

A Survey of Numerical Algorithms that can Solve the Lasso Problems

Yujie Zhao¹ (ORCID ID: 0000-0003-2896-4955) and Xiaoming Huo²

¹ Biostatistics and Research Decision Sciences Department, Merck & Co., Inc

² The Stewart School of Industrial and System Engineering, Georgia Institute of Technology

Article Category

Advanced Review

Conflict of Interest

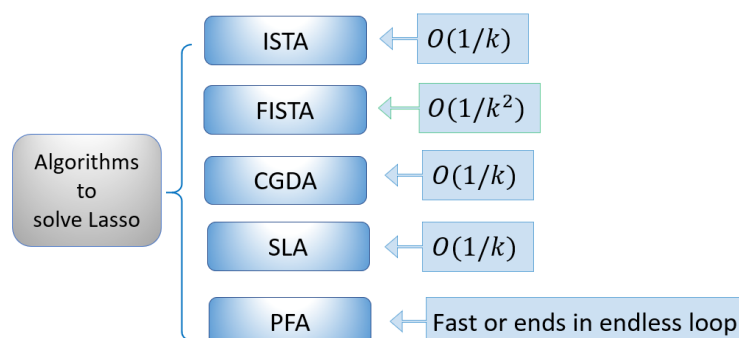
The authors have no conflict of interests.

Abstract

In statistics, the least absolute shrinkage and selection operator (Lasso) is a regression method that performs both variable selection and regularization. There is a lot of literature available, discussing the statistical properties of the regression coefficients estimated by the Lasso method. However, there lacks a comprehensive review discussing the algorithms to solve the optimization problem in Lasso. In this review, we summarize five representative algorithms to optimize the objective function in Lasso, including **iterative shrinkage threshold algorithm (ISTA)**, **fast iterative shrinkage-thresholding algorithms (FISTA)**, **coordinate gradient descent algorithm (CGDA)**, **smooth L1 algorithm (SLA)**, and **path following algorithm (PFA)**. Additionally, we also compare their convergence rate, as well as their potential strengths and weakness.

Keywords: Lasso, ℓ_1 regularization, convergence rate

Graphical/Visual Abstract and Caption



1 Introduction

In the regression analysis, one has the data

$$\mathcal{D} = \{y \in \mathbb{R}^n, X \in \mathbb{R}^{n \times p}\},$$

where y is the response vector and X is the model matrix (of predictors). Here $n, p > 0$ refers to the number of observations and covariates, respectively. Given the above dataset \mathcal{D} , the linear regression model can be written as

$$y = X\beta^* + w,$$

where $\beta^* \in \mathbb{R}^p$ is the ground truth of the regression coefficients desired to be estimated. And the vector $w \in \mathbb{R}^n$ is the white-noise residual, i.e., $w_i \stackrel{i.i.d.}{\sim} N(0, \sigma^2)$ for any $i = 1, \dots, n$.

When the number of covariates is greater than the number of observations, i.e., $p > n$, one prefers to select a subset of covariates and exclude the insignificant covariates. To realize the objective of variable selection, the least absolute shrinkage and selection operator (Lasso) (Tibshirani, 1996; Santosa and Symes, 1986) can be used:

$$\hat{\beta} = \arg \min_{\beta} \left\{ F(\beta) := \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\}, \quad (1)$$

where parameter $\lambda > 0$ controls the trade-off between the sparsity and model's goodness of fit. The objective function in the Lasso method is $F(\beta)$, whose first term is a nice quadratic function and numerically amenable. However, its second term of is not differentiable at the origin. To minimize this objective function $F(\beta)$, there does not exist a closed-form minimizer and the existing algorithms are mostly iterative algorithms.

In this paper, we review representative algorithms to minimize $F(\beta)$ and compare their convergence rates. The convergence rate measures how quickly the sequence $\beta^{(0)}, \beta^{(1)}, \beta^{(2)}, \dots$ approaches its optima $\hat{\beta}$, where $\beta^{(k)}$ is the iterative solution when minimizing $F(\beta)$ after k iterations. And the convergence rate is commonly in terms of k and the big O notation, like $O(1/k), O(1/k^2)$. In theory, an algorithm with convergence rate of $O(1/k^2)$ is more computationally efficient than that of $O(1/k)$. Yet, it does not say anything on the average performance of the algorithm. It is possible that an algorithm with convergence rate of $O(1/k)$ performs better in some cases than an algorithm with convergence rate of $O(1/k^2)$.

Please note that, we do not consider the selection of parameter λ in this review, which by itself has a large literature. Consequently, we don't include λ in the notation $F(\beta)$.

In the remainder of this review, we first introduce some key preliminaries in Section 2. Then we review five representative algorithms in Section 3. In Section 4, we give conclusions.

2 Preliminaries

In this section, we introduce some preliminaries, which helps readers to learn terminologies in statistical computations. The introduced terminologies include (1) Lipschitz continuous gradient, (2) convexity, (3) the (accelerate) gradient descent, (4) first/second order algorithms.

We begin with introducing two terms to describe functions, i.e., Lipschitz continuous gradient and convex functions.

Definition 2.1 (Lipschitz continuous gradient). A differentiable function $f(\cdot)$ has an Lipschitz continuous gradient L if for some $L > 0$, one has

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq L \|x_1 - x_2\|_2,$$

where $\nabla f(x)$ is the gradient of $f(x)$ at x .

Definition 2.2 (convex and strongly convex). A real-valued function $f(\cdot)$ is called convex, if the line segment between any two points on the graph of the function does not lie below the graph between the two points, i.e., $\forall x_1, x_2$ and $\alpha \in [0, 1]$, one has

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2).$$

If there exist $\mu > 0$, such that $\forall x_1, x_2$ we have

$$f(x_2) \geq f(x_1) + \nabla f(x_1)(x_2 - x_1) + \frac{\mu}{2} \|x_2 - x_1\|_2^2,$$

then we call $f(x)$ strongly convex.

For the above definition, one can verify that $f(x) = \log(x)$ is not a convex function, and $f(x) = x$ is convex but not strongly convex. On the other hand, $f(x) = x^2$ is convex and strongly convex. An example of a convex function is visualized in Figure 1. Besides, if a function both have Lipschitz continuous gradient as L and is strongly convex with μ , we call it L -smooth and μ -strongly convex function.

