Package ‘Qest’

January 23, 2024

Type Package

Title Quantile-Based Estimator

Version 1.0.1

Author Gianluca Sottile [aut, cre],
Paolo Frumento [aut]

Maintainer Gianluca Sottile <gianluca.sottile@unipa.it>

Description
Quantile-based estimators (Q-estimators) can be used to fit any parametric distribution, using its quantile function. Q-estimators are usually more robust than standard maximum likelihood estimators. The method is described in: Sottile G. and Frumento P. (2022). Robust estimation and regression with parametric quantile functions. <doi:10.1016/j.csda.2022.107471>.

Depends pch, survival, matrixStats, methods, utils

URL https://www.sciencedirect.com/science/article/abs/pii/S0167947322000512

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-01-23 13:42:53 UTC

R topics documented:

Qest-package *Quantile-Based Estimator*

Description

Quantile-based estimators (Q-estimators) can be used to fit any parametric distribution, using its quantile function. Q-estimators are usually more robust than standard maximum likelihood estimators. The method is described in: Sottile G. and Frumento P. (2022). Robust estimation and regression with parametric quantile functions. <doi:10.1016/j.csda.2022.107471>.

Details

Package:	Qest
Type:	Package
Version:	1.0.1
Date:	2024-01-22
License:	GPL-2

The DESCRIPTION file:

Package:	Qest
Type:	Package
Title:	Quantile-Based Estimator
Version:	1.0.1
Authors@R:	c(person("Gianluca", "Sottile", role=c("aut", "cre"), email = "gianluca.sottile@unipa.it"), person("Paolo", "Fru
Author:	Gianluca Sottile [aut, cre], Paolo Frumento [aut]
Maintainer:	Gianluca Sottile <gianluca.sottile@unipa.it>
Description:	Quantile-based estimators (Q-estimators) can be used to fit any parametric distribution, using its quantile funct
Depends:	pch, survival, matrixStats, methods, utils
URL:	https://www.sciencedirect.com/science/article/abs/pii/S0167947322000512
License:	GPL (>= 2)
Encoding:	UTF-8

Index of help topics:

```
Qcoxph                  Q-Estimation of Proportional Hazards Regression
                        Models
Qcoxph.control          Auxiliary for Controlling Qcoxph Fitting
Qest                    Q-Estimation
Qest-package            Quantile-Based Estimator
Qest.control            Auxiliary for Controlling Qest Fitting
Qfamily                 Family Objects for Qest
Qlm                     Q-Estimation of Linear Regression Models
```

Author(s)

Gianluca Sottile [aut, cre], Paolo Frumento [aut]

Maintainer: Gianluca Sottile <gianluca.sottile@unipa.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

See Also

Qest, Qlm, Qcoxph

Examples

```
## Not run:
Qest(y ~ x, Q, start) # General-purpose Q-estimator
Qlm(y ~ x) # Q-estimation of linear models
Qcoxph(Surv(time, event) ~ x) # Q-estimation of proportional hazards models

## End(Not run)
```

---

invQ *Inverse of Quantile Function*

Description

Auxiliary function to compute cumulative distribution function (CDF) by inverting a quantile function.

Usage

```
invQ(Q, theta, y, data, n.it = 17)
```

Arguments

Q	any parametric quantile function of the form `Q(theta, tau, data)`.
theta	a vector of model parameters.
y	vector of observations to evaluate the CDF.
data	data frame containing the variables used in the Q() function.
n.it	the number of iterations (see “details”).

Details

Given a parametric quantile function $Q(\tau|\theta)$, the CDF is defined as $F(y|\theta) = Q^{-1}(y|\theta)$. Alternatively, $F(y|\theta)$ corresponds to the value $\tau*$ such that $Q(\tau * |\theta) = y$. Starting from $\tau = 0.5$, a bisection algorithm is used to evaluate numerically $\tau*$. The maximum error is given by `1/2^(n.it + 1)`.

Value

a vector of CDF values.

