Introduction to the genlasso package

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

Abstract

We present a short tutorial and introduction to using the R package **genlasso**, which is used for computing the solution path of the generalized lasso problem discussed in Tibshirani and Taylor (2011). Use cases of the generalized lasso include the fused lasso over an arbitrary graph, and trend fitting of any given polynomial order. Our implementation includes a function to solve the generalized lasso is its most general form, as well as special functions to cover the fused lasso and trend filtering subproblems. The general implementation maintains and updates a matrix factorization to successively solve related linear systems; the specialized implementations forego this update routine and exploit subproblem structure, which improves both stability and speed of the computation. Standard S3 methods such as `plot`, `predict`, and `coef` are also included to assist in studying the output solution path objects.

Keywords: generalized lasso, path algorithm, fused lasso, trend filtering.

1. Introduction

A recent paper by Tibshirani and Taylor (2011) introduced the generalized lasso path algorithm, which computes the solution path to the following optimization problem:

$$\hat{\beta} \in \operatorname*{argmin}_{\beta \in \mathbb{R}^p} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|D\beta\|_1, \tag{1}$$

where $y \in \mathbb{R}^n$ is a response vector, $X \in \mathbb{R}^{n \times p}$ is a matrix of predictor variables, $D \in \mathbb{R}^{m \times p}$ is a penalty matrix, and $\lambda \geq 0$ is a regularization parameter. In other words, the path algorithm computes the solution $\hat{\beta} = \hat{\beta}(\lambda)$ as a function of the parameter λ. This is done by solving the equivalent Lagrange dual problem,

$$\hat{u} \in \operatorname*{argmin}_{u \in \mathbb{R}^m} \|XX^+y - (X^+)^T D^T u\|_2^2 \text{ subject to } \|u\|_\infty \leq \lambda, \tag{2}$$

where $X \in \mathbb{R}^{n \times p}$ is assumed to have full column rank[1] and $X^+ = (X^T X)^{-1} X^T$ is the Moore-Penrose generalized inverse of X. The relationship between the primal and dual solutions is

$$\hat{\beta} = (X^T X)^{-1}(X^T y - D^T \hat{u}), \tag{3}$$

and hence solving for $\hat{u}$ in (2) yields the solution $\hat{\beta}$ in (1) via the above relationship. When $X = I$, often called the "signal approximator" case, the dual and primal-dual relationship can

[1] If the predictor matrix X does not have full column rank (occurring, say, if $p > n$) then a small ridge penalty can be added to the criterion in (1), and the problem can be subsequently rewritten as a generalized lasso problem with a new full column rank predictor matrix.

be simplified,

$$\hat{u} \in \operatorname*{argmin}_{u \in \mathbb{R}^m} \|y - D^T u\|_2^2 \text{ subject to } \|u\|_\infty \leq \lambda, \tag{4}$$

and

$$\hat{\beta} = y - D^T \hat{u}. \tag{5}$$

The dual problem (4), and more generally (2) when $X \neq I$, is "easier" to consider because loosely speaking the non-differentiable box constraint is free of any linear transformation. Computing the solution path of (5) (and of (3)) essentially reduces to solving several linear systems that differ by one more or one less variable. This is much like the lasso solution path. See Tibshirani and Taylor (2011) for details.

The problem (1) is solvable at a fixed value of λ (or a fixed sequence of λ values) by several convex optimization tools, both open source and proprietary. The novelty here is that the solution is computed for all values of the tuning parameter λ simultaneously. Perhaps not surprisingly, computation of the full path does not scale as efficiently as does computation at individual λ values. The purpose of this package is not to provide a solver for large scale generalized lasso problems, but instead to deliver the entire solution path when computationally amenable.

There are three main functions: `genlasso`, `fusedlasso`, and `trendfilter`. The first function computes the solution path of any problem of the general form (1); the latter two are specialized functions designed to compute the solution path of the fused lasso and trend filtering subproblems, respectively (additionally, wrapper functions `fusedlasso1d` and `fusedlasso2d` are available to fit the fused lasso over 1d and 2d grids, respectively). These latter functions should be used whenever possible, as they provide a significant improvement in both computational efficiency and numerical stability.

