

See Also

hurstBlock, hurstACVF, dispersion.

Examples

```
RoverS(wmtsa::ocean)
```

spaceTime *Space time separation plot*

Description

This function can be used to generate contours of a space time separation plot. This plot type is a visual tool which can help to determine the correlation time for a particular delay embedding of a given time series.

Usage

```
spaceTime(x, dimension=2, tlag=timeLag(x, method="acfdecor"),
    olag.max=as.integer(min(500,length(x)/20)), probability=0.1)
```

Arguments

x	a vector holding a scalar time series
dimension	the desired embedding dimension. Default: 2.
olag.max	an integer representing the maximum orbital lag ot use in forming the results. Default: as.integer(min(500,length(x)/20)).
probability	a positive numeric scalar on the interval (0,1) which gives the probability associated with the first contour. This input determines the number of contours to be generated (see the output description below). Default: 0.1.
tlag	the delay used to create the delay embedding. Default: timeLag(x, method="acfdecor").

Details

Each contour, $C_p(\Delta t)$, in a space time separation plot corresponds to a particular probability, p, and gives spatial distance between pairs of phase space vectors as a function of their temporal separation. In particular, any pair of vectors seperated in time by Δt are separated in the phase space by distance $C_p(\Delta t)$ with probability p.

Value

an object of class spaceTime.

S3 METHODS

as.matrix convert the output to a matrix.

eda.plot plot a summary of the space-time contours including a density function estimate of the median contour in addition to a suggested range of suitable orbital lags. In the latter case, the most populous values of the median contour are highlighted by a cross-hatched area that covers a plot of the median curve. The suggested range for a suitable orbital lag is based on the range of values that first escape this cross-hatched region. Optional parameters include:

type An integer denoting the type of line to plot ala the par function. Default: "l" (solid line).

density The density of the cross-hatched area ala the `polygon` function. Default: `10`.

add A logical flag. If `TRUE`, the plot is added using the current `par` settings. Otherwise, the `par` settings are adjusted as needed. Default: `FALSE`.

... Additional parameters sent directly to the `par` function.

plot plot the space-time contours for the given `spaceTime` object. Optional arguments include:

lty Line type ala the `par` function. Default: `1`.

color A vector of integers defining the contour line colors. Default: `1:8`.

xlab A character string denoting the x-axis label. Default: `"Orbital Lag"`.

ylab A character string denoting the y-axis label. Default: `"Spatial Separation"`.

main A character string denoting the title label. Default: `NULL` (no title).

cex Character expansion value ala `par`. Default: `1`.

pch An integer representing the plot character ala `par`. Default: `"."`.

add A logical flag. If `TRUE`, the plot is added using the current `par` settings. Otherwise, the `par` settings are adjusted as needed. Default: `FALSE`.

... Additional parameters sent directly to the `par` function.

print print a summary of the `spaceTime` object.

References

Holger Kantz and Thomas Schreiber, *Nonlinear Time Series Analysis*, Cambridge University Press, 1997.

See Also

`embedSeries`, `determinism`, `timeLag`.

Examples

```
## Using the beamchaos data calculate the space
## time separation contours for probabilities
## 1/10, 2/10, ..., 1, for a 3-dimensional
## delay embedding with delay of 10. Plot the
## resulting contours, which will reveal
## periodicity in the data. From the top
## contour in the plot, which corresponds to
## probability 1, we can conclude that any two
## vectors in the chosen delay embedding which
## are separated in time by approximately 90
## time steps are separated by a distance of at
## least 8 in the phase space.
z <- spaceTime(beamchaos, dim=3, tlag=10,
    olag.max=500, probability=1/10)

## print the results
print(z)

## plot the results
plot(z)
```

```
## extended data analysis plot
eda.plot(z)
```

`stationarity` *Testing for stationarity in a time series*

Description

The Priestley-Subba Rao (PSR) test for nonstationarity is based upon examining how homogeneous a set of spectral density function (SDF) estimates are across time, across frequency, or both. The original test was formulated in the terms of localized lag window SDF estimators, but such estimators can suffer from bias due to leakage. To circumvent this potential problem, the SDF estimators are averages of multitaper SDF estimates using orthogonal sinusoidal tapers.

