CS 520 – Final Report

Junxing Shi, Priyank Jain
shi69@purdue.edu, jain206@purdue.edu

April 28, 2017

1. Introduction

We define a general class of linear programs we set out to solve. A linear program can be represented in *general* form as,

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \quad ,$$

where $\mathbf{c}$ is the objective function, $\mathbf{x}$ is the primal variable with a box constraint, i.e. lower bound $\mathbf{l}$ and upper bound $\mathbf{u}$, and $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ constitutes a system of equality constraints. Given such a problem, we always first convert the problem into its *standard* form and simplify the problem via a procedure called **presolving**, and thereafter solve it with a linear programming **solver**. Usually, a **postsolver** is also used to revert the presolved steps. A linear programing solver typically solves the problem in its standard form given by,

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

where now $\mathbf{x}$ absorbs the slack variables that serve to remove the box constraints, $\mathbf{b}$ absorbs the lower and upper bound $\mathbf{l}$ and $\mathbf{u}$, and $\mathbf{c}$ and $\boldsymbol{A}$ are also augmented accordingly. The corresponding dual problem for this standard primal problem is:

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

The vector $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{s}^*)$ is a primal-dual solution for (1), (2) if it satisfies the following KKT conditions:

$$\boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} = \mathbf{c} \tag{3a}$$
$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \tag{3b}$$
$$\mathbf{x}^T\mathbf{s} = 0 \tag{3c}$$
$$\mathbf{x}, \mathbf{s} \geq 0 \tag{3d}$$
(3)

In this report, we documented a software package we developed to tackle the linear programming problems as described above. The package is made to solve such linear programs as robustly as possible, and delivered for further tests with the requested interface. In the following sections, we will describe a series of procedures/methods we implemented to actuate the package.

2. Background of approaches

There are many different approaches to creating an interior point solver (not necessarily distinct), some of them are listed as follows:

1. Potential Reduction Methods: These methods make use of a logarithmic potential function of the form.

$$\rho \log(\mathbf{c}^T\mathbf{x} - Z) - \sum_{i=1}^{n} \log(x_i)$$

where $\rho = n + 1$, n being the length of vector $\mathbf{x}$ and Z is a lower bound on the optimal objective value. Karmarkar [1] proved convergence and complexity results by showing that this function is decreased by at least a constant at each step.

2. Path-following algorithms: We define a central path $\mathcal{C}$, a path of points $(\mathbf{x}_\tau, \boldsymbol{\lambda}_\tau, \mathbf{s}_\tau)$, that leads to the set Ω of primal-dual solutions. Points in Ω satisfy the KKT conditions (3), where points in $\mathcal{C}$ are defined by conditions that differ from the KKT conditions only by the presence of a positive parameter $\tau > 0$, namely,

$$\begin{aligned} \boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} &= \mathbf{c} \\ \boldsymbol{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x}^T\mathbf{s} &= \tau \\ \mathbf{x}, \mathbf{s} &\geq 0 \end{aligned}$$

We introduce a centering parameter $\sigma \in [0, 1]$ and a duality measure μ defined by

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i s_i = \frac{\mathbf{x}^T\mathbf{s}}{n}$$

which measures the average value of the pairwise products $x_i s_i$. The generic step equations are then

$$\begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\boldsymbol{X}\boldsymbol{S}\mathbf{e} + \sigma\mu\mathbf{e} \end{bmatrix} \tag{4}$$

The step $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$ is a Newton step toward the point $(\mathbf{x}_{\sigma\mu}, \boldsymbol{\lambda}_{\sigma\mu}, \mathbf{s}_{\sigma\mu}) \in \mathcal{C}$ at which the pairwise products $x_i s_i$ are all equal to $\sigma\mu$

Path-following methods follow $\mathcal{C}$ in the direction of decreasing τ to the solution set Ω. They do not necessarily stay exactly on $\mathcal{C}$ or even particularly close to it. Rather, they stay within a loose but well-defined neighborhood of C while steadily reducing the duality measure μ to zero. Each search direction is a Newton step toward a point on $\mathcal{C}$, a point for which the duality target measure

τ is equal to or smaller than the current duality measure μ. The target value $\tau = \sigma\mu$ is used. There are three variants of Path following algorithms:

(a) Algorithm SPF: The short-step path-following algorithm chooses a constant value $\sigma_k = \sigma$ for the centering parameter and fixes the step length at $\alpha_k = 1$ for all iterations k.

(b) Algorithm PC: The predictor-corrector algorithm alternates between two types of steps: predictor steps, which improve the value of μ but which also tend to worsen the centrality measure given by

$$\frac{1}{\mu}||\boldsymbol{X}\boldsymbol{S}\mathbf{e} - \mu\mathbf{e}||$$

and corrector steps, which have no effect on the duality measure μ but improve centrality.

(c) Algorithm LPF: The long path-following algorithm makes more aggressive (smaller) choices of centering parameter σ than does Algorithm SPF. Instead of taking unit steps, Algorithm LPF performs a line search along the Newton direction.

3. Infeasible Interior Point Algorithms: Path-following algorithms and potential-reduction methods start from a strictly feasible point $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ in some neighborhood of the central path. Often, it is not easy to find a starting point that satisfies this conditions. One way to avoid this difficulty is to embed the given linear program in a slightly larger problem for which a strictly feasible point is easy to identify. This is termed as homogeneous self-dual reformulation and is a particularly useful embedding tool. However, for the purpose of this project we din't focus much on this tool except than reading the theory behind it.

