Package: tsmarch (via r-universe)

November 19, 2024

Type Package

Title Multivariate ARCH Models

Description Feasible Multivariate Generalized Autoregressive Conditional Heteroscedasticity (GARCH) models including Dynamic Conditional Correlation (DCC), Copula GARCH and Generalized Orthogonal GARCH with Generalized Hyperbolic distribution. A review of some of these models can be found in Boudt, Galanos, Payseur and Zivot (2019) <doi:10.1016/bs.host.2019.01.001>.

Version 1.0.0

Maintainer Alexios Galanos <`alexios@4dscape.com`>

Depends R (>= 3.5.0), methods, tsmethods (>= 1.0.2)

LinkingTo Rcpp (>= 0.10.6), RcppArmadillo, RcppParallel, RcppBessel

SystemRequirements GNU make

Imports Rcpp, RcppParallel, tsgarch (>= 1.0.3), tsdistributions (>= 1.0.2), RcppBessel, Rsolnp, nloptr, numDeriv, abind, shape, Rdpack, xts, zoo, lubridate, sandwich, future.apply, future, stats, utils, data.table

License GPL-2

Encoding UTF-8

LazyData true

BugReports `https://github.com/tsmodels/tsmarch/issues`

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

ByteCompile true

URL `https://github.com/tsmodels/tsmarch`, `https://www.nopredict.com`

RdMacros Rdpack

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tstests

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libjpeg-dev libpng-dev libtiff-dev libxml2-dev libssl-dev

Repository https://tsmodels.r-universe.dev

RemoteUrl https://github.com/tsmodels/tsmarch

RemoteRef HEAD

RemoteSha 74f1ee87d6ddb879de57646f269e49f5f9ab25b6

Contents

bread.cgarch.estimate *Bread Method*

Description

Bread Method

Usage

```
## S3 method for class 'cgarch.estimate'
bread(x, ...)

## S3 method for class 'dcc.estimate'
bread(x, ...)
```

Arguments

x	an object of class “cgarch.estimate” or “dcc.estimate”.
...	not currently used.

Value

The inverse of the numerical hessian of the model.

Author(s)

Alexios Galanos

cgarch_modelspec *Generic Methods for the Copula GARCH model specification*

Description

Generic Methods for the Copula GARCH model specification

Usage

```
cgarch_modelspec(object, ...)
```

Arguments

object	a valid object.
...	additional parameters specific to the method implemented.

Value

an object of some class.

`cgarch_modelspec.tsgarch.multi_estimate`
Copula GARCH model specification

Description

Copula GARCH model specification

Usage

```
## S3 method for class 'tsgarch.multi_estimate'
cgarch_modelspec(
  object,
  dynamics = c("constant", "dcc", "adcc"),
  dcc_order = c(1, 1),
  transformation = c("parametric", "empirical", "spd"),
  copula = c("mvn", "mvt"),
  constant_correlation = c("pearson", "kendall", "spearman"),
  cond_mean = NULL,
  ...
)
```

Arguments

`object`	an object of class “tsgarch.multi_estimate”.
`dynamics`	the type of correlation dynamics to use.
`dcc_order`	the order of the dynamics in case of “dcc” or “adcc” correlation models.
`transformation`	the copula transformation to apply.
`copula`	the copula distribution.
`constant_correlation`	the constant correlation estimator to use. In the case of the “mvt” copula, only Kendall’s tau is a valid choice.
`cond_mean`	an optional matrix of the conditional mean for the series.
`...`	additional arguments passed to the `spd_modelspec` function in the case of the “spd” transformation.

Value

an object of class “cgarch.spec”.

`coef.cgarch.estimate` *Extract Model Coefficients*

Description

Extract the estimated coefficients of a model.

Usage

```
## S3 method for class 'cgarch.estimate'
coef(object, ...)

## S3 method for class 'dcc.estimate'
coef(object, ...)

## S3 method for class 'gogarch.estimate'
coef(object, ...)
```

Arguments

`object`	an estimated object from one of the models in the package.
`...`	none

Value

A numeric named vector of estimated coefficients.

Author(s)

Alexios Galanos

`combn_fast` *Fast combination of n elements, taken m at a time*

Description

Fast combination of n elements, taken m at a time

Usage

```
combn_fast(x, m)
```

Arguments

`x`	integer vector source for combinations
`m`	number of elements to choose.

Details

This is a significantly faster version of `combn`. For an integer vector x of length 1000 and m of 3 which results in a matrix of size 3 x 166,167,000, the speed improvement is about 9 times (11 secs vs 99 secs). This code is written using Armadillo C++ and 64 bit unsingned integers.

Value

a matrix with m rows.

Examples

```
all.equal(combn(1:10, 3), combn_fast(1:10,3))
```

`dcc_modelspec` *Generic Methods for the DCC GARCH model specification*

Description

Generic Methods for the DCC GARCH model specification

Usage

```
dcc_modelspec(object, ...)
```

Arguments

`object`	a valid object.
`...`	additional parameters specific to the method implemented.

Value

an object of some class.

