Package ‘rugarch’

February 15, 2013

Type Package

Title Univariate GARCH models

Version 1.0-16

Date 2013-01-15

Author Alexios Ghalanos <alexios@4dscape.com>

Maintainer Alexios Ghalanos <alexios@4dscape.com>

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

LinkingTo Rcpp, RcppArmadillo

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

SystemRequirements GNU make

Suggests zoo, xts, timeSeries, nloptr, spd, ks


LazyLoad yes

License GPL-3

Repository CRAN

Date/Publication 2013-01-13 08:10:10

NeedsCompilation yes
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rugarch-package</td>
<td>3</td>
</tr>
<tr>
<td>ARFIMA-class</td>
<td>6</td>
</tr>
<tr>
<td>ARFIMADistribution-class</td>
<td>7</td>
</tr>
<tr>
<td>arfimadistribution-methods</td>
<td>8</td>
</tr>
<tr>
<td>ARFIMAfilter-class</td>
<td>10</td>
</tr>
<tr>
<td>arfimafilter-methods</td>
<td>11</td>
</tr>
<tr>
<td>ARFIMAfit-class</td>
<td>13</td>
</tr>
<tr>
<td>arfimafit-methods</td>
<td>14</td>
</tr>
<tr>
<td>ARFIMAforecast-class</td>
<td>17</td>
</tr>
<tr>
<td>arfimaforecast-methods</td>
<td>18</td>
</tr>
<tr>
<td>ARFIMAmultifilter-class</td>
<td>21</td>
</tr>
<tr>
<td>ARFIMAmultifit-class</td>
<td>22</td>
</tr>
<tr>
<td>ARFIMAmultiforecast-class</td>
<td>23</td>
</tr>
<tr>
<td>ARFIMAmultispec-class</td>
<td>24</td>
</tr>
<tr>
<td>ARFIMAprediction-class</td>
<td>24</td>
</tr>
<tr>
<td>arfima-prediction-methods</td>
<td>25</td>
</tr>
<tr>
<td>ARFIMAroll-class</td>
<td>26</td>
</tr>
<tr>
<td>arfima-roll-methods</td>
<td>27</td>
</tr>
<tr>
<td>ARFIMAAsim-class</td>
<td>29</td>
</tr>
<tr>
<td>arfima-asim-methods</td>
<td>30</td>
</tr>
<tr>
<td>ARFIMAAspec-class</td>
<td>31</td>
</tr>
<tr>
<td>arfima-aspec-methods</td>
<td>32</td>
</tr>
<tr>
<td>autoarfima</td>
<td>33</td>
</tr>
<tr>
<td>BerkowitzTest</td>
<td>35</td>
</tr>
<tr>
<td>DACTest</td>
<td>37</td>
</tr>
<tr>
<td>dji30ret</td>
<td>38</td>
</tr>
<tr>
<td>dmbp</td>
<td>39</td>
</tr>
<tr>
<td>ESTest</td>
<td>39</td>
</tr>
<tr>
<td>ForwardDates-methods</td>
<td>41</td>
</tr>
<tr>
<td>GARCHboot-class</td>
<td>42</td>
</tr>
<tr>
<td>GARCHdistribution-class</td>
<td>43</td>
</tr>
<tr>
<td>GARCHfilter-class</td>
<td>43</td>
</tr>
<tr>
<td>GARCHfit-class</td>
<td>44</td>
</tr>
<tr>
<td>GARCHforecast-class</td>
<td>45</td>
</tr>
<tr>
<td>GARCHpath-class</td>
<td>45</td>
</tr>
<tr>
<td>GARCHroll-class</td>
<td>45</td>
</tr>
<tr>
<td>GARCHsim-class</td>
<td>46</td>
</tr>
<tr>
<td>GARCHspec-class</td>
<td>47</td>
</tr>
<tr>
<td>GARCHtests-class</td>
<td>47</td>
</tr>
<tr>
<td>ghyptransform</td>
<td>48</td>
</tr>
<tr>
<td>GMMTest</td>
<td>49</td>
</tr>
<tr>
<td>HLTTest</td>
<td>50</td>
</tr>
<tr>
<td>multifilter-methods</td>
<td>51</td>
</tr>
<tr>
<td>multifit-methods</td>
<td>53</td>
</tr>
<tr>
<td>multiforecast-methods</td>
<td>54</td>
</tr>
<tr>
<td>multispec-methods</td>
<td>55</td>
</tr>
</tbody>
</table>
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.

Details

Package: rugarch
Type: Package
Version: 1.0-16
Date: 2013-01-15
License: GPL
LazyLoad: yes
Depends: R (>= 2.10.0), Rcpp, RcppArmadillo, methods, parallel, numDeriv, chron, Rsolnp
Suggests: zoo, xts, timeSeries, nloptr, spd, ks

While the package has implemented some safeguards, both during pre-estimation as well as the estimation phase, there is no guarantee of convergence in the fitting procedure. As a result, the fit method allows the user to input starting parameters as well as keep any parameters from the spec as fixed (including the case of all parameters fixed). The functionality of the packages is contained in the main methods for defining a specification ugarchspec, fitting ugarchfit, forecasting ugarchforecast, simulation from fit object ugarchsim, path simulation from specification object ugarchpath, parameter distribution by simulation ugarchdistribution, bootstrap forecast ugarchboot and rolling estimation and forecast ugarchroll. There are also some functions which enable multiple fitting of assets in an easy to use wrapper with the option of multicore functionality, namely multispec, multifit, multifilter and multiforecast. Explanations on the available methods for the returned classes can be found in the documentation for those classes.

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

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

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

Some benchmarks (published and comparison with commercial package), are available through the
ugarchbench function. The ‘inst’ folder of the source distribution also contains various tests which
can be sourced and run by the user, also exposing some finer details of the functionality of the pack-
age. 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.

How to cite this package

Whenever using this package, please cite as

@Manual{Ghalanos_2/zero.noslash12,
  author     = {Alexios Ghalanos},
  title      = {{rugarch}: Univariate GARCH models.},
  year       = {2012},
  note       = {R package version 1.0-16.},
}

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 au-
Berkowitz, J. 2001, Testing density forecasts, with applications to risk management, Journal of
Ding, Z., Granger, C.W.J. and Engle, R.F. 1993, A Long Memory Property of Stock Market Returns
of Finance, 48, 1749–1778.
Pascual, L., Romo, J. and Ruiz, E. 2006, Bootstrap prediction for returns and volatilities in GARCH models, Computational Statistics and Data Analysis.

---

**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
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"), mexitdata = 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.
mexitdata 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

Examples

```r
## Not run:
spec = arfimaspec( mean.model = list(armaOrder = c(2,2), include.mean = TRUE, arfima = FALSE), distribution.model = "norm", fixed.pars = list(ar1=0.6, ar2=0.21, ma1=-0.7, ma2=0.3, mu = 0.02, sigma = 0.02))
dist = arfimadistribution(spec, n.sim = 2000, n.start = 100, m.sim = 100, recursive = TRUE, recursive.length = 10000, recursive.window = 1000)
# slots:
slotNames(dist)
# methods:
# summary
show(dist)
# as.data.frame(...., window, which=c("rmse", "stats", "coef", "coefse"))
# default
as.data.frame(dist)
as.data.frame(dist, window = 1, which = "rmse")
as.data.frame(dist, window = 1, which = "stats")
as.data.frame(dist, window = 1, which = "coef")
as.data.frame(dist, window = 1, which = "coefse")
as.data.frame(dist, window = 8, which = "rmse")
as.data.frame(dist, window = 8, which = "stats")
as.data.frame(dist, window = 8, which = "coef")
as.data.frame(dist, window = 8, which = "coefse")
```
# create some plots
#
# nwindows = dist@dist$details$nwindows
# 2000/3000/4000/5000/6000/7000/8000/9000/10000
#
# expected reduction factor in RMSE for sqrt(N) consistency
expexcted.rmsegr = sqrt(2000/seq(3000,10000,by=1000))
#
# actual RMSE reduction
actual.rmsegr = matrix(NA, ncol = 8, nrow = 6)
rownames(actual.rmsegr) = c("mu", "ar1", "ar2", "ma2", "ma2", "sigma")
# start at 2000 (window 1)
rmse.start = as.data.frame(dist, window = 1, which = "rmse")
for(i in 2:nwindows) actual.rmsegr[,i-1] = as.numeric(as.data.frame(dist,
window = i, which = "rmse")/rmse.start)
par(mfrow = c(2,3))
for(i in 1:6){
plot(seq(3000,10000,by=1000),actual.rmsegr[,i], type = "l", lty = 2,
ylab = "RMSE Reduction", xlab = "N (sim)",main = rownames(actual.rmsegr)[i])
lines(seq(3000,10000,by=1000), expexcted.rmsegr, col = 2)
legend("topright", legend = c("Actual", "Expected"), col = 1:2, bty = "m",
  lty = c(2,1))
}
## End(Not run)

