Interior Point Method

1st ZHANG Longwen
ShanghaiTech
Shanghai, China
zhanglw2@shanghaitech.edu.cn 2018533113

***Abstract*—In this project, I provided an implementation for Interior Point Method (IPM), which is fast, robust, and bug-free.**

***Index Terms*—Interior Point Method, Linear Programming, Constraint Optimization**

I. Introduction

Interior Point Method focuses on solving the primal-dual problem with relaxation on complementary conditions.

II. Theoretical Analysis

A. Optimal condition

First, we consider the pure Interior Point Method. The standard form of linear programming could be written as

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{1}$$

, which is the primal problem. And the dual problem is

$$\begin{aligned} \max \quad & \boldsymbol{b}^T\boldsymbol{\lambda} \\ \text{s.t.} \quad & \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{s} = \boldsymbol{c} \\ & \boldsymbol{s} \geq \boldsymbol{0} \end{aligned} \tag{2}$$

Through Strong Duality theorem and Complementary condition, the optimal solution satisfies the following condition

$$\begin{cases} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{s} = \boldsymbol{c} \\ \boldsymbol{x} \geq \boldsymbol{0} \\ \boldsymbol{s} \geq \boldsymbol{0} \\ x_i s_i = 0, \forall i \end{cases}$$

Define the residual function be

$$F(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \begin{bmatrix} \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{s} - \boldsymbol{c} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{X}\boldsymbol{S}\boldsymbol{1} \end{bmatrix}$$

where $\boldsymbol{X}$ is the diagonal form of $\boldsymbol{x}$ and $\boldsymbol{S}$ is the diagonal form of $\boldsymbol{s}$. The optimal solution is achieved when

$$F(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{s}^*) = \boldsymbol{0}$$

B. Newton Iterative Method

For

$$F(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \boldsymbol{0}$$

do first order expansion and get

$$F(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) + \nabla F(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s})^T \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{s} \end{bmatrix} = \boldsymbol{0}$$

, which means

$$\begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{s} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{s} + \boldsymbol{c} \\ -\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \\ -\boldsymbol{X}\boldsymbol{S}\boldsymbol{1} \end{bmatrix}$$

During the iterative update process, we solve

$$\begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}_k & \boldsymbol{0} & \boldsymbol{X}_k \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_k \\ \Delta\boldsymbol{\lambda}_k \\ \Delta\boldsymbol{s}_k \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{s}_k + \boldsymbol{c} \\ -\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b} \\ -\boldsymbol{X}_k\boldsymbol{S}_k\boldsymbol{1} + \tau_k\boldsymbol{1} \end{bmatrix} \tag{3}$$

, where τ_k is the relaxation over iterations. This item is proposed to release complementary constraint to some extent, in which case the residual on the primal and dual problem could reduce fast.

C. Big M

The Big-M form of the standard form of linear programming could be written as

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} + M\boldsymbol{e}^T\boldsymbol{y} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{aligned} \tag{4}$$

, where M is a sufficent large value. The quivalent form is

$$\begin{aligned} \min \quad & \begin{bmatrix} \boldsymbol{c}^T & M\boldsymbol{e}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \\ \text{s.t.} \quad & \begin{bmatrix} \boldsymbol{A} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{aligned} \tag{5}$$

, which is in the same form of the original standard form. In the optimal solution, $\boldsymbol{y}$ is zero since any positive value in $\boldsymbol{y}$ results increases the target function value, as long as it is not sufficent small. When M is selected appropriately, the number of iterations could be reduced. Furthermore, in Interior Point Method, if the convergent solution of a problem results in not sufficent small $\boldsymbol{y}$, it shows the infeasibility in the original problem.

III. Implementation Detail

A. Assumption

In the previous analysis, we assume $\boldsymbol{A}$ has full row rank. But it could have linearly dependent rows in some tricky problems, which brings lots of trouble in implementation, and almost destroys everything we made. So before we start, we will remove the linearly dependent rows at first.

First, $\boldsymbol{A}^T$ is made to be a row echelon form, where the leading 1 of every line indicates a linearly independent column of $\boldsymbol{A}^T$, also a linearly independent row of $\boldsymbol{A}$. So we pick out those indices of linearly independent columns of $\boldsymbol{A}^T$ and select the corresponding rows in $\boldsymbol{A}$ and $\boldsymbol{b}$ to form new $\boldsymbol{A}$ and $\boldsymbol{b}$. We always ensure $\boldsymbol{A}$ has the full row rank before the algorithm starts.

