Abstract

A linear program (or a linear programming problem, or an LP) is an optimization problem that involves minimizing or maximizing a linear function subject to linear constraints. Linear programming plays a fundamental role not only in optimization, but also in economics, strategic planning, analysis of algorithms, combinatorial problems, cryptography, and data compression. When linear programs are formulated in so-called standard form the set of variables satisfying the linear constraints (the feasible set) is a convex polytope. If this polytope is bounded and non-empty, then a solution of the linear program must lie at a vertex. The simplex method for linear programming moves from one vertex to another until an optimal vertex is found. Almost all practical linear programs are degenerate, i.e., there are vertices that lie at the intersection of a set of linearly dependent constraints. The conventional simplex method can cycle at a degenerate vertex, which means that the iterations can be repeated infinitely without changing the variables. In the first part of this thesis, we discuss the effects of degeneracy and investigate several non-simplex methods that avoid cycling by solving a linear least-squares subproblem at each iteration. In the second part, we propose a new active-set method based on adding a penalty term to the linear objective function. Numerical results indicate that the proposed method is computationally efficient.
Contents

1. Introduction .................................................. 4

2. Notation, Definitions and Theorems .......................... 5
   2.1 Notation .................................................... 5
   2.2 Definitions ................................................. 5
   2.3 Theorems and Lemmas ...................................... 6

3. Primal-Dual Non-Negative Least Squares .................. 6
   3.1 Solving the Linear Program ............................... 6
      3.1.1 Finding a feasible ascent direction .................. 6
      3.1.2 The PDNNLS algorithm ................................ 8
      3.1.3 Finding a feasible point ............................. 8
   3.2 Solving the NNLS Sub-problem ............................ 9
      3.2.1 Optimality conditions for NNLS ....................... 9
      3.2.2 The Lawson and Hanson method ....................... 10
      3.2.3 Fast NNLS algorithm ................................. 12

4. Mixed Constraints ............................................. 12
   4.1 Solving the Mixed-Constraint LP ........................ 13
      4.1.1 Optimality conditions ................................. 13
      4.1.2 Finding a descent direction ........................ 14
      4.1.3 The algorithm ......................................... 15
      4.1.4 Finding a feasible point ............................. 16
   4.2 Solving Bounded Least Square Problems .................. 16
      4.2.1 Solving BLS using the singular-value decomposition .. 16
      4.2.2 Dax’s method for BLS ................................. 19

5. A Primal-Dual Active Set Method for Regularized LP .... 20
   5.1 Formulation of the Primal and Dual Problems ............ 21
   5.2 Optimality Conditions and the KKT Equations ............ 21
   5.3 The Primal Active-Set Method ............................ 27
   5.4 The Dual Active-Set Method .............................. 30
   5.5 Degeneracy ............................................... 32
   5.6 Combining Primal and Dual Active-set Methods .......... 32
      5.6.1 Shifting the primal problem ......................... 33
      5.6.2 Shifting the dual problem ........................... 33

6. Matrix Factorizations and Updating .......................... 35
   6.1 Using the QR Decomposition for Solving Least-Squares Problems 36
      6.1.1 Calculating a null-space descent direction .......... 36
      6.1.2 Calculating the projection matrix ................... 36
      6.1.3 Solving the KKT equations ........................... 37
   6.2 Updating the QR Factorization ........................... 37
   6.3 Modification after a Rank-one Change .................... 38
6.4  Adjoining and Deleting a Column .................................. 39
  6.4.1  Adjoining a column ............................................. 39
  6.4.2  Removing a column .............................................. 40
6.5  Adjoining and Deleting a Row ...................................... 41
  6.5.1  Adjoining a row .................................................. 41
  6.5.2  Deleting a row .................................................. 41
6.6  Other Factorization Methods ........................................ 42
  6.6.1  The symmetric indefinite factorization ......................... 42

7.  Numerical Examples .................................................. 43
  7.1  Highly Degenerate Problems ..................................... 44
      7.1.1  Test problems ................................................. 44
      7.1.2  Computational results ...................................... 44
  7.2  Cycling Problems .................................................. 45
      7.2.1  Test problems ................................................. 45
      7.2.2  Computational results ...................................... 45

8.  Summary and Conclusion ............................................. 46

References .............................................................. 47
1. Introduction

In this paper we consider methods for the solution of the linear programming (LP) problem in standard form, i.e.,

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

where \(A\) is a constant \(m \times n\) matrix, and \(b\) and \(c\) are constant vectors with \(m\) and \(n\) components, respectively. The set of variables satisfying the linear constraints of (LP) (the feasible set) is a convex polytope. If this polytope is bounded and non-empty, then a solution of the linear program (LP) must lie at a vertex of the feasible set. A vertex is characterized by a non-negative basic solution of the equations \(Ax = b\) that is calculated by solving an \(m \times m\) system of equations \(Bx_B = b\), where the columns of \(B\) are a subset of the columns of \(A\). The set of indices of the columns of \(A\) used to define \(B\) is known as a basis. The variables associated with the columns of \(B\) are known as the basic variables. The variables with indices that are not in the basis are known as the nonbasic variables.

The simplex method was the first method to be proposed for solving a linear program (see, e.g., [5]). The simplex method moves repeatedly from one vertex to an adjacent vertex until an optimal vertex is found. Each change of vertex involves moving a variable from the nonbasic set to the basic set and vice-versa. Each iteration involves solving two systems of linear equations of order \(m\). Almost all practical linear programs are degenerate, i.e., there are vertices that lie at the intersection of a set of linearly dependent constraints. Unfortunately, the conventional simplex method can cycle at a degenerate vertex, which means that the iterations can be repeated infinitely without changing the variables. A crucial decision associated with the simplex method is the choice of which variable should enter the basis at each iteration. This choice is determined by so-called pivot rules. A good choice of pivot can prevent cycling on degenerate problems and enhance the speed of the method. The “Dantzig rule” [5] is the earliest and simplest pivot rule and chooses the variable with the most negative reduced cost to enter the basis. Unfortunately, the simplex method implemented with the Dantzig rule can cycle. Bland’s pivot rule [3] defines a choice of entering and leaving variable that precludes the possibility of cycling. However, an implementation of the simplex method with Bland’s rule usually requires substantially more iterations to solve an LP than an implementation based on the Dantzig rule. One variation of Dantzig rule is the steepest-edge algorithm [7, 16]. It chooses an entering index with the largest rate of change in the objective. The steepest-edge simplex method can be efficient in terms of reducing the number of simplex iterations, but can also cause the simplex method to cycle.

In the 71 years since the simplex method was first proposed, two other types of method for solving a linear program have been proposed. The first of these is the interior-point method. Interest in these methods was initiated by Karmarkar’s announcement [17] in 1984 of a polynomial-time linear programming method that was 50 times faster than the simplex method. Amid the subsequent flurry of interest in Karmarkar’s method, it was shown in 1985 [11] that there is a formal equivalence
between Karmarkar’s method and the classical logarithmic barrier method applied to linear programming. This resulted in many long-discarded barrier methods being proposed as polynomial-time algorithms for linear programming. One of the main features of an interior method is that every iterate lies in the strict interior of the feasible set, which implies that there is no basis defined. In the past thirty years, many efficient implementations of the interior-point method and its variants have been developed \[18, 20–22\]. Extensive numerical results indicate that robust and efficient implementations of the interior-point method can solve many large-scale LP problems. As all the iterates lie inside the feasible set, it is not possible for an interior-point method to cycle.

This paper will focus on the second alternative to the simplex method, which is an active-set method that does not necessarily move from vertex to vertex. This method is similar to the simplex method in the sense that it uses a basis with independent columns. However, unlike the simplex method, the basis matrix does not have to be square.

Instead of solving (LP) directly by using an active-set method, we consider methods based on solving a non-negative least-squares subproblem of the form

\[
\text{(NNLS)} \quad \begin{array}{ll}
\text{minimize} & \|A_{\text{e}}x - b\|^2 \\
\text{subject to} & x \geq 0,
\end{array}
\]

where \(\cdot\) denotes the two-norm of a vector, and \(A_{\text{e}}\) denotes a working basis matrix.

2. Notation, Definitions and Theorems

2.1. Notation

Unless explicitly indicated otherwise, \(\|\cdot\|\) denotes the vector two-norm or its induced matrix norm. Given vectors \(a\) and \(b\) with the same dimension, \(\min(a, b)\) is a vector with components \(\min(a_i, b_i)\). The vectors \(e\) and \(e_j\) denote, respectively, the column vector of ones and the \(j\)th column of the identity matrix \(I\). The dimensions of \(e\), \(e_i\) and \(I\) are defined by the context. Given vectors \(x\) and \(y\), the column vector consisting of the components of \(x\) augmented by the components of \(y\) is denoted by \((x, y)\).

2.2. Definitions

**Definition 2.1.** The primal and dual linear programs associated with problem (LP) are

\[
\begin{align*}
\text{(P)} & \quad \text{minimize} \quad c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0, \\
\text{(D)} & \quad \text{maximize} \quad b^T y \quad \text{subject to} \quad A^T y \leq c,
\end{align*}
\]

where (D) is the dual of problem (P).
Definition 2.2. (Feasible direction) Let $\bar{y}$ be a feasible point for the linear inequality constraints $A^T y \leq c$. A dual feasible direction at $\bar{y}$ is a vector $p$ satisfying

$$p \neq 0 \quad \text{and} \quad A^T(\bar{y} + \alpha p) \leq c \quad \text{for some} \quad \alpha > 0.$$ 

2.3. Theorems and Lemmas

Theorem 2.1. (Farkas’ Lemma) Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following two systems is feasible:

1. $\{ x : Ax = b, \ x \geq 0 \}$,
2. $\{ y : A^T y \leq 0, \ b^T y > 0 \}$.

Theorem 2.2. (Dual active-set optimality conditions for an LP)

A point $x^*$ is a solution of (LP) if and only if $Ax^* = b, A_B x_B^* = b$ with $x_B^* \geq 0$, and there exists a $y^*$ such that $A^T y^* \leq c$, where $A_B$ is the active-constraint matrix at $y^*$.

3. Primal-Dual Non-Negative Least Squares

In this section we present a modification of the method of Barnes et al. [1] to solve the linear program (LP). The primal-dual non-negative least-squares method (PDNNLS) is based on solving the dual problem (D) instead of the primal problem (P). A strict feasible ascent direction for the dual is found by using the residual vector from the solution of a non-negative least squares sub-problem (NNLS). A suitable step along this direction will increase the objective function of (D) at each iteration.

The main algorithm PDNNLS is described in section 3.1.2 and its corresponding formulation of phase 1 problem is described in Section 3.1.3. In Section 3.2 we present methods for solving (NNLS) subproblem that can be used in PDNNLS.

3.1. Solving the Linear Program

3.1.1. Finding a feasible ascent direction

Theorem 3.1. Let $x^*$ be the solution of the problem

$$(\text{NNLS}) \quad \text{minimize} \ \frac{1}{2} \| A_B x - b \|^2 \quad \text{subject to} \quad x \geq 0,$$

where $A_B$ is the submatrix corresponding to $\mathcal{B}$, and $|\mathcal{B}| = t$. Let $\Delta y = b - A_B x^*$. If $\Delta y \neq 0$ then $\Delta y$ is a feasible ascent direction for (D), i.e. $b^T \Delta y > 0$.

Proof. The Lagrangian of (NNLS) with $y$ as the multiplier is given by

$$L(x, y) = \frac{1}{2} (A_B x - b)^T (A_B x - b) - y^T x.$$
The Kuhn-Tucker optimality conditions imply that

\[
A_B^T(A_Bx^* - b) = y, \quad (3.1a) \\
x^* \geq 0, \quad (3.1b) \\
y \geq 0, \quad (3.1c) \\
y^T x^* = 0. \quad (3.1d)
\]

The optimality conditions for (NNLS) are then given by: (i) the feasibility conditions (3.1b); (ii) the nonnegativity conditions (3.1c) for the multipliers associated with the bounds \( x \geq 0 \); (iii) the stationarity conditions (3.1a); and (iv) the complementarity conditions (3.1d). Then the following holds,

\[
A_B^T(A_Bx^* - b) \geq 0, \quad (3.2a) \\
x^T A_B^T(A_Bx^* - b) = 0. \quad (3.2b)
\]

From the definition of \( \Delta y \), condition (3.2a) can be rewritten as \( A_B^T \Delta y \leq 0 \). The condition (3.2b) then gives

\[
b^T \Delta y = b^T \Delta y + x^T A_B^T(A_Bx^* - b) \\
= -b^T (A_Bx^* - b) + x^T A_B^T(A_Bx^* - b) \\
= (A_Bx^* - b)^T(A_Bx^* - b) \\
= \|A_Bx^* - b\|^2 > 0.
\]
3.1.2. The PDNNLS algorithm

The motivation for the algorithm PDNNLS is to solve problem (NNLS) at each iteration and use $\Delta y = b - A_B x^*$ as an ascent direction for the dual problem. The algorithm is described below.

**Algorithm 1 Primal-Dual Non-Negative Least Squares (PDNNLS).**

1: input $A, b, c$, and a dual feasible point $y$;
2: while not convergent do
3:   Let $B$ be the active set for $A^T y \leq c$, so that $A^T B y = c_B$;
4:   Solve (NNLS) for $x^*$;
5:   if $A_B x^* = b$ then
6:     stop; \hspace{1cm} [(x^*, y^*) is optimal]
7:   else
8:     $\Delta y \leftarrow b - A_B x^*$;
9:     end if
10:  if $A^T N y \leq 0$ then
11:     stop; \hspace{1cm} [The dual is unbounded below]
12:  else
13:     $\alpha \leftarrow \min_{i: i \in N, a_i^T \Delta y > 0} \frac{c_i - a_i^T y}{a_i^T \Delta y}$;
14:     end if
15:  $y \leftarrow y + \alpha \Delta y$;
16: end while
17: return $x, y$;

3.1.3. Finding a feasible point

If $c \geq 0$, then $y_0 = 0$ is feasible for the constraints $A^T y \leq c$ and $y_0$ can be used to initialize Algorithm PDNNLS. Otherwise, an initial dual feasible point can be computed by solving the phase 1 linear program:

$$
\begin{align*}
\min_{y \in \mathbb{R}^m, \theta \in \mathbb{R}} & \quad \theta \\
\text{subject to} & \quad A^T y + \theta e \leq c, \quad \theta \leq 0.
\end{align*}
$$

An initial feasible point for this LP is given by $(y_0, \theta_0)$, where $y_0 = 0$ and $\theta_0 = \min_i c_i$. Consider the quantities $\tilde{A} \in \mathbb{R}^{(m+1) \times (n+1)}$, $\tilde{b} \in \mathbb{R}^{m+1}$, $\tilde{c} \in \mathbb{R}^{n+1}$ and initial dual feasible point $\tilde{y}_0$ such that

$$
\tilde{A} = \begin{pmatrix} A & 0 \\ e^T & 1 \end{pmatrix}, \quad \tilde{b} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \tilde{c} = \begin{pmatrix} c \\ 0 \end{pmatrix}, \quad \tilde{y}_0 = \begin{pmatrix} y_0 \\ \theta_0 \end{pmatrix}.
$$

Use $\tilde{A}$, $\tilde{b}$, $\tilde{c}$, and $\tilde{y}_0$ as input for (PDNNLS). If the associated dual $(D)$ is feasible, then the phase 1 problem will have an optimal solution with zero objective value.
and final iterate \( \bar{y} = (y_1, 0) \). If the optimal phase 1 objective is nonzero the dual is infeasible and the primal (P) is unbounded. If phase 1 terminates with a feasible \( y_1 \), the original coefficient matrix \( A \), cost vector \( c \), and the bound constraint vector \( b \) can be used as input for a second phase to solve using (PDNNLS).

