AN INTERIOR POINT LP SOLVER

Contents

1 Introduction to primal-dual interior-point methods 2
  1.1 LP and KKT conditions ........................................ 2
  1.2 Framework of primal-dual methods .......................... 3
  1.3 Infeasible starting points ...................................... 4
  1.4 Test of simple case ............................................ 4

2 Mehrotra Predictor-Corrector Algorithm 5
  2.1 Motivation for Mehrothra’s algorithm ......................... 5
  2.2 Test of simple case ............................................ 5

3 Implementation details 7
  3.1 Good starting points ........................................... 7
  3.2 Solving the linear system ....................................... 7
  3.3 Steplength scaling parameter ................................. 9
  3.4 Converted to Standard Form .................................. 9

4 Numerical experiments 10

A Matlab codes for starting point 15

B Matlab codes for Modified Cholesky factorization 15

C Matlab codes for solving linear systems (3.10)-(3.12) 17

D Matlab codes for MPC algorithm 17

E Matlab codes for solving the problem lp-afiro 21
1 Introduction to primal-dual interior-point methods

1.1 LP and KKT conditions.

Linear programming (LP, or linear optimization) means a linear objective function and several linear constraints, which may include both equalities and inequalities. Linear programming can be applied to various fields of study. It is used in business and economics, as well as some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proved useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design. Most of the theoretical results in this section are selected from Chapter 2 of the reference [5] and Chapter 14 of the reference [4].

Linear programs are usually stated and analyzed in the following standard form:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b, \ x \geq 0,
\end{align*}
\]

where

\[x, c \in \mathbb{R}^n, \ b \in \mathbb{R}^m, \ A \in \mathbb{R}^{m \times n}, \ m < n.\]

Note that throughout this paper, there is no assumption on the matrix \(A\) other than that it has linearly independent rows.

The vector \(x \in \mathbb{R}^n\) is a solution of (1.1) if and only if there exist vectors \(s \in \mathbb{R}^n\) and \(\lambda \in \mathbb{R}^m\) for which the following KKT conditions hold for \((x, \lambda, x) = (x^*, \lambda^*, x^*)\):

\[
\begin{align*}
A^T \lambda + s = c, \quad & (1.2) \\
Ax = b, & (1.3) \\
x_i s_i = 0, & i = 1, \ldots, n, \quad (1.4) \\
(x, s) \geq 0. & (1.5)
\end{align*}
\]

Note that if \(s_i = 0, x_i > 0\), then the constraint is not tight and could be ignored; if \(s_i > 0, x_i = 0\), then the constraint is tight.

Let us restate the KKT conditions (1.2)-(1.5) in a slight different form by means of a mapping \(F : \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m} :\)

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

\[
(x, s) \geq 0.
\]

where

\[
X := \text{diag}(x_1, x_2, \ldots, x_n), \\
S := \text{diag}(s_1, s_2, \ldots, s_n), \\
e := (1, 1, \ldots, 1)^T.
\]

Note that \(F(x, \lambda, s)\) is actually linear in its first two components and only mildly nonlinear in the third component, which could be solved by Newton method.
1.2 Framework of primal-dual methods

Numerical methods for LP have been extensively studied by a huge number of authors. There are two major computational methods for LP: Simplex Method and Interior Point Method. A subclass of interior-point methods known as primal-dual methods had distinguished themselves as the most efficient practical approaches, and proved to be strong competitors to the simplex method on large problems.

Interior-point methods follow a fundamentally different approach from the simplex method. The simplex approach moves from vertex to vertex, usually improving the objective function on each step. By contrast, the most successful interior-point approaches focus instead on the KKT conditions shown in (1.2)-(1.5), searching for primal and dual variables that satisfy these conditions, and hence solve the primal and dual linear programs concurrently. Primal and dual variables that are required to be nonnegative at the solution are kept strictly positive at each interior-point iteration. That is, the iterates stay interior with respect to these constraints, though some of these variables will approach zero in the limit.

Further, each interior-point iteration is expensive to compute and can make significant progress towards the solution, while the simplex method usually requires a larger number of inexpensive iterations. Geometrically speaking, the simplex method works its way around the boundary of the feasible polytope, testing a sequence of vertices in turn until it finds the optimal one. Interior-point methods approach the boundary of the feasible set only in the limit. They may approach the solution either from the interior or the exterior of the feasible region, but they never actually lie on the boundary of this region.

