Package ‘qrcm’

October 13, 2022

Type Package

Title Quantile Regression Coefficients Modeling

Version 3.0

Date 2021-01-29

Author Paolo Frumento <paolo.frumento@unipi.it>

Maintainer Paolo Frumento <paolo.frumento@unipi.it>

Description Parametric modeling of quantile regression coefficient functions.

Imports stats, utils, graphics

Depends survival (>= 2.4.1), pch (>= 2.0)

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-02-02 06:40:02 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. The package includes a generalization to longitudinal data (Frumento et al 2021). Special functions can be used to eliminate quantile crossing (Sottile and Frumento 2021).

Details

Package:	qrcm
Type:	Package
Version:	3.0
Date:	2021-01-29
License:	GPL-2

The function `iqr` permits specifying regression models for cross-sectional data, allowing for censored and truncated outcomes. The function `iqrL` can be used to analyze longitudinal data in which the same individuals are observed repeatedly.

Two special functions, `slp` and `plf`, can be used for model building. Auxiliary functions for model summary, prediction, and plotting are provided. The generic function `test.fit` is used to assess the model fit.

The function `diagnose.qc` can be applied to `iqr` objects to diagnose quantile crossing, and the option `remove.qc` can be used to remove it, using the algorithm described in `qc.control`.

Author(s)

Paolo Frumento

Maintainer: Paolo Frumento <paolo.frumento@unipi.it>

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.

Frumento, P., Bottai, M., and Fernandez-Val, I. (2021). *Parametric modeling of quantile regression coefficient functions with longitudinal data*. Journal of the American Statistical Association [forthcoming].

Sottile, G., and Frumento, P. (2021). *Parametric estimation of non-crossing quantile functions*. Statistical Modelling [forthcoming].

Examples

```
# iqr(y ~ x) # cross-sectional observations
# iqr(Surv(time, event) ~ x) # censored data
# iqr(Surv(start, stop, event) ~ x) # censored and truncated data
# iqrL(y ~ x, id = id) # repeated measures
```

---

diagnose.qc *Diagnose Quantile Crossing*

Description

Diagnose quantile crossing in a model estimated with iqr.

Usage

```
diagnose.qc(obj)
```

Arguments

obj	an object created with iqr.

Details

The function determines if quantile crossing occurs in your fitted model, and provides a number of diagnostic tools.

Local quantile crossing is defined by `obj$PDF < 0`, and is obtained when the quantile function, say $Q(p|x)$, has negative first derivatives at the values of p that correspond to the observed data. *Global* quantile crossing occurs when the conditional quantile function has negative first derivatives at *some* values of p. To assess global crossing, a grid of approximately 1000 quantiles is used. Note that local crossing is a special case of global crossing.

The function will assess local and global crossing, and return a summary `pcross` of the quantiles at which *global* crossing occurs. It is important to understand that crossing at extremely low or high quantiles is very common, but may be considered irrelevant in practice. For example, if *all* observations have crossing quantiles, implying that global crossing is 100%, but crossing only occurs at quantile above 0.999, the fitted model can be safely used for prediction. Very frequently, crossing occurs at extreme quantiles that do not correspond to any observation in the data.

This command will also compute a `crossIndex`, that represents the average length, across observations, of the sub-intervals $p*$ such that $Q'(p*|x) < 0$. For example, if $Q'(p|x) < 0$ in the interval $p* = (0.3, 0.5)$, the contribution to the `crossIndex` is `0.5 - 0.3 = 0.2`. If crossing is detected at a single quantile, the interval is assumed to have length `1e-6`. In principle, the `crossIndex` is always between `0` (no quantile crossing) and `1` (*all* observations crossing at *all* quantiles, which is clearly impossible). In practice, values of `crossIndex` greater than 0.05 are relatively rare.

Value

A list with the following items:

- `qc` — a data frame with two columns (`qc.local, qc.global`) containing logical indicators of local and global quantile crossing for each observation in the data.
- `qc.local, qc.global` — the absolute number of observations for which local/global quantile crossing was detected.
- `pcross` — a frequency table of the values of p at which global quantile crossing was detected.
- `crossIndex` — the estimated index of crossing described above.