Algorithm IPF: Infeasible interior point algorithm does not require the initial point to be strictly feasible but only requires that its x and s components be strictly positive. This algorithm and the details of its implementation are the focus of the remaining part of the report.

3. The Infeasible Interior Point Solver

To solve a linear program in its *standard* form, we adopted a class of methods called the infeasible primal-dual interior point method. The method proposes to solve the primal and dual problems simultaneously, i.e. gradually reducing the duality gap and residuals of the KKT conditions. It allows updates of infeasible points that only accept the box constraints but potentially violates the equality constraints. The algorithm declares success when the solution point closes the residuals of all the constraints, i.e. KKT conditions are satisfied, within a certain tolerance level. Central to the method is the idea of the logarithmic barrier function. Similar to the primal barrier method, the updates follow a central path, gradually towards the optimal solution.

To illustrate the method, we look at the dual formulation of the exact same linear program, given by (2) where $\boldsymbol{\lambda}$ is the dual variable for the quality constraints, and $\mathbf{s}$ is the complimentary dual for the inequality constraints. The KKT conditions can then be easily derived, as given in (3) The primal-dual interior point method effectively optimize the solution until all the KKT conditions are satisfied within a small tolerance level. However, at every update the equality constraints are not guaranteed. The update direction at every step is computed by solving a system of Newton

equations to reduce the residuals of the KKT conditions,

$$\begin{bmatrix} \boldsymbol{A} & 0 & 0 \\ 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} \tag{5}$$

and

$$\begin{aligned} \mathbf{r}_1 &= \mathbf{b} - \boldsymbol{A}\mathbf{x} \\ \mathbf{r}_2 &= \mathbf{c} - \boldsymbol{A}^T\boldsymbol{\lambda} - \mathbf{s} \\ \mathbf{r}_3 &= \sigma\mu\mathbf{e} - \boldsymbol{X}\boldsymbol{S}\mathbf{e} \\ \mu &= \mathbf{s}^T\mathbf{x}/n, \end{aligned} \tag{6}$$

where μ is a hyper-parameter (duality measure) used to center the path, effectively equivalent to the hyper-parameter in the barrier function of the primal method, σ is a scaling factor (centering parameter), and $\mathbf{e}$ is a vector of ones.

We have implemented a variant of the infeasible interior-point method with the Mehrotra's correction [2], which typically accelerates the convergence by several iterations. The Mehrotra's correction divides the update into two steps: 1) an affine step, and 2) a centering step. The first step involves taking $\sigma = 0$ and hence takes a step along the true-Newton direction. In this step, the iterate moves closer to the solution but loses its centrality. The second step involves taking $\sigma > 0$ and setting the residual terms to zero, and hence the iterate makes no progress towards the solution but the centrality is improved. Precisely, the affine step involves solving the system of equations,

$$\begin{bmatrix} \boldsymbol{A} & 0 & 0 \\ 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{aff} \\ \Delta\boldsymbol{\lambda}^{aff} \\ \Delta\mathbf{s}^{aff} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ -\boldsymbol{X}\boldsymbol{S}\mathbf{e} \end{bmatrix} \tag{7}$$

while the centering step involves solving

$$\begin{bmatrix} \boldsymbol{A} & 0 & 0 \\ 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{cor} \\ \Delta\boldsymbol{\lambda}^{cor} \\ \Delta\mathbf{s}^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu\mathbf{e} - \Delta\boldsymbol{X}^{aff}\Delta\boldsymbol{S}^{aff}\mathbf{e} \end{bmatrix} \tag{8}$$

where

$$\begin{aligned} \mu &= \frac{\mathbf{x}^T\mathbf{s}}{n} \\ \mu^{aff} &= \frac{(\mathbf{x} + \Delta\mathbf{x}^{aff})^T(\mathbf{s} + \Delta\mathbf{s}^{aff})}{n} \\ \sigma &= (\frac{\mu^{aff}}{\mu})^3. \end{aligned} \tag{9}$$

The final Newton update direction is then given by

$$\begin{aligned} \Delta\mathbf{x} &= \Delta\mathbf{x}^{aff} + \Delta\mathbf{x}^{cor} \\ \Delta\boldsymbol{\lambda} &= \Delta\boldsymbol{\lambda}^{aff} + \Delta\boldsymbol{\lambda}^{cor} \\ \Delta\mathbf{s} &= \Delta\mathbf{s}^{aff} + \Delta\mathbf{s}^{cor}. \end{aligned} \tag{10}$$

4. Step Size

Important to the successful implementation of the interior-point method with Mehrotra's correction was the calculation of the step size. At every iteration, we computed the step sizes for the primal and dual variables separately, and ensured the updates still guaranteed the non-negativity of the new solution (i.e. satisfying the box constraints), while preventing the updates to be too much (i.e. following the central path). Precisely, during the affine step, the step size is calculated by,

$$\begin{aligned} \alpha_{aff}^{primal} &= argmax\{\alpha \in [0,1] \mid \mathbf{x} + \alpha\Delta\mathbf{x}^{aff} \geq 0\} \\ \alpha_{aff}^{dual} &= argmax\{\alpha \in [0,1] \mid \mathbf{s} + \alpha\Delta\mathbf{s}^{aff} \geq 0\}, \end{aligned} \tag{11}$$

while during the centering step,

$$\begin{aligned} \alpha_{max}^{primal} &= argmax\{\alpha \geq 0 \mid \mathbf{x} + \alpha\Delta\mathbf{x} \geq 0\} \\ \alpha_{max}^{dual} &= argmax\{\alpha \geq 0 \mid \mathbf{s} + \alpha\Delta\mathbf{s} \geq 0\}, \end{aligned} \tag{12}$$

and

$$\begin{aligned} \alpha^{primal} &= min(0.99\alpha_{max}^{primal}, 1) \\ \alpha^{dual} &= min(0.99\alpha_{max}^{dual}, 1). \end{aligned} \tag{13}$$

