Linear Programming

Gabriele Ciaramella

April 11, 2019

Contents

Preface

These lecture notes are a gentle introduction to the field of linear programming with a numerically oriented perspective. They correspond to the coarse taught by the author at the University of Pavia in March 2019. The goal was to introduce graduate Engineering students to the field of numerical optimization. In particular, these notes can be used by graduate students in Engineering, who wish to go beyond a blind use of black-box routines, and by undergraduate students in mathematics, who wish to get in touch with optimization problems.

The references that I used to write these lecture notes are [10], a standard reference in the field of numerical optimization, and [5], where the reader can find several MATLAB implementation of many algorithms.

I would like to acknowledge Prof. Davide M. Raimondo, who gave me the opportunity to teach this class in Pavia. I thank him for his careful reading of these short notes and for providing many useful feedbacks.

March 2019,
Gabriele Ciaramella
gabriele.ciaramellauni-konstanz.de

Chapter 1

Linear Programming

1.1 Linear programming problems

A linear programming (LP) problem in *standard form* is

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.1}$$

where A is an m by n matrix in $\mathbb{R}^{m\times n}$, with n, m two positive integers, $\mathbf{b}$ is a vector in $\mathbb{R}^m$, $\mathbf{c}$ is a vector in $\mathbb{R}^n$. The solution $\mathbf{x}$ is in $\mathbb{R}^n$ and the inequality $\mathbf{x} \geq 0$ is understood in an element-wise sense:

$$\mathbf{x} \geq 0 \quad \Leftrightarrow \quad (\mathbf{x})_j \geq 0 \ \ \forall j \in \{1, \ldots, n\},$$

which means that each component $(\mathbf{x})_j$ of the vector $\mathbf{x}$ must be non-negative. Clearly, A, $\mathbf{b}$ and $\mathbf{c}$ are given data. The function $f : \mathbb{R}^n \to \mathbb{R}$ defined as $f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x}$ is called *cost function* and it is linear in $\mathbf{x}$.

We introduce the *feasible set* associated to (1.1):

$$F := \{\mathbf{y} \in \mathbb{R}^n \ : \ A\mathbf{y} = \mathbf{b}, \, \mathbf{y} \geq 0\}. \tag{1.2}$$

This is the set of all vectors in $\mathbb{R}^n$ that satisfies the constraints of (1.1). The LP problem (1.1) is equivalent to

$$\min_{\mathbf{x}\in F} \ f(\mathbf{x}). \tag{1.3}$$

Let us consider some examples to illustrate the concept of an LP problem and its feasible set F.

Example 1 (The feasible set is a segment line)**.** *Consider the LP problem* (1.1) *with* $A = \begin{bmatrix} 1 & 2 \end{bmatrix}$, $\mathbf{b} = 3$ *and* $\mathbf{c} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$. *The linear system* $A\mathbf{x} = \mathbf{b}$

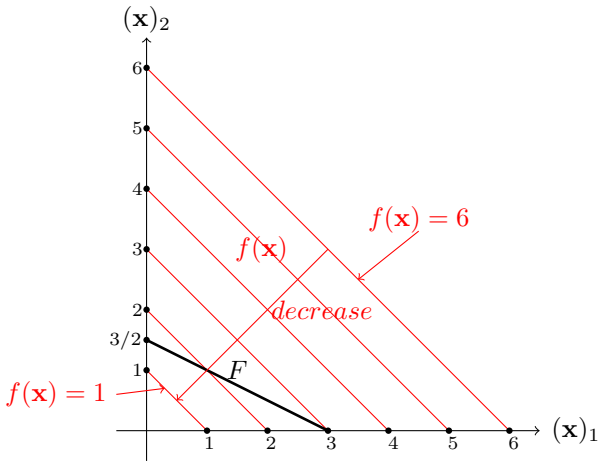

Figure 1.1: The black thick line is the feasible set F. The red lines are the isolines of the cost function $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. The value of $f(\mathbf{x})$ decreases in the direction pointed by the red arrow. The minimum of $f(\mathbf{x})$ over F is clearly attained at $\mathbf{x} = (0, 3/2)$.