If no quantile crossing is detected, `pcross = NULL`, and `crossIndex = 0`.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

References

Sottile, G., and Frumento, P. (2021). *Parametric estimation of non-crossing quantile functions*. Statistical Modelling [forthcoming].

See Also

`iqr`, `qc.control`.

Examples

```
# Using simulated data

n <- 1000
x1 <- runif(n,0,3)
x2 <- rbinom(n,1,0.5)

u <- runif(n)
y <- 1*qexp(u) + (2 + 3*u)*x1 + 5*x2
m <- iqr(y ~ x1 + x2, formula.p = ~ slp(p,7))
diagnose.qc(m)
```

`iqr` *Quantile Regression Coefficients Modeling*

Description

This function implements Frumento and Bottai's (2016, 2017) methods for quantile regression coefficients modeling (qrcm). Quantile regression coefficients are described by (flexible) parametric functions of the order of the quantile. Quantile crossing can be eliminated using the method described in Sottile and Frumento (2021).

Usage

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

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 is `Surv(time,event)` if the data are right-censored, and `Surv(time,time2,event)` if the data are right-censored and 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.
`remove.qc`	either a logical value, or a list created with `qc.control`. See 'Details'.

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:

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

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:

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

where $b_1(p), b_2(p, \ldots)$ are known functions of p. If q is the dimension of $x = (1, x_1, x_2, \ldots)$ and k is that of $b(p) = (1, b_1(p), b_2(p), \ldots)$, 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 $b(p, \ldots)$ can be used, see 'Examples'.

If no censoring and truncation are present, estimation of θ is carried out by minimizing an objective function that corresponds to the integral, with respect to p, of the loss function of standard quantile regression. Details are in Frumento and Bottai (2016). If the data are censored or truncated, instead, θ is estimated by solving the estimating equations described in Frumento and Bottai (2017).

The option `remove.qc` applies the method described by Sottile and Frumento (2021) to remove quantile crossing. You can either choose `remove.qc = TRUE`, or use `remove.qc = qc.control(...)`, which allows to specify the operational parameters of the algorithm. Please read `qc.control` for more details on the method, and use `diagnose.qc` to diagnose quantile crossing.

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`	if the data are neither censored nor truncated, the value of the minimized loss function; otherwise, a meaningful loss function which, however, is not the objective function of the model (see note 3). The number of model parameter is returned as an attribute.
`mf`	the model frame used.
`PDF, CDF`	the fitted values of the conditional probability density function (PDF) and cumulative distribution function (CDF). See note 1 for details.
`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 accessory functions `coefficients`, `formula`, `terms`, `model.matrix`, `vcov` are available to extract information from the fitted model. The special function `diagnose.qc` can be used to diagnose quantile crossing.

Note

NOTE 1 (`PDF`, `CDF`, quantile crossing, and goodness-of-fit). By expressing quantile regression coefficients as functions of p, you practically specify a parametric model for the entire conditional distribution. The induced `CDF` is the value $p*$ such that $y = Q(p * |x)$. The corresponding `PDF` is given by $1/Q'(p * |x)$. Negative values of `PDF` indicate quantile crossing, occurring when the estimated quantile function is not monotonically increasing. If 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`.

NOTE 2 (model intercept). The intercept can be excluded from `formula`, e.g., `iqr(y ~ -1 + x)`. This, however, implies that when `x = 0`, `y` is zero at all quantiles. 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'.

NOTE 3 (censoring, truncation, and loss function). Data are right-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. Additionally, left 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.

Ordinary quantile regression minimizes $L(\beta(p)) = \sum(p - \omega)(t - x'\beta(p))$ where $\omega = I(t \leq x'\beta(p))$. Equivalently, it solves its first derivative, $S(\beta(p)) = \sum x(\omega - p)$. The objective function of `iqr` is simply the integral of $L(\beta(p|\theta))$ with respect to p.

