# Package 'genlasso'

July 11, 2019

**Type** Package

**Title** Path Algorithm for Generalized Lasso Problems

**Version** 1.4

**Date** 2019-07-11

**Author** Taylor B. Arnold and Ryan J. Tibshirani

**Maintainer** Taylor Arnold <taylor.arnold@acm.org>

**Description** Provides fast algorithms for computing the solution path for generalized lasso problems. Important use cases are the fused lasso over an arbitrary graph, and trend fitting of any given polynomial order. Specialized implementations for the latter two problems are given to improve stability and speed.

**License** GPL (>= 2.0)

**Depends** MASS, Matrix, igraph

**ByteCompile** TRUE

**URL** https://github.com/statsmaths/genlasso

**NeedsCompilation** yes

**Repository** CRAN

**RoxygenNote** 6.1.1

**Date/Publication** 2019-07-11 13:32:48 UTC

## R topics documented:

---

`genlasso-package` *Package to compute the solution path of generalized lasso problems*

---

## Description

This package is centered around computing the solution path of the generalized lasso problem, which minimizes the criterion

$$1/2\|y - X\beta\|_2^2 + \lambda\|D\beta\|_1.$$

The solution path is computed by solving the equivalent Lagrange dual problem. The dimension of the dual variable u is the number of rows of the penalty matrix D, and the primal (original) and dual solutions are related by

$$\hat{\beta} = y - D^T\hat{u}$$

when the predictor matrix X is the identity, and

$$\hat{\beta} = (X^TX)^{-1}(X^Ty - D^T\hat{u})$$

for a full column rank predictor matrix X. For column rank deficient matrices X, the solution path is not unique and not computed by this package. However, one can add a small ridge penalty to the above criterion, which can be re-expressed as a generalized lasso problem with full column rank predictor matrix X and hence yields a unique solution path.

Important use cases include the fused lasso, where D is the oriented incidence matrix of some underlying graph (the orientations being arbitrary), and trend filtering, where D is the discrete difference operator of any given order k.

The general function `genlasso` computes a solution path for any penalty matrix D and full column rank predictor matrix X (adding a ridge penalty when X is rank deficient). For the fused lasso and trend filtering problems, the specialty functions `fusedlasso` and `trendfilter` should be used as they deliver a significant increase in speed and numerical stability.

For a walk-through of using the package for statistical modelling see the included package vignette; for the appropriate background material see the generalized lasso paper referenced below.

## Author(s)

Taylor B. Arnold and Ryan J. Tibshirani

## References

Tibshirani, R. J. and Taylor, J. (2011), "The solution path of the generalized lasso", Annals of Statistics 39 (3) 1335–1371.

Arnold, T. B. and Tibshirani, R. J. (2014), "Efficient implementations of the generalized lasso dual path algorithm", arXiv: 1405.3222.

## See Also

`genlasso`, `fusedlasso`, `trendfilter`

`coef.genlasso` *Extract cefficients from a genlasso object*

## Description

This function extracts coefficients from a generalized lasso solution path object, for any set of tuning parameter values along the path. It can return dual coefficients. The requested coefficients can also be parametrized by degrees of freedom value instead of tuning parameter value.

## Usage

```
## S3 method for class 'genlasso'
coef(object, lambda, nlam, df,
     type = c("primal", "dual", "both"), ...)
```

## Arguments

| | |
|---|---|
| `object` | an object of class "genlasso", or an object which inherits this class (i.e., "fusedlasso", "trendfilter"). |
| `lambda` | a numeric vector of tuning parameter values at which coefficients should be calculated. The user can choose to specify one of `lambda`, `nlam`, or `df`; if none are specified, then coefficients are returned at every knot in the solution path. |
| `nlam` | an integer indicating a number of tuning parameters values at which coefficients should be calculated. The tuning parameter values are then chosen to be equally spaced on the log scale over the first half of the solution path (this is if the full solution path has been computed; if only a partial path has been computed, the tuning parameter values are spaced over the entirety of the computed path). |
| `df` | an integer vector of degrees of freedom values at which coefficients should be calculated. In the case that a single degrees of freedom value appears multiple times throughout the solution path, the least regularized solution (corresponding to the smallest value of lambda) is chosen. If a degrees of freedom value does not appear at all in the solution path, this function chooses the solution whose degrees of freedom is largest, subject to being less than or equal to the specified value. |
| `type` | a character string, one of "primal", "dual", or "both", indicating whether primal coefficients, dual coefficients, or both, should be returned. Default is "primal", which corresponds to the solution of the original problem. |
| `...` | additional arguments passed to coef. |

## Value

Returns a list with the following components:

| | |
|---|---|
| `beta` | if the type is "primal" or "both", a matrix containing the primal coefficients, each column corresponding to a value of lambda. |

| | |
|---|---|
| u | if the type is "dual" or "both", a matrix containing the dual coefficients, each column corresponding to a value of lambda. |
| lambda | a numeric vector containing the sequence of tuning parameter values, corresponding to the columns of beta and u. |
| df | an integer vector containing the sequence of degrees of freedom values corresponding to the columns of beta and u. |

## See Also

genlasso, predict.genlasso, plot.genlasso

## Examples

```
# Constant trend filtering (the 1d fused lasso)
set.seed(0)
n = 20
beta0 = rep(sample(1:10,5),each=n/5)
y = beta0 + rnorm(n,sd=0.5)
a = fusedlasso1d(y)

# Get the coefficients that use 3, 4, and 5 degrees
# of freedom
coef(a,df=3:5)
```

---

# cv.trendfilter *Perform k-fold cross-validation to choose a trend filtering model*

---

## Description

This function performs k-fold cross-validation to choose the value of the regularization parameter lambda for a trend filtering problem, given the computed solution path. This function only applies to trend filtering objects with identity predictor matrix (no X passed).