Usage

```
stationarity(x, n.taper=5,
    n.block=max(c(2, floor(logb(length(x), base=2)))), significance=0.05,
    center=TRUE, recenter=FALSE)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series.
`center`	a logical value. If `TRUE`, The mean value of the data is subtracted from the original series prior to analyis. Default: `TRUE`.
`n.block`	the number of non-overlapping blocks with which the time series will be uniformly divided. If the number of samples in the time series is not evenly divisible by `n.block`, then those samples at the of the time series that do not fit into the last block are ignored (rejected for analysis). Default: `max(c(2, floor(logb(length(x), base=2))))`.
`n.taper`	an integer specifying the number of sinusoidal tapers to use in developing the eigenspectra for each block of the time series. Default: 5.
`recenter`	a logical value. If `TRUE`, the mean value of the data is subtracted from the tapered series prior to forming a mutltiaper SDF estimate of the input time series. Default: `TRUE`.
`significance`	the significance is the number of times you expect the underlying hypothesis of stationarity to fail even though stationarity remains true. Essentially, you are allowing for error in the result. A significance of 0.05 means that you are allowing 5 percent error, i.e., you are 95 percent confident in the result. Default: 0.05.

Details

The algorithms is outlined as follows:

1. **Re-centering** The time series `x` is recentered by subtracting the sample mean.
2. **Blocking** The recentered series is then segmented into `n.block` non-overlapping blocks in time.
3. **Mutitaper SDF estimation** For each block, `n.taper` eigenspectra are formed by calculating the periodogram of the block windowed by each of the `n.taper` tapers. These eiegenspectra are then averaged to form a multitaper SDF estimator for the current block.
4. **ANOVA table** A subset of each multitaper SDF estimate is formed by extracting only those values corresponding to frequencies which are approximately uncorrelated (the details of this exercise can be found in the references). Each subset (one per block) is stacked in rows such that an `n.block` x M matrix ($\mathbf{S}$) is formed, where M is the number of (subset) Fourier frequencies. The (two-factor) ANOVA table ($\mathbf{Y}$) is then formed via $\mathbf{Y} \equiv \log(\mathbf{S}) - \psi(\text{n.taper}) + \log(\text{n.taper})$, where $\psi(\cdot)$ is the digamma function and log is the natural logarithm function.
5. **PSR statistics** Using the ANOVA table and (row, column, and grand) means of the ANOVA table, the Priesltey-Subba Rao statistics are generated: one for investigating time effects, one for investigating frequency effects, and one which combines the two to test time-frequency effects. See references for details.
6. **Stationarity tests** The PSR statisitcs are then compared to corresponding chi-square (`(1 - significance)` x 100) percentiles (normalized by $\psi'(\text{n.taper})$ where $\psi'(\cdot)$ is the trigamma function). Specifically, if the PSR statistic is found to be greater than the corresponding chi-square percentile, it indicates that there is a `1 - significance` probability that the data is nonstationary.

Value

an object of class `stationarity`.

S3 METHODS

as.list convert output to a list.

print prints the object. Available options are:

- **justify** text justification ala `prettPrintList`. Default: `"left"`.
- **sep** header separator ala `prettyPrintList`. Default: `":"`.
- **n.digits** number of digits ala `prettyPrintList`. Default: 5.
- **...** Additional print arguments sent directly to the `prettyPrintList` function).

summary prints a summary of the stationarity test results.

References

Priestley, M. B. and Subba Rao, T. (1969) "A Test for Stationarity of Time Series", *Journal of the Royal Statistical Society*, Series B, **31**, pp. 140–9.

See Also

`hurstBlock`.

Examples

```
## assess the stationarity of the ecgrr series
z <- stationarity(ecgrr, n.block=8)

## print the result, noting that all tests fail.
## The strongest failure attributed to the
## time-based fluctations of the eigenspectra
print(z)

## print a summary of the results, including the
## ANOVA table of the eigenspectra
summary(z)
```

`surrogate` *Surrogate data generation*

Description

This function can be used to generate surrogate time series via various frequency domain bootstrapping techniques. Bootstrapping has been used (in the statistics community) to assess the sampling variability of certain statistics. The nonlinear dynamics community typically uses bootstrapping to detect nonlinear structure in stationary time series. Given a time series, this function is used to generate surrogate series via Theiler's Amplitude Adjusted Fourier Transform (AAFT), Theiler's phase randomization, Davies and Harte's Circulant Embedding (CE) technique, or Davison and Hinkley's (DH) phase and amplitude randomization technique.

Theiler's techniques produce so-called *constrained realizations* since some statistical aspect of the original data preserved (the histogram for the AAFT and the periodogram for the phase randomization). The other techniques, ciruclant embedding and Davison-Hinkley, are non-constrained as both the amplitudes and phases of the original series are randomized.

Usage