If the data are censored and truncated, ω is replaced by

$$\omega* = \omega.y + (1 - d)\omega.y(p - 1)/S.y - \omega.z - \omega.z(p - 1)/S.z + p$$

where $\omega.y = I(y \leq x'\beta(p))$, $\omega.z = I(z \leq x'\beta(p))$, $S.y = P(T > y)$, and $S.z = P(T > z)$. The above formula can be obtained from equation (7) of Frumento and Bottai, 2017. Replacing ω with $\omega*$ in $L(\beta(p))$ is **NOT** equivalent to replacing ω with $\omega*$ in $S(\beta(p))$.

The latter option leads to a much simpler computation, and generates the estimating equation used by `iqr`. This means that, if the data are censored or truncated, the `obj.function` returned by `iqr` is **NOT** the objective function being minimized, and should not be used to compare models. However, if one of two models has a much larger value of the `obj.function`, this may be a sign of severe misspecification or poor convergence.

Author(s)

Paolo Frumento `<paolo.frumento@unipi.it>`

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.

Sottile, G., and Frumento, P. (2021). *Parametric estimation of non-crossing quantile functions*. Statistical Modelling [forthcoming].

See Also

`summary.iqr`, `plot.iqr`, `predict.iqr`, for summary, plotting, and prediction, and `test.fit.iqr` 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; `diagnose.qc` to diagnose quantile crossing.

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)
par(mfrow = c(1,2)); plot(m1, ask = FALSE)
# see the documentation for 'summary.iqr', 'test.fit.iqr', 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)))
```

iqrL *Quantile Regression Coefficients Modeling with Longitudinal Data*

Description

This function implements Frumento et al's (2021) method for quantile regression coefficients modeling with longitudinal data.

Usage

```
iqrL(fx, fu = ~ slp(u,3), fz = ~ 1, fv = ~ -1 + I(qnorm(v)),
   id, weights, s.theta, s.phi, data, tol = 1e-5, maxit)
```

Arguments

`fx, fu, fz, fv`	formulas that describe the model (see 'Details').
`id`	a vector of cluster identifiers.
`weights`	an optional vector of weights to be used in the fitting process.
`s.theta, s.phi`	optional 0/1 matrices that permit excluding some model coefficients.
`data`	an optional data frame, list or environment containing the variables in `fx` and `fz`.
`tol`	convergence criterion for numerical optimization.
`maxit`	maximum number of iterations. If missing, a default is computed.

Details

New users are recommended to read Frumento and Bottai's (2018) paper for details on notation and modeling, and to have some familiarity with the `iqr` command, of which `iqrL` is a natural expansion.

The following data-generating process is assumed:

$$Y_{it} = x_{it}\beta(U_{it}) + z_i\gamma(V_i)$$

where x_{it} are level-1 covariates, z_i are level-2 covariates, and (U_{it}, V_i) are independent $U(0,1)$ random variables. This model implies that $\alpha_i = z_i\gamma(V_i)$ are cluster-level effects with quantile function $z_i\gamma(v)$, while $x_{it}\beta(u)$ is the quantile function of $Y_{it} - \alpha_i$.

Both $\beta(u)$ and $\gamma(v)$ are modeled parametrically, using a linear combination of known "basis" functions $b(u)$ and $c(v)$ such that

$$\beta(u) = \beta(u|\theta) = \theta b(u),$$

$$\gamma(u) = \gamma(u|\phi) = \phi c(v),$$

where θ and ϕ are matrices of model parameters.

Model specification is implemented as follows.

- `fx` is a two-sided formula of the form `y ~ x`.
- `fu` is a one-sided formula that describes $b(u)$.
- `fz` is a one-sided formula of the form `~ z`.
- `fv` is a one-sided formula that describes $c(v)$.

By default, `fu = ~ slp(u,3)`, a shifted Legendre's polynomial (see `slp`), and the distribution of α_i is assumed to be Normal (`fv = ~ -1 + I(qnorm(v))`) and to not depend on covariates (`fz = ~ 1`).

Restrictions on θ and ϕ are imposed by setting to zero the corresponding elements of `s.theta` and `s.phi`.

Value

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

- `theta, phi` — estimates of θ and ϕ.
- `obj.function` — the value of the minimized loss function, and, separately, the level-1 and the level-2 loss. The number of model parameters (excluding the individual effects) is returned as an attribute.
- `call` — the matched call.
- `converged` — logical. The convergence status.
- `n.it` — the number of iterations.
- `covar.theta, covar.phi` — the estimated covariance matrices.
- `mf.theta, mf.phi` — the model frames used to fit θ and ϕ, respectively. Note that `mf.theta` is sorted by increasing `id` and, within each `id`, by increasing values of the response variable `y`, while `mf.phi` is sorted by increasing `id`.
- `s.theta, s.phi` — the used '`s.theta`' and '`s.phi`' matrices.
- `fit` — a data.frame with the following variables:
    - `id` the cluster identifier.
    - `y` the response variable.
    - `alpha` the estimated individual effects.
    - `y_alpha` = `y - alpha[id]`, the estimated responses purged of the individual effects.
    - `v` estimates of V_i.
    - `u` estimates of U_{it}.

  Observations are sorted by increasing `id` and, within each `id`, by increasing `y`.

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

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

References

Frumento, P., Bottai, M., and Fernandez-Val, I. (2021). *Parametric modeling of quantile regression coefficient functions with longitudinal data*. Journal of the American Statistical Association [forthcoming].

See Also

summary.iqrL, plot.iqrL, predict.iqrL, for summary, plotting, and prediction, and test.fit.iqrL for goodness-of-fit assessment. plf and slp to define $b(u)$ or $c(v)$ to be piecewise linear functions and shifted Legendre polynomials, respectively.

Examples

