Solving LASSO : A Convex Optimization Perspective

Report for the course *Convex Optimization*

Boen Jiang

Abstract

This report first introduces the LASSO problem and its equivalent form, leading to the ℓ^1 regularization problem. Then, by introducing **strong duality** and the method of **conjugate functions**, the dual problem of the ℓ^1 regularization problem is derived.

In numerical simulations, the ℓ^1 regularization problem and its dual problem are first solved using the commercial solver `Gurobi`, and the relative error of the solution compared with the theoretical optimal solution is compared. Subsequently, this article considers other gradient-based methods for solving the ℓ^1 regularization problem, including **Huber smoothing approximation**, **proximal gradient method**, and **coordinate descent method**, Numerical results are presented to compare the performance of these methods under different scenarios.

Contents

1 LASSO and its equivalent problem

1.1 Sparity and LASSO

In linear regression, one may consider a dataset with N samples, denoted as $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{ip})^\top \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. After centering the columns of the covariate matrix, i.e.,

$$\frac{1}{N}\sum_{i=1}^N x_{ij} = 0, \quad j = 1, 2, ..., p,$$

and centering y_i such that

$$\frac{1}{N}\sum_{i=1}^N y_i = 0,$$

the intercept term can be omitted. [Tib96] defined the LASSO problem as the solution $\hat{\beta}^{\text{lasso}}$ to the optimization problem (1):

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{2N}\sum_{i=1}^N (y_i - x_i^T\beta)^2, \text{ subject to } \sum_{j=1}^p |\beta_j| \le t. \tag{1}$$

[OPT00] considers an equivalent formulation of problem (1), given by (2):

$$\min_{\beta\in\mathbb{R}^p} \left\{ \frac{1}{2N}\sum_{i=1}^N (y_i - x_i^T\beta)^2 + \lambda\|\beta\|_1 \right\} \tag{2}$$

Problems (1) and (2) are equivalent. That is, given λ in problem (2), there exists t such that the solution $\hat{\beta}^{\text{lasso}}$ to problem (1) is the solution to problem (2), and vice versa. [Din24] pointed out that the one-to-one correspondence between the two can be obtained by $t = \sum_{j=1}^p |\hat{\beta}_j^{\text{lasso}}|$. [Tib12] further analyzed the uniqueness conditions of the LASSO solution.

The following parts of this article will be organized as follows:

1. Prove the equivalence of the problems (1) and (2) under strong duality.
2. Derive the dual problem of the ℓ^1 regularization problem (2) using the method of conjugate functions.
3. Numerical simulations to compare the performance of different methods for solving the ℓ^1 regularization problem.

1.2 Prove the equivalence of the problems under strong duality

The LASSO problem (1) can be written in the standard form of the optimization problem as follows:

$$\begin{aligned} &\text{minimize} && f(\beta) \\ &\text{subject to} && g(\beta) \le 0. \end{aligned} \tag{3}$$

where

$$f(\beta) = \frac{1}{2N}\|\mathbf{y} - \mathbf{X}\beta\|_2^2, \quad g(\beta) = \sum_{j=1}^p |\beta_j| - t. \tag{4}$$

This is a convex optimization problem.

Generally, we assert the following proposition 1.

Proposition 1. *Consider the standard form of the optimization problem:*

$$\min_{x\in\mathbb{R}^n} f(x) \quad s.t. \quad h(x) \le \tau, \tag{5}$$

where $f:\mathbb{R}^n \to \mathbb{R}$ and $h:\mathbb{R}^n \to \mathbb{R}$ are convex functions, and $\tau \in \mathbb{R}$. Given strong duality, let x^* be the solution to the standard optimization problem (5), which satisfies the feasibility condition, i.e., $h(x^*) \le \tau$. We claim that x^* is also the solution to the regularized problem (6) and vice versa.

$$\min_{x\in\mathbb{R}^n} f(x) + \lambda h(x). \tag{6}$$

The proof of Proposition 1 is provided in the appendix A.1. Therefore, by applying Proposition 1, we can conclude that the LASSO problem (1) is equivalent to the ℓ^1 regularization problem (2).

2 Dual problem for ℓ^1 regularization

[BV04] pointed out that we are facing an ℓ^1 regularization problem.

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-b\|_2^2 + \lambda\|x\|_1, \tag{7}$$

where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, \lambda \ge 0$.

2.1 Conjugate function of a norm

By the definition of the conjugate function, the conjugate function f^* of an appropriate function f is defined as

$$f^*(y) = \sup_{x\in\mathbf{dom} f} \{y^{\mathrm{T}}x - f(x)\}.$$

Now let's consider the function $f(x) = \|x\|$, where $\|x\|$ is some norm.

We assert the following lemma:

Lemma 1. *For any norm $\|x\|$, its conjugate function is*

$$f^*(y) = \begin{cases} 0, & \|y\|_* \le 1 \\ +\infty, & \|y\|_* > 1 \end{cases} \tag{8}$$

where $\|y\|_*$ is the dual norm of $\|x\|$. Moreover, the dual norm of the ℓ^1 norm is the ℓ^∞ norm.

The proof of Lemma 1 is provided in the appendix A.2.

2.2 Obtain the dual problem via variable substitution and conjugate functions

We assert the following proposition 2. The proof is provided in the appendix A.3.

Proposition 2. *For the regularization problem (7), the dual problem of the ℓ^1 regularization problem is*

$$\max_{\mu\in\mathbb{R}^m} \left\{ b^\top\mu - \frac{1}{2}\|\mu\|_2^2 \right\} \quad \textit{subject to } \|A^\top\mu\|_\infty \le \lambda. \tag{9}$$

Geometrically, the interpretation of the dual problem is shown in Figure 1.

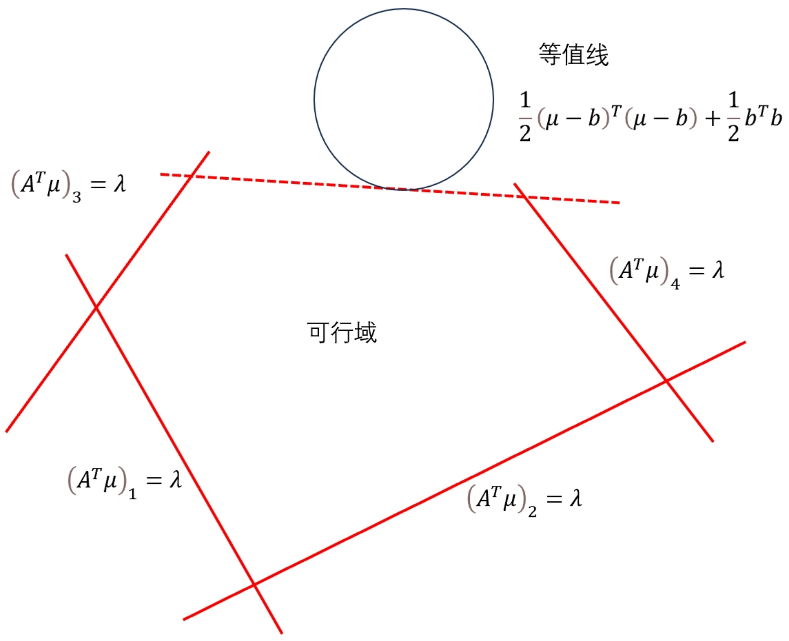

Figure 1: The Lagrange dual function of the ℓ^1 regularization, where the feasible region is $\mathcal{F}_\lambda = \{\mu : \|A^\top \mu\|_\infty \leq \lambda\}$. Clearly, the solution to the unconstrained dual problem is b. The constrained dual problem is to find a point in $\mathcal{F}_\lambda$ that maximizes $b^\top \mu - \frac{1}{2}\|\mu\|_2^2$. The tangential point in the figure prepresents the solution to the dual problem.

