

# Package ‘quantgen’

July 3, 2020

**Type** Package

**Title** Tools for generalized quantile modeling

**Version** 1.0.0

**Date** 2020-06-06

**Author** Ryan Tibshirani

**Maintainer** Ryan Tibshirani <ryantibs@stat.cmu.edu>

**Description** Tools for generalized quantile modeling: penalized quantile regression, noncrossing constraints, cross-validation, and ensembles.

**Encoding** UTF-8

**License** GPL-2

**RoxygenNote** 7.1.1

**Imports** Matrix,  
Rglpk

**Suggests** gurobi

## R topics documented:

coef.quantile_ensemble . . . . .	2
coef.quantile_genlasso . . . . .	2
combine_into_array . . . . .	3
cv_quantile_genlasso . . . . .	3
cv_quantile_lasso . . . . .	4
get_diff_mat . . . . .	6
get_lambda_max . . . . .	6
get_lambda_seq . . . . .	7
log_pad . . . . .	7
plot.cv_quantile_genlasso . . . . .	8
predict.cv_quantile_genlasso . . . . .	8
predict.quantile_ensemble . . . . .	9
predict.quantile_genlasso . . . . .	9
predict.quantile_genlasso_grid . . . . .	10
quantile_ensemble . . . . .	11
quantile_extrapolate . . . . .	13
quantile_genlasso . . . . .	15
quantile_genlasso_grid . . . . .	17
quantile_genlasso_objective . . . . .	18

quantile_lasso . . . . .	18
quantile_lasso_grid . . . . .	20
quantile_lasso_objective . . . . .	21
quantile_loss . . . . .	21
refit_quantile_genlasso . . . . .	22
refit_quantile_lasso . . . . .	23
unif_jitter . . . . .	24

## Index 25

---

coef.quantile\_ensemble

*Coef function for quantile\_ensemble object*

---

### Description

Retrieve ensemble coefficients for estimating the conditional quantiles at given tau values.

### Usage

```
## S3 method for class 'quantile_ensemble'
coef(object, ...)
```

### Arguments

object	The quantile_ensemble object.
...	Additional arguments (not used).

---

coef.quantile\_genlasso

*Coef function for quantile\_genlasso object*

---

### Description

Retrieve generalized lasso coefficients for estimating the conditional quantiles at specified tau or lambda values.

### Usage

```
## S3 method for class 'quantile_genlasso'
coef(object, s = NULL, ...)
```

### Arguments

object	The quantile_genlasso object.
s	Vector of integers specifying the tau and lambda values to consider for coefficients; for each i in this vector, coefficients are returned at quantile level tau[i] and tuning parameter value lambda[i], according to the tau and lambda vectors stored in the given quantile_genlasso object obj. (Said differently, s specifies the columns of obj\$beta to retrieve for the coefficients.) Default is NULL, which means that all tau and lambda values will be considered.
...	Additional arguments (not used).

---

combine_into_array	<i>Combine matrices into an array</i>
--------------------	---------------------------------------

---

### Description

Combine (say)  $p$  matrices, each of dimension  $n \times r$ , into an  $n \times p \times r$  array.

### Usage

```
combine_into_array(mat, ...)
```

### Arguments

mat	First matrix to combine into an array. Alternatively, a list of matrices to combine into an array.
...	Additional matrices to combine into an array. These additional arguments will be ignored if mat is a list.

---



---

cv_quantile_genlasso	<i>Cross-validation for quantile generalized lasso</i>
----------------------	--------------------------------------------------------

---

### Description

Run cross-validation for the quantile generalized lasso on a tau by lambda grid. For each tau, the lambda value minimizing the cross-validation error is reported.

### Usage

```
cv_quantile_genlasso(
  x,
  y,
  d,
  tau,
  lambda = NULL,
  nlambda = 30,
  lambda_min_ratio = 0.001,
  weights = NULL,
  nfolds = 5,
  train_test_inds = NULL,
  intercept = TRUE,
  standardize = TRUE,
  lp_solver = c("glpk", "gurobi"),
  time_limit = NULL,
  warm_starts = TRUE,
  params = list(),
  transform = NULL,
  inv_trans = NULL,
  jitter = NULL,
  verbose = FALSE,
```

```

    sort = FALSE,
    iso = FALSE,
    nonneg = FALSE,
    round = FALSE
  )

```

## Arguments

**nfolds**                Number of cross-validation folds. Default is 5.

**train\_test\_inds**      List of length two, with components named `train` and `test`. Each of `train` and `test` are themselves lists, of the same length; for each `i`, we will consider `train[[i]]` the indices (which index the rows of `x` and elements of `y`) to use for training, and `test[[i]]` as the indices to use for testing (validation). The validation error will then be summed up over all `i`. This allows for fine control of the "cross-validation" process (in quotes, because there need not be any crossing going on here). Default is `NULL`; if specified, takes priority over `nfolds`.

## Details

All arguments through `verbose` (except for `nfolds` and `train_test_inds`) are as in `quantile_genlasso_grid` and `quantile_genlasso`. Past `verbose`, the arguments are as in `predict.quantile_genlasso`, and control what happens with the predictions made on the validation sets.