Author(s)

Maintainer: Gianluca Sottile <gianluca.sottile@unipa.it>

See Also

Qest

Examples

```
# Ex. 1 Normal model

# Quantile function of a linear model.
Qlinmod <- function(theta, tau, data){
  sigma <- exp(theta[1])
  beta <- theta[-1]
  X <- model.matrix( ~ x1 + x2, data = data)
  qnorm(tau, X %*% beta, sigma)
}

n <- 100
x1 <- rnorm(n)
x2 <- runif(n,0,3)
theta <- c(1,4,1,2)

# generate the data
U <- runif(n)
y <- Qlinmod(theta, U, data.frame(x1,x2))

# Given y and theta, evaluate U = F(y)
invQ(Qlinmod, theta, y, data.frame(x1,x2))
```

Qcoxph *Q-Estimation of Proportional Hazards Regression Models*

Description

Fit proportional hazards regression models using Q-estimation.

Usage

```
Qcoxph(formula, weights, start, data, knots, wtau = NULL,
    control = Qcoxph.control(), ...)
```

Arguments

- `formula` an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Use `Surv(time, event) ~ x`, if the data are right-censored, and `Surv(time, time2, event) ~ x`, if the data are right-censored and left-truncated (`time < time2`, `time` can be `-Inf`).
- `weights` an optional vector of weights to be used in the fitting process. The weights will always be normalized to sum to the sample size. This implies that, for example, using double weights will *not* halve the standard errors.
- `start` optional starting values for the coefficients of the linear predictor.
- `data` an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `Qcoxph` is called.
- `knots` knots to create the basis of a piecewise linear function. If `knots` is a vector of at least two elements, it is used to identify the exact position of *all* knots, including boundaries. If `knots` is a scalar, its value is used to determine the number of internal knots (`knots = 0` is allowed, and fits an Exponential model). If `knots` is missing, by default `max(1, min(floor(n.events/30), 3))` internal knots are used. Unless a vector of `knots` is provided by the user, the "optimal" position of the knots will be identified using the method described in Muggeo (2008). If this fails, the knots are positioned at the empirical quantiles of the observed events.
- `wtau` an optional function that assigns a different weight to each quantile. By default, all quantiles in (0,1) have the same weight. Please check the documentation of `wtrunc` for built-in weighting functions.
- `control` a list of operational parameters. This is usually passed through `Qcoxph.control`.
- `...` additional arguments for `wtau`.

Details

This function estimates a proportional hazards model, allowing for right-censored and left-truncated data. The syntax and output of `Qcoxph` are almost identical to those of `coxph`, but the parameters are estimated using Q-estimation (Sottile and Frumento, 2020). This method can be used to obtain outlier-robust estimators of the regression coefficients.

The quantile function of a proportional hazards model is given by

$$Q(\tau|x) = H0^{-1}(-exp{-x'\beta}log(1-\tau))$$

where $H0$ is the baseline cumulative hazard function. In `Qcoxph`, $H0$ is parametrized by a piecewise linear function identified by the provided `knots`. As the number of knots increases, the baseline hazard becomes arbitrarily flexible.

Estimation is carried out by finding the zeroes of a set of integrals equation. The optional argument `wtau` permits assigning a different weight to each quantile in (0,1). It is possible to choose `wtau` to be a discontinuous function (e.g., `wtau = function(tau){tau < 0.95}`). However, this may occasionally result in poorly estimated of the standard errors.

The estimation algorithm is briefly described in the documentation of `Qcoxph.control`.

Value

an object of classes "Qcoxph", "coxph", and "Qest". See `coxph.object` for details. All the S3 methods that are available for "coxph" objects will also work with a "Qcoxph" object.

An object of class "Qcoxph" is a list containing at least the following components:

`coefficients` a named vector of coefficients.

`var` the covariance matrix of the coefficients.

`iter` number of iterations used.

`linear.predictors` the vector of linear predictors, one per subject. Note that this vector has not been centered, see `predict.coxph` for details.

`residuals` the martingale residuals.

`means` vector of column means of the X matrix. Subsequent survival curves are adjusted to this value.

`n` the number of observations used in the fit.

`nevent` the number of events used in the fit.

`concordance` a vector of length 6, containing the number of pairs that are concordant, discordant, tied on x, tied on y, and tied on both, followed by the standard error of the concordance statistic.

`terms, assign, formula, call, y` other objects used for prediction.

`obj.function` the objective function of the model. Please, interpret with care: read the note in the documentation of `Qest`.

`internal` internal objects.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>, Gianluca Sottile <gianluca.sottile@unipa.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

Muggeo VMR (2008). *Segmented: an R package to fit regression models with broken-line relationships.* R News 8/1, 20–25.

See Also

`Qest`, for general Q-estimation, and `Qlm`, for Q-estimation of linear models.

