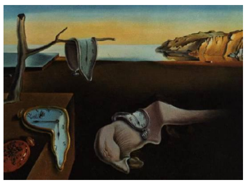

Journal of Statistical Software

August 2024, Volume 110, Issue 6. *doi: 10.18637/jss.v110.i06*

sparsegl: An R Package for Estimating Sparse Group Lasso

Xiaoxuan Liang
University of British Columbia

Aaron Cohen
Indiana University

Anibal Sólon Heinsfeld
The University of Texas at Austin

Franco Pestilli
The University of Texas at Austin

Daniel J. McDonald
University of British Columbia

Abstract

The sparse group lasso is a high-dimensional regression technique that is useful for problems whose predictors have a naturally grouped structure and where sparsity is encouraged at both the group and individual predictor level. In this paper we discuss a new R package for computing such regularized models. The intention is to provide highly optimized solution routines enabling analysis of very large datasets, especially in the context of sparse design matrices.

Keywords: generalized linear model, regularization, sequential strong rule, sparse matrix.

1. Introduction

Regularized linear models are now ubiquitous tools for prediction and, increasingly, inference. When solving such high-dimensional learning problems, adding regularization helps to reduce the chances of overfitting and improve the model performance on unseen data. Sparsity inducing ℓ_1-type penalties such as the lasso (Tibshirani 1996) or the Dantzig selector (Candes and Tao 2007) perform both variable selection and shrinkage, resulting in near-optimal statistical properties. The group lasso (Yuan and Lin 2006) modifies the regularizer, replacing the ℓ_1 penalty with a group-wise sum of ℓ_2 norms. When covariates have natural groupings, such as with genomics data or one-hot encoded factors, this group penalty is preferable, because the resulting estimate will include or exclude entire groups of covariates. To simultaneously attain sparsity at both group and individual feature levels, Simon, Friedman, Hastie, and Tibshirani

(2013) proposed the sparse group-lasso, a convex combination of the ℓ_1 lasso penalty and the group lasso penalty.

While a number of packages exist for solving the sparse group lasso, our R (R Core Team 2024) implementation in **sparsegl** (McDonald, Liang, Solón Heinsfeld, and Cohen 2024) is designed to be fast, especially in the case of large, sparse covariate matrices. This package focuses on finding the optimal solutions to sparse group-lasso penalized learning problems at a sequence of regularization parameters, implements risk estimators in an effort to avoid cross validation if necessary, leverages a fast, compiled Fortran implementation, avoids extraneous data copies, and undertakes a number of additional computational efficiency improvements. In R, there are already excellent implementations of sparse group lasso and group lasso, namely **SGL** (Simon, Friedman, Hastie, and Tibshirani 2019), **gglasso** (Yang, Zou, and Bhatnagar 2024; Yang and Zou 2015), and **biglasso** (Zeng, Wang, Peter, and Breheny 2024; Zeng and Breheny 2020). Of these, only **SGL** employs the additional ℓ_1 sparsity-inducing penalty. However, it has a number of drawbacks that result in much slower performance, even on small data. One major reason is the omission of so-called "strong rules" (Tibshirani *et al.* 2012) that help coordinate descent algorithms to avoid many of the groups which will turn out to have zero coefficient estimates. The **gglasso** and **biglasso** packages are both computationally fast. The former incorporates the strong rule, and the latter involves a hybrid safe-strong rule along with scalable storage and a parallel implementation in C++ (Stroustrup 2013) and R that allows for data that exceeds the size of installed random access memory. Unfortunately, neither allows within-group sparsity (i.e., they perform group lasso, not sparse group lasso). Thus, the estimated coefficients produced by these packages will have some active groups and some inactive groups, but within an active group, generally all the coefficients will be nonzero.

In Python (Van Rossum *et al.* 2011), **asgl** (Civieta, Aguilera-Morillo, and Lillo 2021) implements adaptive sparse group-lasso, which flexibly adjusts the weights in the penalization on the groups of features. Additionally it incorporates quantile loss. As with the other packages mentioned above, it can also solve the special cases (lasso and group lasso). However, for all optimization problems, it directly uses **CVXPY** (Diamond and Boyd 2016; Agrawal, Verschueren, Diamond, and Boyd 2018), a general purpose optimizer, without strong rules or other tricks to relate solutions to each other across values of the tuning parameter.[1]

Our contribution, then, is to provide a package that performs sparse group lasso and is faster than existing implementations. In particular, **sparsegl** has the following benefits:

- Performs Gaussian and logistic regression using fast, compiled code.
- Allows arbitrary generalized linear models using R's '`family`' class, though with slightly less efficiency.
- Allows for interval constraints and differential weights on the coefficients.
- Accommodates a sparse design matrix and returns the coefficient estimates in a sparse matrix.
- Uses strong rules (and active set iteration) for fast computation along a sequence of tuning parameters.

[1]A similar implementation could be achieved in R using **CVXR** (Fu, Narasimhan, and Boyd 2020; Fu, Narasimhan, Kang, Diamond, and Miller 2024).

	Regression & classification	Within group sparsity	Sparse matrices	Strong rules	Avoids copies	Interval constraints
sparsegl	✓	✓	✓	✓	✓	✓
gglasso	✓			✓		
SGL	✓	✓				

Table 1: This table summarizes the features available in **sparsegl** and related R packages.

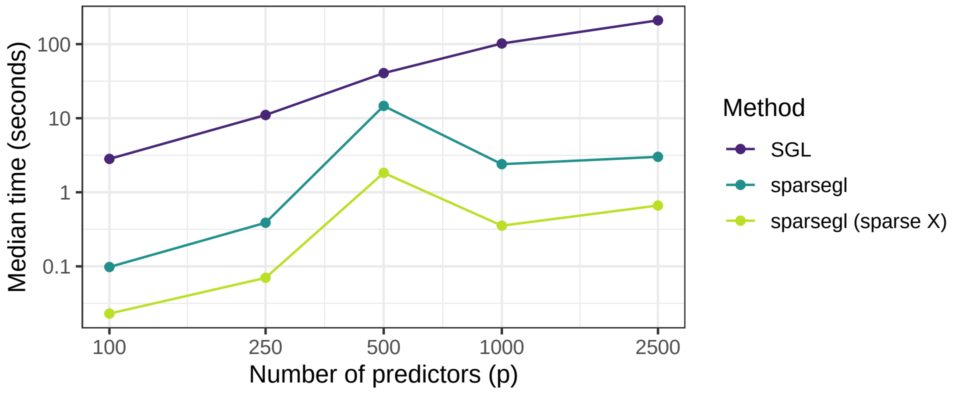

Figure 1: This figure shows the time required to compute sparse group lasso solutions across a number of different problem sizes. In all cases, we use $n = 500$ observations and 100 values of the tuning parameter λ. The median is taken across 5 replications for each method and problem size. Note that both axes are on the log scale.

- Uses **dotCall64** to interface with low-level Fortran functions and avoid unnecessary copying as well as allow for 64-bit integers (see Gerber, Moesinger, and Furrer 2017, 2018).
- Provides information criteria as risk estimators (AIC/BIC/GCV) in addition to cross validation.

A comparison of features of this and related R packages is shown in Table 1. Figure 1 compares the speed of the dense and sparse implementations in **sparsegl** with **SGL** across a number of different problem sizes, finding speedups of 1.5 to 2.5 orders of magnitude.

In Section 2, we describe the algorithmic implementation in detail, paying particular attention to the strong rule. In Section 3, we show how to use the package, running through an example with simulated data. Section 4 demonstrates many of the unique features of **sparsegl** in two applications. We summarize our contributions in Section 5.

2. Methodology, estimation and prediction

Given a sample of n observations of a univariate response y_i and a corresponding vector of features $\mathbf{x}_i \in \mathbb{R}^p$, the standard linear regression setup has

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \sigma\epsilon_i, \quad i = 1, \ldots, n, \tag{1}$$

where ϵ_i is independent standard Gaussian noise and $\sigma > 0$. While ordinary least squares estimates the coefficient vector $\boldsymbol{\beta}$ by solving $\min_\beta \frac{1}{2n}\sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2$, this method tends to behave poorly if $p \gg n$. In what follows, we will write $\mathbf{y} = (y_1, \dots, y_n)$ and let $\mathbf{X}$ be the row-wise concatenation of $\mathbf{x}_1^\top, \dots, \mathbf{x}_n^\top$.

The lasso adds an ℓ_1 penalty to the optimization problem:

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} = \min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\}, \tag{2}$$

where $\|\cdot\|_2$ is the Euclidean (ℓ_2) norm and $\|\cdot\|_1$ is the ℓ_1 norm. The benefit of this penalty is that it tends to allow only a subset of coefficient estimates to be nonzero, hence performing variable selection. Here, λ is a hyperparameter that trades fidelity to the data—small λ emphasizes minimization of the squared-error—with desirable regularization that selects a subset of variables and improves prediction accuracy.