Interior-point methods of the “primal-dual” variety, the type that has been the most successful in practice, generate iterates \((x, \lambda, s)\) with the following properties:

- The inequalities (1.5) are satisfied strictly at every iteration;
- The amount by which the equality conditions (1.2)-(1.3) are violated decreases at each iteration;
- The quantity \(\mu\) defined by
  \[
  \mu = (x)^T s / n
  \]  \hspace{1cm} (1.8)
  known as the duality measure, decreases at each iteration. Note that since inequalities (1.5) are strictly satisfied, this quantity is strictly positive, while because of (1.4), it will approach zero as the iterates approach a solution.
- The pairwise products \(x_i s_i, i = 1, 2, \cdots, n\), are kept roughly in balance. That is, although all these products approach zero as \((x, \lambda, s)\) approaches a primal-dual solution of the KKT equations, no single one of these quantities approaches zero much faster than the others. (From (1.8), we see that the duality measure \(\mu\) is the average value of these pairwise products.)

In later sections, we specify how interior-point methods move from one iterate to the next. The iteration number appears as a superscript: The starting point is \((x^0, \lambda^0, s^0)\) and iterates are denoted by \((x^k, \lambda^k, s^k) \in F^0, k = 0, 1, \cdots\). Here, the feasible set \(F\) and strictly feasible set \(F^0\) are respectively defined by

\[
F := \{(x, \lambda, s) | Ax = b, (x, s) \geq 0\}, \hspace{1cm} (1.9)
\]
\[
F^0 := \{(x, \lambda, s) | Ax = b, (x, s) > 0\}. \hspace{1cm} (1.10)
\]

A general framework for primal-dual path-following algorithm is shown as Algorithm 1.
Algorithm 1 General primal-dual interior point method

1: Initial guess \((x_0, \lambda_0, s_0) \in F^0\).
2: for \(k = 0, 1, 2, \cdots \) do
3:   Set \(\sigma_k \in [0, 1]\) and \(\mu_k = (x_k)^T s_k / n\).
4:   Solve
   \[
   \begin{bmatrix}
   0 & A^T & I \\
   A & 0 & 0 \\
   S^k & 0 & X^k
   \end{bmatrix}
   \begin{bmatrix}
   \Delta x^k \\
   \Delta \lambda^k \\
   \Delta s^k
   \end{bmatrix}
   =
   \begin{bmatrix}
   0 \\
   0 \\
   -X^k S^k e + \sigma_k \mu_k e
   \end{bmatrix},
   \tag{1.11}
   \]
5:   Set
   \[(x_{k+1}, \lambda_{k+1}, s_{k+1}) \leftarrow (x_k, \lambda_k, s_k) + \alpha_k (\Delta x_k, \Delta \lambda_k, \Delta s_k), \tag{1.12}
   \]
   choosing \(\alpha_k\) so that \((x_{k+1}, s_{k+1}) > 0\).
6: end for

1.3 Infeasible starting points.

For most problems, a strictly feasible starting point is difficult to find. The infeasible interior-point methods require noting more of the starting point than positivity of \(x_0\) and \(s_0\).

Define the residuals for the two linear equations as

\[
\begin{align*}
    r_b &= Ax - b, \\
    r_c &= A^T \lambda + s - c.
\end{align*}
\]

Then the step equation (1.11) becomes

\[
\begin{bmatrix}
   0 & A^T & I \\
   A & 0 & 0 \\
   S^k & 0 & X^k
   \end{bmatrix}
   \begin{bmatrix}
   \Delta x \\
   \Delta \lambda \\
   \Delta s
   \end{bmatrix}
   =
   \begin{bmatrix}
   -r_c \\
   -r_b \\
   -X^k S^k e + \sigma \mu e
   \end{bmatrix}, \tag{1.13}
\]

The search direction is still a Newton step toward the point \((x_{\sigma \mu}, \lambda_{\sigma \mu}, s_{\sigma \mu})\). It tries to bite off all the infeasibility in the equality constraints in a single step. If a full step is ever taken (that is \(\alpha = 1\)), the residuals \(r_b\) and \(r_c\) become zero, and all subsequent iterates remain strictly feasible.

Besides, there is a way shown in Section 3.1 to find a more reasonable starting point.

1.4 Test of simple case

Consider the following LP in \(\mathbb{R}^2\):

\[
\begin{align*}
    \text{minimize} & \quad x_1 \\
    \text{subject to} & \quad x_1 + x_2 = 1, \quad (x_1, x_2) \geq 0.
\end{align*}
\]

The exact solution is

\[
x^* = [0, 1], \quad \lambda^* = 0, \quad s^* = [1, 0].
\]

The numerical results are shown in Table 1. It shows that \(\lambda_0\) seems not to affect the results while the bounds \((x_0, s_0) \geq 0\) are essential.
Table 1: Numerical results for Algorithm 1 with different initial values (tol = 1e-6)

