Package 'qrcmNP'

October 30, 2026

Type Package

Title Nonlinear and Penalized Quantile Regression Coefficients Modeling

Version 1.0.0

Description Penalized and nonlinear parametric modeling of quantile regression coefficient functions. The penalized integrated loss minimization estimator (lasso and group lasso penalties) of Sottile, Frumento, Chiodi and Bottai (2020) <doi:10.1177/1471082X19825523> performs covariate selection using the information on all quantiles simultaneously. Nonlinear parametric quantile models, in which the conditional quantile function can be a nonlinear function of the parameters, of the covariates or both, are estimated by integrated loss minimization as in Bottai and Cilluffo (2020) <doi:10.1177/0962280220941159>. Both approaches extend the quantile regression coefficients modeling framework of Frumento and Bottai (2016) <doi:10.1111/biom.12410> implemented in 'qrcm'.

License GPL-2

URL https://github.com/gianluca-sottile/qrcmNP

BugReports https://github.com/gianluca-sottile/qrcmNP/issues

Depends R (>= 4.1.0)

Imports graphics, grDevices, qrcm (>= 3.0), stats, survival (>= 2.41), utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Gianluca Sottile [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9347-7251>),
Paolo Frumento [aut, cph] (author of parts of the code adapted from
'qrcm')

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

Repository CRAN

Date/Publication 2026-10-30 09:50:07 UTC

Contents

coef.piqr 2
cv_piqr 3
gof.niqr 4
gof.piqr 5
niqr 7
niqr.control 10
piqr 11
plot.niqr 14
plot.piqr 15
predict.niqr 16
predict.piqr 17
refit_piqr 19
summary.niqr 20
summary.piqr 21
test.fit.niqr 22

Index **23**

`coef.piqr` *Extract Coefficients of a Penalized QRCM*

Description

Extract Coefficients of a Penalized QRCM

Usage

```
## S3 method for class 'piqr'
coef(object, pos.lambda, lambda, ...)
```

Arguments

`object`	an object of class "`piqr`".
`pos.lambda`	the position of the selected value of λ.
`lambda`	alternatively, a value of λ (the closest value in `object$lambda` is used).
`...`	not used.

Value

The $q \times k$ matrix $\hat{\theta}$ at the selected λ.

See Also

`piqr()`

`cv_piqr` *Cross-Validation for Penalized QRCM*

Description

K-fold cross-validation of the integrated loss along the path of a `piqr()` fit (uncensored data only).

Usage

```
cv_piqr(
  formula,
  formula.p = ~slp(p, 3),
  weights,
  data,
  s,
  penalty = c("lasso", "group"),
  penalty.factor,
  nlambda = 100,
  lambda.min.ratio,
  lambda,
  nfolds = 10,
  foldid,
  ...
)
```

Arguments

`formula`, `formula.p`, `weights`, `data`, `s`, `penalty`, `penalty.factor`, `nlambda`, `lambda.min.ratio`, `lambda`	as in `piqr()`.
`nfolds`	number of folds (default 10).
`foldid`	an optional vector of fold indicators (integers in `1:nfolds`), one for each observation used in the fit (i.e., each row of `data` without missing values in the model variables).
`...`	further arguments passed to `piqr()` (e.g., `tol`, `maxit`).

Details

The cross-validated loss of observation i is $\int_0^1 \rho_p\{y_i - x_i^\top \hat\theta_{-i}\, b(p)\}\, dp$, computed numerically, where $\hat\theta_{-i}$ is estimated without the fold containing i. In each fold the variables are standardized as in the full data and λ is rescaled by the size of the training sample, so that the same value of λ corresponds to the same penalty in all fits. `lambda.1se` is the largest λ whose error is within one standard error of the minimum.

Value

An object of class `"cv.piqr"`, a list with items `lambda`, `cvm` (mean cross-validated loss), `cvsd` (its standard error), `lambda.min`, `lambda.1se`, `pos.min`, `pos.1se` (positions in `lambda`), `foldid` and `fit` (the `piqr()` fit on the full data).

See Also

`piqr()`, `gof.piqr()`

Examples

```
set.seed(1)
n <- 300
dat <- data.frame(x1 = runif(n), x2 = runif(n), x3 = runif(n))
dat$y <- 1 + qnorm(runif(n)) + 2 * dat$x1
cv <- cv_piqr(y ~ x1 + x2 + x3, formula.p = ~ slp(p, 2), data = dat,
              nfolds = 5, nlambda = 20)
cv
plot(cv)
coef(cv$fit, pos.lambda = cv$pos.1se)
```

---

gof.niqr *Information Criteria for Nonlinear Parametric Quantile Models*

Description

Computes the goodness-of-fit criteria of Bottai and Cilluffo (2020, Section 3) for one or more `niqr()` fits.