`dcc_modelspec.tsgarch.multi_estimate` *DCC GARCH model specification*

Description

DCC GARCH model specification

Usage

```
## S3 method for class 'tsgarch.multi_estimate'
dcc_modelspec(
  object,
  dynamics = c("constant", "dcc", "adcc"),
  dcc_order = c(1, 1),
  distribution = c("mvn", "mvt"),
  cond_mean = NULL,
  ...
)
```

Arguments

`object`	an object of class “tsgarch.multi_estimate”.
`dynamics`	the type of correlation dynamics to use.
`dcc_order`	the order of the dynamics in case of “dcc” or “adcc” correlation models.
`distribution`	the multivariate distribution. If using the “mvt”, then the first stage univariate models should use the normal distribution.
`cond_mean`	an optional matrix of the conditional mean for the series.
`...`	not currently used.

Value

an object of class “dcc.spec”.

`dfft` *FFT density, distribution and quantile method*

Description

FFT density, distribution and quantile method

Usage

```
dfft(object, ...)

pfft(object, ...)

qfft(object, ...)

## S3 method for class 'gogarch.fft'
dfft(object, index = 1, ...)

## S3 method for class 'gogarch.fft'
pfft(object, index = 1, ...)
```

```
## S3 method for class 'gogarch.fft'
qfft(object, index = 1, ...)

## S3 method for class 'gogarch.fftsim'
dfft(object, index = 1, sim = 1, ...)

## S3 method for class 'gogarch.fftsim'
pfft(object, index = 1, sim = 1, ...)

## S3 method for class 'gogarch.fftsim'
qfft(object, index = 1, sim = 1, ...)
```

Arguments

`object`	an object of class “gogarch.fft” formed by calling the `tsconvolve` method.
`...`	additional parameters passed to the method.
`index`	the time index on which to generate the function.
`sim`	the simulation draw on which to generate the function (for the predict and simulate objects).

Details

These methods generate smooth approximation to the distribution functions, returning in turn a function object which can be further called with additional arguments expected from the density, distribution and quantile functions. Random number generation can be achieved through the use of uniform random variates and the quantile function (inverse CDF method).

Value

an function object which can then be called to calculate the density distribution or quantile for a particular instance (or time point),

`dji30retw` *Dow Jones 30 Constituents Closing Value log Weekly Return*

Description

Dow Jones 30 Constituents closing value weekly (Friday) 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. When data was not available for a Friday, the closest previous close for which data was available was used.

Usage

```
dji30retw
```

Format

`dji30retw`:
A data frame with 1131 rows and 30 columns, with row names being the YYYY-mm-dd date.

Source

Yahoo Finance

`estfun.cgarch.estimate` *Score Method*

Description

Score Method

Usage

```
## S3 method for class 'cgarch.estimate'
estfun(x, ...)

## S3 method for class 'dcc.estimate'
estfun(x, ...)
```

Arguments

`x`	an object of class “cgarch.estimate” or “dcc.estimate”.
`...`	not currently used.

Details

The function returns the numerical scores of the negative of the log likelihood at the optimal solution. These are used in the calculation of sandwich estimators.

Value

The score matrix

Author(s)

Alexios Galanos

estimate.cgarch.spec *Estimates a model given a specification.*

Description

Method for estimating one of the models in the package given a specification object.

Usage

```
## S3 method for class 'cgarch.spec'
estimate(
  object,
  solver = "solnp",
  control = list(trace = 0),
  return_hessian = TRUE,
  ...
)

## S3 method for class 'dcc.spec'
estimate(
  object,
  solver = "solnp",
  control = list(trace = 0),
  return_hessian = TRUE,
  ...
)

## S3 method for class 'gogarch.spec'
estimate(object, trace = FALSE, ...)
```

Arguments

`object`	an object of class “cgarch.spec”, “dcc.spec” or “gogarch.spec”.
`solver`	the solver to use for the second stage estimation of the multivariate dynamics. Valid choices are `solnp`, `nloptr` and `optim`, with the latter using the “L-BFGS-B” method. This option is inactive for the GOGARCH model which uses the default solver in package “tsgarch” for the estimation of the independent factors.
`control`	solver control parameters.
`return_hessian`	whether to calculate and return the partitioned hessian of the model (see details). Not applicable in the case of the GOGARCH model.
`...`	for the GOGARCH model, additional options passed to the `radical` function.
`trace`	whether to print tracing information for the GOGARCH model estimation.

Details

DCC and Copula Models:

Estimation assumes a 2-stage approach whereby the pre-estimated GARCH models (first stage) are used to estimate the Copula or DCC dynamics. In the case of the constant correlation Gaussian model, there are no parameters to estimate. Whilst this 2-stage approach results in a very fast estimation, the calculation of the standard errors based on the partitioned hessian is quite expensive. The user can create a futures `plan` to take advantage of parallel functionality which for large dimensional problems leads to a large speedup. Additionally, the option of not calculating the hessian ("return_hessian") is available. In that case, the scores are still calculated and the resulting standard errors will be based on the outer product of gradients.

GOGARCH Model:

The independent factors are calculated by first pre-whitening the data (PCA), selecting the number of factors, and then solving for the rotation matrix. A GARCH model is then estimated on each factor. A minimal amount of information is retained in the estimation object and most of the heavy lifting related to co-moment matrices, weighted moments etc is done through dedicated methods. The estimation method makes use of parallel processing for the independent factor GARCH models which can be setup using `plan`. Additionally, the RADICAL algorithm benefits from part parallelization which can be controlled using `setThreadOptions`.

