Package 'rugarch'

September 22, 2024

Type Package

Title Univariate GARCH Models

Version 1.5-3

Date 2024-09-21

Maintainer Alexios Galanos <alexios@4dscape.com>

Depends R (>= 3.5.0), methods, parallel

LinkingTo Rcpp (>= 0.10.6), RcppArmadillo (>= 0.2.34)

Imports Rsolnp, ks, numDeriv, spd, xts, zoo, chron, SkewHyperbolic, Rcpp, graphics, fracdiff, stats, grDevices, utils, nloptr

Suggests knitr, rmarkdown

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

Collate rugarch-imports.R rugarch-cwrappers.R rugarch-solvers.R rugarch-lossfn.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-figarch.R rugarch-csgarch.R rugarch-fgarch.R rugarch-egarch.R rugarch-gjrgarch.R rugarch-aparch.R rugarch-igarch.R rugarch-mcsgarch.R rugarch-realgarch.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 rugarch-cv.R zzz.R

LazyLoad yes

URL http://www.unstarched.net, https://github.com/alexiosg/rugarch

License GPL-3

NeedsCompilation yes

Author Alexios Galanos [aut, cre], Tobias Kley [ctb]

Repository CRAN

Date/Publication 2024-09-22 05:00:02 UTC

Contents

`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 feature rich distributions are used for modelling innovations and documented in the vignette.
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, both of which are now hosted on CRAN (stable) and bitbucket (development).

Details

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 now completely based on the xts package, and only data which can be coerced to such will be accepted by the package. For the estimation and filter routines, some of the main extractors methods will now also return xts objects.
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.
Since version 1.0-14, all parallel estimation is carried out through a user-supplied cluster object,

created from the parallel package, meaning that the user is now in control of managing the cluster lifecycle. This greatly simplifies the parallel estimation process and adds a layer of flexibility to the type of resources supported.
Finally, the global extractor functions `sigma` and `fitted` will now work with almost all returned classes and the return the conditional sigma and mean values, whether these are from an estimated, filtered, forecast, or simulated object (and their multi- function equivalents).

How to cite this package

Whenever using this package, please cite as

```
@Manual{Ghalanos_2014,
 author       = {Alexios Ghalanos},
 title        = {{rugarch}: Univariate GARCH models.},
 year         = {2014},
 note         = {R package version 1.4-0.},}
```

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.
Engle, R. F., and Sokalska, M. E. 2012, Forecasting intraday volatility in the US equity market. Multiplicative component GARCH. *Journal of Financial Econometrics*, **10(1)**, 54–83.
Fisher, T. J., and Gallagher, C. M. 2012, New weighted portmanteau statistics for time series goodness of fit testing, *Journal of the American Statistical Association*, **107(498)**, 777–787.
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

`arfimacv` *ARFIMAX time series cross validation*

Description

Implements a cross validation method for ARFIMAX models

Usage

```
arfimacv(data, indexin, indexout, ar.max = 2, ma.max = 2,
criterion = c("rmse","mae","berkowitzp"),berkowitz.significance = 0.05,
arfima = FALSE, include.mean = NULL, distribution.model = "norm",
cluster = NULL, external.regressors = NULL, solver = "solnp",
solver.control=list(), fit.control=list(), return.best=TRUE)
```

Arguments

data	A univariate xts vector.
indexin	A list of the training set indices
indexout	A list of the testing set indices, the same list length as that of indexin. This should be a numeric index of points immediately after those in the equivalent indexin slot and contiguous (for time series cross validation).
ar.max	Maximum AR order to test for.
ma.max	Maximum MA order to test for.
criterion	The cv criterion on which the forecasts will be tested against the realized values. Currently "rmse", "mae" and experimentally "berkowitzp" are implemented. The latter is the Berkowitz test p-value (maximized) and should not be used if your indexout set is very small.
berkowitz.significance	The significance level at which the Berkowitz test is evaluated at (this has no value at the moment since we are only looking at the p-values, but may be used in futures to instead aggregate across pass-fail).
arfima	Can be TRUE, FALSE or NULL in which case it is tested.
include.mean	Can be TRUE, FALSE or NULL in which case it is tested.
cluster	A cluster object created by calling `makeCluster` from the parallel package. If it is not NULL, then this will be used for parallel estimation.
external.regressors	An xts matrix object containing the pre-lagged external regressors to include in the mean equation with the same indices as those of the data supplied.
distribution.model	The distribution density to use for the innovations (defaults to Normal).
solver	One of either "nlminb", "solnp", "gosolnp" or "nloptr".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine.
return.best	On completion of the cross-validation, should the best model be re-estimated on the complete dataset and returned (defaults to TRUE).

Details

The function evaluates all possible combinations of the ARFIMAX model for all the training and testing sets supplied. For the ARMA orders, the orders are evaluated fully (e.g. for ar.max=2, all possible combinations are evaluated including AR(0,0), AR(0,1), AR(0,2), AR(1,0), AR(2,0) AR(1,2), AR(2,1), and AR(2,2)). For each training set in indexin, all model combinations are evaluated and the 1-ahead rolling forecast for the indexout testing set is produced and compared to the realized values under the 3 criteria listed. Once all training/testing is done on all model combinations, the criteria are averaged across all the sets for each combination and the results returned.

Value

A list with the following items:

bestmodel	The best model based on the criterion chosen is re-estimated on the complete data set and returned.
cv_matrix	The model combinations and their average criteria statistics across the training/testing sets.

Note

Use a cluster...this is an expensive computation, particularly for large ar.max and ma.max orders. The indexin and indexout lists are left to the user to decide how to implement.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
require(xts)
require(parallel)
data(sp500ret)
spx = as.xts(sp500ret)
nn = nrow(spx)
nx = nn-round(0.9*nn,0)
if(nx
h = (nx/50)-1
indexin = lapply(1:h, function(j){ tail(seq(1,(nn-nx)+j*50, by=1),250) })
indexout = lapply(indexin, function(x){ (tail(x,1)+1):(tail(x,1)+50) })
cl = makePSOCKcluster(5)
mod = arfimacv(spx, indexin, indexout, ar.max = 2, ma.max = 2,
criterion = c("rmse","mae","berkowitzp")[1],
berkowitz.significance = 0.05, arfima = FALSE, include.mean = NULL,
distribution.model = "norm", cluster = cl, external.regressors = NULL,
solver = "solnp")
stopCluster(cl)

## End(Not run)
```

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(),
cluster = 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.
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).
cluster	A cluster object created by calling makeCluster from the parallel package. If it is not NULL, then this will be used for parallel estimation.
...	.

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

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

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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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

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

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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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.

vcov `signature(object = "ARFIMAfit")`: Extracts the covariance matrix of the parameters. Additional logical option of 'robust' indicates whether to extract the robust based covariance matrix.

convergence `signature(object = "ARFIMAfit")`: Returns the solver convergence code for the fitted object (zero denotes convergence).

reduce `signature(object = "ARFIMAfit")`: Zeros parameters (fixing to zero in rugarch is equivalent to eliminating them in estimation) with p-values (optional argument "pvalue") greater than 0.1 (default), and re-estimates the model. Additional arguments are passed to `arfimafit`.An additional option "use.robust" (default TRUE) asks whether to use the robust calculated p-values.

getspec `signature(object = "ARFIMAfit")`: Extracts and returns the ARFIMA specification from a fitted object.

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), numderiv.control = list(grad.eps=1e-4,
grad.d=0.0001, grad.zero.tol=sqrt(.Machine$double.eps/7e-7), hess.eps=1e-4, hess.d=0.1,
hess.zero.tol=sqrt(.Machine$double.eps/7e-7), r=4, v=2), ...)
```

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", "gosolnp" or "nloptr".
`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`.
`numderiv.control`	Control arguments passed to the numerical routines for the calculation of the standard errors. See the documentation in the numDeriv package for further details. The arguments which start with 'hess' are passed to the hessian routine while those with 'grad' to the jacobian routine.
`...`	.