```
##### Also see ?iqr for a tutorial on modeling
##### Using simulated data in all examples


##### Example 1

n <- 1000 # n. of observations
n.id <- 100 # n. of clusters
id <- rep(1:n.id, each = n/n.id) # cluster id

x1 <- runif(n) # a level-1 covariate
z1 <- rbinom(n,1,0.5)[id] # a level-2 covariate

V <- runif(n.id) # V_i
U <- runif(n) # U_it

alpha <- (0.5 + z1)*qnorm(V) # or alpha = rnorm(n.id, 0, 0.5 + z1)
y_alpha <- qexp(U) + 3*x1 # or y_alpha = 3*x1 + rexp(n)
y <- y_alpha + alpha[id] # observed outcome
mydata <- data.frame(id = id, y = y, x1 = x1, z1 = z1[id])

# true quantile function: beta0(u) + beta1(u)*x1 + gamma0(v) + gamma1(v)*z1
# beta0(u) = qexp(u)
# beta1(u) = 3
# gamma0(v) = 0.5*qnorm(v)
# gamma1(v) = qnorm(v)


##### Example 1 (cont.) fitting the model

model1 <- iqrL(fx = y ~ x1, fu = ~ I(qexp(u)), fz = ~ z1, fv = ~ -1 + I(qnorm(v)),
  id = id, data = mydata)
summary(model1) # theta, phi
summary(model1, level = 1, p = c(0.1,0.9)) # beta
summary(model1, level = 2, p = c(0.1,0.9)) # gamma
par(mfrow = c(2,2)); plot(model1, ask = FALSE)


##### Example 1 (cont.) - excluding coefficients

s.theta <- rbind(0:1,1:0) # beta0(u) has no intercept, and beta1(u) does not depend on u.
model2 <- iqrL(fx = y ~ x1, fu = ~ I(qexp(u)), fz = ~ z1, fv = ~ -1 + I(qnorm(v)),
  id = id, s.theta = s.theta, data = mydata)
summary(model2)
```

```
test.fit(model2) # testing goodness-of-fit


##### Example 1 (cont.) - flexible modeling using slp for lev. 1, asymm. logistic for lev. 2

model3 <- iqrL(fx = y ~ x1, fu = ~ slp(u,3),
  fz = ~ z1, fv = ~ -1 + I(log(2*v)) + I(-log(2*(1 - v))),
  id = id, data = mydata)
par(mfrow = c(2,2)); plot(model3, ask = FALSE)