Value

An estimated object of one of either "cgarch.estimate", "dcc.estimate" or "gogarch.estimate".

Author(s)

Alexios Galanos

`expected_shortfall` *Expected Shortfall (ES) method for predicted and simulated objects*

Description

Expected Shortfall (ES) method for predicted and simulated objects

Usage

```
expected_shortfall(object, ...)

## S3 method for class 'gogarch.predict'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'dcc.predict'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'cgarch.predict'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)
```

```
## S3 method for class 'gogarch.simulate'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'dcc.simulate'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'cgarch.simulate'
expected_shortfall(object, weights = NULL, alpha = 0.05, ...)
```

Arguments

`object`	an object generated from the predict or simulate methods.
`...`	not used.
`weights`	a vector of weights of length equal to the number of series. If NULL then an equal weight vector is used.
`alpha`	the quantile level for the value at risk. for the GOGARCH model.

Value

a matrix of the expected shortfall. For predict type input objects this will be an xts matrix with index the forecast dates.

fitted.cgarch.estimate *Extract Model Fitted Values*

Description

Extract the fitted values from an object.

Usage

```
## S3 method for class 'cgarch.estimate'
fitted(object, ...)

## S3 method for class 'cgarch.simulate'
fitted(object, ...)

## S3 method for class 'cgarch.predict'
fitted(object, ...)

## S3 method for class 'dcc.estimate'
fitted(object, ...)

## S3 method for class 'dcc.simulate'
```

```
fitted(object, ...)

## S3 method for class 'dcc.predict'
fitted(object, ...)

## S3 method for class 'gogarch.estimate'
fitted(object, ...)

## S3 method for class 'gogarch.predict'
fitted(object, ...)

## S3 method for class 'gogarch.simulate'
fitted(object, ...)
```

Arguments

`object`	an object from one of the estimated, simulated or predicted model classes in the package.
`...`	not currently used.

Value

For an estimated object an “xts” matrix of the fitted values (either zero or a constant), else for simulated or predicted objects a horizon x n_series x n_draws array of the distributional values. The array output includes attributes such as the date index (for the predicted object), horizon, n_draws and series names.

`globalindices` *Global Financial Indices Closing Value log Weekly Return*

Description

A selection of the log weekly returns (Friday) of 34 global financial indices from Yahoo Finance starting on 1991-12-13 and ending on 2024-06-21.

Usage

```
globalindices
```

Format

`globalindices`**:**
A data frame with 1698 rows and 34 columns, with row names being the YYYY-mm-dd date.

Source

Yahoo Finance

gogarch_modelspec *GOGARCH Model specification*

Description

GOGARCH Model specification

Usage

```
gogarch_modelspec(
  y,
  distribution = c("norm", "nig", "gh"),
  model = "garch",
  order = c(1, 1),
  ica = "radical",
  components = NCOL(y),
  lambda_range = c(-5, 5),
  shape_range = c(0.1, 25),
  cond_mean = NULL,
  ...
)
```

Arguments

`y`	an xts matrix of pre-filtered (residuals) stationary data.
`distribution`	a choice for the component distributions. Valid choices are normal, normal inverse gaussian or generalized hyperbolic distribution.
`model`	the GARCH model to use for each factor.
`order`	the GARCH model order.
`ica`	the Independent Component Analysis algorithm. Current only the RADICAL algorithm is available.
`components`	the number of components to extract in the pre-whitening phase,
`lambda_range`	for the generalized hyperbolic distribution, the range of the lambda parameter.
`shape_range`	for the generalized hyperbolic distribution, the range of the shape parameter (zeta).
`cond_mean`	an optional matrix of the conditional mean for the series.
`...`	additional arguments passed to the `radical` function.

Value

an object of class “gogarch.spec”.

logLik.cgarch.estimate

Extract Log-Likelihood

Description

Extract the log likelihood of the model at the estimated optimal parameter values.

Usage

```
## S3 method for class 'cgarch.estimate'
logLik(object, ...)

## S3 method for class 'dcc.estimate'
logLik(object, ...)

## S3 method for class 'gogarch.estimate'
logLik(object, ...)
```

Arguments

`object`	an estimated object from one of the models in the package.
`...`	none

Details

For all models in the package, the log-likelihood is a combination of the univariate log-likelihood and the multivariate log-likelihood. For the GOGARCH model, being an independent factor model, this is the sum of the univariate GARCH log-likelihoods plus a term for the mixing matrix. The number of parameters in the GOGARCH model reported (“df”) represents the univariate independent factor parameters plus the number of parameters in the rotation matrix U of the ICA algorithm.

Value

An object of class “logLik” with attributes for “nobs” and “df”. The latter is equal to the number of estimated parameters, whilst the former is the number of timesteps (i.e. the number of observations per series).

Author(s)

Alexios Galanos

newsimpact.cgarch.estimate

News Impact Surface

Description

News impact surface of a model

Usage

```
## S3 method for class 'cgarch.estimate'
newsimpact(object, epsilon = NULL, pair = c(1, 2), type = "correlation", ...)

## S3 method for class 'dcc.estimate'
newsimpact(object, epsilon = NULL, pair = c(1, 2), type = "correlation", ...)

## S3 method for class 'gogarch.estimate'
newsimpact(
  object,
  epsilon = NULL,
  pair = c(1, 2),
  factor = c(1, 1),
  type = "correlation",
  ...
)
```

Arguments

`object`	an object of one of the estimated model classes in the package.
`epsilon`	not used.
`pair`	the pair of series for which to generate the news impact surface.
`type`	either “correlation” or “covariance”.
`...`	additional parameters passed to the method.
`factor`	the pair of factors for which to generate the news impact surface for the GOGARCH model.