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( \lambda_0 )</th>
<th>( s_0 )</th>
<th>( x )</th>
<th>No. of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.1, 1.1]</td>
<td>1</td>
<td>[1.1, 0.1]</td>
<td>[0.000019, 0.999981]</td>
<td>169</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>100</td>
<td>[1.1, 0.1]</td>
<td>[0.000019, 0.999981]</td>
<td>169</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>-100</td>
<td>[1.1, 0.1]</td>
<td>[0.000019, 0.999981]</td>
<td>169</td>
</tr>
<tr>
<td>[1, 10]</td>
<td>1</td>
<td>[1.1, 0.1]</td>
<td>[0.000019, 0.999981]</td>
<td>213</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>1</td>
<td>[10, 1]</td>
<td>[0.000019, 0.999981]</td>
<td>213</td>
</tr>
<tr>
<td>[1, 10]</td>
<td>1</td>
<td>[10, 1]</td>
<td>[0.000019, 0.999981]</td>
<td>257</td>
</tr>
<tr>
<td>[-1, -1]</td>
<td>1</td>
<td>[0, -1]</td>
<td>[0.999999, 0.000001]</td>
<td>37</td>
</tr>
</tbody>
</table>

2 Mehrotra Predictor-Corrector Algorithm

2.1 Motivation for Mehrotra’s algorithm

Mehrotra’s algorithm [3] produces a sequence of infeasible iterates \( (x^k, \lambda^k, s^k) \) with \( (x^k, s^k) > 0 \). Each iteration step involves the following three components:

- an affine-scaling predictor step which is the pure Newton direction for the nonlinear function \( F(x, \lambda, s) \) defined by (1.6);
- a centering term by means of an adaptively chosen centering parameter \( \sigma \);
- a corrector step that compensates for some nonlinearity in the predictor step direction.

The Mehrotra Predictor-Corrector (MPC) algorithm are shown in Algorithm 2. Note that unlike predictor-corrector techniques in other areas of computational mathematics, however, we do not actually take a step along the “predictor” direction but use it only as a basis for choosing the centering parameter \( \sigma_k \). It would perhaps be more accurate to call it a “probing” direction.

The Matlab codes for Algorithm 2 (MPC), named iplp, is shown in Appendix D.

2.2 Test of simple case

The numerical results for the same test problem as Section 1.4 are shown in Table 2. Comparison between the Tables 1-2 illustrates that MPC Algorithm enjoys the following advantages:

1. A significant reduction in the number of iterations;

2. Flexibility to infeasible initials. Surprisingly, given the totally infeasible initial guess \( x_0 = (-1, -1)^T, s = (0, -1)^T \), the Algorithm 1 gives the wrong answer (spurious solution) while the Algorithm 2 gives the correct answer!
Algorithm 2 Mehrotra Predictor-Corrector Algorithm

1: Initial guess \((x^0, \lambda^0, s^0)\) satisfying only the minimal conditions \((x^0, s^0) > 0\).
2: for \(k = 0, 1, 2, \ldots\) do
3: Compute the duality measure \(\mu_k = (x^k)^T s^k / n\) and the relative residual \(\text{res}\).
4: Solve (1.13) for \(\Delta x^{aff}, \Delta \lambda^{aff}, \Delta s^{aff}\), where \(\sigma = \mu = 0\).
5: Calculate \(\alpha_{aff}^{pri}, \alpha_{aff}^{dual}\) and \(\mu_{aff}\) as follows

\[
\alpha_{aff}^{pri} := \min \left( 1, \min_{i, \Delta x_i < 0} \frac{x_i}{\Delta x_i^{aff}} \right),
\]
\[
\alpha_{aff}^{dual} := \min \left( 1, \min_{i, \Delta s_i < 0} \frac{s_i}{\Delta s_i^{aff}} \right),
\]
\[
\mu_{aff} := (x^k + \alpha_{aff}^{pri} \Delta x^{aff})^T (s^k + \Delta x^{aff} \Delta s^{aff}).
\]
6: Set centering parameter to be

\[
\sigma = \left( \frac{\mu_{aff}}{\mu_k} \right)^{\frac{3}{2}}.
\]
7: Solve (1.13) for \(\Delta x, \Delta \lambda, \Delta s\), where the third component of the right hand side is replaced by \(-X^k S^k - \Delta X^{aff} S S^{aff} e + \sigma \mu_k e\).
8: Calculate the \(\alpha_{k}^{pri}\) and \(\alpha_{k}^{dual}\) as follows

\[
\alpha_{k}^{pri} = \min(1, \eta \alpha_{max}^{pri}),
\]
\[
\alpha_{k}^{dual} = \min(1, \eta \alpha_{max}^{dual}),
\]

where \(\eta \in [0, 99, 1.0]\) and \(\alpha_{max}^{pri}, \alpha_{max}^{dual}\) are given by

\[
\alpha_{max}^{pri} := \min \left( 1, \min_{i, \Delta x_i < 0} \frac{x_i}{\Delta x_i} \right),
\]
\[
\alpha_{max}^{dual} := \min \left( 1, \min_{i, \Delta s_i < 0} \frac{s_i}{\Delta s_i} \right).
\]
9: Set

\[
x^{k+1} = x^k + \alpha_{k}^{pri} \Delta x
\]
\[
(\lambda^{k+1}, s^{k+1}) = (\lambda^k, s^k) + \alpha_{k}^{dual}(\Delta \lambda, \Delta s).
\]
10: end for
Table 2: Numerical results for Algorithm 2 with different initial values (tol = 1e-6)

