Coordinate Descent for SLOPE
Johan Larsson, Quentin Klopfenstein, Mathurin Massias, Jonas Wallin

To cite this version:
Johan Larsson, Quentin Klopfenstein, Mathurin Massias, Jonas Wallin. Coordinate Descent for SLOPE. 26th International Conference on Artificial Intelligence and Statistics - AISTATS 2023, Apr 2023, Valencia, Spain. hal-04211867

HAL Id: hal-04211867
https://hal.science/hal-04211867
Submitted on 20 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Abstract

The lasso is the most famous sparse regression and feature selection method. One reason for its popularity is the speed at which the underlying optimization problem can be solved. Sorted L-One Penalized Estimation (SLOPE) is a generalization of the lasso with appealing statistical properties. In spite of this, the method has not yet reached widespread interest. A major reason for this is that current software packages that fit SLOPE rely on algorithms that perform poorly in high dimensions. To tackle this issue, we propose a new fast algorithm to solve the SLOPE optimization problem, which combines proximal gradient descent and proximal coordinate descent steps. We provide new results on the directional derivative of the SLOPE penalty and its related SLOPE thresholding operator, as well as provide convergence guarantees for our proposed solver. In extensive benchmarks on simulated and real data, we demonstrate our method’s performance against a long list of competing algorithms.

1 INTRODUCTION

In this paper we present a novel numerical algorithm for Sorted L-One Penalized Estimation (SLOPE, Bogdan et al., 2013; Bogdan et al., 2015; Zeng and Figueiredo, 2014), which, for a design matrix $X \in \mathbb{R}^{n \times p}$ and response vector $y \in \mathbb{R}^n$, is defined as

$$
\beta^* \in \arg \min_{\beta \in \mathbb{R}^p} P(\beta) = \frac{1}{2} \|y - X\beta\|^2 + J(\beta) \quad (1)
$$

where

$$
J(\beta) = \sum_{j=1}^{p} \lambda_j |\beta(j)| \quad (2)
$$

is the sorted $\ell_1$ norm, defined through

$$
|\beta(1)| \geq |\beta(2)| \geq \cdots \geq |\beta(p)|, \quad (3)
$$

with $\lambda$ being a fixed non-increasing and non-negative sequence.

The sorted $\ell_1$ norm is a sparsity-enforcing penalty that has become increasingly popular due to several appealing properties, such as its ability to control false discovery rate (Bogdan et al., 2015; Kos and Bogdan, 2020), cluster coefficients (Figueiredo and Nowak, 2016; Schneider and Tardivel, 2020), and recover sparsity and ordering patterns in the solution (Bogdan et al., 2022). Contrary to other coefficient clustering approaches such as the fused Lasso (Tibshirani et al., 2005), it is independent of feature order, and in addition does not require prior knowledge of the number of clusters. Finally, unlike other competing sparse regularization methods such as MCP (Zhang, 2010) and SCAD (Fan and Li, 2001), SLOPE has the advantage of being a convex problem (Bogdan et al., 2015).

In spite of the availability of predictor screening rules (Larsson, Bogdan, and Wallin, 2020; Elvira and Herzet, 2021), which help speed up SLOPE in the high-dimensional regime, current state-of-the-art algorithms...
for SLOPE perform poorly in comparison to those of more established penalization methods such as the lasso ($\ell_1$ norm regularization) and ridge regression ($\ell_2$ norm regularization). As a small illustration of this issue, we compared the speed at which the SLOPE (Larsson et al., 2022) and glmnet (Friedman et al., 2022) packages solve a SLOPE and lasso problem, respectively, for the bcTCA data set. SLOPE takes 43 seconds to reach convergence, whilst glmnet requires only 0.14 seconds\(^1\). This lackluster performance has hampered the applicability of SLOPE to many real-world applications. In this paper, we present a remedy for this issue: an algorithm that reaches convergence in only 2.9 seconds on the same problem\(^2\).

A major reason for why algorithms for solving $\ell_1$-, MCP-, or SCAD-regularized problems enjoy better performance is that they use coordinate descent (Tseng, 2001; Friedman, Hastie, and Tibshirani, 2010; Breheny and Huang, 2011). Current SLOPE solvers, on the other hand, rely on proximal gradient descent algorithms such as FISTA (Beck and Teboulle, 2009) and the alternating direction method of multipliers method (ADMM, Boyd et al., 2010), which have proven to be less efficient than coordinate descent in empirical benchmarks on related problems, such as the lasso (Moreau et al., 2022). In addition to FISTA and ADMM, there has also been research into Newton-based augmented Lagrangian methods to solve SLOPE (Luo et al., 2019). But this method is adapted only to the $p \gg n$ regime and, as we show in our paper, is outperformed by our method even in this scenario. Applying coordinate descent to SLOPE is not, however, straightforward since convergence guarantees for coordinate descent require the non-smooth part of the objective to be coordinate-wise separable, which is not the case for SLOPE. As a result, naive coordinate descent schemes can get stuck (Figure 1).

In this article, we address this problem by introducing a new, highly effective algorithm for SLOPE based on a hybrid proximal gradient and coordinate descent scheme. Our method features convergence guarantees and reduces the time required to fit SLOPE by orders of magnitude in our empirical experiments.

**Notation** Let $(i)^-$ be the inverse of $(i)$ such that $((i)^-) = (i)$; see Table 1 for an example of this operator for a particular $\beta$. This means that

$$J(\beta) = \sum_{j=1}^{p} \lambda_j |\beta_j| = \sum_{j=1}^{p} \lambda_{(j)} - |\beta_j|.$$  

\(^1\)See Appendix B.1 for details on this experiment.

\(^2\)Note that we do not use any screening rule in the current implementation of our algorithm, unlike the SLOPE package, which uses the strong screening rule for SLOPE (Larsson, Bogdan, and Wallin, 2020).

Figure 1: An example of standard coordinate descent getting stuck on a two-dimensional SLOPE problem. The main plot shows level curves for the primal objective (1), with the minimizer $\beta^* = [0, 0]^T$ indicated by the orange cross. The marginal plots display objective values at $\beta_1 = 0.2$ when optimizing over $\beta_2$ and vice versa. At $\beta = [0.2, 0.2]^T$, standard coordinate descent can only move in the directions indicated by the dashed lines—neither of which are descent directions for the objective. As a result, the algorithm is stuck at a suboptimal point.

