

The fSeries Package

October 22, 2006

Version 240.10068

Date 1996 - 2006

Title Rmetrics - The Dynamical Process Behind Markets

Author Diethelm Wuertz and many others, see the SOURCE file

Depends R (>= 1.9.0), methods, mgcv, nnet, fCalendar, fBasics

Maintainer Diethelm Wuertz and Rmetrics Core Team <wuertz@itp.phys.ethz.ch>

Description Environment for teaching “Financial Engineering and Computational Finance”

License GPL Version 2 or later

URL <http://www.rmetrics.org>

R topics documented:

ArmaModelling	1
ArfimaOxInterface	9
UnitrootDistribution	12
UnitrootTests	14
LongRangeDependence	19
GarchDistributions	26
HeavisideFunction	30
GarchModelling	31
GarchOxInterface	37
ChaoticTimeSeries	39
TimeSeriesTests	43
Index	48

Description

A collection and description of simple to use functions to model univariate autoregressive moving average time series processes, including time series simulation, parameter estimation, diagnostic analysis of the fit, and predictions of future values.

The functions are:

<code>armaSim</code>	Simulates an artificial ARMA time series process,
<code>armaFit</code>	Fits the parameters of an ARMA time series process,
<code>print</code>	S3 Print Method,
<code>plot</code>	S3 Plot Method,
<code>summary</code>	S3 Summary Method,
<code>predict</code>	Forecasts and optionally plots an ARMA process,
<code>fitted</code>	S3 Method, returns fitted values,
<code>coef coefficients</code>	S3 Method, returns coefficients,
<code>residuals</code>	S3 Method, returns residuals.

The functions for the analysis of an true ARMA process are:

<code>armaRoots</code>	Roots of the characteristic ARMA polynomial,
<code>armaTrueacf</code>	True autocorrelation function of an ARMA process.

Usage

```
armaSim(model = list(ar = c(0.5, -0.5), d = 0, ma = 0.1), n = 100,
        positions = NULL, innov = NULL, n.start = 100, start.innov = NULL,
        rand.gen = rnorm, rseed = NULL, addControl = FALSE, ...)

armaFit(formula, data, method = c("mle", "ols"), include.mean = TRUE,
        fixed = NULL, title = NULL, description = NULL, ...)

## S3 method for class 'fARMA':
print(x, ...)
## S3 method for class 'fARMA':
plot(x, which = "ask", gof.lag = 10, ...)
## S3 method for class 'fARMA':
summary(object, doplot = TRUE, which = "all", ...)

## S3 method for class 'fARMA':
predict(object, n.ahead = 10, n.back = 50, conf = c(80, 95),
        doplot = TRUE, ...)

## S3 method for class 'fARMA':
fitted(object, ...)
## S3 method for class 'fARMA':
coef(object, ...)
## S3 method for class 'fARMA':
residuals(object, ...)

armaRoots(coefficients, n.plot = 400, digits = 4, ...)
armaTrueacf(model, lag.max = 20, type = c("correlation", "partial", "both"),
            doplot = TRUE)
```

Arguments

`addControl` [`armaSim`] -
a logical value. Should control parameters added to the returned series as a control attribute?

<code>coefficients</code>	[<code>armaRoots</code>] - a numeric vector with the coefficients of the characteristic polynomial.
<code>data</code>	an optional <code>timeSeries</code> or data frame object containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>armaFit</code> is called. If <code>data</code> is an univariate series, then the series is converted into a numeric vector and the name of the response in the formula will be neglected.
<code>description</code>	a character string which allows for a brief description.
<code>digits</code>	[<code>armaRoots</code>] - output precision, an integer value.
<code>doplot</code>	[<code>armaRoots</code>] - a logical. Should a plot be displayed? [<code>predict</code>][<code>summary</code>] - is used by the <code>predict</code> and <code>summary</code> methods. By default, this value is set to <code>TRUE</code> and thus the function calls generate beside written also graphical print-out. Additional arguments required by underlying functions have to be passed through the <code>dots</code> argument.
<code>fixed</code>	[<code>armaFit</code>] - is an optional numeric vector of the same length as the total number of parameters. If supplied, only <code>NA</code> entries in <code>fixed</code> will be varied. In this way subset ARMA processes can be modeled. ARIMA modelling supports this option. Thus for estimating parameters of subset ARMA and AR models the most easiest way is to specify them by the formulas $x \sim \text{ARIMA}(p, 0, q)$ and $x \sim \text{ARIMA}(p, 0, 0)$, respectively.
<code>formula</code>	[<code>armaFit</code>] - a formula specifying the general structure of the ARMA form. Can have one of the forms $x \sim \text{ar}(q)$, $x \sim \text{ma}(p)$, $x \sim \text{arma}(p, q)$, $x \sim \text{arima}(p, d, q)$, or $x \sim \text{fracdiff}(p, q)$. <code>x</code> is the response variable and must appear in the formula expression. In the first case R's function <code>ar</code> from the <code>ts</code> package will be used to estimate the parameters, in the second case R's function <code>arma</code> from the <code>tseries</code> package will be used, in the third case R's function <code>arima</code> from the <code>ts</code> package will be used, and in the last case R's function <code>fracdiff</code> from the <code>fracdiff</code> package will be used. The state space modelling based <code>arima</code> function allows also to fit ARMA models using <code>arima(p, d=0, q)</code> , and AR models using <code>arima(q, d=0, q=0)</code> , or pure MA models using <code>arima(q=0, d=0, p)</code> . (Exogenous variables are also allowed and can be passed through the <code>...</code> argument.)
<code>gof.lag</code>	[<code>print</code>][<code>plot</code>][<code>summary</code>][<code>predict</code>] - the maximum number of lags for a goodness-of-fit test.
<code>include.mean</code>	[<code>armaFit</code>] - Should the ARIMA model include a mean term? The default is <code>TRUE</code> , note that for differenced series a mean would not affect the fit nor predictions.
<code>innov</code>	[<code>armaSim</code>] - is a univariate time series or vector of innovations to produce the series. If not provided, <code>innov</code> will be generated using the random number generator specified by <code>rand.gen</code> . Missing values are not allowed. By default the normal random number generator will be used.
<code>lag.max</code>	[<code>armaTrueacf</code>] - maximum number of lags at which to calculate the acf or pacf, an integer value by default 20.

method	[armaFit] - a character string denoting the method used to fit the model. The default method for all models is the log-likelihood parameter estimation approach, <code>method="mle"</code> . In the case of an AR model the parameter estimation can also be done by ordinary least square estimation, <code>"ols"</code> .
model	[armaSim] - a list with one (AR), two (ARMA) or three (ARIMA, FRACDIFF) elements. <code>ar</code> is a numeric vector giving the AR coefficients, <code>d</code> is an integer value giving the degree of differencing, and <code>ma</code> is a numeric vector giving the MA coefficients. Thus the order of the time series process is (F)ARIMA(p, d, q) with <code>p=length(ar)</code> and <code>q=length(ma)</code> . <code>d</code> is a positive integer for ARIMA models and a numeric value for FRACDIFF models. By default an ARIMA(2, 0, 1) model with coefficients <code>ar=c(0.5, -0.5)</code> and <code>ma=0.1</code> will be generated. [armaTrueacf][armaFischer] - a specification of the ARMA model with two elements: <code>model\$ar</code> is the vector of the AR coefficients, and <code>model\$ma</code> is the vector of the MA coefficients.
n	[armaSim] - an integer value setting the length of the series to be simulated (optional if <code>innov</code> is provided). The default value is 100.
n.ahead, n.back, conf	[print][plot][summary][predict] - are presetted arguments for the <code>predict</code> method. <code>n.ahead</code> determines how far ahead forecasts should be evaluated together with errors on the confidence intervals given by the argument <code>conf</code> . If a forecast plot is desired, which is the default and expressed by <code>doplot=TRUE</code> , then <code>n.back</code> sets the number of time steps back displayed in the graph.
n.plot	[armaRoots] - the number of data point to plot the unit circle; an integer value.
n.start	[armaSim] - gives the number of start-up values discarded when simulating non-stationary models. The start-up innovations will be generated by <code>rand.gen</code> if <code>start.innov</code> is not provided.
object	[summary][predict] - is an object of class <code>fARMA</code> returned by the fitting function <code>armaFit</code> and serves as input for the <code>summary</code> , and <code>predict</code> methods. Some methods allow for additional arguments.
positions	[armaSim] - determines the class of the returned time series. If set to <code>NULL</code> , the default value, then an object of class <code>"ts"</code> will be returned. If <code>positions</code> is a <code>timeDate</code> vector of length <code>n</code> then an object of class <code>"timeSeries"</code> will be returned.
rand.gen	[armaSim] - is the function which is called to generate the innovations. Usually, <code>rand.gen</code> will be a random number generator. Additional arguments required by the random number generator <code>rand.gen</code> , usually the location, scale and/or shape parameter of the underlying distribution function, have to be passed through the <code>dots</code> argument.
rseed	[armaSim] - the random number seed, by default <code>NULL</code> . If this argument is set to an integer value, then the function <code>set.seed(rseed)</code> will be called.

<code>start.innov</code>	[armaSim] - is a univariate time series or vector of innovations to be used as start up values. Missing values are not allowed.
<code>title</code>	a character string which allows for a project title.
<code>type</code>	[armaTrueacf] - a character string, "correlation" to compute the true autocorrelation function, "partial" to compute the true partial autocorrelation function, or "both" if both functions are desired. The start of one of the strings will suffice.
<code>which</code>	[plot][summary] - if <code>which</code> is set to "ask" the function will interactively ask which plot should be displayed. This is the default value for the <code>plot</code> method. If <code>which="all"</code> is specified all plots will be displayed. This is the default setting for the <code>summary</code> method. On the other hand, if a vector of logicals is specified, then those plots will be displayed for which the elements of the vector are set to <code>TRUE</code> .
<code>x</code>	[print][plot] - is an object of class <code>fARMA</code> returned by the fitting function <code>armaFit</code> and serves as input for the <code>predict</code> , <code>print</code> , <code>print.summary</code> , and <code>plot</code> methods. Some methods allow for additional arguments.
<code>...</code>	additional arguments to be passed.

Details

AR - Auto-Regressive Modelling:

The argument `x~ar(p)` calls the underlying functions `ar.yw`, `ar.burg1`, `ar.burg2`, `codear.ols` or `ar.mle`. For definiteness, the AR models are defined through

$$x_t - \mu = a_1(x_{t-1} - \mu) + \dots + a_p(x_{t-p} - \mu) + e_t$$

Order selection can be achieved through the comparison of AIC values for different model specifications. However this may be problematic, as of the methods here only `ar.mle` performs true maximum likelihood estimation. The AIC is computed as if the variance estimate were the MLE, omitting the determinant term from the likelihood. Note that this is not the same as the Gaussian likelihood evaluated at the estimated parameter values. With `method="yw"` the variance matrix of the innovations is computed from the fitted coefficients and the autocovariance of `x`. Burg's method allows for two alternatives `method="burg1"` or `method="burg2"` to estimate the innovations variance and hence AIC. Method 1 is to use the update given by the Levinson-Durbin recursion (Brockwell and Davis, 1991), and follows S-PLUS. Method 2 is the mean of the sum of squares of the forward and backward prediction errors (as in Brockwell and Davis, 1996). Percival and Walden (1998) discuss both.

[stats:ar]

MA - Moving-Average Modelling:

The argument `x~ma(q)` maps the call to the argument `x ~ arima(0, 0, q)`.

ARMA - Auto-Regressive Moving-Average Modelling:

The argument `x~arma(p,q)` maps the call to the argument `x~arima(p, 0, q)`.

ARIMA - Integrated ARMA Modelling:

The argument `x~arima()` calls the underlying function `arima` from R's `ts` package. For definiteness, the AR models are defined through

$$x_t = a_1x_{t-1} + \dots + a_px_{t-p} + e_t + b_1e_{t-1} + \dots + b_qe_{t-q}$$

and so the MA coefficients differ in sign from those of S-PLUS. Further, if `include.mean` is `TRUE`, this formula applies to $x - m$ rather than x . For ARIMA models with differencing, the differenced series follows a zero-mean ARMA model.

The variance matrix of the estimates is found from the Hessian of the log-likelihood, and so may only be a rough guide.

Optimization is done by `optim`. It will work best if the columns in `xreg` are roughly scaled to zero mean and unit variance, but does attempt to estimate suitable scalings. The exact likelihood is computed via a state-space representation of the ARIMA process, and the innovations and their variance found by a Kalman filter. The initialization of the differenced ARMA process uses stationarity. For a differenced process the non-stationary components are given a diffuse prior (controlled by `kappa`). Observations which are still controlled by the diffuse prior (determined by having a Kalman gain of at least `1e4`) are excluded from the likelihood calculations. (This gives comparable results to `arima0` in the absence of missing values, when the observations excluded are precisely those dropped by the differencing.)

Missing values are allowed, and are handled exactly in method "ML".

If `transform.pars` is true, the optimization is done using an alternative parametrization which is a variation on that suggested by Jones (1980) and ensures that the model is stationary. For an AR(p) model the parametrization is via the inverse tanh of the partial autocorrelations: the same procedure is applied (separately) to the AR and seasonal AR terms. The MA terms are not constrained to be invertible during optimization, but they will be converted to invertible form after optimization if `transform.pars` is true.

