Package 'rugarch'

September 10, 2011

Type Package

Title Univariate GARCH models

Version 1.0-1

Date 2011-09-05

Author Alexios Ghalanos <alexios@4dscape.com>

Maintainer Alexios Ghalanos <alexios@4dscape.com>

Depends
R (>= 2.10.0), Rcpp (>= 0.8.5), RcppArmadillo (>= 0.2.5),methods, numDeriv, chron, Rsolnp

LinkingTo Rcpp, RcppArmadillo

Description ARFIMA, in-mean, external regressors and various GARCH
flavours, with methods for fit, forecast, simulation, inference and plotting.

Suggests xts, timeSeries, multicore, snowfall

Collate rugarch-imports.R rugarch-cwrappers.R rugarch-solvers.R
rugarch-distributions.R rugarch-kappa.R rugarch-helperfn.R
rugarch-numderiv.R rugarch-series.R rugarch-startpars.R
rugarch-tests.R rugarch-armafor.R rugarch-graphs.R
rugarch-classes.R rugarch-sgarch.R rugarch-fgarch.R
rugarch-egarch.R rugarch-gjrgarch.R rugarch-aparch.R
rugarch-igarch.R rugarch-multi.R rugarch-plots.R
rugarch-rolling.R rugarch-uncertainty.R rugarch-bootstrap.R
rugarch-methods.R rugarch-benchmarks.R arfima-classes.R
arfima-multi.R arfima-main.R arfima-methods.R zzz.R

LazyLoad yes

License GPL-3

Repository CRAN

Repository/R-Forge/Project rgarch

Repository/R-Forge/Revision 336

Date/Publication 2011-09-10 15:04:24

R topics documented:

`rugarch-package` *The rugarch package*

Description

The rugarch package aims to provide a flexible and rich univariate GARCH modelling and testing environment. Modelling is a simple process of defining a specification and fitting the data. Inference can be made from summary, various tests and plot methods, while the forecasting, filtering and simulation methods complete the modelling environment. Finally, specialized methods are implemented for simulating parameter distributions and evaluating parameter consistency, and a bootstrap forecast method which takes into account both parameter and predictive distribution uncertainty.

The testing environment is based on a rolling backtest function which considers the more general context in which GARCH models are based, namely the conditional time varying estimation of density parameters and the implication for their use in analytical risk management measures.

The mean equation allows for AR(FI)MA, arch-in-mean and external regressors, while the variance equation implements a wide variety of univariate GARCH models as well as the possibility of including external regressors. Finally, a set of rich distributions from the "fBasics" package and Johnson's reparametrized SU from the "gamlss" package are used for modelling innovations.

This package is part of what used to be the rgarch package, which was split into univariate (rugarch) and multivariate (rmgarch) models for easier maintenance and use. The rmgarch package is still under re-write so the old rgarch package should be used in the meantime for multivariate models (and hosted on r-forge).

Details

Package:	rugarch
Type:	Package
Version:	1.00-1
Date:	2011-09-05
License:	GPL
LazyLoad:	yes
Depends:	R (>= 2.10.0), Rcpp (>= 0.8.5), RcppArmadillo (>= 0.2.5), numDeriv, chron, Rsolnp

While the package has implemented some safeguards, both during pre-estimation as well as the estimation phase, there is no guarantee of convergence in the fitting procedure. As a result, the fit method allows the user to input starting parameters as well as keep any parameters from the spec as fixed (including the case of all parameters fixed).

The functionality of the packages is contained in the main methods for defining a specification `ugarchspec`, fitting `ugarchfit`, forecasting `ugarchforecast`, simulation from fit object `ugarchsim`, path simulation from specification object `ugarchpath`, parameter distribution by simulation `ugarchdistribution`, bootstrap forecast `ugarchboot` and rolling estimation and forecast `ugarchroll`. There are also some functions which enable multiple fitting of assets in an easy to use wrapper with the option of multicore functionality, namely `multispec`, `multifit`, `multifilter` and `multiforecast`. Explanations on the available methods for the returned classes can be found in the documentation for those classes.

A separate subset of methods and classes has been included to calculate pure ARFIMA models with constant variance. This subset includes similar functionality as with the GARCH methods, with the exception that no plots are yet implemented, and neither is a forecast based on the bootstrap. These may be added in the future. While there are limited examples in the documentation on the ARFIMA methods, the interested user can search the rugarch.tests folder of the source installation for some tests using ARFIMA models as well as equivalence to the base R arima methods (particularly replication of simulation). Finally, no representation is made about the adequacy of ARFIMA models, particularly the statistical properties of parameters when using distributions which go beyond the Gaussian.

The conditional distributions used in the package are also exposed for the benefit of the user through the `rgarchdist` functions which contain methods for density, distribution, quantile, sampling and fitting. Additionally, `ghyptransform` function provides the necessary parameter transformation and scaling methods for moving from the location scale invariant 'rho-zeta' parametrization with mean and standard deviation, to the standard 'alpha-beta-delta-mu' parametrization of the Generalized Hyperbolic Distribution family.

The type of data handled by the package is quite varied, accepting "timeSeries", "xts", "zoo", "zooreg", "data.frame" with dates as rownames, "matrix" and "numeric" vector with dates as names. For the "numeric" vector and "data.frame" with characterdates in names or rownames, the package tries a variety of methods to try to recognize the type and format of the date else will index the data numerically. The package holds dates internally as class `Date`. This mostly impacts the plots and forecast summary methods. For high frequency data, the user should make use of a non-named representation such as "matrix" or "numeric" as the package has yet to implement methods for checking and working with frequencies higher than daily (and is unlikely to do so). Finally, the functions `ForwardDates` and `WeekDayDummy` offer some simple Date manipulation methods for working with forecast dates and creating day of the week dummy variables for use in GARCH

modelling.
Some benchmarks (published and comparison with commercial package), are available through the ugarchbench function. The 'inst' folder of the source distribution also contains various tests which can be sourced and run by the user, also exposing some finer details of the functionality of the package. The user should really consult the examples supplied in this folder which are quite numerous and instructive with some comments.

How to cite this package

Whenever using this package, please cite as

```
@Manual{Ghalanos_2011,
 author       = {Alexios Ghalanos},
 title        = {{rugarch}: Univariate GARCH models.},
 year         = {2011},
 note        = {R package version 1.00.},}
```

License

The releases of this package is licensed under GPL version 3.

Author(s)

Alexios Ghalanos

References

Baillie, R.T. and Bollerslev, T. and Mikkelsen,H.O. 1996, Fractionally integrated generalized autoregressive conditional heteroskedasticity, *Journal of Econometrics*, 3–30 .
Berkowitz, J. 2001, Testing density forecasts, with applications to risk management, *Journal of Business and Economic Statistics*, **19(4)**, 465–474.
Bollerslev, T. 1986, Generalized Autoregressive Conditional Heteroskedasticity 1986, *Journal of Econometrics*, **31**, 307–327.
Ding, Z., Granger, C.W.J. and Engle, R.F. 1993, A Long Memory Property of Stock Market Returns and a New Model, *Journal of Empirical Finance*, **1**, 83–106.
Engle, R.F. and Ng, V. K. 1993, Measuring and Testing the Impact of News on Volatility, *Journal of Finance*, **48**, 1749–1778.
Glosten, L.R., Jagannathan, R. and Runkle, D.E. 1993, On the Relation between the Expected Value and the Volatility of the Nominal Excess Return on Stocks, *Journal of Finance*, **48(5)**, 1779–1801.
Hansen, B.E. 1990, Langrange Multiplier Tests for Parameter Instability in Non-Linear Models, *mimeo*.
Hentschel, Ludger. 1995, All in the family Nesting symmetric and asymmetric GARCH models, *Journal of Financial Economics*, **39(1)**, 71–104.
Nelson, D.B. 1991, Conditional Heteroskedasticity in Asset Returns: A New Approach, *Econometrica*, **59**, 347–370.
Pascual, L., Romo, J. and Ruiz, E. 2004, Bootstrap predictive inference for ARIMA processes,

Journal of Time Series Analysis.
Pascual, L., Romo, J. and Ruiz, E. 2006, Bootstrap prediction for returns and volatilities in GARCH models, *Computational Statistics and Data Analysis*.
Vlaar, P.J.G. and Palm, F.C. 1993, The Message in Weekly Exchange Rates in the European Monetary System: Mean Reversion Conditional Heteroskedasticity and Jumps, *Journal of Business and Economic Statistics*, **11**, 351–360.

`ARFIMA-class` *class: High Level ARFIMA class*

Description

The virtual parent class of the ARFIMA subset.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "ARFIMA" in the signature.

Author(s)

Alexios Ghalanos

`ARFIMAdistribution-class`
class: ARFIMA Parameter Distribution Class

Description

Class for the ARFIMA Parameter Distribution, objects of which are created by calling function `arfimadistribution`.

Slots

`dist`: Object of class "vector" Details of fitted parameters.

`truecoef`: Object of class "matrix" The actual coefficients.

`model`: Object of class "list" The model specification.

Extends

Class "ARFIMA", directly. Class "rGARCH", by class "ARFIMA", distance 2.

Methods

as.data.frame `signature(x = "ARFIMAdistribution")`: extracts various values from object (see note).

show `signature(object = "ARFIMAdistribution")`: parameter distribution summary.

Note

The `as.data.frame` function takes optionally 2 additional arguments, namely `window` which indicates the particular distribution window number for which data is required (is usually just 1 unless the recursive option was used), and `which` indicating the type of data required. Valid values for the latter are "rmse" for the root mean squared error between simulation fit and actual parameters, "stats" for various statistics computed for the simulations such as log likelihood, persistence, unconditional variance and mean, "coef" for the estimated coefficients (i.e. the parameter distribution and is the default choice), and "coefse" for the estimated robust standard errors of the coefficients (i.e. the parameter standard error distribution).

Author(s)

Alexios Ghalanos

`arfimadistribution-methods`

function: ARFIMA Parameter Distribution via Simulation

Description

Method for simulating and estimating the parameter distribution from an ARFIMA models as well as the simulation based consistency of the estimators given the data size.

Usage

```
arfimadistribution(fitORspec, n.sim = 2000, n.start = 1, m.sim = 100,
recursive = FALSE, recursive.length = 6000, recursive.window = 1000,
prereturns = NA, preresiduals = NA, rseed = NA,
custom.dist = list(name = NA,  distfit = NA, type = "z"), mexsimdata = NULL,
fit.control = list(), solver = "solnp", solver.control = list(),
parallel = FALSE, parallel.control = list(pkg = c("multicore", "snowfall"),
cores = 2), ...)
```

Arguments

fitORspec	Either an ARFIMA fit object of class ARFIMAfit or alternatively an ARFIMA specification object of class ARFIMAspec with valid parameters supplied via the fixed.pars argument in the specification.
n.sim	The simulation horizon.
n.start	The burn-in sample.
m.sim	The number of simulations.
recursive	Whether to perform a recursive simulation on an expanding window.
recursive.length	If recursive is TRUE, this indicates the final length of the simulation horizon, with starting length n.sim.
recursive.window	If recursive is TRUE, this indicates the increment to the expanding window. Together with recursive.length, it determines the total number of separate and increasing length windows which will be simulated and fitted.
prereturns	Allows the starting return data to be provided by the user.
preresiduals	Allows the starting residuals to be provided by the user.
rseed	Optional seeding value(s) for the random number generator.
custom.dist	Optional density with fitted object from which to simulate.
mexsimdata	Matrix of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided.
solver	One of either "nlminb" or "solnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine (as in the arfimafit method).
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Details

This method facilitates the simulation and evaluation of the uncertainty of ARFIMA model parameters. The recursive option also allows the evaluation of the simulation based consistency (in terms of sqrt(N)) of the parameters as the length (n.sim) of the data increases, in the sense of the root mean square error (rmse) of the difference between the simulated and true (hypothesized) parameters. This is an expensive function, particularly if using the recursive option, both on memory and CPU resources, performing many re-fits of the simulated data in order to generate the parameter distribution.

Value

A ARFIMAdistribution object containing details of the ARFIMA simulated parameters distribution.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
spec = arfimaspec( mean.model = list(armaOrder = c(2,2), include.mean = TRUE,
arfima = FALSE), distribution.model = "norm", fixed.pars = list(ar1=0.6,
ar2=0.21, ma1=-0.7, ma2=0.3, mu = 0.02, sigma = 0.02))
dist = arfimadistribution(spec, n.sim = 2000, n.start = 100, m.sim = 100,
recursive = TRUE, recursive.length = 10000, recursive.window = 1000)
# slots:
slotNames(dist)
# methods:
# summary
show(dist)
# as.data.frame(...., window, which=c("rmse", "stats", "coef", "coefse"))
# default
as.data.frame(dist)

as.data.frame(dist, window = 1, which = "rmse")
as.data.frame(dist, window = 1, which = "stats")
as.data.frame(dist, window = 1, which = "coef")
as.data.frame(dist, window = 1, which = "coefse")


as.data.frame(dist, window = 8, which = "rmse")
as.data.frame(dist, window = 8, which = "stats")
as.data.frame(dist, window = 8, which = "coef")
as.data.frame(dist, window = 8, which = "coefse")


# create some plots
#
nwindows = dist@dist$details$nwindows
# 2000/3000/4000/5000/6000/7000/8000/9000/10000

# expected reduction factor in RMSE for sqrt(N) consistency
expexcted.rmsegr = sqrt(2000/seq(3000,10000,by=1000))

# actual RMSE reduction
actual.rmsegr = matrix(NA, ncol = 8, nrow = 6)
rownames(actual.rmsegr) = c("mu", "ar1", "ar2", "ma2", "ma2", "sigma")
# start at 2000 (window 1)
rmse.start = as.data.frame(dist, window = 1, which = "rmse")
for(i in 2:nwindows) actual.rmsegr[,i-1] = as.numeric(as.data.frame(dist,
window = i, which = "rmse")/rmse.start)
par(mfrow = c(2,3))
for(i in 1:6){
plot(seq(3000,10000,by=1000),actual.rmsegr[i,], type = "l", lty = 2,
ylab = "RMSE Reduction", xlab = "N (sim)",main = rownames(actual.rmsegr)[i])
lines(seq(3000,10000,by=1000), expexcted.rmsegr, col = 2)
```

```
legend("topright", legend = c("Actual", "Expected"), col = 1:2, bty = "m",
lty = c(2,1))
}

## End(Not run)
```

ARFIMAfilter-class *class: ARFIMA Filter Class*

Description

Class for the ARFIMA filter.

Slots

`filter`: Object of class `"vector"`

`model`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.data.frame `signature(x = "ARFIMAfilter")`: Extracts the position (dates), data, filtered values and residuals.

coef `signature(object = "ARFIMAfilter")`: Extracts the coefficients.

fitted `signature(object = "ARFIMAfilter")`: Extracts the filtered values.

infocriteria `signature(object = "ARFIMAfilter")`: Calculates and returns various information criteria.

likelihood `signature(object = "ARFIMAfilter")`: Extracts the likelihood.

residuals `signature(object = "ARFIMAfilter")`: Extracts the residuals.

show `signature(object = "ARFIMAfilter")`: Filter summary.

uncmean `signature(object = "ARFIMAfilter")`: Calculates and returns the unconditional mean. Takes additional arguments 'method' with option for "analytical" or "simulation", 'n.sim' for the number of simulations (if that method was chosen, and defaults to 100000) and 'rseed' for the simulation random generator initialization seed. Note that the simulation method is only available for a fitted object or specification with fixed parameters, and not for the filtered object.

Author(s)

Alexios Ghalanos

Examples