3 Data generation and performance comparison

3.1 Data generating process

For the regularization problem (7), consider the following data generating process. Assume $A \in \mathbb{R}^{m\times n}$, where $m = 256, n = 512$. Therefore, this is a "wide matirx", with the number of features n being greater than the number of samples m. The matrix A can be generated from a standard normal distribution. Sine lasso addresses a sparse problem, we assume that the solution is sparse with only 10% of the elements being non-zero. Then b is generate as $b = Au$. The code implementation is shown in the appendix B.1.

3.2 Performance comparison

In general, different solution methods will yield the optimal solution x^* and the optimal value $f(x^*) = \frac{1}{2}\|Ax^* - b\|_2^2 + \lambda\|x^*\|_1$ with a certain computation time.

Since the true (oracle) solution u is attainable in the data generating process, the theoretical optimal value $f(u)$ can be computed. To compare the optimal values, the relative error can be calculated using equation (10).

$$\text{Err} = \left|\frac{f(x^*) - f(u)}{f(u)}\right|. \tag{10}$$

For the optimal solution, since the solution is a sparse vector, [HKP12] suggested using the cosine similarity (11) to compare two sparse vectors.

$$\text{sim}(x^*, u) = \frac{x^* \cdot u}{\|x^*\|_2\|u\|_2}. \tag{11}$$

The "error" of the solution can then be interpreted as $1 - \text{sim}(x^*, u)$. Further more, we can plot the convergence curve of the objective function value with respect to the iteration number

Finally, regarding computation time, the CPU running time can be used for comparison.

4 Solving the ℓ^1 regularization problem

4.1 Solving the ℓ^1 regularization problem using `Gurobi`

4.1.1 Solving the primal problem using `Gurobi`

The `Gurobi` solver requires variable substitution for the ℓ^1 norm part when solving the ℓ^1 regularization problem. By introducing a new variable z, the problem (7) can be transformed into the following equivalent optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|Ax-b\|_2^2 + \lambda \sum_{i=1}^{n} z_i \\ \text{subject to} & z_i \geq x_i, i = 1, 2, ..., n, \\ & z_i \geq -x_i, i = 1, 2, ..., n. \end{array} \tag{12}$$

This is because if $z_i \geq x_i$ and $z_i \geq -x_i$, then $z_i \geq |x_i|$, and hence $\sum_{i=1}^{n} z_i \geq \sum_{i=1}^{n} |x_i|$. Therefore, the solution to the problem (7) is equivalent to the solution of problem (12). The code for solving the primal problem using `Gurobi` is provided in the appendix B.2.

4.1.2 Explaination of the results

In the above code, we solved the problem (12) with $\lambda = 0.1$. The optimal value obtained is 4.512. We found that the relative error of the optimal value is 2.621×10^{-4}, and the cosine error of the solution is 7.477×10^{-8}. The computation time is 33.8279 seconds, and the number of iterations is 13.

This indicates that solving the ℓ^1 regularization problem using `Gurobi` is feasible. However, the computation time is relatively long. This may be due to the fact that `Gurobi` is a general-purpose optimization solver and cannot take full advantage of the sparse structure of the lasso problem. Therefore, it is necessary to compare the performance of different methods for solving the ℓ^1 regularization problem.

4.1.3 Solving the dual problem using `Gurobi`

Based on the previous derivation, the dual problem (21) can also be solved using `Gurobi`. For the constraint $\|A^\top \mu\|_\infty \leq \lambda$, we can express it in the following equivalent form:

$$-\lambda \leq \left(A^\top \mu\right)_i \leq \lambda, \quad \forall i = 1, 2, \ldots, n.$$

Here, $\left(A^\top \mu\right)_i$ represents the i-th element of the vector $A^\top \mu$. The code for solving the dual problem is provided in the appendix B.3.

4.1.4 Explaination of the results

The computation for solving the dual problem is relatively fast, taking only 3.718 seconds. The optimal value obtained is 4.334. The relative error of the optimal value is 3.96%, and the number of iterations is 18.

Since the dimension of the optimization variables in the dual problem is m, which is half of the dimension of the optimization variables in the primal problem, the dual problem is solved faster than the primal problem.

Although the Slater condition is satisfied, then strong duality holds, the optimal value of the dual problem deviates from the optimal value of the primal problem, resulting in a relatively large relative error.

4.2 Solving the ℓ^1 regularization problem using gradient descent

4.2.1 Huber smoothing

Since the objective function of the ℓ^1 regularization problem (7) is not smooth, the gradient cannot be computed directly for some points, making it difficult to apply direct gradient-based optimization methods.

The non-smooth term is $\|x\|_1$, which is the sum of the absolute values of the components of x . To address this, we consider the following one-dimensional smooth function (the Huber function):

$$l_\delta(x) = \begin{cases} \frac{1}{2\delta}x^2, & |x| < \delta, \\ |x| - \frac{\delta}{2}, & \text{otherwise.} \end{cases}$$

This definition is a modification of the Huber loss function. As $\delta \to 0$, the smooth function $l_\delta(x)$ and the absolute value function $|x|$ become closer. The Huber function is shown in Figure 2.

Therefore, we transform the problem into a smoothing problem:

$$\min \quad f_\delta(x) = \frac{1}{2}\|Ax - b\|^2 + \mu L_\delta(x), \quad \text{where} \quad L_\delta(x) = \sum_{i=1}^{n} l_\delta(x_i), \tag{13}$$

where δ is a given smoothing parameter.

The gradient of $f_\delta(x)$ is given by

$$\nabla f_\delta(x) = A^\top(Ax - b) + \mu \nabla L_\delta(x),$$

where $\nabla L_\delta(x)$ is defined element-wise as:

$$(\nabla L_\delta(x))_i = \begin{cases} \operatorname{sign}(x_i), & |x_i| > \delta \\ \frac{x_i}{\delta}, & |x_i| \le \delta \end{cases}$$

The gradient of $f_\delta(x)$ is Lipschitz continuous, with the corresponding constant $L = \left\|A^\top A\right\|_2 + \frac{\mu}{\delta}$. Clearly, f_δ is strongly convex,with the corresponding strong convexity constant $m = \frac{1}{2}\|A\|_2^2$. Therefore, gradient descent can be guaranteed to converge in this case. The code for solving the smoothed ℓ^1 regularization problem using the gradient descent method is provided in the appendix B.4.

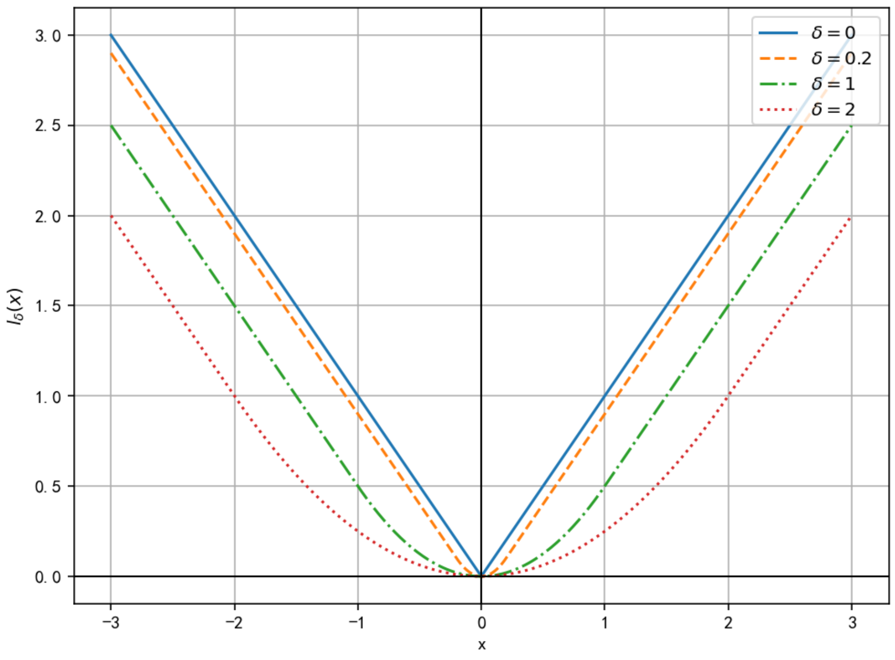

Figure 2: Different curves of $l_\delta(x)$ for different values of δ.