A variant of this, the group lasso (Yuan and Lin 2006) is appropriate when there is a natural grouping structure for the features. That is, we assume that both the design matrix $\mathbf{X}$ and the corresponding vector of coefficients can be partitioned into interpretable non-overlapping groups, and, by analogy with lasso regression, only a few of the groups are active, i.e., have nonzero coefficients. The group lasso thus performs regularization that has the effect of discarding groups of predictors rather than the predictors themselves:

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \|\mathbf{y} - \sum_{g=1}^{G} \mathbf{X}^{(g)} \boldsymbol{\beta}^{(g)}\|_2^2 + \lambda \sum_{g=1}^{G} \sqrt{w_g} \|\boldsymbol{\beta}^{(g)}\|_2 \right\}. \tag{3}$$

Grouping may occur naturally—say with the inclusion of many categorical predictors, groups of genes, or brain regions—or may be a design choice using additive models and basis expansions. Note that in Equation 3, the grouping structure is explicitly stated: The vector of coefficients, $\boldsymbol{\beta}$, is thought of as a concatenation of the coefficient subvectors of the various groups $\boldsymbol{\beta}^{(g)}$, and similarly the data matrix $\mathbf{X}$ is the concatenation of submatrices, each submatrix $\mathbf{X}^{(g)}$ being composed of the columns that correspond to that particular group. Thus the first part of the equation, $\mathbf{y} - \sum_{g=1}^{G} \mathbf{X}^{(g)} \boldsymbol{\beta}^{(g)}$, is identical to the more simply-written equation $\mathbf{y} - \mathbf{X}\boldsymbol{\beta}$, but the notation serves to emphasize the partitioning.

However, the penalty, $\sum_{g=1}^{G} \sqrt{w_g} \|\boldsymbol{\beta}^{(g)}\|_2$, is different from the corresponding part in Equation 2, using instead the sum of the (non-squared) ℓ_2-norms of the coefficient vectors of the various groups. It is the non-differentiability of this expression at $\mathbf{0} \in \mathbb{R}^{|g|}$ (with $|g|$ meaning the size of group g) that accounts for the group-discarding property of the solution, similar to the way that the non-differentiability the absolute value at 0 is responsible for discarding individual predictors in the lasso.

As with Equation 2, there is only a single tuning parameter λ, whose value determines the strength of regularization. Within the second summation are the relative weights of the groups, w_g. These are often taken to be the size of the corresponding group. For simplicity, this notation is suppressed below where the meaning is clear.

Finally, in a group-structured problem as above, it may be desirable to enforce sparsity not only among the groups but also within the groups. The sparse group lasso (Simon *et al.* 2013) does this by combining the penalties in Equations 2 and 3:

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + (1-\alpha)\lambda \sum_{g=1}^{G} \|\boldsymbol{\beta}^{(g)}\|_2 + \alpha\lambda \sum_{g=1}^{G} \|\boldsymbol{\beta}^{(g)}\|_1 \right\}. \tag{4}$$

There is now a second tuning parameter α, which controls the relative emphasis of intra- versus inter-group sparsity in the coefficient estimates. In Equation 4, we have chosen to write the group dependence explicitly in the ℓ_1 component of the penalty, but note that $\sum_{g=1}^{G} \|\boldsymbol{\beta}^{(g)}\|_1 = \|\boldsymbol{\beta}\|_1$. Similar to the weights w_g in the group component, **sparsegl** also allows individual predictor weights in the ℓ_1 component, $\sum_{j=1}^{p} \omega_j |\beta_j|$, but we suppress this generality for clarity, setting $\omega_j = 1$ for all j below.

2.1. The group-wise majorization-minimization algorithm

There is no closed-form solution to the optimization problem in Equation 4, so we require a numerical procedure. Because the problem is convex, a variety of methods may be used. The general framework for our algorithm is the same as the majorized block-wise coordinate descent algorithm developed in (Yang and Zou 2015; Simon *et al.* 2013). What this means is that, for a fixed value of λ, we loop over the groups and update only those coefficients while holding all other groups constant. In particular, instead of using the exact Hessian to determine the step size and direction in every update step, we update according to a simpler expression that majorizes the objective.

For the rest of this section, we describe this majorization algorithm, focusing on a particular group g and holding the coefficients for all other groups fixed. We note here that, because the loss function in Equation 4 is differentiable and the penalty terms are convex and separable (i.e., they can be decomposed into a sum of functions each only involving a single group), this block coordinate descent algorithm is guaranteed to converge to a global optimum (Tseng 2001).

To begin with, we introduce some notation. Let

$$\mathbf{r}_{(-g)} = \mathbf{y} - \sum_{k \neq g} \mathbf{X}^{(k)} \boldsymbol{\beta}^{(k)}$$

be the partial residual without group g where all the group fits besides that of group g are subtracted from $\mathbf{y}$. With all other groups held fixed, we aim to solve:

$$\min_{\boldsymbol{\beta}^{(g)}} \frac{1}{2n} \|\mathbf{r}_{(-g)} - \mathbf{X}^{(g)} \boldsymbol{\beta}^{(g)}\|_2^2 + (1-\alpha)\lambda \|\boldsymbol{\beta}^{(g)}\|_2 + \alpha\lambda \|\boldsymbol{\beta}^{(g)}\|_1. \tag{5}$$

In what follows, we will suppress the (g) notation, with the understanding that we are really referring to only the g^{th} group of the coefficient vector and the partial residual $\mathbf{r}_{(-g)}$. We will also define, the unpenalized loss function

$$\ell(\boldsymbol{\beta}) = \frac{1}{2n} \|\mathbf{r} - \mathbf{X}\boldsymbol{\beta}\|_2^2,$$

so that our objective function for the g^{th} group becomes $\ell(\boldsymbol{\beta}) + (1-\alpha)\lambda \left\|\boldsymbol{\beta}\right\|_2 + \alpha\lambda \left\|\boldsymbol{\beta}\right\|_1$, and we are interested in finding an optimal value, $\hat{\boldsymbol{\beta}}$. This enables the procedure to generalize easily to logistic loss or, in principle, other exponential families.

Any global minimum must satisfy a subgradient equation, similar to a first-derivative test for an optimum, except that $\|\cdot\|_2$ and $\|\cdot\|_1$ are non-differentiable at $\mathbf{0}$. For Equation 5 above, taking the subdifferential and setting equal to zero gives us the following first-order condition:

$$\nabla \ell(\boldsymbol{\beta}) = (1-\alpha)\lambda \mathbf{u} + \alpha\lambda \mathbf{v}, \tag{6}$$

where $\mathbf{u}$ is the subgradient of $\|\boldsymbol{\beta}\|_2$ and $\mathbf{v}$ is the subgradient of $\|\boldsymbol{\beta}\|_1$. The first is defined to be $\boldsymbol{\beta}/\|\boldsymbol{\beta}\|_2$ if $\boldsymbol{\beta}$ is a nonzero vector, and is any vector in the set $\{\mathbf{u} : \|\mathbf{u}\|_2 \leq 1\}$ otherwise; the second, $\mathbf{v}$, is defined coordinate-wise as $v_j = \text{sign}(\beta_j)$ if $v_j \neq 0$, and is any value $v_j \in \{v_j : |v_j| \leq 1\}$ otherwise.

For the Gaussian case, the unpenalized loss $\ell(\boldsymbol{\beta})$ is a quadratic function in $\boldsymbol{\beta}$, so it is equal to its second order Taylor expansion about any point $\boldsymbol{\beta}_0$ in the parameter space. We thus start with the following equality for any given $\boldsymbol{\beta}_0$ (recalling that $\boldsymbol{\beta}_0$ here is only for group g):

$$\forall \boldsymbol{\beta},\ \boldsymbol{\beta}_0, \quad \ell(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}_0) + (\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \nabla \ell(\boldsymbol{\beta}_0) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \mathbf{H}(\boldsymbol{\beta} - \boldsymbol{\beta}_0),$$

where the gradient $\nabla \ell$ is the first total derivative of ℓ (evaluated at $\boldsymbol{\beta}_0$) and $\mathbf{H}$, the Hessian, is the second total derivative. For $\ell(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{r} - \mathbf{X}\boldsymbol{\beta}\|_2^2$, a short computation shows that the Hessian is $\mathbf{H} = \frac{1}{n}\mathbf{X}^\top \mathbf{X}$.

For the large-scale problems motivating this work, the matrix $\mathbf{X}$ is large, so computing $\mathbf{X}^\top \mathbf{X}$, storing it in memory, or inverting it, is computationally prohibitive. Instead, we replace this matrix with a simpler one, $t^{-1}\mathbf{I}$, a diagonal matrix with the value of t selected to be such that this dominates the Hessian (in the sense that $t^{-1}\mathbf{I} - \mathbf{H}$ is positive definite). For our algorithm we choose the largest eigenvalue of the Hessian and use that for t^{-1}. Note that this eigenvalue must be computed for each group $g \in G$, but this computation is relatively simple using the power method or other techniques as implemented with **RSpectra** (Qiu and Mei 2024). This upper bound leads to the following inequality:

$$\forall \boldsymbol{\beta},\ \boldsymbol{\beta}_0, \quad \ell(\boldsymbol{\beta}) \leq \ell(\boldsymbol{\beta}_0) + (\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \nabla \ell(\boldsymbol{\beta}_0) + \frac{1}{2t}(\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top (\boldsymbol{\beta} - \boldsymbol{\beta}_0). \tag{7}$$

Replacing the loss function in the original minimization problem in Equation 5 with the right-hand side of Equation 7 leads to a majorized version of the original problem

$$\ell(\boldsymbol{\beta}_0) + (\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \nabla \ell(\boldsymbol{\beta}_0) + \frac{1}{2t}\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}\|_2^2 + (1-\alpha)\lambda \|\boldsymbol{\beta}\|_2 + \alpha\lambda \|\boldsymbol{\beta}\|_1, \tag{8}$$

which no longer involves operations with the Hessian matrix.

