Course Project: Semi-smooth Newton Algorithms for the standard form LP

Li Xiang, Lin Dachao, Ni Chengzhuo
School of Mathematical Science, Peking University

January 9, 2018

1 The Linear Programming Problem

The standard linear programming problem(LP) is usually written in the following formation,

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \succeq 0, \end{aligned} \tag{1}$$

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

The dual of the above problem can be written as below,

$$\begin{aligned} \text{minimize} \quad & -b^T y \\ \text{subject to} \quad & A^T y + s = c, \\ & s \succeq 0, \end{aligned} \tag{2}$$

2 Augmented Lagrangian Method for solving the dual problem

2.1 Augmented Lagrangian Method

We first write down the augmented Lagrangian function of the dual problem,

$$\mathcal{L}_t(y, s; x) = -b^T y + \mathbf{1}_{\{s\succeq 0\}} + \frac{t}{2}||A^T y + s - c + x||_2^2. \tag{3}$$

To solve the problem with the augmented Lagrangian method, we implement the following three steps repeatedly until some stopping criterion is satisfied,

- Step 1: Calculate (y^{k+1}, s^{k+1}) from the equation $(y^{k+1}, s^{k+1}) = \text{argmin}_{(y,s)}\mathcal{L}_t(y, s; x^k)$,
- Step 2: Set $x^{k+1} = x^k + \gamma(A^T y^{k+1} + s^{k+1} - c)$, where γ is a prefixed scalar,
- Step 3: Set $t = \rho t$, where $\rho \geq 1$ is a prefixed scalar.

Since the second step is trivial in implementation, the main difficulty lies in step 1. To find the minimum point of $\mathcal{L}_t(y, s; x)$, we first fix y and minimize the object function with respect to s, which is equivalent to,

$$\begin{aligned} \text{minimize} \quad & ||A^T y + s - c + x^k||_2^2. \\ \text{subject to} \quad & s \succeq 0 \end{aligned} \tag{4}$$

The solution of the above problem can be derived explicitly,

$$s^*(y) = (c - A^T y - x^k)_+ \tag{5}$$

where $(\cdot)_+$ is the ReLU function $(x)_+ = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$, which is applied elementwisely to each component. After we find s^* given a fixed y, we apply its expression (5) to the original augmented Lagrangian function (3), which then becomes a univariate function of y. Then the value of y^{k+1} can be determined by the following equation,

$$\begin{aligned} y^{k+1} &= \text{argmin}_y \, -b^T y + \frac{t}{2}||(c - A^T y - x^k)_+ - (c - A^T y - x^k)||_2^2 \\ &= \text{argmin}_y \, -b^T y + \frac{t}{2}||(A^T y + x^k - c)_+||_2^2 \\ &\triangleq \text{argmin}_y \varphi(y). \end{aligned} \tag{6}$$

We will use the gradient methods and the semi-smooth Newton method to solve the above optimization problem.

2.2 Gradient Methods

We apply both the original gradient method and the fast gradient method to the problem proposed in the previous section.

2.2.1 The Original Gradient Method

We first derive the gradient of $\varphi(y)$, which can be written as,

$$g \triangleq \nabla\varphi(y) = -b + tA(A^T y + x^k - c)_+ \tag{7}$$

Then the pseudocode of the gradient method is written as below, where we also implement the back-tracking method to determine the stepsize,

Algorithm 1: The gradient method for the inner loop of the dual ALM

Input: A, b, x, t, the initial point y^0, the maximum number of iteration n, the initial stepsize α_0, backtracking parameters β, δ
Output: the optimal point y^*
$k = 0$
while $k \leq n$ *and stopping criterion not satisfied* **do**
  $k = k + 1$
  $g = -b + tA(A^T y^{k-1} + x - c)_+$
  $\alpha = \alpha_0$
  while $\varphi(y - \alpha g) > \varphi(y) - \beta\alpha\|g\|_2^2$ **do**
    $\alpha = \delta\alpha$
  end
  $y^k = y^{k-1} - \alpha g$
end
$y^* = y^k$

In practice, the stopping criterion is related to the norm of the gradient, when the norm of gradient is small than some threshold, the inner loop is ended automatically.

