Package ‘QGARCH’

October 2, 2022

Type Package

Title Quantile autoregressive conditional heteroscedasticity

Version 1.0.0

Date 2022-09-26

Author Songhua Tan

Maintainer Songhua Tan <tansonghua@163.sufe.edu.cn>

Description Programs of the quantile GARCH model in Zhu, Q., Tan, S., Zheng, Y and Li, G.(2022+).

Encoding UTF-8

LazyData true

Depends Rcpp,RcppEigen,parallel,rugarch,dfoptim

LinkingTo Rcpp,RcppEigen

RoxygenNote 7.2.0

R topics documented:

`ASD_QR` *Compute ASD and related covariances of QR estimation*

Description

Compute ASD, Ω_{0w}, Ω_{1w} and Σ_w of QR estimation

Usage

```
ASD_QR(parm, Y, w, tau, h_type = "HS")
```

Arguments

`parm`	Vector. Parameter vector. $(\omega(\tau), \alpha_1(\tau), \beta_1(\tau))'$.
`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau`	Double. Specific quantile level (τ).
`h_type`	Character (`"HS"` or `"B"`). The commonly used bandwidth types for ℓ to estimate the asymptotic covariance $\Sigma_w(\tau)$ at Theorem 3.3. The default value is "HS"

Value

A list of ASD and related covariances of QR estimation.

- ASD: 3-dim vector. The asymptotic standard errors (ASD) of the corresponding parameter vector.
- Omega_0: 3×3 matrix. $\widetilde{\Omega}_{0w}(\tau,\tau) = \frac{1}{n}\sum_{t=1}^{n} w_t^2 \dot{\widetilde{q}}_t\left(\widetilde{\boldsymbol{\theta}}_{wn}(\tau)\right)\dot{\widetilde{q}}_t\left(\widetilde{\boldsymbol{\theta}}_{wn}(\tau)\right)$.
- Omega_1: 3×3 matrix. $\widetilde{\Omega}_{1w}(\tau) = \frac{1}{n}\sum_{t=1}^{n} \widetilde{f}_{t-1}\left(F_{t-1}^{-1}(\tau)\right) w_t \dot{\widetilde{q}}_t\left(\widetilde{\boldsymbol{\theta}}_{wn}(\tau)\right)\dot{\widetilde{q}}_t\left(\widetilde{\boldsymbol{\theta}}_{wn}(\tau)\right)$.
- Sigma: 3×3 matrix. $\widetilde{\Sigma}_w(\tau,\tau) = \tau(1-\tau)\widetilde{\Omega}_{1w}^{-1}(\tau)\widetilde{\Omega}_{0w}(\tau,\tau)\widetilde{\Omega}_{1w}^{-1}(\tau)$.

`const_test` *The Cramér-von Misses (CvM) test*

Description

The Cramér-von Misses (CvM) test for constant persistence coefficient

Usage

```
const_test(Y, w, tau_multi, parm_multi, Omega_1_multi, b = NA)
```

Arguments

`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau_multi`	Vector(k-dim). Multiple quantile levels.
`parm_multi`	Matrix($k \times 3$). Each row in `parm_multi` represents a parameter estimate in corresponding quantile levels.
`Omega_1_multi`	Array($k \times 3 \times 3$). Each `Omega_1_multi[i,,] i=1,...,k` represents The estimate of $\tilde{\Omega}_{1w}$ in the corresponding quantile level.
`b`	Int. The block size for subsampling. If `b=NA`, $b == \sqrt{length(Y)}$.

Details

This function provides a test for constant persistence coefficient.

H_0 : for all $\tau \in \mathcal{T}, R\theta(\tau) = \beta_1$ against H_1 : there exist $\tau \in \mathcal{T}, R\theta(\tau) \neq \beta_1$,

where $R = (0, 0, 1)$ is a row vector, and $\beta_1 \in (0, 1)$ is an unknown constant independent of τ. If the null hypothesis H_0 holds, then $\beta_1(\tau)$ does not vary cross quantiles.

Define the inference process $\nu_n(\tau) = R\left(\tilde{\theta}_{wn}(\tau) - \int_{\mathcal{T}} \tilde{\theta}_{wn}(\tau)d\tau\right)$. To test H_0, the CvM test statistic is constructed as follows

$S_n = n \int_{\mathcal{T}} \nu_n^2(\tau)d\tau$.

Value

A list of test results.