<table>
<thead>
<tr>
<th>$\mathbf{x}_0$</th>
<th>$\lambda_0$</th>
<th>$s_0$</th>
<th>$\mathbf{x}$</th>
<th>$\text{No. of iterations}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.1, 1.1]</td>
<td>1</td>
<td>[1.1, 0.1]</td>
<td>[0.000000, 1.000000]</td>
<td>3</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>100</td>
<td>[1.1, 0.1]</td>
<td>[0.000000, 1.000000]</td>
<td>3</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>-100</td>
<td>[1.1, 0.1]</td>
<td>[0.000000, 1.000000]</td>
<td>3</td>
</tr>
<tr>
<td>[1, 10]</td>
<td>1</td>
<td>[1.1, 0.1]</td>
<td>[0.000000, 1.000000]</td>
<td>3</td>
</tr>
<tr>
<td>[0.1, 1.1]</td>
<td>1</td>
<td>[10, 1]</td>
<td>[0.000000, 1.000000]</td>
<td>4</td>
</tr>
<tr>
<td>[1, 10]</td>
<td>1</td>
<td>[10, 1]</td>
<td>[0.000000, 1.000000]</td>
<td>4</td>
</tr>
<tr>
<td>[-1, -1]</td>
<td>1</td>
<td>[0, -1]</td>
<td>[0.000000, 1.000000]</td>
<td>5</td>
</tr>
</tbody>
</table>

3 Implementation details

In order to improve the performance of the above algorithms, say decreasing the failure rate and reducing the number of interior-point iterations, particular attention should be paid to some details in practical implementation. We list here the most important modifications and enhancements according to my knowledge. Most of the materials shown in this section are chosen from Chapter 11 of [5] and Chapter 8 of [2].

3.1 Good starting points

It seems that in Algorithm 2, we have considerable flexibility in our choice of starting point. However, choice of starting point is an important practical issue with a significant effect on the robustness of the algorithm. A poor choice ($\mathbf{x}_0^0, \lambda_0^0, s_0^0$) satisfying only the minimal conditions ($\mathbf{x}_0^0, s_0^0$) > 0 often leads to failure of convergence.

Generally speaking, a good starting point should also satisfy two other conditions.

- First, the point should be well centered, so that the pairwise products $x_i^0 s_i^0$ are similar for all $i = 1, 2, \cdots, n$.
- Second, the point should not be too infeasible; that is, the ratio $|\langle \mathbf{r}_0^0, \mathbf{r}_0^0 \rangle| / \mu_0$ of infeasibility to duality measure should not be too large.

A popular heuristic for finding ($\mathbf{x}_0^0, \lambda_0^0, s_0^0$) starts by calculating ($\bar{\mathbf{x}}, \bar{\lambda}, \bar{s}$) as the solution of two least-squares problems

$$\min_{\mathbf{x}} \frac{1}{2} \mathbf{x}^T \mathbf{x} \quad \text{subject to } \mathbf{A} \mathbf{x} = \mathbf{b}, \quad (3.1)$$

$$\min_{(\lambda, s)} \frac{1}{2} s^T s \quad \text{subject to } \mathbf{A}^T \lambda + s = \mathbf{c}. \quad (3.2)$$

The practical algorithm of finding the starting point is given in Algorithm 3 and the corresponding Matlab code is in Appendix A.

Note that the computational cost of finding ($\mathbf{x}_0^0, \lambda_0^0, s_0^0$) by this scheme is about the same as one step of the primal-dual method.

3.2 Solving the linear system

In each iteration of the MPC (Algorithm 2) shown above, we need to solve the linear system (1.13) twice (in steps 4 and 7 respectively), with the same coefficient matrix but different right hand sides. It is crucial to solve this system efficiently since for practical LP, it is often very large.
Algorithm 3 Find the starting point \((x^0, \lambda^0, s^0)\)

1: Solve the problems (3.1)-(3.2):
\[
\tilde{x} = A^T(AA^T)^{-1}b, \quad \tilde{\lambda} = (AA^T)^{-1}Ac, \quad \tilde{s} = c - A^T\tilde{\lambda}.
\] (3.3)

2: Get rid of the nonpositive components of \(\tilde{x}\) and \(\tilde{s}\):
\[
\delta_x = \max \left(-\frac{3}{2} \min_i \tilde{x}_i, 0 \right), \quad \hat{x} = \tilde{x} + \delta_x e, \quad (3.4)
\]
\[
\delta_s = \max \left(-\frac{3}{2} \min_i \tilde{s}_i, 0 \right), \quad \hat{s} = \tilde{s} + \delta_s e, \quad (3.5)
\]
where \(e = (1, \cdots, 1)^T\).