2.2.2 Fast Gradient Method

To accelerate the inner loop, we also apply the fast gradient method, whose pseudocode is written as below,

Algorithm 2: The fast gradient method for the inner loop of the dual ALM

Input: A, b, x, t, the initial point y^0, the maximum number of iteration n, the stepsize α
Output: the optimal point y^*
$y^{-1} = y^0$
$k = 0$
while $k \leq n$ *and stopping criterion not satisfied* **do**
  $k = k + 1$
  $z = y^{k-1} + \frac{k-2}{k+1}(y^{k-1} - y^{k-2})$
  $g = -b + tA(A^T z + x - c)_+$
  $y^k = t - \alpha g$
end
$y^* = y^k$

2.3 Semi-smooth Newton Method

To make our code consistent with [2], we apply an alternate form of the dual problem, which is equivalent to our previous result if we take $A' = -A, b' = -b, c' = -c$.

$$\begin{aligned} \min \quad & b^T y \\ \text{subject to} \quad & A^T y - c \succeq 0, \end{aligned} \tag{8}$$

And we can write the augmented Lagrangian function as following,

$$L_\sigma(y, x) = b^T y + \frac{1}{2\sigma}(\|(x - \sigma(A^T y - c))_+\|^2 - \|x\|^2)$$

Under this setting, the update procedure is performed in the following steps,

- Step 1: Calculate y^{k+1} from the equation $y^{k+1} = \text{argmin}_y L_{\sigma_k}(y, x^k)$,
- Step 2: Set $x^{k+1} = (x^k - \sigma_k(A^T y^{k+1} - c))_+$,
- Step 3: Set $\sigma_{k+1} = \rho\sigma_k$, where $\rho \geq 1$ is a prefixed scalar.

The gradient and the the generalized Hessian matrix with respect y can be calculated as below

$$\begin{aligned} \nabla_y L_\sigma(y, x) &= b - A(x - \sigma(A^T y - c))_+ \triangleq g(y) \\ \nabla_y^2 L_\sigma(y, x) &= \sigma A \left(\mathbf{diag}\{\mathbf{sgn}(A^T y + x^k - c)\}\right) A^T \triangleq V(y) \end{aligned} \tag{9}$$

where $\mathbf{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$, is applied elementwisely to x. In step 1, the descent direction $d(y)$ is calculated by solving the equation $(V(y) + \varepsilon I)d(y) = -g(y)$, which is accomplished by calling the CG algorithm. The stepsize is then determined by the backtracking procedure. The semi-smooth Newton method can be written in the following form[2],

---

Algorithm 3: The semi-smooth Newton method for the inner loop of the dual ALM

Input: $\mu \in (0, 1/2)$, $\bar{\eta} \in (0, 1)$, $\tau \in (0, 1]$,$\tau_1, \tau_2 \in (0, 1)$, and $\delta \in (0, 1)$. Perform the jth iteration as follows.

Step 1. Given a maximum number of CG iterations $N_j > 0$, compute

$$\eta_j := \min(\bar{\eta}, \|\nabla\varphi(y^j)\|^{1+\tau}).$$

Apply the conjugate gradient (CG) algorithm ($CG(\eta_j, N_j)$) to find an approximation solution d_j to

$$(V_j + \varepsilon_j I)\, d = -\nabla\varphi(y^j), \tag{10}$$

where $V_j = \nabla^2\varphi(y^j)$ and $\varepsilon_j := \tau_1 \min\{\tau_2, \|\nabla\varphi(y^j)\|\}$.

Step 2. Set $\alpha_j = \delta^{m_j}$, where m_j is the first nonnegative integer m for which

$$\varphi(y^j + \delta^m d_j) \leq \varphi(y^j) + \mu\delta^m \langle \nabla\varphi(y^j), d_j \rangle. \tag{11}$$

Step 3. Set $y^{j+1} = y^j + \alpha_j\, d_j$.