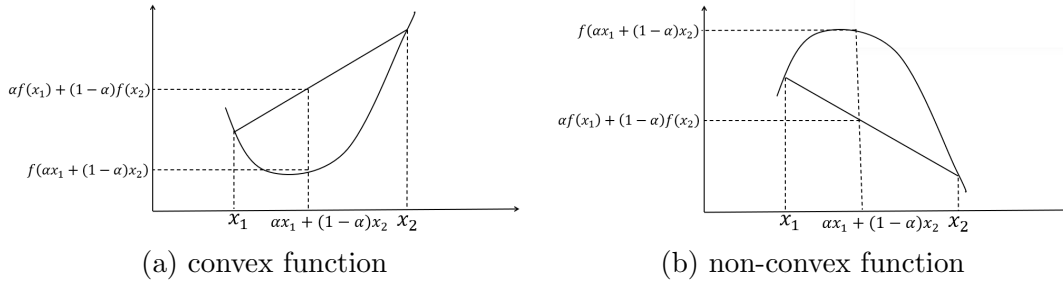


Figure 1: Examples of convex and non-convex functions

Next, we present one version of gradient descent (GD) and accelerated gradient descent (AGD) algorithm, which build the foundations of our reviewed algorithms.

Definition 2.3 (GD algorithm). Suppose one wants to minimize a convex and differentiable function $f : \mathbb{R}^p \rightarrow \mathbb{R}$, and that its gradient is Lipschitz continuous with constant L . To minimize $f(x)$ with respect to x , the gradient descent (GD) algorithm iterates as follows

$$x^{(k)} = x^{(k-1)} - \gamma_{k-1} \nabla f(x^{(k-1)})$$

for $k = 1, 2, \dots$. Here $x^{(k)}$ is the solution after k iterations. And the hyper-parameter $\gamma_k > 0$ is the step size or the learning rate, which can be either fixed throughout the iterations, or decided by the backtracking line search.

In the sequence of the gradient descent iterations, one has a monotonic sequence $f(x^{(0)}) \geq f(x^{(1)}) \geq f(x^{(2)}) \geq \dots$. Promisingly, the sequence $\{x^{(k)}\}_{k=0,1,\dots}$ converges to the desired local minimum. The convergence rate of the GD algorithm is summarized in the following lemma.

Lemma 2.4 (GD convergence rate). *Suppose the function $f : \mathbb{R}^p \rightarrow \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant L . Then if one runs the GD algorithm in Definition 2.3 for k iterations with a fixed step size $\gamma \leq 1/L$, it will yield a solution $f(x^{(k)})$ which satisfies*

$$f(x^{(k)}) - f(x^*) \leq \frac{\|x^{(0)} - x^*\|_2^2}{2\gamma k},$$

where $f(x^*)$ is the optimal value. Otherwise, if one runs GD algorithm for k iterations with step size γ_k chosen by backtracking line search on each iteration k , it will yield a solution $f(x^{(k)})$ which satisfies

$$f(x^{(k)}) - f(x^*) \leq \frac{\|x^{(0)} - x^*\|_2^2}{2\gamma_{\min} k},$$

where $\gamma_{\min} = \min\{1, \beta/L\}$. Here $\beta \in (0, 1)$ is a hyper-parameter in the backtracking line search: in the k -th iteration, we start with $\gamma_k = 1$ and while $f(x - \nabla f(x)) > f(x) - \frac{1}{2}\gamma \|\nabla f(x)\|_2^2$, we update $\gamma_k = \beta\gamma_k$.

Intuitively, the above lemma indicates that the GD algorithm is guaranteed to converge and that it converges with rate $O(1/k)$. Motivated by GD algorithm, researchers (Nesterov, 1983) proposed its accelerated version, called AGD algorithm. It has been widely applied into many optimization problems to speed up the convergence rate (Nesterov, 2005, 2013; Beck and Teboulle, 2009; Li and Lin, 2015). Following please find one version of the AGD algorithm.

Definition 2.5 (AGD algorithm). *Suppose ones wants to minimize a function $f = g + h : \mathbb{R}^p \rightarrow \mathbb{R}$, where g is a convex and differentiable function and h is a convex function. To minimize $f(x)$ with respect to x , the accelerated gradient descent (AGD) algorithm iterates as follows*

$$\begin{aligned} y^{(k)} &= x^{(k-1)} + \frac{k-2}{k+1} [x^{(k-1)} - x^{(k-2)}] \\ x^{(k)} &= \text{prox}_{t_k} \left(y^{(k)} - \gamma_k \nabla g(y^{(k)}) \right), \end{aligned} \quad (2)$$

where

$$\text{prox}_t(x) = \arg \min_{z \in \mathbb{R}^p} \frac{1}{2t} \|x - z\|_2^2 + h(z).$$

Here $x^{(k)}$ is the solution after k iterations. And $y^{(k)}$ is an auxiliary vector after k iterations. The hyper-parameter t_k is the step size or the learning rate, which can be either fixed throughout iterations, or decided by the backtracking line search.

Following the procedure of the AGD algorithm, we further introduce the convergence rate of the AGD algorithm (Lan, 2019, Theorem 3.7).

Lemma 2.6 (AGD convergence rate). *Suppose a function $f = g + h : \mathbb{R}^p \rightarrow \mathbb{R}$ with g as a convex and differentiable function and h as a convex function. Then if one runs the AGD algorithm in Definition 2.5 for k iterations with a fixed step size $\gamma \leq 1/L$, it will yield a solution $f(x^{(k)})$ which satisfies*

$$f(x^{(k)}) - f(x^*) \leq \frac{2\|x^{(0)} - x^*\|_2^2}{\gamma(k+1)^2}.$$

Otherwise, if one runs the AGD algorithm with a backtracking line search generated step size $\gamma_k \leq 1/L$, it will yield a solution $f(x^{(k)})$ which satisfies

$$f(x^{(k)}) - f(x^*) \leq \frac{2 \|x^{(0)} - x^*\|_2^2}{\gamma_{\min}(k+1)^2}.$$

where γ_{\min} is defined in Lemma 2.4.

The above lemma indicates that, the AGD algorithm is guaranteed to converge and that it converges with rate $O(1/k^2)$. And compared with GD algorithm, the AGD algorithm has a faster convergence rate theoretically.

Finally, we introduce the definition of *first order algorithm* and *second order algorithm*.