Conditional sum-of-squares is provided mainly for expositional purposes. This computes the sum of squares of the fitted innovations from observation `n.cond` on, (where `n.cond` is at least the maximum lag of an AR term), treating all earlier innovations to be zero. Argument `n.cond` can be used to allow comparability between different fits. The "part log-likelihood" is the first term, half the log of the estimated mean square. Missing values are allowed, but will cause many of the innovations to be missing.

When regressors are specified, they are orthogonalized prior to fitting unless any of the coefficients is fixed. It can be helpful to roughly scale the regressors to zero mean and unit variance.

Note from `arima`: The functions parse their arguments to the original time series functions available in R's time series library `ts`.

The results are likely to be different from S-PLUS's `arima.mle`, which computes a conditional likelihood and does not include a mean in the model. Further, the convention used by `arima.mle` reverses the signs of the MA coefficients.

```
[stats:arima]
```

FRACDIFF Modelling:

The argument `x~fracdiff()` calls the underlying functions from R's `fracdiff` package. The estimator calculates the maximum likelihood estimators of the parameters of a fractionally-differenced ARIMA (p,d,q) model, together (if possible) with their estimated covariance and correlation matrices and standard errors, as well as the value of the maximized likelihood. The likelihood is approximated using the fast and accurate method of Haslett and Raftery (1989). Note, the number of AR and MA coefficients should not be too large (say < 10) to avoid degeneracy in the model.

The optimization is carried out in two levels: an outer univariate unimodal optimization in `d` over

the interval $[0, 5]$, and an inner nonlinear least-squares optimization in the AR and MA parameters to minimize white noise variance.

`[fracdiff:fracdiff]`

Value

`armaFit`

returns an S4 object of class "fARMA", with the following slots:

<code>call</code>	the matched function call.
<code>data</code>	the input data in form of a data.frame.
<code>description</code>	allows for a brief project description.
<code>fit</code>	the results as a list returned from the underlying time series model function.
<code>method</code>	the selected time series model naming the applied method.
<code>formula</code>	the formula expression describing the model.
<code>parameters</code>	named parameters or coefficients of the fitted model.
<code>title</code>	a title string.

`armaRoots`

returns a three column data frame with the real, the imaginary part and the radius of the roots. The number of rows corresponds to the coefficients.

`armaTrueacf`

returns a two column data frame with the lag and the correlation function.

Note

There is nothing really new in this package. The benefit you will get with this collection is, that all functions have a common argument list with a formula to specify the model and presetted arguments for the specification of the algorithmic method. For users who have already modeled GARCH processes with SPlus, this approach will be quite natural.

The function `armaFit` allows for the following formula arguments:

<code>x ~ ar()</code>	autoregressive time series processes,
<code>x ~ ma()</code>	moving average time series processes,
<code>x ~ arma()</code>	autoregressive moving average processes,
<code>x ~ arima()</code>	autoregressive integrated moving average processes, and
<code>x ~ arfima()</code>	fractionally integrated ARMA processes.

For the first selection `x~ar()` the function `armaFit()` uses the AR modelling algorithm as implemented in R's `stats` package. For the second `x~ma()`, third `x~arma()`, and fourth selection `x~arima()` the function `armaFit()` uses the ARMA modelling algorithm also as implemented in R's `stats` package. For the last selection `x~fracdiff()` the function `armaFit()` uses the fractional ARIMA modelling algorithm from R's contributed `fracdiff` package. Note, that the AR, MA, and ARMA processes can all be modelled by the same algorithm specifying the formula `x~arima(p, d, q)` in the proper way, i.e. setting `d=0` and choosing the orders of `p` and `q` as zero in agreement with the desired model specification.

Alternatively, one can still use the functions from R's "stats" package: `arima.sim` that simulates from an ARIMA time series model, `ar`, `arima`, `arima0` that fit an AR, ARIMA model to an univariate time series, `predict` that forecasts from a fitted model, and `tsdiag` that plots time-series diagnostics. No function from these packages is masked, modified or overwritten.

The output of the `print`, `summary`, and `predict` methods have all the same style of format for each time series model with some additional algorithm specific printing. This makes it easier to interpret the results obtained from different algorithms implemented in different functions.

For `fracdiff` models the following methods are not yet implemented: `plot`, `fitted`, `residuals`, `predict`, and `predictPlot`.

Author(s)

M. Plummer and B.D. Ripley for `ar` functions and code,
 B.D. Ripley for `arima` and `ARMAacf` functions and code,
 C. Fraley and F. Leisch for `fracdiff` functions and code, and
 Diethelm Wuertz for the Rmetrics R-port.

References

- Brockwell, P.J. and Davis, R.A. (1996); *Introduction to Time Series and Forecasting*, Second Edition, Springer, New York.
- Durbin, J. and Koopman, S.J. (2001); *Time Series Analysis by State Space Methods*, Oxford University Press.
- Gardner, G, Harvey, A.C., Phillips, G.D.A. (1980); *Algorithm AS154. An algorithm for exact maximum likelihood estimation of autoregressive-moving average models by means of Kalman filtering*, Applied Statistics, 29, 311–322.
- Hannan E.J. and Rissanen J. (1982); *Recursive Estimation of Mixed Autoregressive-Moving Average Order*. Biometrika 69, 81–94.
- Harvey, A.C. (1993); *Time Series Models*, 2nd Edition, Harvester Wheatsheaf, Sections 3.3 and 4.4.
- Jones, R.H. (1980); *Maximum likelihood fitting of ARMA models to time series with missing observations*, Technometrics, 20, 389–395.
- Percival, D.P. and Walden, A.T. (1998); *Spectral Analysis for Physical Applications*. Cambridge University Press.
- Whittle, P. (1963); *On the fitting of multivariate autoregressions and the approximate canonical factorization of a spectral matrix*. Biometrika 40, 129–134.
- Haslett J. and Raftery A.E. (1989); *Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource (with Discussion)*, Applied Statistics 38, 1–50.

Examples

```
## SOURCE("fSeries.1A-ArmaModelling")
## Not run:
## armaSim/armaFit -
x = armaSim(model = list(ar = c(0.5, -0.5), ma = 0.1), n = 1000)
# Estimate the parameters:
fit = armaFit(x ~ arma(2, 1))
print(fit)
# Diagnostic Analysis:
par(mfrow = c(3, 2), cex = 0.7)
summary(fit)
# 5 Steps ahead Forecasts:
predict(fit, 5)

## armaRoots -
# Calculate and plot the roots of an ARMA process:
par(mfrow = c(2, 2), cex = 0.7)
```

```

coefficients = c(-0.5, 0.9, -0.1, -0.5)
armaRoots(coefficients)

## armaTrueacf -
model = list(ar = c(0.3, +0.3), ma = 0.1)
armaTrueacf(model)
model = list(ar = c(0.3, -0.3), ma = 0.1)
armaTrueacf(model)
## End(Not run)

```

ArfimaOxInterface *R Interface for Garch Arfima*

Description

A collection and description of functions to fit the parameters of an univariate time series to ARFIMA models interfacing the ARFIMA Ox Package.

The ARFIMA time series models include the following functions:

1	arfimaOxFit	Fits parameters of ARFIMA models,
2	print	S3 print method for ARFIMA models,
3	plot	S3 plot method for ARFIMA models,
4	summary	S3 summary method for ARFIMA models,
5	fitted	S3 method to extract the fitted values,
6	residuals	S3 method to extract the residual values,
7	predict	S3 method to predict from an ARFIMA model,
8	predictPlot	S3 method to plot ARFIMA predictions.

Usage

```

arfimaOxFit(formula, data, method = c("mle", "nls", "mpl"),
  trace = TRUE, title = NULL, description = NULL)

```

Arguments

data	an optional timeSeries or data frame object containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which armaFit is called. If data is an univariate series, then the series is converted into a numeric vector and the name of the response in the formula will be neglected.
description	a character string which allows for a brief description.
formula	a formula object which specifies the ARFIMA process.
method	a character string denoting the method used for parameter estimation.
title	a character string which allows for a project title.
trace	a logical flag. Should the estimation process be traced? By default TRUE.

Details

Ox Interface:

The function garchOxFit interfaces a subset of the functionality of the ARFIMA 1.01 Pack-

age written in Ox. ARFIMA is one of the most sophisticated packages for modelling univariate ARFIMA processes.

About Ox:

Ox (tm) is an object-oriented matrix language with a comprehensive mathematical and statistical function library. Many packages were written for Ox including software mainly for econometric modelling. The Ox packages for time series analysis and forecasting, Arfima, Doornik and Ooms [2003], Garch, Laurent and Peters [2005], and State Space Modelling, Koopman, Shepard and Doornik [1998], are especially worth to note. Since most of the R-users wan't to change to another Statistical Computing environment, we made selected parts of the ARFIMA Ox software available for them through an R-Interface. What you have to do, is to read carefully the "Ox citation and copyright" rules and if you agree and fullfill the conditions, then download the OxConsole Software together with the "Arfima" Package, currently implemented Arfima 1.01. If you are not qualified for a free license, order your copy from Timberlake Consultants. We recommend to install the "Setup.exe" for "Ox3" under the path "C:\Ox\Ox3" and to unzip the "Arfima" Package in the directory "C:\Ox\Ox3\Packages".

Distribution:

Ox and Arfima are distributed by Timberlake Consultants Ltd. Timberlake Consultants can be contacted through the following web site: www.timberlake.co.uk.

Installation of the Interface:

In addition you have to copy the files "ArfimaOxFit.ox" and "ArfimaOxPredict.ox" (which is the interface written especially for Rmetrics) from the "fSeries/data/" directory to the Ox library directory "C:\Ox\Ox3\lib".

Ox Citation and Copyright Rules:

Ox and all its components are copyright of Jurgen A. Doornik. The Console (command line) versions may be used freely for academic research and teaching purposes only. Commercial users and others who do not qualify for the free version must purchase the Windows version of Ox and GiveWin with documentation, regardless of which version they use (so even when only using Ox on Linux or Unix). Ox should be cited whenever it is used. Refer to the two references given below. Note, failure to cite the use of Ox in published work may result in loss of the right to use the free version, and an invoice at the full commercial price. Ox is available from Timberlake Consultants. The Ox syntax is public, and you may do with your own Ox code whatever you wish, including the files "ArfimaOx*.ox".

Work to do:

Note, only a small part of the functionalities are interfaced until now to Rmetrics. But, principally it would be possible to interface also other functionalities offered by the Ox Arfima Package. This work is left to the Rmetrics/Ox user.

Value

arfimaOxFit

returns an S4 object of class "fARMA", with the following slots:

call	the matched function call.
data	the input data in form of a data.frame.
description	allows for a brief project description.
fit	the results as a list returned from the underlying time series model function.

method	a character string naming the selected ARFIMA model. Implemented are the exact max-likelihood estimation method, "mle", the non-linear least-square method, "nls", and the "mplik"
formula	the formula expression describing the model.
parameters	named parameters or coefficients of the fitted model.
title	a title string.

For further details please consult `help (ArmaModelling)`.

Note

The R functions were tested only under MS Windows XP. Please report your experience with other operating systems to info@metrics.org. The functions should also run under the newer version "Ox4".

Author(s)

Jurgen A. Doornik for the Ox Environment, www.doornik.com,
 Jurgen A. Doornik and Marius Ooms for the Arfima Ox package, www.doornik.com,
 Diethelm Wuertz for R's Ox Arfima interface.

References

- Doornik J.A. (2002), Object-Oriented Matrix Programming Using Ox, London, 3rd ed.: Timberlake Consultants Press and Oxford: www.doornik.com.
- Doornik J.A., Ooms M. (2001), A Package for Estimating, Forecasting and Simulating Arfima Models, Arfima Package 1.01 for Ox, pp. 32.
- Doornik J.A., Ooms M. (2003), Computational Aspects of Maximum Likelihood Estimation of Autoregressive Fractionally Integrated Moving Average Models, Computational Statistics and Data Analysis 42, 333–348.
- Koopman J.S., Shepard N., Doornik J.A. (1999), Statistical Algorithms for Models in State Space using SsfPack 2.2, Econometrics Journal 2, 113–166.
- Laurent S., Peters J.P., [2005], G@RCH 4.0, Estimating and Forecasting ARCH Models, Timberlake Consultants, www.timberlake.co.uk

See Also

`ArmaModelling`.

Examples

```
## Not run:
## SOURCE("fSeries1.1B-ArfimaOxInterface")

## arfimaOxFit -
x = armaSim(model = list(ar = c(0.5, - 0.5), d = 0.3, ma = 0.1), n = 500)
fit = arfimaOxFit(formula = x ~ arfima(2,1))

print(fit)
plot(fit, which = "all:")
summary(fit, doplot = FALSE)

fitted(fit)[1:10]
```

```

residuals(fit)[1:10]

predict(object)
## End(Not run)

```

UnitrootDistribution
Unit Root Distributions

Description

A collection and description of functions to compute the distribution and and quantile function for the unit root test statistics.

The functions are:

<code>punitroot</code>	the cumulative probability,
<code>qunitroot</code>	the quantiles of the unit root test statistics,
<code>pdfctest</code>	the cumulative probability for the ADF test,
<code>qdfctest</code>	the quantiles for the ADF test.

