Group LASSO Problem

Zexi Fan 2200010816. Lab group: Optimization Methods. Tutor: Zaiwen Wen.

Lab date: 17th December 2024. Report date: 17th December 2024.

Abstract

In this lab report, we solve the group LASSO problem with different solvers, including CVX, Gurobi, Mosek, SGD Primal, ProxGD Primal, FProxGD Primal, ALM Dual, ADMM Dual, and ADMM Primal. Their accuracy, speed, and sparsity are thoroughly investigated and compared. We implemented all these code in Matlab.

1 Problem Formulation

Consider the group LASSO problem[1],

$$\min_{x\in\mathbb{R}^{n\times l}} \quad \frac{1}{2}\|Ax-b\|_F^2+\mu\|x\|_{1,2}, \tag{1}$$

where,

$$A\in\mathbb{R}^{m\times n} \tag{2}$$

$$b\in\mathbb{R}^{m\times l} \tag{3}$$

$$\mu>0 \tag{4}$$

and that,

$$\|x\|_{1,2}=\sum_{i=1}^{n}\|x(i,1:l)\|_2. \tag{5}$$

Here $x(i,1:l), 1\le i\le n$ is the i-th row of matrix x.

2 Experimental Apparatus and Methods

The project has implemented multiple algorithms, and to compare their performance under different conditions, the following metrics are adopted to evaluate each algorithm:

- **Objective Value** (`optival`): Let f_x represent the objective value produced by the algorithm, we print them out and plot them for each solver to compare their minimization efficiency.

- **Error to Exact Solution**(`err-to-exact`): The normalized Frobenius norm difference between the solution obtained by the algorithm and the optimal solution:

  $$\text{Error}(x, u) = \frac{\|x - u\|_F}{1 + \|u\|_F}$$

  In this formula, x denotes the solution obtained by the algorithm, and u is the optimal solution.

- **Error with Respect to CVX-Mosek Solution** (`err-to-cvx-mosek`): The normalized Frobenius norm difference between the solution obtained by the algorithm and the solution computed by CVX-Mosek:

  $$\text{Error}(x, x_{cvx}) = \frac{\|x - x_{cvx}\|_F}{1 + \|x_{cvx}\|_F}$$

  Here, x represents the solution obtained by the algorithm, and x_{cvx} is the solution computed using Mosek via the CVX toolbox.

- **Error with Respect to CVX-Gurobi Solution** (`err-to-cvx-gurobi`): The normalized Frobenius norm difference between the solution obtained by the algorithm and the solution computed by CVX-Gurobi:

  $$\text{Error}(x, x_{cvx}) = \frac{\|x - x_{cvx}\|_F}{1 + \|x_{cvx}\|_F}$$

  Similarly, x represents the solution obtained by the algorithm, and x_{cvx} denotes the solution computed using Gurobi via the CVX toolbox.

- **Sparsity** (`sparsity`): The sparsity of the solution obtained by the algorithm, defined as:

  $$\text{Sparsity} = \frac{\|x\|_0}{n \times l}$$

  Here, $\|x\|_0$ represents the number of nonzero elements in x, with elements whose absolute values are smaller than 10^{-5} treated as zero.

- **Runtime** (`cpu`): The time taken by the algorithm to complete the computation on intel 12500H.

- **Number of Iterations** (`Iter`): The total number of iterations performed by the algorithm.

The solving techniques of each solver are listed as the following.

2.1 CVX-Mosek and CVX-Gurobi

We do not need to reformulate CVX in Matlab.

2.2 Mosek

The group LASSO problem is formulated in MOSEK as follows:

Optimization Problem: Minimize:

$$\frac{1}{2}t + \mu \sum_{i=1}^{n} z_i$$

subject to:

$$\begin{aligned} \text{(1) Linear constraint:} \quad & (I_l \otimes A)\,\mathrm{vec}(x) - \mathrm{vec}(y) = \mathrm{vec}(b), && (6) \\ \text{(2) Quadratic cone constraint 1:} \quad & [1+t; 2 \cdot \mathrm{vec}(y); 1-t] \in \mathcal{Q}, && (7) \\ \text{(3) Quadratic cone constraint 2:} \quad & [z_i; \mathrm{vec}(x_i)] \in \mathcal{Q}, \quad \forall i \in \{1, \dots, n\}, && (8) \\ \text{(4) Bounds on variables:} \quad & z_i \geq 0, \quad \forall i \in \{1, \dots, n\}. && (9) \end{aligned}$$

Variables: - $x \in \mathbb{R}^{n \times l}$: Solution matrix to be optimized, flattened into $\mathrm{vec}(x)$. - $y \in \mathbb{R}^{m \times l}$: Auxiliary variable vector. - $t \in \mathbb{R}$: Scalar variable controlling the quadratic cone. - $z \in \mathbb{R}^n$: Regularization variables corresponding to the group sparsity terms.