```
surrogate(x, method="ce", sdf=NULL, seed=0)
```

Arguments

`x`	a vector containing a uniformly-sampled real-valued time series.
`method`	a character string representing the method to be used to generate surrogate data. Choices are: `"aaft"` Theiler's Amplitude Adjusted Fourier Transform. `"phase"` Theiler's phase randomization. `"ce"` Davies and Harte's Circulant Embedding. `"dh"` Davison and Hinkley's phase and amplitude randomization. Default: `"ce"`.

`sdf` an object of class `SDF`, containing a single-sided spectral density function estimation (corresponding to the original data) over normalized frequencies $f_k = k/(2N)$ for $k = 0, \ldots, N$ where N is the number of samples in the original time series. This argument is only used for the circulant embedding method. Default: `NULL` unless the circulant embedding method is used, and then it is `sapa::SDF(x, method="multitaper", recenter=TRUE, taper=h, single.sided=T)` where `h = taper(type="sine", n.sample=N, n.taper=5, norm=TRUE)`.

`seed` a positive integer representing the initial seed value to use for the random number generator. If `seed=0`, the current time is used as a means of generating a (unique) seed value. Otherwise, the specified seed value is used. Default: `0`.

Details

The algorithms are detailed as follows:

phase The discrete Fourier transform of a time series is calculated and the phase at each frequency is randomized to be uniformly distributed on $[0, 2\pi]$. Phase symmetry is preserved so that an inverse DFT forms a purely real surrogate. Null hypothesis: the original data come from a linear Gaussian process. Side effect: the periodogram of the surrogate and original time series are the same.

aaft An N-point normally distributed realization of a white noise process is created, where N is the length of `x`, and sorted to have the same rank as `x` (e.g., if rank$(x_t) = 5$ it means that x_t is the fifth smallest element of `x`). The result is then phase randomized and its rank (r) is then calculated. The surrogate is then created by rank ordering `x` using r. Null hypothesis: the observed time series is a monotonic nonlinear transformation of a Gaussian process. Side effect: the amplitude distribution (histogram) of the surrogate and original time series are the same.

ce The circulant embedding technique is based upon generating surrogates whose estimated SDF (e.g., a periodogram) is not constrained to be the same as that of the original series (for references for details).

dh The Davison-Hinkley technique is based upon generating surrogates by randomizing both the phases and the amplitudes in the frequency domain followed by an inversion back to the time domain.

Value

an object of class `surrogate`.

S3 METHODS

plot plots the surrogate data realizations. The following options may be used to adjust the plot components:

show. A character string defining the data to display. Choices are `"series"`, `"surrogate"`, or `"both"` for plots corresponding to the original series, surrogate series, or both, respectively. Default: `"surrogate"`.

type Character string denoting the type of data to plot. Options are `"time"` for time history, `"sdf"` for a multitaper spectral density function estimation, `"pdf"` for a probability density function estimation, and `"lag"` for a two-dimensional embedding (lag plot. Default: `"time"`.

stack A logical flag. If TRUE, the stackPlot function is called as opposed to the default plot function. As stackPlot requires a common abscissa, this option is only available for type="time" (time history) or type="sdf" (spectral density function plot). Default: TRUE.

xlab Character string denoting the x-axis label for the "time" and "sdf" "pdf" types. Default: "Time", the series name, and "Frequency (Hz)", respectively.

ylab Character string denoting the y-axis label for the "time" style. Default: the series name.

cex Character expansion factor (same as the cex argument of the par function). Default: 1.

adj.main Title adjustment ala the adj argument of the par function). Default: 1.

line.main Line spacing for title ala the line argument of the text function). Default: 0.5.

col.series A character string or integer denoting the color to use when plotting data corresponding to the original series. See the colors function for more details. Default: "black".

col.surrogate A character string or integer denoting the color to use when plotting data corresponding to the surrogate series. See the colors function for more details. Default: "red".

... Additional plot arguments (set internally by the par function).

print prints a summary of the surrogate data realization. Available options are:

... Additional print arguments used by the standard print function.

References

J. Theiler and S. Eubank and A. Longtin and B. Galdrikian and J.D. Farmer (1992), Testing for nonlinearity in time series: the method of surrogate data, *Physica D: Nonlinear Phenomena*, **58**, 77–94.

Davies,R.B.and Harte,D.S.(1987). Tests for the Hurst effect, *Biometrika*, **74**, 95–102.

D.B. Percival and W.L.B. Constantine (2002), Exact Simulation of Gaussian Time Series from Nonparametric Spectral Estimates with Application to Bootstrapping, *Statistics and Computing*, under review.

D.B. Percival and A. Walden (1993), *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*, Cambridge University Press, Cambridge, UK.

D. B. Percival, S. Sardy and A. C. Davison, *Wavestrapping Time Series: Adaptive Wavelet-Based Bootstrapping*, in W. J. Fitzgerald, R. L. Smith, A. T. Walden and P. C. Young (Eds.), *Nonlinear and Nonstationary Signal Processing*, Cambridge, England: Cambridge University Press, 2001.

D.T. Kaplan (1995), Nonlinearity and Nonstationarity: The Use of Surrogate Data in Interpreting Fluctuations in Heart Rate, *Proceedings of the 3rd Annual Workshop on Computer Applications of Blood Pressure and Heart Rate Signals*, Florence, Italy, 4–5 May.

See Also

infoDim, corrDim.

Examples

