PRIMAL-DUAL INTERIOR-POINT METHOD
CS 520 FINAL PROJECT REPORT

DALI GUO[*] AND XIN YE[†]

Abstract. In this project report, we discuss an implementation of the primal-dual interior-point method for solving linear programming problem. Various pratical issues and implementation aspects are inspected, we will primarily focus on the robustness of our solver as is shown by numerical experiments.

1. Primal-dual method for linear programming.

One of the most frequently used formulation of the linear programming (LP) problem is the following standard form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$, $c, x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. We call x a feasible point if it satisfies the constrans $Ax = b$ and $x \geq 0$, the set of all feasible points is called the feasible set.

A geometric view of an LP is that we are trying to minimize an objective function that has constant gradient everywhere in a feasible polytope defined by the constrains. The fundamental theorem of linear programming states that if an LP (1.1) has solution then there exists a solution which is a basic feasible point. However every basic feasible point corresponds to a vertex of the feasible polytope, this nice property is key to the simplex method, one of the most popular algorithm for solving LP. The algorithm moves from one basic feasible point (vertex of the feasible polytope) to another and meanwhile reduces the value of the objective function. There are finite number of vertices hence the algorithm will eventually reach one solution, but its cost can potentially be exponential in the size of the problem. Unlike the simplex method, the primal-dual interior-point method theoretically has polynomial complexity. It generates iterates that roughly follows the central path and are strictly inside the polytope so that the algorithm only finds an approximation to the true solution. Now we briefly review the primal-dual interior-point method and its prototype algorithm.

Associated with (1.1), which is also called primal, the dual can be constructed by introducing Lagrange multiplier and rearranging data and variables:

$$\begin{aligned} \min \quad & -b^T \lambda \\ \text{s.t.} \quad & A^T\lambda + s = c, \\ & s \geq 0, \end{aligned} \tag{1.2}$$

where $\lambda \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$. Solutions of (1.1) and (1.2) are characterized by the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} A^T\lambda + s &= c, \\ Ax &= b, \\ x^T s &= 0, \\ x, s &\geq 0. \end{aligned} \tag{1.3}$$

[*]School of Industrial Engineering, Purdue University (dlguo@purdue.edu)

[†]Department of Mathematics, Purdue University (ye83@purdue.edu)

They can be organized into a non-linear equation

$$F(x,\lambda,s) = \begin{bmatrix} A^T\lambda + s - c \\ Ax - b \\ XSe \end{bmatrix} = 0, \tag{1.4}$$

where $X = \operatorname{diag}(x)$, $S = \operatorname{diag}(s)$ and $e^T = [1, 1, \ldots, 1]$. The idea in primal-dual method is to apply Newton's method to solve (1.4), if the current point (x, λ, s) is strictly feasible, the search direction is obtained by solving

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe \end{bmatrix}, \tag{1.5}$$

where the matrix on the left-hand side is the Jacobian $J(x, \lambda, s)$ of (1.4). The next iterate will be

$$(x,\lambda,s) + \alpha(\Delta x, \Delta\lambda, \Delta s)$$

for some step length $\alpha \in (0, 1]$. But in order to maintain strict feasibility of all iterates, namely, to keep $x, s > 0$, it is often the case that α can only be very small so the algorithm moves slowly towards the true solution. To remedy this issue, a modified Newton's procedure is used by introducing the idea of central path and a centering parameter.

The central path $\mathcal{C}$ with parameter $\tau > 0$ is defined by the following system:

$$\begin{aligned} A^T\lambda + s &= c, \\ Ax &= b, \\ x^T s &= n\tau, \\ x, s &> 0. \end{aligned} \tag{1.6}$$

Note that the central path (1.6) only differ from the KKT conditions (1.3) at the last two conditions: x and s no longer satisfy the complementary condition, all points on central path are strictly feasible points. It's easy to see that the point $(x_\tau, \lambda_\tau, s_\tau) \in \mathcal{C}$ converges to the solution of primal-dual problem as τ goes to 0. Instead of solving (1.5) for a search direction, we now solve

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \sigma\mu e \end{bmatrix}, \tag{1.7}$$

for some centering parameter $\sigma \in [0, 1]$ and $\mu = x^T s/n$. Note that if $\sigma = 0$, (1.7) becomes (1.5) and we get the same search direction as in the original Newton's method, we can reduce μ but often times we can't go to far on this direction; if $\sigma = 1$, the search direction is towards the point $(x_\tau, \lambda_\tau, s_\tau) \in \mathcal{C}$, the step length can be larger but we make little progress in reducing μ.

Now with all the basic theories metioned above, we can organize everything into Algorithm 1 which is the framework of primal-dual interior-point method.

