FINAL PROJECT REPORT

Student Name: Timur Ibrayev, Gobinda Saha

May 4, 2020

1. Introduction

Linear programs (LP) is the type of optimization problems which involve linear objective functions along with linear equality and/or inequality constraints. Despite there is a certain limitation on what can be described by only linear equations, linear programming remains one of the most widely used optimization tools finding its use in economics, engineering, and management. There are various techniques developed over decades to solve this type of optimization problems with one of the most famous one being Simplex methods. However, there is a class of different methods, called Interior-point (IP) methods, which are shown to be more efficient on certain type of LP problems which might cause Simplex method to be exponentially inefficient in the size of the problem.

In this report we describe our implementation of robust primal-dual interior-point method for solving LP using Julia programming language. To solve problems with varying degree of complexity, we have included various preprocessing steps. Moreover, efficient linear algebra routines are also implemented to tackle issues arising from large matrix computations. Our extensive computational results on Netlib problems demonstrates robustness of our implementation.

We mainly focus on algorithm implementation aspects with emphasis on why each particular feature was implemented and added to the solver. In order to describe logical progression of designing the IP LP solver presented in this work, most of the ideas presented are described to answer the following questions: what is the feature/property to be added, why it is required, and how it is implemented.

2. Interior-Point Solvers

The feasible set of linear problems, that is the set of points which satisfy linear constraints, has the form of polytope in the space of input variables. Interior point method is the way of solving linear problems where a solution is found by traversing the interior space of such feasible polytope. This in contrast to Simplex method of solving linear problems, where the solution is found by traversing the vertices of the feasible polytope. However, similarly to Simplex method, IP method solves LP by searching for points and Lagrangian multipliers which satisfy KKT conditions.

Theory of Primal-dual IP LP Solvers

Given the problem written in standard form, such as:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \tag{1}$$

and its dual problem, such as:

$$\begin{aligned} \underset{\boldsymbol{\lambda}}{\text{maximize}} \quad & \mathbf{b}^T \boldsymbol{\lambda} \\ \text{subject to} \quad & \boldsymbol{A}^T \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq 0 \end{aligned} \tag{2}$$

the solution of the LP problem can be found by finding set of $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{s}^*)$, which would satisfy the KKT conditions:

$$\boldsymbol{A}^T \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c} \tag{3}$$

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \tag{4}$$

$$\mathbf{x}_i \mathbf{s}_i = 0, \quad \forall i \in \{1...n\} \tag{5}$$

$$(\mathbf{x}, \mathbf{s}) \geq 0 \tag{6}$$

Primal-dual IP LP solvers [1, 2] attempt to find the solution by restating the KKT conditions as a system of non-linear equations which can be solved by Newton-Raphson's method. The solution of such system gives the search direction for each variable $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$. By iteratively solving the non-linear system and updating variables, the goal is to find $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{s}^*)$ which would satisfy KKT conditions *in the limit* and inequality constraints $(\mathbf{x}, \mathbf{s}) > 0$ *at every iteration*. This formally can be written in the following way:

1. Rewrite KKT conditions for LP in standard form to have the system of non-linear equations

$$F = \begin{bmatrix} \boldsymbol{A}^T \boldsymbol{\lambda} + \mathbf{s} - \mathbf{c} \\ \boldsymbol{A}\mathbf{x} - \mathbf{b} \\ \boldsymbol{X}\boldsymbol{S}\mathbf{e} \end{bmatrix} \tag{7}$$

   where $\boldsymbol{X} = diag(x_1, x_2, ..., x_n)$, $\boldsymbol{S} = diag(s_1, s_2, ..., s_n)$ and $\mathbf{e} = [1, ..., 1] \in \mathbb{R}^n$.

2. For Newton-Raphson's method compute Jacobian by finding partial derivatives of the system of non-linear equations w.r.t. every variable $(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s})$

$$J = \begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \tag{8}$$

3. According to Newton-Raphson's method, by using Jacobian solve for roots of F to obtain descent direction for every variable

$$J\mathbf{p} = -F \Rightarrow \begin{bmatrix} \Delta_x \\ \Delta_\lambda \\ \Delta_s \end{bmatrix} = -J^{-1}F \tag{9}$$

   and

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \\ \mathbf{s}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_k \\ \boldsymbol{\lambda}_k \\ \mathbf{s}_k \end{bmatrix} + \alpha * \begin{bmatrix} \Delta_x \\ \Delta_\lambda \\ \Delta_s \end{bmatrix} \tag{10}$$

   such that $\mathbf{x}, \mathbf{s} > 0$.

Trade-off between affine scaling and centering direction

One of the major challenges of interior-point LP solvers can be highlighted by two key phrases: *satisfy KKT conditions in the limit* and *satisfy inequality constraints* $(\mathbf{x}, \mathbf{s}) > 0$ *at every iteration.* In order to explain why these two possess difficulties, we have to define the duality measure and a central path.

Duality measure is the quantity which is defined as follows:

$$\mu = \frac{1}{n}\sum_{i}^{n} \mathbf{x}_i \mathbf{s}_i = \frac{\mathbf{x}^T \mathbf{s}}{n} \tag{11}$$

which can be thought of as average value of the pairwise products $\mathbf{x}_i \mathbf{s}_i$ for all $i \in [1...n]$. From KKT condition shown in Eq.(5), it can be seen that at the solution, this quantity should be zero.

In order to define the central path, we have to look at alternative formulation of the primal problem (Eq.(1)) using log-barrier:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} - \tau \textstyle\sum_i^n \ln \mathbf{x}_i \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{x} = \mathbf{b} \end{aligned} \tag{12}$$

Solving this problem is equivalent to solving for KKT conditions similar to Eq.(3-6), but with:

$$\mathbf{x}_i \mathbf{s}_i = \tau, \quad \forall i \in \{1...n\} \tag{13}$$

Then, for some given value of τ we can define the central path as the set of points which satisfy KKT conditions for that τ value. In other words, the central path is:

$$C = \{(\mathbf{x}_\tau, \boldsymbol{\lambda}_\tau, \mathbf{s}_\tau) | \tau > 0\} \tag{14}$$

for some given value of τ. Alternatively, for a feasible points (i.e. satisfying constraints), it can be defined in terms of Eq.(7) as:

$$F = \begin{bmatrix} 0 \\ 0 \\ \tau\mathbf{e} \end{bmatrix} \tag{15}$$

The importance of the central path comes from the observation that if the central path converges to anything in the limit of $\tau \to 0$, it must converge to a solution of the primal problem. By noting that τ is related to duality measure μ, this observation tells us that the central path can serve as a guide for interior-point method to achieve convergence to the solution by maintaining positivity of $\mathbf{x}$ and $\mathbf{s}$ and gradually reducing the duality measure μ.

Moreover, the system of non-linear equations can be modified (by combining Eq.(7) and Eq.(15)) to implicitly accommodate the requirement to follow the central path as follows:

$$F = \begin{bmatrix} \boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} - \mathbf{c} \\ \boldsymbol{A}\mathbf{x} - \mathbf{b} \\ \boldsymbol{X}\boldsymbol{S}\mathbf{e} - \sigma\mu\mathbf{e} \end{bmatrix} \tag{16}$$

where μ is the duality measure and σ is centering step parameter, which essentially defines to what extend we wish to reduce the duality measure during this step.

The above modification eventually explains the trade-off between affine scaling and centering directions and the major challenge associated with two key phrases: *satisfy KKT conditions in the limit* and *satisfy inequality constraints* $(\mathbf{x}, \mathbf{s}) > 0$ *at every iteration*.

