# Package: conquer (via r-universe)

October 12, 2024

**Type** Package

**Title** Convolution-Type Smoothed Quantile Regression

**Version** 1.3.3

**Date** 2023-03-05

**Description** Estimation and inference for conditional linear quantile regression models using a convolution smoothed approach. In the low-dimensional setting, efficient gradient-based methods are employed for fitting both a single model and a regression process over a quantile range. Normal-based and (multiplier) bootstrap confidence intervals for all slope coefficients are constructed. In high dimensions, the conquer method is complemented with flexible types of penalties (Lasso, elastic-net, group lasso, sparse group lasso, scad and mcp) to deal with complex low-dimensional structures.

**Depends** R (>= 3.5.0)

**License** GPL-3

**Encoding** UTF-8

**URL** https://github.com/XiaoouPan/conquer

**SystemRequirements** C++17

**Imports** Rcpp (>= 1.0.3), Matrix, matrixStats, stats

**LinkingTo** Rcpp, RcppArmadillo (>= 0.9.850.1.0)

**RoxygenNote** 7.2.1

**NeedsCompilation** yes

**Author** Xuming He [aut], Xiaoou Pan [aut, cre], Kean Ming Tan [aut], Wen-Xin Zhou [aut]

**Maintainer** Xiaoou Pan <xip024@ucsd.edu>

**Repository** CRAN

**Date/Publication** 2023-03-06 08:40:02 UTC

# Contents



---

`conquer-package` *Conquer: Convolution-Type Smoothed Quantile Regression*

---

## Description

Estimation and inference for conditional linear quantile regression models using a convolution smoothed approach. In the low-dimensional setting, efficient gradient-based methods are employed for fitting both a single model and a regression process over a quantile range. Normal-based and (multiplier) bootstrap confidence intervals for all slope coefficients are constructed. In high dimensions, the conquer methods complemented with $\ell_1$-penalization and iteratively reweighted $\ell_1$-penalization are used to fit sparse models. Commonly used penalities, such as the elastic-net, group lasso and sparse group lasso, are also incorporated to deal with more complex low-dimensional structures.

## Author(s)

Xuming He <xmhe@umich.edu>, Xiaoou Pan <xip024@ucsd.edu>, Kean Ming Tan <keanming@umich.edu>, and Wen-Xin Zhou <wez243@ucsd.edu>

## References

Barzilai, J. and Borwein, J. M. (1988). Two-point step size gradient methods. IMA J. Numer. Anal., 8, 141–148.

Belloni, A. and Chernozhukov, V. (2011). $\ell_1$ penalized quantile regression in high-dimensional sparse models. Ann. Statist., 39, 82-130.

Fan, J., Liu, H., Sun, Q. and Zhang, T. (2018). I-LAMM for sparse learning: Simultaneous control of algorithmic complexity and statistical error. Ann. Statist., 46, 814-841.

Fernandes, M., Guerre, E. and Horta, E. (2021). Smoothing quantile regressions. J. Bus. Econ. Statist., 39, 338-357.

He, X., Pan, X., Tan, K. M., and Zhou, W.-X. (2022+). Smoothed quantile regression for large-scale inference. J. Econometrics, in press.

Koenker, R. (2005). Quantile Regression. Cambridge University Press, Cambridge.

Koenker, R. and Bassett, G. (1978). Regression quantiles. Econometrica, 46, 33-50.

Tan, K. M., Wang, L. and Zhou, W.-X. (2022). High-dimensional quantile regression: convolution smoothing and concave regularization. J. Roy. Statist. Soc. Ser. B, 84(1), 205-233.

conquer *Convolution-Type Smoothed Quantile Regression*

## Description

Estimation and inference for conditional linear quantile regression models using a convolution smoothed approach. Efficient gradient-based methods are employed for fitting both a single model and a regression process over a quantile range. Normal-based and (multiplier) bootstrap confidence intervals for all slope coefficients are constructed.