Usage

```
gof.niqr(object, ...)
```

Arguments

`object`	an object of class `"niqr"`.
`...`	further objects of class `"niqr"` fitted to the same data.

Details

With L_n the minimized (average) loss, k the number of parameters and n the sample size, the criteria are $AIC^* = \log L_n + k/n$ and $BIC^* = \log L_n + k\log(n)/(2n)$ (`AIC.log`, `BIC.log`), and $AIC = nL_n + 2k$, $BIC = nL_n + k\log n$. Smaller values indicate a better fit. Bottai and Cilluffo (2020) write the last two criteria in terms of L_n; here the total loss nL_n is used, otherwise the penalty would dominate. Note that, unlike the log versions, `AIC` and `BIC` depend on the scale of the response; the log versions are recommended.

Value

A data frame with columns `npar`, `loss`, `AIC`, `BIC`, `AIC.log` and `BIC.log` (one row per model).

References

Bottai, M. and Cilluffo, G. (2020). Nonlinear parametric quantile models. *Statistical Methods in Medical Research*, 29(12), 3757-3769. doi:10.1177/0962280220941159

See Also

`niqr()`, `test.fit.niqr()`

`gof.piqr` *Select the Tuning Parameter of a Penalized QRCM by AIC or BIC*

Description

Computes an information criterion along the path of a `piqr()` fit and returns the value of λ at which it is minimized.

Usage

```
gof.piqr(
  object,
  method = c("BIC", "AIC"),
  Cn = 1,
  plot = TRUE,
  logi = TRUE,
  df = NULL,
  ...
)
```

Arguments

`object`	an object of class `"piqr"`.
`method`	the criterion, `"BIC"` (default) or `"AIC"`.

`Cn`	a positive constant (or a character string with an expression in `n`, the sample size, and `q`, the number of columns of the model matrix, e.g. `"log(log(n))"`) multiplying the BIC penalty, see Details.
`plot`	logical: plot the criterion against $\log \lambda$?
`logi`	logical: if `TRUE` (default) the criteria are computed on the logarithm of the loss (Sottile et al., 2020), otherwise on the loss.
`df`	how to count the degrees of freedom: `"parameters"`, the number of nonzero coefficients θ_{jh} (Sottile et al., 2020, Section 3.1), or `"covariates"`, the number of covariates with at least one nonzero coefficient (including the intercept), as in 'qrcmNP' 0.2.x. The default is `"parameters"`, or `"covariates"` for fits with `method = "legacy"`.
`...`	further graphical parameters.

Details

With `logi = TRUE`, the criteria of Sottile et al. (2020, eq. 3.6-3.7) are

$$AIC(\lambda) = \log L(\hat{\theta}_\lambda) + 2\, df(\lambda)/n, \qquad BIC(\lambda) = \log L(\hat{\theta}_\lambda) + C_n \log(n)\, df(\lambda)/n,$$

where L is the minimized integrated loss. With `logi = FALSE`, $AIC = L + 2\,df$ and $BIC = L + C_n \log(n)\, df$. Note that Bottai and Cilluffo (2020) define the log-criteria with df/n and $\log(n) df/(2n)$; the selected model is the same as with the definition above when $C_n = 1/2$ for the BIC.

Value

A list with items `call`, `minLambda` (the selected value of λ), `dfMinLambda` (its degrees of freedom), `posMinLambda` (its position in `object$lambda`), `betaMin` (the coefficients at the selected λ) and `curve` (the criterion along the path).

References

Sottile, G., Frumento, P., Chiodi, M. and Bottai, M. (2020). A penalized approach to covariate selection through quantile regression coefficient models. *Statistical Modelling*, 20(4), 369-385. doi:10.1177/1471082X19825523

Lee, E. R., Noh, H. and Park, B. U. (2014). Model selection via Bayesian information criterion for quantile regression models. *Journal of the American Statistical Association*, 109(505), 216-229. doi:10.1080/01621459.2013.836975

See Also

`piqr()`, `cv_piqr()`

Examples

```
set.seed(1)
n <- 200
x1 <- runif(n); x2 <- runif(n)
y <- 1 + qnorm(runif(n)) + 2 * x1
```

```
fit <- piqr(y ~ x1 + x2, formula.p = ~ slp(p, 2), nlambda = 20)
gof.piqr(fit, method = "AIC")
gof.piqr(fit, method = "BIC", df = "parameters", plot = FALSE)$posMinLambda
```

---

niqr *Nonlinear Parametric Quantile Models*

Description

Estimates a parametric model for the whole conditional quantile function, possibly nonlinear in the parameters, in the covariates, or both, by minimizing the integrated loss function of Frumento and Bottai (2016), as described by Bottai and Cilluffo (2020).