```
showClass("ARFIMAfilter")
```

`arfimafilter-methods` *function: ARFIMA Filtering*

Description

Method for filtering an ARFIMA model.

Usage

```
arfimafilter(spec, data, out.sample = 0, n.old=NULL, ...)
```

Arguments

`data`	A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
`spec`	An ARFIMA spec object of class `ARFIMAspec` with the fixed.pars argument having the model parameters on which the filtering is to take place.
`out.sample`	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (as in `arfimafit` function).
`n.old`	For comparison with ARFIMA models using the out.sample argument, this is the length of the original dataset (see details).
`...`	.

Details

The n.old argument is optional and indicates the length of the original data (in cases when this represents a dataseries augmented by newer data). The reason for using this is so that the old and new datasets agree since the original recursion uses the sum of the residuals to start the recursion and therefore is influenced by new data. For a small augmentation the values converge after x periods, but it is sometimes preferable to have this option so that there is no forward looking information contaminating the study.

Value

A `ARFIMAfilter` object containing details of the ARFIMA filter.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
```

```
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp",
fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist

for(i in 1:9){
cf = coef(fit[[i]])
cfmatrix[i, match(names(cf), colnames(cfmatrix))] =  cf
}
sk = ku = rep(0, 9)
for(i in 1:9){
cf = coef(fit[[i]])
if(fit[[i]]@model$modelinc[16]>0) sk[i] = dskewness(distribution = dist[i],
skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
if(fit[[i]]@model$modelinc[17]>0) ku[i] = dkurtosis(distribution = dist[i],
skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
}
hq = sapply(fit, FUN = function(x) infocriteria(x)[4])
cfmatrix = cbind(cfmatrix, sk, ku, hq)
colnames(cfmatrix)=c(colnames(cfmatrix[,1:7]), "skewness", "ex.kurtosis","HQIC")


# filter the data to check results
filt = vector(mode = "list", length = 9)
for(i in 1:9){
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
setfixed(spec) = as.list(coef(fit[[i]]))
filt[[i]] = arfimafilter(spec = spec, data = sp500ret)
}
print(cfmatrix, digits = 4)
cat("\nARFIMAfit and ARFIMAfilter residuals check:\n")
print(head(sapply(filt, FUN = function(x) residuals(x))) == head(sapply(fit,
FUN = function(x) residuals(x))))
cat("\nas.data.frame method:\n")
print(cbind(head(as.data.frame(filt[[1]])), head(as.data.frame(fit[[1]]))))
cat("\ncoef method:\n")
print(cbind(coef(filt[[1]]), coef(fit[[1]])))
cat("\nfitted method:\n")
print(cbind(head(fitted(filt[[1]])), head(fitted(fit[[1]]))))
cat("\ninfocriteria method:\n")
# For filter, it assumes estimation of parameters else does not make sense!
print(cbind(infocriteria(filt[[1]]), infocriteria(fit[[1]])))
cat("\nlikelihood method:\n")
print(cbind(likelihood(filt[[1]]), likelihood(fit[[1]])))
cat("\nresiduals method:\n")
# Note that we the package will always return the full length residuals and
# fitted object irrespective of the lags (i.e. since this is an ARMA(1,1)
# i.e. max lag = 1, the first row is zero and should be discarded.
```

```
print(cbind(head(residuals(filt[[1]])), head(residuals(fit[[1]]))))
cat("\nuncmean method:\n")
print(cbind(uncmean(filt[[1]]), uncmean(fit[[1]])))
cat("\nuncmean method (by simulation):\n")
# For spec and fit
spec = arfimaspec( mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[1])
setfixed(spec) = as.list(coef(fit[[1]]))
print(cbind(uncmean(spec, method = "simulation", n.sim = 100000, rseed = 100),
uncmean(fit[[1]], method = "simulation", n.sim = 100000, rseed = 100)))
cat("\nsummary method:\n")
show(filt[[1]])
show(fit[[1]])

## End(Not run)
```

ARFIMAfit-class *class: ARFIMA Fit Class*

Description

Class for the ARFIMA fit.

Slots

`fit`: Object of class `"vector"`

`model`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.data.frame `signature(x = "ARFIMAfit")`: Extracts the position (dates), data, fitted values and residuals.

coef `signature(object = "ARFIMAfit")`: Extracts the coefficients.

fitted `signature(object = "ARFIMAfit")`: Extracts the fitted values.

infocriteria `signature(object = "ARFIMAfit")`: Calculates and returns various information criteria.

likelihood `signature(object = "ARFIMAfit")`: Extracts the likelihood.

residuals `signature(object = "ARFIMAfit")`: Extracts the residuals.

show `signature(object = "ARFIMAfit")`: Fit summary.

uncmean `signature(object = "ARFIMAfit")`: Calculates and returns the unconditional mean. Takes additional arguments 'method' with option for "analytical" or "simulation", 'n.sim' for the number of simulations (if that method was chosen, and defaults to 100000) and 'rseed' for the simulation random generator initialization seed.

Author(s)

Alexios Ghalanos

Examples

```
showClass("ARFIMAfit")
```

---

arfimafit-methods *function: ARFIMA Fit*

Description

Method for fitting an ARFIMA models.

Usage

```
arfimafit(spec, data, out.sample = 0, solver = "solnp", solver.control = list(),
fit.control = list(fixed.se = 0, scale = 0), ...)
```

Arguments

data	A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
spec	An ARFIMA spec object of class ARFIMAspec.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (see details).
solver	One of either "nlminb", "solnp" or "gosolnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine. The fixed.se argument controls whether standard errors should be calculated for those parameters which were fixed (through the fixed.pars argument of the arfimaspec function). The scale parameter controls whether the data should be scaled before being submitted to the optimizer.
...	.

Details

The ARFIMA optimization routine first calculates a set of feasible starting points which are used to initiate the ARFIMA Maximum Likelihood recursion. The main part of the likelihood calculation is performed in C-code for speed.
The out.sample option is provided in order to carry out forecast performance testing against actual data. A minimum of 5 data points are required for these tests. If the out.sample option is positive, then the routine will fit only N - out.sample (where N is the total data length) data points, leaving out.sample points for forecasting and testing using the forecast performance measures. In the arfimaforecast routine the n.ahead may also be greater than the out.sample number resulting in a

combination of out of sample data points matched against actual data and some without, which the forecast performance tests will ignore.
The "gosolnp" solver allows for the initialization of multiple restarts of the solnp solver with randomly generated parameters (see documentation in the Rsolnp-package for details of the strategy used). The solver.control list then accepts the following additional (to the solnp) arguments: "n.restarts" is the number of solver restarts required (defaults to 1), "parallel" and "parallel.control" for use of the parallel functionality, "rseed" is the seed to initialize the random number generator, and "n.sim" is the number of simulated parameter vectors to generate per n.restarts.

Value

A ARFIMAfit object containing details of the ARFIMA fit.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp",
fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist

for(i in 1:9){
cf = coef(fit[[i]])
cfmatrix[i, match(names(cf), colnames(cfmatrix))] =  cf
}
sk = ku = rep(0, 9)
for(i in 1:9){
cf = coef(fit[[i]])
if(fit[[i]]@model$modelinc[16]>0) sk[i] = dskewness(distribution = dist[i],
skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
if(fit[[i]]@model$modelinc[17]>0) ku[i] = dkurtosis(distribution = dist[i],
skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
}
hq = sapply(fit, FUN = function(x) infocriteria(x)[4])
cfmatrix = cbind(cfmatrix, sk, ku, hq)
colnames(cfmatrix)=c(colnames(cfmatrix[,1:7]), "skewness", "ex.kurtosis","HQIC")

print(cfmatrix, digits = 4)
# notice that for the student distribution kurtosis is NA since shape (dof) < 4.
cat("\nas.data.frame method:\n")
```

```
head(as.data.frame(fit[[1]]))
cat("\ncoef method:\n")
coef(fit[[1]])
cat("\nfitted method:\n")
head(fitted(fit[[1]]))
cat("\ninfocriteria method:\n")
infocriteria(fit[[1]])
cat("\nlikelihood method:\n")
likelihood(fit[[1]])
cat("\nresiduals method:\n")
# Note that we the package will always return the full length residuals and
# fitted object irrespective of the lags (i.e. since this is an ARMA(1,1)
# i.e. max lag = 1, the first row is zero and should be discarded).
head(residuals(fit[[1]]))
cat("\nuncmean method:\n")
uncmean(fit[[1]])
cat("\nuncmean method (by simulation):\n")
uncmean(fit[[1]], method = "simulation", n.sim = 100000, rseed = 100)
cat("\nsummary method:\n")
show(fit[[1]])

## End(Not run)
```

ARFIMAforecast-class *class: ARFIMA Forecast Class*

Description

Class for the ARFIMA forecast.

Slots

`forecast`: Object of class `"vector"`

`model`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.array `signature(x = "ARFIMAforecast")`: Extracts the forecast array with matrix column dimensions equal to the n.ahead value and row dimension 1 (series forecast), and array dimension equal to the number of rolling forecasts chosen.

as.data.frame `signature(x = "ARFIMAforecast")`: Extracts the forecasts. Takes many additional arguments (see note below).

as.list `signature(x = "ARFIMAforecast")`: Extracts the forecast list with all rollframes.

fpm `signature(object = "ARFIMAforecast")`: Forecast performance measures.

show `signature(object = "ARFIMAforecast")`: Forecast summary returning the 0-roll frame only.

Note

There are 3 main extractor functions for the ARFIMA object which is admittedly the most complex in the package as a result of allowing for rolling forecasts. The `as.array` extracts an array object where each page of the array represents a roll. The `as.list` method works similarly returns instead a list object. There are no additional arguments to these extractor functions and they will return all the forecasts. The `as.data.frame` method on the other hand provides for 4 additional arguments. The `rollframe` option is for the rolling frame to return (with 0 being the default no-roll) and allows either a valid numeric value or alternatively the character value "all" for which additional options then come into play. When "all" is chosen in the `rollframe` argument, the data.frame returned may be time aligned (logical option `aligned`) in which case the logical option `prepad` indicates whether to pad the values prior to the forecast start time with actual values or NA (value `FALSE`). Finally, the `type` option controls whether to return all forecasts (value 0, default), return only those forecasts which have in sample equivalent data (value 1) or return only those values which are truly forecasts without in sample data (value 2). Depending on the intended usage of the forecasts, some or all these options may be useful to the user when extracting data from the forecast object.

Author(s)

Alexios Ghalanos

`arfimaforecast-methods`

function: ARFIMA Forecasting

Description

Method for forecasting from an ARFIMA model.

Usage

```
arfimaforecast(fitORspec, data = NULL, n.ahead = 10, n.roll = 0, out.sample = 0,
external.forecasts = list(mregfor = NULL), ...)
```

Arguments

`fitORspec`	Either an ARFIMA fit object of class `ARFIMAfit` or alternatively an ARFIMA specification object of class `ARFIMAspec` with valid parameters supplied via the `fixed.pars` argument in the specification.
`data`	Required if a specification rather than a fit object is supplied.
`n.ahead`	The forecast horizon.
`n.roll`	The no. of rolling forecasts to create beyond the first one (see details).
`out.sample`	Optional. If a specification object is supplied, indicates how many data points to keep for out of sample testing.
`external.forecasts`	A list with a matrix of forecasts for the external regressors in the mean.
`...`	.

Details

The forecast function has two dispatch methods allowing the user to call it with either a fitted object (in which case the data argument is ignored), or a specification object (in which case the data is required) with the parameters entered via the `set.fixed<-` methods on an `ARFIMAspec` object.
One step ahead forecasts are based on the value of the previous data, while n-step ahead (n>1) are based on the unconditional mean of the model.
The ability to roll the forecast 1 step at a time is implemented with the `n.roll` argument which controls how many times to roll the n.ahead forecast. The default argument of n.roll = 0 denotes no rolling beyond the first forecast and returns the standard n.ahead forecast. Critically, since n.roll depends on data being available from which to base the rolling forecast, the `arfimafit` function needs to be called with the argument `out.sample` being at least as large as the n.roll argument, or in the case of a specification being used instead of a fit object, the `out.sample` argument directly in the forecast function.

Value

A `ARFIMAforecast` object containing details of the ARFIMA forecast. See the class for details on the returned object and methods for accessing it and performing some tests.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
# Long Horizon Forecast
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp",
fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist

for(i in 1:9){
cf = coef(fit[[i]])
cfmatrix[i, match(names(cf), colnames(cfmatrix))] =  cf
}

umean = rep(0, 9)
for(i in 1:9){
umean[i] = uncmean(fit[[i]])
}

forc = vector(mode = "list", length = 9)
```

```
for(i in 1:9){
forc[[i]] = arfimaforecast(fit[[i]], n.ahead = 100)
}

lmean40 = sapply(forc, FUN = function(x) as.numeric(as.data.frame(x)[40,1]))
cfmatrix1 = cbind(cfmatrix, umean, lmean40)
colnames(cfmatrix1) = c(colnames(cfmatrix1[,1:7]), "uncmean", "forecast40")

# forecast with spec to check results
forc2 = vector(mode = "list", length = 9)
for(i in 1:9){
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
setfixed(spec) = as.list(coef(fit[[i]]))
forc2[[i]] = arfimaforecast(spec, data = sp500ret, n.ahead = 100)
}
lmean240 = sapply(forc2, FUN = function(x) as.numeric(as.data.frame(x)[40,1]))
cfmatrix2 = cbind(cfmatrix, umean, lmean240)
colnames(cfmatrix2) = c(colnames(cfmatrix2[,1:7]), "uncmean", "forecast40")

cat("\nARFIMAforecast from ARFIMAfit and ARFIMAspec check:")
cat("\nFit\n")
print(cfmatrix1, digits = 4)
cat("\nSpec\n")
print(cfmatrix2, digits = 4)
# methods and slots
slotNames(forc[[1]])
showMethods(classes="ARFIMAforecast")
# summary
show(forc[[1]])
# Extractor Functions
# as array (array dimension [3] is 1 since n.roll = 0 i.e. no rolling beyond
# the first)
as.array(forc[[1]])
# as.data.frame
as.data.frame(forc[[1]])
# as.list
as.list(forc[[1]])

# Rolling Forecast
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
arfima = FALSE), distribution.model = dist[i])
fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp",
out.sample = 1000, fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist
```

```
for(i in 1:9){
cf = coef(fit[[i]])
cfmatrix[i, match(names(cf), colnames(cfmatrix))] =  cf
}


forc = vector(mode = "list", length = 9)
for(i in 1:9){
forc[[i]] = arfimaforecast(fit[[i]], n.ahead = 1, n.roll = 999)
}
rollforc = sapply(forc, FUN = function(x) t(unlist(as.data.frame(x,
rollframe = "all", aligned = FALSE))))

# forecast performance measures:
fpmlist = vector(mode = "list", length = 9)
for(i in 1:9){
fpmlist[[i]] = fpm(forc[[i]], summary = FALSE)
}

par(mfrow = c(1,2))
dd = rownames(tail(sp500ret, 1250))
clrs = rainbow(9, alpha = 1, start = 0.4, end = 0.95)
plot(as.Date(dd), tail(sp500ret[,1], 1250), type = "l",
ylim = c(-0.02, 0.02), col = "lightgrey", ylab = "", xlab = "",
main = "Rolling 1-ahead Forecasts\nvs Actual")
for(i in 1:9){
tmp = tail(sp500ret[,1], 1250)
tmp[251:1250] = rollforc[1:1000,i]
lines(as.Date(dd), c(rep(NA, 250), tmp[-(1:250)]), col = clrs[i])
}
legend("topleft", legend = dist, col = clrs, fill = clrs, bty = "n")

# plot deviation measures and range
tmp = vector(mode = "list", length = 9)
for(i in 1:9){
tmp[[i]] = fpmlist[[i]][,"AE"]
names(tmp[[i]]) = dist[i]
}
boxplot(tmp, col = clrs, names = dist, range  = 6, notch = TRUE,
main = "Rolling 1-ahead Forecasts\nAbsolute Deviation Loss")

# fpm comparison
compm = matrix(NA, nrow = 3, ncol = 9)
compm = sapply(fpmlist, FUN = function(x) c(mean(x[,"SE"]), mean(x[,"AE"]),
mean(x[,"DAC"])))
colnames(compm) = dist
rownames(compm) = c("MSE", "MAD", "DAC")

cat("\nRolling Forecast FPM\n")
print(compm, digits = 4)
cat("\nMethods Check\n")
as.data.frame(forc[[1]], rollframe = 0)
as.data.frame(forc[[1]], rollframe = 999)
```

```
t(as.data.frame(forc[[1]], rollframe = "all", aligned = FALSE))
fpm(forc[[1]], summary = TRUE)
show(forc[[1]])

## End(Not run)
```