## Usage

```
conquer(
  X,
  Y,
  tau = 0.5,
  kernel = c("Gaussian", "logistic", "uniform", "parabolic", "triangular"),
  h = 0,
  checkSing = FALSE,
  tol = 1e-04,
  iteMax = 5000,
  stepBounded = TRUE,
  stepMax = 100,
  ci = c("none", "bootstrap", "asymptotic", "both"),
  alpha = 0.05,
  B = 1000
)
```

## Arguments

| | |
|---|---|
| X | An $n$ by $p$ design matrix. Each row is a vector of observations with $p$ covariates. Number of observations $n$ must be greater than number of covariates $p$. |
| Y | An $n$-dimensional response vector. |
| tau | (**optional**) The desired quantile level. Default is 0.5. Value must be between 0 and 1. |
| kernel | (**optional**) A character string specifying the choice of kernel function. Default is "Gaussian". Choices are "Gaussian", "logistic", "uniform", "parabolic" and "triangular". |
| h | (**optional**) Bandwidth/smoothing parameter. Default is $\max\{((log(n)+p)/n)^{0.4}, 0.05\}$. The default will be used if the input value is less than or equal to 0. |
| checkSing | (**optional**) A logical flag. Default is FALSE. If checkSing = TRUE, then it will check if the design matrix is singular before running conquer. |
| tol | (**optional**) Tolerance level of the gradient descent algorithm. The iteration will stop when the maximum magnitude of all the elements of the gradient is less than tol. Default is 1e-04. |

| | |
|---|---|
| iteMax | (**optional**) Maximum number of iterations. Default is 5000. |
| stepBounded | (**optional**) A logical flag. Default is TRUE. If stepBounded = TRUE, then the step size of gradient descent is upper bounded by stepMax. If stepBounded = FALSE, then the step size is unbounded. |
| stepMax | (**optional**) Maximum bound for the gradient descent step size. Default is 100. |
| ci | (**optional**) A character string specifying methods to construct confidence intervals. Choices are "none" (default), "bootstrap", "asymptotic" and "both". If ci = "none", then confidence intervals will not be constructed. If ci = "bootstrap", then three types of confidence intervals (percentile, pivotal and normal) will be constructed via multiplier bootstrap. If ci = "asymptotic", then confidence intervals will be constructed based on estimated asymptotic covariance matrix. If ci = "both", then confidence intervals from both bootstrap and asymptotic covariance will be returned. |
| alpha | (**optional**) Miscoverage level for each confidence interval. Default is 0.05. |
| B | (**optional**) The size of bootstrap samples. Default is 1000. |

## Value

An object containing the following items will be returned:

coeff A $(p+1)$-vector of estimated quantile regression coefficients, including the intercept.

ite Number of iterations until convergence.

residual An $n$-vector of fitted residuals.

bandwidth Bandwidth value.

tau Quantile level.

kernel Kernel function.

n Sample size.

p Number of covariates.

perCI The percentile confidence intervals for regression coefficients. Only available if ci = "bootstrap" or ci = "both".

pivCI The pivotal confidence intervals for regression coefficients. Only available if ci = "bootstrap" or ci = "both".

normCI The normal-based confidence intervals for regression coefficients. Only available if ci = "bootstrap" or ci = "both".

asyCI The asymptotic confidence intervals for regression coefficients. Only available if ci = "asymptotic" or ci = "both".

## References

Barzilai, J. and Borwein, J. M. (1988). Two-point step size gradient methods. IMA J. Numer. Anal., 8, 141–148.

Fernandes, M., Guerre, E. and Horta, E. (2021). Smoothing quantile regressions. J. Bus. Econ. Statist., 39, 338-357.

He, X., Pan, X., Tan, K. M., and Zhou, W.-X. (2022+). Smoothed quantile regression for large-scale inference. J. Econometrics, in press.

Koenker, R. and Bassett, G. (1978). Regression quantiles. Econometrica, 46, 33-50.

## See Also

See conquer.process for smoothed quantile regression process.