Usage

```

punitroot(q, n.sample = 0, trend = c("c", "nc", "ct", "ctt"),
  statistic = c("t", "n"), na.rm = FALSE)
qunitroot(p, n.sample = 0, trend = c("c", "nc", "ct", "ctt"),
  statistic = c("t", "n"), na.rm = FALSE)

pdfctest(q, n.sample, trend = c("nc", "c", "ct"), statistic = c("t", "n"))
qdfctest(p, n.sample, trend = c("nc", "c", "ct"), statistic = c("t", "n"))

```

Arguments

<code>n.sample</code>	the number of observations in the sample from which the quantiles are to be computed. [*unitroot] - Specify <code>n.sample=0</code> for asymptotic quantiles. The default is 0.
<code>na.rm</code>	a logical value. If set to <code>TRUE</code> , missing values will be removed otherwise not, the default is <code>FALSE</code> .
<code>p</code>	a numeric vector of probabilities. Missing values are allowed.
<code>q</code>	vector of quantiles or test statistics. Missing values are allowed.
<code>statistic</code>	a character string describing the type of test statistic. Valid choices are "t" for t-statistic, and "n" for normalized statistic, sometimes referred to as the rho-statistic. The default is "t".
<code>trend</code>	a character string describing the regression from which the quantiles are to be computed. Valid choices are: "nc" for a regression with no intercept (constant) nor time trend, and "c" for a regression with an intercept (constant) but no time trend, "ct" for a regression with an intercept (constant) and a time trend. The default is "c".

Value

The function `padftest` returns the cumulative probability of the finite sample distribution of the unit root test statistics.

The function `qadftest` returns the quantiles of the finite sample distribution of the unit root test statistics, given the probabilities.

The function `punitroot` returns the cumulative probability of the asymptotic or finite sample distribution of the unit root test statistics.

The function `qunitroot` returns the quantiles of the asymptotic or finite sample distribution of the unit root test statistics, given the probabilities.

Note

The programs `padf` and `qadf` use the tables from A. Banerjee et al. (1993).

The programs `punitroot` and `qunitroot` use Fortran routines and the tables from J.G. McKinnon (1988). Many thanks to J.G. McKinnon putting his code and tables under the GPL license, which made this implementation possible.

Author(s)

J.G. McKinnon for the underlying Fortran routine and the tables,
Diethelm Wuertz for the Rmetrics R-port.

References

Banerjee A., Dolado J.J., Galbraith J.W., Hendry D.F. (1993); *Cointegration, Error Correction, and the Econometric Analysis of Non-Stationary Data*, Oxford University Press, Oxford.

Dickey, D.A., Fuller, W.A. (1979); *Distribution of the estimators for autoregressive time series with a unit root*, Journal of the American Statistical Association 74, 427–431.

MacKinnon, J.G. (1996); *Numerical distribution functions for unit root and cointegration tests*, Journal of Applied Econometrics 11, 601–618.

Phillips, P.C.B., Perron, P. (1988); *Testing for a unit root in time series regression*, Biometrika 75, 335–346.

Examples

```
## SOURCE("fSeries.2A-UnitrootDistribution")

## qunitroot -
# Asymptotic quantile of t-statistic
qunitroot(0.95, trend = "nc", statistic = "t")

## qunitroot -
# Finite sample quantile of n-statistic
qunitroot(0.95, n.sample = 100, trend = "nc", statistic = "n")

## punitroot -
# Asymptotic cumulative probability of t-statistic
punitroot(1.2836, trend = "nc", statistic = "t")

## punitroot -
# Finite sample cumulative probability of n-statistic
punitroot(1.2836, n.sample = 100, trend = "nc", statistic = "n")
```

```
## dfTable -
# Dickey-Fuller Internal Table:
.dfTable(trend = "nc", statistic = "t")
# Interpolate q and p-values
p = 0.984
n.sample = 78
Trend = Statistic = Q = P = NULL
for (trend in c("nc", "c", "ct")) {
  for (statistic in c("t", "n")) {
    Trend = c(Trend, trend)
    Statistic = c(Statistic, statistic)
    q = qdftest(p, n.sample, trend, statistic)
    Q = c(Q, q)
    P = c(P, pdfctest(q, n.sample, trend, statistic))
  }
}
data.frame(Trend, Statistic, Q, P)
```

UnitrootTests

Unit Root Time Series Tests

Description

A collection and description of functions for unit root testing. The family of tests includes ADF tests based on Banerjee's et al. tables and on J.G. McKinnons' numerical distribution functions. In addition we have included tests written by B. Pfaff.

The functions are:

adfTest	Augmented Dickey-Fuller test for unit roots,
unitrootTest	the same based on McKinnons's test statistics.

Added functions based on the 'urca' package include:

urdfTest	Augmented Dickey-Fuller test for unit roots,
urersTest	Elliott-Rothenberg-Stock test for unit roots,
urkpssTest	KPSS unit root test for stationarity,
urppTest	Phillips-Perron test for unit roots,
urspTest	Schmidt-Phillips test for unit roots,
urzaTest	Zivot-Andrews test for unit roots.

Note, that the contributed R package urca is required!

Usage

```
urTest(x, method = c("unitroot", "adf", "urers", "urkpss", "urpp",
  "ursp", "urza"), title = NULL, description = NULL, ...)
```

```
unitrootTest(x, lags = 1, type = c("nc", "c", "ct"), title = NULL,
  description = NULL)
```

```

adftest(x, lags = 1, type = c("nc", "c", "ct"), title = NULL,
        description = NULL)

urdfTest(x, lags = 1, type = c("nc", "c", "ct"), doplot = TRUE)
urersTest(x, type = c("DF-GLS", "P-test"), model = c("constant", "trend"),
          lag.max = 4, doplot = TRUE)
urkpssTest(x, type = c("mu", "tau"), lags = c("short", "long", "nil"),
           use.lag = NULL, doplot = TRUE)
urppTest(x, type = c("Z-alpha", "Z-tau"), model = c("constant", "trend"),
         lags = c("short", "long"), use.lag = NULL, doplot = TRUE)
urspTest(x, type = c("tau", "rho"), pol.deg = c(1, 2, 3, 4),
         signif = c(0.01, 0.05, 0.1), doplot = TRUE)
urzaTest(x, model = c("intercept", "trend", "both"), lag, doplot = TRUE)

```

Arguments

description	a character string which allows for a brief description.
doplot	[ur*Test] - a logical flag, by default TRUE. Should a diagnostical plot be displayed?
lag.max	[urersTest] - the maximum numbers of lags used for testing of a decent lag truncation for the "P-test", BIC used, or the maximum number of lagged differences to be included in the test regression for "DF-GLS".
lag	[urzaTest] - the highest number of lagged endogenous differenced variables to be included in the test regression.
lags	[urkpssTest][urppTest] - the maximum number of lags used for error term correction.
method	[urTest] - a character string describing the desired method, one of: "unitroot", "adf", "urers", "urkpss", "urpp", "ursp", or "urza".
model	[urersTest] - a character string denoting the deterministic model used for detrending, either "constant", the default, or "trend". [urppTest] - a character string which determines the deterministic part in the test regression, either "constant", the default, or "trend". [urzaTest] - a character string specifying if the potential break occurred in either the "intercept", the linear "trend" or in "both".
pol.deg	[urspTest] - the polynomial degree in the test regression.
signif	[urspTest] - the significance level for the critical value of the test statistic.
title	a character string which allows for a project title.
type	[adftest][unitrootTest] - a character string describing the type of the unit root regression. Valid choices are "nc" for a regression with no intercept (constant) nor time trend, and "c"

	for a regression with an intercept (constant) but no time trend, "ct" for a regression with an intercept (constant) and a time trend. The default is "c".
	[urkpssTest] - a character string which denotes the type of deterministic part, either "mu", the default, or "tau".
	[urppTest] - a character string which specifies the test type, either "Z-alpha", the default, or "Z-tau".
	[urspTest] - a character string which specifies the test type, either "tau", the default, or "rho".
use.lag	[urkpssTest] - a character string specifying the number of lags. Allowed arguments are lags=c("short", "long", "nil"), for more information see the details section. [urppTest] - Use of a different lag number, specified by the user.
x	a numeric vector or time series object.
...	[urTest] - optional arguments passed to the underlying test functions.

Details

ADF Tests:

The `adftest` computes test statistics and p values along the implementation from Trapletti's augmented Dickey–Fuller test for unit roots. In contrast to Trapletti's function three kind of test types can be selected.

Unit Root Tests from Berhard Pfaff's "urca" Package:

Elliott–Rothenberg–Stock Test for Unit Roots:

To improve the power of the unit root test, Elliot, Rothenberg and Stock proposed a local to unity detrending of the time series. ERS developed a feasible point optimal test, "P-test", which takes serial correlation of the error term into account. The second test type is the "DF-GLS" test, which is an ADF-type test applied to the detrended data without intercept. Critical values for this test are taken from MacKinnon in case of `model="constant"` and else from Table 1 of Elliot, Rothenberg and Stock.

[urca:ur.ers]

KPSS Test for Unit Roots:

Performs the KPSS unit root test, where the Null hypothesis is stationarity. The test types specify as deterministic component either a constant "mu" or a constant with linear trend "tau". `lags="short"` sets the number of lags to *root 4 of [4 times (n/100)]*, whereas `lags="long"` sets the number of lags to *root 4 of [12 times (n/100)]*. If `lags="nil"` is chosen, then no error correction is made. Furthermore, one can specify a different number of maximum lags by setting `use.lag` accordingly.

[urca:ur.kpss]

Phillips–Perron Test for Unit Roots:

Performs the Phillips and Perron unit root test. Beside the Z statistics Z-alpha and Z-tau, the Z statistics for the deterministic part of the test regression are computed, too. For correction of the

error term a Bartlett window is used.

[urca:ur.pp]

Schmidt–Phillips Test for Unit Roots:

Performs the Schmidt and Phillips unit root test, where under the Null and Alternative Hypothesis the coefficients of the deterministic variables are included. Two test types are available: the "rho-test" and the "tau-test". Both tests are extracted from the LM principle.

[urca:ur.sp]

Zivot–Andrews Test for Unit Roots:

Performs the Zivot and Andrews unit root test, which allows a break at an unknown point in either the intercept, the linear trend or in both. This test is based upon the recursive estimation of a test regression. The test statistic is defined as the minimum t-statistic of the coefficient of the lagged endogenous variable.

[urca:ur.za]

Value

All tests return an object of class "fHTEST" with the following slots:

@call	the function call.
@data	a data frame with the input data.
@data.name	a character string giving the name of the data frame.
@test	a list object which holds the output of the underlying test function.
@title	a character string with the name of the test.
@description	a character string with a brief description of the test.
\$statistic	the value of the test statistic.
\$parameter	the lag order.
\$p.value	the p-value of the test.
\$method	a character string indicating what type of test was performed.
\$data.name	a character string giving the name of the data.
\$alternative	a character string describing the alternative hypothesis.
\$name	the name of the underlying function, which may be wrapped.
\$output	additional test results to be printed.

Note

The `ur*Test` wrapper functions fulfill the naming conventions of `Rmetrics`, return an S4 object named `fHTEST` as any other hypothesis test from `Rmetrics`, and allow for `timeSeries` objects as input. These are the only differences. The `Rmetrics` wrappers were tested with `urca` version 0.7.9.

If you are running `Rmetrics` under an operating system where the R-package `urca` or one of the other packages from the dependency tree is not available you can load the required functions through the command `xmpfSeries()` and select `funUrca` from the menu. A minimalist copy of "urca" and required dependent functions are saved in the demo file `funUrca.R`.

For further details we refer to the manual pages of the "urca" package.

Author(s)

Adrian Trapletti for the tests adapted from R's "tseries" package,
 Bernhard Pfaff for the tests wrapped from R's "urca" package,
 Diethelm Wuertz for the Rmetrics R-port.

References

Banerjee A., Dolado J.J., Galbraith J.W., Hendry D.F. (1993); *Cointegration, Error Correction, and the Econometric Analysis of Non-Stationary Data*, Oxford University Press, Oxford.

Dickey, D.A., Fuller, W.A. (1979); *Distribution of the estimators for autoregressive time series with a unit root*, Journal of the American Statistical Association 74, 427–431.

Kwiatkowski D., Phillips P.C.B, Schmidt P., Shin Y. (1992); *Testing the Null Hypothesis of Stationarity against the Alternative of a Unit Root*, Journal of Econometrics 54, 159–178.

MacKinnon, J.G. (1996); *Numerical distribution functions for unit root and cointegration tests*, Journal of Applied Econometrics 11, 601–618.

Perron P. (1988); *Trends and Random Walks in Macroeconomic Time Series*, Journal of Economic Dynamics and Control 12, 297–332.

Phillips P.C.B., Perron P. (1988); *Testing for a unit root in time series regression*, Biometrika 75, 335–346.

Said S.E., Dickey D.A. (1984); *Testing for Unit Roots in Autoregressive-Moving Average Models of Unknown Order*, Biometrika 71, 599–607.

Schwert G.W. (1989); *Tests for Unit Roots: A Monte Carlo Investigation*, Journal of Business and Economic Statistics 2, 147–159.

Examples

```
## SOURCE("fSeries.2B-UnitrootTests")

## adfTest -
# A time series which contains no unit-root:
x = rnorm(1000)
# A time series which contains a unit-root:
y = cumsum(c(0, x))
# Test:
adfTest(x)
adfTest(y)

## unitrootTest -
unitrootTest(x)
unitrootTest(y)

## ur*Test -
# Unit Root Tests build on Bernhard Pfaff's Implementation:
# Examples can be found in the demo file "xmpTestUnitRoots".
```

 LongRangeDependence

Long Range Dependence Modelling

Description

A collection and description of functions to investigate the long range dependence or long memory behavior of an univariate time series process. Included are functions to simulate fractional Gaussian noise and fractional ARMA processes, functions to model the true autocorrelations and the spectrum of these processes, and functions to compute the Hurst exponent by several different methods.

