Package 'qrcm'

March 27, 2017

Type Package

Title Quantile Regression Coefficients Modeling

Version 2.1

Date 2017-03-26

Author Paolo Frumento <paolo.frumento@ki.se>

Maintainer Paolo Frumento <paolo.frumento@ki.se>

Description Parametric modeling of quantile regression coefficient functions.
Can be used with censored and truncated data.

Imports stats, utils, graphics

Depends survival, pch (>= 1.2)

License GPL-2

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-03-27 15:34:27 UTC

R topics documented:

qrcm-package *Quantile Regression Coefficients Modeling*

Description

This package implements Frumento and Bottai's (2016, 2017) method for quantile regression coefficient modeling (qrcm), in which quantile regression coefficients are described by (flexible) parametric functions of the order of the quantile. This permits modeling the entire conditional quantile function of a response variable.

Details

Package:	qrcm
Type:	Package
Version:	2.1
Date:	2017-03-26
License:	GPL-2

The function `iqr` permits specifying the regression model, allowing for censored and truncated outcomes. Two special functions, `slp` and `plf`, are provided to facilitate model building. The auxiliary functions `summary.iqr`, `predict.iqr`, and `plot.iqr` can be used to extract information from the fitted model, while `test.fit` is used to assess the model fit.

Author(s)

Paolo Frumento

Maintainer: Paolo Frumento <paolo.frumento@ki.se>

References

Frumento, P., and Bottai, M. (2016). *Parametric modeling of quantile regression coefficient functions*. Biometrics, 72 (1), pp 74-84, doi: 10.1111/biom.12410.

Frumento, P., and Bottai, M. (2017). *Parametric modeling of quantile regression coefficient functions with censored and truncated data*. Biometrics, doi: 10.1111/biom.12675.

Examples

```
# use simulated data

n <- 1000
x <- rexp(n)
y <- runif(n, 0, 1 + x)
model <- iqr(y ~ x, formula.p = ~ p + I(p^2))
summary(model)
```

```
summary(model, p = c(0.1,0.2,0.3))
predict(model, type = "beta", p = c(0.1,0.2,0.3))
predict(model, type = "CDF", newdata = data.frame(x = c(1,2,3), y = c(0.5,1,2)))
predict(model, type = "QF", p = c(0.1,0.2,0.3), newdata = data.frame(x = c(1,2,3)))
predict(model, type = "sim", newdata = data.frame(x = c(1,2,3)))
par(mfrow = c(1,2)); plot(model, ask = FALSE)
test.fit(model, R = 30)
```

`iqr` *Quantile Regression Coefficients Modeling*

Description

This function implements Frumento and Bottai's (2016) method for quantile regression coefficients modeling (qrcm). Quantile regression coefficients are described by (flexible) parametric functions of the order of the quantile. From version 2.0, this function can also be used with censored and truncated data, as described in Frumento and Bottai (2017).

Usage

```
iqr(formula, formula.p = ~ slp(p,3), weights, data, s, tol = 1e-6, maxit)
```

Arguments

`formula`	a two-sided formula of the form `y ~ x1 + x2 + ...`: a symbolic description of the quantile regression model. The left side of the formula should be `Surv(time,event)` if the data are right-censored, and `Surv(time,time2,event)` if the data are left-truncated (`time < time2`, `time` can be `-Inf`).
`formula.p`	a one-sided formula of the form `~ b1(p, ...) + b2(p, ...) + ...`, describing how quantile regression coefficients depend on `p`, the order of the quantile.
`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.
`data`	an optional data frame, list or environment containing the variables in `formula`.
`s`	an optional 0/1 matrix that permits excluding some model coefficients (see 'Examples').
`tol`	convergence criterion for numerical optimization.
`maxit`	maximum number of iterations.

Details

Quantile regression permits modeling conditional quantiles of a response variabile, given a set of covariates. A linear model is used to describe the conditional quantile function:

$$Q(p|x) = \beta_0(p) + \beta_1(p)x_1 + \beta_2(p)x_2 + \dots.$$

The model coefficients $\beta(p)$ describe the effect of covariates on the p-th quantile of the response variable. Usually, one or more quantiles are estimated, corresponding to different values of p.