## Examples

```
n = 500; p = 10
beta = rep(1, p)
X = matrix(rnorm(n * p), n, p)
Y = X %*% beta + rt(n, 2)

## Smoothed quantile regression with Gaussian kernel
fit.Gauss = conquer(X, Y, tau = 0.5, kernel = "Gaussian")
beta.hat.Gauss = fit.Gauss$coeff

## Smoothe quantile regression with uniform kernel
fit.unif = conquer(X, Y, tau = 0.5, kernel = "uniform")
beta.hat.unif = fit.unif$coeff

## Construct three types of confidence intervals via multiplier bootstrap
fit = conquer(X, Y, tau = 0.5, kernel = "Gaussian", ci = "bootstrap")
ci.per = fit$perCI
ci.piv = fit$pivCI
ci.norm = fit$normCI
```

---

# conquer.cv.reg *Cross-Validated Penalized Convolution-Type Smoothed Quantile Regression*

---

## Description

Fit sparse quantile regression models via regularized conquer methods with "lasso", "elastic-net", "group lasso", "sparse group lasso", "scad" and "mcp" penalties. The regularization parameter $\lambda$ is selected via cross-validation.

## Usage

```
conquer.cv.reg(
  X,
  Y,
  lambdaSeq = NULL,
  tau = 0.5,
  kernel = c("Gaussian", "logistic", "uniform", "parabolic", "triangular"),
  h = 0,
  penalty = c("lasso", "elastic", "group", "sparse-group", "scad", "mcp"),
  para.elastic = 0.5,
  group = NULL,
  weights = NULL,
  para.scad = 3.7,
```

```
  para.mcp = 3,
  kfolds = 5,
  numLambda = 50,
  epsilon = 0.001,
  iteMax = 500,
  phi0 = 0.01,
  gamma = 1.2,
  iteTight = 3
)
```

## Arguments

| | |
|---|---|
| `X` | An $n$ by $p$ design matrix. Each row is a vector of observations with $p$ covariates. |
| `Y` | An $n$-dimensional response vector. |
| `lambdaSeq` | (**optional**) A sequence of candidate regularization parameters. If unspecified, the sequence will be generated by a simulated pivotal quantity approach proposed in Belloni and Chernozhukov (2011). |
| `tau` | (**optional**) Quantile level (between 0 and 1). Default is 0.5. |
| `kernel` | (**optional**) A character string specifying the choice of kernel function. Default is "Gaussian". Choices are "Gaussian", "logistic", "uniform", "parabolic" and "triangular". |
| `h` | (**optional**) The bandwidth parameter for kernel smoothing. Default is $\max\{0.5*(log(p)/n)^{0.25}, 0.05\}$. The default will be used if the input value is less than or equal to 0. |
| `penalty` | (**optional**) A character string specifying the penalty. Default is "lasso" (Tibshirani, 1996). The other options are "elastic" for elastic-net (Zou and Hastie, 2005), "group" for group lasso (Yuan and Lin, 2006), "sparse-group" for sparse group lasso (Simon et al., 2013), "scad" (Fan and Li, 2001) and "mcp" (Zhang, 2010). |
| `para.elastic` | (**optional**) The mixing parameter between 0 and 1 (usually noted as $\alpha$) for elastic net. The penalty is defined as $\alpha\|\|\beta\|\|_1 + (1-\alpha)\|\|\beta\|\|_2^2$. Default is 0.5. Setting `para.elastic = 1` gives the lasso penalty, and setting `para.elastic = 0` yields the ridge penalty. Only specify it when `penalty = "elastic"`. |
| `group` | (**optional**) A $p$-dimensional vector specifying group indices. Only specify it if `penalty = "group"` or `penalty = "sparse-group"`. For example, if $p = 10$, and we assume the first 3 coefficients belong to the first group, and the last 7 coefficients belong to the second group, then the argument should be `group = c(rep(1, 3), rep(2, 7))`. If not specified, then the penalty will be the classical lasso. |
| `weights` | (**optional**) A vector specifying groups weights for group Lasso and sparse group Lasso. The length must be equal to the number of groups. If not specified, the default weights are square roots of group sizes. For example , if `group = c(rep(1, 3), rep(2, 7))`, then the default weights are $\sqrt{3}$ for the first group, and $\sqrt{7}$ for the second group. |
| `para.scad` | (**optional**) The constant parameter for "scad". Default value is 3.7. Only specify it if `penalty = "scad"`. |

`para.mcp` (**optional**) The constant parameter for "mcp". Default value is 3. Only specify it if `penalty = "mcp"`.