3: Ensure that the components of \(x^0\) and \(s^0\) are not too close to zero and not too dissimilar:
\[
\hat{\delta}_x = \frac{1}{2} \frac{\tilde{x}^T\hat{x}}{\tilde{x}^T\hat{s}}, \quad x^0 = \hat{x} + \hat{\delta}_x e, \quad (3.6)
\]
\[
\hat{\delta}_s = \frac{1}{2} \frac{\tilde{s}^T\hat{s}}{\tilde{s}^T\hat{x}}, \quad s^0 = \hat{s} + \hat{\delta}_s e. \quad (3.7)
\]

4: Besides,
\[
\lambda^0 = \hat{\lambda}. \quad (3.8)
\]

We apply the reformulation procedures to the following general form of the linear system:
\[
\begin{bmatrix}
0 & A^T & I \\
A & 0 & 0 \\
S & 0 & X
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta \lambda \\
\Delta s
\end{bmatrix}
= \begin{bmatrix}
-r_c \\
r_b \\
-r_{xs}
\end{bmatrix}. \quad (3.9)
\]

Actually, it can be solved efficiently in the following way:
\[
AD^2\Delta \lambda = -r_b - AS^{-1}Xr_c + AS^{-1}r_{xs}, \quad (3.10)
\]
\[
\Delta s = -r_c - A^T\Delta \lambda, \quad (3.11)
\]
\[
\Delta x = -S^{-1}X\Delta s - S^{-1}r_{xs}, \quad (3.12)
\]

where \(D = S^{-1/2}X^{1/2}\).

Practically, I use direct sparse Cholesky algorithms to factor the matrix \(AD^2A^T\), and then perform triangular substitution (forward, then backward) to obtain the step \(\Delta \lambda\) from Eq.(3.10). Then steps \(\Delta s\) and \(\Delta x\) are recovered from Eq.(3.11) and Eq.(3.12).

It is well known that the process of Cholesky factorization followed by triangular substitution is a remarkably stable way to compute solutions of linear systems with symmetric positive definite coefficient matrices. However, during the final stages of a primal-dual algorithm when \((x, s)\) goes close to the solution, the elements of the diagonal weighting matrix \(D^2\) take on both huge and tiny values, resulting in the ill-conditioning of the system (3.10).

Hence, general-purpose sparse Cholesky subroutines (e.g. \texttt{chol} in Matlab) can be applied to \(AD^2A^T\), but modifications are needed because \(AD^2A^T\) may be ill-conditioned or singular. The idea of modified Cholesky factorization is to ensure that \(E = A - LL^T\) is reasonably small if \(A\) is not too far from being indefinite. The Matlab code is shown in Appendix B.

Additional remarks about this issue are listed below:

- Observe that in Steps 4 and 7, both linear system have the same coefficient matrix, which means just a single matrix factorization is required in each iteration.
• It is better to do row permutation on $A^T$ to ensure that the factor $L$ stays reasonably sparse during the Cholesky factorization, i.e.

$$LL^T = PAD^2A^TP^T,$$

(3.13)

where the permutation matrix $P$ can be computed before the first iteration.

• The subroutine `step_solver_chol` for solving the linear system with modified Cholesky factorization is shown in Appendix C. If you run my codes, it is of high probability to see “Warning from Modified Cholesky factorization: The matrix may be not symmetric!”. That is due to the singularity of the matrix $D^2$.

3.3 Steplength scaling parameter

In general, we have considerable freedom in the choice of centering parameters $\sigma_k$ and steplength scaling parameters $\eta_k$ at each iteration. We would like to choose these parameters so as to make good progress in reducing the duality measure $\mu_k$ to zero on the current step (which requires $\sigma_k$ close to zero) while at the same time maintaining steplengths $\alpha_k$ close to 1 (which is more likely to happen when $\sigma_k$ is closer to 1). In addition, it has proven effective in practice to choose $\eta_k$ closer to 1 as we approach the solution but in any case to ensure that this parameter does not drop below a threshold quite close to 1, for example, 0.99.