As before, the optimal value for Equation 8 is determined by its subgradient equation, similar to that of Equation 6:

$$\frac{1}{t}\Big(\boldsymbol{\beta} - (\boldsymbol{\beta}_0 - t\nabla \ell(\boldsymbol{\beta}_0))\Big) + (1-\alpha)\lambda \mathbf{u} + \alpha\lambda \mathbf{v} = \mathbf{0},$$

with $\mathbf{u}$ and $\mathbf{v}$ as defined above. Solving this for $\boldsymbol{\beta}$ in terms of $\boldsymbol{\beta}_0$ results in the following expression:

$$\hat{\boldsymbol{\beta}} = U(\boldsymbol{\beta}_0) = \left(1 - \frac{t(1-\alpha)\lambda}{\|S(\boldsymbol{\beta}_0 - t\nabla \ell(\boldsymbol{\beta}_0),\ t\alpha\lambda)\|_2}\right)_+ S(\boldsymbol{\beta}_0 - t\nabla \ell(\boldsymbol{\beta}_0),\ t\alpha\lambda), \tag{9}$$

where $(z)_+ = \max\{z,\ 0\}$ and S is the coordinate-wise soft threshold operator, on a vector $\boldsymbol{\gamma}$ and scalar b,

$$(S(\boldsymbol{\gamma}, b))_j = \text{sign}(\gamma_j)(|\gamma_j| - b)_+,$$

i.e., for each coordinate in the vector, it shrinks that coordinate in magnitude by the amount b, and sets it to zero if the magnitude of that coordinate was smaller than b to begin with. It is this soft-thresholding operation that encourages within-group sparsity.

Algorithm 1 Sparse group lasso solution for fixed λ, regression version

```
1: Input: $\lambda \geq 0$, $\alpha \in [0,\ 1]$, set of groups $\mathcal{G}$, initial coefficients $\boldsymbol{\beta}$, $\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$
2: while Not converged do
3:     for $g \in \mathcal{G}$ do
4:         Update $\boldsymbol{\beta}^{(g)} = \left(1 - \frac{t(1-\alpha)\lambda}{\|S\left(\boldsymbol{\beta}^{(g)} - t\nabla\ell(\boldsymbol{\beta}^{(g)}),\ t\alpha\lambda\right)\|_2}\right)_+ S\left(\boldsymbol{\beta}^{(g)} - t\nabla\ell(\boldsymbol{\beta}^{(g)}),\ t\alpha\lambda\right)$.
5:         Update $\mathbf{r} = \mathbf{r} - \mathbf{X}^{(g)}\boldsymbol{\beta}^{(g)}$.
6:     end for
7: end while
8: return $\boldsymbol{\beta}$
```

An examination of Equation 9 shows that it is possible for the entire group to be set to zero (made inactive) due to the (hard) threshold operator $(\cdot)_+$ in the first part of the expression. It is also possible for individual components of $\boldsymbol{\beta}^{(b)}$ to be zeroed out by the coordinate-wise (soft) threshold operator S. Therefore, performing this update step tends to enforce coefficient sparsity at both the group- and individual-level.

Above, we have focused on the Gaussian linear model with $\ell(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{r} - \mathbf{X}\boldsymbol{\beta}\|_2^2$, $\nabla\ell(\boldsymbol{\beta}) = -\frac{1}{n}\mathbf{X}^\top(\mathbf{r} - \mathbf{X}\boldsymbol{\beta})$, and $\mathbf{H} \preceq t^{-1}\mathbf{I}$. In the case of logistic regression, we use exactly the same procedure but with $\ell(\boldsymbol{\beta}) = \frac{1}{n}\sum_i \log(1 + \exp\{-r_i\mathbf{x}_i^\top\boldsymbol{\beta}\})$, $\nabla\ell(\boldsymbol{\beta}) = -\frac{1}{n}\sum_i y_i\mathbf{x}_i^\top(1 + \exp\{-r_i\mathbf{x}_i^\top\boldsymbol{\beta}\})^{-1}$, and $\mathbf{H}(\boldsymbol{\beta}) \preceq 4t^{-1}\mathbf{I}$. This procedure is explicitly stated in Algorithm 1. For other exponential families (for example, Poisson, Gamma, or Probit regression), we provide functionality to pass an R '`family`' object. These will generally be much slower than the built-in families described above because they require iteratively reweighted least squares as an outer loop combined with inner majorization-minimization iterations as described here.

While the procedure described so far solves Equation 4 for fixed choices of λ and α, the data analyst does not typically know these ahead of time. Rather, we would like to solve the problem for a collection of values of λ (and perhaps α as well). It turns out that the structure of this optimization problem allows for some heuristics that can perform this sequential optimization with a minimum of additional computational resources, in some cases, solving Equation 4 faster for a sequence of values $\lambda_m \in \{\lambda_1, \ldots, \lambda_M\}$ than for a single choice (Tibshirani *et al.* 2012). We describe our implementation of this procedure next.

2.2. Sequential strong rule, KKT conditions, and active set iteration

For any fixed value of λ, many groups of coefficient estimates will end up being equal to zero. If, somehow, we knew *which* groups, we could completely avoid visiting them in the block-wise coordinate descent updates, and therefore avoid calculating Equation 9 for those groups. This would significantly speed up computations.

Re-examining Equation 6, we can see that the first order condition implies that, for each group g, any solution must satisfy

$$\|S(\nabla\ell(\boldsymbol{\beta}_g),\ \lambda\alpha)\|_2 \leq (1-\alpha)\lambda. \tag{10}$$

This is because, as $\mathbf{u}$ is the subgradient of $\|\boldsymbol{\beta}_g\|_2$, $\|\mathbf{u}\|_2 \leq 1$. Furthermore, if $\|\mathbf{u}\|_2 < 1$, then $\hat{\boldsymbol{\beta}}_j = \mathbf{0}$. In the previous section, we used the sufficiency of the Karush-Kuhn-Tucker (KKT)

Algorithm 2 Sequential strong rule and Majorization Minimization in **sparsegl**

```
 1: Input: $\mathbf{X}$, $\mathbf{y}$, $\mathcal{G}$, and $\{\lambda_1, \ldots, \lambda_M\}$.   Output: $\hat{\boldsymbol{\beta}}$.
 2: Initialize: $\mathcal{A} = \mathcal{S} = \varnothing$, $\hat{\boldsymbol{\beta}} = 0$.
 3: for $m = 1$ to $M$ do
 4:     Update $\mathcal{S} \leftarrow \mathcal{S} \bigcup \left\{ g \in \mathcal{S}^c : \|S(\nabla \ell(\hat{\boldsymbol{\beta}}_g),\ t\alpha\lambda_m)\|_2 > t(1-\alpha)\lambda_m \right\}$.
 5:     Apply Algorithm 1 with $\mathcal{G} = \mathcal{A}$ (MM gradient update).
 6:     Update $\mathcal{A} \leftarrow \mathcal{A} \bigcup \left\{ g \in \mathcal{S} \bigcap \mathcal{A}^c : \|S(\nabla \ell(\hat{\boldsymbol{\beta}}_g),\ t\alpha\lambda_m)\|_2 > t(1-\alpha)\lambda_m \right\}$.
 7:         If there were any violations, go to to Line 5.
 8:     Update $\mathcal{A} \leftarrow \mathcal{A} \bigcup \left\{ g \in \mathcal{S}^c \bigcap \mathcal{A}^c : \|S(\nabla \ell(\hat{\boldsymbol{\beta}}_g),\ t\alpha\lambda_m)\|_2 > t(1-\alpha)\lambda_m \right\}$.
 9:         If there were any violations, go to to Line 5.
10:     Set $\mathcal{S} = \mathcal{S} \bigcup \mathcal{A}$.
11: end for
```

stationarity condition to derive a solution, while Equation 10 is the necessary version. So given a potential solution, it is easy to check its validity. Unfortunately, this is not constructive.

The sequential strong rule (Tibshirani *et al.* 2012) begins from Equation 10 and makes use of the fact that we are solving for a sequence of parameters $\{\lambda_1 > \lambda_2 > \cdots > \lambda_M\}$ rather than a single value. At each λ_m, we rely on the fact that we have already solved the problem at λ_{m-1} and use this information to quickly discard many groups of predictors. Without loss of generality, for the rest of this section, assume that the problem has been solved for λ_{m-1}.