## Value

A list with the following components:

- `qgl_obj`: a `quantile_genlasso` object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
- `cv_mat`: a matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
- `lambda_min`: a vector of optimum lambda values, one per quantile level
- `tau, lambda`: the sequences of tau and lambda values considered

---

cv_quantile_lasso	<i>Cross-validation for quantile lasso</i>
-------------------	--------------------------------------------

---

## Description

Run cross-validation for the quantile lasso on a tau by lambda grid. For each tau, the lambda value minimizing the cross-validation error is reported.

## Usage

```

cv_quantile_lasso(
  x,
  y,
  tau,
  lambda = NULL,
  nlambda = 30,

```

```

lambda_min_ratio = 0.001,
weights = NULL,
no_pen_vars = c(),
nfolds = 5,
train_test_inds = NULL,
intercept = TRUE,
standardize = TRUE,
lp_solver = c("glpk", "gurobi"),
time_limit = NULL,
warm_starts = TRUE,
params = list(),
transform = NULL,
inv_trans = NULL,
jitter = NULL,
verbose = FALSE,
sort = FALSE,
iso = FALSE,
nonneg = FALSE,
round = FALSE
)

```

### Arguments

**nfolds**                Number of cross-validation folds. Default is 5.

**train\_test\_inds**      List of length two, with components named `train` and `test`. Each of `train` and `test` are themselves lists, of the same length; for each `i`, we will consider `train[[i]]` the indices (which index the rows of `x` and elements of `y`) to use for training, and `test[[i]]` as the indices to use for testing (validation). The validation error will then be summed up over all `i`. This allows for fine control of the "cross-validation" process (in quotes, because there need not be any crossing going on here). Default is `NULL`; if specified, takes priority over `nfolds`.

### Details

All arguments through `verbose` (except for `nfolds` and `train_test_inds`) are as in `quantile_lasso_grid` and `quantile_lasso`. Past `verbose`, the arguments are as in `predict.quantile_lasso`, and control what happens with the predictions made on the validation sets.

### Value

A list with the following components:

- `qgl_obj`: a `quantile_genlasso` object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
- `cv_mat`: a matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
- `lambda_min`: a vector of optimum lambda values, one per quantile level

---

get_diff_mat	<i>Difference matrix</i>
--------------	--------------------------

---

### Description

Construct a difference operator, of a given order, for use in trend filtering penalties.

### Usage

```
get_diff_mat(p, k)
```

### Arguments

p	Dimension (number of columns) of the difference matrix.
k	Order of the difference matrix.

### Value

A sparse matrix of dimension  $(p - k) \times p$ .

---

get_lambda_max	<i>Lambda max for quantile generalized lasso</i>
----------------	--------------------------------------------------

---

### Description

Compute lambda max for a quantile generalized lasso problem.

### Usage

```
get_lambda_max(x, y, d, weights = NULL, lp_solver = c("glpk", "gurobi"))
```

### Details

This is not exact, but should be close to the exact value of  $\lambda$  such that  $D\hat{\beta} = 0$  at the solution  $\hat{\beta}$  of the quantile generalized lasso problem. It is derived from the KKT conditions when  $\tau = 1/2$ .

get\_lambda\_seq

*Lambda sequence for quantile generalized lasso***Description**

Compute a lambda sequence for a quantile generalized lasso problem.

**Usage**

```
get_lambda_seq(
  x,
  y,
  d,
  nlambda,
  lambda_min_ratio,
  weights = NULL,
  intercept = TRUE,
  standardize = TRUE,
  lp_solver = c("glpk", "gurobi"),
  transform = NULL
)
```

**Details**

This function returns `nlambda` values log-spaced in between `lambda_max`, as computed by `get_lambda_max`, and `lambda_max * lambda_min_ratio`. If `d` is not specified, we will set it equal to the identity (hence interpret the problem as a quantile lasso problem).

log\_pad

*Convenience functions for log padding***Description**

Functions to map  $y \mapsto \log(a + y)$  and  $x \mapsto \exp(x) - a$ .

**Usage**

```
log_pad(a = 1)
inv_log_pad(a = 1)
```

---

```
plot.cv_quantile_genlasso
```

*Plot function for quantile\_genlasso object*

---

### Description

Plot the cross-validation error curves, for each quantile level, as functions of the tuning parameter value.

### Usage

```
## S3 method for class 'cv_quantile_genlasso'
plot(x, legend_pos = "topleft", ...)
```

### Arguments

x	The cv_quantile_genlasso object.
legend_pos	Position for the legend; default is "topleft"; use NULL to suppress the legend.
...	Additional arguments (not used).

---

```
predict.cv_quantile_genlasso
```

*Predict function for cv\_quantile\_genlasso object*

---

### Description

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients tuned by cross-validation.