B. Initial Point

First, we find an arbitrary solution $\boldsymbol{x}_0$

$$\boldsymbol{x}_0 = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^\dagger \boldsymbol{b}$$

where

$$\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{A}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^\dagger \boldsymbol{b} = \boldsymbol{b}$$

Then we find the corresponding $\boldsymbol{\lambda}_0, \boldsymbol{s}_0$

$$\boldsymbol{c}^T\boldsymbol{x}_0 = \boldsymbol{c}^T\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^\dagger \boldsymbol{b} = \boldsymbol{\lambda}_0^T\boldsymbol{b}$$

$$\Rightarrow \boldsymbol{\lambda}_0 = (\boldsymbol{A}\boldsymbol{A}^T)^\dagger \boldsymbol{A}\boldsymbol{c}$$

$$\boldsymbol{s}_0 = \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{\lambda}_0$$

Thus here we have initial points that satisfy the primal-dual condition except for the complementary condition and could be negative. The second step is to shift $\boldsymbol{x}_0, \boldsymbol{s}_0$ to be non-negative.

For the convenience of later computations, we had better add a small shift again to $\boldsymbol{x}_0, \boldsymbol{s}_0$ to make them a little bit more away from zeros, in which case we have more space for other residuals to optimize. In my implementation I use

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \frac{\boldsymbol{x}_0^T\boldsymbol{s}_0}{\|\boldsymbol{s}_0\|_1 + \epsilon} + \epsilon$$

$$\boldsymbol{s}_1 = \boldsymbol{s}_0 + \frac{\boldsymbol{x}_0^T\boldsymbol{s}_0}{\|\boldsymbol{x}_0\|_1 + \epsilon} + \epsilon$$

where ϵ is a small number and is set to 10^{-9} in practice. So we could avoid some parts being zero.

C. Equation Solving

Consider the updating equation (3). Let

$$\begin{bmatrix} -\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{s}_k + \boldsymbol{c} \\ -\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b} \\ -\boldsymbol{X}_k\boldsymbol{S}_k\boldsymbol{1} + \tau_k\boldsymbol{1} \end{bmatrix} = - \begin{bmatrix} F_k^{\text{dual}} \\ F_k^{\text{primal}} \\ F_k^0 \end{bmatrix}$$

Equation (3) is a linear system as following:

$$\boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k + \boldsymbol{I}\Delta\boldsymbol{s}_k = -F_k^{\text{dual}} \tag{6}$$

$$\boldsymbol{A}\Delta\boldsymbol{x}_k = -F_k^{\text{primal}} \tag{7}$$

$$\boldsymbol{S}_k\Delta\boldsymbol{x}_k + \boldsymbol{X}_k\Delta\boldsymbol{s}_k = -F_k^0 \tag{8}$$

$\boldsymbol{X}_k(6) \Longrightarrow$

$$\boldsymbol{X}_k\boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k + \boldsymbol{X}_k\Delta\boldsymbol{s}_k = -\boldsymbol{X}_kF_k^{\text{dual}} \tag{9}$$

$(9) - (8) \Longrightarrow$

$$\boldsymbol{X}_k\boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k = -\boldsymbol{X}_kF_k^{\text{dual}} + \boldsymbol{S}_k\Delta\boldsymbol{x}_k + F_k^0 \tag{10}$$

$\boldsymbol{A}\boldsymbol{S}_k^{-1}(10) \Longrightarrow$

$$\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k = -\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_kF_k^{\text{dual}} + \boldsymbol{A}\Delta\boldsymbol{x}_k + \boldsymbol{A}\boldsymbol{S}_k^{-1}F_k^0 \tag{11}$$

$(11) + (7) \Longrightarrow$

$$\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k = -\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_kF_k^{\text{dual}} - F_k^{\text{primal}} + \boldsymbol{A}\boldsymbol{S}_k^{-1}F_k^0 \tag{12}$$

$(6) \Longrightarrow$

$$\Delta\boldsymbol{s}_k = -F_k^{\text{dual}} - \boldsymbol{A}^T\Delta\boldsymbol{\lambda}_k \tag{13}$$

$\boldsymbol{S}_k^{-1}(8) \Longrightarrow$

$$\Delta\boldsymbol{x}_k = -\boldsymbol{S}_k^{-1}F_k^0 - \boldsymbol{S}_k^{-1}\boldsymbol{X}_k\Delta\boldsymbol{s}_k \tag{14}$$

So actually we first use equation (12) to solve $\Delta\boldsymbol{\lambda}_k$, then we use equation (13) and (14) to compute $\Delta\boldsymbol{s}_k, \Delta\boldsymbol{x}_k$ directly.