This vignette is intended to get new users quickly up to speed on using the **genlasso** package for statistical modelling. Sections 2–4 give short code snippets for common use cases of the package, solving fused lasso problems, trend filtering problems, and using cross-validation. In a future paper we will reveal the details of various implementation strategies used in this package, as well as provide empirical results on computationally accuracy and run times of the various strategies, to help assist users in deciding what problems can and cannot be reasonably solved with the path algorithm.

2. The standard lasso

As a first toy example to using the **genlasso** package, we show how to compute the solution path of the usual lasso optimization problem. (This is intended as an example only, and we do not recommend this strategy for solving lasso problems!)

We first generate some data, consisting of a predictor matrix X with 10 variables and 100 observations generated by independent draws from a standard normal distribution. The response vector y is made to be a noisy version of the first column of the data matrix X:

```
> library("genlasso")
> set.seed(1)
> n = 100
> p = 10
```

```
> X = matrix(rnorm(n*p), ncol=p)
> y = X[,1] + rnorm(n)
```

In order to write the standard lasso as a generalized lasso problem (1), we construct a penalty matrix D equal to the 10-dimensional identity matrix:

```
> D = diag(1,p)
```

Now we can run the path solution for the (generalized) lasso:

```
> out = genlasso(y, X=X, D=D)
```

Like the `lm` function in the **stats** package, the output of the generalized lasso has a compact standard plot output. It gives the function call information as well as the length of the computed path:

```
> out

Call:
genlasso(y = y, X = X, D = D)

Output:
Path model with 10 total steps.
```

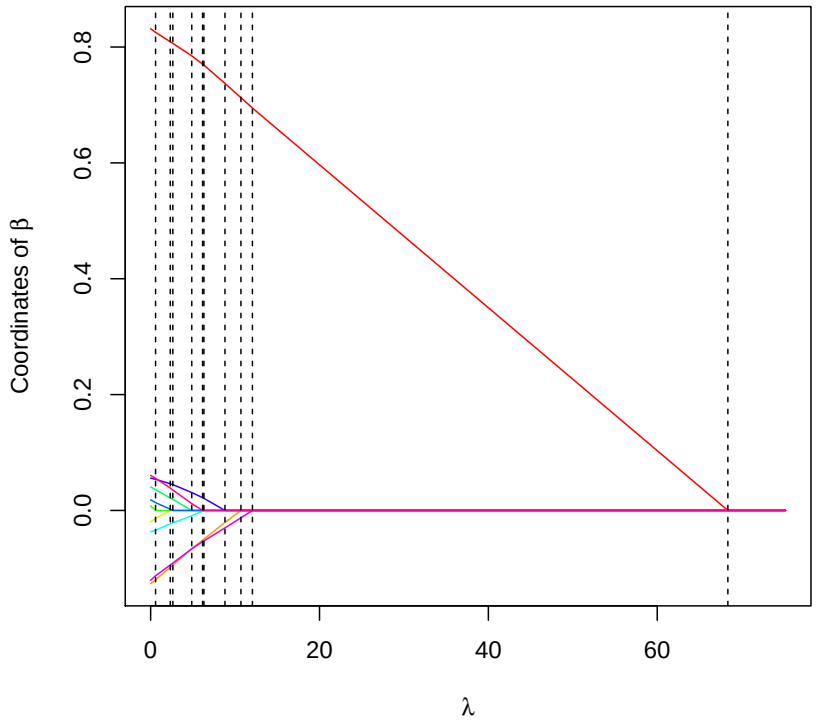
More information about each step, including an unbiased estimate of the degrees of freedom of the fit and the residual sum of squares, can be obtained by printing the summary of the output:

```
> summary(out)

   df   lambda      rss
   90   68.365   160.66
   91   12.054   104.79
   92   10.694   104.09
   93    8.803   102.87
   94    6.307   101.30
   95    6.153   101.21
   96    4.870   100.44
   97    2.630    99.37
   98    2.325    99.26
   99    0.582    98.88
```

A simple plot of the solution path can be produced by calling the `plot` function on the output object:

```
> plot(out)
```



Each color shows a coordinate of the solution path over λ, Note that as λ increases, all coordinates of the primal solution are quickly shrunken to zero, except the first coordinate, drawn in red.