- When $\sigma = 0$, the direction found by solving for roots of Eq.(16) defines affine scaling direction. In this case, we are trying to achieve maximum reduction of duality measure by directly equating it to zero ($\boldsymbol{XS}\mathbf{e} = 0$). However, to *satisfy inequality constraints* $(\mathbf{x}, \mathbf{s}) > 0$ *at every iteration*, usually affine scaling requires small step and consequently does not allow much progress.
- When $\sigma = 1$, the direction defines centering direction. In this case, the reduction of duality measure is the minimal, because we are decreasing $\boldsymbol{XS}\mathbf{e}$ by the current average value of duality measure $\mu\mathbf{e}$. In order to *satisfy KKT conditions in the limit*, the centering parameter cannot be kept at high value all the time. However, by following the central path before following affine scaling it is possible to achieve a significant reduction of duality measure on the next iteration without violating inequality constraints.

Hence, the major challenge can be redefined as the need for mechanism to adaptively switch between high and small values of centering parameter to achieve trade off between reducing μ and increasing effectiveness of each step.

3. Practical Implementation of Interior Point Solver

Despite the previous section might give impression that solving LP using IP solvers is straightforward, practical implementation of even simple IP LP solver which performs computations using heuristically defined values for step length and sigma centering parameter reveals that special care should be taken to account for numerous factors. These factors, which motivate implementing different features explained in this and the next sections, include:

1. As highlighted above, one of the major questions is to how select centering parameter and step length to balance between reducing duality measure and making significant progress.
2. How to find a good starting points, which would ensure that the solver does not diverge in the first couple of iterations due to bad initialization.
3. How to ensure that the problem itself is not infeasible, well defined (has full rank), and does not contain redundancies which can result not only in inefficient computation of the solver but also in the wrong solution.
4. How to ensure that the solver selects the most efficient way of computing single steps involving Newton-Raphson's method and sparse matrices.

All of these factors were considered to our best efforts during implementation of our IPLP solver. The following sections describe the features which aim to tackle one of the factors specified above.

Mehrotra Predictor-Corrector Algorithm

Despite being the first, it was one of the major decisions to build our IPLP solver with the base being implementation of Mehrotra's Predictor-Corrector (MPC) algorithm [3]. The main reason for selecting MPC algorithm as the base is not only the fact that it is used as the base for many

commercially available state-of-the-art interior-point LP solvers, but also the fact that it provides the solution for the first factor listed in the previous section.

Specifically, one of the attracting advantages of MPC algorithm is its heuristic in adaptively choosing centering parameter σ. It comes as a by-product from the main corrector principle of the algorithm, which tries to compensate for the linearization error resulting from pure affine-scaling direction. The idea is to firstly compute affine-scaling direction ("predictor step") and then use that information to estimate both centering required to make a good progress ("centering step") as well as correction required in that direction to better approximate our non-linear system of equations F ("corrector step").

Moreover, MPC algorithm also incorporates the idea of using different step lengths for the primal (i.e. $\mathbf{x}$) and dual (i.e. $\boldsymbol{\lambda}, \mathbf{s}$) variables, which was used in practice even before Mehrotra's work. This improves computational efficiency by allowing independent movement in primal and dual problem progression. Formally, MPC computations can be stated as follows:

0. Define the residual terms as:
$$\mathbf{r}_c = \boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} - \mathbf{c} \tag{17}$$
$$\mathbf{r}_b = \boldsymbol{A}\mathbf{x} - \mathbf{b} \tag{18}$$
as well as the diagonal matrices of $\boldsymbol{X} = diag(\mathbf{x})$ and $\boldsymbol{S} = diag(\mathbf{s})$.

1. "**Predictor step**": compute affine-scaling direction $(\Delta\mathbf{x}^{aff}, \Delta\boldsymbol{\lambda}^{aff}, \Delta\mathbf{s}^{aff})$ by solving the following system:
$$\begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{aff} \\ \Delta\boldsymbol{\lambda}^{aff} \\ \Delta\mathbf{s}^{aff} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c \\ -\mathbf{r}_b \\ -\boldsymbol{X}\boldsymbol{S}\mathbf{e} \end{bmatrix} \tag{19}$$

2. "**Centering step**": compute the centering parameter according to Mehrotra's heuristic:
$$\sigma = (\frac{\mu_{aff}}{\mu})^3 \tag{20}$$
where $\mu = \mathbf{x}_k^T\mathbf{s}_k/n$ is the current duality measure with the current iterates and μ_{aff} is the duality measure that would be obtained if the affine-scaling direction is taken:
$$\mu_{aff} = (\mathbf{x} + \alpha_{aff}^{prim}\Delta\mathbf{x}^{aff})^T(\mathbf{s} + \alpha_{aff}^{dual}\Delta\mathbf{s}^{aff})/n \tag{21}$$
where α_{aff}^{prim} and α_{aff}^{dual} are maximum allowable steplengths for primal and dual variables, respectively, along the affine-scaling direction, which do not violate strict inequality constraints:
$$\alpha_{aff}^{prim} = \min\left(1, \min_{i:\Delta x_i^{aff}<0} -\frac{x_i}{\Delta x_i^{aff}}\right) \tag{22}$$
$$\alpha_{aff}^{dual} = \min\left(1, \min_{i:\Delta s_i^{aff}<0} -\frac{s_i}{\Delta s_i^{aff}}\right) \tag{23}$$

3. "**Center-Corrector step**": compute centered and corrected direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$ by solving the following system:

$$\begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c \\ -\mathbf{r}_b \\ -\boldsymbol{X}\boldsymbol{S}\mathbf{e} - \Delta\boldsymbol{X}^{aff}\Delta\boldsymbol{S}^{aff}\mathbf{e} + \sigma\mu\mathbf{e} \end{bmatrix} \tag{24}$$

4. "**Update step**": after computing the search direction with the correct centering parameter and with account of correction for pure affine-scaling direction, the update of primal and dual variables is performed as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k^{prim}\Delta\mathbf{x}_k, \quad (\boldsymbol{\lambda}_{k+1}, \mathbf{s}_{k+1}) = (\boldsymbol{\lambda}_k, \mathbf{s}_k) + \alpha_k^{dual}(\Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k) \tag{25}$$

where steplengths for primal and dual variables are calculated as follows:

$$\alpha_k^{prim} = \min\left(1, \eta_k\left(\min_{i:\Delta x_i^k<0} -\frac{x_i^k}{\Delta x_i^k}\right)\right) \tag{26}$$

$$\alpha_k^{dual} = \min\left(1, \eta_k\left(\min_{i:\Delta s_i^k<0} -\frac{s_i^k}{\Delta s_i^k}\right)\right) \tag{27}$$

where $\eta_k \in [0.9, 1.0)$ is heuristic chosen so that $\eta_k \to 1$ as $k \to maxiter$. In our implementation it is implemented as:

$$\eta_k = 0.9 + 0.1 \cdot \frac{(k-1)}{maxiter} \tag{28}$$

Despite MPC brings advantage of both correcting linearization error and heuristic for choosing centering parameter, it comes at a cost of requirement to solve two non-linear systems of equations instead of one. However, due to the fact that coefficient matrix is the same in both systems, we can leverage the extra cost by factorizing the coefficient matrix and reusing it in both computations. This motivates us to improve linear algebra computations involved in our IPLP solver by implementing Cholesky factorization. This, in turn, will require not only reconsidering how Eqs.(19, 24) are solved, but also considering possible challenges associated with sparsity of these coefficient matrices as well as with inherent behavior of primal and dual variables within the context of IPLP solvers. These details are described in the next section.

Finding Starting Points

Robustness of the algorithm for solving LP with Interior Point method is largely dependent on the choice of starting point. A poor choice (x^0, λ^0, s^0) satisfying only the minimal conditions $x^0 > 0$ and $s^0 > 0$ often leads to failure of convergence. A *good starting point* should also satisfy the following conditions:

- The points should be well centered so that the pairwise products $x_i^0 s_i^0$ are similar for all $i = 1, 2, ..., n$.
- The points should not be too infeasible; that is the ratio of infeasibility to duality measure $(|(r_b^0, r_c^0)|/\mu_0)$ should not be too large.