3.2. Solving the NNLS Sub-problem

Over the years, various methods have been developed for solving NNLS. Generally, approaches can be divided into three broad categories. The first includes iterative methods such as gradient projection methods and barrier methods. The second category includes methods based on transforming NNLS into a least-distance problem that is solved as a linear inequality constrained least-squares problem. The third category includes other active-set methods that do not transform NNLS into an equivalent least-distance problem.

In this section we focus on active-set methods. Specifically, we present the classical Lawson and Hanson algorithm and its improved version known as the fast non-negative least squares method (FNNLS).

3.2.1. Optimality conditions for NNLS

The classical NNLS algorithm was proposed by Lawson and Hanson \([19]\) in 1974 for solving the following problem

\[
\begin{align*}
\text{minimize} & \quad \|Ax - b\| \\
\text{subject to} & \quad Dx \geq f,
\end{align*}
\]

where \( A \in \mathbb{R}^{m \times n} \) and \( b \in \mathbb{R}^m \). This problem is equivalent to NNLS if \( D = I \) and \( f = 0 \). Consider the problem

\[
\begin{align*}
\text{minimize} & \quad \|Ax - b\| \\
\text{subject to} & \quad x \geq 0.
\end{align*}
\]

which is equivalent to the problem

\[(\text{NNLS}) \quad \text{minimize} \quad \frac{1}{2}\|Ax - b\|^2 \quad \text{subject to} \quad x \geq 0.\]

Proposition 3.1. (Kuhn-Tucker Conditions) A vector \( x \in \mathbb{R}^n \) is an optimal solution for problem (NNLS) if and only if there exists a \( y \in \mathbb{R}^n \) satisfying

\[
\begin{align*}
A^T(Ax - b) &= y, \quad (3.3a) \\
x &\geq 0, \quad (3.3b) \\
y &\geq 0, \quad (3.3c) \\
x^Ty &= 0. \quad (3.3d)
\end{align*}
\]

Proof. The Lagrangian for (NNLS) is

\[
L(x, y) = (Ax - b)^T(Ax - b) - y^T x.
\]
Stationary of the Lagrangian with respect to \( x \) implies that
\[
A^T(Ax - b) - y = 0.
\] (3.4)

The optimality conditions for (NNLS) are then given by: (i) the feasibility conditions (3.3b); (ii) the nonnegativity conditions (3.3c) for the multipliers associated with the bounds \( x \geq 0 \); (iii) the stationarity conditions (3.3a); and (iv) the complementarity conditions (3.3d).

3.2.2. The Lawson and Hanson method

The NNLS algorithm is described below. The inputs include a matrix \( A \in \mathbb{R}^{m \times n} \) and a vector \( b \in \mathbb{R}^m \). The dual vector \( y \) is calculated and the corresponding working set \( B \) and non-working set \( N \) are defined and modified each iteration. The working set is changed as follows: in the main loop, the algorithm calculates the dual vector \( y \), selects the most negative index of the dual vector and adds it to \( B \). In the inner loop, the algorithm tries to satisfy the primal feasibility condition (3.3b) by finding a vector \( z \) such that its subvector \( z_B \) corresponding to \( B \) solves the least-square problem while its subvector \( z_N \) corresponding to \( N \) will always be held at value zero. Inside the primal feasibility loop, \( x_N \) will always be zeros while \( x_B \) will be free to take values other than zero. If there is any variable \( x_i < 0, \ i \in B \), the algorithm will either take a step to change it to a positive value, or set the variable to 0 and move it to \( N \).

**Algorithm 2** Non-Negative Least Squares.

```plaintext
\( B \leftarrow \emptyset; \ N \leftarrow \{1, \ldots, n\}; \ x \leftarrow 0; \) \quad \text{[Fix all variables on their bounds]}
\( y \leftarrow A^T(Ax - b); \)

while \( N \neq \emptyset \) and \( \exists i : y_i < 0 \) do
  \( l \leftarrow \text{argmin}_{i \in N} y_i; \)
  \( B \leftarrow B \cup l, \ N \leftarrow N \setminus l; \)
  \( z_N \leftarrow 0 \) and find \( z_B \) by solving minimize \( \|A_B z - b\|_2 \);

while \( \exists i \in B : z_i < 0 \) do
  \( k \leftarrow \text{argmin}_{i \in B, z_i \leq 0} x_i/(x_i - z_i); \)
  \( \alpha \leftarrow x_k/(x_k - z_k); \)
  \( x \leftarrow x + \alpha(z - x); \)
  \( \mathcal{E} \leftarrow \{i : x_i = 0\}; \)
  \( B \leftarrow B \setminus \mathcal{E}; \ N \leftarrow N \cup \mathcal{E}; \)
  \( z_N \leftarrow 0 \) and find \( z_B \) by solving minimize \( \|A_B z - b\|_2 \);
end while
\( x \leftarrow z; \)
\( y \leftarrow A^T(Ax - b); \)
end while

return \( x, y; \)
```
Proposition 3.2. On termination of Algorithm 2, the solution $x$ and the dual vector $y$ satisfy

$$
\begin{align*}
x_i &> 0, \quad y_i = 0, \quad i \in B, \\
x_i &> 0, \quad y_i \geq 0, \quad i \in N.
\end{align*}
$$

Proof. The resulting solution vector $x$ after the primal feasibility loop will satisfy $x_N = 0$ and $x_B > 0$, where $x_B$ is the solution of

$$
\text{minimize} \quad \|A_B x - b\|.
$$

The vector $x_B$ can be written as $x_B = A_B^\dagger b$, where $A_B^\dagger$ is the Moore-Penrose pseudoinverse of $A_B$. (If $A_B$ has full column rank then $A_B^\dagger = (A_B^T A_B)^{-1} A_B^T b$.) Then

$$
Ax - b = A_B x_B + A_N x_N - b = A_B A_B^\dagger b - b = -(I - A_B A_B^\dagger) b,
$$

and the subvector of $y$ corresponding to $B$ is

$$
y_B = A_B^T (Ax - b) = -A_B^T (I - A_B A_B^\dagger) b = 0,
$$

because the vector $(I - A_B A_B^\dagger) b$ lies in the null space of $A_B^T$. The termination of the main loop implies that $y_N \geq 0$, which implies that $y \geq 0$. }

The NNLS algorithm requires a finite number of iterations to obtain a solution. For the finiteness of NNLS algorithm, we refer readers to [19] for more detail. However, the significant cost of computing the pseudoinverse $A_B^\dagger$ in Steps 6 and 13 implies that the method is slow in practice.
3.2.3. Fast NNLS algorithm

Many improved algorithms have been suggested since 1974. Fast NNLS (FNNLS) [4] is an optimized version of the Lawson and Hanson’s algorithm which implemented in the MATLAB routine lsqnonneg. It uses precomputed cross-product matrices in the normal equation formulation and used them repeatedly.

Algorithm 3 Fast non-negative least squares.

Choose \( x \); Compute \( N = \{ i : x_i = 0 \} \) and \( B = \{ i : x_i > 0 \} \);

repeat

repeat

\( z_B = \text{argmin}_z \| A_B z - b \|; \quad z_N = 0 \);

if \( \min_{j \in B} z_j < 0 \) then

\( k \leftarrow \text{argmin}_{i \in B, z_i < 0} x_i / (x_i - z_i) \);

\( \alpha \leftarrow x_k / (x_k - z_k) \); [Compute the maximum feasible step]

\( N \leftarrow N \cup \{ k \}; \quad B \leftarrow B \setminus \{ k \}; \) [fix \( x_k \) on its bound]

\( x \leftarrow x + \alpha (z - x) \);

else

\( x = z \);

end if

until \( x = z \)

\( y \leftarrow A^T (Ax - b) \);

\( y_{\min} \leftarrow \min_{i \in N} y_i; \quad l \leftarrow \text{argmin}_{i \in N} y_i \);

if \( y_{\min} < 0 \) then

\( B \leftarrow B \cup \{ l \}; \quad N \leftarrow N \setminus \{ l \}; \) [free \( x_l \) from its bound]

end if

until \( y_{\min} \geq 0 \)

return \( x, y \);

4. Mixed Constraints

The idea of solving linear programming by solving a least-squares subproblem for the dual feasibility can be extended to problems with a mixture of equality and inequality constraints. Consider the following problem

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad a_i^T x = b_i, \quad \text{for } i \in \mathcal{E}, \quad a_i^T x \geq b_i, \quad \text{for } i \in \mathcal{I},
\end{align*}
\]

(4.1)

where \( \mathcal{E}, \mathcal{I} \) are partitions of \( \mathcal{M} = \{ 1, \ldots, m \} \) with \( \mathcal{E} \cap \mathcal{I} = \emptyset \). The quantity \( \mathcal{E} \) denotes the index set of the equality constraints, and it is assumed that \( |\mathcal{E}| = m_1 \). Similarly, \( \mathcal{I} \) denotes the index set of the inequality constraints, with \( |\mathcal{I}| = m_2 \). Also, let \( \mathcal{B} \) denote the indices of a basis. Let \( A \) be an \( m \times n \) matrix with \( i \)-th row vector given by \( a_i^T \). Let \( A_E \) and \( A_I \) denote the matrices consisting of rows of \( A \) with indices in \( \mathcal{E} \)
and \( I \), respectively. Therefore, (4.1) is equivalent to

\[
\begin{align*}
\text{(P}_{\text{MC}}) & \quad \text{minimize} \quad c^T x \\
& \quad \text{subject to} \quad A_E x = b_E, \quad A_I x \geq b_I.
\end{align*}
\]

4.1. Solving the Mixed-Constraint LP

4.1.1. Optimality conditions

**Proposition 4.1.** A vector \( x \in \mathbb{R}^n \) is an optimal solution for problem (P_{MC}) if and only if there exists a \( y \in \mathbb{R}^m \) such that

\[
\begin{align*}
A_E x &= b_E, \quad (4.2a) \\
A_I x &\geq b_I, \quad (4.2b) \\
A^T y &= c, \quad (4.2c) \\
y_I &\geq 0, \quad (4.2d) \\
y_I^T (A_I x - b_I) &= 0, \quad (4.2e)
\end{align*}
\]

where \( y_E \) and \( y_I \) are the vectors of components of the Lagrangian multiplier \( y \) associated with \( E \) and \( I \), respectively.

The Lagrangian is given by

\[
L(x, y) = c^T x - y_E^T (A_E x - b_E) - y_I^T (A_I x - b_I)
\]

\[
= x^T (c - A_E^T y - A_I^T y_I) + b_E^T y_E + b_I^T y_I
\]

\[
= b_E^T y_E + b_I^T y_I + x^T (c - A^T y)
\]

\[
= b^T y - x^T (A^T y - c),
\]

which implies that the dual problem is

\[
\text{(D}_{\text{MC}}) \quad \text{maximize} \quad \frac{b^T y}{y \in \mathbb{R}^m} \\
\text{subject to} \quad A^T y = c, \quad y_I \geq 0.
\]

From the formulation of (D_{MC}), the following proposition holds.

**Proposition 4.2.** Let \( B \) denote a working basis. If \( y_i \geq 0 \) for \( i \in I \cap B \), and \( y_i = 0 \) for \( i \in N \), then \( y \) solves problem (D_{MC}).

**Proof.** The complementary slackness conditions (4.2e) are satisfied when \( a_i^T x - b_i = 0 \), for \( i \in I \cap B \). The non-negativity of the multipliers associated with the inequality constraints (4.2d) requires \( y_i \geq 0 \) for \( i \in I \cap B \). For \( i \in M \setminus B \) it holds that \( a_i^T x \neq b_i \) and \( y_i \) must be zero to satisfy the complementarity conditions.

By Proposition 4.2, the mixed constraint problem (4.1) is equivalent to solving the bound constrained linear least-squares subproblem

\[
\text{(BLS)} \quad \text{minimize} \quad \| A_{p}^T y - c \|^2 \\
\text{subject to} \quad y_i \geq 0, \quad i \in I \cap B.
\]
14 Anti-Cycling Methods for Linear Programming

If the subvector $y^*_B$ is the optimal solution of problem (BLS) and $y^*_N = 0$, then $y^*$ is the optimal solution of problem (D_{MC}). Otherwise, we find a feasible descent direction for the primal problem (P_{MC}) and update the current iterate. An appropriate method for solving problem (BLS) is described in Section 4.2.

4.1.2. Finding a descent direction

The method for solving a problem with mixed constraints uses two different types of descent direction. Each of these directions is the residual vector at the solutions of a linear least-squares problem.

A Type-1 descent direction is computed by solving the unconstrained least-squares problem

$$
\min_{y \in \mathbb{R}^t} \| A_B^T y - c \|^2.
$$

If $y^*$ is the optimal solution of this problem, then $\Delta x = A_B^T y^* - c$ is a feasible descent direction for the primal. For a proof, we refer the reader to [10]. The search direction may be regarded as a gradient descent direction in range($A_B$).

A Type-2 descent direction is found by solving the bound-constrained problem (BLS). To establish that the optimal residual of this problem is a feasible descent direction, we first state the optimality conditions of (BLS).

**Proposition 4.3.** A vector $y \in \mathbb{R}^t$ is an optimal solution for problem (BLS) if and only if there exists a vector $\lambda \in \mathbb{R}^n$ satisfying the following conditions.

$$
\begin{align*}
  y_i & \geq 0, \quad i \in I \cap B, \\
  A_B(A_B^T y - c) & = \lambda, \\
  \lambda & \geq 0, \\
  \lambda^T y & = 0.
\end{align*}
$$

The motivation for finding the feasible descent direction is similar to the idea described in Section 3.1. We use the residual vectors as the search direction. The following proposition proves that such direction is a feasible descent direction for the (P_{MC}).