In the reference paper, the loop is continued until one of the following conditions is satisfied,

$$\begin{aligned} &\text{(A). } \varphi_k(y^{k+1}) - \inf\varphi_k \leq \frac{\epsilon_k^2}{2\sigma_k}, \epsilon_k \geq 0, \sum_{k=0}^{\infty} \epsilon_k < \infty \\ &\text{(B). } \varphi_k(y^{k+1}) - \inf\varphi_k \leq \frac{\delta_k^2}{2\sigma_k}\|X^{k+1} - X^k\|^2, \delta_k \geq 0, \sum_{k=0}^{\infty} \delta_k < \infty \\ &\text{(B'). } \|\nabla\varphi_k(y^{k+1})\| \leq \frac{\delta_k'}{\sigma_k}\|X^{k+1} - X^k\|, 0 \leq \delta_k' \to 0 \end{aligned} \tag{12}$$

where $\epsilon_k, \delta_k, \delta_k'$ are all prefixed sequences and $\varphi_k(y) = L_{\sigma_k}(y, x^k)$. To make the tuning process less tricky, we apply a rather simpler criterion,

$$
\begin{aligned}
&\text{(A). } \frac{\|Ax^k - b\|}{1 + \|b\|} < e_1 \\
&\text{(B). } \frac{\| - A^T y^k + s^k + c\|}{1 + \|c\|} < e_2 \\
&\text{(C). } \frac{|c^T x^k - b^T y^k|}{1 + |c^T x^k| + |b^T y^k|} < e_3
\end{aligned} \tag{13}
$$

where $s^k = -(-\frac{x^k}{\sigma_k} + A^T y^k - c)_+$, e_1, e_2, e_3 are three prefixed scalars.

In practice, however, we find that direct application of CG method causes the problems of imprecise solution and high computation cost. In order to accelerate computation as well as improving the quality of the solution, we apply the following two modifications,

2.3.1 Cut the matrix size

We notice that it is a common case that $D(y) \triangleq \mathbf{diag}\{\mathbf{sgn}(A^T y + x^k - c)\}$ has several diagonal components whose values are zero. Without loss of generality, we assume that the non-zero components are the first k components, and denote the first k columns of A as A_1 and the last $n-k$ columns of A as A_2. Then the expression of $V(y)$ can be rewritten as

$$
V(y) = \sigma A D(y) A^T = \sigma \begin{pmatrix} A_1 & A_2 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} A_1^T \\ A_2^T \end{pmatrix} = \sigma A_1 D_k A_1^T
$$

Therefore, to compute $V(y)$, we need only to select the columns whose corresponding value in D is not zero and then use the sub-matrix composed of these columns to calculate the value of $D(y)$.

2.3.2 Incomplete Cholesky actorization

We find that direct application of CG can often be slow and fail to get a precise solution to the equation $Ad = -\nabla\varphi(y) \triangleq -g$. To solve the problem, we instead use the preconditioned conjugate gradients method. We first transform the problem into an equivalent one of solving the following equation,

$$
M^{-1} A (M^T)^{-1} M^T d = -M^{-1} g
$$

where M is an invertible matrix. We need to choose an appropriate form of M to make $M^{-1}A(M^T)^{-1}$ as close to the identity matrix as possible. To achieve this aim, we use the Incomplete Cholesky factorization method, which decomposes A as

$$
A \approx LL^T \tag{14}
$$

where L is a lower triangle matrix. We then set $M = L$, and apply the CG method. We find a significant improvement in the performance of our algorithm after such change.

3 Semi-smooth Newton method based on solving a fixed-point equation

3.1 Douglas-Rachford splitting(DRS) algorithm for the primal problem

In order to use the standard DRS algorithm, we adjust the (1) to the following formulation,

$$\min \quad c^T x + \mathbf{1}_{\{x \succeq 0\}} + \mathbf{1}_{\{Ax=b\}} \tag{15}$$

We denote $g(x) = c^T x + \mathbf{1}_{\{x \succeq 0\}}$ and $h(x) = \mathbf{1}_{\{Ax=b\}}$, then the updating procedure can be written in the following form,

$$\begin{aligned} x^{k+1} &= \text{prox}_{th}(z^k) \\ u^{k+1} &= \text{prox}_{tg}(2x^{k+1} - z^k) \\ z^{k+1} &= z^k + u^{k+1} - x^{k+1} \end{aligned} \tag{16}$$

There also exists another formation where we update u at first,

$$\begin{aligned} u^{k+1} &= \text{prox}_{tg}(2x^k - z^k) \\ z^{k+1} &= z^k + u^{k+1} - x^k \\ x^{k+1} &= \text{prox}_{th}(z^{k+1}) \end{aligned} \tag{17}$$