Sorted $\ell_1$ norm penalization leads to solution vectors with clustered coefficients in which the absolute values of several coefficients are set to exactly the same value. To this end, for a fixed $\beta$ such that $|\beta_j|$ takes $m$ distinct values, we introduce $C_1, C_2, \ldots, C_m$ and $c_1, c_2, \ldots, c_m$ for the indices and coefficients respectively of the $m$ clusters of $\beta$, such that $C_i = \{j : |\beta_j| = c_i\}$ and $c_1 > c_2 > \cdots > c_m \geq 0$. For a set $C$, let $\bar{C}$ denote its complement. Furthermore, let $(e_i)_{i \in [d]}$ denote the canonical basis of $\mathbb{R}^d$, with $[d] = \{1, 2, \ldots, d\}$. Let $X_i$ and $X_{i:}$ denote the $i$-th row and column, respectively, of the matrix $X$. Finally, let $\text{sign}(x) = x/|x|$ (with the convention $0/0 = 1$) be the scalar sign, that acts entrywise on vectors.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\beta_i$</th>
<th>$(i)$</th>
<th>$(i)^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>-5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
2 COORDINATE DESCENT FOR SLOPE

Proximal coordinate descent cannot be applied to Problem 1 because the non-smooth term is not separable. If the clusters $C_1^*, \ldots, C_m^*$ and signs of the solution $\beta^*$ were known, however, then the values $c_1^*, \ldots, c_m^*$ taken by the clusters of $\beta^*$ could be computed by solving

$$
\min_{z \in \mathbb{R}^m^*} \frac{1}{2} \|y - X \sum_{i=1}^{m^*} \sum_{j \in C_i^*} z_j \text{sign}(\beta_j^*) e_j - X \beta^* \|_2^2
+ \sum_{i=1}^{m^*} |z_i| \sum_{j \in C_i^*} \lambda_j.
$$

(4)

Conditionally on the knowledge of the clusters and the signs of the coefficients, the penalty becomes separable (Dupuis and Tardivel, 2022), which means that coordinate descent could be used.

Based on this idea, we derive a coordinate descent update for minimizing the SLOPE problem (1) with respect to the coefficients of a single cluster at a time (Section 2.1). Because this update is limited to updating and, possibly, merging clusters, we intertwine it with proximal gradient descent in order to correctly identify the clusters (Section 2.2). In Section 2.3, we present this hybrid strategy and show that it is guaranteed to converge. In Section 3, we show empirically that our algorithm outperforms competing alternatives for a wide range of problems.

2.1 Coordinate Descent Update

In the sequel, let $\beta$ be fixed with $m$ clusters $C_1, \ldots, C_m$ corresponding to values $c_1, \ldots, c_m$. In addition, let $k \in [m]$ be fixed and $s_k = \text{sign} \beta_{C_k}$. We are interested in updating $\beta$ by changing only the value taken on the $k$-th cluster. To this end, we define $\beta(z) \in \mathbb{R}^p$ by:

$$
\beta_i(z) = \begin{cases} 
\text{sign}(\beta_i) z, & \text{if } i \in C_k, \\
\beta_i, & \text{otherwise.}
\end{cases}
$$

(5)

Minimizing the objective in this direction amounts to solving the following one-dimensional problem:

$$
\min_{z \in \mathbb{R}} \left( G(z) = P(\beta(z)) = \frac{1}{2} \|y - X \beta(z)\|_2^2 + H(z) \right),
$$

where

$$
H(z) = |z| \sum_{j \in C_k} \lambda_j | \beta_j | + \sum_{j \notin C_k} \lambda_j | \beta_j | (7)
$$

is the partial sorted $\ell_1$ norm with respect to the $k$-th cluster and where we write $\lambda_{(j)z}$ to indicate that the inverse sorting permutation $(j)_z^-$ is defined with respect to $\beta(z)$. The optimality condition for Problem (6) is

$$
\forall \delta \in \{-1,1\}, \quad G'(z; \delta) \geq 0,
$$

where $G'(z; \delta)$ is the directional derivative of $G$ in the direction $\delta$. Since the first part of the objective is differentiable, we have

$$
G'(z; \delta) = \delta \sum_{j \notin C_k} X_j^T (X \beta(z) - y) + H'(z; \delta),
$$

where $H'(z; \delta)$ is the directional derivative of $H$.

Throughout the rest of this section, we derive the solution to (6). To do so, we will introduce the directional derivative for the sorted $\ell_1$ norm with respect to the coefficient of the $k$-th cluster. First, as illustrated in Figure 2, note that $H$ is piecewise affine, with breakpoints at 0 and all $c_i$’s for which $i \neq k$. Hence, the partial derivative is piecewise constant, with jumps at these points; in addition, $H'(-1) = H'(-1)$ except at these points.

![Figure 2: Graph of the partial sorted $\ell_1$ norm with $\beta = \{-3,1,3,2\}^T$, $k = 1$, and so $c_1, c_2, c_3 = (3,2,1)$.](image)

Let $C(z)$ be the function that returns the cluster of $\beta(z)$ corresponding to $|z|$, that is

$$
C(z) = \{ j : |\beta(z)_j| = |z| \}.
$$

(8)

Remark 2.1. Note that if $z$ is equal to some $c_i$, then $C(z) = C_i \cup C_k$, and otherwise $C(z) = C_k$. Related to the piecewise affineness of $H$ is the fact that the permutation* corresponding to $\beta(z)$ is

$$
\begin{cases} 
C_k, C_m, \ldots, C_1, & \text{if } z \in (0, c_m), \\
C_m, \ldots, C_i, C_k, C_{i-1}, \ldots, C_1, & \text{if } z \in (c_i, c_{i-1}), \\
C_m, \ldots, C_1, C_k, & \text{if } z \in (c_i, +\infty),
\end{cases}
$$

and that this permutation also reorders $\beta(z \pm h)$ for $z \neq c_i$ $(i \neq k)$ and $h$ small enough. The only change in permutation happens when $z = 0$ or $z = c_i$ $(i \neq k)$. Finally, the permutations differ between $\beta(z + h)$ and $\beta(z - h)$ for arbitrarily small $h$ if and only if $z = c_i \neq 0$.

*the permutation is in fact not unique, without impact on our results. This is discussed when needed in the proofs.
We can now state the directional derivative of $H$.

**Theorem 2.2.** Let $c^{<k}$ be the set containing all elements of $c$ except the $k$-th one: $c^{<k} = \{c_1, \ldots, c_{k-1}, c_{k+1}, \ldots, c_m\}$. Let $\varepsilon_c > 0$ such that

$$
\varepsilon_c < |c_i - c_j|, \quad \forall i \neq j \text{ and } \varepsilon_c < c_m \text{ if } c_m \neq 0. \quad (9)
$$

The directional derivative of the partial sorted $\ell_1$ norm with respect to the $k$-th cluster, $H$, in the direction $\delta$ is

$$
H'(z; \delta) = \begin{cases} 
\sum_{j \in C(\varepsilon_c)} \lambda_j \delta c_j & \text{if } z = 0, \\
\text{sign}(z) \sum_{j \in C(z+\varepsilon_c, \delta)} \lambda_j \delta c_j & \text{if } |z| \in c^{<k} \setminus \{0\}, \\
\text{sign}(z) \sum_{j \notin C(z)} \lambda_j \delta c_j & \text{otherwise}.
\end{cases}
$$

The proof is in Appendix A.1; in Figure 3, we show an example of the directional derivative and the objective function.

