Package ‘rmgarch’

February 20, 2015

Type Package

Title Multivariate GARCH models

Version 1.2-8

Date 2014-06-11

Author Alexios Ghalanos <alexios@4dscape.com>

Maintainer Alexios Ghalanos <alexios@4dscape.com>

Depends R (>= 3.0.2), methods, rugarch

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

Imports Rsolnp, MASS, parallel, Matrix, zoo, xts, Bessel, ff, shape, pcaPP, spd, Rcpp

Description
Feasible multivariate GARCH models including DCC, GO-GARCH and Copula-GARCH.

Collate rmgarch-var.R rmgarch-functions.R rmgarch-classes.R rmgarch-ica.R rmgarch-series.R rmgarch-mmean.R gogarch-classes.R gogarch-distributions.R gogarch-main.R gogarch-methods.R rdcc-classes.R rdcc-main.R rdcc-likelihoods.R rdcc-plots.R fdcc-likelihoods.R fdcc-main.R rdcc-methods.R rdcc-mdistributions.R rdcc-postestimation.R rdcc-solver.R copula-classes.R copula-distributions.R copula-likelihoods.R copula-solver.R copula-fn.R copula-transformations.R copula-main.R copula-postestimation.R copula-methods.R rmgarch-tests.R rmgarch-scenario.R zzz.R

LazyLoad yes

URL http://www.unstarched.net, https://bitbucket.org/alexiosg

License GPL-3

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-06-30 18:55:51

R topics documented:

`rmgarch-package` *The rmgarch package*

Description

The rmgarch provides a selection of multivariate GARCH models with methods for fitting, filtering, forecasting and simulation with additional support functions for working with the returned objects. At present, the Generalized Orthogonal GARCH using Independent Components Analysis (*ICA*) and Dynamic Conditional Correlation (with multivariate Normal, Laplace and Student distributions) models are fully implemented, with methods for spec, fit, filter, forecast, simulation, and rolling estimation and forecasting, as well as specialized functions to calculate and work with the weighted portfolio conditional density. The Copula-GARCH model is also implemented with the multivariate Normal and Student distributions, with dynamic (DCC) and static estimation of the correlation.

Details

Package:	rmgarch
Type:	Package
Version:	1.2-8
Date:	2014-06-11
License:	GPL
LazyLoad:	yes
Depends:	R (>= 3.0.2), metho
LinkingTo:	Rcpp (>= 0.10.6), R
Imports:	Rsolnp, MASS, par
Description: Feasible multivariate GARCH models including DCC, GO-GARCH and Copula-GARCH.	

The main package functionality, currently supports the *GO-GARCH* with ICA method, and is available through the `gogarchspec`, `gogarchfit`, `gogarchfilter`, `gogarchforecast`, `gogarchsim` and `gogarchroll` functions. The *DCC* with multivariate Normal, Laplace and Student distributions is also supported with the main functionality in `dccspec`, `dccfit`, `dccfilter`, `dccforecast`, `dccsim` and `dccroll`. The Normal and Student Copula-GARCH, with dynamic or static correlation, is implemented with the main functionality in `cgarchspec`, `cgarchfit`, `cgarchfilter`, and `cgarchsim`. Usual extractor and support methods for the multivariate GARCH models are documented in the class of the returned objects.

.

How to cite this package

Whenever using this package, please cite as

```
@Manual{Ghalanos_2014,
 author       = {Alexios Ghalanos},
 title        = {{rmgarch}: Multivariate GARCH models.},
 year         = {2014},
 note        = {R package version 1.2-8.}}
```

License

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

Author(s)

Alexios Ghalanos

References

Bollerslev, T. 1990, Modelling the coherence in short-run nominal exchange rates: a multivariate generalized ARCH model, *The Review of Economics and Statistics*, **72(3)**, 498–505.
Broda, S.A. and Paolella, M.S. 2009, CHICAGO: A Fast and Accurate Method for Portfolio Risk Calculation, *Journal of Financial Econometrics* **7(4)**, 412–436 .
Cappiello, L., Engle, R.F. and Sheppard, K. 2006, Asymmetric dynamics in the correlations of global equity and bond returns, *Journal of Financial Econometrics* **4**, 537–572.
Croux, C. and Joossens, K. 2008, Robust estimation of the vector autoregressive model by a least trimmed squares procedure, *COMPSTAT*, 489–501.
Chen, Y., Hardle, W., and Spokoiny, V. 2010, GHICA - Risk analysis with GH distributions and independent components, *Journal of Empirical Finance*, **17(2)**, 255–269.
de Athayde, G.M. and Flores Jr, R.G. 2002, On Certain Geometric Aspects of Portfolio Optimisation with Higher Moments, *mimeo*.
Engle, R.F. 2002, Dynamic conditional correlation, *Journal of Business and Economic Statistics* **20**, 339–350.
Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.
Genest, C., Ghoudi, K. and Rivest, L. 1995, A semiparametric estimation procedure of dependence parameters in multivariate families of distributions, *Biometrika*, **82**, 543–552.
Ghalanos, A., Rossi, E., and Urga, G. (2014). *Independent Factor Autoregressive Conditional Density Model*, **Econometric Reviews**.
Paolella, M.S. 2007, Intermediate Probability - A Computational Approach, *Wiley-Interscience*.
Schmidt, R., Hrycej, T. and Stutzle 2006, Multivariate distribution models with generalized hyperbolic margins, *Computational Statistics \& Data Analysis* **50(8)**, 2065–2096.

cGARCHfilter-class *class: Copula Filter Class*

Description

The class is returned by calling the function cgarchfilter.

Slots

`mfilter`: Object of class "vector" Multivariate filter list.

`model`: Object of class "vector" Model specification list.

Extends

Class "mGARCHfilter", directly. Class "GARCHfilter", by class "mGARCHfilter", distance 2. Class "rGARCH", by class "mGARCHfilter", distance 3.

Methods

coef `signature(object = "cGARCHfilter")`: The coefficient vector (see note).

fitted `signature(object = "cGARCHfilter")`: The conditional mean filtered data (xts object).

likelihood `signature(object = "cGARCHfilter")`: The joint likelihood.

rcor `signature(object = "cGARCHfilter")`: The conditional correlation array with third dimension labels the time index.

rcov `signature(object = "cGARCHfilter")`: The conditional covariance array with third dimension labels the time index.

residuals `signature(object = "cGARCHfilter")`: The model residuals (xts object).

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

sigma `signature(object = "cGARCHfilter")`: The model conditional sigma (xts object).

rshape `signature(object = "cGARCHfilter")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "cGARCHfilter")`: The multivariate distribution skew parameter(s).

Note

The 'coef' method takes additional argument 'type' with valid values 'garch' for the garch parameters, 'dcc' for the second stage parameters and by default returns all the parameters in a named vector.

Author(s)

Alexios Ghalanos

References

Joe, H. *Multivariate Models and Dependence Concepts*, 1997, Chapman \& Hall, London.
Genest, C., Ghoudi, K. and Rivest, L. *A semiparametric estimation procedure of dependence parameters in multivariate families of distributions*, 1995, Biometrika, 82, 543-552.

`cgarchfilter-methods` *function: Copula-GARCH Filter*

Description

Method for creating a Copula-GARCH filter object.

Usage

```
cgarchfilter(spec, data, out.sample = 0, filter.control = list(n.old = NULL),
spd.control = list(lower = 0.1, upper = 0.9, type = "pwm", kernel = "epanech"),
cluster = NULL, varcoef = NULL, realizedVol = NULL, ...)
```

Arguments

`spec`	A `cGARCHspec` object created by calling `cgarchspec` with fixed parameters for the coeffficients.
`data`	A multivariate xts data object or one which can be coerced to such.
`out.sample`	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
`filter.control`	Control arguments passed to the filtering routine (see note below).
`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 (remember to stop the cluster on completion).
`spd.control`	If the spd transformation was chosen in the specification, the spd.control passes its arguments to the `spdfit` routine of the `spd` package.
`varcoef`	If a VAR model was chosen, then this is the VAR coefficient matrix which must be supplied. No checks are done on its dimension or correctness so it is up to the user to perform the appropriate checks.
`realizedVol`	Required xts matrix for the realGARCH model.
`...`	.

Value

A `cGARCHfilter` object containing details of the Copula-GARCH filter and sharing most of the methods of the `cGARCHfit` class.

Note

The 'n.old' option in the `filter.control` argument is key in replicating conditions of the original fit. That is, if you want to filter a dataset consisting of an expanded dataset (versus the original used in fitting), but want to use the same assumptions as the original dataset then the 'n.old' argument denoting the original number of data points passed to the `cgarchfit` function must be provided. This is then used to ensure that some calculations which make use of the full dataset (unconditional starting values for the garch filtering, the dcc model and the copula transformation methods) only use the first 'n.old' points thus replicating the original conditions making filtering appropriate for rolling 1-ahead forecasting.
For extensive examples look in the 'rmgarch.tests' folder.

Author(s)

Alexios Ghalanos

`cGARCHfit-class` *class: Copula Fit Class*

Description

The class is returned by calling the function `cgarchfit`.

Slots

`mfit`**:** Object of class `"vector"` Multivariate fit list.

`model`**:** Object of class `"vector"` Model specification list.

Extends

Class `"mGARCHfit"`, directly. Class `"GARCHfit"`, by class "mGARCHfit", distance 2. Class `"rGARCH"`, by class "mGARCHfit", distance 3.

Methods

coef `signature(object = "cGARCHfit")`: The coefficient vector (see note).

fitted `signature(object = "cGARCHfit")`: The conditional mean fitted data (xts object).

likelihood `signature(object = "cGARCHfit")`: The joint likelihood.

rcor `signature(object = "cGARCHfit")`: The conditional correlation array with third dimension labels the time index.

rcov `signature(object = "cGARCHfit")`: The conditional covariance array with third dimension labels the time index.

rshape `signature(object = "cGARCHfit")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "cGARCHfit")`: The multivariate distribution skew parameter(s).

residuals `signature(object = "cGARCHfit")`: The model residuals (xts object).

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

sigma `signature(object = "cGARCHfit")`: The model conditional sigma (xts object).

Note

The 'coef' method takes additional argument 'type' with valid values 'garch' for the garch parameters, 'dcc' for the second stage parameters and by default returns all the parameters in a named vector.

Author(s)

Alexios Ghalanos

References

Joe, H. *Multivariate Models and Dependence Concepts*, 1997, Chapman \& Hall, London.
Genest, C., Ghoudi, K. and Rivest, L. *A semiparametric estimation procedure of dependence parameters in multivariate families of distributions*, 1995, Biometrika, 82, 543-552.

`cgarchfit-methods` *function: Copula-GARCH Fit*

Description

Method for creating a Copula-GARCH fit object.

Usage

```
cgarchfit(spec, data, spd.control = list(lower = 0.1, upper = 0.9, type = "pwm",
kernel = "epanech"), fit.control = list(eval.se = TRUE, stationarity = TRUE,
scale = FALSE), solver = "solnp", solver.control = list(), out.sample = 0,
cluster = NULL, fit = NULL, VAR.fit = NULL, realizedVol = NULL,...)
```

Arguments

`spec`	A `cGARCHspec` A `cGARCHspec` object created by calling `cgarchspec`.
`data`	A multivariate xts data object or one which can be coerced to such.
`out.sample`	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
`solver`	Either "nlminb", "solnp", "gosolnp" or "lbfgs". It can also optionally be a vector of length 2 with the first solver being used for the first stage univariate GARCH estimation (in which case the option of "hybrid" is also available).
`solver.control`	Control arguments list passed to optimizer.
`fit.control`	Control arguments passed to the fitting routine. The 'eval.se' option determines whether standard errors are calculated (see details below). The 'scale' option is for the first stage univariate GARCH fitting routine.
`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 (remember to stop the cluster on completion).

`fit`	(optional) A previously estimated univariate `uGARCHmultifit` object (see details).
`VAR.fit`	(optional) A previously estimated VAR list returned from calling the `varxfit` function.
`spd.control`	If the spd transformation was chosen in the specification, the spd.control passes its arguments to the `spdfit` routine of the `spd` package.
`realizedVol`	Required xts matrix for the realGARCH model.
`...`	.

Details

The Copula-GARCH models implemented can either be time-varying of DCC variety else static. The multivariate Normal and Student distributions are used in the construction of the copulas, and 3 transformation methods are available (parametric, semi-parametric, and empirical). For the semi-parametric case the 'spd' package of the author is available to download from CRAN and fits a Gaussian kernel in the interior and gpd distribution for the tails (see that package for more details). The static copula allows for the estimation of the correlation matrix either by Maximum Likelihood or the Kendall method for the multivariate Student.
Note that the 'cgarchfit' method will assign to the global environment the `uGARCHmultifit` once that is estimated in order to allow the routine to be restarted should something go wrong (it should show up as '.fitlist').

Value

A `cGARCHfit` Object containing details of the Copula-GARCH fit.

Note

There is no check on the VAR.fit list passed to the method so particular care should be exercised so that the same data used in the fitting routine is also used in the VAR fit routine. This must have been called with the option `postpad` 'constant'. The ability to pass this list of the pre-calculated VAR model is particularly useful when comparing different models (such as DCC GARCH, GO GARCH etc) using the same dataset and VAR method (i.e. the same first stage conditional mean filtration). Though the classical VAR estimation is very fast and may not require this extra step, the robust method is slow and therefore benefits from calculating this only once.
For extensive examples look in the 'rmgarch.tests' folder.

Author(s)

Alexios Ghalanos

cGARCHsim-class *class: Copula Simulation Class*

Description

The class is returned by calling the function `cgarchsim`.

Slots

`msim`: Object of class `"vector"` Multivariate simulation list.

`model`: Object of class `"vector"` Model specification list.

Extends

Class `"mGARCHsim"`, directly. Class `"GARCHsim"`, by class "mGARCHsim", distance 2. Class `"rGARCH"`, by class "mGARCHsim", distance 3.

Methods

fitted `signature(object = "cGARCHsim")`: The simulated conditional returns matrix given. Takes optional argument "sim" indicating the simulation run to return (from the m.sim option of the `cgarchsim` method.

sigma `signature(object = "cGARCHfit")`: The simulated conditional sigma matrix given. Takes optional argument "sim" indicating the simulation run to return (from the m.sim option of the `cgarchsim` method.

rcor `signature(object = "cGARCHsim")`: The simulated conditional correlation array (for DCC type). Takes optional argument "sim" indicating the simulation run to return (from the m.sim option of the `cgarchsim` method

rcov `signature(object = "cGARCHsim")`: The simulated conditional covariance array. Takes optional argument "sim" indicating the simulation run to return (from the m.sim option of the `cgarchsim` method.