is equivalent to $(\mathbf{x})_2 = 3/2 - (\mathbf{x})_1/2$. *Therefore, to guarantee that* $(\mathbf{x})_2 \geq 0$, $(\mathbf{x})_1$ *must be bounded by* 3. *Moreover, since* $\mathbf{x} \geq 0$, $(\mathbf{x})_1$ *must be non-negative. Hence,* $(\mathbf{x})_1 \in [0, 3]$ *and the feasible set is*

$$F = \{\mathbf{x} \in \mathbb{R}^2 \,:\, (\mathbf{x})_1 \in [0, 3],\, (\mathbf{x})_2 = 3/2 - (\mathbf{x})_1/2\}$$

and corresponds to the line segment shown in Figure 1.1 (black thick line). In Figure 1.1 the red lines are the isolines of the cost function $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. *In particular, the isolines* $f(\mathbf{x}) = 1$ *and* $f(\mathbf{x}) = 6$ *are indicated by two arrows. The isolines decrease in the direction pointed by the red arrow. Therefore, one can easily see that the minimum of* $f(\mathbf{x})$ *over* F *is clearly attained at* $\mathbf{x} = (0, 3/2)$, *which is the unique solution to* (1.1). *To see it, one can also rewrite* $f(\mathbf{x})$ *as a function of the* $(\mathbf{x})_1$ *component only by using that* $(\mathbf{x})_2 = 3/2 - (\mathbf{x})_1/2$*:*

$$f(\mathbf{x}) = (\mathbf{x})_1 + (\mathbf{x})_2 = 3/2 + (\mathbf{x})_1/2.$$

The minimum of this function for $(\mathbf{x})_1 \in [0, 3]$ *is attained at* $(\mathbf{x})_1 = 0$. *The corresponding value of* f *at* $\mathbf{x} = [0,\, 3/2]^\top$ *is* $3/2$.

Example 2 (The feasible set is a single point)**.** *Consider the LP problem* (1.1) *with* $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$ *and* $\mathbf{c} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$. *Notice that the matrix* A *is invertible (*$\det A \neq 0$*). Therefore, the linear system* $A\mathbf{x} = \mathbf{b}$ *is uniquely solved by* $\mathbf{x}^\star = A^{-1}\mathbf{b} = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top$. *This vector satisfies the constraint* $\mathbf{x}^\star \geq 0$. *Since* $\mathbf{x}^\star$ *is the unique vector that satisfies the constraint* $A\mathbf{x} = \mathbf{b}$, *the feasible set* F *contains only the point* $\mathbf{x}^\star$*:* $F = \{\mathbf{x}^\star\}$. *Moreover, it is clear that the minimum of* $f(\mathbf{x})$ *over* F *is* $\mathbf{x}^\star$.

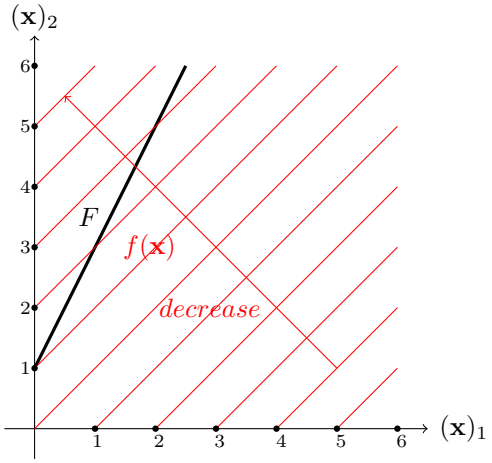

Figure 1.2: The black thick line is the feasible set F. The red lines are the isolines of the cost function $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. The value of $f(\mathbf{x})$ decreases in the direction pointed by the red arrow. Notice that the cost function $f(\mathbf{x})$ becomes negative and arbitrarily small along the feasible set F, which indicates that the problem is unbounded and has no solution.