Examples

```
# A proportional-hazards Weibull model

n <- 100
x <- runif(n,0,3)
shape <- 2
t <- rweibull(n, shape = shape, scale = (1/exp(2 + 2*x))^(1/shape)) # time-to-event
c <- runif(n,0,1) # censoring variable
y <- pmin(t,c) # observed response
d <- (t <= c) # event indicator

require(survival)
m1 <- coxph(Surv(y,d) ~ x) # standard Cox model
m2 <- Qcoxph(Surv(y,d) ~ x) # Q-estimator
```

`Qcoxph.control` *Auxiliary for Controlling Qcoxph Fitting*

Description

Auxiliary function for controlling `Qcoxph` fitting. Estimation proceeds in three steps: (i) evaluation of starting points; (iia) stochastic gradient-based optimization (iib) standard gradient-based optimization; and (iii) Newton-Raphson. Step (i) is based on a preliminary fit of a Cox model via `coxph`. Steps (iia) and (iib) find an approximate solution, and make sure that the Jacobian matrix is well-defined. Finally, step (iii) finds a more precise solution.

Usage

```
Qcoxph.control(tol = 1e-8, maxit, safeit, alpha0, display = FALSE)
```

Arguments

`tol`	tolerance for convergence of Newton-Raphson algorithm, default is 1e-8.
`maxit`	maximum number of iterations of Newton-Raphson algorithm. If not provided, a default is computed as `50 + 25*npar`, where `npar` is the total number of parameters.
`safeit`	maximum number of iterations of gradient-search algorithm. If not provided, a default is computed as `10 + 5*npar`, where `npar` is the total number of parameters.
`alpha0`	step size for the preliminary gradient-based iterations. If estimation fails, you can try choosing a small value of `alpha0`. If `alpha0` is missing, an adaptive choiche will be made internally.
`display`	Logical. If `TRUE`, tracing information on the progress of the optimization is printed on screen. Default is `FALSE`.

Details

If called with no arguments, `Qcoxph.control()` returns a list with the current settings of these parameters. Any arguments included in the call sets those parameters to the new values, and then silently returns.

Value

A list with named elements as in the argument list

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it> Paolo Frumento <paolo.frumento@unipi.it>

See Also

`Qcoxph`

`Qest` *Q-Estimation*

Description

An implementation of the quantile-based estimators described in Sottile and Frumento (2022).

Usage

```
Qest(formula, Q, weights, start, data, ntau = 199, wtau = NULL,
  control = Qest.control(), ...)
```

Arguments

`formula`	a two-sided formula of the form `y ~ x`. Note that the parametric model is identified through `Q`, and not through `formula`, that only identifies the response and the predictors. Use `Surv(time, event)`, if the data are right-censored, and `Surv(start, stop, event)`, if the data are right-censored and left-truncated (`start` < `stop`, `start` can be `-Inf`).
`Q`	a parametric quantile function of the form `Q(theta, tau, data)`. Alternatively, a character string naming a `Qfamily` function, a `Qfamily` function itself, or the result of a call to a `Qfamily` function. See `Qfamily` for details.
`weights`	an optional vector of weights to be used in the fitting process. The weights will always be normalized to sum to the sample size. This implies that, for example, using double weights will *not* halve the standard errors.
`start`	a vector of starting values. `NA`s are allowed, but will be internally replaced by zeroes. Make sure that the quantile function is well-defined at `theta` = `start`. The size of `start` is also used to identify the number of parameters in the model. You *must* supply starting points, unless you are fitting a model defined by a `Qfamily`.

data	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `Qest` is called.
ntau	the number of points for numerical integration (see "Details"). Default `ntau = 199`.
wtau	an optional function that assigns a different weight to each quantile. By default, all quantiles in (0,1) have the same weight. Please check the documentation of `wtrunc` for built-in weighting functions.
control	a list of operational parameters. This is usually passed through `Qest.control`.
...	additional arguments for `wtau` and `Q`.

Details

A parametric model, $Q(\tau|\theta, x)$, is used to describe the conditional quantile function of an outcome Y, given a vector x of covariates. The model parameters, θ, are estimated by minimizing the (weighted) integral, with respect to τ, of the loss function of standard quantile regression. If the data are censored or truncated, θ is estimated by solving a set of estimating equations. In either case, numerical integration is required to calculate the objective function: a grid of `ntau` points in `(0,1)` is used. The estimation algorithm is briefly described in the documentation of `Qest.control`.