A popular heuristic [2, 3] for finding (x^0, λ^0, s^0) starts by calculating $(\tilde{x}, \tilde{\lambda}, \tilde{s})$ as the solution of two least-squares problems:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{x} \quad \text{subject to} \quad \boldsymbol{A}\mathbf{x} = \mathbf{b} \tag{29}$$

$$\underset{\mathbf{s}}{\text{minimize}} \quad \frac{1}{2}\mathbf{s}^T\mathbf{s} \quad \text{subject to} \quad \boldsymbol{A}^T\lambda + \mathbf{s} = \mathbf{c} \tag{30}$$

The full algorithm for finding the starting point is given in Algorithm 1. We implemented it in our solver as `starting_point` function. In our implementation, we add small value to $\hat{\mathbf{x}}$ and $\hat{\mathbf{s}}$ if all the elements are found to be zero for numerical stability. The computational cost of finding (x^0, λ^0, s^0) by this scheme is about the same as one step of the primal-dual method.

Algorithm 1 Find the starting points (x^0, λ^0, s^0)

1: Solve the problems in Equation (29) and (30)

$$\tilde{\mathbf{x}} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\mathbf{b}, \quad \tilde{\lambda} = (\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\mathbf{c}, \quad \tilde{\mathbf{s}} = \mathbf{c} - \boldsymbol{A}^T\tilde{\lambda} \tag{31}$$

2: Eliminate the non-positive components of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{s}}$:

$$\delta_x = \max(-\frac{3}{2}\min_i \tilde{\mathbf{x}}_i, 0), \quad \hat{\mathbf{x}} = \tilde{\mathbf{x}} + \delta_x\mathbf{e} \tag{32}$$

$$\delta_s = \max(-\frac{3}{2}\min_i \tilde{\mathbf{s}}_i, 0), \quad \hat{\mathbf{s}} = \tilde{\mathbf{s}} + \delta_s\mathbf{e} \tag{33}$$

where $\mathbf{e} = [1, 1, ...1]^T$

3: Make sure that the components of $\mathbf{x}^0$ and $\mathbf{s}^0$ are not too close to zero and not too dissimilar:

$$\hat{\delta}_x = \frac{1}{2}\frac{\hat{\mathbf{x}}^T\hat{\mathbf{s}}}{\mathbf{e}^T\hat{\mathbf{s}}}, \quad \mathbf{x}^0 = \hat{\mathbf{x}} + \hat{\delta}_x\mathbf{e} \tag{34}$$

$$\hat{\delta}_s = \frac{1}{2}\frac{\hat{\mathbf{x}}^T\hat{\mathbf{s}}}{\mathbf{e}^T\hat{\mathbf{x}}}, \quad \mathbf{s}^0 = \hat{\mathbf{s}} + \hat{\delta}_s\mathbf{e} \tag{35}$$

4: Besides,

$$\lambda^0 = \tilde{\lambda} \tag{36}$$

4. Linear Algebra

Implementing MPC as our base algorithm brought challenges in terms of solving non-linear systems of equations. Having to compute non-linear system two times (for affine-scaling and center-corrector directions) doubles the computation time requirements if systems are solved in a straightforward fashion as they are stated in equations describing MPC algorithm. This motivated implementing Cholesky factorization, which would factorize coefficient matrices.

Implementation of Cholesky factorization removes the need to compute the inverse of coefficient matrix, which is not available for all matrices and is computationally expensive procedure. Moreover,

having computed factors once, it can be reused for both affine-scaling and center-correct steps. However, the first step was to reconsider how Eqs.(19, 24) are solved.

Normal Equations Form

Let us consider general case of Eqs.(19, 24) for estimation of direction for single step update:

$$\begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c \\ -\mathbf{r}_b \\ -\mathbf{r}_{xs} \end{bmatrix} \tag{37}$$

where $\mathbf{r}_{xs}$ can change depending on whether it is affine-scaling direction estimation or center-corrector direction estimation. This is called *unreduced form.*

By defining $\boldsymbol{D}$ such that:

$$\boldsymbol{D} = \boldsymbol{S}^{-1/2}\boldsymbol{X}^{1/2} \Rightarrow \boldsymbol{D}^2 = diag(\mathbf{x}./\mathbf{s}) \tag{38}$$

where ./ is element wise division, we can rewrite unreduced form into two more equivalent forms.

The second form is known as the *augmented system.* It is achieved by eliminating (or, more precisely, detaching) $\Delta\mathbf{s}$ from it as follows:

$$\begin{bmatrix} -\boldsymbol{D}^{-2} & \boldsymbol{A}^T \\ \boldsymbol{A} & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c + \boldsymbol{X}^{-1}\mathbf{r}_{xs} \\ -\mathbf{r}_b \end{bmatrix} \tag{39}$$

$$\Delta\mathbf{s} = -\boldsymbol{X}^{-1}\mathbf{r}_{xs} - \boldsymbol{X}^{-1}\boldsymbol{S}\Delta\mathbf{x} \tag{40}$$

The third form is achieved by eliminating $\Delta\mathbf{x}$ from the augmented system:

$$\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\Delta\boldsymbol{\lambda} = -\mathbf{r}_b - \boldsymbol{A}\boldsymbol{X}\boldsymbol{S}^{-1}\mathbf{r}_c + \boldsymbol{A}\boldsymbol{S}^{-1}\mathbf{r}_{xs} \tag{41}$$

$$\Delta\mathbf{s} = -\mathbf{r}_c - \boldsymbol{A}^T\Delta\boldsymbol{\lambda} \tag{42}$$

$$\Delta\mathbf{x} = -\boldsymbol{S}^{-1}\mathbf{r}_{xs} - \boldsymbol{X}\boldsymbol{S}^{-1}\Delta\mathbf{s} \tag{43}$$

which is known as *normal-equations form.*

In our implementation we went with common strategy of utilizing normal-equations form and using Cholesky factorization on $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$ coefficient matrix used to compute $\Delta\boldsymbol{\lambda}$. This way we avoid computing inverse of large matrices (except for $\boldsymbol{S}$, for which inverse does exist and easy to compute because it is just diagonal matrix of positive values $\mathbf{s}$) and consequently speed up overall computation.

If we define $\boldsymbol{M} = \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$, Cholesky factorization of this matrix will give us lower triangular matrix $\boldsymbol{L}$, such that:

$$\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{M} = \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T \tag{44}$$

If we define RHS used to compute $\Delta\boldsymbol{\lambda}$ as $\mathbf{r}_l = -\mathbf{r}_b - \boldsymbol{A}\boldsymbol{X}\boldsymbol{S}^{-1}\mathbf{r}_c + \boldsymbol{A}\boldsymbol{S}^{-1}\mathbf{r}_{xs}$, factors of $\boldsymbol{M}$ can then be used to solve for $\Delta\boldsymbol{\lambda}$ by forward and backward substitution as follows:

1. Solve for $\mathbf{y}$ by forward substitution:

$$\boldsymbol{L}\mathbf{y} = \mathbf{r}_l \tag{45}$$

2. And solve for $\boldsymbol{\lambda}$ by backward substitution:

$$\boldsymbol{L}^T\boldsymbol{\lambda} = \mathbf{y} \tag{46}$$

Considering that the matrix $\boldsymbol{M} = \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$ is the same for both predictor step and center-corrector step, factors obtained from Cholesky factorization can be re-utilized for both cases with modification of $\mathbf{r}_l$ to account for a change of $\mathbf{r}_{xs}$.

Modified Cholesky Factorization for IP LP Solver

Implementation of Cholesky factorization for IP LP solvers has some challenges associated with the behavior of values $\mathbf{x}$ and $\mathbf{s}$ as well as associated matrix $\boldsymbol{D}^2$. As it was stated in the theory of interior-point methods, reducing the duality measure is the way to reach convergence. However, in terms of behavior of $\mathbf{x}$ and $\mathbf{s}$, this results in very small pivot values of matrix $\boldsymbol{M}$. Consequently, Cholesky factorization fails due to ill-conditioning of the input matrix $\boldsymbol{M}$ and occurrence of very small pivot elements, especially towards the end of convergence. Hence, instead of conventional Cholesky factorization, we implemented two versions of modified Cholesky.