Algorithm 1 *Primal-dual framework*

```
Get starting point $(x^{(0)}, \lambda^{(0)}, s^{(0)}) \in \mathcal{F}^o$
for $k = 0, 1, 2, \ldots$ do
    Pick $\sigma_k \in [0, 1]$
    Set $\mu_k = (x^k)^T s^k / n$ and solve (1.7)
        $$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^{(k)} & 0 & X^{(k)} \end{bmatrix} \begin{bmatrix} \Delta x^{(k)} \\ \Delta \lambda^{(k)} \\ \Delta s^{(k)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^{(k)} S^{(k)} e + \sigma_k \mu_k e \end{bmatrix}$$
    Set $\left(x^{(k+1)}, \lambda^{(k+1)}, s^{(k+1)}\right) \leftarrow \left(x^{(k)}, \lambda^{(k)}, s^{(k)}\right) + \alpha_k \left(\Delta x^{(k)}, \Delta \lambda^{(k)}, \Delta s^{(k)}\right)$
                                        $\triangleright$ choose $\alpha_k$ so that $\left(x^{(k+1)}, s^{(k+1)}\right) > 0$
    Check convergence criteria
end for
```

2. Practical and implementation issues. Note that Algorithm 1 is just a framework of the primal-dual method and a practical implementation actually differs a lot from it. Various practical and implementation issues have to be considered before even getting the algorithm to work, not alone solving an LP fast and accurately.

In this section, we will discuss the part of the problems we encounted when implementing the algorithm and how we dealt with them. Subsections are arranged in the order where they appear in the algorithm.

2.1. Presolve stage. Sparse matrix A in system (1.7) has a very high dimension for large scale problem. To relieve the numerical issues may happen in solving system (1.7), we do a set a presolve stage before optimization. By reducing the size of A, hopefully we can eliminate features could lead to numerical difficulties.

Our presolver performs these checks:

- *Zero rows in A*: When the row of A is zero, that means this constraint is not impacted by variables. Then we check if corresponding b is zero. If the b entry is zero, then we delete this row and corresponding b. Otherwise, this problem is infeasible.
- *Zero columns in A*: When we have zero column in A, that means the correspond variable x is not constrained by A and b. If the corresponding objective coefficient c is greater than zero, x takes its lower bound. If $c < 0$, x takes its higher bound. If $c = 0$, x can be any value.
- *Duplicated rows in A*: Duplicated rows in A means two constraints are the same. If they have the same b as well, we can remove arbitrary one of them and keep the other. If b's are different, the problem is infeasible.
- *Duplicated columns in A*: Duplicated columns means two variables have same constraint coefficients. We then delete these columns and variables and only keep one with smallest objective coefficient c since such variable can minimize the objective at most.

After these checks, the size of A is reduced and A becomes more robust in the next steps.

2.2. Converting to standard form. It is not hard to see that the problem description in LPnetlib is a little bit different from an LP in standard form, namely, both lower and upper bound on x exist. If we treat both of them as finite numbers and naively tranform it into standard form by adding slack variables. This immediately caused us trouble that looks like a numerical instability issue, a lot of the numbers in the iteration goes wildly large or small.

A carefull examination of the input data, take "afiro" as an example, shows that the higher bounds on x are all 10^{308} which is supposed to be infinity, so there is actually no upper bound and the problem is already in standard form. But we still want our code to handle all kinds of cituations no matter what lower and upper bound are given. This motivated an extra procedure to detect the following four types of variables before converting the problem to standard form:

1. free variable,
2. only lower bound exists,
3. only higher bound exists,
4. both lower and upper bounds exist,

then they are dealt with by different means.

Suppose the input data has the form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & l \le x \le h \end{aligned} \tag{2.1}$$

and (under some permutation) $x^T = [x_1^T, x_2^T, x_3^T, x_4^T]$ where x_i corresponds to the i-th catagory above, and similarly let $A = [A_1, A_2, A_3, A_4]$ and $c^T = [c_1^T, c_2^T, c_3^T, c_4^T]$. The inequility constrain becomes

$$l_2 \le x_2, \quad x_3 \le h_3, \quad l_4 \le x_4 \le h_4.$$

Now let $y_1 = x_1^+$ and $y_2 = x_1^-$ where $x_1 = x_1^+ - x_1^-$, and $y_3 = x_2 - l_2$, $y_4 = h_3 - x_3$, $y_5 = x_4 - l_4$ and $y_6 = h_4 - x_4$, the new variable $y^T = [y_1^T, y_2^T, \ldots, y_6^T]$ satisfies $y \ge 0$ and

$$\begin{aligned} A_1 y_1 - A_1 y_2 + A_2 y_3 - A_3 y_4 + A_4 y_5 &= b - A_2 l_2 - A_3 h_3 - A_4 l_4, \\ y_5 + y_6 &= h_4 - l_4, \end{aligned}$$

or, in matrix form,

$$\tilde{A} y = \tilde{b},$$

where

$$\tilde{A} = \begin{pmatrix} A_1 & -A_1 & A_2 & -A_3 & A_4 & \\ & & & & I & I \end{pmatrix}, \quad \tilde{b} = \begin{pmatrix} b - A_2 l_2 - A_3 h_3 - A_4 l_4 \\ h_4 - l_4 \end{pmatrix}$$