Usage

```
niqr(
  formula,
  data,
  fun,
  fun2 = NULL,
  x0,
  weights,
  cluster,
  control = niqr.control(),
  ...
)
```

Arguments

`formula`	a two-sided formula `y ~ x1 + x2 + ...`; the model matrix (including the intercept, if any) is passed to `fun2` as `X`.
`data`	an optional data frame containing the variables in `formula` (and, possibly, `weights` and `cluster`).
`fun`	a function `fun(theta, p)` that returns the matrix of coefficient functions $\beta(p \mid \theta)$, with one row for each element of the vector `p` and one column for each column of the model matrix. It must be vectorized in `p`.
`fun2`	an optional function `fun2(beta, X)` returning the quantile function $Q(p_i \mid x_i, \theta)$, where `beta` is the matrix returned by `fun` (row i evaluated at p_i) and `X` the model matrix (row i = x_i). The default (`NULL`) is the model linear in the covariates, $Q(p \mid x) = x^\top \beta(p)$, i.e. `rowSums(beta * X)`.
`x0`	starting values for θ. If named, the names are used to label the estimates.
`weights`	optional non-negative weights: a vector, or a variable in `data` given without quotes. They are normalized to have mean 1 and are treated as sampling (inverse-probability) weights in the sandwich formula.

`cluster`	an optional vector, or a variable in `data` given without quotes, identifying clusters of correlated observations. It is used to compute cluster-robust standard errors (sandwich) or a cluster bootstrap.
`control`	a list of control parameters, see `niqr.control()`.
`...`	not used. Arguments of the interface of 'qrcmNP' 0.2.x (`X`, `y`) give an informative error.

Details

Let $Q(p \mid x, \theta)$ be a parametric model for the conditional quantile function of y given x. The estimator $\hat{\theta}$ minimizes

$$L_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} w_i \int_0^1 \rho_p\{y_i - Q(p \mid x_i, \theta)\}\, dp,$$

where $\rho_p(u) = u\{p - I(u < 0)\}$. The integral is computed with the trapezoidal rule on a grid of `n.p` values of p (denser in the tails), splitting the interval containing $p_i^* = F(y_i \mid x_i, \theta)$, where the integrand is not differentiable. The gradient $-n^{-1} \sum_i w_i \int_0^1 \{p - I(p \geq p_i^*)\}\, \partial_\theta Q(p \mid x_i, \theta)\, dp$ is computed in the same way, with numerical derivatives of `fun` (or of `fun2`) with respect to θ. The loss is minimized by a damped Newton algorithm that uses the Hessian

$$H_n(\theta) = \frac{1}{n} \sum_i w_i \, \frac{\partial_\theta Q(p_i^* \mid x_i, \theta)\, \partial_\theta Q(p_i^* \mid x_i, \theta)^\top}{Q'(p_i^* \mid x_i, \theta)},$$

where Q' is the derivative with respect to p, i.e. the inverse of the conditional density. H_n is positive semi-definite and estimates the Hessian of the expected loss at the true parameter.

The asymptotic covariance matrix of $\hat{\theta}$ is estimated by the sandwich formula $H^{-1}\Omega H^{-1}/n$ (Bottai and Cilluffo, 2020, Theorem 3), where Ω is the covariance of the individual contributions to the gradient (summed within clusters if `cluster` is supplied), or by the (cluster) bootstrap (`niqr.control(se = "boot")`).

The model is assumed to be monotonically increasing in p for all observations. A warning is given if quantile crossing is detected. Good starting values `x0` are important: the loss function may have several local minima when Q is not one-to-one in θ (Bottai and Cilluffo, 2020, Theorem 1).

Value

An object of class `"niqr"`, a list with items:

`coefficients`	the estimated θ.
`se`	standard errors.
`covar`	the estimated covariance matrix of $\hat{\theta}$.
`obj.function`	the minimized loss $L_n(\hat{\theta})$.
`gradient`, `hessian`, `omega`	gradient, Hessian and outer product of the gradient at $\hat{\theta}$.
`CDF`, `PDF`	fitted conditional CDF and density at the observed responses.
`converged`, `n.it`	convergence indicator and number of iterations.
`call`, `terms`, `X`, `y`, `weights`, `cluster`, `fun`, `fun2`, `control`	the elements of the model.

Note

The matrix interface `niqr(fun, fun2, x0, X, y)` of 'qrcmNP' 0.2.x was removed in version 1.0.0: use `niqr(y ~ x, data, fun, fun2, x0)`, where the model matrix of the formula plays the role of the old `X`.