In my implementation, I keep both $\boldsymbol{x}_k, \boldsymbol{s}_k$ away from zero. This generates huge convenience for calculation. For example, $\boldsymbol{S}_k^{-1}$ always exists, and $\boldsymbol{X}_k\boldsymbol{S}_k^{-1}$ always has full rank. Thus we compute $\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T$ first each time. Note $\boldsymbol{x}_k, \boldsymbol{s}_k > 0$ in the iterations and $\boldsymbol{A}$ has full rank so $\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T$ shall be symmetric positive definite. The second step is to do Cholesky decomposition $\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T = \boldsymbol{L}_k\boldsymbol{L}_k^T$ and then solve the equation (12).

In practice, there is always a chance for $\boldsymbol{X}_k\boldsymbol{S}_k^{-1}$ to be numerically unstable, where $\boldsymbol{A}\boldsymbol{S}_k^{-1}\boldsymbol{X}_k\boldsymbol{A}^T$ is no longer positive definite. In this case, we shall use the Least-Square solution instead. Often in the next iteration, things become normal as usual.

D. Primal-Dual Update

For current $\boldsymbol{x}_k$ and updating direction $\Delta\boldsymbol{x}_k$, the updating step size α_k^{primal} should satisfies

$$\boldsymbol{x}_k + \alpha_k^{\text{primal}}\Delta\boldsymbol{x}_k > \boldsymbol{0}$$

So we choose the step size to be

$$\alpha_k^{\text{primal}} < -\frac{(\boldsymbol{x}_k)_i}{(\Delta\boldsymbol{x}_k)_i}, \forall(\Delta\boldsymbol{x}_k)_i < 0$$

, which means

$$\alpha_k^{\text{primal}} < \min_{i:(\Delta\boldsymbol{x}_k)_i<0} \frac{(\boldsymbol{x}_k)_i}{-(\Delta\boldsymbol{x}_k)_i}$$

In practice, to ensure $\boldsymbol{x}_k$ is away from zero, α_k^{primal} is set to

$$\alpha_k^{\text{primal}} \leftarrow (1-\epsilon)\min\{\min_{i:(\Delta\boldsymbol{x}_k)_i<0} \frac{(\boldsymbol{x}_k)_i}{-(\Delta\boldsymbol{x}_k)_i}, 1\}$$

, where ϵ is a small number and is set to 10^{-9} in practice. This is similar for α_k^{dual}

$$\alpha_k^{\text{dual}} \leftarrow (1-\epsilon)\min\{\min_{i:(\Delta \boldsymbol{s}_k)_i<0} \frac{(\boldsymbol{s}_k)_i}{-(\Delta \boldsymbol{s}_k)_i}, 1\}$$

In my implementation, I solve updating directions in two steps.

First, let $\tau_k = 0$, i.e. solve (3) without any relaxation, to get the strict direction $\Delta \boldsymbol{x}_k^{\text{strict}}, \Delta \boldsymbol{s}_k^{\text{strict}}$ and corresponding step size from previous rules $\alpha_k^{\text{strict-primal}}, \alpha_k^{\text{strict-dual}}$. From these data, then we calculate the relaxation as following (n is the number of variables in the original problem, i.e. the number of columns of $\boldsymbol{A}$)

$$\tau_k = \frac{((\boldsymbol{x}_k + \alpha_k^{\text{strict-primal}} \Delta \boldsymbol{x}_k^{\text{strict}})^T (\boldsymbol{s}_k + \alpha_k^{\text{strict-dual}} \Delta \boldsymbol{s}_k^{\text{strict}}))^2}{n \boldsymbol{x}_k^T \boldsymbol{s}_k}$$

where τ_k approximates the relaxation in the current iteration by the ratio between the dot product of strict updating and the current dot product, since dot product $\boldsymbol{x}_k^T \boldsymbol{s}_k$ appropriates the scale of residual of complementary condition.