### Usage

```
## S3 method for class 'cv_quantile_genlasso'
predict(
  object,
  newx,
  s = NULL,
  sort = FALSE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

### Details

This just calls the predict function on the quantile\_genlasso that is stored within the given cv\_quantile\_genlasso object.



---

predict.quantile\_ensemble

*Predict function for quantile\_ensemble object*


---

### Description

Predict the conditional quantiles at a new set of ensemble realizations, using the ensemble coefficients at given tau values.

### Usage

```
## S3 method for class 'quantile_ensemble'
predict(
  object,
  newq,
  s = NULL,
  sort = TRUE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

### Arguments

object	The quantile_ensemble object.
newq	Array of new predicted quantiles, of dimension (number of new prediction points) x (number of ensemble components) x (number of quantile levels).
sort	Should the returned quantile estimates be sorted? Default is TRUE.
iso	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over sort.
nonneg	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
round	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.
...	Additional arguments (not used).

---

predict.quantile\_genlasso

*Predict function for quantile\_genlasso object*


---

### Description

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients at specified tau or lambda values.

**Usage**

```
## S3 method for class 'quantile_genlasso'
predict(
  object,
  newx,
  s = NULL,
  sort = FALSE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

**Arguments**

object	The quantile_genlasso object.
newx	Matrix of new predictor variables at which predictions should be made.
s	Vector of integers specifying the tau and lambda values to consider for predictions; for each i in this vector, predictions are made at quantile level tau[i] and tuning parameter value lambda[i], according to the tau and lambda vectors stored in the given quantile_genlasso object obj. (Said differently, s specifies the columns of object\$beta to use for the predictions.) Default is NULL, which means that all tau and lambda values will be considered.
sort	Should the returned quantile estimates be sorted? Default is FALSE. Note: this option only makes sense if the values in the stored tau vector are distinct, and sorted in increasing order.
iso	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over sort. Note: this option only makes sense if the values in the stored tau vector are distinct, and sorted in increasing order.
nonneg	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
round	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.
...	Additional arguments (not used).

---

predict.quantile\_genlasso\_grid

*Predict function for quantile\_genlasso\_grid object*

---

**Description**

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients at given tau or lambda values.

**Usage**

```
## S3 method for class 'quantile_genlasso_grid'
predict(
  object,
  newx,
  sort = FALSE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

**Details**

This function operates as in the `predict.quantile_genlasso` function for a `quantile_genlasso` object, but with a few key differences. First, the output is reformatted so that it is an array of dimension (number of prediction points) x (number of tuning parameter values) x (number of quantile levels). This output is generated from the full set of tau and lambda pairs stored in the given `quantile_genlasso_grid` object `obj` (selecting a subset is disallowed). Second, the arguments `sort` and `iso` operate on the appropriate slices of this array: for a fixed lambda value, we sort or run isotonic regression across all tau values.

---

quantile_ensemble	<i>Quantile ensemble</i>
-------------------	--------------------------

---

**Description**

Fit ensemble weights, given a set of quantile predictions.

**Usage**

```
quantile_ensemble(
  qarr,
  y,
  tau,
  weights = NULL,
  tau_groups = rep(1, length(tau)),
  intercept = FALSE,
  nonneg = TRUE,
  unit_sum = TRUE,
  noncross = TRUE,
  q0 = NULL,
  lp_solver = c("glpk", "gurobi"),
  time_limit = NULL,
  params = list(),
  verbose = FALSE
)
```

**Arguments**

qarr	Array of predicted quantiles, of dimension (number of prediction points) x (number or ensemble components) x (number of quantile levels).
y	Vector of responses (whose quantiles are being predicted by qarr).
tau	Vector of quantile levels at which predictions are made. Assumed to be distinct, and sorted in increasing order.
weights	Vector of observation weights (to be used in the loss function). Default is NULL, which is interpreted as a weight of 1 for each observation.
tau_groups	Vector of group labels, having the same length as tau. Common labels indicate that the ensemble weights for the corresponding quantile levels should be tied together. Default is rep(1, length(tau)), which means that a common set of ensemble weights should be used across all levels. See details.
intercept	Should an intercept be included in the ensemble model? Default is FALSE.
nonneg	Should the ensemble weights be constrained to be nonnegative? Default is TRUE.
unit_sum	Should the ensemble weights be constrained to sum to 1? Default is TRUE.
noncross	Should noncrossing constraints be enforced? Default is TRUE. Note: this option only matters when there is more than group of ensemble weights, as determined by tau_groups. See details.
q0	Array of points used to define the noncrossing constraints. Must have dimension (number of points) x (number of ensemble components) x (number of quantile levels). Default is NULL, which means that we consider noncrossing constraints at the training points qarr.
lp_solver	One of "glpk" or "gurobi", indicating which LP solver to use. If possible, "gurobi" should be used because it is much faster and more stable; default is "glpk"; however, because it is open-source.
time_limit	This sets the maximum amount of time (in seconds) to allow Gurobi or GLPK to solve any single quantile generalized lasso problem (for a single tau and lambda value). Default is NULL, which means unlimited time.
params	A list of control parameters to pass to Gurobi or GLPK. Default is list() which means no additional parameters are passed. For example: with Gurobi, we can use list(Threads=4) to specify that Gurobi should use 4 threads when available. (Note that if a time limit is specified through this params list, then its value will be overridden by the last argument time_limit, assuming the latter is not NULL.)
verbose	Should progress be printed out to the console? Default is FALSE.