References

Bottai, M. and Cilluffo, G. (2020). Nonlinear parametric quantile models. *Statistical Methods in Medical Research*, 29(12), 3757-3769. doi:10.1177/0962280220941159

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

See Also

`niqr.control()`, `summary.niqr()`, `predict.niqr()`, `plot.niqr()`, `gof.niqr()`, `test.fit.niqr()`.

Examples

```
## Model linear in the covariates, nonlinear in the parameters:
## beta0(p) = theta1 + exp(theta2 * p), beta1(p) = theta3 + theta4 * p
set.seed(1234)
n <- 300
x <- runif(n)
fun <- function(theta, p) {
  cbind(theta[1] + exp(theta[2] * p), theta[3] + theta[4] * p)
}
u <- runif(n)
beta <- fun(c(1, 1, 1, 1), u)
y <- beta[, 1] + beta[, 2] * x
m <- niqr(y ~ x, fun = fun, x0 = c(a = 0.5, b = 0.5, c = 0.5, d = 0.5))
summary(m)
predict(m, type = "QF", p = c(0.1, 0.5, 0.9), newdata = data.frame(x = 0.5))

## Model nonlinear in the covariate: Hill dose-response quantile function
## Q(p | dose) = b0(p) + (b1 - b0(p)) / (1 + 10^(b3 - log10(dose)))

set.seed(1)
dose <- rep(10^seq(-2, 2, length.out = 20), each = 15)
fun.h <- function(theta, p) {
  cbind(theta[1] + exp(theta[2]) * qnorm(p), theta[3], theta[4])
}
fun2.h <- function(beta, X) {
  beta[, 1] + (beta[, 2] - beta[, 1]) / (1 + 10^(beta[, 3] - log10(X[, 2])))
}
yh <- fun2.h(fun.h(c(1, log(0.5), 10, 0), runif(length(dose))), cbind(1, dose))
mh <- niqr(yh ~ dose, fun = fun.h, fun2 = fun2.h, x0 = c(0, 0, 5, 0.5))
summary(mh)
```

`niqr.control` *Control Parameters for niqr*

Description

Control Parameters for niqr

Usage

```
niqr.control(
  tol = 1e-06,
  maxit = 100,
  n.p = 199,
  eps = 1e-04,
  h = .Machine$double.eps^(1/3),
  se = c("sandwich", "boot", "none"),
  R = 200,
  verbose = FALSE
)
```

Arguments

`tol`	convergence tolerance on the relative change of the parameters (default `1e-6`).
`maxit`	maximum number of iterations (default 100).
`n.p`	number of grid points used to compute the integrals over p (default 199).
`eps`	the grid covers $[\epsilon, 1-\epsilon]$ (default `1e-4`).
`h`	relative step for numerical derivatives (default `.Machine$double.eps^(1/3)`).
`se`	how to estimate the covariance matrix: `"sandwich"` (default), `"boot"` (nonparametric bootstrap, or cluster bootstrap if `cluster` is supplied), or `"none"`.
`R`	number of bootstrap replications (default 200).
`verbose`	logical: print the progress of the algorithm?

Value

A list of control parameters.

See Also

`niqr()`

piqr *Penalized Quantile Regression Coefficients Modeling*

Description

Fits the penalized integrated loss minimization (PILM) estimator of Sottile et al. (2020) along a path of values of the tuning parameter λ. Quantile regression coefficients are modelled as parametric functions of the quantile order p, and covariates are selected using the information on all quantiles simultaneously.

Usage

```
piqr(
  formula,
  formula.p = ~slp(p, 3),
  weights,
  data,
  s,
  nlambda = 100,
  lambda.min.ratio = ifelse(nobs < nvars, 0.01, 1e-04),
  lambda,
  tol = 1e-06,
  maxit = 100,
  penalty = c("lasso", "group"),
  penalty.factor,
  method = c("pnewton", "legacy"),
  verbose = FALSE,
  ...
)
```

Arguments

formula	a two-sided formula of the form y ~ x1 + x2 + ... The response can be a numeric vector or a survival::Surv() object of type "right" (right censoring) or "counting" (left truncation and right censoring), as in qrcm::iqr().
formula.p	a one-sided formula of the form ~ b1(p, ...) + b2(p, ...) + ... describing how the quantile regression coefficients depend on p. The default is a shifted Legendre polynomial of degree 3, see qrcm::slp().
weights	an optional vector of non-negative weights.
data	an optional data frame, list or environment containing the variables in formula.
s	an optional $q \times k$ 0/1 matrix that permits excluding some model coefficients (q = number of columns of the model matrix, k = number of basis functions in formula.p).
nlambda	number of values of λ (default 100).

`lambda.min.ratio`
: smallest value of λ, as a fraction of λ_{max}. The default is `1e-4` if the number of observations is larger than the number of columns of the model matrix, and `0.01` otherwise.

