Midterm report

Ruicheng Ao 1900012179

March 1, 2022

Contents

1 l_1 minimization

1.1 Problem settings

In this section, we consider the following optimization problem

$$\min_x \quad \mu\|x\|_1 + \|Ax-b\|_1, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ are given.

Problem (1) can be reformulated into a constrained problem by introducing constraint $y = Ax - b$:

$$\begin{array}{cc} \min_{x,y} & \mu\|x\|_1 + \|y\|_1 \\ \text{s.t.} & y = Ax-b \end{array} \tag{2}$$

with its dual problem given by

$$\begin{array}{cc} \max_\lambda & -b^\top\lambda \\ \text{s.t.} & \|\lambda\|_1 \le 1, \|A^\top\lambda\|_1 \le 1 \end{array} \tag{3}$$

1.2 Agumented Lagragian method

In the past, penalty function methods had been prevailed by directly adding a constraint violation penalty to the target function [2, section 10] and converting the constrained problem into an unconstrained problem. For example, thet quadratic penalty method is welcomed due to its simple form and easy implementation. However, these kinds of methods come up with disadvantages such as generating ill-consitioned problems when the penalty factors are large, which impairs the convergence.

The augmented Lagrangian method, however, relieves such difficulty by introducing the Lagrangian multipliers combined with quadratic penalty function. To speak more specifically, by applying such technique, an augmented Lagrangian function of primal problem (1) is of the form

$$L(x, y, \lambda, \tau) = \mu\|x\|_1 + \|y\|_1 + \lambda^\top(Ax - b - y) + \frac{\tau}{2}\|Ax - b - y\|_2^2. \tag{4}$$

One of the most obvious advantage of this kind of method is implied in that, throughout the optimization process, there is no necessity for the augmented Lagrangian method to set gradually *tau* to infinity in order to get the optimal point, which avoids ill-conditioned subproblems. Algorithm 1 shows a general framework of augmented Lagrangian method.

Algorithm 1 Augmented Lagrangian method

```
1: Initialize $x^0, y^0, \lambda^0$
2: while Not converge do
3:     Solve $(x^{k+1}, y^{k+1}) = \arg\min_{x,y} L(x, y, \lambda^k, \tau))$
4:     Update Lagrangian multiplier $\lambda^{k+1} = \lambda^k + \tau(Ax^{k+1} - b - y^{k+1})$
5:     $k = k + 1$
6: end while
7: return $x^k, y^k$
```

It should be mentioned that, in our form of constraint $y = Ax - b$, one can show that the augmented Lagrangian method is equivalent to the well-known Bregman iterative method (Algorithm 2) when $\tau = 1$ in ALM. [5] has proved the following results of convergence for Bregman iterative method.

Theorem 1. *(Yin)*

(a) Suppose the iterate (x^k, y^k) in Algorithm 2 satisfies $Ax^k - y^k = b$, then (x^k, y^k) are the solutions of primal problem (2).

(b) There exists a number $K < \infty$, such that for any $k > K$, (x^k, y^k) is a solution of (2).

Algorithm 2 Bregman iterative method

```
1: Initialize $x^0, y^0, f^0$
2: while Not converge do
3:     Update $f^{k+1} = f^k - (Ax^k - y - b)$
4:     Solve $(x^{k+1}, y^{k+1}) = \arg\min_{x,y} H(x, y, f, \tau) = \mu\|x\|_1 + \|y\|1 + \frac{\tau}{2}\|Ax - y - f^k\|_2^2)$
5:     $k = k + 1$
6: end while
7: return $x^k, y^k$
```

We apply the fast iterative shrinkage-thresholding algorithm (FISTA) [1] for solving the subproblem in Algorithm 1, which is illustrated in Algorithm 3. We split the augmented Lagrangian function into two parts in order to use FISTA, i.e., $f(x,y) = \lambda^\top(Ax-b-y) + \frac{\tau}{2}\|Ax-b-y\|_2^2, g(x,y) = \mu\|x\|_1 + \|y\|_1$. The proximal operator of $tg(x.y), t > 0$ composes of two parts. One is $\text{prox}_{t\|\cdot\|_1}(x)$, which can be expressed explicitly as

$$\text{prox}_{t\|\cdot\|_1}(x)_i = \begin{cases} x_i + t, & x_i < -t, \\ 0, & |x_i| \le t, \\ x_i - t, & x_i > t, \end{cases} \tag{5}$$

where the computational complexity of each iteration is $\mathcal{O}(mn)$.