Assume that each coefficient can be expressed as a parametric function of p of the form:

$$\beta(p|\theta) = \theta_0 + \theta_1 b_1(p) + \theta_2 b_2(p) + \dots$$

where $b_1(p), b_2(p, \dots)$ are known functions of p. If q is the dimension of $x = (1, x_1, x_2, \dots)$ and k is that of $b(p) = (1, b_1(p), b_2(p), \dots)$, the entire conditional quantile function is described by a $q \times k$ matrix θ of model parameters.

Users are required to specify two formulas: `formula` describes the regression model, while `formula.p` identifies the 'basis' $b(p)$. By default, `formula.p = ~ slp(p, k = 3)`, a 3rd-degree shifted Legendre polynomial (see `slp`). Any user-defined function $b(p, \dots)$ can be used, see 'Examples'.

Estimation of θ is carried out by minimizing an integrated loss function, corresponding to the integral, over p, of the loss function of standard quantile regression. This motivates the acronym `iqr` (integrated quantile regression).

From version 2.0 of the package, censored and truncated data are also allowed. Data are censored when, instead of a response variable T, one can only observe $Y = \min(T, C)$ and $d = I(T \leq C)$. Here, C is a censoring variable that is assumed to be conditionally independent of T. Truncation occurs if Y can only be observed when it exceeds another random variable Z. For example, in the prevalent sampling design, subjects with a disease are enrolled; those who died before enrollment are not observed. If the data are censored or truncated, θ is estimated by solving a system of estimating equation as described in Frumento and Bottai (2017).

Value

An object of class "`iqr`", a list containing the following items:

`coefficients`	a matrix of estimated model parameters describing the fitted quantile function.
`converged`	logical. The convergence status.
`n.it`	the number of iterations.
`call`	the matched call.
`obj.function`	the value of the minimized integrated loss function (`NULL` if the data are censored or truncated).
`mf`	the model frame used.
`PDF, CDF`	the fitted values of the conditional probability density function (`PDF`) and cumulative distribution function (`CDF`).
`covar`	the estimated covariance matrix.
`s`	the used '`s`' matrix.

Use `summary.iqr`, `plot.iqr`, and `predict.iqr` for summary information, plotting, and predictions from the fitted model. The function `test.fit` can be used for goodness-of-fit assessment. The generic accessor functions `coefficients`, `formula`, `terms`, `model.matrix`, `vcov` are available to extract information from the fitted model.

Note

By expressing quantile regression coefficients as functions of p, a parametric model for the conditional quantile function is specified. The induced PDF and CDF can be used as diagnostic tools. Negative values of PDF indicate quantile crossing, i.e., the conditional quantile function is not monotonically increasing. Null values of PDF indicate observations that lie outside the estimated support of the data, defined by quantiles of order 0 and 1. If null or negative PDF values occur for a relatively large proportion of data, the model is probably misspecified or ill-defined. If the model is correct, the fitted CDF should approximately follow a Uniform(0,1) distribution. This idea is used to implement a goodness-of-fit test, see `test.fit`.

The intercept can be excluded from `formula`, e.g., `iqr(y ~ -1 + x)`. This, however, implies that when `x = 0`, `y` is always 0. See example 5 in 'Examples'. The intercept can also be removed from `formula.p`. This is recommended if the data are bounded. For example, for strictly positive data, use `iqr(y ~ 1, formula.p = -1 + slp(p,3))` to force the smallest quantile to be zero. See example 6 in 'Examples'.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

References

Frumento, P., and Bottai, M. (2016). *Parametric modeling of quantile regression coefficient functions*. Biometrics, 72 (1), pp 74-84, doi: 10.1111/biom.12410.

Frumento, P., and Bottai, M. (2017). *Parametric modeling of quantile regression coefficient functions with censored and truncated data*. Biometrics, doi: 10.1111/biom.12675.

See Also

`summary.iqr`, `plot.iqr`, `predict.iqr`, for summary, plotting, and prediction, and `test.fit` for goodness-of-fit assessment. `plf` and `slp` to define $b(p)$ to be a piecewise linear function and a shifted Legendre polynomial basis, respectively.