Example 3 (The feasible set is empty)**.** *Consider the LP problem* (1.1) *with* $A = \begin{bmatrix} 1 & 2 \end{bmatrix}$, $\mathbf{b} = -3$ *and* $\mathbf{c} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$. *Notice that these data coincide with the ones of Example 1, but we switched the sign of* $\mathbf{b}$. *The linear system* $A\mathbf{x} = \mathbf{b}$ *is equivalent to* $(\mathbf{x})_2 = -3/2 - (\mathbf{x})_1/2$. *This implies that for every* $(\mathbf{x})_1 \geq 0$ *it holds that* $(\mathbf{x})_2 \leq -3/2 < 0$. *Therefore, there are no non-negative vectors in* $\mathbb{R}^2$ *that satisfy the constraint* $A\mathbf{x} = \mathbf{b}$. *This means that the feasible set* F *is empty and the LP problem has no solution.*

Example 4 (The feasible set is a straight line)**.** *Consider the LP problem* (1.1) *with* $A = \begin{bmatrix} -2 & 1 \end{bmatrix}$, $\mathbf{b} = 1$ *and* $\mathbf{c} = \begin{bmatrix} 1 & -1 \end{bmatrix}^\top$. *The linear system* $A\mathbf{x} = \mathbf{b}$ *is equivalent to* $(\mathbf{x})_2 = 1 + 2(\mathbf{x})_1$. *This implies that* $(\mathbf{x})_2 > 0$ *for every* $(\mathbf{x})_1 \geq 0$. *Therefore, there the feasible set* F *is the straight line* $(\mathbf{x})_2 = 1 + 2(\mathbf{x})_1$ *with* $(\mathbf{x})_1 \geq 0$, *which is depicted in Figure 1.2 with a black thick line. The LP problem corresponding to this example is unbounded because* $f(\mathbf{x}) = (\mathbf{x})_1 - (\mathbf{x})_2 = -(\mathbf{x})_1 - 1$ *for any* $\mathbf{x} \in F$, *which implies that* $\lim_{\mathbf{x}\to\infty, \mathbf{x}\in F} f(\mathbf{x}) = -\infty$. *This problem has therefore no solution.*

Having studied the above examples, the next definition follows naturally.

Definition 1 (Infeasible and unbounded LP problems)**.** *An LP problem of the form* (1.1) *is said to be infeasible if its feasible set* F *is empty* ($F = \emptyset$).

An LP problem of the form (1.1) *is said to be unbounded if there exists a feasible sequence* $\{\mathbf{x}_k\}_{k\in\mathbb{N}}$ ($\mathbf{x}_k \in F$) *such that* $\lim_{k\to\infty} f(\mathbf{x}_k) = -\infty$.

Clearly infeasible and unbounded LP problems admit no solution.

In several applications a LP problem is formulated in forms that look different from the standard form (1.1). An LP problem in *canonical form* is

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.4}$$

where we remark the sign "$\leq$" in $A\mathbf{x} \leq \mathbf{b}$ (instead of "="). Another very common form of LP problems is

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq \mathbf{b}, \end{aligned} \tag{1.5}$$

where the constraint $\mathbf{x} \geq 0$ is missing. Other examples are also possible.

Luckily, these LP problems can be reformulated in standard form. Consider problem (1.4). For any $\mathbf{x} \in \mathbb{R}^n$ such that $A\mathbf{x} \leq \mathbf{b}$ there exists a vector $\mathbf{s} \in \mathbb{R}^m$ such that

$$A\mathbf{x} + \mathbf{s} = \mathbf{b}.$$

This vector is called *slack variable* and is clearly non-negative

$$\mathbf{s} = \mathbf{b} - A\mathbf{x} \geq 0,$$

where we used the constraint $A\mathbf{x} \leq \mathbf{b}$. Let us now introduce the variables

$$\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix} \in \mathbb{R}^{n+m} \quad \text{and} \quad \mathbf{d} = \begin{bmatrix} \mathbf{c} \\ 0 \end{bmatrix} \in \mathbb{R}^{n+m},$$

and the matrix

$$\widetilde{A} = \begin{bmatrix} A & I \end{bmatrix} \in \mathbb{R}^{m\times(n+m)},$$

where I is the $m \times m$ identity matrix. The problem (1.4) is then equivalent to

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^{n+m}} \quad & \mathbf{d}^\top \mathbf{y} \\ \text{s.t.} \quad & \widetilde{A}\mathbf{y} = \mathbf{b} \\ & \mathbf{y} \geq 0, \end{aligned} \tag{1.6}$$

which is in standard form.