**Proposition 4.4.** Assume that $y^*$ solves (BLS) and let $r = A_B^T y^* - c$. If $r = 0$, then $x$ solves the mixed-constraint linear program (4.1). Otherwise, $r$ is a feasible descent direction for (P_{MC}).

**Proof.** Assume that $r = 0$. Let $p$ be a feasible direction, i.e., $p$ satisfies $a_i^T p \geq 0$ for $i \in I \cap B$ and $a_i^T p = 0$ for $i \in E$. Then

$$
c^T p = (A_B^T y^*)^T p = y^* (A_B^T p) = \sum_{i \in I \cap B} y^*_i (a_i^T p) + \sum_{i \in E} y^*_i (a_i^T p) = \sum_{i \in I \cap B} y^*_i (a_i^T p) \geq 0.
$$

It follows that every feasible direction is a direction of increase for the objective, which implies that $x$ solves (4.1).
If \( r \neq 0 \), it follows directly from Proposition 4.3 that

\[
A_B r = A_B (A_B^T y^* - c) = \lambda \geq 0.
\]

Also, since \( c = A_B^T y^* - r \), and conditions (4.3b) and (4.3d) hold, we have

\[
c^T r = (A_B^T y^* - r)^T r = y^T A_B r - \| r \|^2 = y^T \lambda - \| r \|^2 = -\| r \|^2 < 0.
\]

This proves the claim. \( \blacksquare \)

### 4.1.3. The algorithm

Algorithm 4 below gives the steps of the algorithm for solving an LP with mixed constraints. In the main loop, a Type-1 direction is used until the stationarity conditions are satisfied, at which point the inner loop is entered. A Type-2 descent direction is found in the inner loop. Then the algorithm tests if the problem is unbounded above by checking if the search direction \( \Delta y \) satisfies \( A_B^T \Delta y \geq 0 \). The step size \( \alpha \) is then found by calculating the largest step along \( \Delta y \) that maintains the dual feasibility.

The use of a mixture of Type-1 and Type-2 directions significantly reduces the number of times that it is necessary to solve problem (BLS). A similar scheme can be applied to problem PDNNLS to improve the efficiency.

**Algorithm 4** Mixed Constraints.

```
Input A, b, c, a feasible point x, and I
while not converged do
    Let B be the active set such that \( A_B x = b_B \);
    Solve minimize \( \| A_B^T y - c \|^2 \) for y;
    if \( A_B^T y = c \) then
        if \( y_i \geq 0, i \in I \cap B \) then
            Stop; \[optimal solution\]
        end if
    end if
    Compute y, the solution of
    minimize \( y \in \mathbb{R}^t \) \( \| A_B^T y - c \|^2 \) subject to \( y_i \geq 0, i \in I \cap B \);
    end if
    \( \Delta x \leftarrow A_B^T y - c \);
    if \( A_N \Delta x \geq 0 \) then
        stop; \[unbounded above\]
    else
        \( \alpha \leftarrow \min_{i : i \in N, a_i^T \Delta x < 0} \frac{b_i - a_i^T x}{a_i^T \Delta x} \);
    end if
    \( x \leftarrow x + \alpha \Delta x \);
end while
return x, y;
```
4.1.4. Finding a feasible point

If necessary, a phase 1 linear program must be solved for an initial feasible point. Consider choosing $\mathcal{B}$ such that $|\mathcal{B}| \leq n$ and $\mathcal{E} \subseteq \mathcal{B}$. One choice of such $\mathcal{B}$ is to let $\mathcal{B} = \mathcal{E}$. Let $b_\mathcal{B}, b_\mathcal{E}$ be subvectors of $b$ associated with $\mathcal{B}$ and $\mathcal{E}$. The initial $x_0$ is calculated by solving $A_\mathcal{B}x_0 = b_\mathcal{B}$. If $A_\mathcal{E}x_0 \geq b_\mathcal{E}$, then there is no need to solve a Phase 1 problem, i.e., $x_0$ is feasible. Otherwise, we define the auxiliary variable $\theta_0$ as

$$
\theta_0 = -\min_{i \in \mathcal{N}} (a_i^T x_0 - b_i).
$$

The phase 1 problem for mixed constraints is

$$
\begin{align*}
\text{minimize} & \quad \theta \\
\text{subject to} & \quad A_E x = b_E, \quad A_I x + \theta e \geq b_I, \quad \theta \geq 0.
\end{align*}
$$

Define $\bar{A} \in \mathbb{R}^{(m+1)\times(n+1)}$, $\bar{b} \in \mathbb{R}^{m+1}$, $\bar{c} \in \mathbb{R}^{n+1}$ and initial feasible point $\bar{x}_0$ such that

$$
\bar{A} = \begin{pmatrix} A & e_N \\ 0 & 1 \end{pmatrix}, \quad \bar{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad \bar{c} = \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix}, \quad \bar{x}_0 = \begin{pmatrix} x_0 \\ \theta_0 \end{pmatrix},
$$

where $e_N \in \mathbb{R}^m$ with $[e_N]_i = 1$ if $i \in \mathcal{N}$ and 0 otherwise.

If the optimal solution $(x^*, \theta^*)$ satisfies $\theta^* = 0$, then $x^*$ is a feasible point for $(P_{MC})$. Then we use $x^*$ as the initial feasible point for the phase 1 problem.

4.2. Solving Bounded Least Square Problems

In this section, we describes two methods for solving the (BLS). The method in Section 4.2.1 uses the singular value decomposition, and transforms the (BLS) to a least distance problem (LDP). It also takes the advantages of the classical (NNLS). In section 4.2.2, we present a modified method of (NNLS) based on Dax’s paper [6].

4.2.1. Solving BLS using the singular-value decomposition

Let $D$ be a $t \times t$ diagonal matrix with $D_{ii} = 1$ if $i \in \mathcal{I} \cap \mathcal{B}$ and $D_{ii} = 0$, otherwise. Therefore, (BLS) is equivalent to

$$
\begin{align*}
\text{minimize} & \quad \|A_B^T y - c\|^2 \\
\text{subject to} & \quad Dy \geq 0.
\end{align*}
$$

One way to solve (BLS) is to solve the least-distance problem (LDP) in the following form in instead,

$$
\begin{align*}
\text{(LDP)} \quad \text{minimize} & \quad \|z\| \\
\text{subject to} & \quad E z \geq f.
\end{align*}
$$
Proposition 4.5. Solving (BLS) is equivalent to solving (LDP).

Proof. Given $A_B$ is a matrix of dimension $t \times n$, the singular value decomposition gives

$$A_B^T = \hat{U} \Sigma V^T = (U_1 \quad U_2) \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T,$$

where $U$ and $V$ are orthogonal with dimension $n \times n$ and $t \times t$ respectively. Besides, $U_1$ is of dimension $n \times t$, $U_2$ is of dimension $n \times (n-t)$, and $\Sigma$ is a diagonal matrix of dimension $t \times t$. The objective function of (BLS) can be written as

$$\| A_B^T y - c \|^2 = \| U \Sigma V^T y - c \|^2 = \| U \Sigma V^T y - U^T c \|^2 = \| \begin{pmatrix} \Sigma V^T y \\ 0 \end{pmatrix} - \begin{pmatrix} U_1^T c \\ U_2^T c \end{pmatrix} \|^2 = \| \Sigma V^T y - U_1^T c \|^2 + \| U_2^T c \|^2 = \| z \|^2 + \| U_2^T c \|^2,$$

where $z = \Sigma V^T y - U_1^T c$. By change of variables, $y = V \Sigma^{-1} (z + U_1^T c)$, the constraint of (BLS) becomes

$$DV \Sigma^{-1} z \geq -DV \Sigma^{-1} U_1^T c.$$

If the constant second term $\| U_2^T c \|^2$ is omitted, then (LDP) and (BLS) are equivalent for $E = DV \Sigma^{-1}$ and $f = -DV \Sigma^{-1} U_1^T c$. 

The classical NNLS algorithm can be used to compute the solution for (LDP). Instead of computing $z$ directly, we can compute a $t$ vector $\hat{u}$ solves the problem

$$\min_{u \in \mathbb{R}^t} \left\| \begin{pmatrix} E^T \\ f^T \end{pmatrix} u - e_{t+1} \right\|^2 \text{ subject to } u \geq 0,$$

where $e_{t+1}$ is a $t+1$ vector with everywhere zeros except its $(t+1)$ entry. Further, if $r = \begin{pmatrix} E^T \\ f^T \end{pmatrix} \hat{u} - e_{t+1}$, then the following proposition holds.

Proposition 4.6. If $z_i = r_i / r_{t+1}$, $i = 1, 2, \ldots, t$, then $z$ solves (LDP).

Proof. The gradient of the objective function $\frac{1}{2} \left\| \begin{pmatrix} E^T \\ f^T \end{pmatrix} u - e_{t+1} \right\|^2$ at $\hat{u}$ is

$$g = (E \quad f) \left\{ \begin{pmatrix} E^T \\ f^T \end{pmatrix} \hat{u} - e_{t+1} \right\} = (E \quad f) r.$$

From the KKT condition for the NNLS problem, the complementary slackness condition (3.3d) requires $g^T \hat{u} = 0$. Then

$$\| r \|^2 = r^T r = r^T \left\{ \begin{pmatrix} E^T \\ f^T \end{pmatrix} \hat{u} - e_{t+1} \right\} = g^T \hat{u} - r_{t+1} = r_{t+1} > 0.$$
As \( z_i = -r_i/r_{t+1} \), for \( i = 1, 2, \ldots, t \), we have

\[
  r = -r_{t+1} \begin{pmatrix} z \\ -1 \end{pmatrix}.
\]

The dual feasibility condition (3.3c) requires \( g \geq 0 \),

\[
g = (E \ f) r = - (E \ f) \begin{pmatrix} z \\ -1 \end{pmatrix} r_{t+1} = (Ez - f)(-r_{t+1}) \geq 0.
\]

The strict inequality \( r_{t+1} > 0 \) implies that \( Ez \geq f \), which establishes the feasibility of constraints.

It remains to show that \( z \) is the minimizer. The KKT conditions for (LDP) require that \( z \) must be a nonnegative linear combination of rows of \( E \).

\[
z = - \frac{1}{r_{n+1}} \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} = - \frac{1}{r_{n+1}} E^T \hat{u} = \frac{1}{\|r\|^2} E^T \hat{u},
\]

where \( \hat{u} \geq 0 \) from the primal feasibility condition of the non-negative least squares problem (3.3b). Therefore, \( z \) is the solution of Problem (LDP).

By change of variable, i.e., \( y = V\Sigma^{-1}(z + U_1^T c) \), (LDP) can be solved as well. The BLS algorithm using NNLS is described below.

**Algorithm 5** Bounded Least Square-NNLS.

- **Input** \( A_B, c \);
- Calculate the economy-size singular value decomposition of \( A_B^T \)
  
  \[
  A_B^T = U_1\Sigma V^T;
  \]

  \( E \leftarrow DV\Sigma^{-1} \); \( f \leftarrow -DV\Sigma^{-1}U_1^T c \);

  Use NNLS to compute \( \hat{u} \), the solution of

  \[
  \text{minimize}_{u \in \mathbb{R}^t} \left\| \begin{pmatrix} E^T \\ f^T \end{pmatrix} u - e_{t+1} \right\|^2 \text{ subject to } u \geq 0;
  \]

  \( r \leftarrow \begin{pmatrix} E^T \\ f^T \end{pmatrix} \hat{u} - e_{t+1} \);

- **if** \( r = 0 \) **then**
  
  stop; \( [Ez \geq f \text{ is not compatible}] \)

  **else**

  \( z_i \leftarrow -r_i/r_{t+1}, i = 1, \ldots, t \);

  **end if**

  \( y \leftarrow V\Sigma^{-1}(z + U_1^T c) \);

  **return** \( y \);
This algorithm performs well for low dimensional problems. However, since computing the singular value decomposition involves approximately \( m^2n + n^3 \) floating-point operations for a matrix of dimension \( m \times n \), it is not efficient to compute the singular-value decomposition from \( A_B^T \) directly. Instead, we use the economy-size QR factorization in the outer loop if \( n \gg t \) and compute the singular value decomposition of the square upper-triangular matrix \( R \). This gives the decomposition
\[
R = U\Sigma V^T,
\]
which implies that \( A_B^T \) can be written as
\[
A_B^T = Q_1R = Q_1\bar{U}\Sigma V^T = U_1\Sigma V^T, \quad \text{with} \quad U_1 = Q_1\bar{U}.
\]
The singular-value decomposition of \( R \) requires \( 2t^3 \) floating-point operations.

### 4.2.2. Dax’s method for BLS

Without loss of generality, consider solving the (BLS) in the following form
\[
\text{(BLS)} \quad \begin{array}{ll}
\text{minimize} & \|Ax - b\|^2 \\
\text{subject to} & x_i \geq 0, \quad i \in \mathcal{I},
\end{array}
\]
where \( A \in \mathbb{R}^{m \times n} \) and \( \mathcal{I} \subset \{1, 2, \ldots, n\} \). To solve the mixed constraints problems using the following algorithm, simply substitute \( A, x, \) and \( b \) with \( A_B^T, y \) and \( c \) respectively.

From Proposition 4.3 and using an appropriate substitution, the following result holds.

**Proposition 4.7.** If \( x \) is a feasible point for (BLS) and the following equations are satisfied
\[
y_i = 0, \quad \text{if} \quad i \in \mathcal{E}, \\
y_i = 0, \quad \text{if} \quad i \in \mathcal{I} \cap \mathcal{B}, \quad x_i > 0, \\
y_i > 0, \quad \text{if} \quad i \in \mathcal{I} \cap \mathcal{B}, \quad x_i = 0,
\]
then \( x \) is the optimal solution for (BLS).