Examples

```
##### Using simulated data in all examples


##### Example 1

n <- 1000
x <- runif(n)
y <- rnorm(n, 1 + x, 1 + x)
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
  # beta0(p) = beta1(p) = 1 + qnorm(p)

# fit the true model: b(p) = (1 , qnorm(p))
m1 <- iqr(y ~ x, formula.p = ~ I(qnorm(p)))
# the fitted quantile regression coefficient functions are
```

```
  # beta0(p) = m1$coef[1,1] + m1$coef[1,2]*qnorm(p)
  # beta1(p) = m1$coef[2,1] + m1$coef[2,2]*qnorm(p)

# a basis b(p) = (1, p), i.e., beta(p) is assumed to be a linear function of p
m2 <- iqr(y ~ x, formula.p = ~ p)

# a 'rich' basis b(p) = (1, p, p^2, log(p), log(1 - p))
m3 <- iqr(y ~ x, formula.p = ~ p + I(p^2) + I(log(p)) + I(log(1 - p)))

# 'slp' creates an orthogonal spline basis using shifted Legendre polynomials
m4 <- iqr(y ~ x, formula.p = ~ slp(p, k = 3)) # note that this is the default

# 'plf' creates the basis of a piecewise linear function
m5 <- iqr(y ~ x, formula.p = ~ plf(p, knots = c(0.1,0.9)))


summary(m1)
summary(m1, p = c(0.25,0.5,0.75))
test.fit(m1, R = 25)
par(mfrow = c(1,2)); plot(m1, ask = FALSE)
# see the documentation for 'summary.iqr', 'test.fit', and 'plot.iqr'




##### Example 2 ### excluding coefficients

n <- 1000
x <- runif(n)
qy <- function(p,x){(1 + qnorm(p)) + (1 + log(p))*x}
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
  # beta0(p) = 1 + qnorm(p)
  # beta1(p) = 1 + log(p)

y <- qy(runif(n), x) # to generate y, plug uniform p in qy(p,x)
iqr(y ~ x, formula.p = ~ I(qnorm(p)) + I(log(p)))

# I would like to exclude log(p) from beta0(p), and qnorm(p) from beta1(p)
# I set to 0 the corresponding entries of 's'

s <- matrix(1,2,3); s[1,3] <- s[2,2] <- 0
iqr(y ~ x, formula.p = ~ I(qnorm(p)) + I(log(p)), s = s)




##### Example 3 ### excluding coefficients when b(p) is singular

n <- 1000
x <- runif(n)
qy <- function(p,x){(1 + log(p) - 2*log(1 - p)) + (1 + log(p/(1 - p)))*x}
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
  # beta0(p) = 1 + log(p) - 2*log(1 - p)
```

```
  # beta1(p) = 1 + log(p/(1 - p))

y <- qy(runif(n), x) # to generate y, plug uniform p in qy(p,x)

iqr(y ~ x, formula.p = ~ I(log(p)) + I(log(1 - p)) + I(log(p/(1 - p))))
# log(p/(1 - p)) is dropped due to singularity

# I want beta0(p) to be a function of log(p) and log(1 - p),
# and beta1(p) to depend on log(p/(1 - p)) alone

s <- matrix(1,2,4); s[2,2:3] <- 0
iqr(y ~ x, formula.p = ~ I(log(p)) + I(log(1 - p)) + I(log(p/(1 - p))), s = s)
# log(p/(1 - p)) is not dropped




##### Example 4 ### using slp to test deviations from normality

n <- 1000
x <- runif(n)
y <- rnorm(n, 2 + x)
# the true model is normal, i.e., b(p) = (1, qnorm(p))

summary(iqr(y ~ x, formula.p = ~ I(qnorm(p)) + slp(p,3)))
# if slp(p,3) is not significant, no deviation from normality




##### Example 5 ### formula without intercept

n <- 1000
x <- runif(n)
y <- runif(n, 0,x)

# True quantile function: Q(p | x) = p*x, i.e., beta0(p) = 0, beta1(p) = p
# When x = 0, all quantiles of y are 0, i.e., the distribution is degenerated
# To explicitly model this, remove the intercept from 'formula'

iqr(y ~ -1 + x, formula.p = ~ p)

# the true model does not have intercept in b(p) either:

iqr(y ~ -1 + x, formula.p = ~ -1 + p)




##### Example 6 ### no covariates, strictly positive outcome

n <- 1000
y <- rgamma(n, 3,1)
```

```
# you know that Q(0) = 0
# remove intercept from 'formula.p', and use b(p) such that b(0) = 0

summary(iqr(y ~ 1, formula.p = ~ -1 + slp(p,5))) # shifted Legendre polynomials
summary(iqr(y ~ 1, formula.p = ~ -1 + sin(p*pi/2) + I(qbeta(p,2,4)))) # unusual basis
summary(iqr(y ~ 1, formula.p = ~ -1 + I(sqrt(p))*I(log(1 - p)))) # you can include interactions




##### Example 7 ### revisiting the classical linear model

n <- 1000
x <- runif(n)
y <- 2 + 3*x + rnorm(n,0,1) # beta0 = 2, beta1 = 3

iqr(y ~ x, formula.p = ~ I(qnorm(p)), s = matrix(c(1,1,1,0),2))
# first column of coefficients: (beta0, beta1)
# top-right coefficient: residual standard deviation




##### Example 8 ### censored data

n <- 1000
x <- runif(n,0,5)

u <- runif(n)
beta0 <- -log(1 - u)
beta1 <- 0.2*log(1 - u)
t <- beta0 + beta1*x  # time variable
c <- rexp(n,2)        # censoring variable
y <- pmin(t,c)        # observed events
d <- (t <= c)         # 1 = event, 0 = censored

iqr(Surv(y,d) ~ x, formula.p = ~ I(log(1 - p)))

##### Example 8 (cont.) ### censored and truncated data

z <- rexp(n,10)   # truncation variable
w <- which(y > z) # only observe z,y,d,x when y > z
z <- z[w]; y <- y[w]; d <- d[w]; x <- x[w]

iqr(Surv(z,y,d) ~ x, formula.p = ~ I(log(1 - p)))
```