`kfolds` (**optional**) Number of folds for cross-validation. Default is 5.

`numLambda` (**optional**) Number of $\lambda$ values for cross-validation if `lambdaSeq` is unspeficied. Default is 50.

`epsilon` (**optional**) A tolerance level for the stopping rule. The iteration will stop when the maximum magnitude of the change of coefficient updates is less than `epsilon`. Default is 0.001.

`iteMax` (**optional**) Maximum number of iterations. Default is 500.

`phi0` (**optional**) The initial quadratic coefficient parameter in the local adaptive majorize-minimize algorithm. Default is 0.01.

`gamma` (**optional**) The adaptive search parameter (greater than 1) in the local adaptive majorize-minimize algorithm. Default is 1.2.

`iteTight` (**optional**) Maximum number of tightening iterations in the iteratively reweighted $\ell_1$-penalized algorithm. Only specify it if the penalty is scad or mcp. Default is 3.

## Value

An object containing the following items will be returned:

`coeff.min` A $(p+1)$ vector of estimated coefficients including the intercept selected by minimizing the cross-validation errors.

`coeff.1se` A $(p+1)$ vector of estimated coefficients including the intercept. The corresponding $\lambda$ is the largest $\lambda$ such that the cross-validation error is within 1 standard error of the minimum.

`lambdaSeq` The sequence of regularization parameter candidates for cross-validation.

`lambda.min` Regularization parameter selected by minimizing the cross-validation errors. This is the corresponding $\lambda$ of `coeff.min`.

`lambda.1se` The largest regularization parameter such that the cross-validation error is within 1 standard error of the minimum. This is the corresponding $\lambda$ of `coeff.1se`.

`deviance` Cross-validation errors based on the quantile loss. The length is equal to the length of `lambdaSeq`.

`deviance.se` Estimated standard errors of `deviance`. The length is equal to the length of `lambdaSeq`.

`bandwidth` Bandwidth value.

`tau` Quantile level.

`kernel` Kernel function.

`penalty` Penalty type.

`n` Sample size.

`p` Number of covariates.

## References

Belloni, A. and Chernozhukov, V. (2011). $\ell_1$ penalized quantile regression in high-dimensional sparse models. Ann. Statist., 39, 82-130.

Fan, J. and Li, R. (2001). Variable selection via nonconcave regularized likelihood and its oracle properties. J. Amer. Statist. Assoc., 96, 1348-1360.

Fan, J., Liu, H., Sun, Q. and Zhang, T. (2018). I-LAMM for sparse learning: Simultaneous control of algorithmic complexity and statistical error. Ann. Statist., 46, 814-841.

Koenker, R. and Bassett, G. (1978). Regression quantiles. Econometrica, 46, 33-50.

Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2013). A sparse-group lasso. J. Comp. Graph. Statist., 22, 231-245.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. J. R. Statist. Soc. Ser. B, 58, 267–288.

Tan, K. M., Wang, L. and Zhou, W.-X. (2022). High-dimensional quantile regression: convolution smoothing and concave regularization. J. Roy. Statist. Soc. Ser. B, 84, 205-233.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables., J. Roy. Statist. Soc. Ser. B, 68, 49-67.

Zhang, C.-H. (2010). Nearly unbiased variable selection under minimax concave penalty. Ann. Statist., 38, 894-942.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. J. R. Statist. Soc. Ser. B, 67, 301-320.

## See Also

See `conquer.reg` for regularized quantile regression with a prescribed $lambda$.