5. Stopping Criteria - Detecting Convergence

The stopping criteria entailed checking the KKT conditions at every iteration. Namely, we computed the residuals of the primal constraints, the dual constraints, and the complementarity, respectively defined as

$$\begin{aligned} \mathbf{r}_p &= \frac{\|\boldsymbol{A}\mathbf{x} - \mathbf{b}\|}{\|\mathbf{b}\|} \\ \mathbf{r}_d &= \frac{\|\boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} - \mathbf{c}\|}{\|\mathbf{c}\|} \\ \mathbf{r}_c &= \frac{\|\mathbf{s}^T\mathbf{x}\|}{n}. \end{aligned} \tag{14}$$

If all residuals fell within a small pre-defined tolerance level (e.g. 1e-8) within the maximum number of allowed iterations, then the solver declared success and the solution was optimal.

6. Stopping Criteria - Finite Termination

We tried our hands on finite termination algorithm, but had to limit our focus on this part due to time constraints. As in Ye [3], once our duality measure drops below a certain threshold, we can project the interior point into the optimal face. Let $\mathbf{x}^k$ correspond to the current interior point iterate and $\mathbf{x}^*$ correspond to the exact solution lying on the optimal face. The exact solution can be computed by solving the following norm minimization least squares problem.

$$\begin{aligned} &\min ||\mathbf{x}_B^* - \mathbf{x}_B^k|| \\ &\text{s.t. } \boldsymbol{B}\mathbf{x}_B^* = \mathbf{b} \\ &\quad \text{and} \\ &\min ||\boldsymbol{\lambda}_B^* - \boldsymbol{\lambda}^k|| \\ &\text{s.t. } \boldsymbol{B}^T\boldsymbol{\lambda}^* = \mathbf{c}_B \\ &\text{where } \mathscr{B}(\mathbf{x},\mathbf{s}) = \{i \in \{1,2,...,n\} | x_i^k \geq s_i^k\} \\ &\mathscr{N}(\mathbf{x},\mathbf{s}) = \{1,2,...,n\} \backslash \mathscr{B}(\mathbf{x},\mathbf{s}) \end{aligned} \tag{15}$$

where $\boldsymbol{B}$ is the matrix formed by columns in A corresponding to the set of indices $\mathscr{B}$ and the rest form the matrix $\boldsymbol{N}$. We denote the corresponding variables by $\mathbf{x}_B, \mathbf{s}_B, \boldsymbol{\lambda}_B$ and $\mathbf{x}_N, \mathbf{s}_N, \boldsymbol{\lambda}_N$ respectively.

The first norm-minimization problem is to project $\mathbf{x}_B^k$ into the hyperplane $\{\mathbf{x}_B^* : \boldsymbol{B}\mathbf{x}_B^* = \mathbf{b}\}$ and the second problem is to project $\boldsymbol{\lambda}^k$ into the hyperplane $\{\boldsymbol{\lambda}^* : \boldsymbol{B}^T\boldsymbol{\lambda}^* = \mathbf{c}_B\}$. The amount of work in solving these problems is like one iteration of the predictor-corrector algorithm. If the resulting solutions $\mathbf{x}_B^*$ and $\mathbf{s}_N^*$ satisfy:

$$\mathbf{x}_B^* > 0$$

and

$$\mathbf{s}_N^* = \mathbf{c}_N - \boldsymbol{N}^T\boldsymbol{\lambda}^* > 0$$

then, $\mathbf{x}^* = (\mathbf{x}_B^*, 0), \mathbf{s}^* = (\mathbf{s}_N^*, 0), \boldsymbol{\lambda}^*$ are exact optimal solutions for the original LP problem.

The two problems can be rewritten as

$$\begin{aligned} &\min ||\mathbf{x}_B^* - \mathbf{x}_B^k|| \\ &\text{s.t. } \boldsymbol{B}(\mathbf{x}_B^* - \mathbf{x}_B^k) = \mathbf{b} - \boldsymbol{B}\mathbf{x}_B^k = \boldsymbol{N}\mathbf{x}_N^k \end{aligned}$$

with solution

$$\mathbf{x}_B^* - \mathbf{x}_B^k = \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{B}^T)^{-1}\boldsymbol{N}\mathbf{x}_N^k$$

and

$$\begin{aligned} &\min ||\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^k|| \\ &\text{s.t. } \boldsymbol{B}^T(\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^k) = \mathbf{c}_B - \boldsymbol{B}^T\boldsymbol{\lambda}_B^k = \mathbf{s}_B^k \end{aligned}$$

with solution

$$\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^k = \boldsymbol{B}(\boldsymbol{B}^T\boldsymbol{B})^{-1}\mathbf{s}_B^k.$$

Both problems can be solved using one *QR factorization* of the matrix $\boldsymbol{B}$. As per [4], linearly dependent rows and/or columns of B are deleted and the corresponding components in $\mathbf{x}_B^*$ and $\boldsymbol{\lambda}^*$ are then set to zero.

This strategy is successful if the iterate is sufficiently advanced for the index sets $\mathscr{B}$ and $\mathscr{N}$ to be well resolved. If not, the strategy will produce a point that violates either $\mathbf{x}_B^* > 0$ or $\mathbf{s}_N^* > 0$. No harm is done when the strategy fails, however, we can simply return to the path-following

predictor-corrector method and take a few more steps before attempting finite termination again.