## Usage

```
cv.trendfilter(object, k = 5, mode = c("lambda", "df"),
               approx = FALSE, rtol = 1e-07, btol = 1e-07,
               verbose = FALSE)
```

## Arguments

| | |
|---|---|
| object | the solution path object, of class "trendfilter", as returned by the trendfilter function. |
| k | an integer indicating the number of folds to split the data into. Must be between 2 and n-2 (n being the number of observations), default is 5. It is generally not a good idea to pass a value of k much larger than 10 (say, on the scale of n); see "Details" below. |

| | |
|---|---|
| mode | a character string, either "lambda" or "df". Specifying "lambda" means that the cross-validation error will be computed and reported at each value of lambda that appears as a knot in the solution path. Specifying "df" means that the cross-validation error will be computed and reported for every of degrees of freedom value (actually, estimate) incurred along the solution path. In the case that the same degrees of freedom value is visited multiple times, the model with the most regularization (smallest value of lambda) is considered. Default is "lambda". |
| approx | a logical variable indicating if the approximate solution path should be used (with no dual coordinates leaving the boundary). Default is FALSE. |
| rtol | a numeric variable giving the relative tolerance used in the calculation of the hitting and leaving times. A larger value is more conservative, and may cause the algorithm to miss some hitting or leaving events (do not change unless you know what you're getting into!). Defaultis 1e-7. |
| btol | similar to rtol but in absolute terms. If numerical instability is detected, first change rtol; then adjust btol if problems persist. |
| verbose | a logical variable indicating if progress should be reported after each knot in the path. |

## Details

For trend filtering (with an identity predictor matrix), the folds for k-fold cross-validation are chosen by placing every kth point into the same fold. (Here the points are implicitly ordered according to their underlying positions—either assumed to be evenly spaced, or explicitly passed through the pos argument.) The first and last points are not included in any fold and are always included in building the predictive model. As an example, with n=15 data points and k=4 folds, the points are assigned to folds in the following way:

$$x\ 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1\ x$$

where $x$ indicates no assignment. Therefore, the folds are not random and running cv.trendfilter twice will give the same result. In the calculation of the cross-validated error, the predicted value at a point is given by the average of the fits at this point's two neighbors (guaranteed to be in a different fold).

Running cross-validation in modes "lambda" and "df" often yields very similar results. The mode "df" simply gives an alternative parametrization for the sequence of cross-validated models and can be more convenient for some applications; if you are confused about its function, simply leave the mode equal to "lambda".

## Value

Returns and object of class "cv.trendfilter", a list with the following components:

| | |
|---|---|
| err | a numeric vector of cross-validated errors. |
| se | a numeric vector of standard errors (standard deviations of the cross-validation error estimates). |
| mode | a character string indicating the mode, either "lambda" or "df". |
| lambda | if mode="lambda", the values of lambda at which the cross-validation errors in err were computed. |

| | |
|---|---|
| lambda.min | if mode="lambda", the value of lambda at which the cross-validation error is minimized. |
| lambda.1se | if mode="lambda", the value of lambda chosen by the one standard error rule (the largest value of lambda such that the cross-validation error is within one standard error of the minimum). |
| df | if mode="df", the degrees of freedom values at which the cross-validation errors in err were computed. |
| df.min | if mode="df", the degrees of freedom value at which the cross-validation error is minimized. |
| df.1se | if mode="df", the degrees of freedom value chosen by the one standard error rule (the smallest degrees of freedom value such that cross-validation error is within one standard error of the minimum). |
| i.min | the index of the model minimizing the cross-validation error. |
| i.1se | the index of the model chosen by the one standard error rule. |
| call | the matched call. |

## See Also

trendfilter, plot.cv.trendfilter, plot.trendfilter

## Examples

```
# Constant trend filtering (the 1d fused lasso)
set.seed(0)
n = 50
beta0 = rep(sample(1:10,5),each=n/5)
y = beta0 + rnorm(n,sd=0.8)
a = fusedlasso1d(y)
plot(a)

# Choose lambda by 5-fold cross-validation
cv = cv.trendfilter(a)
plot(cv)
plot(a,lambda=cv$lambda.min,main="Minimal CV error")
plot(a,lambda=cv$lambda.1se,main="One standard error rule")

## Not run:
# Cubic trend filtering
set.seed(0)
n = 100
beta0 = numeric(100)
beta0[1:40] = (1:40-20)^3
beta0[40:50] = -60*(40:50-50)^2 + 60*100+20^3
beta0[50:70] = -20*(50:70-50)^2 + 60*100+20^3
beta0[70:100] = -1/6*(70:100-110)^3 + -1/6*40^3 + 6000
beta0 = -beta0
beta0 = (beta0-min(beta0))*10/diff(range(beta0))
y = beta0 + rnorm(n)
a = trendfilter(y,ord=3,maxsteps=150)
```

```
plot(a,nlam=5)

# Choose lambda by 5-fold cross-validation
cv = cv.trendfilter(a)
plot(cv)
plot(a,lambda=cv$lambda.min,main="Minimal CV error")
plot(a,lambda=cv$lambda.1se,main="One standard error rule")

## End(Not run)
```

---

`fusedlasso` *Compute the fused lasso solution path for a general graph, or a 1d or 2d grid*

---

## Description

These functions produce the solution path for a general fused lasso problem. The `fusedlasso` function takes either a penalty matrix or a graph object from the `igraph` package. The `fusedlasso1d` and `fusedlasso2d` functions are convenience functions that construct the penalty matrix over a 1d or 2d grid.