ARFIMAmultifilter-class

class: ARFIMA Multiple Filter Class

Description

Class for the ARFIMA Multiple filter.

Slots

`filter`: Object of class `"vector"`

`desc`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

fitted `signature(object = "ARFIMAmultifilter")`: extracts the fitted values.

residuals `signature(object = "ARFIMAmultifilter")`: extracts the residuals.

coef `signature(object = "ARFIMAmultifilter")`: extracts the coefficients.

likelihood `signature(object = "ARFIMAmultifilter")`: extracts the likelihood.

show `signature(object = "ARFIMAmultifilter")`: filter summary.

Author(s)

Alexios Ghalanos

ARFIMAmultifit-class *class: ARFIMA Multiple Fit Class*

Description

Class for the ARFIMA Multiple fit.

Slots

`fit`: Object of class `"vector"`

`desc`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

coef `signature(object = "ARFIMAmultifit")`: extracts the coefficients.

likelihood `signature(object = "ARFIMAmultifit")`: extracts the likelihood.

fitted `signature(object = "ARFIMAmultifit")`: extracts the fitted values.

residuals `signature(object = "ARFIMAmultifit")`: extracts the residuals.

show `signature(object = "ARFIMAmultifit")`: fit summary.

Author(s)

Alexios Ghalanos

ARFIMAmultiforecast-class *class: ARFIMA Multiple Forecast Class*

Description

Class for the ARFIMA Multiple forecast.

Slots

`forecast`: Object of class `"vector"`

`desc`: Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.array `signature(x = "ARFIMAmultiforecast")`: extracts the forecast array with matrix column dimensions equal to the number of assets, row dimension the n.ahead and array dimension equal to the number of rolling forecasts chosen.

as.list `signature(x = "ARFIMAmultiforecast")`: extracts the forecast list of length equal to the number of assets, sublists equal to n.roll, row dimension of each sublist equal to n.ahead and column dimension equal to 1 (series forecasts).

show `signature(object = "ARFIMAmultiforecast")`: forecast summary.

Author(s)

Alexios Ghalanos

ARFIMAmultispec-class *class: ARFIMA Multiple Specification Class*

Description

Class for the ARFIMA Multiple specification.

Slots

`spec`: Object of class `"vector"`

`type`: Object of class `"character"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

show `signature(object = "ARFIMAmultispec")`: specification summary.

Author(s)

Alexios Ghalanos

ARFIMApath-class *class: ARFIMA Path Simulation Class*

Description

Class for the ARFIMA Path simulation.

Slots

`path`: Object of class `"vector"`

`model`: Object of class `"vector"`

`seed`: Object of class `"integer"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.data.frame `signature(x = "ARFIMApath")`: Extracts the simulated path values (see note).

show `signature(object = "ARFIMApath")`: path simulation summary.

Note

The `as.data.frame` function takes optionally 1 additional arguments, namely `which`, indicating the type of simulation path series to extract. Valid values "series" for the simulated series and "residuals" for the simulated residuals. The dimension of the `data.frame` will be `n.sim` by `m.sim`.

Author(s)

Alexios Ghalanos

arfimapath-methods *function: ARFIMA Path Simulation*

Description

Method for simulating the path of an ARFIMA model. This is a convenience function which does not require a fitted object (see note below).

Usage

```
arfimapath(spec, n.sim = 1000, n.start = 0, m.sim = 1, prereturns = NA,
preresiduals = NA, rseed = NA,
custom.dist=list(name = NA, distfit = NA, type = "z"), mexsimdata=NULL, ...)
```

Arguments

spec	An ARFIMA object of class ARFIMAspec with the required parameters of the model supplied via the fixed.pars list argument.
n.sim	The simulation horizon.
n.start	The burn-in sample.
m.sim	The number of simulations.
prereturns	Allows the starting return data to be provided by the user.
preresiduals	Allows the starting residuals to be provided by the user.
rseed	Optional seeding value(s) for the random number generator.
custom.dist	Optional density with fitted object from which to simulate. The "type" argument denotes whether the standardized innovations are passed ("z") else the innovations (anything other than "z").
mexsimdata	Matrix of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided.
...	.

Details

This is a convenience method to allow path simulation of ARFIMA models without the need to supply a fit object as in the arfimasim method. Instead, an arfima spec object is required with the model parameters supplied via the setfixed<- argument to the spec.

Value

A ARFIMApath object containing details of the ARFIMA path simulation.

Author(s)

Alexios Ghalanos

ARFIMAroll-class *class: ARFIMA Rolling Forecast Class*

Description

Class for the ARFIMA rolling forecast.

Slots

roll: Object of class "vector"

forecast: Object of class "vector"

model: Object of class "vector"

Extends

Class "ARFIMA", directly. Class "rGARCH", by class "ARFIMA", distance 2.

Methods

as.ARFIMAforecast `signature(object = "ARFIMAroll")`: extracts and converts the forecast object contained in the roll object to one of ARFIMAforecast given the refit number supplied by additional argument 'refit' (defaults to 1).

as.data.frame `signature(x = "ARFIMAroll")`: extracts various values from object (see note).

fpm `signature(object = "ARFIMAroll")`: Forecast performance measures.

report `signature(object = "ARFIMAroll")`: roll backtest reports (see note).

Note

The `as.data.frame` extractor method allows the extraction of a variety of values from the object. Additional arguments are:
which indicates the type of value to return. Valid values are "coefs" returning the parameter coefficients for all refits, "density" for the parametric density, "coefmat" for the parameter coefficients with their respective standard errors and t- and p- values, "LLH" for the likelihood across the refits, and "VaR" for the Value At Risk measure if it was requested in the roll function call.
n.ahead for the n.ahead forecast horizon to return if `which` was used with arguments "density" or 'VaR'.
refit indicates which refit window to return the "coefmat" if that is chosen. If "series" is chosen under via the *which* argument, then the forecast series is returned for a particular refit, else when "all" is used it returns the complete forecasted series across all refits.
The `report` method takes the following additional arguments:
type for the report type. Valid values are "VaR" for the Value at Risk report based on the unconditional and conditional coverage tests for VaR exceedances (discussed below) and "fpm" for forecast performance measures.
n.ahead for the rolling n.ahead forecasts (defaults to 1).
VaR.alpha for the Value at Risk backtest report, this is the tail probability and defaults to 0.01.
conf.level the confidence level upon which the conditional coverage hypothesis test will be based on (defaults to 0.95).
Kupiec's unconditional coverage test looks at whether the amount of expected versus actual exceedances given the tail probability of VaR actually occur as predicted, while the conditional coverage test of Christoffersen is a joint test of the unconditional coverage and the independence of the exceedances. Both the joint and the separate unconditional test are reported since it is always possible that the joint test passes while failing either the independence or unconditional coverage test.
The "fpm" does not take any additional arguments, but instead returns the forecast performance measures for all "n.ahead" values.

Author(s)

Alexios Ghalanos

`arfimaroll-methods` *function: ARFIMA Rolling Density Forecast and Backtesting*

Description

Method for creating rolling density forecast from ARFIMA models with option for refitting every n periods and some multicore parallel functionality.

Usage

```
arfimaroll(spec, data, n.ahead = 1, forecast.length = 500, refit.every = 25,
refit.window = c("recursive", "moving"), parallel = FALSE,
parallel.control = list(pkg = c("multicore", "snowfall"), cores = 2),
solver = "solnp", fit.control = list(), solver.control = list(),
calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.05), ...)
```

Arguments

`spec`	An ARFIMA spec object specifiying the desired model for testing.
`data`	A univariate dataset.
`n.ahead`	The number of periods to forecast.
`forecast.length`	The length of the total forecast for which out of sample data from the dataset will be excluded for testing.
`refit.every`	Determines every how many periods the model is re-estimated.
`refit.window`	Whether the refit is done on an expanding window including all the previous data or a moving window, the length of the window determined by the argument above (*refit.every*).
`parallel`	Whether to make use of parallel processing on multicore systems.
`parallel.control`	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
`solver`	The solver to use.
`fit.control`	Control parameters parameters passed to the fitting function.
`solver.control`	Control parameters passed to the solver.
`calculate.VaR`	Whether to calculate forecast Value at Risk during the estimation.
`VaR.alpha`	The Value at Risk tail level to calculate.
`...`	.

Details

ARFIMA models generate a partially time varying density based on the variation in the conditional mean values (sigma, skewness and shape are not time varying). The function first generates rolling forecasts of the ARFIMA model and then rescales the density from a standardized (0, 1, skew, shape) to the one representing the underlying return process (mu, sigma, skew, shape). Given this information it is then a simple matter to generate any measure of risk through the analytical evaluation of some type of function of the density. The function calculates one such measure (VaR), but since the full density parameters are returned, the user can calculate many others.
The argument *refit.every* determines every how many periods the fit is recalculated and the total forecast length actually calculated. For example, for a forecast length of 500 and *refit.every* of 25, this is 20 windows of 25 periods each for a total actual forecast length of 500. However, for a *refit.every* of 30, we take the floor of the division of 500 by 30 which is 16 windows of 30 periods each for a total actual forecast length of 480 (16 x 30). The important thing to remember about the refit.every is that it acts like the n.roll argument in the `arfimaforecast` function as it determines the number of rolls to perform. For example for n.ahead of 1 and refit.every of 25, the forecast is rolled every day using the filtered (actual) data of the previous period while for n.ahead of 1 and refit.every of 1 we will get 1 n.ahead forecasts for every day after which the model is refitted and reforecast for a total of 500 refits (when length.forecast is 500)!

Value

An object of class `ARFIMAroll`.

Author(s)

Alexios Ghalanos

`ARFIMAsim-class` *class: ARFIMA Simulation Class*

Description

Class for the ARFIMA simulation.

Slots

`simulation`: Object of class `"vector"`

`model`: Object of class `"vector"`

`seed`: Object of class `"integer"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

as.data.frame `signature(x = "ARFIMAsim")`: extracts the simulated values (see note).

show `signature(object = "ARFIMAsim")`: simulation summary.

Note

The `as.data.frame` function takes optionally 1 additional arguments, namely `which`, indicating the type of simulation series to extract. Valid values are "series" for the simulated series and "residuals" for the simulated residuals. The dimension of the `data.frame` will be `n.sim` by `m.sim`.

Author(s)

Alexios Ghalanos

`arfimasim-methods` *function: ARFIMA Simulation*

Description

Method for simulation from ARFIMA models.

Usage

```
arfimasim(fit, n.sim = 1000, n.start = 0, m.sim = 1, startMethod =
c("unconditional", "sample"), prereturns = NA, preresiduals = NA,
rseed = NA, custom.dist = list(name = NA, distfit = NA, type = "z"),
mexsimdata = NULL, ...)
```

Arguments

`fit`	An ARFIMA fit object of class `ARFIMAfit`.
`n.sim`	The simulation horizon.
`n.start`	The burn-in sample.
`m.sim`	The number of simulations.
`startMethod`	Starting values for the simulation.
`prereturns`	Allows the starting return data to be provided by the user.
`preresiduals`	Allows the starting residuals to be provided by the user.
`rseed`	Optional seeding value(s) for the random number generator.
`custom.dist`	Optional density with fitted object from which to simulate. The "type" argument denotes whether the standardized innovations are passed ("z") else the innovations (anything other than "z"). See notes below for details.
`mexsimdata`	Matrix of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this can be provided else will be ignored.
`...`	.

Details

The custom.dist option allows for defining a custom density which exists in the users workspace with methods for "r" (sampling, e.g. rnorm) and "d" (density e.g. dnorm). It must take a single fit object as its second argument. Alternatively, custom.dist can take any name in the name slot (e.g."sample") and a matrix in the fit slot with dimensions equal to m.sim (columns) and n.sim (rows).

Value

A `ARFIMAsim` object containing details of the ARFIMA simulation.

Author(s)

Alexios Ghalanos

`ARFIMAspec-class` *class: ARFIMA Specification Class*

Description

Class for the ARFIMA specification.

Slots

`model`**:** Object of class `"vector"`

Extends

Class `"ARFIMA"`, directly. Class `"rGARCH"`, by class "ARFIMA", distance 2.

Methods

show `signature(object = "ARFIMAspec")`: Specification summary.

setfixed<- `signature(object = "ARFIMAspec", value = "vector")`: Sets the fixed parameters (which must be supplied as a named list).

setstart<- `signature(object = "ARFIMAspec", value = "vector")`: Sets the starting parameters (which must be supplied as a named list).

uncmean `signature(object = "ARFIMAspec")`: Returns the unconditional mean of a specification which has been assigned fixed parameters.

Author(s)

Alexios Ghalanos

`arfimaspec-methods` *function: ARFIMA Specification*

Description

Method for creating an ARFIMA specification object prior to fitting.