While implementing finite termination algorithm, the values of elements of $\mathbf{x}_B^*$ were tending to zero from both negative and positive sides. This resulted in the finite termination strategy failing. However, since we were asked to find points that satisfy the KKT conditions up to a given tolerance level, we stopped pursuing finite termination algorithm any further.

7. Convergence analysis

We refer readers to Wright [2] for a formal proof of convergence of infeasible interior point methods and predictor-corrector algorithm. Convergence of our infeasible interior point method is proved by showing that there is a constant $\bar{\alpha} > 0$ such that the step size $\alpha > \bar{\alpha}$ for every iteration k. The following Armijo condition holds for every iteration:

$$\mu_k(\alpha) \leq (1 - 0.01\alpha)\mu_k$$

Because of this condition, it follows that

$$\mu_{k+1} \leq (1 - 0.01\alpha_k)\mu_k \leq (1 - 0.01\bar{\alpha})\mu_k \text{ for all } k \geq 0$$

so that the sequence $\{\mu_k\}$ of duality gaps converges Q-linearly to zero. We refer interested readers to Page 110-111 of Primal-Dual interior point methods [2] for a more rigorous proof.

8. Detecting Infeasibility

Not all linear programs have solutions. Linear programs can be infeasible, or their objective functions can be unbounded from below on the feasible region. There are three generally used methods for handling infeasible starting points and infeasible programs:

1. Apply an infeasible interior point algorithm. When presented with an infeasible problem, these methods are unable to reduce the residuals below a certain (non-zero level), so the iterates $(\mathbf{x}^k, \mathbf{s}^k)$ diverge - their norm $||\mathbf{x}^T\mathbf{s}||$ approaches ∞. This behavior is not particularly graceful, but practical codes can check for it and accurately diagnose infeasibility in most cases.

2. Homogeneous Self-Dual (HSD) formulation: The second way to deal with an infeasible problem is to embed the given problem in a slightly larger problem for which a strictly feasible point is easy to find. The solution to the augmented problem should tell us everything we need to know about the original problem, including the following. If the original problem is infeasible, the augmented solution should tell us so. In fact, it should tell us whether the infeasibility occurs in the primal problem, the dual problem or both. The homogeneous self-dual (HSD) formulation of Ye, Todd and Mizuno [5] possesses all these desirable properties and is quite efficient as well.

3. Simplified HSD formulation: A variant of the HSD formulation, known as simplified HSD formulation, due to Xu, Hung and Ye [6], combines the infeasible interior point approaches and HSD approaches. Like the HSD form, it always has a solution even if the original problem does not. Unlike the HSD form, however, there are no strictly feasible starting points, so an infeasible interior point method must be used to solve the problem. Like the HSD, the simplified HSD yields either a solution of the original linear program (if one exists) or an indication of the infeasibility of

the linear program.

For our implementation, we used the first approach. If the maximum allowed iterations have been reached and the residuals did not drop below the specified threshold, the solver declares failure and the solution is not optimal or infeasible. Moreover, if during the optimization we identify the duality gap, $\|\mathbf{s}^T\mathbf{x}\|/n$, exceeds a pre-defined threshold (e.g. 1e64), the solver also declares infeasibility because the duality gap explodes.

9. Presolving Procedure

Certainly, all of the commercial linear programing packages include some sorts of the presolving procedure to simplify the linear programing problems on hand before actually proceeding to optimize it. Even though it requires additional computations, the trade-off is that it reduces the overall complexity by thinning the matrix to be solved during the Newton's step, which is the most expensive step in solving linear programs. It also effectively removes some potential numerical difficulties that might appear later during the optimization. In our implementation of the presolving strategy, we modified the full-fledged procedure from common literatures [7,8] to include only *six* iterative steps. In general, a flag was kept for each constraint (row) and each variable (column), denoting if it was still alive or dead. Once dead, it was treated as non-existent. Another flag was used to keep track of any change made during the presolving steps. The presolver terminated if a full pass of all the steps made no further change to the system. Our strategy was a simplified version of the common strategy proposed in the literature. As we eliminated the bounds and formulated the standard form in early steps, we didn't and didn't have to consider tightening the primal and dual bounds later. We only focused on removing redundancies in the system, especially row redundancies that might cause numerical difficulties. Though such simplified strategy might have caused heavier computational burdens to the Newton step due to larger dimensionality, and slowed down the optimization compared to the efficient commercial packages, it should find the same optimal solution in principle for most of the problems, if not all. We illustrate the performed steps in the following bullet points:

1) Splitting variables that are unbounded from below is useful for converting to the standard form. That is, the system of equations remains stable when we compute $\mathbf{b} - \boldsymbol{A}\mathbf{l}$ to eliminate the lower bounds on $\mathbf{x}$. Otherwise, if for some $l_i = -\infty$, such operation would cause unexpected and problematic effects on the system of constrains. Concretely, if $x_i \in (-\infty, u]$ or $x_i \in (-\infty, \infty)$, we let $x_i = x_i^+ - x_i^-$, where $x_i^+ \in [0, u]$ or $x_i^+ \in [0, \infty)$, and $x_i^- \in [0, \infty)$.

