The fArma Package

October 27, 2008

Version 270.74

Date 1997 - 2008

Title ARMA Time Series Modelling

Author Diethelm Wuertz and many others, see the SOURCE file

Depends R (>= 2.4.0), methods, fBasics

Maintainer Rmetrics Core Team <Rmetrics-core@r-project.org>

Description Environment for teaching "Financial Engineering and Computational Finance"

NOTE SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS FOR ARGUMENTS AND RETURN VALUES.

LazyLoad yes

LazyData yes

License GPL (>= 2)

URL http://www.rmetrics.org

R topics documented:

ArmaInterface *Integrated ARMA Time Series Modelling*

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:

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

Usage

```
armaSim(model = list(ar = c(0.5, -0.5), d = 0, ma = 0.1), n = 100,
    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, ...)

## S4 method for signature 'fARMA':
show(object)

## 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, ...)
```

Arguments

addControl	[armaSim] - a logical value. Should control parameters added to the returned series as a control attribute?
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.
doplot	[armaRoots] - a logical. Should a plot be displayed? [predict][summary] - is used by the predict and summary methods. By default, this value is set to TRUE and thus the function calls generate beside written also graphical printout. Additional arguments required by underlying functions have to be passed through the dots argument.
fixed	[armaFit] - is an optional numeric vector of the same length as the total number of parameters. If supplied, only NA entries in fixed 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~ARIMA(p, 0, q) and x~ARIMA(p, 0, 0), respectively.
formula	[armaFit] - a formula specifying the general structure of the ARMA form. Can have one of the forms x ~ ar(q), x ~ ma(p), x ~ arma(p, q), x ~ arima(p, d, q), or x ~ arfima(p, q). x is the response variable optionally to appear in the formula expression. In the first case R's function ar from the ts package will be used to estimate the parameters, in the second case R's function arma from the tseries package will be used, in the third case R's function arima from the ts package will be used, and in the last case R's function fracdiff from the fracdiff package will be used. The state space modelling based arima function allows also to fit ARMA models using arima(p, d=0, q), and AR models using arima(q, d=0, q=0), or pure MA models using arima(q=0, d=0, p). (Exogenous variables are also allowed and can be passed through the ... argument.)
gof.lag	[print][plot][summary][predict] - the maximum number of lags for a goodness-of-fit test.
include.mean	[armaFit] - Should the ARIMA model include a mean term? The default is TRUE, note that for differenced series a mean would not affect the fit nor predictions.

`innov` [armaSim] -
is a univariate time series or vector of innovations to produce the series. If not provided, `innov` will be generated using the random number generator specified by `rand.gen`. Missing values are not allowed. By default the normal random number generator will be used.

`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, `method="mle"`. In the case of an AR model the parameter estimation can also be done by ordinary least square estimation, `"ols"`.

`model` [armaSim] -
a list with one (AR), two (ARMA) or three (ARIMA, FRACDIFF) elements . `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 (F)ARIMA(p, d, q) with `p=length(ar)` and `q=length(ma)`. `d` is a positive integer for ARIMA models and a numeric value for FRACDIFF models. By default an ARIMA(2, 0, 1) model with coefficients `ar=c(0.5, -0.5)` and `ma=0.1` will be generated.

`n` [armaSim] -
an integer value setting the length of the series to be simulated (optional if `innov` is provided). The default value is 100.

`n.ahead, n.back, conf`
[print][plot][summary][predict] -
are presetted arguments for the `predict` method. `n.ahead` determines how far ahead forecasts should be evaluated together with errors on the confidence intervals given by the argument `conf`. If a forecast plot is desired, which is the default and expressed by `doplot=TRUE`, then `n.back` sets the number of time steps back displayed in the graph.

`n.start` [armaSim] -
gives the number of start-up values discarded when simulating non-stationary models. The start-up innovations will be generated by `rand.gen` if `start.innov` is not provided.