Denoting $w^k = z^k - x^k$, we get an alternate form of the DR iteration,

$$\begin{aligned} u^{k+1} &= \text{prox}_{tg}(x^k - w^k) \\ x^{k+1} &= \text{prox}_{th}(u^{k+1} + w^k) \\ w^{k+1} &= w^k + u^{k+1} - x^{k+1} \end{aligned} \tag{18}$$

which can be written explicitly in our problem,

$$\begin{aligned} u^{k+1} &= (x^k - w^k - tc)_+ \\ x^{k+1} &= (I - A^T(AA^T)^{-1}A)(u^{k+1} + w^k) + A^T(AA^T)^{-1}b \\ w^{k+1} &= w^k + u^{k+1} - x^{k+1} \end{aligned} \tag{19}$$

3.2 ADMM for solving the dual problem

The updating procedure for the ADMM method can be written as

$$\begin{aligned} s^{k+1} &= \text{argmin}_s L_t(y^k, s; x^k) \\ y^{k+1} &= \text{argmin}_y \mathcal{L}_t(y, s^{k+1}; x^k) \\ x^{k+1} &= x^k + t(A^T y^{k+1} + s^{k+1} - c) \end{aligned} \tag{20}$$

which can be written explicitly,

$$\begin{aligned} s^{k+1} &= (c - \frac{x^k}{t} - A^T y^k)_+ \\ y^{k+1} &= (tAA^T)^{-1}(tA(c - s^{k+1} - \frac{x^k}{t}) + b) \\ x^{k+1} &= x^k + t(A^T y^{k+1} + s^{k+1} - c) \end{aligned} \tag{21}$$

3.3 The relationship between the variables of DRS and ADMM

We will show that (18) is equivalent to (20) in the LP if some conditions are satisfied. Firstly, we show the update rules are equivalent. We will prove a more general case,

$$\begin{aligned} \min_{x_1,x_2} \quad & f_1(x_1) + f_2(x_2) \\ s.t. \quad & A_1x_1 + A_2x_2 = c \end{aligned} \tag{22}$$

It is easy to derive the dual problem,

$$\min_{z} \quad \underbrace{c^Tz + f_1^*(-A_1^Tz)}_{g(z)} + \underbrace{f_2^*(-A_2^Tz)}_{h(z)} \tag{23}$$

For the dual problem, the alternate form of DR iteration is

$$\begin{aligned} &\text{Step 1: } y^+ = \text{prox}_{tg}(z-w) \\ &\text{Step 2: } z^+ = \text{prox}_{th}(y^+ + w) \\ &\text{Step 3: } w^+ = w + y^+ - z^+ \end{aligned} \tag{24}$$

In Step 1, we have

$$\begin{aligned} y^+ &= \text{prox}_{tg}(z-w) \\ &= \text{argmin}_y g(y) + \frac{1}{2t}\|y-(z-w)\|_2^2 \\ &= \text{argmin}_y c^Ty + f_1^*(-A_1^Ty) + \frac{1}{2t}\|y-(z-w)\|_2^2 \end{aligned} \tag{25}$$

Taking derivative with respect to y in the right hand side of the above equation and setting the gradient to zero, we get

$$y^+ - z + w + tc - tA_1\nabla f_1^*(-A_1^Ty^+) = 0 \tag{26}$$

We take $\hat{x}_1 \in \nabla f_1^*(-A_1^Ty^+)$ and further assume f_1 is closed and convex, then $-A_1^Ty^+ \in \nabla f_1(\hat{x}_1)$, which yields

$$\nabla f_1(\hat{x}_1) + A_1^Tz + tA_1^T(A_1\hat{x}_1 - c - w/t) = 0 \tag{27}$$

The above equation means that

$$\begin{aligned} \hat{x}_1 &= \arg\min_{x_1} f_1(x_1) + z^T(A_1x_1 + A_2x_2 - c) + \frac{t}{2}\|A_1x_1 - c - w/t\|_2^2 \\ y^+ &= z - w + t(A_1\hat{x_1} - c) \end{aligned} \tag{28}$$

In Step 2, similarly, we compute

$$
\begin{aligned}
z^+ &= \mathrm{prox}_{th}(y^+ + w) \\
&= \mathrm{prox}_{th}(z + t(A_1\hat{x_1} - c)) \\
&= \mathrm{argmin}_{z^*} h(z^*) + \frac{1}{2t}\|z^* - (z + t(A_1\hat{x_1} - c))\|_2^2 \\
&= \mathrm{argmin}_{z^*} f_2^*(-A_2^T z^*) + \frac{1}{2t}\|z^* - (z + t(A_1\hat{x_1} - c))\|_2^2
\end{aligned} \tag{29}
$$