2) Checking the bounding conditions entails two special scenarios: *Infeasible bound* and *fixed variable*. In the first case, the problem could be directly declared *infeasible* if $l_i > u_i$ for some variable i. In the second case, where $l_i = u_i$, the variable must be fixed at its bound, and thus could be directly eliminated from the problem, reducing the dimension by one. However, it is important to also subtract the value from the set of equality constraints, i.e. if x_i was fixed, e.g. $x_i = l_i$, then we eliminate x_i and immediately update $\mathbf{b} = \mathbf{b} - \boldsymbol{A}_{*i}x_i$.

3) To convert the linear program into its standard form, we subtract the lower bound from the variables, i.e. $\mathbf{y} = \mathbf{x} - \mathbf{l}$, and form

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T(\mathbf{y} + \mathbf{l}) \\ \text{subject to} & \boldsymbol{A}(\mathbf{y} + \mathbf{l}) = \mathbf{b} \\ & \mathbf{l} \le \mathbf{y} + \mathbf{l} \le \mathbf{u} \end{array},$$

equivalently,

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{y} \\ \text{subject to} & \boldsymbol{A}\mathbf{y} = \mathbf{b} - \boldsymbol{A}\mathbf{l} = \mathbf{b}' \\ & 0 \le \mathbf{y} \le \mathbf{u} - \mathbf{l} = \mathbf{u}' \end{array} .$$

Furthermore, we introduce *slack variables* to eliminate all the *finite* upper bounds $\mathbf{u}'$. This step requires augmenting the $\boldsymbol{A}$, $\mathbf{c}$ and $\mathbf{b}'$ accordingly, but we have therefore converted the problem into its standard form, where $\mathbf{y}$ were bounded non-negative only. Doing so simplifies the following steps and yields a problem in the form suitable for the infeasible interior point method.

4) To simplify the rows, we simply loop over the constraints and count how many non-zeros entries are present in each constraint. If a constraint is empty, it is removed; otherwise if it has only one non-zero entry at index j, the variable x_j is fixed at b_i/A_{ij} in order to satisfy the equality. We then remove the variable from the problem, and set the solution to be the fixed value. Similarly, immediately after the removal, we update the system of equations by $\mathbf{b} = \mathbf{b} - \boldsymbol{A}_{*j}x_j$.

5) A loop over the variables should identify the empty columns, but an empty column may result in different outcomes. If the j-th column is empty, then x_j is a free variable bounded only by its own box. However, depending on the objective function $\mathbf{c}$, if $c_j < 0$ then $x_j = u_j$ or the problem is unbounded; if $c_j > 0$ then $x_j = l_j$ or the problem is unbounded; or if $c_j = 0$, then x_j can be set to any value within its bounds, e.g. to its lower bound in our case.

6) Detecting redundant rows is not a trivial process. While Gaussian elimination should do the job, it is computationally expensive to be used in presolving, because the presolving steps are to be repeated many times. The common strategy described in the literature [7,8] mostly consists of the elimination of duplicate rows, where a duplicate row is defined as a row being a simple constant multiple of the others. In this case, the duplicate row can be removed and the dimension is reduced by one. However, this process doesn't guarantee the eliminations of all linearly dependent rows. We implement this process by using each row in turn as a basis, by which every other row with identical pattern was subtracted, so that at least one entry in the current row is zeroed out. The identical pattern is determined by the pattern of the non-zero entries. A pseudo-code is presented in the following:

```
m ← length(rows)
for i = 1 : m do
    basis ← rows[i]
    for j = i + 1 : m do
        current ← rows[j]
        if identical non-zero pattern then
            determine the constant factor α
            current ← current − α ∗ basis
        end if
    end for
end for
```

It is worth noting that this particular step has increased the runtime complexity by $O(m^2)$, whereas all other presolving steps require only linear runtime complexity.

10. Postsolver

For each presolving step, there should be an appropriate corresponding postsolving step, attempting to revert the optimal solution under the presolved condition back to the original problem. Here, since our presolving strategy was simplified, our postsolving strategy also requires only minimal work. Precisely, we take the solution up to the dimension before adding the slack variables in *step 3*, and revert the solution by $\mathbf{x} = \mathbf{y} + \mathbf{l}$ (see *step 3*). Then, we merge the split variables (see *step 1*) by exactly $\mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-$. Finally, we fill in the fixed variables removed during the presolving procedure.

11. Initialization

To initialize a reasonably close point to begin the optimization, we solve the minimum-norm least squares for the primal variables after the presolving procedure,

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{x}\|^2 \\ &\text{subject to } \boldsymbol{A}\mathbf{x} = \mathbf{b} \\ &\text{with approximate solution} \\ &\mathbf{x}_0 = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T + \epsilon \boldsymbol{I})^{-1}\mathbf{b}, \end{aligned} \tag{16}$$

and for the dual variables,

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{s}\|^2 \\ &\text{subject to } \boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} = \mathbf{c} \\ &\text{with approximate solution} \\ &\boldsymbol{\lambda}_0 = (\boldsymbol{A}\boldsymbol{A}^T + \epsilon \boldsymbol{I})^{-1}\boldsymbol{A}\mathbf{c} \\ &\mathbf{s}_0 = \mathbf{c} - \boldsymbol{A}^T\boldsymbol{\lambda}_0. \end{aligned} \tag{17}$$

Note that the notation corresponds to the system after presolving, and a regularization parameter $\epsilon = 1e-6$ is used to prevent singular inversion, which is likely due to rank deficiency. Furthermore, we must ensure the positivity of $\mathbf{x}$ and $\mathbf{s}$ in order to use the infeasible interior point method. To do so, we simply correct their values by adding a constant to the variables that were negative,

$$\begin{aligned} \mathbf{x} &= \mathbf{x} + max(-\frac{3}{2} * min(\mathbf{x}), 0) \\ \mathbf{s} &= \mathbf{s} + max(-\frac{3}{2} * min(\mathbf{s}), 0), \end{aligned}$$

such that the most negative values are perturbed to be strictly positive, while nothing is changed if there was no need for perturbation. Notice that the choice of $\frac{3}{2}$ is rather arbitrary, and any value that might ensure strict positivity may do the job.