`lambda`
: an optional user-supplied decreasing sequence of values of λ (on the internal scale, see Details). It overrides `nlambda` and `lambda.min.ratio`.

`tol`
: convergence tolerance (default `1e-6`).

`maxit`
: maximum number of Newton iterations for each λ (default 100).

`penalty`
: either `"lasso"` (default), the ℓ_1 penalty $\lambda \sum_{j,h} w_j |\theta_{jh}|$ of Sottile et al. (2020, eq. 3.4), or `"group"`, the group lasso penalty $\lambda \sum_j w_j \sqrt{k_j}\, \|\theta_{j\cdot}\|_2$ that selects covariates (all the coefficients $\theta_{j1}, \ldots, \theta_{jk}$ of the j-th covariate are either zero or nonzero). Groups are the columns of the model matrix (each dummy variable of a factor is a separate group), and k_j is the number of free coefficients of column j. Ignored if `method = "legacy"`.

`penalty.factor`
: optional vector of non-negative penalty weights w_j, one for each column of the model matrix except the intercept (which is never penalized). A zero weight leaves the covariate unpenalized (it is always in the model). The default is 1 for all covariates. Different weights can be used, for example, to compute an adaptive lasso.

`method`
: `"pnewton"` (default) for the proximal Newton algorithm described in Details, or `"legacy"` for the algorithm of 'qrcmNP' 0.2.x, kept for reproducibility of published results.

`verbose`
: logical: if `TRUE`, the progress along the path is printed.

`...`
: not used (an error is given for the removed argument `display`).

Details

Under the quantile regression coefficients model (QRCM) of Frumento and Bottai (2016), the conditional quantile function is

$$Q(p \mid x, \theta) = x^\top \theta\, b(p),$$

where $b(p)$ is a set of k known functions of p (defined by `formula.p`) and θ is a $q \times k$ matrix of parameters. The penalized estimator minimizes

$$L_{PEN}(\theta) = \int_0^1 \sum_i w_i \rho_p\{y_i - Q(p \mid x_i, \theta)\}\, dp + \lambda\, \mathrm{pen}(\theta),$$

where ρ_p is the check loss of quantile regression. As suggested by Sottile et al. (2020), x and $b(p)$ are standardized, and the response is mapped to the interval $[0, 10]$; the values of `lambda` refer to this internal scale. The estimated coefficients are returned on the original scale.

With `method = "pnewton"`, for each λ the integrated loss is replaced by its second-order expansion (gradient and Hessian are available in closed form) and the resulting penalized quadratic problem is solved by coordinate descent (`"lasso"`) or block coordinate descent (`"group"`); a backtracking line search guarantees the decrease of the penalized objective (proximal Newton method). The path starts from λ_{max}, the smallest value at which all penalized coefficients are zero, and uses warm starts, sequential strong rules and a final check of the Karush-Kuhn-Tucker conditions. With censored or truncated data, the estimating equations of Frumento and Bottai (2017) replace the

gradient of the loss; since they are not the gradient of a loss function, the symmetric part of their Jacobian is used in the quadratic approximation, and the iterate with the smallest violation of the KKT conditions is returned (convergence is declared when the average violation is below `5e-4`).

The intercept (the first row of θ) is never penalized. Use `gof.piqr()` to select λ by AIC or BIC, `cv_piqr()` for cross-validation, and `refit_piqr()` to re-estimate the selected model without penalty and obtain standard errors (Sottile et al., 2020, Section 3.2).

Value

An object of class `"piqr"`, a list with items:

`call`	the matched call.
`lambda`	the sequence of values of λ.
`coefficients`	a list (one element for each λ) of $q \times k$ matrices of estimated coefficients.
`covar`	a list of covariance matrices (only with `method = "legacy"`, otherwise `NULL`; use `refit_piqr()` for inference).
`obj.function`	the minimized (unpenalized) integrated loss, on the original scale, for each λ. For censored and truncated data this is the loss function of 'qrcm', which is not the function minimized.
`df`	number of covariates (including the intercept) with nonzero coefficients, for each λ.
`npar`	number of nonzero coefficients, for each λ.
`dl`	matrix of the ℓ_1-norms of the rows of the (unpenalized) gradient, one column for each λ.
`seqS`	a list of 0/1 matrices indicating the nonzero coefficients.
`penalty`, `penalty.factor`, `method`	as in the call.
`converged`	logical vector, convergence at each λ.
`internal`	a list of objects used by the methods.