## Usage

```
fusedlasso(y, X, D, graph, gamma = 0, approx = FALSE, maxsteps = 2000,
           minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
   verbose = FALSE)
fusedlasso1d(y, pos, X, gamma = 0, approx = FALSE, maxsteps = 2000,
             minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
     verbose = FALSE)
fusedlasso2d(y, X, dim1, dim2, gamma = 0, approx = FALSE, maxsteps = 2000,
     minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-4,
     verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `y` | a numeric response vector. Alternatively, for `fusedlasso2d` with no matrix `X` passed, `y` can be a matrix (its dimensions corresponding to the underlying 2d grid). Note that when `y` is given as a vector in `fusedlasso2d`, with no `X` passed, it should be in column major order. |
| `pos` | only for `fusedlasso1d`, these are the optional positions of the positions in the 1d grid. If missing, the 1d grid is assumed to have unit spacing. |
| `X` | an optional matrix of predictor variables, with observations along the rows, and variables along the columns. If the passed `X` has more columns than rows, then a warning is given, and a small ridge penalty is added to the generalized lasso criterion before the path is computed. If `X` has less columns than rows, then its rank is not checked for efficiency, and (unlike the `genasso` function) a ridge penalty is not automatically added if it is rank deficient. Therefore, a tall, rank deficient `X` may cause errors. |

| | |
|---|---|
| D | only for fusedlasso, this is the penalty matrix, i.e., the oriented incidence matrix over the underlying graph (the orientation of each edge being arbitrary). Only one of D or graph needs to be specified. |
| graph | only for fusedlasso, this is the underlying graph as an igraph object from the igraph package. Only one of D or graph needs to be specified. |
| dim1 | only for fusedlasso2d, this is the number of rows in the underlying 2d grid. If missing and y is given as a matrix, it is assumed to be the number of rows of y. |
| dim2 | only for fusedlasso2d, this is the number of columns in the underlying 2d grid. If missing and y is given as a matrix, it is assumed to be the number of columns of y. |
| gamma | a numeric variable greater than or equal to 0, indicating the ratio of the two tuning parameters, one for the fusion penalty, and the other for the pure $\ell_1$ penalty. Default is 0. See "Details" for more information. |
| approx | a logical variable indicating if the approximate solution path should be used (with no dual coordinates leaving the boundary). Default is FALSE. Note that for the 1d fused lasso, with identity predicor matrix, this approximate path is the same as the exact solution path. |
| maxsteps | an integer specifying the maximum number of steps for the algorithm to take before termination. Default is 2000. |
| minlam | a numeric variable indicating the value of lambda at which the path should terminate. Default is 0. |
| rtol | a numeric variable giving the tolerance for determining the rank of a matrix: if a diagonal value in the R factor of a QR decomposition is less than R, in absolute value, then it is considered zero. Hence making rtol larger means being less stringent with determination of matrix rank. In general, do not change this unless you know what you are getting into! Default is 1e-7. |
| btol | a numeric variable giving the tolerance for accepting "late" hitting and leaving times: future hitting times and leaving times should always be less than the current knot in the path, but sometimes for numerical reasons they are larger; any computed hitting or leaving time larger than the current knot + btol is thrown away. Hence making btol larger means being less stringent withthe determination of hitting and leaving times. Again, in general, do not change this unless you know what you are getting into! Default is 1e-7. |
| eps | a numeric variable indicating the multiplier for the ridge penalty, in the case that X is wide (more columns than rows). If numeric problems occur, make eps larger. Default is 1e-4. |
| verbose | a logical variable indicating if progress should be reported after each knot in the path. |

## Details

The fused lasso estimate minimizes the criterion

$$1/2\sum_{i=1}^{n}(y_i - x_i^T\beta_i)^2 + \lambda \sum_{(i,j)\in E} |\beta_i - \beta_j| + \gamma \cdot \lambda \sum_{i=1}^{p} |\beta_i|,$$

where $x_i$ is the ith row of the predictor matrix and $E$ is the edge set of the underlying graph. The solution $\hat{\beta}$ is computed as a function of the regularization parameter $\lambda$, for a fixed value of $\gamma$. The default is to set $\gamma = 0$, which corresponds to pure fusion of the coefficient vector $\beta$. A choice $\gamma > 0$ introduces both sparsity and fusion in the coefficient vector, with a higher value placing more priority on sparsity.

If the predictor matrix is the identity, and the primal solution path $\beta$ is desired at several levels of the ratio parameter $\gamma$, it is much more efficient to compute the solution path once with $\gamma = 0$, and then use soft-thresholding via the `softthresh` function.

Finally, for the image denoising problem, i.e., the fused lasso over a 2d grid with identity predictor matrix, it is easy to specify a huge graph with a seemingly small amount of data. For instance, running the 2d fused lasso (with identity predictor matrix) on an image at standard 1080p HD resolution yields a graph with over 2 million edges. Moreover, in image denoising problems—somewhat unlike most other applications of the fused lasso (and generalized lasso)—a solution is often desired near the dense end of the path ($\lambda = 0$) as opposed to the regularized end ($\lambda = \infty$). The dual path algorithm implemented by the `fusedlasso2d` function begins at the fully regularized end and works its way down to the dense end. For a problem with many edges (dual variables), if a solution at the dense is desired, then it must usually pass through a huge number knots in the path. Hence it is not advisable to run `fusedlasso2d` on image denoising problems of large scale, as the dual solution path is computationally infeasible. It should be noted that a faster algorithm for the 2d fused lasso solution path (when the predictor matrix is the identity), which begins at the dense end of the path, is available in the `flsa` package.

## Value

The function returns an object of class "fusedlasso", and subclass "genlasso". This is a list with at least following components:

| | |
|---|---|
| `lambda` | values of lambda at which the solution path changes slope, i.e., kinks or knots. |
| `beta` | a matrix of primal coefficients, each column corresponding to a knot in the solution path. |
| `fit` | a matrix of fitted values, each column corresponding to a knot in the solution path. |
| `u` | a matrix of dual coefficients, each column corresponding to a knot in the solution path. |
| `hit` | a vector of logical values indicating if a new variable in the dual solution hit the box contraint boundary. A value of `FALSE` indicates a variable leaving the boundary. |
| `df` | a vector giving an unbiased estimate of the degrees of freedom of the fit at each knot in the solution path. |
| `y` | the observed response vector. Useful for plotting and other methods. |
| `completepath` | a logical variable indicating whether the complete path was computed (terminating the path early with the `maxsteps` or `minlam` options results in a value of `FALSE`). |
| `bls` | the least squares solution, i.e., the solution at lambda = 0. This can be `NULL` when `completepath` is `FALSE`. |
| `gamma` | the value of the lambda ratio. |
| `call` | the matched call. |

## Author(s)

Taylor B. Arnold and Ryan J. Tibshirani

## References

Tibshirani, R. J. and Taylor, J. (2011), "The solution path of the generalized lasso", Annals of Statistics 39 (3) 1335–1371.

Arnold, T. B. and Tibshirani, R. J. (2014), "Efficient implementations of the generalized lasso dual path algorithm", arXiv: 1405.3222.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J. and Knight, K. (2005), "Sparsity and smoothness via the fused lasso", Journal of the Royal Statistics Society: Series B 67(1), 91–108.

## See Also

softthresh, genlasso

## Examples