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" (logical), "pkg" (either snowfall or multicore) and "cores" (the number of cores or workers to use) for use of 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

`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

fitted `signature(x = "ARFIMAforecast")`: The n.ahead by n.roll+1 matrix of conditional mean forecasts. The column names are the T[0] dates.

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

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

Note

Since versions 1.01-3, the `fitted` method has been introduced which extracts the n.ahead by (n.roll+1) matrix of conditional mean forecasts, with column names the T[0] time index. This is unlike the old `data.frame` which returned the T+1 etc dates. This method is the default extractor in rugarch package for the conditional mean (whether from an estimated, filtered, forecast or simulated object) and the other method, namely `as.data.frame` is now deprecated with the exception of a few classes where it is still used (`ARFIMAdistribution` and `ARFIMAroll`).
The fpm method returns the Mean Squared Error (MSE), Mean Absolute Error (MAE), Directional Accuracy (DAC) and number of points used for the calculation (N), of forecast versus realized returns, if the extra `summary` option is set to TRUE (default). This is a 4 x (n.roll+1) matrix, with row headings the T[0] time index, and requires at least 5 points to calculate the summary measures else will return NA. When n.ahead>1, this method calculates the measures on the n.ahead>1 unconditional forecast, but if n.ahead=1 with n.roll>4, it will calculate the measures on the rolling forecast instead. Finally, when summary is set to FALSE, the method will return a list of length n.roll+1 of xts objects with the loss functions (Squared Error and Absolute Error and Directional Hits).

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

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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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

fitted `signature(x = "ARFIMAmultiforecast")`: Extracts the conditional mean forecast from the object, and returns an array of the n.ahead by (n.roll+1) by n.assets.

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

fitted `signature(x = "ARFIMApath")`: Extracts the simulated path values as a matrix of dimension n.sim by m.sim.

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

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

`forecast`: 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 = "ARFIMAroll")`: extracts various values from object (see note).

resume `signature(object = "ARFIMAroll")`: Resumes a rolling backtest which has non-converged windows using alternative solver and control parameters.

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

coef `signature(object = "ARFIMAroll")`: Extracts the list of coefficients for each estimated window in the rolling backtest.

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

show `signature(object = "ARFIMAroll")`: Summary.

Note

The `as.data.frame` extractor method allows the extraction of either the conditional forecast density or the VaR. It takes additional argument `which` with valid values either "density" or "VaR".
The `coef` method will return a list of the coefficients and their robust standard errors (assuming the keep.coef argument was set to TRUE in the ugarchroll function), and the ending date of each estimation window.
The `report` method takes the following additional arguments:
1.*type* for the report type. Valid values are "VaR" for the VaR report based on the unconditional and conditional coverage tests for exceedances (discussed below) and "fpm" for forecast performance measures.
2.*VaR.alpha* (for the VaR backtest report) is the tail probability and defaults to 0.01.
3.*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 method (separately from report) takes additional logical argument *summary*, which when TRUE will return the mean squared error (MSE), mean absolute error (MAE) and directional accuracy of the forecast versus realized returns. When FALSE, it will return a data.frame of the time series of squared (SE) errors, absolute errors (AE), directional hits (HITS), and a VaR Loss function described in Gonzalez-Rivera, Lee, and Mishra (2004) for each coverage level where it was calculated. This can then be compared, with the VaR loss of competing models using such tests as the model confidence set (MCS) of Hansen, Lunde and Nason (2011).

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 with parallel functionality.

Usage

```
arfimaroll(spec, data, n.ahead = 1, forecast.length = 500,
n.start = NULL, refit.every = 25, refit.window = c("recursive", "moving"),
window.size = NULL, solver = "hybrid", fit.control = list(),
solver.control = list(), calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.05),
cluster = NULL, keep.coef = TRUE, ...)
```

Arguments

spec	A univariate ARFIMA specification object.
data	A univariate dataset, ideally with time based index.
n.ahead	The number of periods to forecast (only n.ahead=1 supported).
forecast.length	The length of the total forecast for which out of sample data from the dataset will be used for testing.
n.start	Instead of forecast.length, this determines the starting point in the dataset from which to initialize the rolling forecast.
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 where all previous data is used for the first estimation and then moved by a length equal to refit.every (unless the window.size option is used instead).
window.size	If not NULL, determines the size of the moving window in the rolling estimation, which also determines the first point used.

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.
cluster	A cluster object created by calling makeCluster from the parallel package. If it is not NULL, then this will be used for parallel estimation of the refits (remember to stop the cluster on completion).
keep.coef	Whether to return the list of coefficients and their robust standard errors.
...	.

Details

This is a wrapper function for creating rolling forecasts of the conditional ARFIMA density, and optionally calculating the Value at Risk at specified levels. The argument refit.every determines every how many periods the model is re-estimated. Given a dataset of length N, it is possible to choose either how many periods from the end to use for out of sample forecasting (using the forecast.length option), or the starting point for initializing the rolling forecast (and using all the data after that for the out of sample forecast). Only rolling 1-ahead forecasts are supported spanning the dataset, which should be useful for backtesting models. Anything more complicated should be wrapped by the user by making use of the underlying functions in the package. The function has 2 main methods for viewing the data, a standard plot method and a report methods (see class ARFIMAroll for details on how to use these methods). In case of no-convergence in some of all the windows, a new method called resume now allows to pass the returned (non-converged) object with new solver and control parameters to be re-estimated (only the non-converged windows are re-estimated). Parallel functionality is now based entirely on the parallel package, and it is up to the user to pass a cluster object, and then stop it once the routine is completed.

Value

An object of class ARFIMAroll.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
spec = arfimaspec(distribution.model = "std")
mod = arfimaroll(spec, data = sp500ret, n.ahead = 1,
n.start = 1000,  refit.every = 100, refit.window = "moving",
solver = "hybrid", fit.control = list(),
calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05),
keep.coef = TRUE)
report(sp500.bktest, type="VaR", VaR.alpha = 0.01, conf.level = 0.95)
report(sp500.bktest, type="fpm")
```

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

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

fitted `signature(x = "ARFIMAsim")`: extracts the simulated values as a matrix of dimension n.sim by m.sim.

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

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).

setbounds<- `signature(object = "ARFIMAspec", value = "vector")`: Sets the parameters lower and upper bounds, which must be supplied as a named list with each parameter being a numeric vector of length 2 i.e. "ar1"=c(-1,1)). If the vector is of length 1, then this is assumed to be the lower bound, and the upper bound will be set to its default value prior to estimation.

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 (0<d<0.5).
`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	`arfima`

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

`autoarfima` *Automatic Model Selection for ARFIMA models*

Description

Select best fitting ARFIMA models based on information criteria.

Usage

```
autoarfima(data, ar.max = 2, ma.max = 2, criterion = c("AIC","BIC","SIC","HQIC"),
method = c("partial", "full"), arfima = FALSE, include.mean = NULL,
distribution.model = "norm", cluster = NULL, external.regressors = NULL,
solver = "solnp", solver.control=list(), fit.control=list(), return.all = FALSE)
```

Arguments

`data`	A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
`ar.max`	Maximum AR order to test for.
`ma.max`	Maximum MA order to test for.
`criterion`	Information Criterion to use for selecting the best model.
`method`	The partial method tests combinations of consecutive orders of AR and MA i.e. 1:2, 1:3 etc, while the full method tests all possible combinations within the consecutive orders thus enumerating the complete combination space of the MA and AR orders. .
`arfima`	Can be TRUE, FALSE or NULL in which case it is tested.
`include.mean`	Can be TRUE, FALSE or NULL in which case it is tested.
`cluster`	A cluster object created by calling `makeCluster` from the parallel package. If it is not NULL, then this will be used for parallel estimation.
`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 (defaults to Normal).
`solver`	One of either “nlminb”, “solnp”, “gosolnp” or “nloptr”.
`solver.control`	Control arguments list passed to optimizer.
`fit.control`	Control arguments passed to the fitting routine.
`return.all`	Whether to return all the fitted models or only the best one.

Value

A list with the following items:

fit	Either the best fitted model or all the fitted models if the option 'return.all' was selected.
rank.matrix	Either a sorted matrix of the models and their information criterion, else an unsorted matrix of the models and their information criterion if the option 'return.all' was selected.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
fit = autoarfima(data = sp500ret[1:1000,], ar.max = 2, ma.max = 2,
criterion = "AIC", method = "full")

## End(Not run)
```

BerkowitzTest *Berkowitz Density Forecast Likelihood Ratio Test*

Description

Implements the Berkowitz Density Forecast Likelihood Ratio Test.