## Examples

```
n = 100; p = 200; s = 5
beta = c(rep(1.5, s), rep(0, p - s))
X = matrix(rnorm(n * p), n, p)
Y = X %*% beta + rt(n, 2)

## Cross-validated regularized conquer with lasso penalty at tau = 0.7
fit.lasso = conquer.cv.reg(X, Y, tau = 0.7, penalty = "lasso")
beta.lasso = fit.lasso$coeff.min

## Cross-validated regularized conquer with elastic-net penalty at tau = 0.7
fit.elastic = conquer.cv.reg(X, Y, tau = 0.7, penalty = "elastic", para.elastic = 0.7)
beta.elastic = fit.elastic$coeff.min

## Cross-validated regularized conquer with scad penalty at tau = 0.7
fit.scad = conquer.cv.reg(X, Y, tau = 0.7, penalty = "scad")
beta.scad = fit.scad$coeff.min

## Regularized conquer with group lasso at tau = 0.7
beta = c(rep(1.3, 2), rep(1.5, 3), rep(0, p - s))
err = rt(n, 2)
```

```
Y = X %*% beta + err
group = c(rep(1, 2), rep(2, 3), rep(3, p - s))
fit.group = conquer.cv.reg(X, Y,tau = 0.7, penalty = "group", group = group)
beta.group = fit.group$coeff.min
```

---

`conquer.process` *Convolution-Type Smoothed Quantile Regression Process*

---

## Description

Fit a smoothed quantile regression process over a quantile range. The algorithm is essentially the same as `conquer`.

## Usage

```
conquer.process(
  X,
  Y,
  tauSeq = seq(0.1, 0.9, by = 0.05),
  kernel = c("Gaussian", "logistic", "uniform", "parabolic", "triangular"),
  h = 0,
  checkSing = FALSE,
  tol = 1e-04,
  iteMax = 5000,
  stepBounded = TRUE,
  stepMax = 100
)
```

## Arguments

| | |
|---|---|
| `X` | An $n$ by $p$ design matrix. Each row is a vector of observations with $p$ covariates. Number of observations $n$ must be greater than number of covariates $p$. |
| `Y` | An $n$-dimensional response vector. |
| `tauSeq` | (**optional**) A sequence of quantile values (between 0 and 1). Default is $\{0.1, 0.15, 0.2, ..., 0.85, 0.9\}$. |
| `kernel` | (**optional**) A character string specifying the choice of kernel function. Default is "Gaussian". Choices are "Gaussian", "logistic", "uniform", "parabolic" and "triangular". |
| `h` | (**optional**) The bandwidth/smoothing parameter. Default is $\max\{((log(n) + p)/n)^{0.4}, 0.05\}$. The default will be used if the input value is less than or equal to 0. |
| `checkSing` | (**optional**) A logical flag. Default is FALSE. If `checkSing = TRUE`, then it will check if the design matrix is singular before running conquer. |
| `tol` | (**optional**) Tolerance level of the gradient descent algorithm. The iteration will stop when the maximum magnitude of all the elements of the gradient is less than `tol`. Default is 1e-04. |

| | |
|---|---|
| `iteMax` | (**optional**) Maximum number of iterations. Default is 5000. |
| `stepBounded` | (**optional**) A logical flag. Default is TRUE. If `stepBounded = TRUE`, then the step size of gradient descent is upper bounded by `stepMax`. If `stepBounded = FALSE`, then the step size is unbounded. |
| `stepMax` | (**optional**) Maximum bound for the gradient descent step size. Default is 100. |

## Value

An object containing the following items will be returned:

`coeff` A $(p+1)$ by $m$ matrix of estimated quantile regression process coefficients, including the intercept. m is the length of `tauSeq`.

`bandwidth` Bandwidth value.

`tauSeq` The sequence of quantile levels.

`kernel` The choice of kernel function.

`n` Sample size.

`p` Number the covariates.

## References

Barzilai, J. and Borwein, J. M. (1988). Two-point step size gradient methods. IMA J. Numer. Anal., 8, 141–148.

Fernandes, M., Guerre, E. and Horta, E. (2021). Smoothing quantile regressions. J. Bus. Econ. Statist., 39, 338-357.

He, X., Pan, X., Tan, K. M., and Zhou, W.-X. (2022+). Smoothed quantile regression for large-scale inference. J. Econometrics, in press.

Koenker, R. and Bassett, G. (1978). Regression quantiles. Econometrica, 46, 33-50.

## See Also

See `conquer` for single-index smoothed quantile regression.