Value

An object of class “tsmarch.newsimpact” which has a plot method.

`pit.gogarch.fft` *Probability Integral Transform (PIT) for weighted FFT densities*

Description

Probability Integral Transform (PIT) for weighted FFT densities

Usage

```
## S3 method for class 'gogarch.fft'
pit(object, actual, ...)
```

Arguments

`object`	an object of class “gogarch.fft” or “gogarch.fftsim”.
`actual`	a vector of realized values representing the weighted values of the series for the period under consideration.
`...`	not used.

Value

a matrix.

`plot.dcc.estimate` *Dynamic Correlation Model Plots*

Description

Dynamic Correlation Model Plots

Usage

```
## S3 method for class 'dcc.estimate'
plot(
  x,
  y = NULL,
  series = NULL,
  index_format = "%Y",
  labels = FALSE,
  cex_labels = 0.8,
  ...
)

## S3 method for class 'cgarch.estimate'
plot(
  x,
  y = NULL,
  series = NULL,
  index_format = "%Y",
  labels = FALSE,
  cex_labels = 0.8,
  ...
)
```

Arguments

x	an object of class "dcc.estimate" or "cgarch.estimate".
y	not used
series	for the constant correlation a vector of length 2 indicating the series numbers to use for the pairwise correlation plot. For the dynamic correlation model, if NULL will include all series, else a vector of integers of the series to include.
index_format	for the dynamic correlation plot the x-axis label formatting.
labels	whether to include y-axis labels. For a large number of series it is best to leave this as FALSE.
cex_labels	the shrink factor for the y-axis labels if included.
...	additional parameters passed to the plotting functions.

Value

plots of the correlation.

`plot.tsmarch.newsimpact` *News Impact Surface Plot*

Description

Plot method for newsimpact class.

Usage

```
## S3 method for class 'tsmarch.newsimpact'
plot(x, y = NULL, ...)
```

Arguments

x	an object of class "tsmarch.newsimpact".
y	not used.
...	not currently supported.

Value

a plot of the newsimpact surface

---

predict.cgarch.estimate

Model Prediction

Description

Prediction function for estimated objects.

Usage

```
## S3 method for class 'cgarch.estimate'
predict(
  object,
  h = 1,
  nsim = 1000,
  sim_method = c("parametric", "bootstrap"),
  forc_dates = NULL,
  cond_mean = NULL,
  seed = NULL,
  ...
)

## S3 method for class 'dcc.estimate'
predict(
  object,
  h = 1,
  nsim = 1000,
  sim_method = c("parametric", "bootstrap"),
  forc_dates = NULL,
  cond_mean = NULL,
  seed = NULL,
  ...
)

## S3 method for class 'gogarch.estimate'
predict(
  object,
  h = 1,
  nsim = 1000,
  sim_method = c("parametric", "bootstrap"),
  forc_dates = NULL,
  cond_mean = NULL,
  seed = NULL,
  ...
)
```

Arguments

`object`	an estimated object from one of the models in the package.
`h`	the forecast horizon.
`nsim`	the number of sample paths to generate.
`sim_method`	white noise method for generating random sample for the multivariate distribution. The default "parametric" samples random normal variates whilst the "bootstrap" samples from the whitened innovations of the fitted model.
`forc_dates`	an optional vector of forecast dates equal to h. If NULL will use the implied periodicity of the data to generate a regular sequence of dates after the last available date in the data.
`cond_mean`	an optional matrix (h x n_series) of the predicted conditional mean for the series which is used to recenter the simulated predictive distribution.
`seed`	an integer that will be used in a call to set.seed before simulating.
`...`	no additional arguments currently supported.

Details

For the Copula GARCH model, the prediction is based on simulation due to the nonlinear transformation present in the model.

Value

A prediction class object for which methods exists for extracting relevant statistics such as the correlation, covariance, etc.

`print.summary.cgarch.estimate` *Model Estimation Summary Print method*

Description

Print method for class "summary.cgarch.estimate" or "summary.dcc.estimate"

Usage

```
## S3 method for class 'summary.cgarch.estimate'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

## S3 method for class 'summary.dcc.estimate'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)

## S3 method for class 'summary.gogarch.estimate'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)
```

Arguments

`x`	an object of class "summary.cgarch.estimate" or "summary.dcc.estimate"
`digits`	integer, used for number formatting. Optionally, to avoid scientific notation, set 'options(scipen=999)'.
`signif.stars`	logical. If TRUE, 'significance stars' are printed for each coefficient.
`...`	not currently used.

Value

Invisibly returns the original summary object.

`radical` *The Robust Accurate, Direct ICA ALgorithm (RADICAL)*

Description

The ICA algorithm of Learned-Miller (2003), is based on an efficient entropy estimator (due to Vasicek (1976)) which is robust to outliers and requires no strong characterization assumptions about the data generating process.