Definition 2.7 (first/second-order algorithm). *Any algorithm that requires at most the gradient/first order derivative is a first order algorithm. Any algorithm that uses any second order derivative is a second order algorithm. The accelerated first order algorithm is a particular type of algorithms that use multiple steps of gradients/first order derivatives.*

The well-known Newton–Raphson method is a second-order algorithm since it uses the second order derivatives (i.e., the Hessian). The GD algorithm is a first order algorithm. The AGD algorithm is motivated by the first order algorithms to learn more “history.” Specifically, the classical first order algorithm uses the gradient at the immediate previous solution. While the accelerated first order algorithms take advantage of the gradients at the previous two solutions, to learn from the “history.” Since it uses the historic information, it is also referred to as the *momentum* algorithm.

3 Review of Existing Algorithms to Solve Lasso

In this section, we presents five representative algorithms to solve Lasso: (1) iterative shrinkage threshold algorithm (ISTA), (2) fast iterative shrinkage-thresholding algorithms (FISTA), (3) coordinate gradient descent algorithm (CGDA), (4) smooth L1 algorithm (SLA) and (5) path following algorithm (PFA).

3.1 ISTA

ISTA (Daubechies et al., 2004) is a first order method (see Defition 2.7), which targets on the minimization of a summation of two functions

$$f(\beta) + g(\beta),$$

where $f(\beta) : \mathbb{R}^p \rightarrow \mathbb{R}$ is smooth convex with a Lipschitz continuous gradient and $g(\beta) : \mathbb{R}^p \rightarrow \mathbb{R}$ is continuous convex. If we let

$$f(\beta) = \frac{1}{2n} \|y - X\beta\|_2^2, \quad g(\beta) = \lambda \|\beta\|_1$$

with $f(\beta)$ ’s Lipschitz continuous gradient L taking the largest eigenvalue of matrix $X'X/n$, then we can verify that the objective function in Lasso $F(\beta)$ is a special case of ISTA.

To minimize $F(\beta)$, at the k -th iteration, ISTA updates $\beta^{(k+1)}$ from $\beta^{(k)}$ by using the quadratic approximation function of $f(\beta)$ at value $\beta^{(k)}$:

$$\beta^{(k+1)} = \arg \min_{\beta} f(\beta^{(k)}) + \langle (\beta - \beta^{(k)}), \nabla f(\beta^{(k)}) \rangle + \frac{\sigma_{\max}(X'X/n)}{2} \|\beta - \beta^{(k)}\|_2^2 + \lambda \|\beta\|_1. \quad (3)$$

Here $\sigma_{\max}(X'X/n)$ is the maximal eigen-value of the matrix $X'X/n$. Simple algebra shows that (ignoring constant terms in β), minimization of (3) is equivalent to the minimization problem in the following equation:

$$\beta^{(k+1)} = \arg \min_{\beta} \frac{\sigma_{\max}(X'X/n)}{2} \left\| \beta - \left(\beta^{(k)} - \frac{\frac{1}{n}(X'X\beta^{(k)} - X'y)}{\sigma_{\max}(X'X/n)} \right) \right\|_2^2 + \lambda \|\beta\|_1, \quad (4)$$

where the soft-thresholding function in equation (5) can be used to solve the problem in equation (4).

$$S(x, \alpha) = \begin{cases} x - \alpha, & \text{if } x \geq \alpha, \\ x + \alpha, & \text{if } x \leq -\alpha, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

Specifically, one can update $\beta^{(k+1)}$ from $\beta^{(k)}$ as

$$\beta^{(k+1)} = S \left(\beta^{(k)} - \frac{1}{n\sigma_{\max}(X'X/n)} (X'X\beta^{(k)} - X'y), \lambda/\sigma_{\max}(X'X/n) \right).$$

The summary of ISTA algorithm is presented in Algorithm 1.

Algorithm 1: Iterative Shrinkage-Thresholding Algorithms (ISTA)

Input: $y_{n \times 1}, X_{n \times p}, L = \sigma_{\max}(X'X/n)$
Output: $\beta^{(K)}$: an estimator of β after K iterations
1 initialization: $\beta^{(0)}$
2 for $k = 0, 1, \dots, K$ **do**
3 $\beta^{(k+1)} = S(\beta^{(k)} - \frac{1}{nL}(X'X\beta^{(k)} - X'y), \lambda/L)$

In addition to the implementation of ISTA, we also develop the convergence analysis of ISTA in the following equation (Beck and Teboulle, 2009, Theorem 3.1). To make it more clear, we list (Beck and Teboulle, 2009, Theorem 3.1) below with several changes of notation. The notations are changed to be consistent with the terminology that are used in this paper.

Theorem 3.1. *Let $\{\beta^{(k)}\}$ be the sequence generated by Algorithm 1. Then for any $k \geq 1$, we have*

$$F(\beta^{(k)}) - F(\hat{\beta}) \leq \frac{\sigma_{\max}(X'X/n) \|\beta^{(0)} - \hat{\beta}\|_2^2}{2k}. \quad (6)$$

Accordingly, the convergence rate of ISTA is $O(1/k)$.

From the above theorem and Algorithm 1, we find the computational complexity of ISTA is of order $O(kp^2)$, where k is the number of iterations and p is the number of coveriates. This is because that, line 3 in Algorithm 1 shows the number of operations in one loop of ISTA is $O(p^2)$. This is because that the main computation of each loop in ISTA is the matrix multiplication in $X'X\beta^{(k)}$. Note that the matrix $X'X$ can be pre-calculated and saved, therefore, in one loop, ISTA's order of computational complexity is $p(2p - 1)$ (Boyd and Vandenberghe, 2016). Accordingly, the computational complexity of ISTA is $O(kp^2)$.

3.2 FISTA

Motivated by ISTA, Beck and Teboulle (2009) developed an accelerated version FISTA. The major difference of ISTA and FISTA is that ISTA is a first order method (see Definition 2.7), which only uses the gradient at the immediate previous solution. On the other hand, FISTA is a accelerated first order algorithm method, which takes advantage of the gradients at previous two solutions, in order to learn from the history.