Again we set the gradient of the right hand side of the above equation to be zero, then we get

$$
z^+ - z - t(A_1\hat{x_1} - c) - tA_2\nabla f_2^*(-A_2^T z^+) = 0 \tag{30}
$$

We take $\hat{x}_2 \in \nabla f_2^*(-A_2^T z^+)$ and also assume that f_2 is closed and convex, then we get $-A_2^T z^+ \in \nabla f_2(\hat{x}_2)$, which yields

$$
\nabla f_2(\hat{x}_2) + A_2^T z + tA_2^T(A_1\hat{x}_1 + A_2\hat{x}_2 - c) = 0 \tag{31}
$$

The above equation means that

$$
\begin{aligned}
\hat{x}_2 &= \arg\min_{x_2} f_2(x_2) + z^T(A_1\hat{x}_1 + A_2x_2 - c) + \frac{t}{2}\|A_1\hat{x}_1 + A_2x_2 - c\|_2^2 \\
z^+ &= z + t(A_1\hat{x}_1 + A_2\hat{x}_2 - c)
\end{aligned} \tag{32}
$$

In Step 3, combining the results in Step 1 and Step 2, the updating rule reduces to

$$
w^+ = w + y^+ - z^+ = w + (z - w + t(A_1\hat{x}_1 - c)) - (z + t(A_1\hat{x}_1 + A_2x_2 - c)) = -tA_2\hat{x}_2 \tag{33}
$$

Combining the results of the three steps, we get the updating rule as following,

$$
\begin{aligned}
\hat{x}_1 &= \arg\min_{x_1} f_1(x_1) + z^T(A_1x_1 + A_2x_2 - c) + \frac{t}{2}\|A_1x_1 - c - w/t\|_2^2 \\
\hat{x}_2 &= \arg\min_{x_2} f_2(x_2) + z^T(A_1\hat{x}_1 + A_2x_2 - c) + \frac{t}{2}\|A_1\hat{x}_1 + A_2x_2 - c\|_2^2 \\
z^+ &= z + t(A_1\hat{x}_1 + A_2\hat{x}_2 - c) \\
w^+ &= -tA_2\hat{x}_2
\end{aligned} \tag{34}
$$

which is actually the form of ADMM for (22). Now we apply the above results to the linear programming problem, where

$$
\begin{aligned}
&f_1(x_1) = \mathbf{1}_{\{x_1 \succeq 0\}} \\
&f_2(x_2) = -b^T x_2 \\
&A_1 = I \ , \ A_2 = A^T \\
&x_1 = s \ , \ x_2 = y \ , \ z = x
\end{aligned} \tag{35}
$$

Then the equivalence between DRS for dual and ADMM for primal can be proved as a special case of the above result.

3.4 Regularized Semi-smooth Newton Method

We first review the DRS iteration,

$$\begin{aligned} x^{k+1} &= \text{prox}_{th}(z^k) \\ u^{k+1} &= \text{prox}_{tg}(2x^{k+1} - z^k) \\ z^{k+1} &= z^k + u^{k+1} - x^{k+1} \end{aligned} \tag{36}$$

which is equivalent to the fixed-point iteration

$$z^{k+1} = T_{\text{DRS}}(z^k)$$

where $T_{\text{DRS}} = I + \text{prox}_{tg}(2\text{prox}_{th} - I) - \text{prox}_{th}$. It can be derived that $F = I - T_{\text{DRS}}$ is strongly semi-smooth and monotone with a positive semidefinite generalized Jacobi matrix J, according to [1] in section 3.2, so we can apply the regularized semi-smooth Newton method to approximate the solution

$$F(z) = 0 \tag{37}$$

The calculation result shows that when we set $h(x) = c^T x + \mathbf{1}_{\{x \succeq 0\}}$ and $g(x) = \mathbf{1}_{\{Ax=b\}}$, the result will have a simpler form than the case where $g(x) = c^T x + \mathbf{1}_{\{x \succeq 0\}}$ and $h(x) = \mathbf{1}_{\{Ax=b\}}$. In the former case, $F(z)$ has the following formation,

$$\begin{aligned} F(z) &= \text{prox}_{th}(z) - \text{prox}_{tg}(2\text{prox}_{th}(z) - z) \\ &= (z - tc)_+ - (I - A^T(AA^T)^{-1}A)(2(z - tc)_+ - z) - A^T(AA^T)^{-1}b \\ &= U(z - tc)_+ + Vz - S \end{aligned} \tag{38}$$

where $U = 2A^T(AA^T)^{-1}A - I, V = I - A^T(AA^T)^{-1}A, S = -A^T(AA^T)^{-1}b$The Jacobi matrix of $F(z)$, denoted by J, is

$$J(z) = UM + V \tag{39}$$

where $M = \mathbf{diag}(\mathbf{sgn}(z - tc))$. The overall procedure of the algorithm is shown below,