Finally, it is easy to extract the coefficients β for a particular value (or values) of λ. Here we calculate the coefficients for the tuning parmeter $\lambda = \sqrt{n \log(p)}$, as is suggested for model selection consistency by recent theory:

```
> coef(out, lambda=sqrt(n*log(p)))

$beta
              15.174
 [1,]  6.562310e-01
 [2,]  5.583701e-17
 [3,]  3.926975e-17
 [4,] -6.339336e-17
 [5,]  7.858477e-17
 [6,] -3.960272e-17
 [7,] -3.699918e-17
 [8,]  4.592585e-17
 [9,] -6.083313e-17
[10,]  9.324401e-17
```

```
$lambda
[1] 15.17427
```

This is reassuring, as it essentially recovers the true model from which we had simulated the data. Also, as expected, the lasso has somewhat underestimated the magnitude of the nonzero coefficient due to the shrinking nature of the ℓ_1 penalty.

3. The fused lasso

3.1. The 1d fused lasso

A simple example of a problem which fits naturally into the generalized lasso structure is the 1d fused lasso. In the common signal approximator case, $X = I$, we assume that the observed data $y = (y_1, \dots y_n) \in \mathbb{R}^n$ is generated from a process whose mean changes at only a smaller number of locations, when ordered sequentially from 1 to n. The goal is hence to find a piecewise constant vector of coefficients, of course, fitting well to y. This is done by solving the following minimization problem:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 + \lambda \sum_{i=1}^{n-1} |\beta_{i+1} - \beta_i|, \tag{6}$$

which is a version of (1) with D equal to the matrix of first differences:

$$D = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 \\ & & & \dots & & \\ 0 & 0 & 0 & \dots & -1 & 1 \end{bmatrix}. \tag{7}$$

We generate a data set with four change points:

```
> set.seed(1)
> n = 100
> i = 1:n
> y = (i > 20 & i < 30) + 5*(i > 50 & i < 70) +
+   rnorm(n, sd=0.1)
```

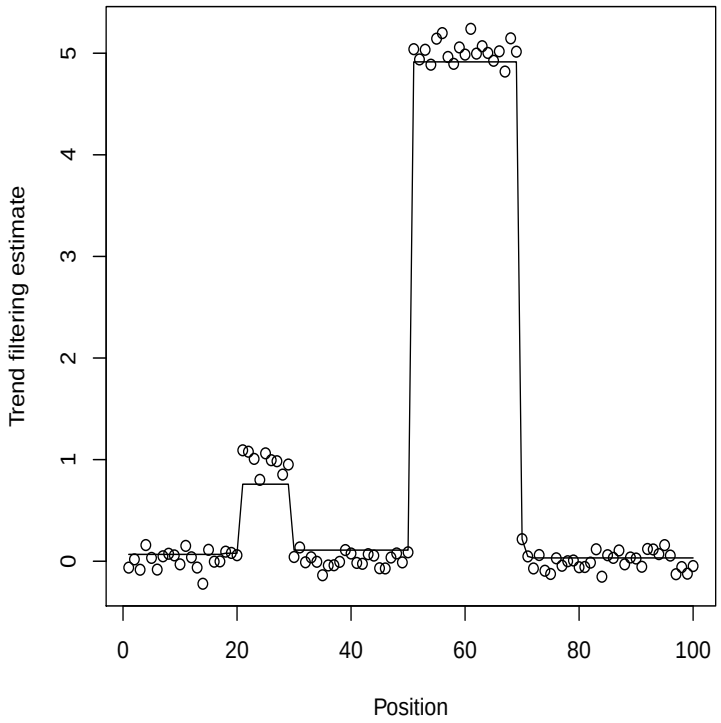
Now to fit the 1d fused lasso, we simply call the `fusedlasso1d` function (and we pass no X matrix, with indicates that $X = I$):

```
> out = fusedlasso1d(y)
```

An alternative method would be to call the `genlasso` function with argument `D=getD1d(n)`, where `getD1d` is a convenience function to construct the first difference matrix in (7). However, this is much less efficient and less numerically stable, and it is highly recommended to use the speciality implementations `fusedlasso` (wrappers `fusedlasso1d` and `fusedlasso2d`) and `trendfilter` whenever possible.