The updating rule of FISTA from $\beta^{(k)}$ to $\beta^{(k+1)}$ is articulated as follows. FISTA first employs an auxiliary variable $\alpha^{(k)}$ in the second-order Taylor expansion step (i.e., the one in equation (3)):

$$\beta^{(k+1)} = \arg \min_{\alpha} f(\alpha^{(k)}) + \langle (\alpha - \alpha^{(k)}), \nabla f(\alpha^{(k)}) \rangle + \frac{\sigma_{\max}(X'X/n)}{2} \|\alpha - \alpha^{(k)}\|_2^2 + \lambda \|\alpha\|_1, \quad (7)$$

where $\alpha^{(k)}$ is a specific linear combination of the previous two estimator $\beta^{(k-1)}, \beta^{(k-2)}$. In particular, we have

$$\alpha^{(k)} = \beta^{(k-1)} + \frac{t_{k-1} - 1}{t_k} [\beta^{(k-1)} - \beta^{(k-2)}].$$

In this way, FISTA constructs a “momentum” term $\beta^{(k-1)} - \beta^{(k-2)}$, and learns more historic information. This idea improves the computational complexity, as you will see in the reminder of this section. For completeness, we present FISTA in Algorithm 2.

Algorithm 2: Fast Iterative Shrinkage-Thresholding Algorithms (FISTA)

Input: $y_{n \times 1}, X_{n \times p}, L = \sigma_{\max}(X'X/n)$

Output: $\beta^{(K)}$: an estimator of β after K iterations

1 **initialization;**

2 $\beta^{(0)} \quad \alpha^{(1)} = \beta^{(0)}, t_1 = 1$

3 **for** $k = 1, \dots, K$ **do**

4 $\beta^{(k)} = S(\alpha^{(k)} - \frac{1}{nL}(X'X\alpha^{(k)} - X'y), \lambda/L)$

5 $t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$

6 $\alpha^{(k+1)} = \beta^{(k)} + \frac{t_k - 1}{t_{k+1}}(\beta^{(k)} - \beta^{(k-1)})$

FISTA has an improved convergence rate as $O(1/k^2)$, as compared $O(1/k)$ from ISTA (Beck and Teboulle, 2009, Theorem 4.4). To make it more clear, we list (Beck and Teboulle, 2009, Theorem 4.4) below with several changes of notation. The notations are changed to be consistent with the terminology that are used in this paper.

Theorem 3.2. *Let $\{\beta^{(k)}\}$ be a sequence generated by Algorithm 2. Then for any $k \geq 1$, we have that*

$$F(\beta^{(k)}) - F(\hat{\beta}) \leq \frac{2\sigma_{\max}(X'X/n)\|\beta^{(0)} - \hat{\beta}\|_2^2}{(k+1)^2}. \quad (8)$$

Accordingly, the convergence rate of FISTA is $O(1/k^2)$.

By combining the conclusion in the above theorem with Algorithm 2, we find FISTA's computational complexity after k iterations is $O(kp^2)$. This is because, the number of operations in one iteration of FISTA is still $O(p^2)$ (see line 4 in Algorithm 2). Although for both ISTA and FISTA, they have the same number of operation in one loop, FISTA has improved convergence rate than ISTA. Specifically, after running both k iterations, FISTA's output is more closer to the optima than that from ISTA, given its faster convergence rate.

3.3 CGDA

The aforementioned two algorithms (ISTA and FISTA) update their estimates globally in one iteration loop. The algorithm introduced in this section, CGDA, updates the estimates one coordinate at a time. Specifically it cyclically chooses one coordinate at a time and performs a simple analytical update. Such an approach is called *coordinate gradient descent*. This approach has been proposed for the Lasso problem for a number of times, but only after Friedman et al. (2010), was its power fully appreciated. Early research work on the coordinate descent include the discovery by Hildreth (1957) and Warga (1963), and the convergence analysis by Tseng (2001). There are research work done on the applications of coordinate descent on Lasso problems, such as Fu (1998), Shevade and Keerthi (2003), Friedman et al. (2007), Wu et al. (2008), and so on.

CGDA is widely used and the corresponding R package is named *glmnet* (Friedman et al., 2010). In CGDA, the updating rule from $\beta^{(k)}$ to $\beta^{(k+1)}$ is that, it optimizes with respect to only the j -th entry of $\beta^{(k+1)}$ for a selected $j = 1, \dots, p$. And the gradient at $\beta_j^{(k)}$ is used for the updating process:

$$\frac{\partial}{\partial \beta_j} F(\beta^{(k)}) = \frac{1}{n} \left(e_j' X' X \beta^{(k)} - y' X e_j \right) + \lambda \text{sign}(\beta_j) \quad (9)$$

where e_j is a vector of length p , whose entries are all zero except that the j -th entry is equal to 1. Imposing the gradient in (9) to be 0, we can solve for $\beta_j^{(k+1)}$ as follows:

$$\beta_j^{(k+1)} = S \left(y' X e_j - \sum_{l \neq j} (X' X)_{jl} \beta_l^{(k)}, n\lambda \right) / (X' X)_{jj},$$

where $S(\cdot)$ is the soft-thresholding function defined in (5) and $(X' X)_{ij}$ is the (i, j) -th entry of the matrix $X' X$. The above implementation is summarized in Algorithm 3.

Algorithm 3: Coordinate Gradient Descent Algorithm (CGDA)

Input: $y_{n \times 1}, X_{n \times p}, \lambda$
Output: $\beta^{(K)}$: an estimator of β after K iterations
1 initialization: $\beta^{(0)}$
2 for $k = 0, 1, \dots, K$ **do**
3 for $j = 1 \dots p$ **do**
4 $\beta_j^{(k+1)} = S \left(y' X e_j - \sum_{l \neq j} (X' X)_{jl} \beta_l^{(k)}, n\lambda \right) / (X' X)_{jj}$

As reflected by Corollary 3.8 in Beck and Tetrushvili (2013), we find the convergence rate of CGDA is $O(1/k)$. Here we list this corollary as a theorem below. We changed several notations to adopt the terminology in this paper.

Theorem 3.3. *Let $\{\beta^{(k)}\}$ be the sequence generated by in Algorithm 3. Then we have that*

$$F(\beta^{(k)}) - F(\hat{\beta}) \leq \frac{4\sigma_{\max}(X' X/n)(1+p)\|\beta^{(0)} - \hat{\beta}\|_2^2}{k + (8/p)}. \quad (10)$$

Accordingly, the convergence rate of CGDA is $O(1/k)$.

Compared with ISTA and FISTA, we find CGDA share the same order of convergence rate as ISTA, which is less efficient than FISTA. This is because that, both ISTA and CGDA are first order algorithm (see Definition 2.7), while the FISTA is the accelerated first order algorithm. For the first order algorithm, it utilize the previous gradient, while the accelerated first order algorithm takes advantage of the previous two gradients and learns more history from the previous two gradients. This momentum-learning ability allows FISTA to have faster convergence rate.