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

Author(s)

Alexios Ghalanos

References

Joe, H. *Multivariate Models and Dependence Concepts*, 1997, Chapman \& Hall, London.
Genest, C., Ghoudi, K. and Rivest, L. *A semiparametric estimation procedure of dependence parameters in multivariate families of distributions*, 1995, Biometrika, 82, 543-552.

`cgarchsim-methods` *function: Copula-GARCH Simulation*

Description

Method for creating a Copula-GARCH simulation object.

Usage

```
cgarchsim(fit, n.sim = 1000, n.start = 0, m.sim = 1,
startMethod = c("unconditional", "sample"), presigma = NULL, preresiduals = NULL,
prereturns = NULL, preR = NULL, preQ = NULL, preZ = NULL, rseed = NULL,
mexsimdata = NULL, vexsimdata = NULL, cluster = NULL, only.density = FALSE,
prerealized = NULL, ...)
```

Arguments

`fit`	A `cGARCHfit` object created by calling `cgarchfit`.
`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. This is mostly related to the univariate GARCH dynamics.
`presigma`	Allows the starting sigma values to be provided by the user for the univariate GARCH dynamics.
`prereturns`	Allows the starting return data to be provided by the user for the conditional mean simulation.
`preresiduals`	Allows the starting residuals to be provided by the user and used in the GARCH dynamics simulation.
`preR`	Allows the starting correlation to be provided by the user and mostly useful for the static copula.
`preQ`	Allows the starting 'DCC-Q' value to be provided by the user and though unnecessary for the first 1-ahead simulation using the "sample" option in the `startMethod`, this is key to obtaining a rolling n-ahead forecast type simulation (see details below).
`preZ`	Allows the starting transformed standardized residuals (used in the DCC model) to be provided by the user and though unnecessary for the first 1-ahead simulation using the "sample" option in the `startMethod`, this is key to obtaining a rolling n-ahead forecast type simulation (see details below).
`rseed`	Optional seeding value(s) for the random number generator. This should be of length equal to m.sim.

mexsimdata	A list (equal to the number of asset) of matrices of simulated external regressor-in-mean data with row length equal to n.sim + n.start. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed to be zero.
vexsimdata	A list (equal to the number of asset) of matrices of simulated external regressor-in-variance data with row length equal to n.sim + n.start. If the fit object contains external regressors in the variance equation, this must be provided else will be assumed to be 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 (remember to stop the cluster on completion).
only.density	Whether to return only the simulated returns (discrete time approximation to the multivariate density). This is sometimes useful in order to control memory management for large simulations not requiring any other information.
prerealized	Allows the starting realized volatility values to be provided by the user for the univariate GARCH dynamics.
...	.

Details

Since there is no explicit forecasting routine, the user should use this method for incrementally building up n-ahead forecasts by simulating 1-ahead, obtaining the means of the returns, sigma, Rho etc and feeding them to the next round of simulation as starting values. The 'rmgarch.tests' folder contains specific examples which illustrate this particular point.

Value

A `cGARCHsim` object containing details of the Copula-GARCH simulation.

Author(s)

Alexios Ghalanos

References

Joe, H. *Multivariate Models and Dependence Concepts*, 1997, Chapman \& Hall, London.
Genest, C., Ghoudi, K. and Rivest, L. *A semiparametric estimation procedure of dependence parameters in multivariate families of distributions*, 1995, Biometrika, 82, 543-552.

`cGARCHspec-class` *class: Copula Specification Class*

Description

The class is returned by calling the function `cgarchspec`.

Slots

`model`**:** Object of class `"vector"` The multivariate model specification

`umodel`**:** Object of class `"uGARCHmultispec"` The univariate model specification.

Extends

Class `"mGARCHspec"`, directly. Class `"GARCHspec"`, by class "mGARCHspec", distance 2. Class `"rGARCH"`, by class "mGARCHspec", distance 3.

Methods

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

setfixed<- `signature(object = "cGARCHspec", value = "vector")`: Set fixed second stage parameters.

setstart<- `signature(object = "cGARCHspec", value = "vector")`: Set starting second stage parameters.

Author(s)

Alexios Ghalanos

References

Joe, H. *Multivariate Models and Dependence Concepts*, 1997, Chapman \& Hall, London.
Genest, C., Ghoudi, K. and Rivest, L. *A semiparametric estimation procedure of dependence parameters in multivariate families of distributions*, 1995, Biometrika, 82, 543-552.

cgarchspec-methods *function: Copula-GARCH Specification*

Description

Method for creating a Copula-GARCH specification object prior to fitting.

Usage

```
cgarchspec(uspec, VAR = FALSE, robust = FALSE, lag = 1, lag.max = NULL,
lag.criterion = c("AIC", "HQ", "SC", "FPE"), external.regressors = NULL,
robust.control = list(gamma = 0.25, delta = 0.01, nc = 10, ns = 500),
dccOrder = c(1, 1), asymmetric = FALSE,
distribution.model = list(copula = c("mvnorm", "mvt"),
method = c("Kendall", "ML"), time.varying = FALSE,
transformation = c("parametric", "empirical", "spd")),
start.pars = list(), fixed.pars = list())
```

Arguments

- `uspec` — A `uGARCHmultispec` object created by calling `multispec` on a list of univariate GARCH specifications.
- `VAR` — Whether to fit a VAR model for the conditional mean.
- `robust` — Whether to use the robust version of VAR.
- `lag` — The VAR lag.
- `lag.max` — The maximum VAR lag to search for best fit.
- `lag.criterion` — The criterion to use for choosing the best lag when lag.max is not NULL.
- `external.regressors` — Allows for a matrix of common pre-lagged external regressors for the VAR option.
- `robust.control` — The tuning parameters to the robust regression including the proportion to trim ("gamma"), the critical value for reweighted estimator ("delta"), the number of subsets ("ns") and the number of C-steps ("nc".
- `dccOrder` — The DCC autoregressive order.
- `asymmetric` — Whether to include an asymmetry term to the DCC model (thus estimating the aDCC).
- `distribution.model` — The Copula distribution model. Currently the multivariate Normal and Student Copula are supported.
- `time.varying` — Whether to fit a dynamic DCC Copula.
- `transformation` — The type of transformation to apply to the marginal innovations of the GARCH fitted models. Supported methods are parametric (Inference Function of Margins), empirical (Pseudo ML), and Semi-Parametric using a kernel interior and GPD tails (via the 'spd' package).

start.pars (optional) Starting values for the DCC parameters (starting values for the univariate garch specification should be passed directly via the 'uspec' object).

fixed.pars (optional) Fixed DCC parameters.

Details

The transformation method allows for parametric (Inference-Functions for Margins), empirical (Pseudo-Likelihood) and semi-parametric (via the spd package).
When the Student Copula is jointly estimated with student margins having so that a common shape parameter is obtained, this results in the multivariate Student distribution. When estimating the Student Copula with disparate margins, a meta-student distribution is obtained. Additionally, the correlation parameter in the static Student Copula may be estimated either by Kendall's tau transformation or Maximum Likelihood.
The `robust` option allows for a robust version of VAR based on the multivariate Least Trimmed Squares Estimator described in Croux and Joossens (2008).

Value

A `cGARCHspec` object containing details of the Copula-GARCH specification.

Author(s)

Alexios Ghalanos

`cordist` *A Correlation Distance Measure*

Description

Given an array of correlation snapshots in time, returns a matrix of some rolling distance measure on the correlations.

Usage

```
cordist(R, distance = c("ma","ms","meda","meds","eigen", "cmd"), n = 25,
plot = TRUE, dates = NULL, title = NULL)
```

Arguments

`R`	An array of correlations.
`distance`	The measure to use to capture the change between 2 correlation matrices (see details).
`n`	The distance between 2 correlation matrices.
`plot`	Whether to create a heatmap plot of the result.
`dates`	A `POSIXct` vector of dates to use for the heatmap (recommend to supply).
`title`	Title for the heatmap plot.

Details

This function provides for a visualization of dynamic correlation distance between periods with a number of plausible measures including "ma" (mean absolute), "ms" (mean squared), "meda" (median absolute), "meds" (median squared) "eigen" (largest eigenvalue difference) and "cmd" (correlation matrix distance). See the references for more details.

Value

A symmetric matrix of the rolling distance measure for each period.

Author(s)

Alexios Ghalanos

References

Munnix, M. C., Shimada, T., Schafer, R., Leyvraz, F., Seligman, T. H., Guhr, T., & Stanley, H. E. (2012). Identifying states of a financial market. *Scientific Reports 2*.
Herdin, M., Czink, N., Ozcelik, H., & Bonek, E. (2005). Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels. *Vehicular Technology Conference, 2005, IEEE 61st*, **1**, 136–140.

`DCCfilter-class` *class: DCC Filter Class*

Description

The class is returned by calling the function `dccfilter`.

Slots

`mfilter`**:** Object of class `"vector"`. Multivariate filter list.

`model`**:** Object of class `"vector"`. Model specification list.

Extends

Class `"mGARCHfilter"`, directly. Class `"GARCHfilter"`, by class "mGARCHfilter", distance 2. Class `"rGARCH"`, by class "mGARCHfilter", distance 3.

Methods

coef `signature(object = "DCCfilter")` The coefficient vector (see note).

likelihood `signature(object = "DCCfilter")`: The joint likelihood.

rshape `signature(object = "DCCfilter")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "DCCfilter")`: The multivariate distribution skew parameter(s).

fitted `signature(object = "DCCfilter")`: The filtered conditional mean xts object.

sigma `signature(object = "DCCfilter")`: The filtered conditional sigma xts object.

residuals `signature(object = "DCCfilter")`: The filtered conditional mean residuals xts object.

plot `signature(x = "DCCfilter", y = "missing")`: Plot method, given additional arguments 'series' and 'which'.

infocriteria `signature(object = "DCCfilter")`: Information criteria.

rcor `signature(object = "DCCfilter")`: The filtered dynamic conditional correlation array given additional argument 'type' (either "R" for the correlation else will return the "Q" matrix). The third dimension label of the array gives the time index (from which it is then possible to construct pairwise xts objects for example).

rcov `signature(object = "DCCfilter")`: The filtered dynamic conditional covariance array. The third dimension label of the array gives the time index (from which it is then possible to construct pairwise xts objects for example).

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

nisurface `signature(object = "DCCfilter")`: The news impact surface plot given additional arguments 'type' with either "cov" or "cor" (for the covariance and correlation news impact respectively), 'pair' denoting the asset pair (defaults to c(1,2)), 'plot' (logical) and 'plot.type' with a choice of either "surface" or "contour".

Note

The 'coef' method takes additional argument 'type' with valid values 'garch' for the univariate garch parameters, 'dcc' for the second stage dcc parameters and by default returns all the parameters in a named vector.

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

`dccfilter-methods` *function: DCC-GARCH Filter*

Description

Method for creating a DCC-GARCH filter object.

Usage

```
dccfilter(spec, data, out.sample = 0, filter.control = list(n.old = NULL),
cluster = NULL, varcoef = NULL, realizedVol = NULL, ...)
```

Arguments

spec	A DCCspec object created by calling dccspec with fixed parameters for the coefficients.
data	A multivariate data object of class xts, or one which can be coerced to such.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
filter.control	Control arguments passed to the filtering routine (see note.
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 (remember to stop the cluster on completion).
varcoef	If a VAR model was chosen, then this is the VAR coefficient matrix which must be supplied. No checks are done on its dimension or correctness so it is up to the user to perform the appropriate checks.
realizedVol	Required xts matrix for the realGARCH model.
...	.

Value

A DCCfilter object containing details of the DCC-GARCH filter.

Note

The 'n.old' option in the filter.control argument is key in replicating conditions of the original fit. That is, if you want to filter a dataset consisting of an expanded dataset (versus the original used in fitting), but want to use the same assumptions as the original dataset then the 'n.old' argument denoting the original number of data points passed to the dccfit function must be provided. This is then used to ensure that some calculations which make use of the full dataset (unconditional starting values for the garch filtering and the dcc model) only use the first 'n.old' points thus replicating the original conditions making filtering appropriate for rolling 1-ahead forecasting.
For extensive examples look in the 'rmgarch.tests' folder.

Author(s)

Alexios Ghalanos

DCCfit-class *class: DCC Fit Class*

Description

The class is returned by calling the function dccfit.

Slots

mfit: Object of class "vector" Multivariate filter list.

model: Object of class "vector" Model specification list.

Extends

Class "`mGARCHfit`", directly. Class "`GARCHfit`", by class "mGARCHfit", distance 2. Class "`rGARCH`", by class "mGARCHfit", distance 3.

Methods

coef `signature(object = "DCCfit")` The coefficient vector (see note).

likelihood `signature(object = "DCCfit")`: The joint likelihood.

rshape `signature(object = "DCCfit")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "DCCfit")`: The multivariate distribution skew parameter(s).

fitted `signature(object = "DCCfit")`: The fitted conditional mean xts object.

sigma `signature(object = "DCCfit")`: The fitted conditional GARCH sigma xts object.

residuals `signature(object = "DCCfit")`: The fitted conditional mean residuals xts object.

plot `signature(x = "DCCfit", y = "missing")`: Plot method, given additional arguments 'series' and 'which'.

infocriteria `signature(object = "DCCfit")`: Information criteria.

rcor `signature(object = "DCCfit")`: The fitted dynamic conditional correlation array given additional argument 'type' (either "R" for the correlation else will return the Q matrix). The third dimension label of the array gives the time index (from which it is then possible to construct pairwise xts objects for example).

rcov `signature(object = "DCCfit")`: The fitted dynamic conditional covariance array. The third dimension label of the array gives the time index (from which it is then possible to construct pairwise xts objects for example).

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

nisurface `signature(object = "DCCfit")`: The news impact surface plot given additional arguments 'type' with either "cov" or "cor" (for the covariance and correlation news impact respectively), 'pair' (defaults to c(1,2)), 'plot' (logical) and 'plot.type' with a choice of either "surface" or "contour".

Note

The 'coef' method takes additional argument 'type' with valid values 'garch' for the univariate garch parameters, 'dcc' for the second stage dcc parameters and by default returns all the parameters in a named vector.

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

dccfit-methods *function: DCC-GARCH Fit*

Description

Method for creating a DCC-GARCH fit object.