![Graph of G(z) and G'(z; δ)](image)

**Theorem 2.3** (The SLOPE Thresholding Operator). Define $S(x) = \sum_{j \in C(x)} \lambda_j \delta c_j$ and let

$$
T(\gamma; \omega, c, \lambda) = \begin{cases} 
0 & \text{if } |\gamma| \leq S(\varepsilon_c), \\
\text{sign}(\gamma) c_i & \text{if } |\gamma| \leq \omega_c + S(\varepsilon_c), \\
\frac{\text{sign}(\gamma)}{\omega} (|\gamma| - S(\varepsilon_c)) & \text{if } |\gamma| \leq \omega_c + S(\varepsilon_c), \omega_c < \omega < 1, \\
\frac{\text{sign}(\gamma)}{\omega} (|\gamma| - S(\varepsilon_c)) & \text{if } |\gamma| \geq \omega_c + S(\varepsilon_c), \omega_c < \omega.
\end{cases}
$$

with $\varepsilon_c$ defined as in (9). Let $\tilde{x} = X \beta \text{sign}(\beta c_k)$ and $r = y - X \beta$. Then

$$
T \left( c_k \|\tilde{x}\|^2 + \tilde{x}^T r; \|x\|^2, c^{<k}, \lambda \right) = \arg\min_{z \in \mathbb{R}} G(z). \quad (10)
$$

An illustration of this operator is given in Figure 4.

**Remark 2.4.** The minimizer is unique because $G$ is the sum of a quadratic function in one variable and a norm.

**Remark 2.5.** In the lasso case where the $\lambda_i$'s are all equal, the SLOPE thresholding operator reduces to the soft thresholding operator.

In practice, it is rarely necessary to compute all sums in Theorem 2.3. Instead, we first check in which direction we need to search relative to the current order for the cluster and then search in that direction until we find the solution. The complexity of this operation depends on how far we need to search and the size of the current cluster and other clusters we need to consider. In practice, the cost is typically larger at the start of optimization and becomes marginal as the algorithm approaches convergence and the cluster permutation stabilizes.

### 2.2 Proximal Gradient Descent Update

The coordinate descent update outlined in the previous section updates the coefficients of each cluster in unison, which allows clusters to merge—but not to split. This means that the coordinate descent updates are not guaranteed to identify the clusters of the solution on their own, and thus are not guaranteed to converge to a solution of (1). To circumvent this issue, we combine these coordinate descent steps with a full proximal gradient step (Bogdan et al., 2015). This enables the algorithm to identify the cluster structure (Liang, Fadili, and Peyré, 2014) due to the partial smoothness property of the sorted $\ell_1$ norm that we prove in Appendix A.4. A similar idea has previously been used in...
Bareilles, Iutzeler, and Malick (2022), wherein Newton steps are taken on the problem structure identified after a proximal gradient descent step.

### 2.3 Hybrid Strategy

We now present the proposed solver in Algorithm 1. For the first and every \( \nu \)-th iteration\(^4\), we perform a proximal gradient descent update. For the remaining iterations, we take coordinate descent steps.

The combination of the proximal gradient steps and proximal coordinate descent allows us to overcome the problem of vanilla proximal coordinate descent getting stuck because of non-separability and allows us to enjoy the speed-up provided by making local updates on each cluster, as we illustrate in Figure 5.

We now state that our proposed hybrid algorithm converges to a solution of Problem (1).

**Lemma 2.6.** Let \( \beta^{(t)} \) be an iterate generated by Algorithm 1. Then

\[
\lim_{t \to \infty} (P(\beta^{(t)}) - P(\beta^*)) = 0.
\]

**Computational Complexity** We examine the complexity of the proximal gradient step and the coordinate

\(^4\)Our experiments suggest that \( \nu \) has little impact on performance as long as it is at least 3 (Appendix B.2). We have therefore set it to 5 in our experiments.

\[
\begin{align*}
\text{Algorithm 1 Hybrid coordinate descent and proximal gradient descent algorithm for SLOPE} \\
\text{input: } X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \{\mathbb{R}^p : \lambda_1 \geq \lambda_2 \geq \cdots \geq 0\}, \nu \in \mathbb{N}, \beta \in \mathbb{R}^p \\
\text{for } t \leftarrow 0, 1, \ldots \text{ do} \\
\quad \text{if } t \mod \nu = 0 \text{ then} \\
\quad\quad \beta \leftarrow \text{prox}_{f/\|x\|_2} \left( \beta - \frac{1}{\|x\|_2} X^T (X \beta - y) \right) \\
\quad\quad \text{Update } c, C \\
\quad \text{else} \\
\quad\quad k \leftarrow 1 \\
\quad\quad \text{while } k \leq |C| \text{ do} \\
\quad\quad\quad \tilde{x}_k \leftarrow X_{C_k} \text{sign}(\beta_{C_k}) \\
\quad\quad\quad z \leftarrow T(c_k \|x\|^2 - \tilde{x}^T (X \beta - y); \|x\|^2, c^{(k)}, \lambda) \\
\quad\quad\quad \beta_{C_k} \leftarrow z \text{sign}(\beta_{C_k}) \\
\quad\quad\quad \text{Update } c, C \\
\quad\quad k \leftarrow k + 1 \\
\text{return } \beta
\end{align*}
\]

descent separately. For the proximal gradient step, the complexity is \( O(np + p \log(p)) \), where \( np \) comes from the matrix-vector multiplication and \( p \log(p) \) from the computation of the proximal operator of the sorted \( \ell_1 \) norm (Zeng and Figueiredo, 2014). For the coordinate descent step, the worst case complexity is \( O(np + m(m + n)) \). As we will see in Section 3, however, the cost of the coordinate descent step turns out to be much lower in practice. The reason for this is

![Figure 4: An example of the SLOPE thresholding operator for \( \beta = [0.5, -0.5, 0.3, 0.7]^T, c = (0.7, 0.5, 0.3) \) with an update for the second cluster (\( k = 2 \), such that \( c^{(k)} = (0.5, 0.3) \). Across regions where the function is constant, the operator sets the result to be either exactly 0 or to the value of one of the elements of \( \pm c^{(k)} \).](image-url)
that the order of the clusters becomes increasingly stable during optimization. If, for instance, the order of the clusters is unchanged with respect to the previous step, then the complexity, in our implementation of Algorithm 1, reduces to $O(np + mn)$.

**Alternative Datafits** So far we have only considered sorted $\ell_1$-penalized least squares regression. In Appendix C, we consider possible extensions to alternative datafits.

## 3 Experiments

To investigate the performance of our algorithm, we performed an extensive benchmark against the following competitors:

- Alternating direction method of multipliers (ADMM, Boyd et al., 2010). We considered several alternative for the choice of the augmented Lagrangian parameter: an adaptive method to update the parameter throughout the algorithm (Boyd et al., 2010, Sec. 3.4.1) and fixed values. In the following sections, we only kept the ADMM solver with a fixed value of 100 for the augmented Lagrangian parameter. We present in Appendix B.3 a more detailed benchmarks for ADMM solvers with different values of this parameter and the adaptive setting. Choosing this parameter is not straightforward and the best value changes across datasets and regularization strengths.
- Anderson acceleration for proximal gradient descent (Anderson PGD, Zhang, O’Donoghue, and Boyd, 2020)
- Proximal gradient descent (PGD, Combettes and Wajs, 2005)
- Fast Iterative Shrinkage-Thresholding Algorithm (FISTA, Beck and Teboulle, 2009)
- Semismooth Newton-Based Augmented Lagrangian (Newt-ALM, Luo et al., 2019)
- The hybrid (our) solver (see Algorithm 1) combines proximal gradient descent and coordinate descent to overcome the non-separability of the SLOPE problem. We perform a coordinate descent step every fifth iteration ($v = 5$) in the algorithm. (See Section 2.3.)
- The oracle solver (oracle CD) solves Problem (4) with coordinate descent, using the clusters obtained via another solver. Note that it cannot be used in practice as it requires knowledge of the solution’s clusters.