First modified Cholesky (shown in Alg. 2) factorization was designed around Julia in-built Cholesky factorization with the main motivation being to utilize inherent speed of in-built functions. This modified Cholesky is used for two types of problems:

1. For small well-conditioned positive definite matrices which can be factorized with in-built Cholesky without any need for modifications.
2. For slightly ill-conditioned almost positive definite matrices, which might have one or two slightly negative diagonal elements. We fix such matrices by adding matrix $\tau \cdot \boldsymbol{I}$, where τ is the small correction factor used to rectify negative diagonal elements of original (not positive definite) matrix $\boldsymbol{M}$.

Experimental results showed that this modified Cholesky factorization is beneficial to have on small and medium size matrices as well as sometimes on large problems at the initial couple of iterations.

Second modified Cholesky (shown in Alg. 3) factorization was designed specifically to address small pivots arising as a result of convergence of IPLP solver. It utilizes skipping strategy presented in [1] and used by industry solvers, such as LIPSOL and PCx. The idea is to dynamically during factorization process skip factorization step when small pivot is encountered. We implemented slightly different version of this strategy, by replacing small pivot by huge numerical value and setting all off-diagonal elements of that column to zero. This also requires keeping track of such manipulations during factorization and zeroing out corresponding elements of $\Delta\boldsymbol{\lambda}$ vector, which we try to compute using factors of $\boldsymbol{M}$. Experiments show that this strategy works effectively on large scope of problems without too much overhead caused by such manipulations. It should be noted that these manipulations result in factors with huge values on the diagonal elements, which in turn results in large factorization residual. However, this is cancelled out by zeroing out corresponding elements of $\Delta\boldsymbol{\lambda}$ vector.

In addition, to leverage sparsity of coefficient matrix $\boldsymbol{M}$ we add the feature of computing approximate minimum degree permutations. This results in permutation of matrix $\boldsymbol{M}$ with the most sparsity and allows to gain extra benefit when performing Cholesky factorization. This is implemented in our solver by the help of Julia AMD package.

Algorithm 2 Modified Cholesky 1 (`modchol1`) of matrix M

Input: Coefficient matrix, $\boldsymbol{M} \in \mathbb{R}^{m \times m}$.
Let, $\beta = 1.0e-3$ be some small value
if $\min_i m_{ii} > 0$ **then**
    set $\tau_0 \leftarrow 0$
else
    $\tau_0 = \min(a_{ii}) + \beta$
end if
for k = 1 . . . 1000 **do**
    $\boldsymbol{M} \leftarrow \boldsymbol{M} + \tau_k \boldsymbol{I}$
    if $\boldsymbol{M}$ *positive definite* **then**
        $\boldsymbol{L} = Cholesky(\boldsymbol{M})$ ▷ Julia in-built Cholesky function
        return $\boldsymbol{L}$
    else
        $\tau_{k+1} \leftarrow \max(2\tau_k, \beta)$
    end if
end for

Algorithm 3 Modified Cholesky 2 (`modchol2`) of matrix M

Input: Coefficient matrix, $\boldsymbol{M} \in \mathbb{R}^{m \times m}$.
Let, $\beta = 1.0e-8$ be some small value
Initialize $zero\_out = ones(m)$
for i = 1 . . . m **do**
    if $m_{i,i} < \beta$ **then** ▷ Skip factorization with this pivot
        $m_{i,i} = 1.0e64$
        $m_{i+1:m,i} = 0.0$
        record $zero\_out_i = 0$
    else
        $m_{i,i} = sqrt(m_{i,i})$
        $m_{i+1:m,i} = m_{i+1:m,i}/m_{i,i}$
        $m_{i+1:m,i+1:m} = m_{i+1:m,i+1:m} - m_{i+1:m,i} * m_{i+1:m,i}^T$
    end if
end for
return $\boldsymbol{L} = sparse(LowerTriangular(\boldsymbol{M}))$ and $zero\_out$

5. Preprocessing of LP Problems

The purpose of preprocessing [4] is to transform a given linear program into the standard form and get rid of some redundancy from the formulation before generating starting point and solving it. In this process we also check obvious (trivial) infeasibility. We perform the following preprocessing steps:

1. **Trivial infeasibility**: For given lower bound vector, lo and upper bound vector, hi, we check if

$$\exists j : \quad lo_j > hi_j, \tag{47}$$

then the problem is trivially infeasible.

2. **Remove empty rows**: We check and remove the empty rows form the constraint matrix. At the same time we check if all the corresponding entries in b are also zero or not. If we find any empty row, i, with non-zero b_i than we declare the problem as (trivially) infeasible.

3. **Remove dependent rows**: Since our starting point estimation involves solution of Equation (31), we want the constraint matrix, $\boldsymbol{A}$, to be full rank so that AA^T is invertible. For that, after removing the empty rows we check if the matrix is full rank or not. If the matrix is not full rank, we detect and remove the dependent rows from the constraint matrix, $\boldsymbol{A}$. For finding linearly dependent rows, we used the algorithm proposed by Andersen [5]. The pseudo-code of the algorithm is given in Algorithm 4. We later figured out that, Cholesky decomposition on $\boldsymbol{A}$ can be used to find out rows with extremely small pivots. Those rows are dependent rows and can be removed from $\boldsymbol{A}$. Thus, Cholesky decomposition can be also be as a substitute for Andersen's method for finding linearly dependent rows. We perform empty and dependent row removal by `remove_redundancy_sparse` function in our code.

Algorithm 4 Find linearly dependent rows in A

Input: Constraint matrix, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
Let, $\boldsymbol{I} \in \mathbb{R}^{m \times m}$ and augment $\boldsymbol{A}$ as

$$\boldsymbol{A} = [\boldsymbol{I}, \boldsymbol{A}]$$

Let, $\beta = \nu$, where ν contain the indices of the artificial columns. β contain the indices of the basis columns and $\boldsymbol{B}$ is the corresponding matrix. e_i is the i^{th} column of the identity matrix.
for each $i \in (\beta \cup \nu)$ **do**
    $\pi = \mathbf{e}_i^T \boldsymbol{B}^{-1}$
    if $\exists j : |\pi^T a_{.j}| > 0 \quad j \notin \beta \cup \nu$ **then**
        $\beta = \beta \cup \{j\} \setminus \{i\}$
    else
        The i^{th} row is dependent
    end if
end for

4. **Detect and substitute fixed variable:** A variable (x_j) is called fixed variable if its corresponding lower bound and upper bound are equal ($lo_j = hi_j$). We detect and substitute such variable out of the problem.

5. **Remove empty column:** We also detect if a column, j, is empty or not. If such columns exist, we remove them from the problem. Dependent on the bounds on corresponding variable x_j and its objective coefficient c_j, variable x_j is fixed at one of its bounds. Otherwise we detect the problem as unbounded. We use `preprocess_LP` function in our code to find fixed variables and empty columns.

6. **Detect free variable:** A variable (x_j) is called free variable if it does not have any constraints ($-\infty \leq x_j \leq \infty$). We represent such variable as difference of two nonnegative variables ($x_j = x_j^+ - x_j-$) and substitute back in the problem to comply with the standard form of LP. In our code we implement this in `free_variable` function.