4.2.2 Numerical results

This study examines the performance of the gradient descent method for two cases: $\delta = 0.01$ and $\delta = 0.001$. From Figure 3, it can be observed that the gradient method exhibits linear convergence. The relative error decreases as the number of iterations increases. However, the convergence rate is slow, and the algorithm requires a large number of iterations to reach the convergence condition, which indicates that smootghing the ℓ^1 regularization problem is not an efficient approach. For $\delta = 0.01$, the iteration time is 68.595 seconds, while for $\delta = 0.0001$, the iteration time is 103.610 seconds.

Additionally, we demonstrate the case where the backtracking line search parameters are not properly set. When the parameter α is set to 1, the algorithm does not converge, as shown in Figure 4. The algorithm stops after 5 iterations, taking 0.1162 seconds, and fails to converge.

4.3 Solving the ℓ^1 regularization problem using proximal gradient method

4.3.1 Composite functions and proximal gradient descent

For a convex function h, its proximal operator is defined as

$$\operatorname{prox}_h(x) = \underset{u}{\operatorname{argmin}} \left(h(u) + \frac{1}{2}\|u - x\|_2^2 \right)$$

We assert the following lemma 2, which is proved in the appendix A.4.

Lemma 2. *$u = \operatorname{prox}_h(x)$ is equivalent to $x - u \in \partial h(u)$.*

Now taking $h(x) = \|x\|_1$, then the optimality condition for the proximity operator $u = \operatorname{prox}_{th}(x)$

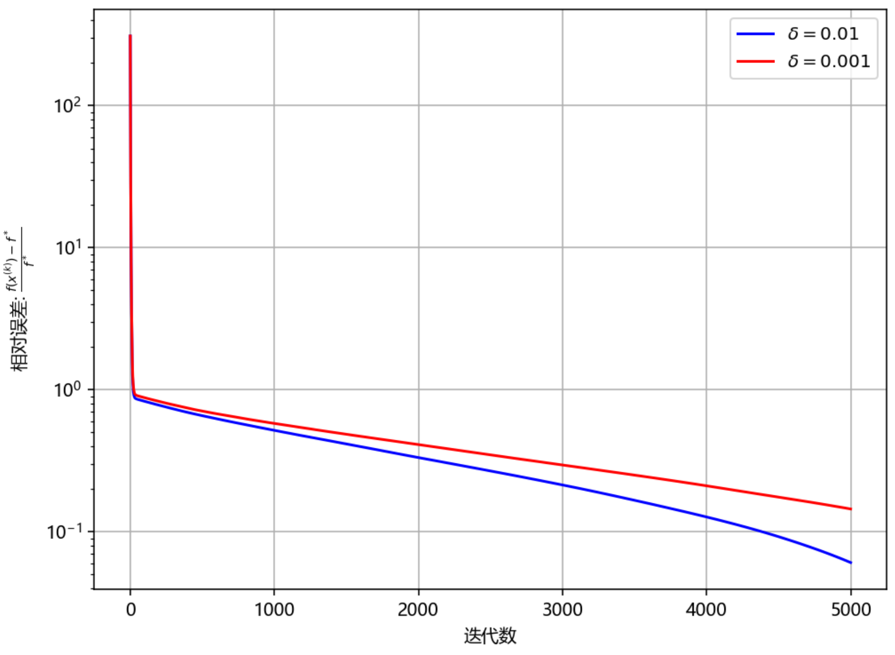

Figure 3: Iterative process of solving the smoothed ℓ^1 regularization problem using the gradient descent method.

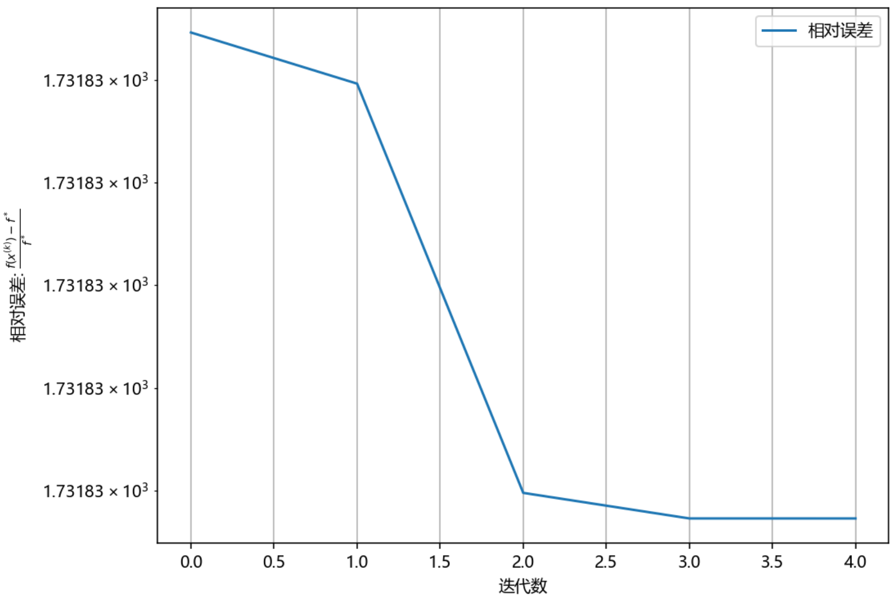

Figure 4: Iterative process of solving the smoothed ℓ^1 regularization problem using the gradient descent method with improper backtracking line search parameters.

is

$$x - u \in t\partial\|u\|_1 = \begin{cases} \{t\}, & u > 0 \\ [-t, t], & u = 0 \\ \{-t\}, & u < 0 \end{cases}$$

which implies

$$\operatorname{prox}_{th}(x) = \operatorname{sign}(x) \max\{|x| - t, 0\}.$$

Proximal gradient method is a first-order optimization algorithm for solving the optimization problem:

$$\min \quad \psi(x) = f(x) + h(x),$$

where f is differentiable, with its domian $\operatorname{dom} f = \mathbb{R}^n$, Function h is convex, and can be non-smooth, with an easily obtained proximal operator. In this sense, ℓ^1 is a typical example for the proximal gradient method. Its algorithm can be written as Algorithm 1. This is equivalent to performing explicit gradient descent on the smooth part and implicit gradient descent on the non-smooth part [Ber15]. When the gradient of f is Lipschitz continuous, with Lipschitz constant L, the proximal gradient method only requires $t_k \leq \dfrac{1}{L}$ to ensure convergence.

Algorithm 1 Proximal Gradient Method

```
Require: function $f(x), h(x)$, initial point $x^0$. Initialize $k = 0$.
1: while not converged do
2:     $x^{k+1} = \operatorname{prox}_{t_k h}\left(x^k - t_k \nabla f\left(x^k\right)\right)$.
3:     $k = k + 1$.
4: end while
```

For the ℓ^1 regularization problem (7),

$$\nabla f(x) = A^\top(Ax - b)$$
$$\operatorname{prox}_{t_k h}(x) = \operatorname{sign}(x) \max\left\{|x| - t_k\lambda, 0\right\}.$$

The proximal gradient method for solving the ℓ^1 regularization problem can be given by the following iterative formula:

$$y^k = x^k - t_k A^\top\left(Ax^k - b\right),$$
$$x^{k+1} = \operatorname{sign}\left(y^k\right) \max\left\{\left|y^k\right| - t_k\lambda, 0\right\},$$

where t_k is the step size. Namely, the proximal gradient method run gradient descent on the smooth part and proximal operator on the non-smooth part. The code for solving the ℓ^1 regularization problem using the proximal gradient method is provided in the appendix B.5.

4.3.2 Numerical results

The algorithm required 18292 iterations to converge, taking 6.236 seconds. This means that, although the convergence speed of the algorithm is relatively slow, it can still converge in a relatively short of time in this problem. At this point, the relative error of the optimal value is 6.528×10^{-7}, and the cosine error of the solution is 4.875×10^{-7}.