**Details**

This function solves the following quantile ensemble optimization problem, over quantile levels  $\tau_k, k = 1, \dots, r$ :

$$\begin{aligned}
 & \underset{\alpha_j, j=1, \dots, p}{\text{minimize}} && \sum_{k=1}^r \sum_{i=1}^n w_i \psi_{\tau_k} \left( y_i - \sum_{j=1}^p \alpha_j q_{ijk} \right) \\
 & \text{subject to} && \sum_{j=1}^p \alpha_j = 1, \alpha_j \geq 0, j = 1, \dots, p
 \end{aligned}$$

for a response vector  $y$  and quantile array  $q$ , where  $q_{ijk}$  is an estimate of the quantile of  $y_i$  at the level  $\tau_k$ , from ensemble component member  $j$ . Here  $\psi_{\tau}(v) = \max\{\tau v, (\tau - 1)v\}$  is the "pinball"

or "tilted  $\ell_1$ " loss. A more advanced version allows us to estimate a separate ensemble weight  $\alpha_{jk}$  per component method  $j$ , per quantile level  $k$ :

$$\begin{aligned} & \text{minimize } \alpha_{jk}, j = 1, \dots, p, k = 1, \dots, r \sum_{k=1}^r \sum_{i=1}^n w_i \psi_{\tau_k} \left( y_i - \sum_{j=1}^p \alpha_{jk} q_{ijk} \right) \\ & \text{subject to } \sum_{j=1}^p \alpha_{jk} = 1, k = 1, \dots, r, \alpha_{jk} \geq 0, j = 1, \dots, p, k = 1, \dots, r \end{aligned}$$

As a form of regularization, we can additionally incorporate noncrossing constraints into the above optimization, which take the form:

$$\alpha_{\bullet, k}^T q \leq \alpha_{\bullet, k+1}^T q, \quad k = 1, \dots, r-1, \quad q \in \mathcal{Q}$$

where the quantile levels  $\tau_k, k = 1, \dots, r$  are assumed to be in increasing order, and  $\mathcal{Q}$  is a collection of points over which to enforce the noncrossing constraints. Finally, somewhere in between these two extremes is to allow one ensemble weight per component member  $j$ , per quantile group  $g$ . This can be interpreted as a set of further constraints which enforce equality between  $\alpha_{jk}$  and  $\alpha_{j\ell}$ , for all  $k, \ell$  that are in the same group  $g$ .

### Value

A list with the following components:

- alpha: vector or matrix of ensemble weights. If tau\_groups has only one unique label, then this is a vector of length = (number of ensemble components); otherwise, it is a matrix, of dimension (number of ensemble components) x (number of quantile levels)
- tau: vector of quantile levels used
- weights, tau\_groups, ..., params: values of these other arguments used in the function call

---

quantile\_extrapolate    *Quantile extrapolater*

---

### Description

Extrapolate a set of quantiles at new quantile levels: parametric in the tails, nonparametric in the middle.

### Usage

```
quantile_extrapolate(
  tau,
  qvals,
  tau_out = c(0.01, 0.025, seq(0.05, 0.95, by = 0.05), 0.975, 0.99),
  sort = TRUE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  qfun_left = qnorm,
  qfun_right = qnorm,
  n_tau_left = 1,
```

```

    n_tau_right = 1,
    middle = c("cubic", "linear"),
    param0 = NULL,
    param1 = NULL,
    grid_size = 1000,
    tol = 0.01,
    max_iter = 10
)

```

### Arguments

<code>tau</code>	Vector of quantile levels. Assumed to be distinct, and sorted in increasing order.
<code>qvals</code>	Vector or matrix quantiles; if a matrix, each row is a separate set of quantiles, at the same (common) quantile levels, given by <code>tau</code> .
<code>tau_out</code>	Vector of quantile levels at which to perform extrapolation. Default is a sequence of 23 quantile levels from 0.01 to 0.99.
<code>sort</code>	Should the returned quantile estimates be sorted? Default is TRUE.
<code>iso</code>	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over <code>sort</code> .
<code>nonneg</code>	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
<code>round</code>	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.
<code>qfun_left, qfun_right</code>	Quantile functions on which to base extrapolation in the left and right tails, respectively; each must be a function whose first two arguments are a quantile level and a distribution parameter (such as a mean parameter); these are assumed to be vectorized in the first argument when the second argument is fixed, and also vectorized in the second argument when the first argument is fixed. Default is <code>qnorm</code> . See details for further explanation.
<code>n_tau_left, n_tau_right</code>	Integers between 1 and the length of <code>tau</code> , indicating how many elements quantile levels from the left and right ends, respectively, to use in defining the tails. For example, if <code>n_tau_left=1</code> , the default, then only the leftmost quantile is used for the left tail extrapolation; if <code>n_tau_left=2</code> , then the two leftmost quantiles are used, etc; and similarly for <code>n_tau_right</code> . See details for further explanation.
<code>middle</code>	One of "cubic" or "linear", indicating the interpolation method to use in the middle (outside of the tails, as determined by <code>n_tau_left</code> and <code>n_tau_right</code> ). If "cubic", the default, then a monotone cubic spline interpolant is fit to the given quantiles, and used to estimate quantiles in the middle. If "linear", then linear interpolation is used to estimate quantiles in the middle.
<code>param0, param1</code>	TODO
<code>grid_size, tol, maxiter</code>	TODO