The Functions and methods are:

Functions to simulate long memory time series processes:

fnmSim	Simulates fractional Brownian motion,
- mvn	from the numerical approximation of the stochastic integral,
- chol	from the Choleski's decomposition of the covariance matrix,
- lev	using the method of Levinson,
- circ	using the method of Wood and Chan,
- wave	using the wavelet synthesis,
fgnSim	Simulates fractional Gaussian noise,
- beran	using the method of Beran,
- durbin	using the method Durbin and Levinson,
- paxson	using the method of Paxson,
farimaSim	simulates FARIMA time series processes.

Functions to model the true autocorrelation function and spectrum:

fgnTrueacf	Returns true FGN covariances,
fgnTruefft	returns true FGN fast Fourier transform,
farimaTrueacf	returns true FARIMA covariances,
farimaTruefft	returns true FARIMA fast Fourier transform.

Functions to estimate the Hurst exponent:

aggvarFit	Aggregated variance method,
diffvarFit	Differenced aggregated variance method,
absvalFit	aggregated absolute value (moment) method,
higuchiFit	Higuchi's or fractal dimension method,
pengFit	Peng's or variance of residuals method,
rsFit	R/S Rescaled Range Statistic method,
perFit	periodogram method,
boxperFit	boxed (modified) periodogram method,
whittleFit	Whittle estimator,
hurstSlider	Interactive Display of Hurst Estimates.

Function for the wavelet estimator:

waveletFit	wavelet estimator.
------------	--------------------

Usage

```

fbmSim(n = 100, H = 0.7, method = c("mvn", "chol", "lev", "circ", "wave"),
       waveJ = 7, seed = NULL, doplot = TRUE, fgn = FALSE)
fgnSim(n = 1000, H = 0.7, method = c("beran", "durbin", "paxson"))
farimaSim(n = 1000, model = list(ar = c(0.5, -0.5), d = 0.3, ma = 0.1),
         method = c("freq", "time"), ...)

fgnTrueacf(n, H)
fgnTruefft(n, H)
farimaTrueacf(n, H)
farimaTruefft(n, H)

aggvarFit(x, levels = 50, minnpts = 3, cut.off = 10^c(0.7, 2.5),
          doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
diffvarFit(x, levels = 50, minnpts = 3, cut.off = 10^c(0.7, 2.5),
           doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
absvalFit(x, levels = 50, minnpts = 3, cut.off = 10^c(0.7, 2.5), moment = 1,
          doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
higuchiFit(x, levels = 50, minnpts = 2, cut.off = 10^c(0.7, 2.5),
           doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
pengFit(x, levels = 50, minnpts = 3, cut.off = 10^c(0.7, 2.5),
        method = c("mean", "median"),
        doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
rsFit(x, levels = 50, minnpts = 3, cut.off = 10^c(0.7, 2.5),
      doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
perFit(x, cut.off = 0.1, method = c("per", "cumper"),
       doplot = FALSE, title = NULL, description = NULL)
boxperFit(x, nbox = 100, cut.off = 0.10,
          doplot = FALSE, trace = FALSE, title = NULL, description = NULL)
whittleFit(x, order = c(1, 1), subseries = 1, method = c("fgn", "farma"),
           trace = FALSE, spec = FALSE, title = NULL, description = NULL)
hurstSlider(x = fgnSim())

waveletFit(x, length = NULL, order = 2, octave = c(2, 8),
           doplot = FALSE, title = NULL, description = NULL)

show.fHURST(object)

```

Arguments

cut.off	[*Fit] - a numeric vector with the lower and upper cut-off points for the estimation. They should be chosen to define a linear range. The default values are c(0.7, 2.5), i.e. 10 ^{0.7} and 10 ^{2.5} , respectively.
description	[*Fit] - a character string which allows for a brief description.
doplot	[*Fit] - a logical flag, by default FALSE. Should a plot be displayed?
fgn	[fbmSim] - a logical flag, if FALSE, the functions returns a FBM series otherwise a FGN series.

H	[fgnSim] - the Hurst exponent, a numeric value between 0.5 and 1, by default 0.7.
length	[waveletFit] - the length of data to be used, must be power of 2. If set to NULL, the previous power will be used.
levels	[*Fit] - the number of aggregation levels or number of blocks from which the variances or moments are computed.
method	[fbmSim] - the method how to generate the FBM time series sequence, one of the following five character strings: "mvn", "chol", "lev", "circ", or "wave". [fgnSim] - the method how to generate the FGN time series sequence, one of the following three character strings: "beran", "durbin", or "paxson". [farimaSim] - the method how to generate the time series sequence, one of the following two character strings: "freq", or "time". [pengFit] - a string naming the method how to do the averaging, either calculating the "mean" or the "median". [perFit] - a string naming the method how to fit the data, either using the periodogram itself "per", or using the cumulated periodogram "cumper". [whittleFit] - a string naming the underlying time series process to be estimated, either "fgn" for FGN processes, or "farima" for FARIMA models.
minnpts	[*Fit] - the minimum number of points or blocksize to be used to estimate the variance or moments at any aggregation level.
model	a list with model parameters ar, ma and d. ar is a numeric vector giving the AR coefficients, d is an integer value giving the degree of differencing, and ma is a numeric vector giving the MA coefficients. Thus the order of the time series process is FARMA(p, d, q) with $p = \text{length}(\text{ar})$ and $q = \text{length}(\text{ma})$. d is a fractional value for FARMA models. By default an FARMA(2, d, 1) model with coefficients $\text{ar} = c(0.5, -0.5)$, $\text{ma} = 0.1$, and $d = 0.3$ will be generated.
moment	[absvalHurst] - an integer value, by default 1 which denotes absolute values. For values larger than one this argument determines what absolute moment should be calculated.
n	[fgnSim][farimaSim] - number of data points to be simulated, a numeric value, by default 1000.
nbox	[boxperFit] - is the number of boxes to divide the data into. A numeric value, by default 100.
object	an object of class fHurst.
octave	[waveletFit] - beginning and ending octave for estimation. An integer vector with two elements. By default $c(2, 8)$. If the upper value is too large, it will be replaced by the maximum allowed value.
order	[waveletFit] - the order of the wavelet. An integer value, by default 2.

seed	[fbmSim] - an integer value setting the seed for the random number generation.
spec	[whittleFit] - Should the periodogram be returned? A logical flag, by default FALSE.
subseries	[whittleFit] - allows optionally to subdivide the series into subseries. A numeric value, by default 1.
title	a character string which allows for a project title.
trace	a logical value, by default FALSE. Should the estimation process be traced?
waveJ	[fbmSim] - an integer parameter for the simulation of FBM using the wavelet method.
x	[*Fit] - the numeric vector of data, an object of class <code>timeSeries</code> , or any other object which can be transformed into a numeric vector by the function <code>as.vector</code> .
...	arguments to be passed.

Details

I: Functions to Simulate Long Memory Processes:

Fractional Gaussian Noise:

The function `fgnSim` simulates a series of fractional Gaussian noise, FGN. FGN provides a parsimonious model for stationary increments of a self-similar process parameterised by the Hurst exponent H and variance. Fractional Gaussian noise with $H < 0.5$ demonstrates negatively auto-correlated or anti-persistent behaviour, and FGN with $H > 0.5$ demonstrates $1/f$, long memory or persistent behaviour, and the special case. The case $H = 0.5$ corresponds to the classical Gaussian white noise. One can select from three different methods. The first generator named "beran" uses the fast Fourier transform to generate the series based on SPLUS code written originally by J. Beran [1994]. The second generator named "durbin" produces a FGN series by using the Durbin-Levinson coefficients. The algorithm was reimplemented in pure S based on the C source code written by V. Teverovsky [199x]. The third generator named "paxson" was proposed by V. Paxson [199x], this approximate method is a very fast and requires low storage. However, the algorithm reveals some weakness in the method which was discussed by D.A. Rolls [2001].

Fractional ARIMA Processes:

The function `farimaSim` is a generator for fractional ARIMA time series processes. A Gaussian FARIMA(0,d,0) series can be created, where d is related to the Hurst exponent H through $d=H-0.5$. This is a particular case of the more general Gaussian FARIMA(p,d,q) process which follows the same asymptotic relations for their autocovariance and the spectral density as do the Gaussian FARIMA(0,d,0) processes. Two different generators are implemented in S. The first named "freq" works in the frequency domain and generates the series from the fast Fourier transform based on SPLUS code written originally by J. Beran [1994]. The second method creates the series in the time domain, therefore named "time". The algorithm was reimplemented in pure S based on the Fortran source code from the R's `fracdiff` package originally written by C. Fraley [1991]. Details for the algorithm are given in J Haslett and A.E. Raftery [1989].

II: Functions to Model True Correlations and Spectrum:

The functions `fgnTrueacf` and `farimaTrueacf` return the true covariances of an FGN and Gaussian FARIMA(0,d,0) time series process. The functions `fgnTruefft` and `farimaTruefft`

return the true fast Fourier transform of an FGN and Gaussian FARIMA(0,d,0) time series process. The R functions are implemented from SPlus code written by J. Beran [1994].

III: Functions to Estimate the Hurst Exponent:

These are 9 functions as described by M.S. Taqqu, V. Teverovsky, and W. Willinger [1995] to estimate the self similarity parameter and/or the intensity of long-range dependence in a time series.

Aggregated Variance Method:

The function `aggvarFit` computes the Hurst exponent from the variance of an aggregated FGN or FARIMA time series process. The original time series is divided into blocks of size m . Then the sample variance within each block is computed. The slope $\beta = 2H - 2$ from the least square fit of the logarithm of the sample variances versus the logarithm of the block sizes provides an estimate for the Hurst exponent H .

Differenced Aggregated Variance Method:

To distinguish jumps and slowly decaying trends which are two types of non-stationary, from long-range dependence, the function `diffvarFit` differences the sample variances of successive blocks. The slope $\beta = 2H - 2$ from the least square fit of the logarithm of the differenced sample variances versus the logarithm of the block sizes provides an estimate for the Hurst exponent H .

Aggregated Absolute Value/Moment Method:

The function `absvalFit` computes the Hurst exponent from the moments $\text{moment} = M$ of absolute values of an aggregated FGN or FARIMA time series process. The first moment $M = 1$ coincides with the absolute value method, and the second moment $M = 2$ with the aggregated variance method. Again, the slope $\beta = M(H - 1)$ of the regression line of the logarithm of the statistic versus the logarithm of the block sizes provides an estimate for the Hurst exponent H .

Higuchi or Fractal Dimension Method:

The function `higuchiFit` implements a technique which is very similar to the absolute value method. Instead of blocks a sliding window is used to compute the aggregated series. The function involves the calculation of the length of a path and, in principle, finding its fractal dimension D . The slope $D = 2 - H$ from the least square fit of the logarithm of the expected path lengths versus the logarithm of the block (window) sizes provides an estimate for the Hurst exponent H .

Peng or Variance of Residuals Method:

The function `pengFit` uses the method described by peng. In Peng's variance of residuals method the series is also divided into blocks of size m . Within each block the cumulated sums are computed up to t and a least-squares line $a + b \cdot t$ is fitted to the cumulated sums. Then the sample variance of the residuals is computed which is proportional to m^{2H} . The "mean" or "median" are computed over the blocks. The slope $\beta = 2H$ from the least square provides an estimate for the Hurst exponent H .

The R/S Method:

The function `rsFit` implements the algorithm named *rescaled range analysis* which is discussed for example in detail by B. Mandelbrot and Wallis [199x], B. Mandelbrot [199x] and B. Mandelbrot and M.S. Taqqu [199x].

The Periodogram Method:

The function `perFit` estimates the Hurst exponent from the periodogram. In the finite variance case, the periodogram is an estimator of the spectral density of the time series. A series with long range dependence will show a spectral density with a lower law behavior in the frequency. Thus, we expect that a log-log plot of the periodogram versus frequency will display a straight line, and the slope can be computed as $1-2H$. In practice one uses only the lowest 10% of the frequencies, since the power law behavior holds only for frequencies close to zero. Varying this cut off may provide additional information. Plotting H versus the cut off, one should select that cut off where the curve flattens out to estimate H . This approach can be selected by the argument `method="per"`. Alternatively we can select `method="cumper"`. In this case, instead of using the periodogram itself, the cumulative periodogram will be investigated. The slope of the double logarithmic fit is given by $2-2H$. More details can be found in the work of J. Geweke and S. Porter-Hudak [1983] and in Taqqu [?].

The Boxed or Modified Periodogram Method:

The function `boxperFit` is a modification of the periodogram method. The algorithm divides the frequency axis into logarithmically equally spaced boxes and averages the periodogram values corresponding to the frequencies inside the box.

The Whittle Estimator:

The function `whittleFit` performs also a periodogram analysis. The algorithm is based on the minimization of a likelihood function defined in the frequency domain. For FGN and FARIMA(0,d,0) processes the parameter H or d is the unknown parameter which minimizes the function. This approach also allows to compute confidence intervals. Unlike the previous eight estimators the Whittle estimator is not a graphical method, it just returns the values of H or d together with their confidence intervals. The function allows also to investigate FARIMA(p,d,q) models, then the parameter set to be optimized is enlarged by the AR and MA coefficients. It is worth to remark, that the empirical series is required to be a Gaussian process and that the underlying form must be specified.