Dax’s method is given in Algorithm 6 below. The method performs well when there is no degeneracy. However, if a problem is degenerate, a large number of iterations may be required to solve the bounded least squares problem.
Algorithm 6 Bounded least squares.

function BOUNDED_LEAST_SQUARES(A, b, I, x₀)
  Define a feasible point \(x\) as
  \[
  x_i = \begin{cases} 
  0, & [x₀]_i < 0, \ i \in I; \\
  [x₀]_i, & \text{otherwise};
  \end{cases}
  \]

  while not convergent do
    \(\mathcal{N} \leftarrow \{i : x_i = 0, \ i \in I\}; \ \mathcal{B} \leftarrow \{1, \ldots, n\} \setminus \mathcal{N};\)
    \(r \leftarrow Ax - b;\)
    \(\Delta x_N \leftarrow 0 \text{ and find } \Delta x_B \text{ by solving minimize } x \in \mathbb{R}^n \|A_B x - r\|;\)
    if \(\|A \Delta x - r\| \geq \|r\|\) then
      \(y \leftarrow A^T r;\)
      \(y_i^* = \begin{cases} 
      0, & i \in \mathcal{N}, \ y_i > 0; \\
      y_i, & \text{otherwise};
      \end{cases}\)
      if \(y^* = 0\) then
        stop; \quad \text{[x is the optimal solution]}\)
      else
        \(j \leftarrow \arg\max_{i \not\in \mathcal{N}} |y_i^*|; \ \Delta x = -\frac{y_j^*}{\|s_j\|} s_j;\)
      end if
    end if
    Find the largest step \(\alpha \in (0, 1]\) such that \(x + \alpha \Delta x \geq 0;\)
    \(x \leftarrow x + \alpha \Delta x;\)
  end while
  return \(x, y;\)
end function

The algorithm described above starts with a feasible point satisfying \(x_i \geq 0,\) for \(i \in I,\) and calculates the search direction by solving an unconstrained linear least-squares problem. If the search direction is a descent direction, then it is used to perform the line search. Otherwise, a dual vector \(y^*\) is calculated and the optimality conditions of Proposition 4.7 are checked. An auxiliary dual vector \(y\) is used to calculate \(y^*\). If the current point is not optimal, the index of the largest dual violation is selected use to compute a new primal search direction. The step length \(\alpha\) satisfies \(\alpha \in (0, 1],\) which implies that the new iterate always satisfies the primal feasibility condition, i.e., \(x + \alpha \Delta x \geq 0.\)

5. A Primal-Dual Active Set Method for Regularized LP

The method considered in this section is based on applying a quadratic regularization term to the linear program (LP) and solving the resulting regularized problem using a primal-dual quadratic programming (QP) method. (For more details on solving convex quadratic programming problems, see [8].) The regularized linear program
(RLP) is given by
\[
\begin{align*}
\text{minimize} & \quad c^T x + \frac{1}{2} \mu y^T y \\
\text{subject to} & \quad Ax + \mu y = b, \quad x \geq 0,
\end{align*}
\tag{5.1}
\]
where \(0 < \mu < 1\) is a fixed regularization parameter, and \(A, b\) and \(c\) are the quantities associated with the linear program (LP). A solution of the regularized linear program (5.1) is an approximate solution of (LP).

### 5.1. Formulation of the Primal and Dual Problems

For given constant vectors \(q\) and \(r\), consider the pair of convex quadratic programs

\[
\begin{align*}
(\text{PLP}_{q,r}) & \quad \text{minimize} \quad \frac{1}{2} \mu y^T y + c^T x + r^T x \\
\text{subject to} & \quad Ax + \mu y = b, \quad x \geq -q,
\end{align*}
\]
and

\[
\begin{align*}
(\text{DLP}_{q,r}) & \quad \text{maximize} \quad -\frac{1}{2} \mu y^T y + b^T y - q^T z \\
\text{subject to} & \quad A^T y + z = c, \quad z \geq -r.
\end{align*}
\]

(The significance of the shifted constraints \(x \geq -q\) and \(z \geq -r\) is discussed below.)

The following result gives joint optimality conditions for the triple \((x, y, z)\) such that \((x, y)\) is optimal for (PLP\(_{q,r}\)), and \((x, y, z)\) is optimal for (DLP\(_{q,r}\)). If \(q\) and \(r\) are zero, then (PLP\(_{0,0}\)) and (DLP\(_{0,0}\)) are the primal and dual problems associated with (5.1). For arbitrary \(q\) and \(r\), (PLP\(_{q,r}\)) and (DLP\(_{q,r}\)) are essentially the dual of each other, the difference is only an additive constant in the value of the objective function.

### 5.2. Optimality Conditions and the KKT Equations

**Proposition 5.1.** Let \(q \in \mathbb{R}^n\) and \(r \in \mathbb{R}^m\) denote given constant vectors. If \((x, y, z)\) is a given triple in \(\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n\), then \((x, y)\) is optimal for (PLP\(_{q,r}\)) and \((x, y, z)\) is optimal for (DLP\(_{q,r}\)) if and only if

\[
\begin{align*}
Ax + \mu y - b &= 0, \\
\quad c - A^T y - z &= 0, \\
x + q &\geq 0, \\
z + r &\geq 0, \\
(x + q)^T (z + r) &= 0.
\end{align*}
\tag{5.2}
\]

In addition, it holds that \(\text{optval}(\text{PLP}_{q,r}) - \text{optval}(\text{DLP}_{q,r}) = -q^T r\). Finally, (5.2) has a solution if and only if the primal and dual feasible sets

\[
\{ (x, y, z) : A^T y + z = c, \quad z \geq -r \} \quad \text{and} \quad \{ x : Ax + \mu y = b, \quad x \geq -q \}
\]

are both nonempty.
Proof. Without loss of generality, let \( \tilde{y} \) denote the Lagrange multipliers for the constraints \( Ax + \mu y - b = 0 \). Let \( z + r \) be the multipliers for the bounds \( x + q \geq 0 \). With these definitions, a Lagrangian function \( L(x, y, \tilde{y}, z) \) associated with \((\text{PLP}_{q,r})\) is given by

\[
L(x, y, \tilde{y}, z) = (c + r)^T x + \frac{1}{2} \mu y^T y - \tilde{y}^T (Ax + \mu y - b) - (z + r)^T (x + q),
\]

with \( z + r \geq 0 \). Stationarity of the Lagrangian with respect to \( x \) and \( y \) implies that

\[
\begin{align*}
&c + r - A^T \tilde{y} - z - r = c - A^T \tilde{y} - z = 0, \\
&\mu y - \mu \tilde{y} = 0.
\end{align*}
\]

The optimality conditions for \((\text{PLP}_{q,r})\) are then given by: (i) the feasibility conditions \((5.2a)\) and \((5.2c)\); (ii) the nonnegativity conditions \((5.2d)\) for the multipliers associated with the bounds \( x + q \geq 0 \); (iii) the stationarity conditions \((5.3)\); and (iv) the complementarity conditions \((5.2e)\). The vector \( y \) appears only in the term \( \mu y \) of \((5.2a)\) and \((5.3b)\). In addition, \((5.3b)\) implies that \( \mu y = \mu \tilde{y} \), in which case we may choose \( y = \tilde{y} \). This common value of \( y \) and \( \tilde{y} \) must satisfy \((5.3a)\), which is then equivalent to \((5.2b)\). The optimality conditions \((5.2)\) for \((\text{PLP}_{q,r})\) follow directly.

With the substitution \( \tilde{y} = y \), the expression for the primal Lagrangian may be rearranged so that

\[
L(x, y, z) = -\frac{1}{2} \mu y^T y + b^T y - q^T z + (c - A^T y - z)^T x - q^T r.
\]

Substituting \( y = \tilde{y} \) in \((5.3)\), the dual objective is given by \((5.4)\) as

\[
\begin{align*}
&\frac{1}{2} \mu y^T y + b^T y - q^T z - q^T r, \\
&\text{and the dual constraints are} \ c - A^T y - z = 0 \ \text{and} \ z + r \geq 0.
\end{align*}
\]

It follows that \((\text{DLP}_{q,r})\) is equivalent to the dual of \((\text{PLP}_{q,r})\), the only difference is the constant term \(-q^T r\) in the objective, which is a consequence of the shift \( z + r \) in the dual variables. Consequently, strong duality implies \( \text{optval}(\text{PLP}_{q,r}) - \text{optval}(\text{DLP}_{q,r}) = -q^T r \). In addition, the variables \( x, y \) and \( z \) satisfying \((5.2)\) are feasible for \((\text{PLP}_{q,r})\) and \((\text{DLP}_{q,r})\) with the difference in the objective function value being \(-q^T r\). It follows that \((x, y, z)\) is optimal for \((\text{DLP}_{q,r})\) as well as \((\text{PLP}_{q,r})\). Finally, feasibility of both \((\text{PLP}_{q,r})\) and \((\text{DLP}_{q,r})\) is both necessary and sufficient for the existence of optimal solutions. 

If \( \mathcal{I} \) denotes the index set \( \mathcal{I} = \{1, 2, \ldots, n\} \), let \( \mathcal{B} \) and \( \mathcal{N} \) denote a partition of \( \mathcal{I} \). The basic and non-basic variables are defined as the subvectors \( x_B \) and \( x_N \), where \( x_B \) and \( x_N \) denote the components of \( x \) associated with \( \mathcal{B} \) and \( \mathcal{N} \) respectively. A set \( \mathcal{B} \) is associated with a unique solution \((x, y, z)\) that satisfies the equations

\[
\begin{align*}
&c - A_T^T y - z = 0, \quad x_N + q_N = 0, \\
&Ax + \mu y - b = 0, \quad z_B + r_B = 0.
\end{align*}
\]

If \( A_B \) and \( A_N \) denote the matrices of columns of \( A \) associated with \( \mathcal{B} \) and \( \mathcal{N} \) respectively, then the equations above can be written as

\[
\begin{align*}
&c_B - A_B^T y - z_B = 0, \quad x_N + q_N = 0, \\
&c_N - A_N^T y - z_N = 0, \quad z_B + r_B = 0, \\
&A_B x_B + A_N x_N + \mu y - b = 0.
\end{align*}
\]
These equations may be written in terms of the linear system

\[
\begin{pmatrix}
0 & A^T_B \\
A_B & -\mu I
\end{pmatrix}
\begin{pmatrix}
x_B \\
y
\end{pmatrix}
=
\begin{pmatrix}
-c_B - r_B \\
A_N q_N + b
\end{pmatrix}
\]  
(5.8)

by substituting \(x_N = -q_N\) from the equation \(x_N + q_N = 0\). Once \(y\) is known, \(z_N\) can be calculated as \(z_N = c_N - A^T_B y\).

It follows that if the system (5.8) has a unique solution, then the systems (5.5) and (5.6) have a unique solution \((x, y, z)\), and the matrix \(K_B\) is nonsingular. As in Gill and Wong [13], any set \(B\) such that \(K_B\) is nonsingular is known as a second-order consistent basis. The next result shows that a second-order consistent basis may be found by identifying an index set of independent columns of \(A\).

**Proposition 5.2.** If \(A_B\) has linearly independent columns then the matrix \(K_B = \begin{pmatrix} 0 & A^T_B \\ A_B & -\mu I \end{pmatrix}\) is nonsingular.

**Proof.** Suppose that \(A_B \in \mathbb{R}^{n \times r}\). Consider the matrix decomposition

\[
\begin{pmatrix}
0 & A^T_B \\
A_B & -\mu I
\end{pmatrix} = \begin{pmatrix} I_r & \frac{1}{\mu} A^T_B A_B \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\mu} A_B A_B & 0 \\ 0 & -\mu I_m \end{pmatrix} \begin{pmatrix} I_r & 0 \\ -\frac{1}{\mu} A_B & I_r \end{pmatrix}.
\]

The full-rank assumption on \(A_B\) implies that \(A^T_B A_B\) is an \(r \times r\) symmetric positive-definite matrix. Let \(A^T_B A_B = LL^T\) be its corresponding Cholesky decomposition. Then \(K_B\) can be written as

\[
K_B = \begin{pmatrix} I_r & \frac{1}{\mu} A^T_B \\ 0 & I_m \end{pmatrix} \begin{pmatrix} L & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\mu} I_r & 0 \\ 0 & -\mu I_m \end{pmatrix} \begin{pmatrix} I_r & 0 \\ -\frac{1}{\mu} A_B & I_r \end{pmatrix}.
\]

The product of the eigenvalues of \(K_B\) is the product of the eigenvalues of the diagonal entries of each of the five matrices above, which are all nonzero. It follows that \(K_B\) is nonsingular.

The proposed primal-dual method generates a sequence \((x_k, y_k, z_k)\) that satisfies the identities (5.5) and (5.6) at every iteration. In order to satisfy the optimality conditions of Proposition 5.1 the search direction \((\Delta x, \Delta y, \Delta z)\) must always satisfy

\[
A^T \Delta y - \Delta z = 0, \quad \Delta x_N = 0, \quad A \Delta x + \mu \Delta y = 0, \quad \Delta z_B = 0.
\]  
(5.9, 5.10)
The iterates are designed to converge to optimal points that also satisfy the requirements \( x_B + q_B \geq 0 \) and \( z_N + r_N \geq 0 \). This provides the motivation for an algorithm that changes the basis by removing and adding column vectors while maintaining the conditions mentioned above. Consider a new partition \( \mathcal{B} \cup \mathcal{N} \cup \{l\} = \{1, 2, \ldots, n\} \), where the index \( l \) has been removed from either the previous \( \mathcal{B} \) or \( \mathcal{N} \). The conditions (5.9) and (5.10) can be written in matrix form as

\[
\begin{pmatrix}
0 & 0 & a_l^T \\
0 & 0 & A_B^T \\
a_l & A_B & -\mu I
\end{pmatrix}
\begin{pmatrix}
\Delta x_l \\
\Delta x_B \\
-\Delta y
\end{pmatrix}
= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \Delta z_l, \text{ and } \Delta z_N = -A_N^T \Delta y.
\]

In the following discussion we use \( K_l \) to denote the matrix

\[
K_l = \begin{pmatrix}
0 & 0 & a_l^T \\
0 & 0 & A_B^T \\
a_l & A_B & -\mu I
\end{pmatrix}.
\]

**Proposition 5.3.** If \( \Delta z_l > 0 \) and \( K_l \) is nonsingular, then the following holds.

1. If the index \( l \) is such that \( z_l + r_l < 0 \), then \((\Delta x, \Delta y)\) is a strict descent direction for \((\text{PLP}_{q,r})\);

2. If the index \( l \) is such that \( x_l + q_l < 0 \), then \((\Delta y, \Delta z)\) is a strict ascent direction for \((\text{DLP}_{q,r})\).