12. Linear Algebra Issues and Cholesky Factorization

Most of the computational effort in implementation of primal-dual methods is taken up in solving linear systems of the forms given by (5). The coefficient matrix in these systems is usually large and sparse, since the constraint matrix $\boldsymbol{A}$ is itself large and sparse in most applications. The special structure in the step equations (7) and (8) allows us to reformulate them as systems with more compact symmetric coefficient matrices, which are easier and cheaper to solve than the original form. The reformulation procedures is simple, as we show by applying them to the predictor and corrector systems.

When we use the infeasible primal-dual interior point method to solve linear programs, in every iteration, a system of Newton's equations needs to be solved in order to determine the direction to move to reach the solution. A widely used approach to tackle this specific problem is to use the *normal equation* formulation. Therefore, we derive the normal equations for the Mehrotra's predictor-corrector algorithm. For the predictor step, the Newton's system are given by,

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

where

$$\begin{aligned} \mathbf{r}_b &= \boldsymbol{A}\mathbf{x} - \mathbf{b}, \\ \mathbf{r}_c &= \boldsymbol{A}^T\boldsymbol{\lambda} + \mathbf{s} - \mathbf{c}, \end{aligned}$$

and $\mathbf{e}$ is a vector of ones. Since the current point $(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s})$ has $\mathbf{x}$ and $\mathbf{s}$ strictly positive, the diagonal matrices $\boldsymbol{X}$ and $\boldsymbol{S}$ are nonsingular. Hence by eliminating $\Delta \mathbf{s}^{aff}$ from the above system, we obtained the following equivalent system:

$$\begin{aligned} \begin{bmatrix} 0 & \boldsymbol{A} \\ \boldsymbol{A}^T & -\boldsymbol{D}^{-1} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\lambda}^{aff} \\ \Delta \mathbf{x}^{aff} \end{bmatrix} &= \begin{bmatrix} -\mathbf{r}_b \\ -\mathbf{r}_c + \mathbf{s} \end{bmatrix} \\ \Delta \mathbf{s}^{aff} &= -\mathbf{s} - \boldsymbol{X}^{-1}\boldsymbol{S}\Delta \mathbf{x}^{aff}, \end{aligned}$$

where we introduced the notation

$$\boldsymbol{D} = \boldsymbol{S}^{-1}\boldsymbol{X}.$$

This form of the step equations usually is known as the augmented system. Since the matrix $\boldsymbol{X}^{-1}\boldsymbol{S}$ is also diagonal and nonsingular, we can go a step further, eliminating $\Delta \mathbf{x}^{aff}$ from the above system, to obtain the normal equations:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^T\Delta \boldsymbol{\lambda}^{aff} &= -\mathbf{r}_b + \boldsymbol{A}(\mathbf{x} - \boldsymbol{D}\mathbf{r}_c) \\ \Delta \mathbf{s}^{aff} &= -\mathbf{r}_c - \boldsymbol{A}^T\Delta \boldsymbol{\lambda}^{aff} \\ \Delta \mathbf{x}^{aff} &= -\mathbf{x} - \boldsymbol{D}\Delta \mathbf{s}^{aff}. \end{aligned}$$

The corrector step has a slightly different formulation, given by

$$\begin{bmatrix} 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{A} & 0 & 0 \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}^{cor} \\ \Delta \boldsymbol{\lambda}^{cor} \\ \Delta \mathbf{s}^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{r} \end{bmatrix},$$

where $\mathbf{r} = \sigma\mu\mathbf{e} - \Delta\boldsymbol{X}^{aff}\Delta\boldsymbol{S}^{aff}\mathbf{e}$. After eliminating $\Delta\mathbf{s}^{cor}$ from the above system, we obtained the following equivalent system:

$$\begin{bmatrix} 0 & \boldsymbol{A} \\ \boldsymbol{A}^T & -\boldsymbol{D}^{-1} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\lambda}^{cor} \\ \Delta\mathbf{x}^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{X}^{-1}\mathbf{r} \end{bmatrix}$$
$$\Delta\mathbf{s}^{cor} = \boldsymbol{X}^{-1}(\mathbf{r} - \boldsymbol{S}\Delta\mathbf{x}^{cor}),$$

where we again introduced the notation

$$\boldsymbol{D} = \boldsymbol{S}^{-1}\boldsymbol{X},$$

and the corresponding normal equations were,

$$\begin{aligned} \boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^T\Delta\boldsymbol{\lambda}^{cor} &= -\boldsymbol{A}\boldsymbol{S}^{-1}\mathbf{r} \\ \Delta\mathbf{s}^{cor} &= -\boldsymbol{A}^T\Delta\boldsymbol{\lambda}^{cor} \\ \Delta\mathbf{x}^{cor} &= \boldsymbol{S}^{-1}\mathbf{r} - \boldsymbol{D}\Delta\mathbf{s}^{cor}. \end{aligned}$$

Then we factorize the left-hand-side $\boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^T$ using *Chelosky factorization*, $\boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^T = \boldsymbol{L}\boldsymbol{L}^T$, where $\boldsymbol{L}$ was a lower triangular matrix. Then the solution could be calculated using the triangular matrices instead.