The original functions were written by V. Teverovsky and W. Willinger for SPLUS calling internal functions written in C. The software can be found on M. Taqqu's home page:

<http://math.bu.edu/people/murad/>

In addition the Whittle estimator uses SPlus functions written by J. Beran. They can be found in the appendix of his book or on the StatLib server:

<http://lib.stat.cmu.edu/S/>

Note, all nine R functions and internal utility functions are reimplemented entirely in S.

IV: Functions to perform a Wavelet Analysis:

The function `waveletFit` computes the Discrete Wavelet Transform, averages the squares of the coefficients of the transform, and then performs a linear regression on the logarithm of the average, versus the log of the scale parameter of the transform. The result should be directly proportional to H providing an estimate for the Hurst exponent.

Value

`fgnSim` and `farimaSim` return a numeric vector of length n , the FGN or FARIMA series.

`fgnTrueacf(n, H)`, `fgnTruefft(n, H)`, `farimaTrueacf(n, H)`, and `farimaTruefft(n, H)` return the true covariance and the true spectrum of the FGN or FARIMA time series process.

*Fit returns an S4 object of class `fHURST` with the following slots:

```
@call          the function call.
@method        a character string with the selected method string.
@hurst         a list with at least one element, the Hurst exponent named H. Optional values
               may be the value of the fitted slope beta, or information from the fit.
@parameters    a list with a varying number of elements describing the input parameters from
               the argument list.
@data          a list holding the input data.
@fit           a list object with all information of the fit.
@plot          a list object which holds information to create a plot of the fit.
@title         a character string with the name of the test.
@description   a character string with a brief description of the test.

waveletFit
```

Author(s)

V. Paxson, code as listed in the Appendix of his paper 1995,
 J. Beran, ported by Maechler, code as listed in the Appendix of his Book,
 M.S. Taqqu et al. for the S-Plus and C code concerned with the Hurst exponent,
 C. Fraley for the FARIMA simulation code,
 Guy Nason for the functions from the R package 'wavethresh',
 Diethelm Wuertz for the Rmetrics R-port.

References

Beran J. (1992); *Statistics for Long-Memory Processes*, Chapman and Hall, New York, 1994.
 Haslett J., Raftery A.E. (1989); *Space-Time Modelling with Long-Memory Dependence: Assessing Ireland's Wind Power Resource*, Applied Statistics 38, pp. 1–50.
 Paxson V. (1995); *Fast Approximation of Self-Similar Network Traffic*, Technical report, LBL-36750/UC-405, Berkeley, and Computer Communication Review 27, p.5–18, 1997.
 Rolls D.A. (2001); *Improved Fast Approximate Synthesis of Fractional Gaussian Noise*, Thesis, Department of Mathematics and Statistics, Queen's University at Kingston, Kingston, Ontario, Canada, 5 pages.
 Taqqu M., et al. *Hurst Exponent*, Several Preprints.

Examples

```
## SOURCE("fSeries.3A-LongRangeDependence")

## fgnSim -
par(mfrow = c(3, 1), cex = 0.75)

# Beran's Method:
plot(fgnSim(n = 200, H = 0.75), type = "l",
     ylim = c(-3, 3), xlab = "time", ylab = "x(t)", main = "Beran")

# Durbin's Method:
plot(fgnSim(n = 200, H = 0.75, method = "durbin"), type = "l",
```

```

ylim = c(-3, 3), xlab = "time", ylab = "x(t)", main = "Durbin")

# Paxson's Method:
plot(fgnSim(n = 200, H = 0.75, method = "paxson"), type = "l",
     ylim = c(-3, 3), xlab = "time", ylab = "x(t)", main = "Paxson")

```

GarchDistributions *GARCH Distributions and Parameter Estimation*

Description

A collection and description of functions to compute density, distribution function, quantile function and to generate random variates for the skew normal, the skew Student-t, and skew generalized error distribution. In addition maximum likelihood estimators are available to fit the parameters of a distribution and to compute basic statistical properties.

The functions are:

[dpqr]norm	Normal distribution from R's base package,
[dpqr]snorm	Skew Normal distribution,
[dpqr]std	Symmetric Student-t Distribution,
[dpqr]sstd	Skew Student-t Distribution,
[dpqr]ged	Symmetric GED distribution,
[dpqr]sged	Skew GED distribution.

The estimators are:

normFit	MLE parameter fit for a Normal distribution,
snormFit	MLE parameter fit for a skew Normal distribution,
stdFit	MLE parameter fit for a Student-t distribution,
sstdFit	MLE parameter fit for a skew Student-t distribution,
gedFit	MLE parameter fit for a generalized error distribution,
nigFit	MLE parameter fit for a skew generalized error distribution.

Utility Function:

absMoments	Computes absolute moments of a symmetric density.
------------	---

Usage

```

dsnorm(x, mean = 0, sd = 1, xi = 1.5)
psnorm(q, mean = 0, sd = 1, xi = 1.5)
qsnorm(p, mean = 0, sd = 1, xi = 1.5)
rsnorm(n, mean = 0, sd = 1, xi = 1.5)

dstd(x, mean = 0, sd = 1, nu = 5)
pstd(q, mean = 0, sd = 1, nu = 5)
qstd(p, mean = 0, sd = 1, nu = 5)
rstd(n, mean = 0, sd = 1, nu = 5)

```

```

dsstd(x, mean = 0, sd = 1, nu = 5, xi = 1.5)
psstd(q, mean = 0, sd = 1, nu = 5, xi = 1.5)
qsstd(p, mean = 0, sd = 1, nu = 5, xi = 1.5)
rsstd(n, mean = 0, sd = 1, nu = 5, xi = 1.5)

dged(x, mean = 0, sd = 1, nu = 2)
pged(q, mean = 0, sd = 1, nu = 2)
qged(p, mean = 0, sd = 1, nu = 2)
rged(n, mean = 0, sd = 1, nu = 2)

dsged(x, mean = 0, sd = 1, nu = 2, xi = 1.5)
psged(q, mean = 0, sd = 1, nu = 2, xi = 1.5)
qsged(p, mean = 0, sd = 1, nu = 2, xi = 1.5)
rsged(n, mean = 0, sd = 1, nu = 2, xi = 1.5)

normFit(x, ...)
snormFit(x, ...)
stdFit(x, ...)
sstdFit(x, ...)
gedFit(x, ...)
sgedFit(x, ...)

absMoments(n, density = c("dnorm", "dged", "dstd"), ...)

```

Arguments

<code>density</code>	[absMoments] - a character string naming the symmetric density function.
<code>mean, sd, nu, xi</code>	location parameter <code>mean</code> , scale parameter <code>sd</code> , shape parameter <code>nu</code> , skewness parameter <code>xi</code> .
<code>n</code>	[rnorm][r*ged][r*std] - the number of observations. [absMoments] - the number of absolute Moments.
<code>p</code>	a numeric vector of probabilities.
<code>x, q</code>	a numeric vector of quantiles.
<code>...</code>	[*Fit] - parameters parsed to the optimization function <code>nlm</code> . [absMoments] - parameters passed to the density function.

Details

Symmetric Normal Distribution:

The functions for the normal distribution are part of R's base package. The functions for the symmetric Student-t distribution are rescaled in such a way that they have unit variance in contrast to the Student-t family `dt`, `pt`, `qt` and `rt` which are part of R's base package. The generalized error distribution functions are defined as described by Nelson (1991).

Skew Normal Distribution:

The skew normal distribution functions are defined as described by Fernandez and Steel (2000).
`cr`

Parameter Estimation:

The function `nlm` is used to minimize the "negative" maximum log-likelihood function. `nlm` carries out a minimization using a Newton-type algorithm.

Value

`d*` returns the density, `p*` returns the distribution function, `q*` returns the quantile function, and `r*` generates random deviates, all values are numeric vectors.

`*Fit` return a list with the following components:

<code>estimate</code>	the point at which the maximum value of the log likelihood function is obtained.
<code>objective</code>	the value of the estimated maximum, i.e. the value of the log likelihood function.
<code>message</code>	an integer indicating why the optimization process terminated.
<code>code</code>	an integer indicating why the optimization process terminated. 1: relative gradient is close to zero, current iterate is probably solution; 2: successive iterates within tolerance, current iterate is probably solution; 3: last global step failed to locate a point lower than <code>estimate</code> . Either <code>estimate</code> is an approximate local minimum of the function or <code>steptol</code> is too small; 4: iteration limit exceeded; 5: maximum step size <code>stepmax</code> exceeded five consecutive times. Either the function is unbounded below, becomes asymptotic to a finite value from above in some direction or <code>stepmax</code> is too small.
<code>gradient</code>	the gradient at the estimated maximum.
<code>steps</code>	number of function calls.

`absMoments` returns a numeric vector of length `n` with the values of the absolute moments of the density function.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Nelson D.B. (1991); *Conditional Heteroscedasticity in Asset Returns: A New Approach*, *Econometrica*, 59, 347–370.

Fernandez C., Steel M.F.J. (2000); *On Bayesian Modelling of Fat Tails and Skewness*, Preprint, 31 pages.

Examples

```
## SOURCE("fSeries.4A-GarchDistributions")

## snorm -
par(mfrow = c(2, 2), cex = 0.75)
```

```

set.seed(1953)
r = rsnorm(n = 1000, mean = 1, sd = 0.5, xi = 1.5)
plot(r, type = "l", main = "snorm: xi = 1.5")
# Plot empirical density and compare with true density:
hist(r, n = 25, probability = TRUE, border = "white", col = "steelblue")
x = seq(-4, 6, 0.1)
lines(x, dsnorm(x = x, mean = 1, sd = 0.5, xi = 1.5))
# Plot df and compare with true df:
plot(sort(r), (1:1000/1000), main = "Probability", col = "steelblue")
lines(x, psnorm(x, mean = 1, sd = 0.5, xi = 1.5))
# Compute quantiles:
qsnorm(psnorm(q = -4:6, mean = 1, sd = 0.5, xi = 1.5),
       mean = 1, sd = 0.5, xi = 1.5)

## sstd -
par(mfrow = c(2, 2), cex = 0.75)
set.seed(1953)
r = rsstd(n = 1000, nu = 4, xi = 1.5)
# Print Variance:
var(r)
plot(r, type = "l", main = "sstd: xi = 1.5")
# Plot empirical density and compare with true density:
hist(r, n = 30, xlim = c(-5, 5), probability = TRUE,
     border = "white", col = "steelblue")
x = seq(-5, 5, 0.1)
lines(x, dsnorm(x = x, xi = 1.5))
# Plot df and compare with true df:
plot(sort(r), (1:1000/1000), main = "Probability", col = "steelblue")
lines(x, psstd(x, xi = 1.5))
# Compute quantiles:
qsstd(psst(x = -5:5, xi = 1.5), xi = 1.5)

## sged -
par(mfrow = c(2, 2), cex = 0.75)
set.seed(1953)
r = rsged(n = 1000, mean = 1, sd = 0.5, xi = 1.5)
plot(r, type = "l", main = "sged: xi = 1.5")
# Plot empirical density and compare with true density:
hist(r, n = 25, probability = TRUE, border = "white", col = "steelblue")
x = seq(-1, 5, 0.1)
lines(x, dsGED(x = x, mean = 1, sd = 0.5, xi = 1.5))
# Plot df and compare with true df:
plot(sort(r), (1:1000/1000), main = "Probability", col = "steelblue")
lines(x, psGED(x, mean = 1, sd = 0.5, xi = 1.5))
# Compute quantiles:
qsged(psged(q = -1:5, mean = 1, sd = 0.5, xi = 1.5),
      mean = 1, sd = 0.5, xi = 1.5)

## snormFit -
options(warn = -1) # suppress negative logs from nlm
normFit(rnorm(1000))
snormFit(rnorm(1000))

## sstdFit -
sstdFit(rsstd(1000, mean = 1, sd = 1.5, nu = 7, xi = 2/3))

## sgedFit -

```

```
sgedFit(rsged(1000, mean = -1, sd = 0.5, nu = 3, xi = 3/2),
print.level = 2)
```

HeavisideFunction *Heaviside and Related Functions*

Description

A collection and description of functions which compute the Heaviside and related functions. These include the sign function, the delta function, the boxcar function, and the ramp function.

The functions are:

Heaviside	Computes Heaviside unit step function,
Sign	Just another signum function,
Delta	Computes delta function,
Boxcar	Computes boxcar function,
Ramp	Computes ramp function.

Usage

```
Heaviside(x, a = 0)
Sign(x, a = 0)
Delta(x, a = 0)
Boxcar(x, a = 0.5)
Ramp(x, a = 0)
```

Arguments

a	a numeric value, the location of the break.
x	a numeric vector.

Details

The Heaviside step function `Heaviside` is 1 for $x > a$, $1/2$ for $x = a$, and 0 for $x < a$.

The Sign function `Sign` is 1 for $x > a$, 0 for $x = a$, and -1 for $x < a$.

The delta function `Delta` is defined as: $\text{Delta}(x) = d/dx H(x-a)$.

The boxcar function `Boxcar` is defined as: $\text{Boxcar}(x) = H(x+a) - H(x-a)$.

The ramp function is defined as: $\text{Ramp}(x) = (x-a) * H(x-a)$.

Value

returns the function values of the selected function.

Note

The Heaviside function is used in the implementation of the skew Normal, Student-t, and Generalized Error distributions, distributions functions which play an important role in modelling GARCH processes.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Weisstein W. (2004); <http://mathworld.wolfram.com/HeavisideStepFunction.html>, Mathworld.

See Also

GarchDistribution, GarchDistributionFits.