Usage

```
radical(
  X,
  components = NCOL(X),
  demean = TRUE,
  pca_cov = c("ML", "LW", "EWMA"),
  k = 150,
  augment = FALSE,
  replications = 30,
  sigma = 0.175,
  first_eigen = NULL,
  last_eigen = NULL,
  E = NULL,
  D = NULL,
  Z = NULL,
  K = NULL,
  L = NULL,
  seed = NULL,
  trace = FALSE,
  ...
)
```

Arguments

`X`	the data matrix (n x m) where n is the number of samples and m is the number of signals.
`components`	the number of independent components to extract.
`demean`	whether to demean the data.
`pca_cov`	the method used to calculate the covariance matrix for PCA. Options are “ML” (maximum likelihood), “LW” (Ledoit-Wolf) shrinkage method and “EWMA” (exponentially weighted moving average).
`k`	the number of angles at which to evaluate the contrast function. The contrast function will be evaluated at K evenly spaced rotations from -Pi/4 to Pi/4.
`augment`	whether to augment the data (as explained in paper). For large datasets of >10,000 points this should be set to FALSE.
`replications`	the number of replicated points for each original point if augment is set to TRUE. The larger the number of points in the data set, the smaller this value can be. For data sets of 10,000 points or more, point replication should be de-activated by setting augment to FALSE.
`sigma`	the standard deviation (noise) of the replicated points when using the augmentation option (which sets the standard deviation for the random normal generator).
`first_eigen`	This and `last_eigen` specify the range for eigenvalues that are retained, `first_eigen` is the index of largest eigenvalue to be retained. Making use of this option overwrites `components`.
`last_eigen`	the index of the last (smallest) eigenvalue to be retained and overwrites `component` argument.
`E`	(optional) Eigen vector from the PCA decomposition. If this is provided then `D` must also be provided.
`D`	(optional) Eigen values from the PCA decomposition.
`Z`	(optional) provided whitened signal matrix. If this is provided then both `K` and `L` must also be provided.
`K`	(optional) whitening matrix.
`L`	(optional) de-whitening matrix.
`seed`	the random seed for the random number generator.

`trace`	whether to print out progress information.
`...`	additional arguments to be passed to the covariance matrix calculation. For arguments passed to the “EWMA” method, it optionally takes an additional argument `lambda` which is the exponential decay parameter (default is 0.96). The “LW” takes an additional argument `shrink` which is the shrinkage parameter (default is to calculate this).

Details

Steps to the general algorithm are as follows (see P.1284 of Learned-Miller (2003) for specific details of RADICAL implementation):

1. Demean the data if required: $M = X - \mu$
2. Calculate the covariance matrix Σ using one of the methods provided.
3. Use an eigen decomposition to calculate the eigenvalues and eigenvectors of the covariance matrix: $\Sigma = EDE'$
4. Select the range of eigenvalues to retain (dimensionality reduction).
5. Calculate the whitening matrix $K = D^{-1/2}E'$ and the dewhitening matrix $L = ED^{1/2}$.
6. Whiten the data: $Z = MK'$. Unwhitening is done by $M = ZL'$.
7. Run the RADICAL algorithm to calculate the rotation matrix U, the mixing matrix: $A = U^{-1}L$ and the unmixing matrix $W = K'U$.
8. Calculate the independent components: $S = MW + \mathbf{1}\mu W$ where **1** is a matrix of ones with dimension (samples x 1).

Notice that in calculating the mixing (A) and unmixing (W) matrices we have combined the whitening (K) and un-whitening (L) matrices with the rotation matrix U.

Value

A list with the following components:

`A`	the mixing matrix
`W`	the unmixing matrix
`S`	the independent components
`U`	the rotation matrix
`K`	the whitening matrix
`L`	the dewhitening matrix
`C`	the covariance matrix
`Z`	the whitened signal
`mu`	the mean of the mixed signal (X)
`elapsed`	the time taken to run the algorithm

Note

Replications carries a significant computational cost. The algorithm is written in C++ and uses “RcppParallel” for the most expensive calculations. See `setThreadOptions` for setting the number of threads to use.

References

Learned-Miller EG, Fisher III JW (2003). "ICA using spacings estimates of entropy." *Journal of Machine Learning Research*, **4**, 1271–1295. Ledoit O, Wolf M (2004). "A well-conditioned estimator for large-dimensional covariance matrices." *Journal of Multivariate Analysis*, **88**, 365–411. Vasicek O (1976). "A test for normality based on sample entropy." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **38**(1), 54–59.

`residuals.cgarch.estimate` *Extract Model Residuals*

Description

Extract the residuals of the estimated model.

Usage

```
## S3 method for class 'cgarch.estimate'
residuals(object, standardize = FALSE, type = "standard", ...)

## S3 method for class 'dcc.estimate'
residuals(object, standardize = FALSE, type = "standard", ...)

## S3 method for class 'gogarch.estimate'
residuals(object, standardize = FALSE, type = "standard", ...)
```

Arguments

`object`	an object of class "cgarch.estimate", "dcc.estimate" or "gogarch.estimate"
`standardize`	logical. Whether to standardize the residuals by the conditional volatility (only valid for the "standard" type).
`type`	a choice of "standard" (default), "model", or "whitened" residuals. The first choice is the default and represents the residuals from the first stage univariate GARCH models. The second choice is only useful for the copula model and represents the residuals from the copula after the transformation. In the case of the DCC model this will return the standard type residuals (since they are the same). The last choice represents the whitened (ZCA based) residuals which are the standard residuals multiplied by the inverse of the square root of the conditional covariance matrix.
`...`	not currently used.