Usage

```
dccfit(spec, data, out.sample = 0, solver = "solnp", solver.control = list(),
fit.control = list(eval.se = TRUE, stationarity = TRUE, scale = FALSE),
cluster = NULL, fit = NULL, VAR.fit = NULL, realizedVol = NULL, ...)
```

Arguments

`spec`	A `DCCspec` object created by calling `dccspec`.
`data`	A multivariate data object of class xts or one which can be coerced to such.
`out.sample`	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
`solver`	Either "nlminb", "solnp", "gosolnp" or "lbfgs". It can also optionally be a vector of length 2 with the first solver being used for the first stage univariate GARCH estimation (in which case the option of "hybrid" is also available).
`solver.control`	Control arguments list passed to optimizer.
`fit.control`	Control arguments passed to the fitting routine. The 'eval.se' option determines whether standard errors are calculated (see details below). The 'stationarity' option is for the univariate stage GARCH fitting routine, whilst for the second stage DCC this is imposed by design. The 'scale' option is also for the first stage univariate GARCH fitting routine.
`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 (remember to stop the cluster on completion).
`fit`	(optional) A previously estimated univariate `uGARCHmultifit` object (see details).
`VAR.fit`	(optional) A previously estimated VAR object returned from calling the `varxfit` function.
`realizedVol`	Required xts matrix for the realGARCH model.
`...`	.

Details

The 2-step DCC estimation fits a GARCH-Normal model to the univariate data and then proceeds to estimate the second step based on the chosen multivariate distribution. Because of this 2-step approach, standard errors are expensive to calculate and therefore the use of parallel functionality, built into both the fitting and standard error calculation routines is key. The switch to turn off

the calculation of standard errors through the 'fit.control' option could be quite useful in rolling estimation such as in the `dccroll` routine.
The optional 'fit' argument allows to pass your own `uGARCHmultifit` object instead of having the routine estimate it. This is very useful in cases of multiple use of the same fit and problems in convergence which might require a more hands on approach to the univariate fitting stage. However, it is up to the user to ensure consistency between the 'fit' and supplied 'spec'.

Value

A `DCCfit` object containing details of the DCC-GARCH fit.

Note

There is no check on the VAR.fit list passed to the method so particular care should be exercised so that the same data used in the fitting routine is also used in the VAR fit routine. This this must have been called with the option `postpad` 'constant'. The ability to pass this list of the pre-calculated VAR model is particularly useful when comparing different models (such as copula-GARCH, GO-GARCH etc) using the same dataset and VAR method (i.e. the same first stage conditional mean filtration). Though the classical VAR estimation is very fast and may not require this extra step, the robust method is slow and therefore benefits from calculating this only once.
For extensive examples look in the 'rmgarch.tests' folder.

Author(s)

Alexios Ghalanos

`DCCforecast-class` *class: DCC Forecast Class*

Description

The class is returned by calling the function `dccforecast`.

Slots

`mforecast`: Object of class `"vector"` Multivariate forecast list.

`model`: Object of class `"vector"` Model specification list.

Extends

Class `"mGARCHforecast"`, directly. Class `"GARCHforecast"`, by class "mGARCHforecast", distance 2. Class `"rGARCH"`, by class "mGARCHforecast", distance 3.

Methods

rshape `signature(object = "DCCforecast")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "DCCforecast")`: The multivariate distribution skew parameter(s).

fitted `signature(object = "DCCforecast")`: The conditional mean forecast array of dimensions n.ahead x n.assets by (n.roll+1). The thirds dimension of the array has the T+0 index label.

sigma `signature(object = "DCCforecast")`: The conditional sigma forecast array of dimensions n.ahead x n.assets by (n.roll+1). The thirds dimension of the array has the T+0 index label.

plot `signature(x = "DCCforecast", y = "missing")`: Plot method, given additional arguments 'series' and 'which'.

rcor `signature(object = "DCCforecast")`: The forecast dynamic conditional correlation list of arrays of length (n.roll+1), with each array of dimensions n.assets x n.assets x n.ahead. The method takes on one additional argument 'type' (either "R" for the correlation else will return the DCC Q matrix).

rcov `signature(object = "DCCforecast")`: The forecast dynamic conditional correlation list of arrays of length (n.roll+1), with each array of dimensions n.assets x n.assets x n.ahead.

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

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

`dccforecast-methods` *function: DCC-GARCH Forecast*

Description

Method for creating a DCC-GARCH forecast object.

Usage

```
dccforecast(fit, n.ahead = 1, n.roll = 0,
external.forecasts = list(mregfor = NULL, vregfor = NULL), cluster = NULL, ...)
```

Arguments

`fit`	A `DCCfit` object created by calling `dccfit`.
`n.ahead`	The forecast horizon.
`n.roll`	The no. of rolling forecasts to create beyond the first one (see details).
`external.forecasts`	A list with forecasts for the external regressors in the mean and/or variance equations if specified (see details).
`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 (remember to stop the cluster on completion).
`...`	.

Details

When using `n.roll`, it is assumed that `dccfit` was called with argument 'out.sample' being large enough to cover n-rolling forecasts.
When n.roll = 0, all forecasts are based on an unconditional n-ahead forecast routine based on the approximation method described in ENGLE and SHEPPARD (2001) paper (see reference below). If any external regressors are present, then the user must pass in their unconditional forecasts in the 'external.forecasts' list, as matrices with dimensions equal to n.ahead x n.assets. This assumes that the univariate GARCH specifications share common external regressors (this may change in the future).
When n.roll>0 and n.ahead = 1, then this is a pure rolling forecast based on the available out.sample data provided for in the call to the fit routine. It is also assumed that if any external regressors were passed to the fit routine that they contained enough values to cover the out.sample period so that they could be used in this forecast scenario.
The case of n.roll > 0 AND n.ahead > 1 is not implemented.

Value

A `DCCforecast` object containing details of the DCC-GARCH forecast.

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

DCCroll-class *class: DCC Roll Class*

Description

The class is returned by calling the function `dccroll`.

Slots

`mforecast`: Object of class `"vector"` Multivariate forecast list.

`model`: Object of class `"vector"` Model specification list.

Extends

Class `"mGARCHroll"`, directly. Class `"GARCHroll"`, by class "mGARCHroll", distance 2. Class `"rGARCH"`, by class "mGARCHroll", distance 3.

Methods

coef `signature(object = "DCCroll")`: The coefficient array across the rolling estimations with a T+0 3rd dimension index label.

fitted `signature(object = "DCCroll")`: The conditional mean forecast xts object (with the actual T+i forecast dates as index).

likelihood `signature(object = "DCCroll")`: The log-likelihood across rolling estimations.

plot `signature(x = "DCCroll", y = "missing")`: Plot method, given additional arguments 'series' and 'which'.

rcor `signature(object = "DCCroll")`: The forecast dynamic conditional correlation array, with the T+i forecast index in the 3rd dimension label. Optional argument 'type' determines whether to return "R" for the correlation else will the DCC Q matrix.

rcov `signature(object = "DCCroll")`: The forecast dynamic conditional covariance array, with the T+i forecast index in the 3rd dimension label.

rshape `signature(object = "DCCroll")`: The multivariate distribution shape parameter(s).

rskew `signature(object = "DCCroll")`: The multivariate distribution skew parameter(s).

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

sigma `signature(object = "DCCroll")`: The conditional sigma forecast xts object (with the actual T+i forecast dates as index).

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

dccroll-methods *function: DCC-GARCH Rolling Forecast*

Description

Method for creating a DCC-GARCH rolling forecast object.

Usage

```
dccroll(spec, data, n.ahead = 1, forecast.length = 50, refit.every = 25,
n.start = NULL, refit.window = c("recursive", "moving"), window.size = NULL,
solver = "solnp", solver.control = list(),
fit.control = list(eval.se = TRUE, stationarity = TRUE, scale = FALSE),
cluster = NULL, save.fit = FALSE, save.wdir = NULL, realizedVol = NULL, ...)
```

Arguments

`spec`	A `DCCspec` object with fixed parameters.
`data`	A multivariate xts dataset or one which can be coerced to such.
`n.ahead`	The number of periods to forecast.
`forecast.length`	The length of the total forecast for which out of sample data from the dataset will be used for testing.
`n.start`	Instead of forecast.length, this determines the starting point in the dataset from which to initialize the rolling forecast.
`refit.every`	Determines every how many periods the model is re-estimated.
`refit.window`	Whether the refit is done on an expanding window including all the previous data or a moving window where all previous data is used for the first estimation and then moved by a length equal to refit.every (unless the window.size option is used instead).
`window.size`	If not NULL, determines the size of the moving window in the rolling estimation, which also determines the first point used.
`solver`	The solver to use.
`fit.control`	Control parameters parameters passed to the fitting function.
`solver.control`	Control parameters passed to the solver.
`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).
`save.fit`	Whether to save the fitted objects of class `DCCfit` during the estimation of each (“refit.every”). If true, the directory to save must be provided. The function will not save this by default for reasons of memory management, but can save it as an “.rda” file in the user’s chosen directory for further analysis.

save.wdir	If "save.fit" is true, the directory in which to save the DCCfit objects (1 for each "refit.every").
realizedVol	Required xts matrix for the realGARCH model.
...	.

Value

A DCCroll object containing details of the DCC-GARCH rolling forecast.

Author(s)

Alexios Ghalanos

DCCsim-class *class: DCC Forecast Class*

Description

The class is returned by calling the function dccsim.

Slots

msim: Object of class "vector" Multivariate simulation list.

model: Object of class "vector" Model specification list.

Extends

Class "mGARCHsim", directly. Class "GARCHsim", by class "mGARCHsim", distance 2. Class "rGARCH", by class "mGARCHsim", distance 3.

Methods

fitted signature(object = "DCCsim"): The conditional mean simulated data matrix given additional argument 'sim' denoting the simulation run (m.sim) to return values for.

rcor signature(object = "DCCsim"): The simulated dynamic conditional correlation array given additional arguments 'sim' denoting the simulation run (m.sim) to return values for, and 'type' (either "R" for the correlation else will return the Q matrix).

rcov signature(object = "DCCsim"): The simulated dynamic conditional covariance array given additional argument 'sim' denoting the simulation run (m.sim) to return values for.

sigma signature(object = "DCCsim"): The univariate simulated conditional sigma matrix given additional argument 'sim' (m.sim) denoting the simulation run to return values for.

show signature(object = "DCCsim"): Summary.

Author(s)

Alexios Ghalanos

References

Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

`dccsim-methods` *function: DCC-GARCH Simulation*

Description

Method for creating a DCC-GARCH simulation object.

Usage

```
dccsim(fitORspec, n.sim = 1000, n.start = 0, m.sim = 1,
startMethod = c("unconditional", "sample"), presigma = NULL, preresiduals = NULL,
prereturns = NULL, preQ = NULL, preZ = NULL, Qbar = NULL, Nbar = NULL,
rseed = NULL, mexsimdata = NULL, vexsimdata = NULL, cluster = NULL,
VAR.fit = NULL, prerealized = NULL, ...)
```

Arguments

`fitORspec`	A `DCCspec` or `DCCfit` object created by calling either `dccspec` with fixed parameters or `dccfit`.
`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 (for the dispatch method using a specification, “sample” is not relevant).
`presigma`	Allows the starting sigma values to be provided by the user for the univariate GARCH dynamics.
`prereturns`	Allows the starting return data to be provided by the user for the conditional mean simulation.
`preresiduals`	Allows the starting residuals to be provided by the user and used in the GARCH dynamics simulation.
`preQ`	Allows the starting ‘DCC-Q’ value to be provided by the user and though unnecessary for the first 1-ahead simulation using the “sample” option in the `startMethod`, this is key to obtaining a rolling n-ahead forecast type simulation (see details below).
`preZ`	Allows the starting standardized residuals to be provided by the user and though unnecessary for the first 1-ahead simulation using the “sample” option in the `startMethod`, this is key to obtaining a rolling n-ahead forecast type simulation (see details below).

Qbar
: The DCC dynamics unconditional Q matrix, required for the specification dispatch method.

Nbar
: The aDCC dynamics unconditional asymmetry matrix, required for the specification dispatch method.

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.

mexsimdata
: A list (equal to the number of asset) of matrices of simulated external regressor-in-mean data with row length equal to n.sim + n.start. If the fit object contains external regressors in the mean equation, this must be provided else will be assumed to be zero.

vexsimdata
: A list (equal to the number of asset) of matrices of simulated external regressor-in-variance data with row length equal to n.sim + n.start. If the fit object contains external regressors in the variance equation, this must be provided else will be assumed to be 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 (remember to stop the cluster on completion).

VAR.fit
: An VAR.fit list returned from calling the `varxfilter` or `varxfit` function with `postpad` set to "constant". This is required for the specification dispatch method.

prerealized
: Allows the starting realized volatility values to be provided by the user for the univariate GARCH dynamics.

...
: .

Details

In order to pass a correct specification to the filter routine, you must ensure that it contains the appropriate 'fixed.pars' in both the multivariate DCC part of the specification as well as the multiple univariate specification part, for which the method `setfixed<-` should be used.

Value

A `DCCsim` object containing details of the DCC-GARCH simulation.

Author(s)

Alexios Ghalanos

DCCspec-class *class: DCC Specification Class*

Description

The class is returned by calling the function dccspec.

Slots

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

`umodel`: Object of class "vector" The univariate model specification list.

Extends

Class "mGARCHspec", directly. Class "GARCHspec", by class "mGARCHspec", distance 2. Class "rGARCH", by class "mGARCHspec", distance 3.

Methods

setfixed<- `signature(object = "DCCspec", value = "vector")`: Set fixed second stage parameters.

setstart<- `signature(object = "DCCspec", value = "vector")`: Set starting second stage parameters.

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

Note

The 'umodel' list is absorbed into the 'model' list in all other DCC classes.

Author(s)

Alexios Ghalanos

References

Croux, C. and Joossens, K. 2008, Robust estimation of the vector autoregressive model by a least trimmed squares procedure, *COMPSTAT*, 489–501.
Cappiello, L., Engle, R.F. and Sheppard, K. 2006, Asymmetric dynamics in the correlations of global equity and bond returns, *Journal of Financial Econometrics* **4**, 537–572.
Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

dccspec-methods *function: DCC-GARCH Specification*

Description

Method for creating a DCC-GARCH specification object prior to fitting.