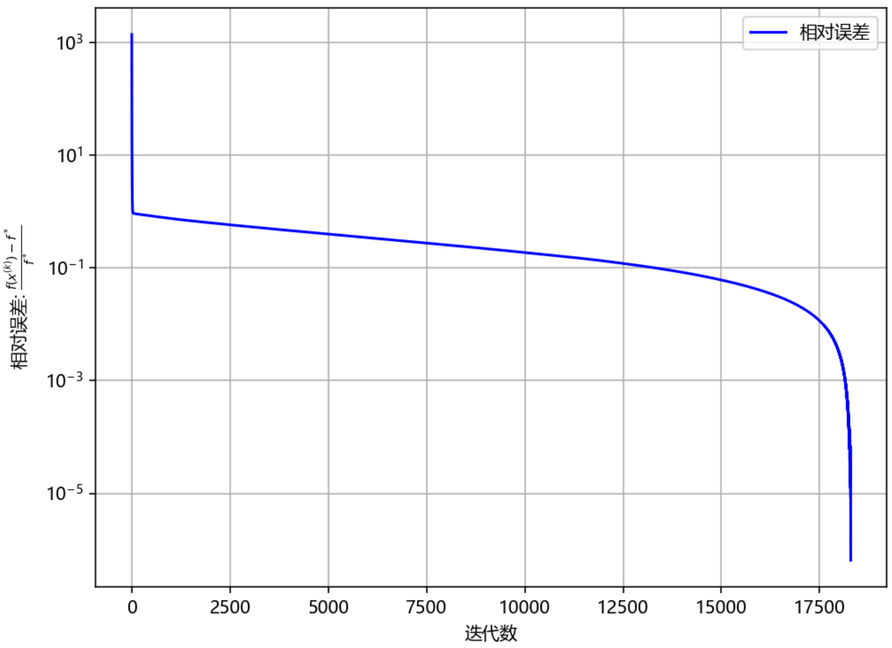

Figure 5: Iterative process of solving the ℓ^1 regularization problem using the proximal gradient method.

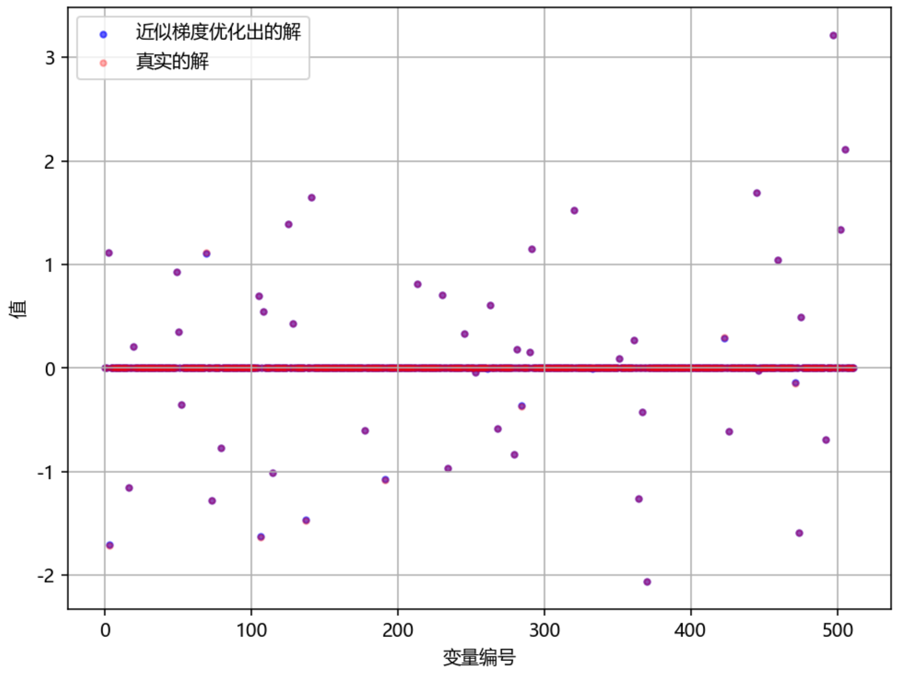

Figure 6: Comparison of the variables obtained by the proximal gradient method with the oracle solutions.

4.4 Solving the ℓ^1 regularization problem using coordinate descent method

4.4.1 Soft-thresholding and coordinate descent

Coordinate descent is particularly suitable for ℓ^1 regularization problems, and this method can be used to solve the lasso problem quickly. For an initial value $x^{(0)} \in \mathbb{R}^n$, the update formula for the coordinate descent method is given by:

$$x_i^{(k)} = \operatorname{argmin}_{x_i} f\left(x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)}\right), \text{ where } i = 1, \ldots, n \text{ and } k = 1, 2, 3 \ldots \tag{14}$$

Here, i represents the i-th element in the vector, and k denotes the iteration number. In other words, we minimize the function f with respect to one element x_i, then substitute it back into the function f to update the next element.

Here, we consider $f(x) = g(x) + \sum_{i=1}^{n} h_i(x_i)$, where g is a smooth convex function, and h_i is a convex non-smooth function. Thus, this method is applicable to the ℓ^1 regularization problem.

[Tse01] proved that when f is continuous on the compact set $\{x : f(x) \leq f(x^{(0)})\}$ and the minimum value of f is attainable, then the sequence of solutions $\{x^{(k)}\}$ generated by the coordinate descent method converges to the minimum value of f. It can be shown that the coordinate descent method satisfies the optimal condition of the optimization problem. The detailed proof is provided in the appendix A.5.

For the regularization problem (7), consider updating x_i, it can be shown that the updating formula is given by

$$x_j^{(\text{new})} = \operatorname{sign}(\hat{x}_{j,0}) \left(|\hat{x}_{j,0}| - \frac{\lambda}{A_i^\top A_i}\right)_+ . \tag{15}$$

The detailed proof is provided in the appendix A.6.

In the statistical literature, the transformation in Equation (15) is often referred to as the "soft thresholding operator" [HTW15]. According to the terminology in convex optimization, it is the "proximity operator" of the ℓ^1 norm. Also, the algorithm above is called the "ISTA" or "Iterative Soft-Thresholding Algorithm" in the statistical literature. This is an interesting connection between the two fields. The code for solving the ℓ^1 regularization problem using the coordinate descent method is provided in the appendix B.6.

4.4.2 Numerical results

The algorithm converged after 2019 iterations, taking 26.525 seconds. At this point, the cosine error is 7.151×10^{-8}, and the relative error is 4.894×10^{-3}.

As shown in the iterative process in Figure 7, the coordinate descent method converges quickly and exhibits quadratic convergence. Compared with the proximal gradient method, the coordinate descent method converges faster and reaches the quadratic convergence stage more quickly. However, the coordinate descent method runs slower than the proximal gradient method.

5 Conclusions

The lasso problem is a classical problem in convex optimization and is also a very interesting one. Its interest lies in the fact that, as a convex problem, its objective function is non-smooth,

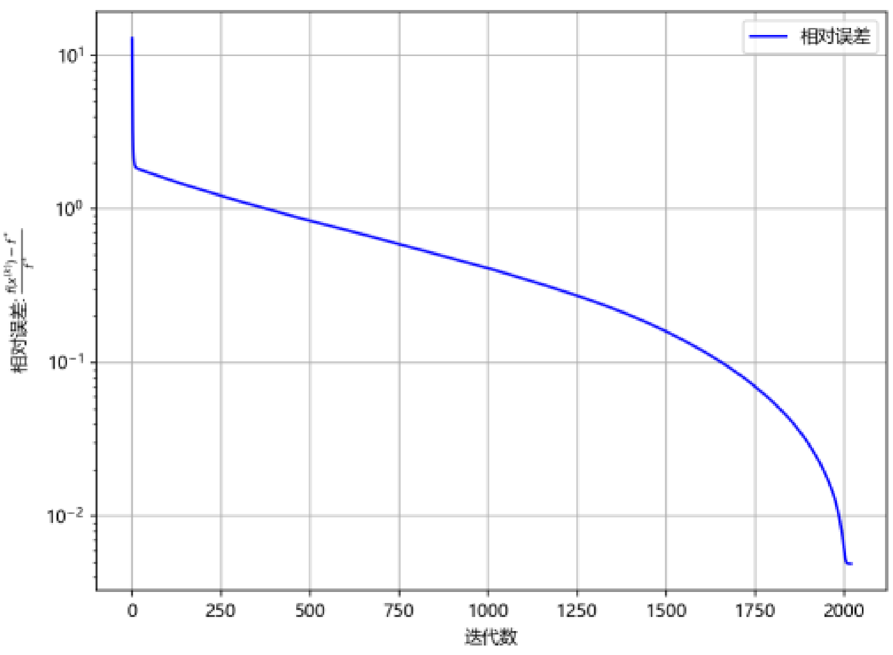

Figure 7: Iterative process of solving the ℓ^1 regularization problem using the coordinate descent method.