```
# Fused lasso on a custom graph
set.seed(0)
edges = c(1,2,1,3,1,5,2,4,2,5,3,6,3,7,3,8,6,7,6,8)
gr = graph(edges=edges,directed=FALSE)
plot(gr)
y = c(1,1,0,1,1,0,0,0) + rnorm(8,0.1)

# Can either pass the graph object directly, or
# first construct the penalty matrix, and then
# pass this
a1 = fusedlasso(y,graph=gr)
D = getDgSparse(gr)
a2 = fusedlasso(y,D=D)

plot(a1,numbers=TRUE)

## Not run:
# The 2d fused lasso with a predictor matrix X
set.seed(0)
dim1 = dim2 = 16
p = dim1*dim2
n = 300
X = matrix(rnorm(n*p),nrow=n)
beta0 = matrix(0,dim1,dim2)
beta0[(row(beta0)-dim1/2)^2 + (col(beta0)-dim2/2)^2 <=
(min(dim1,dim2)/3)^2] = 1
y = X %*% as.numeric(beta0) + rnorm(n)

# Takes about 30 seconds for the full solution path
out = fusedlasso2d(y,X,dim1=dim1,dim2=dim2)

# Grab the solution at 8 values of lambda over the path
a = coef(out,nlam=8)
```

```
# Plot these against the true coefficients
par(mar=c(1,1,2,1),mfrow=c(3,3))
cols = terrain.colors(30)
zlim = range(c(range(beta0),range(a$beta)))
image(beta0,col=cols,zlim=zlim,axes=FALSE)

for (i in 1:8) {
  image(matrix(a$beta[,i],nrow=dim1),col=cols,zlim=zlim,
  axes=FALSE)
  mtext(bquote(lambda==.(sprintf("%.3f",a$lambda[i]))))
}

## End(Not run)
```

---

`genlasso` *Compute the generalized lasso solution path for arbitrary penalty matrix*

---

## Description

This function computes the solution path of the generalized lasso problem for an arbitrary penalty matrix. Speciality functions exist for the trend filtering and fused lasso problems; see `trendfilter` and `fusedlasso`.

## Usage