The optional argument `wtau` can be used to attribute a different weight to each quantile. Although it is possible to choose `wtau` to be a discontinuous function (e.g., `wtau = function(tau){tau < 0.95}`), this may occasionally result in poorly estimated standard errors.

The quantile function `Q` must have at least the following three arguments: `theta, tau, data`, in this order. The first argument, `theta`, is a vector (not a matrix) of parameters' values. The second argument, `tau`, is the order of the quantile. When `Q` receives a `n*ntau` matrix of `tau` values, it must return a `n*ntau` matrix of quantiles. The third argument, `data`, is a data frame that includes the predictors used by `Q`.

If `Q` is identified by one `Qfamily`, everything becomes much simpler. It is not necessary to implement your own quantile function, and the starting points are not required. Note that `ntau` is ignored if `Q = Qnorm` or `Q = Qunif`.

Please check the documentation of `Qfamily` to see the available built-in distributions. A convenient Q-based implementation of the standard linear regression model is offered by `Qlm`. Proportional hazards models are implemented in `Qcoxph`.

Value

a list with the following elements:

coefficients	a named vector of coefficients.
std.errs	a named vector of estimated standard errors.
covar	the estimated covariance matrix of the estimators.
obj.function	the value of the minimized loss function. If the data are censored or truncated, a meaningful loss function which, however, is not the function being minimized (see "Note").

`ee`	the values of the estimating equations at the solution. If the data are neither censored nor truncated, the partial derivatives of the loss function.
`jacobian`	the jacobian at the solution. If the data are neither censored nor truncated, the matrix of second derivatives of the loss function.
`CDF, PDF`	the fitted values of the cumulative distribution function (CDF) and the probability density function (PDF).
`converged`	logical. The convergence status.
`n.it`	the number of iterations.
`internal`	internal elements.
`call`	the matched call.

Note

NOTE 1. If the data are censored or truncated, estimation is carried out by solving estimating equations, and no associated loss is present. In this case, a meaningful value of `obj.function` is the integrated loss [equation 1 of Sottile and Frumento (2022)] in which the indicator function $I(y \leq Q(\tau|\theta, x))$ has been replaced with one of the expressions presented in equations 6 and 7 of the paper. The resulting loss, however, is not the function being minimized.

NOTE 2. To prevent computational problems, avoid situations in which some of the estimated parameters are expected to be very small or very large. For example, standardize the predictors, and normalize the response. Avoid as much as possible parameters with bounded support. For example, model a variance/rate/shape parameter on the log scale, e.g., $\sigma = exp(\theta)$. Carefully select the starting points, and make sure that `Q(start, ...)` is well-defined. If `Q` is identified by one `Qfamily`, all these recommendations can be ignored.

NOTE 3. You should *not* use `Qest` to fit parametric models describing discrete distributions, where the quantile function is piecewise constant. You can try, but the optimization algorithm will most likely fail. The predefined family `Qpois` allows to fit a Poisson distribution by using a continuous version of its quantile function (see `Qfamily`).

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>, Gianluca Sottile <gianluca.sottile@unipa.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

See Also

`Qest.control`, for operational parameters, and `summary.Qest`, for model summary. `Qfamily`, for the available built-in distributions. `wtrunc` for built-in weighting functions (`wtau` argument). `Qlm`, for Q-estimation of the standard normal (linear) regression model; `Qcoxph`, for proportional hazards models.