- stat_CvM: the CvM test statistic
- stat_b_CvM: the CvM test statistics in each step of subsampling
- p_value: p-value of the CvM test
- information: summary information

`fit1_optim` *An optimization function of self-weighted QR estimation given fixed initial value.*

Description

This function provide an optimization function of self-weighted QR with a fixed initial value. Either method, derivative optimization using `stats::optim()` or derivative-free optimization using `dfoptim::hjkb()`, can be used. To avoid error or non-convergence in optimization, we add an innovation to the initial value and re-optimize this problem. If the optimization fails on the fixed initial value, a more complicated optimization function `fit1_optim_grid` based on greedy search.

Usage

```
fit1_optim(
  par = NULL,
  Y,
  w,
  tau,
  lower = c(NA, NA, 0.001),
  upper = c(NA, NA, 1 - 0.001),
  method = "optim",
  iter_max_1 = 10,
  iter_max_2 = 20,
  seed = 1234
)
```

Arguments

`par`	Vector. Initial value for optimization.
`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau`	Double. Specific quantile level.
`lower`	Vector. Lower bound for parameter. The default value is `c(NA,NA,1e-3)`.
`upper`	Vector. Upper bound for parameter. The default value is `c(NA,NA,1-1e-3)`.
`method`	Character. If `method="optim"`, derivative optimization by `stats::optim()` is used. If `method="dfoptim"`, derivative-free optimization by `dfoptim::hjkb()` is used.
`iter_max_1`	Int. If the optimization function does not converge or the parameter is at the boundary, then re-optimize. Maximum number of repetitions of this step is `iter_max_1`.
`iter_max_2`	Int. If the condition in `iter_max_1` cannot be satisfied, then relax the boundary condition and estimate again. Maximum number of repetitions of this step is `iter_max_2-iter_max_1`.
`seed`	Double. Random seed is used to generate an innovation to perturb the initial value.

Value

A list of optimization results returned from the `optim()` or `hjkb()` function.

Note

1. The selection of initial values.

Since the QGARCH model is extended from the classical GARCH model, the initial value can be chosen based on GARCH model. Specifically,

- Estimate the parameters of GARCH(1,1) model (2.1) with $r_t = \sqrt{|y_t|}\,(sgn(y_t))$ using Gaussian QMLE and $Q_\tau(\eta_t)$ using empirical quantile of $\hat{\eta}_t$ based on package `rugarch`.
- The initial value can be chosen as $\left(\frac{\hat{a}_0}{1-\hat{b}_1}\hat{Q}_\tau(\varepsilon_t), \hat{a}_1\hat{Q}_\tau(\varepsilon_t), \hat{b}_1\right)$, where $\hat{\varepsilon}_t = \hat{\eta}_t^2 sgn(\hat{\eta}_t)$.

2. The necessity of random seed.

To avoid error or non-convergence in optimization, we add an innovation to the initial value and re-optimize this problem, the random seed is set for reproducible results.

`fit1_optim_grid` *An optimization function of self-weighted QR given multiple initial values*

Description

This function extends the optimization function `fit1_optim()` given with a single initial value to multiple initial values. The method is based on greedy algorithm. In this framework, we start by listing all the possible parameter values with interval `1/D`. Then the loss function is evaluated at all points, and the points with the least loss are selected as the starting points for our optimization.

Usage

```
fit1_optim_grid(
  Y,
  w,
  tau,
  lower = c(NA, NA, 0.001),
  upper = c(NA, NA, 1 - 0.001),
  D = 10,
  num_best = 10
)
```

Arguments

`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau`	Double. Specific quantile level.
`lower`	Vector. Lower bound for parameter. The default value is `c(NA,NA,1e-3)`.
`upper`	Vector. Upper bound for parameter. The default value is `c(NA,NA,1-1e-3)`.
`D`	Int. `1/D` is the interval for partition. The default value is 10.
`num_best`	Int. The number of the selected points. The default value is 10.

Value

A list of optimization results returned from the `optim()` or `hjkb()` function.

fit2_optim *An optimization function of self-weighted CQR estimation given fixed initial value.*

Description

This function provide an optimization function of self-weighted CQR with a fixed initial value. Either method, derivative optimization using `stats::optim()` or derivative-free optimization using `dfoptim::hjkb()`, can be used. To avoid error or non-convergence in optimization, we add an innovation to the initial value and re-optimize this problem. If the optimization fails on initial value, a more complicated optimization function `fit2_optim_grid` based on greedy search.