**ARFIMAfilter-class**  
*class: ARFIMA Filter Class*

**Description**

Class for the ARFIMA filter.

**Slots**

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

**Extends**

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

**Methods**

- `as.data.frame` signature(x = "ARFIMAfilter"): Extracts the position (dates), data, filtered values and residuals.
- `coef` signature(object = "ARFIMAfilter"): Extracts the coefficients.
- `fitted` signature(object = "ARFIMAfilter"): Extracts the filtered values.
infocriteria signature(object = "ARFIMAfilter"): Calculates and returns various information criteria.

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

residuals signature(object = "ARFIMAfilter"): Extracts the residuals. 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 datasets augmented by newer data). The reason for using this is so that the old and new datasets agree since the original recursion uses the sum of the residuals to start the recursion and therefore is influenced by new data. For a small augmentation the values converge after x periods, but it is sometimes preferable to have this option so that there is no forward looking information contaminating the study.

Value

A ARFIMAfilter object containing details of the ARFIMA filter.

Author(s)

Alexios Ghalanos

Examples

```R
## Not run:
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE, arfima = FALSE), distribution.model = dist[i])
  fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp", fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist
for(i in 1:9){
  cf = coef(fit[[i]])
  cfmatrix[i, match(names(cf), colnames(cfmatrix))] = cf
}
sk = ku = rep(0, 9)
for(i in 1:9){
  cf = coef(fit[[i]])
  if(fit[[i]]@model$modelfinc[16]>0) sk[i] = dskewness(distribution = dist[i], skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
  if(fit[[i]]@model$modelfinc[17]>0) ku[i] = dkurtosis(distribution = dist[i], skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
}
hq = sapply(fit, FUN = function(x) infocriteria(x)[4])
cfmatrix = cbind(cfmatrix, sk, ku, hq)
colnames(cfmatrix) = c(colnames(cfmatrix), "skewness", "ex.kurtosis", "HQIC")

# filter the data to check results
filt = vector(mode = "list", length = 9)
```
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE, arfima = FALSE), distribution.model = dist[i])
  setfixed(spec) = as.list(coef(fit[[i]]))
  filt[[i]] = arfimafilter(spec = spec, data = sp5/zero.noslash/zero.noslashret)
}
print(cfmatrix, digits = 4)
cat("\nARFIMAfit and ARFIMAfilter residuals check:\n")
print(head(sapply(filt, FUN = function(x) residuals(x))) == head(sapply(fit, FUN = function(x) residuals(x))))
cat("\nas.data.frame method:\n")
print(cbind(head(as.data.frame(filt[[1]]))), head(as.data.frame(fit[[1]]))))
cat("\ncoef method:\n")
print(cbind(coef(filt[[1]]), coef(fit[[1]])))
cat("\nfitted method:\n")
print(cbind(head(fitted(filt[[1]]))), head(fitted(fit[[1]]))))
cat("\ninfcriteria method:\n")
# For filter, it assumes estimation of parameters else does not make sense!
print(cbind(infocriteria(filt[[1]]), infocriteria(fit[[1]])))
cat("\nlikelihood method:\n")
print(cbind(likelihood(filt[[1]]), likelihood(fit[[1]]))))
cat("\nresiduals method:\n")
# Note that we the package will always return the full length residuals and
# fitted object irrespective of the lags (i.e. since this is an ARMA(1,1)
# i.e. max lag = 1, the first row is zero and should be discarded.
print(cbind(head(residuals(filt[[1]]))), head(residuals(fit[[1]]))))
cat("\nuncmean method\n")
print(cbind(uncmean(filt[[1]]), uncmean(fit[[1]])))
cat("\nuncmean method (by simulation):\n")
# For spec and fit
spec = arfimaspec( mean.model = list(armaOrder = c(1,1), include.mean = TRUE, arfima = FALSE), distribution.model = dist[i])
setfixed(spec) = as.list(coef(fit[[i]]))
print(cbind(uncmean(spec, method = "simulation", n.sim = 100000, rseed = 100), uncmean(fit[[1]], method = "simulation", n.sim = 100000, rseed = 100)))
cat("\nssummary method:\n")
show(filt[[1]])
show(fit[[1]])
## End(Not run)
model: Object of class "vector"

Extends

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

Methods

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

coeff 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).

Author(s)

Alexios Ghalanos

Examples

showClass("ARFIMAfit")
arfimafit-methods

Usage

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

Arguments

- **data**: A univariate data object. Can be a numeric vector, matrix, data.frame, zoo, xts, timeSeries, ts or irts object.
- **spec**: An ARFIMA spec object of class ARFIMAspec.
- **out.sample**: A positive integer indicating the number of periods before the last to keep for out of sample forecasting (see details).
- **solver**: One of either “nlminb”, “solnp”, “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.

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
Examples

```r
## Not run:
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE, arfima = FALSE), distribution.model = dist[i])
  fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp", fit.control = list(scale = 1))
}
cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist
for(i in 1:9){
  cf = coef(fit[[i]])
  cfmatrix[i, match(names(cf), colnames(cfmatrix))] = cf
}
}
sk = ku = rep(NA, 9)
for(i in 1:9){
  if(fit[[i]]@model$modelinc[16] > 0) sk[i] = dskewness(distribution = dist[i], skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
  if(fit[[i]]@model$modelinc[17] > 0) ku[i] = dkurtosis(distribution = dist[i], skew = cf["skew"], shape = cf["shape"], lambda = cf["ghlambda"])
}
hq = sapply(fit, FUN = function(x) infocriteria(x)[4])
cfmatrix = cbind(cfmatrix, sk, ku, hq)
colnames(cfmatrix) = c("skewness", "ex.kurtosis", "HQIC")
print(cfmatrix, digits = 4)
# notice that for the student distribution kurtosis is NA since shape (dof) < 4.
cat("as.data.frame method:"
head(as.data.frame(fit[[1]])))
cat("coef method:"
head(coef(fit[[1]])))
cat("fitted method:"
head(fitted(fit[[1]])))
cat("infocriteria method:"
infocriteria(fit[[1]]))
cat("likelihood method:"
likelihood(fit[[1]]))
cat("residuals method:"
residuals(fit[[1]]))
cat("uncmean method:"
uncmean(fit[[1]]))
cat("uncmean method (by simulation):"
uncmean(fit[[1]], method = "simulation", n.sim = 100000, rseed = 100)
```
Class for the ARFIMA forecast.