Now, we wish to formulate problem (1.5) in standard form. To do so, we introduce the variables $\mathbf{x}^+$ and $\mathbf{x}^-$ in $\mathbb{R}^n$ and rewrite any vector $\mathbf{x} \in \mathbb{R}^n$ as $\mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-$ with $\mathbf{x}^+ \geq 0$ and $\mathbf{x}^- \geq 0$. Thus we write $A\mathbf{x} \geq \mathbf{b}$ as

$$A\mathbf{x}^+ - A\mathbf{x}^- \geq \mathbf{b}.$$

As before, we can introduce a slack variable $\mathbf{s} \in \mathbb{R}^m$ such that

$$A\mathbf{x}^+ - A\mathbf{x}^- - \mathbf{s} = \mathbf{b}$$

with $\mathbf{s} \geq 0$. If we introduce the vectors

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \\ \mathbf{s} \end{bmatrix} \in \mathbb{R}^{2n+m} \quad \text{and} \quad \mathbf{d} = \begin{bmatrix} \mathbf{c} \\ -\mathbf{c} \\ 0 \end{bmatrix} \in \mathbb{R}^{2n+m},$$

and the matrix

$$\widetilde{A} = \begin{bmatrix} A & -A & I \end{bmatrix} \in \mathbb{R}^{m \times (2n+m)},$$

where I is the $m \times m$ identity matrix, then problem (1.5) is then equivalent to

$$\begin{aligned} \min_{\mathbf{y} \in \mathbb{R}^{2n+m}} \quad & \mathbf{d}^\top \mathbf{y} \\ \text{s.t.} \quad & \widetilde{A}\mathbf{y} = \mathbf{b} \\ & \mathbf{y} \geq 0, \end{aligned}$$

which is in standard form.

Notice that when one formulates (1.4) or (1.5) in standard form the dimension of the unknown vector is augmented. In particular, transforming (1.4) in standard form, the dimension n is augmented to $n+m$, while transforming (1.5) in standard form, the dimension n is augmented to $2n + m$.

Let us now give a very practical example/exercise of a classical LP problem considered in engineering.

Exercise 1 (Product mix problem[1])**.** *The product mix problem is*

$$\begin{aligned} \min_{x_1, x_2} \quad & 20x_1 + 30x_2 \\ \text{s.t.} \quad & x_1 \leq 70 \\ & x_2 \leq 50 \\ & x_1 + 2x_2 \leq 120 \\ & x_1 + x_2 \leq 90 \\ & x_1 \geq 0 \\ & x_2 \geq 0. \end{aligned} \tag{1.7}$$

As an exercise, the reader can rewrite (1.7) *in standard and canonical forms.*

Exercise 2. *Consider the following LP problem*

$$\begin{aligned} \min_{x_1, x_2, x_3} \quad & c_1x_1 + c_2x_2 + c_3x_3 \\ \text{s.t.} \quad & x_1 \leq 70 \\ & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & a_{22}x_2 + a_{23}x_3 \geq b_2 \\ & a_{31}x_1 + a_{32}x_3 = b_3 \\ & x_1 \geq 0 \\ & x_2 \leq 0. \end{aligned} \tag{1.8}$$

Rewrite (1.8) *in standard and canonical forms.*

[1]The Product mix problem will be explained by Prof. Raimondo in his lectures.

1.2 Geometry of feasible sets

In this section, we discuss geometrical properties of feasible sets. We begin by defining hyperplanes.

Definition 2 (Hyperplane and supporting hyperplane)**.** *Let* $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a} \neq 0$ *and* $b \in \mathbb{R}$. *The set*

$$H := \{\mathbf{x} \in \mathbb{R}^n \,:\, \mathbf{a}^\top \mathbf{x} = b\}$$

is called hyperplane. Consider the half spaces

$$H^+ := \{\mathbf{x} \in \mathbb{R}^n \,:\, \mathbf{a}^\top \mathbf{x} \geq b\} \text{ and } H^- := \{\mathbf{x} \in \mathbb{R}^n \,:\, \mathbf{a}^\top \mathbf{x} \leq b\}.$$

The boundary of H^+ *and* H^- *is called supporting hyperplane.*

We can now introduce the definitions of a polyhedron and a polytope.