The objective function becomes

$$\begin{aligned} c^T x &= (c_1^T y_1 - c_1^T y_2 + c_2^T y_3 - c_3^T y_4 + c_4^T y_5) + (c_2^T l_2 + c_3^T h_3 + c_4^T l_4) \\ &= \tilde{c}^T y + \text{constant}, \end{aligned}$$

where $\tilde{c}^T = [c_1^T, -c_1^T, c_2^T, -c_3^T, c_4^T, 0^T]$. Thus the new LP in standard form is as follows

$$\begin{aligned} \min \quad & \tilde{c}^T y \\ \text{s.t} \quad & \tilde{A} y = \tilde{b} \\ & y \ge 0 \end{aligned} \tag{2.2}$$

The whole procedure is done by examining each elements of l and h and then collect index sets for each of the four types of variables. Once the new LP (2.2) is solved, the original solution vector x in (2.1) can be recovered using the index sets information and the corresponding parts of l and h.

2.3. Detecting infeasibility. In the case that the problem is infeasible, system (1.7) does not have a solution. Therefore we need to detect infeasible problem before solving the problem. Infeasibility can be detected by solving a so-called *Phase I* problem. Suppose the original problem is:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.3}$$

Now we consider a new problem:

$$\begin{aligned} \min \quad & e^T s \\ \text{s.t} \quad & Ax + s = b \\ & x, s \geq 0 \end{aligned} \tag{2.4}$$

We can prove that problem (2.3) is feasible if and only if problem (2.4) has a optimal solution where $s = 0$. If the original problem is feasible, we know that the new problem is also feasible because any feasible x in original problem plus $s = 0$ is a feasible solution for the new problem. Since $s = 0$ is the most minimized solution for $e^T s$, if the optimal value for (2.5) is non-zero which means some $s \neq 0$, there must exist a gap between Ax and b. That makes the original problem infeasible.

2.4. Starting point. In Algorithm 1, we need a starting point $(x^{(0)}, \lambda^{(0)}, s^{(0)}) \in \mathcal{F}^o$ to start the entire iteratino process. Similar to other iterative algorithm, the choice of starting point has significant effect on the algorithm, a poor starting point may result in slow or even no convergence as well as other numerical issues.

First we find $\tilde{x}$ with minimum norm and satisfying the primal constraint $Ax = b$, that is

$$\begin{aligned} \min_x \quad & x^T x \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{2.5}$$

Similarly, $\tilde{\lambda}$ and $\tilde{s}$ will satisfy the dual constraint and $\tilde{s}$ has the minimum norm

$$\begin{aligned} \min_{\lambda, s} \quad & s^T s \\ \text{s.t.} \quad & A^T \lambda + s = c. \end{aligned} \tag{2.6}$$

We see that (2.5) is a constrained least squares problem and (2.6) can be easily converted to a standard least squares problem, explicit solutions to both problems are available:

$$\begin{aligned} \tilde{x} &= A^T (AA^T)^{-1} b, \\ \tilde{\lambda} &= (AA^T)^{-1} Ac, \\ \tilde{s} &= c - A^T \tilde{\lambda}. \end{aligned} \tag{2.7}$$

The solutions $\tilde{x}$ and $\tilde{s}$ are not guarenteed to have all positive components, they are not ready to use as a starting point yet. Define

$$\tilde{\delta}_x = \max\left(-\frac{3}{2}\min \tilde{x}_i, 0\right),$$
$$\tilde{\delta}_s = \max\left(-\frac{3}{2}\min \tilde{s}_i, 0\right),$$

then modify $\tilde{x}$ and $\tilde{s}$ with

$$\begin{aligned}\hat{x} &= \tilde{x} + \tilde{\delta}_x e,\\ \hat{s} &= \tilde{s} + \tilde{\delta}_s e.\end{aligned} \tag{2.8}$$

Now $\hat{x}, \hat{s} \geq 0$, we still need to ensure that the starting point is not to close to zero and due to some heuristic reason x and s need to be not too dissimilar. We modify them again with

$$\hat{\delta}_x = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{s}}, \quad \hat{\delta}_s = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{x}},$$

and finally we get the desired point to start the algorithm with

$$x^{(0)} = \hat{x} + \hat{\delta}_x e, \quad \lambda^{(0)} = \tilde{\lambda}, \quad s^{(0)} = \hat{s} + \hat{\delta}_s e. \tag{2.9}$$

2.5. Solution to Newton equation. As can be seen in Algorithm 1, the majority of computation lies in solving the linear system in Newton step in each iterations. In fact in the final pratical version of the algorithm, two linear systems with the same matrix are solved in each iterations. A direct solver seems like the natural choice here, since once a factorization of the matrix is obtained it can be used multiple times for different right-hand side.