The model output works essentially the same as that produced by the `genlasso` function discussed in Section 2, giving printing, summaries, and coefficients in the exactly the same way. Additionally, a new plot function is available to show how the piecewise constant model fits the data at specified values of λ:

```
> plot(out, lambda=1)
```



The estimate at $\lambda = 1$ fits the data fairly well, and also provides a good estimate of the change point locations. To retrieve coordinate plots as in Section 2, use the `plot` function with argument `style="path"`.

3.2. The 2d fused lasso and arbitrary graphs

The 1d fused lasso can be extended to an arbitrary graph structure, where the absolute difference between the coefficients between neighboring nodes is penalized:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 + \lambda \sum_{(i,j) \in E} |\beta_i - \beta_j|, \tag{8}$$

where E is the edge set of the graph. Note that this is still a generalized lasso problem (1), with the penalty matrix D still a difference matrix between the appropriate pairs of

coefficients. In the graph framework, underlying the 1d fused lasso is the chain graph, or 1d grid. Another common structure is the 2d grid, and with this underlying graph problem (8) is called the 2d fused lasso, a technique used for image denoising.

We generate data in the form of a 16 by 16 grid of points; the mean of the data is equal to 2 for points within a distance of 4 from the middle of the grid, and 1 for all other points.

```
> set.seed(1)
> y = matrix(runif(256), 16, 16)
> i = (row(y) - 8.5)^2 + (col(y) - 8.5)^2 <= 4^2
> y[i] = y[i] + 1
```

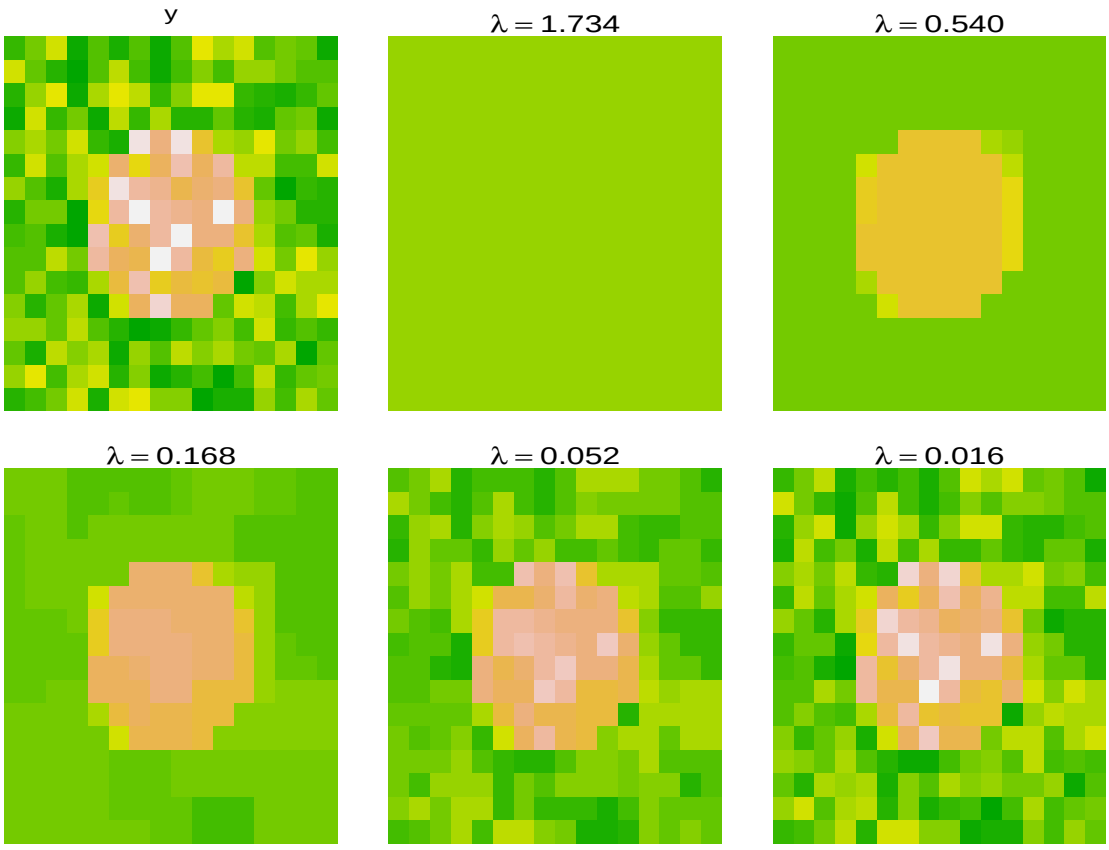
To compute the solution path of this problem, we call the `fusedlasso2d` function:

```
> out = fusedlasso2d(y)
```