Define $c_g(\lambda) = S(\nabla \ell(\boldsymbol{\beta}_g),\ \lambda\alpha)$. Now, we make the assumption that $c_g(\lambda)$ is $(1-\alpha)$-Lipschitz, i.e., that

$$\forall \lambda,\ \lambda' \quad \|c_g(\lambda) - c_g(\lambda')\|_2 \leq (1-\alpha)|\lambda - \lambda'|.$$

This Lipschitz assumption appears unintuitive, and in fact, is not always true, but it turns out to be useful.

By Equation 10, if we knew that $\|c_g(\lambda_m)\|_2 < (1-\alpha)\lambda_m$ then we could safely ignore it. But we already have the solution at λ_{m-1}. By the triangle inequality (first) and the Lipschitz assumption (second),

$$\|c_g(\lambda_m)\|_2 \leq \|c_g(\lambda_m) - c_g(\lambda_{m-1})\|_2 + \|c_g(\lambda_{m-1})\|_2 \leq (1-\alpha)(\lambda_{m-1} - \lambda_m) + \|c_g(\lambda_{m-1})\|_2.$$

We want to be able to assert that $(1-\alpha)(\lambda_{m-1} - \lambda_m) + \|c_g(\lambda_{m-1})\|_2 \leq (1-\alpha)\lambda_m$, allowing us to ignore group g, and this assertion holds precisely when

$$\|c_g(\lambda_{m-1})\|_2 \leq (1-\alpha)(2\lambda_m - \lambda_{m-1}).$$

Applying this logic to Equation 8 gives the sequential strong rule for the sparse group lasso:

$$\|S(\nabla \ell(\boldsymbol{\beta}_g),\ t\alpha\lambda_{m-1})\|_2 \leq t(1-\alpha)(2\lambda_m - \lambda_{m-1}). \tag{11}$$

For more details in related settings, see Tibshirani *et al.* (2012). If Equation 11 holds, then we ignore group g when solving the problem at λ_m. That is to say, when we move from λ_{m-1} to λ_m, we first check this condition using the previously computed solution for $\hat{\boldsymbol{\beta}}(\lambda_{m-1})$, and then perform block-wise coordinate descent, using only those groups that failed this inequality.

This discarding rule is fast, because it uses the previously computed solution combined with a simple inequality, and, in practice, it tends to accurately discard large numbers of groups.

However, we should reiterate that it is possible for the strong rule to fail. The Lipschitz assumption is not a guarantee. Because of this, it is critical that, after discarding some of the groups and running the algorithm on the others, the KKT condition is checked on all discarded groups. If there are no violations, then we have the solution.

To minimize gradient computations for groups that will eventually be determined to be inactive, we actually keep track of two sets: The strong set $\mathcal{S}$ and the active set $\mathcal{A}$. The active set collects all groups that have ever had non-zero coefficients at previous values of λ. We first iterate over previously active groups, then check the strong set to see if we missed any, and finally check all the remaining groups. When the number of groups is very large, this avoids onerous computations for as many groups as possible. The complete algorithm including this active set iteration is shown in Algorithm 2.

2.3. Risk estimation

For many regularized prediction methods, tuning parameter selection is largely performed with cross validation. However, cross validation can be computationally expensive when the data set is large enough that the initial fit is slow. For this reason, **sparsegl** provides information criteria as well as cross validation routines.

In the Gaussian linear regression model given by Equation 1, if σ is unknown then a general form for a family of information criteria is given by

$$\mathrm{info}(C_n, g) = \log\left(\frac{1}{n}\|\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2\right) + C_n\, g(\mathrm{df}), \tag{12}$$

where C_n depends only on n, $g : [0, \infty) \to \mathbb{R}$ is a fixed function, and the degrees of freedom (df) measures the complexity of the estimation procedure. The choices $C_n = 2/n$ or $C_n = \log(n)/n$ with $g(x) = x$ are commonly referred to as AIC (Akaike 1973) and BIC (Schwarz 1978), respectively. Additionally, generalized cross validation (Golub, Heath, and Wahba 1979, GCV) is defined as

$$\mathrm{GCV} = \frac{\frac{1}{n}\|\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2}{(1 - \mathrm{df}/n)^2}.$$

Written on the log scale, GCV takes the form of Equation 12 with $g(x) = \log(1 - x/n)$ and $C_n = -2$.

The key components for all three information criteria are the negative log likelihood and the degrees of freedom. The first is simply a function of the in-sample (training) error. On the other hand, the degrees of freedom, while simple in the unregularized linear model (it is the number of parameters), is less obvious for the sparse group lasso. In general, the degrees of freedom for any predictor $\hat{\mathbf{y}}$ of $\mathbf{y}$ is defined as (Efron 1986)

$$\mathrm{df}(\hat{\mathbf{y}}) = \frac{1}{\sigma^2}\sum_{i=1}^{n} \mathrm{Cov}(\mathbf{y}_i,\ \hat{\mathbf{y}}_i).$$

In the case of any linear predictor, $\hat{\mathbf{y}} = \mathbf{A}\mathbf{y}$ for some matrix $\mathbf{A}$, it is easy to see that $\mathrm{df} = \mathrm{tr}(\mathbf{A})$. Vaiter *et al.* (2012) gives an explicit formula for the group lasso (without intra-group sparsity), but only minor modifications are required for the sparse group lasso. We give a simplified version of their result here.

Proposition 1 (Vaiter *et al.* 2012)**.** *Suppose that for a fixed $\lambda > 0$, the active set of $\hat{\beta}$ is $\mathcal{A}$ and that $\mathbf{X}_\mathcal{A}$ is the set of columns associated to $\mathcal{A}$. Assume that $\mathbf{X}_\mathcal{A}$ has full column rank. Then,*

$$\mathrm{df} = \mathrm{tr}\left(\mathbf{X}_\mathcal{A}\left(\mathbf{X}_\mathcal{A}^\top\mathbf{X}_\mathcal{A} + \lambda\mathbf{K}\right)^{-1}\mathbf{X}_\mathcal{A}^\top\right).$$

Here, $\mathbf{K} \in \mathbb{R}^{\mathcal{A}\times\mathcal{A}}$ is a block diagonal matrix with each block corresponding to a group g having at least 1 nonzero $\hat{\beta}$. For such a group g, denote $\hat{\beta}_{g|\mathcal{A}}$ the subvector of nonzero coefficient estimates. Then

$$\mathbf{K}_g = \frac{1}{\|\hat{\boldsymbol{\beta}}_{g|\mathcal{A}}\|_2}\left(\mathbf{I} - \frac{\hat{\boldsymbol{\beta}}_{g|\mathcal{A}}\hat{\boldsymbol{\beta}}_{g|\mathcal{A}}^\top}{\|\hat{\boldsymbol{\beta}}_{g|\mathcal{A}}\|_2^2}\right).$$

As long as the number of nonzero coefficients $|\mathcal{A}|$ is reasonably small, the degrees of freedom can be efficiently calculated for each value of λ. However, this calculation is generally cubic in $|\mathcal{A}|$. In these cases, an approximation may be desired. We have found, in practice, that $\lambda\mathbf{K} \approx \mathbf{0}$ is reasonably accurate, suggesting that $\mathrm{df} \approx |\mathcal{A}|$ is also reasonable. This approximation is exact for the lasso with $\alpha = 1$ (Zou, Hastie, and Tibshirani 2007).

3. Example usage

This section provides a simple illustration of using the **sparsegl** package (McDonald *et al.* 2024) to fit the regularization path for sparse group-lasso penalized learning problems. We first examine the linear regression model when the response variable is continuous and then briefly go over the logistic regression case. The package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=sparsegl` and can be installed and loaded in the usual manner:[2]

```
R> install.packages("sparsegl")
R> library("sparsegl")
```

We first create a small simulated dataset along with a vector indicating the grouping structure.

```
R> set.seed(1010)
R> n <- 100
R> p <- 200
R> X <- matrix(rnorm(n*p), nrow = n, ncol = p)
R> beta <- c(rep(5, 5), c(5, -5, 2, 0, 0), rep(-5, 5), c(2, -3, 8, 0, 0),
+    rep(0, (p - 20)))
R> groups <- rep(1:(p / 5), each = 5)
R> eps <- rnorm(n, mean = 0, sd = 1)
R> y <- X %*% beta + eps
R> pr <- 1 / (1 + exp(-X %*% beta))
R> y0 <- rbinom(n, 1, pr)
```

The **sparsegl** package is mainly used with calls to two functions:

[2]The development version of the package is hosted at `https://github.com/dajmcdon/sparsegl` with accompanying documentation at `https://dajmcdon.github.io/sparsegl`.