The linear systems involved in the algorithm all have the following form

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -r_{xs} \end{bmatrix}. \tag{2.10}$$

The matrix is a sparse matrix with much larger dimension compared to the constraint matrix A, the cost to factorize it can be too high especially dynamic pivoting stratergy needs to be used in each iterations when S and X vary.

In order to make the primal-dual method more pratical to deal with large scale LPs, two other mathematically equivalent formulations of (2.10) are often used. We will briefly review them now.

We'd like to mention that due to multiply reasons, we are still using the unreduced form (2.10) in our implementation which is much slower but more reliable and easier to implement than other two forms.

2.5.1. Augmented system form. By partially eliminate Δs from (2.10), we can decouple the unreduced system and obtain the augmented form

$$\begin{bmatrix} -SX^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_c + X^{-1}r_{xs} \\ -r_b \end{bmatrix}, \tag{2.11}$$

$$\Delta s = -X^{-1}(r_{xs} + S\Delta x). \tag{2.12}$$

When A has full row rank which is guarenteed by the presolve stage and $X, S > 0$ since all iterates are strictly feasible, the matrix in (2.11) is symmetric indefinite. An LDL factorization that has lower computation cost than a naive factorization on the unreduced matrix in (2.10) can be used but (2.11) suffers more from conditioning especially when the algorithm is close to a solution.

In our very first version, we tried to generate this augmented system in the algorithm and called the Julia built-in function `ldltfact` to factorize it. But turned out this function, when applied to sparse matrix, only works for symmetric definite case, it returns an error when encounters a negative diagonal entry. Thus after a few days, we abbandoned this stratergy and tried the next form.

2.5.2. Normal equation form. By continueing decouple the two variables in (2.11), we can then obtain the normal equation form of the Newton system

$$AS^{-1}XA^T\Delta\lambda = -r_b + AS^{-1}(Xr_c + r_{xs}), \tag{2.13}$$

$$\Delta s = -r_c - A^T\Delta\lambda, \tag{2.14}$$

$$\Delta x = -S^{-1}(r_x s + X\Delta s). \tag{2.15}$$

The matrix $AS^{-1}XA^T$ in (2.13) is symmetric positive-definite and has the smallest dimension among all three forms, the Cholesky factorization on it has the lowest computation cost but it suffers the most from conditioning.

Luckily several modified Cholesky factorization techniques can be applied and also provides a realiable search direction. A smart way to deal with small pivots while avoiding dynamic diagonal pivoting is to simply skip the entire column if a diagonal element is too small compare to previous diagonal entries. Then in the solution stage, we just need to set the corresponding elements in the solution to be zero. This algorithm only differs very little in implementation compared to the basic Cholesky factorization, and it gives the best computation cost among all three forms, thus we tried this after failed in augmented system. But after debugging our code, the solution failed to provide a good search direction and it's hard to investigate into the issue. Thus, finally, we have no choice but to go back to the unreduced form and simply call a sparse LU factorization to solve the Newton step.

2.6. Mehrotra's predictor-corrector algorthim. We will follow Mehrotra's predictor-corrector (MPC) algorithm in our implementation, it enhances the bacis Newton step by adding a correction to the search direction and has a systematic way of picking a suitable centering parameter. The algorithm will generate infeasible points (x, λ, s) in the process but ensures $x, s > 0$.

During a single iteration, we first find the affine-scaling direction by solving the basic Newton step

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta\lambda^{\text{aff}} \\ \Delta s^{\text{aff}} \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -r_{xs} \end{bmatrix}, \tag{2.16}$$

where the right-hand side consists of residuals of the KKT conditions

$$r_b = Ax - b, \quad r_c = A^T\lambda + s - c, \quad r_{xs} = XSe.$$

Note that this system is (1.7) by setting $\sigma = 0$. Then we find the maximum step length on the affine-scaling direction towards boundary

$$\begin{aligned} \alpha_{\text{aff}}^{\text{pri}} &= \operatorname{argmax}\{\alpha \in [0,1] | x + \alpha\Delta x^{\text{aff}} \geq 0\}, \\ \alpha_{\text{aff}}^{\text{dual}} &= \operatorname{argmax}\{\alpha \in [0,1] | s + \alpha\Delta s^{\text{aff}} \geq 0\}. \end{aligned} \tag{2.17}$$

Assume we take both full length for the primal and dual variables, then the hypothetical value of μ

$$\mu_{\text{aff}} = (x + \alpha_{\text{aff}}^{\text{pri}} \Delta x^{\text{aff}})^T (s + \alpha_{\text{aff}}^{\text{dual}} \Delta s^{\text{aff}})/n \tag{2.18}$$

can be used to measure the effectiveness of affine-scaling direction. If μ_{aff} is much smaller compared to $\mu = x^T s/n$ in the current step, then the affine-scaling direction can already provide significant reduction thus a comparatively small centering parameter σ is used. Mehrotra suggested $\sigma = (\mu_{\text{aff}}/\mu)^3$ which is proved to be effective in practice.