Usage

```
BerkowitzTest(data, lags = 1, significance = 0.05, tail.test = FALSE, alpha = 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.
tail.test	Whether to use the tail test of Berkowitz using a censored likelihood.
alpha	The quantile level for the tail.test cuttoff.

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 (not calculated when tail.test is used).
JB	The Jarque-Bera Test of Normality Statistic (not calculated when tail.test is used).
JBp	The Jarque-Beta Test Statistic p-value (not calculated when tail.test is used).

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). When tail.test is used, the test of the tail at the "alpha" quantile level is performed using a censored normal likelihood.

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. 1987, 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.numeric(fitted(pred)),as.numeric(sigma(pred)),
rep(coef(fit)["skew"],1000), rep(coef(fit)["shape"],1000))
colnames(dmatrix) = c("mu", "sigma", "skew", "shape")
# Get Realized (Oberved) Data
obsx = tail(dji30ret[,1], 1000)
# 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)
test1 = BerkowitzTest(data = nvector, lags = 1, significance = 0.05)
test2 = BerkowitzTest(data = nvector, alpha = 0.05, significance = 0.05,
tail.test=TRUE)
test3 = BerkowitzTest(data = nvector, alpha = 0.01, significance = 0.05,
tail.test=TRUE)

## 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.

Examples

```
## Not run:
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)
# Get Realized (Oberved) Data
obsx = tail(dji30ret[,1], 1000)
forc = as.numeric(as.data.frame(pred,rollframe="all",align=FALSE,which="series"))
print(DACTest(forc, obsx, test = "PT", conf.level = 0.95))
print(DACTest(forc, obsx, test = "AG", conf.level = 0.95))

## End(Not run)
```

DateTimeUtilities *A small set of utilities to work with some time and date classes.*

Description

These utilities will likely be useful for working with the forecast objects of the package which have a rather complex structure. In addition, the ftseq function is of particular value in generating intraday regularly spaced time and date sequences within a specific interval of times (e.g. 09:30 to 16:00).

Usage

```
move(index, by=1)
generatefwd(T0, length.out = 1, by = "days")
ftseq(T0, length.out, by, interval, exclude.weekends = TRUE)
```

Arguments

index	A POSIXct, Date or numeric vector.
T0	A single POSIXct, Date or numeric value from which to generate forward values (the returned vector will exclude this value). For the ftseq function, this must be a Date AND Time object of class POSIXct.
by	For the move function, the length by which to shift the index forward, truncating the first values and extending the last by this amount. For the generatefwd or ftseq function, either a character (see Date and Time classes for valid values), numeric or difftime object (see details).
length.out	The length of the forward generated indices (excluding T0 which is not returned).
interval	A character vector of the regularly sampled times which define the trading day (see example).
exclude.weekends	Whether to exclude the weeekends.

Details

Every object returned by one of the main methods in rugarch (including ugarchfit, ugarchfilter, ugarchforecast and ugarchsim) has a model slot attached which in turns hold details on the time index of the original dataset used (including a difftime object). In addition, extractors for the forecast class, uGARCHforecast, will usually return a matrix with the (n.roll+1) columns having the T+0 dates, and the rows names represented as characters 'T+1,...,T+n' indicating the forecast periods following the T+0 date.

For the rolling forecast, it is a simple matter to shift the T+0 date by 1 to obtain the actual forecast date. Because rolling forecasts are made using the 'out.sample' switch, this means that there is always an actual date attached to this forecast based on the realized out.sample data (with the exception of the case when n.roll=out.sample in which case the last forecast is completely out of the range of the dataset). One quick way of obtaining the actual T+1 rolling dates is to just pass the

vector of T+0 dates to the move function as shown in the examples.
For the n.ahead>1 unconditional forecasts, there may or may not be actual dates in the dataset covering the period, depending on whether out.sample was used, n.roll was also used, and how these all come together to form a complex object of moving and unconditional forecasts (making this the most complex of forecast cases). One way to quickly generate a sequence of dates is to use the generatefwd function with the T+0 starting date, the 'length' as the n.ahead horizon and the 'by' the difftime object from the model slot, as shown in the examples.
Note that for both the move and generatefwd functions, weekends are excluded in order to try to return a more realistic value.
Finally, when working with Date/Time objects remember to set your time zone with `Sys.setenv(TZ=)`.

Value

A vector of Date/Time/Numeric indices of the same class as used in the input.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(sp500ret)
spec = ugarchspec()
fit = ugarchfit(spec, sp500ret, out.sample=10)
forc = ugarchforecast(fit, n.ahead = 25, n.roll = 10)
f = fitted(forc)
# this is a 25 x 11 matrix [n.ahead x (n.roll+1)]
# colnames: T+0 date index
T0 = as.POSIXct(colnames(f))
rollT1 = move(T0, by=1)
# rolling estimation
plot(xts(f["T+1",],rollT1))
# unconditional estimates:
par(mfrow=c(3,4))
for(i in 1:11){
# difftime is always in model$modeldata$period
D=generatefwd(T0[i], length.out = 25, by = forc@model$modeldata$period)
plot(xts(f[,i], D), main=paste("T+0:",as.character(T0[i]),sep=""), auto.grid=FALSE)
}
############################
## Intraday Sequency Example
############################
T0 = as.POSIXct("2001-01-01 16:00:00")
# remember to remove the backslash from the code below
interval = format(seq(as.POSIXct("2001-01-01 09:30:00"), as.POSIXct("2001-01-01 16:00:00"),
by="min"), "%H:%M:%S")
by = "mins"
length.out=1000
R = ftseq(T0, length.out, by, interval)

## End(Not run)
```

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.

References

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

ESTest *Expected Shortfall Test.*

Description

Implements the Expected Shortfall Test of McNeil and Frey.

Usage

```
ESTest(alpha = 0.05, actual, ES, VaR, conf.level = 0.95,
boot = FALSE, n.boot = 1000)
```

Arguments

alpha	The quantile (coverage) used for the VaR.
actual	A numeric vector of the actual (realized) values.
ES	The numeric vector of the Expected Shortfall (ES).
VaR	The numeric vector of VaR.
conf.level	The confidence level at which the Null Hypothesis is evaluated.
boot	Whether to bootstrap the test.
n.boot	Number of bootstrap replications to use.

Details

The Null hypothesis is that the excess conditional shortfall (excess of the actual series when VaR is violated), is i.i.d. and has zero mean. The test is a one sided t-test against the alternative that the excess shortfall has mean greater than zero and thus that the conditional shortfall is systematically underestimated. Using the bootstrap to obtain the p-value should alleviate any bias with respect to assumptions about the underlying distribution of the excess shortfall.

Value

A list with the following items:

expected.exceed	The expected number of exceedances (length actual x coverage).
actual.exceed	The actual number of exceedances.
H1	The Alternative Hypothesis of the one sided test (see details).
boot.p.value	The bootstrapped p-value (if used).
p.value	The p-value.
Decision	The one-sided test Decision on H0 given the confidence level and p-value (not the bootstrapped).