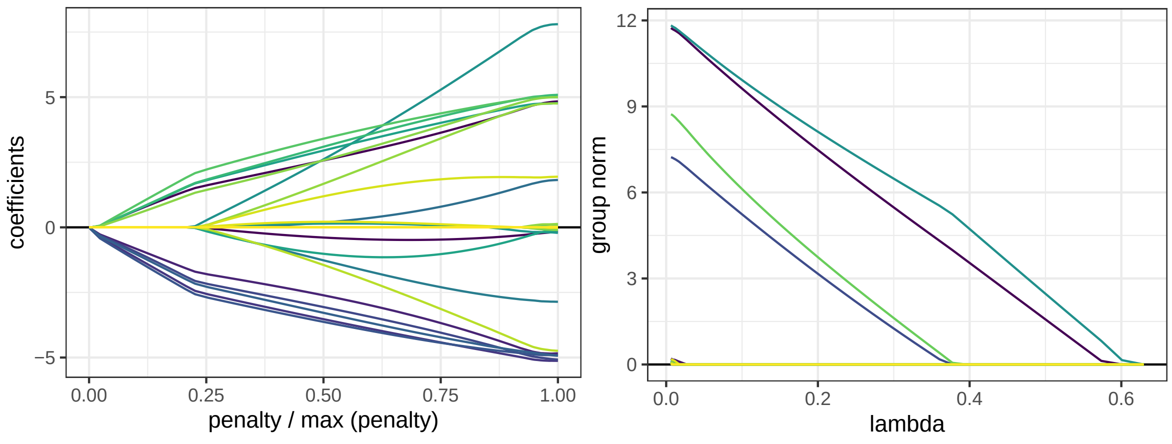

Figure 2: The left panel plots the estimated coefficients against the sparse group penalty; while the right plots the ℓ_2-norm of each group against λ.

- `sparsegl()`: Fits sparse group regularized regression and classification models;
- `cv.sparsegl()`: Repeatedly calls `sparsegl()` for the purposes of tuning parameter selection via cross validation.

The interface is intended to closely mimic that available in other R packages for regularized linear models, most notably **glmnet** (Friedman *et al.* 2023). To perform the regularization path fitting at a sequence of regularization parameters, `sparsegl()` takes as required inputs, only `x`, the design matrix, and `y`, the response vector. Other optional arguments are the grouping vector `group`, the `family` (either `"gaussian"` or `"binomial"`), a penalty vector for group weights other than the size, the relative weight of lasso penalty, desired lower or upper bounds for coefficient estimates, and other optional configurations.

```
R> fit <- sparsegl(X, y, group = groups)
```

We include a number of S3 methods for `sparsegl` typical for linear models: `plot()`, `coef()`, `predict()` and `print()`. The `plot()` function displays either the coefficients or the group norms on the y-axis against either $\{\lambda_m\}_{m=1}^M$ or the scaled penalty on the x-axis.[3] The resulting plots are shown in Figure 2.

```
R> plot(fit, y_axis = "coef", x_axis = "penalty", add_legend = FALSE)
R> plot(fit, y_axis = "group", x_axis = "lambda", add_legend = FALSE)
```

The `coef()` and `predict()` methods give the coefficients or predicted values for a new design matrix $\widetilde{\mathbf{X}}$ at the requested λ's, potentially allowing for λ values different from those used at the fitting stage.

```
R> coef(fit, s = c(0.02, 0.03))[c(1, 3, 25, 29), ]
```

```
4 x 2 sparse Matrix of class "dgCMatrix"
```

[3] We have chosen to implement plotting throughout the package using **ggplot2** (Wickham *et al.* 2024b).

```
                     s1         s2
(Intercept) -0.05189536 -0.1718156
V2           4.71082485  4.6339817
V24          .          .
V28          .          .
```

```
R> predict(fit, newx = tail(X), s = fit$lambda[2:3])
```

```
              s1        s2
 [95,] -3.894658 -2.966973
 [96,] -3.906349 -2.945468
 [97,] -4.119689 -4.241786
 [98,] -4.184564 -4.555082
 [99,] -4.175593 -4.382721
[100,] -4.071804 -4.091689
```

```
R> print(fit)
```

```


Call:  sparsegl(x = X, y = y, group = groups)

Summary of Lambda sequence:
         lambda index nnzero active_grps
Max.    0.62948     1      0           0
3rd Qu. 0.19676    26     20           4
Median  0.06443    50     19           4
1st Qu. 0.02014    75     25           5
Min.    0.00629   100    111          23
```

The `cv.sparsegl()` function implements K-fold cross validation and has a similar signature to `sparsegl()`. It allows the user to choose the number of splits and the loss function for measuring prediction or classification accuracy on the held-out sets. Here, the S3 `plot()` method displays the cross-validation curve with upper and lower confidence bounds calculated as ± 1 standard error across the folds for each λ in the regularization path (Figure 3).

```
R> cv_fit <- cv.sparsegl(X, y, groups, nfolds = 15)
R> plot(cv_fit)
```

The `coef()` and `predict()` methods work similarly to those above. The only difference being that they can additionally accept the strings `lambda.min` or `lambda.1se`, respectively the λ that minimizes the average cross validation error and the largest λ such that the cross-validated prediction error is within one standard error of the minimum.

```
R> coef(cv_fit, s = "lambda.1se")[c(1, 3, 25, 29), ]
```

```
(Intercept)          V2         V24         V28
0.004435981 4.740139458 0.000000000 0.000000000
```

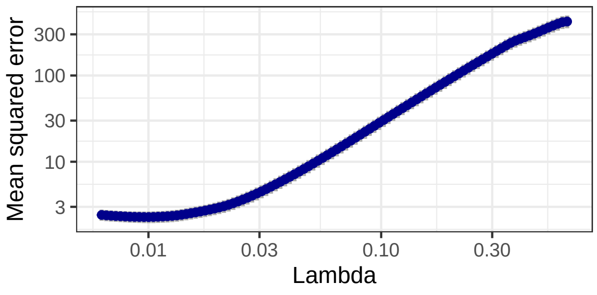

Figure 3: The cross validation estimate of out-of-sample prediction mean-squared error is displayed against the sequence of λ values.

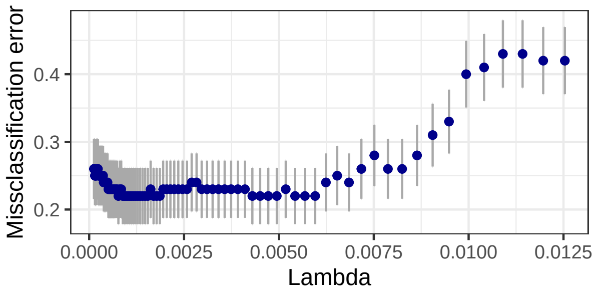

Figure 4: The cross validation estimate of out-of-sample error is displayed against the sequence of λ values. For logistic regression, misclassification error may be used.

```
R> predict(cv_fit, newx = tail(X), s = "lambda.min") |> c()

[1]  11.364725  39.985246   4.635314 -34.832413  -6.602096 -16.138344
```

For logistic regression, only a different `family` is required. Cross validation can be implemented with misclassification or deviance loss (Figure 4).

```
R> fit_logit <- sparsegl(X, y0, groups, family = "binomial")
R> cv_fit_logit <- cv.sparsegl(X, y0, groups, family = "binomial",
+    pred.loss = "misclass")
R> plot(cv_fit_logit, log_axis = "none")
```

In some cases, when computations are at a premium, cross validation my be too demanding for the purposes of risk estimation. For this reason, **sparsegl** provides an `estimate_risk()` function. It can be used to compute any of AIC (Akaike 1973), BIC (Schwarz 1978), and GCV (Golub *et al.* 1979). All three are computed by combining the log-likelihood with a penalty term for model flexibility. In addition to a fitted `sparsegl` model, `estimate_risk()` also needs the original design matrix. Because the exact degrees-of-freedom can be computationally expensive, setting `approx_df = TRUE` uses the number of non-zero coefficient estimates, which can be reasonably accurate.

```
R> er <- estimate_risk(fit, X)
```

In this simulation, the λ that minimizes AIC is 0.013 while the CV minimizer is 0.01. The estimated risk curves are plotted against λ in Figure 5.

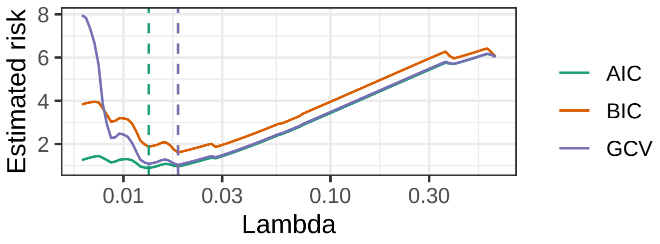

Figure 5: For Gaussian loss, AIC, BIC, and GCV (solid lines) along with their minima (vertical dashed lines) can be estimated.

Additional documentation and examples are provided on the package website at `https://dajmcdon.github.io/sparsegl`.

4. Applications

We examine two applications for which sparse group lasso is a natural estimator. The first uses data regarding COVID-19 and trustworthiness of information sources, which is included in the package. The second uses a very large though sparse data set from neuroimaging. Finally, we briefly investigate the accuracy of **sparsegl** relative to **gglasso** and **CVXR**.