Slots

forecast: Object of class "vector"
model: Object of class "vector"

Extends

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

Methods

as.array signature(x = "ARFIMAforecast"): Extracts the forecast array with matrix column dimensions equal to the n.ahead value and row dimension 1 (series forecast), and array dimension equal to the number of rolling forecasts chosen.
as.data.frame signature(x = "ARFIMAforecast"): Extracts the forecasts. Takes many additional arguments (see note below).
as.list signature(x = "ARFIMAforecast"): Extracts the forecast list with all rollframes.
fpm signature(object = "ARFIMAforecast"): Forecast performance measures.
show signature(object = "ARFIMAforecast"): Forecast summary returning the 0-roll frame only.

Note

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

Author(s)
Alexios Ghalanos

arfimaforecast-methods

function: ARFIMA Forecasting

Description
Method for forecasting from an ARFIMA model.

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

Arguments

gfitORspec Either an ARFIMA fit object of class ARFIMApred function: ARFIMA Fit Object or alternatively an ARFIMA specification object of class ARFIMApred function: ARFIMA Specification 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.pars function on an ARFIMA object. One step ahead forecasts are based on the value of the previous data, while n-step ahead (n>1) are based on the unconditional mean of the model. The ability to roll the forecast 1 step at a time is implemented with the n.roll argument which controls how many times to roll the n.ahead forecast. The default argument of n.roll = 0 denotes no rolling beyond the first forecast and returns the standard n.ahead forecast. Critically, since n.roll
depends on data being available from which to base the rolling forecast, the `arfimafit` function needs to be called with the argument `out.sample` being at least as large as the `n.roll` argument, or in the case of a specification being used instead of a fit object, the `out.sample` argument directly in the forecast function.

Value

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

Author(s)

Alexios Ghalanos

Examples

```r
## Not run:
# Long Horizon Forecast
data(sp500ret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
                                           arfima = FALSE), distribution.model = dist[i])
  fit[[i]] = arfimafit(spec = spec, data = sp500ret, solver = "solnp",
                      fit.control = list(scale = 1))
  cfmatrix = matrix(NA, nrow = 9, ncol = 7)
colnames(cfmatrix) = c("mu", "ar1", "ma1", "sigma", "skew", "shape", "ghlambda")
rownames(cfmatrix) = dist
  for(i in 1:9){
    cf = coef(fit[[i]])
    cfmatrix[i, match(names(cf), colnames(cfmatrix))] = cf
  }
  umean = rep(0, 9)
  for(i in 1:9){
    umean[i] = uncmean(fit[[i]])
  }
  forc = vector(mode = "list", length = 9)
  for(i in 1:9){
    forc[[i]] = arfimaforecast(fit[[i]], n.ahead = 100)
  }
  lmean40 = sapply(forc, FUN = function(x) as.numeric(as.data.frame(x)[40,1]))
  cfmatrix1 = cbind(cfmatrix, umean, lmean40)
  colnames(cfmatrix1) = c(colnames(cfmatrix1[,1:7]), "uncmean", "forecast40")
}
```

# forecast with spec to check results
forc2 = vector(mode = "list", length = 9)
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
                      arfima = FALSE), distribution.model = dist[i])
  setfixed(spec) = as.list(coef(fit[[i]]))
  forc2[[i]] = arfimaforecast(spec, data = sp5/zero.noslash/zero.noslashret, n.ahead = 100)
}
lmean240 = sapply(forc2, FUN = function(x) as.numeric(as.data.frame(x)[4/zero.noslash,1]))
cfmatrix2 = cbind(cfmatrix, umean, lmean240)
colnames(cfmatrix2) = c(colnames(cfmatrix[,1:7]), "uncmean", "forecast4/zero.noslash")
cat("\nARFIMAforecast from ARFIMAfit and ARFIMAspec check:")
cat("\nFit\n")
print(cfmatrix1, digits = 4)
cat("\nSpec\n")
print(cfmatrix2, digits = 4)
# methods and slots
slotNames(forc[[1]])
showMethods(classes="ARFIMAforecast")
# summary
show(forc[[1]])
# Extractor Functions
# as array (array dimension [3] is 1 since n.roll = 0 i.e. no rolling beyond
# the first)
as.array(forc[[1]])
# as.data.frame
as.data.frame(forc[[1]])
# as.list
as.list(forc[[1]])

# Rolling Forecast

# Rolling Forecast

# as array (array dimension [3] is 1 since n.roll = 0 i.e. no rolling beyond
# the first)
as.array(forc[[1]])
# as.data.frame
as.data.frame(forc[[1]])
# as.list
as.list(forc[[1]])

# Rolling Forecast

data(sp5/zero.noslash/zero.noslashret)
fit = vector(mode = "list", length = 9)
dist = c("norm", "snorm", "std", "sstd", "ged", "sged", "nig", "ghyp", "jsu")
for(i in 1:9){
  spec = arfimaspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE,
                      arfima = FALSE), distribution.model = dist[i])
  fit[[i]] = arfima(armaOrder = c(1,1), include.mean = TRUE,
                      arfima = FALSE), distribution.model = dist[i])
  out.sample = 1000, fit.control = list(scale = 1))
}

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

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

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

par(mfrow = c(1,2))
dd = rownames(tail(sp5, 1250))
clrs = rainbow(9, alpha = 1, start = 0.1, end = 0.95)
plot(as.Date(dd), tail(sp5[,1], 1250), type = "l",
ylim = c(-2, 2), col = "lightgrey", ylab = "", xlab = "",
main = "Rolling 1-ahead Forecasts vs Actual")
for(i in 1:9){
  tmp = tail(sp5[,1], 1250)
  tmp[251:1250] = rollforc[1:1,i]
  lines(as.Date(dd), c(rep(NA, 25), tmp[-(1:25)], col = clrs[i])
}
legend("topleft", legend = dist, col = clrs, fill = clrs, bty = "n")

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

# fpm comparison
compm = matrix(NA, nrow = 3, ncol = 9)
compm = sapply(fpmlist, FUN = function(x) c(mean(x[,"SE"]), mean(x[,"AE"]),
mean(x[,"DAC"])))
rownames(compm) = c("MSE", "MAD", "DAC")
cat("\nRolling Forecast FPM\n")
print(compm, digits = 4)
cat("\nMethods Check\n")
as.data.frame(forc[[1]], rollframe = 0)
as.data.frame(forc[[1]], rollframe = 999)
t(as.data.frame(forc[[1]], rollframe = "all", aligned = FALSE))
fpm(forc[[1]], summary = TRUE)
show(forc[[1]])

## End(Not run)
ARFIMAmultifilter-class

class: ARFIMA Multiple Filter Class

Description

Class for the ARFIMA Multiple filter.

Slots

filter: Object of class "vector"
desc: Object of class "vector"

Extends

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

Methods

fitted signature(object = "ARFIMAmultifilter"): Extracts the fitted values.
residuals signature(object = "ARFIMAmultifilter"): Extracts the residuals. 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: ARIMA Multiple Forecast Class*

---

Description

Class for the ARFIMA Multiple forecast.

Slots

- **forecast**: Object of class "vector"
- **desc**: Object of class "vector"

Extends

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

Methods

- **as.array** signature(x = "ARFIMAmultiforecast"): extracts the forecast array with matrix column dimensions equal to the number of assets, row dimension the n.ahead and array dimension equal to the number of rolling forecasts chosen.
- **as.list** signature(x = "ARFIMAmultiforecast"): extracts the forecast list of length equal to the number of assets, sublists equal to n.roll, row dimension of each sublist equal to n.ahead and column dimension equal to 1 (series forecasts).
- **show** signature(object = "ARFIMAmultiforecast"): forecast summary.

Author(s)

Alexios Ghalanos
ARFIMAmultispec-class  
\textit{class: ARIMA Multiple Specification Class}  

\textbf{Description}  
Class for the ARFIMA Multiple specification.  

\textbf{Slots}  
\begin{itemize}  
\item \texttt{spec}: Object of class "vector"  
\item \texttt{type}: Object of class "character"  
\end{itemize}  

\textbf{Extends}  
Class "\texttt{ARFIMA}"" directly. Class "\texttt{rGARCH}"" by class "\texttt{ARFIMA}", distance 2.  