---

plf *Basis of a Piecewise Linear Function*

Description

Generates $b_1(p), b_2(p), \ldots$ such that, for `0 < p < 1`,

$$\theta_1 * b_1(p) + \theta_2 * b_2(p) + \ldots$$

is a piecewise linear function with slopes $(\theta_1, \theta_2, \ldots)$.

Usage

```
plf(p, knots)
```

Arguments

`p`	a numeric vector of values between `0` and `1`.
`knots`	a set of *internal* knots between `0` and `1`. It can be `NULL` for no internal knots.

Details

This function permits computing a piecewise linear function on the unit interval. A different slope holds between each pair of knots, and the function is continuous at the knots.

Value

A matrix with one row for each element of `p`, and `length(knots) + 1` columns. The knots are returned as `attr(, "knots")`. Any linear combination of the basis matrix is a piecewise linear function where each coefficient represents the slope in the corresponding sub-interval (see 'Examples').

Note

This function is typically used within a call to `iqr`. A piecewise linear function can be used to describe how quantile regression coefficients depend on the order of the quantile.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

See Also

`slp`, for shifted Legendre polynomials.

Examples

```
p <- seq(0,1, 0.1)

a1 <- plf(p, knots = NULL) # returns p

a2 <- plf(p, knots = c(0.2,0.7))
plot(p, 3 + 1*a2[,1] - 1*a2[,2] + 2*a2[,3], type = "l")
  # intercept = 3; slopes = (1,-1,2)
```

plot.iqr *Plot Quantile Regression Coefficients*

Description

Plots quantile regression coefficients $\beta(p)$ as a function of p, based on a fitted model of class "iqr".

Usage

```
## S3 method for class 'iqr'
plot(x, conf.int = TRUE, polygon = TRUE, which = NULL, ask = TRUE, ...)
```

Arguments

x	an object of class "iqr", typically the result of a call to iqr.
conf.int	logical. If TRUE, asymptotic 95% confidence intervals are added to the plot.
polygon	logical. If TRUE, confidence intervals are represented by shaded areas via polygon. Otherwise, dashed lines are used.
which	an optional numerical vector indicating which coefficient(s) to plot. If which = NULL, all coefficients are plotted.
ask	logical. If which = NULL and ask = TRUE (the default), you will be asked interactively which coefficients to plot.
...	additional graphical parameters, that can include xlim, ylim, xlab, ylab, col, lwd, cex.lab, cex.axis, axes, frame.plot. See par.