4.1. Geographic distribution of trust in experts

Two typical uses for sparse group lasso are (1) additive models where continuous predictors are expanded in a basis and (2) discrete factors as predictors. Here we demonstrate an example using both at the same time. We examine data from The Delphi Group at Carnegie Mellon University U.S. COVID-19 Trends and Impact Survey (CTIS, `https://cmu-delphi.github.io/delphi-epidata/symptom-survey/contingency-tables.html`), in partnership with Facebook. In particular, we examine the publicly available contingency table reports, which break down survey responses by age, race/ethnicity, gender, and other demographic variables of interest. The necessary data to reproduce this analysis is included in **sparsegl** as `trust_experts`.

In particular, we will focus on the "estimated percentage of respondents who trust ... to provide accurate news and information about COVID-19." This survey item is reported for a variety of different potential sources of information—personal doctors/nurses, scientists, the Centers for Disease Control (CDC), government health officials, politicians, journalists, friends, and religious leaders. In this analysis, we average the first 4, characterize the combination as "experts", and use this as the response variable in a linear model.

We regress "trust in experts" on 5 factor predictors representing month of report, state of residence, age group, race/ethnicity, and self-reported gender identity. We also include two continuous predictors: "estimated percentage of people with Covid-like illness" and "estimated percentage of people reporting Covid-like illness in their local community, including their household" to control for the amount of exposure that respondents may have been having to the pandemic. Both continuous predictors are incorporated with a B-spline basis expansion and 10 degrees-of-freedom. The result is largely similar to a generalized additive model as implemented with **mgcv** (Wood 2017, 2023). The design matrix can be created using the following code:

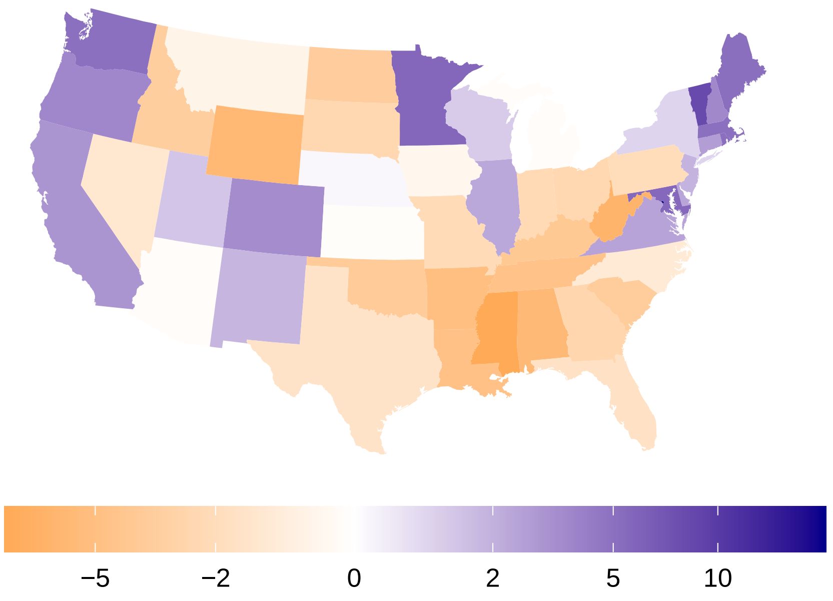

Figure 6: This figure displays estimates for each state's level of trust in experts' advice about Covid-19. The value displayed represents the change relative the U.S.-wide average.

```
R> library("dplyr")
R> library("splines")
R> data("trust_experts", package = "sparsegl")
R> trust_experts <- mutate(trust_experts, across(
+    where(is.factor),
+    ~ `attr<-`(.x, "contrasts", contr.sum(nlevels(.x), contrasts = FALSE))
+  ))
R> x <- Matrix::sparse.model.matrix(
+    trust_experts ~ 0 + region + age + gender + raceethnicity + period +
+      bs(cli, df = 10) + bs(hh_cmnty_cli, df = 10),
+    data = trust_experts, drop.unused.levels = TRUE)
```

After omitting both structural and otherwise missing data, the final model is estimated with 9759 observations on 101 predictors. As shown in the code, we did not use contrasts, fully expanding each factor in a one-hot encoding. This allows all estimated coefficients to be interpreted as deviations from the grand mean conditional on continuous predictors, which is natural. Such a formulation (along with the group penalty) is closely related to Bayesian linear models with separate Gaussian priors centered at 0 for each level of the factor. Other contrasts could be used by modifying the above, but the interpretation is more complicated. Encoded as a sparse matrix, this requires about 2.1 MB of random-access memory (RAM) to store, as opposed to 8.5 MB if it were dense. We estimated the model using `cv.sparsegl()` and default arguments. Finally, we chose λ to be the largest lambda within one standard error of the CV minimum (`lambda.1se`), resulting in a sparser model. Figure 6 displays the estimated coefficients for the state-of-residence predictors. Even controlling for age, race, gender, and the amount of circulating Covid-like illness, the United States displays strong geographic disparities when it comes to citizens' trust in scientists and other health authorities.

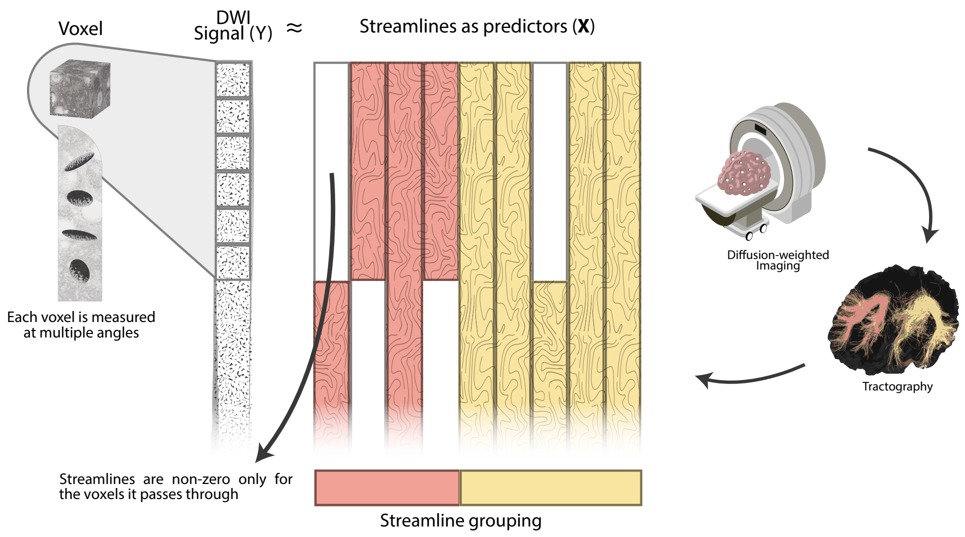
Voxel
DWI Signal (Y) ≈
Streamlines as predictors (**X**)
Each voxel is measured at multiple angles
Streamlines are non-zero only for the voxels it passes through
Streamline grouping
Diffusion-weighted Imaging
Tractography

Figure 7: This graphic illustrates how the streamlines and voxels are converted from a diffusion-weighted image to a linear model. Each voxel is measured on 90 angles, so it occupies 90 rows in the data. When a streamline (column of **X**) passes through a voxel, the values within that voxel are given by a physical model based on the direction of passage. Otherwise, if the streamline does not cross the voxel, the respective rows are zero.

4.2. Estimating white matter connectivity

Pestilli *et al.* (2014) formulated an optimization model that takes as input a set of brain connections generated using tractography algorithms and predicts the MRI diffusion signal via a linear model (Pestilli *et al.* 2014; Daducci, Dal Palù, Lemkaddem, and Thiran 2015). The Pestilli *et al.* (2014) model had no regularization, but Aminmansour *et al.* (2019) extended the problem to include group-regularization (this is an approach recently followed up by Schiavi *et al.* 2020). In this study, we re-implemented the Aminmansour *et al.* (2019) formulation using **sparsegl** to illustrate the feasibility and efficiency of the DWI modeling.

The neurological model predicts the DWI signal using the tractogram, apportioning the image signal at each voxel to the streamlines according to the measured gradient field. This preprocessing is shown pictorially Figure 7. We estimate streamline weights using sparse group lasso, allowing the amount of regularization applied to each group to be proportional to their cardinality. For our study, we used one subject from the Human Connectome Project (Van Essen *et al.* 2012). The full brain tractogram has 3M streamlines, though we used only the streamlines identified as being part of the Arcuate Fasciculus for illustration.[4]

Aminmansour *et al.* (2019) used an algorithm based on Orthogonal Matching Pursuit to estimate a related model. The data used in that study measures diffusion in 11,823 voxels using 96 magnetic field angles and attempts to reconstruct the image using the ENCODE method (Caiafa, Sporns, Saykin, and Pestilli 2017), resulting in 1057 orientations and 868 fascicles. This results in a linear regression problem with $n \approx 1M$ and $p = 868$. In comparison, our data contains 77,630 voxels measured on 90 angles, with a tractography of 10,244 streamlines.

[4]The processed data used to estimate the sparse group lasso is available at `https://doi.org/10.6084/m9.figshare.20314917`.