Parameters: - $A \in \mathbb{R}^{m \times n}$: Constraint matrix. - $b \in \mathbb{R}^{m \times l}$: Observation matrix, reshaped as $\mathrm{vec}(b)$. - $\mu > 0$: Regularization parameter. - $\mathcal{Q}$: The standard quadratic cone defined as:

$$\mathcal{Q} = \left\{ v = [v_0; v_1; \dots; v_k] \in \mathbb{R}^{k+1} : v_0 \geq \|[v_1; \dots; v_k]\|_2 \right\}.$$

Objective Function: The objective consists of a trade-off between a quadratic term involving t and a regularization term involving the sum of z_i's.

Constraints: 1. The linear constraint ensures the relationship between x, y, and b. 2. The first quadratic cone constraint imposes the condition:

$$\|\mathrm{vec}(y)\|_2 \leq t \quad \text{and} \quad t \geq 0.$$

3. The second set of quadratic cone constraints enforces group sparsity by ensuring that for each group i, the norm of the corresponding row $\mathrm{vec}(x_i)$ is bounded by z_i. 4. The bounds $z_i \geq 0$ ensure non-negativity of the regularization variables.

2.3 Gurobi

The group LASSO problem is formulated in Gurobi as follows:

Optimization Problem: Minimize:

$$\frac{1}{2} \sum_{j=1}^{l} \|y_j\|_2^2 + \mu \sum_{i=1}^{n} z_i$$

subject to:

$$\begin{aligned} \text{(1) Linear constraint:} \quad & (I_l \otimes A)\,\mathrm{vec}(x) - \mathrm{vec}(y) = \mathrm{vec}(b), && (10) \\ \text{(2) Quadratic cone constraints:} \quad & [z_i; x_{i1}; x_{i2}; \dots; x_{il}] \in \mathcal{Q}, \quad \forall i \in \{1, \dots, n\}, && (11) \\ \text{(3) Bounds on variables:} \quad & z_i \geq 0, \quad \forall i \in \{1, \dots, n\}. && (12) \end{aligned}$$

Variables: - $x \in \mathbb{R}^{n \times l}$: Solution matrix with x_{ij} as its elements. - $y \in \mathbb{R}^{m \times l}$: Auxiliary variable vector with y_{ij} representing components in group j. - $z \in \mathbb{R}^n$: Regularization variables corresponding to the group sparsity terms.

Parameters: - $A \in \mathbb{R}^{m \times n}$: Constraint matrix. - $b \in \mathbb{R}^{m \times l}$: Observation matrix, reshaped as vec(b). - $\mu > 0$: Regularization parameter. - $\mathcal{Q}$: The standard quadratic cone defined as:

$$\mathcal{Q} = \left\{ v = [v_0; v_1; \dots; v_k] \in \mathbb{R}^{k+1} : v_0 \geq \|[v_1; \dots; v_k]\|_2 \right\}.$$

Objective Function: The objective function consists of two parts: 1. A quadratic term:

$$\frac{1}{2} \sum_{j=1}^{l} \|y_j\|_2^2,$$

which penalizes the auxiliary variable y to enforce sparsity. 2. A linear regularization term:

$$\mu \sum_{i=1}^{n} z_i,$$

which encourages group-wise sparsity through z.

Constraints: 1. The linear constraint ensures the coupling between x, y, and the observed b:

$$(I_l \otimes A)\, \mathrm{vec}(x) - \mathrm{vec}(y) = \mathrm{vec}(b),$$

where $\otimes$ denotes the Kronecker product. 2. The quadratic cone constraints enforce group sparsity:

$$\|[x_{i1}, x_{i2}, \dots, x_{il}]\|_2 \leq z_i, \quad \forall i = 1, \dots, n.$$

3. The bounds ensure $z_i \geq 0$ for all i.

Structure in Gurobi: - The quadratic part of the objective is represented using a matrix Q, where only the y-related terms have a nonzero diagonal 0.5. - The quadratic cone constraints are directly added using Gurobi's `quadcon` structure, specifying the cone constraints for each group i.

2.4 SGD

The Subgradient Descent (SGD) algorithm is a generalization of the classical gradient descent method. Unlike standard gradient methods, the SGD algorithm leverages subgradients instead of gradients, allowing it to handle objective functions containing non-smooth components. For the current problem, where the non-smooth term arises in the second part of the objective function, the SGD approach efficiently addresses this challenge. The pseudocode of the algorithm is shown in Algorithm 1.