```
genlasso(y, X, D, approx = FALSE, maxsteps = 2000, minlam = 0,
         rtol = 1e-07, btol = 1e-07, eps = 1e-4, verbose = FALSE,
         svd = FALSE)
```

## Arguments

| | |
|---|---|
| `y` | a numeric response vector. |
| `X` | an optional matrix of predictor variables, with observations along the rows, and variables along the columns. If missing, `X` is assumed to be the identity matrix. If the passed `X` does not have full column rank, then a warning is given, and a small ridge penalty is added to the generalized lasso criterion before the path is computed. |
| `D` | a penalty matrix. Its number of columns must be equal to the number of columns of `X`, or if no `X` is given, the length of `y`. This can be a sparse matrix from `Matrix` package, but this will be ignored (converted to a dense matrix) if `D` is row rank deficient or if `X` is specified. See "Details" below. |
| `approx` | a logical variable indicating if the approximate solution path should be used (with no dual coordinates leaving the boundary). Default is `FALSE`. |
| `maxsteps` | an integer specifying the maximum number of steps for the algorithm to take before termination. Default is 2000. |

`minlam` a numeric variable indicating the value of lambda at which the path should terminate. Default is 0.

`rtol` a numeric variable giving the tolerance for determining the rank of a matrix: if a diagonal value in the R factor of a QR decomposition is less than R, in absolute value, then it is considered zero. Hence making rtol larger means being less stringent with determination of matrix rank. In general, do not change this unless you know what you are getting into! Default is 1e-7.

`btol` a numeric variable giving the tolerance for accepting "late" hitting and leaving times: future hitting times and leaving times should always be less than the current knot in the path, but sometimes for numerical reasons they are larger; any computed hitting or leaving time larger than the current knot + btol is thrown away. Hence making btol larger means being less stringent withthe determination of hitting and leaving times. Again, in general, do not change this unless you know what you are getting into! Default is 1e-7.

`eps` a numeric variable indicating the multiplier for the ridge penalty, in the case that `X` is column rank deficient. Default is 1e-4.

`verbose` a logical variable indicating if progress should be reported after each knot in the path.

`svd` a logical variable indicating if the genlasso function should use singular value decomposition to solve least squares problems at each path step, which is slower, but should be more stable.

## Details

The generalized lasso estimate minimizes the criterion

$$1/2\|y - X\beta\|_2^2 + \lambda\|D\beta\|_1.$$

The solution $\hat{\beta}$ is computed as a function of the regularization parameter $\lambda$. The advantage of the `genlasso` function lies in its flexibility, i.e., the user can specify any penalty matrix `D` of their choosing. However, for a trend filtering problem or a fused lasso problem, it is strongly recommended to use one of the speciality functions, `trendfilter` or `fusedlasso`. When compared to these functions, `genlasso` is not as numerically stable and much less efficient.

Note that, when `D` is passed as a sparse matrix, the linear systems that arise at each step of the path algorithm are solved separately via a sparse solver. The usual strategy (when `D` is simply a matrix) is to maintain a matrix factorization of `D`, and solve these systems by (or downdating) this factorization, as these linear systems are highly related. Therefore, when `D` is sufficiently sparse and structured, it can be advantageous to pass it as a sparse matrix; but if `D` is truly dense, passing it as a sparse matrix will be highly inefficient.

## Value

Returns an object of class "genlasso", a list with at least following components:

`lambda` values of lambda at which the solution path changes slope, i.e., kinks or knots.

`beta` a matrix of primal coefficients, each column corresponding to a knot in the solution path.

| | |
|---|---|
| `fit` | a matrix of fitted values, each column corresponding to a knot in the solution path. |
| `u` | a matrix of dual coefficients, each column corresponding to a knot in the solution path. |
| `hit` | a vector of logical values indicating if a new variable in the dual solution hit the box contraint boundary. A value of `FALSE` indicates a variable leaving the boundary. |
| `df` | a vector giving an unbiased estimate of the degrees of freedom of the fit at each knot in the solution path. |
| `y` | the observed response vector. Useful for plotting and other methods. |
| `completepath` | a logical variable indicating whether the complete path was computed (terminating the path early with the `maxsteps` or `minlam` options results in a value of `FALSE`). |
| `bls` | the least squares solution, i.e., the solution at lambda = 0. |
| `call` | the matched call. |

## Author(s)

Taylor B. Arnold and Ryan J. Tibshirani

## References

Tibshirani, R. J. and Taylor, J. (2011), "The solution path of the generalized lasso", Annals of Statistics 39 (3) 1335–1371.

Arnold, T. B. and Tibshirani, R. J. (2014), "Efficient implementations of the generalized lasso dual path algorithm", arXiv: 1405.3222.

## See Also

`trendfilter`, `fusedlasso`, `coef.genlasso`, `predict.genlasso`, `plot.genlasso`

## Examples

```
# Using the generalized lasso to run a standard lasso regression
# (for example purposes only! for pure lasso problems, use LARS
# instead)
set.seed(1)
n = 100
p = 10
X = matrix(rnorm(n*p),nrow=n)
y = 3*X[,1] + rnorm(n)
D = diag(1,p)
out = genlasso(y,X,D)
coef(out, lambda=sqrt(n*log(p)))
```

getDxx *Helper functions for constructing generalized lasso penalty matrices*

## Description

These are utility functions for creating penalty matrices for the fused lasso and trend filtering problems. Most users will not need to explicitly construct these as they are created internally by the `fusedlasso` or `trendfilter` functions. The sparse variants output sparse matrices, which should be used whenever possible because of a significant savings in both construction speed and memory usage.

The function getGraph is an inverse function for fused lasso problems, returning an `igraph` object (from the `igraph` package), the graph corresponding to the passed penalty matrix.

## Usage

```
getD1d(n)
getD1dSparse(n)
getD2d(dim1, dim2)
getD2dSparse(dim1, dim2)
getDg(graph)
getDgSparse(graph)
getDtf(n, ord)
getDtfSparse(n, ord)
getDtfPos(n, ord, pos)
getDtfPosSparse(n, ord, pos)
getGraph(D)
```

## Arguments

| | |
|---|---|
| | The arguments for the sparse variants are identical to those for the regular variants, which are described below. |
| `n` | for `getD1d`, `getDtf`, and `getDtfPos`, the number of points in the 1d sequence. |
| `dim1, dim2` | for `getD2d`, the number of rows and columns in the 2d grid, respectively. |
| `graph` | for `getDg`, an `igraph` object from the `igraph` package, upon which the penalty matrix should be based (the penalty matrix is the oriented incidence matrix of the graph, with arbitrary orientations assigned to each edge). |
| `ord` | for `getDtf`, and `getDtfPos`, the order of the polynomial. E.g., ord=0 is the 1d fused lasso and ord=1 is linear trend filtering. Hence the returned matrix is the discrete (ord+1)st derivative operator. |
| `pos` | for `getDtfPos`, a numeric vector giving the positions of points in the 1d sequence. Must have length `n`. |
| `D` | for `getGraph`, a fused lasso penalty matrix, the incidence matrix of an undirected graph, with arbitrary edge orientations. |

## Value

All functions except getGraph return a penalty matrix, either in standard R matrix format or as a sparse matrix of class dgCMatrix via the Matrix package. The function getGraph returns an igraph object from the igraph package.

## See Also

fusedlasso, trendfilter

## Examples