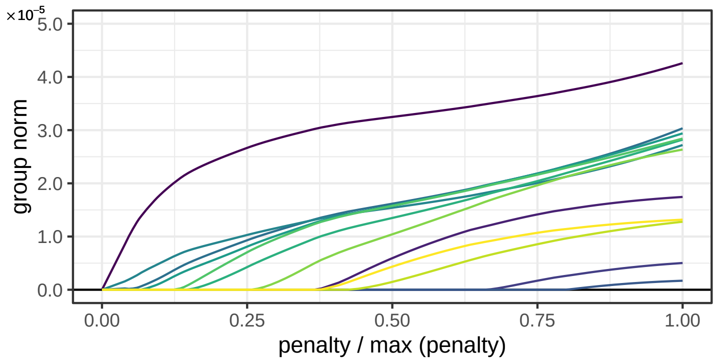

Figure 8: The group norm of the 12 groups based on neuroanatomical structure is plotted against the magnitude of the penalty.

The resulting linear model has $n \approx 6.9M$ and $p \approx 88K$, around 700× the size of the previous analysis. The design matrix would occupy over 500 GB if it were dense, but since it is only about 0.02% non-sparse, it requires 1.8 GB of memory when stored in CSC format.

Estimating the group lasso using **sparsegl** with 12 groups and 100 values of λ required a little over one minute and about 6 GB of peak memory usage on an Intel i7-11700K PC with 64 GB of RAM. The previous method required nearly a day for a single value of λ. Figure 8 displays the group norms of the 12 groups against the magnitude of the penalty.

4.3. Accuracy on synthetic problems

A small-scale simulated comparison illustrates that **sparsegl** is highly accurate in many regimes of interest. We generate synthetic data from a linear model and examine the objective function for the estimated model across a range of λ. Specifically, we first generate the predictors $\mathbf{X}$ by simulating each element x_{ij} i.i.d. standard Gaussian. We generate $n = 100$ observations and $p = \{50,\ 100,\ 150\}$ predictors. We use 10 groups in all cases, with

$$\beta = (\underbrace{1,\ldots,1}_{\text{group 1}},\ \underbrace{0,\ldots,0}_{\text{group 2}},\ \underbrace{1,\ldots,1}_{\text{group 3}},\ \ldots,\ \underbrace{0,\ldots,0}_{\text{group 10}}).$$

The groups are all of size $p/10$. The expected signal is then equal to $p/2$. We simulate the response from Equation 1 with σ chosen to produce expected signal-to-noise ratios (SNR) of $\{0.1,\ 1,\ 10\}$. We note that this design produces group sparsity, even-numbered groups have no effect on the response, but it does not produce within-group sparsity. Figure 9 shows the results for the nine combinations of these conditions across 20 values of λ determined automatically by **sparsegl** and reused for the other packages. Note that **gglasso** is optimizing a different objective function than **CVXR** or **sparsegl** which both use $\alpha = 0.2$. Despite this discrepancy, the objective values for **gglasso** and **sparsegl** are generally quite close, with **gglasso** tending to be slightly higher, as expected, but by no more than 0.75%. On the other hand **CVXR** is often much less accurate, especially for large values of λ. This divergence is due to its inability to recover exact 0 solutions, tending to instead produce estimates nearly, but not exactly, equal to zero. When $p = n = 100$, **CVXR** is actually slightly more accurate, especially for large or small λ. Inspecting the estimated $\widehat{\beta}$ in this setting, it seems to have slightly less bias on the non-zero groups, producing estimates with slightly larger magnitude.

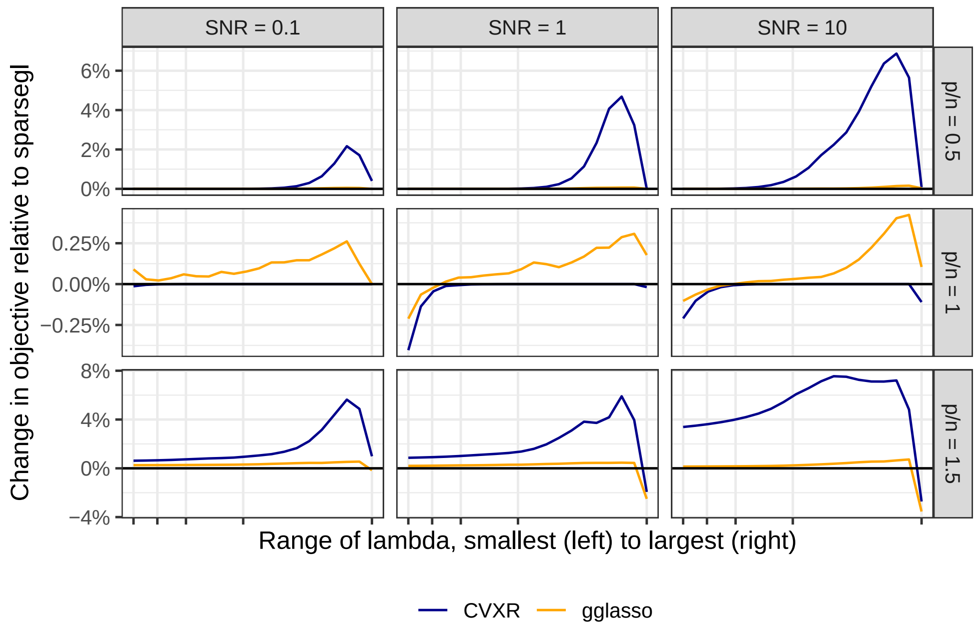

Figure 9: This figure shows the % change in the objective function for **gglasso** and **CVXR** relative to **sparsegl**. The x-axis is scaled to make the λ range comparable across conditions.

5. Discussion

We developed this package for solving sparse group lasso optimization problems using group ℓ_2 and ℓ_1 penalties with an eye toward computational efficiency for very large, potentially sparse design matrices. This efficiency is achieved through a customized Fortran implementation, avoidance of deep copy behavior, and the use of sequential strong rules for the regularization parameter. We also provide heuristics for tuning parameter selection without the need for refitting inherent in cross-validation and enable some simple extensions such as differential weights in the ℓ_1 penalty and boundary constraints on the coefficients.

Acknowledgments

In addition to the R packages described above, this manuscript and **sparsegl** benefited from **devtools** (Wickham, Hester, Chang, and Bryan 2022), **usethis** (Wickham, Bryan, Barrett, and Teucher 2024a), **testthat** (Wickham 2024), **knitr** (Xie 2024), and **rticles** (Allaire *et al.* 2024).

Franco Pestilli was supported with grants from the National Science Foundation (OAC-1916518, IIS-1912270, IIS-1636893, BCS-1734853), National Institutes of Health (NIMH R01MH126699, NIBIB R01EB030896, NIBIB R01EB029272), a Microsoft Investigator Fellowship, and a gift from the Kavli Foundation. Daniel J. McDonald was partially supported by the National Science Foundation (DMS–1753171) and the National Sciences and Engineering Research Council of Canada (RGPIN-2021-02618). The analysis in Section 4.1 is based on survey results from Carnegie Mellon University's Delphi Group.

References

Agrawal A, Verschueren R, Diamond S, Boyd S (2018). "A Rewriting System for Convex Optimization Problems." *Journal of Control and Decision*, **5**(1), 42–60. doi:10.1080/23307706.2017.1397554.

Akaike H (1973). "Information Theory and an Extension of the Maximum Likelihood Principle." In BN Petrov, F Csaki (eds.), *Proceedings of the 2nd International Symposium of Information Theory*, pp. 267–281.

Allaire JJ, Xie Y, Dervieux C, R Foundation, Wickham H, Journal of Statistical Software, Vaidyanathan R, Association for Computing Machinery, Boettiger C, Elsevier, Broman K, Mueller K, Quast B, Pruim R, Marwick B, Wickham C, Keyes O, Yu M, Emaasit D, Onkelinx T, Gasparini A, Desautels MA, Leutnant D, MDPI, Taylor, rancis, Öğreden O, Hance D, Nüst D, Uvesten P, Campitelli E, Muschelli J, Hayes A, Kamvar ZN, Ross N, Cannoodt R, Luguern D, Kaplan DM, Kreutzer S, Wang S, Hesselberth J, Hyndman R (2024). ***rticles**: Article Formats for R Markdown.* doi:10.32614/CRAN.package.rticles. R package version 0.27.

Aminmansour F, Patterson A, Le L, Peng Y, Mitchell D, Pestilli F, Caiafa CF, Greiner R, White M (2019). "Learning Macroscopic Brain Connectomes via Group-Sparse Factorization." In H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox, R Garnett (eds.), *Advances in Neural Information Processing Systems 32*, volume 32. URL https://proceedings.neurips.cc/paper/2019/hash/0bfce127947574733b19da0f30739fcd-Abstract.html.

Caiafa CF, Sporns O, Saykin A, Pestilli F (2017). "Unified Representation of Tractography and Diffusion-Weighted MRI Data Using Sparse Multidimensional Arrays." In I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. URL https://proceedings.neurips.cc/paper/2017/hash/ccbd8ca962b80445df1f7f38c57759f0-Abstract.html.