The following theorem in [1] demonstrates the general convergence of FISTA:

Theorem 2. *Let $\{x^k\}$ be the iterates of FISTA, then for any $k \ge 1$, $F(x) = f(x) + g(x)$ satisfies*

$$F(x^k) - F(x^*) \le \frac{2L\|x^0 - x^*\|_2^2}{(k+1)^2}, \forall x^* \textit{ optimal.} \tag{6}$$

Thus the computation complexity for finding an $\epsilon-$optimal point is $\mathcal{O}(\frac{mnL^{\frac{1}{2}}}{\epsilon^{\frac{1}{2}}})$ with the cost per iteration $\mathcal{O}(mn)$.

Algorithm 3 Fast iterative shrinkage-thresholding algorithm

```
1: Given $f, g$, initial point $x^0$, Lipschitz coefficient $L$ of $\nabla f$
2: Set $y^1 = x^1, t^1 = 1$
3: while Not converge do
4:     Solve $x^k = \arg\min_x \text{prox}_{g(x)/L}(y^k - \frac{1}{L}\nabla f(y^k))$
5:     Update $t^{k+1} = \frac{1+\sqrt{1+4(t^{(k)})^2}}{2}$
6:     Update $y^{k+1} = x^k + \frac{t^k-1}{t^k+1}(x^k - x^{k-1})$
7:     $k = k + 1$
8: end while
9: return $x^k$
```

To the end of this section, we illustrate the practical implementation of augmented Lagrangian method in Algorithm 4.

Algorithm 4 Augmented Lagrangian method

```
1: Given tolerance $tol, \varepsilon^0$, penalty parameter $\tau^0$, initial point $(x^0, y^0)$, $\lambda^0$
2: while $|\mu\|x\|_1 + \|y\|_1 + b^\top\lambda| + \|Ax - b - y\|_2^2 > tol$ do
3:     Using FISTA to obtain $\epsilon^k-$optimal solution $(x^k, y^k)$ of $L(x, y, \lambda^k, \tau^k)$
4:     Update Lagrangian multiplier $\lambda^{k+1} = \lambda^k + \tau^k(Ax^k - b - y^k)$
5:     Choose initial point for FISTA of the next subproblem and updata tolerance$\epsilon^{k+1} \le \epsilon^k$, penalty parameter
       $\tau^{k+1} \ge \tau^k$.
6:     $k = k + 1$
7: end while
8: return $x^k, y^k$
```

1.3 Alternating direction method of multipliers

In this section, we demonstrate alternating direction metod of multipliers (ADMM) for solving primal problem (2). We rewrite the augmented Lagrangian function as follows

$$L(x, y, \lambda, \tau) = \mu\|x\|_1 + \|y\|_1 + \lambda^\top(Ax - b - y) + \frac{\tau}{2}\|Ax - b - y\|_2^2. \tag{7}$$

In each iteration of ALM, (x^k, y^k) are updated simultaneously, which leads to costly computation. To address this problem, ADMM modifies the iteration by updating x^k, y^k, λ^k separately. We demonstrate the framework of ADMM in Algorithm 5. The convergence of ADMM is established in [3].