Author(s)

Alexios Ghalanos

References

McNeil, A.J. and Frey, R. and Embrechts, P. (2000), Estimation of tail-related risk measures for heteroscedastic financial time series: an extreme value approach, *Journal of Empirical Finance*,**7**, 271–300.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "sstd")
fit = ugarchfit(spec, data = dji30ret[1:1000, 1, drop = FALSE])
spec2 = spec
setfixed(spec2)<-as.list(coef(fit))
filt = ugarchfilter(spec2, dji30ret[1001:2500, 1, drop = FALSE], n.old = 1000)
actual = dji30ret[1001:2500,1]
# location+scale invariance allows to use [mu + sigma*q(p,0,1,skew,shape)]
VaR = fitted(filt) + sigma(filt)*qdist("sstd", p=0.05, mu = 0, sigma = 1,
skew  = coef(fit)["skew"], shape=coef(fit)["shape"])
# calculate ES
f = function(x) qdist("sstd", p=x, mu = 0, sigma = 1,
skew  = coef(fit)["skew"], shape=coef(fit)["shape"])
ES = fitted(filt) + sigma(filt)*integrate(f, 0, 0.05)$value/0.05
print(ESTest(0.05, actual, ES, VaR, boot = TRUE))

## 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.

`GMMTest` *The GMM Orthogonality Test of Hansen*

Description

Implements the GMM Orthogonality Test of Hansen.

Usage

```
GMMTest(z, lags = 1, skew=0, kurt=3, conf.level = 0.95)
```

Arguments

`z`	A numeric vector the standardized residuals.
`lags`	The number of lags to test for.
`skew`	The skewness of the standardized residuals (derived from the estimated model). This can be either a scalar or numeric vector the same size as z.
`kurt`	The kurtosis (not excess) of the standardized residuals (derived from the estimated model). This can be either a scalar or numeric vector the same size as z.
`conf.level`	The confidence level at which the Null Hypothesis is evaluated.

Details

This is a mispecification test based on Hansen's GMM procedure. Under a correctly specified model, certain population moment conditions should be satisfied and hold in the sample using the standardized residuals. The moment conditions can be tested both individually using a t-test or jointly using a Wald test (the vignette gives more details). The test returns a matrix (moment.mat) containing the first 4 moments statistics, their standard errors and t-values (2-sided t-test with alternative hypothesis that the value is not equal to zero). The matrix of joint conditions (joint.mat) contains the t-values and critical values of 'Q2', 'Q3' and 'Q4' representing the autocorrelation, given the chosen lags in the second, third and fourth moments and distributed as chi-squared with n.lag d.o.f, and the joint test ('J') for all moment conditions distributed chi-squared with 4+(n.lagx3) d.o.f.

Value

A list with the following items:

joint.mat	The matrix of the joint tests.
moment.mat	The matrix of the individual moment tests.
H0	The Null Hypothesis.
Decision	Whether to reject or not the Null given the conf.level.

Author(s)

Alexios Ghalanos

References

Hansen, L. (1982), Large Sample Properties of Generalized Method of Moments Estimators, *Econometrica*, **50(4)**, 1029–1054.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "sstd")
fit = ugarchfit(spec, data = dji30ret[, 1, drop = FALSE])
z = residuals(fit)\/sigma(fit)
skew = dskewness("sstd",skew = coef(fit)["skew"], shape= coef(fit)["shape"])
# add back 3 since dkurtosis returns the excess kurtosis
kurt = 3+dkurtosis("sstd",skew = coef(fit)["skew"], shape= coef(fit)["shape"])
print(GMMTest(z, lags = 1, skew=skew, kurt=kurt))

## End(Not run)
```

HLTest *The Non-Parametric Density Test of Hong and Li*

Description

Implements the Non-Parametric Density Test of Hong and Li.

Usage

```
HLTest(PIT, lags = 4, kernel = "quartic", conf.level = 0.95)
```

Arguments

PIT	This represents the actual data transformed into a U(0,1) series by applying the distribution function of the estimated model conditional on the parameters.
lags	The number of lags to use for testing the joint hypothesis.
kernel	The kernel to use for the comparison against the PIT series (only the 'quartic' currently implemented).
conf.level	The confidence level at which the Null Hypothesis is evaluated.

Details

A novel method to analyze how well a conditional density fits the underlying data is through the probability integral transformation (PIT) discussed in Rosenblatt (1952) and used in the BerkowitzTest. More recently, Hong and Li (2005) introduced a nonparametric portmanteau test, building on the work of Ait-Sahalia (1996), which tests the joint hypothesis of i.i.d and uniformity for a series of PIT transformed data. To achieve this, it tests for misspecification in the conditional moments of the model transformed standardized residuals, and is distributed as N(0, 1) under the Null of a correctly specified model. These moment tests are reported as 'M(1,1)' to 'M(4,4)' in the output, with 'M(1,2)' related to ARCH-in-mean effects, and 'M(2,1)' to leverage, while 'W' is the Portmanteu type test statistic for general misspecification (using p lags) and also distributed as N(0, 1) under the Null of a correctly specified model. Only upper tail critical values are used in this test. The interested reader is referred to the paper for more details.

Value

A list with the following items:

statistic	The individual moment and joint test statistics.
Decision	Whether to reject or not the Null given the conf.level.

Author(s)

Alexios Ghalanos

References

Ait-Sahalia, Y. (1996), Testing continuous-time models of the spot interest rate, *Review of Financial Studies*, **9(2)**, 385–426.
Berkowitz, J. (2001), Testing density forecasts, with applications to risk management, *Journal of Business and Economic Statistics*, **19(4)**, 465–474.
Hong, Y., and Li, H. (2005), Nonparametric specification testing for continuous-time models with applications to term structure of interest rates, *Review of Financial Studies*, **18(1)**, 37–84.
Rosenblatt, M. (1952), Remarks on a multivariate transformation, *The Annals of Mathematical Statistics*, **23(3)**, 470–472.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "sstd")
fit = ugarchfit(spec, data = dji30ret[, 1, drop = FALSE])
z = residuals(fit)/sigma(fit)
PIT = pdist("sstd",z, mu = 0, sigma = 1, skew = coef(fit)["skew"],
shape=coef(fit)["shape"])
print(HLTest(PIT, lags=4))

## End(Not run)
```

`mcsTest` *Model Confidence Set Test*

Description

Implements the Model Confidence Set Test procedure of Hansen, Lunde and

Usage

```
mcsTest(losses, alpha, nboot = 100, nblock = 1, boot = c("stationary", "block"))
```

Arguments

`losses`	A matrix of losses from competing models.
`alpha`	The p-value used in the test.
`nboot`	The number of bootstrap replications.
`nblock`	The block length to use in the bootstrap.
`boot`	A choice of either the stationary or block boostrap.

Details

Calculates and returns the results of both the R (range) and SQ (semi-quadratic) statistics.

Value

A list with the following items:

`includedR`	The models included based on the R statistic.
`pvalsR`	The final p-values of each model under the R statistic.
`excludedR`	The excluded models based on the R statistic.
`includedSQ`	The models included based on the SQ statistic.
`pvalsSQ`	The final p-values of each model under the SQ statistic.
`excludedSQ`	The excluded models based on the SQ statistic.

Author(s)

Alexios Ghalanos

References

Hansen, P. R., Lunde, A., and Nason, J. M., 2011. The model confidence set. *Econometrica*, **79(2)**, 453–497.