References

Sottile, G., Frumento, P., Chiodi, M. and Bottai, M. (2020). A penalized approach to covariate selection through quantile regression coefficient models. *Statistical Modelling*, 20(4), 369-385. doi:10.1177/1471082X19825523

Frumento, P. and Bottai, M. (2016). Parametric modeling of quantile regression coefficient functions. *Biometrics*, 72(1), 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*, 73(4), 1179-1188. doi:10.1111/biom.12675

Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1-22. doi:10.18637/jss.v033.i01

Lee, J. D., Sun, Y. and Saunders, M. A. (2014). Proximal Newton-type methods for minimizing composite functions. *SIAM Journal on Optimization*, 24(3), 1420-1443. doi:10.1137/130921428

See Also

gof.piqr(), cv_piqr(), refit_piqr(), summary.piqr(), plot.piqr(), predict.piqr().

Examples

```
set.seed(1234)
n <- 300
x1 <- rexp(n)
x2 <- runif(n, 0, 5)
x3 <- rnorm(n)                      # irrelevant covariate
# true model: beta0(p) = 1 + qnorm(p), beta1(p) = 1 + 2 p, beta2(p) = 1
u <- runif(n)
y <- 1 + qnorm(u) + (1 + 2 * u) * x1 + x2
dat <- data.frame(y, x1, x2, x3)

fit <- piqr(y ~ x1 + x2 + x3, formula.p = ~ slp(p, 2), data = dat, nlambda = 30)
fit
best <- gof.piqr(fit, method = "BIC", plot = FALSE)
coef(fit, pos.lambda = best$posMinLambda)

# group lasso: covariates enter or leave the model as a whole
fitg <- piqr(y ~ x1 + x2 + x3, formula.p = ~ slp(p, 2), data = dat,
             nlambda = 30, penalty = "group")
plot(fitg, xvar = "lambda")

# unpenalized re-estimation of the selected model (with standard errors)

m <- refit_piqr(fit, pos.lambda = best$posMinLambda)
summary(m)
```

---

plot.niqr *Plot a Nonlinear Parametric Quantile Model*

Description

Plots the estimated coefficient functions (the columns of fun(theta, p)) with pointwise confidence bands, and a uniform QQ-plot of the fitted CDF.

Usage

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

Arguments

x	an object of class "niqr".
conf.int	logical: draw 95% confidence bands?
which	an optional vector of indices of the coefficient functions to plot (use length(beta) + 1 for the QQ-plot). If missing and ask = TRUE, a menu is shown.
ask	logical: choose interactively what to plot? Default interactive().
polygon	logical: draw the confidence bands as shaded areas?
...	additional graphical parameters.

Value

No return value, called for its side effects.

See Also

niqr()

---

plot.piqr *Plot a Penalized QRCM*

Description

Coefficient paths, gradient paths and estimated coefficient functions $\hat{\beta}(p)$ of a piqr() fit.

Usage

```
## S3 method for class 'piqr'
plot(
  x,
  xvar = c("lambda", "objective", "grad", "beta"),
  pos.lambda,
  label = FALSE,
  which = NULL,
  ask = interactive(),
  polygon = TRUE,
  ...
)
```

Arguments

x	an object of class "piqr".
xvar	what to plot: "lambda" (coefficient paths against $\log\lambda$), "objective" (coefficient paths against the minimized loss), "grad" (ℓ_1-norm of the gradient of each covariate against $\log\lambda$), or "beta" (the coefficient functions $\hat{\beta}_j(p)$ at pos.lambda).

pos.lambda	the position of the selected λ (for xvar = "beta").
label	logical: label the curves (paths only)?
which	for xvar = "beta", an optional vector of indices of the coefficients to plot. If missing and ask = TRUE, a menu is shown.
ask	logical: for xvar = "beta", choose interactively which coefficient to plot? Defaults to interactive().
polygon	logical: for xvar = "beta" with standard errors, draw confidence bands as shaded areas?
...	additional graphical parameters (xlab, ylab, col, lwd, xlim, ylim, main, ...).

Value

No return value, called for its side effects.

See Also

piqr()

Examples

```
set.seed(1)
n <- 200
x1 <- runif(n); x2 <- runif(n); x3 <- runif(n)
y <- 1 + qnorm(runif(n)) + 2 * x1 - x2
fit <- piqr(y ~ x1 + x2 + x3, formula.p = ~ slp(p, 2), nlambda = 20)
oldpar <- par(mfrow = c(1, 2))
plot(fit, xvar = "lambda")
plot(fit, xvar = "grad")
par(oldpar)
plot(fit, xvar = "beta", pos.lambda = 20, which = 2)
```

---

predict.niqr *Prediction after Nonlinear Parametric Quantile Models*

Description

Prediction after Nonlinear Parametric Quantile Models

Usage

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

Arguments

object	an object of class "niqr".
type	the type of prediction: "beta" (the coefficient functions, i.e. the columns of fun(theta, p)), "CDF" and "PDF" (fitted conditional distribution and density at the observed response), "QF" (conditional quantile function) or "sim" (simulated responses).
newdata	an optional data frame (for type = "CDF" and "PDF" it must contain the response).
p	a vector of quantile orders (for type = "beta" and "QF").
se	logical: compute delta-method standard errors ("beta" and "QF")?
...	not used.