Details

Using iqr, each quantile regression coefficient $\beta(p)$ is described by a linear combination of known parametric functions of p. With this command, a plot of $\beta(p)$ versus p is created. If ask = TRUE, an additional option permits plotting a Q-Q plot of the fitted cumulative distribution function (CDF), that should follow a U(0,1) distribution if the model is correctly specified. If the data are censored or truncated, this is assessed applying the Kaplan-Meier estimator to the fitted CDF values. See also test.fit for a formal test of uniformity.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

See Also

iqr for model fitting; summary.iqr and predict.iqr for model summary and prediction.

Examples

```
# using simulated data

n <- 1000
x <- runif(n)
qy <- function(p,x){p^2 + x*log(p)}
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
   # beta0(p) = p^2
   # beta1(p) = log(p)
y <- qy(runif(n), x) # to generate y, plug uniform p in qy(p,x)

par(mfrow = c(1,2))
plot(iqr(y ~ x, formula.p = ~ slp(p,3)), ask = FALSE)
# flexible fit with shifted Legendre polynomials
```

`predict.iqr` *Prediction After Quantile Regression Coefficients Modeling*

Description

Predictions from an object of class "`iqr`".

Usage

```
## S3 method for class 'iqr'
predict(object, type = c("beta", "CDF", "QF", "sim"), newdata, p, se = TRUE, ...)
```

Arguments

`object`	an object of class "`iqr`", the result of a call to `iqr`.
`type`	a character string specifying the type of prediction. See 'Details'.
`newdata`	an optional data frame in which to look for variables with which to predict. If omitted, the data are used. For `type = "CDF"`, it must include the response variable. Ignored if `type = "beta"`.
`p`	a numeric vector indicating the order(s) of the quantile to predict. Only used if `type = "beta"` or `type = "QF"`.
`se`	logical. If `TRUE` (the default), standard errors of the prediction will be computed. Only used if `type = "beta"` or `type = "QF"`.
`...`	for future methods.

Details

Using `iqr`, quantile regression coefficients $\beta(p)$ are modeled as parametric functions of p, the order of the quantile. This implies that the model parameter is *not* $\beta(p)$ itself. The function `predict.iqr` permits computing $\beta(p)$ and other quantities of interest, as detailed below.

- if `type = "beta"` (the default), $\beta(p)$ is returned at the supplied value(s) of `p`. If `p` is missing, a default `p = (0.01, ..., 0.99)` is used.
- if `type = "CDF"`, the value of the fitted CDF (cumulative distribution function) and PDF (probability density function) are computed.
- if `type = "QF"`, the fitted values $x'\beta(p)$, corresponding to the conditional quantile function, are computed at the supplied values of `p`.
- if `type = "sim"`, data are simulated from the fitted model. To simulate the data, the fitted conditional quantile function is computed at randomly generated `p` following a Uniform(0,1) distribution.

Value

- if `type = "beta"` a list with one item for each covariate in the model. Each element of the list is a data frame with columns (`p, beta, se, low, up`) reporting $\beta(p)$, its estimated standard error, and the corresponding 95% confidence interval. If `se = FALSE`, the last three columns are not computed.
- if `type = "CDF"`, a two-columns data frame (`CDF,PDF`).
- if `type = "QF"` and `se = FALSE`, a data frame with one row for each observation, and one column for each value of `p`. If `se = TRUE`, a list of two data frames, `fit` (predictions) and `se.fit` (standard errors).
- if `type = "sim"`, a vector of simulated data.

Note

Prediction may generate quantile crossing if the support of the new covariates values supplied in `newdata` is different from that of the observed data.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

See Also

`iqr`, for model fitting; `summary.iqr` and `plot.iqr`, for summarizing and plotting `iqr` objects.