`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,
rec.init = "all", cluster = NULL, ...)
```

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).
`rec.init`	Recursion initialization method (as in `ugarchfit` function), valid only for GARCH models, and can be a vector of length equal to the number of assets being modelled.
`cluster`	A cluster object created by calling `makeCluster` from the parallel package. If it is not NULL, then this will be used for parallel estimation.
`...`	.

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, rec.init = "all"),
cluster = NULL, ...)
```

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 of class xts or coercible to such.
`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, while the rec.init option controls the recursion initialization method and only valid for GARCH models.
`cluster`	A cluster object created by calling `makeCluster` from the parallel package. If it is not NULL, then this will be used for parallel estimation.
`...`	.

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),
cluster = NULL, ...)
```

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.
`cluster`	A cluster object created by calling `makeCluster` from the parallel package. If it is not NULL, then this will be used for parallel estimation of the refits (remember to stop the cluster on completion).
`...`	.

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 ) )
```

qnig *Functions exported for use in rmgarch*

Description

Quantile for NIG and GH distributions in alpha, beta, delta, mu and lambda parametrizations.

Usage

```
qnig(p, alpha = 1, beta = 0, delta = 1, mu = 0)
qgh(p, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)
```

Arguments

p	Probabilities
alpha	parameter in alpha, beta, delta, mu, lambda parametrization of the gh distribution.
beta	parameter in alpha, beta, delta, mu, lambda parametrization of the gh distribution.
delta	parameter in alpha, beta, delta, mu, lambda parametrization of the gh distribution.
mu	parameter in alpha, beta, delta, mu, lambda parametrization of the gh distribution.
lambda	parameter in alpha, beta, delta, mu, lambda parametrization of the gh distribution.

Details

For use internally by rmgarch.

Value

A vector

Author(s)

Alexios Galanos

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)
distplot(distribution = "snorm", skewbounds = NULL, shapebounds = NULL,
n.points = NULL)
skdomain(distribution = "nig", kurt.max = 30, n.points = 25, lambda = 1,
plot = TRUE, legend = NULL)
```

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.
skewbounds	The skewed distribution skew bounds for the plot. Leaving it NULL will use a good set of defaults for display purposes.
shapebounds	The shaped distribution shape bounds for the plot. Leaving it NULL will use a good set of defaults for display purposes.
n.points	The number of points between the lower and upper bounds of the skew and shape parameters for which to evaluate the skewness and excess kurtosis. For the skdomain function this determines the kurtosis interval (3-max.kurt) for which to determine (using a solver) the maximum skewness.
kurt.max	The maximum kurtosis for which to determine the bounds for the skewness-kurtosis domain.
plot	Whether to plot the results.
legend	Whether to include a legend with the plot in the skdomain.

Details

For the "nig" 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. The distplot provides illustrative plots (or surfaces) of skewness and kurtosis for any of the distributions supported (with the exception of the GH which has 2 shape and 1 skew parameters and hence is impractical to represent).

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. skdomain returns the authorized domain 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 SP500 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

`spyreal` *data: SPDR Standard and Poors 500 Open-Close Daily Return and Realized Kernel Volatility*

Description

The SPDR SP500 index open-close return and the realized kernel volatility for the period 2002-01-02 to 2008-08-29 from the paper of Hansen, Huang and Shek (2011). Used for illustrating the implementation of the Realized GARCH model in rugarch.

Usage

```
data(spyreal)
```

Format

An xts object.

Source

Journal of Applied Econometrics Data Archive

References

Hansen, P. R., Huang, Z., and Shek, H. H. (2012). Realized GARCH: a joint model for returns and realized measures of volatility. *Journal of Applied Econometrics*, **27(6)**, 877–906.

`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

```
## Not run:
ugarchbench( benchmark = "published" )

## End(Not run)
```

---

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"),
sampling = c("raw", "kernel", "spd"), spd.options = list(upper = 0.9,
lower = 0.1, type = "pwm", kernel = "normal"), 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), mexsimdata = NULL,
vexsimdata = NULL, cluster = NULL, verbose = FALSE)
```

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).
sampling	Whether to sample from the raw residuals, the kernel-fitted distribution of the residuals or the spd-fitted distribution of the residuals.
spd.options	If sampling is from the SPD distribution, this controls the options for fitting this distribution to the residuals (see spd package for details).
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.
mexsimdata	List of matrices (size of list n.bootpred, with each matrix having n.ahead 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 n.bootpred, with each matrix having n.ahead 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.

cluster	A cluster object created by calling makeCluster from the parallel package. If it is not NULL, then this will be used for parallel estimation of the refits (remember to stop the cluster on completion).
verbose	Whether to print out progress messages.

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 (2006), 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 choice of parallel computation via a cluster (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.
If using external regressors, the routine requires both the forecast (of length n.ahead as in the ugarchforecast routine) and a list of simulated forecasts as in the ugarchsim routine (else with be assumed zero). Finally, it is possible to resample based on 3 schemes, namely the "raw" innovations as in the original paper of Pascual et al (2006), "kernel" fits a Gaussian kernel to the innovations from the ks package in order to then generate random samples, and the "spd" fits a semi-parametric distribution to the innovations based on the spd package in order to generate the random samples, for which an optional list (spd.options) may be further passed to the spd fitting routine.

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(), cluster = NULL, ...)
```

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).
cluster	A cluster object created by calling makeCluster from the parallel package. If it is not NULL, then this will be used for parallel estimation of the refits (remember to stop the cluster on completion).
...	.

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

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

residuals `signature(object = "uGARCHfilter")`: Extracts the residuals. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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.

quantile `signature(x = "uGARCHfilter")`: Calculates and returns, given a vector of probabilities (additional argument "probs"), the conditional quantiles of the filtered object (x).

pit `signature(object = "uGARCHfilter")`: Calculates and returns the conditional probability integral transform given the data and estimated density.

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, rec.init = 'all',
trunclag = 1000, ...)
```

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).
`rec.init`	The recursion initialization method (see `ugarchfit` for explanation).
`trunclag`	The truncation lags for the binomial expansion in the FIGARCH model.
`...`	For the multiplicative component sGARCH model (mcsGARCH), the additional argument 'DailyVar' is required and should be an xts object of the daily forecasted variance to use with the intraday data.

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

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

cofint `signature(object = "uGARCHfit")`: Similar to the stats S3 method `confint`, extracts coefficient confidence intervals taking additional optional arguments `parm` and `level`, as well as `robust` (default: FALSE) indicating whether to use the robust covariance matrix for the calculations.

vcov `signature(object = "uGARCHfit")`: Extracts the covariance matrix of the parameters. Additional logical option of 'robust' indicates whether to extract the robust based covariance matrix.

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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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.

convergence `signature(object = "uGARCHfit")`: Returns the solver convergence code for the fitted object (zero denotes convergence).

quantile `signature(x = "uGARCHfit")`: Calculates and returns, given a vector of probabilities (additional argument "probs"), the conditional quantiles of the fitted object (x).

pit `signature(object = "uGARCHfit")`: Calculates and returns the conditional probability integral transform given the data and estimated density.

reduce `signature(object = "uGARCHfit")`: Zeros parameters (fixing to zero in rugarch is equivalent to eliminating them in estimation) with p-values (optional argument "pvalue") greater than 0.1 (default), and re-estimates the model. Additional arguments are passed to `ugarchfit`.An additional option "use.robust" (default TRUE) asks whether to use the robust calculated p-values.