We used Benchopt (Moreau et al., 2022) to obtain the convergence curves for the different solvers. Benchopt is a collaborative framework that allows reproducible and automatic benchmarks. The repository to reproduce the benchmark is available at [github.com/klopfe/benchmark_slope](http://github.com/klopfe/benchmark_slope).

Unless we note otherwise, we used the Benjamini–Hochberg method to compute the $\lambda$ sequence (Bogdan et al., 2015), which sets $\lambda_j = \eta^{-1}(1 - q \times j/(2p))$ for $j = 1, 2, \ldots, p$ where $\eta^{-1}$ is the probit function. For the rest of the experiments section, the parameter $q$ of this sequence has been set to 0.1 if not stated otherwise.\(^5\) We let $\lambda_{\text{max}}$ be the $\lambda$ sequence such that $\beta^* = 0$, but for

\(^5\)We initially experimented with various settings for $q$ but found that they made little difference to the relative performance of the algorithms.
which any scaling with a strictly positive scalar smaller than one produces a solution with at least one non-zero coefficient. We then parameterize the experiments by scaling $\lambda_{\max}$, using the fixed factors $1/2$, $1/10$, and $1/50$, which together cover the range of very sparse solutions to the almost-saturated case.

We pre-process datasets by first removing features with less than three non-zero values. Then, we center and scale each feature by its mean and standard deviation respectively. For sparse data, we scale each feature by its maximum absolute value. This approach is in line with recommendations in, for instance, Pedregosa et al. (2022).

Each solver was coded in python, using numpy (Harris et al., 2020) and numba (Lam, Pitrou, and Seibert, 2015) for performance-critical code. The code is available at github.com/jolars/slopecd. In Appendix D, we provide additional details on the implementations of some of the solvers used in our benchmarks.

The computations were carried out on a computing cluster with dual Intel Xeon CPUs (28 cores) and 128 GB of RAM.

3.1 Simulated Data

The design matrix $X$ was generated such that features had mean one and unit variance, with correlation between features $j$ and $j'$ equal to $0.6^{|j-j'|}$. We generated $\beta \in \mathbb{R}^p$ such that $k$ entries, chosen uniformly at random throughout the vector, were sampled from a standard Gaussian distribution. The response vector, meanwhile, was set to $y = X\beta + \epsilon$, where $\epsilon$ was sampled from a multivariate Gaussian distribution with variance such that $\|X\beta\|/\|\epsilon\| = 3$. The different scenarios for the simulated data are described in Table 2.

Table 2: Scenarios for the simulated data in our benchmarks, including the number of rows ($n$), columns ($p$), signals ($k$), and the fraction of non-zero entries (density) of $X$.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$n$</th>
<th>$p$</th>
<th>$k$</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>20,000</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>20,000</td>
<td>200</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>200,000</td>
<td>20</td>
<td>0.001</td>
</tr>
</tbody>
</table>

In Figure 6, we present the results of the benchmarks on simulated data. We see that for smaller fractions of $\lambda_{\max}$, our hybrid algorithm allows significant speedup in comparison to its competitors mainly when the number of features is larger than the number of samples. On very large scale data such as in simulated data setting 3, we see that the hybrid solver is faster than its competitors by one or two orders of magnitude.

For the second scenario, notice that all solvers take considerably longer than the oracle CD method to reach convergence. This gap is a consequence of Cholesky factorization in the case of ADMM and computation of $\|X\|_2$ in the remaining cases. For the hybrid method, we can avoid this cost, with little impact on performance, since $\|X\|_2$ is used only in the PGD step.

3.2 Real data

The datasets used for the experiments have been described in Table 3 and were obtained from Chang and Lin (2011), Chang and Lin (2022), and Breheny (2022).

Table 3: List of real datasets used in our experiments, including the number of rows ($n$), columns ($p$), signals ($k$), and the fraction of non-zero entries (density) of the corresponding $X$ matrices. See Table 6 in Appendix E for references on these datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$p$</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcTCGA</td>
<td>536</td>
<td>17,322</td>
<td>1</td>
</tr>
<tr>
<td>news20</td>
<td>19,996</td>
<td>1,355,191</td>
<td>0.00034</td>
</tr>
<tr>
<td>rcv1</td>
<td>20,242</td>
<td>44,504</td>
<td>0.0017</td>
</tr>
<tr>
<td>Rhee2006</td>
<td>842</td>
<td>360</td>
<td>0.025</td>
</tr>
</tbody>
</table>

Figure 7 shows the suboptimality for the objective function $P$ as a function of the time for the four different datasets. We see that when the regularization parameter is set at $\lambda_{\max}/2$ and $\lambda_{\max}/10$, our proposed solver is faster than all its competitors—especially when the datasets become larger. This is even more visible for the news20 dataset where we see that our proposed method is faster by at least one order of magnitude.

When the parametrization value is set to $\lambda_{\max}/50$, our algorithm remains competitive on the different datasets. It can be seen that the different competitors do not behave consistently across the datasets. For example, the Newt-ALM method is very fast on the bcTCGA dataset but is very slow on the news20 dataset whereas the hybrid method remains very efficient in both settings.

4 DISCUSSION

In this paper, we have presented a new, fast algorithm for solving Sorted L-One Penalized Estimation (SLOPE). Our method relies on a combination of proximal gradient descent to identify the cluster structure of the solution and coordinate descent to allow the algorithm to take large steps. In our results, we have shown that our method often outperforms all competitors by orders of magnitude for high-to-medium levels of regularization and typically performs among the best algorithms for low levels of regularization.
We have not, in this paper, considered using screening rules for SLOPE (Larsson, Bogdan, and Wallin, 2020; Elvira and Herzet, 2021). Although screening rules work for any algorithm considered in this article, they are particularly effective when used in tandem with coordinate descent (Fercoq, Gramfort, and Salmon, 2015) and, in addition, easy to implement due to the nature of coordinate descent steps. Coordinate descent is moreover especially well-adapted to fitting a path of \( \lambda \) sequences (Friedman et al., 2007; Friedman, Hastie, and Tibshirani, 2010), which is standard practice during cross-validating to obtain an optimal \( \lambda \) sequence.

Future research directions may include investigating alternative strategies to split clusters, for instance by considering the directional derivatives with respect to the coefficients of an entire cluster at once. Another potential approach could be to see if the full proximal gradient steps might be replaced with batch stochastic gradient descent in order to reduce the costs of these steps. It would also be interesting to consider whether gap safe screening rules might be used not only to screen predictors, but also to deduce whether clusters are able to change further during optimization. Finally, combining cluster identification of proximal gradient descent with solvers such as second order ones as in Bareilles, Iutzeler, and Malick (2022) is a direction of interest.