```
## create surrogate data sets using circulant
## embedding method
surr <- surrogate(beamchaos, method="ce")

## print the result
print(surr)

## plot and compare various statistics of the
## surrogate and original time series
plot(surr, type="time")
plot(surr, type="sdf")
plot(surr, type="lag")
plot(surr, type="pdf")

## create comparison time history
plot(surr, show="both", type="time")
```

---

timeLag *Estimate the proper time lag for single variable delay embeddings*

Description

Given the time series X_t, the embedding dimension E, and the time lag τ, the embedding coordinates are defined as X_t, $X_{t-\tau}$, ..., $X_{t-(E-1)\tau}$. This function can be used to estimate the time lag τ using a variety of statistical methods.

Usage

```
timeLag(x, method="acfzero", plot.data=FALSE)
```

Arguments

x	a vector containing a uniformly-sampled real-valued time series.
method	character string denoting the method to use in estimating the time delay. Supported methods are: "acfzero" First zero crossing of the autocorrelation function. "acfdecor" First $1/\exp$ of the autocorrelation function. "acfnadir" First nadir of the autocorrelation function. "mutual" First nadir of the average mutual information function. Default: "acfzero".
plot.data	a logical value. If TRUE, a plot of the time lag selection process is displayed. Default: FALSE.

Details

Currently, there exists no single method which yields an optimal time lag estimation although there are some basic criteria that are used: if the lag is chosen too small, the coordinates will be too highly correlated and the embedding will cluster tightly around the diagonal in the phase space. If the lag is chosen too large, the resulting coordinates may be almost uncorrelated and the resulting embedding may become unduly complicated, even if the underlying attractor is not. The goal is to find a lag which falls in-between these scenarios.

In addition the autocorrelation-based methods this function supports an estimation method based on the time-delayed mutual infomation (TDMI), which can be thought of as a nonlinear generalization of the autocorrelation function. For a random process X_t the TDMI, $I(\tau)$, is a measure of the information about $X_{t+\tau}$ contained in X_t. The first nadir of $I(\tau)$ gives the delay, τ_0, such that $X_{t+\tau_0}$ adds maximal information to that already known from X_t. This τ_0 is returned as an estimate of the proper time lag for a delay embedding of the given time series.

Value

an integer representing the the estimated time lag.

References

Holger Kantz and Thomas Schreiber (1997), *Nonlinear Time Series Analysis*, Cambridge University Press.

J.B. Bassingthwaighte and L.S. Liebovitch and B.J. West (1994), *Fractal Physiology*, Oxford University Press, New York.

A.M. Fraser and H.L. Swinney (1986), Independent coordinates for strange attractors from mutual information, *Physical Review A*, **33**, 1134–40.

M. Casdagli, S. Eubank, J. D. Farmer, and J. Gibson (1991), State Space Reconstruction in the Presence of Noise, *Physica D*, **51**, 52- 98.

See Also

`embedSeries`, `infoDim`, `corrDim`, `lyapunov`, `findNeighbors`, `KDE`, `determinism`.

Examples

```
## estimate the proper time lag for an embedding
## of the beamchaos data using the first zero
## crossing of the ACF
as.numeric(timeLag(beamchaos, method="acfzero", plot=TRUE))
```

Index