Since the input data `y` is a matrix, the function knows the dimensions of the 2d grid. Otherwise we could specify the dimensions as inputs to the function call, `dim1` and `dim2`. We can extract 5 solutions along the solution path (evenly spaced on the log scale):

```
> co = coef(out, nlam=5)
```

and plotting them, along with the original data y (in the top-left corner), gives a rough understanding of what the solution path looks like:



We can see that the algorithm does a pretty good job of finding the region with a different mean, particularly for the 2nd largest value of λ displayed in the plot. With more regularization, the entire image is assigned a constant coefficient; with less, the coefficients are overly noisy.

Finally, it is possible to specify a generic underlying graph for the fused lasso problem. The function `fusedlasso` (for which both `fusedlasso1d` and `fusedlasso2d` simply act as wrappers) takes either a generic difference matrix D—i.e., a matrix such that each row contains a single -1 and 1 and all 0s otherwise—or an `igraph` graph object from the **igraph** package (Csardi and Nepusz 2006). The function `fusedlasso` (as well as `fusedlasso1d` and `fusedlasso2d`) also takes an optional argument `X` in the case that a non-identity predictor matrix X should be included. As with the `genlasso`, the predictor matrix X should have full column rank; however, this is not check here, for efficiency.

3.3. Sparse fused lasso and soft-thresholding

A common variant of the fused lasso is employs an additional ℓ_1 penalty on the coefficients themselves; e.g., the signal approximator fused lasso problem in (8) can be extended as in

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 + \lambda \sum_{(i,j) \in E} |\beta_i - \beta_j| + \gamma \cdot \lambda \sum_{i=1}^{n} |\beta_i|. \tag{9}$$

Here $\gamma \geq 0$ is another parameter that controls the ratio between the fusion and sparsity penalty terms. Note that (9) also fits into the generalized lasso framework, as it simply concatenates (a multiple of) the identity matrix to the rows of a fused lasso penalty matrix. For a single fixed value of γ, the solution path of the sparse fused lasso problem (9) can be computed using by setting the `gamma` argument in `fusedlasso` (or `fusedlasso1d`, `fusedlasso2d`). (Additionally including a non-identity predictor matrix poses no problems, and is done by setting the `X` argument, as before.)

When $X = I$, a particularly simple relationship exists between the solutions of the problem (8) and its sparse variant (9): at any λ, the solution of (9) is simply given by soft-thresholding the solution of (8) by an amount $\gamma \cdot \lambda$, as shown in Friedman, Hastie, Hoefling, and Tibshirani (2007). If the solution path of (9) is desired at several levels of γ (or even at a single nonzero level, actually) it is more efficient to solve the problem with $\gamma = 0$ and then soft-threshold to yield the solutions.