**Acknowledgements**

The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg (Varrette et al., 2022) (see [hpc.uni.lu](https://hpc.uni.lu)).

The results shown here are in whole or part based upon data generated by the TCGA Research Network: [https://www.cancer.gov/tcga](https://www.cancer.gov/tcga).

**References**

Bareilles, Gilles, Franck Iutzeler, and Jérôme Malick (2022). “Newton acceleration on manifolds identified by proximal gradient methods”. In: Mathematical Programming, pp. 1–34.


Bogdan, Malgorzata, Ewout van den Berg, Weijie Su, and Emmanuel Candès (2013). “Statistical Estima-
Figure 7: Benchmark on real datasets. The plots show suboptimality (difference between the objective at the current iterate, $P(\beta)$, and the optimum, $P(\beta^*)$) as a function of time for SLOPE on multiple simulated datasets and $\lambda$ sequences of varying strength.
Coordinate Descent for SLOPE


A PROOFS

A.1 Proof of Theorem 2.2

Let $c \setminus k$ be the set containing all elements of $c$ except the $k$-th one: $c \setminus k = \{c_1, \ldots, c_{k-1}, c_{k+1}, \ldots, c_m\}$.

From the observations in Remark 2.1, we have the following cases to consider: $|z| \in c \setminus k$, $|z| = 0$, and $|z| \notin \{0\} \cup c \setminus k$.

Since $C(z + \delta h) = C(z) = C_k$ and $\text{sign}(z + \delta h) = \text{sign}(z)$ for $h$ small enough,

$$H(z + \delta h) - H(z) = \sum_{j=1}^{p} |\beta(z + \delta h)_j| \lambda_{(j)}_{z + \delta h} - \sum_{j=1}^{p} |\beta(z)_j| \lambda_{(j)}_z$$

$$= \sum_{j=1}^{p} (|\beta(z + \delta h)_j| - |\beta(z)_j|) \lambda_{(j)}_z$$

$$= \sum_{j \in C(z)} (|\beta(z + \delta h)_j| - |\beta(z)_j|) \lambda_{(j)}_z$$

$$= \sum_{j \in C(z)} \text{sign}(\beta(z)_j)(z + \delta h - z) \lambda_{(j)}_z$$

$$= \sum_{j \in C_k} \text{sign}(z) \delta h \lambda_{(j)}_z$$

$$= \sum_{j \in C_k} \text{sign}(z) \delta h \lambda_{(j)}_z.$$  \hspace{1cm} (11)

**Case 2** Then if $z \neq 0$ and $|z|$ is equal to one of the $c_i$’s, $i \neq k$, one has $C(z) = C_k \cup C_i$, $C(z + \delta h) = C_k$, and $\text{sign}(z + \delta h) = \text{sign}(z)$ for $h$ small enough. Thus

$$H(z + \delta h) - H(z) = \sum_{j=1}^{p} |\beta(z + \delta h)_j| \lambda_{(j)}_{z + \delta h} - \sum_{i=1}^{p} |\beta(z)_j| \lambda_{(j)}_z$$

$$= \sum_{j \in C_k \cup C_i} \left( |\beta(z + \delta h)_j| \lambda_{(j)}_{z + \delta h} - |\beta(z)_j| \lambda_{(j)}_z \right)$$

$$= \sum_{j \in C_k} (c_i + \delta h) \lambda_{(j)}_{z + \delta h} - c_i \lambda_{(j)}_z + \sum_{j \in C_i} \left( c_i \lambda_{(j)}_{z + \delta h} - c_i \lambda_{(j)}_z \right).$$  \hspace{1cm} (12)

Note that there is an ambiguity in terms of permutation, since, due to the clustering, there can be more than one permutation reordering $\beta(z)$. However, choosing any such permutation result in the same values for the computed sums.
We now proceed to show that there is a solution to Algorithm 1, that is, the solution for the last case follows using reasoning analogous to that of the third case. To prove the lemma, we will show that
\[ H(z + \delta h) - H(z) = h \sum_{j \in k} \lambda_{(j) z + \delta h}. \]

If \( c_m \neq 0 \), then the computation proceeds exactly as in case 1.

**A.2 Proof of Theorem 2.3**

Recall that \( G(z) : \mathbb{R} \rightarrow \mathbb{R} \) is a convex, continuous piecewise-differentiable function with breakpoints whenever \( |z| > c_i^k \) or \( z = 0 \). Let \( \gamma = c_k \| \hat{x} \|^2 + \hat{x}^T r \) and \( \omega = \| \tilde{x} \| \) and note that the optimality criterion for (6) is
\[ \delta(z - \gamma) + H'(z; \delta) \geq 0, \quad \forall \delta \in \{-1, 1\}, \]
which is equivalent to
\[ \omega z - H'(z; -1) \leq \gamma \leq \omega z + H'(z; 1). \]
We now proceed to show that there is a solution \( z^* \in \arg\min_{z \in \mathbb{R}} H(z) \) for every interval over \( \gamma \in \mathbb{R} \).

First, assume that the first case in the definition of \( T \) holds and note that this is equivalent to (14) with \( z = 0 \) since \( C(\varepsilon_c) = C(-\varepsilon_c) \) and \( \lambda_{(j) -} = \lambda_{(j) -e_c} \). This is sufficient for \( z^* = 0 \).

Next, assume that the second case holds and observe that this is equivalent to (14) with \( z = c_i^k \), since \( C(c_i + \varepsilon_c) = C(-c_i - \varepsilon_c) \) and \( C(-c_i + \varepsilon_c) = C(c_i - \varepsilon_c) \). Thus \( z^* = \text{sign}(\gamma)c_i^k \).

For the third case, we have
\[ \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j) -} = \sum_{j \in C(c_i - \varepsilon_c)} \lambda_{(j) -} \]
and therefore (14) is equivalent to
\[ c_i < \frac{1}{\omega} \left( |\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j) -} \right) < c_i. \]

Now let
\[ z^* = \frac{\text{sign}(\gamma)}{\omega} \left( |\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j) -} \right) \]
and note that \( |z^*| \in (c_i^k, c_i^k - 1) \) and hence
\[ \frac{1}{\omega} \left( |\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j) -} \right) = \frac{1}{\omega} \left( |\gamma| - \sum_{j \in C(z^*)} \lambda_{(j) -} \right). \]

Furthermore, since \( G \) is differentiable in \( (c_i^k, c_i^k - 1) \), we have
\[ \frac{\partial}{\partial z} G(z) \left|_{z = z^*} \right. = \omega z^* - \gamma + \text{sign}(z^*) \sum_{j \in C(z^*)} \lambda_{(j) -} = 0, \]
and therefore (15) must be the solution.

The solution for the last case follows using reasoning analogous to that of the third case.