Examples

```
# Ex1. Normal model

# Quantile function of a linear model
Qlinmod <- function(theta, tau, data){
  sigma <- exp(theta[1])
  beta <- theta[-1]
  X <- model.matrix( ~ x1 + x2, data = data)
  qnorm(tau, X %*% beta, sigma)
}

n <- 100
x1 <- rnorm(n)
x2 <- runif(n,0,3)
theta <- c(1,4,1,2)
y <- Qlinmod(theta, runif(n), data.frame(x1,x2)) # generate the data

m1 <- Qest(y ~ x1 + x2, Q = Qlinmod, start = c(NA,NA,NA,NA)) # User-defined quantile function
summary(m1)

m2 <- Qest(y ~ x1 + x2, Q = Qnorm) # Qfamily
summary(m2)

m3 <- Qlm(y ~ x1 + x2)
summary(m3) # using 'Qlm' is much simpler and faster, with identical results




# Ex2. Weibull model with proportional hazards

# Quantile function
QWeibPH <- function(theta, tau, data){
  shape <- exp(theta[1])
  beta <- theta[-1]
  X <- model.matrix(~ x1 + x2, data = data)
  qweibull(tau, shape = shape, scale = (1/exp(X %*% beta))^(1/shape))
}

n <- 100
x1 <- rbinom(n,1,0.5)
x2 <- runif(n,0,3)
theta <- c(2,-0.5,1,1)

t <- QWeibPH(theta, runif(n), data.frame(x1,x2)) # time-to-event
c <- runif(n,0.5,1.5) # censoring variable
y <- pmin(t,c) # observed response
d <- (t <= c) # event indicator

m1 <- Qest(Surv(y,d) ~ x1 + x2, Q = QWeibPH, start = c(NA,NA,NA,NA))
summary(m1)
```

```
m2 <- Qcoxph(Surv(y,d) ~ x1 + x2)
summary(m2) # using 'Qcoxph' is much simpler and faster (but not identical)




# Ex3. A Gamma model

# Quantile function
Qgm <- function(theta, tau, data){
  a <- exp(theta[1])
  b <- exp(theta[2])
  qgamma(tau, shape = a, scale = b)
}
n <- 100
theta <- c(2,-1)
y <- rgamma(n, shape = exp(theta[1]), scale = exp(theta[2]))

m1 <- Qest(y ~ 1, Q = Qgm, start = c(NA, NA)) # User-defined quantile function
m2 <- Qest(y ~ 1, Q = Qgamma) # Qfamily
m3 <- Qest(y ~ 1, Q = Qgamma, wtau = function(tau, h) dnorm((tau - 0.5)/h), h = 0.2)
# In m3, more weight is assigned to quantiles around the median




# Ex4. A Poisson model

# Quantile function
n <- 100
x1 <- runif(n)
x2 <- rbinom(n,1,0.5)
y <- rpois(n, exp(1.5 -0.5*x1 + x2))
m1 <- Qest(y ~ x1 + x2, Q = Qpois) # Use a Qfamily! See "Note"
m2 <- Qest(y + runif(n) ~ x1 + x2, Q = Qpois) # Use jittering! See the documentation of "Qfamily"
```

`Qest.control` *Auxiliary for Controlling Qest Fitting*

Description

Auxiliary function for controlling `Qest` fitting. Estimation proceeds in three steps: (i) evaluation of starting points; (iia) stochastic gradient-based optimization (iib) standard gradient-based optimization; and (iii) Newton-Raphson. Step (i) is initialized at the provided starting values (the `start` argument of `Qest`), and utilizes a preliminary flexible model, estimated with `pchreg`, to generate a cheap guess of the model parameters. If you have good starting points, you can skip step (i) by setting `restart = FALSE`. Steps (iia) and (iib) find an approximate solution, and make sure that the Jacobian matrix is well-defined. Finally, step (iii) finds a more precise solution.

Usage

```
Qest.control(tol = 1e-8, maxit, safeit, alpha0, display = FALSE, restart = FALSE)
```

Arguments

`tol`	tolerance for convergence of Newton-Raphson algorithm, default is 1e-8.
`maxit`	maximum number of iterations of Newton-Raphson algorithm. If not provided, a default is computed as `50 + 25*npar`, where `npar` is the number of parameters.
`safeit`	maximum number of iterations of gradient-search algorithm. If not provided, a default is computed as `10 + 5*npar`, where `npar` is the number of parameters.
`alpha0`	step size for the preliminary gradient-based iterations. If estimation fails, you can try choosing a small value of `alpha0`. If `alpha0` is missing, an adaptive choiche will be made internally.
`display`	Logical. If `TRUE`, tracing information on the progress of the optimization is printed on screen. Default is `FALSE`.
`restart`	Logical. If `FALSE` (the default), step (i) is not performed, and the provided starting points are directly passed to step (ii). This may save you some time, but is not recommended unless you are confident about your choice of initial values. When `restart = TRUE`, the provided starting points are used to initialize step (i).

Details

If called with no arguments, `Qest.control()` returns a list with the current settings of these parameters. Any arguments included in the call sets those parameters to the new values, and then silently returns.

Value

A list with named elements as in the argument list

Note

Step (i) is not performed, and `restart` is ignored, if the quantile function is one of the available `Qfamily`.