`object` [summary][predict] -
is an object of class `fARMA` returned by the fitting function `armaFit` and serves as input for the `summary`, and `predict` methods. Some methods allow for additional arguments.

`rand.gen` [armaSim] -
is the function which is called to generate the innovations. Usually, `rand.gen` will be a random number generator. Additional arguments required by the random number generator `rand.gen`, usually the location, scale and/or shape parameter of the underlying distribution function, have to be passed through the `dots` argument.

`rseed` [armaSim] -
the random number seed, by default NULL. If this argument is set to an integervalue, then the function `set.seed(rseed)` will be called.

start.innov	[armaSim] - is a univariate time series or vector of innovations to be used as start up values. Missing values are not allowed.
title	a character string which allows for a project title.
which	[plot][summary] - if which is set to "ask" the function will interactively ask which plot should be displayed. This is the default value for the plot method. If which="all" is specified all plots will be displayed. This is the default setting for the summary 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 TRUE.
x	[print][plot] - is an object of class fARMA returned by the fitting function armaFit and serves as input for the predict, print, print.summary, and plot methods. Some methods allow for additional arguments.
...	additional arguments to be passed to the output timeSeries. (charvec, units, ...)

Details

AR - Auto-Regressive Modelling:

The argument x~ar(p) calls the underlying functions ar.mle or codear.ols depending on the method's choice. For definiteness, the AR models are defined through

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

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

$$x_t = a_1 x_{t-1} + \cdots + a_p x_{t-p} + e_t + b_1 e_{t-1} + \ldots + b_q e_{t-q}$$

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]`

ARFIMA/FRACDIFF Modelling:

The argument `x~arfima()` 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:

`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`	the selected time series model naming the applied method.
`formula`	the formula expression describing the model.
`parameters`	named parameters or coefficients of the fitted model.
`title`	a title string.

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 R/Rmetrics and SPlus/Finmetrics, this approach will be quite natural.

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

`x ~ ar()`	autoregressive time series processes,
`x ~ ma()`	moving average time series processes,
`x ~ arma()`	autoregressive moving average processes,
`x ~ arima()`	autoregressive integrated moving average processes, and
`x ~ arfima()`	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~arfima()` 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 simu-

lates 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 `arfima` 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

```
## armaSim -
   # Simulation:
   x = armaSim(model = list(ar = c(0.5, -0.5), ma = 0.1), n = 1000)

## armaFit -
   # Estimate the Parameters:
   fit = armaFit(~ arma(2, 1), data = x)
   print(fit)
```

```
## summary -
   # Diagnostic Analysis:
   par(mfrow = c(2, 2), cex = 0.7)
   summary(fit, which =  "all")

## plot -
   # Interactive Plots:
   # par(mfrow = c(1, 1))
   # plot(fit)

## predict -
   # Forecast 5 Steps Ahead:
   par(mfrow = c(1, 1))
   predict(fit, 5)

## armaFit -
   # Alternative Calls:
   TS = as.timeSeries(data(msft.dat))
   armaFit(formula = diff(log(Close)) ~ ar(5), data = TS)
   armaFit(Close ~ ar(5), data = returns(TS, digits = 12))
   TS.RET = returns(TS, digits = 12)
   armaFit(Close ~ ar(5), TS.RET)
   armaFit(Close ~ ar(5), as.data.frame(TS.RET))
   armaFit(~ ar(5), as.vector(TS.RET[, "Close"]))
   armaFit(~ ar(5), as.ts(TS.RET)[, "Close"])
   attach(TS.RET)
   armaFit(Close ~ ar(5))
   detach(TS.RET)
```

`ArmaStatistics` *Statistics of the True ARMA Process*

Description

A collection and description of functions to compute statistics of a true ARMA time series process.

The functions are:

`armaRoots`	Roots of the characteristic ARMA polynomial,
`armaTrueacf`	True autocorrelation function of an ARMA process.

