

Package ‘DMQ’

July 21, 2025

Type Package

Title Dynamic Multiple Quantile (DMQ) Model

Version 0.1.2

Date 2023-10-25

Maintainer Leopoldo Catania <leopoldo.catania@econ.au.dk>

Description Perform estimation, prediction, and simulations using the Dynamic Multiple Quantile model of Catania and Luati (2023) <[doi:10.1016/j.jeconom.2022.11.002](https://doi.org/10.1016/j.jeconom.2022.11.002)>. Can be used to estimate a set of conditional time-varying quantiles of a time series that do not cross.

License GPL-3

Imports Rcpp (>= 0.12.17)

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.6.0), Rsolnp, DEoptim, MASS, parallel

NeedsCompilation yes

Author Leopoldo Catania [cre, aut] (ORCID:

<<https://orcid.org/0000-0002-0981-1921>>),

Alessandra Luati [ctb] (ORCID: <<https://orcid.org/0000-0001-6407-9385>>)

Repository CRAN

Date/Publication 2023-10-28 15:20:05 UTC

Contents

DMQ-package	2
EstimateDMQ	2
fn.DEoptim	6
fn.optim	8
fn.solnp	9
ForecastDMQ	10
MomentsDMQ	11
MSFT	12
SimulateDMQ	13
UpdateDMQ	14
Index	16

DMQ-package

Dynamic Multiple Quantiles model in R

Description

The DMQ package allows us to simulate, estimate and forecast using the Dynamic Multiple Quantile (DMQ) model of Catania, L., & Luati, A. (2022). Semiparametric modeling of multiple quantiles. Journal of Econometrics, see [doi:10.1016/j.jeconom.2022.11.002](https://doi.org/10.1016/j.jeconom.2022.11.002).

Note

Please cite Catania and Luati (2022) in working papers and published papers that use DMQ. Use `citation("DMQ")`.

Author(s)

Leopoldo Catania [aut,cre], Alessandra Luati [aut]

Maintainer: Leopoldo Catania <leopoldo.catania@econ.au.dk>

References

Catania, L, and Luati, A. (2023). "Semiparametric modeling of multiple quantiles." Journal of Econometrics [doi:10.1016/j.jeconom.2022.11.002](https://doi.org/10.1016/j.jeconom.2022.11.002).

EstimatedDMQ

Estimate the Dynamic Multiple Quantile (DMQ) model.

Description

Estimate the parameters of the DMQ model using the estimator detailed in Catania and Luati (2023).

Usage

```
EstimatedDMQ(vY, vTau, iTau_star = NULL, vPn_Starting = NULL,
             FixReference = FALSE, FixOthers = FALSE,
             ScalingType = "InvSqrt",
             vQ_0 = NULL,
             fn.optimizer = fn.DEoptim,
             cluster = NULL, smooth = NULL, ...)
```

Arguments

<code>vY</code>	numeric vector of length $T \times 1$ containing the time series of observations.
<code>vTau</code>	numeric vector of length $J \times 1$ containing probability levels at which quantiles are estimated.
<code>iTau_star</code>	Integer indicating the position in <code>vTau</code> where the reference quantile is placed. For instance, if <code>vTau = seq(0.01, 0.99, 0.01)</code> then <code>iTau_star = 50</code> means that the median is used as the reference quantile.
<code>vPn_Starting</code>	numeric named vector of length 4×1 with starting values for the optimizer. For example <code>vPn_Starting = c("phi" = 0.9, "gamma" = 0.05, "alpha" = 0.01, "beta" = 0.7)</code> .
<code>FixReference</code>	logical. Should the reference quantile be fixed? By default <code>FixReference = FALSE</code> .
<code>FixOthers</code>	logical. Should the quantiles other than the reference quantile be fixed? By default <code>FixOthers = FALSE</code> .
<code>ScalingType</code>	character Indicating the scaling mechanism for the conditional quasi score. Possible choices are "Identity", "Inv", "InvSqrt". When <code>ScalingType = "InvSqrt"</code> quasi scores are scaled by their standard deviation. When <code>ScalingType = "Inv"</code> quasi scores are scaled by their variance. When <code>ScalingType = "Identity"</code> quasi scores are not scaled. Default value <code>ScalingType = "InvSqrt"</code> .
<code>vQ_0</code>	numeric. Vector of limiting quantiles evaluated at <code>vTau</code> . By default <code>FixOthers = NULL</code> meaning that empirical unconditional quantiles are used.
<code>fn.optimizer</code>	function. This is a generic optimization function that can be provided by the user. By default <code>fn.optimizer = fn.DEoptim</code> where <code>fn.DEoptim</code> is a wrapper to the <code>DEoptim</code> function of the package <code>DEoptim</code> . See the Details and Examples sections for user defined optimization routines.
<code>cluster</code>	A cluster object created calling using the <code>paralell</code> package. If supplied parallel processing is used to speed up the computations if <code>fn.optimizer</code> makes use of it. When <code>fn.optimizer = fn.DEoptim</code> parallel computation can be used.
<code>smooth</code>	logical. Should a smooth version of the objective function should be used? If using a gradient based optimizer like <code>fn.optimizer = fn.optim</code> is it advised to set <code>smooth = TRUE</code> . By default, when <code>fn.optimizer = fn.DEoptim</code> we set <code>smooth = FALSE</code> and when <code>fn.optimizer = fn.optim</code> or <code>fn.optimizer = fn.solnp</code> we set <code>smooth = TRUE</code> .
<code>...</code>	Additional arguments to be passed to <code>fn.optimizer</code> .