Second, use computed τ_k as the relaxation to solve (3) and get the true updating direction $\Delta \boldsymbol{x}_k, \Delta \boldsymbol{s}_k$ and corresponding step size from previous rules $\alpha_k^{\text{primal}}, \alpha_k^{\text{dual}}$. Note the step size from previous rules only ensures $\boldsymbol{x}_k, \boldsymbol{s}_k$ not to be zero.

E. Line Search

In this step, we will ensure two constraints on updating directions.

First, we want the updated $\boldsymbol{x}_k$ and $\boldsymbol{s}_k$ to be loose to some extent, i.e. have a residual not too small. In practice and previous implementations, when $\boldsymbol{X}_k \boldsymbol{S}_k \boldsymbol{1}$ is too close to zero, it is hard for primal residual F_k^{primal} and dual residual F_k^{dual} to decline, since in most cases a super small $\boldsymbol{X}_k \boldsymbol{S}_k \boldsymbol{1}$ restrict the other variables a lot and even a small step will violate the complementary constraint. So we decide to keep the distance. The first constraint the step should satisfy is

$$\|(\boldsymbol{X}_k + \alpha_k^{\text{primal}} \Delta \boldsymbol{X}_k)(\boldsymbol{S}_k + \alpha_k^{\text{dual}} \Delta \boldsymbol{S}_k)\boldsymbol{1}\|_\infty > \gamma \frac{\boldsymbol{x}_k^T \boldsymbol{s}_k}{n}$$

where γ is a decay constant and is set to be 0.9 in practice.

Second, we want residuals to be truly reduced after updating. So the Armijo condition is modified to consider the infinite norm as following

$$\begin{aligned}&\|F(\boldsymbol{x}_k + \alpha_k^{\text{primal}} \Delta \boldsymbol{x}_k, \boldsymbol{\lambda}_k + \alpha_k^{\text{dual}} \Delta \boldsymbol{\lambda}_k, \boldsymbol{s}_k + \alpha_k^{\text{dual}} \Delta \boldsymbol{s}_k)\|_\infty \\ &< \left\| F_k + c_1 \nabla F(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{s}_k)^T \begin{bmatrix} \alpha_k^{\text{primal}} \Delta \boldsymbol{x}_k \\ \alpha_k^{\text{dual}} \Delta \boldsymbol{\lambda}_k \\ \alpha_k^{\text{dual}} \Delta \boldsymbol{s}_k \end{bmatrix} \right\|_\infty\end{aligned}$$

where c_1 is a constant and is set to be 0.0001 in practice.

To satisfy both constraints, we use direct backtracking. Each time we multiply a decay constant γ to both α_k^{primal} and α_k^{dual}. Under infinite norm, both constraints could be satisfied when α_k^{primal} and α_k^{dual} are sufficent small.

F. Terminal conditions

There are several different terminal conditions and status set in my implementation. They are

1) *DONE*: The problem is successfully solved.
2) *HARD*: The problem cannot converge to the optimal solution.
3) *EXPIRED*: The problem runs out of maximum iterations.
4) *UNBOUNDED*: The problem has no optimal solutions.
5) *VIOLATED*: The problem is infeasible.

At any moment, if

$$\|F(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{s}_k)\|_\infty < p$$

the status is set to *DONE*, where p is the terminal precision and is set to 10^{-6} in practice. This means the residual is small enough.

To decide if a problem is infeasible,

1) if we are using Big-M form now, then after we set the status to *DONE*, we check the last m artificial variables in the solution. If they are sufficiently small, such as smaller than p, we consider the problem is truly solved. Otherwise, the problem is infeasible and we set the status to *VIOLATED*.
2) if we are not using Big-M form, then *DONE* means truly solved. If the status is other than *DONE*, we construct a subproblem to solve

$$\begin{aligned} \min \quad & \boldsymbol{0}^T x \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{15}$$

, where we simply want to know if it could converge. If this subproblem could not result in good solving, then the original problem is infeasible and we set the status to *VIOLATED*.