Algorithm 5 Alternating direction method of multipliers

```
1: Initialize $x^0, y^0, \lambda^0, \gamma \in (0, \frac{\sqrt{5}+1}{2})$
2: while Not converge do
3:     Solve $x^{k+1} = \arg\min_x L(x, y^k, \lambda^k, \tau)$
4:     Solve $y^{k+1} = \arg\min_y L(x^{k+1}, y, \lambda^k, \tau)$
5:     Update Lagrangian multiplier $\lambda^{k+1} = \lambda^k + \gamma\tau(Ax^{k+1} - b - y^{k+1})$
6: end while
7: return $x^k, y^k$
```

However, in some situations, solving the subproblem $x^{k+1} = \arg\min_x L(x, y^k, \lambda^k, \tau)$ is intractable. [4] proposed a linearized ADMM scheme demonstrated in Algorithm 6. Each iteration can be proceed quickly by applying proximal operators as mentioned before. Also, since there is no difficulty to directly compute y^{k+1} with the help of proximal operators, we can update y^k by just solving the subproblem and get

$$y^{k+1} = \text{prox}_{\|\cdot\|_1/\tau}(Ax^{k+1} - b + \lambda^k/\tau). \tag{8}$$

The following theorem in [4] establishes the convergence of linearized ADMM.

Theorem 3. *Suppose the stepsize α, γ satisfy $\alpha\lambda_{\max} + \gamma < 2$, where $\lambda_{\max}$ denotes the largest eigenvalue of $A^\top A$. Then for primal problem $\mu\|x\|_1 + \|y\|_2^2$, the iterates generated by linearized ADMM converges globally to the optimal points.*

Note that the convergence analysis is applied to quadratic function of y, nevertheless, it does not hamper the performance of such method. Since the iterates always converge fast to the optimum, where $y = 0$.

Algorithm 6 Linearized ADMM

```
1: Initialize $x^0, y^0, \lambda^0, \gamma \in (0, \frac{\sqrt{5}+1}{2})$, stepsize $\alpha$
2: while Not converge do
3:     Solve $x^{k+1} = \arg\min_x \mu\|x\|_1 + \langle A^\top(\lambda + \tau(Ax^k - b - y^k)), x\rangle + \frac{1}{2\alpha}\|x - x^k\|_2^2$
4:     Solve $y^{k+1} = \arg\min_y L(x^{k+1}, y, \lambda^k, \tau)$
5:     Update Lagrangian multiplier $\lambda^{k+1} = \lambda^k + \gamma\tau(Ax^{k+1} - b - y^{k+1})$
6: end while
7: return $x^k, y^k$
```

1.4 Numerical results

We generate data for experiments through MATLAB code provided in project requirement, i.e.,

$$\begin{aligned}&\mathrm{A}=\ \mathrm{randn}\ (\mathrm{m},\mathrm{n});\\&\mathrm{u}=\ \mathrm{sprandn}\ (\mathrm{n},1,0.1);\\&\mathrm{b}=\mathrm{A}*\mathrm{u};\\&\mathrm{mu}=1\mathrm{e}-2\end{aligned}$$

The random seed is fixed for so that the experiments can be repeated. $(m, n) = (256, 128), (512, 256), (1024, 512), (2048, 1024)$ are chosen. We compare the performance of Mosek, Gurobi on CVX, augmented Lagrangian method and linearized ADMM on the primal problem. ALM is implemented with both FISTA/ ISTA for comparison. For parameters, we choose $\epsilon^k = 1e-8, \tau = 1e-3$ and $\gamma = \frac{\sqrt{5}+1}{2}, \alpha = \frac{1}{\tau\lambda_{\max}}$ for ADMM. We use randomly generated initial points. The stopping toleration is chosen to be $1e-10$ based on the difference of two following steps.

The numerical results are displayed in Table 1 , we do not list the number of iterations since the time shows much more importance. From the table, we see that the ALM and linearized ADMM outperform the CVX toolbox. High-quality solutions are obtained with negligible violataion of constraints. For ALM, the superiority of FISTA can be found in the results. In all scales of problems, linearized ADMM shows the fastest convergence speed, which partly derives from its exploitation of the problem structure and less proximate operators used.