Definition 3 (Polyhedron)**.** *A polyhedron in* $\mathbb{R}^n$ *is the intersection of a finite (but positive) number of half spaces in* $\mathbb{R}^n$. *If* K *is a polyhedron, then there exists a matrix* $A \in \mathbb{R}^{m \times n}$ *and a vector* $\mathbf{b} \in \mathbb{R}^m$ *such that*

$$K = \{\mathbf{x} \in \mathbb{R}^n \,:\, A\mathbf{x} \leq \mathbf{b}\}.$$

Definition 4 (Polytope)**.** *A polytope* K *is a bounded*[2] *polyhedron.*

Some examples of polyhedra and polytopes in $\mathbb{R}^2$ are given in Figure 1.3. Three-dimensional examples are given in Figure 1.4. To create these plots the reader can use the MATLAB package 'MPT3'. To install this package it is sufficient to download the file

`install_mpt3.m`

from the website

`https://www.mpt3.org/Main/Installation`

and run it in Matlab. Once the installation is completed, one can easily plot the polyhedron depicted in Figure 1.4 (right) by running the following MATLAB script.[3].

```
A=[1 1 1/2; 1/2 -1/3 1];
b=[1;1/2];
A=[A;-eye(3)];
b=[b;zeros(3,1)];
polyh=Polyhedron('A',A,'b',b);
figure();
set(gca,'FontSize',22);
plot(polyh);
xlabel('$$(\bf x)_1$$','interpreter','latex','FontSize',20);
ylabel('$$(\bf x)_2$$','interpreter','latex','FontSize',20);
zlabel('$$(\bf x)_3$$','interpreter','latex','FontSize',20);
view(70,45);
```

[2]Bounded means that there exists a ball B of finite radius such that $K \subset B$.

[3]Notice that, if one simply copy/pastes the script in MATLAB there could be a error due to the symbol of the apostrophe that must be replaced with " ' "

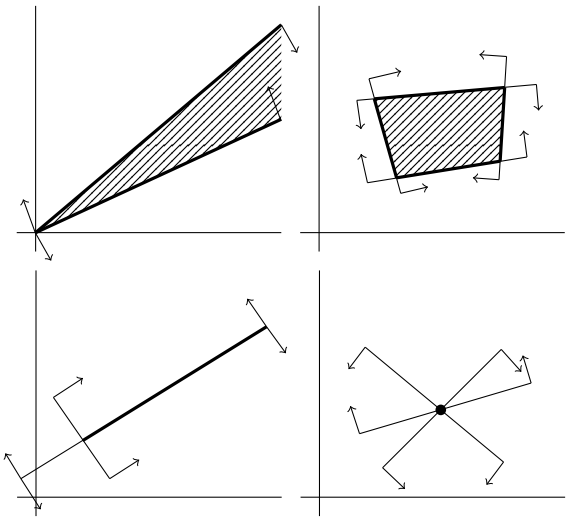

Figure 1.3: Four examples of polyhedra obtained as the intersection of half spaces. Arrows indicate half spaces. Solid thick black lines and hatch filling represent the polyhedra. Top left: an unbounded polyhedron obtained as the intersection of two half spaces. Top right: a bounded polyhedron (a polytope) obtained as the intersection of four half spaces. Bottom left: an unbounded polyhedron (the straight thick black line) obtained as the intersection of three half spaces. Bottom right: a polyhedron given by a point and obtained as the intersection of three half spaces.

The polyhedra depicted in Figure 1.4 (left and middle) can be obtained similarly.

Let us now consider some remarks.

Remark 1.