**Proof.** Without loss of generality, assume that \( \Delta z_l = 1 \). The equations for \( \Delta x_l \), \( \Delta x_B \) and \( \Delta y \) are then

\[
\begin{pmatrix}
0 & 0 & a_l^T \\
0 & 0 & A_B^T \\
a_l & A_B & -\mu I
\end{pmatrix}
\begin{pmatrix}
\Delta x_l \\
\Delta x_B \\
-\Delta y
\end{pmatrix}
= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{5.11}
\]

As \( K_l \) is nonsingular by assumption, \( A_B \) must have full column rank and the matrix \( A_B^T A_B \) is nonsingular. This implies that the solution of (5.11) may be computed as

\[
\Delta x_l = \frac{\mu}{a_l^T P a_l},
\]

\[
\Delta x_B = -\frac{\mu}{a_l^T P a_l} (A_B^T A_B)^{-1} A_B^T a_l,
\]

\[
\Delta y = -\frac{1}{a_l^T P a_l} P a_l,
\]

\[
\Delta z_N = \frac{1}{a_l^T P a_l} A_N^T P a_l,
\]

where \( P \) is the orthogonal projection \( P = I - A_B (A_B^T A_B)^{-1} A_B^T \).

The objective function of primal problem is the quadratic function

\[
f_P(\tilde{p}) = \frac{1}{2} \mu y^T y + c^T x + r^T x = \tilde{g}^T \tilde{p} + \frac{1}{2} \tilde{p}^T \tilde{H} \tilde{p}.
\]
with \( \hat{p} = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \hat{\theta} = \begin{pmatrix} 0 & 0 \\ 0 & \mu I \end{pmatrix} \) and \( \hat{g} = \begin{pmatrix} c + r \\ 0 \end{pmatrix} \). The gradient of \( f_\rho \) at \( \hat{p} \) is

\[
\nabla f_\rho = \hat{g} + \hat{\theta} \hat{p} = \begin{pmatrix} c + r \\ \mu y \end{pmatrix}.
\]

Then the directional derivative at \( \hat{p} \) is

\[
(\nabla f_\rho)^T \hat{p} = (c + r)^T \Delta x + \mu y^T \Delta y
\]

\[
= \frac{\mu}{a_l^T P_{a_l}} (c_l + r_l) - \frac{\mu}{a_l^T P_{a_l}} (c_b + r_b)^T (A_b^T A_b)^{-1} A_b^T a_l - \frac{\mu}{a_l^T P_{a_l}} y^T P_{a_l}
\]

\[
= \frac{\mu}{a_l^T P_{a_l}} ((c_l + r_l) - y^T A_b (A_b^T A_b)^{-1} A_b^T a_l - y^T (I - A_b (A_b^T A_b)^{-1} A_b^T) a_l)
\]

\[
= \frac{\mu}{a_l^T P_{a_l}} ((c_l + r_l) - y^T a_l)
\]

\[
= \frac{\mu}{a_l^T P_{a_l}} ((c_l + r_l) - (c_l - z_l)) = \frac{\mu}{a_l^T P_{a_l}} (z_l + r_l).
\]

The assumption that \( z_l + r_l < 0 \) implies that the directional derivative is negative and \( \hat{p} \) is a descent direction for the primal problem, as required.

Similarly, let \( \overline{p} = \begin{pmatrix} \Delta z \\ \Delta y \end{pmatrix} \), \( \overline{\theta} = \begin{pmatrix} 0 & 0 \\ 0 & \mu I \end{pmatrix} \), \( \overline{g} = \begin{pmatrix} q \\ -b \end{pmatrix} \). The dual objective is then

\[
f_D(\overline{p}) = -\frac{1}{2} \mu y^T y + b^T y - q^T z = -\overline{g}^T \overline{p} - \frac{1}{2} \overline{p}^T \overline{\theta} \overline{p},
\]

which has the gradient

\[
\nabla f_D = -(\overline{g} + \overline{\theta} \overline{p}) = \begin{pmatrix} -q \\ -\mu y + b \end{pmatrix}.
\]

The directional derivative at \( \overline{p} \) is

\[
(\nabla f_D)^T \overline{p} = (-q^T - (\mu y - b)^T) \begin{pmatrix} \Delta z \\ \Delta y \end{pmatrix}
\]

\[
= -\frac{1}{a_l^T P_{a_l}} q_N^T A_N^T P_{a_l} - q_l \Delta z_l - (-\mu y^T + b^T) \frac{1}{a_l^T P_{a_l}} P_{a_l}
\]

\[
= -\frac{1}{a_l^T P_{a_l}} q_N^T A_N^T P_{a_l} - q_l \Delta z_l - (Ax)^T \frac{1}{a_l^T P_{a_l}} P_{a_l}
\]

\[
= -\frac{1}{a_l^T P_{a_l}} (q_N^T A_N^T P_{a_l} + x^T A^T P_{a_l}) - q_l \Delta z_l
\]

\[
= -\frac{1}{a_l^T P_{a_l}} ((q_N + x_N)^T A_N^T P_{a_l} + x_b^T A_b^T P_{a_l} + x_N^T A_N^T P_{a_l} + x_l a_l^T P_{a_l}) - q_l
\]

\[
= -(x_l + q_l) > 0.
\]
The last equality follows from the condition \( q_N + x_N = 0 \) of (5.5), and the identity \( A_B^T \bar{P} = 0 \), which holds because \( \bar{P} \) is the projection onto \( \text{null}(A^T) \).

It is important to maintain the nonsingularity of \( K_l \) by enforcing the linear independence of the columns of \( A_B \). Therefore, if \( l \) is to be added to \( \mathcal{B} \), it may be necessary to remove some indices from \( \mathcal{B} \) before appending \( a_l \) to ensure that \( a_l \) is not in \( \text{range}(A_B) \).

The primal-dual search direction \( (\Delta x_B, \Delta y) \) defined by the equations

\[
\begin{pmatrix}
0 & A_B^T \\
A_B & -\mu I
\end{pmatrix}
\begin{pmatrix}
\Delta x_B \\
-\Delta y
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
a_l
\end{pmatrix}
\tag{5.12}
\]

will always satisfy (5.9) and (5.10) if \( \Delta z_B = 0 \) and \( \Delta x_l = 1 \).

**Proposition 5.4.** If \( A_B \) has linearly independent columns then the search direction \( (\Delta y, \Delta z) \) computed from (5.12) is nonzero if and only if \( a_l \) is linearly independent of the columns of \( A_B \).

**Proof.** Given the full rank assumption of \( A_B \), the matrix \( K_B \) is nonsingular from Proposition 5.2. The solution \( (\Delta x_B^T, -\Delta y^T) \) is unique, and can be calculated as

\[
\Delta x_B = -(A_B^T A_B)^{-1} A_B^T a_l,
\]

\[
\Delta y = -\frac{1}{\mu} (I - A_B (A_B^T A_B)^{-1} A_B^T) a_l.
\]

Suppose that \( a_l \) is linearly dependent on the columns of \( A_B \), i.e., \( a_l \in \text{range}(A_B) \). As \( (I - A_B (A_B^T A_B)^{-1} A_B^T) \) projects \( a_l \) onto the orthogonal complement of \( \text{range}(A_B) \), it follows that

\[
\Delta y = 0, \quad \text{and} \quad \Delta z_N = -A_N^T \Delta y = 0.
\]

Conversely, if \( a_l \) is linearly independent of the columns of \( A_B \), then \( \Delta y \neq 0 \) and \( \Delta z \neq 0 \). □

**Proposition 5.5.** If \( a_l \) is linearly independent of the columns of \( A_B \), and \( \Delta x_l > 0 \) then the following holds:

1. If \( z_l + r_l < 0 \), then \( (\Delta x, \Delta y) \) is a strict descent direction for \( (\text{PLP}_{q,r}) \).

2. If \( x_l + q_l < 0 \), then \( (\Delta y, \Delta z) \) is a strict ascent direction for \( (\text{DLP}_{q,r}) \).

**Proof.** Without loss of generality, assume \( \Delta x_l = 1 \). The proof follows directly from Proposition 5.3 by replacing the constant factor \( \mu/a_l^T P a_l \) by one. □
5.3. The Primal Active-Set Method

The primal active-set method starts and ends with a primal feasible point \((x, y, z)\) that satisfies

\[
\begin{align*}
c - A^T y - z &= 0, \\
x_N + q_N &= 0, \\
x_B + q_B &\geq 0, \\
A x + \mu y - b &= 0, \\
z_B + r_B &= 0,
\end{align*}
\]

(5.13a)

(5.13b)

where \(B\) is constructed such that \(A_B\) has linearly independent columns. Over the course of an iteration, an index \(l\) of a dual infeasible constraint \(z_l + r_l < 0\) is identified and removed from \(N\). This infeasibility is driven to zero, at which point the index \(l\) is added to \(B\).

An iteration begins with a so-called base sub-iteration in which a step is taken along a search direction computed from the linear system (5.12). It follows from Proposition 5.5 that the search direction is always a strict descent direction for \((PLP_{q,r})\). The base sub-iteration continues with the calculation of a step length \(\alpha\) that is the smaller of \(\alpha_s\) and \(\alpha_{\text{max}}\), where \(\alpha_s\) is the step that moves \(z_l + r_l\) to zero and \(\alpha_{\text{max}}\) is the largest step that maintains the feasibility of the primal constraints \(x \geq -q\). If \(\alpha = \alpha_s\) the index \(l\) is added to \(B\) and the iteration is complete. Otherwise, \(\alpha = \alpha_{\text{max}}\) and the index of a blocking constraint \(x_k \geq -q_k\) is moved from \(B\) to \(N\).

In this case, a sequence of one or more intermediate sub-iterations is calculated. An intermediate sub-iteration involves taking a step along a search direction computed from the linear system (5.11) involving \(K_l\). Proposition 5.3 shows that the computed search direction is always a strict descent direction for \((PLP_{q,r})\). As in the base sub-iteration, the step length \(\alpha\) is the smaller of \(\alpha_s\) and \(\alpha_{\text{max}}\), where \(\alpha_s\) is the step that moves \(z_l + r_l\) to zero and \(\alpha_{\text{max}}\) is the largest step that maintains primal feasibility. If \(\alpha = \alpha_{\text{max}}\), then the index of a blocking constraint is moved from \(B\) to \(N\) and the intermediate sub-iterations are repeated until \(\alpha = \alpha_s\), at which point \(z_l + r_l = 0\) and the index \(l\) is added to \(B\).

**Proposition 5.6.** On completion of the base sub-iteration, \(a_l\) is linearly independent of the columns of \(A_B\).

**Proof.** There are two cases to consider.

**Case 1:** Suppose that \(l\) is chosen such that \(a_l\) and columns of \(A_B\) are linearly independent. If \(\alpha = \alpha_s\), then \(z_l + r_l\) will move to zero. In this case, no index is removed from \(B\) in the base sub-iteration and \(A_B\) does not change. If \(\alpha = \alpha_{\text{max}}\), the vector \(a_l\) will continue to be independent of the columns of the new \(A_B\) after the index of the blocking constraint is removed.

**Case 2:** Suppose \(l\) is chosen such that \(a_l\) and columns of \(A_B\) are linearly dependent. From Proposition 5.4, this must imply that \(\Delta z_l = 0\), in which case \(\alpha_s = \infty\). If the primal problem is bounded below, then \(\alpha_{\text{max}}\) will be taken as the step. By the linear dependence assumption and that index \(k \in B\) is chosen such that \(\Delta x_k < 0\), we have

\[
a_l = \sum_{i \in B} \Delta x_i a_i = \sum_{i \in B \setminus \{k\}} \Delta x_i a_i + \Delta x_k a_k.
\]
After removing the index \( k \) from \( \mathcal{B} \), the column \( a_l \) cannot be written as a linear combination of columns of the new \( A_B \), which implies that they must be linearly independent.

**Proposition 5.7.** The matrix \( K_l \) will remain nonsingular during a sequence of intermediate sub-iterations.

**Proof.** Follows from Proposition 5.2 and Proposition 5.6.

It is important to note that if \( \alpha = \alpha_{\text{max}} \) in the base sub-iteration, then at least one intermediate sub-iteration is needed because the dual infeasibility could not be eliminated by taking the longer step \( \alpha = \alpha_s \). It is also possible for \( \alpha \) to be zero in a base or intermediate sub-iteration. However, once a zero step is taken, an index will be deleted from \( \mathcal{B} \), and the execution of an intermediate sub-iteration must follow. The intermediate sub-iterations continue until a nonzero step is taken and \( z_l + r_l \) becomes non-negative. The degenerate case is considered in Section 5.5 below, where it is shown that in the worst case, if all the indices are deleted from \( \mathcal{B} \) by the intermediate sub-iterations, then the search direction must be nonzero by Proposition 5.10 and \( \alpha > 0 \) will drive \( z_l + r_l \) to zero.

The primal active-set method is described in Algorithm 7.
Algorithm 7 A Primal Active-Set Method for Linear Programming.

while ∃ l : z_l + r_l < 0 do
    N ← N \ {l};
    PRIMAL_BASE(ℬ, N, l, x, y, z);  [returns ℬ, N, x, y, z]
    while z_l + r_l < 0 do
        PRIMAL_INTERMEDIATE(ℬ, N, l, x, y, z);  [returns ℬ, N, x, y, z]
    end while
    ℬ ← ℬ \ {l};
end while

function PRIMAL_BASE(ℬ, N, l, x, y, z)
Δx_l ← 1;  Solve \( \begin{pmatrix} 0 & A_B^T \ A_B & -μI \\ A_B & -μI \end{pmatrix} \begin{pmatrix} Δx_l \\ Δx_B \end{pmatrix} = - \begin{pmatrix} 0 \\ a_l \end{pmatrix} \);
Δz_N ← -A_N^T Δy;
Δz_l ← -a_l \Delta y;  [Δz_l ≥ 0]
α_\* ← -(z_l + r_l)/Δz_l;  [α_\* ← +∞ if Δz_l = 0]
α_{max} ← \min_{i: Δx_i < 0} (x_i + q_i)/(−Δx_i);  k ← \arg\min_{i: Δx_i < 0} (x_i + q_i)/(−Δx_i);
α ← \min(α_\*, α_{max});
if α = +∞ then
    stop;  [(DLP_{q,r}) is infeasible]
end if
x_l ← x_l + αΔx_l;  x_B ← x_B + αΔx_B;
y ← y + αΔy;  z_l ← z_l + αΔz_l;  z_N ← z_N + αΔz_N;
if z_l + r_l < 0 then
    ℬ ← ℬ \ {k};  N ← N \cup \{k\};
end if
return ℬ, N, x, y, z;
end function

function PRIMAL_INTERMEDIATE(ℬ, N, l, x, y, z)
Δz_l ← 1;  Solve \( \begin{pmatrix} 0 & 0 & a_l \Delta y \\ 0 & 0 & 1 \\ a_l & A_B & -μI \end{pmatrix} \begin{pmatrix} Δx_l \\ Δx_B \\ a_l \Delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \);  [Δx_l ≥ 0]
Δz_N ← -A_N^T Δy;
α_\* ← -(z_l + r_l);
α_{max} ← \min_{i: Δx_i < 0} (x_i + q_i)/(−Δx_i);  k ← \arg\min_{i: Δx_i < 0} (x_i + q_i)/(−Δx_i);
α ← \min(α_\*, α_{max});
x_l ← x_l + αΔx_l;  x_B ← x_B + αΔx_B;
y ← y + αΔy;  z_l ← z_l + αΔz_l;  z_N ← z_N + αΔz_N;
if z_l + r_l < 0 then
    ℬ ← ℬ \ {k};  N ← N \cup \{k\};
end if
return ℬ, N, x, y, z;
end function
5.4. The Dual Active-Set Method

The dual active-set method starts and ends with a dual feasible point \((x, y, z)\) that satisfies the equations

\[
\begin{align*}
c - A^T y - z &= 0, \quad x_N + q_N = 0, \quad (5.14a) \\
Ax + \mu y - b &= 0, \quad z_B + r_B = 0, \quad z_N + r_N \geq 0, \quad (5.14b)
\end{align*}
\]

where \(B\) is constructed so that \(A_B\) has linearly independent columns. At each iteration, an index \(l\) of a primal-infeasible constraint \(x_l + q_l < 0\) is identified and removed from \(B\). This infeasibility is driven to zero over the course of an iteration, at which point the index \(l\) is added to \(N\).