```
getD1d(9)
getDtfSparse(10,2)

graph = getGraph(getD2dSparse(4,4))
plot(graph)
```

---

# iterate *Iterate a genlasso object*

---

## Description

Given an incomplete genlasso path object, this function continues the path computation from the last computed knot, either until the complete path has been computed or the step limit specified by moresteps has been reached. All options are assumed to be the same as those in the initial call to a genlasso function (as in genlasso, fusedlasso, or trendfilter), with the exception of minlam and verbose, which can be changed with a call to iterate.

## Usage

```
iterate(object, moresteps=200, minlam=0, verbose=FALSE)
```

## Arguments

| | |
|---|---|
| object | a genlasso object with an incomplete path. |
| moresteps | an integer specifying the number of additional steps to take, starting from termination point of the passed (incomplete) path object. |
| minlam | a numeric variable indicating the value of lambda at which the path should terminate. Default is 0. |
| verbose | a logical variable indicating if progress should be reported after each knot in the path. |

## Value

Returns an list of the same class typing and same structure as the passed object.

## See Also

genlasso, trendfilter, fusedlasso

## Examples

```
# Sparse 2d fused lasso
library(genlasso)
set.seed(1)
dim1 = dim2 = 10
n = 100
y = as.numeric(row(diag(dim1)) > 5 & col(diag(dim2)) > 5) * 3 + rnorm(n)

a10 = fusedlasso2d(y, dim1=dim1, dim2=dim2, gamma=0.5, maxsteps=10)
a20 = fusedlasso2d(y, dim1=dim1, dim2=dim2, gamma=0.5, maxsteps=20)
a30 = fusedlasso2d(y, dim1=dim1, dim2=dim2, gamma=0.5, maxsteps=30)
b20 = iterate(a10, moresteps=10)
b30 = iterate(b20, moresteps=10)

# Check for equality; should match on all but 'call'
b20$call = a20$call
b30$call = a30$call
all.equal(target=a20, current=b20)
all.equal(target=a30, current=b30)
```

---

# plot.genlasso *Plotting methods for generalized lasso objects*

---

## Description

The function `plot.genlasso` produces a plot of the coordinate paths for objects of class "genlasso". This can be helpful for visualizing the full solution path for small problems; however, for moderate or large problems, the plot produced can be quite dense and difficult to interpret. The function `plot.trendfilter` applies to objects of class "trendfilter", and plots trend filtering coefficients at a single value of lambda (or multiple values, as specified by the user) as a function of the input positions (which, recall, are assumed to be evenly spaced if not specified). The function `plot.cv.trendfilter` plots the output of `cv.trendfilter`.

## Usage

```
## S3 method for class 'genlasso'
plot(x, type = c("primal", "dual", "both"), numbers = FALSE,
     vlines = TRUE, xlab, ylab, ...)
## S3 method for class 'trendfilter'
plot(x, style = c("trend", "path"), lambda, nlam, df, xlab,
     ylab, ...)
## S3 method for class 'cv.trendfilter'
plot(x, legendpos = "top", xlab, ylab, ...)
```

## Arguments

| | |
|---|---|
| x | an object of the appropriate class ("genlasso" or anything class inherits this for plot.genlasso, "trendfilter" for plot.trendfilter, and "cv.trendfilter" for plot.cv.trendfilter). |
| type | for plot.genlasso, a character string, one of "primal", "dual", or "both", indicating which solution path system(s) should be plotted. Default is "primal". |
| numbers | for plot.genlasso, a logical variable indicating if coordinate paths should be labeled by their numbers. Default is FALSE. |
| vlines | for plot.genlasso, a logical variable indicating if dashed lines should be drawn at knots in the path, with black lines for hitting events, and red lines for leaving events. Default is TRUE. |
| style | for plot.trendfilter, a character string, either "trend" or "path". If "trend", then trend filtering coefficients are plotted according to their underlying positions. If "path", then a plot of the coordinate paths is produced with the function plot.genlasso. Default is "trend". |
| lambda, nlam, df | for plot.trendfilter, these arguments work exactly as they do in coef.genlasso, and they are used to specify which solutions should be extracted and plotted from the computed solution path stored in x. The only difference is, if all three are missing, then nlam is set to 10 (whereas in coef.genlasso, the default is to set lambda equal to the full set of knots along the solution path). |
| legendpos | for plot.cv.trendfilter, a character string indicating the position of the legend. Default is "top". |
| xlab | an optional character string label for the x-axis. |
| ylab | an optional character string label for the y-axis. |
| ... | additional arguments. |

## Value

For plot.trendfilter, with style set to "trend", a coefficient object is silently returned as specified by lambda, nlam, or df.

## See Also

genlasso, trendfilter, cv.trendfilter

## Examples

```
# Constant trend filtering (the 1d fused lasso)
set.seed(0)
n = 100
beta0 = rep(sample(1:10,5),each=n/5)
y = beta0 + rnorm(n,sd=0.8)
a = fusedlasso1d(y)
cv = cv.trendfilter(a)

plot(a,style="path")
```

```
plot(cv)
plot(a,lambda=cv$lambda.1se)
```

---

`predict.genlasso` *Make predictions given a genlasso object*

---

## Description

This predict method for the genlasso class makes a prediction for the fitted values at new predictor measurements. Hence it is really only useful when the generalized lasso model has been fit with a nonidentity predictor matrix. In the case that the predictor matrix is the identity, it does the same thing as `coef.genlasso`.

## Usage