Examples

```
## SOURCE("fSeries1.4B-HeavisideFunction")

## Heaviside -
x = sort(round(c(-1, -0.5, 0, 0.5, 1, 5*rnorm(5)), 2))
h = Heaviside(x)

## Sign -
s = Sign(x)

## Delta -
d = Delta(x)

## Boxcar -
Pi = Boxcar(x)

## ramp -
r = Ramp(x)
cbind(x = x, Step = h, Signum = s, Delta = d, Pi = Pi, R = r)
```

Description

A collection and description of functions to simulate artificial ARCH time series processes, to fit the parameters of univariate time series to ARCH models, to perform a diagnostic analysis of the fit, and to predict future values of the time series.

The family of GARCH time series models includes the following processes:

garch	generalized AR conditional heteroskedastic models,
aparch	asymmetric power GARCH models.

Included functions are:

<code>garchSpec</code>	Specifies a GARCH model,
<code>garchSim</code>	simulates a GARCH process,
<code>garchFit</code>	fits parameters of a GARCH model,
<code>garchKappa</code>	computes ARCH expectation value.

Included methods are:

<code>print</code>	Prints results from a GARCH fit,
<code>summary</code>	creates a summary report from a GARCH fit,
<code>plot</code>	creates diagnostic plots for a fitted GARCH process,
<code>residuals</code>	returns residuals for a GARCH model,
<code>fitted</code>	returns fitted values for a GARCH model,
<code>predict</code>	predicts mean and variance from a GARCH model.

Note: This collection is still under a complete reconstruction.

Usage

```
garchSpec(model = list(omega = 1.0e-6, alpha = 0.1, beta = 0.8),
  presample = NULL, cond.dist = c("rnorm", "rged", "rstd", "rsnorm",
  "rsged", "rsstd"), rseed = NULL)
## S3 method for class 'garchSpec':
print(x, ...)

garchSim(model = list(omega = 1.0e-6, alpha = 0.1, beta = 0.8), n = 100,
  n.start = 100, presample = NULL, cond.dist = c("rnorm", "rged", "rstd",
  "rsnorm", "rsged", "rsstd"), rseed = NULL)

garchFit(formula, data, init.rec = c("mci", "uev"), delta = 2, skew = 1,
  shape = 4, cond.dist = c("dnorm", "dsnrm", "dged", "dsged", "dstd", "dsstd"),
  include.mean = TRUE, include.delta = NULL, include.skew = NULL,
  include.shape = NULL, leverage = NULL, trace = TRUE,
  algorithm = c("sqp", "nlminb", "lbfgsb", "nlminb+nm", "lbfgsb+nm"),
  control = list(), title = NULL, description = NULL, ...)

garchKappa(cond.dist = c("dnorm", "dged", "dstd", "dsnrm", "dsged", "dsstd"),
  gamma = 0, delta = 2, skew = NA, shape = NA)

## S3 method for class 'fGARCH':
print(x, ...)
## S3 method for class 'fGARCH':
plot(x, which = "ask", ...)
## S3 method for class 'fGARCH':
summary(object, ...)
## S3 method for class 'fGARCH':
residuals(object, ...)
## S3 method for class 'fGARCH':
fitted(object, ...)
## S3 method for class 'fGARCH':
predict(object, n.ahead = 10, trace = FALSE, ...)
```

Arguments

<code>algorithm</code>	[garchFit] - a string parameter that determines the algorithm used for maximum likelihood estimation. Allowed values are "sqp", "nlminb", and "bfgs" where the first is the default setting.
<code>cond.dist</code>	[garchSpec, garchSim, garchFit] - a character string naming the desired conditional distribution. Valid values are "dnorm", "dged", "dstd", "dsnrm", "dsged", "dsstd". The default value is the normal distribution.
<code>control</code>	[garchFit] - control parameters, the same as used for the functions from <code>nlminb</code> , and 'bfgs' and 'Nelder-Mead' from <code>optim</code> .
<code>data</code>	an optional <code>timeSeries</code> or data frame object containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>armaFit</code> is called. If <code>data</code> is an univariate series, then the series is converted into a numeric vector and the name of the response in the formula will be neglected.
<code>delta, include.delta</code>	[garchFit][kappa] - the exponent <code>delta</code> of the variance recursion. By default, this value will be fixed, otherwise the exponent will be estimated together with the other model parameters if <code>include.delta=FALSE</code> .
<code>description</code>	[garchFit] - a character string which allows for a brief description.
<code>formula</code>	[garchFit] - formula object describing the mean and variance equation of the ARMA-GARCH/APARCH model. A pure GARCH(1,1) model is selected when e.g. <code>formula=~garch(1,1)</code> . To specify for example an ARMA(2,1)-APARCH(1,1) use <code>formula = ~arma(2,1)+apaarch</code>
<code>gamma</code>	[kappa] - APARCH leverage parameter entering into the formula for calculating the expectation value.
<code>include.mean</code>	this flag determines if the parameter for the mean will be estimated or not. If <code>include.mean=TRUE</code> this will be the case, otherwise the parameter will be kept fixed during the process of parameter optimization.
<code>include.skew, include.shape</code>	this flag determines if the parameters for the skew and shape of the conditional distribution will be estimated or not. If <code>include.skew=TRUE</code> and/or <code>include.shape=TRUE</code> this will be the case, otherwise the parameters will be kept fixed during the process of parameter optimization.
<code>init.rec</code>	[garchFit] - a character string indicating the method how to initialize the mean and variance recursion relation.
<code>leverage</code>	a logical flag for APARCH models. Should the model be leveraged? By default <code>leverage=TRUE</code> .
<code>model</code>	[garchSpec, garchSim] - List of GARCH model parameters: omega - the constant coefficient of the variance equation; alpha - the vector of autoregressive coefficients; beta - the vector of variance coefficients;

Further Optional Values:

mu - the mean value;
 ar - the autoregressive ARMA coefficients;
 ma - the moving average ARMA coefficients;

The default model is Bollerslev's GARCH(1,1) model.

n	[garchSim] - length of output series, an integer value. An integer value, by default n=100.
n.ahead	[predict] - number of steps to be forecasted, an integer value, by default 10.
n.start	[garchSim] - length of "burn-in" period, by default 100.
object	[predict][summary] - an object of class fGARCH as returned from the function garchFit().
presample	a numeric three column matrix with start values for the series, innovations, and conditional variances. For an ARMA(m,n)-GARCH(p,q) process the number of rows must be at least max(m,n,p,q), longer presamples are cutted.
rseed	[garchSpec][garchSpec] - single integer argument, the seed for the initialization of the random number generator for the innovations.
skew, shape	[garchFit][kappa] - skewness and shape parameter of the conditional distribution.
title	[garchFit][predict] - a character string which allows for a project title.
trace	[garchFit] - a logical flag. Should the optimization process of fitting the model parameters be printed? By default trace=TRUE. [predict] - a logical flag. Should the prediction process be printed? By default trace=FALSE.
which	[plot] - if which is set to "ask" the function will interactively ask which plot should be displayed. This is the default value and then those plots will be displayed for which the elements in the logical vector which are set to TRUE; by default all four elements are set to "all".
x	[print][plot] - either an object of class garchSpec for printing specification structures, or an object of class fGARCH for printing fitted GARCH/APARCH models or plotting results from the diagnostic analysis of fitted models.
...	additional arguments to be passed.

Details**Parameter Estimation:**

garchFit uses the nlminb() optimizer to find the maximum likelihood estimates.

Residuals Methods:

The method for the computation of the residuals may have an additional argument named standardize which allows to standardize the residuals by the conditional variances.

Value

`garchSpec`

returns a S4 object of class `fGARCH` with the following slots:

<code>@call</code>	the call of the <code>garch</code> function.
<code>@formula</code>	a list with two formula entries for the mean and variance equation.
<code>@model</code>	a list with the model parameters.
<code>@presample</code>	a numeric matrix with presample values.
<code>@distribution</code>	a character string with the name of the conditional distribution.
<code>@call</code>	the call of the <code>garch</code> function.
<code>@formula</code>	a list with two formula entries, one for the mean and the other one for the variance equation.
<code>@method</code>	a string denoting the optimization method, by default the returned string is "Max Log-Likelihood Estimation".
<code>@data</code>	a list with one entry named <code>x</code> , containing the data of the time series to be estimated, the same as given by the input argument <code>series</code> .
<code>@fit</code>	a list with the results from the parameter estimation. The entries of the list depend on the selected algorithm, see below.
<code>@residuals</code>	a numeric vector with the residual values.
<code>@fitted.values</code>	a numeric vector with the fitted values.
<code>@cvariances</code>	a numeric vector with the conditional variances.
<code>@title</code>	a title string.
<code>@description</code>	a string with a brief description.
<code>\$par</code>	The best set of parameters found.
<code>\$value</code>	The value of the max log likelihood function corresponding to the optimal values of <code>\$par</code> .
<code>\$counts</code>	A two-element integer vector giving the number of calls to 'fn' and 'gr' respectively. This excludes those calls needed to compute the Hessian, if requested, and any calls to 'fn' to compute a finite-difference approximation to the gradient.
<code>\$convergence</code>	An integer code. '0' indicates successful convergence. Values different from zero indicate an error code. For details, please consult the help page of <code>optim</code> .
<code>message</code>	
<code>hessian</code>	Only if control list argument <code>hessian</code> is true. A symmetric matrix giving an estimate of the Hessian at the solution found. Note that this is the Hessian of the unconstrained problem even if the box constraints are active.
<code>\$par</code>	The best set of parameters found.
<code>\$value</code>	The value of the max log likelihood function corresponding to the optimal values of <code>\$par</code> .

Note**What's next?**

The software will be extended to further GARCH models including integrated GARCH models, EGARCH models, GARCH-in-Mean models as well as fractionally integrated GARCH models. In the next version also the fixing of individual ARMA-GARCH/APARCH coefficients will become available. Furthermore, we will provide additional conditional distribution functions including for example members from the family of the hyperbolic distribution. Concerning performance analysis we plan to offer in addition several performance measures for the residuals like Theil's inequality coefficient, and Mincer and Zarnowitz's R2 coefficient. Concerning hypothesis testing we will add some specific tests including Engle's LM ARCH test to test the presence of ARCH effects, Engle and Ng's test to investigate possible misspecifications of the conditional variance equation, and Nyblom's test to check the constancy of model parameters over time.

Author(s)

Diethelm Wuertz for the Rmetrics R-port,
 R Core Team for the 'optim' R-port,
 Douglas Bates and Deepayan Sarkar for the 'nlminb' R-port,
 Bell-Labs for the underlying PORT Library,
 Ladislav Luksan for the underlying SQP Routine,
 Zhu, Byrd, Lu-Chen and Nocedal for the underlying L-BFGS-B Routine.

References

- ATT (1984); *PORT Library Documentation*, <http://netlib.bell-labs.com/netlib/port/>.
- Bera A.K., Higgins M.L. (1993); *ARCH Models: Properties, Estimation and Testing*, J. Economic Surveys 7, 305–362.
- Bollerslev T. (1986); *Generalized Autoregressive Conditional Heteroscedasticity*, Journal of Econometrics 31, 307–327.
- Byrd R.H., Lu P., Nocedal J., Zhu C. (1995); *A Limited Memory Algorithm for Bound Constrained Optimization*, SIAM Journal of Scientific Computing 16, 1190–1208.
- Engle R.F. (1982); *Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation*, Econometrica 50, 987–1008.
- Nash J.C. (1990); *Compact Numerical Methods for Computers*, Linear Algebra and Function Minimisation, Adam Hilger.
- Nelder J.A., Mead R. (1965); *A Simplex Algorithm for Function Minimization*, Computer Journal 7, 308–313.
- Nocedal J., Wright S.J. (1999); *Numerical Optimization*, Springer, New York.

Examples

```
## SOURCE("fSeries1.4C-GarchModelling")

# garchSpec -
# garchSim -
# garchFit -

# For examples we refer to: demo/xmpDWChapter34.R ...
```

GarchOxInterface *R Interface for Garch Ox*

Description

A collection and description of functions to fit the parameters of an univariate time series to GARCH models interfacing the G@RCH Ox Package.

The family of GARCH time series models includes the following processes:

- 1 garch generalized AR conditional heteroskedastic models,
- 2 egarch exponential GARCH models,
- 3 aparch asymmetric power ARCH models.

Usage

```
garchOxFit(formula, data, cond.dist = c("gaussian", "t", "ged", "skewed-t"),
  include.mean = TRUE, trace = TRUE, control = list(), title = NULL,
  description = NULL)

## S3 method for class 'garchOx':
print(x, digits, ...)
## S3 method for class 'garchOx':
summary(object, ...)
## S3 method for class 'garchOx':
plot(x, ...)
```

Arguments

<code>cond.dist</code>	a character string describing the distribution of innovations. By default the optimization is based on gaussian log likelihood parameter optimization denoted by "gaussian". Alternatively, a Student-t "t", a generalized error "sged", or a skewed Student-t "skewed-t" can be chosen.
<code>control</code>	a list of additional control parameters: <code>truncation</code> - the number of truncation points, by default 100, <code>xscale</code> - should the time series be scaled by the standard deviation ?
<code>data</code>	an optional timeSeries or data frame object containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>armaFit</code> is called. If <code>data</code> is an univariate series, then the series is converted into a numeric vector and the name of the response in the formula will be neglected.
<code>description</code>	a character string which allows for a brief description.
<code>digits</code>	the number of digits to be printed.
<code>formula</code>	[<code>garchFit</code>] - formula object describing the mean and variance equation of the ARMA-GARCH/APARCH model. A pure GARCH(1,1) model is selected when e.g. <code>formula=~garch(1,1)</code> . To specify for example an ARMA(2,1)-APARCH(1,1) use <code>formula = ~arma(2,1)+aparch</code>
<code>include.mean</code>	should the mean be included? By default TRUE.