However, a direct implementation of the Cholesky factorization from a standard library is usually not enough for applications in solving linear programs. The major issue related to it is when the solution moves closer to the optimum, where the strict complementarity is usually established, the diagonal elements in $\boldsymbol{D}$ take both huge (e.g. $\mathbf{s} \to 0$) and tiny (e.g. $\mathbf{x} \to 0$) values, or even small negative values due to round-off errors. Such behaviors will very likely throw errors during pivoting in Cholesky factorization. There are several common approaches to handle this issue [2]:

1) One approach requires the swapping of rows and columns whenever a invalid (small) pivot is encountered. A condition is constantly checked to determine if the pivot is valid or not. After all rows and columns are examined, the decomposed triangular matrix is approximated by the valid rows and columns, while neglecting those with negligible pivots.

2) Obviously, the first approach, with constant permutations of rows and columns on the fly, is not preferable. Another approach resolves the issue by simply skipping the small pivots and setting its row and column in the triangular matrix to zero. This approach effectively approximates the rows and columns of small pivots to be zero.

3) One other approach that requires only trivial modification to the code is essentially identical to the second approach, by replacing the small pivot to a huge value. Doing so effectively approximates the sub-diagonals in the triangular matrix to be zero to the machine precision.

In our implementation, we have adopted the third approach, which is simple and requires trivial modification to the code. Precisely, during pivoting, if a pivot $x_{ii} < 1e{-}12$, then we set $x_{ii} = 1e128$, and continue with the procedure.

Moreover, it also becomes inefficient to formulate and solve the system of normal equations when $\boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^T$ is much denser than $\boldsymbol{A}$. This situation arises when $\boldsymbol{A}$ has one or more dense columns. We can modify the normal equation strategy by excluding the dense columns from the matrix-matrix

product $\boldsymbol{ADA}^T$ and correcting for this omission in a subsequent calculation. However, we should note here that our implementation of the Cholesky factorization did not take advantage of the sparse nature of the matrix, and have therefore left the dense columns alone.

The augmented system formulation has received less attention than the normal equation system, mainly because algorithms and software for factoring sparse symmetric indefinite matrices are more complicated, slower and less readily available than (sparse) Cholesky algorithms. This situation is however changing. The augmented system formulation is more cleaner and flexible than normal equations formulation in a number of respects: The difficulty caused by dense columns in A does not arise, and free variables can be handled without resorting to the various artificial devices needed in the normal equations form.

13. Software

Our delivered software package contains three major functions, including *presolve*, *solve* and *postsolve*, where *solve* contains two sub-functions, 1) finding step sizes and 2) computing the modified Cholesky factorization. All the functionality is wrapped into an interfacing function *iplp*, which takes as arguments the problem structure, the tolerance level, and the maximum number of iterations. To use the package, users should *include* the source code file into the current Julia workspace, and call the function *iplp*. Source code has been provided as a separate material to the report.

14. Test Cases

We tested our code on the following **85** problems from LPnetlib collection:

`lp_bnl1, lp_fit1p, lp_czprob, lp_scfxm3, lp_gfrd_pnc, lp_modszk1, lp_perold, lp_qap8, lp_scrs8, lp_scsd6, lp_25fv47, lp_adlittle, lp_afiro, lp_agg, lp_agg2, lp_agg3, lp_bandm, lp_beaconfd, lp_blend, lp_bore3d, lp_brandy, lp_capri, lp_degen2, lp_degen3, lp_e226, lp_etamacro, lp_ganges, lp_fffff800, lp_finnis, lp_fit1d, lp_grow15, lp_grow22, lp_grow7, lp_israel, lp_kb2, lp_lotfi, lp_pilot4, lp_recipe, lp_sc105, lp_sc205, lp_sc50a, lp_sc50b, lp_scagr25, lp_scagr7, lp_scfxm1, lp_scfxm2, lp_scorpion, lp_scsd1, lp_scsd8, lp_sctap1, lp_share1b, lp_share2b, lp_shell, lp_ship04s, lp_stair, lp_sctap2, lp_ship04l, lp_ship08s, lp_ship12s, lp_stocfor1, lp_tuff, lp_vtp_base, lpi_bgdbg1, lpi_bgetam, lpi_bgprtr, lpi_box1, lpi_chemcom, lpi_cplex2, lpi_ex72a, lpi_ex73a, lpi_forest6, lpi_galenet, lpi_itest2, lpi_itest6, lpi_klein1, lpi_klein2, lpi_mondou2, lpi_pang, lpi_pilot4i, lpi_qual, lpi_reactor, lpi_refinery, lpi_vol1, lpi_woodinfe, lpi_klein3.`

Out of which *lp_bore3d* gives the wrong objective value, *lpi_woodinfe* is reported to be feasible when it is not, and following 7 problems do not converge in 100 iterations:

`lp_capri, lp_etamacro, lp_modszk1, lp_perold, lp_shell, lp_stair, lp_vtp_base.`

The rest **76** problems (89.41%) give solutions approximately equal to the solution of the Clp simplex code.