```
## S3 method for class 'genlasso'
predict(object, lambda, nlam, df, Xnew, ...)
```

## Arguments

| | |
|---|---|
| `object` | object of class "genlasso", or an object which inherits this class (i.e., "fusedlasso", "trendfilter"). |
| `lambda` | a numeric vector of tuning parameter values at which coefficients should be calculated. The user can choose to specify one of `lambda`, `nlam`, or `df`; if none are specified, then coefficients are returned at every knot in the solution path. |
| `nlam` | an integer indicating a number of tuning parameters values at which coefficients should be calculated. The tuning parameter values are then chosen to be equally spaced on the log scale over the first half of the solution path (this is if the full solution path has been computed; if only a partial path has been computed, the tuning parameter values are spaced over the entirety of the computed path). |
| `df` | an integer vector of degrees of freedom values at which coefficients should be calculated. In the case that a single degrees of freedom value appears multiple times throughout the solution path, the least regularized solution (corresponding to the smallest value of lambda) is chosen. If a degrees of freedom value does not appear at all in the solution path, the least regularized solution at which this degrees of freedom value is not exceeded is chosen. |
| `Xnew` | a numeric matrix X, containing new predictor measurements at which predictions should be made. If missing, it is assumed to be the same as the existing predictor measurements in `object`. |
| `...` | additional arguments passed to predict. |

## Value

Returns a list with the following components:

| | |
|---|---|
| `fit` | a numeric matrix of predictor values, one column for each value of lambda. |
| `lambda` | a numeric vector containing the sequence of tuning parameter values, corresponding to the columns of `fit`. |
| `df` | if `df` was specified, an integer vector containing the sequence of degrees of freedom values corresponding to the columns of `fit`. |

## See Also

`coef.genlasso`

---

`softthresh` *Fit a sparse variant of the fused lasso*

---

## Description

This function computes solution path to a fused lasso problem of the form

$$1/2\sum_{i=1}^{n}(y_i-\beta_i)^2+\lambda\sum_{(i,j)\in E}|\beta_i-\beta_j|+\gamma\cdot\lambda\sum_{i=1}^{p}|\beta_i|,$$

given the solution path corresponding to $\gamma = 0$. Note that the predictor matrix here is the identity, and in this case the new solution path is given by a simple soft-thresholding operation (Friedman et al. 2007).

## Usage

```
softthresh(object, lambda, gamma)
```

## Arguments

| | |
|---|---|
| `object` | an object of class "fusedlasso", fit with no predictor matrix `X` (taken to mean that the predictor matrix is the identity) and with `gamma` set to 0. Other objects will issue a warning that soft-thresholding does not give the exact primal solution path to a sparsified generalized lasso problem. |
| `lambda` | a numeric vector giving the values of lambda at which the solution should be computed and returned; if missing, defaults to the knots in the solution path stored in `object`. |
| `gamma` | a numeric variable giving the ratio of the fusion and sparsity tuning parameters, must be greater than or equal to 0. |

## Value

Returns a numeric matrix of primal solutions, one column for each value of lambda.

## References

Friedman J., Hastie T., Hoefling H. and Tibshirani, R. (2007), "Pathwise coordinate optimization", Annals of Applied Statistics 1 (2) 302–332.

## See Also

fusedlasso

## Examples

```
# The 1d fused lasso
set.seed(0)
n = 100
beta0 = rep(sample(1:10,5),each=n/5)
beta0 = beta0-mean(beta0)
y = beta0 + rnorm(n,sd=0.8)
a = fusedlasso1d(y)

lambda = 4
b1 = coef(a,lambda=lambda)$beta

gamma = 0.5
b2 = softthresh(a,lambda=lambda,gamma=gamma)

plot(1:n,y)
lines(1:n,b1)
lines(1:n,b2,col="red")
legend("topright",lty=1,col=c("black","red"),
       legend=c(expression(gamma==0),expression(gamma==0.5)))
```

---

# trendfilter *Compute the trend filtering solution path for any polynomial order*

---

## Description

This function computes the solution path for the trend filtering problem of an arbitrary polynomial order. When the order is set to zero, trend filtering is equivalent to the 1d fused lasso, see fusedlasso1d.

## Usage