### Details

TODO

**Value**

A matrix of dimension (number of rows in `qvals`) x (length of `tau_out`), where each row is the extrapolation of the set of quantiles in the corresponding row of `qvals`, at the quantile levels specified in `tau_out`.

---

quantile_genlasso	<i>Quantile generalized lasso</i>
-------------------	-----------------------------------

---

**Description**

Compute quantile generalized lasso solutions.

**Usage**

```
quantile_genlasso(
  x,
  y,
  d,
  tau,
  lambda,
  weights = NULL,
  intercept = TRUE,
  standardize = TRUE,
  noncross = FALSE,
  x0 = NULL,
  lp_solver = c("glpk", "gurobi"),
  time_limit = NULL,
  warm_starts = TRUE,
  params = list(),
  transform = NULL,
  inv_trans = NULL,
  jitter = NULL,
  verbose = FALSE
)
```

**Arguments**

<code>x</code>	Matrix of predictors. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization.
<code>y</code>	Vector of responses.
<code>d</code>	Matrix defining the generalized lasso penalty; see details. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization. A convenience function <code>get_diff_mat</code> for constructing trend filtering penalties is provided.
<code>tau, lambda</code>	Vectors of quantile levels and tuning parameter values. If these are not of the same length, the shorter of the two is recycled so that they become the same length. Then, for each <code>i</code> , we solve a separate quantile generalized lasso problem at quantile level <code>tau[i]</code> and tuning parameter value <code>lambda[i]</code> . The most common use cases are: specifying one <code>tau</code> value and a sequence of <code>lambda</code> values; or specifying a sequence of <code>tau</code> values and one <code>lambda</code> value.

weights	Vector of observation weights (to be used in the loss function). Default is NULL, which is interpreted as a weight of 1 for each observation.
intercept	Should an intercept be included in the regression model? Default is TRUE.
standardize	Should the predictors be standardized (to have zero mean and unit variance) before fitting? Default is TRUE.
noncross	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is FALSE. If TRUE, then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument <code>x0</code> . Note: this option only makes sense if the values in the <code>tau</code> vector are distinct, and sorted in increasing order.
<code>x0</code>	Matrix of points used to define the noncrossing constraints. Default is NULL, which means that we consider noncrossing constraints at the training points <code>x</code> .
lp_solver	One of "glpk" or "gurobi", indicating which LP solver to use. If possible, "gurobi" should be used because it is much faster and more stable; default is "glpk"; however, because it is open-source.
time_limit	This sets the maximum amount of time (in seconds) to allow Gurobi or GLPK to solve any single quantile generalized lasso problem (for a single <code>tau</code> and <code>lambda</code> value). Default is NULL, which means unlimited time.
warm_starts	Should warm starts be used in the LP solver (from one LP solve to the next)? Only supported for Gurobi.
params	A list of control parameters to pass to Gurobi or GLPK. Default is <code>list()</code> which means no additional parameters are passed. For example: with Gurobi, we can use <code>list(Threads=4)</code> to specify that Gurobi should use 4 threads when available. (Note that if a time limit is specified through this <code>params</code> list, then its value will be overridden by the last argument <code>time_limit</code> , assuming the latter is not NULL.)
transform, inv_trans	The first is a function to transform <code>y</code> before solving the quantile generalized lasso; the second is the corresponding inverse transform. For example: for count data, we might want to model $\log(1+y)$ (which would be the transform, and the inverse transform would be $\exp(x)-1$ ). Both <code>transform</code> and <code>inv_trans</code> should be vectorized. Convenience functions <code>log_pad</code> and <code>inv_log_pad</code> are provided.
jitter	Function for applying random jitter to <code>y</code> , which might help optimization. For example: for count data, there can be lots of ties (with or without transformation of <code>y</code> ), which can make optimization more difficult. The function <code>jitter</code> should take an integer <code>n</code> and return <code>n</code> random draws. A convenience function <code>unif_jitter</code> is provided.
verbose	Should progress be printed out to the console? Default is FALSE.