## Examples

```
n = 500; p = 10
beta = rep(1, p)
X = matrix(rnorm(n * p), n, p)
Y = X %*% beta + rt(n, 2)

## Smoothed quantile regression process with Gaussian kernel
fit.Gauss = conquer.process(X, Y, tauSeq = seq(0.2, 0.8, by = 0.05), kernel = "Gaussian")
beta.hat.Gauss = fit.Gauss$coeff

## Smoothe quantile regression with uniform kernel
fit.unif = conquer.process(X, Y, tauSeq = seq(0.2, 0.8, by = 0.05), kernel = "uniform")
beta.hat.unif = fit.unif$coeff
```

conquer.reg *Penalized Convolution-Type Smoothed Quantile Regression*

## Description

Fit sparse quantile regression models in high dimensions via regularized conquer methods with "lasso", "elastic-net", "group lasso", "sparse group lasso", "scad" and "mcp" penalties. For "scad" and "mcp", the iteratively reweighted $\ell_1$-penalized algorithm is complemented with a local adpative majorize-minimize algorithm.

## Usage

```
conquer.reg(
  X,
  Y,
  lambda = 0.2,
  tau = 0.5,
  kernel = c("Gaussian", "logistic", "uniform", "parabolic", "triangular"),
  h = 0,
  penalty = c("lasso", "elastic", "group", "sparse-group", "scad", "mcp"),
  para.elastic = 0.5,
  group = NULL,
  weights = NULL,
  para.scad = 3.7,
  para.mcp = 3,
  epsilon = 0.001,
  iteMax = 500,
  phi0 = 0.01,
  gamma = 1.2,
  iteTight = 3
)
```

## Arguments

| | |
|---|---|
| X | An $n$ by $p$ design matrix. Each row is a vector of observations with $p$ covariates. |
| Y | An $n$-dimensional response vector. |
| lambda | (**optional**) Regularization parameter. Can be a scalar or a sequence. If the input is a sequence, the function will sort it in ascending order, and run the regression accordingly. Default is 0.2. |
| tau | (**optional**) Quantile level (between 0 and 1). Default is 0.5. |
| kernel | (**optional**) A character string specifying the choice of kernel function. Default is "Gaussian". Choices are "Gaussian", "logistic", "uniform", "parabolic" and "triangular". |
| h | (**optional**) Bandwidth/smoothing parameter. Default is $\max\{0.5*(log(p)/n)^{0.25}, 0.05\}$. The default will be used if the input value is less than or equal to 0. |

`penalty` (**optional**) A character string specifying the penalty. Default is "lasso" (Tibshirani, 1996). The other options are "elastic" for elastic-net (Zou and Hastie, 2005), "group" for group lasso (Yuan and Lin, 2006), "sparse-group" for sparse group lasso (Simon et al., 2013), "scad" (Fan and Li, 2001) and "mcp" (Zhang, 2010).

`para.elastic` (**optional**) The mixing parameter between 0 and 1 (usually noted as $\alpha$) for elastic-net. The penalty is defined as $\alpha||\beta||_1 + (1-\alpha)||\beta||_2^2$. Default is 0.5. Setting `para.elastic = 1` gives the lasso penalty, and setting `para.elastic = 0` yields the ridge penalty. Only specify it when `penalty = "elastic"`.

`group` (**optional**) A $p$-dimensional vector specifying group indices. Only specify it if `penalty = "group"` or `penalty = "sparse-group"`. For example, if $p = 10$, and we assume the first 3 coefficients belong to the first group, and the last 7 coefficients belong to the second group, then the argument should be `group = c(rep(1, 3), rep(2, 7))`. If not specified, then the penalty will be the classical lasso.

`weights` (**optional**) A vector specifying groups weights for group Lasso and sparse group Lasso. The length must be equal to the number of groups. If not specified, the default weights are square roots of group sizes. For example , if `group = c(rep(1, 3), rep(2, 7))`, then the default weights are $\sqrt{3}$ for the first group, and $\sqrt{7}$ for the second group.

`para.scad` (**optional**) The constant parameter for "scad". Default value is 3.7. Only specify it if `penalty = "scad"`.