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it> Paolo Frumento <paolo.frumento@unipi.it>

See Also

`Qest` and `Qlm`

Qfamily *Family Objects for Qest*

Description

Family objects are used to specify the model to be fitted by Qest.

Usage

```
Qnorm()
Qgamma()
Qpois(offset = NULL)
Qunif(min = TRUE)
```

Arguments

offset	an optional vector of offsets for a Poisson model.
min	logical. If TRUE, fit a $U(a, b)$ distribution. If FALSE, fit a $U(0, b)$ distribution.

Details

A Qfamily object can be used to identify a certain type of distribution within a call to Qest. You can supply either the name of the family, or the function itself, or a call to it. For example, the following are equivalent: Qest(formula, "Qpois"), Qest(formula, Qpois), and Qest(formula, Qpois()). The latter syntax can be used to pass additional arguments, if any.

The Qnorm family fits a normal homoskedastic model in which the mean is described by a linear predictor. The parameters are: log(sigma), beta. Qest(formula, Qnorm) is equivalent to Qlm(formula), but returns a very basic output. However, Qest allows for censored and truncated data, while Qlm does not.

The Qgamma family fits a Gamma distribution in which the log-scale is modeled by a linear predictor. The model parameters are: log(shape), beta.

The Qpois family fits a Poisson distribution in which the log-rate is modeled by a linear predictor. In reality, to obtain a continuous quantile function, qpois is replaced by the inverse, *with respect to* y, of the upper regularized gamma function, $Q(y, \lambda)$. It is recommended to apply Qpois to a jittered response (i.e., y + runif(n)).

The Qunif family fits a Uniform distribution $U(a, b)$ in which both a and b are modeled by linear predictors. The design matrix, however, is the same for a and b. Use Qunif(min = FALSE) to fit a $U(0, b)$ model. The parameters are: beta_a, beta_b, or only beta_b if min = FALSE.

The families Qnorm and Qgamma can be used when the data are censored or truncated, while Qpois and Qunif cannot. All families can be estimated without covariates, using formula = ~ 1.

Value

An object of class "Qfamily" that contains all the necessary information to be passed to Qest.

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it>, Paolo Frumento <paolo.frumento@unipi.it>

See Also

Qest.

Examples

```
n <- 250
x <- runif(n)
eta <- 1 + 2*x # linear predictor

# Normal model
y <- rnorm(n, eta, exp(1))
m1 <- Qest(y ~ x, Qnorm)
# Use Qlm(y ~ x) instead!

# Gamma model
y <- rgamma(n, shape = exp(1), scale = exp(eta))
m2 <- Qest(y ~ x, Qgamma)

# Poisson model
y <- rpois(n, exp(eta))
m3 <- Qest(y ~ x, Qpois)
m4 <- Qest(y + runif(n) ~ x, Qpois) # Jittering is recommended

# Uniform model
y <- runif(n, 0, eta)
m5 <- Qest(y ~ x, Qunif(min = TRUE))  # U(a,b)
m6 <- Qest(y ~ x, Qunif(min = FALSE)) # U(0,b)
```

Qlm *Q-Estimation of Linear Regression Models*

Description

Use Q-estimation to fit a Normal model in which the mean is a linear function of the predictors, and the variance is constant.

Usage

```
Qlm(formula, data, subset, weights, na.action, start = NULL, contrasts = NULL,
  wtau = NULL, control = Qest.control(), ...)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `Qlm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. The weights will always be normalized to sum to the sample size. This implies that, for example, using double weights will *not* halve the standard errors.
`na.action`	a function which indicates what should happen when the data contain `NA`s. See `lm`.
`start`	optional starting values for the regression coefficients.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`wtau`	an optional function that assigns a different weight to each quantile. By default, all quantiles in (0,1) have the same weight. Please check the documentation of `wtrunc` for built-in weighting functions.
`control`	a list of operational parameters. See `Qest.control` for details.
`...`	additional arguments for `wtau`.

Details

This function is used exactly as `lm`, but estimates the model parameters as in `Qest`. Using Q-estimation allows to obtain outlier-robust estimators of the regression coefficients. The optional argument `wtau` permits assigning a different weight to each quantile in (0,1). It is possible to choose `wtau` to be a discontinuous function (e.g., `wtau = function(tau){tau < 0.95}`). However, this may occasionally result in poorly estimated of the standard errors.

Note that `Qlm`, like `lm`, does not support censored or truncated data.

Value

Qlm returns an object of classes "Qlm", "lm", and "Qest". The generic accessor functions `summary`, `coefficients`, `fitted.values`, and `residuals` can be used to extract infromation from a "Qlm" object.

An object of class "Qlm" is a list containing at least the following elements:

`coefficients`	a named vector of coefficients.
`std.errs`	a named vector of standard errors.
`covar`	the estimated covariance matrix of the estimators.
`dispersion`	the estimated dispersion parameter (residual variance).
`residuals`	the working residuals.
`rank`	the estimated degrees of freedom.

`fitted.values`	the fitted values.
`df.residual`	the residual degrees of freedom.
`obj.function`	the value of the minimized loss function.
`gradient`	the first derivatives of the minimized loss function.
`hessian`	the matrix of second derivatives of the minimized loss function.
`convergence`	logical. The convergence status.
`n.it`	the number of iterations.
`control`	control parameters.
`xlevels`	(only where relevant) a record of the levels of the factors used in fitting.
`call`	the matched call.
`terms`	the “terms” object used.
`model`	if requested (the default), the model frame used.

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it>, Paolo Frumento <paolo.frumento@unipi.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

See Also

`Qest`, for general Q-estimation.

Examples