## Details

This function solves the quantile generalized lasso problem, for each pair of quantile level  $\tau$  and tuning parameter  $\lambda$ :

$$\underset{\beta_0, \beta}{\text{minimize}} \sum_{i=1}^n w_i \psi_{\tau}(y_i - \beta_0 - x_i^T \beta) + \lambda \|D\beta\|_1$$

for a response vector  $y$  with components  $y_i$ , predictor matrix  $X$  with rows  $x_i$ , and penalty matrix  $D$ . Here  $\psi_{\tau}(v) = \max\{\tau v, (\tau - 1)v\}$  is the "pinball" or "tilted  $\ell_1$ " loss. When noncrossing



constraints are applied, we instead solve one big joint optimization, over all quantile levels and tuning parameter values:

$$\underset{\beta_{0k}, \beta_k, k=1, \dots, r}{\text{minimize}} \sum_{k=1}^r \left( \sum_{i=1}^n w_i \psi_{\tau_k}(y_i - \beta_{0k} - x_i^T \beta_k) + \lambda_k \|D\beta_k\|_1 \right)$$

$$\text{subject to } \beta_{0k} + x^T \beta_k \leq \beta_{0,k+1} + x^T \beta_{k+1} \quad k = 1, \dots, r-1, \quad x \in \mathcal{X}$$

where the quantile levels  $\tau_k, k = 1, \dots, r$  are assumed to be in increasing order, and  $\mathcal{X}$  is a collection of points over which to enforce the noncrossing constraints.

Either problem is readily converted into a linear program (LP), and solved using either Gurobi (which is free for academic use, and generally fast) or GLPK (which free for everyone, but slower).

### Value

A list with the following components:

- beta: a matrix of generalized lasso coefficients, of dimension = (number of features + 1) x (number of quantile levels) assuming intercept=TRUE, else (number of features) x (number of quantile levels). Note: these coefficients will always be on the appropriate scale; they are always on the scale of original features, even if standardize=TRUE
- status: vector of status flags returned by Gurobi's or GLPK's LP solver, of length = (number of quantile levels)
- tau, lambda: vectors of tau and lambda values used
- weights, intercept, ..., jitter: values of these other arguments used in the function call

### Author(s)

Ryan Tibshirani

---

quantile\_genlasso\_grid

*Quantile generalized lasso on a tau by lambda grid*

---

### Description

Convenience function for computing quantile generalized lasso solutions on a tau by lambda grid.

### Usage

```
quantile_genlasso_grid(
  x,
  y,
  d,
  tau,
  lambda = NULL,
  nlambda = 30,
  lambda_min_ratio = 0.001,
  weights = NULL,
  intercept = TRUE,
  standardize = TRUE,
```

```

lp_solver = c("glpk", "gurobi"),
time_limit = NULL,
warm_starts = TRUE,
params = list(),
transform = NULL,
inv_trans = NULL,
jitter = NULL,
verbose = FALSE
)

```

### Arguments

**nlambda**                Number of lambda values to consider, for each quantile level. Default is 30.

**lambda\_min\_ratio**        Ratio of the minimum to maximum lambda value, for each quantile levels. Default is 1e-3.

### Details

This function forms a lambda vector either determined by the `nlambda` and `lambda_min_ratio` arguments, or the `lambda` argument; if the latter is specified, then it takes priority. Then, for each `i` and `j`, we solve a separate quantile generalized lasso problem at quantile level `tau[i]` and tuning parameter value `lambda[j]`, using the `quantile_genlasso` function. All arguments (aside from `nlambda` and `lambda_min_ratio`) are as in the latter function; noncrossing constraints are disallowed.

---

<code>quantile_genlasso_objective</code>	<i>Quantile generalized lasso objective</i>
------------------------------------------	---------------------------------------------

---

### Description

Compute generalized lasso objective for a single tau and lambda value.

### Usage

```
quantile_genlasso_objective(x, y, d, beta, tau, lambda)
```

---

<code>quantile_lasso</code>	<i>Quantile lasso.</i>
-----------------------------	------------------------

---

### Description

Compute quantile lasso solutions.

**Usage**

```
quantile_lasso(
  x,
  y,
  tau,
  lambda,
  weights = NULL,
  no_pen_vars = c(),
  intercept = TRUE,
  standardize = TRUE,
  noncross = FALSE,
  x0 = NULL,
  lp_solver = c("glpk", "gurobi"),
  time_limit = NULL,
  warm_starts = TRUE,
  params = list(),
  transform = NULL,
  inv_trans = NULL,
  jitter = NULL,
  verbose = FALSE
)
```

**Arguments**

<code>x</code>	Matrix of predictors. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization.
<code>y</code>	Vector of responses.
<code>tau, lambda</code>	Vectors of quantile levels and tuning parameter values. If these are not of the same length, the shorter of the two is recycled so that they become the same length. Then, for each $i$ , we solve a separate quantile lasso problem at quantile level $\tau[i]$ and tuning parameter value $\lambda[i]$ . The most common use cases are: specifying one $\tau$ value and a sequence of $\lambda$ values; or specifying a sequence of $\tau$ values and one $\lambda$ value.
<code>weights</code>	Vector of observation weights (to be used in the loss function). Default is <code>NULL</code> , which is interpreted as a weight of 1 for each observation.
<code>no_pen_vars</code>	Indices of the variables that should be excluded from the lasso penalty. Default is <code>c()</code> , which means that no variables are to be excluded.