Some cases in the wild will lead to both

$$\alpha_k^{\text{primal}} < \epsilon$$

$$\alpha_k^{\text{dual}} < \epsilon$$

, which means the step is too small and cannot converge to the optimal solution anymore. There is a chance that the iteration goes normally and just be trapped once. So we count the number of times that the step is small. If the number of times is greater than 3, we consider this situation happened. There are two more situations. If $\|F(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{s}_k)\|_\infty$ is relatively small, we consider it hard to converge, which seldom happened. In this case, we set the status to *HARD*. But if $\| \begin{bmatrix} \Delta \boldsymbol{x}_k \\ \Delta \boldsymbol{\lambda}_k \\ \Delta \boldsymbol{s}_k \end{bmatrix} \|_\infty$ is relatively large, we consider it be unbounded since the updating directions are large. In this case, we set the status to *UNBOUNDED*.

Also, *EXPIRED* is prepared for some cases I have never seen before but may exist and show up in the future. The maximum number of iterations is set to 500 in practice.

G. Big M

The M value is difficult to choose. If M is too small, the objective value is incorrect since the artificial variables are not zero. If M is too large, it is hard for iterations to converge, since the scale of iterative variables will grow large as M.

In practice, M is chosen as following

$$M = \max\{\|c\|_\infty \hat{b} m, m\}$$

, where m is the number of constraints, i.e. the number of rows of $\boldsymbol{A}$, and $\hat{b}$ is the mean of element-wise absolute $\boldsymbol{b}$. This considers the largest cost factor, the average scale of $\boldsymbol{b}$ and the number of added artificial variables.

IV. Evaluation and Performance

A. Usage

Here is the help of the PYTHON version of my implementation.

```
> python code/main.py -h
usage: main.py [-h] [--detail] [--solution] [-M] data_folder

positional arguments:
  data_folder  a folder containing `A.csv`, `b.csv` and
               `c.csv` [and `x_star.csv`]

optional arguments:
  -h, --help   show this help message and exit
  --detail     show detailed information each iteration
  --solution   show solution if solved
  -M           use the big-M form
```

For example, to load provided *data1*, run following

```
> python code/main.py data/data1
```

B. Examples

1) For given *Example 1*,

$$\boldsymbol{A} = \begin{bmatrix} 6 & 1 & -2 & -1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 6 & 4 & -2 & 0 & 0 & -1 \end{bmatrix}$$

$$\boldsymbol{b} = \begin{bmatrix} 5 \\ 4 \\ 10 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 5 \\ 2 \\ -4 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> python code/main.py data/example1 --solution -M
total solving wall time = 0.029999494552612305 sec
status                  = DONE : successfully solved
primal-dual optimal solution:
optimal objective value = 3.0000
numbers of iterations   = 30
solution x              = {
    1.0000,     1.6667,     1.3333,     0.0000,
    0.0000,     0.0000
}
solver terminated successfully
```

The optimal solution given by the problem is

$$\boldsymbol{x}^* = \begin{bmatrix} 5/3 & 4/3 & 1 \end{bmatrix}^T$$

$$\text{optimal objective} = 3$$

and my solution is

$$\boldsymbol{x} = \begin{bmatrix} 1.0000 \\ 1.6667 \\ 1.3333 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

$$\text{optimal objective} = 3.0000$$

, which are the same. (And very fast through the number of iterations is over 30.)

2) For given *Example 2*,

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 20 & 1 & 0 & 0 & 1 & 0 \\ 200 & 20 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{b} = \begin{bmatrix} 1 \\ 100 \\ 10000 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} -100 \\ -10 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> python code/main.py data/example2 --solution -M
total solving wall time = 0.03599977493286133 sec
status                  = DONE : successfully solved
primal-dual optimal solution:
optimal objective value = -10000.0000
numbers of iterations   = 37
solution x              = {
    0.0000,     0.0000, 10000.0000,     1.0000,
  100.0000,     0.0000
}
solver terminated successfully
```

The optimal solution given by the problem is

$$\boldsymbol{x}^* = \begin{bmatrix} 0 & 0 & 10000 \end{bmatrix}^T$$

$$\text{optimal objective} = -10000$$

and my solution is

$$\boldsymbol{x} = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 10000.0000 \\ 1.0000 \\ 100.0000 \\ 0.0000 \end{bmatrix}$$

$$\text{optimal objective} = -10000.0000$$

, which are the same. (And very fast through the number of iterations is over 30.)