Usage

```
dccspec(uspec, VAR = FALSE, robust = FALSE, lag = 1, lag.max = NULL,
lag.criterion = c("AIC", "HQ", "SC", "FPE"), external.regressors = NULL,
robust.control = list("gamma" = 0.25, "delta" = 0.01, "nc" = 10, "ns" = 500),
dccOrder = c(1,1), model = c("DCC", "aDCC", "FDCC"), groups = rep(1, length(uspec@spec)),
distribution = c("mvnorm", "mvt", "mvlaplace"), start.pars = list(), fixed.pars = list())
```

Arguments

- `uspec` A `uGARCHmultispec` object created by calling `multispec` on a list of univariate GARCH specifications.
- `VAR` Whether to fit a VAR model for the conditional mean.
- `robust` Whether to use the robust version of VAR.
- `lag` The VAR lag.
- `lag.max` The maximum VAR lag to search for best fit.
- `lag.criterion` The criterion to use for choosing the best lag when lag.max is not NULL.
- `external.regressors` Allows for a matrix of common pre-lagged external regressors for the VAR option.
- `robust.control` The tuning parameters to the robust regression including the proportion to trim ("gamma"), the critical value for re-weighted estimator ("delta"), the number of subsets ("ns") and the number of C-steps ("nc".
- `dccOrder` The DCC autoregressive order.
- `model` The DCC model to use, with a choice of the symmetric DCC, asymmetric (aDCC) and the Flexible DCC (FDCC). See notes for more details.
- `groups` The groups corresponding to each asset in the FDCC model, where these are assumed and checked to be contiguous and increasing (unless only 1 group).
- `distribution` The multivariate distribution. Currently the multivariate Normal, Student and Laplace are implemented, and only the Normal for the FDCC model.
- `start.pars` (optional) Starting values for the DCC parameters (starting values for the univariate garch specification should be passed directly via the 'uspec' object).
- `fixed.pars` (optional) Fixed DCC parameters. This is required in the `dccfilter`, `dccforecast`, `dccsim` with spec, and `dccroll` methods.

Details

The `robust` option allows for a robust version of VAR based on the multivariate Least Trimmed Squares Estimator described in Croux and Joossens (2008).

Value

A `DCCspec` object containing details of the DCC-GARCH specification.

Note

The FDCC model of Billio, Caporin and Gobbo (2006) allows different DCC parameters to govern the dynamics of the correlation of distinct groups. The drawback is a somewhat larger parameter set, and no correlation targeting. Still, it remains a feasible model for not too large a number of groups, and avoids the unrealistic assumption, particularly for large datasets, of one parameter governing all the dynamics, as in the DCC model. Note that the group indices must be increasing (unless all 1), which means that you should arrange your dataset so that the assets are ordered by their groups.

Author(s)

Alexios Ghalanos

References

Billio, M., Caporin, M., & Gobbo, M. 2006, Flexible dynamic conditional correlation multivariate GARCH models for asset allocation, *Applied Financial Economics Letters*, **2(02)**, 123–130.
Croux, C. and Joossens, K. 2008, Robust estimation of the vector autoregressive model by a least trimmed squares procedure, *COMPSTAT*, 489–501.
Cappiello, L., Engle, R.F. and Sheppard, K. 2006, Asymmetric dynamics in the correlations of global equity and bond returns, *Journal of Financial Econometrics* **4**, 537–572.
Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

`DCCtest` *Engle and Sheppard Test of Dynamic Correlation*

Description

A test of non-constant correlation based on Engle and Sheppard (2001).

Usage

```
DCCtest(Data, garchOrder = c(1,1), n.lags = 1, solver = "solnp",
solver.control = list(), cluster = NULL, Z = NULL)
```

Arguments

`Data`	A multivariate data matrix.
`garchOrder`	The first stage common GARCH order.
`n.lags`	The number of lags to test for the presence of non-constant correlation.
`solver`	Either "solnp" or "nlminb" .
`solver.control`	Control arguments list passed to optimizer.
`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 (remember to stop the cluster on completion).
`Z`	(Optional) The standardized residuals from a constant correlation model. If supplied the model is not estimated since this is the only input the test requires.

Details

The test effectively equates to estimating a multivariate dataset using the Constant Conditional Correlation (CCC) model of Bollerslev (1990) and after which the standardized residuals (standardized by the symmetric square root decomposition of the estimated constant correlation matrix) should be i.i.d. with covariance the identity matrix. Testing for this can be done using a series of artificial regressions on the outer and lagged product of these residuals and a constant. In the rmgarch package, the CCC model is calculated using a static GARCH copula (Normal) model.

Value

A list with the proposed Null hypothesis (H0), the test statistic and its p-value.

Author(s)

Alexios Ghalanos

References

Bollerslev, T. 1990, Modelling the coherence in short-run nominal exchange rates: a multivariate generalized ARCH model, *The Review of Economics and Statistics*, **72(3)**, 498–505.
Engle, R.F. and Sheppard, K. 2001, Theoretical and empirical properties of dynamic conditional correlation multivariate GARCH, *NBER Working Paper*.

`dji30retw` *data: 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

```
data(dji30retw)
```

Format

A data.frame containing 30x1141 observations.

Source

Yahoo Finance

`fastica` *Fast Fixed Point ICA*

Description

The fast fixed point algorithm for independent component analysis and projection pursuit based on the direct translation to R of the FastICA program of the original authors at the Helsinki University of Technology.

Usage

```
fastica(X, approach = c("symmetric", "deflation"), n.comp = dim(X)[2], demean = TRUE,
pca.cov = c("ML", "LW", "ROB", "EWMA"), gfun = c("pow3", "tanh", "gauss", "skew"),
finetune = c("none", "pow3", "tanh", "gauss", "skew"), tanh.par = 1, gauss.par = 1,
step.size = 1, stabilization = FALSE, epsilon = 1e-4, maxiter1 = 1000, maxiter2 = 5,
A.init = NULL, pct.sample = 1, firstEig = NULL, lastEig = NULL,
pcaE = NULL, pcaD = NULL, whiteSig = NULL, whiteMat = NULL, dewhiteMat = NULL,
rseed = NULL, trace = FALSE, ...)
```

Arguments

`X`	The multidimensional signal matrix, where each column of matrix represents one observed signal.
`approach`	The decorrelation approach to use, with “symmetric” estimating the components in parallel while “deflation” estimating one-by-one as in projection pursuit.
`n.comp`	Number of independent components to estimate, defaults to the dimension of the data (rows). Is overwritten by `firstEig` and `lastEig`.
`demean`	(Logical) Whether the data should be centered.
`pca.cov`	The method to use for the calculation of the covariance matrix during the PCA whitening phase. “ML” is the standard maximum likelihood method, “LW” is the Ledoit and Wolf method, “ROB” is the robust method from the MASS package and “EWMA” an exponentially weighted moving average estimator. Optional parameters passed via the (...) argument.
`gfun`	The nonlinearity algorithm to use in the fixed-point algorithm.

finetune — The nonlinearity algorithm for fine-tuning.

tanh.par — Control parameter used when nonlinearity algorithm equals “tanh”.

gauss.par — Control parameter used when nonlinearity algorithm equals “gauss”.

step.size — Step size. If this is anything other than 1, the program will use the stabilized version of the algorithm.

stabilization — Controls whether the program uses the stabilized version of the algorithm. If the stabilization is on, then the value of `step.size` can momentarily be halved if the program estimates that the algorithm is stuck between two points (this is called a stroke). Also if there is no convergence before half of the maximum number of iterations has been reached then the `step.size` will be halved for the rest of the rounds.

epsilon — Stopping criterion. Default is 0.0001.

maxiter1 — Maximum number of iterations for `gfun` algorithm.

maxiter2 — Maximum number of iterations for `finetune` algorithm.

A.init — Initial guess for the mixing matrix A. Defaults to a random (standard normal) filled matrix (no.signals by no.factors).

pct.sample — Percentage [0-1] of samples used in one iteration. Samples are chosen at random.

firstEig — This and `lastEig` specify the range for eigenvalues that are retained, `firstEig` is the index of largest eigenvalue to be retained. Making use of this option overwrites `n.comp`.

lastEig — This is the index of the last (smallest) eigenvalue to be retained and overwrites `n.comp` argument.

pcaE — Optionally provided eigenvector (must also supply `pcaD`).

pcaD — Optionally provided eigenvalues (must also supply `pcaE`).

whiteSig — Optionally provided Whitened signal.

whiteMat — Optionally provided Whitening matrix (no.factors by no.signals).

dewhiteMat — Optionally provided dewhitening matrix (no.signals by no.factors).

rseed — Optionally provided seed to initialize the mixing matrix A (when `A.init` not provided).

trace — To report progress in the console, set this to ‘TRUE’.

... — Optional arguments passed to the pca.cov methods.

Details

The fastica program is a direct translation into R of the FastICA Matlab program of Gaevert, Hurri, Saerelae, and Hyvaerinen with some extra features. All computations are currently implemented in R so for very large dimensional sets alternative implementations may be faster. Porting part of the code to C++ may be implemented in a future version.

Value

A list containing the following values:

- `A` Estimated Mixing Matrix (no.signals by no.factors).
- `W` Estimated UnMixing Matrix (no.factors by no.signals).
- `U` Estimated rotation Matrix (no.factors by no.factors).
- `S` The column vectors of estimated independent components (no.obs by no.factors).
- `C` Estimated Covariance Matrix (no.signals by no.signals).
- `whiteningMatrix` The Whitening matrix (no.factors by no.signals).
- `dewhiteningMatrix` The de-Whitening matrix (no.signals by no.factors).
- `rseed` The random seed used (if any) for initializing the mixing matrix A.
- `elapsed` The elapsed time.

Note

Since version 1.0-3 the multidimensional signal matrix is now the usual row by column matrix, where the rows represent observations and columns the signals. Before this version, the reverse was true in keeping with the original version of the program.
Dimensionality reduction can be achieved in the PCA stage by use of either `n.comp` in which case the n.comp largest eigenvalues are chosen, else by selection of `firstEig` and `lastEig` which overwrites the choice of `n.comp`.

Author(s)

Hugo Gaevert, Jarmo Hurri, Jaakko Saerelae, and Aapo Hyvaerinen for the original FastICA package for matlab.
Alexios Ghalanos for this R-port.

References

Hyvaerinen, A. and Oja,.E , 1997, A fast fixed-point algorithm for independent component analysis, *Neural Computation*, **9(7)**, 1483-1492. Reprinted in *Unsupervised Learning*, G. Hinton and T. J. Sejnowski, 1999, MIT Press.

Examples

```
## Not run:
# create a set of independent signals S, glued together by a mixing matrix A
# (note the notation and matrix multiplication direction as we are dealing with
# row rather than column vectors)
set.seed(100)
S <- matrix(runif(10000), 5000, 2)
A <- matrix(c(1, 1, -1, 2), 2, 2, byrow = TRUE)
# the mixed signal X
X = S %*% t(A)
```

```
# The function centers and whitens (by the eigenvalue decomposition of the
# unconditional covariance matrix)  the data before applying the theICA algorithm.
IC <- fastica(X, n.comp = 2, approach = "symmetric", gfun = "tanh", trace  = TRUE,
A.init = diag(2))

# demeaned data:
X_bar = scale(X, scale = FALSE)

# whitened data:
X_white = X_bar %*% t(IC$whiteningMatrix)

# check whitening:
# check correlations are zero
cor(X_white)
# check diagonals are 1 in covariance
cov(X_white)

# check that the estimated signals(S) multiplied by the
# estimated mxing matrix (A) are the same as the original dataset (X)
round(head(IC$S %*% t(IC$A)), 12) == round(head(X), 12)

# do some plots:
par(mfrow = c(1, 3))
plot(IC$S %*% t(IC$A), main = "Pre-processed data")
plot(X_white, main = "Whitened and Centered components")
plot(IC$S, main = "ICA components")

## End(Not run)
```

---

fMoments-class *Class* "fMoments"

Description

Object returned from calling fmoments.

Objects from the Class

Objects can be created by calls of the form new("fMoments", ...).

Slots

moments**:** Object of class "vector" A list with the (roll+1) n-ahead forecast moment matrices.

model**:** Object of class "vector" A list with details of data generating process.

Methods

show signature(object = "fMoments"): Summary method.

fitted signature(object = "fMoments"): Conditional mean forecast matrix.

rcov `signature(object = "fMoments")`: Conditional covariance forecast array.

rcoskew `signature(object = "fMoments")`: Conditional third co-moment array.

rcokurt `signature(object = "fMoments")`: Conditional fourth co-moment array.

Author(s)

Alexios Ghalanos

Examples

```
showClass("fMoments")
```

`fmoments-methods` *Moment Based Forecast Generation*

Description

Generates n-ahead forecast moment matrices given a choice of data generating processes.

Usage

```
fmoments(spec, Data, n.ahead = 1, roll  = 0, solver = "solnp",
solver.control = list(), fit.control = list(eval.se = FALSE),
cluster = NULL, save.output = FALSE, save.dir = getwd(),
save.name = paste("M", sample(1:1000, 1), sep = ""), ...)
```

Arguments

`Data`	An n-by-m data matrix or data.frame.
`spec`	Either a DCCspec or GOGARCHspec.
`n.ahead`	The n.ahead forecasts (n.ahead>1 is unconditional).
`roll`	Whether to fit the data using (n - roll) periods and then return a (roll+1) n-ahead rolling forecast moments.
`solver`	The choice of solver to use for all models but "var", and includes 'solnp', 'nlminb' and 'nloptr'.
`solver.control`	Optional control options passed to the appropriate solver chosen.
`fit.control`	Control arguments passed to the fitting routine.
`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).
`save.output`	Whether output should be saved to file instead of being returned to the workspace.
`save.dir`	The directory to save output if save.output is TRUE.
`save.name`	The name of the file to save the output list.
`...`	Additional parameters passed to the model fitting routines. In particular, for the 'gogarch' model additional parameters are passed to the ICA routines, whereas for the 'dcc' and 'cgarch' models this would include the 'realizedVol' xts matrix for the realGARCH model.

Details

The function allows to generate forecast covariance matrices for use in the QP based EV model, and also for the “gogarch” model higher co-moment matrices for use in the Utility maximization model implemented separately.

Value

A `fMoments` object containing the forecast moments (list of length roll+1) and the model details (list).

Author(s)

Alexios Ghalanos

`fScenario-class` *Class* `"fScenario"`

Description

Object returned from calling `fscenario`.

Objects from the Class

Objects can be created by calls of the form `new("fScenario", ...)`.

Slots

`scenario`**:** Object of class `"vector"` A list with the (roll+1) scenario matrices.

`model`**:** Object of class `"vector"` A list with details of data generating process.