<code>object</code>	an object of class <code>garchOx</code> as returned from the function <code>codegarchOxFit</code> .
<code>title</code>	a character string which allows for a project title.
<code>trace</code>	a logical flag. Should the estimation process be traced? By default TRUE.
<code>x</code>	an object of class <code>garchOx</code> as returned from the function <code>garchOxFit</code> .
<code>...</code>	additional arguments to be passed to the <code>print</code> , <code>summary</code> , and <code>plot</code> methods.

Details

Ox Interface:

The function `garchOxFit` interfaces a subset of the functionality of the G@ARCH 4.0 Package written in Ox. G@RCH 4.0 is one of the most sophisticated packages for modelling univariate GARCH processes including GARCH, EGARCH, GJR, APARCH, IGARCH, FIGARCH, FIEGARCH, FIAPARCH and HYGARCH models. Parameters can be estimated by approximate (Quasi-) maximum likelihood methods under four assumptions: normal, Student-t, GED or skewed Student-t errors.

About Ox:

Ox (tm) is an object-oriented matrix language with a comprehensive mathematical and statistical function library. Many packages were written for Ox including software mainly for econometric modelling. The Ox packages for time series analysis and forecasting, Arfima, Doornik and Ooms [2003], Garch, Laurent and Peters [2005], and State Space Modelling, Koopman, Shepard and Doornik [1998], are especially worth to note. Since most of the R-users wan't to change to another Statistical Computing environment, we made selected parts of the G@RCH Ox software available for them through an R-Interface. What you have to do, is to read carefully the "Ox citation and copyright" rules and if you agree and fulfill the conditions, then download the Ox-Console Software together with the "OxGarch" Package, currently G@RCH 4.0. If you are not qualified for a free license, order your copy from Timberlake Consultants. We recommend to install the "Setup.exe" under the path "C:\Ox\Ox3" and to unzip the OxGarch Package in the directory "C:\Ox\Ox3\Packages". An Update to Ox4 has not yet be done.

Distribution:

Ox and G@RCH are distributed by Timberlake Consultants Ltd. Timberlake Consultants can be contacted through the following web site: www.timberlake.co.uk.

Installation of the Interface:

In addition you have to copy the file "GarchOxModelling.ox" (which is the interface written especially for Rmetrics) from the "fSeries/ox/" directory to the Ox library directory "C:\Ox\lib".

Ox Citation and Copyright Rules:

Ox and all its components are copyright of Jurgen A. Doornik. The Console (command line) versions may be used freely for academic research and teaching purposes only. Commercial users and others who do not qualify for the free version must purchase the Windows version of Ox and GiveWin with documentation, regardless of which version they use (so even when only using Ox on Linux or Unix). Ox should be cited whenever it is used. Refer to the two references given below. Note, failure to cite the use of Ox in published work may result in loss of the right to use the free version, and an invoice at the full commercial price. Ox is available from Timberlake Consultants. The Ox syntax is public, and you may do with your own Ox code whatever you wish, including the file "GarchOxModelling.ox".

Work to do:

Note, only a small part of the functionalities are interfaced until now to R. But, principally it would be possible to interface also other functionalities offered by the Ox Garch Package. This work is left to the Ox/Rmetrics user.

Author(s)

Jurgen A. Doornik for the Ox Environment, www.doornik.com,
 Sebastian Laurent for the Ox Garch package, www.garch.org,
 Diethelm Wuertz for R's Ox Garch interface.

References

- Doornik J.A. (2002), Object-Oriented Matrix Programming Using Ox, London, 3rd ed.: Timberlake Consultants Press and Oxford: www.doornik.com.
- Doornik J.A., Ooms M. (2003), Computational Aspects of Maximum Likelihood Estimation of Autoregressive Fractionally Integrated Moving Average Models, *Computational Statistics and Data Analysis* 42, 333–348.
- Koopman J.S., Shepard N., Doornik J.A. (1999), Statistical Algorithms for Models in State Space using SsfPack 2.2, *Econometrics Journal* 2, 113–166.
- Laurent S., Peters J.P. (2002); G@RCH 2.2: An Ox Package for Estimating and Forecasting Various ARCH Models, *Journal of Economic Surveys* 16, 447–485.
- Laurent S., Peters J.P., [2005], G@RCH 4.0, Estimating and Forecasting ARCH Models, Timberlake Consultants, www.timberlake.co.uk

Examples

```
## SOURCE("fSeries1.4D-GarchOxModelling")

## Not run:
## garchOxFit -

# Load Benchmark Data Set:
data(dem2gbp)
x = dem2gbp[, 1]

# Fit GARCH(1,1):
garchOxFit(formula.mean = ~arma(0,0), formula.var = ~garch(1,1))
## End(Not run)
```

ChaoticTimeSeries *Chaotic Time Series Modelling*

Description

A collection and description of functions to investigate the chaotic behavior of time series processes. Included are functions to simulate different types of chaotic time series maps.

Chaotic Time Series Maps:

tentSim	Simulates data from the Tent Map,
---------	-----------------------------------

henonSim	simulates data from the Henon Map,
ikedaSim	simulates data from the Ikeda Map,
logisticSim	simulates data from the Logistic Map,
lorentzSim	simulates data from the Lorentz Map,
roesslerSim	simulates data from the Roessler Map.

Functions to Analyse Chaotic Time Series:

mutualPlot	Returns mutual information,
falsennPlot	returns false nearest neighbours,
recurrencePlot	returns a recurrence plot,
separationPlot	returns a space-time separation plot,
lyapunovPlot	computes maximum lyapunov exponent.

Usage

```
tentSim(n = 1000, n.skip = 100, parms = c(a = 2), start = runif(1),
        doplot = FALSE)
henonSim(n = 1000, n.skip = 100, parms = c(a = 1.4, b = 0.3),
         start = runif(2), doplot = FALSE)
ikedaSim(n = 1000, n.skip = 100, parms = c(a = 0.4, b = 6.0, c = 0.9),
         start = runif(2), doplot = FALSE)
logisticSim(n = 1000, n.skip = 100, parms = c(r = 4), start = runif(1),
            doplot = FALSE)
lorentzSim(times = seq(0, 40, by = 0.01), parms = c(sigma = 16, r = 45.92,
            b = 4), start = c(-14, -13, 47), doplot = TRUE, ...)
roesslerSim(times = seq(0, 100, by = 0.01), parms = c(a = 0.2, b = 0.2, c = 8.0)
            start = c(-1.894, -9.920, 0.0250), doplot = TRUE, ...)
```

```
mutualPlot(x, partitions = 16, lag.max = 20, doplot = TRUE, ...)
falsennPlot(x, m, d, t, rt = 10, eps = NULL, doplot = TRUE, ...)
recurrencePlot(x, m, d, end.time, eps, nt = 10, doplot = TRUE, ...)
separationPlot(x, m, d, mdt, idt = 1, doplot = TRUE, ...)
lyapunovPlot(x, m, d, t, ref, s, eps, k = 1, doplot = TRUE, ...)
```

Arguments

d	[*Plot] - an integer value setting the value of the time delay.
eps	[falsennPlot] - a numeric value setting the value of the neighbour diameter. If NULL, which is the default value, then the value will be automatically setted to $\text{eps} = \text{sd}(x) / 10$. [lyapunovPlot] - the radius where to find nearest neighbours.
doplot	[recurrencePlot] - the neighbourhood threshold. a logical flag. Should a plot be displayed? [*Plot] - By default TRUE.
end.time	[*Sim] - By default TRUE. [recurrencePlot] - ending time as number of observations.

<code>idt</code>	[<code>separationPlot</code>] - an integer value setting the number of observation steps in each iterations. By default 1.
<code>k</code>	[<code>lyapunovPlot</code>] - an integer setting the number of considered neighbours. By default 1.
<code>lag.max</code>	[<code>mutualPlot</code>] - an integer value setting the number of maximum lags, by default 20.
<code>m</code>	[* <code>Plot</code>] - an integer value setting the value of the maximum embedding dimension.
<code>mdt</code>	[<code>separationPlot</code>] - an integer value setting the number of iterations.
<code>n, n.skip</code>	[<code>henonSim</code>][<code>ikedaSim</code>][<code>logisticSim</code>] - the number of chaotic time series points to be generated and the number of initial values to be skipped from the series.
<code>nt</code>	[<code>recurrencePlot</code>] - observations in each step which will be plotted, by default 10. Increasing <code>nt</code> reduces number of points plotted which is useful especially with highly sampled data.
<code>parms</code>	the named parameter vector characterizing the chaotic map.
<code>rt</code>	[<code>falsennPlot</code>] - an integer value setting the value for the escape factor. By default 10.
<code>partitions</code>	[<code>mutualPlot</code>] - an integer value setting the number of bins, by default 16.
<code>ref</code>	[<code>lyapunovPlot</code>] - the number of points to take into account.
<code>s</code>	[<code>lyapunovPlot</code>] - the iterations along which follow the neighbours of each point.
<code>start</code>	the vector of start values to initiate the chaotic map.
<code>t</code>	[* <code>Plot</code>] - an integer value setting the value for the Theiler window.
<code>times</code>	[<code>lorenzSim</code>][<code>roesslerSim</code>] - the sequence of time series points at which to generate the map.
<code>x</code>	[* <code>Plot</code>] - a numeric vector, or an object either of class 'ts' or of class 'timeSeries'.
<code>...</code>	arguments to be passed.

Details

Phase Space Representation:

The function `mutualPlot` estimates and plots the mutual information index of a given time series for a specified number of lags. The joint probability distribution function is estimated with a simple bi-dimensional density histogram.

The function `falsennPlot` uses the Method of false nearest neighbours to help deciding the optimal embedding dimension.

Non-Stationarity:

The function `recurrencePlot` creates a recurrence plot as proposed by Eckmann et al. [1987]. The function `separationPlot` creates a space-time separation plot qs introduced by Provenzale et al. [1992]. It plots the probability that two points in the reconstructed phase-space have distance smaller than epsilon in function of epsilon and of the time between the points, as iso-lines at levels 10, 20, ..., 100 percent levels. The plot can be used to decide the Theiler time window.

Lyapunov Exponents:

The function `lyapunovPlot` evaluates and plots the largest Lyapunov exponent of a dynamic system from a univariate time series. The estimate of the Lyapunov exponent uses the algorithm of Kantz. In addition, the function computes the regression coefficients of a user specified segment of the sequence given as input.

Dimensions and Entropies:

The function `C2` computes the sample correlation integral on the provided time series for the specified length scale and Theiler window. It uses a naive algorithm: simply returns the fraction of points pairs nearer than eps. It is preferable to use the function `d2`, which takes roughly the same time, but computes the correlation sum for multiple length scales and embedding dimensions at once.

The function `d2` computes the sample correlation integral over given length scales `neps` for embedding dimensions `1 : m` for a given Theiler window. The slope of the linear segment in the log-log plot gives an estimate of the correlation dimension.

Value

*Sim -

All functions return invisible a vector of time series data.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

- Brock, W.A., Dechert W.D., Sheinkman J.A. (1987); *A Test of Independence Based on the Correlation Dimension*, SSRI no. 8702, Department of Economics, University of Wisconsin, Madison.
- Eckmann J.P., Oliffson Kamphorst S., Ruelle D. (1987), *Recurrence plots of dynamical systems*, Europhys. Letters 4, 973.
- Hegger R., Kantz H., Schreiber T. (1999); *Practical implementation of nonlinear time series methods: The TISEAN package*, CHAOS 9, 413–435.
- Kennel M.B., Brown R., Abarbanel H.D.I. (1992); *Determining embedding dimension for phase-space reconstruction using a geometrical construction*, Phys. Rev. A45, 3403.
- Rosenstein M.T., Collins J.J., De Luca C.J. (1993); *A practical method for calculating largest Lyapunov exponents from small data sets*, Physica D 65, 117.

See Also

RandomInnovations.

Examples

```

## SOURCE("fSeries.035A-ChaoticTimeSeries")

## bdsTest -
# iid Time Series:
par(mfrow = c(3, 1))
x = rnorm(100)
plot(x, type = "l", main = "iid Time Series")
bdsTest(x, m = 3)
# Non Identically Distributed Time Series:
x = c(rnorm(50), runif(50))
plot(x, type = "l", main = "Non-iid Time Series")
bdsTest(x, m = 3)
# Non Independent Innovations from Quadratic Map:
x = rep(0.2, 100)
for (i in 2:100) x[i] = 4*(1-x[i-1])*x[i-1]
plot(x, type = "l", main = "Quadratic Map")
bdsTest(x, m = 3)

## tnnTest -
# Time Series Non-linear in "mean" regression
par(mfrow = c(2, 1))
n = 1000
x = runif(1000, -1, 1)
tnnTest(x)
# Generate time series which is nonlinear in "mean"
x[1] = 0.0
for (i in (2:n)) {
  x[i] = 0.4*x[i-1] + tanh(x[i-1]) + rnorm(1, sd = 0.5) }
plot(x, main = "Teraesvirta Test", type = "l")
tnnTest(x)

## wnnTest -
# Time Series Non-Linear in "mean" Regression
par(mfrow = c(2, 1))
n = 1000
x = runif(1000, -1, 1)
wnnTest(x)
# Generate time series which is nonlinear in "mean"
x[1] = 0.0
for (i in (2:n)) {
  x[i] = 0.4*x[i-1] + tanh(x[i-1]) + rnorm(1, sd = 0.5) }
plot(x, main = "White Test", type = "l")
wnnTest(x)

```

TimeSeriesTests *Time Series Tests*

Description

A collection and description of functions for testing various aspects of univariate time series, including independence, and neglected nonlinearities.