Value

- type = "beta": a list with one data frame for each column of fun, with columns p, beta, and (if se = TRUE) se, low, up (95% pointwise confidence limits).
- type = "CDF" or "PDF": a vector.
- type = "QF": a data frame with one column for each p or, if se = TRUE, a list with items fit and se.fit.
- type = "sim": a vector of simulated responses.

See Also

niqr()

---

predict.piqr *Prediction after Penalized QRCM*

Description

Predictions from a piqr() fit at a selected value of λ.

Usage

```
## S3 method for class 'piqr'
predict(
  object,
  pos.lambda,
  type = c("beta", "CDF", "QF", "sim"),
  newdata,
  p,
  se = !is.null(object$covar),
  lambda,
  ...
)
```

Arguments

`object`	an object of class `"piqr"`.
`pos.lambda`	the position of the selected value of λ.
`type`	the type of prediction: `"beta"` (quantile regression coefficients $\beta(p)$), `"CDF"` (fitted conditional distribution and density functions at the observed response), `"QF"` (conditional quantile function) or `"sim"` (simulated responses).
`newdata`	an optional data frame in which to look for the covariates (and, for `type = "CDF"`, the response). If omitted, the data used to fit the model are used.
`p`	a vector of quantile orders (for `type = "beta"` and `type = "QF"`).
`se`	logical: should standard errors be computed? They are available only with `method = "legacy"`; see `refit_piqr()` otherwise.
`lambda`	alternatively to `pos.lambda`, a value of λ.
`...`	not used.

Value

- `type = "beta"`: a list with one data frame for each covariate, with columns `p`, `beta` (and `se`, `low`, `up` if `se = TRUE`).
- `type = "CDF"`: a data frame with columns `CDF` and `PDF`.
- `type = "QF"`: a data frame of fitted quantiles (one column for each `p`) or, if `se = TRUE`, a list with items `fit` and `se.fit`.
- `type = "sim"`: a vector of simulated responses.

See Also

`piqr()`

Examples

```
set.seed(1)
n <- 200
x <- runif(n)
y <- qexp(runif(n)) * (1 + x)
fit <- piqr(y ~ x, formula.p = ~ slp(p, 3), nlambda = 10)
predict(fit, pos.lambda = 10, type = "QF", p = c(0.25, 0.5),
        newdata = data.frame(x = c(0, 0.5, 1)))
head(predict(fit, pos.lambda = 10, type = "CDF"))
```

`refit_piqr` *Unpenalized Re-estimation of the Model Selected by a Penalized QRCM*

Description

Following Sottile et al. (2020, Section 3.2), the penalized estimator is used to select the covariates, and the selected model is re-estimated without penalty with `qrcm::iqr()`, which provides asymptotic standard errors, Wald tests and goodness-of-fit diagnostics.

Usage

```
refit_piqr(
  object,
  pos.lambda,
  criterion = c("BIC", "AIC"),
  keep = c("covariates", "coefficients"),
  data,
  ...
)
```

Arguments

`object`	an object of class `"piqr"`.
`pos.lambda`	the position of the selected value of λ. If missing, it is selected with `gof.piqr()` using `criterion`.
`criterion`	`"BIC"` (default) or `"AIC"`, used if `pos.lambda` is missing.
`keep`	`"covariates"` (default): the selected covariates are re-estimated with all the basis functions allowed by the original `s`; `"coefficients"`: only the nonzero coefficients are re-estimated.
`data`	an optional data frame; by default, the `data` of the original call.
`...`	further arguments passed to `gof.piqr()`.

Value

An object of class `"iqr"` (see `qrcm::iqr()`), with an additional attribute `"pos.lambda"`.

Note

The standard errors are conditional on the selected model and do not account for the selection step.

See Also

`piqr()`, `gof.piqr()`, `qrcm::iqr()`, `qrcm::summary.iqr()`

Examples

```
set.seed(1)
n <- 300
x1 <- runif(n); x2 <- runif(n)
dat <- data.frame(y = 1 + qnorm(runif(n)) + 2 * x1, x1, x2)
fit <- piqr(y ~ x1 + x2, formula.p = ~ slp(p, 2), data = dat, nlambda = 20)
m <- refit_piqr(fit)
summary(m)
```