Methods

show `signature(object = "fScenario")`: Summary method.

goget `signature(object = "fScenario")`: Get a specified ‘arg’ from the object (only ‘scenario’ used).

fitted `signature(object = "fScenario")`: Returns an array of the simulated scenario returns, of dimensions n.sim by n.assets by (roll+1), with third dimension labels the actual forecast time index, and second dimension labels the asset names. The last forecast scenario will always be completely out of sample so the time index label for that is generated using the `generatefwd` function in the rugarch package.

Author(s)

Alexios Ghalanos

fscenario-methods *Scenario Generation*

Description

Generates a 1-ahead forecast scenario given a choice of data generating processes (for use in stochastic programming or risk management).

Usage

```
fscenario(Data, sim = 1000, roll = 0,
model = c("gogarch", "dcc", "cgarch", "var", "mdist"),
spec = NULL,
var.model = list(lag = 1, lag.max = NULL,
lag.criterion = c("AIC", "HQ", "SC", "FPE"),
robust = FALSE, robust.control = list("gamma" = 0.25,
"delta" = 0.01, "nc" = 10, "ns" = 500)),
mdist.model = list(distribution = c("mvn", "mvt", "manig"),
AR = TRUE, lag = 1),
spd.control = list(lower = 0.1, upper = 0.9, type = "pwm",
kernel = "epanech"),
cov.method = c("ML", "LW", "EWMA", "MVE", "MCD", "MVT", "BS"),
cov.options = list(shrinkage=-1, lambda = 0.96),
solver = "solnp", solver.control = list(),
fit.control = list(eval.se = FALSE),
cluster = NULL, save.output = FALSE, save.dir = getwd(),
save.name = paste("S", sample(1:1000, 1), sep = ""), rseed  = NULL, ...)
```

Arguments

`Data`	An n-by-m data matrix or data.frame.
`sim`	The size of the simulated 1-ahead forecast.
`roll`	Whether to fit the data using (n - roll) periods and then return a (roll+1) 1-ahead rolling simulated scenarios.
`model`	A choice of 5 models for generating scenarios.
`spec`	Required if choosing ‘gogarch’, ‘dcc’ or ‘cgarch’, in which case this represents a specification object (see rmgarch package) .
`var.model`	Required if model is var.
`mdist.model`	Required if model is mdist, and provides details for the model estimation (not yet implemented).
`spd.control`	Required if model is “cgarch” and transformation is spd.
`cov.method`	For model “var” this represents the choice of covariance matrix to use to generate random deviates.
`cov.options`	For model “var” this provides the optional parameters to certain types of covariance estimation methods.

`solver`	The choice of solver to use for all models but “var”, and includes ‘solnp’, ‘nlminb’ and ‘nloptr’.
`solver.control`	Optional control options passed to the appropriate solver chosen.
`fit.control`	Control arguments passed to the fitting routine.
`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).
`save.output`	Whether output should be saved to file instead of being returned to the workspace.
`save.dir`	The directory to save output if save.output is TRUE.
`save.name`	The name of the file to save the output list.
`rseed`	A vector of length sim to initiate the random number generator.
`...`	Additional parameters passed to the model fitting routines. In particular, for the ‘gogarch’ model additional parameters are passed to the ICA routines, whereas for the ‘dcc’ and ‘cgarch’ models this would include the ‘realizedVol’ xts matrix for the realGARCH model.

Details

The functionality here provides some wrapper functions, to create 1-ahead (and optionally rolling, useful for backtesting) scenarios for use in portfolio optimization using stochastic programming methods. The nature of these data generating processes (as implemented here) and resulting optimization problems results in the so called anticipative class of stochastic programming models. If save.output is chosen, and given a save.dir, the scenario is saved (using save.name) and an object is returned containing an empty list for the scenario but with a model details list and the seed values. This can then be passed on to the `goload` function which can read from the directory and return a complete object with the scenario.

Value

A `fScenario` object containing the scenario and the model details (list). The scenario list contains a list of the (roll+1) simulated forecast scenarios, the list of (roll+1) simulated forecast residuals, the forecast conditional mean, the forecast covariance and the list of random generator seed values used for replication. In addition, for the gogarch model the ICA whitening (K) and rotation matrices are also returned and required for replication of results (these may be entered in the ‘gogarchspec’ function). Use the `fitted` method on the object to extract the simulated returns forecast.

Author(s)

Alexios Ghalanos

goGARCHfft-class *Class: GO-GARCH portfolio density*

Description

Class for the GO-GARCH portfolio density

Objects from the Class

The class is returned by calling the function `convolution` on objects of class `goGARCHfit`, `goGARCHfilter`, `goGARCHforecast`, `goGARCHsim` and `goGARCHroll`

Slots

`dist`: A list with the portfolio density and other details.

`model`: A list with the model details carried across objects.

Methods

dfft `signature(object = "goGARCHfft")`: The takes additional argument "index" to indicate the particular time point, and returns an interpolated density function which may be called like any other "d" type density function.

pfft `signature(object = "goGARCHfft")` The takes additional argument "index" to indicate the particular time point, and returns an interpolated distribution function which may be called like any other "p" type distribution function.

qfft `signature(object = "goGARCHfft")` This takes additional argument "index" to indicate the particular time point, and returns an interpolated quantile function which may be called like any other "q" type quantile function. This may also be used to generate pseudo-random variables from the distribution by using random standard uniform numbers as inputs.

nportmoments `signature(object = "goGARCHfft")`: Calculate and returns a matrix of the first 4 standardized moments by evaluation of the portfolio density using quadrature based method (i.e. calling R's "integrate" function on the portfolio FFT based density). Depending on the GOGARCH class the density was based (e.g. goGARCHfit vs goGARCHforecast), the format of the output will be different, and generally follow the format 'rules' of that class.

notes

In the case that `convolution` was called on a `goGARCHforecast` or `goGARCHroll` object, the `dist` slot will contain the max of n.ahead or n.roll. There should be no confusion here since the multivariate forecast methods in rmgarch only allow either n.ahead>1 with n.roll = 0 (pure unconditional), or n.ahead = 1 with n.roll>=0 (pure rolling), and only the latter in the case of a `gogarchroll`. While the `nportmoments` method reconstitutes the forecasts into a more familiar form (n.ahead x n.moments x (n.roll+1)), this does not make sense for the distribution methods (d*, p*, and q*), and it is understood that when the user calls for example `dfft(object, index=5)` on an object created from a forecast with n.ahead=10 and n.roll=0, the index is meant to indicate the unconditional density forecast at time T+5. Similarly, when calling codedfft(object, index=0) on an object created from a forecast with n.ahead=1 and n.roll = 1 (remember that n.roll is zero based), the index is meant to indicate the first (of two, since rolls = 0:1) rolling forecast density.

Author(s)

Alexios Ghalanos

`goGARCHfilter-class` *class: GO-GARCH Filter Class*

Description

Class for the GO-GARCH filtered object.

Objects from the Class

The class is returned by calling the function `gogarchfilter` and is mainly called by `gogarchfit` when the "out.sample" option is used.

Slots

`mfilter`**:** Multivariate filter object.

`model`**:** Object of class `"vector"` containing details of the GOGARCH model specification.

Extends

Class `"mGARCHfilter"`, directly. Class `"GARCHfilter"`, by class "mGARCHfilter", distance 2. Class `"rGARCH"`, by class "mGARCHfilter", distance 3.

Methods

as.matrix `signature(x = "goGARCHfilter")`:
: function:
  as.matrix(x, which = "A")
  This returns four types of matrices relating to the estimation of the independent components in the GO-GARCH model. Valid choices are "A" for the mixing matrix, "W" for the unmixing matrix, "U" for the rotational matrix and "K" for the whitening matrix, "Kinv" for the de-whitening matrix.

likelihood `signature(object = "goGARCHfilter")`: The quasi log-likelihood of the model, which being an independent factor model is the sum of the univariate GARCH log-likelihoods plus a term for the mixing matrix. For a dimensionality reduced system, this is NA.

coef `signature(object = "goGARCHfilter")`: Extraction of independent factor GARCH model coefficients.

fitted `signature(object = "goGARCHfilter")`: Extracts the conditional mean equation filtered values.

residuals `signature(object = "goGARCHfilter")`: Extracts the conditional mean equation residuals.

convolution `signature(object = "goGARCHfilter")`:
function:
convolution(object, weights, fft.step = 0.001, fft.by = 0.0001, fft.support = c(-1, 1), support.method = c("user", "adaptive"), use.ff = TRUE, cluster = NULL, trace = 0,...)
The convolution method takes a goGARCHfit object and a weights vector or matrix and calculates the weighted density. If a vector is given, it must be the same length as the number of assets, otherwise a matrix with row dimension equal to the row dimension of the filtered dataset (i.e. less any lags). In the case of the multivariate normal distribution, this simply returns the linear and quadratic transformation of the mean and covariance matrix, while in the multivariate affine NIG distribution this is based on the numerical inversion by FFT of the characteristic function. In that case, the "fft.step" option determines the stepsize for tuning the characteristic function inversion, "fft.by" determines the resolution for the equally spaced support given by "fft.support", while the use of the "ff" package is recommended to avoid memory problems on some systems and is turned on via the "use.ff" option. The "support.method" option allows either a fixed support range to be given (option 'user'), else an adaptive method is used based on the min and max of the assets at each point in time at the 0.00001 and 1-0.00001 quantiles. The range is equally spaced subject to the "fft.by" value but the returned object no longer makes of the "ff" package returning instead a list. Finally, the option for parallel computation is available via the use of a cluster object as elsewhere in this package.

nisurface `signature(object = "goGARCHfilter")`:
function:
nisurface(object, type = "cov", pair = c(1, 2), factor = c(1,2), plot = TRUE)
Creates the covariance or correlation (determined by "type" being either "cov" or "cor") news impact surface for a pair of assets and factors. Since the shocks impact the factors independently, the news impact surface is a combination of the independent news impact curves of the factors which when combined via the mixing matrix A, create the dynamics for the underlying asset-factor surface function

portmoments `signature(object = "goGARCHfilter")`:
function:
gportmoments(object, weights)
Calculates the first 4 portfolio moments using the geometric properties of the model, given a vector or matrix of asset weights. If a matrix is given it must have row dimension equal to the row dimension of the filtered dataset (i.e. less any lags), else if a vector is given it will be replicated for all time points.

rcoskew `signature(object = "goGARCHfilter")` function:
rcoskew(object, standardize = TRUE, from = 1, to = 1)
Returns the 'time-varying' NxN^2 coskewness tensor in array format. The "from" and "to" options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to 100 indices per call.

rcokurt `signature(object = "goGARCHfilter")` function:
rcokurt(object, standardize = TRUE, from = 1, to = 1)
Returns the 'time-varying' NxN^3 cokurtosis tensor in array format. The "from" and "to" options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to models with less than 100 assets.

rcov `signature(object = "goGARCHfilter")`: Returns the time-varying NxN covariance matrix in array format.

rcor `signature(object = "goGARCHfilter")`: Returns the time-varying NxN correlation matrix in array format.

betacovar `signature(object = "goGARCHfilter")`: function:
betacovar(object, weights, asset = 1, cluster = NULL)
Returns the covariance beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

betacoskew `signature(object = "goGARCHfilter")`: function:
betacoskew(object, weights, asset = 1, cluster = NULL)
Returns the coskewness beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

betacokurt `signature(object = "goGARCHfilter")`: function:
betacokurt(object, weights, asset = 1, cluster = NULL)
Returns the cokurtosis beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

show `signature(object = "goGARCHfilter")`: Summary method.

Note

The reference by Paolella (2007) contains more details on the algorithm for the characteristic function inversion via FFT. The application of this method in a related model can be found in Chen (2007). The de Athayde and Flores (2002) paper is the basis for the geometric properties of the higher moment tensors in finance.

Author(s)

Alexios Ghalanos

References

de Athayde, G.M. and Flores Jr, R.G. 2002, On Certain Geometric Aspects of Portfolio Optimisation with Higher Moments, *mimeo*.
Broda, S.A. and Paolella, M.S. 2009, CHICAGO: A Fast and Accurate Method for Portfolio Risk Calculation, *Journal of Financial Econometrics* **7(4)**, 412–436 .
Paolella, M.S. 2007, Intermediate Probability - A Computational Approach, *Wiley-Interscience*.
Schmidt, R., Hrycej, T. and Stutzle 2006, Multivariate distribution models with generalized hyperbolic margins, *Computational Statistics \& Data Analysis* **50(8)**, 2065-2096.

gogarchfilter-methods *function: GO-GARCH Filter*

Description

Method for filtering the GO-GARCH model.

Usage

```
gogarchfilter(fit, data, out.sample = 0, n.old = NULL, cluster = NULL, ...)
```

Arguments

`fit`	A GO-GARCH fit object of class `goGARCHfit`.
`data`	A multivariate data object. Can be a matrix or data.frame or timeSeries.
`out.sample`	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
`n.old`	For comparison with goGARCHfit models using the out.sample argument, this is the length of the original dataset.
`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 (remember to stop the cluster on completion).
`...`	.