\textbf{Methods}  
\begin{itemize}  
\item \texttt{show} signature(object = "ARFIMA\texttt{multispec}"): specification summary.  
\end{itemize}  

\textbf{Author(s)}  
Alexios Ghalanos  

---  

ARFIMApath-class  
\textit{class: ARIMA Path Simulation Class}  

\textbf{Description}  
Class for the ARFIMA Path simulation.  

\textbf{Slots}  
\begin{itemize}  
\item \texttt{path}: Object of class "vector"  
\item \texttt{model}: Object of class "vector"  
\item \texttt{seed}: Object of class "integer"  
\end{itemize}  

\textbf{Extends}  
Class "\texttt{ARFIMA}"" directly. Class "\texttt{rGARCH}"" by class "\texttt{ARFIMA}"", distance 2.  

\textbf{Methods}  
\begin{itemize}  
\item \texttt{as.data.frame} signature(x = "ARFIMA\texttt{path}"): Extracts the simulated path values (see note).  
\item \texttt{show} signature(object = "ARFIMA\texttt{path}"): path simulation summary.  
\end{itemize}
arfimapath-methods

Note

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

Author(s)

Alexios Ghalanos

arfimapath-methods  function: ARFIMA Path Simulation

Description

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

Usage

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

Arguments

`spec`  An ARFIMA object of class `ARFIMAspec` with the required parameters of the model supplied via the `fixed.pars` list argument.

`n.sim`  The simulation horizon.

`n.start`  The burn-in sample.

`m.sim`  The number of simulations.

`prereturns`  Allows the starting return data to be provided by the user.

`preresiduals`  Allows the starting residuals to be provided by the user.

`rseed`  Optional seeding value(s) for the random number generator.

`custom.dist`  Optional density with fitted object from which to simulate. The “type” argument denotes whether the standardized innovations are passed (“z”) else the innovations (anything other than “z”).

`mexsimdata`  Matrix of simulated external regressor-in-mean data. If the fit object contains external regressors in the mean equation, this must be provided.

...  

Details

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

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.

coeff 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 coeff method will return a list of the coefficients and their robust standard errors (assuming the keep.coeff 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. \textit{VaR.alpha} (for the VaR backtest report) is the tail probability and defaults to 0.01.
3. \textit{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 \texttt{fpm} method (separately from report) takes additional logical argument \textit{summary}, which when \texttt{TRUE} will return the mean squared error (MSE), mean absolute error (MAE) and directional accuracy of the forecast versus realized returns. When \texttt{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 \texttt{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 \texttt{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 \texttt{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 estima-
tion, 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
ARFIMAsim-class

Examples

```r
## 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

- `as.data.frame` signature(x = "ARFIMAsim"): extracts the simulated values (see note).
- `show` signature(object = "ARFIMAsim"): simulation summary.

Note

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

Author(s)

Alexios Ghalanos
arfimasim-methods  

function: ARFIMA Simulation

Description
Method for simulation from ARFIMA models.

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

Arguments

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

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

Value
A ARFIMAsim object containing details of the ARFIMA simulation.
Author(s)
Alexios Ghalanos

---

**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"=(-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 distribution, “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 implemented 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
Mean Model:

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

Distribution Model:

dlamba     dlamba (for GHYP distribution)
skew        skew
shape       shape

Value

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

Author(s)

Alexios Ghalanos

Description

Select best fitting ARFIMA models based on information criteria.

Usage

```r
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 equa-
                      tion 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 ‘re-
              turn.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)
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 values 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


Examples

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

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

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

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

Examples

```r
## 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 (Observed) 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)
```

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
Description

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

Usage

data(dmbp)

Format

A data.frame containing 2x1974 observations.

Source

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

References


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

## Examples

```r
## 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[1:2500, 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)
```

---

### ForwardDates-methods

**function: Generate Future Dates**

#### Description

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

#### Usage

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

#### Arguments

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

#### Value

A POSIXct vector of future dates.
Note

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

Author(s)

Alexios Ghalanos

Examples

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

GARCHboot-class

**class:** GARCH Bootstrap Class

Description

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

Objects from the Class

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

Extends

Class "rGARCH", directly.

Methods

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

Author(s)

Alexios Ghalanos
Examples

showClass("GARCHboot")

GARCHdistribution-class

class: GARCH Parameter Distribution Class

Description

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

Objects from the Class

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

Extends

Class "rGARCH", directly.

Methods

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

Author(s)

Alexios Ghalanos

Examples

showClass("GARCHdistribution")

GARCHfilter-class

class: GARCH Filter Class

Description

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

Objects from the Class

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

Extends

Class "rGARCH", directly.
Methods

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

Author(s)

Alexios Ghalanos

Examples

showClass("GARCHfilter")

GARCHfit-class

class: GARCH Fit Class

Description

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

Objects from the Class

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

Extends

Class "rGARCH", directly.

Methods

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

Author(s)

Alexios Ghalanos

Examples

showClass("GARCHfit")
GARCHforecast-class  

**class: GARCH Forecast Class**

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

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

**Extends**
Class "rGARCH", directly.

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

**Author(s)**
Alexios Ghalanos

**Examples**

```r
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**

**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**

**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**

```r
ghyptransform(mu = /zero.noslash, 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 nrow(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**

The GMM Orthogonality Test of Hansen

**Description**

Implements the GMM Orthogonality Test of Hansen.

**Usage**

\[
\text{GMMTest}(z, \text{lags} = 1, \text{skew}=0, \text{kurt}=3, \text{conf.level} = 0.95)
\]

**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 containing the first 4 moments statistics, their standard error and t-values, ‘M1’ to ‘M4’, the t-value of the test of the the joint squared residuals under the specified number of lags, ‘Q2’, and the t-value of the test of joint nullness, ‘J’. The joint tests (‘Q2’ and ‘J’) are distributed chi-squared with n.lag and 4 + n.lag d.o.f. respectively, while the individual moment conditions may be tested for significance using a standard t-test.

**Value**

A list with the following items:

- **statmat**: The matrix of the statistics (see details).
- **H0**: The Null Hypothesis.
- **critical.value**: The critical value for each statistic.
- **Decision**: Whether to reject or not the Null given the conf.level.
HLTest

Author(s)

Alexios Ghalanos

References


Examples

```r
## 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 Portmanteau 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

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.

Examples

```r
## 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)
```
**Description**

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

**Usage**

```r
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

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. Can be a matrix or data.frame object, no other class supported at present.
solver One of either “nlminb” or “solnp”.
solver.control Control arguments list passed to optimizer.
fit.control Control arguments passed to the fitting routine. Stationarity (only for the GARCH case) explicitly imposes the variance stationarity constraint during optimization. The fixed.se argument controls whether standard errors should be calculated for those parameters which were fixed (through the fixed.pars argument of the ugarchspec or arfimaspec functions). The scale parameter controls whether the data should be scaled before being submitted to the optimizer, 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

```r
## Not run:
data(dji3ret)
spec = ugarchspec()
mspec = multispec(replicate(spec, n = 4))
fitlist = multifit(multispec = mspec, data = dji3ret[,1:4])
## End(Not run)
```