Value

An xts matrix.

Note

In the case of the GOGARCH model, the residuals are calculated as εA, where A is the mixing matrix applied to the independent component residuals. These will be equal to the residuals of the original series only if there is no dimensionality reduction.

---

simulate.cgarch.estimate *Model Simulation*

Description

Simulates paths of a model.

Usage

```
## S3 method for class 'cgarch.estimate'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  h = 100,
  burn = 0,
  Q_init = NULL,
  Z_init = NULL,
  init_method = c("start", "end"),
  cond_mean = NULL,
  sim_method = c("parametric", "bootstrap"),
  ...
)

## S3 method for class 'dcc.estimate'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  h = 100,
  burn = 0,
  Q_init = NULL,
  Z_init = NULL,
  init_method = c("start", "end"),
  cond_mean = NULL,
  sim_method = c("parametric", "bootstrap"),
  ...
)

## S3 method for class 'gogarch.estimate'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  h = 100,
  burn = 0,
  cond_mean = NULL,
  sim_method = c("parametric", "bootstrap"),
  ...
)
```

Arguments

`object`	an estimated object from one of the models in the package.
`nsim`	the number of sample paths to generate.
`seed`	an integer that will be used in a call to set.seed before simulating.
`h`	the number of time steps to simulate paths for.
`burn`	burn in. Will be discarded before returning the output.
`Q_init`	an optional array of matrices of dimension n_series x n_series x maxpq for initializing the DCC model (not relevant in the constant correlation case), where maxpq is the maximum DCC model order.
`Z_init`	an optional matrix of size maxpq x m of initialization values for the standardized innovations of the DCC model. For this copula model, care should be taken as these represent the DCC copula standardized innovations, not the univariate GARCH innovations.
`init_method`	method to initialize the DCC and GARCH recursion (unless "Q_init" and "Z_init" are not NULL in which case those take priority for those inputs). The "start" method initializes the recursion with the same values used during estimation, whereas the "end" method uses the last values of the estimated model to initialize the recursion. In the constant correlation case, only the the GARCH initialization is relevant.
`cond_mean`	an optional matrix (h x n_series) of the simulated conditional mean for the series which is used to recenter the simulated distribution.
`sim_method`	white noise method for generating random sample for the multivariate distribution. The default "parametric" samples random variates from the underlying error distribution whilst the "bootstrap" samples from the whitened innovations of the fitted model.
`...`	no additional arguments currently supported.

Details

Part of the code makes use of parallel functionality via the "future" package (see `plan`). The dimension the parallel execution operates on is the number of series (for the individual GARCH series simulation), so unless you have more than 100 series then it is possible that using a parallel back-end may actually result in slower execution due to the overhead involved.

Value

A simulation class object for which methods exists for extracting relevant statistics such as the correlation, covariance, etc.

`summary.cgarch.estimate` *Model Estimation Summary*

Description

Summary method for class “cgarch.estimate” or “dcc.estimate”.

Usage

```
## S3 method for class 'cgarch.estimate'
summary(object, vcov_type = "OP", ...)

## S3 method for class 'dcc.estimate'
summary(object, vcov_type = "OP", ...)

## S3 method for class 'gogarch.estimate'
summary(object, vcov_type = "OP", ...)
```

Arguments

`object`	an object of class “cgarch.estimate” or “dcc.estimate”.
`vcov_type`	the type of standard errors based on the vcov estimate (see `vcov`).
`...`	not currently used.

Value

A list with summary information of class “summary.cgarch.estimate” or “summary.dcc.estimate”.

`tsaggregate.cgarch.estimate` *Weighted Moments Aggregation*

Description

Calculates and returns the weighted moments of the estimated, simulated or predicted object.

Usage

```
## S3 method for class 'cgarch.estimate'
tsaggregate(object, weights = NULL, ...)

## S3 method for class 'cgarch.simulate'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)

## S3 method for class 'cgarch.predict'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)

## S3 method for class 'dcc.estimate'
tsaggregate(object, weights = NULL, ...)

## S3 method for class 'dcc.simulate'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)

## S3 method for class 'dcc.predict'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)

## S3 method for class 'gogarch.estimate'
tsaggregate(object, weights = NULL, ...)

## S3 method for class 'gogarch.predict'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)

## S3 method for class 'gogarch.simulate'
tsaggregate(object, weights = NULL, distribution = TRUE, ...)
```

Arguments

`object`	an object of one of the model classes in the package.
`weights`	an optional weighting vector. If NULL will use an equal weight vector. It is also possible to pass a time varying weighting matrix with time along the row dimension and equal to the number of time points or horizon.
`...`	not currently used.
`distribution`	for the predicted and simulated objects whether to return the simulated distribution of the weighted moments else the average.

Value

A list with the weighted moments. For an estimated object class these will be xts vectors whilst for the simulated and predicted class these will be objects of class “tsmodel.distribution” capturing the distributional uncertainty and for which a default plot method exists, unless argument “distribution” is set to FALSE.

tscokurt.gogarch.estimate

Cokurtosis Extractor

Description

Extracts the conditional cokurtosis matrices.