After reviewing the algorithm of CGDA, we develop its computational complexity. Firstly, the number of operations in each loop of CGDA is $O(p^2)$. It can be explained by the following two reasons. (i) While updating $\beta_j^{(k+1)}$ (line 4 in Algorithm 3), it costs $O(p)$ operations because of $\sum_{l \neq j} (X'X)_{jl} \beta_l^{(k)}$. (ii) From line 3 in Algorithm 3, we can see that all p entries of $\beta^{(k+1)}$ are updated one by one. Combining (i) and (ii), we can see that the number of operations need in one loop of CD is of the order $O(p^2)$. Therefore, CGDA's computational complexity after k iterations is $O(kp^2)$.

3.4 SLA

The aforementioned three methods (ISTA, FISTA, CGDA) targets directly at the minimization of $F(\beta)$. Different from them, SLA (Schmidt et al., 2007) aims to minimize a smooth surrogate of $F(\beta)$. And the surrogate commonly happens to the ℓ_1 penalty term. Recall the ℓ_1 penalty is essentially the absolute function:

$$|x| = \max\{x, 0\} + \max\{-x, 0\}, \quad \forall x \in \mathbb{R}. \quad (11)$$

And without much pain, one can find the non-differentiability of $|x|$ at the origin makes it hard to enable fast convergence rate when applying the gradient descent method.

To get rid of non-differentiability of $|x|$, Schmidt et al. (2007) proposed one of its surrogate function:

$$\begin{aligned} |x| &\approx \left[x + \frac{1}{\alpha} \log(1 + \exp(-\alpha x)) \right] + \left[-x + \frac{1}{\alpha} \log(1 + \exp(\alpha x)) \right] \\ &= \underbrace{\frac{1}{\alpha} [\log(1 + \exp(-\alpha x)) + \log(1 + \exp(\alpha x))]}_{\phi_\alpha(x)} \end{aligned} \quad (12)$$

Here α is a hyper-parameter controlling the closeness between $|x|$ and its surrogates $\phi_\alpha(x)$. The curve of this surrogate function $\phi_\alpha(x)$ is available in Figure 2. From this figure, one can tell that a large value of α makes better approximation to $|x|$. As suggested by Schmidt et al. (2007), to select an appropriate α , one can always begins from a small α where the quadratic approximation from $\phi_\alpha(x)$ is appropriate, and terminates at a sufficiently large value of α .

The above surrogate function is motivated by the non-negative projection operator $\max\{x, 0\}$ in (11), which can be smoothly approximated by the integral of a sigmoid function as shown in (12).

A nice feature of the surrogate function in (12) is its twice differentiability. Consequently, the AGD algorithm (see Definition 2.5) is applicable to the surrogate function:

$$F_\alpha(\beta) = \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \sum_{i=1}^p \phi_\alpha(\beta_i),$$

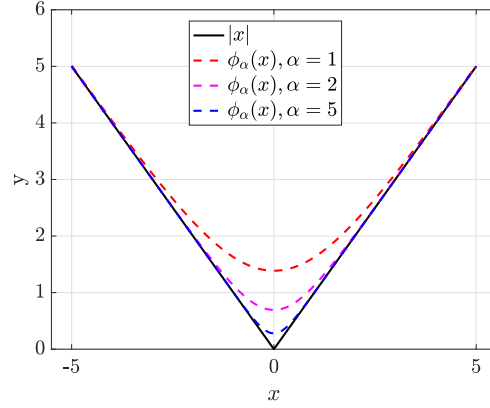


Figure 2: The closeness between $\phi_\alpha(x)$ and $|x|$ under different value of α

where $\phi_\alpha(\beta_i) = \frac{1}{\alpha} [\log(1 + \exp(-\alpha\beta_i)) + \log(1 + \exp(\alpha\beta_i))]$ for any $i = 1, \dots, p$. The pseudo code of SLA algorithm is displayed in Algorithm 4.

Algorithm 4: Smooth L1 algorithm (SLA)

Input: $y_{n \times 1}, X_{n \times p}, \mu = [\sigma_{\max}^2(X/\sqrt{n}) + \lambda\alpha/2]^{-1}$
Output: $\beta^{(K)}$: an estimator of β after K iterations
1 initialization: $\beta^{(0)}$
2 for $k = 1, \dots, K$ **do**
3 $w^{(k+1)} = \beta^{(k)} + \frac{k-2}{k+1}(\beta^{(k)} - \beta^{(k-1)})$
4 $\beta^{(k+1)} = w^{(k+1)} - \mu \nabla F_\alpha(w^{(k)})$

As proved by Mukherjee and Seelamantula (2016), the approximation error of $\beta^{(k)}$ in SLA is shown in equation (13).

Theorem 3.4. *Let $\{\beta^{(k)}\}$ be a sequence generated by Algorithm 4. Then we have*

$$F(\beta^{(k)}) - F(\hat{\beta}) \leq \frac{4 \|\beta^{(0)} - \hat{\beta}\|_2^2 \sigma_{\max}^2(\frac{X}{\sqrt{n}})}{k^2} + \frac{4\sqrt{2\lambda n \log 2} \|\beta^{(0)} - \hat{\beta}\|_2}{k}. \quad (13)$$

Accordingly, the convergence rate of SLA is $O(1/k)$.

From the above theorem, we find the convergence rate of SLA is decided by the summation of two terms. The first term origins from the convergence rate of AGD (recall Lemma 2.6). And the second term is caused by the difference between $F_\alpha(\beta)$ and $F(\beta)$: if one aims at the minimization of $F_\alpha(\beta)$, then the convergence rate is $O(1/k^2)$; however, if one aims at the minimization of $F(\beta)$, then the convergence rate is slowed by the difference between $F(\beta)$ and $F_\alpha(\beta)$.

For SLA's computational complexity, it mainly lies in the calculation of $\nabla F_\alpha(w) = \frac{X'X}{n}w - \frac{X'y}{n} + v$, where the v is a vector of length p , whose i -th entry is $\frac{-2}{w_i^2} \log(1 + e^{\alpha w_i}) + \frac{2\alpha e^{\alpha w_i}}{w_i(1 + e^{\alpha w_i})} - 1$. Accordingly, the main computational effort of one loop of SLA is the matrix multiplication in $X'Xw^{(k)}$, which cost $O(p^2)$ operations. Thus, SLA's computational complexity after k iterations is of order $O(kp^2)$.