Then the centering step is combined with a correction step which will bring the element-wise product of x and s closer to target value zero into a second linear system

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cc}} \\ \Delta \lambda^{\text{cc}} \\ \Delta s^{\text{cc}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma \mu e - \Delta X^{\text{aff}} \Delta S^{\text{aff}} e \end{bmatrix}. \tag{2.19}$$

Note that $\sigma\mu e$ corresponds to the centering step and $-\Delta X^{\text{aff}} \Delta S^{\text{aff}} e$ corresponds to the corrector.

Now with the affine-scaling direction and the centering-corrector direction, the search direction in this iteration is

$$(\Delta x, \Delta\lambda, \Delta s) = (\Delta x^{\text{aff}}, \Delta\lambda^{\text{aff}}, \Delta s^{\text{aff}}) + (\Delta x^{\text{cc}}, \Delta\lambda^{\text{cc}}, \Delta s^{\text{cc}}), \tag{2.20}$$

and the step lengths for primal and dual varibles can be obtained similar to (2.17):

$$\begin{aligned} \alpha_{\max}^{\text{pri}} &= \text{argmax}\{\alpha \geq 0 | x + \alpha \Delta x^k \geq 0\}, \\ \alpha_{\max}^{\text{dual}} &= \text{argmax}\{\alpha \geq 0 | s + \alpha \Delta s^k \geq 0\}. \end{aligned} \tag{2.21}$$

Mehrotra defines a heuristic in which this scaling factor applied to primal and dual variables approaches 1 on later iterations. This can help the algorithm converges more quickly than simply using fixed scaling factor. The method is realized by choosing this factor to be the largest value for which the "critical" pairwise products $x_i s_i$ which reaches maxima in the previous method are larger than $\gamma_f \mu_+$, where γ_f is a small constant and μ_+ is an approximation to μ_{k+1}. The algorithm is actually that given a parameter γ_f close to 0 and the $(\Delta x^k, \Delta\lambda^k, \Delta s^k), \alpha_{\max}^{\text{pri}}, \alpha_{\max}^{\text{dual}}$ from (2.21). Calculate

$$\mu_+ = (x_k + \alpha_{\max}^{\text{pri}} \Delta x^k)^T (s^k + \alpha_{\max}^{\text{dual}} \Delta s^k)/n$$

For particular index i for which $x_i^k + \alpha_{\max}^{\text{pri}} \Delta x_i^k = 0$, define f^{pri} by

$$(x_i^k + f^{\text{pri}} \alpha_{\max}^{\text{pri}} \Delta x_i^k)(s_i^k + \alpha_{\max}^{\text{dual}} \Delta s_i^k) = \gamma_f \mu_+$$

for the particular index i for which $s_i^k + \alpha_{\max}^{\text{dual}} \Delta s_i^k = 0$, define f^{dual} by

$$(x_i^k + \alpha_{\max}^{\text{pri}} \Delta x_i^k)(s_i^k + f^{\text{dual}} \alpha_{\max}^{\text{dual}} \Delta s_i^k) = \gamma_f \mu_+$$

set

$$\alpha_k^{\text{pri}} = \max(1 - \gamma_f, f^{\text{pri}}) \alpha_{\max}^{\text{pri}}, \;\; \alpha_k^{\text{dual}} = \max(1 - \gamma_f, f^{\text{dual}}) \alpha_{\max}^{\text{dual}}$$

In our solver, we set $\gamma_f = 0.01$

2.7. Finite termination. One feature of the primal-dual method that is different from simplex method is that it never computes an exact solution but only an approximated one, thus some convergence criteria to determine when to stop the iteration.

Optimality conditions come in handy in this case. Note that a solution (x^*, λ^*, s^*) satisfies the KKT conditions (1.3)

$$\begin{aligned} Ax^* &= b, \\ A^T\lambda^* + s^* &= c, \end{aligned}$$

and the primal and dual problems have the same objective value, that is

$$c^T x^* = b^T \lambda^*.$$

Thus a point (x, λ, s) is accepted as an approximated solution if the relatively residuals of the above three equation are all within a given tolerance $\epsilon > 0$, i.e., if

$$\begin{aligned} \frac{\|Ax - b\|}{1 + \|b\|} &< \epsilon, \\ \frac{\|A^T\lambda + s - c\|}{1 + \|c\|} &< \epsilon, \\ \frac{|c^T x - b^T y|}{1 + |c^T x|} &< \epsilon. \end{aligned} \tag{2.22}$$

2.8. Practical algorithm. With all the practical and implementation issues discussed before, we are ready to introduce a more practical primal-dual algorithm in Algorithm 2.