`para.mcp` (**optional**) The constant parameter for "mcp". Default value is 3. Only specify it if `penalty = "mcp"`.

`epsilon` (**optional**) A tolerance level for the stopping rule. The iteration will stop when the maximum magnitude of the change of coefficient updates is less than `epsilon`. Default is 0.001.

`iteMax` (**optional**) Maximum number of iterations. Default is 500.

`phi0` (**optional**) The initial quadratic coefficient parameter in the local adaptive majorize-minimize algorithm. Default is 0.01.

`gamma` (**optional**) The adaptive search parameter (greater than 1) in the local adaptive majorize-minimize algorithm. Default is 1.2.

`iteTight` (**optional**) Maximum number of tightening iterations in the iteratively reweighted $\ell_1$-penalized algorithm. Only specify it if the penalty is scad or mcp. Default is 3.

## Value

An object containing the following items will be returned:

`coeff` If the input `lambda` is a scalar, then `coeff` returns a $(p+1)$ vector of estimated coefficients, including the intercept. If the input `lambda` is a sequence, then `coeff` returns a $(p+1)$ by $nlambda$ matrix, where $nlambda$ refers to the length of `lambda` sequence.

`bandwidth` Bandwidth value.

`tau` Quantile level.

`kernel` Kernel function.

`penalty` Penalty type.

`lambda` Regularization parameter(s).

`n` Sample size.

`p` Number of the covariates.

## References

Belloni, A. and Chernozhukov, V. (2011). $\ell_1$ penalized quantile regression in high-dimensional sparse models. Ann. Statist., 39, 82-130.

Fan, J. and Li, R. (2001). Variable selection via nonconcave regularized likelihood and its oracle properties. J. Amer. Statist. Assoc., 96, 1348-1360.

Fan, J., Liu, H., Sun, Q. and Zhang, T. (2018). I-LAMM for sparse learning: Simultaneous control of algorithmic complexity and statistical error. Ann. Statist., 46, 814-841.

Koenker, R. and Bassett, G. (1978). Regression quantiles. Econometrica, 46, 33-50.

Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2013). A sparse-group lasso. J. Comp. Graph. Statist., 22, 231-245.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. J. R. Statist. Soc. Ser. B, 58, 267–288.

Tan, K. M., Wang, L. and Zhou, W.-X. (2022). High-dimensional quantile regression: convolution smoothing and concave regularization. J. Roy. Statist. Soc. Ser. B, 84, 205-233.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables., J. Roy. Statist. Soc. Ser. B, 68, 49-67.

Zhang, C.-H. (2010). Nearly unbiased variable selection under minimax concave penalty. Ann. Statist., 38, 894-942.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. J. R. Statist. Soc. Ser. B, 67, 301-320.

## See Also

See `conquer.cv.reg` for regularized quantile regression with cross-validation.

## Examples

```
n = 200; p = 500; s = 10
beta = c(rep(1.5, s), rep(0, p - s))
X = matrix(rnorm(n * p), n, p)
Y = X %*% beta + rt(n, 2)

## Regularized conquer with lasso penalty at tau = 0.7
fit.lasso = conquer.reg(X, Y, lambda = 0.05, tau = 0.7, penalty = "lasso")
beta.lasso = fit.lasso$coeff

## Regularized conquer with elastic-net penalty at tau = 0.7
fit.elastic = conquer.reg(X, Y, lambda = 0.1, tau = 0.7, penalty = "elastic", para.elastic = 0.7)
beta.elastic = fit.elastic$coeff
```

```
## Regularized conquer with scad penalty at tau = 0.7
fit.scad = conquer.reg(X, Y, lambda = 0.13, tau = 0.7, penalty = "scad")
beta.scad = fit.scad$coeff

## Regularized conquer with group lasso at tau = 0.7
beta = c(rep(1.3, 5), rep(1.5, 5), rep(0, p - s))
err = rt(n, 2)
Y = X %*% beta + err
group = c(rep(1, 5), rep(2, 5), rep(3, p - s))
fit.group = conquer.reg(X, Y, lambda = 0.05, tau = 0.7, penalty = "group", group = group)
beta.group = fit.group$coeff
```

# Index