7. **Conversion to standard form:** Many practical problems are not presented in standard form in Equation (1) but in the following way, instead,

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{x} = \mathbf{b} \\ & lo \leq \mathbf{x} \leq hi \end{aligned} \tag{48}$$

Thus after removing the redundancies form the problem by following all the preprocessing steps (1-6) we finally convert this problem into standard form. We first replace $\mathbf{x}$ with

$$\mathbf{x} = \mathbf{y} + lo \tag{49}$$

to shift the nonzero lower bounds to zero. This changes the problem to

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{y} + \mathbf{c}^T lo \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{y} = \mathbf{b} - \boldsymbol{A} * lo \\ & 0 \leq \mathbf{y} \leq hi - lo \end{aligned} \tag{50}$$

Since $\mathbf{c}^T lo$ is constant it does not affect the optimum solution points, hence can be kept out of the optimization and added to the optimum objective value at the end of the optimization as a postsolving step. Now, let's take a variable $\mathbf{z}$ such that $\mathbf{z} \geq 0$ and

$$\mathbf{y} + \mathbf{z} = hi - lo \tag{51}$$

In our implementation, we only add this slack z to the inequalities that has finite upper bounds. Using this equality constraint we can rewrite our LP objective with constraints as follows:

$$\begin{aligned} \underset{\mathbf{x}_s}{\text{minimize}} \quad & \mathbf{c}_s^T\mathbf{x}_s \\ \text{subject to} \quad & \boldsymbol{A}_s\mathbf{x}_s = \mathbf{b}_s \\ & \mathbf{x}_s \geq 0 \end{aligned} \tag{52}$$

Where,

$$\mathbf{x}_s = \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}, \mathbf{c}_s = \begin{bmatrix} \mathbf{c} \\ 0 \end{bmatrix}, \boldsymbol{A}_s = \begin{bmatrix} \boldsymbol{A} & 0 \\ \boldsymbol{I} & \boldsymbol{I} \end{bmatrix}, \mathbf{b}_s = \begin{bmatrix} \mathbf{b} - \boldsymbol{A} * lo \\ hi - lo \end{bmatrix}$$

In our code this conversion is performed in `convert_to_standard` function. After this conversion step, we check if the $\boldsymbol{A}_s$ is full rank or not. If not, there is redundancy in rows which is a by product of the intermediate preprocessing steps (4-5). We call the `remove_redundancy_sparse` function one last time to remove such dependency.

6. Infeasibility Detection

For a given linear program, primal-dual solution might not exist. We want to detect such problems as infeasible problems. The class of algorithms that uses strictly feasible starting point, from the analysis or existence of such starting points feasibility/infeasibility of the linear program can be detected. Since we have used Mehrotra's infeasible-interior point framework, where starting point is not strictly feasible, we can not decide on the feasibility of the problem from starting point analysis.

However, we know from the convergence theory that when a solution exists, this algorithms will find it [1]. When presented with an infeasible problem, method such as MPC are unable to reduce the residuals below a certain (nonzero) level, so the iterates $(\mathbf{x}^k, \mathbf{s}^k)$ diverge: their norm $||(\mathbf{x}^k, \mathbf{s}^k)||$ approaches ∞. Formally, simple termination test for such algorithms described in [1] as: For some positive constants ϵ (small) and ω (large), terminate the algorithm if

$$\mu_k \leq \epsilon \quad \text{or} \quad ||\mathbf{x}^k, \mathbf{s}^k||_1 \geq \omega. \tag{53}$$

One of the two conditions in (53) eventually holds for some k sufficiently large (Theorem 9.7 in [1]), so the algorithm is guaranteed to terminate after a finite number of iterations. If termination happens because of the first condition in (53), we are close to an approximate solution. If termination occurs because of the second condition, chances are that the set of strictly feasible point for the problem is empty (hence the problem might be infeasible). From the discussion in [1], we find that, ω depends on the initial norm $||(\mathbf{x}^0, \mathbf{s}^0)|| = \omega^0$. We use this relationship in our heuristic method for setting the value of ω for detecting infeasibility.

We also notice that, $||(\mathbf{x}^k, \mathbf{s}^k)||$ of a feasible problem might diverge due to instability of computation (as discussed in section 4). We detect such cases by checking whether the duality measure is less than the given tolerance or not ($\mu_k \leq \epsilon$). In case of moderate norm blow-up due to such instability issues, if the duality measure is still under the given tolerance, we detect such problems as feasible. Combining all these ideas, we propose a heuristic for detecting infeasibility below :

$$\text{Condition 1:} \quad 10e^3\omega^0 < \omega^k \leq 10e^9\omega^0 \quad \text{and} \quad \mu_k > \epsilon \tag{54}$$

$$\text{Condition 2:} \quad \omega^k > 10e^9\omega^0 \tag{55}$$

Condition 1 is detecting infeasibility for moderate blow-up but with diverging duality. Whereas, Condition 2 is detecting infeasibility due to high blow-up. Both this conditions are valid for large enough k. In our implementation we check these conditions after maximum number of iterations has been exhausted. To detect infeasibility before maximum iteration reached, we also include a 'greedy' criteria that checks:

$$\text{Condition 3:} \quad \omega^k > 10e^6\omega^0 \quad \text{and} \quad \mu_k > \epsilon \tag{56}$$

after each iteration and if satisfied terminates the solver and report the problem as infeasible. In practice, for Netlib problems, our method detected all the infeasible LP problems correctly except `lpi_cplex2` (which is almost a feasible problem and difficult to detect). Thus our method of finding infeasibility works well in practice and robust.

7. Stopping and Postsolving LP

Convergence Criterion

For solving LP problems our implementation requires three user inputs. The LP problem itself, the tolerance (tol) and maximum number of allowable iterations (maxiter). Based on the tolerance limit set by the user, we check for the convergence of our algorithm by using the following two conditions:

$$\text{Duality Measure:} \quad \frac{(\mathbf{x}_s * \mathbf{s})}{n} \leq tol \tag{57}$$

$$\text{Norm Residual: } \frac{\text{norm}([\boldsymbol{A}_s * \lambda + \mathbf{s} - \mathbf{c}_s; \boldsymbol{A}_s * \mathbf{x}_s - \mathbf{b}_s; \mathbf{x}_s. * \mathbf{s}])}{\text{norm}([\mathbf{b}_s; \mathbf{c}_s])} \leq tol \tag{58}$$

If both of these conditions are satisfied then we terminate the solver and return the solutions. If not, we terminate after maximum number of iterations specified by the user. User can check the status of the solution (whether it converged or not) by checking if `soln.flag` is `true` or `false`.

Postsolving

Since we preprocess the problem before solving, we need to postprocess the solution after the MPCsolver returns a solution (x_s) to get back x of the original problem before preprocessing and obtain the final objective value. We store our presolved variables in `presolved` which is of `IplpPresolved` data type. Our postsolving steps involves: (i) undoing the lower bound shift in (49), (ii) substitution of the original free variables, (iii) inserting back the presolved fixed variables and variables corresponding to the empty columns from `presolved.xpsol` in correct order. These steps ensures that that we get back the solution in original x format which can be checked by `soln.x`, where `soln` is the final returned solution to the user of type `Iplpsolution`. We also correct the standard solution, x_s from the solver by undoing the lower bound shift in (49) and augmenting the presolved solutions (`presolved.xpsol`) and return xs. We also return cs with the corresponding changes in c_s. User can get the objective value returned from our solver after convergence by any one of the following equations:

$$\text{objective value} = (\texttt{soln.cs})^T(\texttt{soln.xs}) \tag{59}$$

$$\text{objective value} = (\texttt{problem.c})^T(\texttt{soln.x}) \tag{60}$$

We used `get_original_x` and `get_standard_xs_cs` functions during postsolving.

8. Results and Discussions

To assess the robustness of our implementation, we have tested LP problems from Netlib [6]. For specified tolerance of $1e^{-08}$ and maximum iterations of 100, we have shown convergence for 95 problems. The results are shown in Table (1,2). In the tables we report the 'optimal value' of the LP problem returned by our solver after convergence. We also specify number of iterations required for convergence in 'Iter' column. In Figure (1), for nine example Netlib LP problems, we have plotted how objective values, duality measure and norm residual vary as we converge towards the solutions.

The problems that we tested has varying degree of complexity. In general, convergence of moderate size problems were very fast. Even for some of the biggest problems such as `lp_osa_60`, `lp_truss` etc. we observed fast convergence. This is primarily due to that fact that these problem were well-conditioned hence our linear algebra routine involving Cholesky decomposition was very efficient in computation resulting in fast convergence. In Table (2) we listed results for some of the largest LP problems from Netlib. Though these problems took longer for convergence, we got accurate solution (compared with reference objective value after solution from Netlib and LIPSOL [7]). Accurate results after such extensive testing thus validate the robustness of our implementation.