Usage

```
arfimaspec(mean.model = list(armaOrder = c(1, 1), include.mean = TRUE,
    arfima = FALSE, external.regressors = NULL), distribution.model = "norm",
    start.pars = list(), fixed.pars = list(), ...)
```

Arguments

`mean.model`	List containing the mean model specification: `armaOrder` The autoregressive (ar) and moving average (ma) orders (if any). `include.mean` Whether to include the mean. `arfima` Whether to include arfima. `external.regressors` A matrix object containing the external regressors to include in the mean equation with as many rows as will be included in the data (which is passed in the fit function).
`distribution.model`	The distribution density to use for the innovations. Valid choices are "norm" for the normal distibution, "snorm" for the skew-normal distribution, "std" for the student-t, "sstd" for the skew-student-t, "ged" for the generalized error distribution, "sged" for the skew-generalized error distribution, "nig" for the normal inverse gaussian distribution, "ghyp" for the Generalized Hyperbolic, and "jsu" for Johnson's SU distribution. Note that some of the distributions are taken from the fBasics package and implenented locally here for convenience. The "jsu" distribution is the reparametrized version from the "gamlss" package.
`start.pars`	List of staring parameters for the optimization routine. These are not usually required unless the optimization has problems converging.
`fixed.pars`	List of parameters which are to be kept fixed during the optimization. It is possible that you designate all parameters as fixed so as to quickly recover just the results of some previous work or published work. The optional argument "fixed.se" in the `arfimafit` function indicates whether to calculate standard errors for those parameters fixed during the post optimization stage.
`...`	.

Details

The specification allows for flexibility in ARFIMA modelling.
In order to understand which parameters can be entered in the start.pars and fixed.pars optional arguments, the list below exposes the names used for the parameters:(note that when a parameter is followed by a number, this represents the order of the model. Just increment the number for higher

orders):
Mean Model:

constant	`mu`
AR term	`ar1`
MA term	`ma1`
exogenous regressors	`mxreg1`
arfima	`darfima`

Distribution Model:

dlambda	`dlambda (for GHYP distribution)`
skew	`skew`
shape	`shape`

Value

A `ARFIMAspec` object containing details of the ARFIMA specification.

Author(s)

Alexios Ghalanos

`BerkowitzLR` *Berkowitz Density Forecast Likelihood Ratio Test*

Description

Implements the Berkowitz Density Forecast Likelihood Ratio Test.

Usage

```
BerkowitzLR(data, lags = 1, significance = 0.05)
```

Arguments

`data`	A univariate vector of standard normal transformed values (see details and example).
`lags`	The number of autoregressive lags (positive and greater than 0).
`significance`	The level of significance at which the Null Hypothesis is evaluated.

Details

See not below.

Value

A list with the following items:

uLL	The unconditional Log-Likelihood of the maximized values.
rLL	The restricted Log-Likelihood with zero mean, unit variance and zero coefficients in the autoregressive lags.
LR	The Likelihood Ratio Test Statistic.
LRp	The LR test statistic p-value (distributed chisq with 2+lags d.o.f).
H0	The Null Hypothesis.
Test	The test of the Null Hypothesis at the requested level of significance.
mu	The estimated mean of the model.
sigma	The estimated sd of the model.
rho	The estimated autoregressive coefficients of the model.
JB	The Jarque Bera Test of Normality Statistic.
JBp	The Jarque Beta Test Statistic p-value.

Note

The data must first be transformed before being submitted to the function as described here. Given a forecast density (d*) at time t, transform the actual(observed) realizations of the data by applying the distribution function of the forecast density (p*). This will result in a set of uniform values (see Rosenblatt (1952)). Transform those value into standard normal variates by applying the standard normal quantile function (qnorm). The example below hopefully clarifies this. The function also returns the Jarque Bera Normality Test statistic as an additional check of the normality assumption which the test does not explicitly account for (see Dowd reference).

Author(s)

Alexios Ghalanos

References

Berkowitz, J. 2001, Testing density forecasts, with applications to risk management, *Journal of Business and Economic Statistics*, **19(4)**, 465–474.
Dowd, K. 2004, A modified Berkowitz back-test, *RISK Magazine*, **17(4)**, 86–87.
Jarque, C.M. and Bera, A.K. 1987m A test for normality of observations and regression residuals, *International Statistical Review*, **55(2)**, 163–172.
Rosenblatt, M. 1952, Remarks on a multivariate transformation, *The Annals of Mathematical Statistics*, **23(3)**, 470–472.

Examples

```
## Not run:
#  A univariate GARCH model is used with rolling out of sample forecasts.
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(6,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "nig")
fit = ugarchfit(spec, data = dji30ret[, 1, drop = FALSE], out.sample = 1000)
pred = ugarchforecast(fit, n.ahead = 1, n.roll = 999)
dmatrix = cbind(as.array(pred)[,2,],as.array(pred)[,1,], coef(fit)["skew"],
coef(fit)["shape"])
colnames(dmatrix) = c("mu", "sigma", "skew", "shape")

# Get Realized (Oberved) Data
obsx = tail(dji30ret[,1], 1000)
# you can check that this is correct by looking at the dates of the first and
# last predictions:
as.data.frame(pred, rollframe = 0)
head(tail(dji30ret[,1, drop = FALSE], 1000), 1)

as.data.frame(pred, rollframe = 999)
tail(dji30ret[,1, drop = FALSE], 1)

# Transform to Uniform
uvector = apply(cbind(obsx,dmatrix), 1, FUN = function(x) pdist("nig", q = x[1],
mu = x[2], sigma = x[3], skew = x[4], shape = x[5]))

# hist(uvector)
# transform to N(0,1)
nvector = qnorm(uvector)
test = BerkowitzLR(data = nvector, lags = 1, significance = 0.05)
print(test)
# We fail to reject Null at the 5% level, but not at 10%.
# plot(density(nvector))

## End(Not run)
```

DACTest *Directional Accuracy Test*

Description

Implements the Directional Accuracy Test of Pesaran and Timmerman and Excess Profitability Test of Anatolyev and Gerko.

Usage

```
DACTest(forecast, actual, test = c("PT", "AG"), conf.level = 0.95)
```

Arguments

forecast	A numeric vector of the forecasted values.
actual	A numeric vector of the actual (realized) values.
test	Choice of Pesaran and Timmermann ('PT') or Anatolyev and Gerko ('AG') tests.
conf.level	The confidence level at which the Null Hypothesis is evaluated.

Details

See the references for details on the tests. The Null is effectively that of independence, and distributed as N(0,1).

Value

A list with the following items:

Test	The type of test performed.
Stat	The test statistic.
p-value	The p-value of the test statistic.
H0	The Null Hypothesis.
Decision	Whether to reject or not the Null given the conf.level.
DirAcc	The directional accuracy of the forecast.

Author(s)

Alexios Ghalanos

References

Anatolyev, S. and Gerko, A. 2005, A trading approach to testing for predictability, *Journal of Business and Economic Statistics*, **23(4)**, 455–461.
Pesaran, M.H. and Timmermann, A. 1992, A simple nonparametric test of predictive performance, *Journal of Business and Economic Statistics*, **10(4)**, 461–465.

---

dji30ret *data: Dow Jones 30 Constituents Closing Value Log Return*

Description

Dow Jones 30 Constituents closing value log returns from 1987-03-16 to 2009-02-03 from Yahoo Finance. Note that AIG was replaced by KFT (Kraft Foods) on September 22, 2008. This is not reflected in this data set as that would bring the starting date of the data to 2001.

Usage

```
data(dji30ret)
```

Format

A data.frame containing 30x5521 observations.

Source

Yahoo Finance

---

dmbp *data: Deutschemark/British pound Exchange Rate*

Description

The Bollerslev-Ghysel benchmark dataset. The variables in the data set are:
1. The daily percentage nominal returns computed as 100 [ln(Pt) - ln(Pt-1)], where Pt is the bilateral Deutschemark/British pound rate constructed from the corresponding U.S. dollar rates.
2. A dummy variable that takes the value of 1 on Mondays and other days following no trading in the Deutschemark or British pound/ U.S. dollar market during regular European trading hours and 0 otherwise.

Usage

```
data(dmbp)
```

Format

A data.frame containing 2x1974 observations.

Source

JBES Data Archive ftp://www.amstat.org/jbes/View/

References

Bollerslev, T. and Ghysels, E. 1996, Periodic Autoregressive Conditional Heteroscedasticity , *Journal of Business and Economic Statistics*, **14**, 139–151.

ForwardDates-methods *function: Generate Future Dates*

Description

Given a starting date, this helper function generates a set of future dates (excl.weekends) for use in forecasting and simulation when using external regressors.

Usage

```
ForwardDates(Dates, n.ahead, date.format, periodicity = "days")
```

Arguments

Dates	A character vector of dates. The last date is used as the starting date for the forward date creation.
n.ahead	The number of dates to generate forward.
date.format	The format of the dates e.g. "%Y-%m-%d"" .
periodicity	Currently only days is supported.

Value

A POSIXct vector of future dates.

Note

This is a helper function particularly useful when used with the weekday dummy variable for simulation and forecasting in light of weekday dummy external regressors in the mean or variance equation. For example, if fitting a GARCH model with a "Monday" dummy variable in the mean equation, then for simulation or forecasting, one needs a set of forward deterministic dummy variables for the Mondays going forward.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
Dates = rownames(sp500ret)
# generate the 100 forward non-weekend days
fwd100 = ForwardDates(Dates, n.ahead=100, date.format = "%Y-%m-%d",
periodicity = "days")
# create a dummy vector for those forward days which are Mondays
fwdMonday = WeekDayDummy(as.character(fwd100), date.format = "%Y-%m-%d",
weekday = "Monday")

## End(Not run)
```

GARCHboot-class *class: GARCH Bootstrap Class*

Description

High Level GARCH bootstrap class to hold the univariate and multivariate boot objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHboot" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHboot")
```

GARCHdistribution-class *class: GARCH Parameter Distribution Class*

Description

High Level GARCH parameter distribution class to hold the univariate and multivariate boot objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHdistribution" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHdistribution")
```

GARCHfilter-class *class: GARCH Filter Class*

Description

High Level GARCH filter class to hold the univariate and multivariate filter objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHfilter" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHfilter")
```

GARCHfit-class *class: GARCH Fit Class*

Description

High Level GARCH fit class to hold the univariate and multivariate fits objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHfit" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHfit")
```

GARCHforecast-class *class: GARCH Forecast Class*

Description

High Level GARCH forecast class to hold the univariate and multivariate forecast objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHforecast" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHforecast")
```

GARCHpath-class *class: GARCH Path Simulation Class*

Description

High Level GARCH Path simulation class to hold the univariate and multivariate path simulation objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHpath" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHpath")
```

GARCHroll-class *class: GARCH Roll Class*

Description

High Level GARCH roll class to hold the univariate and multivariate roll objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHroll" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHroll")
```

GARCHsim-class *class: GARCH Simulation Class*

Description

High Level GARCH simulation class to hold the univariate and multivariate simulation objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHsim" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHsim")
```

GARCHspec-class *class: GARCH Spec Class*

Description

High Level GARCH spec class to hold the univariate and multivariate spec objects.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHspec" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHspec")
```

GARCHtests-class *class: GARCH Tests Class*

Description

GARCH High level inference and other tests class.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "rGARCH", directly.

Methods

No methods defined with class "GARCHtests" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("GARCHtests")
```

ghyptransform *Distribution: Generalized Hyperbolic Transformation and Scaling*

Description

The function scales the distributions from the (0, 1) zeta-rho GARCH parametrization to the alpha-beta parametrization and performs the appropriate scaling to the parameters given the estimated sigma and mu.

Usage

```
ghyptransform(mu = 0, sigma = 1,  skew = 0, shape = 3, lambda = -0.5)
```

Arguments

mu	Either the conditional time-varying (vector) or unconditional mean estimated from the GARCH process.
sigma	The conditional time-varying (vector) sigma estimated from the GARCH process.
skew, shape, lambda	The conditional non-time varying skewness (rho) and shape (zeta) parameters estimated from the GARCH process (zeta-rho), and the GHYP lambda parameter ('dlambda' in the estimation).

Details

The GHYP transformation is taken from Rmetrics internal function and scaled as in Blaesild (see references).

Value

A matrix of size nrows(sigma) x 4 of the scaled and transformed parameters to be used in the alpha-beta parametrized GHYP distribution functions.

Author(s)

Diethelm Wuertz for the Rmetrics R-port of the nig transformation function.
Alexios Ghalanos for rugarch implementation.

References

Blaesild, P. 1981, The two-dimensional hyperbolic distribution and related distributions, with an application to Johannsen's bean data, *Biometrika*, **68**, 251–263.
Eberlein, E. and Prauss, K. 2000, The Generalized Hyperbolic Model Financial Derivatives and Risk Measures, *Mathematical Finance Bachelier Congress*, 245–267.

---

multifilter-methods *function: Univariate GARCH and ARFIMA Multiple Filtering*

Description

Method for multiple filtering of a variety of univariate GARCH and ARFIMA models.

Usage

```
multifilter(multifitORspec, data = NULL, out.sample = 0, n.old = NULL,
parallel = FALSE, parallel.control = list(pkg = c("multicore", "snowfall"),
cores = 2), ...)
```

Arguments

multifitORspec	Either a univariate GARCH or ARFIMA multiple fit object of class uGARCHmultifit and ARFIMAmultifit, or alternatively a univariate GARCH or ARFIMA multiple specification object of class uGARCHmultispec and ARFIMAmultispec with valid parameters supplied via the fixed.pars argument in the individual specifications.
data	Required if a multiple specification rather than a multiple fit object is supplied. A multivariate data object. Can be a matrix or data.frame object, no other class supported at present.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (as in ugarchfit function).
n.old	For comparison with uGARCHfit or ARFIMAfit models using the out.sample argument, this is the length of the original dataset (see details).
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Value

A uGARCHmultifilter object containing details of the multiple GARCH filter. A ARFIMAmultifilter object containing details of the multiple ARFIMA filter.

Author(s)

Alexios Ghalanos

---

multifit-methods *function: Univariate GARCH and ARFIMA Multiple Fitting*

Description

Method for multiple fitting a variety of univariate GARCH and ARFIMA models.

Usage

```
multifit(multispec, data, out.sample = 0,
solver = "solnp", solver.control = list(),
fit.control = list(stationarity = 1, fixed.se = 0, scale = 0),
parallel = FALSE, parallel.control = list(pkg = c("multicore", "snowfall"),
cores = 2), ...)
```

Arguments

multispec	A multiple GARCH or ARFIMA spec object of class uGARCHmultispec and ARFIMAmultispec.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (see details).
data	A multivariate data object. Can be a matrix or data.frame object, no other class supported at present.
solver	One of either "nlminb" or "solnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine. Stationarity (only for the GARCH case) explicitly imposes the variance stationarity constraint during optimization. The fixed.se argument controls whether standard errors should be calculated for those parameters which were fixed (through the fixed.pars argument of the ugarchspec or arfimaspec functions). The scale parameter controls whether the data should be scaled before being submitted to the optimizer.
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Value

A uGARCHmultifit or ARFIMAmultifit object containing details of the GARCH or ARFIMA fits.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec()
mspec = multispec( replicate(spec, n = 4) )
fitlist = multifit(multispec = mspec, data = dji30ret[,1:4])

## End(Not run)
```

multiforecast-methods *function: Univariate GARCH and ARFIMA Multiple Forecasting*

Description

Method for multiple forecasting from a variety of univariate GARCH and ARFIMA models.