### Description

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

### Usage

```r
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

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

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

class: rGARCH Class

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

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

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

author(s)
Alexios Ghalanos

examples
showClass("rGARCH")

rgarchdist  

Distribution: rugarch distribution functions

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

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

usage
ddist(distribution = "norm", y, mu = 0, sigma = 1, lambda = -0.5, skew = 1, shape = 5)
pdist(distribution = "norm", q, mu = 0, sigma = 1, lambda = -0.5, skew = 1, shape = 5)
qdist(distribution = "norm", p, mu = 0, sigma = 1, lambda = -0.5, skew = 1, shape = 5)
rdist(distribution = "norm", n, mu = 0, sigma = 1, lambda = -0.5, skew = 1, shape = 5)
```r
fitdist(distribution = "norm", x, control=list())
dskewness(distribution = "norm", skew = 1, shape = 5, lambda = -0.5)
dkurtosis(distribution = "norm", skew = 1, shape = 5, lambda = -0.5)
```

**Arguments**

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

**Details**

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

**Value**

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

`fitdist` returns a list with the following components:

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

`dskewness` returns the skewness of the distribution. `dkurtosis` returns the excess kurtosis of the distribution.
Author(s)

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


---

data: Standard and Poors 500 Closing Value Log Return

Description

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

Usage

data(sp500ret)

Format

A data.frame containing 1x5523 observations.

Source

Yahoo Finance

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 benchmark 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

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

References


Examples

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

## End(Not run)
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. 2006, Bootstrap prediction for returns and volatilities in GARCH models, Computational Statistics and Data Analysis.

See Also

Classes uGARCHforecast, uGARCHfit and uGARCHspec.

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. 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

```r
## 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), mexitdata = NULL, vexitdata = NULL,
fit.control = list(), solver = "solnp", solver.control = list(), cluster = NULL, ...)

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)
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.
**persistence** signature(object = "uGARCHfilter", pars = "missing", distribution = "missing", model = "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")
Calculates and returns the long run unconditional variance of the garch filter given a uGARCHfilter object.

**plot** signature(x = "uGARCHfilter", y = "missing")
Filter plots

**show** signature(object = "uGARCHfilter")
Filter summary.

**Note**

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

**Author(s)**

Alexios Ghalanos

**Examples**

```r
## 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)

(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', ...)

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).
...

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

```r
## 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**

### Description

Class for the univariate GARCH fit.

### Objects from the Class

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

### Extends

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

### Slots

- **fit**: Object of class "vector" Holds data on the fitted model.
- **model**: Object of class "vector" The model specification common to all objects.

### Methods

- **as.data.frame** signature(x = "uGARCHfit"): Extracts the position (dates), data, fitted values, residuals and conditional sigma.
- **coef** signature(object = "uGARCHfit"): Extracts the coefficients.
- **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.

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"): 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"): 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).

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

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

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

```r
## Not run:
# Basic GARCH(1,1) Spec
data(dm)bp
spec = uGARCHspec()
fit = uGARCHfit(data = dm, spec = spec)
fit
```
# object fit:
slotNames(fit)
# sublist fit@fit
names(fit@fit)

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

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

## End(Not run)
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. 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.

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.

Value

A `uGARCHfit` object containing details of the GARCH fit.

Note

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

Description

Class for the univariate GARCH forecast.

Author(s)

Alexios Ghalanos

See Also

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

Examples

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

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

Extends

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

Methods

- **as.array** signature(x = "uGARCHforecast"): Extracts the forecast array with matrix column dimensions equal to the n.ahead value and row dimension 2 (sigma and series forecast), and array dimension equal to the number of rolling forecasts chosen.
- **as.data.frame** signature(x = "uGARCHforecast"): Extracts the forecasts. Takes many additional arguments (see note below).
- **as.list** signature(x = "uGARCHforecast"): Extracts the forecast list with all rollframes.
- **plot** signature(x = "uGARCHforecast", y = "missing"): Forecast plots with n.roll optional argument indicating the rolling sequence to plot.
- **fpm** signature(object = "uGARCHforecast"): Forecast performance measures.
- **show** signature(object = "uGARCHforecast"): Forecast summary returning the 0-roll frame only.

Note

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

The plot method takes additional arguments which and n.roll indicating which roll frame to plot.

Author(s)

Alexios Ghalanos

See Also

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

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

## End(Not run)
```

ugarchforecast-methods

*function: Univariate GARCH Forecasting*

Description

Method for forecasting from a variety of univariate GARCH models.

Usage

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

Arguments

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

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

## End(Not run)
```
Description

Class for the univariate GARCH Multiple filter.

Extends

Class "GARCHfilter", directly. Class "rGARCH", by class "GARCHfit", 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.
Methods

c\_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

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

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

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

Alexios Ghalanos

See Also

Classes uGARCHmultifilter, uGARCHmultifit and uGARCHmultispec.

---

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.
Description

Class for the univariate GARCH Path simulation.

Objects from the Class

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

Extends

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

Methods

as.data.frame signature(x = "uGARCHpath"): extracts the simulated path values (see note).
plot signature(x = "uGARCHpath", y = "missing"): path simulation plots.
show signature(object = "uGARCHpath"): path simulation summary.

Note

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

Author(s)

Alexios Ghalanos

See Also

Classes uGARCHsim, uGARCHfit and uGARCHspec.
function: Univariate GARCH Path Simulation

Description

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

Usage

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

Arguments

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

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

Value

A `uGARCHpath` object containing details of the GARCH path simulation.

Author(s)

Alexios Ghalanos

See Also

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

Examples

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

**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” will 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
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 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 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). 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 ugarhsprec, fitting ugarhfit, filtering ugarhfilter, forecasting ugarhforecast, simulation ugarhsim, parameter distribution and uncertainty ugarhdistribution, bootstrap forecast ugarhboot.

Examples

```r
## Not run:
data(sp500ret)
spec = ugarhsprec(distribution.model = "std")
mod = ugarhroll(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)
```
### uGARCHsim-class

**class: Univariate GARCH Simulation Class**

**Description**

Class for the univariate GARCH simulation.

**Extends**

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

**Slots**

- simulation: Object of class "vector" Holds data on the simulation.
- model: Object of class "vector" The model specification common to all objects.
- seed: Object of class "integer" The random seed used.

**Methods**

- **as.data.frame** signature(x = "uGARCHsim"): Extracts the simulated values (see note).
- **plot** signature(x = "uGARCHsim", y = "missing"): Simulation plots.
- **show** signature(object = "uGARCHsim"): Simulation summary.

**Note**

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

**Author(s)**

Alexios Ghalanos

**See Also**

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

**Examples**

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

## End(Not run)

## Function: Univariate GARCHSimulation

### Description

Method for simulation from a variety of univariate GARCH models.

### Usage

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

### Arguments

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

...
Details

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

Value

A uGARCHsim object containing details of the GARCH simulation.

Author(s)

Alexios Ghalanos

See Also

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

Examples

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

## End(Not run)
```

uGARCHspec-class

class: Univariate GARCH Specification Class

Description

Class for the univariate GARCH specification.

Extends

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

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

Methods

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

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

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

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 = ugarspec()
spec
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 “ALL-
GARCH”.
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 Nu-
meric) 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 distribution, “snorm” for the skew-normal distribution, “std” for the student-t, “ssstd” 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 implemented 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 regressors 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
Distribution Model:

ghlambda lambda (for GHYP distribution)
skew skew
shape shape

Variance Model (common specs):

constant omega
ARCH term alpha1
GARCH term beta1
exogenous regressors vxreg1

Variance Model (GJR, EGARCH):

assymetry term gamma1

Variance Model (APARCH):

assymetry term gamma1
power term delta

Variance Model (FGARCH):

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

Variance Model (csGARCH):

permanent component autoregressive term (rho) eta11
permanent component shock term (phi) eta21
permanent component intercept (omega) \( \omega \)
transitory component ARCH term \( \alpha_1 \)
transitory component GARCH term \( \beta_1 \)

**Value**

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

**Author(s)**

Alexios Ghalanos

**Examples**

```r
# 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))
```

**Description**

Implements the VaR Duration Test of Christoffersen and Pelletier.

**Usage**

```r
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


Examples

```r
## 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)