Usage

```
## S3 method for class 'gogarch.estimate'
tscokurt(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)

## S3 method for class 'gogarch.predict'
tscokurt(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)

## S3 method for class 'gogarch.simulate'
tscokurt(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)
```

Arguments

`object`	an object class from one of the models in the package.
`index`	the time index (integer) from which to extract a subset of the cokurtosis array rather than the whole time series.
`distribution`	whether to return the full simulated cokurtosis distribution for the predicted and simulated objects, else the average cokurtosis across each horizon.

`standardize`	whether to standardize the 4th co-moment so that it represents the cokurtosis.
`folded`	whether to return the result as a folded or unfolded array. The folded array is n_series x n_series x n_series x n_series x horizon (x simulation if predicted or simulated object). The unfolded array is a n_series x (n_series^3) x horizon array. Calculations such as weighted co-moments are based on the unfolded array using the Kronecker operator.
`...`	none

Details

The calculation of the cokurtosis array from the independent factors is very expensive in terms of memory footprint as well as computation time. While it does take advantage of multiple threads if required (see `setThreadOptions`), in the case of many series this will quickly become difficult for systems low RAM. Because of this, there is the option to extract a specific point in time output using the `index` argument.

Value

the cokurtosis (see details).

Author(s)

Alexios Galanos

`tsconvolve.gogarch.estimate`

Convolution

Description

Generates the weighted density of the GOGARCH NIG or GH model.

Usage

```
## S3 method for class 'gogarch.estimate'
tsconvolve(
  object,
  weights = NULL,
  fft_step = 0.001,
  fft_by = 1e-04,
  fft_support = c(-1, 1),
  ...
)

## S3 method for class 'gogarch.predict'
tsconvolve(
  object,
  weights = NULL,
  fft_step = 0.001,
  fft_by = 1e-04,
  fft_support = c(-1, 1),
  distribution = FALSE,
  ...
)

## S3 method for class 'gogarch.simulate'
tsconvolve(
  object,
  weights = NULL,
  fft_step = 0.001,
  fft_by = 1e-04,
  fft_support = c(-1, 1),
  distribution = FALSE,
  ...
)
```

Arguments

`object`	an object of class “gogarch.estimate”, “gogarch.predict” or “gogarch.simulate”.
`weights`	A vector of weights of length equal to the number of series. If NULL then an equal weight vector is used. A time varying matrix of weights is also allowed with the correct number of rows (time points or horizon).
`fft_step`	determines the step size for tuning the characteristic function inversion.
`fft_by`	determines the resolution for the equally spaced support given by `fft_support`.
`fft_support`	allows either a fixed support range to be given for the inversion else this is calculated (if NULL) by examining the upper and lower quantiles of each independent factor modeled. For the Generalized Hyperbolic distribution, it is not recommended to leave this as NULL since it is quite expensive to calculate the quantiles and will significantly slow down execution time.
`...`	not currently supported.
`distribution`	for the simulated and predicted object, whether to apply to each draw or on the average across draws (for the predicted object this is the analytic solution rather than the average).

Details

The Fast Fourier Transformation (FFT) is used to approximate the weighted density based on its characteristic function. The weighted density is based on the convolution of the scaled densities of the independent factors, by using the Jacobian transformation (for more details see the vignette). The returned object will be a list with the convoluted density for each time point (or each time point and draw). This can then be passed to the `dfft`, `pfft` or `qfft` methods which create smooth distributional functions.

Value

an object of class “gogarch.fft” or “gogarch.fftsim”.

tscor.cgarch.estimate *Correlation Extractor*

Description

Extracts the conditional correlation matrices.

Usage

```
## S3 method for class 'cgarch.estimate'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'cgarch.simulate'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'cgarch.predict'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'dcc.estimate'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'dcc.simulate'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'dcc.predict'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.estimate'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.predict'
tscor(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.simulate'
tscor(object, distribution = TRUE, ...)
```

Arguments

`object`	an object class from one of the models in the package.
`distribution`	whether to return the full simulated correlation distribution for the predicted and simulated objects, else the average covariance across each horizon.
`...`	none

Details

Estimation Object:

An array of correlation matrices with time as the third dimension. The returned object has attributes ‘index’ representing the datetime and ‘series’ representing the series names.

Simulation and Prediction Objects:

A 4-d array of dimensions (n_series x n_series x horizon x n_draws). If `distribution` is FALSE, then the average covariance across all draws, an array of dimensions (n_series x n_series x horizon).

Value

the correlation (see details).

Author(s)

Alexios Galanos

`tscoskew.gogarch.estimate` *Coskewness Extractor*

Description

Extracts the conditional coskewness matrices.

Usage

```
## S3 method for class 'gogarch.estimate'
tscoskew(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)

## S3 method for class 'gogarch.predict'
tscoskew(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)
```

```
## S3 method for class 'gogarch.simulate'
tscoskew(
  object,
  index = NULL,
  distribution = FALSE,
  standardize = TRUE,
  folded = TRUE,
  ...
)
```

Arguments

`object`	an object class from one of the models in the package.
`index`	the time index (integer) from which to extract a subset of the coskewness array rather than the whole time series.
`distribution`	whether to return the full simulated coskewness distribution for the predicted and simulated objects.
`standardize`	whether to standardize the 3th co-moment so that it represents the coskewness.
`folded`	whether to return the result as a folded or unfolded array. The folded array is n_series x n_series x n_series x horizon (x simulation if predicted or simulated object). The unfolded array is a n_series x (n_series^2) x horizon array. Calculations such as weighted co-moments are based on the unfolded array using the Kronecker operator.
`...`	none

Details

The calculation of the coskewness array from the independent factors is very expensive in terms of memory footprint as well as computation time. While it does take advantage of multiple threads if required (see `setThreadOptions`), in the case of many series this will quickly become difficult for systems low RAM. Because of this, there is the option to extract a specific point in time output using the `index` argument.