Usage

```
multiforecast(multifitORspec, data = NULL, n.ahead = 1, n.roll = 0,
out.sample = 0,
external.forecasts = list(mregfor = NULL, vregfor = NULL), parallel = FALSE,
parallel.control = list(pkg = c("multicore", "snowfall"), cores = 2), ...)
```

Arguments

multifitORspec	Either a univariate GARCH or ARFIMA multiple fit object uGARCHmultifit and ARFIMAmultifit, or alternatively a univariate GARCH or ARFIMA multiple specification object of class uGARCHmultispec and ARFIMAmultispec with valid parameters supplied via the setfixed<- function in the individual specifications.
data	Required if a multiple specification rather than a multiple fit object is supplied. A multivariate data object. Can be a matrix or data.frame object, no other class supported at present.
n.ahead	The forecast horizon.
n.roll	The no. of rolling forecasts to create beyond the first one.
out.sample	Optional. If a specification object is supplied, indicates how many data points to keep for out of sample testing. If this is not a vector equal to the column dimension of the data, then it will be replicated to that dimension, else it must be of same length as the data column dimension.
external.forecasts	A list with forecasts for the external regressors in the mean and/or variance equations if specified.
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Value

A uGARCHmultiforecast or ARFIMAmultiforecast object containing details of the multiple GARCH or ARFIMA forecasts. See the class for details.

Author(s)

Alexios Ghalanos

`multispec-methods` *function: Univariate multiple GARCH Specification*

Description

Method for creating a univariate multiple GARCH or ARFIMA specification object prior to fitting.

Usage

```
multispec( speclist )
```

Arguments

`speclist`	A list with as many univariate GARCH or ARFIMA specifications of class `uGARCHspec` and `ARFIMAspec` as there will be columns in the data object passed to one of the other methods which uses a multiple specification object (fitting, filtering and forecasting).

Value

A `uGARCHmultispec` or `ARFIMAmultispec` object containing details of the multiple GARCH or ARFIMA specifications.

Author(s)

Alexios Ghalanos

Examples

```
# how to make a list with 2 uGARCHspec objects of the same type
spec = ugarchspec()
mspec = multispec( replicate(2, spec) )
# note that replicate(spec, 2) does not work...be careful about the order
# else explicity name 'n' (i.e. n = 2)

# or simply combine disparate objects
spec1 = ugarchspec(distribution = "norm")
spec2 = ugarchspec(distribution = "std")
mspec = multispec( c( spec1, spec2 ) )
```

rGARCH-class *class: rGARCH Class*

Description

Highest Level Virtual Package Class to which all other classes belong.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "rGARCH" in the signature.

Author(s)

Alexios Ghalanos

Examples

```
showClass("rGARCH")
```

rgarchdist *Distribution: rugarch distribution functions*

Description

Density, distribution function, quantile function, random generation and fitting from the univariate distributions implemented in the rugarch package, with functions for skewness and excess kurtosis given density skew and shape parameters.

`rgarchdist`	rugarch univariate distributions,
`fitdist`	MLE parameter fit for the rugarch univariate distributions,

Usage

```
ddist(distribution = "norm", y, mu = 0, sigma = 1, lambda = -0.5, skew = 1,
shape = 5)
pdist(distribution = "norm", q, mu = 0, sigma = 1, lambda = -0.5, skew = 1,
shape = 5)
qdist(distribution = "norm", p, mu = 0, sigma = 1, lambda = -0.5, skew = 1,
shape = 5)
rdist(distribution = "norm", n, mu = 0, sigma = 1, lambda = -0.5, skew = 1,
shape = 5)
```

```
fitdist(distribution = "norm", x, control=list())
dskewness(distribution = "norm", skew = 1, shape = 5, lambda = -0.5)
dkurtosis(distribution = "norm", skew = 1, shape = 5, lambda = -0.5)
```

Arguments

distribution	The distribution name. Valid choices are "norm", "snorm", "std", "sstd", "ged", "sged", "nig", "jsu".
mu, sigma, skew, shape	location, scale and skewness and shape parameters (see details).
lambda	The additional shape parameter for the Generalized Hyperbolic and NIG distributions.
n	The number of observations.
p	A numeric vector of probabilities.
y, q	A numeric vector of quantiles.
x	A univariate dataset (for fitting routine).
control	Control parameters passed to the solnp solver.

Details

For the dQuotenig and "ghyp" distributions, the shape, skew and lambda are transformed from the 'zeta-rho' to the 'alpha-beta' parametrization and then scaled by the mean and standard deviation. The fitting routines use the solnp solver and minimize the negative of the log-likelihood. The "dskewness" and "dkurtosis" functions take as inputs the distribution name, skew and shape parameters and return the skewneness and excess kurtosis of the distribution. The functions are not at present vectorized.

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.

fitdist returns a list with the following components:

par	The best set of parameters found.
value	The likelihood values of the optimization (vector whose length represents the number of major iterations).
convergence	An integer code. 0 indicates successful convergence.
lagrange	The lagrange multiplier value at convergence.
h	The hessian at the solution.
xineq0	The value of the inequality constraint multiplier (NULL for the distribution fit problems).

dskewness returns the skewness of the distribution. dkurtosis returns the excess kurtosis of the distribution.

Author(s)

Diethelm Wuertz for the Rmetrics R-port of the "norm", "snorm", "std", "sstd", "ged", "sged" and "nig" distrbutions.
Rigby, R. A. and Stasinopoulos D. M for the JSU distribution in the `gamlss` package.
Alexios Ghalanos for rugarch implementation and higher moment distribution functions.

References

Johnson, N. L. 1954, Systems of frequency curves derived from the first law of Laplace, *Trabajos de Estadistica*, **5**, 283–291.
Barndorff-Nielsen, O. E. 1995, Normal inverse Gaussian processes and the modeling of stock returns, *mimeo: Univ.of Aarhus Denmark*.
Fernandez C. and Steel, M.F.J. 1998, On Bayesian Modelling of Fat Tails and Skewness, *Journal of the American Statistical Association*, 359–371.

`sp500ret` *data: Standard and Poors 500 Closing Value Log Return*

Description

The S\&P500 index closing value log return from 1987-03-10 to 2009-01-30 from yahoo finance.

Usage

```
data(sp500ret)
```

Format

A data.frame containing 1x5523 observations.

Source

Yahoo Finance

`ugarchbench` *Benchmark: The Benchmark Test Suite*

Description

Function for running the rugarch benchmark suite.

Usage

```
ugarchbench( benchmark = c("commercial", "published") )
```

Arguments

benchmark The type of benchmark to run against (see details).

Details

Currently, 2 benchmark suites are available. The "commercial" option runs the standard GARCH, apARCH and gjrGARCH against a commercial based product and reports the results. The data for this bechmarks is "AA" in the dji30ret dataset. The "published" option is based on the published benchmark of Bollerslev and Ghysels for the standard and exponential GARCH models on the dmbp data.

Author(s)

Alexios Ghalanos

Source

http://www.stanford.edu/~clint/bench/index.htm

References

Brooks, C. 1997, GARCH Modelling in Finance: A review of the Software Options, *Economic Journal*, **107(443)**, 1271–1276.

Examples

```
ugarchbench( benchmark = "published")
```

---

uGARCHboot-class *class: Univariate GARCH Bootstrap Class*

Description

Class for the univariate GARCH Bootstrap based Forecasts.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "GARCHboot", directly. Class "rGARCH", by class "GARCHboot", distance 2.

Methods

as.data.frame `signature(x = "uGARCHboot")`: extracts various values from object (see note).

plot `signature(x = "uGARCHboot", y = "missing")`: bootstrap forecast plots.

show `signature(object = "uGARCHboot")`: bootstrap forecast summary.

Note

The as.data.frame function takes optionally the arguments which, being either "sigma" or "series", the argument type, with the options "raw" for the bootstrapped series, "summary" for summary statistics per n.ahead, and "q" for the quantiles of the n.ahead bootstrapped series, for which the option qtile is then required and takes a numeric vector of quantiles (e.g. c(0.05, 0.95)).
The plot method provides for a Parameter Density Plots (only valid for the "full" method), and the series and sigma forecast plots with quantile error lines from the bootstrapped n.ahead distribution. The plot option which relates to either a numeric choice (1:3), an interactive choice ("ask" which is the default) and an all plot choice ("all") for which only plots 2 and 3 are included.

Author(s)

Alexios Ghalanos

References

Pascual, L., Romo, J. and Ruiz, E. 2004, Bootstrap predictive inference for ARIMA processes, *Journal of Time Series Analysis*.
Pascual, L., Romo, J. and Ruiz, E. 2006, Bootstrap prediction for returns and volatilities in GARCH models, *Computational Statistics and Data Analysis*.

See Also

Classes uGARCHforecast, uGARCHfit and uGARCHspec.

---

ugarchboot-methods *function: Univariate GARCH Forecast via Bootstrap*

Description

Method for forecasting the GARCH density based on a bootstrap procedures (see details and references).

Usage

```
ugarchboot(fitORspec, data = NULL, method = c("Partial", "Full"), n.ahead = 10,
n.bootfit = 100, n.bootpred = 500, out.sample = 0, rseed = NA, solver = "solnp",
solver.control = list(), fit.control = list(),
external.forecasts = list(mregfor = NULL, vregfor = NULL), parallel = FALSE,
parallel.control = list(pkg = c("multicore", "snowfall"), cores = 2))
```

Arguments

fitORspec	Either a univariate GARCH fit object of class uGARCHfit or alternatively a univariate GARCH specification object of class uGARCHspec with valid parameters supplied via the setfixed<- function in the specification.
data	Required if a specification rather than a fit object is supplied.
method	Either the full or partial bootstrap (see note).
n.ahead	The forecast horizon.
n.bootfit	The number of simulation based re-fits used to generate the parameter distribution (i.e the parameter uncertainty). Not relevant for the "Partial" method.
n.bootpred	The number of bootstrap replications per parameter distribution per n.ahead forecasts used to generate the predictive density. If this is for the partial method, simply the number of random samples from the empirical distribution to generate per n.ahead.
out.sample	Optional. If a specification object is supplied, indicates how many data points to keep for out of sample testing.
rseed	A vector of seeds to initialize the random number generator for the resampling with replacement method (if supplied should be equal to n.bootfit + n.bootpred).
solver	One of either "nlminb" or "solnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine (as in the ugarchfit method).
external.forecasts	A list with forecasts for the external regressors in the mean and/or variance equations if specified.
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Details

There are two main sources of uncertainty about n.ahead forecasting from GARCH models, namely that arising from the form of the predictive density and due to parameter estimation. The bootstrap method considered here, is based on resampling innovations from the empirical distribution of the fitted GARCH model to generate future realizations of the series and sigma. The "full" method, based on the referenced paper by Pascual et al, takes into account parameter uncertainty by building a simulated distribution of the parameters through simulation and refitting. This process, while more accurate, is very time consuming which is why the parallel option (as in the ugarchdistribution is available and recommended). The "partial" method, only considers distribution uncertainty and while faster, will not generate prediction intervals for the sigma 1-ahead forecast for which only the parameter uncertainty is relevant in GARCH type models.

Value

A uGARCHboot object containing details of the GARCH bootstrapped forecast density.

Author(s)

Alexios Ghalanos

References

Pascual, L., Romo, J. and Ruiz, E. 2004, Bootstrap predictive inference for ARIMA processes, *Journal of Time Series Analysis*.
Pascual, L., Romo, J. and Ruiz, E. 2006, Bootstrap prediction for returns and volatilities in GARCH models, *Computational Statistics and Data Analysis*.

See Also

For specification ugarchspec, fitting ugarchfit, filtering ugarchfilter, forecasting ugarchforecast, simulation ugarchsim, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(variance.model=list(model="gjrGARCH", garchOrder=c(1,1)),
mean.model=list(armaOrder=c(1,1), arfima=FALSE, include.mean=TRUE,
archm = FALSE, archpow = 1), distribution.model="std")
ctrl = list(tol = 1e-7, delta = 1e-9)
fit = ugarchfit(data=dji30ret[, "BA", drop = FALSE], out.sample = 0,
spec = spec, solver = "solnp", solver.control = ctrl,
fit.control = list(scale = 1))
bootpred = ugarchboot(fit, method = "Partial", n.ahead = 120, n.bootpred = 2000)
bootpred
# as.data.frame(bootpred, which = "sigma", type = "q", qtile = c(0.01, 0.05))

## End(Not run)
```

---

uGARCHdistribution-class

class: Univariate GARCH Parameter Distribution Class

Description

Class for the univariate GARCH Parameter Distribution.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "GARCHdistribution", directly. Class "rGARCH", by class "GARCHdistribution", distance 2.

Methods

as.data.frame `signature(x = "uGARCHdistribution")`: Extracts various values from object (see note).

plot `signature(x = "uGARCHdistribution", y = "missing")`: Parameter Distribution Plots.

show `signature(object = "uGARCHdistribution")`: Parameter Distribution Summary.

Note

The `as.data.frame` function takes optionally 2 additional arguments, namely `window` which indicates the particular distribution window number for which data is required (is usually just 1 unless the recursive option was used), and `which` indicating the type of data required. Valid values for the latter are "rmse" for the root mean squared error between simulation fit and actual parameters, "stats" for various statistics computed for the simulations such as log likelihood, persistence, unconditional variance and mean, "coef" for the estimated coefficients (i.e. the parameter distribution and is the default choice), and "coefse" for the estimated robust standard errors of the coefficients (i.e. the parameter standard error distribution).
The plot method offers 4 plot types, namely, Parameter Density Plots (take `window` as additional argument), Bivariate Plots (take `window` as additional argument), Stats and RMSE (only when recursive option used) Plots. The standard option for `which` is used, allowing for a numeric arguments to one of the four plot types else interactive choice via "ask".

Author(s)

Alexios Ghalanos

See Also

Classes `uGARCHforecast`, `uGARCHfit` and `uGARCHspec`.

Examples

```
## Not run:
data(sp500ret)
spec = ugarchspec(variance.model=list(model="gjrGARCH", garchOrder=c(1,1)),
mean.model=list(armaOrder=c(1,1), arfima=FALSE, include.mean=TRUE,
archm = FALSE, archpow = 1), distribution.model="std")

fit = ugarchfit(data=sp500ret[, 1, drop = FALSE], out.sample = 0,
spec = spec, solver = "solnp")

dist = ugarchdistribution(fit, n.sim = 2000, n.start = 50, m.sim = 5)