```r
spec = ugarchspec(mean.model = list(armaOrder = c(1,1), include.mean = TRUE),
                   variance.model = list(model = "gjrGARCH"), distribution.model = "norm")
fit = ugarchfit(spec, data = dji3/zero.noslashret[1:10000, 1, drop = FALSE])
spec2 = spec
setfixed(spec2)<-as.list(coef(fit))
filt = ugarchfilter(spec2, dji3/zero.noslashret[1:2500, 1, drop = FALSE], n.old = 1000)
actual = dji3/zero.noslashret[1:2500,1]
# location+scale invariance allows to use \([\mu + \sigma\cdot q(p,0,1,\text{skew},\text{shape})]\)
VaR = fitted(filt) + sigma(filt)\cdot qdist(\text{"norm"}, p=0.05, \mu = 0, \sigma = 1)
print(VaRDurTest(0.05, actual, VaR))
```

```{r}
## End(Not run)
```

---

### VaRTeset

#### Description

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

#### Usage

```r
VaRTeset(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.
VaRTest

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

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, actual, VaR))
## End(Not run)
**WeekDayDummy-methods**

**function: Create Dummy Day-of-Week Variable**

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Author(s)**

Alexios Ghalanos

**Examples**

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

*Topic classes

ARFIMA-class, 6
ARFIMAdistribution-class, 7
ARFIMAFilter-class, 10
ARFIMAfit-class, 13
ARFIMAforecast-class, 17
ARFIMAmultifilter-class, 22
ARFIMAmultifit-class, 22
ARFIMAmultiforecast-class, 23
ARFIMAmultispec-class, 24
ARFIMAprediction-class, 24
ARFIMArain-class, 26
ARFIMAsim-class, 29
ARFIMAspec-class, 31
GARCHboot-class, 42
GARCHdistribution-class, 43
GARCHfilter-class, 43
GARCHfit-class, 44
GARCHforecast-class, 45
GARCHpath-class, 45
GARCHroll-class, 46
GARCHsim-class, 47
GARCHspec-class, 47
GARCHtests-class, 48
rGARCH-class, 57
uGARCHboot-class, 60
uGARCHdistribution-class, 64
uGARCHfilter-class, 67
uGARCHfit-class, 70
uGARCHforecast-class, 75
uGARCHmultifilter-class, 78
uGARCHmultifit-class, 79
uGARCHmultiforecast-class, 80
uGARCHmultispec-class, 81
uGARCHpath-class, 82
uGARCHroll-class, 84
uGARCHsim-class, 88
uGARCHspec-class, 90

*Topic datasets

dji30ret, 38
dmbp, 39
sp500ret, 59

*Topic methods

arfimadistribution-methods, 8
arfimafilter-methods, 11
arfimafit-methods, 14
arfimaforecast-methods, 18
arfimapath-methods, 25
arfimaroll-methods, 27
arfimasim-methods, 30
arfimaspec-methods, 32
ForwardDates-methods, 41
multifit-methods, 54
multiforecast-methods, 55
multispec-methods, 56
ugarchboot-methods, 61
ugarchdistribution-methods, 65
ugarchfit-methods, 73
ugarchforecast-methods, 77
ugarchpath-methods, 83
ugarchroll-methods, 86
ugarchsim-methods, 89
ugarchsfit-methods, 92
WeekDayDummy-methods, 99

ARFIMA, 7, 10, 14, 17, 22–24, 26, 29, 31
ARFIMA-class, 6
ARFIMAdistribution, 9
arfimadistribution, 7
arfimadistribution
(arfimadistribution-methods), 8
arfimadistribution,ANY-method
(arfimadistribution-methods), 8
arfimadistribution,ARFIMAfit-method
(arfimadistribution-methods), 8
arfimadistribution,ARFIMAspec-method
(arfimadistribution-methods), 8
ARFIMAdistribution-class, 7
arfimadistribution-methods, 8
INDEX

ARFIMAfilter, 12
arfimafilter (arfimafilter-methods), 11
arfimafilter, ANY-method
 (arfimafilter-methods), 11
arfimafilter, ARFIMAspec-method
 (arfimafilter-methods), 11
ARFIMAfilter-class, 10
arfimafilter-methods, 11
ARFIMAfit, 8, 15, 18, 30
arfimafit, 11, 19, 32
arfimafit (arfimafit-methods), 14
arfimafit, ANY-method
 (arfimafit-methods), 14
arfimafit, ARFIMAspec-method
 (arfimafit-methods), 14
ARFIMAfit-class, 13
arfimafit-methods, 14
ARFIMAforecast, 19
arfimaforecast, 15
arfimaforecast
 (arfimaforecast-methods), 18
arfimaforecast, ANY-method
 (arfimaforecast-methods), 18
arfimaforecast, ARFIMAfit-method
 (arfimaforecast-methods), 18
arfimaforecast, ARFIMAspec-method
 (arfimaforecast-methods), 18
ARFIMAforecast-class, 17
arfimaforecast-methods, 18
ARFIMAmultifilter, 53
ARFIMAmultifilter-class, 21
ARFIMAmultifit, 53-55
ARFIMAmultifit-class, 22
ARFIMAmultiforecast, 56
ARFIMAmultiforecast-class, 23
ARFIMAmultispec, 53-56
ARFIMAmultispec-class, 24
ARFIMApath, 26
arfimapath (arfimapath-methods), 25
arfimapath, ANY-method
 (arfimapath-methods), 25
arfimapath, ARFIMAspec-method
 (arfimapath-methods), 25
ARFIMApath-class, 24
arfimapath-methods, 25
ARFIMARoll, 28
arfimaroll (arfimaroll-methods), 27
arfimaroll, ANY-method
 (arfimaroll-methods), 27
arfimaroll, ARFIMAspec-method
 (arfimaroll-methods), 27
ARFIMARoll-class, 26
arfimaroll-methods, 27
ARFIMASim, 30
arfimasim, 25
arfimasim (arfimasim-methods), 30
arfimasim, ANY-method
 (arfimasim-methods), 30
arfimasim, ARFIMAfit-method
 (arfimasim-methods), 30
ARFIMASim-class, 29
arfimasim-methods, 30
ARFIMAspec, 8, 11, 15, 18, 25, 33, 56
arfimaspect, 15, 54
arfimaspect (arfimaspect-methods), 32
arfimaspect, ANY-method
 (arfimaspect-methods), 32
ARFIMAspec-class, 31
arfimaspect-methods, 32
as.array, ARFIMAforecast-method
 (ARFIMAforecast-class), 17
as.array, ARFIMAmultiforecast-method
 (ARFIMAmultiforecast-class), 23
as.array, uGARCHforecast-method
 (uGARCHforecast-class), 75
as.array, uGARCHmultiforecast-method
 (uGARCHmultiforecast-class), 80
as.data.frame, ARFIMAdistribution-method
 (ARFIMAdistribution-class), 7
as.data.frame, ARFIMAfilter-method
 (ARFIMAfilter-class), 10
as.data.frame, ARFIMAfit-method
 (ARFIMAfit-class), 13
as.data.frame, ARFIMAforecast-method
 (ARFIMAforecast-class), 17
as.data.frame, ARFIMApath-method
 (ARFIMApath-class), 24
as.data.frame, ARFIMARoll-method
 (ARFIMARoll-class), 26
as.data.frame, ARFIMASim-method
 (ARFIMASim-class), 29
as.data.frame, uGARCHboot-method
 (uGARCHboot-class), 60
as.data.frame, uGARCHdistribution-method
 (uGARCHdistribution-class), 64
as.data.frame, uGARCHfilter-method
INDEX