The critical component of implementing the SGD algorithm for this problem is the computation of the subgradient of the $\ell_{1,2}$-norm, denoted as $\partial_{\ell_{1,2}}(x^{(k)})$. The $\ell_{1,2}$-norm is defined as:

$$\|x\|_{1,2} = \sum_{i=1}^{n} \|x_{(i,:)}\|_2,$$

Algorithm 1 SGD Solver

```
Input: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m\times l}$, $\mu > 0$, maximum iterations max_iter, tolerance tol
Initialize $x^{(0)}$ (random initialization)
Set Lipschitz constant $L$ (e.g., $L = \|A^T A\|_2$)
for $k = 0, 1, 2, \ldots,$ max_iter do
    Compute the subgradient $g^{(k)}$:
        $$g^{(k)} = A^T(Ax^{(k)} - b) + \mu \cdot \partial_{\ell_{1,2}}(x^{(k)})$$
    Update $x^{(k+1)}$ using the proximal operator:
        $$x^{(k+1)} = \text{prox}_{\frac{1}{L}\mu\|\cdot\|_{1,2}}\left(x^{(k)} - \frac{1}{L}g^{(k)}\right)$$
    Check convergence: if $\|x^{(k+1)} - x^{(k)}\|_F <$ tol, terminate
end for
Output: $x^{(k+1)}$
```

where $x_{(i,:)}$ represents the i-th row of the matrix x. The subgradient of the $\ell_{1,2}$-norm at $x^{(k)}$ can therefore be decomposed into a summation over row-wise ℓ_2-norm subgradients:

$$\partial_{\ell_{1,2}}(x^{(k)}) = \sum_{i=1}^{n} \partial_{\ell_2}(x^{(k)}_{(i,:)}),$$

where $\partial_{\ell_2}(x^{(k)}_{(i,:)})$ is the subgradient of the ℓ_2-norm and is defined as:

$$\partial_{\ell_2}(x^{(k)}_{(i,:)}) = \begin{cases} \frac{x^{(k)}_{(i,:)}}{\|x^{(k)}_{(i,:)}\|_2} & \text{if } x^{(k)}_{(i,:)} \neq 0, \\ \text{any } v \in \mathbb{R}^l \text{ with } \|v\|_2 \leq 1 & \text{if } x^{(k)}_{(i,:)} = 0. \end{cases}$$

The proximal operator used in the update step involves the $\ell_{1,2}$-norm regularization. This step ensures the sparsity-inducing effect in group-wise variables, which is crucial for problems involving structured sparsity. The choice of the Lipschitz constant L, which bounds the gradient's smoothness, further stabilizes the updates.

In practice, the subgradient descent method may converge slower compared to gradient-based methods for smooth objectives. However, it offers significant flexibility for non-smooth regularizers like the $\ell_{1,2}$-norm, which arise in structured sparsity problems such as group LASSO. By iteratively applying the subgradient and proximal operator, the algorithm can effectively balance data fidelity and regularization, achieving structured sparsity in the solution.

2.5 ProxGD

The Proximal Gradient Descent (ProxGD) algorithm combines the gradient descent method with a proximal operator to handle objective functions with non-smooth components. This approach decomposes the optimization process into two steps: a gradient descent step for the smooth part and a proximal operator step for the non-smooth part. The pseudocode of the ProxGD algorithm is presented in Algorithm 2.

Algorithm 2 ProxGD Solver

```
Input: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^{m\times l}$, $\mu > 0$, maximum iterations max_iter, tolerance tol
Initialize $x^{(0)}$ (random initialization)
for $k = 0, 1, 2, \dots,$ max_iter do
    Compute gradient $g^{(k)}$:
        $$g^{(k)} = A^T(Ax^{(k)} - b)$$
    Compute proximal operator:
        $$x_{\text{prox}} = \text{prox}_{\frac{\mu}{L}\|\cdot\|_{1,2}}\left(x^{(k)} - \frac{1}{L}g^{(k)}\right)$$
    Update $x^{(k+1)}$ using proximal operator:
        $$x^{(k+1)} = x_{\text{prox}}$$
    Check convergence: if $\|x^{(k+1)} - x^{(k)}\|_F <$ tol, terminate
end for
Output: $x^{(k+1)}$
```

In the ProxGD algorithm, the objective function is decomposed into two components: a smooth term $\phi(x) = \frac{1}{2}\|Ax - b\|_F^2$ and a non-smooth term $h(x) = \mu\|x\|_{1,2}$. The algorithm applies gradient descent to the smooth term while using the proximal operator for the non-smooth term. The iterative update rule is given by:

$$x^{(k+1)} = \text{prox}_{t_k h(\cdot)}\left(x^{(k)} - t_k \nabla\phi(x^{(k)})\right),$$

where t_k is the step size.