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 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(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, rec.init = 'all',
trunclag = 1000),
numderiv.control = list(grad.eps=1e-4, grad.d=0.0001,
grad.zero.tol=sqrt(.Machine$double.eps/7e-7), hess.eps=1e-4, hess.d=0.1,
hess.zero.tol=sqrt(.Machine$double.eps/7e-7), r=4, v=2),...)
```

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", "lbfgs", "gosolnp", "nloptr" or "hybrid" (see notes).
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. For the FIGARCH model this imposes the positivity constraint. 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. The rec.init option determines the type of initialization for the variance recursion. Valid options are 'all' which uses all the values for the unconditional variance calculation, an integer greater than or equal to 1 denoting the number of data points to use for the calculation, or a positive numeric value less than one which determines the weighting for use in an exponential smoothing backcast. The trunclag is the truncation lags for the binomial expansion in the FIGARCH model.
numderiv.control	Control arguments passed to the numerical routines for the calculation of the standard errors. See the documentation in the numDeriv package for further details. The arguments which start with 'hess' are passed to the hessian routine while those with 'grad' to the jacobian routine.
...	For the multiplicative component sGARCH model (mcsGARCH), the additional argument 'DailyVar' is required and should be an xts object of the daily forecasted variance to use with the intraday data.

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" (logical), "pkg" (either snowfall or multicore) and "cores" (the number of cores or workers to use) for use of 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.
The "hybrid" strategy solver first tries the "solnp" solver, in failing to converge then tries then "nlminb", the "gosolnp" and finally the "nloptr" solvers. Solver control parameters can be passed for all the solvers in the solver.control list as one long list which will be filtered for each solver's specific options as and when that solver is called during the hybrid strategy optimization. It is still possible that the Hessian at the optimal found cannot be inverted, in which case a warning is printed and there will not be any standard errors. In this case it is suggested that the problem is re-run with different solver parameters. It is also possible that the solution, while still 'almost' optimal may be at a saddle-point very near the global optimum in which case the Hessian may still be invertible but one eigenvalue is negative. The uGARCHfit object has a value in the fit slot called condH (object@fit$condH) which indicates the approximate number of decimal places lost to roundoff/numerical estimation error. When this is NaN, this indicates the case just described of one negative eigenvalue/saddlepoint (this previously flagged a warning but is now silenced and it is upto to the user to decide whether it is worth investigating further).

Value

A uGARCHfit object containing details of the GARCH fit.

Note

The nloptr solver takes the following options in the solver.control list:

ftol_rel	function value relative tolerance	default: 1e-8
xtol_rel	parameter value relative tolerance	default: 1e-6
maxeval	maximum function evaluations	default: 25000
print_level	trace level	default: 1
solver	the nloptr solver to use	default: 1 ('SBPLX').

The solver option for nloptr has 10 different choices (1:10), which are 1:'COBYLA', 2:'BOBYQA', 3:'PRAXIS', 4:'NELDERMEAD', 5:'SBPLX', 6:'AUGLAG'+'COBYLA', 7:'AUGLAG'+'BOBYQA', 8:'AUGLAG'+'PRAXIS', 9:'AUGLAG'+'NELDERMEAD' and 10:'AUGLAG'+'SBPLX'. As always, your mileage will vary and care should be taken on the choice of solver, tuning parameters etc. If you do use this solver try 9 or 10 first.

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(sigma(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

sigma `signature(x = "uGARCHforecast")`: The n.ahead by n.roll+1 matrix of conditional sigma forecasts. The column names are the T[0] dates.

fitted `signature(x = "uGARCHforecast")`: The n.ahead by n.roll+1 matrix of conditional mean forecasts. The column names are the T[0] dates.

quantile `signature(x = "uGARCHforecast")`: Calculates and returns, given a scalar for the probability (additional argument "probs"), the conditional quantile of the forecast object as an n.ahead by n.roll+1 matrix (with the same type of headings as the sigma and fitted methods).

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

Since versions 1.01-3, a `sigma` and `fitted` methods have been introduced which extract the n.ahead by (n.roll+1) matrix of conditional sigma and mean forecasts respectively, with column names the T[0] time index. This is unlike the old `data.frame` which returned the T+1 etc dates. These two methods are the default extractors in rugarch (used on estimated, filtered, forecast and simulation class objects) and the other methods, namely `as.data.frame` is now deprecated with the exception of a few classes where it is still used (`uGARCHdistribution`, `uGARCHboot` and `uGARCHroll`).
The plot method takes additional arguments `which` and `n.roll` indicating which roll frame to plot. The fpm method returns the Mean Squared Error (MSE), Mean Absolute Error (MAE), Directional Accuracy (DAC) and number of points used for the calculation (N), of forecast versus realized returns, if the extra `summary` option is set to TRUE (default). This is a 4 x (n.roll+1) matrix, with row headings the T[0] time index, and requires at least 5 points to calculate the summary measures else will return NA. When n.ahead>1, this method calculates the measures on the n.ahead>1 unconditional forecast, but if n.ahead=1 with n.roll>4, it will calculate the measures on the rolling forecast instead. Finally, when summary is set to FALSE, the method will return a list of length n.roll+1 of xts objects with the loss functions (Squared Error and Absolute Error and Directional Hits).

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, out.sample = 100)
forc1 = ugarchforecast(fit, n.ahead=100, n.roll = 100)
```

```
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),
trunclag = 1000, ...)
```

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.
trunclag	The truncation lag for the binomial expansion in the FIGARCH model. Only used when the dispatch is based on a uGARCHspec object, otherwise will be read from the already defined value in the fitted object.
...	For the multiplicative component sGARCH model (mcsGARCH), the additional argument 'DailyVar' is required and should be an xts object of the daily forecasted variance for the period under consideration to be used with the intraday data. For the realized GARCH model (realGARCH), the additional argument 'RealizedVol', an xts object, is required when using a specification object for fitORspec. Additionally, the optional argument 'n.sim' denotes the number of simulations required for n.ahead>1 forecast (see vignette for this model's representation), whilst 'returnDistribution' is a logical argument (default TRUE) denoting whether to return the simulated distribution of the sigma and realized forecast values.

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(sigma(forc))
head(fitted(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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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. Optional logical argument `standardize` (default is FALSE) allows to extract the standardized 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

sigma `signature(x = "uGARCHmultiforecast")`: extracts the n.ahead by (n.roll+1) by n.assets array of conditional sigma forecasts.

fitted `signature(x = "uGARCHforecast")`: extracts the n.ahead by (n.roll+1) by n.assets array of conditional mean 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

sigma `signature(object = "uGARCHpath")`: Extracts the conditional sigma simulated values as a matrix of size n.sim x m.sim.

fitted `signature(object = "uGARCHpath")`: Extracts the conditional mean simulated values as a matrix of size n.sim x m.sim.

quantile `signature(x = "uGARCHpath")`: Calculates and returns, given a scalar for the probability (additional argument "probs"), the conditional quantile of the simulated object as an n.sim by m.sim matrix (with the same type of headings as the sigma and fitted methods).

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

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

Note

The `sigma` and `fitted` methods are used to extract the matrix of simulated conditional sigma and mean values. The `as.data.frame` method is globally deprecated as an extractor method in rugarch with the exception of a few classes which still makes sense to use them.