Algorithm 4: The regularized semi-smooth Newton method

Input: $\lambda_0 > 0$, $0 < \eta_1 < \eta_2 < 1$, $1 < \gamma_1 < \gamma_2$ and $0 < \tau < 1$. Perform the kth iteration as follows.

Step 1. solve the below equation for a descent direction d^k by regularized Newton method.

$$(J_k + \lambda_k ||F(z^k)||_2 I)d = -F(z^k),$$

with tolerance $r^k = (J_k + \lambda_k ||F(z^k)||_2 I)d^k + F(z^k) \leq \tau \min\{1, \lambda_k \|F(z^k)\|_2 \|d^k\|_2\}$

Step 2. then a trial point is obtained as

$$u^k = z^k + d^k,$$

and calculate the ratio

$$\rho_k = \frac{-\langle F(u^k), d^k \rangle}{\|d^k\|_2^2}$$

Step 3. update z according to the follow form

$$z^{k+1} = \begin{cases} u^k, \text{ if } \rho_k \geq \eta_1 \text{ and } \|F(u^k)\|_2 \leq \tau \|F(\bar{u})\|_2 & \text{[Newton step]} \\ v^k, \text{ if } \rho_k \geq \eta_1 \text{ and } \|F(u^k)\|_2 \geq \tau \|F(\bar{u})\|_2 & \text{[projection step]} \\ z^k, \text{ otherwise} & \text{[unsuccessful iteration]} \end{cases} \tag{40}$$

where $v^k = z^k - \frac{\langle F(u^k), z^k - u^k \rangle}{\|F(u^k)\|_2^2} F(u^k)$.

and the reference point $\bar{u}$ is the iteration from the last Newton step.More specially, when $\rho_k \geq \eta_1$ and $\|F(u^k)\|_2 \leq \tau \|F(\bar{u})\|_2$, we take $z^{k+1} = u^k$ and update $\bar{u} = u^k$.

Step 4. update the regularization parameter λ

$$\lambda_{k+1} \in \begin{cases} (\lambda_0, \lambda_k), & \text{if } \rho_k \geq \eta_2 \\ [\lambda_k, \gamma_1 \lambda_k], & \text{if } \eta_1 \leq \rho_k \leq \eta_2 \\ (\gamma_1 \lambda_k, \gamma_2 \lambda_k], & \text{otherwise} \end{cases} \tag{41}$$

4 Numerical Results

We first implement algorithms mentioned above on the synthetic data. Due to the low dimension of the data, all algorithms do a satisfactory performance. Then we attempt to test the algorithm performance in the Netlib datasets. But the result is quite undesirable. As a less-than-ideal alternative, we focus on finetuning one algorithm (here we choose the ALM newton method but with some modifications) to work in as many as possible datasets in Netlib.

4.1 Synthetic Data

The data is generated using the following matlab code:

```
rng(5)
n = 100;
m = 20;
A = rand(m, n);
xs = full(abs(sprandn(n, 1, m / n)));
b = A * xs;
y = randn(m, 1);
s = rand(n, 1) .* (xs == 0);
c = A' * y + s;
```

We can find that primal variable x_s (denoted by xs in codes above), dual variables y and s has already

satisfied the following KKT conditions (42). In other words, we can safely conclude that x_s is the true solution to the original problem, which could be used to measure the absolute accuracy of different algorithms.

$$\begin{aligned} \text{primal constraints: } & Ax_s = b \ , \ x_s \geq 0 \\ \text{dual constraints: } & A^T y + s = c \ , \ s \geq 0 \\ \text{complementary slackness: } & s^T x_s = 0 \end{aligned} \tag{42}$$

In table (1), we list the results of all of our algorithms. The cvx-gurobi method is the most accurate method, and alm-fgrad is the fastest.