(uGARCHfilter-class), 67
as.data.frame, uGARCHfit-method
(uGARCHfit-class), 70
as.data.frame, uGARCHforecast-method
(uGARCHforecast-class), 75
as.data.frame, uGARCHpath-method
(uGARCHpath-class), 82
as.data.frame, uGARCHroll-method
(uGARCHroll-class), 84
as.data.frame, uGARCHsim-method
(uGARCHsim-class), 88
as.list, ARFIMAforecast-method
(ARFIMAforecast-class), 17
as.list, ARFIMAmultiforecast-method
(ARFIMAmultiforecast-class), 23
as.list, uGARCHforecast-method
(uGARCHforecast-class), 75
as.list, uGARCHmultiforecast-method
(uGARCHmultiforecast-class), 80
autoarfima, 33

dist, BerkowitzTest, 35, 52
coeff, ARFIMAfilter-method
(ARFIMAfilter-class), 10
coeff, ARFIMAfilt-method
(ARFIMAfilt-class), 13
coeff, ARFIMAmultifilter-method
(ARFIMAmultifilter-class), 22
coeff, ARFIMAmultifit-method
(ARFIMAmultifit-class), 22
coeff, ARFIMAroll-method
(ARFIMAroll-class), 26
coeff, uGARCHfilter-method
(uGARCHfilter-class), 67
coeff, uGARCHfit-method
(uGARCHfit-class), 70
coeff, uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
coeff, uGARCHmultifit-method
(uGARCHmultifit-class), 79
coeff, uGARCHroll-method
(uGARCHroll-class), 84
convergence, uGARCHfit-method
(uGARCHfit-class), 70
convergence, uGARCHforecast-method
(uGARCHforecast-class), 75
convergence, uGARCHroll-method
(uGARCHroll-class), 84
convergence, uGARCHsim-method
(uGARCHsim-class), 88
fitdist, BerkowitzTest, 35, 52
fitdist, ARFIMAforecast-method
(ARFIMAforecast-class), 17
fitdist, ARFIMAfilt-method
(ARFIMAfilt-class), 13
fitdist, ARFIMAmultifilter-method
(ARFIMAmultifilter-class), 22
fitdist, ARFIMAmultifit-method
(ARFIMAmultifit-class), 22
fitdist, uGARCHfit-method
(uGARCHfit-class), 70
fitdist, uGARCHforecast-method
(uGARCHforecast-class), 75
fitdist, uGARCHroll-method
(uGARCHroll-class), 84
fitdist, uGARCHsim-method
(uGARCHsim-class), 88
fitdist, uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
fitdist, uGARCHmultifit-method
(uGARCHmultifit-class), 79
ForwardDates, 5
ForwardDates, ARFIMAfit-method
(ARFIMAfit-class), 13
ForwardDates, ARFIMAmultifilter-method
(ARFIMAmultifilter-class), 22
ForwardDates, ARFIMAmultifit-method
(ARFIMAmultifit-class), 22
ForwardDates, ARFIMAroll-method
(ARFIMAroll-class), 26
ForwardDates, uGARCHfit-method
(uGARCHfit-class), 70
ForwardDates, uGARCHforecast-method
(uGARCHforecast-class), 75
ForwardDates, uGARCHroll-method
(uGARCHroll-class), 84
ForwardDates, uGARCHsim-method
(uGARCHsim-class), 88
ForwardDates, uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
ForwardDates, uGARCHmultifit-method
(uGARCHmultifit-class), 79
ForwardDates, uGARCHroll-method
(uGARCHroll-class), 84
ForwardDates, uGARCHsim-method
(uGARCHsim-class), 88
ForwardDates, uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
ForwardDates, uGARCHmultifit-method
(uGARCHmultifit-class), 79
ForwardDates, uGARCHroll-method
(uGARCHroll-class), 84
INDEX

fpm, uGARCHroll-method
  (uGARCHroll-class), 84

GARCHboot. 60
GARCHboot-class, 42
GARCHdistribution, 64
GARCHdistribution-class, 43
GARCHfilter, 67, 79
GARCHfilter-class, 43
GARCHfit, 70, 79
GARCHfit-class, 44
GARCHForecast, 76, 80
GARCHForecast-class, 45
GARCHpath-class, 45
GARCHroll, 84
GARCHroll-class, 46
GARCHsim, 88
GARCHsim-class, 47
GARCHspec, 81, 90
GARCHspec-class, 47
GARCHtests-class, 48
getspec (uGARCHfit-class), 70
getspec, ANY-method (uGARCHfit-class), 70
getspec, ARFIMAfit-method
  (ARFIMAfit-class), 13
getspec, uGARCHfit-method
  (uGARCHfit-class), 70
ghyptransform, 4, 49
GMMTest. 50
gof (uGARCHfit-class), 70
gof, ANY, ANY-method (uGARCHfit-class), 70
gof, uGARCHFilter, numeric-method
  (uGARCHfit-class), 67
gof, uGARCHfit, numeric-method
  (uGARCHfit-class), 70
halflife (uGARCHfit-class), 70
halflife, ANY, ANY, ANY, ANY-method
  (uGARCHfit-class), 70
halflife, missing, numeric, character, character
  (uGARCHfit-class), 70
halflife, uGARCHfilter, missing, missing, missing,
  uGARCHfilter-class), 67
halflife, uGARCHfit, missing, missing, missing,
  uGARCHfit-class), 70
halflife, uGARCHspec, missing, missing, missing,
  uGARCHspec-class), 90
HLTest. 51
infocriteria (uGARCHfit-class), 70
infocriteria, ANY-method
  (uGARCHfit-class), 70
infocriteria, ARFIMAfilter-method
  (ARFIMAfilter-class), 10
infocriteria, ARFIMAfit-method
  (ARFIMAfit-class), 13
infocriteria, uGARCHfilter-method
  (uGARCHfilter-class), 67
infocriteria, uGARCHfit-method
  (uGARCHfit-class), 70
likelihood (uGARCHfit-class), 70
likelihood, ANY-method
  (uGARCHfit-class), 70
likelihood, ARFIMAfilter-method
  (ARFIMAfilter-class), 10
likelihood, ARFIMAfit-method
  (ARFIMAfit-class), 13
likelihood, ARFIMAmultifilter-method
  (ARFIMAmultifilter-class), 22
likelihood, ARFIMAmultifit-method
  (ARFIMAmultifit-class), 22
likelihood, uGARCHfilter-method
  (uGARCHfilter-class), 67
likelihood, uGARCHfit-method
  (uGARCHfit-class), 70
likelihood, uGARCHmultifilter-method
  (uGARCHmultifilter-class), 78
likelihood, uGARCHmultifit-method
  (uGARCHmultifit-class), 79
multifilter, 4
multifilter (multifilter-methods), 53
multifilter, ANY-method
  (multifilter-methods), 53
multifilter, ARFIMAmultifit-method
  (multifilter-methods), 53
multifilter, ARFIMAmultispec-method
  (multifilter-methods), 53
multifilter, uGARCHmultifit-method
  (multifilter-methods), 53
multifilter, uGARCHmultispec-method
  (multifilter-methods), 53
multifit, 4
multifit (multifit-methods), 54
multifit, ANY-method (multifit-methods), 54