We want to emphasize the necessity of preprocessing steps before solution. The problem, such as `lp_brandy, lp_tuff` etc. has zero rows. Thus without removal of the zero rows we could not even get the starting points. Also, `lp_cre_b` has dependent columns in addition to zero rows which

needed to be removed as well. Moreover, for `lp_tuff` our preprocessing steps detected free variables, fixed variables and number of constraints to add to the original problem for efficient solution in standard from. Thus, in our opinion, preprocessing is absolutely necessary to deal with problems of various structure and complexity.

However, there is scope for improving the speed of the implementation. For example, `lp_fit2p` has dense columns. Efficient handling of dense columns before Cholesky decomposition will achieve significant speed-up in convergence [7]. Since our main focus was to implement a robust solver we did not include such modifications for problems with dense columns.

Moreover, few LP problems such as `lp_cycle, lp_capri, lp_perold, lp_greenbea` did not converge for the given tolerance of $1e^{-08}$. However, for lower tolerance value ($1e^{-06}$) they all converged with small error in solution. For `lp_pilot4, lp_pilot_ja, lp_pilot_we, lp_greenbeb` problems we did not achieved convergence even after lowering the tolerance. However, for these problems we observe that, objective value after reaching maximum iterations remain close to expected values but do not exactly match them possibly due to structure of the problem resulting in numerical instability.

As discussed in section 6, we have included necessary steps in our algorithm for detecting infeasible LP problems. We have tested all the infeasible LP problems from Netlib. For the conditions that we use for infeasibility detection (54-56), we detected all the infeasible problems correctly except `lpi_cplex2` (as mentioned in [6], this is an almost feasible problem and difficult to detect, thus require more sophisticated technique). We observe that, different problem satisfies different infeasibility conditions. For example, `lpi_chemcom` satisfied infeasibility Condition 3 (56) within 7 iterations whereas `lpi_pilot4i` satisfied infeasibility Condition 1 (54) after reaching maximum iterations. For large infeasible problems such as `lpi_bgindy`, early detection of infeasibility using Condition 3 saved a lot of computations.

References

[1] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, USA, 1997.

[2] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[3] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 11 1992.

[4] Erling D. Andersen and Knud D. Andersen. Presolving in linear programming. Technical report, 1993.

[5] Erling D. Andersen. Finding all linearly dependent rows in large-scale linear programming. *Optimization Methods and Software*, 6(3):219–227, 1995.

[6] UF Sparse Matrix Collection. https://www.cise.ufl.edu/research/sparse/matrices/LPnetlib/. Accessed: 2020-05-04.

[7] Yin Zhang. Solving large-scale linear programs by interior-point methods under the matlab environment. *Optimization Methods and Software*, 10(1):1–31, 1998.

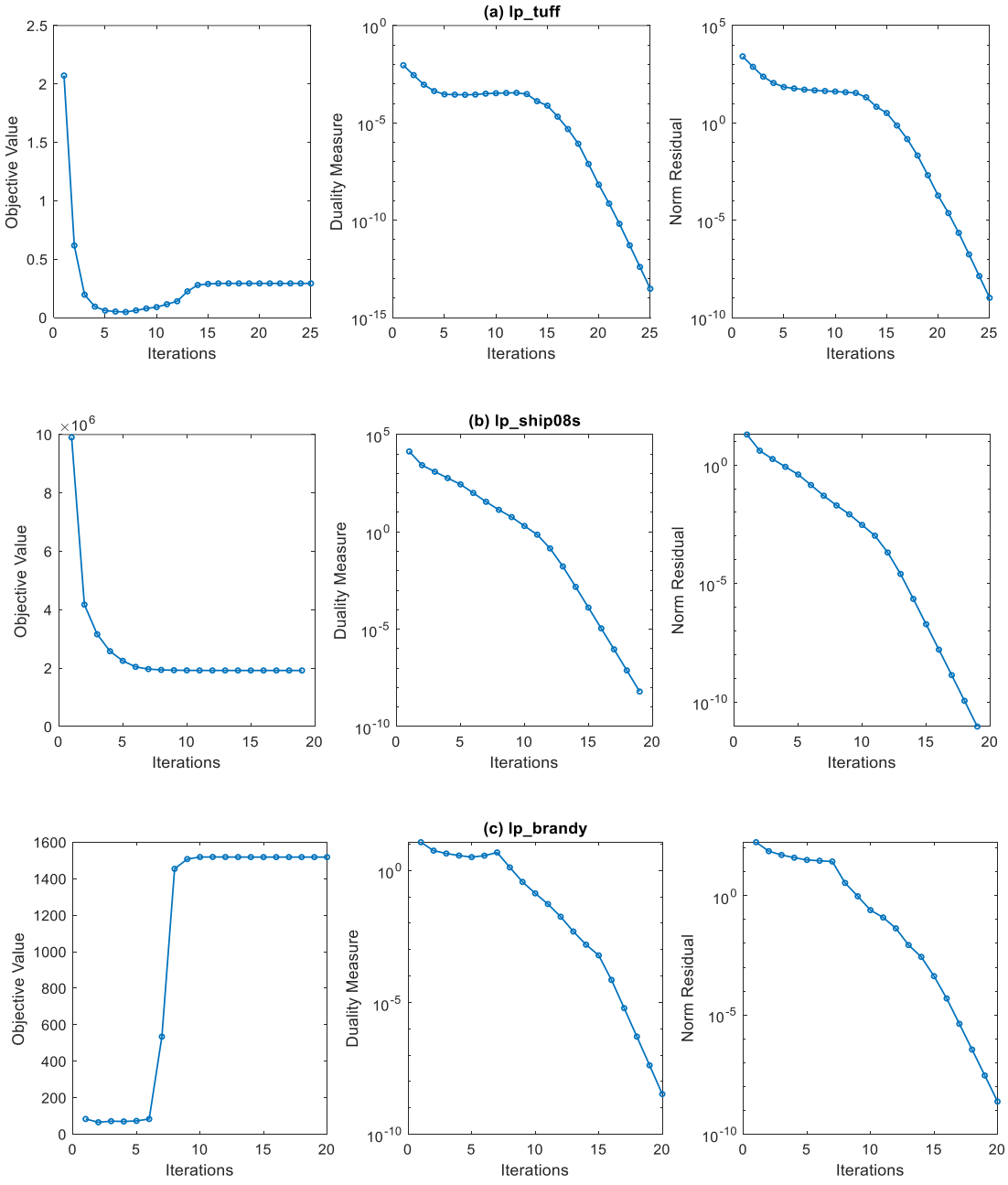
	Test Results (Tolerance: 1e-08)					
Problem	Rows	Cols	Iter	Duality Measure	Norm Residual	Optimal value
25fv47	821	1876	30	7.33e-10	7.19e-12	5.5018458890e+03
80bau3b	2262	12061	43	3.82e-09	1.56e-12	9.8722419242e+05
tuff	333	628	25	3.01e-14	1.03e-09	2.9214776510e-01
brandy	221	249	20	3.33e-09	2.35e-09	1.5185098967e+03
agg	489	163	37	1.68e-09	2.36e-13	-3.5991767287e+07
fit1d	24	1049	23	8.98e-10	1.45e-11	-9.1463780916e+03
fit1p	627	1677	24	1.08e-09	2.69e-11	9.1463780942e+03
ship08s	778	2467	19	6.27e-09	9.48e-12	1.9200982079e+06
adlittle	56	138	18	1.31e-09	1.54e-12	2.2549496316e+05
bandm	306	472	19	7.54e-09	1.47e-09	-1.5862801791e+02
afiro	27	51	13	8.67e-09	8.74e-11	-4.6475314268e+02
blend	74	114	13	5.19e-09	4.02e-09	-3.0812149779e+01
scfxm2	660	1200	24	8.56e-10	1.31e-10	3.6660261566e+04
ganges	1309	1706	23	1.54e-09	1.63e-11	-1.0958573613e+05
scsd8	397	2750	14	1.20e-09	5.73e-10	9.0500000214e+02
agg2	516	758	29	7.02e-09	7.39e-14	-2.0239252356e+07
agg3	516	758	28	8.44e-09	8.94e-14	1.0312115935e+07
beaconfd	173	295	13	2.94e-09	8.90e-10	3.3592485808e+04
bnl1	643	1586	35	1.83e-09	3.35e-11	1.9776295624e+03
bnl2	2324	4486	38	2.99e-09	2.31e-10	1.9776295624e+03
pilot	1441	4860	38	3.33e-09	3.01e-11	-5.5748972166e+02
bore3d	233	334	22	3.25e-09	2.36e-10	1.3730803944e+03
cre_a	3516	7248	34	3.03e-09	5.32e-13	2.3595407061e+07
cre_c	3068	6411	35	3.58e-09	1.33e-12	2.5275116141e+07
czprob	929	3562	37	4.59e-09	3.61e-11	2.1851966989e+06
d2q06c	2171	5831	33	2.20e-09	7.05e-10	1.2278421082e+05
d6cube	415	6184	23	8.90e-10	1.08e-11	3.1549167198e+02
e226	223	472	22	2.25e-09	6.21e-10	-1.8751928737e+01
etamacro	400	816	32	4.46e-09	1.38e-11	-7.5571523104e+02
fffff800	524	1028	53	1.22e-09	1.05e-12	5.5567956482e+05
finnis	497	1064	27	2.41e-09	5.04e-12	1.7279106559e+05
gfrd_pnc	616	1160	24	7.75e-10	3.80e-14	6.9022359995e+06
grow15	300	645	21	1.17e-09	4.12e-13	-1.0687094129e+08
grow22	440	946	21	3.59e-09	3.11e-12	-1.6083433648e+08
grow7	140	301	19	4.30e-09	2.75e-14	-4.7787811815e+07
israel	174	316	29	1.72e-09	5.45e-14	-8.9664482186e+05
kb2	43	68	26	4.47e-09	2.03e-10	-1.7499001298e+03
ken_07	2426	3602	22	2.08e-09	3.63e-13	-6.7952044338e+08
lotfi	153	366	17	6.19e-09	2.28e-09	-2.5264704769e+01
maros	846	1966	30	6.06e-09	3.76e-10	-5.8063743696e+04
fit2d	25	10524	25	4.30e-09	4.75e-11	-6.8464293252e+04
modszk1	687	1620	26	1.83e-09	1.68e-10	3.2061965548e+02
pds_02	2953	7716	33	3.91e-09	5.13e-13	2.8857862010e+10
vtp_base	198	346	50	5.23e-10	7.22e-13	1.2983146246e+05