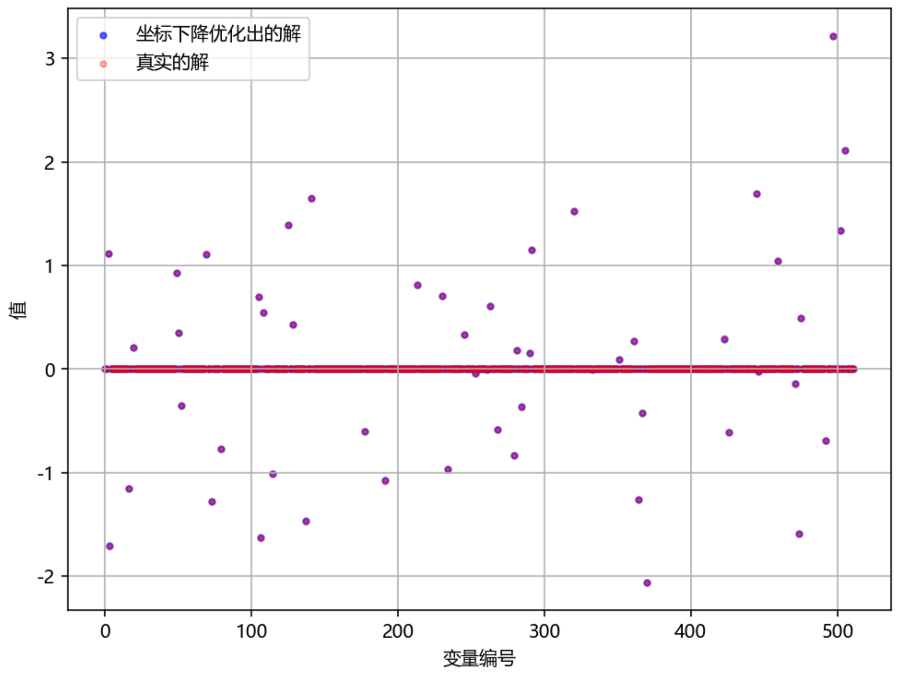

Figure 8: Comparison of the variables obtained by the coordinate descent method with the true solution.

Thus, lasso serves as a good example for connecting various optimization algorithms.

In this project, we started with the lasso problem, which is a convex optimization problem with an ℓ^1 regularization term. Then, we have derived the dual problem of the ℓ^1 problem and used the commercial solver to solve it. We have implemented three optimization algorithms for solving the lasso problem: gradient descent, proximal gradient method, and coordinate descent method. In the numerical experiments presented in this paper, the proximal gradient method and the coordinate descent method are more efficient than the gradient descent method and both of the two methods out perform the commercial solver.

References

[Ber15] Dimitri Bertsekas. *Convex Optimization Algorithms*. en. Athena Scientific, Feb. 2015. ISBN: 978-1-886529-28-1.

[BV04] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. en. Cambridge, UK ; New York: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3.

[Din24] Peng Ding. *Linear Model and Extensions*. arXiv:2401.00649. Jan. 2024. URL: http://arxiv.org/abs/2401.00649 (visited on 10/16/2024).

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. "2 - Getting to Know Your Data". In: *Data Mining (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, Jan. 2012, pp. 39–82. ISBN: 978-0-12-381479-1. DOI: 10.1016/B978-0-12-381479-1.00002-2.

[HTW15] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. New York: Chapman and Hall/CRC, May 2015. ISBN: 978-0-429-17158-1. DOI: 10.1201/b18401.

[OPT00] Michael R. Osborne, Brett Presnell, and Berwin A. Turlach. "On the LASSO and Its Dual". In: *Journal of Computational and Graphical Statistics* 9.2 (2000). Publisher: [American Statistical Association, Taylor & Francis, Ltd., Institute of Mathematical Statistics, Interface Foundation of America], pp. 319–337. ISSN: 1061-8600. DOI: 10.2307/1390657. URL: https://www.jstor.org/stable/1390657.

[Tib12] Ryan Tibshirani. "The Lasso Problem and Uniqueness". In: *Electronic Journal of Statistics* 7 (June 2012). DOI: 10.1214/13-EJS815.

[Tib96] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996). Publisher: [Royal Statistical Society, Oxford University Press], pp. 267–288. ISSN: 0035-9246. URL: https://www.jstor.org/stable/2346178.

[Tse01] P. Tseng. "Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization". en. In: *Journal of Optimization Theory and Applications* 109.3 (June 2001), pp. 475–494. ISSN: 1573-2878. DOI: 10.1023/A:1017501703105. URL: https://doi.org/10.1023/A:1017501703105.

A Proofs

A.1 Proof of Proposition 1

To prove this, we consider the Lagrangian of the constrained optimization problem:

$$L(x,\alpha) = f(x) + \alpha(h(x) - \tau)$$

The Lagrange dual function is

$$g(\alpha) = \inf_x L(x,\alpha) = \inf_x \{f(x) + \alpha(h(x) - \tau)\}.$$

Hence, the dual optimization problem is

$$\max_\alpha g(\alpha) \quad \text{s.t.} \quad \alpha \geq 0.$$

Let α^* be the solution to the dual optimization problem, satisfying the dual feasibility condition, i.e., $\alpha^* \geq 0$. Given the strong duality of the constrained optimization problem (e.g., satisfying the Slater condition), we have

$$f(x^*) = g(\alpha^*).$$

This implies the following chain of equalities:

$$\begin{aligned} f(x^*) &= g(\alpha^*) \\ &= \inf_x \{f(x) + \alpha^*(h(x) - \tau)\} \\ &\leq f(x^*) + \underbrace{\alpha^*}_{\geq 0} \underbrace{(h(x^*) - \tau)}_{\leq 0} \\ &\leq f(x^*) \end{aligned}$$

This forces all the inequalities above to be equalities. Thus,

$$\begin{aligned} \inf_x \{f(x) + \alpha^*(h(x) - \tau)\} &= f(x^*) + \alpha^*(h(x^*) - \tau) \\ \Leftrightarrow \inf_x \{f(x) + \alpha^* h(x)\} &= f(x^*) + \alpha^* h(x^*) \end{aligned}$$

This implies that x^* is the solution to the regularized problem (6) when $\lambda = \alpha^*$.

Conversely, if $x^\star$ is the solution to the regularized problem (6), then $x^\star$ is also the solution to the standard form of the optimization problem (5) with $\tau = h(x^\star)$, which satisfies the complementary slackness condition in the KKT conditions. This completes the proof of the equivalence of the two problems.