Algorithm	$(m,n)=(128,256)$					
	time	optvalue	violation	optgap	error to mosek	$\|x\|_1$
Mosek	1.16e+00	1.76e-07	-	-	-	2.33e+01
Gurobi	9.00e-01	2.33e-09	-	-	3.74e-08	2.33e+01
ALM with ISTA	3.98e-02	4.74e-09	7.75e-11	1.51e-11	3.65e-08	2.33e+01
ALM with FISTA	4.66e-02	3.51e-09	5.64e-11	2.00e-11	3.65e-08	2.33e+01
ADMM	4.28e-02	2.76e-09	2.31e-09	4.38e-12	3.65e-08	2.33e+01
Algorithm	$(m,n)=(256,512)$					
	time	optvalue	violation	optgap	error to mosek	$\|x\|_1$
Mosek	2.11e+00	3.10e-10	-	-	-	4.39e+01
Gurobi	1.51e+00	7.53e-09	-	-	6.32e-10	4.39e+01
ALM with ISTA	1.03e-01	1.94e-09	4.10e-11	3.98e-11	5.65e-11	4.39e+01
ALM with FISTA	1.08e-01	2.96e-09	6.12e-11	3.18e-11	1.18e-10	4.39e+01
ADMM	3.80e-02	1.06e-10	2.46e-10	7.12e-13	4.86e-11	4.39e+01
	time	optvalue	violation	optgap	error to mosek	$\|x\|_1$
Mosek	2.81e+00	5.19e-08	-	-	-	8.25e+01
Gurobi	4.79e+00	2.46e-09	-	-	1.99e-08	8.25e+01
ALM with ISTA	2.31e-01	3.44e-09	8.85e-11	3.83e-12	6.30e-09	8.25e+01
ALM with FISTA	1.78e-01	3.76e-09	8.14e-11	2.39e-12	6.41e-09	8.25e+01
ADMM	1.26e-01	1.81e-10	1.08e-09	4.79e-12	6.35e-09	8.25e+01
Algorithm	$(m,n)=(1024,2048)$					
	time	optvalue	violation	optgap	error to mosek	$\|x\|_1$
Mosek	8.87e+00	1.00e-08	-	-	-	1.69e+02
Gurobi	7.15e+01	7.11e-07	-	-	1.06e-07	1.69e+02
ALM with ISTA	1.90e+00	5.46e-10	1.89e-11	2.93e-11	1.24e-08	1.69e+02
ALM with FISTA	1.36e+00	2.26e-09	8.18e-11	6.48e-12	1.24e-08	1.69e+02
ADMM	1.20e+00	7.95e-12	1.34e-10	1.07e-12	1.24e-08	1.69e+02

Table 1: Numerical results for L_1 minimization. Violation means relative constraint violation defined by $\frac{\|Ax-y-b\|_2}{\|b\|_2}$. CPU time is recorded. Optvaule represents $\frac{|f(x,y)-f(x^*,y^*)}{|f(x^*,y^*)|}$, where $f(x,y)=\mu\|x\|_1+\|y\|_1$. Error to mosek denotes the relative l_2 difference with the solution of mosek.

2 Low-rank recovery

2.1 Problem settings

In this section, we consider the following problem

$$\min_{X\in\mathbb{R}^{m\times n}} \quad \mu\|X\|_* + \textstyle\sum_{(i,j)\in\Omega}(X_{ij}-M_{ij})^2, \tag{9}$$

where $\|X\|_* = \sum_i \sigma_i(X)$ denotes the nuclear norm (sum of singular values). The projection operator is defined as

$$[P_\Omega(X)]_{ij} = \begin{cases} B_{ij} & (i,j)\in\Omega \\ 0 & (i,j)\notin\Omega \end{cases}$$