Examples

```
# using simulated data

n <- 1000
x <- runif(n)
y <- rlogis(n, 1 + x, 1 + x)
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
  # beta0(p) = beta1(p) = 1 + log(p/(1 - p))

model <- iqr(y ~ x, formula.p = ~ I(log(p)) + I(log(1 - p)))
# (fit asymmetric logistic distribution)
```

```
# predict beta(0.25), beta(0.5), beta(0.75)
predict(model, type = "beta", p = c(0.25,0.5, 0.75))

# predict the CDF and the PDF at new values of x and y
predict(model, type = "CDF", newdata = data.frame(x = c(.1,.2,.3), y = c(1,2,3)))

# computes the quantile function at new x, for p = (0.25,0.5,0.75)
predict(model, type = "QF", p = c(0.25,0.5,0.75), newdata = data.frame(x = c(.1,.2,.3)))

# simulate data from the fitted model
ysim <- predict(model, type = "sim") # 'newdata' can be supplied

# if the model is correct, the distribution of y and that of ysim should be similar
qy <- quantile(y, prob = seq(.1,.9,.1))
qsim <- quantile(ysim, prob = seq(.1,.9,.1))
plot(qy, qsim); abline(0,1)
```

`slp` *Shifted Legendre Polynomials*

Description

Computes shifted Legendre polynomials.

Usage

```
slp(p, k = 3, intercept = FALSE)
```

Arguments

`p`	the variable for which to compute the polynomials. Must be `0 <= p <= 1`.
`k`	the degree of the polynomial.
`intercept`	logical. If `TRUE`, the polynomials include the constant term.

Details

Shifted Legendre polynomials (SLP) are orthogonal polynomial functions in (0,1) that can be used to build a spline basis, typically within a call to `iqr`. The constant term is omitted unless `intercept = TRUE`: for example, the first two SLP are `(2*p - 1, 6*p^2 - 6*p + 1)`, but `slp(p, k = 2)` will only return `(2*p, 6*p^2 - 6*p)`.

Value

An object of class "`slp`", i.e., a matrix with the same number of rows as `p`, and with `k` columns named `slp1, slp2, ...` containing the SLP of the corresponding orders. The value of `k` is reported as attribute.

Note

The estimation algorithm of `iqr` is optimized for objects of class "slp", which means that using `formula.p = ~ slp(p, k)` instead of `formula.p = ~ p + I(p^2) + ... + I(p^k)` will result in a quicker computation, even with `k = 1`, with equivalent results. The default for `iqr` is `formula.p = ~ slp(p, k = 3)`.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

References

Refaat El Attar (2009), *Legendre Polynomials and Functions*, CreateSpace, ISBN 978-1-4414-9012-4.

See Also

`plf`, for piecewise linear functions in the unit interval.

Examples

```
p <- seq(0,1,0.1)
slp(p, k = 1) # = 2*p
slp(p, k = 1, intercept = TRUE) # = 2*p - 1 (this is the true SLP of order 1)
slp(p, k = 3) # a linear combination of (p, p^2, p^3), with slp(0,k) = 0
```

---

summary.iqr *Summary After Quantile Regression Coefficients Modeling*

Description

Summary of an object of class "`iqr`".

Usage

```
## S3 method for class 'iqr'
summary(object, p, cov = FALSE, ...)
```

Arguments

`object`	an object of class "`iqr`", the result of a call to `iqr`.
`p`	an optional vector of quantiles.
`cov`	logical. If `TRUE`, the covariance matrix of $\beta(p)$ is reported. Ignored if `p` is missing.
`...`	for future methods.

Details

If p is missing, a summary of the fitted model is reported. This includes the estimated coefficients, their standard errors, and other summaries (see 'Value'). If p is supplied, the quantile regression coefficients of order p are extrapolated and summarized.

Value

If p is supplied, a standard summary of the estimated quantile regression coefficients is returned for each value of p. If `cov = TRUE`, the covariance matrix is also reported.

If p is missing (the default), a list with the following items:

`converged`	logical value indicating the convergence status.
`n.it`	the number of iterations.
`n`	the number of observations.
`free.par`	the number of free parameters in the model.
`coefficients`	the matrix of estimated coefficients. Each row corresponds to a covariate, while each column corresponds to an element of $b(p)$, the set of functions that describe how quantile regression coefficients vary with the order of the quantile. See 'Examples'.
`se`	the estimated standard errors.
`test.x`	Wald test for the covariates. Each *row* of `coefficients` is tested for nullity.
`test.p`	Wald test for the building blocks of the quantile function. Each *column* of `coefficients` is tested for nullity.
`obj.function`	the minimized loss function (`NULL` if the data are censored or truncated.
`call`	the matched call.