Usage

```
fit2_optim(
  par,
  Y,
  w,
  tau,
  lower = c(1e-09, 0.001, 0.001, -Inf),
  upper = c(+Inf, 1 - 0.001, 1 - 0.001, +Inf),
  method = "optim",
  iter_max_1 = 10,
  iter_max_2 = 20,
  seed = 1234
)
```

Arguments

`par`	Vector. Initial value for optimization.
`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau`	Vector. Composite quantile levels.
`lower`	Vector. Lower bound for parameter vector. The default value is c(1e-9,1e-3,1e-3,NA).
`upper`	Vector. Upper bound for parameter vector. The default value is c(NA,NA,1-1e-3,NA).
`method`	Character. If `method="optim"`, derivative optimization by `stats::optim()` is used. If `method="dfoptim"`, derivative-free optimization by `dfoptim::hjkb()` is used.
`iter_max_1`	Int. If the optimization function does not converge or the parameter vector is at the boundary, then re-optimize. Maximum number of repetitions of this step is `iter_max_1`. The default is 10.
`iter_max_2`	Int. If the condition in `iter_max_1` cannot be satisfied, then relax the boundary condition and estimate again. Maximum number of repetitions of this step is `iter_max_2-iter_max_1`. The default of `iter_max_2` is 20.
`seed`	Double. Random seed is used to generate an innovation to perturb the initial value.

Value

A list of optimization result returned from the `optim()` function.

`fit2_optim_grid` *An optimization function of self-weighted CQR given multiple initial values*

Description

This function extends the optimization function `fit2_optim()` given with a single initial value to multiple initial values. The method is based on greedy algorithm. In this framework, we start by listing all the possible parameter values with interval `1/D`. Then the loss function is evaluated at all points, and the points with the least loss are selected as the starting points for our optimization.

Usage

```
fit2_optim_grid(
  Y,
  w,
  tau,
  lower = c(1e-09, 0.001, 0.001, -Inf),
  upper = c(+Inf, 1 - 0.001, 1 - 0.001, +Inf),
  D = 10,
  num_best = 10
)
```

Arguments

`Y`	Vector. Data.
`w`	Vector. Self-weight.
`tau`	Vector. Composite quantile level.
`lower`	Vector. Lower bound for parameter vector. The default value is `c(1e-9,1e-3,1e-3,+Inf)`.
`upper`	Vector. Upper bound for parameter vector. The default value is `c(-Inf,-Inf,1-1e-3,-Inf)`.
`D`	Int. `1/D` is the interval for partition. The default value is 10.
`num_best`	Int. The number of the selected points. The default value is 10.

Value

A list of optimization result returned from the `optim()` or `hjkb()` function.

g_tau — *Transformation function in CQR*

Description

Transformation function in CQR

Usage

```
g_tau(parm_CQR, tau)
```

Arguments

parm_CQR	Vector. Parameter vector (4-dim), i.e. $(a_0, a_1, b_1, \lambda)'$.
tau	Vector. Observations.

Value

Vector. Transformed parameter vector (3-dim), i.e. $(a_0 Q_\lambda(\lambda)/(1-b_1), a_1 Q_\lambda, b_1)'$.

Loss_CQR — *The loss function of CQR*

Description

The loss function of CQR

Usage

```
Loss_CQR(parm, Y, w, tau)
```

Arguments

parm	4-dim vector. $(a_0, a_1, b_1, \lambda)'$.
Y	Vector. Data.
w	Vector. Self-weights.
tau	Vector. Composite quantile levels.

Value

Double. The loss of CQR given a parameter vector

`Loss_CQR_gr` *The derivative of CQR loss function*

Description

The derivative of CQR loss function

Usage

```
Loss_CQR_gr(parm, Y, w, tau)
```

Arguments

`parm`	Vector (4-dim). $(a_0, a_1, b_1, \lambda)'$.
`Y`	Vector. Data
`w`	Vector. Self-weights.
`tau`	Vector. Composite quantile levels.

Value

Vector. The vector of the gradient of loss function for self-weighted CQR estimation given a parameter vector.

`Loss_QR` *Loss function for self-weighted QR estimation*

Description

Loss function for self-weighted QR estimation

Usage

```
Loss_QR(parm, Y, w, tau)
```

Arguments

`parm`	Vector. Parameter vector (3-dim), i.e. $(\omega(\tau), \alpha_1(\tau), \beta_1(\tau))'$.
`Y`	Vector. Data.
`w`	Vector. Self-weights.
`tau`	Double. Specific quantile level.