```
trendfilter(y, pos, X, ord = 1, approx = FALSE, maxsteps = 2000,
            minlam = 0, rtol = 1e-07, btol = 1e-07, eps = 1e-04,
            verbose = FALSE)
```

## Arguments

| | |
|---|---|
| y | a numeric response vector. |
| pos | an optional numeric vector specifying the positions of the observations, and missing pos is assumed to mean unit spacing. |
| X | an optional matrix of predictor variables, with observations along the rows, and variables along the columns. If the passed X has more columns than rows, then a warning is given, and a small ridge penalty is added to the generalized lasso criterion before the path is computed. If X has less columns than rows, then its rank is not checked for efficiency, and (unlike the genasso function) a ridge penalty is not automatically added if it is rank deficient. Therefore, a tall, rank deficient X may cause errors. |
| ord | an integer specifying the desired order of the piecewise polyomial produced by the solution of the trend filtering problem. Must be non-negative, and the default to 1 (linear trend filtering). |
| approx | a logical variable indicating if the approximate solution path should be used (with no dual coordinates leaving the boundary). Default is FALSE. Note that for the 1d fused lasso (zeroth order trend filtering), with identity predictor matrix, this approximate path is the same as the exact solution path. |
| maxsteps | an integer specifying the maximum number of steps for the algorithm to take before termination. Default is 2000. |
| minlam | a numeric variable indicating the value of lambda at which the path should terminate. Default is 0. |
| rtol | a numeric variable giving the tolerance for determining the rank of a matrix: if a diagonal value in the R factor of a QR decomposition is less than R, in absolute value, then it is considered zero. Hence making rtol larger means being less stringent with determination of matrix rank. In general, do not change this unless you know what you are getting into! Default is 1e-7. |
| btol | a numeric variable giving the tolerance for accepting "late" hitting and leaving times: future hitting times and leaving times should always be less than the current knot in the path, but sometimes for numerical reasons they are larger; any computed hitting or leaving time larger than the current knot + btol is thrown away. Hence making btol larger means being less stringent withthe determination of hitting and leaving times. Again, in general, do not change this unless you know what you are getting into! Default is 1e-7. |
| eps | a numeric variable indicating the multiplier for the ridge penalty, in the case that X is wide (more columns than rows). If numeric problems occur, make eps larger. Default is 1e-4. |
| verbose | a logical variable indicating if progress should be reported after each knot in the path. |

## Details

When the predictor matrix is the identity, trend filtering fits a piecewise polynomial to linearly ordered observations. The result is similar to that of a polynomial regression spline or a smoothing spline, except the knots in the piecewise polynomial (changes in the (k+1)st derivative, if the polynomial order is k) are chosen adaptively based on the observations. This is in contrast to regression

splines, where the knots are prespecified, and smoothing splines, which place a knot at every data point.

With a nonidentity predictor matrix, the trend filtering problem enforces piecewise polynomial smoothness along successive components of the coefficient vector. This can be used to fit a kind of varying coefficient model.

We note that, in the signal approximator (identity predictor matrix) case, fitting trend filtering estimate with arbitrary positions `pos` is theoretically no harder than doing so on an evenly spaced grid. However in practice, with differing gaps between points, the algorithm can become numerically unstable even for large (or moderately large) problems. This is especially true as the polynomial order increases. Hence, use the positions argument `pos` with caution.

## Value

Returns an object of class "trendfilter", a subclass of "genlasso". This is a list with at least following components:

| | |
|---|---|
| `lambda` | values of lambda at which the solution path changes slope, i.e., kinks or knots. |
| `beta` | a matrix of primal coefficients, each column corresponding to a knot in the solution path. |
| `fit` | a matrix of fitted values, each column corresponding to a knot in the solution path. |
| `u` | a matrix of dual coefficients, each column corresponding to a knot in the solution path. |
| `hit` | a vector of logical values indicating if a new variable in the dual solution hit the box contraint boundary. A value of `FALSE` indicates a variable leaving the boundary. |
| `df` | a vector giving an unbiased estimate of the degrees of freedom of the fit at each knot in the solution path. |
| `y` | the observed response vector. Useful for plotting and other methods. |
| `completepath` | a logical variable indicating whether the complete path was computed (terminating the path early with the `maxsteps` or `minlam` options results in a value of `FALSE`). |
| `bls` | the least squares solution, i.e., the solution at lambda = 0. This can be `NULL` when `completepath` is `FALSE`. |
| `ord` | the order of the piecewise polyomial that has been fit. |
| `call` | the matched call. |

## Author(s)

Taylor B. Arnold and Ryan J. Tibshirani

## References

Tibshirani, R. J. and Taylor, J. (2011), "The solution path of the generalized lasso", Annals of Statistics 39 (3) 1335–1371.

Tibshirani, R. J. (2014), "Adaptive piecewise polynomial estimation via trend filtering", Annals of Statistics 42 (1): 285–323.

Arnold, T. B. and Tibshirani, R. J. (2014), "Efficient implementations of the generalized lasso dual path algorithm", arXiv: 1405.3222.

Kim, S.-J., Koh, K., Boyd, S. and Gorinevsky, D. (2009), "l1 trend filtering", SIAM Review 51 (2), 339–360.

## See Also

fusedlasso1d, genlasso, cv.trendfilter, plot.trendfilter

## Examples

```
# Constant trend filtering (the 1d fused lasso)
set.seed(0)
n = 100
beta0 = rep(sample(1:10,5),each=n/5)
y = beta0 + rnorm(n,sd=0.8)
a = fusedlasso1d(y)
plot(a)

# Linear trend filtering
set.seed(0)
n = 100
beta0 = numeric(n)
beta0[1:20] = (0:19)*4/19+2
beta0[20:45] = (25:0)*3/25+3
beta0[45:80] = (0:35)*9/35+3
beta0[80:100] = (20:0)*4/20+8
y = beta0 + rnorm(n)
a = trendfilter(y,ord=1)
plot(a,df=c(2,3,4,10))

# Cubic trend filtering
set.seed(0)
n = 100
beta0 = numeric(100)
beta0[1:40] = (1:40-20)^3
beta0[40:50] = -60*(40:50-50)^2 + 60*100+20^3
beta0[50:70] = -20*(50:70-50)^2 + 60*100+20^3
beta0[70:100] = -1/6*(70:100-110)^3 + -1/6*40^3 + 6000
beta0 = -beta0
beta0 = (beta0-min(beta0))*10/diff(range(beta0))
y = beta0 + rnorm(n)
a = trendfilter(y,ord=3)
plot(a,nlam=5)
```

# Index