```
set.seed(1234)
n <- 100
x1 <- rnorm(n)
x2 <- runif(n,0,3)
theta <- c(1,4,1,2)
y <- rnorm(n, 4 + x1 + 2*x2, 1)

m1 <- Qlm(y ~ x1 + x2)
summary(m1)
```

`Qlm.fit` *Fitter Functions for Quantile-based Linear Models*

Description

This is the basic computing engine called by “Qlm” used to fit quantile-based linear models. This function should only be used directly by experienced users.

Usage

```
Qlm.fit(y, X, w = rep(1, nobs), start = NULL, wtau = NULL,
  control = Qest.control(), ...)
```

Arguments

`y`	vector of observations of length n.
`X`	design matrix of dimension n * p.
`w`	an optional vector of weights to be used in the fitting process.
`start`	starting values for the parameters in the linear predictor.
`wtau`	an optional function that assigns a different weight to each quantile. By default, all quantiles in (0,1) have the same weight.
`control`	a list of operational parameters. This is usually passed through `Qest.control`.
`...`	additional arguments for `wtau`.

Value

a “list” with components

`coefficients`	p vector
`std.errs`	p vector
`covar`	p x p matrix
`dispersion`	estimated dispersion parameter
`residuals`	n vector
`rank`	integer, giving the rank
`fitted.values`	n vector
`qr`	the QR decomposition, see “qr”
`df.residual`	degrees of freedom of residuals
`obj.function`	the minimized loss function
`gradient`	p vector
`hessian`	p x p matrix
`convergence`	logical. The convergence status
`n.it`	the number of iterations
`control`	control elements

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it>, Paolo Frumento <paolo.frumento@unipi.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

See Also

Qlm

Examples

```
# Ex. 1 Normal model

set.seed(1234)
n <- 100
x1 <- rnorm(n)
x2 <- runif(n,0,3)
y <- rnorm(n, 4 + x1 + 2*x2, 1)
X <- cbind(1, x1, x2)
w <- rep.int(1, n)

m <- Qlm.fit(y = y, X = X, w = w, control = Qest.control(display = TRUE))
```

---

summary.Qest *Summarizing Q-estimators*

Description

Summary method for class “Qest”.

Usage

```
## S3 method for class 'Qest'
summary(object, covar = FALSE, ...)
```

Arguments

object	an object of class “Qest”.
covar	logical; if TRUE, the variance covariance matrix of the estimated parameters is returned.
...	for future methods.

Details

This function returns a summary of the most relevant information on model parameters, standard errors, and convergence status.

Value

The function `summary.Qest` computes and returns a list of summary statistics of the fitted model given in object, using the "call" and "terms" from its argument, plus

`coefficients`	a matrix with 4 columns reporting the estimated coefficients, the estimated standard errors, the corresponding z-values (coef/se), and the two-sided p-values.
`obj.function`	the value of the minimized loss function (see `Qest` for details).
`n`	the number of observations.
`npar`	the number of free parameters.
`iter`	the number of iterations.
`covar`	only if `covar = TRUE`, the estimated covariance matrix.
`call`	the matched call.
`type`	a character string defined as follows: `"c"` for right-censored data; `"ct"` for left-truncated, right-censored data; and `"u"` otherwise.

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it>

References

Sottile G, and Frumento P (2022). *Robust estimation and regression with parametric quantile functions.* Computational Statistics and Data Analysis. <doi:10.1016/j.csda.2022.107471>

See Also

`Qest`, for model fitting.

Examples