Details

Starting values for the optimizer are by default set as `c("phi" = 0.94, "gamma" = 0.10, "alpha" = 0.05, "beta" = 0.95)`.

The user is free to employ his/her own optimization routine via the `fn.optimizer` argument. `fn.optimizer` accepts a function object. The user provided optimizer has to satisfy strict requirements. The arguments of the `fn.optimizer` are:

`par0` a vector of starting values,
`vY` the data provided,

FUN the objective function,
 LB vector of lower bounds for the parameters,
 UB vector of upper bounds for the parameters.
 ... additional arguments.

The output of `fn.optimizer` has to be an object of the class `list` with four named elements:

`pars` a numeric vector where the estimated parameters are stored,
`value` a numeric containing the value of the objective function evaluated at its minimum,
`hessian` a numeric matrix containing the Hessian matrix evaluated at the minimum of the objective function, this is used for inferential purposes,
`convergence` a numeric variable reporting information about the convergence of the optimization.
`convergence = 0` has to indicate successful completion.

The user is allowed to not include the last two elements of the output of the `fn.optimizer` function, that is, the values `hessian = NULL` and `convergence = NULL` are admissible. In the case of `hessian = NULL`, no standard errors will be computed.

Value

A list with, among others, elements:

<code>lFilter</code>	A list containing the output from the filtering procedure. For instance filtered quantiles, hit variables, and losses.
<code>vPn</code>	numeric named vector of estimated parameters.
<code>optimizer</code>	A list with the output from <code>fn.optimizer</code> .
<code>Inference</code>	A list with output from the inferential procedure.

Author(s)

Leopoldo Catania

References

Catania, L, and Luati, A. (2023). "Semiparametric modeling of multiple quantiles." *Journal of Econometrics* doi:[10.1016/j.jeconom.2022.11.002](https://doi.org/10.1016/j.jeconom.2022.11.002).

Examples

```
# Load Microsoft Corporation logarithmic percentage returns from December 8,
# 2010 to November 15, 2018 for a total of T = 2000 observation
data("MSFT")
```

```
#####
##### Estimate DMQ #####
#####
```

```
# Deciles
vTau = seq(0.1, 0.9, 0.1)
```

```

# Reference quantile to the median
iTau_star = 5

# Fix the reference quantile to a constant
FixReference = TRUE

# Estimate DMQ
Fit_solnp = EstimateDMQ(vY = vY,
                        vTau = vTau,
                        iTau_star = iTau_star,
                        FixReference = FixReference,
                        fn.optimizer = fn.solnp,
                        cluster = cluster)

Fit_solnp$vPn
Fit_solnp$optimizer$value

## Not run:
#### Estimate DMQ using different optimizers

# With the DEoptim optimizer

# parallel computation
iG = 7
cluster = makeCluster(iG)

set.seed(123)

# Estimate DMQ
Fit_DEoptim = EstimateDMQ(vY = vY,
                          vTau = vTau,
                          iTau_star = iTau_star,
                          FixReference = FixReference,
                          fn.optimizer = fn.DEoptim,
                          cluster = cluster)

Fit_DEoptim$vPn
Fit_DEoptim$optimizer$value

# Estimate the model with a user defined optimizer.
# Let's use the gosolnp() optimizer from the Rsolnp package.

library("Rsolnp")
fn.gosolnp <- function(par0, vY, FUN, LB, UB, ...) {

  foo = list(...)
  if (!is.null(foo$cluster)) {
    cluster = foo$cluster
    clusterEvalQ(cluster, library(DMQ))
  }

  optimiser = gosolnp(