On one hand, there is a formula for choosing $\sigma_k$ adaptively in Step 6 of Algorithm 2 (see Eq.(2.4)). On the other hand, I just use the following simple assignment to choose $\eta_k$

$$\eta = \max(0.995, 1 - \mu_k).$$

(3.14)

3.4 Converted to Standard Form

Many practical problems are not presented in standard form (1.1) but in this way, instead,

$$\min_{x} c^T x$$

subject to $A x = b$, $low \leq x \leq high$.

I use two steps to convert (3.15) to standard form.

First, set $y = x - low$, then it becomes

$$\min_{y} c^T y$$

subject to $A y = b - A * low$, $0 \leq y \leq high - low$.

Second, choose $z$ such that $y + z = high - low$, then it becomes

$$\min_{y} c^T y$$

subject to $A y = b - A * low$, $y + z = high - low$, $y, z \geq 0$.

Let us denote

$$\tilde{x} = \begin{bmatrix} y \\ z \end{bmatrix}, \quad \tilde{A} = \begin{bmatrix} A & 0 \\ I & I \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} b - A * low \\ high - low \end{bmatrix}, \quad \tilde{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}.$$

Then the problem (3.17) can be written as

$$\min_{\tilde{x}} \tilde{c}^T \tilde{x}$$

subject to $\tilde{A} \tilde{x} = \tilde{b}$, $\tilde{x} \geq 0$,

which is of standard form.
4 Numerical experiments

The test problems all come from the University of Florida Sparse Matrix repository [1]:
1. LPnetlib/lp-adlittle,
2. LPnetlib/lp-afiro,
3. LPnetlib/lp-agg,
4. LPnetlib/lp-brandy,
5. LPnetlib/lp-fit1d,
6. LPnetlib/lp-ganges,
7. LPnetlib/lp-stocfor1.

An example to illustrate how to use Matlab routine ip1p to solve the problem LPnetlib/lp-afiro is shown in Appendix E.

The numerical results are shown in Table 3 and Figs. 1-7. Note that in Table 3, \(N\) is the number of iterations, \(\mu\) is duality measure defined in (1.8) and Res means relative residual defined by

\[
\text{res} = \frac{\|r_c; r_b; \mu\|_2}{\|b; c\|_2}.
\]

Besides, the exact optimal values are taken from the “PROBLEM SUMMARY TABLE” given by the UF Sparse Matrix Collection.

We can see that all of the numerical optimal values are very close to the exact ones. The duality measures and relative residuals are also acceptable while the number of iterations is not very large, which verifies that these problems are efficiently solved by the ip1p.

In addition, the history of iteration for problems “lp-brandy” and “lp-ganges” (see Figs. 4 and 6) seems strange. I think the reason is that the starting points is not feasible. That is why the values of object functions are increasing at the beginning.

<table>
<thead>
<tr>
<th>Name</th>
<th>Optimal values</th>
<th>(N)</th>
<th>(\mu)</th>
<th>Res</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp-adlittle</td>
<td>2.254996316e+05</td>
<td>11</td>
<td>1.0271e-05</td>
<td>1.2113e-08</td>
</tr>
<tr>
<td>lp-afiro</td>
<td>-4.6475314286e+02</td>
<td>8</td>
<td>2.7618e-06</td>
<td>2.4086e-08</td>
</tr>
<tr>
<td>lp-agg</td>
<td>-3.5901767287e+07</td>
<td>42</td>
<td>4.2471e-03</td>
<td>2.1351e-08</td>
</tr>
<tr>
<td>lp-brandy</td>
<td>1.5185099370e+03</td>
<td>17</td>
<td>1.5037e-04</td>
<td>1.4875e-05</td>
</tr>
<tr>
<td>lp-fit1d</td>
<td>-9.143780924e+03</td>
<td>18</td>
<td>6.5158e-07</td>
<td>7.5149e-09</td>
</tr>
<tr>
<td>lp-ganges</td>
<td>-1.0958636356e+05</td>
<td>26</td>
<td>1.6098e-06</td>
<td>2.4209e-07</td>
</tr>
<tr>
<td>lp-lp-stocfor1</td>
<td>-4.1131976219e+04</td>
<td>18</td>
<td>7.3545e-07</td>
<td>1.2340e-08</td>
</tr>
</tbody>
</table>
Figure 1: Numerical results of lp-adlittle.

Figure 2: Numerical results of lp-afiro.
Figure 3: Numerical results of lp-agg.

Figure 4: Numerical results of lp-brandy.
Figure 5: Numerical results of lp-fit1d.

Figure 6: Numerical results of lp-ganges.
References


APPENDIX

Here are the Matlab codes of all the algorithms mentioned in previous sections. If you have any questions related to run these codes, please let me know.
A Matlab codes for starting point

function [x, lam, s] = starting_point(A, b, c)
% We describe here a heuristic that finds a starting point that
% satisfies the equality constraints in the primal and dual problems reasonably well,
% while maintaining positivity of the x and s components
% and avoiding excessively large values of these components.