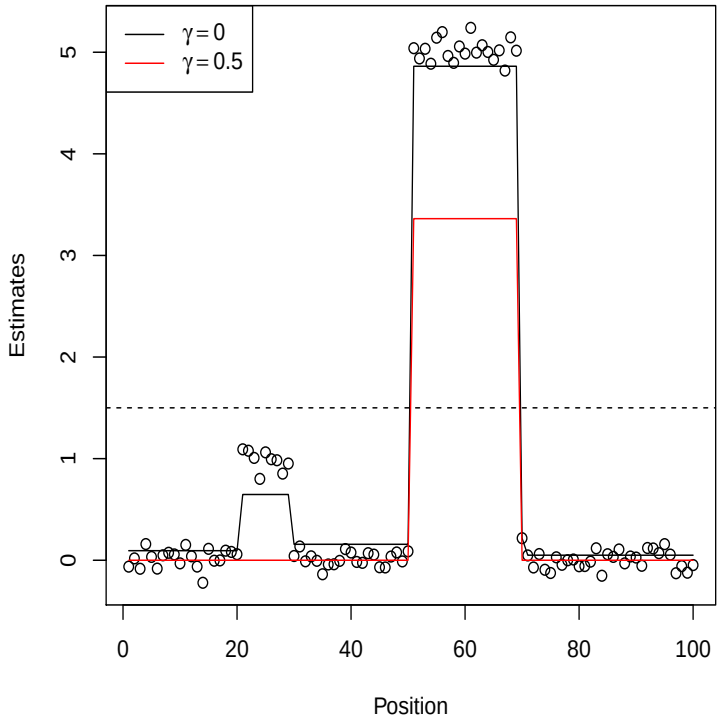
We demonstrate this by revisiting the 1d fused lasso example of Section 3.1:

```
> set.seed(1)
> n = 100
> i = 1:n
> y = (i > 20 & i < 30) + 5*(i > 50 & i < 70) +
+   rnorm(n, sd=0.1)
> out = fusedlasso1d(y)
> beta1 = coef(out, lambda=1.5)$beta
```

To compute the solution at $\lambda = 1.5$ and $\gamma = 1$, we could call `fusedlasso1d` with `gamma=1`, or simply soft-threshold:

```
> beta2 = softthresh(out, lambda=1.5, gamma=1)
```

The effect of soft-thresholding is apparent when looking at the original and thresholded estimates (with the dashed line showing the thresholding level $\gamma \cdot \lambda = 1.5$):



4. Trend filtering

4.1. Beyond piecewise constant fits

Like the 1d fused lasso, trend filtering in the signal approximator case $X = I$ assumes that the data $y = (y_1, \ldots y_n) \in \mathbb{R}^n$ is meaningfully ordered from 1 to n, and fits a piecewise polynomial of a specified degree. For example, linear trend filtering (with $X = I$) solves the following minimization problem:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^n}{\operatorname{argmin}} \; \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_i)^2 + \lambda \sum_{i=1}^{n-2} |\beta_i - 2\beta_{i+1} + \beta_{i+1}|. \tag{10}$$

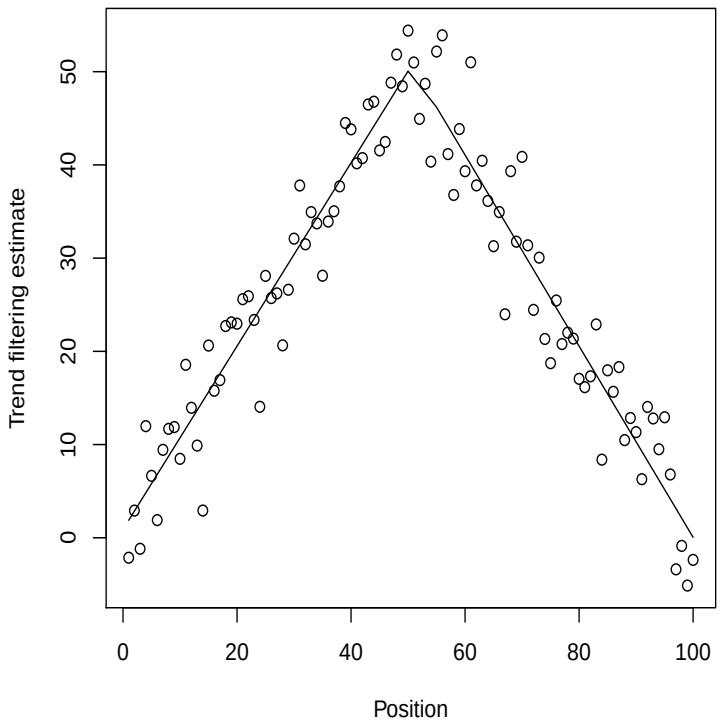
Notice that here the discrete second deriative is penalized, as opposed to the discrete first derivative in the 1d fused lasso criterion (6). Quadratic and cubic trend filtering are defined similarly, by penalizing the discrete third and fourth derivative, respectively. (In this light, we can think of the 1d fused lasso as constant or zeroth order trend filtering.)