Usage

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

Arguments

coefficients	[armaRoots] - a numeric vector with the coefficients of the characterisitic polynomial.
digits	[armaRoots] - output precision, an integer value.
doplot	[armaRoots] - a logical. Should a plot be displayed?
lag.max	[armaTrueacf] - maximum number of lags at which to calculate the acf or pacf, an integer value by default 20.
model	[armaTrueacf] - a specification of the ARMA model with two elements: model$ar is the vector of the AR coefficients, and model$ma is the vector of the MA coefficients.
n	[armaSim] - an integer value setting the length of the series to be simulated (optional if innov is provided). The default value is 100.
n.plot	[armaRoots] - the number of data points to plot the unit circle; an integer value.
type	[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.
...	additional arguments to be passed.

Value

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.

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

```
## 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)
```

`LrdModelling` *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, and functions to estimate 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 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, 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"), ...)

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)

## S4 method for signature 'fHURST':
show(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. $10\hat{0}.7$ and $10\hat{2}.5$, 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 tow 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 peridogram 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=length(ar)` and `q=length(ma)`. `d` is a fractional value for FARMA models. By default an FARMA(2, d, 1) model with coefficients `ar=c(0.5, -0.5)`, `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`.
`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 defaul 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 `timeSeries`, or any other object which can be transofrmed into a numeric vector by the function `as.vector`.
`...`	arguments to be passed.

Details

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 autocorrelated 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 approaximate 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 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 implement in S. The first named `"freq"` works in the frequence 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].

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=2*H-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=2*H-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 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 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*t is fitted to the cumulated sums. Then the sample variance of the residuals is computed which is proportional to m^(2*H). The "mean" or "median" are computed over the blocks. The slope beta=2*H 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 dicussed 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 slopw 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 periodgram itself, the cmulative periodgram 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 devides 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.

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.

`*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 Communcation Review27, 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

```
## 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")
```

LongRangeDependence

Statistics of the True LRD Process

Description

A collection and description of functions to investigate the true statistics of the long range dependence or long memory behavior of an univariate time series process.

The Functions are:

Functions to model the true autocorrelation function and spectrum:

`fgnTrueacf`	Returns true FGN covariances,
`fgnTruefft`	returns true FGN fast Fourier transform,
`fgnStatsSlider`	returns a plot of true FGN Statistics,
`farimaTrueacf`	returns true FARIMA covariances,
`farimaTruefft`	returns true FARIMA fast Fourier transform,
`farimaStatsSlider`	returns a plot of true FARIMA Statistics.

Usage

```
fgnTrueacf(n = 100, H = 0.7)
fgnTruefft(n = 100, H = 0.7)
fgnStatsSlider()

farimaTrueacf(n = 100, H = 0.7)
farimaTruefft(n = 100, H = 0.7)
farimaStatsSlider()
```

Arguments

`H`	the Hurst exponent, a numeric value between 0.5 and 1, by default 0.7.
`n`	number of data points to be simulated, a numeric value, by default 100.

Details

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].

Value

`fgnTrueacf`
`farimaTrueacf`
return the true covariance of an FGN or FARIMA time series process.

`fgnTruefft`
`farimaTruefft`
return the true spectrum of an FGN or FARIMA time series process.

fgnStatsSlider()
farimStatsSlider()
interactively display the true covariance and the true spectrum of an FGN or FARIMA time series process.

Author(s)

J. Beran, ported by Maechler, code as listed in the Appendix of his Book,
Diethelm Wuertz for the Rmetrics R-port.

References

Beran J. (1992); *Statistics for Long-Memory Processes*, Chapman and Hall, New York, 1994.

Examples

```
## fgnTrueacf -
   fgnTrueacf(n = 20, H = 0.8)

## fgnTruefft -
   fgnTruefft(n = 20, H = 0.8)

## farimaTrueacf -
   farimaTrueacf(n = 20, H = 0.8)

## farimaTruefft -
   farimaTruefft(n = 20, H = 0.8)
```

Index