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, trunclag=1000, ...)
```

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. It is now recommended not to provide a value (i.e., keep the default of rseed == NA) and to call set.seed only once in the beginning of the session, which will ensure reproducibility.
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 variance equation, this must be provided else will be assumed zero.
trunclag	This is the truncation lags for the binomial expansion in the FIGARCH model

... If the model is the "csGARCH", then preq can be used to denote the previous value of the permanent component of the variance model (q, e.g. tail(fit@fit$q,1)) so that the ugarchpath method with all pre-values included will evaluate to the same result as the ugarchsim method with method equal to "sample" (assuming the same random seeding values are used).

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. The mcsGARCH model is not supported for the path method-use ugarchsim instead.

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.

Slots

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

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

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).

resume `signature(object = "uGARCHroll")`: Resumes a rolling backtest which has non-converged windows using alternative solver and control parameters.

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

coef `signature(object = "uGARCHroll")`: Extracts the list of coefficients for each estimated window in the rolling backtest.

quantile `signature(x = "uGARCHroll")`: Calculates and returns, given a vector of probabilities (additional argument "probs"), the conditional quantiles of the rolling object as an xts matrix.

pit `signature(object = "uGARCHroll")`: Calculates and returns the conditional probability integral transform given the realized data and forecast density.

convergence `signature(object = "uGARCHroll")`: Returns the convergence code for the estimation windows, with 0 indicating that all have converged and 1 that there were non-converged windows. In the latter case the 'nonconverged' attribute is also printed of those windows which failed to converge.

show `signature(object = "uGARCHroll")`: Summary.

Note

The `as.data.frame` extractor method allows the extraction of either the conditional forecast density or the VaR. It takes additional argument `which` with valid values either "density" or "VaR".
The `coef` method will return a list of the coefficients and their robust standard errors (assuming the keep.coef argument was set to TRUE in the ugarchroll function), and the ending date of each estimation window.
The `plot` method takes the following additional arguments:
1.*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.
2.*VaR.alpha* for the Value at Risk backtest plot, this is the tail probability and defaults to 0.01.
3.*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:
1.*type* for the report type. Valid values are "VaR" for the VaR report based on the unconditional and conditional coverage tests for exceedances (discussed below) and "fpm" for forecast performance

measures.
2.*VaR.alpha* (for the VaR backtest report) is the tail probability and defaults to 0.01.
3.*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` method (separately from report) takes additional logical argument *summary*, which when TRUE will return the mean squared error (MSE), mean absolute error (MAE) and directional accuracy of the forecast versus realized returns. When FALSE, it will return a data.frame of the time series of squared (SE) errors, absolute errors (AE), directional hits (HITS), and a VaR Loss function described in Gonzalez-Rivera, Lee, and Mishra (2004) for each coverage level where it was calculated. This can then be compared, with the VaR loss of competing models using such tests as the model confidence set (MCS) of Hansen, Lunde and Nason (2011).

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 with parallel functionality.

Usage

```
ugarchroll(spec, data, n.ahead = 1, forecast.length = 500,
n.start = NULL, refit.every = 25, refit.window = c("recursive", "moving"),
window.size = NULL, solver = "hybrid", fit.control = list(),
solver.control = list(), calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.05),
cluster = NULL, keep.coef = TRUE, ...)
```

Arguments

`spec`	A univariate GARCH specification object.
`data`	A univariate dataset, ideally with time based index.
`n.ahead`	The number of periods to forecast (only n.ahead=1 supported).
`forecast.length`	The length of the total forecast for which out of sample data from the dataset will be used for testing.

n.start	Instead of forecast.length, this determines the starting point in the dataset from which to initialize the rolling forecast.
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 where all previous data is used for the first estimation and then moved by a length equal to refit.every (unless the window.size option is used instead).
window.size	If not NULL, determines the size of the moving window in the rolling estimation, which also determines the first point used.
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.
cluster	A cluster object created by calling makeCluster from the parallel package. If it is not NULL, then this will be used for parallel estimation of the refits (remember to stop the cluster on completion).
keep.coef	Whether to return the list of coefficients and their robust standard errors.
...	In the case of the realized GARCH (realGARCH) model, the 'realizedVol' is required (an xts object), and optionally the 'n.sim' argument indicates the samples to generate for the realized vol forecast (does not affect the 1-ahead sigma forecast).

Details

This is a wrapper function for creating rolling forecasts of the conditional GARCH density, and optionally calculating the Value at Risk at specified levels. The argument refit.every determines every how many periods the model is re-estimated. Given a dataset of length N, it is possible to choose either how many periods from the end to use for out of sample forecasting (using the forecast.length option), or the starting point for initializing the rolling forecast (and using all the data after that for the out of sample forecast). Only rolling 1-ahead forecasts are supported spanning the dataset, which should be useful for backtesting models. Anything more complicated should be wrapped by the user by making use of the underlying functions in the package. The function has 2 main methods for viewing the data, a standard plot method and a report methods (see class uGARCHroll for details on how to use these methods). In case of no-convergence in some of all the windows, a new method called resume now allows to pass the returned (non-converged) object with new solver and control parameters to be re-estimated (only the non-converged windows are re-estimated). Non-convergence here implies both a failure of the solver to converge to a solution (global failure) OR a failure to invert the resulting Hessian (local failure). The convergence method can be used on an object (aside from the printed warning) to print out the number of the non-converged estimation windows.
Parallel functionality is now based entirely on the parallel package, and it is up to the user to pass a cluster object, and then stop it once the routine is completed.

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)
spec = ugarchspec(distribution.model = "std")
mod = ugarchroll(spec, data = sp500ret, n.ahead = 1,
n.start = 1000,  refit.every = 500, refit.window = "recursive",
solver = "hybrid", fit.control = list(),
calculate.VaR = TRUE, VaR.alpha = c(0.01, 0.025, 0.05),
keep.coef = TRUE)
report(mod, type="VaR", VaR.alpha = 0.01, conf.level = 0.95)
report(mod, 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

sigma signature(object = "uGARCHsim"): Extracts the conditional sigma simulated values as a matrix of size n.sim x m.sim.

fitted signature(object = "uGARCHsim"): Extracts the conditional mean simulated values as a matrix of size n.sim x m.sim.

quantile `signature(object = "uGARCHsim", probs="numeric")`: Calculates and returns, given a scalar for the probability (additional argument "probs"), the conditional quantile of the simulated object as an n.sim by m.sim matrix (with the same type of headings as the sigma and fitted methods).

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

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

Note

The `sigma` and `fitted` methods are used to extract the matrix of simulated conditional sigma and mean values. The `as.data.frame` method is globally deprecated as an extractor method in rugarch with the exception of a few classes which still makes sense to use them.

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
head(sigma(sim))

## 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. It is now recommended not to provide a value (i.e., keep the default of rseed == NA) and to call set.seed only once in the beginning of the session, which will ensure reproducibility.
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.
...	For the multiplicative component sGARCH model (mcsGARCH), the additional argument 'DailyVar' is required and should be an xts object of length floor(n.sim/increments-per-day) by m.sim of the the daily simulated variance to use with the intraday data. In the case of the realized GARCH (realGARCH) model, the optional argument 'prerealized' allows to pass starting values of the realized volatility (should be of length q as was set in the 'garchOrder(q,p)' in the specification)

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). It is understood that what is supplied are the standardized (0,1) innovations and not the unstandardized residuals. 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
head(sigma(sim))

## 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).

setbounds<- `signature(object = "uGARCHspec", value = "vector")`: Sets the parameters lower and upper bounds, which must be supplied as a named list with each parameter being a numeric vector of length 2 i.e. "alpha1"=c(0,1)). If the vector is of length 1, then this is assumed to be the lower bound, and the upper bound will be set to its default value prior to estimation. Some of the parameters in the fGARCH model are not allowed to take on custom bounds (since they determine the class of the model) nor the beta parameter(s) in the iGARCH model.

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, archex = FALSE),
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" and "csGARCH".
`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` (Logical or Numeric) If logical, indicates whether to use variance targeting for the conditional variance intercept "omega", else if numeric, the value provided is used instead of the unconditional variance for the calculation of the intercept (in combination with the persistence value). Care should be taken if using the numeric option for apARCH and fGARCH models since the intercept is not the variance but sigma raised to the power of some positive value. Finally, if scaling is used (from the fit.control option in `ugarchfit`), the value provided is adjusted accordingly by the routine.

`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).
`archex` (integer) Whether to multiply the last 'archex' external regressors by the conditional standard deviation.

`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. There is also an option to multiply the external regressors by the conditional standard deviation, which may be of use for example in calculating the correlation coefficient in a CAPM type setting.
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, with the exception of the component sGARCH permanent component parameters which are fixed to have a lag-1 autoregressive structure.):

- ***Mean Model***
  - constant: `mu`
  - AR term: `ar1`
  - MA term: `ma1`
  - ARCH-in-mean: `archm`
  - exogenous regressors: `mxreg1`
  - arfima: `arfima`
- ***Distribution Model***
  - skew: `skew`
  - shape: `shape`
  - ghlambda: `lambda (for GHYP distribution)`

- ***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`
- ***Variance Model (csGARCH)***
  - permanent component autoregressive term (rho): `eta11`
  - permanent component shock term (phi): `eta21`
  - permanent component intercept: `omega`
  - transitory component ARCH term: `alpha1`
  - transitory component GARCH term: `beta1`

The terms defined above are better explained in the vignette which provides each model's specification and exact representation. For instance, in the eGARCH model, both alpha and gamma jointly determine the assymetry, and relate to the magnitude and sign of the standardized innovations.

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))
```

VaRDurTest *VaR Duration Test*

Description

Implements the VaR Duration Test of Christoffersen and Pelletier.