## End(Not run)
```

ugarchdistribution-methods

function: Univariate GARCH Parameter Distribution via Simulation

Description

Method for simulating and estimating the parameter distribution from a variety of univariate GARCH models as well as the simulation based consistency of the estimators given the data size.

Usage

```
ugarchdistribution(fitORspec, n.sim = 2000, n.start = 1,
m.sim = 100, recursive = FALSE, recursive.length = 6000, recursive.window = 1000,
presigma = NA, prereturns = NA, preresiduals = NA, rseed = NA,
custom.dist = list(name = NA, distfit = NA), mexsimdata = NULL, vexsimdata = NULL,
fit.control = list(), solver = "solnp", solver.control = list(), parallel = FALSE,
parallel.control = list(pkg = c("multicore", "snowfall"), cores = 2), ...)
```

Arguments

fitORspec	Either a univariate GARCH fit object of class uGARCHfit or alternatively a univariate GARCH specification object of class uGARCHspec with valid parameters supplied via the setfixed<- function in the specification.
n.sim	The simulation horizon.
n.start	The burn-in sample.
m.sim	The number of simulations.
recursive	Whether to perform a recursive simulation on an expanding window.
recursive.length	If recursive is TRUE, this indicates the final length of the simulation horizon, with starting length n.sim.
recursive.window	If recursive is TRUE, this indicates the increment to the expanding window. Together with recursive.length, it determines the total number of separate and increasing length windows which will be simulated and fitted.
presigma	Allows the starting sigma values to be provided by the user.
prereturns	Allows the starting return data to be provided by the user.
preresiduals	Allows the starting residuals to be provided by the user.
rseed	Optional seeding value(s) for the random number generator.
custom.dist	Optional density with fitted object from which to simulate.
mexsimdata	Matrix of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided.
vexsimdata	Matrix of simulated external regressor-in-variance data. If the fit object contains external regressors in the variance equation, this must be provided.

solver	One of either "nlminb" or "solnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine (as in the ugarchfit method).
parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
...	.

Details

This method facilitates the simulation and evaluation of the uncertainty of GARCH model parameters. The recursive option also allows the evaluation of the simulation based consistency (in terms of sqrt(N)) of the parameters as the length (n.sim) of the data increases, in the sense of the root mean square error (rmse) of the difference between the simulated and true (hypothesized) parameters. This is a very expensive function, particularly if using the recursive option, both on memory and cpu resources, performing many re-fits of the simulated data in order to generate the parameter distribution and it is therefore suggested that, if available, the parallel functionality should be used (in a system with ideally many cores and at least 4GB of RAM for the recursion option...).

Value

A uGARCHdistribution object containing details of the GARCH simulated parameters distribution.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec, fitting ugarchfit, filtering ugarchfilter, forecasting ugarchforecast, simulation ugarchsim, rolling forecast and estimation ugarchroll, bootstrap forecast ugarchboot.

---

uGARCHfilter-class *class: Univariate GARCH Filter Class*

Description

Class for the univariate GARCH filter.

Extends

Class "GARCHfilter", directly. Class "rGARCH", by class "GARCHfilter", distance 2.

Methods

as.data.frame `signature(x = "uGARCHfilter")`: extracts the position (dates), data, fitted values, residuals and conditional sigma.

fitted `signature(object = "uGARCHfilter")`: extracts the fitted values.

residuals `signature(object = "uGARCHfilter")`: extracts the residuals.

sigma `signature(object = "uGARCHfilter")`: extracts the conditional sigma values.

coef `signature(object = "uGARCHfilter")`: extracts the coefficients.

infocriteria `signature(object = "uGARCHfilter")`: calculates and returns various information criteria.

newsimpact `signature(object = "uGARCHfilter")`: calculates and returns the news impact curve.

likelihood `signature(object = "uGARCHfilter")`: extracts the likelihood.

signbias `signature(object = "uGARCHfilter")`: calculates and returns the sign bias test of Engle and Ng (1993).

gof `signature(object = "uGARCHfilter", groups = "numeric")`: calculates and returns the adjusted goodness of fit statistic and p-values for the fitted distribution based on the Vlaar and Palm paper (1993). Groups is a numeric vector of bin sizes.

persistence `signature(object = "uGARCHfilter", pars = "missing", distribution = "missing", model = "missing", submodel = "missing")`: calculates and returns the persistence of the garch filter model.

halflife `signature(object = "uGARCHfilter", pars = "missing", distribution = "missing", model = "missing")`: calculates and returns the halflife of the garch fit variance given a `uGARCHfilter` object.

uncmean `signature(object = "uGARCHfilter")`: calculates and returns the unconditional mean of the conditional mean equation (constant, ARMAX, arch-in-mean).

uncvariance `signature(object = "uGARCHfilter", pars = "missing", distribution = "missing", model = "missing", submodel = "missing")`: calculates and returns the long run unconditional variance of the garch filter given a `uGARCHfilter` object.

plot `signature(x = "uGARCHfilter", y = "missing")`: filter plots

show `signature(object = "uGARCHfilter")`: filter summary.

Note

The `uGARCHfilter` class contains almost all the methods available with the `uGARCHfit` with the exception of those requiring the scores of the likelihood (i.e the optimization process) such as the nyblom test.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(dji30ret)
ctrl = list(rho = 1, delta = 1e-8, outer.iter = 100, inner.iter = 650,
tol = 1e-6)
spec = ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
distribution.model = "std")
sgarch.fit = ugarchfit(data = dji30ret[,"AA",drop=FALSE], spec = spec,
solver = "solnp", solver.control = ctrl)

spec = ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
distribution.model = "std", fixed.pars = as.list(coef(sgarch.fit)))
sgarch.filter = ugarchfilter(data = dji30ret[,"AA",drop=FALSE], spec = spec)

c(likelihood(sgarch.filter), likelihood(sgarch.fit))
c(uncmean(sgarch.filter), uncmean(sgarch.fit))
c(uncvariance(sgarch.filter), uncvariance(sgarch.fit))

## End(Not run)
```

---

ugarchfilter-methods *function: Univariate GARCH Filtering*

Description

Method for filtering a variety of univariate GARCH models.

Usage

```
ugarchfilter(spec, data, out.sample = 0, n.old=NULL, ...)
```

Arguments

data	A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
spec	A univariate GARCH spec object of class uGARCHspec with the fixed.pars argument having the model parameters on which the filtering is to take place.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (as in ugarchfit function).
n.old	For comparison with uGARCHfit models using the out.sample argument, this is the length of the original dataset (see details).
...	.

Details

The n.old argument is optional and indicates the length of the original data (in cases when this represents a series augmented by newer data). The reason for using this is so that the old and new datasets agree since the original recursion uses the sum of the residuals to start the recursion and therefore is influenced by new data. For a small augmentation the values converge after x periods, but it is sometimes preferable to have this option so that there is no forward looking information contaminating the study.

Value

A uGARCHfilter object containing details of the GARCH filter.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec, fitting ugarchfit, forecasting ugarchforecast, simulation ugarchsim, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
## Not run:
data(sp500ret)
ctrl = list(RHO = 1,DELTA = 1e-8,MAJIT = 100,MINIT = 650,TOL = 1e-6)
spec = ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1,1)),
mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
distribution.model = "std")
egarch.fit = ugarchfit(data = sp500ret[,1,drop=FALSE], spec = spec,
solver = "solnp", solver.control = ctrl)

spec = ugarchspec(variance.model = list(model = "eGARCH", garchOrder = c(1,1)),
mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
distribution.model = "std", fixed.pars = as.list(coef(egarch.fit)))
egarch.filter = ugarchfilter(data = sp500ret[,1,drop=FALSE], spec = spec)

## End(Not run)
```

---

uGARCHfit-class *class: Univariate GARCH Fit Class*

Description

Class for the univariate GARCH fit.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class GARCHfit, directly. Class rGARCH, by class GARCHfit, distance 2.

Slots

`fit`: Object of class `"vector"` Holds data on the fitted model.

`model`: Object of class `"vector"` The model specification common to all objects.

Methods

as.data.frame `signature(x = "uGARCHfit")`: Extracts the position (dates), data, fitted values, residuals and conditional sigma.

coef `signature(object = "uGARCHfit")`: Rxtracts the coefficients.

infocriteria `signature(object = "uGARCHfit")`: Calculates and returns various information criteria.

nyblom `signature(object = "uGARCHfit")`: Calculates and returns the Hansen-Nyblom stability test (1990).

gof `signature(object = "uGARCHfit", groups = "numeric")`: Calculates and returns the adjusted goodness of fit statistic and p-values for the fitted distribution based on the Vlaar and Palm paper (1993). Groups is a numeric vector of bin sizes.

newsimpact `signature(object = "uGARCHfit")`: Calculates and returns the news impact curve.

signbias `signature(object = "uGARCHfit")`: Calculates and returns the sign bias test of Engle and Ng (1993).

likelihood `signature(object = "uGARCHfit")`: Extracts the likelihood.

sigma `signature(object = "uGARCHfit")`: Extracts the conditional sigma values.

fitted `signature(object = "uGARCHfit")`: Extracts the fitted values.

residuals `signature(object = "uGARCHfit")`: Extracts the residuals.

getspec `signature(object = "uGARCHfit")`: Extracts and returns the GARCH specification from a fit object.

uncvariance `signature(object = "uGARCHfit", pars = "missing", distribution="missing", model = "missing", vexdata = "missing")`: Calculates and returns the long run unconditional variance of the GARCH fit given a uGARCHfit object.

uncvariance `signature(object = "missing", pars = "numeric", distribution = "character", model = "character", submodel = "ANY", vexdata = "ANY")`: Calculates and returns the long run unconditional variance of the GARCH fit given a named parameter vector as returned by the fit, a distribution model name and a GARCH model name with a submodel included if the model is of the nested type such as fGARCH and any external regressor data.

uncmean `signature(object = "uGARCHfit")`: Calculates and returns the unconditional mean of the conditional mean equation (constant, ARMAX, arch-in-mean).

persistence `signature(object = "uGARCHfit", pars = "missing", distribution = "missing", model = "missing")`: Calculates and returns the persistence of the GARCH fit model given a `uGARCHfit` object.

persistence `signature(object = "missing", pars = "numeric", distribution = "character", model = "character")`: Calculates and returns the persistence of the GARCH fit model given a named parameter vector as returned by the fit, a distribution model name and a GARCH model name with a submodel included if the model is of the nested type such as fGARCH.

halflife `signature(object = "uGARCHfit", pars = "missing", distribution = "missing", model = "missing")`: Calculates and returns the halflife of the GARCH fit variance given a `uGARCHfit` object.

halflife `signature(object = "missing", pars = "numeric", distribution = "character", model = "character")`: Calculates and returns the halflife of the GARCH fit variance given a named parameter vector as returned by the fit, a distribution model name and a GARCH model name with a submodel included if the model is of the nested type such as fGARCH.

plot `signature(x = "uGARCHfit", y = "missing")`: Fit plots.

show `signature(object = "uGARCHfit")`: Fit summary.

Note

Methods for coef, likelihood, fitted, sigma and residuals provide extractor functions for those values.
Method for show gives detailed summary of GARCH fit with various tests.
Method for plot provides for interactive choice of plots, option of choosing a particular plot (option "which" equal to a valid plot number) or a grand plot including all subplots on one page (option "which"="all").
The data.frame method returns a data frame with 4 columns, the original data, the fitted data, the residuals and the sigma values, indexed (rownames) by the same values as provided in the original data provided to the fit function (e.g. dates).
The infocriteria method calculates and returns the information criteria (AIC, BIC etc) of the GARCH fit.
The nyblom method calculates and returns the Hansen-Nyblom joint and individual coefficient stability test statistic and critical values.
The gof methods calculates and returns the adjusted goodness of fit statistic and p-values for the fitted distribution. The groups parameter is a numeric vector of grouped bin sizes for the test. See the references in the package introduction for the original paper by Vlaar and Palm explaining the test.
The signbias methods calculates and returns the sign bias test of Engle and Ng (see the references in the package introduction).
Methods for calculating and extracting persistence, unconditional variance and half-life of the GARCH shocks exist and take either the GARCH fit object as a single value otherwise you may provide a named parameter vector (see `uGARCHspec` section for parameter names of the various GARCH models), a distribution name and the GARCH model (with submodel argument for the fGARCH model).
Unconditional mean and variance of the model may be extracted by means of the uncmean and uncvariance methods. The uncvariance may take either a fit object or a named parameter list, distribution and GARCH model name. The uncmean will only take a fit object due to the complexity of the calculation requiring much more information than the uncoditional variance.

The news impact method returns a list with the calculated values (zx, zy) and the expression (xexpr, yexpr) which can be used to illustrate the plot.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHforecast, uGARCHsim and uGARCHspec.

Examples

```
## Not run:
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
fit

# object fit:
slotNames(fit)
# sublist fit@fit
names(fit@fit)

coef(fit)
infocriteria(fit)
likelihood(fit)
nyblom(fit)
signbias(fit)
head(as.data.frame(fit))
head(sigma(fit))
head(residuals(fit))
head(fitted(fit))
gof(fit,c(20,30,40,50))
uncmean(fit)
uncvariance(fit)
#plot(fit,which="all")

# news impact example
spec = ugarchspec(variance.model=list(model="apARCH"))
fit = ugarchfit(data = dmbp[,1], spec = spec)
# note that newsimpact does not require the residuals (z) as it
# will discover the relevant range to plot against by using the min/max
# of the fitted residuals.
ni=newsimpact(z = NULL, fit)
#plot(ni$zx, ni$zy, ylab=ni$yexpr, xlab=ni$xexpr, type="l", main = "News Impact Curve")

## End(Not run)
```

ugarchfit-methods *function: Univariate GARCH Fitting*

Description

Method for fitting a variety of univariate GARCH models.

Usage

```
ugarchfit(spec, data, out.sample = 0, solver = "solnp", solver.control = list(),
fit.control = list(stationarity = 1, fixed.se = 0, scale = 0), ...)
```

Arguments

data	A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
spec	A univariate GARCH spec object of class uGARCHspec.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting (see details).
solver	One of either "nlminb", "solnp" or "gosolnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine. Stationarity explicitly imposes the variance stationarity constraint during optimization. The fixed.se argument controls whether standard errors should be calculated for those parameters which were fixed (through the fixed.pars argument of the ugarchspec function). The scale parameter controls whether the data should be scaled before being submitted to the optimizer.
...	.

Details

The GARCH optimization routine first calculates a set of feasible starting points which are used to initiate the GARCH recursion. The main part of the likelihood calculation is performed in C-code for speed.

The out.sample option is provided in order to carry out forecast performance testing against actual data. A minimum of 5 data points are required for these tests. If the out.sample option is positive, then the routine will fit only N - out.sample (where N is the total data length) data points, leaving out.sample points for forecasting and testing using the forecast performance measures. In the ugarchforecast routine the n.ahead may also be greater than the out.sample number resulting in a combination of out of sample data points matched against actual data and some without, which the forecast performance tests will ignore.

The "gosolnp" solver allows for the initialization of multiple restarts of the solnp solver with randomly generated parameters (see documentation in the Rsolnp-package for details of the strategy used). The solver.control list then accepts the following additional (to the solnp) arguments: "n.restarts" is the number of solver restarts required (defaults to 1), "parallel" and "parallel.control" for use of the parallel functionality, "rseed" is the seed to initialize the random number generator, and "n.sim" is the number of simulated parameter vectors to generate per n.restarts.

Value

A uGARCHfit object containing details of the GARCH fit.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec,filtering ugarchfilter, forecasting ugarchforecast, simulation ugarchsim, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
fit
coef(fit)
head(as.data.frame(fit))
#plot(fit,which="all")
# in order to use fpm (forecast performance measure function)
# you need to select a subsample of the data:
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec, out.sample=100)
forc = ugarchforecast(fit, n.ahead=100)
# this means that 100 data points are left from the end with which to
# make inference on the forecasts
fpm(forc)
```

---

uGARCHforecast-class *class: Univariate GARCH Forecast Class*

Description

Class for the univariate GARCH forecast.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class GARCHforecast, directly. Class rGARCH, by class GARCHforecast, distance 2.