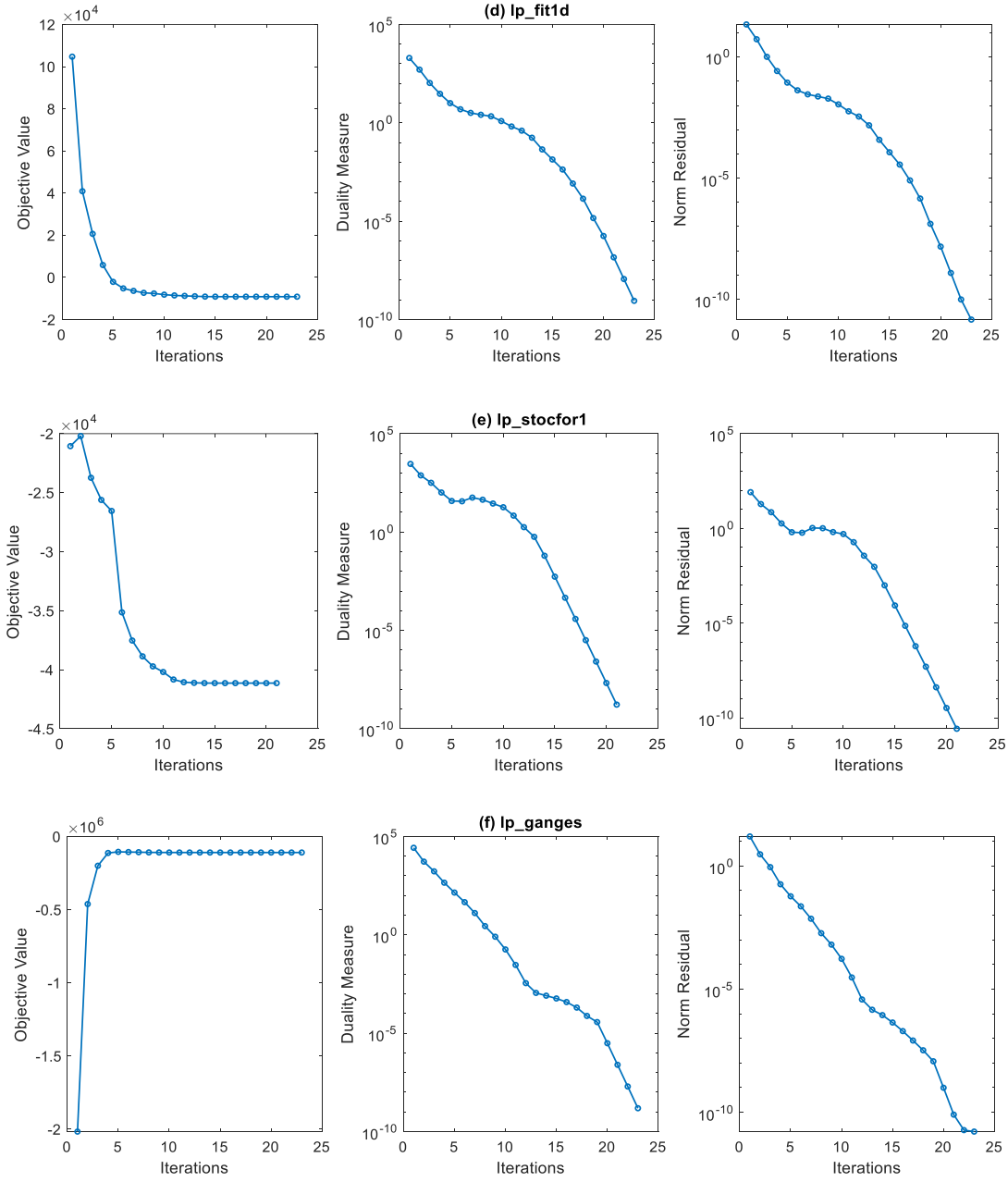
	Test Results (Tolerance: 1e-08)					
Problem	Rows	Cols	Iter	Duality Measure	Norm Residual	Optimal value
osa_07	1118	25067	38	3.64e-09	1.56e-12	5.3572251668e+05
osa_14	2337	54797	52	5.10e-10	2.49e-13	1.1064628444e+06
osa_30	4350	104374	53	5.27e-10	6.85e-13	2.1421398732e+06
osa_60	10280	243246	59	9.74e-09	9.48e-11	4.0440724739e+06
pilotnov	975	2446	26	6.52e-10	7.71e-09	-4.4972761880e+03
recipe	91	204	15	1.75e-09	2.51e-10	-2.6661599992e+02
sc105	105	163	13	1.17e-09	8.43e-10	-5.2202061189e+01
sc205	205	317	15	6.94e-10	1.16e-09	-5.2202061143e+01
sc50a	50	78	12	4.71e-09	1.83e-10	-6.4575076970e+01
sc50b	50	78	12	1.44e-09	1.12e-10	-6.9999999961e+01
scagr25	471	671	23	3.45e-09	2.20e-12	-1.4753433061e+07
scagr7	129	185	19	8.31e-09	5.41e-12	-2.3313898243e+06
scfxm1	330	600	22	4.22e-09	8.77e-11	1.8416759029e+04
scfxm3	990	1800	24	1.10e-09	4.88e-10	5.4901254551e+04
scorpion	388	466	16	1.05e-09	2.98e-11	1.8781248229e+03
scrs8	490	1275	23	3.31e-09	1.25e-11	9.0429695722e+02
scsd1	77	760	11	8.81e-09	4.02e-09	8.6666729141e+00
scsd6	147	1350	13	2.14e-09	9.82e-10	5.0500002424e+01
share1b	117	253	26	3.38e-09	1.78e-10	-7.6589318579e+04
share2b	96	162	17	4.97e-10	3.56e-09	-4.1573224073e+02
ship04l	402	2166	17	8.90e-09	1.87e-11	1.7933245382e+06
ship04s	402	1506	18	3.79e-09	6.41e-12	1.7987147005e+06
ship08l	778	4363	19	8.83e-09	2.06e-11	1.9090552288e+06
ship12l	1151	5533	21	1.05e-09	9.20e-12	1.4701879191e+06
ship12s	1151	2869	21	1.37e-09	5.69e-12	1.4892361341e+06
stair	356	614	26	3.95e-09	2.93e-09	-2.5126695057e+02
standata	359	1274	22	2.24e-09	2.06e-09	1.2576994996e+03
standgub	361	1383	23	7.33e-09	6.59e-09	1.2576994986e+03
standmps	467	1274	28	9.41e-09	6.24e-10	1.4060175001e+03
stocfor1	117	165	21	1.69e-09	2.72e-11	-4.1131976219e+04
stocfor2	2157	3045	26	2.24e-09	6.53e-10	-3.9024408536e+04
truss	1000	8806	22	4.44e-09	2.86e-10	4.5881584676e+05
wood1p	244	2595	36	2.19e-09	4.01e-09	1.4429024116e+00
woodw	1098	8418	45	2.12e-14	3.37e-09	1.3044763332e+00
degen2	444	757	16	1.43e-09	1.92e-10	-1.4351779998e+03
degen3	1503	2604	19	8.63e-10	3.80e-10	-9.8729399916e+02
sctap1	300	660	19	7.28e-09	6.61e-10	1.4122500020e+03
sctap2	1090	2500	17	8.85e-09	5.69e-10	1.7248071566e+03
sctap3	1480	3340	17	8.95e-09	6.49e-10	1.4240000238e+03
shell	536	1777	28	7.38e-09	6.65e-13	1.2088253460e+09
sierra	1227	2735	24	4.58e-09	4.60e-13	1.5394362184e+07
qap12	3192	8856	20	1.86e-09	4.27e-09	5.2289430293e+02
qap8	912	1632	12	1.26e-09	1.17e-09	2.0350000025e+02