```

```

    pars = par0,
    fun = FUN, vY = vY,
    n.sim = 1000,
    n.restarts = 5,
    LB = LB,
    UB = UB, control = list(trace = 1),
    ...)

  out = list(pars = optimiser$pars,
            value = tail(optimiser$values, 1),
            hessian = optimiser$hessian,
            convergence = optimiser$convergence)

  return(out)
}

set.seed(123)
# Estimate DMQ
Fit_gosolnp = EstimateDMQ(vY = vY,
                          vTau = vTau,
                          iTau_star = iTau_star,
                          FixReference = FixReference,
                          fn.optimizer = fn.gosolnp,
                          cluster = cluster,
                          smooth = TRUE)

Fit_gosolnp$vPn
Fit_gosolnp$optimizer$value

stopCluster(cluster)

## End(Not run)

```

fn.DEoptim	<i>A wrapper to the DEoptim function of the DEoptim package of Mullen et al. (2011).</i>
------------	--

Description

This function is a wrapper to the [DEoptim](#) function of the DEoptim package of Mullen et al. (2011).

Usage

```
fn.DEoptim(par0, vY, FUN, LB, UB, ...)
```

Arguments

par0	numeric vector of named model coefficients.
vY	numeric vector or matrix of data.
FUN	A function to optimize.
LB	A vector of lower bounds for the parameters.
UB	A vector of upper bounds for the parameters.
...	Additional arguments to provide to DEoptim .

Details

The following control parameters are used: trace = 0, rho = 1, outer.iter = 400, inner.iter = 1800, delta = 1e-08, tol = 1e-08. See the documentation of [DEoptim](#).

Value

It returns a named list with four elements: i) pars: a numeric vector where the estimated parameters are stored, ii) value: a numeric containing the value of the objective function evaluated at its minimum, iii) hessian, a numeric matrix containing the Hessian matrix evaluated at the minimum of the objective function, and iv) convergence a numeric element indicating the convergence (convergence is always reached by [DEoptim](#), i.e. convergence = 1).

Author(s)

Leopoldo Catania

References

Katharine Mullen, David Ardia, David Gil, Donald Windover, James Cline (2011). 'DEoptim': An R Package for Global Optimization by Differential Evolution. Journal of Statistical Software, 40(6), 1-26. doi:10.18637/jss.v040.i06.

Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2010). Differential Evolution with 'DEoptim': An Application to Non-Convex Portfolio Optimization. R Journal, 3(1), 27-34. doi:10.32614/RJ-2011-005.

See Also

help(DEoptim)

fn.optim	<i>A wrapper to the optim function.</i>
----------	---

Description

This function is a wrapper to the standard [optim](#) optimizer with method = "BFGS".

Usage

```
fn.optim(par0, vY, FUN, LB, UB, ...)
```

Arguments

par0	numeric vector of named model coefficients.
vY	numeric vector of data.
FUN	A function to optimize.
LB	A vector of lower bounds for the parameters.
UB	A vector of upper bounds for the parameters.
...	Additional arguments to provide to optim .

Details

The following control parameters are used for control:

- trace = 0
- abstol = 1e-8

See the documentation of [optim](#).

Value

It returns a named list with four elements: i) pars: a numeric vector where the estimated parameters are stored, ii) value: a numeric containing the value of the objective function evaluated at its minimum, iii) hessian, a numeric matrix containing the Hessian matrix evaluated at the minimum of objective function, iv) convergence a numeric element indicating the convergence results of [optim](#).

Author(s)

Leopoldo Catania

See Also

`help(optim)`

fn.solnp	<i>A wrapper to the solnp function of the Rsolnp package of Ghalanos and Theussl (2016).</i>
----------	--

Description

This function is a wrapper to the [solnp](#) function of the Rsolnp package of Ghalanos and Theussl (2016).

Usage

```
fn.solnp(par0, vY, FUN, LB, UB, ...)
```

Arguments

par0	numeric vector of named model coefficients.
vY	numeric vector or matrix of data.
FUN	A function to optimize.
LB	A vector of lower bounds for the parameters.
UB	A vector of upper bounds for the parameters.
...	Additional arguments to provide to solnp .

Details

The following control parameters are used: trace = 0, rho = 1, outer.iter = 400, inner.iter = 1800, delta = 1e-08, tol = 1e-08. See the documentation of [solnp](#).