##### Example 2 - revisiting the classical linear random-effects model

n <- 1000 # n. of observations
n.id <- 100 # n. of clusters
id <- rep(1:n.id, each = n/n.id) # id

x1 <- runif(n,0,5)
E <- rnorm(n) # level-1 error
W <- rnorm(n.id, 0, 0.5) # level-2 error
y <- 2 + 3*x1 + E + W[id] # linear random-intercept model

s.theta <- rbind(1, 1:0)
linmod <- iqrL(fx = y ~ x1, fu = ~ I(qnorm(u)), id = id, s.theta = s.theta)
summary(linmod)
```

---

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@unipi.it>

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@unipi.it`>

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
```

`plot.iqrL` *Plot Quantile Regression Coefficients with Longitudinal Data*

Description

Plots quantile regression coefficients $\beta(u)$ and $\gamma(v)$, based on a fitted model of class "`iqrL`".

Usage

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

Arguments

`x`	an object of class "`iqrL`", the result of a call to `iqrL`.
`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 options will permit creating Q-Q plots of `u` or `v`, which should be independently distributed according to a Uniform(0,1) distribution. The option `ppplot(u,v)` will generate a P-P plot that compares the empirical distribution of `(u,v)` with its theoretical value, `F(u,v)` = `uv`, at a discrete grid of points.
`...`	additional graphical parameters, that can include `xlim, ylim, xlab, ylab, col, lwd, cex.lab, cex.axis, axes, frame.plot`. See `par`.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

See Also

`iqrL` for model fitting; `summary.iqrL` and `predict.iqrL` for model summary and prediction.

Examples

```
# using simulated data

n <- 1000 # n. of observations
n.id <- 100 # n. of clusters
id <- rep(1:n.id, each = n/n.id) # cluster id

x1 <- runif(n) # a level-1 covariate
z1 <- rnorm(n.id) # a level-2 covariate

V <- runif(n.id) # V_i
U <- runif(n) # U_it

alpha <- 2*(V - 1) + z1 # alpha
y_alpha <- 1 + 2*qnorm(U) + 3*U*x1 # y - alpha
y <- y_alpha + alpha[id] # observed outcome
mydata <- data.frame(id = id, y = y, x1 = x1, z1 = z1[id])
```

```
model <- iqrL(fx = y ~ x1, fu = ~ I(qnorm(u)) + u,
  fz = ~ z1, fv = ~ -1 + I(qnorm(v)), id = id, data = mydata)
par(mfrow = c(2,2))
plot(model, ask = FALSE)
```

---

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@unipi.it>

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)
```

---

predict.iqrL *Prediction After Quantile Regression Coefficients Modeling with Longitudinal Data*

Description

Predictions from an object of class "iqrL".

Usage

```
## S3 method for class 'iqrL'
predict(object, level, type = c("coef", "CDF", "QF", "sim"), newdata, p, se = FALSE, ...)
```

Arguments

object	an object of class "iqrL", the result of a call to iqrL.
level	a numeric scalar. Use level = 1 to predict $y_{it} - \alpha_i$, and level = 2 to predict α_i (see iqrL for the notation).
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 (ignored if type = "coef"). For type = "CDF", newdata must include a response variable named 'y_alpha', if level = 1, and 'alpha' if level = 2. If newdata is omitted, the observed data will be used, and y_alpha and alpha will be taken from object$fit.
p	a numeric vector indicating the order(s) of the quantile to predict. Only used if type = "coef" or type = "QF".
se	logical. If TRUE (the default), standard errors of the prediction will be computed. Only used if type = "coef" or type = "QF".
...	for future methods.

Details

- if type = "coef" (the default), quantile regression coefficients are returned: if level = 1, $\beta(p)$; and if level = 2, $\gamma(p)$. 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 level = 1, these refer to the distribution of $Y_{it} - \alpha_i = x_{it}\beta(U_{it})$, and the CDF is an estimate of U_{it}. If level = 2, they refer to the distribution of $\alpha_i = z_i\gamma(V_i)$, and the CDF is an estimate of V_i.
- if type = "QF", the fitted values $x\beta(p)$ (if level = 1), or $z\gamma(p)$ (if level = 2).
- if type = "sim", data are simulated from the fitted model. If level = 1, simulated values are from the distribution of $Y_{it} - \alpha_i$, while if level = 2, they are from the distribution of α_i.

Value

- if type = "coef" a list with one item for each covariate. Each element of the list is a data frame with columns (u, beta, se, low, up), if level = 1, and (v, gamma, se, low, up), if level = 2. 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

If no newdata are supplied, the observed data are used and predictions are ordered as follows:

- if level = 1, by increasing id and, within each id, by increasing values of the response variable y. Rownames will indicate the position in the original data frame.
- if level = 2, by increasing id.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

See Also

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

Examples