**A.3 Proof of Lemma 2.6**

To prove the lemma, we will show that \( \lim_{t \rightarrow \infty} \beta(t) \in \Omega = \{ \beta : 0 \in \partial P(\beta) \} \) using Convergence Theorem A in Zangwill (1969, p. 91). For simplicity, we assume that the point to set map \( A \) is generated by \( v \) iterations of Algorithm 1, that is \( A(\beta(0)) = \{ \beta^{(v)} \}_{i=0}^\infty \). To be able to use the theorem, we need the following assumptions to hold.
1. The set of iterates, $A(\beta^{(0)})$ is in a compact set.

2. $P$ is continuous and if $\beta \notin \Omega = \{ \beta : 0 \in \partial P(\beta) \}$, then for any $\hat{\beta} \in A(\beta)$ it holds that $P(\hat{\beta}) < P(\beta)$.

3. If $\beta \in \Omega = \{ \beta : 0 \in \partial P(\beta) \}$, then for any $\hat{\beta} \in A(\beta)$ it holds that $P(\hat{\beta}) \leq P(\beta)$.

Before tackling these three assumptions, we decompose the map into two parts: $v - 1$ coordinate descent steps, $T_{CD}$, and one proximal gradient decent step, $T_{PGD}$. This clearly means that

$$P(T_{CD}(\beta)) \leq P(\beta)$$

for all $\beta \in \mathbb{R}^p$. For $T_{PGD}$, we have two useful properties: first, if $||T_{PGD}(\beta) - \beta|| = 0$, then by Lemma 2.2 in Beck and Teboulle (2009) it follows that $\beta \in \Omega$. Second, by Lemma 2.3 in Beck and Teboulle (2009), using $x = y$, it follows that

$$P(T_{PGD}(\beta)) - P(\beta) \leq \frac{-L(f)}{2}||T_{PGD}(\beta) - \beta||^2,$$

where $L(f)$ is the Lipschitz constant of the gradient of $f(\beta) = \frac{1}{2}||y - X\beta||^2$.

We are now ready to prove that the three assumptions hold.

- Assumption 1 follows from the fact that the level sets of $P$ are compact and from $P(T_{PGD}(\beta)) \leq P(\beta)$ and $P(T_{CD}(\beta)) \leq P(\beta)$.

- Assumption 2 holds since if $\beta \notin \Omega$, it follows that $||T_{PGD}(\beta) - \beta|| > 0$ and thus $P(T_{PGD}(\beta)) < P(\beta)$.

- Assumption 3 follows from $P(T_{PGD}(\beta)) \leq P(\beta)$ and $P(T_{CD}(\beta)) \leq P(\beta)$.

Using Theorem 1 from Zangwill (1969), this means that Algorithm 1 converges as stated in the lemma.

### A.4 Partial Smoothness of the Sorted $\ell_1$ Norm

In this section, we prove that the sorted $\ell_1$ norm $J$ is partly smooth (Lewis, 2002). This allows us to apply results about the structure identification of the proximal gradient algorithm.

**Definition A.1.** Let $J$ be a proper closed convex function and $x$ a point of its domain such that $\partial J(x) \neq \emptyset$. $J$ is said to be partly smooth at $x$ relative to a set $\mathcal{M}$ containing $x$ if:

1. $\mathcal{M}$ is a $C^2$-manifold around $x$ and $J$ restricted to $\mathcal{M}$ is $C^2$ around $x$.

2. The tangent space of $\mathcal{M}$ at $x$ is the orthogonal of the parallel space of $\partial J(x)$.

3. $\partial J$ is continuous at $x$ relative to $\mathcal{M}$.

Because the sorted $\ell_1$ norm is a polyhedral, it follows immediately that it is partly smooth (Vaiter et al., 2017, Example 18). But since we believe a direct proof is interesting in and of itself, we provide and prove the following proposition here.

**Proposition A.2.** Suppose that the regularization parameter $\lambda$ is a strictly decreasing sequence. Then the sorted $\ell_1$ norm is partly smooth at any point of $\mathbb{R}^p$.

**Proof.** Let $m$ be the number of clusters of $x$ and $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be those clusters, and let $c_1 > \cdots > c_m > 0$ be the value of $|x|$ on the clusters.

We define $\varepsilon_c$ as in Equation (9) and let $\mathcal{B} = \{u \in \mathbb{R}^p : \|u - x\|_\infty < \varepsilon_c/2\}$. Let $v_k \in \mathbb{R}^p$ for $k \in [m]$ be equal to be sign$(x_{\mathcal{C}_k})$ on $\mathcal{C}_k$ and to 0 outside, such that $x = \sum_{k=1}^m c_k v_k$. We define

$$\mathcal{M} = \begin{cases} \text{span}(v_1, \ldots, v_m) \cap \mathcal{B} & \text{if } c_m \neq 0, \\ \text{span}(v_1, \ldots, v_{m-1}) \cap \mathcal{B} & \text{otherwise}. \end{cases}$$

We will show that $J$ is partly smooth at $x$ relative to $\mathcal{M}$.
As a first statement, we prove that any \( u \in M \) shares the same clusters as \( x \). For any \( u \in M \) there exists \( c' \in \mathbb{R}^m \), \( u = \sum_{k=1}^m c'_k v_k \) (with \( c'_m = 0 \) if \( c_m = 0 \)). Suppose that there exist \( k \neq k' \) such that \( c'_k = c'_{k'} \). Then since 
\[
\|x - u\|_\infty = \max_k |c_k - c'_{k}| \quad \text{and} \quad |c_k - c'_{k}| > \varepsilon_c,
\]
one has:
\[
\varepsilon_c < |c_k - c'_{k}| = |c_k - c'_{k} + c'_{k'} - c'_{k'}| \\
\leq |c_k - c'_{k}| + |c'_{k'} - c'_{k'}| \\
\leq 2\|x - u\|_\infty \\
\leq \varepsilon_c.
\]
This shows that clusters of any \( u \in M \) are equal to clusters of \( x \). Further, clearly the tangent space of \( M \) at \( x \) is \( \text{span}(v_1, \ldots, v_m) \) if \( c_m \neq 0 \) and \( \text{span}(v_1, \ldots, v_{m-1}) \) otherwise.

1. The set \( M \) is then the intersection of a linear subspace and an open ball, and hence is a \( C^2 \) manifold. Since the clusters of any \( u \in M \) are the same as the clusters of \( x \), we can write that
\[
J(u) = \sum_{k=1}^m \left( \sum_{j \in C_k} \lambda_j \right) c'_k,
\]
and hence \( J \) is linear on \( M \) and thus \( C^2 \) around \( x \).

2. We let \( x_\downarrow \) denote a version of \( x \) sorted in non-increasing order and let \( R : \mathbb{R}^p \to \mathbb{N}^p \) be the function that returns the ranks of the absolute values of its argument. The subdifferential of \( J \) at \( x \) (Larsson, Bogdan, and Wallin, 2020, Thm. 1)\(^6\) is the set of all \( g \in \mathbb{R}^p \) such that
\[
g_{C_i} \in G_i \triangleq \left\{ s \in \mathbb{R}^{|C_i|} : \begin{cases} \text{cumsum}(|s|_1 - \lambda_{R(g)_{C_i}}) \leq 0 & \text{if } x_{C_i} = 0, \\ \text{cumsum}(|s|_1 - \lambda_{R(g)_{C_i}}) \leq 0 \\ \text{and } \sum_{j \in C_i} (|s_j| - \lambda_{R(g)_{C_i}}) = 0 \\ \text{and } \text{sign}(x_{C_i}) = \text{sign}(s) \end{cases} \right. \quad \text{otherwise.}
\]

Hence, the problem can be decomposed over clusters. We will restrict the analysis to a single \( C_i \) without loss of generality and proceed in \( \mathbb{R}^{|C_i|} \).