3) For given *data1*,

There is no space to print the data here.

```
> python code/main.py data/data1 --solution -M
total solving wall time = 0.03099966049194336 sec
status                  = DONE : successfully solved
primal-dual optimal solution:
optimal objective value = 196200.0000
numbers of iterations   = 26
solution x              = {
    0.0000,     0.0000,     0.0000,     0.0000,
  137.6533,  1100.0000,   162.3467,     0.0000,
```

```
 1200.0000,   600.0000,   400.0000,     0.0000,
    0.0000,   400.0000,   900.0000,     0.0000,
    0.0000,     0.0000,  1562.3467,     0.0000,
  437.6533
}
given x*:||x_k-x*||_inf = 137.6533206895622
                (optimal solution may not be unique)
solver terminated successfully
```

The optimal solution given by the problem is

$$\boldsymbol{x}^* = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 1100.0 \\ 300.0 \\ 0.0 \\ 1200.0 \\ 600.0 \\ 400.0 \\ 0.0 \\ 0.0 \\ 400.0 \\ 900.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 1700.0 \\ 0.0 \\ 300.0 \end{bmatrix}$$

$$\text{optimal objective} = 196200$$

and my solution is

$$\boldsymbol{x} = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 137.6533 \\ 1100.0000 \\ 162.3467 \\ 0.0000 \\ 1200.0000 \\ 600.0000 \\ 400.0000 \\ 0.0000 \\ 0.0000 \\ 400.0000 \\ 900.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 1562.3467 \\ 0.0000 \\ 437.6533 \end{bmatrix}$$

$$\text{optimal objective} = 196200.0000$$

index	name	shape of $\boldsymbol{A}$	shape of $\boldsymbol{b}$	shape of $\boldsymbol{c}$
1	adlittle	56×138	56×1	138×1
2	afiro	27×51	27×1	51×1
3	bandm	305×472	305×1	472×1
4	beaconfd	173×295	173×1	295×1
5	blend	74×114	74×1	114×1
6	bore3d	233×334	233×1	334×1
7	brandy	220×303	220×1	303×1
8	capri	271×482	271×1	482×1
9	degen2	444×757	444×1	757×1
10	e226	223×472	223×1	472×1
11	grow7	140×301	140×1	301×1
12	israel	174×316	174×1	316×1
13	kb2	43×68	43×1	68×1
14	lotfi	153×366	153×1	366×1
15	lpi_bgdbg1	348×629	348×1	629×1
16	lpi_bgprtr	20×40	20×1	40×1
17	lpi_box1	231×261	231×1	261×1
18	lpi_chemcom	288×744	288×1	744×1
19	lpi_cplex2	224×378	224×1	378×1
20	lpi_ex72a	197×215	197×1	215×1
21	lpi_ex73a	193×211	193×1	211×1
22	lpi_forest6	66×131	66×1	131×1
23	lpi_galenet	8×14	8×1	14×1
24	lpi_itest2	9×13	9×1	13×1
25	lpi_itest6	11×17	11×1	17×1
26	lpi_klein2	477×531	477×1	531×1
27	lpi_mondou2	312×604	312×1	604×1
28	lpi_reactor	318×808	318×1	808×1
29	lpi_woodinfe	35×89	35×1	89×1
30	nug05	210×225	210×1	225×1
31	nug06	372×486	372×1	486×1
32	nug07	602×931	602×1	931×1
33	recipe	91×204	91×1	204×1
34	sc105	105×163	105×1	163×1
35	sc205	205×317	205×1	317×1
36	sc50a	50×78	50×1	78×1
37	sc50b	50×78	50×1	78×1
38	scagr25	471×671	471×1	671×1
39	scagr7	129×185	129×1	185×1
40	scfxm1	330×600	330×1	600×1
41	scorpion	388×466	388×1	466×1
42	scsd1	77×760	77×1	760×1
43	sctap1	300×660	300×1	660×1
44	share1b	117×253	117×1	253×1
45	share2b	96×162	96×1	162×1
46	stocfor1	117×165	117×1	165×1
47	vtp_base	198×346	198×1	346×1

TABLE I
TESTING DATASET SHAPES

The optimal objective values are the same. (And very fast.) The solutions are not the same since in this case, the optimal solution is not unique.