Usage

```
VaRDurTest(alpha, actual, VaR, conf.level = 0.95)
```

Arguments

alpha	The quantile (coverage) used for the VaR.
actual	A numeric vector of the actual (realized) values.
VaR	The numeric vector of VaR.
conf.level	The confidence level at which the Null Hypothesis is evaluated.

Details

The duration of time between VaR violations (no-hits) should ideally be independent and not cluster. Under the null hypothesis of a correctly specified risk model, the no-hit duration should have no memory. Since the only continuous distribution which is memory free is the exponential, the test can conducted on any distribution which embeds the exponential as a restricted case, and a likelihood ratio test then conducted to see whether the restriction holds. Following Christoffersen and Pelletier (2004), the Weibull distribution is used with parameter 'b=1' representing the case of

the exponential. A future release will include the choice of using a bootstrap method to evaluate the p-value, and until then care should be taken when evaluating series of length less than 1000 as a rule of thumb.

Value

A list with the following items:

b	The estimated Weibull parameter which when restricted to the value of 1 results in the Exponential distribution.
uLL	The unrestricted Log-Likelihood value.
rLL	The restricted Log-Likelihood value.
LRp	The Likelihood Ratio Test Statistic.
H0	The Null Hypothesis.
Decision	The on H0 given the confidence level

Author(s)

Alexios Ghalanos

References

Christoffersen, P. and Pelletier, D. 2004, Backtesting value-at-risk: A duration-based approach, *Journal of Financial Econometrics*, **2(1)**, 84–108.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "sstd")
fit = ugarchfit(spec, data = dji30ret[1:1000, 1, drop = FALSE])
spec2 = spec
setfixed(spec2)<-as.list(coef(fit))
filt = ugarchfilter(spec2, dji30ret[1001:2500, 1, drop = FALSE], n.old = 1000)
actual = dji30ret[1001:2500,1]
# location+scale invariance allows to use [mu + sigma*q(p,0,1,skew,shape)]
VaR = fitted(filt) + sigma(filt)*qdist("sstd", p=0.05, mu = 0, sigma = 1,
skew  = coef(fit)["skew"], shape=coef(fit)["shape"])
print(VaRDurTest(0.05, actual, VaR))

# Try with the Normal Distribution (it fails)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "norm")
fit = ugarchfit(spec, data = dji30ret[1:1000, 1, drop = FALSE])
spec2 = spec
setfixed(spec2)<-as.list(coef(fit))
filt = ugarchfilter(spec2, dji30ret[1001:2500, 1, drop = FALSE], n.old = 1000)
actual = dji30ret[1001:2500,1]
```

```
# location+scale invariance allows to use [mu + sigma*q(p,0,1,skew,shape)]
VaR = fitted(filt) + sigma(filt)*qdist("norm", p=0.05, mu = 0, sigma = 1)
print(VaRDurTest(0.05, actual, VaR))

## End(Not run)
```

`VaRloss` *Value at Risk loss function of Gonzalez-Rivera, Lee, and Mishra (2004)*

Description

Returns the VaR loss function described in Gonzalez-Rivera, Lee, and Mishra (2004) which is an appropriate function on which to compare models using such methods as the Model Confidence Set (MCS).

Usage

```
VaRloss(alpha, actual, VaR)
```

Arguments

`alpha`	The quantile (coverage) used for the VaR.
`actual`	A numeric vector of the actual (realized) values.
`VaR`	The numeric vector of VaR.

Author(s)

Alexios Ghalanos

References

Gonzalez-Rivera, G., Lee, T. H., and Mishra, S. 2004, Forecasting volatility: A reality check based on option pricing, utility function, value-at-risk, and predictive likelihood. *International Journal of Forecasting*, **20(4)**, 629–645.

VaRplot *Value at Risk Exceedances plot*

Description

Plot the VaR at a given coverage rate against the realized returns for the same period, highlighting the exceedances.

Usage

```
VaRplot(alpha, actual, VaR, title = paste("Daily Returns and Value-at-Risk
Exceedances\n","(alpha=", alpha,")",sep=""), ylab = "Daily Log Returns",
xlab = "Time")
```

Arguments

alpha	The quantile (coverage) used for the VaR.
actual	An xts object of the realized returns.
VaR	An xts object of the forecast VaR, at the given coverage rate p, with the same index as the actual.
title	Plot title.
xlab	Plot x-axis label.
ylab	Plot y-axis label.

Author(s)

Alexios Ghalanos

VaRTest *Value at Risk Exceedances Test*

Description

Implements the unconditional and conditional coverage Value at Risk Exceedances Test.

Usage

```
VaRTest(alpha = 0.05, actual, VaR, conf.level = 0.95)
```

Arguments

alpha	The quantile (coverage) used for the VaR.
actual	A numeric vector of the actual (realized) values.
VaR	The numeric vector of VaR.
conf.level	The confidence level at which the Null Hypothesis is evaluated.

Details

The test implements both the unconditional (Kupiec) and conditional(Christoffersen) coverage tests for the correct number of exceedances. See the references for further details.

Value

A list with the following items:

expected.exceed	The expected number of exceedances (length actual x coverage).
actual.exceed	The actual number of exceedances.
uc.H0	The unconditional coverage test Null Hypothesis.
uc.LRstat	The unconditional coverage test Likelihood Ratio statistic.
uc.critical	The unconditional coverage test critical value.
uc.LRp	The unconditional coverage test p-value.
uc.H0	The unconditional coverage test Null Hypothesis.
uc.Decision	The unconditional coverage test Decision on H0 given the confidence level.
cc.H0	The conditional coverage test Null Hypothesis.
cc.LRstat	The conditional coverage test Likelihood Ratio statistic.
cc.critical	The conditional coverage test critical value.
cc.LRp	The conditional coverage test p-value.
cc.H0	The conditional coverage test Null Hypothesis.
cc.Decision	The conditional coverage test Decision on H0 given the confidence level.

Author(s)

Alexios Ghalanos

References

Christoffersen, P. (1998), Evaluating Interval Forecasts, *International Economic Review*, **39**, 841–862.
Christoffersen, P., Hahn,J. and Inoue, A. (2001), Testing and Comparing Value-at-Risk Measures, *Journal of Empirical Finance*, **8**, 325–342.

Examples

```
## Not run:
data(dji30ret)
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
variance.model = list(model = "gjrGARCH"), distribution.model = "sstd")
fit = ugarchfit(spec, data = dji30ret[1:1000, 1, drop = FALSE])
spec2 = spec
setfixed(spec2)<-as.list(coef(fit))
filt = ugarchfilter(spec2, dji30ret[1001:2500, 1, drop = FALSE], n.old = 1000)
```

```
actual = dji30ret[1001:2500,1]
# location+scale invariance allows to use [mu + sigma*q(p,0,1,skew,shape)]
VaR = fitted(filt) + sigma(filt)*qdist("sstd", p=0.05, mu = 0, sigma = 1,
skew  = coef(fit)["skew"], shape=coef(fit)["shape"])
print(VaRTest(0.05, as.numeric(actual), as.numeric(VaR)))

## End(Not run)
```

Index