	objective value	time(s)	err to true sol	$\|\|Ax-b\|\|_2$	iter(outer)
cvx-mosek	-24.736639679	0.62	3.16e-15	1.28e-14	
cvx-gurobi	-24.736639679	0.22	2.22e-15	6.34e-15	
mosek	-24.736639689	0.46	1.14e-07	6.68e-07	
gurobi	-24.736639679	0.09	2.59e-15	6.22e-15	
alm	-24.736633679	1.71	6.74e-06	7.48e-06	500
alm-fgrad	-24.736639692	0.07	8.03e-09	6.02e-09	128
alm-newton	-24.736639868	0.09	9.48e-09	9.48e-08	2
admm	-24.736639679	0.25	7.85e-12	1.69e-11	11788
drs	-24.736639682	0.12	7.75e-09	9.04e-09	10000
drs-newton	-24.736639679	0.09	4.42e-09	1.06e-08	215

Table 1: Perfomance of different algorithms in the synthetic data. We compare the accuracy, efficiency, primal feasibility of these methods. Due to the data generating rule, the true optimal solution is x_s and the error to true solution is calculated by $\frac{\|x-x_s\|_2}{\max(1,\|x_s\|_2)}$.

4.2 Netlib Test Problems

Note that the problems in the Netlib may not be in the standard form, which are usually in the form

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & b_l \preceq Ax \preceq b_u, \\ & t_l \preceq x \preceq t_u, \end{aligned} \tag{43}$$

However, this problem can be transformed into the standard form with the following procedure. We first add the relax variables s_1, s_2, s_3, s_4, after which the problem becomes

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax + s_1 = b_u, \\ & Ax - s_2 = b_l \\ & x + s_3 = t_l \\ & x - s_4 = t_u, \\ & s_1, s_2, s_3, s_4 \succeq 0 \end{aligned} \tag{44}$$

then we rewrite $x = x_1 - x_2$, where x_1, x_2 are all elementwisely bigger than zero,

$$
\begin{aligned}
\text{minimize} \quad & c^T x_1 - c^T x_2 \\
\text{subject to} \quad & A(x_1 - x_2) + s_1 = b_u, \\
& A(x_1 - x_2) - s_2 = b_l \\
& x_1 - x_2 + s_3 = t_u \\
& x_1 - x_2 - s_4 = t_l, \\
& x_1, x_2, s_1, s_2, s_3, s_4 \succeq 0
\end{aligned}
\tag{45}
$$

Therefore, the standard form is

$$
\begin{aligned}
\text{minimize} \quad & \begin{pmatrix} c^T & -c^T & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \\
\text{subject to} \quad & \begin{pmatrix} A & -A & I_m & 0 & 0 & 0 \\ A & -A & 0 & -I_m & 0 & 0 \\ I_n & -I_n & 0 & 0 & I_n & 0 \\ I_n & -I_n & 0 & 0 & 0 & -I_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} b_u \\ b_l \\ t_u \\ t_l \end{pmatrix} \\
& x_1, x_2, s_1, s_2, s_3, s_4 \succeq 0
\end{aligned}
\tag{46}
$$

The algorithm results see the Table (2) and corresponding parameters see the Table (3).

References

[1] Xiantao Xiao, Yongfeng Li, Zaiwen Wen, and Liwei Zhang. A regularized semi-smooth newton method with projection steps for composite convex programs. *arXiv preprint arXiv:1603.07870*, 2016.

[2] Xin-Yuan Zhao, Defeng Sun, and Kim-Chuan Toh. A newton-cg augmented lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, 20(4):1737–1765, 2010.