Value

Double. The value of the loss function for self-weighted QR estimation given a parameter vector.

Loss_QR_gr *Gradient of the loss function for self-weighted QR estimation*

Description

Gradient of the loss function for self-weighted QR estimation

Usage

```
Loss_QR_gr(parm, Y, w, tau)
```

Arguments

parm	Vector. Parameter vector (3-dim), i.e. $(\omega(\tau), \alpha_1(\tau), \beta_1(\tau))'$.
Y	Vector. Data.
w	Vector. Self-weights.
tau	Double. Specific quantile level.

Value

Vector. The vector of the gradient of loss function for self-weighted QR estimation given a parameter vector.

q_y *Conditional quantile function of Y_t*

Description

Conditional quantile function of Y_t, i.e. $Q_\tau(y_t|\mathcal{F}_{t-1})$.

Usage

```
q_y(parm, Y)
```

Arguments

parm	Vector. Parameter vector (3-dim), i.e. $(\omega(\tau), \alpha_1(\tau), \beta_1(\tau))'$.
Y	Vector. Data.

Value

Vector. $(Q_\tau(y_1|\mathcal{F}_0), \ldots, Q_\tau(y_N|\mathcal{F}_{N-1}))'$.

`VaR_forecasting_CQR` *Rolling forecasting for conditional quantiles based on self-weighted CQR*

Description

One-step-ahead conditional quantile forecast based on using a rolling forecast procedure.

Usage

```
VaR_forecasting_CQR(
  data,
  N_train,
  N_val,
  fixTau = c(0.005, 0.01, 0.05, 0.95, (1 - 0.01), (1 - 0.005)),
  H = 1:10/100,
  seed = 1234
)
```

Arguments

`data`	Vector. Data.
`N_train`	Int. The size of rolling window.
`N_val`	Int. The size of validation set.
`fixTau`	Vector. Multiple quantile levels.
`H`	Vector. Discrete points for grid search.
`seed`	Double. Select random seed to generate initial value for optimization.

Details

The framework is similar to `VaR_forecasting_QR`, see more details in `VaR_forecasting_QR`. Moreover, since the self-weighted CQR needs to choose an optimal h in advance, we divide the dataset into the training set with size `N_train`, validation set with size `N_val` and test set with size `length(data)-N_train-N_val`, and choose the optimal h that minimizes the check loss in the validation set.

Value

Matrix with `length(data)-N_train-N_val` rows and `length(fixTau)` columns. Each column saves the conditional quantile forecasts at a specific quantile level for the test set.

`VaR_forecasting_QR` *Rolling forecasting for conditional quantiles based on self-weighted QR*

Description

One-step-ahead conditional quantile forecast based on self-weighted QR using a rolling forecast procedure.

Usage

```
VaR_forecasting_QR(
  data,
  N_train,
  fixTau = c(0.005, 0.01, 0.05, 0.95, (1 - 0.01), (1 - 0.005)),
  seed = 1234
)
```

Arguments

`data`	Vector. Data.
`N_train`	Int. The size of rolling window.
`fixTau`	Vector. Multiple quantile levels.
`seed`	Double. Select random seed for optimization.

Details

We begin with the forecast origin $t_0 = N_{train}+1$, and obtain the fitted quantile GARCH(1,1) model using the data from the beginning to the forecast origin (exclusive). For each fitted model, we calculate the one-step-ahead conditional quantile forecast for the next trading day by $\widetilde{Q}_\tau(y_{t_0}|\mathcal{F}_{n_0}) = \widetilde{\omega}_{wn_0}(\tau) + \widetilde{\alpha}_{1wn_0}(\tau) \sum_{j=1}^{n_0} \left(\widetilde{\beta}_{1wn_0}(\tau)\right)^{j-1} |y_{N_{train}-j}|$ based on the self-weighted QR. We then advance the forecast origin by one and repeat the estimation and forecasting until all data are utilized.

Value

Matrix with `length(data)-N_train` rows and `length(fixTau)` columns. Each columns is the conditional quantile forecast result.

`weight_function_HeYi2021_cpp` *Self-weight function based on He & Yi (2021)*

Description

Self-weight function based on He & Yi (2021)

Usage

```
weight_function_HeYi2021_cpp(Y, C)
```

Arguments

Y	Vector. Data.
C	Double. Quantile of Y or |Y|.

Index