For the specific problem at hand, the proximal operator can be expressed analytically as:

$$\text{prox}_{t_k h(\cdot)}\left(x_k - t_k\nabla\phi(x_k)\right) = \begin{cases} \frac{x_k - t_k\nabla\phi(x_k)}{1+t_k\mu}, & \text{if } x_k - t_k\nabla\phi(x_k) \neq 0, \\ \text{any } v \in \mathbb{R}^l \text{ with } \|v\|_2 \leq 1, & \text{if } x_k - t_k\nabla\phi(x_k) = 0. \end{cases} \tag{13}$$

The proximal operator step efficiently enforces the sparsity-inducing effect of the $\ell_{1,2}$-norm regularization, which is crucial in applications involving structured sparsity, such as group-wise variable selection. The smoothness of $\phi(x)$ ensures that the gradient descent step converges, while the proximal operator guarantees that the non-smooth term is properly handled.

Compared to standard gradient descent, ProxGD achieves a balance between minimizing the smooth part of the objective and enforcing regularization through its proximal updates. This makes it particularly suitable for problems where sparsity and structure in the solution are desired.

2.6 FProxGD

Nesterov acceleration is a widely used acceleration method, where the result from the previous iteration is leveraged to speed up the current iteration. Its iteration scheme is as follows:

$$x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k),$$

$$y_{k+1} = x_{k+1} + \frac{k}{k+3}(x_{k+1} - x_k).$$

In this project, the Proximal Gradient Descent (ProxGD) algorithm can incorporate Nesterov acceleration, resulting in the Fast Proximal Gradient Descent (FProxGD) algorithm. However, in practice, the convergence speed of the FProxGD algorithm does not outperform the ProxGD algorithm. This phenomenon may be attributed to the use of the continuation strategy and BB step size update strategy, which already enhance the convergence speed of ProxGD significantly. Consequently, introducing Nesterov acceleration could potentially increase the number of iterations, thereby slowing down the overall convergence speed.

2.7 ALM and ADMM Algorithms

The Augmented Lagrangian Method (ALM) is a popular approach for solving constrained optimization problems. It transforms the original constrained problem into an unconstrained form by introducing Lagrange multipliers and an additional penalty term, which allows the original problem to be approximated effectively.

The Alternating Direction Method of Multipliers (ADMM), on the other hand, is a variant of ALM that splits the original problem into two subproblems that can be solved iteratively and alternately. This decomposition allows ADMM to efficiently handle large-scale or distributed optimization problems.

For the given primal problem in this project, the dual problem can be formulated as:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2}\|z\|_F^2 + \langle b, z\rangle \\ \text{s.t.} \quad & \left\|A^T z\right\|_{\infty,2} \le \mu, \end{aligned} \tag{14}$$

where $\|\cdot\|_{\infty,2}$ is the mixed $\ell_{\infty,2}$-norm constraint.

This problem is equivalent to introducing an auxiliary variable s as follows:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2}\|z\|_F^2 + \langle b, z\rangle \\ \text{s.t.} \quad & \|s\|_{\infty,2} \le \mu, \quad s = A^T z. \end{aligned} \tag{15}$$

The augmented Lagrangian function for the above constrained optimization problem can then be expressed as:

$$\mathcal{L}_\rho(z, s, \lambda) = \frac{1}{2}\|z\|_F^2 + \langle b, z\rangle - \langle \lambda, s - A^T z\rangle + \frac{\rho}{2}\|s - A^T z\|_F^2, \quad \text{s.t. } \|s\|_{\infty,2} \le \mu, \tag{16}$$

where λ is the Lagrange multiplier and $\rho > 0$ is a penalty parameter that controls the weight of the quadratic penalty term.

Based on the augmented Lagrangian formulation (16), the ALM and ADMM iterative processes can be derived as follows:

1. ALM Iteration: The ALM framework alternates between minimizing the augmented Lagrangian function $\mathcal{L}_\rho(z, s, \lambda)$ with respect to primal variables z and s, and updating the dual variable λ. The iterations proceed as:

$$\begin{aligned} z^{(k+1)} &= \arg\min_{z} \mathcal{L}_\rho(z, s^{(k)}, \lambda^{(k)}), \\ s^{(k+1)} &= \arg\min_{s:\|s\|_{\infty,2}\le\mu} \mathcal{L}_\rho(z^{(k+1)}, s, \lambda^{(k)}), \\ \lambda^{(k+1)} &= \lambda^{(k)} - \rho(s^{(k+1)} - A^T z^{(k+1)}). \end{aligned}$$

2. ADMM Iteration: In the ADMM framework, the primal variables z and s are updated sequentially, followed by the dual variable update. The steps are:

$$\begin{aligned} z^{(k+1)} &= \arg\min_{z} \left(\frac{1}{2}\|z\|_F^2 + \langle b, z\rangle + \frac{\rho}{2}\|s^{(k)} - A^T z - \lambda^{(k)}/\rho\|_F^2 \right), \\ s^{(k+1)} &= \arg\min_{s:\|s\|_{\infty,2}\le\mu} \frac{\rho}{2}\|s - A^T z^{(k+1)} - \lambda^{(k)}/\rho\|_F^2, \\ \lambda^{(k+1)} &= \lambda^{(k)} - \rho(s^{(k+1)} - A^T z^{(k+1)}). \end{aligned}$$

The ALM and ADMM algorithms both leverage the augmented Lagrangian formulation to iteratively enforce the constraints while minimizing the objective. However, ADMM's alternating minimization strategy simplifies the problem by solving subproblems in sequence, which often allows for closed-form solutions or easier numerical implementations.