Value

A `goGARCHfilter` object containing details of the GO-GARCH filter.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(dji30ret)
spec = gogarchspec()
fit = gogarchfit(spec = spec, data = dji30ret[,1:4], gfun = "tanh")
filter = gogarchfilter(fit, data = dji30ret[,1:4])

## End(Not run)
```

goGARCHfit-class *class: GO-GARCH Fit Class*

Description

Class for the GO-GARCH fitted object.

Objects from the Class

The class is returned by calling the function `gogarchfit`.

Slots

`mfit`**:** Multivariate fit object.

`model`**:** Object of class `"vector"` containing details of the GO-GARCH model specification.

Extends

Class `"mGARCHfit"`, directly. Class `"GARCHfit"`, by class "mGARCHfit", distance 2. Class `"rGARCH"`, by class "mGARCHfit", distance 3.

Methods

as.matrix `signature(x = "goGARCHfit")`: function:
as.matrix(x, which = "A")
This returns four types of matrices relating to the estimation of the independent components in the GO-GARCH model. Valid choices are "A" for the mixing matrix, "W" for the unmixing matrix, "U" for the rotational matrix and "K" for the whitening matrix, "Kinv" for the dewhitening matrix.

coef `signature(object = "goGARCHfit")`: extraction of independent factor GARCH model coefficients.

convolution `signature(object = "goGARCHfit")`:
function:
convolution(object, weights, fft.step = 0.001, fft.by = 0.0001, fft.support = c(-1, 1), support.method = c("user", "adaptive"), use.ff = TRUE, cluster = NULL, trace = 0,...)
The convolution method takes a goGARCHfit object and a weights vector or matrix and calculates the weighted density. If a vector is given, it must be the same length as the number of assets, otherwise a matrix with row dimension equal to the row dimension of the filtered dataset (i.e. less any lags). In the case of the multivariate normal distribution, this simply returns the linear and quadratic transformation of the mean and covariance matrix, while in the multivariate affine NIG distribution this is based on the numerical inversion by FFT of the characteristic function. In that case, the "fft.step" option determines the stepsize for tuning the characteristic function inversion, "fft.by" determines the resolution for the equally spaced support given by "fft.support", while the use of the "ff" package is recommended to avoid memory problems on some systems and is turned on via the "use.ff" option. The "support.method" option allows either a fixed support range to be given (option 'user'), else an

adaptive method is used based on the min and max of the assets at each point in time at the 0.00001 and 1-0.00001 quantiles. The range is equally spaced subject to the “fft.by” value but the returned object no longer makes of the “ff” package returning instead a list. Finally, the option for parallel computation is available via the use of a cluster object as elsewhere in this package.

fitted `signature(object = "goGARCHfit")`: Extracts the conditional mean equation fitted values.

residuals `signature(object = "goGARCHfit")`: Extracts the conditional mean equation residuals.

likelihood `signature(object = "goGARCHfit")`: The quasi log-likelihood of the model, which being an independent factor model is the sum of the univariate GARCH log-likelihoods plus a term for the mixing matrix. For a dimensionality reduced system, this is NA.

nisurface `signature(object = "goGARCHfit")`:
function:
nisurface(object, type = "cov", pair = c(1, 2), factor = c(1,2), plot = TRUE)
Creates the covariance or correlation (determined by “type” being either “cov” or “cor”) news impact surface for a pair of assets and factors. Since the shocks impact the factors independently, the news impact surface is a combination of the independent news impact curves of the factors which when combined via the mixing matrix A, create the dynamics for the underlying asset-factor surface function.

gportmoments `signature(object = "goGARCHfit")`:
function:
gportmoments(object, weights)
Calculates the first 4 portfolio moments using the geometric properties of the model, given a vector or matrix of asset weights. If a matrix is given it must have row dimension equal to the row dimension of the filtered dataset (i.e. less any lags), else if a vector is given it will be replicated for all time points.

rcoskew `signature(object = "goGARCHfit")` function:
rcoskew(object, standardize = TRUE, from = 1, to = 1)
Returns the ’time-varying’ NxN^2 coskewness tensor in array format. The “from” and “to” options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to 100 indices per call.

rcokurt `signature(object = "goGARCHfit")` function:
rcokurt(object, standardize = TRUE, from = 1, to = 1)
Returns the ’time-varying’ NxN^3 cokurtosis tensor in array format. The “from” and “to” options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to models with less than 100 assets.

rcov `signature(object = "goGARCHfit")`: Returns the time-varying NxN covariance matrix in array format.

rcor `signature(object = "goGARCHfit")`: Returns the time-varying NxN correlation matrix in array format.

betacovar `signature(object = "goGARCHfit")`: function:
betacovar(object, weights, asset = 1, cluster = NULL)
Returns the covariance beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

betacoskew `signature(object = "goGARCHfit")`: function:
betacoskew(object, weights, asset = 1, cluster = NULL)
Returns the coskewness beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

betacokurt `signature(object = "goGARCHfit")`: function:
betacokurt(object, weights, asset = 1, cluster = NULL)
Returns the cokurtosis beta given a matrix (of length equal to the number of rows of the original data, or vector which is then recycled to the number of rows of the original data) of benchmark weights and the asset number.

show `signature(object = "goGARCHfit")`: Summary method.

Note

The reference by Paolella (2007) contains more details on the algorithm for the characteristic function inversion via FFT. The application of this method in a related model can be found in Chen (2007). The de Athayde and Flores (2002) paper is the basis for the geometric properties of the higher moment tensors in finance.

Author(s)

Alexios Ghalanos

References

de Athayde, G.M. and Flores Jr, R.G. 2002, On Certain Geometric Aspects of Portfolio Optimisation with Higher Moments, *mimeo*.
Broda, S.A. and Paolella, M.S. 2009, CHICAGO: A Fast and Accurate Method for Portfolio Risk Calculation, *Journal of Financial Econometrics* **7(4)**, 412–436 .
Paolella, M.S. 2007, Intermediate Probability - A Computational Approach, *Wiley-Interscience*.
Schmidt, R., Hrycej, T. and Stutzle 2006, Multivariate distribution models with generalized hyperbolic margins, *Computational Statistics \& Data Analysis* **50(8)**, 2065-2096.

Examples

```
## Not run:
data(dji30ret)
spec = gogarchspec(mean.model = list(demean = "constant"),
variance.model = list(model = "sGARCH", garchOrder = c(1,1), submodel = NULL),
distribution.model = list(distribution = "manig"), ica = "fastica")
fit = gogarchfit(spec = spec, data  = dji30ret[,1:4, drop = FALSE],
out.sample = 50, gfun = "tanh")
# The likelihood of the model
likelihood(fit)
# the GARCH coefficients of the independent factors
coef(fit)
# a news-impact surface plot
#ni = nisurface(fit, type = "cov", pair = c(1, 2), factor = c(1,2), plot = TRUE)
```

```
# the time varying correlation array
mc = rcor(fit)
# plot(mc[1,2,], type = "l")
# The moments of an equally weighted portfolio (subtract the out.sample from dimension)
gm = gportmoments(fit, weights = matrix(1/4, ncol = 4,
nrow = dim(dji30ret)[1]-50), debug = TRUE)

## End(Not run)
```

---

gogarchfit-methods *function: GO-GARCH Filter*

Description

Method for filtering the GO-GARCH model.

Usage

```
gogarchfit(spec, data, out.sample = 0, solver = "solnp",
fit.control = list(stationarity = 1), solver.control = list(), cluster = NULL,
VAR.fit = NULL, ARcoef = NULL, ...)
```

Arguments

spec	A GO-GARCH spec object of class goGARCHspec.
data	A multivariate data object. Can be a matrix or data.frame or timeSeries.
out.sample	A positive integer indicating the number of periods before the last to keep for out of sample forecasting.
solver	One of either "nlminb", "solnp" or "gosolnp".
solver.control	Control arguments list passed to optimizer.
fit.control	Control arguments passed to the fitting routine. Stationarity explicitly imposes the variance stationarity constraint during optimization.
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 (remember to stop the cluster on completion).
VAR.fit	(optional) A previously estimated VAR list returned from calling the varxfilter function.
ARcoef	An optional named matrix of the fitted AR parameters obtained from calling the `arfimafit` function on each series and then extracting the coefficients (the normal distribution should be used for the AR estimation). The number of columns should be equal to the number of series, and the rows should include the AR coefficients (common lag for all series), 'sigma', and if included the mean ('mu'). The option to pass the coefficients directly rather than letting the function estimate them may be useful for example when there are convergence problems in the arfima routine and user control of each series estimation is desirable.
...	Additional arguments passed to the ICA functions.

Value

A goGARCHfit object containing details of the GO-GARCH fit.

Note

There is no check on the VAR.fit list passed to the method so particular care should be exercised so that the same data used in the fitting routine is also used in the VAR filter routine. The ability to pass this list of the pre-calculated VAR model is particularly useful when comparing different models (such as copula GARCH, DCC GARCH etc) using the same dataset and VAR method. Though the classical VAR estimation is very fast and may not require this extra step, the robust method is slow and therefore benefits from calculating this only once.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(dji30ret)
spec = gogarchspec(mean.model = list(demean = "constant"),
variance.model = list(model = "sGARCH", garchOrder = c(1,1), submodel = NULL),
distribution.model = list(distribution = "manig"),ica = "fastica")

fit = gogarchfit(spec = spec, data  = dji30ret[,1:4, drop = FALSE],
out.sample = 50, gfun = "tanh")
fit

## End(Not run)
```

---

goGARCHforecast-class *class: GO-GARCH Forecast Class*

Description

Class for the GO-GARCH forecast.

Objects from the Class

The class is returned by calling the function gogarchforecast.

Slots

`mforecast`: Multivariate forecast object.

`model`: Object of class "vector" containing details of the GOGARCH model specification.

Extends

Class "mGARCHforecast", directly. Class "GARCHforecast", by class "mGARCHforecast", distance 2. Class "rGARCH", by class "mGARCHforecast", distance 3.

Methods

convolution `signature(object = "goGARCHforecast")`:
function:
convolution(object, weights, fft.step = 0.001, fft.by = 0.0001, fft.support = c(-1, 1), support.method = c("user", "adaptive"), use.ff = TRUE, cluster = NULL, trace = 0,...)
The convolution method takes a goGARCHforecast object and a weights vector or matrix and calculates the weighted density. If a vector is given, it must be the same length as the number of assets, otherwise a matrix with row dimension equal to the total forecast horizon. In the case of the multivariate normal distribution, this simply returns the linear and quadratic transformation of the mean and covariance matrix, while in the multivariate affine NIG distribution this is based on the numerical inversion by FFT of the characteristic function. In that case, the "fft.step" option determines the stepsize for tuning the characteristic function inversion, "fft.by" determines the resolution for the equally spaced support given by "fft.support", while the use of the "ff" package is recommended to avoid memory problems on some systems and is turned on via the "use.ff" option. The "support.method" option allows either a fixed support range to be given (option 'user'), else an adaptive method is used based on the min and max of the assets at each point in time at the 0.00001 and 1-0.00001 quantiles. The range is equally spaced subject to the "fft.by" value but the returned object no longer makes of the "ff" package returning instead a list. The option for parallel computation is available via the use of a cluster object as elsewhere in this package. There is no special treatment of the forecast type here (unconditional or rolling), since either n.ahead with no roll or rolling with 1-ahead only choices are available for the `gogarchforecast` method. This means that the stored object does not distringuish between an unconditional or rolling forecast, calculating the density for all points (see note).

gportmoments `signature(object = "goGARCHforecast")`:
function:
gportmoments(object, weights)
Calculates the first 4 standardized portfolio moments using the geometric properties of the model, given a matrix of asset weights with row dimension equal to the forecast n.ahead or n.roll horizon. Returns an array of dimensions n.ahead x 4 (moments) x n.roll, with the third array dimension labelled with the T+0 index times. If the number of assets > 100, then the kurtosis is not returned (see cokurtosis restrictions below).

rcoskew `signature(object = "goGARCHforecast")`:
function:
rcoskew(object, standardize = TRUE, from = 1, to = 1, roll = 0)
Returns the 'time-varying' NxN^2 (coskewness tensor) x (to:from|roll) in array format. The "from" and "to" options indicate the time indices for which to return the array and "roll" the rolling index (base=0). The third dimension array label denotes the T+i (i=from:to) forecast horizon given the T+0 roll index which is returned as an attribute (attr(,"T+0")) of the array. The "standardize" option indicates whether the coskewness should be standardized by the conditional sigma (see equations in vignette). It is also possible to set roll to the character 'all' in which case all the rolling 1-ahead forecasts are returned in an n by n^2 by (n.roll+1) array with 3rd dimension label the T+0 dates (instead of being an attribute).

rcokurt `signature(object = "goGARCHforecast")`:
function:
rcokurt(object, standardize = TRUE, from = 1, to = 1, roll = 0)
Returns the 'time-varying' NxN^3 (cokurtosis tensor) x (to:from|roll) in array format. The "from" and "to" options indicate the time indices for which to return the array and "roll" the rolling index (base=0). Because of memory issues, this is only returned when the number of assets are less than 100. The third dimension array label denotes the T+i (i=from:to) forecast horizon given the T+0 roll index which is returned as an attribute (attr(,"T+0")) of the array. The "standardize" option indicates whether the cokurtosis should be standardized by the conditional sigma (see equations in vignette). It is also possible to set roll to the character 'all' in which case all the rolling 1-ahead forecasts are returned in an n by n^3 by (n.roll+1) array with 3rd dimension label the T+0 dates (instead of being an attribute.

rcov `signature(object = "goGARCHforecast")`: Returns the conditional covariances, in a list of length (n.roll+1), with names the T+0 index, and each list slot having an array of dimensions n.asset x n.asset x n.ahead, with the third array dimension labelled as T+i (i>0).

rcor `signature(object = "goGARCHforecast")`: Returns the conditional correlations, in a list of length (n.roll+1), with names the T+0 index, and each list slot having an array of dimensions n.asset x n.asset x n.ahead, with the third array dimension labelled as T+i (i>0).

coef `signature(object = "goGARCHforecast")`: Extraction of independent factor GARCH model coefficients saved from the goGARCHfit object.

fitted `signature(object = "goGARCHforecast")`: Extracts the conditional mean forecast values. Returns an n.ahead x n.assets x (n.roll+1) array where the third dimension array labels are the T+0 index times.

sigma `signature(object = "goGARCHforecast")`: Extracts the conditional sigma forecast values. Returns an n.ahead x n.assets x (n.roll+1) array where the third dimension array labels are the T+0 index times. Takes optional argument "factors" (default TRUE) denoting whether to return the factor conditional sigma or the transformed sigma for the assets.

as.matrix `signature(x = "goGARCHforecast")`:
function:
as.matrix(x, which = "A")
This returns four types of matrices relating to the estimation of the independent components in the GO-GARCH model. Valid choices are "A" for the mixing matrix, "W" for the unmixing matrix, "U" for the rotational matrix and "K" for the whitening matrix, "Kinv" for the de-whitening matrix.

betacovar `signature(object = "goGARCHforecast")`: function:
betacovar(object, weights, asset = 1)
Returns the covariance beta given a matrix (of length equal to the number of rows of the forecast horizon, or vector which is then recycled to the number of rows of the forecast horizon) of benchmark weights and the asset number.

betacoskew `signature(object = "goGARCHforecast")`: function:
betacoskew(object, weights, asset = 1)
Returns the coskewness beta given a matrix (of length equal to the number of rows of the forecast horizon, or vector which is then recycled to the number of rows of the forecast horizon) of benchmark weights and the asset number.

betacokurt `signature(object = "goGARCHforecast")`: function:
betacokurt(object, weights, asset = 1)

Returns the cokurtosis beta given a matrix (of length equal to the number of rows of the forecast horizon, or vector which is then recycled to the number of rows of the forecast horizon) of benchmark weights and the asset number.

show `signature(object = "goGARCHforecast")`: Summary method.