We generate a data set with a piecewise linear mean:

```
> set.seed(1)
> n = 100
> y = 50 - abs(1:n-n/2) + rnorm(n, sd=5)
```

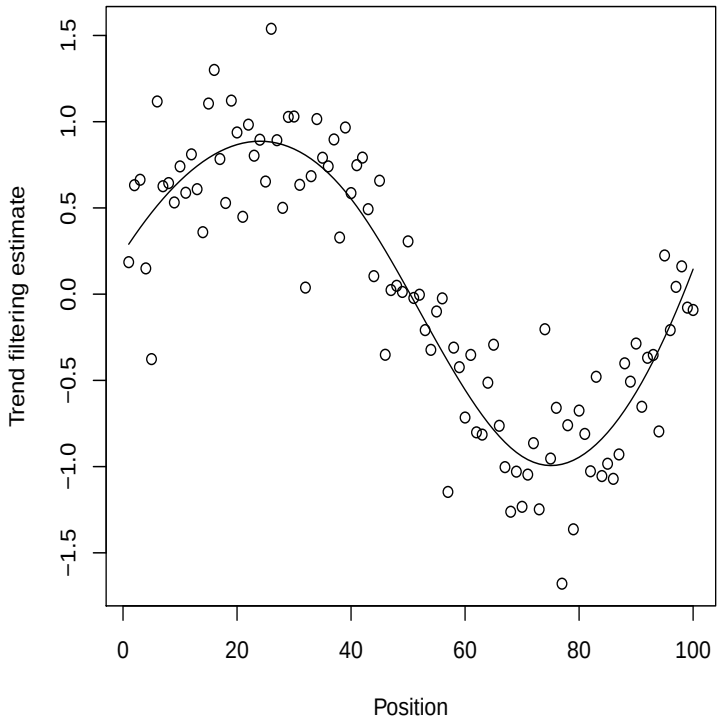
We use the function `trendfilter` to fit trend filtering model of order 1 (i.e., linear). It has the same plotting functionality as the `fusedlasso1d` function:

```
> out = trendfilter(y, ord=1)
> plot(out, lambda=n)
```



Higher order plots are just as easy, as shown by the following cubic fit to data generated with a mean given by a sine curve:

```
> n = 100
> y = sin(1:n/n*2*pi) + rnorm(n, sd=0.3)
> out = trendfilter(y, ord=3)
> plot(out, lambda=n)
```



Note that trend filtering fits can also be produced by using the `genlasso` with `D=getDtf(n,k)`, where `getDtf` is a convenience function to construct a trend filtering penalty matrix of a specified order (`k=1` being linear). However, this is not recommended because the specialized implementation `trendfilter` is significantly more efficient and stable. Trend filtering can also be fit with a non-identity predictor; simply use the `X` argument.

4.2. Cross-validation

For automatic choice of λ in trend filtering problems there is an easy to use, pre-built cross-validation function. Here, we again generate some data which that has a piecewise constant mean, and fit a piecewise constant model:

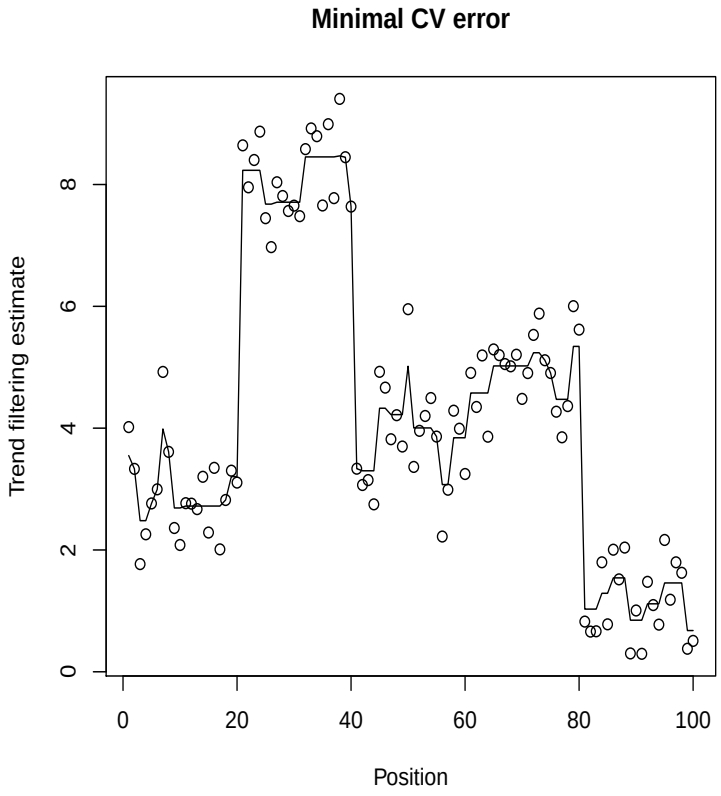
```
> set.seed(1)
> n = 100
> y = rep(sample(1:8,5), each=n/5) + rnorm(n, sd=0.8)
> out = trendfilter(y, ord=0)
```