In this problem, the constraint $\|s\|_{\infty,2} \le \mu$ introduces sparsity in the solution, which is particularly useful in applications requiring structured sparsity. The parameter ρ balances the accuracy of constraint enforcement and the optimization convergence. Proper tuning of ρ and initialization of dual variables λ are critical to achieving faster convergence.

Overall, the ADMM algorithm offers a more computationally efficient alternative for large-scale problems due to its decomposable nature, making it well-suited for practical applications in constrained optimization.

2.8 Auxiliary

2.8.1 Continuation Strategy

For the LASSO problem in this project, given as:

$$\min_{x\in\mathbb{R}^{n\times l}} \frac{1}{2}\|Ax - b\|_F^2 + \mu\|x\|_{1,2}, \tag{17}$$

the continuation strategy[2] starts from a large regularization parameter μ_t and gradually decreases it to μ_0 such that:

$$\mu_1 \ge \mu_2 \ge \cdots \ge \mu_{t-1} \ge \mu_t \ge \cdots \ge \mu_0.$$

At each step, a new LASSO problem is solved with the current μ_t:

$$\min_{x\in\mathbb{R}^{n\times l}} \frac{1}{2}\|Ax - b\|_F^2 + \mu_t\|x\|_{1,2}. \tag{18}$$

The main benefit of this strategy is that the solution of the previous LASSO problem with μ_{t-1} serves as a good approximation to the solution of the current problem with μ_t. This significantly reduces the computational time, as warm-starting the solver accelerates convergence. Larger μ_t values correspond to easier LASSO problems, so the continuation strategy effectively solves a sequence of simpler problems to approximate the original problem.

The regularization parameter μ_{t+1} is updated according to:

$$\mu_{t+1} = \max\{\mu_0, \mu_t \eta\},$$

where $\eta \in (0, 1)$ is a scaling factor.

In this project: - $\mu_0 = 0.01$, - The initial value $\mu_1 = 100$, - The scaling factor $\eta = 0.1$.

The continuation strategy is applied to the SGD, ProxGD, and FProxGD algorithms, where the solvers are wrapped in an outer loop that iteratively solves the LASSO problem for each decreasing μ_t.

2.8.2 BB Step Size Update Strategy

The Barzilai-Borwein (BB) step size update strategy is employed to improve convergence speed. It adjusts the step size α dynamically based on the gradient and iterate differences. The step size update can be expressed in two forms:

$$x^{k+1} = x^k - \alpha_{\mathrm{BB1}}^k \nabla f\left(x^k\right), \tag{19}$$

$$x^{k+1} = x^k - \alpha_{\mathrm{BB2}}^k \nabla f\left(x^k\right), \tag{20}$$

where x^k is the solution at iteration k, and $g^k = \nabla f(x^k)$ is the gradient. The step sizes α_{BB1}^k and α_{BB2}^k are defined as:

$$\alpha_{\mathrm{BB1}}^k = \frac{\left(s^{k-1}\right)^{\mathrm{T}} y^{k-1}}{\left(y^{k-1}\right)^{\mathrm{T}} y^{k-1}}, \quad \alpha_{\mathrm{BB2}}^k = \frac{\left(s^{k-1}\right)^{\mathrm{T}} s^{k-1}}{\left(s^{k-1}\right)^{\mathrm{T}} y^{k-1}},$$

where:

$$s^{k-1} = x^k - x^{k-1}, \quad y^{k-1} = \nabla f\left(x^k\right) - \nabla f\left(x^{k-1}\right).$$

The BB step size strategy requires only information from two consecutive iterates and gradients, making it simple yet effective for accelerating convergence. It adapts the step size dynamically based on the progress of the algorithm.