3.5 PFA

Another representative algorithm to minimize $F(\beta)$ utilizes the path following idea (Park and Hastie, 2007; Rosset and Zhu, 2007; Tibshirani et al., 2011), and we call this type of algorithms as PFA. As the name suggests, PFA forms a path of the penalty parameter $\lambda_0, \lambda_1, \dots, \lambda_K$. Accordingly, it gets a sequence of β estimated under this sequence of λ . And we denote this sequence of $\hat{\beta}$ as $\hat{\beta}(\lambda_0), \hat{\beta}(\lambda_1), \dots, \hat{\beta}(\lambda_K)$.

The first key block of PFA is to identify the sequence of the the penalty parameter, i.e., $\lambda_0, \lambda_1, \dots, \lambda_K$. Before introducing the identification of λ sequence, we introduce a terminology called *support set*, which is useful in the identification.

Definition 3.5 (support set). *For any $\beta \in \mathbb{R}^p$, its support set is the collection of indexes, whose entries are non-zero:*

$$S(\beta) = \{i : \beta_i \neq 0\}.$$

Here β_i is the i -th entry of β . And $|S(\beta)|$ measures the number of elements in the set $S(\beta)$.

To identify the λ sequence, PFA begins with a large λ_0 , which makes the estimated $\hat{\beta}(\lambda_0) = 0$, and accordingly its support set is an empty set, i.e., $S(\hat{\beta}(\lambda_0)) = \emptyset$. Then it tries to identify a sequence of the penalty parameter λ as follows:

$$\lambda_0 > \lambda_1 > \lambda_2 > \dots > \lambda_{K-1} > \lambda_K = 0,$$

such that for any $k \geq 1$, when we have $\lambda \in [\lambda_k, \lambda_{k-1}]$, the support set of $\hat{\beta}(\lambda)$ (which is a function of λ) i.e., S_k , remains unchanged. Moreover, within the interval $[\lambda_k, \lambda_{k-1}]$, vector $\hat{\beta}(\lambda)$ elementwisely is a linear function of λ . However, when one is over the kink point, the support is changed/enlarged, i.e., we have $S_k \neq S_{k-1}$ or even $S_k \subseteq S_{k-1}$.

The second key block of PFA is the estimation of $\hat{\beta}(\lambda_k)$ given $\lambda_0, \dots, \lambda_k$. Instead of estimating β directly under λ_k , PFA takes advantage of the correlation between $\hat{\beta}(\lambda_k)$ and $\hat{\beta}(\lambda_{k-1})$. In the reminder of this section, we show this correlation and a concrete example is available in Section 3.5.1. For a general solution derived by PFA, we know $\hat{\beta}(\lambda)$ is the minimizer of (1). So it must satisfy the first order condition of (1):

$$q - \lambda \text{sign}(\hat{\beta}(\lambda)) = X'X\hat{\beta}(\lambda), \quad (14)$$

where $q = X'y$ and $\text{sign}(\hat{\beta}(\lambda))$ is a vector, whose i -th component is the sign function of $\hat{\beta}_i(\lambda)$:

$$\text{sign}(\hat{\beta}_i(\lambda)) = \begin{cases} 1 & \text{if } \hat{\beta}_i(\lambda) > 0 \\ -1 & \text{if } \hat{\beta}_i(\lambda) < 0 \\ [-1, 1] & \text{if } \hat{\beta}_i(\lambda) = 0 \end{cases}.$$

If we divide the indices of q, β, X into $S = \{i : \hat{\beta}_i(\lambda) \neq 0, \forall i = 1, \dots, p\}$ and its complements S^c , then we can rewrite (14) as

$$\begin{pmatrix} q_S \\ q_{S^c} \end{pmatrix} - \begin{pmatrix} \lambda \text{sign}(\hat{\beta}_S(\lambda)) \\ \lambda \text{sign}(\hat{\beta}_{S^c}(\lambda)) \end{pmatrix} = \begin{pmatrix} X'_S X_S & X'_S X_{S^c} \\ X'_{S^c} X_S & X'_{S^c} X_{S^c} \end{pmatrix} \begin{pmatrix} \hat{\beta}_S(\lambda) \\ 0 \end{pmatrix},$$

where $\hat{\beta}_S(\lambda)$ is the subvector of β only contains elements whose indices are in S and $\hat{\beta}_{S^c}(\lambda)$ is the complement of β_S . Besides, $\text{sign}(\hat{\beta}_S(\lambda))$ is the subset of $\text{sign}(\hat{\beta}(\lambda))$, only contains

the elements whose indices are in S , and $\text{sign}(\widehat{\beta}_{S^c}(\lambda))$ is the complement to $\text{sign}(\widehat{\beta}_S(\lambda))$. Matrix X_S is the columns of X whose indices are in S , and X_{S^c} is the complement of X_S .

Suppose we are interested in parameter estimated under λ and $\lambda - \Delta$, i.e., $\widehat{\beta}(\lambda)$, $\widehat{\beta}(\lambda - \Delta)$, for any $\Delta \in (0, \lambda)$. Then $\widehat{\beta}(\lambda)$, $\widehat{\beta}(\lambda - \Delta)$ must satisfy the following two system of equations:

$$\begin{cases} q_S - \lambda \text{sign}(\widehat{\beta}_S(\lambda)) &= X'_S X_S \widehat{\beta}_S(\lambda) \\ q_{S^c} - \lambda \text{sign}(\widehat{\beta}_{S^c}(\lambda)) &= X'_{S^c} X_S \widehat{\beta}_S(\lambda) \end{cases}, \quad (15)$$

$$\begin{cases} q_S - (\lambda - \Delta) \text{sign}(\widehat{\beta}_S(\lambda - \Delta)) &= X'_S X_S \widehat{\beta}_S(\lambda - \Delta) \\ q_{S^c} - (\lambda - \Delta) \text{sign}(\widehat{\beta}_{S^c}(\lambda - \Delta)) &= X'_{S^c} X_S \widehat{\beta}_S(\lambda - \Delta) \end{cases}. \quad (16)$$

From the above two system of equations, we have the following:

$$-(\lambda - \Delta) \text{sign}(\widehat{\beta}_{S^c}(\lambda - \Delta)) = -\lambda \text{sign}(\widehat{\beta}_{S^c}(\lambda)) + \Delta X'_{S^c} X_S (X'_S X_S)^{-1} \text{sign}(\widehat{\beta}_S(\lambda)). \quad (17)$$

That is, if one decreases λ to $\lambda - \Delta$, one must strictly follow (17). The above equation is useful to find the support set of $\widehat{\beta}(\lambda - \Delta)$. After the support set is available, one can use the linear regression method, restricted to the support set, to get the estimation of $\widehat{\beta}(\lambda - \Delta)$.

$$-(\lambda - \Delta) \text{sign}(\widehat{\beta}_{S^c}(\lambda - \Delta)) = -\lambda \text{sign}(\widehat{\beta}_{S^c}(\lambda)) + \Delta X'_{S^c} X_S (X'_S X_S)^{-1} \text{sign}(\widehat{\beta}_S(\lambda)).$$

The pseudo code of the path-following algorithm is summarized as in Algorithm 5.