The complexity of my implementation with Big-M is approximately $\mathcal{O}(k(m+n)^3 + s(m+n)^2)$, where k is the average iteration times and s is the average line search times, and k is approximate linear to $\sqrt{m+n}$.

C. Extra Data

I collect some test data from the famous COAP website http://users.clas.ufl.edu/hager/coap/format.html. There 47 test cases are displayed in Table I.

The optimal solutions of these test cases are computed using CPLEX for validation. My PYTHON version solver runs all of these test cases one by one, and the result is shown in Table II.

index	name	time cost (s)	my solution	CPLEX solution
1	adlittle	0.1631	225494.9632	225494.9632
2	afiro	0.0450	-464.7531	-464.7531
3	bandm	1.0308	-158.6280	-158.6280
4	beaconfd	0.4210	33592.4858	33592.4858
5	blend	0.1310	-30.8121	-30.8121
6	bore3d	0.3460	0.0000	0.0000
7	brandy	1.4790	1518.5099	1518.5098
8	capri	1.3361	1912.6214	1912.6213
9	degen2	4.0693	-1435.1780	-1435.1780
10	e226	0.8841	-18.7519	-18.7519
11	grow7	0.5260	UNBOUNDED	UNBOUNDED
12	israel	0.7370	-896644.8219	-896644.8219
13	kb2	0.2580	UNBOUNDED	UNBOUNDED
14	lotfi	0.6821	-25.2647	-25.2647
15	lpi_bgdbg1	1.0692	VIOLATED	VIOLATED
16	lpi_bgprtr	0.0750	VIOLATED	VIOLATED
17	lpi_box1	0.2900	0.0000	0.0000
18	lpi_chemcom	1.3003	190.5401	190.5400
19	lpi_cplex2	0.4468	VIOLATED	VIOLATED
20	lpi_ex72a	0.2570	0.0000	0.0000
21	lpi_ex73a	0.2630	0.0000	0.0000
22	lpi_forest6	0.1850	270961.7553	270961.7552
23	lpi_galenet	0.0420	0.0000	0.0000
24	lpi_itest2	0.0220	VIOLATED	VIOLATED
25	lpi_itest6	0.0440	VIOLATED	VIOLATED
26	lpi_klein2	1.1186	VIOLATED	VIOLATED
27	lpi_mondou2	1.0324	174903255.9998	174903256.0000
28	lpi_reactor	1.1853	UNBOUNDED	UNBOUNDED
29	lpi_woodinfe	0.0630	0.0000	0.0000
30	nug05	0.4750	50.0000	50.0000
31	nug06	1.2362	86.0000	86.0000
32	nug07	4.8290	148.0000	148.0000
33	recipe	0.2340	UNBOUNDED	UNBOUNDED
34	sc105	0.2360	-52.2021	-52.2020
35	sc205	0.4850	-52.2021	-52.2020
36	sc50a	0.0970	-64.5751	-64.5750
37	sc50b	0.0960	-70.0000	-70.0000
38	scagr25	1.6147	-14753433.0608	-14753433.0600
39	scagr7	0.2767	-2331389.8243	-2331389.8240
40	scfxm1	1.2706	18416.7590	18416.7590
41	scorpion	1.3917	1878.1248	1878.1248
42	scsd1	0.3997	8.6667	8.6666
43	sctap1	1.0286	1412.2500	1412.2500
44	share1b	0.5284	UNBOUNDED	-76589.3185
45	share2b	0.1860	-415.7322	-415.7322
46	stocfor1	0.5194	UNBOUNDED	-41131.9762
47	vtp_base	0.7473	11120.6821	11120.6821

TABLE II
TESTING DATASET PERFORMANCE

Note among all 47 test cases, my solver successfully solve 45 of them, either getting the optimal solution or correct non-solvable information, which is impressive. The performance of PYTHON is not good enough, while the CPP version runs an average of 10 to 100 times faster and is provided.

V. CONCLUSION

In this project, I implement an Interior Point Method solver with many optimizations, which is user-friendly and robust at last. After that, a comparison with the CPLEX solver on a batch of test cases was made. It shows the great success of my solver that it could correctly detect the type of a problem, such as infeasible, unbounded. In conclusion, it is a good solver this year.