Then the loss function can be written as

$$f(X) = \underbrace{\|P_\Omega(M) - P_\Omega(X)\|_F^2}_{g(X)} + \underbrace{\mu\|X\|_*}_{h(X)}, \tag{10}$$

which is splitted into two parts so that we can apply proximal gradient method or convert (10) into a constrained problem by introducing additional variable such that

$$\begin{array}{ll} \min_{X,Y} & f(X,Y) = \underbrace{\|Y\|_F^2}_{g} + \underbrace{\mu\|X\|_*}_{h} \\ \text{s.t.} & P_\Omega(X) - P_\Omega(M) = Y \end{array} \tag{11}$$

2.2 Proximal gradient method

The problem form in (10) naturally splits the objective funciton into two part, namely, $g(X)$ is smooth and convex in X. Therefore, proximal radient method can be implemented to solve such problem.

For g, the gradient is simply

$$\nabla g(X) = 2(P_\Omega(X) - P_\Omega(M)). \tag{12}$$

To find the explicit solution to proximal point

$$\text{prox}_{th}(X) = \arg\min_Z \frac{1}{2t}\|Z - X\|_F^2 + \mu\|Z\|_*. \tag{13}$$

Fortunately, in our previous assignment, we have already known its expression

$$\text{prox}_{th}(X) = S_{\mu t}(X), \tag{14}$$

where $S_{\mu t}$ is the shrinkage operator defined by $S_\lambda(X) = U\Sigma_\lambda V^\top$, $X = U\Sigma V^\top$ is the SVD and Σ is diagonal with $(\Sigma)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\}$. Then vanilla proximal gradient or Algorithm 3 can be implemented to solve the problem.

2.3 Alternating direction method of multipliers

In this section, we demonstrate alternating direction method of multipliers (ADMM) for solving primal problem (11). We rewrite the augmented Lagrandian function as follows

$$L(X, Y, \Lambda, \tau) = \mu\|X\|_* + \|Y\|_F^2 + \langle \Lambda, P_\Omega(X - M) - Y\rangle + \frac{\tau}{2}\|P_\Omega(X - M) - Y\|_F^2. \tag{15}$$

The framework of ADMM in such matrix form is given in Algorithm 7.

Algorithm 7 Alternating direction method of multipliers

```
1: Initialize $X^0, Y^0, \lambda^0, \gamma \in (0, \frac{\sqrt{5}+1}{2})$
2: while Not converge do
3:     Solve $X^{k+1} = \arg\min_X L(X, Y^k, \Lambda^k, \tau)$
4:     Solve $Y^{k+1} = \arg\min_Y L(X^{k+1}, Y, \Lambda^k, \tau)$
5:     Update Lagrangian multiplier $\Lambda^{k+1} = \Lambda^k + \gamma\tau(P_\Omega(X^{k+1}) - P_\Omega(M) - Y^{k+1})$
6: end while
7: return $X^k, Y^k$
```

However, in this matrix completion task, solving the subproblem $X^{k+1} = \arg\min_X L(X, Y^k, \Lambda^k, \tau)$ is intractable by many times of SVD. [4] proposed a linearized ADMM scheme demonstrated in Algorithm 6. Each iteration can be proceed quickly by applying proximal operators as mentioned before. Also, since there is no difficulty to directly compute Y^{k+1}:

$$Y^{k+1} = (2+\tau)^{-1}(\tau P_\Omega(X-M)+\Lambda). \tag{16}$$

By applying proximal operator of nuclear norm induced before, each updata can be computed quickly.

Algorithm 8 Linearized ADMM

```
1: Initialize $X^0, Y^0, \lambda^0, \gamma \in (0, \frac{\sqrt{5}+1}{2})$, stepsize $\alpha$
2: while Not converge do
3:     Solve $X^{k+1} = \arg\min_X \mu\|X\|_* + \langle (P_\Omega(\Lambda + \tau(X^k - M - Y)), X\rangle + \frac{1}{2\alpha}\|X - X^k\|_2^2$
4:     Solve $Y^{k+1} = \arg\min_Y L(X^{k+1}, Y, \lambda^k, \tau)$
5:     Update Lagrangian multiplier $\Lambda^{k+1} = \Lambda^k + \gamma\tau(P_\Omega(X^{k+1}) - P_\Omega(M) - Y^{k+1})$
6: end while
7: return $x^k, y^k$
```