%% first step
AA = A*A';
xb = AA\b;
xt = A'*xb;
lam = AA\(A*c);
st = c - A'*lam;

%% second step
dx = -1.5*min(xt);
dx = max(0,dx);

ds = -1.5*min(st);
ds = max(0,ds);

xh = xt + dx;
sh = st + ds;

%% third step
xs = xh'*sh;
dxh = 0.5*xs/sum(sh);
dsh = 0.5*xs/sum(xh);

x = xh + dxh;
s = sh + dsh;

B Matlab codes for Modified Cholesky factorization

function [R, indef, E] = cholmod(A)
% CHOLMOD Modified Cholesky factorization
% R = cholmod(A) returns the upper Cholesky factor of A (same as CHOL)
% if A is (sufficiently) positive definite, and otherwise returns a
% modified factor R with diagonal entries >= sqrt(delta) and
% offdiagonal entries <= beta in absolute value,
% where delta and beta are defined in terms of size of diagonal and
% offdiagonal entries of A and the machine precision.

if sum(sum(abs(A-A'))) > eps
fprintf('
 Warning from Modified Cholesky factorization: The matrix may be not symmetric! 
'); end

% set parameters governing bounds on L and D (eps is machine epsilon)
\begin{verbatim}
% initialize d and L

d = zeros(n,1);
if issparse(A)
    L = speye(n);  \% sparse identity
else
    L = eye(n);
end

% there are no inner for loops, everything implemented with
% vector operations for a reasonable level of efficiency

for j = 1:n
    \% column index: all columns to left of diagonal
    K = 1:j-1;  \% d(K) doesn't work in case K is empty
    djtemp = A(:,j) - L(j,K)*d(K,1).*L(j,K)';  \% C(j,j) in book
    if j < n
        I = j+1:n;  \% row index: all rows below diagonal
        Ccol = A(I,j) - L(I,K)*d(K,1).*L(j,K)';  \% C(I,j) in book
        theta = max(abs(Ccol));  \% guarantees d(j) not too small and L(I,j) not too big
        \%
        d(j) = max([abs(djtemp), (theta/beta)^2, delta]);
        L(I,j) = Ccol/d(j);
    else
        d(j) = max([abs(djtemp), delta]);
    end
end

% convert to usual output format: replace L by L*sqrt(D) and transpose
for j=1:n
    L(:,j) = L(:,j)*sqrt(d(j));  \% L = L*diag(sqrt(d)) bad in sparse case
end;
R = L';
if nargout == 3
    E = A - R'*R;
end
\end{verbatim}
C  Matlab codes for solving linear systems (3.10)-(3.12)

function [dx,dlam,ds] = step_solver_chol_order(A,R,dxs,ss,rc,rb,rxs,ordering)
%%% solve the linear system
%%% [ 0 A' I ] [ dx ] = [ -rc
%%% A 0 0 dlam ] = [ -rb
%%% S 0 X ] ds [ ] = -rxs ]
%%% where A: m*n
%%% and S,X: n*n, S = diag(s), X = diag(x)
%%% e = ones(n,1)
%%% using Modified Cholesky decomposition

flam = -rb - A*(dxs.*rc-ss.*rxs);

dlam(ordering) = R\(R'\flam(ordering));
dlamb = dlam(:);

ds = -rc - A'dlam;
dx = -ss.*rxs - dxs.*ds;

D  Matlab codes for MPC algorithm

function [x,flag,cs,As,bs,xs,lam,s,hist] = iplp(Problem,tol,maxit)

% IPLP Interior Point LP Solver
%
% x = iplp(Problem) solve the linear program:
% minimize c'*x where Ax = b and lo <= x <= hi
% where the variables are stored in the following struct:
% Problem.A
% Problem.aux.c
% Problem.aux.lo
% Problem.aux.hi
% Problem.b
%
% [x,flag,cs,As,bs,xs,lam,s] = iplp(Problem)
% should also returns a flag indicating whether or not the
% solution succeeded (flag = 0 => success and flag ~= 0 => failure),
% along with the solution for the problem converted to standard form (xs):
% minimize cs'*xs where As*xs = bs and xs >= 0
% and the associated Lagrange multipliers (lam, s).
%
% [...] = iplp(Problem,tol,maxit) solves the problem to a point where
% the duality measure (xs'*s)/n <= tol and the normalized residual
% norm([As'*lam + s - cs; As*xs - bs; xs.*s])/norm([bs;cs]) <= tol
% and fails if this takes more than maxit iterations.
% hist: history of iterations
%
% Remark: Mehrotra Predictor-Corrector (MPC) algorithm

if nargin==0
    fprintf('n Please input Problem. \n')
elseif nargin==1
    tol = 1e-7;
    maxit = 100;
elseif nargin==2
    maxit = 100;
end