A.2 Proof of Lemma 1

The dual norm is defined as $\|y\|_* = \sup_{\|x\| \leq 1} x^\top y$. Given that

$$\sup_{\|x\|=1} x^\top y \leq \sup_{\|x\|\leq 1} x^\top y = \sup_{x\neq 0, \|x\|\leq 1} x^\top y = \sup_{x\neq 0, \|x\|\leq 1} \|x\| \left(\frac{x}{\|x\|}\right)^\top y \leq \sup_{x\neq 0, \|x\|\leq 1} \left(\frac{x}{\|x\|}\right)^\top \leq \sup_{\|x\|=1} x^\top y,$$

it follows that for $x \in \mathbb{R}^n$, we have

$$\sup_{\|x\|=1} x^\top y = \sup_{\|x\|\leq 1} x^\top y. \tag{16}$$

In order to compute

$$f^*(y) = \sup_{x \in \operatorname{dom} f} \left\{ y^\top x - \|x\| \right\}$$

we consider two cases:

Case 1. $\sup\limits_{\|x\| \leq 1} x^\top y = \|y\|_* \leq 1$. From equation (16), $\sup\limits_{\|x\|=1} x^\top y \leq 1$. Therefore $y^\top x \leq 1 = \|x\|$ for any $x \in \mathbb{R}^n$, and the equality holds when $x = 0$. Consequently, $f^*(y) = \sup\limits_{x \in \operatorname{dom} f} \left\{ y^\top x - \|x\| \right\} = 0$;

Case 2. $\sup\limits_{\|x\| \leq 1} x^\top y = \|y\|_* > 1$. In this case, there exists at least one x such that $\|x\| \leq 1$ and $x^T y > 1$. For $t > 0$, we have

$$f^*(y) \geq y^\top (tx) - \|tx\| = t \left(y^\top x - \|x\| \right),$$

The right-hand side goes to infinity as $t \to +\infty$, implying $f^*(y) = \infty$.

In conclusion, for any norm $\|x\|$, its conjugate function is

$$f^*(y) = \begin{cases} 0, & \|y\|_* \leq 1 \\ +\infty, & \|y\|_* > 1 \end{cases} \tag{17}$$

Next, we demonstrate that the dual norm of the ℓ^1 norm is the ℓ^∞ norm, i.e., the element with the maximum absolute value among the elements of a vector.

Firstly, we establish the upper bound. When $\|x\|_1 \leq 1$, we have

$$z^\top x = \sum_{i=1}^n z_i x_i \leq \sum_{i=1}^n |z_i| \, |x_i| \leq \left(\max_{i=1,\dots,n} |z_i| \right) \sum_{i=1}^n |x_i| = \|z\|_\infty \|x\|_1 \leq \|z\|_\infty.$$

Thus $\|z\|_{1*} = \sup\limits_{\|x\|_1 \leq 1} \{z^\top x\} \leq \|z\|_\infty$.

Secondly, we demonstrate the achievability of the upper bound. To show that the upper bound is tight, consider the case where $z_i = \|z\|_\infty$. By selecting $x = \operatorname{sign}(z_i) e_i$, where e_i is the i-th standard basis vector, we have$\|x\|_1 = 1$ and

$$\|z\|_{1*} \geq z^\top x = z^\top \operatorname{sign}(z_i) \, e_i = \operatorname{sign}(z_i) \, z_i = |z_i| = \|z\|_\infty.$$

Combining the results from both steps, it follows that $\|z\|_{1*} = \|z\|_\infty, z \in \mathbb{R}^n$.

A.3 Proof of Proposition 2

For the regularization problem (7), let $r = Ax - b$. The problem is equivalent to

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & \frac{1}{2}\|r\|_2^2 + \lambda \|x\|_1 \\ \text{subject to} & r = Ax - b. \end{array} \tag{18}$$

At this point, the Lagrangian is given by

$$\begin{aligned} L(\mu | x, r) &= \frac{1}{2}\|r\|_2^2 + \lambda \|x\|_1 - \mu^\top (Ax - b - r) \\ &= \frac{1}{2}\|r\|_2^2 + \mu^\top r + \lambda \|x\|_1 - (A^\top \mu)^\top x + b^\top \mu. \end{aligned}$$

Thus, the dual function is

$$\begin{aligned}
g(\mu) &= \inf_{x,r} L(\mu|x,r) \\
&= b^\top \mu + \inf_{x,r}\left\{\frac{1}{2}\|r\|_2^2 + \mu^\top r + \lambda\|x\|_1 - (A^\top\mu)^\top x\right\} \\
&= b^\top \mu + \inf_{x}\left\{\lambda\|x\|_1 - (A^\top\mu)^\top x\right\} + \inf_{r}\left\{\frac{1}{2}\|r\|_2^2 + \mu^\top r\right\} \\
&= b^\top \mu - \frac{1}{2}\|\mu\|_2^2 + \inf_{x}\left\{\lambda\|x\|_1 - (A^\top\mu)^\top x\right\} \\
&= b^\top \mu - \frac{1}{2}\|\mu\|_2^2 - \sup_{x}\left\{(A^\top\mu)^\top x - \lambda\|x\|_1\right\}.
\end{aligned}$$

Thus, based on Lemma 1, we have

$$\sup_x \left\{(A^\top\mu)^\top x - \lambda\|x\|_1\right\} = \begin{cases} 0, & \|A^\top\mu\|_\infty \le \lambda, \\ +\infty, & \|A^\top\mu\|_\infty > \lambda. \end{cases} \tag{19}$$

Therefore, the dual function of the ℓ^1 regularization problem is

$$g(\mu) = \inf_{x,r} L(\mu|x,r) = \begin{cases} b^\top\mu - \frac{1}{2}\|\mu\|^2, & \left\|A^\top\mu\right\|_\infty \le \lambda \\ -\infty, & \text{otherwise.} \end{cases} \tag{20}$$

Hence, the dual problem of the ℓ^1 regularization problem is

$$\max_{\mu\in\mathbb{R}^m} \left\{b^\top\mu - \frac{1}{2}\|\mu\|_2^2\right\} \quad \text{subject to } \|A^\top\mu\|_\infty \le \lambda. \tag{21}$$

A.4 Proof of Lemma 2

If $u = \operatorname{prox}_h(x)$, then by the optimality condition for subgradients, we have $0 \in \partial h(u) + (u - x)$, which implies $x - u \in \partial h(u)$. Conversely, if $x - u \in \partial h(u)$ then by the definition of the subgradient, we obtain

$$h(v) \ge h(u) + (x-u)^\top(v-u), \quad \forall v \in \operatorname{dom} h$$

Adding $\frac{1}{2}\|v - x\|^2$ to both sides yields

$$\begin{aligned}
h(v) + \frac{1}{2}\|v-x\|^2 &\ge h(u) + (x-u)^\top(v-u) + \frac{1}{2}\|(v-u) - (x-u)\|^2 \\
&\ge h(u) + \frac{1}{2}\|u-x\|^2, \quad \forall v \in \operatorname{dom} h
\end{aligned}$$

Thus, by the definition of the proximity operator, we conclude that $u = \operatorname{prox}_h(x)$ is equivalent to $x - u \in \partial h(u)$.

A.5 Optimality conditions in the coordinate descent method

To prove that the coordinate descent method can achieve optimality conditions, for each coordinate i, we have

$$\begin{aligned}
&0 \in \nabla_i g(x) + \partial h_i(x_i) \\
&\iff -\nabla_i g(x) \in \partial h_i(x_i) \\
&\iff h_i(y_i) \ge h_i(x_i) - \nabla_i g(x)(y_i - x_i) \quad \text{(By the definition of subgradient)} \\
&\iff \nabla_i g(x)(y_i - x_i) + h_i(y_i) - h_i(x_i) \ge 0
\end{aligned} \tag{22}$$

By the smoothness and convexity of g, we have

$$f(y)-f(x)=g(x)-g(y)+\sum_{i=1}^{n}\left[h_i\left(y_i\right)-h_i\left(x_i\right)\right] \geq \sum_{i=1}^{n}\left[\nabla_i g(x)\left(y_i-x_i\right)+h_i\left(y_i\right)-h_i\left(x_i\right)\right] \geq 0 \quad (23)$$

That completes the proof.