Algorithm 2 *Practical primal-dual algorithm*

```
Input: $A, b, c, l, h$ (data of the LP), $\epsilon$ (tolerance), $k_{max}$ (max number of iteration)
Output: $x$ (solution vector)
Presolve stage to preprocess input constraint matrix $A$
        ▷ return false flag if infeasibility in constraint matrix is detected
Convert the problem into standard form
Detect infeasibility
        ▷ return false flag if infeasibility in the problem is detected
Obtain starting point $(x^{(0)}, \lambda^{(0)}, s^{(0)})$ from (2.7)–(2.9)
for $k = 0, 1, 2, \ldots, k_{\max}$ do
    Factorize the matrix in (2.16) and store the factors
    Solve for affine-scaling direction in (2.16)
    Compute $\mu_{\text{aff}}$ in (2.18) and $\mu$
    Set centering parameter $\sigma = (\mu_{\text{aff}}/\mu)^3$
    Solve for centering-corrector direction in (2.19)
            ▷ factors from affine-scaling step is reused here
    Compute step length by (2.17) and update iterates
    Check convergence criteria (2.22)
end for
Recover solution $x$ to the original LP
        ▷ Reverse the processes of converting and presolve
```

3. Numerical experiments. In this section, we test the baby problem from lecture and the required test problem set posted in the website. Most tests are passed and our solver performs well in all tests.

3.1. Baby problem. First we implement our algorithm on the baby problem provided in lecture. That is:

$$\begin{aligned} \min \quad & -x_1 - 2x_2 \\ \text{s.t.} \quad & -2x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 7 \\ & x_1 \leq 3 \end{aligned}$$

We solve this problem by two types of step-length algorithms. One is Mehrotra heuristic scaling and the other is fixed scaling. Both of them converges to the optimal solution. The result is shown as follows:

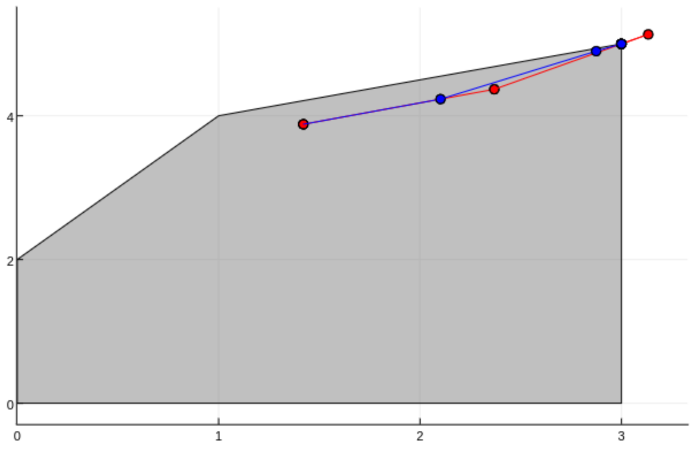

FIG. 3.1. *Iteration traces of two step-length methods. Mehrotra heuristic scaling factor method is in red and fixed scaling factor method is in blue.*

In Figure 3.1, Mehrotra heuristic scaling factor method converges to optimal solution in 4 steps, however fixed scaling factor method uses 6 steps to get converged. We can take a closer look around the optimal solution,

Figure 3.2 gives us a closer look around the optimal solution. Mehrotra heuristic scaling factor method goes beyond the feasible region and then goes back to the optimal solution in the next step. However, fixed scaling factor method takes a extremely step before getting the optima. As this example, Mehrotra heuristic scaling factor method takes advantage of larger step length in finding optima.

If we take a more closer look at the convergence of these two algorithms, we can find that the fixed scaling factor method converges better than the heuristic one even they have the same termination criteria. At the last step, the fixed scaling factor method converges to exact $(3, 5)$ but there is still a gap for heuristic scaling factor method. The heuristic scaling factor method is even "worse" than the second last step for fixed scaling factor method.

That is because the heuristic method has a step length close to 1 at the end of the iteration. This can help algorithm converges earlier but the last step cannot guarantee an accurate step directly to the point. However the fixed scaling factor method is

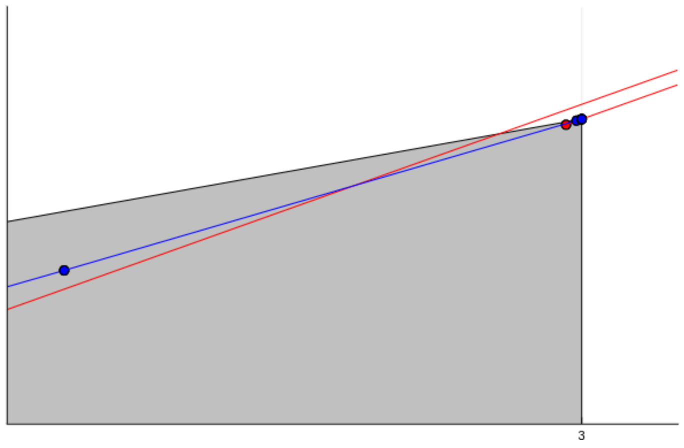

Fig. 3.2. *Closer look around the optimal solution for two methods. Mehrotra heuristic scaling factor method is in red and fixed scaling factor method is in blue.*

relatively conservative which prefers to take small steps around the optima. We can image when the problem has a very high dimension, the fixed scaling factor method will cost more time.