**Details**

This function solves the quantile lasso problem, for each pair of quantile level  $\tau$  and tuning parameter  $\lambda$ :

$$\underset{\beta_0, \beta}{\text{minimize}} \sum_{i=1}^n w_i \psi_{\tau}(y_i - \beta_0 - x_i^T \beta) + \lambda \|\beta\|_1$$

for a response vector  $y$  with components  $y_i$ , and predictor matrix  $X$  with rows  $x_i$ . Here  $\psi_{\tau}(v) = \max\{\tau v, (\tau - 1)v\}$  is the "pinball" or "tilted  $\ell_1$ " loss. When noncrossing constraints are applied, we instead solve one big joint optimization, over all quantile levels and tuning parameter values:

$$\underset{\beta_{0k}, \beta_k, k=1, \dots, r}{\text{minimize}} \sum_{k=1}^r \left( \sum_{i=1}^n w_i \psi_{\tau_k}(y_i - \beta_{0k} - x_i^T \beta_k) + \lambda_k \|\beta_k\|_1 \right)$$

subject to  $\beta_{0k} + x^T \beta_k \leq \beta_{0,k+1} + x^T \beta_{k+1} \quad k = 1, \dots, r-1, x \in \mathcal{X}$

where the quantile levels  $\tau_j, j = 1, \dots, k$  are assumed to be in increasing order, and  $\mathcal{X}$  is a collection of points over which to enforce the noncrossing constraints.

Either problem is readily converted into a linear program (LP), and solved using either Gurobi (which is free for academic use, and generally fast) or GLPK (which free for everyone, but slower).

All arguments not described above are as in the `quantile_genlasso` function. The associated `coef` and `predict` functions are just those for the `quantile_genlasso` class.

## Value

A list with the following components:

- `beta`: a matrix of lasso coefficients, of dimension = (number of features + 1) x (number of quantile levels) assuming `intercept=TRUE`, else (number of features) x (number of quantile levels). Note: these coefficients will always be on the appropriate scale; they are always on the scale of original features, even if `standardize=TRUE`
- `status`: vector of status flags returned by Gurobi's or GLPK's LP solver, of length = (number of quantile levels)
- `tau, lambda`: vectors of tau and lambda values used
- `weights, no_pen_vars, ..., jitter`: values of these other arguments used in the function call

## Author(s)

Ryan Tibshirani

---

`quantile_lasso_grid`     *Quantile lasso on a tau by lambda grid*

---

## Description

Convenience function for computing quantile lasso solutions on a tau by lambda grid.

## Usage

```
quantile_lasso_grid(
  x,
  y,
  tau,
  lambda = NULL,
  nlambdas = 30,
  lambda_min_ratio = 0.001,
  weights = NULL,
  no_pen_vars = c(),
  intercept = TRUE,
  standardize = TRUE,
  lp_solver = c("glpk", "gurobi"),
  time_limit = NULL,
  warm_starts = TRUE,
  params = list(),
  transform = NULL,
```

```

    inv_trans = NULL,
    jitter = NULL,
    verbose = FALSE
)

```

### Arguments

`nlambda`            Number of lambda values to consider, for each quantile level. Default is 30.

`lambda_min_ratio`       Ratio of the minimum to maximum lambda value, for each quantile levels. Default is 1e-3.

### Details

This function forms a lambda vector either determined by the `nlambda` and `lambda_min_ratio` arguments, or the `lambda` argument; if the latter is specified, then it takes priority. Then, for each `i` and `j`, we solve a separate quantile lasso problem at quantile level `tau[i]` and tuning parameter value `lambda[j]`, using the `quantile_lasso` function. All arguments (aside from `nlambda` and `lambda_min_ratio`) are as in the latter function; noncrossing constraints are disallowed. The associated predict function is just that for the `quantile_genlasso_grid` class.

---

quantile_lasso_objective	<i>Quantile lasso objective</i>
--------------------------	---------------------------------

---

### Description

Compute lasso objective for a single tau and lambda value.

### Usage

```
quantile_lasso_objective(x, y, beta, tau, lambda)
```

---

quantile_loss	<i>Quantile loss</i>
---------------	----------------------

---

### Description

Compute the quantile (tilted absolute) loss for a single tau value.

### Usage

```
quantile_loss(yhat, y, tau)
```

---

refit\_quantile\_genlasso

*Refit function for cv\_quantile\_genlasso object*


---

## Description

Refit generalized lasso solutions at a new set of quantile levels, given an existing `cv_quantile_genlasso` object.