15. Experiments

We demonstrated the solutions given by our package on the problem sets posted on the website in comparison with the Clp simplex results. There is a total of 9 test cases. Direct comparison in terms of objective value and elapsed time is shown in the following chart. Though our code ran much slower than the standard package, it identified solutions that were equally optimal to the simplex method, and thereby proven its correctness.

```
======================================>
Problem: lp_afiro

Clp solver:
Solution status: Optimal
Optimal objective value: -464.75314285714285
elapsed time: 0.184935074 seconds

Our solver:
Problem size: (27,51)
Added 0 slack variables.
Presolved problem size: (27,51)
Solution converged in 10 iterations.
Solution status: Optimal
Optimal objective value: -464.753142838679
elapsed time: 3.972409261 seconds

======================================>
Problem: lp_brandy

Clp solver:
Solution status: Optimal
Optimal objective value: 1518.5098964881286
elapsed time: 0.007183795 seconds

Our solver:
Problem size: (220,303)
Added 0 slack variables.
Presolved problem size: (137,247)
Solution converged in 21 iterations.
Solution status: Optimal
Optimal objective value: 1518.5098970098225
elapsed time: 1.45867023 seconds

======================================>
Problem: lp_fit1d

Clp solver:
Solution status: Optimal
Optimal objective value: -9146.378092420953
elapsed time: 0.008728097 seconds
```

```
Our solver:
Problem size: (24,1049)
Added 1026 slack variables.
Presolved problem size: (1050,2075)
Solution converged in 21 iterations.
Solution status: Optimal
Optimal objective value: -9146.378091739809
elapsed time: 118.012789751 seconds

======================================>
Problem: lp_adlittle

Clp solver:
Solution status: Optimal
Optimal objective value: 225494.96316238036
elapsed time: 0.001864751 seconds

Our solver:
Problem size: (56,138)
Added 0 slack variables.
Presolved problem size: (55,137)
Solution converged in 14 iterations.
Solution status: Optimal
Optimal objective value: 225494.96316240696
elapsed time: 0.155672099 seconds

======================================>
Problem: lp_agg

Clp solver:
Solution status: Optimal
Optimal objective value: -3.59917672865765e7
elapsed time: 0.004712411 seconds

Our solver:
Problem size: (488,615)
Added 0 slack variables.
Presolved problem size: (488,615)
Solution converged in 35 iterations.
Solution status: Optimal
Optimal objective value: -3.5991767286504164e7
elapsed time: 10.911391668 seconds

======================================>
Problem: lp_ganges

Clp solver:
Solution status: Optimal
```

```
Optimal objective value: -109585.73612927883
elapsed time: 0.014657076 seconds

Our solver:
Problem size: (1309,1706)
Added 397 slack variables.
Presolved problem size: (1522,1919)
Solution converged in 20 iterations.
Solution status: Optimal
Optimal objective value: -109585.73509288603
elapsed time: 404.589811653 seconds

======================================>
Problem: lp_stocfor1

Clp solver:
Solution status: Optimal
Optimal objective value: -41131.97621943641
elapsed time: 0.001654987 seconds

Our solver:
Problem size: (117,165)
Added 0 slack variables.
Presolved problem size: (106,154)
Solution converged in 18 iterations.
Solution status: Optimal
Optimal objective value: -41131.97621937284
elapsed time: 0.402112258 seconds

======================================>
Problem: lp_25fv47

Clp solver:
Solution status: Optimal
Optimal objective value: 5501.845888286745
elapsed time: 0.243783799 seconds

Our solver:
Problem size: (821,1876)
Added 0 slack variables.
Presolved problem size: (793,1849)
Solution converged in 27 iterations.
Solution status: Optimal
Optimal objective value: 5501.8458893128345
elapsed time: 144.253545344 seconds

======================================>
Problem: lpi_chemcom
```

```
Clp solver:
Solution status: Infeasible
Optimal objective value: nothing
elapsed time: 0.024353188 seconds

Our solver:
Problem size: (288,744)
Added 144 slack variables.
Presolved problem size: (432,888)
Exceed maximum iteration.
Solution status: Infeasible
elapsed time: 39.088137954 seconds
```

16. Conclusion

For the current deliverable, we attempted to implement many other features of the linear programming solver. For example, we attempted to implement a sophisticated presolving strategy by following the literature, while it turned out to be problematic and requires substantially more amount of work to make it function properly. We ended up using a simplified version because in principle either version should yield the same result only with different efficiency. Another issue with efficiency relates to the implementation of Cholesky factorization, for a minimum order sparse Cholesky is preferred over the current version in the cases of sparse left-hand-side. We also tried to incorporate a cross-over algorithm that seeks a solution at the vertex in finite terminations after the convergence of the interior point method. However, given the time constraint we were unable to successfully implement it, so we left the solution, likely not unique, as identified by the interior point method within a tolerance level.

17. References

[1] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica, 4 (1984), pp. 373-395.

[2] Wright, S. J. (1997). Primal-dual interior-point methods. Society for Industrial and Applied Mathematics.

[3] Ye, "On the finite convergence of interior-point algorithms for linear programming", Mathematical Programming, 57 (1992), pp. 325-336.

[4] S. Mehrotra and Y. Ye, "On finding the optimal face of linear programs, "TR91-10, Department of IE/MS, Northwestern University (Evanston, IL., 1991).

[5] Y. Ye, M. J. Todd, and S. Mizuno, An O($\sqrt{n}$)-iteration homogeneous and self-dual linear programming algorithms, Mathematics of Operations Research, 19 (1994), pp. 53-67.

[6] X. Xu, P. Hung, and Y. Ye, A simplified homogeneous and self-dual linear programming algorithm and its implementationm Annals of Operations Research, 62 (1996), pp. 151 - 172

[7] Andersen, E. D., and Andersen, K. D. (1995). Presolving in linear programming. Mathematical Programming, 71(2), 221-245.

[8] Gondzio, J. (1997). Presolve analysis of linear programs prior to applying an interior point method. INFORMS Journal on Computing, 9(1), 73-91.