2.4 Numerical results

We implement the numerical experiments on the first three datasets provided by the project document with $\mu = 0.1, 0.01, 0.001$. Mosek, Proximal gradient method (ISTA), Fast Proximal gradient method (FISTA) and linearized ADMM on primal problem are compared. We apply continuum strategy with a start $\mu = 1000$ each experiment. Step size is chosen to be 0.1 for proximal gradient and $\frac{0.1}{\tau}$ for ADMM, while $\tau = 1$. The toleration is chosen to be $1e-8$. Comparison involves the CPU time, the optimal value, the relative error of $\|P_\Omega(X) - P_\Omega(M)\|_F / \|P_\Omega(M)\|_F$ and the recovery error $\|X - A\|_F / \|A\|$, where A is the primitive matrix. The results are listed in Table 2-4.

Algorithm	μ =0.1				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_\mathcal{F}$	Error with A in norm $\|\cdot\|_\mathcal{F}$	$\|x\|_\mathcal{F}$
Mosek	$2.40e+00$	$1.19e+01$	$5.43e-03$	$1.58e-01$	$1.19e+02$
Proximal (ISTA)	$3.36e-01$	$1.39e+01$	$5.81e-03$	$6.25e-01$	$1.37e+02$
Proximal (FISTA)	$2.81e-01$	$1.21e+01$	$5.44e-03$	$1.61e-01$	$1.19e+02$
ADMM	$2.34e-01$	$1.19e+01$	$5.47e-03$	$1.73e-01$	$1.19e+02$
Algorithm	μ =0.01				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_\mathcal{F}$	Error with A in norm $\|\cdot\|_\mathcal{F}$	$\|x\|_\mathcal{F}$
Mosek	$1.28e+00$	$1.20e+00$	$5.76e-04$	$1.56e-01$	$1.20e+02$
Proximal (ISTA)	$3.62e-01$	$1.67e+00$	$6.37e-04$	$7.76e-01$	$1.67e+02$
Proximal (FISTA)	$3.13e-01$	$1.20e+00$	$5.71e-04$	$1.39e-01$	$1.20e+02$
ADMM	$3.79e-01$	$1.20e+00$	$5.79e-04$	$2.02e-01$	$1.20e+02$
Algorithm	μ =0.001				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_\mathcal{F}$	Error with A in norm $\|\cdot\|_\mathcal{F}$	$\|x\|_\mathcal{F}$
Mosek	$1.19e+00$	$1.20e-01$	$5.77e-05$	$1.56e-01$	$1.20e+02$
Proximal (ISTA)	$3.23e-01$	$1.74e-01$	$6.52e-05$	$7.94e-01$	$1.74e+02$
Proximal (FISTA)	$2.89e-01$	$1.20e-01$	$5.90e-05$	$2.36e-01$	$1.20e+02$
ADMM	$2.09e-01$	$1.20e-01$	$5.91e-05$	$2.50e-01$	$1.20e+02$