Candes EJ, Tao T (2007). "The Dantzig Selector: Statistical Estimation When p Is Much Larger than n." *The Annals of Statistics*, **35**(6), 2313–2351. doi:10.1214/009053606000001523.

Civieta AM, Aguilera-Morillo MC, Lillo RE (2021). "Adaptive Sparse Group LASSO in Quantile Regression." *Advances in Data Analysis and Classification*, **15**, 547–573. doi:10.1007/s11634-020-00413-8.

Daducci A, Dal Palù A, Lemkaddem A, Thiran JP (2015). "COMMIT: Convex Optimization Modeling for Microstructure Informed Tractography." *IEEE Transactions on Medical Imaging*, **34**, 246–257. doi:10.1109/tmi.2014.2352414.

Diamond S, Boyd S (2016). "**CVXPY**: A python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5. doi:10.1007/978-3-319-42056-1_7.

Efron B (1986). "How Biased Is the Apparent Error Rate of a Prediction Rule?" *Journal of the American Statistical Association*, **81**(394), 461–470. doi:10.1080/01621459.1986.10478291.

Friedman J, Hastie T, Tibshirani R, Narasimhan B, Tay K, Simon N, Yang J (2023). ***glmnet**: Lasso and Elastic-Net Regularized Generalized Linear Models*. doi:10.32614/CRAN.package.glmnet. R package version 4.1-8.

Fu A, Narasimhan B, Boyd S (2020). "**CVXR**: An R Package for Disciplined Convex Optimization." *Journal of Statistical Software*, **94**(14), 1–34. doi:10.18637/jss.v094.i14.

Fu A, Narasimhan B, Kang DW, Diamond S, Miller J (2024). ***CVXR**: Disciplined Convex Optimization*. doi:10.32614/CRAN.package.CVXR. R package version 1.0-14.

Gerber F, Moesinger K, Furrer R (2017). "Extending R Packages to Support 64-Bit Compiled Code: An Illustration with **spam64** and GIMMS NDVI3g Data." *Computer & Geoscience*, **104**, 109–119. doi:10.1016/j.cageo.2016.11.015.

Gerber F, Moesinger K, Furrer R (2018). "**dotCall64**: An R Package Providing an Efficient Interface to Compiled C, C++, and Fortran Code Supporting Long Vectors." *SoftwareX*, **7**, 217–221. doi:10.1016/j.softx.2018.06.002.

Golub GH, Heath M, Wahba G (1979). "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter." *Technometrics*, **21**(2), 215–223. doi:10.1080/00401706.1979.10489751.

McDonald DJ, Liang X, Solón Heinsfeld A, Cohen A (2024). ***sparsegl**: Sparse Group Lasso*. doi:10.32614/CRAN.package.sparsegl. R package version 1.1.1.

Pestilli F, Yeatman JD, Rokem A, Kay KN, Wandell BA (2014). "Evaluation and Statistical Inference for Human Connectomes." *Nature Methods*, **11**(10), 1058–1063. doi:10.1038/nmeth.3098.

Qiu Y, Mei J (2024). ***RSpectra**: Solvers for Large-Scale Eigenvalue and SVD Problems*. doi:10.32614/CRAN.package.RSpectra. R package version 0.16-2.

R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL https://www.R-project.org/.

Schiavi S, Ocampo-Pineda M, Barakovic M, Petit L, Descoteaux M, Thiran JP, Daducci A (2020). "A New Method for Accurate in Vivo Mapping of Human Brain Connections Using Microstructural and Anatomical Information." *Science Advances*, **6**(31), eaba8245. doi:10.1126/sciadv.aba8245.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**, 461–464. doi:10.1214/aos/1176344136.

Simon N, Friedman J, Hastie T, Tibshirani R (2013). "A Sparse-Group Lasso." *Journal of Computational and Graphical Statistics*, **22**(2), 231–245. doi:10.1080/10618600.2012.681250.

Simon N, Friedman J, Hastie T, Tibshirani R (2019). ***SGL**: Fit a GLM (or Cox Model) with a Combination of Lasso and Group Lasso Regularization.* doi:10.32614/CRAN.package.SGL. R package version 1.3.

Stroustrup B (2013). *The C++ Programming Language.* 4th edition. Addison-Wesley.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, **58**(1), 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x.

Tibshirani R, Bien J, Friedman J, Hastie T, Simon N, Taylor J, Tibshirani RJ (2012). "Strong Rules for Discarding Predictors in Lasso-Type Problems." *Journal of the Royal Statistical Society B*, **74**(2), 245–266. doi:10.1111/j.1467-9868.2011.01004.x.

Tseng P (2001). "Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization." *Journal of Optimization Theory and Applications*, **109**(3), 475–494. doi:10.1023/a:1017501703105.

Vaiter S, Deledalle C, Peyré G, Fadili J, Dossal C (2012). "The Degrees of Freedom of the Group Lasso for a General Design." *arXiv 1212.6478*, arXiv.org E-Print Archive. doi:10.48550/arXiv.1212.6478.

Van Essen DC, Ugurbil K, Auerbach E, Barch D, Behrens TEJ, Bucholz R, Chang A, Chen L, Corbetta M, Curtiss SW, Della Penna S, Feinberg D, Glasser MF, Harel N, Heath AC, Larson-Prior L, Marcus D, Michalareas G, Moeller S, Oostenveld R, Petersen SE, Prior F, Schlaggar BL, Smith SM, Snyder AZ, Xu J, Yacoub E (2012). "The Human Connectome Project: A Data Acquisition Perspective." *NeuroImage*, **62**(4), 2222–2231. doi:10.1016/j.neuroimage.2012.02.018.

Van Rossum G, *et al.* (2011). *Python Programming Language.* URL https://www.python.org/.

Wickham H (2024). ***testthat**: Unit Testing for R.* doi:10.32614/CRAN.package.testthat. R package version 3.2.1.1.

Wickham H, Bryan J, Barrett M, Teucher A (2024a). ***usethis**: Automate Package and Project Setup.* doi:10.32614/CRAN.package.usethis. R package version 3.0.0.

Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K, Yutani H, Dunnington D, Van den Brand T (2024b). ***ggplot2**: Create Elegant Data Visualisations Using the Grammar of Graphics.* doi:10.32614/CRAN.package.ggplot2. R package version 3.5.1.

Wickham H, Hester J, Chang W, Bryan J (2022). ***devtools**: Tools to Make Developing R Packages Easier.* doi:10.32614/CRAN.package.devtools. R package version 2.4.5.

Wood S (2023). ***mgcv**: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation.* doi:10.32614/CRAN.package.mgcv. R package version 1.9-1.

Wood SN (2017). *Generalized Additive Models: An Introduction with R.* 2nd edition. Chapman & Hall/CRC, New York.

Xie Y (2024). ***knitr**: A General-Purpose Package for Dynamic Report Generation in R.* doi:10.32614/CRAN.package.knitr. R package version 1.48.

Yang Y, Zou H (2015). "A Fast Unified Algorithm for Solving Group-Lasso Penalized Learning Problems." *Statistics and Computing*, **25**(6), 1129–1141. doi:10.1007/s11222-014-9498-5.

Yang Y, Zou H, Bhatnagar S (2024). ***gglasso**: Group Lasso Penalized Learning Using a Unified BMD Algorithm.* doi:10.32614/CRAN.package.gglasso. R package version 1.5.1.

Yuan M, Lin Y (2006). "Model Selection and Estimation in Regression with Grouped Variables." *Journal of the Royal Statistical Society B*, **68**(1), 49–67. doi:10.1111/j.1467-9868.2005.00532.x.

Zeng Y, Breheny P (2020). "The **biglasso** Package: A Memory- And Computation-Effcient Solver for Lasso Model Fitting with Big Data in R." *The R Journal*, **12**(2), 6–19. doi:10.32614/rj-2021-001.

Zeng Y, Wang C, Peter T, Breheny P (2024). ***biglasso**: Extending Lasso Model Fitting to Big Data.* doi:10.32614/CRAN.package.biglasso. R package version 1.6.0.

Zou H, Hastie T, Tibshirani R (2007). "On the 'Degrees of Freedom' of the Lasso." *The Annals of Statistics*, **35**(5), 2173–2192. doi:10.1214/009053607000000127.

Affiliation:

Xiaoxuan Liang
University of British Columbia
Department of Statistics
Vancouver, BC, Canada

Aaron Cohen
Indiana University
Department of Statistics
Bloomington, IN, United States of America

Anibal Sólon Heinsfeld
The University of Texas at Austin
Department of Computer Science
Austin, TX, United States of America

Franco Pestilli
The University of Texas at Austin
Department of Psychology
Center for Perceptual Systems
Austin, TX, United States of America

Daniel J. McDonald
University of British Columbia
Department of Statistics
Vancouver, BC, Canada
E-mail: `daniel@stat.ubc.ca`
URL: `https://dajmcdon.github.io/`

Journal of Statistical Software `https://www.jstatsoft.org/`
published by the Foundation for Open Access Statistics `https://www.foastat.org/`
August 2024, Volume 110, Issue 6 *Submitted:* 2022-11-24
`doi:10.18637/jss.v110.i06` *Accepted:* 2023-10-10