```
# using simulated data

n <- 1000 # n. of observations
n.id <- 100 # n. of clusters
id <- rep(1:n.id, each = n/n.id) # cluster id

x1 <- runif(n) # a level-1 covariate
z1 <- rbinom(n.id,1,0.5) # a level-2 covariate

V <- runif(n.id) # V_i
U <- runif(n) # U_it

alpha <- qlogis(V)*(0.5 + z1) # alpha
y_alpha <- 1 + 2*qexp(U) + 3*x1 # y - alpha
y <- y_alpha + alpha[id] # observed outcome
mydata <- data.frame(id = id, y = y, x1 = x1, z1 = z1[id])

# true model: Y_it = beta0(U_it) + beta1(U_it)*x1 + gamma0(V_i) + gamma1(V_i)*z1
# beta0(u) = 1 + 2*pexp(u)
# beta1(u) = 3
# gamma0(v) = 0.5*qlogis(v)
```

```
# gamma1(v) = qlogis(V)

model <- iqrL(fx = y ~ x1, fu = ~ I(qexp(u)), fz = ~ z1, fv = ~ -1 + I(qlogis(v)),
  id = id, data = mydata)

# predict beta(0.25), beta(0.5), beta(0.75)
predict(model, level = 1, type = "coef", p = c(0.25,0.5,0.75))

# predict gamma(0.1), gamma(0.9)
predict(model, level = 2, type = "coef", p = c(0.1,0.9))

# predict the CDF (u) and the PDF of (y - alpha), at new values of x1
predict(model, level = 1, type = "CDF",
  newdata = data.frame(x1 = c(.1,.2,.3), y_alpha = c(1,2,3)))

# predict the CDF (v) and the PDF of alpha, at new values of z1
predict(model, level = 2, type = "CDF",
  newdata = data.frame(z1 = c(0,1), alpha = c(-1,1)))

# computes the quantile function of (y - alpha) at new x1, for u = (0.25,0.5,0.75)
predict(model, level = 1, type = "QF", p = c(0.25,0.5,0.75),
  newdata = data.frame(x1 = c(.1,.2,.3)))

# computes the quantile function of alpha at new z1, for v = (0.25,0.5,0.75)
predict(model, level = 2, type = "QF", p = c(0.25,0.5,0.75),
  newdata = data.frame(z1 = c(.1,.2,.3)))


# simulate data from the fitted model
y_alpha_sim <- predict(model, level = 1, type = "sim")
alpha_sim <- predict(model, level = 2, type = "sim")
y_sim = y_alpha_sim + alpha_sim[id]
```

`qc.control` *Estimate Non-Crossing Quantile Functions*

Description

This function generates a list of arguments to be used as operational parameters for `remove.qc` within a call to `iqr`. Additionally, this R documentation page contains a short description of the algorithm, which is presented in details in Sottile and Frumento (2021).

Usage