Table 2: Numerical result on example 1 with different μ

Algorithm	μ =0.1				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$7.81e+02$	$2.05e+02$	$6.82e-04$	$8.66e-04$	$2.05e+03$
Proximal (ISTA)	$1.07e+01$	$3.52e+02$	$1.39e-03$	$6.36e-01$	$3.51e+03$
Proximal (FISTA)	$1.51e+01$	$2.05e+02$	$6.82e-04$	$8.66e-04$	$2.05e+03$
ADMM	$4.08e+00$	$2.05e+02$	$6.82e-04$	$8.66e-04$	$2.05e+03$
Algorithm	$\mu=0.01$				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$8.59e+02$	$2.05e+01$	$6.83e-05$	$8.72e-05$	$2.05e+03$
Proximal (ISTA)	$2.33e+01$	$2.62e+01$	$7.62e-05$	$8.81e-05$	$2.12e+03$
Proximal (FISTA)	$1.25e+01$	$2.05e+01$	$6.85e-05$	$8.92e-05$	$2.05e+03$
ADMM	$3.29e+00$	$2.05e+01$	$6.82e-05$	$8.67e-05$	$2.05e+03$
Algorithm	μ =0.001				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$1.04e+03$	$2.05e+00$	$6.83e-06$	$9.87e-06$	$2.05e+03$
Proximal (ISTA)	$8.50e+00$	$4.13e+00$	$1.44e-05$	$7.45e-01$	$4.13e+03$
Proximal (FISTA)	$1.38e+01$	$2.42e+00$	$1.45e-05$	$2.51e-01$	$2.42e+03$
ADMM	$5.23e+00$	$2.42e+00$	$1.32e-05$	$2.57e-01$	$2.42e+03$

Table 4: Numerical result on example 3 with different μ

Algorithm	μ =0.1				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$9.70e+00$	$1.99e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
Proximal (ISTA)	$2.04e+00$	$3.41e+01$	$4.64e-03$	$6.14e-01$	$3.36e+02$
Proximal (FISTA)	$1.80e+00$	$2.00e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
ADMM	$7.61e-01$	$1.99e+01$	$1.86e-03$	$2.33e-03$	$1.99e+02$
Algorithm	μ =0.01				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$1.11e+01$	$2.00e+00$	$1.86e-04$	$2.33e-04$	$2.00e+02$
Proximal (ISTA)	$2.02e+00$	$4.49e+00$	$5.11e-04$	$7.94e-01$	$4.49e+02$
Proximal (FISTA)	$1.89e+00$	$2.00e+00$	$2.84e-04$	$7.54e-03$	$2.00e+02$
ADMM	$8.40e-01$	$2.00e+00$	$1.86e-04$	$2.33e-04$	$2.00e+02$
Algorithm	μ =0.001				
	CPU time	Opt value	Error with M in norm $\|\cdot\|_{\mathcal{F}}$	Error with A in norm $\|\cdot\|_{\mathcal{F}}$	$\|x\|_{\mathcal{F}}$
Mosek	$1.31e+01$	$2.00e-01$	$1.86e-05$	$2.37e-05$	$2.00e+02$
Proximal (ISTA)	$4.76e+00$	$4.72e-01$	$5.25e-05$	$8.13e-01$	$4.72e+02$
Proximal (FISTA)	$3.63e+00$	$2.06e-01$	$4.10e-05$	$3.73e-02$	$2.06e+02$
ADMM	$2.89e+00$	$2.00e-01$	$2.99e-05$	$1.49e-02$	$2.00e+02$

Table 3: Numerical result on example 2 with different μ

We observe that among all these algorithms, ADMM performs the best, both in time and the recovery, while all of our proposed algorithms outperform the baseline Mosek many times except ISTA, which only finds an approximate solution in a very short time. Note that sometimes FISTA costs more time than ISTA, which is because ISTA gets its maximal number of iterations thus leaves an inaccurate solution. The results is consistent with our previous knowledge that ADMM truly has its superiorty, while FISTA has good acceleration and pertains the quality of solution compared with ISTA. Although in some large-scale problem such as example 3, it is exhaustive to get a accurate solution, ADMM still approximates the primitive matrix in 1/200 time less than mosek.

References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Roland Glowinski and Patrick Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. SIAM, 1989.

[4] Junfeng Yang and Yin Zhang. Alternating direction algorithms for \ell_1-problems in compressive sensing. *SIAM journal on scientific computing*, 33(1):250–278, 2011.

[5] Wotao Yin, Stanley Osher, Donald Goldfarb, and Jerome Darbon. Bregman iterative algorithms for \ell_1-minimization with applications to compressed sensing. *SIAM Journal on Imaging sciences*, 1(1):143–168, 2008.