Value

the coskewness (see details).

Author(s)

Alexios Galanos

tscov.cgarch.estimate *Covariance Extractor*

Description

Extracts the conditional covariance matrices.

Usage

```
## S3 method for class 'cgarch.estimate'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'cgarch.simulate'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'cgarch.predict'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'dcc.estimate'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'dcc.simulate'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'dcc.predict'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.estimate'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.predict'
tscov(object, distribution = TRUE, ...)

## S3 method for class 'gogarch.simulate'
tscov(object, distribution = TRUE, ...)
```

Arguments

`object`	an object class from one of the models in the package.
`distribution`	whether to return the full simulated covariance distribution for the predicted and simulated objects, else the average covariance across each horizon.
`...`	none

Details

Estimation Object:

An array of covariance matrices with time as the third dimension. The returned object has attributes 'index' representing the datetime and 'series' representing the series names.

Simulation and Prediction Objects:
A 4-d array of dimensions (n_series x n_series x horizon x n_draws). If `distribution` is FALSE, then the average covariance across all draws, an array of dimensions (n_series x n_series x horizon).

Value

the covariance (see details).

Author(s)

Alexios Galanos

---

tsfilter.cgarch.estimate

Model Filtering

Description

Filters new data based on an already estimated model.

Usage

```
## S3 method for class 'cgarch.estimate'
tsfilter(
  object,
  y = NULL,
  newxreg = NULL,
  update = TRUE,
  cond_mean = NULL,
  ...
)

## S3 method for class 'dcc.estimate'
tsfilter(
  object,
  y = NULL,
  newxreg = NULL,
  update = TRUE,
  cond_mean = NULL,
  ...
)

## S3 method for class 'gogarch.estimate'
tsfilter(object, y = NULL, newxreg = NULL, cond_mean = NULL, ...)
```

Arguments

`object`	an object of class “cgarch.estimate” or “dcc.estimate”.
`y`	an xts matrix of new values to filter.
`newxreg`	not used in these models.
`update`	whether to update certain values using the most recent information less than the new data (see details).
`cond_mean`	an optional matrix of the filtered conditional mean values.
`...`	additional arguments for future expansion.

Details

The method filters new data and updates the object with this new information so that it can be called recursively as new data arrives. The “update” argument allows values such as the intercept matrices and transformation estimates (for the “spd” and “empirical” methods) in the dynamic case, and the constant correlation in the constant case, to use information up to and include time T, where T is the time stamp just preceding the new `y` timestamps. In this way, the filter method can be called recursively and the user can selectively choose to either use the updating scheme or use the original estimated values. Whatever the case, this ensures that there is no look-ahead bias when filtering new information.

Value

A “cgarch.estimate” or “dcc.estimate” object with updated information. All values in the object are updated with the exception of the hessian and scores which remain at their estimation set values.

`value_at_risk` *Value at Risk (VaR) method for predicted and simulated objects*

Description

Value at Risk (VaR) method for predicted and simulated objects

Usage

```
value_at_risk(object, ...)

## S3 method for class 'gogarch.predict'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'dcc.predict'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'cgarch.predict'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)
```

```
## S3 method for class 'gogarch.simulate'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'dcc.simulate'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)

## S3 method for class 'cgarch.simulate'
value_at_risk(object, weights = NULL, alpha = 0.05, ...)
```

Arguments

`object`	an object generated from the predict or simulate methods.
`...`	not used.
`weights`	a vector of weights of length equal to the number of series. If NULL then an equal weight vector is used.
`alpha`	the quantile level for the value at risk.

Value

a matrix of the value at risk. For predict type input objects this will be an xts matrix with index the forecast dates.

`vcov.cgarch.estimate` *The Covariance Matrix of the Estimated Parameters*

Description

The Covariance Matrix of the Estimated Parameters

Usage

```
## S3 method for class 'cgarch.estimate'
vcov(object, adjust = FALSE, type = c("H", "OP", "QMLE", "NW"), ...)

## S3 method for class 'dcc.estimate'
vcov(object, adjust = FALSE, type = c("H", "OP", "QMLE", "NW"), ...)
```

Arguments

`object`	an object of class “cgarch.estimate” or “dcc.estimate”.
`adjust`	logical. Should a finite sample adjustment be made? This amounts to multiplication with n/(n-k) where n is the number of observations and k the number of estimated parameters.
`type`	valid choices are “H” for using the numerical hessian for the bread, “OP” for the outer product of gradients, “QMLE” for the Quasi-ML sandwich estimator (Huber-White), and “NW” for the Newey-West adjusted sandwich estimator (a HAC estimator).

...	additional parameters passed to the Newey-West bandwidth function to determine the optimal lags.

Value

The variance-covariance matrix of the estimated parameters.

Index