Algorithm 5: Path following algorithm (PFA)

Input: $y_{n \times 1}, X_{n \times p}, \lambda$
Output: an estimator of β under penalty parameter λ
1 initialization: $\lambda^{(0)}, k = 0$
2 while $\lambda^{(k)} > \lambda$ **do**
3 $\lambda_{k+1} = \sup \left\{ \lambda : \begin{array}{l} \lambda < \lambda_k \text{ and} \\ \forall \lambda', \lambda'' \in (\lambda, \lambda_k), S(\lambda') = S(\lambda'') \text{ and} \\ \forall \lambda' \in [0, \lambda), S(\lambda') \neq S(\lambda) \end{array} \right\}$
4 $\widehat{\beta}(\lambda_{k+1}) = \arg \min_{\beta} \left\{ \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda_{k+1} \|\beta\|_1 \right\}$
5 $k = k + 1$

For the computational effort, it mainly decided by the length of the λ sequence, i.e., K . If K is small, then PFA is efficient: it only requires $O(nKp^2)$ numerical operations. Compared with ISTA, FISTA, CDGA and SLA, when their number of iterations k is larger than nK , then PFA is more computationally efficient theoretically. In particular, if the size of supports are strictly increasing, i.e., we have

$$|S_{k-1}| < |S_k| \quad \forall k \geq 1,$$

then we have $K \leq p$, and accordingly the total number of numerical operations of PFA can be bounded by $O(np^3)$. Under this scenario, PFA is theoretically faster than ISTA, FISTA, CSDA and SLA, if they iterate more than np iterations.

Yet, the contemporary literature indicates that the upper bound of K is an open question (Tibshirani et al., 2011; Rosset and Zhu, 2007). With unknown K , it is not

theoretically guaranteed PFA converges. And it is possible that its convergence rate is low, considering that the maximum number of K can be as large as 2^p .

In addition to the unpredictable convergence rate, PFA has another limitation: it doesn't work for general cases. The current literature only establishes PFA in special situations. Yet, it might fail to deliver the solution under some cases. In Section 3.5.1, we give a counter example where PFA fails.

3.5.1 A Counter Example where PFA Fails

In this section, we offer a concrete counter example where PFA fails. This counter example represents a general category of design matrix X and coefficient β . And we use the following counter example to argue that PFA does not work in the most general setting.

The counter example is designed as follows. Suppose

$$\beta_1 > \beta_2 > \beta_3 > \beta_4 = \beta_5 = \dots = \beta_p = 0.$$

The model matrix $X = (X_1, X_2, X_3, \dots, X_p)$, where $X_1, X_2 \in \mathbb{R}^n$ is the first two columns from a orthogonal matrix $(X_1, X_2, \tilde{X}_3, \dots, \tilde{X}_p)$, and for $j \geq 3$, we have $X_j = \alpha_j X_1 + (1 - \alpha_j) X_2 + \sqrt{1 - \alpha_j^2 - (1 - \alpha_j)^2} \tilde{X}_j$ with $\alpha_j \in (0, 1)$. The response vector y is generated by

$$y = \sum_{j=1}^p \beta_j X_j.$$

If β_1, β_2 are very large number, say, 200, 100, and β_3 is not that large, say, 1. Then PFA works as follows:

- Loop 0: We start with $\lambda_0 = +\infty$, then we know that $\hat{\beta}(\lambda_0) = 0$ and $S_0 = \emptyset$.
- Loop 1: When λ changes from $\lambda_0 = +\infty$ to $\lambda_1 = \|q\|_\infty$, from (14), we know that $S_1 = \{1\}$.
- Loop 2: Similar to the first loop, when λ decrease to λ_2 , we have $S_2 = \{1, 2\}$.
- Loop 3: This is where problem happens. From (17), we know that $\forall \lambda_2 - \Delta \in (\lambda_3, \lambda_2]$, we have

$$\text{sign}(\hat{\beta}_{S_2^c}(\lambda - \Delta)) = X_{S_2}^T X_{S_2} (X_{S_2}^T X_{S_2})^{-1} \text{sign}(\hat{\beta}_{S_2}(\lambda)).$$

Since $\text{sign}(\hat{\beta}_{S_2}(\lambda)) = (1, 1)'$ and $X_{S_2} = (X_1, X_2)$, $X_{S_2^c} = (X_3, \dots, X_p)$, we have the right hand side of the above equation as a all-one vector, i.e, $(1, 1, \dots, 1)'$. To make the left hand side $\text{sign}(\hat{\beta}_{S_2^c}(\lambda_2 - \Delta))$ equals to $(1, 1, \dots, 1)'$, we can only take $\Delta = \lambda_2$, which gives us $S_3 = \{1, 2, 3, \dots, p\}$.

However, from the data generalization, we know that the true support set is $\{1, 2, 3\}$. Therefore, one will not be able to develop a PFA to realize correct support set recovery. At least not in the sense of inserting one at a time to the support set. In the above example, since a path following approach can only visit three possible subsets, it won't solve the Lasso in general.

3.6 LARS

In the statistical community, there has been an algorithm universally used in the last decades. It is called LARS, which can be regarded as an advanced forward selection method. It is originally developed by Efron et al. (2004) and later a R package named *lars* full filled its implementations. Nowadays, the LARS has decreasing popularity given its limitation in computation efficiency. So we will briefly introduce this algorithm in this review.