A.6 Updating formula in the coordinate descent method

For the regularization problem (7), consider updating x_i, then the optimization problem becomes

$$\frac{1}{2}\sum_{i=1}^{n}\left(\sum_{k\neq j} a_{ik}\hat{x}_k + a_{ij}x_j - b_i\right)^2 + \lambda|x_j| + \sum_{k\neq j}\lambda|\hat{x}_k|.$$

Denote $r_{ij} = b_i - \sum\limits_{k\neq j} a_{ik}\hat{x}_k$. Then the optimization problem becomes

$$\frac{1}{2}\sum_{i=1}^{n}\left(a_{ij}x_j - r_{ij}\right)^2 + \lambda|x_j|.$$

Denote

$$\hat{x}_{j,0} = \frac{\sum_{i=1}^{n} a_{ij}r_{ij}}{\sum_{i=1}^{n} a_{ij}^2} = \frac{A_j^T r}{A_j^T A_j},$$

where A_j is the$j$-th column of matrix A, A_{-j} represents matrix A without the j-th column. r denotes the vector $b - A_{-j}\hat{x}_{-j}$. Then

$$\sum_{i=1}^{n} a_{ij}(x_j - \hat{x}_{j,0})(a_{ij}\hat{x}_{j,0} - r_{ij}) = (x_j - \hat{x}_{j,0})\sum_{i=1}^{n}\left\{a_{ij}\left(a_{ij}\frac{\sum_{i=1}^{n} a_{ij}r_{ij}}{\sum_{i=1}^{n} a_{ij}^2} - r_{ij}\right)\right\} = 0.$$

By adding and substracing, we have

$$\begin{aligned}
\sum_{i=1}^{n}(a_{ij}x_j - r_{ij})^2 &= \sum_{i=1}^{n}(a_{ij}(x_j - \hat{x}_{j,0}) + a_{ij}\hat{x}_{j,0} - r_{ij})^2 \\
&= \sum_{i=1}^{n}(a_{ij}(x_j - \hat{x}_{j,0}))^2 + 2\sum_{i=1}^{n} a_{ij}(x_j - \hat{x}_{j,0})(a_{ij}\hat{x}_{j,0} - r_{ij}) + \sum_{i=1}^{n}(a_{ij}\hat{x}_{j,0} - r_{ij})^2 \\
&= (x_j - \hat{x}_{j,0})^2\left(\sum_{i=1}^{n} a_{ij}^2\right) + \sum_{i=1}^{n}(a_{ij}\hat{x}_{j,0} - r_{ij})^2.
\end{aligned}$$

By using the results in proximal gradient, we have

$$\arg\min_{b\in\mathbb{R}} \frac{1}{2}\left(b - b_0\right)^2 + \lambda|b| = \operatorname{sign}\left(b_0\right)\left(|b_0| - \lambda\right)_+.$$

This gives (15) and completes the proof.

B Code Implementations

B.1 Python code for DGP

Under this data generating process, one can compute the optimal value of the optimization problem, denoted as f^*. For example, if we set the random seed to 42, we can obtain $f^* = 4.513$.

```python
import numpy as np

# 设置随机种子以确保结果可复现
np.random.seed(42)

# 定义参数
n = 512   # 特征数量
m = 256   # 样本数量

# 生成随机矩阵 A
A = np.random.randn(m, n)

# 稀疏性: 生成一个随机稀疏向量 u, 假设 u 中有 10% 的非零元素
k = int(0.1 * n)  # 非零元素的数量
u_indices = np.random.choice(n, k, replace=False)  # 随机选择非零元素的索引
u_values = np.random.randn(k)  # 随机生成非零元素的值
u = np.zeros(n)
u[u_indices] = u_values  # 将随机值赋给 u 的非零元素位置

# 计算 b = Au
b = np.dot(A, u)
```

B.2 Python code for solving the ℓ^1 regularization problem using Gurobi

```python
def l1_cvx_gurobi(x0, A, b, mu):
    m, n = A.shape
    # 创建一个 Gurobi 模型
    model = gp.Model("l1_regularization")
    # 创建变量
    x = model.addVars(n, lb=-GRB.INFINITY, name="x")  # x 的下界设为负无穷
    # 设置目标函数
    # 目标是最小化 (1/2) * ||Ax - b||_2^2 + mu * ||x||_1
    # 我们需要引入一个额外的变量来处理 ||x||_1
    z = model.addVars(n, name="z")  # z 用于表示 |x|
    # 添加目标函数
    model.setObjective(
        (1/2) * gp.quicksum((gp.quicksum(A[i, j] * x[j] for j in range(n)) - b[i]) ** 2 for i in range(m)) +
        mu * gp.quicksum(z[j] for j in range(n)),
        GRB.MINIMIZE
    )
    # 添加约束: z[j] >= x[j] 和 z[j] >= -x[j]
    for j in range(n):
        model.addConstr(z[j] >= x[j])
        model.addConstr(z[j] >= -x[j])

    # 求解模型
    model.optimize()
    # 输出结果
    if model.status == GRB.OPTIMAL:
        value_float = model.ObjVal
        iteration_num = model.BarIterCount ## 由于使用了内点法, 用这个属性才能记录迭代次数
        solution = model.getAttr('x', x)
        # 提取 tupledict 中的数据
        data = [(value, key) for key, value in solution.items()]
        # 将数据转换为列表
        data_list = [item[0] for item in data]
        # 将列表转换为 NumPy 数组
        solution_ndarray = np.array(data_list)
        print(" 找出最优解.")
        return(solution_ndarray, value_float, iteration_num)
    else:
        print(" 没有找出最优解.")
```

Here, when the regularization parameter $\lambda = 0.1$, we can solve the problem (12) using the following code. The solver does not use an initial point x_0. The running results are given in the comments of the code below. The results of the computation are provided in the comments aside the code.

```python
mu = 0.1  # 正则化参数
x0 = np.random.randn(n)
# 测试

import time

start_time = time.time()
solution, value, iter_num = l1_cvx_gurobi(x0, A, b, mu)
end_time = time.time()

## 最优值比较
np.abs((value - oracle_value) / oracle_value)
# 0.00026211821497879446

## 解的比较
def cosine_similarity(vector_a, vector_b):
    dot_product = np.dot(vector_a, vector_b)

    norm_a = np.linalg.norm(vector_a)
    norm_b = np.linalg.norm(vector_b)

    cosine_sim = dot_product / (norm_a * norm_b)
    return cosine_sim

1 - cosine_similarity(solution, u)
# 7.476662444716453e-08

## 求解时间
print(f" 求解时间: {end_time - start_time:.4f}秒")
# 求解时间: 33.8279 秒

## 迭代次数
iter_num
# 13
```

B.3 Python code for solving the ℓ^1 regularization dual problem using `Gurobi`

```python
def l1_dual_gurobi(A, b, lambda_val):
    m, n = A.shape
    # 创建一个 Gurobi 模型
    model = gp.Model("l1_regularization_dual")

    # 创建对偶变量 mu
    mu = model.addVars(m, name="mu")  # 对偶变量

    # 设置目标函数: b^T * mu - 1/2 * ||mu||_2^2
    model.setObjective(
        gp.quicksum(b[i] * mu[i] for i in range(m)) - 0.5 * gp.quicksum(mu[i] * mu[i] for i in range(m)),
        GRB.MAXIMIZE
    )
    # 添加约束: ||A^T * mu||_∞ <= lambda
    # 对每个特征的线性组合, 需要限制 |A^T * mu| <= lambda
    A_T = A.T  # A^T 转置
    for j in range(n):
        model.addConstr(
            gp.quicksum(A_T[j, i] * mu[i] for i in range(m)) <= lambda_val,  # 上界
            name=f"upper_bound_{j}"
        )
        model.addConstr(
            gp.quicksum(A_T[j, i] * mu[i] for i in range(m)) >= -lambda_val,  # 下界
            name=f"lower_bound_{j}"
        )

    # 求解模型
    model.optimize()
    # 输出结果
    if model.status == GRB.OPTIMAL:
        iteration_num = model.BarIterCount ## 由于使用了内点法, 用这个属性才能记录迭代次数
        optimal_mu = np.array([mu[i].x for i in range(m)])  # 获取最优解
        optimal_value = model.ObjVal  # 获取最优目标值
        print("找到最优解:")
        print(f"最优值: {optimal_value}")
        return optimal_mu, optimal_value, iteration_num
    else:
        print("没有找到最优解.")
        return None, None
```