In practice, the BB step size is combined with a non-monotone line search to further enhance performance. The non-monotone line search allows temporary increases in the objective function value to avoid overly conservative step sizes. The algorithm can be described as follows:

Algorithm 3 Non-Monotone Line Search with BB Step Size

```
Initialize $x^0$, step size $\alpha > 0$, parameters $M \geq 0$, $c_1, \beta, \varepsilon \in (0, 1)$, and set $k = 0$.
while $\|\nabla f(x^k)\| > \varepsilon$ do
    while $f\left(x^k - \alpha \nabla f(x^k)\right) \geq \max_{0 \leq j \leq \min(k,M)} f\left(x^{k-j}\right) - c_1 \alpha \|\nabla f(x^k)\|^2$ do
        Update $\alpha \leftarrow \beta\alpha$.
    end while
    Update $x^{k+1} = x^k - \alpha \nabla f(x^k)$.
    Compute BB step size $\alpha$ using (19) or (20) and truncate it to $[\alpha_m, \alpha_M]$.
    Set $k \leftarrow k + 1$.
end while
```

The BB step size strategy is applied to the SGD, ProxGD, and FProxGD algorithms in this project. Its simplicity and efficiency make it suitable for accelerating convergence. By integrating it with the non-monotone line search, the algorithms achieve better practical performance.

The combination of the continuation strategy and the BB step size update provides a robust framework for solving the LASSO problem efficiently, leveraging both warm-starting and adaptive step size updates.

3 Results

We implement these algorithms in Matlab and related toolboxes based on [3]. The results are listed as the following:

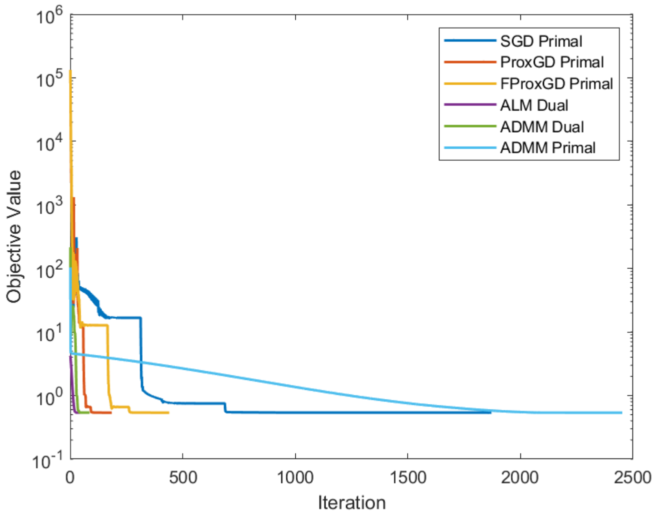

Figure 1: All solvers

Method	CPU (sec)	Iterations	Optval	Sparsity	Err to Exact	Err to CVX-Mosek	Err to CVX-Gurobi
CVX-Mosek	1.45	-1	5.38327E-01	0.115	4.21E-05	0.00E+00	7.49E-08
CVX-Gurobi	0.64	-1	5.38327E-01	0.115	4.21E-05	7.49E-08	0.00E+00
Mosek	12.90	-1	5.38327E-01	0.114	4.20E-05	9.87E-08	7.25E-08
Gurobi	15.73	-1	5.38327E-01	0.115	4.22E-05	2.17E-07	2.50E-07
SGD Primal	0.39	1871	5.38331E-01	0.164	6.33E-05	2.59E-05	2.59E-05
ProxGD Primal	0.06	185	5.38327E-01	0.114	4.20E-05	1.65E-07	1.33E-07
FProxGD Primal	0.07	440	5.38327E-01	0.114	4.20E-05	1.74E-07	1.44E-07
ALM Dual	0.30	61	5.38345E-01	0.100	7.84E-05	4.41E-05	4.41E-05
ADMM Dual	0.09	86	5.38341E-01	0.100	7.50E-05	3.84E-05	3.84E-05
ADMM Primal	8.80	2452	5.38327E-01	0.114	4.20E-05	2.11E-07	1.90E-07

Table 1: Optimization Performance Comparison

Table Description and Comparison:

- **CPU Time:** The time in seconds taken by each method to converge to a solution. The ProxGD Primal solver is the fastest, taking only 0.06 seconds, followed by ADMM Dual Primal (0.09 seconds). The methods that employ optimization methods directly (e.g. ProxGD, FProxGD) have relatively low CPU times compared to toolboxes like Gurobi and Mosek. However, ADMM Primal is very slow as well, indicating the improvement of formulating to dual problem.

- **Iterations:** Indicates how many iterations each method requires to reach a solution. Methods such as SGD and ADMM Primal require significantly more iterations, especially ADMM Primal, which requires 2452 iterations. This reflects the complexity of solving the problem using primal methods than dual iterations.

- **Optimal Value (Optval):** The optimal value achieved by each method is similar across most solvers, hovering around 5.38327E-01, demonstrating the consistency of the solution across different algorithms.

- **Sparsity:** The sparsity of the solution indicates how many zero elements are present. Most methods show a sparsity between 0.114 and 0.115, except for SGD Primal, which results in slightly higher sparsity of 0.164. This is likely due to the regularization used in SGD.