`summary.niqr` *Summarize a Nonlinear Parametric Quantile Model*

Description

Summarize a Nonlinear Parametric Quantile Model

Usage

```
## S3 method for class 'niqr'
summary(object, p, ...)

## S3 method for class 'summary.niqr'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`object`	an object of class `"niqr"`.
`p`	an optional vector of quantile orders: if supplied, the summary reports the coefficient functions $\beta(p \mid \hat{\theta})$ (the columns of `fun`) at each `p`, with delta-method standard errors.
`...`	for future methods.
`x`	an object of class `"summary.niqr"`.
`digits`	number of significant digits.

Value

An object of class `"summary.niqr"`. Without `p`: the table of estimated parameters with standard errors, z-values and p-values, the minimized loss and the information criteria of `gof.niqr()`. With `p`: a list with one table for each value of `p`.

See Also

`niqr()`, `gof.niqr()`

summary.piqr *Summarize a Penalized QRCM*

Description

Summary of a `piqr()` fit at a selected value of λ.

Usage

```
## S3 method for class 'piqr'
summary(object, pos.lambda, SE = FALSE, p, cov = FALSE, lambda, ...)

## S3 method for class 'summary.piqr'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`object`	an object of class `"piqr"`.
`pos.lambda`	the position of the selected value of λ in `object$lambda` (e.g., `gof.piqr(object)$posMinLambda`).
`SE`	logical: if `TRUE`, standard errors of θ are reported (available only with `method = "legacy"`; they ignore the selection step, see `refit_piqr()`).
`p`	an optional vector of quantile orders. If supplied, the summary reports $\beta(p)$ at each p.
`cov`	logical: if `TRUE` and `p` is supplied, the covariance matrix of $\beta(p)$ is reported (only with `method = "legacy"`).
`lambda`	alternatively to `pos.lambda`, a value of λ; the closest value in `object$lambda` is used.
`...`	for future methods.
`x`	an object of class `"summary.piqr"`.
`digits`	number of significant digits.

Value

An object of class `"summary.piqr"`. Without `p`: a list with the selected `lambda`, the matrix of `coefficients`, their standard errors `se` (if requested and available), the minimized loss `obj.function`, the number of observations `n` and of nonzero parameters `free.par`. With `p`: a list with one element for each value of `p`, containing the estimated $\beta(p)$ (and standard errors, if available).

See Also

`piqr()`, `gof.piqr()`, `refit_piqr()`.

Examples

```
set.seed(1)
n <- 200
x1 <- runif(n); x2 <- runif(n)
y <- 1 + qnorm(runif(n)) + 2 * x1
fit <- piqr(y ~ x1 + x2, formula.p = ~ slp(p, 2), nlambda = 20)
best <- gof.piqr(fit, plot = FALSE)
summary(fit, pos.lambda = best$posMinLambda)
summary(fit, pos.lambda = best$posMinLambda, p = c(0.1, 0.5, 0.9))
```

`test.fit.niqr` *Goodness-of-Fit Test for Nonlinear Parametric Quantile Models*

Description

Under a correctly specified model, the fitted values of the conditional CDF, $F(y_i \mid x_i, \hat{\theta})$, are approximately uniformly distributed. This function computes the Kolmogorov-Smirnov and Cramer-von Mises statistics of their empirical distribution, and obtains p-values by parametric bootstrap, as in `qrcm::test.fit()`.

Usage

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

Arguments

`object`	an object of class `"niqr"`.
`R`	number of bootstrap replications (default 100).
`trace`	logical: print the progress?
`...`	not used.

Value

A matrix with the test statistics and their bootstrap p-values.

See Also

`niqr()`, `gof.niqr()`

Index

coef.piqr, 2
cv_piqr, 3
cv_piqr(), *6*, *13, 14*

gof.niqr, 4
gof.niqr(), *9*, *20*, *22*
gof.piqr, 5
gof.piqr(), *4*, *13, 14*, *19*, *21*

niqr, 7
niqr(), *4, 5*, *10*, *15*, *17*, *20*, *22*
niqr.control, 10
niqr.control(), *8, 9*

piqr, 11
piqr(), *3–6*, *15–19*, *21*
plot.niqr, 14
plot.niqr(), *9*
plot.piqr, 15
plot.piqr(), *14*
predict.niqr, 16
predict.niqr(), *9*
predict.piqr, 17
predict.piqr(), *14*
print.summary.niqr *(*summary.niqr*)*, 20
print.summary.piqr *(*summary.piqr*)*, 21

qrcm::iqr(), *11*, *19*
qrcm::slp(), *11*
qrcm::summary.iqr(), *19*
qrcm::test.fit(), *22*

refit_piqr, 19
refit_piqr(), *13, 14*, *18*, *21*

summary.niqr, 20
summary.niqr(), *9*
summary.piqr, 21
summary.piqr(), *14*
survival::Surv(), *11*

test.fit.niqr, 22
test.fit.niqr(), *5*, *9*