dataset	m	n	objective ratio	time(s)	err to cvx-gurobi	pfeasibility	dfeasibility	iter(outer)
sebapre	19	36	-3.17e-11	0.02	5.20e-11	4.19e-09	5.57e-12	2
sc50bpre	34	90	4.44e-15	0.01	4.40e-15	5.87e-12	4.10e-16	2
sc50apre	31	92	-1.48e-13	0.01	1.03e-13	6.51e-10	3.96e-13	4
afiropre	29	78	-8.88e-16	0.01	1.96e-02	2.75e-12	1.15e-15	4
sc105pre	61	182	1.11e-15	0.04	9.97e-16	1.36e-12	2.78e-16	13
beaconfdpre	70	188	-7.77e-16	0.18	3.44e-01	1.62e-11	5.40e-15	20
vtp_basepre	168	324	-1.31e-12	0.68	3.52e-01	1.23e-07	3.70e-12	4
sc205pre	124	362	1.83e-12	1.74	1.80e-12	1.02e-07	4.82e-11	170
recipepre	177	380	3.77e-15	0.22	1.88e-01	3.79e-12	1.73e-15	15
scagr7pre	247	532	-3.48e-08	7.85	1.41e-01	8.56e-08	5.59e-04	40
kb2pre	76	194	-3.93e-06	17.64	6.04e-01	9.60e-09	1.15e-04	1001
scorpionpre	172	446	-2.05e-07	1.23	1.17e+00	1.39e-06	1.09e-04	5
stocfor1pre	122	324	4.32e-12	0.36	9.57e-01	4.77e-06	3.16e-09	3
adlittlepre	162	482	4.20e-11	0.37	1.49e+00	7.10e-08	2.65e-12	3
share2bpre	188	502	-2.54e-12	2.83	2.50e-01	1.74e-07	9.30e-13	12
lotfipre	480	1362	1.91e-09	37.40	2.77e-01	3.51e-04	1.28e-12	44
standatapre	677	1652	5.25e-12	19.62	4.92e+00	1.16e-05	7.74e-09	10
standgubpre	677	1652	-4.91e-12	27.77	3.18e+01	1.10e-06	7.92e-10	11
sctap1pre	608	1894	-1.79e-11	18.43	3.87e+00	1.74e-06	1.34e-09	12
bore3dpre	142	320	-1.52e-07	77.04	6.63e-01	5.55e-06	1.12e-09	678
scagr25pre	949	2044	-8.15e-13	231.54	1.13e-01	1.78e-07	3.46e-12	49
boeing2pre	374	884	-1.02e-11	186.65	3.49e-01	4.56e-06	1.01e-12	320
scsd1pre	914	3194	-2.99e-13	29.50	9.75e-01	3.17e-10	5.31e-11	5
aggpre	313	756	-2.89e-15	39.81	9.57e-01	1.40e-04	2.14e-13	148
brandypre	360	900	-1.11e-16	6.50	1.96e-03	8.56e-09	5.17e-13	17
standmpspre	1481	4232	-8.76e-13	764.85	2.15e+00	2.32e-06	2.32e-08	140
degen2pre	1077	2656	2.83e-12	140.54	1.38e+02	5.60e-06	3.75e-09	14
finnispre	716	2038	2.88e-09	90.27	4.58e+00	9.89e-08	1.02e-05	59
share1bpre	357	962	-1.92e-03	89.53	6.05e-02	1.82e+04	1.18e-02	101
gangespre*	1715	3406	-2.16e-14	328.96	8.59e-02	4.45e-09	4.27e-13	9
finnispre*	716	2038	1.71e-08	27.06	8.34e-01	2.12e-06	1.15e-05	10
capripre*	487	1096	1.11e-12	93.54	4.23e-02	8.46e-08	2.23e-13	38
degen2pre*	1077	2656	8.00e-13	190.22	2.13e+02	3.51e-09	3.76e-05	9

Table 2: Our algorithm performance on Netlib datasets. The obective ratio is the relative objective error, calculated by $\frac{\text{our objective}}{\text{cvx-gurobi objective}} - 1$, whose logarithm is a rough approximation of significant digits. Denoting the cvx-gurobi solution as xg and our algorithm solution as x, we define the relative err of x to x_g as $\frac{\|x-x_g\|_2}{\max(1,\|x_g\|_2)}$. The pfeasibility and dfeasibility respectively measure the feasibility condition for the primal problem and the dual problem. The pfeasibility is $\|Ax-b\|_2$ and the dfeasibility is $\|A^Ty-s-c\|_2$.

dataset	sig	iters	inner_iters	normg
default	1,500	1,000	50	1e-8
default $*$	10,000	100	100	1e-7
lotfipre	1000,000	1,000	50	1e-8
share1bpre	1500,000	100	50	1e-7
standmpspre	10,000	200	50	1e-8

Table 3: Parameters for Netlib problems. In Table (2), the default parameters should be used unless otherwise specified. Problems with $*$ use the default $*$ parameters. The rest problems other than the default and the default $*$ use above specified parameters.