Methods

as.array `signature(x = "uGARCHforecast")`: Extracts the forecast array with matrix column dimensions equal to the n.ahead value and row dimension 2 (sigma and series forecast), and array dimension equal to the number of rolling forecasts chosen.

as.data.frame `signature(x = "uGARCHforecast")`: Extracts the forecasts. Takes many additional arguments (see note below).

as.list `signature(x = "uGARCHforecast")`: Extracts the forecast list with all rollframes.

plot `signature(x = "uGARCHforecast", y = "missing")`: Forecast plots with `n.roll` optional argument indicating the rolling sequence to plot.

fpm `signature(object = "uGARCHforecast")`: Forecast performance measures.

show `signature(object = "uGARCHforecast")`: Forecast summary returning the 0-roll frame only.

Note

There are 3 main extractor functions for the uGARCHforecast object which is admittedly the most complex in the package as a result of allowing for rolling forecasts. The `as.array` extracts an array object where each page of the array represents a roll. The `as.list` method works similarly returns instead a list object. There are no additional arguments to these extractor functions and they will return all the forecasts. The `as.data.frame` method on the other hand provides for 5 additional arguments. The argument `which` indicates the type of forecast value to return(with valid valued being "sigma" and "series"). The `rollframe` option is for the rolling frame to return (with 0 being the default no-roll) and allows either a valid numeric value or alternatively the character value "all" for which additional options then come into play. When "all" is chosen in the `rollframe` argument, the data.frame returned may be time aligned (logical option `aligned`) in which case the logical option `prepad` indicates whether to pad the values prior to the forecast start time with actual values or NA (value `FALSE`). Finally, the `type` option controls whether to return all forecasts (value 0, default), return only those forecasts which have in sample equivalent data (value 1) or return only those values which are truly forecasts without in sample data (value 2). Depending on the intended usage of the forecasts, some or all these options may be useful to the user when extracting data from the forecast object.
The plot method takes additional arguments `which` and `n.roll` indicating which roll frame to plot.

Author(s)

Alexios Ghalanos

See Also

Classes `uGARCHfit`, `uGARCHsim` and `uGARCHspec`.

Examples

```
## Not run:
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
```

```
forc = ugarchforecast(fit, n.ahead=20)
forc
head(as.data.frame(forc))
#plot(forc, which = "all")

## End(Not run)
```

---

ugarchforecast-methods

function: Univariate GARCH Forecasting

Description

Method for forecasting from a variety of univariate GARCH models.

Usage

```
ugarchforecast(fitORspec, data = NULL, n.ahead = 10, n.roll = 0, out.sample = 0,
external.forecasts = list(mregfor = NULL, vregfor = NULL), ...)
```

Arguments

fitORspec	Either a univariate GARCH fit object of class uGARCHfit or alternatively a univariate GARCH specification object of class uGARCHspec with valid fixed parameters.
data	Required if a specification rather than a fit object is supplied.
n.ahead	The forecast horizon.
n.roll	The no. of rolling forecasts to create beyond the first one (see details).
out.sample	Optional. If a specification object is supplied, indicates how many data points to keep for out of sample testing.
external.forecasts	A list with forecasts for the external regressors in the mean and/or variance equations if specified.
...	.

Details

The forecast function has two dispatch methods allowing the user to call it with either a fitted object (in which case the data argument is ignored), or a specification object (in which case the data is required) with fixed parameters.
The forecast is based on the expected value of the innovations and hence the density chosen. One step ahead forecasts are based on the value of the previous data, while n-step ahead (n>1) are based on the unconditional expectation of the models.
The ability to roll the forecast 1 step at a time is implemented with the n.roll argument which controls how many times to roll the n.ahead forecast. The default argument of n.roll = 0 denotes no rolling and returns the standard n.ahead forecast. Critically, since n.roll depends on data being

available from which to base the rolling forecast, the ugarchfit function needs to be called with the argument out.sample being at least as large as the n.roll argument, or in the case of a specification being used instead of a fit object, the out.sample argument directly in the forecast function.

Value

A uGARCHforecast object containing details of the GARCH forecast. See the class for details on the returned object and methods for accessing it and performing some tests.

Author(s)

Alexios Ghalanos

See Also

For filtering ugarchfilter,simulation ugarchsim, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
## Not run:
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
forc = ugarchforecast(fit, n.ahead=20)
forc
head(as.data.frame(forc))
#plot(forc,which="all")

## End(Not run)
```

---

uGARCHmultifilter-class

class: Univariate GARCH Multiple Filter Class

Description

Class for the univariate GARCH Multiple filter.

Extends

Class "GARCHfilter", directly. Class "rGARCH", by class "GARCHfilter", distance 3.

Methods

fitted `signature(object = "uGARCHmultifilter")`: Extracts the fitted values.

residuals `signature(object = "uGARCHmultifilter")`: Extracts the residuals.

sigma `signature(object = "uGARCHmultifilter")`: Extracts the conditional sigma values.

coef `signature(object = "uGARCHmultifilter")`: Extracts the coefficients.

likelihood `signature(object = "uGARCHmultifilter")`: Extracts the likelihood.

show `signature(object = "uGARCHmultifilter")`: Filter summary.

Author(s)

Alexios Ghalanos

See Also

Classes `uGARCHmultiforecast`, `uGARCHmultifit` and `uGARCHmultispec`.

---

uGARCHmultifit-class *class: Univariate GARCH Multiple Fit Class*

Description

Class for the univariate GARCH Multiple fit.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class `GARCHfit`, directly. Class `rGARCH`, by class `GARCHfit`, distance 3.

Methods

coef `signature(object = "uGARCHmultifit")`: Extracts the coefficients.

likelihood `signature(object = "uGARCHmultifit")`: Extracts the likelihood.

sigma `signature(object = "uGARCHmultifit")`: Extracts the conditional sigma values.

fitted `signature(object = "uGARCHmultifit")`: Extracts the fitted values.

residuals `signature(object = "uGARCHmultifit")`: Extracts the residuals.

show `signature(object = "uGARCHmultifit")`: Fit summary.

Note

Methods for coef, likelihood, fitted, sigma and residuals provide extractor functions for those values.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHmultiforecast, uGARCHmultispec and uGARCHmultifilter.

---

uGARCHmultiforecast-class
class: Univariate GARCH Multiple Forecast Class

Description

Class for the univariate GARCH Multiple forecast.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class GARCHforecast, directly. Class rGARCH, by class GARCHforecast, distance 3.

Methods

as.array `signature(x = "uGARCHmultiforecast")`: extracts the forecast array with matrix column dimensions equal to the number of assets, row dimension the n.ahead and array dimension equal to the number of rolling forecasts chosen. The optional argument "which" allows to choose from "sigma" and "series" to return the forecasts for.

as.list `signature(x = "uGARCHforecast")`: extracts the forecast list of length equal to the number of assets, sublists equal to n.roll, row dimension of each sublist equal to n.ahead and column dimension equal to 2 (sigma and series forecasts).

show `signature(object = "uGARCHforecast")`: forecast summary.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHmultifilter, uGARCHmultifit and uGARCHmultispec.

uGARCHmultispec-class *class: Univariate GARCH Multiple Specification Class*

Description

Class for the univariate GARCH Multiple specification.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "GARCHspec", directly. Class "rGARCH", by class "GARCHspec", distance 3.

Methods

show `signature(object = "uGARCHmultispec")`: specification summary.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHmultiforecast, uGARCHmultifit and uGARCHmultifilter.

uGARCHpath-class *class: Univariate GARCH Path Simulation Class*

Description

Class for the univariate GARCH Path simulation.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "uGARCHpath", directly. Class "rGARCH", by class "GARCHpath", distance 2.

Methods

as.data.frame `signature(x = "uGARCHpath")`: extracts the simulated path values (see note).

plot `signature(x = "uGARCHpath", y = "missing")`: path simulation plots.

show `signature(object = "uGARCHpath")`: path simulation summary.

Note

The as.data.frame function takes optionally 1 additional arguments, namely which, indicating the type of simulation path series to extract. Valid values are "sigma" for the simulated sigma, "series" for the simulated series and "residuals" for the simulated residuals. The dimension of the data.frame will be n.sim by m.sim.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHsim, uGARCHfit and uGARCHspec.

---

ugarchpath-methods *function: Univariate GARCH Path Simulation*

Description

Method for simulating the path of a GARCH model from a variety of univariate GARCH models. This is a convenience function which does not require a fitted object (see note below).

Usage

```
ugarchpath(spec, n.sim=1000, n.start=0, m.sim=1, presigma=NA, prereturns=NA,
preresiduals=NA, rseed=NA, custom.dist=list(name=NA,distfit=NA), mexsimdata=NULL,
vexsimdata=NULL, ...)
```

Arguments

spec	A univariate GARCH spec object of class uGARCHspec with the required parameters of the model supplied via the fixed.pars list argument or setfixed<- method.
n.sim	The simulation horizon.
n.start	The burn-in sample.
m.sim	The number of simulations.
presigma	Allows the starting sigma values to be provided by the user.
prereturns	Allows the starting return data to be provided by the user.
preresiduals	Allows the starting residuals to be provided by the user.
rseed	Optional seeding value(s) for the random number generator. For m.sim>1, it is possible to provide either a single seed to initialize all values, or one seed per separate simulation (i.e. m.sim seeds). However, in the latter case this may result in some slight overhead depending on how large m.sim is.
custom.dist	Optional density with fitted object from which to simulate. See notes below for details.

mexsimdata	List of matrices (size of list m.sim, with each matrix having n.sim rows) of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed zero.
vexsimdata	List of matrices (size of list m.sim, with each matrix having n.sim rows) of simulated external regressor-in-variance data. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed zero.
...	.

Details

This is a convenience method to allow path simulation of various GARCH models without the need to supply a fit object as in the ugarchsim method. Instead, a GARCH spec object is required with the fixed model parameters.

Value

A uGARCHpath object containing details of the GARCH path simulation.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec, fitting ugarchfit, filtering ugarchfilter, forecasting ugarchforecast, simulation ugarchsim, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
## Not run:
# create a basic sGARCH(1,1) spec:
spec=ugarchspec(variance.model=list(model="sGARCH", garchOrder=c(1,1)),
mean.model=list(armaOrder=c(0,0), include.mean=TRUE, garchInMean =
FALSE, inMeanType = 2), distribution.model="sstd",
fixed.pars=list(mu=0.001,omega=0.00001, alpha1=0.05, beta1=0.90,
shape=4,skew=2))
# simulate the path
path.sgarch = ugarchpath(spec, n.sim=3000, n.start=1, m.sim=1)

## End(Not run)
```

uGARCHroll-class *class: Univariate GARCH Rolling Forecast Class*

Description

Class for the univariate GARCH rolling forecast.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "GARCHroll", directly. Class "rGARCH", by class "GARCHroll", distance 2.

Methods

as.data.frame `signature(x = "uGARCHroll")`: Extracts various values from object (see note).

plot `signature(x = "uGARCHroll", y = "missing")`: Roll result backtest plots (see note).

report `signature(object = "uGARCHroll")`: Roll backtest reports (see note).

fpm `signature(object = "uGARCHroll")`: Forecast performance measures.

as.uGARCHforecast `signature(object = "uGARCHroll")`: Extracts and converts the forecast object contained in the roll object to one of uGARCHforecast given the refit number supplied by additional argument 'refit' (defaults to 1).

Note

The `as.data.frame` extractor method allows the extraction of a variety of values from the object. Additional arguments are: `which` indicates the type of value to return. Valid values are "coefs" returning the parameter coefficients for all refits, "density" for the parametric density, "coefmat" for the parameter coefficients with their respective standard errors and t- and p- values, "LLH" for the likelihood across the refits, and "VaR" for the Value At Risk measure if it was requested in the roll function call.
`n.ahead` for the n.ahead forecast horizon to return if `which` was used with arguments "density" or "VaR".
`refit` indicates which refit window to return the "coefmat" if that was chosen.
The plot method takes the following additional arguments:
which allows for either a numeric value of 1:4, else will default to "ask" for interactive printing of the options in the command windows. Additionally, the value of "all" wil create a 2x2 chart with all plots.
n.ahead for the rolling n.ahead forecasts (defaults to 1).
VaR.alpha for the Value at Risk backtest plot, this is the tail probability and defaults to 0.01.
density.support the support for the time varying density plot density, defaults to c(-0.15, 0.15) but you should change this to something more appropriate for your data and period under consideration.
The report method takes the following additional arguments:

type for the report type. Valid values are "VaR" for the Value at Risk report based on the unconditional and conditional coverage tests for VaR exceedances (discussed below) and "fpm" for forecast performance measures.
n.ahead for the rolling n.ahead forecasts (defaults to 1).
VaR.alpha for the Value at Risk backtest report, this is the tail probability and defaults to 0.01.
conf.level the confidence level upon which the conditional coverage hypothesis test will be based on (defaults to 0.95).
Kupiec's unconditional coverage test looks at whether the amount of expected versus actual exceedances given the tail probability of VaR actually occur as predicted, while the conditional coverage test of Christoffersen is a joint test of the unconditional coverage and the independence of the exceedances. Both the joint and the separate unconditional test are reported since it is always possible that the joint test passes while failing either the independence or unconditional coverage test.

Author(s)

Alexios Ghalanos

`ugarchroll-methods` *function: Univariate GARCH Rolling Density Forecast and Backtesting*

Description

Method for creating rolling density forecast from ARMA-GARCH models with option for refitting every n periods and some multicore parallel functionality.