Value

It returns a named list with four elements: i) pars: a numeric vector where the estimated parameters are stored, ii) value: a numeric containing the value of the objective function evaluated at its minimum, iii) hessian, a numeric matrix containing the Hessian matrix evaluated at the minimum of the objective function, and iv) convergence a numeric element indicating the convergence results of [solnp](#).

Author(s)

Leopoldo Catania

References

Alexios Ghalanos and Stefan Theussl (2015). "Rsolnp: General Non-linear Optimization Using Augmented Lagrange Multiplier Method". R package version 1.16.

See Also

`help(solnp)`

ForecastDMQ

*Forecast with univariate DMQ model***Description**

Compute the H-steap ahead prediction of the quantile processes.

Usage

```
ForecastDMQ(Fit, H)
```

Arguments

Fit	The output of the function EstimateDMQ .
H	numeric, forecast horizon.

Value

A numeric matrix of dimension HxJ, where J is the number of quantiles.

Author(s)

Leopoldo Catania

Examples

```
# Load Microsoft Corporation logarithmic percentage returns from December 8,
# 2010 to November 15, 2018 for a total of T = 2000 observation
data("MSFT")

#####
##### Estimate DMQ #####
#####

# Estimate DMQ at tau_j = 0.05, 0.10, ..., 0.95
# with fixed median as reference quantile.
Fit = EstimateDMQ(vY = vY,
                  vTau = seq(0.05, 0.95, 0.05),
                  iTau_star = 10,
                  FixReference = TRUE,
                  fn.optimizer = fn.solnp)

# Compute 20-step ahead predictions
mQ_pred = ForecastDMQ(Fit, H = 20)

mQ_pred
```

MomentsDMQ

*Estimate conditional moments using DMQ***Description**

Compute DMQ implied conditional moments. At each point in time moments are computed using the discretized distribution implied by the estimated conditional quantiles.

Usage

```
MomentsDMQ(Fit)
```

Arguments

`Fit` The output of the function [EstimateDMQ](#) or [UpdateDMQ](#).

Details

Moments are computed using the following approximation:

$$\mathbb{E}[g(x)] \approx \sum_{j=1}^J (\tau_j - \tau_{j-1}) g(\hat{q}_t^{\tau_j}),$$

with $\tau_0 = 0$, where $\hat{q}_t^{\tau_j}$ are estimated quantiles.

Value

A list of four elements:

<code>mMoments</code>	a Tx4 numeric matrix with columns containing first, second, third, and fourth moments.
<code>mCenterdMoments</code>	a Tx4 numeric matrix with columns containing first, second, third, and fourth central moments.
<code>vSkew</code>	a numeric vector of length T of estimated skewness coefficients.
<code>vKurt</code>	a numeric vector of length T estimated kurtosis coefficients.

Author(s)

Leopoldo Catania

Examples

```
# Load Microsoft Corporation logarithmic percentage returns from December 8,
# 2010 to November 15, 2018 for a total of T = 2000 observation
data("MSFT")

#####
##### Estimate DMQ #####
#####

# Estimate DMQ on the in sample period
Fit = EstimateDMQ(vY = vY,
                  vTau = seq(0.01, 0.99, 0.01),
                  iTau_star = 50,
                  FixReference = TRUE,
                  fn.optimizer = fn.solnp)

# Compute estimated moments

Moments = MomentsDMQ(Fit)
```

MSFT

data: Microsoft Corporation logarithmic percentage returns from December 8, 2010 to November 15, 2018 for a total of T = 2000 observation downloaded from Yahoo finance.

Description

This dataset is the one used in Catania and Luati (2023).

Dow Jones 30 Constituents closing value log returns from 1987-03-16 to 2009-02-03 from Yahoo Finance. Note that AIG was replaced by KFT (Kraft Foods) on September 22, 2008. This is not reflected in this data set as that would bring the starting date of the data to 2001.

Usage

```
data("MSFT")
```

Format

An xts object of 2000 logarithmic percentage returns.

Source

Yahoo Finance

References

Catania, L, and Luati, A. (2023). "Semiparametric modeling of multiple quantiles." Journal of Econometrics [doi:10.1016/j.jeconom.2022.11.002](https://doi.org/10.1016/j.jeconom.2022.11.002).

SimulatedMQ

Simulate from the DMQ model

Description

Approximate simulation from the DMQ model. Allows to simulate quantiles and observations.