Note

The reference by Chen et al (2010) and Paolella (2007) contains more details on the algorithm for the characteristic function inversion via FFT. The de Athayde and Flores (2002) paper is the basis for some of the geometric properties of the higher moment tensors. The paper by Ghalanos et al (2013) contains more specific details.
Forecasts are carried out on the time varying parameters of the factor distributions, and then scaled and transformed to those of the assets after adding back the mean forecast (which is either a constant or the AR/VAR mean forecast).

Author(s)

Alexios Ghalanos

References

Chen, Y., Hardle, W., and Spokoiny, V. 2010, GHICA-Risk analysis with GH distributions and independent components, *Journal of Empirical Finance*, **17(2)**, 255–269.
de Athayde, G.M. and Flores Jr, R.G. 2002, On Certain Geometric Aspects of Portfolio Optimisation with Higher Moments, *mimeo*.
Ghalanos, A., Rossi, E., and Urga, G. (2013). *Independent Factor Autoregressive Conditional Density Model*, **forthcoming**.
Paolella, M.S. 2007, Intermediate Probability - A Computational Approach, *Wiley-Interscience*.

`gogarchforecast-methods`

function: GO-GARCH Forecast

Description

Method for forecasting from the GO-GARCH model.

Usage

```
gogarchforecast(fit, n.ahead = 10, n.roll = 0,
external.forecasts = list(mregfor = NULL), cluster = NULL, ...)
```

Arguments

- `fit` A GO-GARCH fit object of class `goGARCHfit`.
- `n.ahead` The forecast horizon.
- `n.roll` The no. of rolling forecasts to create beyond the first one.
- `external.forecasts` A list with a matrix object of the external lagged forecasts (if used). These must contain (n.roll+1) x n.ahead forecasts.
- `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 (remember to stop the cluster on completion).
- `...` .

Value

A `goGARCHforecast` object containing details of the GO-GARCH forecast.

Author(s)

Alexios Ghalanos

Examples

```
## Not run:
data(dji30ret)
spec = gogarchspec()
fit = gogarchfit(spec = spec, data = dji30ret[,1:4], out.sample = 10,
gfun = "tanh")
forecast = gogarchforecast(fit, n.ahead = 1, n.roll = 9)

## End(Not run)
```

`goGARCHroll-class` *class: GO-GARCH Roll Class*

Description

Class for the GO-GARCH Roll.

Objects from the Class

The class is returned by calling the function `gogarchroll`.

Slots

- `forecast`: Object of class `"vector"` which contains the rolling forecasts of the distributional parameters for each factor.
- `model`: Object of class `"vector"` containing details of the GOGARCH model specification.

Extends

Class "mGARCHroll", directly. Class "GARCHroll", by class "mGARCHroll", distance 2. Class "rGARCH", by class "mGARCHroll", distance 3.

Methods

coef `signature(object = "goGARCHroll")`: Extraction of independent factor GARCH model coefficients saved from the goGARCHfit objects(returns a list).

fitted `signature(object = "goGARCHroll")`: Extracts the conditional fitted forecast values (returns an xts object with index the actual forecast T+1 times).

sigma `signature(object = "goGARCHroll")`: Extracts the conditional sigma forecast values (returns an xts object with index the actual forecast T+1 times). Takes optional argument "factors" (default TRUE) denoting whether to return the factor conditional sigma or the transformed sigma for the assets.

rcov `signature(object = "goGARCHroll")`: Returns the time-varying n.asset x n.asset x (n.roll+1) covariance matrix in array format, where the third dimension labels are now the actual rolling n.ahead=1 forecast time indices (T+1).

rcor `signature(object = "goGARCHroll")`: Returns the time-varying n.asset x n.asset x (n.roll+1) correlation matrix in array format, where the third dimension labels are now the actual rolling n.ahead=1 forecast time indices (T+1).

rcoskew `signature(object = "goGARCHroll")`: Returns the time-varying n.asset x n.asset^2 x (n.roll+1) coskewness matrix in array format, where the third dimension labels are now the actual rolling n.ahead=1 forecast time indices (T+1). There is a "standardize" option which indicates whether the coskewness should be standardized by the conditional sigma (see equations in vignette).

rcokurt `signature(object = "goGARCHroll")`: Returns the time-varying n.asset x n.asset^3 x (n.roll+1) cokurtosis matrix in array format, where the third dimension labels are now the actual rolling n.ahead=1 forecast time indices (T+1). There is a "standardize" option which indicates whether the cokurtosis should be standardized by the conditional sigma (see equations in vignette).

gportmoments `signature(object = "goGARCHroll")`:
function:
gportmoments(object, weights)
Calculates the first 4 standardized portfolio moments using the geometric properties of the model, given a matrix of asset weights with row dimension equal to the total rolling forecast horizon. Returns an xts object of dimensions (total rolling forecast) x 4 (moments), with the index denoting the T+1 actual forecast time. If the number of assets > 100, then the kurtosis is not returned (see cokurtosis restrictions below).

convolution `signature(object = "goGARCHroll")`:
function:
convolution(object, weights, fft.step = 0.001, fft.by = 0.0001, fft.support = c(-1, 1), support.method = c("user", "adaptive"), use.ff = TRUE, cluster = NULL, trace = 0,...)
The convolution method takes a goGARCHroll object and a weights vector or matrix and calculates the weighted density. If a vector is given, it must be the same length as the number of assets, otherwise a matrix with row dimension equal to the row dimension of total forecast horizon. In the case of the multivariate normal distribution, this simply returns the linear and

quadratic transformation of the mean and covariance matrix, while in the multivariate affine NIG distribution this is based on the numerical inversion by FFT of the characteristic function. In that case, the "fft.step" option determines the stepsize for tuning the characteristic function inversion, "fft.by" determines the resolution for the equally spaced support given by "fft.support", while the use of the "ff" package is recommended to avoid memory problems on some systems and is turned on via the "use.ff" option. The "support.method" option allows either a fixed support range to be given (option 'user'), else an adaptive method is used based on the min and max of the assets at each point in time at the 0.00001 and 1-0.00001 quantiles. The range is equally spaced subject to the "fft.by" value but the returned object no longer makes of the "ff" package returning instead a list. The option for parallel computation is available via the use of a cluster object as elsewhere in this package. Passing this object to the distribution methods (e.g. qfft) follows the same rules as the goGARCHforecast object, namely that the index is zero based.

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

Author(s)

Alexios Ghalanos

`gogarchroll-methods` *function: GO-GARCH Rolling Estimation*

Description

Method for performing rolling estimation of the GO-GARCH model.

Usage

```
gogarchroll(spec, data, n.ahead = 1, forecast.length = 50, n.start = NULL,
refit.every = 25, refit.window = c("recursive", "moving"), window.size = NULL,
solver = "solnp", solver.control = list(), fit.control = list(), rseed = NULL,
cluster = NULL, save.fit = FALSE, save.wdir = NULL, ...)
```

Arguments

`spec`	A GO-GARCH spec object of class `goGARCHspec`.
`data`	A multivariate data object. Can be a matrix or data.frame or timeSeries.
`n.ahead`	The forecast horizon (only 1-ahead supported for rolling forecasts).
`forecast.length`	The length of the total forecast for which out of sample data from the dataset will be excluded for testing.
`n.start`	Instead of forecast.length, this determines the starting point in the dataset from which to initialize the rolling forecast.
`refit.every`	Determines every how many periods the model is re-estimated.

refit.window	Whether the refit is done on an expanding window including all the previous data or a moving window where all previous data is used for the first estimation and then moved by a length equal to refit.every (unless the window.size option is used instead).
window.size	If not NULL, determines the size of the moving window in the rolling estimation, which also determines the first point used.
solver	The solver to use.
fit.control	Control parameters parameters passed to the fitting function.
solver.control	Control parameters passed to the solver.
rseed	Initialization seed for first ICA fit. The rest of the ICA fits are initialized with the previous mixing matrix (using A.init).
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 (remember to stop the cluster on completion).
save.fit	Whether to save the fitted objects of class `goGARCHfit` during the estimation of each ("refit.every"). If true, the directory to save must be provided (see below). The function will not save this by default for reasons of memory management, but can save it as an ".rda" file in the user's chosen directory for further analysis.
save.wdir	If "save.fit" is true, the directory in which to save the `goGARCHfit` objects (1 for each "refit.every").
...	.

Value

An object of class `goGARCHroll`.

Author(s)

Alexios Ghalanos

`goGARCHsim-class` *class: GO-GARCH Simultion Class*

Description

Class for the GO-GARCH Simulation.

Objects from the Class

The class is returned by calling the function `gogarchsim`.

Slots

`msim`: Object of class `"vector"` The multivariate simulation list.

`model`: Object of class `"vector"` containing details of the GOGARCH model specification.

Extends

Class "mGARCHsim", directly. Class "GARCHsim", by class "mGARCHsim", distance 2. Class "rGARCH", by class "mGARCHsim", distance 3.

Methods

convolution `signature(object = "goGARCHsim")`:
function:
convolution(object, weights, fft.step = 0.001, fft.by = 0.0001, fft.support = c(-1, 1), support.method = c("user", "adaptive"), use.ff = TRUE, sim = 1, cluster = NULL, trace = 0,...)
The convolution method takes a goGARCHsim object and a weights vector and calculates the weighted density. The vector must be the same length as the number of assets. The “sim” option indicates the simulation index to use, given the “m.sim” option chosen in the call to the simulation function. In the case of the multivariate normal distribution, this simply returns the linear and quadratic transformation of the mean and covariance matrix, while in the multivariate affine NIG distribution this is based on the numerical inversion by FFT of the characteristic function. In that case, the “fft.step” option determines the stepsize for tuning the characteristic function inversion, “fft.by” determines the resolution for the equally spaced support given by “fft.support”, while the use of the “ff” package is recommended to avoid memory problems on some systems and is turned on via the “use.ff” option. The “support.method” option allows either a fixed support range to be given (option ‘user’), else an adaptive method is used based on the min and max of the assets at each point in time at the 0.00001 and 1-0.00001 quantiles. The range is equally spaced subject to the “fft.by” value but the returned object no longer makes use of the “ff” package returning instead a list. Finally, the option for parallel computation is available via the use of a cluster object as elsewhere in this package.

gportmoments `signature(object = "goGARCHsim")`:
function:
gportmoments(object, weights, sim = 1)
Calculates the first 3 portfolio moments using the geometric properties of the model, given a matrix of asset weights with row dimension equal to the row dimension of the filtered dataset (i.e. less any lags). The “sim” option indicates the simulation index to use, given the “m.sim” option chosen in the call to the simulation function.

rcoskew `signature(object = "goGARCHsim")`:
function:
rcoskew(object, from = 1, to = 1, sim = 1)
Returns the 'time-varying' NxN^2 coskewness tensor in array format. The “from” and “to” options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to 100 indices. The “sim” option indicates the simulation index to use, given the “m.sim” option chosen in the call to the simulation function.

rcokurt `signature(object = "goGARCHsim")`:
function:
rcokurt(object, standardize = TRUE, from = 1, to = 1)
Returns the 'time-varying' NxN^3 cokurtosis tensor in array format. The “from” and “to” options indicate the time indices for which to return the arrays. Because of memory issues, this is limited to models with less than 20 assets.

rcov `signature(object = "goGARCHsim")`: Returns the time-varying NxN covariance matrix in array format. There is an additional “sim” option which indicates the simulation index to use, given the “m.sim” option chosen in the call to the simulation function.

rcor `signature(object = "goGARCHsim")`: Returns the time-varying NxN correlation matrix in array format. There is an additional “sim” option which indicates the simulation index to use, given the “m.sim” option chosen in the call to the simulation function.

as.matrix `signature(x = "goGARCHsim")`:
function:
as.matrix(x, which = "A")
This returns four types of matrices relating to the estimation of the independent components in the GO-GARCH model. Valid choices are “A” for the mixing matrix, “W” for the unmixing matrix, “U” for the rotational matrix and “K” for the whitening matrix, “Kinv” for the de-whitening matrix.

Author(s)

Alexios Ghalanos

`gogarchsim-methods` *function: GO-GARCH Simulation*

Description

Method for simulation from a fitted GO-GARCH model.

Usage

```
gogarchsim(fit, n.sim = 1, n.start = 0, m.sim = 1,
startMethod = c("unconditional", "sample"), prereturns = NA, preresiduals = NA,
presigma = NA, mexsimdata = NULL, rseed = NULL, cluster = NULL, ...)
```

Arguments

`fit`	A GO-GARCH fit object of class `goGARCHfit`.
`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.
`prereturns`	Allows the starting return data to be provided by the user.
`preresiduals`	Allows the starting factor residuals to be provided by the user.
`presigma`	Allows the starting conditional factor sigma to be provided by the user.
`mexsimdata`	A list of matrices with the simulated lagged external variables (if any). The list should be of size m.sim and the matrices each have n.sim + n.start rows.

`rseed`	Optional seeding value(s) for the random number generator.
`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 (remember to stop the cluster on completion).
`...`	.

Value

A `goGARCHsim` object containing details of the GO-GARCH simulation.

Author(s)

Alexios Ghalanos

`goGARCHspec-class` *class: GO-GARCH Specification Class*

Description

Class for the GO-GARCH specification.

Objects from the Class

The class is returned by calling the function `goGARCHspec`.

Slots

`model`**:** Multivariate model specification.

`umodel`**:** Univariate model specification.

Extends

Class `"mGARCHspec"`, directly. Class `"GARCHspec"`, by class "mGARCHspec", distance 2. Class `"rGARCH"`, by class "mGARCHspec", distance 3.

Methods

show `signature(object = "goGARCHspec")`: Summary method.

Note

The mixing matrix in the GO-GARCH model implemented in the rmgarch package is based on non-parametric independent component analysis (ICA) methodology. The estimation is a 2-stage methodology described in Broda and Paolella (2009) and Zhang and Chan (2009). The extension to the use of the full multivariate affine GH distribution is detailed in Ghalanos et al (2011).

Author(s)

Alexios Ghalanos

References

van der Weide, R. 2002, GO-GARCH: a multivariate generalized orthogonal GARCH model, *Journal of Applied Econometrics*, 549–564.
Zhang, K. and Chan, L. 2009, Efficient factor GARCH models and factor-DCC models, *Quantitative Finance*, 71–91.
Broda, S.A. and Paolella, M.S. 2009, CHICAGO: A Fast and Accurate Method for Portfolio Risk Calculation, *Journal of Financial Econometrics*, 412–436.
Ghalanos, A. and Rossi, E. and Urga, G. 2011, Independent Factor Autoregressive Conditional Density Model, *Pending–submitted*.

`gogarchspec-methods` *function: GO-GARCH Specification*

Description

Method for creating a GO-GARCH specification object prior to fitting.