The main idea of LARS is articulated as follows. We start with all coefficients equal to zero and find the predictor most correlated with the response, say x_{j_1} . We take the largest step possible in the direction of this predictor until some other predictor, say x_{j_2} , has as much correlation with the current residual. At this point LARS parts company with forward selection. Instead of continuing along x_{j_1} , LARS proceeds in a direction equiangular between the two predictors until a third variable x_{j_3} earns its way into the “most correlated” set. LARS then proceeds equiangularly between x_{j_1} , x_{j_2} and x_{j_3} , that is, along the “least angle direction,” until a fourth variable enters, and so on.

Its detailed implementation is listed as follows (Tibshirani, 2009).

- Step 1: start with all coefficients β equal to zero.
- Step 2: find the predictor x_j most correlated with y .
- Step 3: increase the coefficient β_j in the direction of the sign of its correlation with y . Take residuals $r = y - \hat{y}$ along the way. Stop when some other predictor x_k has as much correlation with r as x_j has.
- Step 4: increase (β_j, β_k) in their joint least squares direction, until some other predictor x_m has as much correlation with the residual r .
- Step 5: increase $(\beta_j, \beta_k, \beta_m)$ in their joint least squares direction, until some other predictor x_n has as much correlation with the residual r .
- ...

The above procedure continues until all predictors are in the model. However, to our best knowledge, it is not clear when LARS stops, viewing from the existing literature. The LARS procedure may take many steps before it stops (see examples in Turlach (2004)). And to be worse, LARS is not workable for all cases (Turlach, 2004). Given this, the LARS algorithm is not as frequently used as the other algorithms reviewed in Section 3.1 - 3.4.

4 Conclusions

Lasso is a regression method, which can realize both variable selection and regularization. When one aims at the minimization of the objective function in Lasso, the ℓ_1 penalty raises the computational challenge due to its non-differentiability. To overcome this challenge, various iterative algorithms are proposed, including first order algorithm (like ISTA, CSDA, SLA) and accelerated first order algorithm (like FISTA). Additionally, the path following idea is utilized to solve Lasso (like PFA). Comparing the convergence rate of the five algorithms, we find FISTA gives the best and relatively robust performance. Specifically, FISTA’s convergence rate is $O(1/k^2)$, while the convergence rate of ISTA, CGDA and SLA

are all $O(1/k)$. For PFA, its might be faster than FISTA under some special cases. However, there is no theoretical guarantee that it is faster than FISTA under general cases (see a counter example in Section 3.5.1). The above comparison is summarized in Table 1 and hope this comprehensive summary helps readers to learn more details about optimization with ℓ_1 penalty and to facilitate their future research.

Table 1: Pros and cons of the five reviewed algorithm to solve Lasso

	Pros	Cons
ISTA	Convergence rate is $O(1/k)$.	Computationally inefficient than FISTA or PFA under special cases.
FISTA	Convergence rate is $O(1/k^2)$ which is faster than ISTA, CGDA and SLA.	Could be computationally inefficient than PFA under special cases.
CGDA	Convergence rate is $O(1/k)$.	Computationally inefficient than FISTA or PFA under special cases.
SLA	Convergence rate is $O(1/k)$.	(1) Computationally inefficient than FISTA or PFA under special cases; (2) The output is not the exactly $\hat{\beta}$ desired in Lasso.
Path-following	(1) Could be computationally more efficient than FISTA under special cases. (2) Once the solution path is available, one can get the estimation of β under any value of λ .	(1) Not workable for generalized cases; (2) Computation complexity can be unbounded.

Funding Information

This project is partially supported by the Transdisciplinary Research Institute for Advancing Data Science (TRIAD), <http://triad.gatech.edu>, which is a part of the TRIPADS program at NSF and locates at Georgia Tech, enabled by the NSF grant CCF-1740776. The authors are also partially sponsored by NSF grants 1613152 and 2015363.

Acknowledgments

The authors would like to thank the editors of WIREs Computational Statistics for reviewing our proposal, and the editorial office for tracking our paper submission status.

References

- Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202.
- Beck, A. and Tetrushvili, L. (2013). On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060.
- Boyd, S. and Vandenberghe, L. (2016). Numerical linear algebra background. 2010. <http://www.seas.ucla.edu/~vandenbe/ee236b/lectures/num-lin-alg.pdf>.

- Daubechies, I., Defrise, M., and De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457.
- Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *The Annals of statistics*, 32(2):407–499.
- Friedman, J., Hastie, T., Höfling, H., Tibshirani, R., et al. (2007). Pathwise coordinate optimization. *The annals of applied statistics*, 1(2):302–332.
- Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.
- Fu, W. J. (1998). Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, 7(3):397–416.
- Hildreth, C. (1957). A quadratic programming procedure. *Naval research logistics quarterly*, 4(1):79–85.
- Lan, G. (2019). Lectures on optimization. methods for machine learning. *H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA*.
- Li, H. and Lin, Z. (2015). Accelerated proximal gradient methods for nonconvex programming. In *Advances in neural information processing systems*, pages 379–387.
- Mukherjee, S. and Seelamantula, C. S. (2016). Convergence rate analysis of smoothed Lasso. In *Communication (NCC), 2016 Twenty Second National Conference on*, pages 1–6. IEEE.
- Nesterov, Y. (2005). Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152.
- Nesterov, Y. (2013). Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161.
- Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547.
- Park, M. Y. and Hastie, T. (2007). L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677.
- Rosset, S. and Zhu, J. (2007). Piecewise linear regularized solution paths. *The Annals of Statistics*, pages 1012–1030.
- Santosa, F. and Symes, W. W. (1986). Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330.
- Schmidt, M., Fung, G., and Rosales, R. (2007). Fast optimization methods for L1 regularization: A comparative study and two new approaches. In *European Conference on Machine Learning*, pages 286–297. Springer.

- Shevade, S. K. and Keerthi, S. S. (2003). A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics*, 19(17):2246–2253.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.
- Tibshirani, R. (2009). A simple explanation of the lasso and least angle regression. Technical report, Technical report, Stanford University.[Online.
- Tibshirani, R. J., Taylor, J., et al. (2011). The solution path of the generalized Lasso. *The Annals of Statistics*, 39(3):1335–1371.
- Tseng, P. (2001). Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494.
- Turlach, B. A. (2004). [least angle regression]: Discussion. *The Annals of Statistics*, 32(2):481–490.
- Warga, J. (1963). Minimizing certain convex functions. *Journal of the Society for Industrial and Applied Mathematics*, 11(3):588–593.
- Wu, T. T., Lange, K., et al. (2008). Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244.