- First we treat the case where \( |C_i| = 1 \) and \( x_{C_i} \neq 0 \). The set \( G_i \) is then the singleton \( \{ \text{sign}(x_{C_i}) \lambda_{R(s)_{C_i}} \} \) and its parallel space is simply \( \{0\} \). Hence, \( \text{par}(G_i) = \mathbb{R} = \text{span}(\text{sign}(x_{C_i})) \).

- Then, we study the case where \( |C_i| \neq 1 \) and \( x_{C_i} \neq 0 \). Since for all \( j \in [p] \), \( \lambda_j \neq 0 \) and \( \lambda \) is a strictly decreasing sequence, we have that for \( \varepsilon > 0 \) small enough, the \( |C_i| - 1 \) points \( \lambda_{R(g)_{C_i}} + \varepsilon [-\text{sign}(x_{C_i})_1, \text{sign}(x_{C_i})_2, 0, \ldots, 0]^T, \lambda_{R(g)_{C_i}} + \varepsilon [0, -\text{sign}(x_{C_i})_2, \text{sign}(x_{C_i})_3, \ldots, 0]^T, \ldots, \lambda_{R(g)_{C_i}} + \varepsilon [0, 0, 0, \ldots, -\text{sign}(x_{C_i})_{|C_i|-1}, \text{sign}(x_{C_i})_{|C_i|}]^T \) belong to \( G_i \). Since these vectors are linearly independent, and using the last equality in the feasible set that, we have that
\[
\sum_{j \in C_i} \text{sign}(x_j)s_j = \sum_{j \in C_i} \lambda_{R(g)_{C_i}}.
\]

Its parallel space is simply the set \( \{ s \in \mathbb{R}^{|C_i|} : \sum_{j \in C_i} \text{sign}(x_j)s_j = 0 \} \), that is just \( \text{span}(\text{sign}(x_{C_i})) \). Hence \( \text{par}(G_i) = \text{span}(\text{sign}(x_{C_i})) \).

- Finally, we study the case where \( x_{C_m} = 0 \). Then the \( \ell_\infty \) ball \( \{ s \in \mathbb{R}^{|C_m|} : \|s\|_\infty \leq \lambda_p \} \) is contained in the feasible set of the differential, hence the parallel space of \( G_m \) is \( \mathbb{R}^{|C_m|} \) and its orthogonal is reduced to \( \{0\} \).

We can now prove that \( \text{par}(\partial J(x)) = \text{span}(\text{sign}(x_{C_i})) \). From the decomposability of \( \partial J \) (Equation (17)), one has that \( u \in \text{par}(\partial J(x)) \) if and only if \( u_{C_i} = \text{par}(G_i) = \text{par}(\text{sign}(x_{C_i})) \) for all \( i \in [m] \).

\(^6\)We believe there to be a typo in the definition of the subgradient in (Larsson, Bogdan, and Wallin, 2020, Thm. 1). We believe the argument of \( R \) should be \( g \), not \( s \), since otherwise there is a dimension mismatch.
If $c_m > 0$, we have
\[
\text{par}(\partial J(x))^\perp = \{ u \in \mathbb{R}^p : \forall i \in [m], \ u_{C_i} \in \text{par}(G_i)^\perp \} \\
= \{ u \in \mathbb{R}^p : \forall i \in [m], \ u_{C_i} \in \text{span}(\text{sign}(x_{C_i})) \} \\
= \text{span}(v_1, \ldots, v_m). \tag{18}
\]

If $c_m = 0$, we have
\[
\text{par}(\partial J(x))^\perp = \{ u \in \mathbb{R}^p : \forall i \in [m], \ u_{C_i} \in \text{par}(G_i)^\perp \} \\
= \{ u \in \mathbb{R}^p : \forall i \in [m-1], \ u_{C_i} \in \text{span}(\text{sign}(x_{C_i})) \ \& \ u_{C_m} = 0 \} \\
= \text{span}(v_1, \ldots, v_{m-1}). \tag{19}
\]

3. The subdifferential of $J$ is a constant set locally around $x$ along $\mathcal{M}$ since the clusters of any point in the neighborhood of $x$ in $\mathcal{M}$ shares the same clusters with $x$. This shows that it is continuous at $x$ relative to $\mathcal{M}$.

Remark A.3. We believe that the assumption $\lambda_1 > \cdots > \lambda_p$ can be lifted, since for example the $\ell_1$ and $\ell_\infty$ norms are particular instances of $J$ that violate this assumption, yet are still partly smooth. Hence this assumption could probably be lifted in a future work using a slightly different proof.

B ADDITIONAL EXPERIMENTS

B.1 glmnet versus SLOPE Comparison

In this experiment, we ran the glmnet (Friedman et al., 2022) and SLOPE (Larsson et al., 2022) packages on the bcTCGA dataset, selecting the regularization sequence $\lambda$ such that there were 100 nonzero coefficients and clusters at the optimum for glmnet and SLOPE respectively. We used a duality gap of $10^{-6}$ as stopping criteria. The features were centered by their means and scaled by their standard deviation. The code is available at https://github.com/jolars/slopecd.

B.2 Study on Proximal Gradient Descent Frequency

To study the impact of the frequency at which the PGD step in the hybrid solver is used, we performed a comparative study with the rcv1 dataset. We set this parameter to values ranging from 1 i.e., the PGD algorithm, to 9 meaning that a PGD step is taken every 9 epochs. The sequence of $\lambda$ has been set with the Benjamini-Hochberg method and parametrized with $0.1 \lambda_{\text{max}}$.

Figure 8 shows the suboptimality score as a function of the time for the different values of the parameter controlling the frequency at which a PGD step is going to be taken. A first observation is that as long as this parameter is greater than 1 meaning that we perform some coordinate descent steps, we observe a significant speed-up. For all our experiments, this parameter was set to 5. The figure also shows that any choice between 3 and 9 would lead to similar performance for this example.

B.3 Benchmark with Different Parameters for the ADMM Solver

We reproduced the benchmarks setting described in Section 3 for the simulated and real data. We compared the ADMM solver with our hybrid algorithm for different values of the augmented Lagrangian parameter $\rho$. We tested three different values 10, 100 and 1000 as well as the adaptive method (Boyd et al., 2010, Sec. 3.4.1).

We present in Figure 9 and Figure 10 the suboptimality score as a function the time for the different solvers. We see that the best value for $\rho$ depends on the dataset and the regularization strength. The value chosen for the main benchmark (Section 3) performs well in comparison to other ADMM solvers. Nevertheless, our hybrid approach is consistently faster than the different ADMM solvers.
Figure 8: Suboptimality score as a function of the time for different frequencies of the PDG step inside the hybrid solver for the rcv1 dataset.