## Usage

```
refit_quantile_genlasso(
  obj,
  x,
  y,
  d,
  tau_new = c(0.01, 0.025, seq(0.05, 0.95, by = 0.05), 0.975, 0.99),
  weights = NULL,
  intercept = TRUE,
  standardize = TRUE,
  noncross = FALSE,
  x0 = NULL,
  lp_solver = NULL,
  time_limit = NULL,
  warm_starts = NULL,
  params = NULL,
  transform = NULL,
  inv_trans = NULL,
  jitter = NULL,
  verbose = FALSE
)
```

## Arguments

<code>obj</code>	The <code>cv_quantile_genlasso</code> object to start from.
<code>x</code>	Matrix of predictors.
<code>y</code>	Vector of responses.
<code>d</code>	Matrix defining the generalized lasso penalty.
<code>tau_new</code>	Vector of new quantile levels at which to fit new solutions. Default is a sequence of 23 quantile levels from 0.01 to 0.99.
<code>noncross</code>	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is <code>FALSE</code> . If <code>TRUE</code> , then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument <code>x0</code> .
<code>x0</code>	Matrix of points used to define the noncrossing constraints. Default is <code>NULL</code> , which means that we consider noncrossing constraints at the training points <code>x</code> .
<code>verbose</code>	Should progress be printed out to the console? Default is <code>FALSE</code> .

**Details**

This function simply infers, for each quantile level in `tau_new`, a (very) roughly-CV-optimal tuning parameter value, then calls `quantile_genlasso` at the new quantile levels and corresponding tuning parameter values. If not specified, the arguments `weights`, `intercept`, `standardize`, `lp_solver`, `time_limit`, `warm_starts`, `params`, `transform`, `inv_transorm`, `jitter` are all inherited from the given `cv_quantile_genlasso` object.

**Value**

A `quantile_genlasso` object, with solutions at quantile levels `tau_new`.

---

<code>refit_quantile_lasso</code>	<i>Refit function for <code>cv_quantile_lasso</code> object</i>
-----------------------------------	-----------------------------------------------------------------

---

**Description**

Refit lasso solutions at a new set of quantile levels, given an existing `cv_quantile_lasso` object.

**Usage**

```
refit_quantile_lasso(
  obj,
  x,
  y,
  tau_new = c(0.01, 0.025, seq(0.05, 0.95, by = 0.05), 0.975, 0.99),
  weights = NULL,
  no_pen_vars = NULL,
  intercept = TRUE,
  standardize = TRUE,
  noncross = FALSE,
  x0 = NULL,
  lp_solver = NULL,
  time_limit = NULL,
  warm_starts = NULL,
  params = NULL,
  transform = NULL,
  inv_trans = NULL,
  jitter = NULL,
  verbose = FALSE
)
```

**Arguments**

<code>obj</code>	The <code>cv_quantile_lasso</code> object to start from.
<code>x</code>	Matrix of predictors.
<code>y</code>	Vector of responses.
<code>tau_new</code>	Vector of new quantile levels at which to fit new solutions. Default is a sequence of 23 quantile levels from 0.01 to 0.99.

noncross	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is FALSE. If TRUE, then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument <code>x0</code> .
<code>x0</code>	Matrix of points used to define the noncrossing constraints. Default is NULL, which means that we consider noncrossing constraints at the training points <code>x</code> .
verbose	Should progress be printed out to the console? Default is FALSE.

### Details

This function simply infers, for each quantile level in `tau_new`, a (very) roughly-CV-optimal tuning parameter value, then calls `quantile_lasso` at the new quantile levels and corresponding tuning parameter values. If not specified, the arguments `weights`, `no_pen_vars`, `intercept`, `standardize`, `lp_solver`, `time_limit`, `warm_start`, `params`, `transform`, `inv_transorm`, `jitter` are all inherited from the given `cv_quantile_lasso` object.

### Value

A `quantile_lasso` object, with solutions at quantile levels `tau_new`.

---

unif\_jitter

*Convenience function for uniform jitter*


---

### Description

Function to generate random draws from  $\text{Unif}[a, b]$ .

### Usage

```
unif_jitter(a = 0, b = 0.01)
```



# Index

`coef.quantile_ensemble`, [2](#)  
`coef.quantile_genlasso`, [2](#)  
`combine_into_array`, [3](#)  
`cv_quantile_genlasso`, [3](#)  
`cv_quantile_lasso`, [4](#)  
  
`get_diff_mat`, [6](#)  
`get_lambda_max`, [6](#)  
`get_lambda_seq`, [7](#)  
  
`inv_log_pad(log_pad)`, [7](#)  
  
`log_pad`, [7](#)  
  
`plot.cv_quantile_genlasso`, [8](#)  
`predict.cv_quantile_genlasso`, [8](#)  
`predict.quantile_ensemble`, [9](#)  
`predict.quantile_genlasso`, [9](#)  
`predict.quantile_genlasso_grid`, [10](#)  
  
`quantile_ensemble`, [11](#)  
`quantile_extrapolate`, [13](#)  
`quantile_genlasso`, [15](#)  
`quantile_genlasso_grid`, [17](#)  
`quantile_genlasso_objective`, [18](#)  
`quantile_lasso`, [18](#)  
`quantile_lasso_grid`, [20](#)  
`quantile_lasso_objective`, [21](#)  
`quantile_loss`, [21](#)  
  
`refit_quantile_genlasso`, [22](#)  
`refit_quantile_lasso`, [23](#)  
  
`unif_jitter`, [24](#)