Usage

```
gogarchspec(mean.model = list(model = c("constant", "AR", "VAR"), robust = FALSE,
lag = 1, lag.max = NULL, lag.criterion = c("AIC", "HQ", "SC", "FPE"),
external.regressors = NULL,
robust.control = list("gamma" = 0.25, "delta" = 0.01, "nc" = 10, "ns" = 500)),
variance.model = list(model = "sGARCH", garchOrder = c(1,1), submodel = NULL,
variance.targeting = FALSE), distribution.model = c("mvnorm", "manig", "magh"),
ica = c("fastica", "radical"),
ica.fix = list(A = NULL, K = NULL), ...)
```

Arguments

`mean.model`	The mean specification. Allows for either a constant filtration of the return series, a univariate AR for each series with common lag (via the “lag” argument) else a classical or robust Vector Autoregressive Model (VAR). The ‘robust’ option allows for a robust version of VAR based on the multivariate Least Trimmed Squares Estimator described in Croux and Joossens (2008). The ‘robust.control’ includes additional tuning parameters to the robust regression including the proportion to trim (“gamma”), the critical value for Reweighted estimator (“delta”), the number of subsets (“ns”) and the number of C-steps (“nc”). The external.regressors argument allows for a matrix of common external regressors in the constant, AR or VAR formulations.
`variance.model`	The univariate variance specification for the independent factors of the GO-GARCH model.

distribution.model
: The distributions supported are the multivariate normal ("mvnorm") and the multivariate affine NIG ("manig") and GHYP ("magh") distributions of Schmidt et al (see references).

ica
: The algorithm to use for extracting the independent components. The `fastica` and `radical` algorithms are the only ICA algorithms currently allowed and locally implemented. See their documentation for a list of additional arguments possible, which may be passed in the `gogarchfit` method.

ica.fix
: This allows the option of supplying the mixing matrix (A) and optionally the whitening Matrix (K). This is likely to be use when comparing different models (with the same mean filtration and dataset but different variance models) and you wish to use the same independent factors.

...
: .

Value

A `goGARCHspec` object containing details of the GO-GARCH specification.

Author(s)

Alexios Ghalanos

`goload-methods` *Load Scenario from File*

Description

Loads a previously saved fScenario from file and returns a `fScenario` or `fMoments` object.

Usage

```
goload(object, ...)
```

Arguments

object
: A `fScenario` or `fMoments` object which was created with save.output set to TRUE.

...
: not used.

Details

There are times when it is more efficient to save large scenarios to file (particularly when creating them in parallel), rather than returning them to the user workspace. The save.output option in the `fscenario` and `fmoments` allows to do just that, returning instead a lighter object with an empty scenario slot, but with the model slot included, containing the details of the location and name of the saved scenario (or moments list). The goload function then takes this object, reads the location and name and loads the scenario (or moments) into its slot in the object and returns this to the user's workspace.

Value

A `fScenario` or `fMoments` object with the scenario or moments slot now filled with the saved data from file.

Author(s)

Alexios Ghalanos

`last-methods` *First and Last methods for accessing objects*

Description

Functions for accessing first-n and last-n values of an object (similar to head and tail).

Usage

```
last(x, index = 1, ...)
first(x, index = 1, ...)
```

Arguments

`x`	Currently only arrays supported.
`index`	First or Last n-indices to return values for.
`...`	For expansion to other classes.

Methods

`signature(x = "ANY")`

`signature(x = "array")`

Author(s)

Alexios Ghalanos

`mGARCHfilter-class` *Class: Multivariate GARCH Filter Class*

Description

High Level multivariate GARCH filter class.

Objects from the Class

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

Extends

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

Methods

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

Author(s)

Alexios Ghalanos

`mGARCHfit-class` *Class: Multivariate GARCH Fit Class*

Description

High Level multivariate GARCH fit class.

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.

Methods

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

Author(s)

Alexios Ghalanos

mGARCHforecast-class *Class: Multivariate GARCH Forecast Class*

Description

High Level multivariate GARCH forecast class.

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

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

Author(s)

Alexios Ghalanos

mGARCHroll-class *Class: Multivariate GARCH Roll Class*

Description

High Level multivariate GARCH roll class.

Objects from the Class

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

Extends

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

Methods

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

Author(s)

Alexios Ghalanos

mGARCHsim-class *Class: Multivariate GARCH Simulation Class*

Description

High Level multivariate GARCH simulation class.

Objects from the Class

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

Extends

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

Methods

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

Author(s)

Alexios Ghalanos

mGARCHspec-class *Class: Multivariate GARCH Specification*

Description

High Level multivariate GARCH specification class.

Objects from the Class

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

Extends

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

Methods

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

Author(s)

Alexios Ghalanos

radical *The Robust Accurate, Direct ICA aLgorithm (RADICAL).*

Description

An ICA algorithm based on an efficient entropy estimator (due to Vasicek) which is robust to outliers.

Usage

```
radical(X, n.comp = dim(X)[2], demean = TRUE, pca.cov = c("ML", "LW", "ROB", "EWMA"),
k = 150, augment = FALSE, replications = 30, sd = 0.175, firstEig = 1,
lastEig = dim(X)[1], pcaE = NULL, pcaD = NULL, whiteSig = NULL, whiteMat = NULL,
dewhiteMat = NULL, rseed = NULL, trace = FALSE, ...)
```

Arguments

`X`	The multidimensional signal matrix, where each column of matrix represents one observed signal.
`n.comp`	Number of independent components to estimate, defaults to the dimension of the data (rows). Is overwritten by `firstEig` and `lastEig`.
`demean`	(Logical) Whether the data should be centered.
`pca.cov`	The method to use for the calculation of the covariance matrix during the PCA whitening phase. "ML" is the standard maximum likelihood method, "LW" is the Ledoit-Wolf method, "ROB" is the robust method from the MASS package and "EWMA" an exponentially weighted moving average estimator. Optional parameters passed via the ... argument.
`k`	The number of angles at which to evaluate the contrast function. The ICA 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`	This is the number of replicated points for each original point. The default value is 30. 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.
`sd`	This is the standard deviation (noise) of the replicated points when using the augmentation option.
`firstEig`	This and `lastEig` specify the range for eigenvalues that are retained, `firstEig` is the index of largest eigenvalue to be retained. Making use of this option overwrites `n.comp`.
`lastEig`	This is the index of the last (smallest) eigenvalue to be retained and overwrites `n.comp` argument.
`pcaE`	Optionally provided eigenvector (must also supply `pcaD`).
`pcaD`	Optionally provided eigenvalues (must also supply `pcaE`).

`whiteSig`	Optionally provided Whitened signal.
`whiteMat`	Optionally provided Whitening matrix (no.factors by no.signals).
`dewhiteMat`	Optionally provided dewhitening matrix (no.signals by no.factors).
`rseed`	Optionally provided seed to initialize the augmented data matrix.
`trace`	To report progress in the console, set this to `TRUE`.
`...`	Optional arguments passed to the pca.cov methods.

Details

The interested reader should consult the paper in the references section for details on the properties of the algorithm.
The algorithm is quite slow, despite partial implementation in C++, and should only be used on small to medium sized sets.

Value

A list containing the following values:

`A`	Estimated Mixing Matrix (no.signals by no.factors).
`W`	Estimated UnMixing Matrix (no.factors by no.signals).
`U`	Estimated rotation Matrix (no.factors by no.factors).
`S`	The column vectors of estimated independent components (no.obs by no.factors).
`C`	Estimated Covariance Matrix (no.signals by no.signals).
`whiteningMatrix`	The Whitening matrix (no.factors by no.signals).
`dewhiteningMatrix`	The de-Whitening matrix (no.signals by no.factors).
`rseed`	The random seed used (if any) for initializing the mixing matrix A.
`elapsed`	The elapsed time.

Author(s)

Erik G. Learned-Miller for the Radical algorithm and Matlab package.
Alexios Ghalanos for this R-port.

References

Learned-Miller, A.G and Fisher III, J.W., 2003, ICA Using Spacings Estimates of Entropy, *Journal of Machine Learning Research*, **4**, 1271-1295. `http://www.cs.umass.edu/~elm/ICA/`

Examples

```
## Not run:
# create a set of independent signals S, glued together by a mixing matrix A
# (note the notation and matrix multiplication direction as we are dealing with
# row rather than column vectors)
set.seed(100)
S <- matrix(runif(10000), 5000, 2)
A <- matrix(c(1, 1, -1, 2), 2, 2, byrow = TRUE)
# the mixed signal X
X = S %*% t(A)
# The function centers and whitens (by the eigenvalue decomposition of the
# unconditional covariance matrix) the data before applying the theICA algorithm.
IC <- radical(X, n.comp = 2)

# demeaned data:
X_bar = scale(X, scale = FALSE)

# whitened data:
X_white = X_bar %*% t(IC$whiteningMatrix)

# check whitening:
# check correlations are zero
cor(X_white)
# check diagonals are 1 in covariance
cov(X_white)

# check that the estimated signals(S) multiplied by the
# estimated mxing matrix (A) are the same as the original dataset (X)
round(head(IC$S %*% t(IC$A)), 12) == round(head(X), 12)

# do some plots:
par(mfrow = c(1, 3))
plot(IC$S %*% t(IC$A), main = "Pre-processed data")
plot(X_white, main = "Whitened and Centered components")
plot(IC$S, main = "ICA components")

## End(Not run)
```

`varxfit` *VARX Fit/Filter/Forecast/Simulation Functions*

Description

Vector Autoregressive (VAR) with Constant and Optional Exogenous Regressors (X) Fit, Filter, Forecast and Simulation functions for use with multivariate GARCH models.

Usage

```
varxfit(X, p, constant = TRUE, exogen = NULL, robust = FALSE, gamma = 0.25,
delta = 0.01, nc = 10, ns = 500, postpad = c("none", "constant", "zero", "NA"),
```

```
cluster = NULL)
varxfilter(X, p, Bcoef, exogen = NULL, postpad = c("none", "constant", "zero", "NA"))
varxforecast(X, Bcoef, p, out.sample, n.ahead, n.roll, mregfor)
varxsim (X, Bcoef, p, n.sim, n.start, prereturns, resids, mexsimdata)
```

Arguments

`X`	A multivariate data matrix.
`p`	The number of autoregressive lags.
`constant`	Whether to include a constant.
`exogen`	An optional matrix of exogenous regressors with as many rows as X, and appropriately lagged.
`Bcoef`	A matrix of coefficients for the varxfilter function.
`robust`	Whether to use the robust version of VAR based on the multivariate Least Trimmed Squares Estimator described in Croux and Joossens (2008).
`gamma`	Proportion to trim in the robust method.
`delta`	The critical value for Reweighted estimator for the robust method.
`ns`	The number of subsets to use for the robust method.
`nc`	The number of C-steps to use for the robust method.
`postpad`	(defaults to 'none') Whether to postpad the fitted/filtered values (and hence calculation of residuals) with the estimated constant, zeros or NA's, thus returning matrices of the same size as the input data (rather than input data size less the number of lags).
`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 in the case of the robust VAR version (remember to stop the cluster on completion).
`out.sample`	The number of points kept for out of sample rolling forecast.
`n.ahead`	The forecast horizon.
`n.roll`	In combination with `out.sample`, determines the number of times to roll forward the n.ahead forecast using data left out of sample.
`mregfor`	Matrix of external regressor forecasts (with appropriate lag structure).
`n.sim`	Simulation horizon.
`n.start`	Simulation burn-in sample.
`prereturns`	Optionally supplied pre-return matrix with "p" lags to initialize simulation.
`resids`	Matrix of randomly generated residuals of size n.sim+n.start.
`mexsimdata`	Matrix of external regressor pre-generated random values to use in the simulation (if NULL then assumed zero).

Details

This are convenience functions to be optionally used when using the multivariate GARCH methods.

Value

A list with the following items:

- `Bcoef` [varxfit, varxfilter]
  The coefficient matrix with rows equal to number of assets, and columns equal to number of assets x number of lags plus 1 (constant) plus number of exogenous regressors.
- `xfitted` [varxfit, varxfilter]
  The fitted/filtered series (conditional mean series).
- `xresiduals` [varxfit, varxfilter]
  The residuals.
- `Bcov` [varxfit]
  The covariance matrix of the coefficients.
- `se` [varxfit]
  The standard error of the coefficients.
- `tstat` [varxfit]
  The t-stat of the s.e.
- `pstat` [varxfit]
  The p-values of the s.e.
- `lag` [varxfit, varxfilter]
  The number of autoregressive lags.
- `mxn` [varxfit]
  The number of exogenous regressors .
- `meansim` [varxsim]
  The simulated conditional mean.

The varxsim returns an n.sim x n.asset matrix of the simulated conditional means, while the varxforecast returns an n.ahead x n.assets x (n.roll+1) array of the forecast conditional means.

Note

Part of the varxfit functionality and structure is inspired from the 'vars' package, but the estimation method is implemented in a very quick way without calling 'lm'. The robust method is based on the matlab program of Christophe Croux available from his website and the option of using parallel computation is implemented for this particular choice.The `postpad` option is used when the returned data needs to be of the same size as the inputed data for easier manipulation/comparison (since padding is done post-estimation, there is no bias introduced during estimation).

Author(s)

Alexios Ghalanos

References

Lutkepohl, H. 2005, New introduction to multiple time series analysis, *Springer*.
Croux, C. and Joossens, K. 2008, Robust estimation of the vector autoregressive model by a least trimmed squares procedure, *COMPSTAT*, 489–501.

`wmargin` *Weighted Distribution Margin*

Description

Return the weighted margin of one of 3 elliptical distributions given a matrix of weights.

Usage

```
wmargin(distribution = "mvnorm", weights, mean, Sigma, shape = NA, skew = NA)
```

Arguments

`distribution`	One of 'mvnorm', 'mvlaplace' or 'mvt'.
`weights`	Either a vector or matrix of weights, in the latter case must be of the same row dimension as the covariance array.
`mean`	Wither a vector or matrix of conditional distribution means, in the latter case must be of the same row dimension as the covariance array.
`Sigma`	An array of covariances, usually returned by calling the 'rcov' method on one of the multivariate GARCH fitted objects.
`shape`	The shape (d.o.f.) parameter of the multivariate student distribution.
`skew`	Not currently required for the 3 distributions used.

Details

This is just a convenience function to return the weighted variance and mean of the three elliptical distributions given a set of weights.

Value

A matrix with each row representing the conditional weighted marginal density with corresponding parameters.

Author(s)

Alexios Ghalanos

Index