- **Error to Exact:** The difference between the computed solution and the exact solution. The errors across methods are generally very small, with most solvers achieving errors on the order of 10^{-5}.

- **Error to CVX-Mosek and CVX-Gurobi:** These errors indicate how close the computed solution is to the results from the two commercial solvers (Mosek and Gurobi). Methods like CVX-Mosek and CVX-Gurobi have zero error when compared to themselves, while other methods such as SGD Primal show errors in the range of 10^{-5} to 10^{-7}. This is a reasonable trade-off, considering the computational savings and flexibility of the other methods.

Optimization Perspective:

From an optimization viewpoint, the primal methods (ProxGD, FProxGD) offer faster computation times but at the cost of slightly higher iteration counts and more approximation errors, especially in terms of sparsity and error compared to CVX solvers. The dual methods, ALM and ADMM, balance between computational cost and accuracy, with ALM providing a relatively fast solution and ADMM showing efficiency despite a higher iteration count. ADMM, while requiring more time and iterations, produces a solution very close to that of commercial solvers, demonstrating its effectiveness in practical applications.

Conclusion

In this lab report, we investigated various solvers for solving the group LASSO problem, which is a structured sparse optimization problem that arises frequently in high-dimensional statistics and machine learning. The objective was to compare both traditional commercial solvers (CVX-Mosek and CVX-Gurobi) and modern first-order optimization techniques (SGD, ProxGD, FProxGD, ALM, and ADMM) in terms of computational performance, solution accuracy, and convergence behavior.

Key Observations from Solver Comparison

1. CVX-Mosek and CVX-Gurobi: The commercial solvers, CVX-Mosek and CVX-Gurobi, performed exceptionally well in terms of both accuracy and speed. They achieved optimal values very close to the exact solution, with negligible errors (on the order of 10^{-7} for CVX-Gurobi). These solvers also demonstrated fast convergence, with CVX-Gurobi completing the problem in 0.64 seconds, and CVX-Mosek taking 1.45 seconds. Their computational efficiency and precision make them reliable choices for solving large-scale group LASSO problems, particularly when exact solutions are needed.

2. Mosek and Gurobi (Standalone Solvers): While Mosek and Gurobi alone took longer to converge (12.90 and 15.73 seconds, respectively), they still produced accurate solutions with similar sparsity and errors to the CVX-based solvers. These results highlight that, in scenarios where commercial solvers are available, they remain the most reliable options for obtaining exact solutions, although they require more time compared to modern first-order methods.

3. First-Order Methods (SGD, ProxGD, FProxGD):

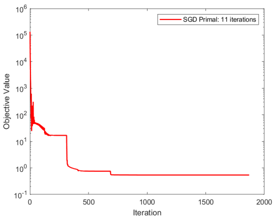

Figure 2: SGD-P

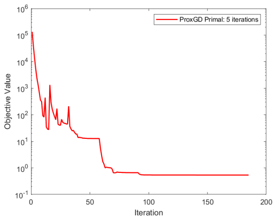

Figure 3: ProxGD-P

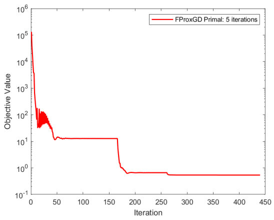

Figure 4: FProxGD-P

Among the first-order methods, SGD, ProxGD, and FProxGD showed promising results in terms of computational efficiency. SGD, for example, completed the optimization in just 0.39 seconds, though it required a higher number of iterations (1871). The optimal value was very close to that of the commercial solvers, but the sparsity in SGD's solution was slightly higher, indicating less precision in the regularization. ProxGD and FProxGD, although slightly slower than SGD, achieved solutions that were very close to the commercial solvers in terms of accuracy and sparsity. These methods, being more flexible and scalable, are particularly useful in situations where the problem size is large, and the computational budget is limited.

4. Dual Methods (ALM and ADMM):

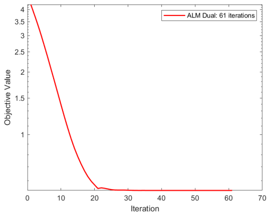

Figure 5: ALM-D

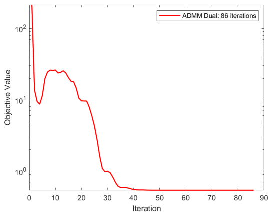

Figure 6: ADMM-D

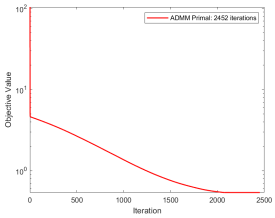

Figure 7: ADMM-P

The dual methods, ALM and ADMM, demonstrated a balance between solution accuracy and computational efficiency. ALM, in particular, achieved a fast convergence in 0.30 seconds with a reasonable error to the exact solution. ADMM, while slower (taking 8.80 seconds), produced very accurate solutions and had lower errors compared to the primal methods. ALM and ADMM are highly effective for structured problems, where dual formulations allow for faster convergence with fewer iterations, making them suitable for cases where precision and computational efficiency are both critical.