Therefore, we choose Mehrotra heuristic scaling factor method for selecting step length at each step.

3.2. Suggested test problems. Then we test 9 problems in LPnetlib as required. The results are shown below,

Table 3.1
Test result of problems from LPnetlib

Problem statistics					Our MPC IP Solver		
name	full-rank	feasible	iter #*	obj. value*	iter #	obj. value	time (s)**
afiro	✓	✓	6	-464.75	7	-464.75	4.49
brandy	×	✓	292	1518.51	19	1518.51	5.27
fit1d	✓	✓	NA	-9146.38	17	-9146.38	7.35
adlittle	✓	✓	123	2.25×10^5	10	2.25×10^5	4.67
agg	✓	✓	NA	-3.60×10^7	31	-3.60×10^7	7.80
ganges	✓	✓	NA	-1.09×10^5	17	-1.09×10^5	48.59
stocfor1	✓	✓	NA	-4.11×10^4	16	-4.11×10^4	4.79
25fv47	×	✓	NA	5501.85	26	5501.85	26.5
chemcom	✓	×	–	–	–	–	–

* The iteration numbers and objective value in Problem statistics is solved by MINOS 5.0.
** The results are computed with Julia 0.5.1 ×86_64-linux-gnu in Intel®Core ™i5 5200U CPU @ 2.20GHz × 4

In table 3.2 we can find that all problems are solved. Then each specific case is elaborated

- ***A* is full rank and problem is feasible.** Take problem "afiro" as an example. The size of A before presolving is 27×51. Since A is full rank, the size of A does not reduce after presolving. We can look at the iteration procedure in the next table:

TABLE 3.2
Problem "afiro"

ITER	MU	RESIDUAL	ALPHAX	ALPHAS
0	1.54e+03	1.61e+01	0	0
1	1.54e+03	1.61e+01	0.8451	1.122
2	2.32e+02	2.56e+00	1.017	0.848
3	3.11e+01	3.22e-01	2.483	0.8137
4	5.96e+00	7.02e-02	0.9661	0.9099
5	4.95e-01	6.09e-03	0.8809	1.012
6	3.49e-02	4.29e-04	0.9997	1
7	7.75e-06	8.97e-08	1	1

Problem "afiro" converges in 7 iterations. After the residual reaches the tolerance the iteration terminates. The step length for primal and dual variables, ALPHAX and ALPHAS, start from 0. Finally the two step length reach 1 as a full step.

- ***A* is not full rank and problem is feasible.** Take problem "brandy" as an example. The size of A before presolving is 220×303. In presolving stage, the algorithm finds 7 replicated rows. That means these constraints are equivalent. Therefore we simply pick one from the equivalent constraints if they have the same b and drop all others. Otherwise, the problem is infeasible. After the presolving stage, the size of A is reduced to 193×303 which has a full row rank. The solving procedure is as followed, The algorithm converges after 19 iterations. As test problem "afiro", the step length for both variables become 1 at the end of the algorithm.
- **Problem is infeasible.** We take problem "chemcom" as an example. Stage one is designed in our solver to detect infeasibility. We expect the objective value as 0 at phase one however in this problem it gives 9824. Then the algorithm is terminated and throw an error indicating that this problem is infeasible.
- **Problem is unbounded.** Since the test problem set does not have an unbounded problem. Then we remove the last constraint in problem (3.1) converting it into an unbounded problem. The new problem is,

$$\begin{array}{rl} \min & -x_1 - 2x_2 \\ \text{s.t.} & -2x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 7 \end{array}$$

  Then we solve this new problem in our solver.
  We can find that in the third iteration, the algorithm throws an error because the step length for primal variable is infinite which means the constraints are unbounded for our problem and we can get a infinity as the objective value for this problem. The test for unbounded case works here.

3.3. Extended test problems from LPnetlib. Some extended test problems are tested in our project.

Table 3.3 shows some extra problems tested in our solver. Some special cases are

Table 3.3
Problem "brandy"