INDEX

resume,uGARCHroll-method
(uGARCHroll-class), 84
rGARCH, 6, 7, 10, 14, 17, 22–24, 26, 29, 31, 42–48, 60, 64, 67, 70, 76, 79–82, 84, 88, 90
rGARCH-class, 57
rgarchdist, 4, 57
rugarch (rugarch-package), 3
rugarch-package, 3
setbounds<-,uGARCHspec-class, 90
setbounds<-,ANY,ANY-method (uGARCHspec-class), 90
setbounds<-,ARFIMA.spec,vector-method (ARFIMA.spec-class), 31
setbounds<-,uGARCHspec,vector-method (uGARCHspec-class), 90
setfixed<-,uGARCHspec-class, 90
setfixed<-,ANY,ANY-method (uGARCHspec-class), 90
setfixed<-,ARFIMA.spec,vector-method (ARFIMA.spec-class), 31
setfixed<-,uGARCHspec,vector-method (uGARCHspec-class), 90
setstart<-,uGARCHspec-class, 90
setstart<-,ANY,ANY-method (uGARCHspec-class), 90
setstart<-,ARFIMA.spec,vector-method (ARFIMA.spec-class), 31
setstart<-,uGARCHspec,vector-method (uGARCHspec-class), 90
show,ARFIMA_roll-method
(ARFIMA_roll-class), 26
show,ARFIMA_sim-method
(ARFIMA_sim-class), 29
show,ARFIMA.spec-method
(ARFIMA.spec-class), 31
show,uGARCHboot-method
(uGARCHboot-class), 60
show,uGARCHdistribution-method
(uGARCHdistribution-class), 64
show,uGARCHfilter-method
(uGARCHfilter-class), 67
show,uGARCHfit-method
(uGARCHfit-class), 70
show,uGARCHforecast-method
(uGARCHforecast-class), 75
show,uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
show,uGARCHmultifit-method
(uGARCHmultifit-class), 79
show,uGARCHmultiforecast-method
(uGARCHmultiforecast-class), 80
show,uGARCHmultispec-method
(uGARCHmultispec-class), 81
show,uGARCHpath-method
(uGARCHpath-class), 82
show,uGARCHroll-method
(uGARCHroll-class), 84
show,uGARCHsim-method
(uGARCHsim-class), 88
show,uGARCHspec-method
(uGARCHspec-class), 90
sigma (uGARCHfit-class), 70
sigma,ANY-method (uGARCHfit-class), 70
sigma,uGARCHfilter-method
(uGARCHfilter-class), 67
sigma,uGARCHfit-method
(uGARCHfit-class), 70
sigma,uGARCHmultifilter-method
(uGARCHmultifilter-class), 78
sigma,uGARCHmultifit-method
(uGARCHmultifit-class), 79
signbias (uGARCHfit-class), 70
signbias,ANY-method (uGARCHfit-class), 70
signbias,uGARCHfilter-method
(uGARCHfilter-class), 67
signbias,uGARCHfit-method
(uGARCHfit-class), 70
(ugARCHfit-class), 70
signbias-methods (ugARCHfit-class), 70
sp500ret, 59
ugarchbench, 5, 59
ugARCHboot, 63
ugarchboot, 4, 67, 69, 75, 78, 84, 87, 90
ugarchboot (ugarchboot-methods), 61
ugarchboot, ANY-method
(ugarchboot-methods), 61
ugarchboot, uGARCHfit-method
(ugarchboot-methods), 61
ugarchboot, uGARCHspec-method
(ugarchboot-methods), 61
ugARCHboot-class, 60
ugarchboot-methods, 61
uGARCHdistribution, 67
ugarchdistribution, 4, 63, 69, 75, 78, 84, 87, 90
ugarchdistribution
(ugarchdistribution-methods), 65
ugarchdistribution, ANY-method
(ugarchdistribution-methods), 65
ugarchdistribution, uGARCHfit-method
(ugarchdistribution-methods), 65
ugarchdistribution, uGARCHspec-method
(ugarchdistribution-methods), 65
uGARCHdistribution-class, 64
ugarchdistribution-methods, 65
uGARCHfilter, 68, 69
ugarchfilter, 63, 67, 75, 78, 84, 87, 90
ugarchfilter (ugarchfilter-methods), 69
ugarchfilter, ANY-method
(ugarchfilter-methods), 69
ugarchfilter, uGARCHspec-method
(ugarchfilter-methods), 69
uGARCHfilter-class, 67
ugarchfilter-methods, 69
uGARCHfit, 61, 62, 65, 66, 68, 71, 74, 76, 77, 82, 88, 89, 91
ugarchfit, 4, 53, 63, 67, 69, 78, 84, 87, 90, 92, 93
ugarchfit (ugarchfit-methods), 73
ugarchfit, ANY-method
(ugarchfit-methods), 73
ugarchfit, uGARCHspec-method
(ugarchfit-methods), 73
uGARCHfit-class, 70
ugarchfit-methods, 73
uGARCHforecast, 61, 65, 72, 78, 88, 91
ugarchforecast, 4, 63, 67, 69, 74, 75, 84, 87, 90
ugarchforecast
(ugarchforecast-methods), 77
ugarchforecast, ANY-method
(ugarchforecast-methods), 77
ugarchforecast, uGARCHfit-method
(ugarchforecast-methods), 77
ugarchforecast, uGARCHspec-method
(ugarchforecast-methods), 77
uGARCHforecast-class, 75
ugarchforecast-methods, 77
uGARCHmultifilter, 53, 80, 81
uGARCHmultifilter-class, 78
uGARCHmultifit, 53–55, 79, 81
uGARCHmultifit-class, 79
uGARCHmultiforecast, 56, 79–81
uGARCHmultiforecast-class, 80
uGARCHmultispec, 53–56, 79–81
uGARCHmultispec-class, 81
uGARCHpath, 82, 84
ugarchpath, 4
ugarchpath (ugarchpath-methods), 83
ugarchpath, ANY-method
(ugarchpath-methods), 83
ugarchpath, uGARCHspec-method
(ugarchpath-methods), 83
uGARCHpath-class, 82
ugarchpath-methods, 83
uGARCHroll, 87
ugarchroll, 4, 63, 67, 69, 75, 78, 84, 90
ugarchroll (ugarchroll-methods), 86
ugarchroll, ANY-method
(ugarchroll-methods), 86
ugarchroll, uGARCHspec-method
(ugarchroll-methods), 86
uGARCHroll-class, 84
ugarchroll-methods, 86
uGARCHsim, 72, 76, 82, 90, 91
ugarchsim, 4, 63, 67, 69, 75, 78, 84, 87
ugarchsim (ugarchsim-methods), 89
ugarchsim, ANY-method
(ugarchsim-methods), 89
ugarchsim, uGARCHFit-method
  (ugarchsim-methods), 89
ugARCHsim-class, 88
ugarchsim-methods, 89
uGARCHspec, 56, 61, 62, 63, 66, 69, 72, 73, 76, 77, 82, 83, 88, 91, 95
ugarchs.spec, 4, 54, 63, 67, 69, 74, 75, 84, 87, 90
ugarchs.spec (ugarchs.spec-methods), 92
ugarchs.spec, ANY-method
  (ugarchs.spec-methods), 92
uGARCHspec-class, 90
ugarchs.spec-methods, 92
uncmean (uGARCHfit-class), 70
uncmean, ANY-method (uGARCHfit-class), 70
uncmean, ARFIMAfilter-method
  (ARFIMAfilter-class), 10
uncmean, ARFIMAfit-method
  (ARFIMAfit-class), 13
uncmean, ARFIMAspec-method
  (ARFIMAspec-class), 31
uncmean, uGARCHfilter-method
  (uGARCHfilter-class), 67
uncmean, uGARCHfit-method
  (uGARCHfit-class), 70
uncmean, uGARCHspec-method
  (uGARCHspec-class), 90
uncvariance (uGARCHfit-class), 70
uncvariance, ANY, ANY, ANY, ANY, ANY-method
  (uGARCHfit-class), 70
uncvariance, missing, numeric, character, character, ANY, ANY-method
  (uGARCHfit-class), 70
uncvariance, uGARCHfilter, missing, missing, missing, missing, missing-method
  (uGARCHfilter-class), 67
uncvariance, uGARCHfit, missing, missing, missing, missing, missing-method
  (uGARCHfit-class), 70
uncvariance, uGARCHspec, missing, missing, missing, missing, missing-method
  (uGARCHspec-class), 90

VarDurTest, 95
VarTest, 97
vcov, ARFIMAfit-method
  (ARFIMAfit-class), 13
vcov, uGARCHfit-method
  (uGARCHfit-class), 70

WeekDayDummy, 5
WeekDayDummy (WeekDayDummy-methods), 99