The family of time series tests includes the following hypothesis tests:

`bdsTest` Brock–Dechert–Scheinkman test for iid series,
`tnnTest` Teraesvirta NN test for neglected nonlinearity,
`wnnTest` White NN test for neglected nonlinearity.

Usage

```

bdsTest(x, m = 3, eps = NULL, title = NULL, description = NULL)
tnnTest(x, lag = 1, title = NULL, description = NULL)
wnnTest(x, lag = 1, qstar = 2, q = 10, range = 4, title = NULL, description = NU
  
```

Arguments

`description` optional description string, or a vector of character strings.
`eps` [`bdsTest`] -
 a numeric vector of epsilon values for close points. The BDS test is computed for each element of `eps`. It should be set in terms of the standard deviation of `x`. If `eps` is `NULL`, then the four default values `seq(0.5*sd(x), 2*sd(x), length = 4)` are used.
`lag` [`tnnTest`][`wnnTest`] -
 an integer which specifies the model order in terms of lags.
`m` [`bdsTest`] -
 an integer indicating that the BDS test statistic is computed for embedding dimensions $2, \dots, m$.
`q` [`wnnTest`] -
 an integer representing the number of phantom hidden units used to compute the test statistic.
`qstar` [`wnnTest`] -
 the test is conducted using `qstar` principal components of the phantom hidden units. The first principal component is omitted since in most cases it appears to be collinear with the input vector of lagged variables. This strategy preserves power while still conserving degrees of freedom.
`range` [`wnnTest`] -
 the input to hidden unit weights are initialized uniformly over $[-\text{range}/2, \text{range}/2]$.
`title` an optional title string, if not specified the inputs data name is deparsed.
`x` a numeric vector or an object of class "timeseries".

Details

Brock–Dechert–Sheinkman Test:

The `bdsTest` test examines the *spatial dependence* of the observed series. To do this, the series is embedded in m -space and the dependence of `x` is examined by counting *near* points. Points for which the distance is less than `eps` are called near. The BDS test statistic is asymptotically standard Normal. Note, that missing values are not allowed. There is a special print method for objects of class "bdsTest" which by default uses 4 digits to format real numbers.

```
[tseries:bds.test]
```

Teraesvirta Neural Network Test:

The null is the hypotheses of linearity in mean. This test uses a Taylor series expansion of the activation function to arrive at a suitable test statistic. If `type` equals "F", then the F-statistic instead of the Chi-Squared statistic is used in analogy to the classical linear regression. Missing values are not allowed.

```
[tseries:teraesvirta.test]
```

White Neural Network Test:

The null is the hypotheses of linearity in "mean". This type of test is consistent against arbitrary nonlinearity in mean. If `type` equals "F", then the F-statistic instead of the Chi-Squared statistic is used in analogy to the classical linear regression.

```
[tseries:white.test]
```

Value

In contrast to R's output report from S3 objects of class "htest" a different output report is produced. The tests here return an S4 object of class "fHTEST". The object contains the following slots:

<code>@call</code>	the function call.
<code>@data</code>	the data as specified by the input argument(s).
<code>@test</code>	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class "htest".
<code>@title</code>	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
<code>@description</code>	a character string with an optional user defined description. By default just the current date when the test was applied will be returned.
<code>statistic</code>	the value(s) of the test statistic.
<code>p.value</code>	the p-value(s) of the test.
<code>parameters</code>	a numeric value or vector of parameters.
<code>estimate</code>	a numeric value or vector of sample estimates.
<code>conf.int</code>	a numeric two row vector or matrix of 95
<code>method</code>	a character string indicating what type of test was performed.
<code>data.name</code>	a character string giving the name(s) of the data.

Note

There is nothing really new in this package. The benefit you will get from this help page, that we bring together a collection of time series tests from several R-packages which may be of interest for economists and financial engineers.

On the other hand the user can still use the underlying function calls from the imported R-packages.

The output of the various hypothesis tests is an object of class `htest`. The associated `print` method gives a unique report about the test results.

Author(s)

Adrian Trapletti for the tests from R's `tseries` package,
 Blake LeBaron for the `bds` C program,
 Diethelm Wuertz for the `Rmetrics` R-port.

References

- Brock, W.A., Dechert W.D., Sheinkman J.A. (1987); *A Test of Independence Based on the Correlation Dimension*, SSRI no. 8702, Department of Economics, University of Wisconsin, Madison.
- Conover, W.J. (1980); *Practical Nonparametric Statistics*, New York, Wiley.
- Cromwell J.B., Labys W.C., Terraza M. (1994); *Univariate Tests for Time Series Models*, Sage, Thousand Oaks, CA, pages 32–36.
- Lee T.H., White H., Granger C.W.J. (1993); *Testing for neglected nonlinearity in time series models*, *Journal of Econometrics* 56, 269–290.
- Teraesvirta T., Lin C.F., Granger C.W.J. (1993); *Power of the Neural Network Linearity Test*, *Journal of Time Series Analysis* 14, 209–220.

Examples

```
## SOURCE("fSeries.5B-TimeSeriesTests")

## bdsTest -
# iid Time Series:
par(mfrow = c(3, 1))
x = rnorm(100)
plot(x, type = "l", main = "iid Time Series")
bdsTest(x, m = 3)
# Non Identically Distributed Time Series:
x = c(rnorm(50), runif(50))
plot(x, type = "l", main = "Non-iid Time Series")
bdsTest(x, m = 3)
# Non Independent Innovations from Quadratic Map:
x = rep(0.2, 100)
for (i in 2:100) x[i] = 4*(1-x[i-1])*x[i-1]
plot(x, type = "l", main = "Quadratic Map")
bdsTest(x, m = 3)

## tnnTest -
# Time Series Non-linear in "mean" regression
par(mfrow = c(2, 1))
n = 1000
x = runif(1000, -1, 1)
tnnTest(x)
# Generate time series which is nonlinear in "mean"
x[1] = 0.0
for (i in (2:n)) {
  x[i] = 0.4*x[i-1] + tanh(x[i-1]) + rnorm(1, sd = 0.5) }
plot(x, main = "Teraesvirta Test", type = "l")
tnnTest(x)

## wnnTest -
# Time Series Non-Linear in "mean" Regression
par(mfrow = c(2, 1))
n = 1000
x = runif(1000, -1, 1)
wnnTest(x)
# Generate time series which is nonlinear in "mean"
x[1] = 0.0
for (i in (2:n)) {
  x[i] = 0.4*x[i-1] + tanh(x[i-1]) + rnorm(1, sd = 0.5) }
```

```
plot(x, main = "White Test", type = "l")  
wnnTest(x)
```

Index

*Topic **distribution**

GarchDistributions, 26
UnitrootDistribution, 12

*Topic **htest**

TimeSeriesTests, 43
UnitrootTests, 14

*Topic **models**

ArfimaOxInterface, 9
ArmaModelling, 1
ChaoticTimeSeries, 39
GarchModelling, 31
GarchOxInterface, 36
LongRangeDependence, 18

*Topic **programming**

HeavisideFunction, 29

absMoments (*GarchDistributions*),
26

absvalFit (*LongRangeDependence*),
18

adfTest (*UnitrootTests*), 14

aggvarFit (*LongRangeDependence*),
18

ar.ols, 5

arfimaOxFit (*ArfimaOxInterface*), 9

ArfimaOxInterface, 9

arma0, 6

armaFit (*ArmaModelling*), 1

ArmaModelling, 1

armaRoots (*ArmaModelling*), 1

armaSim (*ArmaModelling*), 1

armaTrueacf (*ArmaModelling*), 1

bdsTest (*TimeSeriesTests*), 43

Boxcar (*HeavisideFunction*), 29

boxperFit (*LongRangeDependence*),
18

ChaoticTimeSeries, 39

coef.fARMA (*ArmaModelling*), 1

Delta (*HeavisideFunction*), 29

dfTable (*UnitrootDistribution*), 12

dged (*GarchDistributions*), 26

diffvarFit (*LongRangeDependence*),
18

dsged (*GarchDistributions*), 26

dsnrm (*GarchDistributions*), 26

dsstd (*GarchDistributions*), 26

dstd (*GarchDistributions*), 26

falsennPlot (*ChaoticTimeSeries*),
39

farimaSim (*LongRangeDependence*),
18

farimaTrueacf
(*LongRangeDependence*), 18

farimaTruefft
(*LongRangeDependence*), 18

fARMA (*ArmaModelling*), 1

fARMA-class (*ArmaModelling*), 1

fbmSim (*LongRangeDependence*), 18

fGARCH-class (*GarchModelling*), 31

fgnSim (*LongRangeDependence*), 18

fgnTrueacf (*LongRangeDependence*),
18

fgnTruefft (*LongRangeDependence*),
18

fHURST (*LongRangeDependence*), 18

fHURST-class
(*LongRangeDependence*), 18

fitted.fARMA (*ArmaModelling*), 1

fitted.fGARCH (*GarchModelling*), 31

GarchDistributions, 26

garchFit (*GarchModelling*), 31

garchKappa (*GarchModelling*), 31

GarchModelling, 31

garchOxFit (*GarchOxInterface*), 36

GarchOxInterface, 36

garchSim (*GarchModelling*), 31

garchSpec (*GarchModelling*), 31

garchSpec-class (*GarchModelling*),
31

gedFit (*GarchDistributions*), 26

Heaviside (*HeavisideFunction*), 29

HeavisideFunction, 29

- henonSim (*ChaoticTimeSeries*), 39
- higuchiFit (*LongRangeDependence*), 18
- hurstSlider (*LongRangeDependence*), 18
- ikedaSim (*ChaoticTimeSeries*), 39
- logisticSim (*ChaoticTimeSeries*), 39
- LongRangeDependence, 18
- lorentzSim (*ChaoticTimeSeries*), 39
- lyapunovPlot (*ChaoticTimeSeries*), 39
- mutualPlot (*ChaoticTimeSeries*), 39
- nlm, 28
- normFit (*GarchDistributions*), 26
- optim, 6
- pdfTest (*UnitrootDistribution*), 12
- pengFit (*LongRangeDependence*), 18
- perFit (*LongRangeDependence*), 18
- pped (*GarchDistributions*), 26
- plot.fARMA (*ArmaModelling*), 1
- plot.fGARCH (*GarchModelling*), 31
- plot.garchOx (*GarchOxInterface*), 36
- predict.fARMA (*ArmaModelling*), 1
- predict.fGARCH (*GarchModelling*), 31
- print.fARMA (*ArmaModelling*), 1
- print.fGARCH (*GarchModelling*), 31
- print.garchOx (*GarchOxInterface*), 36
- print.garchSpec (*GarchModelling*), 31
- psged (*GarchDistributions*), 26
- psnorm (*GarchDistributions*), 26
- psstd (*GarchDistributions*), 26
- pstd (*GarchDistributions*), 26
- punitroot (*UnitrootDistribution*), 12
- qdfTest (*UnitrootDistribution*), 12
- qged (*GarchDistributions*), 26
- qsged (*GarchDistributions*), 26
- qsnorm (*GarchDistributions*), 26
- qsstd (*GarchDistributions*), 26
- qstd (*GarchDistributions*), 26
- qunitroot (*UnitrootDistribution*), 12
- Ramp (*HeavisideFunction*), 29
- recurrencePlot (*ChaoticTimeSeries*), 39
- residuals.fARMA (*ArmaModelling*), 1
- residuals.fGARCH (*GarchModelling*), 31
- rged (*GarchDistributions*), 26
- roesslerSim (*ChaoticTimeSeries*), 39
- rsFit (*LongRangeDependence*), 18
- rsged (*GarchDistributions*), 26
- rsnorm (*GarchDistributions*), 26
- rsstd (*GarchDistributions*), 26
- rstd (*GarchDistributions*), 26
- separationPlot (*ChaoticTimeSeries*), 39
- sgedFit (*GarchDistributions*), 26
- show, fHURST-method (*LongRangeDependence*), 18
- show.fHURST (*LongRangeDependence*), 18
- Sign (*HeavisideFunction*), 29
- snormFit (*GarchDistributions*), 26
- sstdFit (*GarchDistributions*), 26
- stdFit (*GarchDistributions*), 26
- summary.fARMA (*ArmaModelling*), 1
- summary.fGARCH (*GarchModelling*), 31
- summary.garchOx (*GarchOxInterface*), 36
- tentSim (*ChaoticTimeSeries*), 39
- TimeSeriesTests, 43
- tnnTest (*TimeSeriesTests*), 43
- tsTest (*TimeSeriesTests*), 43
- UnitrootDistribution, 12
- unitrootTest (*UnitrootTests*), 14
- UnitrootTests, 14
- urdfTest (*UnitrootTests*), 14
- urersTest (*UnitrootTests*), 14
- urkpssTest (*UnitrootTests*), 14
- urppTest (*UnitrootTests*), 14
- urspTest (*UnitrootTests*), 14
- urTest (*UnitrootTests*), 14
- urzaTest (*UnitrootTests*), 14
- waveletFit (*LongRangeDependence*), 18
- whittleFit (*LongRangeDependence*), 18
- wnnTest (*TimeSeriesTests*), 43