Figure 9: Benchmark on simulated datasets. Suboptimality score as a function of time for SLOPE on multiple simulated datasets and for multiple sequence of $\lambda$. 
Figure 10: **Benchmark on simulated datasets.** Suboptimality score as a function of time for SLOPE on multiple simulated datasets and for multiple sequence of $\lambda$.

### B.4 SLOPE Path Benchmarks

The choice of $\lambda$ sequence in SLOPE is usually made via cross-validation on a training partition of the data across a grid of the $q$ parameter, which controls the shape of the $\lambda$ sequence, and $\alpha$, a factor that scales the $\lambda$ sequence. The $\alpha$ grid is typically a decreasing sequence where the first and last values correspond to the intercept-only and almost-saturated models respectively. The set of models generated from fitting SLOPE across this grid of $\alpha$ values is called the SLOPE path. In this section, we report benchmarks on the performance of our algorithm in the case of fitting a SLOPE path to the simulated and real data sets used in Section 3.

We use the same path setup that is used by default for the lasso in the glmnet package (Friedman, Hastie, and Tibshirani, 2010). This entails using a grid of 100 $\alpha$ values spaced evenly on the log-e-scale. The first value, $\alpha_1$, is chosen such that it leads to the intercept-only model, that is $\alpha_1 \lambda = \lambda_{\text{max}}$ (see Section 3). The last value, $\alpha_{100}$ is set to $10^{-4}$ if $p > n$ and $10^{-2}$ otherwise. We terminate the path early if the number of unique nonzero coefficients exceed $n^7$, if the increase in the coefficient of determination is less than $10^{-4}$ between two subsequent $\alpha$s on the path, or if the coefficient of determination equals or exceeds 0.999. The $\lambda$ sequence setup, preprocessing, and data simulation setup is exactly the same as in Section 3.

For ADMM, we used $\rho = 100$ following the results in Appendix B.3. The Newt-ALM solver is missing from these

---

7Here we deviate from the standard lasso path setup because the lasso can at most select $n$ nonzero coefficients, whilst SLOPE can select at most $n$ nonzero unique coefficients.
benchmarks because we encountered several issues with convergence.

Our method outperforms the other methods for all of the real datasets (Table 4). In the case of bcTCGA, the difference is particularly striking, with our method taking less than one twentieth of the time taken for the runner-up. In the other cases, our method is roughly twice as fast as the second-best method.

Table 4: Time in seconds to fit a full SLOPE path to real data sets. See Table 3 for information about the datasets. For ADMM we set $\rho = 100$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rhee2006</th>
<th>bcTCGA</th>
<th>news20</th>
<th>rcv1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMM</td>
<td>47</td>
<td>9252</td>
<td>139415</td>
<td>6852</td>
</tr>
<tr>
<td>Anderson (PGD)</td>
<td>30</td>
<td>9867</td>
<td>29867</td>
<td>592</td>
</tr>
<tr>
<td>FISTA</td>
<td>335</td>
<td>12682</td>
<td>138363</td>
<td>1988</td>
</tr>
<tr>
<td>hybrid (ours)</td>
<td>14</td>
<td>379</td>
<td>11574</td>
<td>391</td>
</tr>
</tbody>
</table>

For simulated data (Table 5), our method performs best for the $p > n$ scenarios (1 and 2), being roughly ten and five times faster, respectively, than the runner up. In the case of Scenario 2 where $n > p$, however, Anderson (PGD) instead comes out on top.

Table 5: Time in seconds to fit a full SLOPE path to simulated data sets. See Table 2 for information on what the different scenarios mean. For ADMM we set $\rho = 100$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMM</td>
<td>593</td>
<td>732</td>
<td>649</td>
</tr>
<tr>
<td>Anderson (PGD)</td>
<td>539</td>
<td>36</td>
<td>451</td>
</tr>
<tr>
<td>FISTA</td>
<td>589</td>
<td>67</td>
<td>279</td>
</tr>
<tr>
<td>hybrid (ours)</td>
<td>54</td>
<td>83</td>
<td>49</td>
</tr>
</tbody>
</table>

This experiment was run on a dedicated high-performance computing cluster, using two Intel Xeon E5-2650 v3 (2.3 Ghz, 10-core) CPUs and 64 GB of memory. The computations were enabled by resources provided by LUNARC.

C EXTENSIONS TO OTHER DATAFITS

Our algorithm straightforwardly generalizes to problems where the quadratic datafit $\frac{1}{2}\|y - X\beta\|^2$ is replaced by $F(\beta) = \sum_{i=1}^n f_i(X_i^\top \beta)$, where the $f_i$’s are $L$ smooth (and so $F$ is $L \ast \|X\|^2_2$-smooth), such as logistic regression.

In that case, one has by the descent lemma applied to $F(\beta(z))$, using $F(\beta) = F(\beta(c_k))$,

$$F(\beta(z)) + H(z) \leq F(\beta) + \sum_{j \in C_k} \nabla_j F(\beta) \text{sign}_j(z - c_k) + \frac{L\|\hat{x}\|^2}{2} (z - c_k)^2 + H(z)$$

and so a majorization-minimization approach can be used, by minimizing the right-hand side instead of directly minimizing $F(\beta(z)) + H(z)$. Minimizing the RHS, up to rearranging, is of the form of Problem (6).

D IMPLEMENTATION DETAILS OF SOLVERS

D.1 ADMM

Our implementation of the solver is based on Boyd et al. (2011). For high-dimensional sparse $X$, we use the numerical LSQR algorithm (Paige and Saunders, 1982) instead of the typical direct linear system solver. We originally implemented the solver using the adaptive step size ($\rho$) scheme from Boyd et al. (2010) but discovered that it performed poorly. Instead, we used $\rho = 100$ and have provided benchmarks of the alternative configurations in Appendix B.3.
D.2 Newt-ALM

The implementation of the solver is based on the pseudo-code provided in Luo et al. (2019). According to the authors’ suggestions, we use the Matrix inversion lemma for high-dimensional and sparse $X$ and the preconditioned conjugate gradient method if, in addition, $n$ is large. Please see the source code for further details regarding hyper-parameter choices for the algorithm.

After having completed our own implementation of the algorithm, we received an implementation directly from the authors. Since our own implementation performed better, however, we opted to use it instead.

E REFERENCES AND SOURCES FOR DATASETS

In Table 6, we list the reference and source (from which the data was gathered) for each of the real datasets used in our experiments.

Table 6: Sources and references for the real data sets used in our experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Reference</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcTCGA</td>
<td>National Cancer Institute (2022)</td>
<td>Breheny (2022)</td>
</tr>
<tr>
<td>news20</td>
<td>Keerthi and DeCoste (2005)</td>
<td>Chang and Lin (2022)</td>
</tr>
<tr>
<td>rcv1</td>
<td>Lewis et al. (2004)</td>
<td>Chang and Lin (2022)</td>
</tr>
<tr>
<td>Rhee2006</td>
<td>Rhee et al. (2006)</td>
<td>Breheny (2022)</td>
</tr>
</tbody>
</table>