Table 1: Test Results for LP problems solved with our IPLPSolver form NetLib.

	Test Results (Tolerance: 1e-08)					
Problem	Rows	Cols	Iter	Duality Measure	Norm Residual	Optimal value
cre_b	9648	77137	48	1.15e-09	2.06e-13	2.3129639887e+07
cre_d	8926	73948	50	4.45e-09	1.12e-12	2.4454969765e+07
dfl001	6071	12230	55	5.47e-09	1.87e-12	1.1266395582e+07
fit2p	3000	13525	23	1.27e-09	1.81e-11	6.8464293304e+04
maros_rt	3136	9408	19	7.88e-09	2.11e-12	1.4971851665e+06
pilot87	2030	6680	42	3.33e-09	4.47e-11	3.0171037101e+02
stocfor3	16675	23541	34	4.26e-09	5.93e-09	-3.9976783915e+04
qap15	6330	22275	23	1.70e-09	5.27e-09	1.0409951055e+03

Table 2: Test results for large LP problems.


(a) lp_tuff
Objective Value
Duality Measure
Norm Residual
Iterations
(b) lp_ship08s
×10^6
Objective Value
Duality Measure
Norm Residual
Iterations
(c) lp_brandy
Objective Value
Duality Measure
Norm Residual
Iterations


(d) lp_fit1d
(e) lp_stocfor1
(f) lp_ganges
Objective Value
Duality Measure
Norm Residual
Iterations
×10^4
×10^6

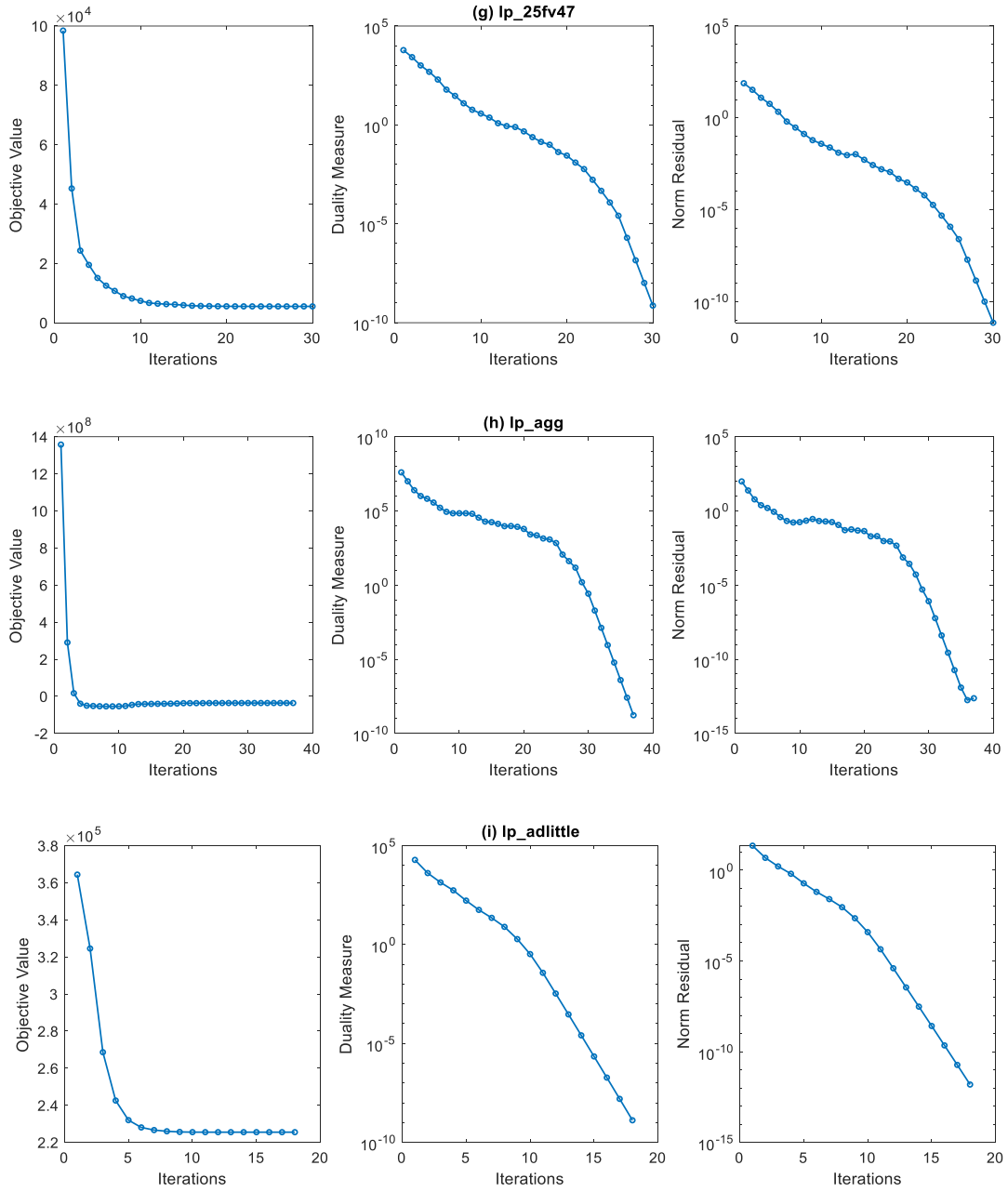


Figure 1: Illustration of the convergence of our IPLPsolver to optimum objective values for different the LP problems (from Netlib).