```
# Quantile function of an Exponential model
Qexp <- function(theta, tau, data){
  qexp(tau, exp(theta))
}

y <- rexp(100, exp(1))
m1 <- Qest(y ~ 1, Q = Qexp, start = NA)
summary(m1)
summary(m1, covar = TRUE)
```

wtrunc *Weighting Function for* Qest, Qlm, *and* Qcoxph.

Description

This function can be used within a call to Qest, Qlm, or Qcoxph to assign a different weight to each quantile.

Usage

```
wtrunc(tau, delta.left = 0.05, delta.right = 0.05, smooth = FALSE, sigma = 0.01)
```

Arguments

- tau: a vector of quantiles.
- delta.left, delta.right: proportion of quantiles to be removed from the left and righ tail. The weighting function is 1 in the interval (delta.left, 1 - delta.right), and zero elsewhere. Default is delta.left = 0.05 and delta.right = 0.05. When a weighting function is used to counteract the effect of extreme observations, delta.left is a guess for the proportion of outliers on the left tail; and delta.right is a guess for the proportion of outliers on the right tail.
- smooth: if smooth = TRUE the indicator functions used to construct wtrunc(tau) are replaced by integrated Gaussian kernels. Default smooth = FALSE.
- sigma: the bandwith of a Gaussian kernel. This parameter controls the smoothness of the weighting function, and is ignored if smooth = FALSE. Default sigma = 0.01.

Details

Within a call to Qest, Qlm, or Qcoxph, one may want to assign a different weight to each quantile through the optional argument wtau. This can be done for reasons of efficiency, or to counteract the presence of outliers. While wtau can be any user-defined function, one can use wtrunc as a shortcut to construct a weighting function that truncates a certain subset of quantiles in the tails of the distribution. For instance, the estimator defined by Qest(..., wtau = wtrunc, delta.left = 0.05, delta.right = 0.1) only uses quantiles in the interval (0.05, 0.90) to fit the model. In this example, delta.left = 0.05 is a guess for the proportion of outliers on the left tail; and delta.right is a guess for the proportion of outliers on the right tail. Use smooth = TRUE to replace the indicator functions involved in wtrunc with smooth functions. Introducing a weighting function that only assigns a positive weight to the quantiles that correspond to the "healthy" part of the distribution allows to deal with any level of contamination by outliers.

Value

A vector of weights assigned to each quantile.

Author(s)

Gianluca Sottile <gianluca.sottile@unipa.it>, Paolo Frumento <paolo.frumento@unipi.it>

See Also

Qest, Qlm, Qcoxph.

Examples

```
## Not run:
taus <- seq(0, 1, length.out = 1000)

### zero weight to quantiles above 0.95
plot(taus, wtrunc(taus, delta.left = 0, delta.right = 0.05),
  type = "l", lwd = 1.5)
# smooth counterpart
lines(taus, wtrunc(taus, delta.left = 0, delta.right = 0.05,
  smooth = TRUE, sigma = .01), col = 2, lwd = 1.5)
lines(taus, wtrunc(taus, delta.left = 0, delta.right = 0.05,
  smooth = TRUE, sigma = .05), col = 3, lwd = 1.5)

### zero weight to quantiles below 0.05
plot(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0),
  type = "l", lwd = 1.5)
# smooth counterpart
lines(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0,
  smooth = TRUE, sigma = .01), col = 2, lwd = 1.5)
lines(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0,
  smooth = TRUE, sigma = .05), col = 3, lwd = 1.5)


### zero weight to quantiles below 0.05 and above 0.90
plot(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0.10),
  type = "l", lwd = 1.5)
# smooth counterpart
lines(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0.10,
  smooth = TRUE, sigma = .01), col = 2, lwd = 1.5)
lines(taus, wtrunc(taus, delta.left = 0.05, delta.right = 0.10,
  smooth = TRUE, sigma = .05), col = 3, lwd = 1.5)


### Use wtrunc in Qest, Qlm, Qcoxph
# Qest(..., wtau = wtrunc, delta.left = 0.05, delta.right = 0.1)

## End(Not run)
```

Index