An iteration begins with a so-called base sub-iteration in which a step is taken along a direction computed from the linear system \((5.11)\). It follows from Proposition 5.3 that the search direction is always a strict ascent direction for \((\text{DLP}_{q,r})\). The base sub-iteration continues with the calculation of a step length \(\alpha\) that is the smaller of \(\alpha_*\) and \(\alpha_{\text{max}}\), where \(\alpha_*\) is the step that moves \(x_l + q_l\) to zero, and \(\alpha_{\text{max}}\) is the largest step that maintains the feasibility of the dual constraints \(z \geq -r\). If \(\alpha = \alpha_*\) the index \(l\) is added to \(B\) and the iteration is complete. Otherwise, \(\alpha = \alpha_{\text{max}}\), and the index of a blocking constraint \(z_k \geq -r_k\) is moved from \(N\) to \(B\). In this case a sequence of one or more intermediate sub-iterations is calculated. An intermediate sub-iteration involves taking a step along the primal-dual search direction \((\Delta x_B, \Delta y)\) satisfying the equations \((5.12)\). This direction satisfies the equations \((5.11)\) with \(\Delta x_l = 1\) and \(\Delta z_l = 0\). It follows from Proposition 5.5 that the search direction is a strict ascent direction for \((\text{DLP}_{q,r})\). As in the base sub-iteration, the step \(\alpha\) is the smaller of \(\alpha_*\) and \(\alpha_{\text{max}}\), where \(\alpha_*\) is the step that moves \(z_l + r_l\) to zero, and \(\alpha_{\text{max}}\) is the largest step that maintains dual feasibility. If \(\alpha = \alpha_{\text{max}}\) the index of a blocking constraint is moved from \(N\) to \(B\) and the intermediate sub-iterations are repeated until \(\alpha = \alpha_*\), in which case \(x_l + q_l = 0\) and the index \(l\) is added to \(N\).

If \(B\) is empty at the beginning of the dual algorithm, then only the base sub-iteration is needed to drive \(x_l + q_l\) to zero. Otherwise, if \(B\) is nonempty and \(\alpha = \alpha_{\text{max}}\) in the base sub-iteration then at least one intermediate sub-iteration is needed.

**Proposition 5.8.** The matrix \(A_B\) always has linearly independent columns at the end of the dual algorithm.

**Proof.** If \(a_l\) and the columns of \(A_B\) are linearly dependent at the beginning of an intermediate sub-iteration, then Proposition 5.4 gives \(\Delta z = 0\). In this case, the set \(\{i : \Delta z_i < 0\}\) is empty and \(\alpha\) must take the value \(\alpha_*\), then the updated primal violation \(x_l + q_l\) is zero. At this point no more indices will be added to \(B\), the index \(l\) is added to \(N\), and the full-rank matrix is not changed. 

The dual active-set method is given in Algorithm 8.
Algorithm 8 A dual active-set method for linear programming.

while \( \exists l : x_l + q_l < 0 \) do
    \( \mathcal{B} \leftarrow \mathcal{B} \setminus \{l\} \);
    DUAL_BASE(\( \mathcal{B} \), \( \mathcal{N} \), \( l \), \( x \), \( y \), \( z \)); \hspace{1cm} \text{[Base sub-iteration]}
    while \( x_l + q_l < 0 \) do
        DUAL_INTERMEDIATE(\( \mathcal{B} \), \( \mathcal{N} \), \( l \), \( x \), \( y \), \( z \)); \hspace{1cm} \text{[Intermediate sub-iteration]}
    end while
    \( \mathcal{N} \leftarrow \mathcal{N} \cup \{l\} \);
end while

function DUAL_BASE(\( \mathcal{B} \), \( \mathcal{N} \), \( l \), \( x \), \( y \), \( z \))

\[ \Delta z_l \leftarrow 1; \] \hspace{1cm} \text{Solve}
\[ \begin{pmatrix} 0 & 0 & a_l^T \\ 0 & 0 & A_B^T \\ a_l & A_B & -\mu I \end{pmatrix} \begin{pmatrix} \Delta x_l \\ \Delta x_B \\ -\Delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}; \hspace{1cm} [\Delta x_l \geq 0] \]

\[ \Delta z_N \leftarrow -A_N^T \Delta y; \]
\[ \alpha_* \leftarrow -(x_l + q_l)/\Delta x_l; \hspace{1cm} [\alpha_* \leftarrow +\infty \text{ if } \Delta x_l = 0] \]
\[ \alpha_{\text{max}} \leftarrow \min_{i: \Delta z_i < 0} (z_i + r_i)/(-\Delta z_i); \hspace{1cm} k \leftarrow \arg\min_{i: \Delta z_i < 0} (z_i + r_i)/(-\Delta z_i); \]
\[ \alpha \leftarrow \min (\alpha_*, \alpha_{\text{max}}); \]
if \( \alpha = +\infty \) then
    stop; \hspace{1cm} \text{[(PLP}_{q,r} \text{) is infeasible]}
end if
\[ x_l \leftarrow x_l + \alpha \Delta x_l; \quad x_B \leftarrow x_B + \alpha \Delta x_B; \]
\[ y \leftarrow y + \alpha \Delta y; \quad z_l \leftarrow z_l + \alpha \Delta z_l; \quad z_N \leftarrow z_N + \alpha \Delta z_N; \]
if \( x_l + q_l < 0 \) then
    \( \mathcal{B} \leftarrow \mathcal{B} \cup \{k\}; \quad \mathcal{N} \leftarrow \mathcal{N} \setminus \{k\}; \)
end if
return \( \mathcal{B} \), \( \mathcal{N} \), \( x \), \( y \), \( z \);
end function

function DUAL_INTERMEDIATE(\( \mathcal{B} \), \( \mathcal{N} \), \( l \), \( x \), \( y \), \( z \))

\[ \Delta z_l \leftarrow 1; \] \hspace{1cm} \text{Solve}
\[ \begin{pmatrix} 0 & A_B^T \\ A_B & -\mu I \end{pmatrix} \begin{pmatrix} \Delta x_B \\ -\Delta y \end{pmatrix} = -\begin{pmatrix} 0 \\ a_l \end{pmatrix}; \hspace{1cm} [\Delta z_l \geq 0] \]

\[ \Delta z_N \leftarrow -A_N^T \Delta y; \]
\[ \Delta z_N \leftarrow -A_N^T \Delta y; \]
\[ \alpha_* \leftarrow -(x_l + q_l); \]
\[ \alpha_{\text{max}} \leftarrow \min_{i: \Delta z_i < 0} (z_i + r_i)/(-\Delta z_i); \hspace{1cm} k \leftarrow \arg\min_{i: \Delta z_i < 0} (z_i + r_i)/(-\Delta z_i); \]
\[ \alpha \leftarrow \min (\alpha_*, \alpha_{\text{max}}); \]
\[ x_l \leftarrow x_l + \alpha \Delta x_l; \quad x_B \leftarrow x_B + \alpha \Delta x_B; \]
\[ y \leftarrow y + \alpha \Delta y; \quad z_l \leftarrow z_l + \alpha \Delta z_l; \quad z_N \leftarrow z_N + \alpha \Delta z_N; \]
if \( x_l + q_l < 0 \) then
    \( \mathcal{B} \leftarrow \mathcal{B} \cup \{k\}; \quad \mathcal{N} \leftarrow \mathcal{N} \setminus \{k\}; \)
end if
return \( \mathcal{B} \), \( \mathcal{N} \), \( x \), \( y \), \( z \);
end function
5.5. Degeneracy

An iteration is said to be degenerate if a zero step is defined in each of the base and intermediate sub-iterations. Specifically, the primal problem is nondegenerate if a nonzero step is taken at every base sub-iteration. The nonsingularity of the regularized term $\mu I$ will guarantee that the primal problem is nondegenerate.

We require that the constrained gradient $A_{\mu}$ to be linearly independent from the beginning, and our method for adding and deleting constraints from working set for subsequent working set will still maintain the property of linearly independence.

**Proposition 5.9.** If $\Delta z_l > 0$, $K_l$ nonsingular and $\mu \neq 0$, then the primal step is nondegenerate.

**Proof.** The proof follows from the fact that $\Delta x_l = \mu/a_l^T P a_l > 0$, where $P = (I - A_{\mu}(A_{\mu}^T A_{\mu})^{-1} A_{\mu}^T)$.

**Proposition 5.10.** The primal-dual active-set method does not cycle.

**Proof.** Suppose that at the end of the intermediate sub-iterations, every variable in $B$ is deleted. The equation becomes

$$
\begin{pmatrix}
0 & a_l^T \\
a_l & -\mu I
\end{pmatrix}
\begin{pmatrix}
\Delta x_l \\
-\Delta y
\end{pmatrix} =
\begin{pmatrix}
1 \\
0
\end{pmatrix}.
$$

$(\Delta x_l - \Delta y)^T$ has the unique solution

$$
\begin{pmatrix}
\Delta x_l \\
-\Delta y
\end{pmatrix} = \frac{1}{\|a_l\|^2} \begin{pmatrix}
\mu \\
a_l
\end{pmatrix}
$$

as $\mu > 0$, $a_l > 0$.

5.6. Combining Primal and Dual Active-set Methods

The primal active-set method proposed in Section 5.3 may be used to solve $(\text{PLP}_{q,r})$ for a given initial basis $B$ satisfying the conditions (5.13). An appropriate initial point may be found by solving a conventional phase-1 linear program. Alternatively, the dual active-set method of Section 5.4 may be used in conjunction with an appropriate phase 1 procedure to solve the quadratic program $(\text{PLP}_{q,r})$ for a given initial basis $B$ satisfying the conditions (5.14). An alternative to the conventional phase-1/phase-2 approach is to create a pair of coupled quadratic programs from the original by simultaneously shifting the bound constraints. The idea is to solve a shifted dual problem to define an initial feasible point for the primal, or vice-versa. This strategy provides an alternative to the conventional phase-1/phase-2 approach that utilizes the phase-1 objective function while finding a feasible point.

An algorithm that combines the primal and dual active-set methods is given in Algorithm 9. The algorithm begins with a second-order consistent basis satisfying...
(5.5) and (5.6). If the initial point is both primal and dual feasible then the triple \((x, y, z)\) is an optimal solution and the algorithm is terminated. If \(x\) is primal feasible, then the primal active-set method \text{PRIMAL\_LP} of Algorithm 7 is used to find an optimal solution. If \(z\) is dual feasible, then the dual active-set method \text{DUAL\_LP} of Algorithm 8 is used to find an optimal solution. If the initial point is neither primal nor dual feasible, then one of the shifts \(q\) or \(r\) can be defined to make the initial point either primal or dual feasible.

5.6.1. Shifting the primal problem

Assume that the given initial point is neither primal nor dual feasible. If we choose to shift the primal problem, the shifts are defined as follows.

1. Define the shifts \(r\) and \(q\) such that \(r = 0\) and \(q_N = 0, q_B = \max\{-x_B, 0\}\).
2. Compute the solution of \((\text{PLP}_{q,r})\).
3. Compute a point \((x^{(1)}, y^{(1)}, z^{(1)})\) satisfying (5.5) and (5.6).
4. Starting at the point \((x^{(1)}, y^{(1)}, z^{(1)})\), solve \((\text{DLP}_0)\).

After applying the shifts in Step 1, the point \(x\) satisfies \(x + q \geq 0\), i.e., \(x\) is an initial primal-feasible point for \((\text{PLP}_{q,r})\). The optimal solution of the primal problem \((\text{PLP}_{q,r})\) solved in Step 2 satisfies

\[
\begin{align*}
    c - A^T y^{(1)} - z^{(1)} &= 0, \\
    x_N^{(1)} + q_N &= 0, \\
    x_B^{(1)} + q_B &\geq 0, \\
    Ax^{(1)} + \mu y^{(1)} - b &= 0, \\
    z_N^{(1)} &= 0, \\
    z_B^{(1)} &= 0.
\end{align*}
\]

Without the shift, this solution may violate (5.5) and (5.6), in which case \(x^{(1)}\) must be recomputed in Step 3 by solving the linear least-squares problem

\[
\min_{x \in \mathbb{R}^t} \| A_B x - b + \mu y^{(1)} \|.
\]

The point \((x^{(1)}, y^{(1)}, z^{(1)})\) found in Step 3 will be dual feasible for the original problem and Step 4 will compute the required optimal solution.