%% Load the data
A = Problem.A;
b = Problem.b;
c = Problem.aux.c;
low = Problem.aux.lo;
high = Problem.aux.hi;

[m,n] = size(A);
jh = 0;

%% Convert to standard form
if (low==0 & high==inf)
    fprintf('n The original problem is already in standard form. \n')
    As = sparse(A);
cs = c;
bs = b;
else if (find(low ~= 0))
    b = b - A*low;
    high = high - low;
    bs = b;
    As = sparse(A);
end
if (find(high ~= inf))
    Jhigh = find(high ~= inf);
    Whigh = high(Jhigh);
    jh = length(Jhigh);
    B1 = zeros(m,jh);
    B2 = eye(jh);
    B3 = zeros(jh,n);
    B3(:,Jhigh) = B2;
    As = [A,B1;B3,B2];
    As = sparse(As);
    cs = [c;zeros(jh,1)];
    bs = [b;Whigh];
end
clow = c'*low;

%% find the tarting point
[x0, lam0, s0] = starting_point(As, bs, cs);

n1 = n + jh;
m1 = m + jh;

%% Primal-Dual Interior Point method with Mehrotra predictor-corrector algorithm

hist_k = [];
hist_f = [];
hist_mu = [];
hist_nres = [];

% main loop
flag = 1;
k=0;

nbc = norm([bs; cs]);

etaMin = 0.999;
maxDiag = 5e+15;
sig0 = 1e-4;
sig1 = 0.75;
mu = 1;
nres = 1;
ordering = colamd(As');

while ( k<maxit)
% predictor
rxs = x0.*s0;
muk = mean(rxs);
% sigma = min(0.1,100*muk);
rc = As'*lam0 + s0-cs;
rb = As*x0 - bs;
nres = norm([rc; rb; rxs])/nbc;

if(nres <= tol & muk <= tol) break; end;

dxs = min(maxDiag, x0./s0);
ss = min(maxDiag, 1./s0);
B = As*sparse(1:n1,1:n1,dxs)*As';

% R = cholmod(B);
R = cholmod(B(ordering,ordering));
% [dx,dlam,ds] = step_solver_chol(As,R,dxs,ss,rc,rb,rxs);
% [dx,dlam,ds] = step_solver_chol_order(As,R,dxs,ss,rc,rb,rxs,ordering);

ax = step_length(x0,dx);
as = step_length(s0,ds);

% corrector
mu = mean((x0 + ax*dx).*(s0 + as*ds));
sigma = (mu/muk)^3;
sigma = sig0*(sigma<sig0) + sig1*(sigma>sig1) + sigma*(sigma>sig0 & sigma<sig1);

rxs = rxs + dx.*ds - sigma*muk;

% [dx,dlam,ds] = step_solver_chol(As,R,dxs,ss,rc,rb,rxs);
% [dx,dlam,ds] = step_solver_chol_order(As,R,dxs,ss,rc,rb,rxs,ordering);

ax = step_length(x0,dx);
as = step_length(s0,ds);
eta = max(etaMin, 1-muk);
axk = min(1,eta*ax);
ask = min(1,eta*as);

% update
x0 = x0 + axk*dx;
s0 = s0 + ask*ds;
lam0 = lam0 + ask*dlam;

k=k+1;

% update the history
f0 = cs'*x0 + clow;
hist_f = [hist_f;f0];
hist_k = [hist_k;k];
hist_mu = [hist_mu;muk];
hist_nres = [hist_nres;nres];
end

% flag
if (muk<tol & nres<tol)
    flag = 0;
end

%% output
xs = x0;
s = s0;
lam = lam0;
x = xs(1:n) + low;

hist = [hist_k,hist_f,hist_mu,hist_nres];
function test_afiro

%% problem lp_afiro
load lp_afiro.mat
[x,flag,cs,As,bs,xs,lambda,s,hist] = iplp(Problem,1e-7,100);

flag
k = hist(:,1);
f = hist(:,2);
mu = hist(:,3);
res = hist(:,4);

P = 1;

subplot(1,3,1)
plot(k(P:end),f(P:end),'-bs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',2)
set(gca,'FontSize',15)
xlabel('Iterations')
ylabel('object function values')

subplot(1,3,2)
semilogy(k(P:end),mu(P:end),'-bs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',2)
set(gca,'FontSize',15)
xlabel('Iterations')
ylabel('log(mu)')
title('LP: afiro')

subplot(1,3,3)
semilogy(k(P:end),res(P:end),'-bs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',2)
set(gca,'FontSize',15)
xlabel('Iterations')
ylabel('log(residual)')