Characteristics of the Group LASSO Problem

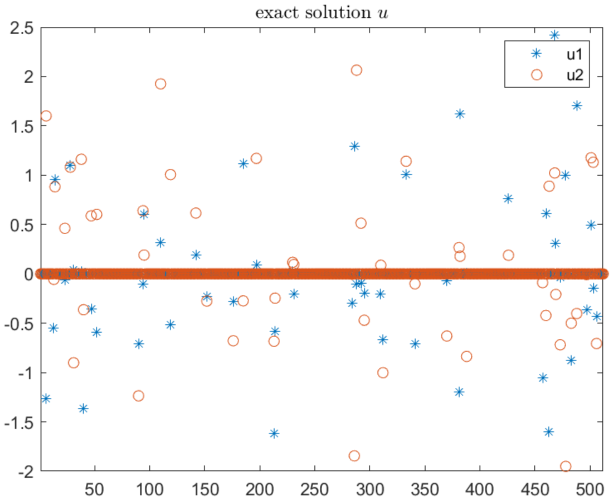

Figure 8: Distribution of first two columns of u

The group LASSO problem is characterized by the presence of both group-wise sparsity and a regularization term that encourages sparse solutions. This problem exhibits strong structure, with regularization being applied to groups of variables rather than individual variables. Thus, solvers must effectively exploit this structure to ensure both computational efficiency and solution accuracy.

In this lab, the comparison of solvers was impacted by the fact that the group LASSO problem naturally favors methods that handle sparsity well. Commercial solvers like Mosek and Gurobi, which implement advanced optimization techniques and exploit problem structure, performed excellently in terms of both accuracy and efficiency. However, the flexibility of first-order methods such as SGD and ProxGD offers a significant advantage in terms of scalability, especially for large-scale problems where commercial solvers may become computationally expensive.

Optimization Techniques Employed

The use of advanced optimization techniques such as **SGD**, **ProxGD**, and **FProxGD** demonstrated the power of modern first-order methods in efficiently solving sparse optimization problems. These methods rely on gradient information, which is computationally efficient to obtain, allowing for faster convergence compared to more traditional solvers. Moreover, the **continuation strategy** employed in the SGD, ProxGD, and FProxGD algorithms was effective in accelerating the solution process, especially when transitioning from a larger regularization parameter to a smaller one. This approach reduced the solution space complexity and improved the convergence speed by leveraging previously computed solutions as good initial guesses.

On the other hand, dual methods like **ALM** and **ADMM** provided a more robust approach to handle the group LASSO problem's dual formulation. The ALM method, by augmenting the Lagrangian and solving the resulting problem iteratively, struck a balance between the primal and dual formulations. ADMM, which splits the problem into simpler subproblems and alternates between them, demonstrated strong performance in both precision and efficiency.

Conclusion

In conclusion, while traditional solvers like CVX-Mosek and CVX-Gurobi offer the best solution accuracy and fast convergence for smaller to medium-sized problems, modern first-order optimization techniques such as SGD, ProxGD, and FProxGD offer significant computational savings and are highly scalable for large-scale problems. Among these, ProxGD and FProxGD achieved high accuracy with relatively fast runtimes. The dual methods (ALM and ADMM) showed strong performance in handling the structure of the group LASSO problem, offering a balance of efficiency and solution accuracy.

For practical applications where computational efficiency is critical, especially with large datasets or when scalability is a concern, first-order methods (especially ProxGD and FProxGD) should be preferred. However, for problems requiring high precision or where computational resources are available, commercial solvers such as CVX-Mosek and CVX-Gurobi remain the gold standard. The use of continuation strategies in first-order methods provides an additional advantage, speeding up convergence and reducing the computational cost of the problem.

Future work could involve further refining these algorithms for better handling of extremely large problems, potentially integrating hybrid methods that combine the strengths of both primal and dual optimization techniques.

References

[1] Zaiwen Wen. Program submitting guide, 11 2020. URL http://faculty.bicmr.pku.edu.cn/~wenzw/opt2015/homework5-req.pdf.

[2] Zaiwen Wen. Continuization of lasso. URL http://faculty.bicmr.pku.edu.cn/~wenzw/optbook/pages/LASSO_con/LASSO_con.html.

[3] AkexStar. Algorithms for group lasso problem. https://github.com/AkexStar/Algorithms-group-LASSO-problem.git, 2024. Accessed: 2024-12-17.