(Here we demonstrate that `trendfilter` with `ord=0` is equivalent to calling `fusedlasso`.) Now we use the function `cv.trendfilter` to perform k-fold cross-validation in to choose λ. This places every kth point in the same fold, so the folds are non-random and calling `cv.trendfilter` twice will yield the same result. The default is $k = 10$ folds.

```
> cv = cv.trendfilter(out)
```

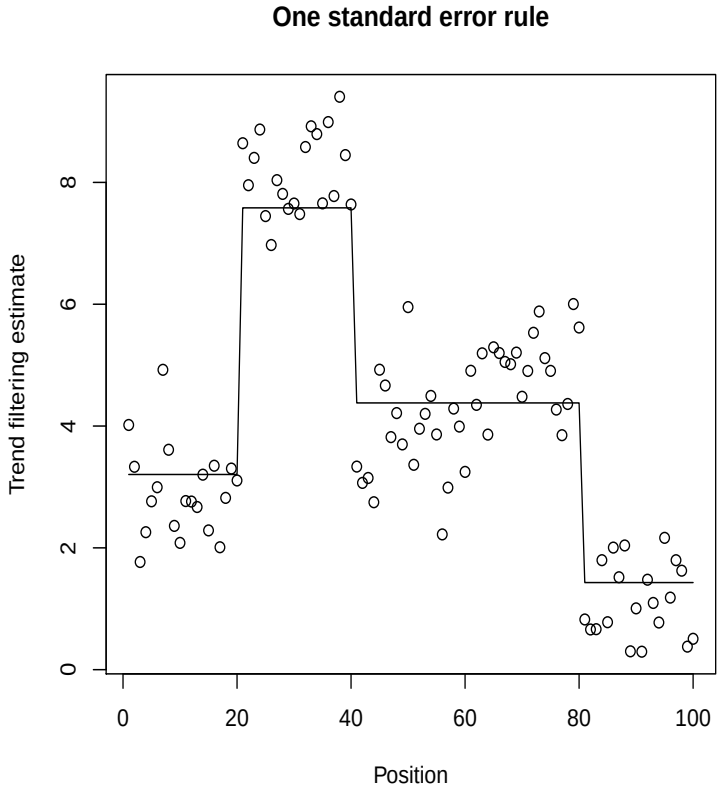
Now we plot the estimate at the value of λ the minimizes the cross-validated error:

```
> plot(out, lambda=cv$lambda.min, main="Minimal CV error")
```



Note that there is a considerable amount of noise here and this does not provide a good estimate of the change points. Using the value of λ chosen by the one standard error rule produces a more regularized estimate:

```
> plot(out, lambda=cv$lambda.1se, main="One standard error rule")
```



This gives a cleaner estimate of the change points.

References

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. URL http://igraph.sf.net.

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

Tibshirani RJ, Taylor J (2011). "The Solution Path of the Generalized Lasso." *Annals of Statistics*, **39**(3), 1335–1371.

Affiliation:

Ryan Tibshirani
Department of Statistics
Carnegie Mellon University
Email: ryantibs@cmu.edu

Taylor B. Arnold
Department of Statistics
Yale University
Email: taylor.arnold@yale.edu