```
qc.control(maxTry = 25, trace = FALSE, lambda = NULL)
```

Arguments

`maxTry`	maximum number of attempts of the algorithm.
`trace`	logical: should the progress be printed on screen?

`lambda`	an optional positive scalar to be used as tuning parameter (see "Details"). By default, `lambda = NULL`.

Details

Quantile crossing occurs when the first derivative of the estimated quantile function is negative at some value of p. The argument `remove.qc` of the `iqr` function can be used to eliminate quantile crossing.

The algorithm proceeds as follows. A penalization that reflects the severity of crossing is added to the loss function. The weight of the penalty term is determined by a tuning parameter λ. If λ is too small, the penalization has no effect. However, if λ is too large, the objective function may lose its convexity, causing a malfunctioning of the algorithm. In general, the value of λ is *not* user-defined. The algorithm starts with an initial guess for the tuning parameter, and proceeds adaptively until it finds a suitable value. The maximum number of iterations is determined by the `maxTry` argument of this function (default `maxTry = 25`). The algorithm stops automatically when the `crossIndex` of the model (see `diagnose.qc`) is zero, or when no further progress is possible.

It is possible to supply a user-defined value of λ, e.g., `lambda = 7.5`. If this happens, the model is estimated **once**, using the requested `lambda`, while the `maxTry` argument is ignored.

This method allows for censored or truncated data, that are supported by `iqr`. Full details are provided in Sottile and Frumento (2021).

Value

The function performs a sanity check and returns its arguments.

Note

Occasionally, the loss of the penalized model is smaller than that of the unconstrained fit. This is either an artifact due to numerical approximations or lack of convergence, or is explained by the fact that, if the quantile function is ill-defined, so is the loss function of the model. With censored or truncated data, however, it can also be explained by the fact that the `obj.function` of the model is **NOT** the function being minimized (see note 3 in the documentation of `iqr`).

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

References

Sottile, G., and Frumento, P. (2021). *Parametric estimation of non-crossing quantile functions*. Statistical Modelling [forthcoming].

See Also

`iqr`, `diagnose.qc`.

Examples

```
# Using simulated data

set.seed(1111)
n <- 1000
x1 <- runif(n,0,3)
x2 <- rbinom(n,1,0.5)

u <- runif(n)
y <- 1*qexp(u) + (2 + 3*u)*x1 + 5*x2

# This model is likely to suffer from quantile crossing
m <- iqr(y ~ x1 + x2, formula.p = ~ slp(p,7))
diagnose.qc(m)

# Repeat estimation with remove.qc = TRUE
m2 <- iqr(y ~ x1 + x2, formula.p = ~ slp(p,7), remove.qc = TRUE)
diagnose.qc(m2)

# Use remove.qc = qc.control(trace = TRUE) to see what is going on!
# You can set a larger 'maxTry', if the algorithm failed to remove
# quantile crossing entirely, or a smaller one, if you want to stop
# the procedure before it becomes 'too expensive' in terms of loss.
```

`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@unipi.it>

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 unreliable and has been removed from the subsequent versions. Use `test.fit`.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

See Also

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

`summary.iqrL` *Summary After Quantile Regression Coefficients Modeling With Longitudinal Data*

Description

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

Usage

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

Arguments

`object`	an object of class "`iqrL`", the result of a call to `iqrL`.
`p`	an optional vector of quantiles.
`level`	a numeric scalar. Use `level = 1` to summarize $\beta(u)$, and `level = 2` to summarize $\gamma(v)$. Ignored if `p` is missing.
`cov`	logical. If `TRUE`, the covariance matrix of the coefficients or is reported. Ignored if `p` is missing.
`...`	for future methods.

Value

If p is supplied, a standard summary of the estimated quantile regression coefficients is returned for each value of p: if `level = 1`, a summary of `beta(p)`, and if `level = 2`, a summary of `gamma(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.
- `n.id` the number of unique `id`s.
- `free.par` the number of free parameters in the model, excluding fixed effects.
- `theta` the estimate of θ.
- `se.theta` the estimated standard errors associated with `theta`.
- `phi` the estimate of ϕ.
- `se.phi` the estimated standard errors associated with `phi`.
- `test.row.theta`, `test.row.phi` Wald test for the covariates. Each *row* of `theta` and `phi` is tested for nullity.
- `test.col.theta`, `test.col.phi` Wald test for the building blocks of the quantile function. Each *column* of `theta` and `phi` is tested for nullity.
- `obj.function` the minimized loss function.
- `call` the matched call.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

See Also

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

Examples