Usage

```
SimulateDMQ(iT, vQ_0, vTau, iTau_star, vPn, ScalingType = "InvSqrt", fSim = NULL)
```

Arguments

iT	Number of observations to simulate.
vQ_0	numeric vector of limiting quantiles.
vTau	numeric vector of length Jx1 containing probability levels at which quantiles are estimated.
iTau_star	Integer indicating the position in vTau where the reference quantile is placed. For instance, if vTau = seq(0.01, 0.99, 0.01) then iTau_star = 50 means that the median is used as the reference quantile.
vPn	numeric named vector of length 4x1 with starting values for the optimizer. For example vPn = c("phi" = 0.9, "gamma" = 0.05, "alpha" = 0.01, "beta" = 0.7).
ScalingType	character Indicating the scaling mechanism for the conditional quasi score. Possible choices are "Identity", "Inv","InvSqrt". When ScalingType = "InvSqrt" quasi scores are scaled by their standard deviation. When ScalingType = "Inv" quasi scores are scaled by their variance. When ScalingType = "Identity" quasi scores are not scaled. Default value ScalingType = "InvSqrt".
fSim	function to simulate from the discretized distribution implied by the simulated quantiles. By default fSim = NULL meaning that an internal simulation scheme is employed. See details.

Details

Given a set of simulated quantiles a Uniform variable drawn. The discretized quantile function is linearly interpolated at the simulated Uniform draw to obtain an observations. When the Uniform draw is outside the range spanned by vTau a Gaussian quantile function is used. The mean and variance of the Gaussian quantile distribution are set to those implied by the simulated quantiles using the same scheme of [MomentsDMQ](#).

Value

A list with two elements:

`vY` A numeric vector of T simulated observations.
`mQ` A numeric TxJ matrix of simulated quantiles.

Author(s)

Leopoldo Catania

Examples

```
set.seed(123)

# Simulate 500 observations from the DMQ model.

# Use the percentiles
vTau = seq(0.01, 0.99, 0.01)

# Median as reference quantile
iTau_star = 50

# Standard Gaussian limiting distribution
vQ_0 = qnorm(vTau)

# vector of parameters
vPn = c("phi" = 0.95, "gamma" = 0.10, "alpha" = 0.01, "beta" = 0.7)

lSim = SimulatedMQ(iT = 500, vQ_0, vTau, iTau_star, vPn)

plot.ts(lSim$vY)
plot.ts(lSim$mQ, plot.type = "single")
```

UpdateDMQ

Update filtered quantiles

Description

Filter dynamic quantiles using an estimated model and an updated dataset.

Usage

```
UpdateDMQ(Fit, vY)
```

Arguments

`Fit` The output of the function [EstimateDMQ](#).
`vY` numeric vector containing past and new observations.

Details

The function can be used to compute a sequence of one-step-ahead rolling predictions, without updating the parameters of the model, see Examples.

Value

An output like the one of [EstimateDMQ](#) with updated quantile estimated.

Author(s)

Leopoldo Catania

Examples

```
# Load Microsoft Corporation logarithmic percentage returns from December 8,
# 2010 to November 15, 2018 for a total of T = 2000 observation
data("MSFT")

# Divide the sample in two equal parts
vY_is = vY[1:1000]

#####
##### Estimate DMQ #####
#####

# Estimate DMQ over the deciles on the in sample period
Fit = EstimateDMQ(vY = vY_is,
                  vTau = seq(0.1, 0.9, 0.1),
                  iTau_star = 5,
                  FixReference = TRUE,
                  fn.optimizer = fn.solnp)

# compute a sequence of one-step-ahead rolling predictions over the out of sample

Roll = UpdateDMQ(Fit, vY)

# one steap ahead predictions from time t = 1001 to 2001 are
mForecast = t(Roll$filter$mQ)[1001:2001, ]
```

Index

- * **datasets**

- MSFT, [12](#)

- * **package**

- DMQ-package, [2](#)

DEoptim, [3](#), [6](#), [7](#)

DMQ (DMQ-package), [2](#)

DMQ-package, [2](#)

EstimateDMQ, [2](#), [10](#), [11](#), [14](#), [15](#)

fn.DEoptim, [3](#), [6](#)

fn.optim, [8](#)

fn.solnp, [9](#)

ForecastDMQ, [10](#)

function, [7–9](#)

MomentsDMQ, [11](#), [13](#)

MSFT, [12](#)

optim, [8](#)

SimulateDMQ, [13](#)

solnp, [9](#)

UpdateDMQ, [11](#), [14](#)

vY (MSFT), [12](#)