Note

In version 1.0 of the package, a chi-squared goodness-of-fit test was provided. The test appeared to be very unreliable and has now been removed. Use `test.fit`.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

See Also

`iqr`, for model fitting; `predict.iqr` and `plot.iqr`, for predicting and plotting objects of class "iqr". `test.fit` for a goodness-of-fit test.

Examples

```
# using simulated data

set.seed(1234); n <- 1000
x1 <- rexp(n)
```

```
x2 <- runif(n)
qy <- function(p,x){qnorm(p)*(1 + x)}
# true quantile function: Q(p | x) = beta0(p) + beta1(p)*x, with
   # beta0(p) = beta1(p) = qnorm(p)

y <- qy(runif(n), x1) # to generate y, plug uniform p in qy(p,x)
                      # note that x2 does not enter

model <- iqr(y ~ x1 + x2, formula.p = ~ I(qnorm(p)) + p + I(p^2))
# beta(p) is modeled by linear combinations of b(p) = (1, qnorm(p),p,p^2)

summary(model)
# interpretation:
  # beta0(p) = model$coef[1,]*b(p)
  # beta1(p) = model$coef[2,]*b(p); etc.
# x2 and (p, p^2) are not significant


summary(model, p = c(0.25, 0.75)) # summary of beta(p) at selected quantiles
```

`test.fit` *Goodness-of-Fit Test*

Description

Goodness-of-fit test for a model fitted with `iqr`. The Kolmogorov-Smirnov statistic and the Cramer-Von Mises statistic are computed. Their distribution under the null hypothesis is estimated with Monte Carlo.

Usage

```
test.fit(object, R = 100, zcmodel = 1, trace = FALSE)
```

Arguments

`object`	an object of class "`iqr`".
`R`	number of Monte Carlo replications.
`zcmodel`	a numeric value indicating how to model the joint distribution of censoring (C) and truncation (Z). Only used when data are censored and truncated. See 'Details'.
`trace`	logical. If `TRUE`, the progress be printed.

Details

This function permits assessing goodness of fit by testing the null hypothesis that the CDF values follow a $U(0,1)$ distribution, indicating that the model is correctly specified. Since the CDF values depend on estimated parameters, the distribution of the test statistic is not known. To evaluate it, the model is fitted on `R` simulated datasets generated under the null hypothesis.

If the data are censored and truncated, `object$CDF` is as well a censored and truncated outcome, and its quantiles must be estimated with Kaplan-Meier. The fitted survival curve is then compared with a $U(0, 1)$.

To run Monte Carlo simulations when data are censored or truncated, the distribution of the censoring and that of the truncation variable must be estimated: the function `pchreg` from the **pch** package is used, with its option `splinex = splinex()`.

The joint distribution of the censoring variable (C) and the truncation variable (Z) can be specified in two ways:

- If `zcmodel = 1` (the default), it is assumed that $C = Z + U$, where U is a positive variable and is independent of Z, given covariates. This is the most common situation, and is verified when censoring occurs at the end of the follow-up. Under this scenario, C and Z are correlated with $P(C > Z) = 1$.
- If `zcmodel = 2`, it is assumed that C and Z are conditionally independent. This situation is more plausible when all censoring is due to drop-out.

The testing procedure is described in details by Frumento and Bottai (2016, 2017).

Value

a matrix with columns `statistic` and `p.value`, reporting the Kolmogorov-Smirnov and Cramer-Von Mises statistic and the associated p-values evaluated with Monte Carlo.

Author(s)

Paolo Frumento <paolo.frumento@ki.se>

References

Frumento, P., and Bottai, M. (2016). *Parametric modeling of quantile regression coefficient functions*. Biometrics, 72 (1), pp 74-84, doi: 10.1111/biom.12410.

Frumento, P., and Bottai, M. (2017). *Parametric modeling of quantile regression coefficient functions with censored and truncated data*. Biometrics, doi: 10.1111/biom.12675.

Examples

```
y <- rnorm(1000)
m1 <- iqr(y ~ 1, formula.p = ~ I(qnorm(p))) # correct
m2 <- iqr(y ~ 1, formula.p = ~ p)  # misspecified
test.fit(m1, R = 25)
test.fit(m2, R = 25)
```

Index