ITER	MU	RESIDUAL	ALPHAX	ALPHAS
0	4.00e+01	5.98e+02	0	0
1	4.00e+01	5.98e+02	0.7876	0.9254
2	8.70e+00	1.27e+02	0.3635	0.2454
3	6.04e+00	8.09e+01	0.2953	0.2953
4	4.81e+00	5.70e+01	0.2868	0.5371
5	3.87e+00	4.07e+01	0.3246	0.6519
6	2.49e+00	2.75e+01	0.01581	0.02317
7	2.73e+00	2.70e+01	0.0358	0.06346
8	3.45e+00	2.61e+01	0.05096	0.04037
9	3.47e+00	2.48e+01	0.2217	0.1505
10	3.28e+00	1.93e+01	0.5794	0.5949
11	2.31e+00	8.12e+00	0.804	0.854
12	4.40e-01	1.59e+00	0.7872	0.6645
13	1.18e-01	3.39e-01	0.7419	0.8397
14	2.91e-02	8.74e-02	0.8377	0.6946
15	7.05e-03	1.42e-02	0.9401	0.8594
16	1.65e-03	8.60e-04	0.7742	0.5626
17	6.56e-04	1.99e-04	0.7878	0.9977
18	6.83e-05	4.12e-05	0.9923	0.9944
19	4.95e-07	3.17e-07	1	1

Table 3.4
Problem (3.2)

ITER	MU	RESIDUAL	ALPHAX	ALPHAS
0	4.80e+00	1.74e+00	0	0
1	4.80e+00	1.74e+00	1.426	0.546
ERROR: This problem is unbounded				

tested here to ensure the robustness. The result shows that most cases are solved perfectly. Then some special cases not covered in required problem are analyzed.

- **Large scale problem.**. All given test problems have size less than 1000×1000. Problem "ship12l" is tested as a large-scale problem. The size of A is 1151×5533. Our algorithm terminates at the 17^{th} step and converges to the optima. 114.5 seconds are taken to find the solution. 4.3 G of memory is allocated and totally 71.35 G of memory is used to compute the result. The solving procedure is as follows:
  From Table 3.3 we can find that high dimensional problem often costs more computation. But in our solver, the iteration steps often do not increase with the size of the problem. That shows a good feature of find right Δx and Δs with appropriate step length.
- **Non full rank A and infeasible.** Here we find problem "gran" has a non-full-rank A and infeasible constraints. Size of A in this problem is 2658×2525. After presolving, A is converted to full-rank matrix. In phase one the algorithm detects the infeasiblity and return a false flag as result.

TABLE 3.5
Extended test result of problems from LPnetlib

Problem statistics					Our MPC IP Solver		
name	full-rank	feasible	iter #*	obj. value*	iter #	obj. value	time (s)**
agg2	✓	✓	NA	-2.02×10^7	19	-2.02×10^7	14.03
bandm	✓	✓	362	-158.63	17	-158.63	5.88
capri	✓	✓	273	2690.01	23	2690.01	5.74
e226	✓	✓	570	-18.75	22	-18.75	5.63
fffff800	✓	✓	NA	5.56×10^5	44	5.56×10^5	12.50
bnl1	×	✓	NA	1977.63	28	1977.63	16.62
tuff	×	✓	NA	0.29	23	0.29	6.70
ship12l	×	✓	NA	1.47×10^6	17	1.47×10^6	114.50
cplex2	✓	×	–	–	–	–	–
reactor	✓	×	–	–	–	–	–
gran	×	×	–	–	–	–	–
etamacro	✓	✓	904	-756.72	–	–	–

* The iteration numbers and objective value in Problem statistics is solved by MINOS 5.0.
** The results are computed with Julia 0.5.1 ×86_64-linux-gnu in Intel®Core $^{\text{TM}}$i5 5200U CPU @ 2.20GHz × 4

TABLE 3.6
Problem "ship12l"

ITER	MU	RESIDUAL	ALPHAX	ALPHAS
0	6.00e+04	2.50e+02	0	0
1	6.00e+04	2.50e+02	0.921	0.7022
2	6.29e+03	3.05e+01	0.8796	0.8461
3	1.02e+03	5.49e+00	0.7545	0.1721
4	4.21e+02	2.47e+00	0.07744	0.3953
5	3.20e+02	1.79e+00	0.4794	0.8365
6	1.43e+02	7.41e-01	0.5521	0.8003
7	6.71e+01	3.43e-01	0.6391	0.531
8	3.94e+01	1.87e-01	0.6735	0.7269
9	1.58e+01	7.37e-02	0.6429	0.8362
10	6.98e+00	3.23e-02	0.08856	0.8014
11	5.36e+00	2.64e-02	0.5582	0.6968
12	2.39e+00	1.18e-02	0.7692	0.8512
13	6.35e-01	3.06e-03	0.9858	0.7394
14	1.14e-01	5.49e-04	0.8298	0.7967
15	2.61e-02	1.26e-04	0.9242	0.9134
16	2.41e-03	1.17e-05	0.7099	0.9926
17	6.31e-04	2.97e-06	1	1

- **Cannot be solved.** Problem "etamacro" cannot be solved by our solver. The solver checks the rank of A and it is full rank. That means there is no rank deficiency issue. Also, this problem passes the infeasiblity test since it has a solution. But our solver encounters a singular error solving the linear system. Not sure why such error happens.

REFERENCES

[1] J. Nocedal, S. Wright, *Numerical optimization*, Springer, 2006.
[2] S. Wright, *Primal-dual interior-point methods*, SIAM, 1997.