5.6.2. Shifting the dual problem

Again, assume that the given initial point is neither primal nor dual feasible. If we choose to shift the dual problem, the shifts are defined as follows:

1. Define the shifts \(q\) and \(r\) such that \(q = 0, r_B = 0\) and \(r_N = \max\{-z_N, 0\}\).
2. Compute a solution of \((\text{DLP}_{q,r})\).
3. Compute a point \((x^{(1)}, y^{(1)}, z^{(1)})\) satisfying (5.5) and (5.6).
4. Starting at the point \((x^{(1)}, y^{(1)}, z^{(1)})\), solve \((\text{PLP}_0)\).
After applying the shifting procedure of Step 1, the point $z$ satisfies $z + r \geq 0$, i.e., $z$ is an initial dual feasible point for (DLP$_{q,r}$). The optimal solution of (DLP$_{q,r}$) satisfies

\[
\begin{align*}
  c - A^T y^{(1)} - z^{(1)} &= 0, & x_B^{(1)} &= 0, & x_B^{(1)} &\geq 0, \\
  Ax^{(1)} + \mu y^{(1)} - b &= 0, & z_B^{(1)} + r_B &= 0, & z_N^{(1)} + r_N &\geq 0.
\end{align*}
\]

Without the shift, this solution may violate (5.5) and (5.6), in which case $y^{(1)}$ and $z^{(1)}$ must be recomputed in Step 3 using

\[
y^{(1)} = \frac{1}{\mu} (b - A_B x_B^{(1)}) \quad \text{and} \quad z_N^{(1)} = c_N - A_N^T y^{(1)}.
\]

The point $(x^{(1)}, y^{(1)}, z^{(1)})$ computed in Step 3 will be primal feasible, and Step 4 will compute an optimal solution for the original problem.
6. Matrix Factorizations and Updating

When solving the linear least-squares problem $\min_{x \in \mathbb{R}^m} \| Ax - b \|$ it is crucial that a factorization of the matrix $A$ is used instead of directly computing the inverse of $A^T A$ for the normal equations. Consider an $m \times n$ matrix $A$ with $m > n$. The QR factorization of $A$ is given by

$$ Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad (6.1) $$

where $Q$ is an orthogonal matrix of dimension $m \times m$, and $R$ is an upper-triangular matrix of dimension $n \times n$.

It is also necessary to update this factorization when columns or the rows of $A$ are added, deleted, or both.
6.1. Using the QR Decomposition for Solving Least-Squares Problems

6.1.1. Calculating a null-space descent direction

In addition to using QR decomposition to calculate a least-squares solution, it can also be used to calculate the matrix \( Z \) such that \( A^T B Z = 0 \).

Lemma 6.1. Consider the QR factorization (6.1) of \( A_B \), with \( Q = (Q_1 \quad Q_2) \), where \( Q_1 = (q_1 \quad q_2 \ldots \quad q_n) \), \( Q_2 = (q_{n+1} \ldots \quad q_m) \), and the \( q_i \), \( i = 1, 2, \ldots, m \) are independent orthonormal column vectors. The matrix \( Z = Q_2 \) satisfies \( A^T B Z = 0 \).

Proof. Substituting the QR factors of \( A_B \) in \( A_B Z \) and using the identities \( Q_1^T Q_2 = 0 \) and \( Q_2^T Q_2 = I_{m-n} \) gives

\[
A^T_B Z = (R^T \quad 0) \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} Q_2 = (R^T \quad 0) \begin{pmatrix} Q_1^T Q_2 \\ Q_2^T Q_2 \end{pmatrix} = (R^T \quad 0) \begin{pmatrix} 0 \\ I_{m-n} \end{pmatrix} = 0.
\]

6.1.2. Calculating the projection matrix

Let \( A_B \) be a matrix of size \( m \times n \) with \( m \geq n \). Let \( b \) be a vector of size \( m \). Assume \( A_B \) has linearly independent columns. The least-squares solution to the unconstrained optimization problem

\[
\text{minimize } \| A_B x - b \|^2 \quad \text{subject to } x \in \mathbb{R}^n \quad (6.2)
\]

is

\[
x = (A^T_B A_B)^{-1} A^T_B b, \quad (6.3)
\]

where \( A^+_B = (A^T_B A_B)^{-1} A^T_B \) is the Moore-Penrose pseudoinverse of \( A_B \).

We now consider how to use the QR decomposition to calculate the pseudoinverse and projection matrix in the full-rank case. The full-size QR factorization is

\[
A_B = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1 \quad Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix}.
\]

Let \( Q_1 R = A_B \) be the corresponding “economy-size” QR factorization, where \( R \) is a \( n \times n \) upper-triangular matrix and \( Q_1 \) is the \( m \times n \) submatrix of the orthonormal matrix \( Q \). Note that \( Q_1^T Q_1 = I_k \) but \( Q_1 Q_1^T \neq I_n \). Furthermore, by the full-rank assumption of \( A_B \), \( R \) must be invertible, and its diagonal entries are all nonzero. Then (6.3) can be solved as

\[
x = (A^+_B A_B)^{-1} A^T_B b = (R^T Q_1^T Q_1 R)^{-1} R^T Q_1^T b
\]

\[
R^T Rx = R^T Q_1^T b
\]

\[
Rx = Q_1^T b.
\]

This implies that forward substitution can be used to compute \( x \). The cost of calculating \( Q_1^T b \) is \( O(mn) \), and one iteration of backward substitution requires \( O(m^2) \). Then, if \( Q \) and \( R \) are already known, the complexity of solving (6.2) is \( O(m^2) \).
Now consider the calculation of the orthogonal projection matrix onto range\((A_B)\),

\[
P = A_B (A_B^T A_B)^{-1} A_B^T = Q_1 R (R^T R)^{-1} R^T Q_1^T
\]

\[
= Q_1 R R^{-1} R^{-T} R^T Q_1^T
\]

\[
= Q_1 I_n Q_1^T
\]

\[
= Q_1 Q_1^T.
\]

Similarly, the orthogonal projection onto null\((A_B^T)\) can be calculated as

\[
M = I_n - Q_1 Q_1^T, \quad (6.4)
\]

or

\[
M = Z (Z^T Z)^{-1} Z^T, \quad (6.5)
\]

where the columns of \(Z\) form a basis for null\((A_B^T)\). By using the result from (6.1)

\[
Z (Z^T Z)^{-1} Z^T = Q_2 (Q_2^T Q_2)^{-1} Q_2^T = Q_2 I_{m-n} Q_2^T = Q_2 Q_2^T.
\]

When \(m \gg n\) it is more efficient to store only \(Q_1\) and calculate the steepest-descent direction in the null space, i.e., \(p = -Z (Z^T Z)^{-1} Z^T c\) using (6.4),

\[
p = (Q_1 Q_1^T - I) c.
\]

The cost of calculating the steepest-descent direction in null\((A_B^T)\) is approximately \(2mn + m\) flops.

### 6.1.3. Solving the KKT equations

By using the results above, the solution of the linear system (5.11) can be written as

\[
\Delta x_l = \beta \mu,
\]

\[
\Delta x_B = -\beta R^{-1} Q_1^T a_l,
\]

\[-\Delta y = \beta (a_l + A_B \Delta x_B) = \beta (a_l - Q_1 Q_1^T a_l),
\]

where

\[
\beta = a_l^T M a_l = a_l^T (I_n - Q_1 Q_1^T) a_l.
\]

The cost of computing (5.11) by given QR decomposition is \(O(m^2)\).

### 6.2. Updating the QR Factorization

It is critical to consider the modification as recomputing the QR factorization is too expensive as it requires approximately \(mn^2\) floating-point operations.

Three kinds of modification of the matrix \(A_k\) are considered:

1. a rank-one change of \(A_k\);
2. the deletion and addition of a column of \(A_k\); and
3. the deletion and addition of a row of $A_k$.

First, consider two basic transformations that are essential tools for matrix updating—Givens rotations and Householder transformations.

**Lemma 6.2. (Givens Rotation)** Given $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, with at least one of the quantities $x_1$ and $x_2$ nonzero, then there exists an orthogonal matrix

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

with $c^2 + s^2 = 1$ such that

$$Gx = \left(\frac{x_1^2 + x_2^2}{2}\right)^{1/2} \begin{pmatrix} x_1 \\ 0 \end{pmatrix}.$$

**Proof.** The result follows directly with $c$ and $s$ defined as

$$c = \frac{x_1}{(x_1^2 + x_2^2)^{1/2}} \quad \text{and} \quad s = \frac{x_2}{(x_1^2 + x_2^2)^{1/2}}.$$

(6.6)

To exploit the symmetric structure, we instead define the symmetric Givens matrix

$$G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

with the same $c$ and $s$ as in (6.6).

**Lemma 6.3. (Householder transformation)** Given a non-zero vector $v \in \mathbb{R}^m$, there exists an orthogonal matrix $Q$ such that

$$Qv = -\sigma \|v\| e_1$$

with $e_1 = (1, 0, \ldots, 0)^T$ and $\sigma = \text{sign}(v_1)$.

**Proof.** Define

$$Q = I_m - 2\frac{uu^T}{u^Tu}, \quad \text{with} \quad u = v + \sigma \|v\| e_1.$$

6.3. Modification after a Rank-one Change

Given the QR decomposition of the matrix $A_k \in \mathbb{R}^{m \times n}$

$$A_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix},$$
we want to compute the QR decomposition after the rank-one change

\[ A_{k+1} = A_k + uv^T = Q_{k+1} \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}. \]

First, compute \( w = Q_k^T u \in \mathbb{R}^m \) and define a sequence of Givens rotations \( G_{k,k+1} \), \( k = 1, 2, \ldots, m - 1 \) such that

\[ G_{1,2}^T \cdots G_{m-1,m}^T w = \pm \|w\| e_1. \]

These transformation will produce an upper-Hessenberg matrix when apply to \( R \)-factor

\[ H = G_{1,2}^T \cdots G_{m-1,m}^T \begin{pmatrix} R_k \\ 0 \end{pmatrix}. \]

For \( A_{k+1} \), we have

\[ A_{k+1} = A_k + uv^T = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix} + wv^T, \]

\[ Q_k^T A_{k+1} = \begin{pmatrix} R_k \\ 0 \end{pmatrix} + wv^T, \]

\[ G_{1,2}^T \cdots G_{m-1,m}^T Q_k A_{k+1} = H \pm \|w\| e_1 v^T. \]

Adding \( e_1 v^T \) to an upper-Hessenberg matrix is still an upper-Hessenberg matrix because only the first row of \( H \) is modified. Let \( \tilde{H} = H \pm \|w\| e_1 v^T \) and define a sequence of Givens rotations \( \tilde{G}_{k,k+1} \), \( k = 1, 2, \ldots, n \) that "zero out" the \((k, k+1)\)-th entry of \( \tilde{H} \), i.e.,

\[ \tilde{G}_{n,n+1}^T \cdots \tilde{G}_{1,2}^T \tilde{H} = \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}. \]

Finally, the orthogonal matrix \( Q_{k+1} \) can be found by

\[ Q_{k+1} = Q_k G_{m-1,m} \cdots G_{1,2} \tilde{G}_{1,2} \cdots \tilde{G}_{n,n+1}. \]

The matrices \( Q_{k+1} \) and \( R_{k+1} \) are the desired updated QR factors. A total of \( m^2 \) flops are required to compute \( w \), and \( 4n^2 \) flops are required to calculate \( H \) and \( R_{k+1} \). Finding \( Q_{k+1} \) requires \( 4(m^2 + mn) \) flops. In total, \( 5m^2 + 4mn \) flops are required for one rank-one update. However, as recalculating the QR decomposition for a general \( m \times n \) matrix requires \( O(mn^2) \) flops, when \( m \gg n \) it may be beneficial to compute the factorization from scratch rather than perform a matrix update.

6.4. Adjoining and Deleting a Column

6.4.1. Adjoining a column

Given the QR decomposition of \( A_k \in \mathbb{R}^{m \times n} \)

\[ A_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix}, \]
we require the QR decomposition after appending one column vector \( a_{k+1} \), i.e.,

\[
A_{k+1} = \begin{pmatrix} A_k & a_{k+1} \end{pmatrix}
\]

First, let \( b_{k+1} = Q_k^T a_{k+1} \). Premultiply \( A_{k+1} \) by \( Q_k \) and partition the matrix

\[
Q^T_k A_{k+1} = \begin{pmatrix} R_k & b_{k+1} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} R_k & s_k \\ 0 & v \end{pmatrix} \text{ with } b_{k+1} = \begin{pmatrix} s_k \\ v \end{pmatrix},
\]

where \( s_k \) is a vector of dimension \( n \) and \( v \) is a vector of dimension \( m - n \). Using a Householder transformation of the form (6.3) we define

\[
u = v + \sigma ||v||e_1,
\]

\[
\hat{H} = I_{m-k} - 2\frac{uu^T}{u^Tu},
\]

with \( e_1 \) the \((m-n)\)-vector \( e_1 = (1, 0, \ldots, 0)^T \). Let

\[
H = \begin{pmatrix} I_m & 0 \\ 0 & \hat{H} \end{pmatrix}.
\]

As \( H \) is symmetric, the equation (6.7) can be premultiplied by \( HT \) to give

\[
H^T Q^T_k A_{k+1} = \begin{pmatrix} I_m & 0 \\ 0 & \hat{H} \end{pmatrix} \begin{pmatrix} R_k & s_k \\ 0 & v \end{pmatrix} = \begin{pmatrix} R_k & s_k \\ 0 & -\sigma ||v||e_1 \\ 0 & 0 \end{pmatrix},
\]

with \( \sigma = \text{sign}(v_1) \), and \( Q_{k+1} = Q_k H \).

### 6.4.2. Removing a column

Given the QR decomposition of \( A_k \in \mathbb{R}^{m\times n} \)

\[
A_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix},
\]

we require the QR factorization when the \( j \)-th column is removed from \( A_k \), where \( 1 \leq j \leq n \). An explicit method proposed by Golub and Saunders [14] defines

\[
A_{k+1} = \begin{pmatrix} a_1 & \cdots & a_{j-1} & a_{j+1} & \cdots & a_n \end{pmatrix},
\]

where \( a_i \) is the \( i \)-th column of \( A_k \). Premultiplying by \( Q_k^T \) gives

\[
Q^T_k A_{k+1} = \begin{pmatrix} r_1 & \cdots & r_{j-1} & r_{j+1} & \cdots & r_n \end{pmatrix},
\]

with \( r_i \) is the \( i \)-th column vector of \( R_k \). We then define a sequence of Givens transformations \( G_{i,i+1}^T, i = j, \ldots, n - 1 \), where \( G_{i,i+1}^T \) zeros out the \( i \)-th element of the \((i+1)\)-th row. The orthogonal matrix \( Q_{k+1} \) is found by

\[
Q_{k+1} = Q_k G_{j,j+1}^T \cdots G_{n-1,n}^T.
\]
The matrix $R_{k+1}$ is found by explicitly calculating the last $(n - j + 1)$-th columns as

$$Q^T_{k+1}A_{k+1} = \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix},$$

where the first $(j - 1)$ columns of $R_{k+1}$ are the same as those of $R_k$.