B.4 Implementation of the gradient descent method

Based on the Huber smoothing defined above, we can implement the gradient descent method to solve the ℓ^1 regularization problem (7). When implementing the piecewise function, there are two methods: using an 'if-else' statement or multiplying by an indicator function. The later is a clever trick.

```
# 定义 Huber 函数
def huber(x, delta):
    if np.abs(x) < delta:
        return 0.5 * x**2 / delta
    else:
        return np.abs(x) - 0.5 * delta

# 计算 L_delta(x) 的梯度
def grad_L_delta(x, delta):
    return np.sign(x) * (np.abs(x) > delta) + x / delta * (np.abs(x) <= delta)

# 目标函数的梯度
def grad_f_delta(x, A, b, mu, delta):
    grad_f = A.T @ (A @ x - b)
    grad_L = mu * grad_L_delta(x, delta)
    return grad_f + grad_L

# 计算目标函数值
def f_delta(x, A, b, mu, delta):
    residual = np.linalg.norm(A @ x - b) ** 2 / 2
    L_delta = np.sum([huber(xi, delta) for xi in x])
    return residual + mu * L_delta
```

Next, we implement backtracking line search (Amijo rule) to determine the step size. Here the parameter 'c' in the 'check descent condition' is the constant α in Algorithm 9.2 of the textbook [BV04]. When selecting the parameter α, it should be chosen from the interval $\alpha \in (0, 0.5)$. Here, I have chosen $\alpha = 0.25$.

```python
# 回溯线搜索
def backtracking_line_search(x, grad, A, b, mu, delta, step_size=1.0, beta=0.5, c=0.25):
    alpha = step_size
    f_current = f_delta(x, A, b, mu, delta)
    while True:
        x_new = x - alpha * grad
        f_new = f_delta(x_new, A, b, mu, delta)
        # 检查下降条件
        if f_new <= f_current - c * alpha * np.linalg.norm(grad) ** 2:
            break
        alpha *= beta  # 步长衰减
    return alpha
```

Here is the main program of the gradient descent algorithm:

```python
# 梯度下降法（使用回溯线搜索）
def gradient_descent(A, b, mu, delta, x0, oracle_value, max_iter=5000, tol_f=1e-8, tol_grad=1e-6, step_size=1.0):
    x = x0
    f_prev = f_delta(x, A, b, mu, delta)

    relative_errors = []  # 用于记录相对误差
    start_time = time.time()  # 计时开始

    for k in range(1, max_iter + 1):
        grad = grad_f_delta(x, A, b, mu, delta)

        # 使用回溯线搜索来确定步长
        alpha = backtracking_line_search(x, grad, A, b, mu, delta, step_size)

        # 更新 x
        x_new = x - alpha * grad

        # 计算当前目标函数值
        f_curr = f_delta(x_new, A, b, mu, delta)

        # 计算相对误差
        relative_error = np.abs(f_curr - oracle_value) / oracle_value
        relative_errors.append(relative_error)

        # 判断是否满足终止条件
        if np.abs(f_curr - f_prev) < tol_f:
            print(f"Converged by function value change after {k} iterations.")
            break
        if np.linalg.norm(grad) < tol_grad:
            print(f"Converged by gradient norm after {k} iterations.")
            break

        # 更新变量
        x = x_new
        f_prev = f_curr

    # 计时结束
    elapsed_time = time.time() - start_time
    print(f"Total time taken: {elapsed_time:.4f} seconds")

    return x, relative_errors, elapsed_time
```

This function can store the relative errors and computation times produced at each iterations. The core consissts of two parts: computing the gradient and backtracking line search.

B.5 Implementation of the proximal gradient method

When writing the function, backtracking line search is used to determine the step size in order to ensure convergence. Additionally, since the proximal gradient method converges relatively slowly, the maximum number of iterations is set to 50000. to ensure taht the algorithm converges within a finite amount of time.

```
# 目标函数计算
def objective(x, A, b, lambda_):
    return 0.5 * np.linalg.norm(A @ x - b) ** 2 + lambda_ * np.linalg.norm(x, 1)

# L1 范数收缩操作，执行 x 的软阈值操作
def prox_l1(x, lambda_, t_k):
    return np.sign(x) * np.maximum(np.abs(x) - lambda_ * t_k, 0)

def backtracking_line_search(x, grad, A, b, lambda_, step_size=1.0, beta=0.5, c=1e-5):
    alpha = step_size
    f_current = objective(x, A, b, lambda_)
    while True:
        x_new = x - alpha * grad
        f_new = objective(x_new, A, b, lambda_)
        # 检查下降条件
        if f_new <= f_current - c * alpha * np.linalg.norm(grad) ** 2:
            break
        alpha *= beta  # 步长衰减
    return alpha

# 近似点梯度法
def approx_point_gradient(A, b, lambda_, x0, oracle_value, max_iter=100000, tol=1e-6):
    x = x0
    relative_errors = []
    f_prev = objective(x, A, b, lambda_)
    # 记录开始时间
    start_time = time.time()

    for k in range(max_iter):
        # 计算梯度下降步
        grad = A.T @ (A @ x - b)
        alpha = backtracking_line_search(x, grad, A, b, lambda_)
        y_k = x - alpha * grad

        # L1 收缩操作
        x_new = prox_l1(y_k, lambda_, t_k)
        # 计算目标函数值并记录相对误差
        f_x_k = objective(x_new, A, b, lambda_)
        relative_error = np.abs(f_x_k - oracle_value) / oracle_value
        relative_errors.append(relative_error)

        # 检查收敛性
        if np.abs(f_x_k - f_prev) < tol:
            print(f"Converged by function value change after {k} iterations.")
            break
        if np.linalg.norm(x_new - x, ord=2) < tol:
            print(f'Converged at iteration {k}')
```

```python
                break
        if relative_error < tol:
            print(" 退出")
            break
        f_prev = f_x_k
        x = x_new
    elapsed_time = time.time() - start_time
    return x, relative_errors, elapsed_time
```

B.6 Implementation of the coordinate descent method

The following code implements the coordinate descent method.

```python
# 软阈值算子
def soft_thresholding(x, lambda_):
    return np.sign(x) * np.maximum(np.abs(x) - lambda_, 0)

# 坐标下降法

def coordinate_descent(A, b, lambda_, x0, max_iter=3000, tol=1e-6):
    x = x0
    m, n = A.shape
    residuals = []  # 用于记录残差
    relative_errors = [0.5 * np.linalg.norm(A @ x - b) + lambda_ * np.sum(np.abs(x))]
    start_time = time.time()  # 计时开始
    for it in range(max_iter):
        x_old = x.copy()  # 保存当前解

        # 对每个坐标更新
        for i in range(n):
            # 计算 xi_new
            A_i = A[:, i]
            r_i = b - A @ x + A_i * x[i]  # 计算残差（去除当前坐标的贡献）
            x[i] = soft_thresholding(A_i.T @ r_i / np.linalg.norm(A_i)**2, lambda_ / np.linalg.norm(A_i)**2)

        # 计算当前残差和目标函数值
        residual = np.linalg.norm(A @ x - b)
        relative_error = 0.5 * residual + lambda_ * np.sum(np.abs(x))
        residuals.append(residual)
        relative_errors.append(relative_error)
        # 检查终止条件（目标函数变化或残差小于容忍值）
        if np.linalg.norm(x - x_old) < tol:
            print(f" 在第 {it+1} 次迭代收敛")
            break

    # 计时结束
    elapsed_time = time.time() - start_time
    print(f" 坐标下降时间: {elapsed_time:.4f} 秒")

    return x, residuals, elapsed_time, relative_errors
```