Usage

```
ugarchroll(spec, data, n.ahead = 1, forecast.length = 500, refit.every = 25,
refit.window = c("recursive", "moving"), parallel = FALSE,
parallel.control = list(pkg = c("multicore", "snowfall"), cores = 2), solver = "solnp",
fit.control = list(), solver.control = list(), calculate.VaR = TRUE,
VaR.alpha = c(0.01, 0.05), ...)
```

Arguments

`spec`	A univariate GARCH spec object specifiying the desired model for testing.
`data`	A univariate dataset.
`n.ahead`	The number of periods to forecast.
`forecast.length`	The length of the total forecast for which out of sample data from the dataset will be excluded for testing.
`refit.every`	Determines every how many periods the model is re-estimated.
`refit.window`	Whether the refit is done on an expanding window including all the previous data or a moving window, the length of the window determined by the argument above ("refit.every").

parallel	Whether to make use of parallel processing on multicore systems.
parallel.control	The parallel control options including the type of package for performing the parallel calculations ('multicore' for non-windows O/S and 'snowfall' for all O/S), and the number of cores to make use of.
solver	The solver to use.
fit.control	Control parameters parameters passed to the fitting function.
solver.control	Control parameters passed to the solver.
calculate.VaR	Whether to calculate forecast Value at Risk during the estimation.
VaR.alpha	The Value at Risk tail level to calculate.
...	.

Details

GARCH models generate a partially time varying density based on the variation in the conditional sigma and mean values (skewness and shape are usually not time varying in GARCH models unless the underlying distribution has an interaction with the conditional sigma). The function first generates rolling forecasts of the ARMA-GARCH model and then rescales the density from a standardized (0, 1, skew, shape) to the one representing the underlying return process (mu, sigma, skew, shape). Given this information it is then a simple matter to generate any measure of risk through the analytical evaluation of some type of function of the time varying density. The function calculates one such measure (VaR), but since the full time varying density parameters are returned, the user can calculate many others (see for example partial moments based measures or the Pedersen-Satchell family of measures).
The argument `refit.every` determines every how many periods the fit is recalculated and the total forecast length actually calculated. For example, for a forecast length of 500 and `refit.every` of 25, this is 20 windows of 25 periods each for a total actual forecast length of 500. However, for a `refit.every` of 30, we take the floor of the division of 500 by 30 which is 16 windows of 30 periods each for a total actual forecast length of 480 (16 x 30). The important thing to remember about the refit.every is that it acts like the n.roll argument in the `ugarchforecast` function as it determines the number of rolls to perform. For example for n.ahead of 1 and refit.every of 25, the forecast is rolled every day using the filtered (actual) data of the previous period while for n.ahead of 1 and refit.every of 1 we will get 1 n.ahead forecasts for every day after which the model is refitted and reforecast for a total of 500 refits (when length.forecast is 500)!
The function has 2 main methods for viewing the data, a standard plot method and a new report methods (see class `uGARCHroll` for details on how to use these methods).

Value

An object of class `uGARCHroll`.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec, fitting ugarchfit, filtering ugarchfilter, forecasting ugarchforecast, simulation ugarchsim, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
## Not run:
data(sp500ret)
ctrl = list(rho = 1, delta = 1e-9, outer.iter = 100, tol = 1e-7)
spec = ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
mean.model = list(armaOrder = c(0,0), include.mean = TRUE),
distribution.model = "std")

sp500.bktest = ugarchroll(spec, data = sp500ret, n.ahead = 1,
forecast.length = 100, refit.every = 25, refit.window = "recursive",
solver = "solnp", fit.control = list(), solver.control = ctrl,
calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05))
report(sp500.bktest, type="VaR", n.ahead = 1, VaR.alpha = 0.01,
conf.level = 0.95)
report(sp500.bktest, type="fpm")

## End(Not run)
```

---

uGARCHsim-class *class: Univariate GARCH Simulation Class*

Description

Class for the univariate GARCH simulation.

Extends

Class "GARCHsim", directly. Class "rGARCH", by class "GARCHsim", distance 2.

Slots

`simulation`: Object of class "vector" Holds data on the simulation.

`model`: Object of class "vector" The model specification common to all objects.

`seed`: Object of class "integer" The random seed used.

Methods

as.data.frame `signature(x = "uGARCHsim")`: Extracts the simulated values (see note).

plot `signature(x = "uGARCHsim", y = "missing")`: Simulation plots.

show `signature(object = "uGARCHsim")`: Simulation summary.

Note

The as.data.frame function takes optionally 1 additional arguments, namely which, indicating the type of simulation series to extract. Valid values are "sigma" for the simulated sigma, "series" for the simulated series and "residuals" for the simulated residuals. The dimension of the data.frame will be n.sim by m.sim.

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHforecast, uGARCHfit and uGARCHspec.

Examples

```
## Not run:
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
sim = ugarchsim(fit,n.sim=1000, n.start=1, m.sim=1, startMethod="sample")
sim
# plot(sim, which="all")
# as.data.frame takes an extra argument which
# indicating one of "sigma", "series" and "residuals"
head(as.data.frame(sim, which = "sigma"))

## End(Not run)
```

---

ugarchsim-methods *function: Univariate GARCH Simulation*

Description

Method for simulation from a variety of univariate GARCH models.

Usage

```
ugarchsim(fit, n.sim = 1000, n.start = 0, m.sim = 1,
startMethod = c("unconditional", "sample"), presigma = NA, prereturns = NA,
preresiduals = NA, rseed = NA, custom.dist = list(name = NA, distfit = NA),
mexsimdata = NULL, vexsimdata = NULL, ...)
```

Arguments

fit	A univariate GARCH fit object of class uGARCHfit.
n.sim	The simulation horizon.
n.start	The burn-in sample.
m.sim	The number of simulations.
startMethod	Starting values for the simulation. Valid methods are "unconditional" for the expected values given the density, and "sample" for the ending values of the actual data from the fit object.
presigma	Allows the starting sigma values to be provided by the user.
prereturns	Allows the starting return data to be provided by the user.
preresiduals	Allows the starting residuals to be provided by the user.
rseed	Optional seeding value(s) for the random number generator. For m.sim>1, it is possible to provide either a single seed to initialize all values, or one seed per separate simulation (i.e. m.sim seeds). However, in the latter case this may result in some slight overhead depending on how large m.sim is.
custom.dist	Optional density with fitted object from which to simulate. See notes below for details.
mexsimdata	List of matrices (size of list m.sim, with each matrix having n.sim rows) of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed zero.
vexsimdata	List of matrices (size of list m.sim, with each matrix having n.sim rows) of simulated external regressor-in-variance data. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed zero.
...	.

Details

The custom.dist option allows for defining a custom density which exists in the users workspace with methods for "r" (sampling, e.g. rnorm) and "d" (density e.g. dnorm). It must take a single fit object as its second argument. Alternatively, custom.dist can take any name in the name slot (e.g."sample") and a matrix in the fit slot with dimensions equal to m.sim (columns) and n.sim (rows). The usefulness of this becomes apparent when one is considering the copula-GARCH approach or the bootstrap method.

Value

A uGARCHsim object containing details of the GARCH simulation.

Author(s)

Alexios Ghalanos

See Also

For specification ugarchspec, fitting ugarchfit, filtering ugarchfilter, forecasting ugarchforecast, rolling forecast and estimation ugarchroll, parameter distribution and uncertainty ugarchdistribution, bootstrap forecast ugarchboot.

Examples

```
## Not run:
# Basic GARCH(1,1) Spec
data(dmbp)
spec = ugarchspec()
fit = ugarchfit(data = dmbp[,1], spec = spec)
sim = ugarchsim(fit,n.sim=1000, n.start=1, m.sim=1, startMethod="sample")
sim
# plot(sim, which="all")
# as.data.frame takes an extra argument which
# indicating one of "sigma", "series" and "residuals"
head(as.data.frame(sim, which = "sigma"))

## End(Not run)
```

---

uGARCHspec-class *class: Univariate GARCH Specification Class*

Description

Class for the univariate GARCH specification.

Extends

Class "GARCHspec", directly. Class "rGARCH", by class "GARCHspec", distance 2.

Slots

model: Object of class "vector" The model specification common to all objects.

Methods

show `signature(object = "uGARCHspec")`: Specification summary.

setfixed<- `signature(object = "uGARCHspec", value = "vector")`: Sets the fixed parameters (which must be supplied as a named list).

setstart<- `signature(object = "uGARCHspec", value = "vector")`: Sets the starting parameters (which must be supplied as a named list).

uncmean `signature(object = "uGARCHspec")`: Unconditional mean of model for a specification with fixed.pars list.

uncvariance `signature(object = "uGARCHspec")`: Unconditional variance of model for a specification with fixed.pars list.

uncvariance `signature(object = "uGARCHspec", pars = "missing", distribution = "missing", model = "missing", submodel = "missing", vexdata = "missing")`: Calculates and returns the long run unconditional variance of the GARCH fit given a `uGARCHfit` object.

halflife `signature(object = "uGARCHspec", pars = "missing", distribution = "missing", model = "missing")`: Calculates and returns the halflife of the GARCH fit variance given a `uGARCHspec` object with fixed parameters.

persistence `signature(object = "uGARCHfit", pars = "missing", distribution = "missing", model = "missing")`: Calculates and returns the persistence of the GARCH fit model given a `uGARCHspec` object with fixed parameters.

Author(s)

Alexios Ghalanos

See Also

Classes `uGARCHfit`, `uGARCHsim` and `uGARCHforecast`.

Examples

```
# Basic GARCH(1,1) Spec
spec = ugarchspec()
spec
```

`ugarchspec-methods` *function: Univariate GARCH Specification*

Description

Method for creating a univariate GARCH specification object prior to fitting.

Usage

```
ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1),
submodel = NULL, external.regressors = NULL, variance.targeting = FALSE),
mean.model = list(armaOrder = c(1, 1), include.mean = TRUE, archm = FALSE,
archpow = 1, arfima = FALSE, external.regressors = NULL),
distribution.model = "norm", start.pars = list(), fixed.pars = list(), ...)
```

Arguments

`variance.model` List containing the variance model specification:
`model` Valid models (currently implemented) are "sGARCH", "fGARCH", "eGARCH", "gjrGARCH", "apARCH" and "iGARCH".
`garchOrder` The ARCH (q) and GARCH (p) orders.
`submodel` If the model is "fGARCH", valid submodels are "GARCH", "TGARCH",

"AVGARCH", "NGARCH", "NAGARCH", "APARCH","GJRGARCH" and "ALLGARCH".
`external.regressors` A matrix object containing the external regressors to include in the variance equation with as many rows as will be included in the data (which is passed in the fit function). `variance.targeting` Indicates whether to use variance targeting for the sigma intercept "omega".).

`mean.model` List containing the mean model specification:
`armaOrder` The autoregressive (ar) and moving average (ma) orders (if any).
`include.mean` Whether to include the mean.
`archm` Whether to include ARCH volatility in the mean regression.
`archpow` Indicates whether to use st.deviation (1) or variance (2) in the ARCH in mean regression.
`arfima` Whether to fractional differencing in the ARMA regression.
`external.regressors` A matrix object containing the external regressors to include in the mean equation with as many rows as will be included in the data (which is passed in the fit function).

`distribution.model`
The conditional density to use for the innovations. Valid choices are "norm" for the normal distibution, "snorm" for the skew-normal distribution, "std" for the student-t, "sstd" for the skew-student, "ged" for the generalized error distribution, "sged" for the skew-generalized error distribution, "nig" for the normal inverse gaussian distribution, "ghyp" for the Generalized Hyperbolic, and "jsu" for Johnson's SU distribution. Note that some of the distributions are taken from the fBasics package and implenented locally here for convenience. The "jsu" distribution is the reparametrized version from the "gamlss" package.

`start.pars` List of staring parameters for the optimization routine. These are not usually required unless the optimization has problems converging.

`fixed.pars` List of parameters which are to be kept fixed during the optimization. It is possible that you designate all parameters as fixed so as to quickly recover just the results of some previous work or published work. The optional argument "fixed.se" in the `ugarchfit` function indicates whether to calculate standard errors for those parameters fixed during the post optimization stage.

`...` .

Details

The specification allows for a wide choice in univariate GARCH models, distributions, and mean equation modelling. For the "fGARCH" model, this represents Hentschel's omnibus model which subsumes many others.
For the mean equation, ARFIMAX is fully supported in fitting, forecasting and simulation.
The "iGARCH" implements the integrated GARCH model. For the "EWMA" model just set "omega" to zero in the fixed parameters list.
The asymmetry term in the rugarch package, for all implemented models, follows the order of the arch parameter `alpha`.
Variance targeting, referred to in Engle and Mezrich (1996), replaces the intercept "omega" in the variance equation by 1 minus the persistence multiplied by the unconditional variance which is calculated by its sample counterpart in the squared residuals during estimation. In the presence of

external regressors in the variance equation, the sample average of the external regresssors is multiplied by their coefficient and subtracted from the variance target.
In order to understand which parameters can be entered in the start.pars and fixed.pars optional arguments, the list below exposes the names used for the parameters across the various models:(note that when a parameter is followed by a number, this represents the order of the model. Just increment the number for higher orders):
Mean Model:

constant	`mu`
AR term	`ar1`
MA term	`ma1`
ARCH in mean	`archm`
exogenous regressors	`mxreg1`
arfima	`arfima`

Distribution Model:

ghlambda	`lambda (for GHYP distribution)`
skew	`skew`
shape	`shape`

Variance Model (common specs):

constant	`omega`
ARCH term	`alpha1`
GARCH term	`beta1`
exogenous regressors	`vxreg1`

Variance Model (GJR, EGARCH):

assymetry term	`gamma1`

Variance Model (APARCH):

assymetry term	`gamma1`
power term	`delta`

Variance Model (FGARCH):

assymetry term1 (rotation)	`eta11`
assymetry term2 (shift)	`eta21`
power term1(shock)	`delta`
power term2(variance)	`lambda`

Value

A `uGARCHspec` object containing details of the GARCH specification.

Author(s)

Alexios Ghalanos

Examples

```
# a standard specification
spec1 = ugarchspec()
spec1
# an example which keep the ar1 and ma1 coefficients fixed:
spec2 = ugarchspec(mean.model=list(armaOrder=c(2,2),
fixed.pars=list(ar1=0.3,ma1=0.3)))
spec2
# an example of the EWMA Model
spec3 = ugarchspec(variance.model=list(model="iGARCH", garchOrder=c(1,1)),
mean.model=list(armaOrder=c(0,0), include.mean=TRUE),
distribution.model="norm", fixed.pars=list(omega=0))
```

`WeekDayDummy-methods` *function: Create Dummy Day-of-Week Variable*

Description

Helper function to create a dummy, day of the week variable given a set of dates.

Usage

```
WeekDayDummy(Dates, date.format, weekday = "Monday")
```

Arguments

`Dates`	A character vector of dates .
`date.format`	The format of the dates e.g. "%Y-%m-%d"" .
`weekday`	Character string indicating day of week.

Value

A numeric vector of 0s and 1s (date-dummy variable)/

Author(s)

Alexios Ghalanos

Examples

```
data(sp500ret)
Dates=rownames(sp500ret)
# create Monday dummy
monday=WeekDayDummy(Dates, date.format="%Y-%m-%d", weekday = "Monday")
```

Index