### 6.5. Adjoining and Deleting a Row

#### 6.5.1. Adjoining a row

Given a QR decomposition of $A_k \in \mathbb{R}^{m \times n}$

$$A_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix},$$

we require the QR decomposition after adjoining the row vector $a_{k+1} \in \mathbb{R}^n$, i.e.,

$$A_{k+1} = \begin{pmatrix} A_k \\ a_{k+1}^T \end{pmatrix} = Q_{k+1} \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}.$$

First, we apply an orthogonal matrix diag($Q_k, 1$),

$$\begin{pmatrix} Q_k \\ 0 \end{pmatrix} \begin{pmatrix} A_k \\ a_{k+1}^T \end{pmatrix} = \begin{pmatrix} R_k \\ 0 \end{pmatrix} = \Pi_{n+1,m+1} \begin{pmatrix} R_k \\ a_{k+1}^T \end{pmatrix},$$

where $\Pi_{n+1,m+1}$ is a permutation matrix that interchanges rows $n + 1$ and $m + 1$.

Then use a sequence of Givens rotations $G_{i,n+1}, i = 1, \ldots, n$ where $G_{i,n+1}$ eliminates the $i$-th element of the $(n+1)$-th row,

$$G_{n,n+1}G_{n-1,n+1} \cdots G_{1,n+1} \Pi_{n+1,m+1} \begin{pmatrix} R_k \\ a_{k+1}^T \end{pmatrix} = \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix},$$

and the updated $Q_{k+1}$ becomes

$$Q_{k+1} = \begin{pmatrix} Q_k \\ 0 \end{pmatrix} \Pi_{n+1,m+1} G_{1,n+1}^T \cdots G_{n,n+1}^T.$$

The updating procedure is stable and requires approximately $2n^2$ operations with standard Givens rotations.

#### 6.5.2. Deleting a row

Given a QR decomposition of the matrix $A_k \in \mathbb{R}^{m \times n}$

$$A_k = \begin{pmatrix} a_1^T \\ A_{k+1} \end{pmatrix} = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix},$$
without loss of generality, we want to compute the QR decomposition of $A_{k+1}$, i.e., the submatrix of $A_{k+1}$ without the first row,

$$A_{k+1} = Q_{k+1} \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}.$$ 

Note that the removal of $k$-th row can be done by the same procedure except that we premultiply a permutation matrix $P_{1,k}$ to $A_k$ and $Q_k$ in order to interchange row 1 and row $k$. The algorithm given below is then applied to $P_{1,k}A_k$ and $P_{1,k}Q_k$ instead.

First find the QR decomposition of $(e_1 \ A_k)$,

$$Q_k^T (e_1 \ A_k) = \begin{pmatrix} q_1 \\ R_k \\ 0 \end{pmatrix}.$$ 

where $q_1$ is the first row of $Q_k$.

Then define a sequence of Givens transformation $G_{i,i+1}, i = 1, \ldots, m - 1$ such that

$$G_{1,2} \cdots G_{m-1,m} q_1 = \alpha e_1, \quad \|\alpha\| = 1.$$ 

Then we have

$$G_{1,2} \cdots G_{m-1,m} \begin{pmatrix} q_1 \\ R_k \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha \\ v^T \\ 0 \end{pmatrix} \begin{pmatrix} R_{k+1} \\ 0 \end{pmatrix}.$$ 

In addition, we compute $Q_{k+1}$ as below,

$$Q_k G_{m-1,m} \cdots G_{1,2} = \begin{pmatrix} \alpha \\ 0 \\ Q_{k+1} \end{pmatrix}.$$ 

Hence, we have

$$A_k = \begin{pmatrix} a_1^T \\ A_{k+1} \end{pmatrix} = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix} = Q_k G_{m-1,m} \cdots G_{1,2} G_{1,2}^T \cdots G_{m-1,m}^T \begin{pmatrix} R_k \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ Q_{k+1} \end{pmatrix} \begin{pmatrix} v^T \\ R_{k+1} \\ 0 \end{pmatrix}.$$ 

This algorithm for removing a row can be regarded as a special case of updating after a rank-one change with $u = -e_1$ and $v = a_1$.

6.6. Other Factorization Methods

6.6.1. The symmetric indefinite factorization

When a matrix $K \in \mathbb{R}^{n \times n}$ is symmetric positive definite, the Cholesky factorization $K = LL^T$ can be used to solve the linear system $Kx = f$. The Cholesky algorithm
is efficient because it requires no pivoting. It only involves about $\frac{1}{6}n^3$ floating-point operations. However, if $K$ is indefinite, the Cholesky factors do not exist because the matrix has negative eigenvalues. Instead, we consider an alternative factorization

$$\Pi K \Pi^T = LDL^T,$$

where $\Pi$ is a permutation matrix and $D$ is a block-diagonal matrix with either $1 \times 1$ or $2 \times 2$ diagonal blocks.

The calculation of the symmetric indefinite factorization requires approximately $\frac{1}{6}n^3$ floating-point operations, which is comparable to the cost of the Cholesky factorization.

Given the symmetric indefinite factorization of $K$, the solution of the linear equations $Kx = f$ can be calculated by solving the sequence of triangular and block-diagonal systems

$$Lw = \Pi f, \quad (6.10a)$$
$$Dy = w, \quad (6.10b)$$
$$L^Tz = y, \quad (6.10c)$$
$$x = \Pi^Tz. \quad (6.10d)$$

(6.10a) and (6.10c) can be solved by forward and backward substitution respectively, and the solution of (6.10b) can be found by explicitly calculating the inverse of each diagonal block.

7. Numerical Examples

In this section, we present some numerical experiments for simple MATLAB implementation applied to two types of problems.

The first set of examples are randomly generated assignment problems that are highly degenerate. The second set of examples are problems that have been known to cycle when solved by the conventional simplex method.

In both parts, we focus on comparing the methods PDNNLS and PDLP described in Sections 3 and 5. All the problems are formulated in standard form. For each problem, we record the row number $m$, column number $n$ of the coefficient matrix $A$, and the optimal objective. For PDNNLS, we set $q$ and $r$ in (5.1) to be zero, and use $\mu = 10^{-8}$. We test both the “primal-first” and “dual-first” options of PDLP and record the corresponding primal and iteration numbers as well as the total running time. If the problem is primal feasible at the beginning, we record only the iteration numbers and running time for the primal method and leave dual entries blank. An analogous annotation is used when the initial point is dual active.

For testing problems using PDLP, we use the phase-1/phase-2 scheme starting with the initial dual point $y_0 = 0$. We use the MATLAB routine lsqnonneg to solve the non-negative linear least squares problem. We record the iteration numbers for the main algorithm of PDNNLS, the number of times a NNLS subproblem is solved, the number of subiterations in the NNLS algorithm, and the corresponding running time. All running times are given in units of a second. Tolerance levels are set to be $\epsilon_{tol} = 10^{-6}$ and $\epsilon_{fea} = 10^{-6}$ in both implementations.
7.1. Highly Degenerate Problems

7.1.1. Test problems

Below we give the pseudocode for the assignment problem. The pseudocode returns the matrix $A$, and vectors $b$ and $c$ associated with a linear program in standard form

$$\minimize_{x \in \mathbb{R}^n} c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0.$$ 

Given an integer $n$, the assignment algorithm returns $A \in \mathbb{R}^{(2n-1) \times n^2}$, $b \in \mathbb{R}^{2n-1}$ and $c \in \mathbb{R}^{n^2}$, where $A$ is sparse with unit entries. The matrix $A$ has significantly more columns than rows, which increases the chance of selecting a dependent basis in a conventional active-set method. The problem is highly degenerate.

Algorithm 10  The assignment problem.

```text
function ASSIGN(n)
    $C = \text{rand}(n)$;
    for $i = 1$ to $n$ do
        $b(i, 1) = 1$;
        $b(n + i, 1) = 1$;
    for $j = 1$ to $n$ do
        $c(n(i - 1) + j, 1) = C(i, j)$;
        $A(i, n(i - 1) + j) = 1$;
        $A(n + i, n(j - 1) + i) = 1$;
    end for
    return $A$, $b$, $c$;
end function
```

7.1.2. Computational results

In this section we present the results of numerical experiments on the assignment problems for $n = 25, 50, 75, 100, 125, 150, 200$, and $300$. As the cost vector $c$ is non-negative, the initial point $y_0 = 0$ is dual feasible for PDNNLS and it is necessary to solve only the phase 1 problem for this method. For this reason, in all the tables for PDNNLS, only the iteration numbers and running times for the phase 1 problem are recorded.
Table 1: Results for PDNNLS on Assignment Problems

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Objective</th>
<th>PHASE 1</th>
<th>PHASE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Itn NNLS-Itn</td>
<td>NNLS-Sub</td>
</tr>
<tr>
<td>49</td>
<td>625</td>
<td>1.2766622E+00</td>
<td>37 622</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>2500</td>
<td>1.6111842E+00</td>
<td>113 537</td>
<td>268</td>
</tr>
<tr>
<td>149</td>
<td>5625</td>
<td>1.5963300E+00</td>
<td>185 1461</td>
<td>362</td>
</tr>
<tr>
<td>199</td>
<td>10000</td>
<td>1.7456917E+00</td>
<td>225 2072</td>
<td>610</td>
</tr>
<tr>
<td>249</td>
<td>15625</td>
<td>1.6131899E+00</td>
<td>257 2825</td>
<td>737</td>
</tr>
<tr>
<td>299</td>
<td>25000</td>
<td>1.5795925E+00</td>
<td>303 3894</td>
<td>909</td>
</tr>
<tr>
<td>399</td>
<td>40000</td>
<td>1.6880131E+00</td>
<td>504 10940</td>
<td>3938</td>
</tr>
<tr>
<td>599</td>
<td>90000</td>
<td>1.5619877E+00</td>
<td>693 21208</td>
<td>7497</td>
</tr>
</tbody>
</table>

Table 2: Results for PDLP on Assignment Problems

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>Objective</th>
<th>Primal-First</th>
<th>Dual-First</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Itn P-Itn</td>
<td>D-Itn</td>
</tr>
<tr>
<td>49</td>
<td>625</td>
<td>1.2766622E+00</td>
<td>64 46 18</td>
<td>0.0322</td>
</tr>
<tr>
<td>99</td>
<td>2500</td>
<td>1.6111842E+00</td>
<td>122 22 100</td>
<td>0.1468</td>
</tr>
<tr>
<td>149</td>
<td>5625</td>
<td>1.5963300E+00</td>
<td>226 62 164</td>
<td>0.4080</td>
</tr>
<tr>
<td>199</td>
<td>10000</td>
<td>1.7456917E+00</td>
<td>444 133 311</td>
<td>1.0075</td>
</tr>
<tr>
<td>249</td>
<td>15625</td>
<td>1.6131899E+00</td>
<td>374 59 315</td>
<td>1.5745</td>
</tr>
<tr>
<td>299</td>
<td>25000</td>
<td>1.5795925E+00</td>
<td>846 258 588</td>
<td>3.3927</td>
</tr>
<tr>
<td>399</td>
<td>40000</td>
<td>1.6880131E+00</td>
<td>1358 348 1010</td>
<td>9.5163</td>
</tr>
<tr>
<td>599</td>
<td>90000</td>
<td>1.5619877E+00</td>
<td>1934 518 1416</td>
<td>37.0491</td>
</tr>
</tbody>
</table>

7.2. Cycling Problems

7.2.1. Test problems

In addition to the assignment problem, we also give results for PDNNLS and PDLP on 12 problems that are known to cycle when solved by the simplex method. For more details about the formulation of these problems, we refer the reader to [9] and [15].

7.2.2. Computational results

Table 3: Results for PDNNLS on Cycling Problems

<table>
<thead>
<tr>
<th>Name</th>
<th>m</th>
<th>n</th>
<th>Objective</th>
<th>PHASE 1</th>
<th>PHASE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Itn NNLS-Itn</td>
<td>NNLS-Sub</td>
</tr>
<tr>
<td>HUFFMAN</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>2 4 1</td>
<td>1</td>
</tr>
<tr>
<td>BEALE</td>
<td>3</td>
<td>7</td>
<td>-0.05</td>
<td>3 6 1</td>
<td>2</td>
</tr>
<tr>
<td>Y&amp;G-1</td>
<td>3</td>
<td>7</td>
<td>-0.5</td>
<td>4 13 2</td>
<td>1</td>
</tr>
<tr>
<td>Y&amp;G-2</td>
<td>3</td>
<td>7</td>
<td>-1</td>
<td>3 10 2</td>
<td>1</td>
</tr>
<tr>
<td>KUHN</td>
<td>3</td>
<td>7</td>
<td>-2</td>
<td>5 11 0</td>
<td>1</td>
</tr>
<tr>
<td>M&amp;S-1</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>M&amp;S-2</td>
<td>4</td>
<td>7</td>
<td>-2</td>
<td>4 14 1</td>
<td>1</td>
</tr>
<tr>
<td>SOLOW</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>3 7 1</td>
<td>1</td>
</tr>
<tr>
<td>SIERKSA</td>
<td>2</td>
<td>6</td>
<td>unbounded</td>
<td>3 6 0</td>
<td>1</td>
</tr>
<tr>
<td>CHVATAL</td>
<td>3</td>
<td>7</td>
<td>unbounded</td>
<td>3 6 0</td>
<td>1</td>
</tr>
<tr>
<td>NAT</td>
<td>3</td>
<td>8</td>
<td>unbounded</td>
<td>4 9 1</td>
<td>2</td>
</tr>
<tr>
<td>HAM</td>
<td>4</td>
<td>6</td>
<td>unbounded</td>
<td>0 0 0</td>
<td>1</td>
</tr>
</tbody>
</table>
### 8. Summary and Conclusion

In this work, we have investigated two non-simplex methods for solving linear programs, one for constraints in standard form, and one for a mixture of general equality and inequality constraints. Both of these methods involve solving a linear least-squares subproblem and terminate in finitely many iterations. We also propose a new active-set method based on adding a penalty term to the linear objective function.

The numerical results given in Section 7 indicate that the proposed method is computationally efficient. It is shown that both PDLP and PDNNLS perform well on highly degenerate problems. However, PDLP is more efficient than PDNNLS in terms of the number of iterations needed to solve the linear least-squares problem, and it also requires less time as the size of the problem increases. In addition, the results presented in Table 2 indicate that the “primal-first” option of PDLP is more efficient than the “dual-first” option in terms of both iteration numbers and running times. In general, the relative efficiency of the “primal-first” and “dual-first” options will depend on the linear program being solved. For degenerate problems, both methods converge in finitely many iterations.

### Acknowledgements

I would like to thank Professor Philip Gill for taking the time to advise me on this one-year-long project. I am deeply grateful for his patience and encouragement. I believe the experience and knowledge that he has given me will last a lifetime.

I would also like to thank Professor Anders Forsgren for the source code of the assignment problem, which helped me a lot in generating the numerical results.

---

1Some entries are intentionally left blank because the initial points calculated for the initial basis are either primal feasible or dual feasible. If the points are primal feasible at the beginning, only entries for “primal-first” are recorded, and vice versa.
References