```
# using simulated data

n <- 1000 # n. of observations
n.id <- 100 # n. of clusters
id <- rep(1:n.id, each = n/n.id) # cluster id

x <- rexp(n) # a covariate

V <- runif(n.id) # V_i
U <- runif(n) # U_it
y <- 1 + 2*log(U) + 3*x + 0.5*qnorm(V)
```

```
# true quantile function: Q(u,v | x) = beta0(u) + beta1(u)*x + gamma0(v), with
   # beta0(u) = 1 + 2*log(u)
   # beta1(u) = 3
   # gamma0(v) = 0.5*qnorm(v)

model <- iqrL(fx = y ~ x, fu = ~ 1 + I(log(u)), fz = ~ 1, fv = ~ -1 + I(qnorm(v)), id = id)
summary(model)
summary(model, level = 1, p = c(0.25, 0.75)) # summary of beta(u) at selected quantiles
summary(model, level = 2, p = c(0.1, 0.9)) # summary of gamma(v) at selected quantiles
```

---

test.fit *Goodness-of-Fit Test*

Description

Generic method for goodness-of-fit test.

Usage

```
test.fit(object, ...)
```

Arguments

`object`	an object of class "`iqr`" or "`iqrL`".
`...`	additional arguments to be supplied to `test.fit.iqr` or `test.fit.iqrL`.

Details

This function will simply call `test.fit.iqr` or `test.fit.iqrL` depending on `class(object)`.

Value

The test statistic(s) and the associated p-values evaluated with Monte Carlo.

See Also

`test.fit.iqr`, `test.fit.iqrL`

test.fit.iqr *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 evaluated with Monte Carlo.

Usage

```
## S3 method for class 'iqr'
test.fit(object, R = 100, zcmodel = 1, trace = FALSE, ...)
```

Arguments

`object`	an object of class "`iqr`".
`R`	number of Monte Carlo replications. If `R = 0`, the function only returns the test statistics.
`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 will be printed.
`...`	for future arguments.

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 default settings.

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@unipi.it>

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)
test.fit(m2)
```

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

Description

Goodness-of-fit test for a model fitted with `iqrL`. The Kolmogorov-Smirnov statistic is computed and its distribution under the null hypothesis is evaluated with Monte Carlo.

Usage

```
## S3 method for class 'iqrL'
test.fit(object, R = 100, trace = FALSE, ...)
```

Arguments

`object`	an object of class "`iqrL`".
`R`	number of Monte Carlo replications. If `R = 0`, the function only returns the test statistic.
`trace`	logical. If `TRUE`, the progress will be printed.
`...`	for future arguments.

Details

This function permits assessing goodness of fit by testing the null hypothesis that the estimated `(u,v)` values are independent uniform variables. To evaluate the distribution of the test statistic under the true model, a Monte Carlo method is used (Frumento et al, 2021).

Value

a vector with entries `statistic` and `p.value`, reporting the Kolmogorov-Smirnov statistic (evaluated on a grid) and the associated p-value.

Author(s)

Paolo Frumento <paolo.frumento@unipi.it>

References

Frumento, P., Bottai, M., and Fernandez-Val, I. (2021). *Parametric modeling of quantile regression coefficient functions with longitudinal data*. Journal of the American Statistical Association [forthcoming].

Examples

```
id <- rep(1:50, each = 10)
y <- rnorm(500) + rnorm(50)[id]
m1 <- iqrL(fx = y ~ 1, fu = ~ I(qnorm(u)), id = id) # correct
m2 <- iqrL(fx = y ~ 1, fu = ~ u, id = id)  # misspecified

test.fit(m1, R = 20)
test.fit(m2, R = 20)

# Warning: this procedure may be time-consuming.
```

Index