- *A polytope is a convex set.*
- *The feasible set of an LP problem is a polyhedron; see, e.g.,* (1.5).
- *Let* $A \in \mathbb{R}^{m\times n}$ *and* $\mathbf{b} \in \mathbb{R}^m$ *define the polyhedron* $K := \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$. *The pair* $(A, \mathbf{b})$ *that induces* K *is not unique:*
  - *If* $\widetilde{A}$ *and* $\widetilde{\mathbf{b}}$ *are obtained by row permutations of* A *and* $\mathbf{b}$, *then* $\{\mathbf{x} \in$

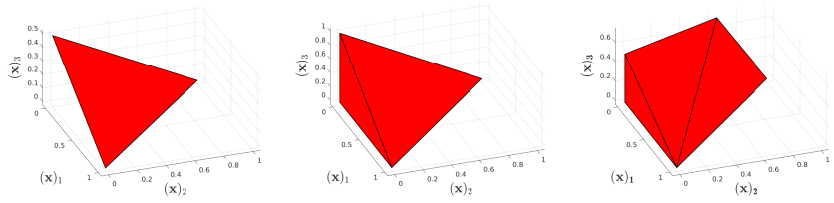

Figure 1.4: Examples of three-dimensional polytopes.

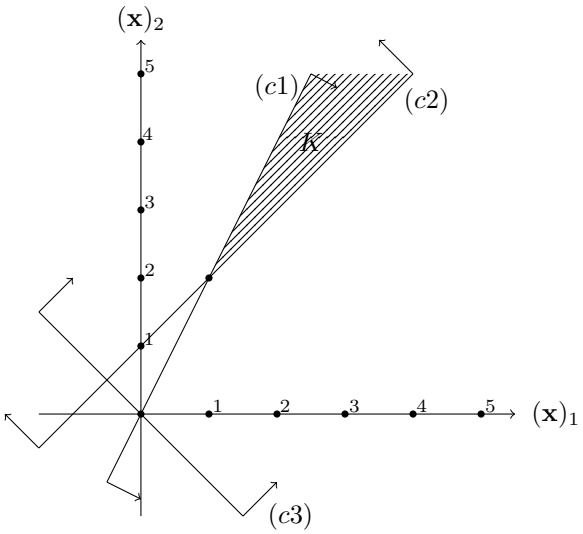

Figure 1.5: Polyhedron K of Example 5. The polyhedron K is obtained as the intersection of the half spaces corresponding to the constraints $(c1)$ and $(c2)$. Notice that the constraint $(c3)$ is redundant because the intersection of the three half spaces corresponding to $(c1)$, $(c2)$ and $(c3)$ is equal to the intersection of the half spaces corresponding to $(c1)$ and $(c2)$.

$$\mathbb{R}^n \,:\, \widetilde{A}\mathbf{x} \leq \widetilde{\mathbf{b}}\} = K.$$

– *The pair* $(\alpha A, \alpha \mathbf{b})$ *for any* $\alpha > 0$ *defines* K.

- *A constraint in* $A\mathbf{x} \leq \mathbf{b}$ *is redundant if* K *does not change when this constraint is removed. See Example 5.*

Example 5. *Consider the matrix* $A = \begin{bmatrix} -2 & 1 \\ 1 & -1 \\ -1 & -1 \end{bmatrix}$ *and the vector* $\mathbf{b} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ *and the polyhedron* $K = \{\mathbf{x} \in \mathbb{R}^n \,:\, A\mathbf{x} \leq \mathbf{b}\}$. *The inequality constraint* $A\mathbf{x} \leq \mathbf{b}$ *is equivalent to*

$$\begin{aligned} &(c1) \quad (\mathbf{x})_2 \leq 2(\mathbf{x})_1, \\ &(c2) \quad (\mathbf{x})_2 \geq (\mathbf{x})_1 + 1, \\ &(c3) \quad (\mathbf{x})_2 \geq -(\mathbf{x})_1. \end{aligned}$$

If one draws the three half spaces defined via the three inequalities (c1), (c2) and (c3), then one obtains the picture given in Figure 1.5. It is clear from the picture that the intersection of the three half spaces is equal to the intersection of the two half spaces induced by (c1) and (c2). This means that the constraint (c3) is redundant.

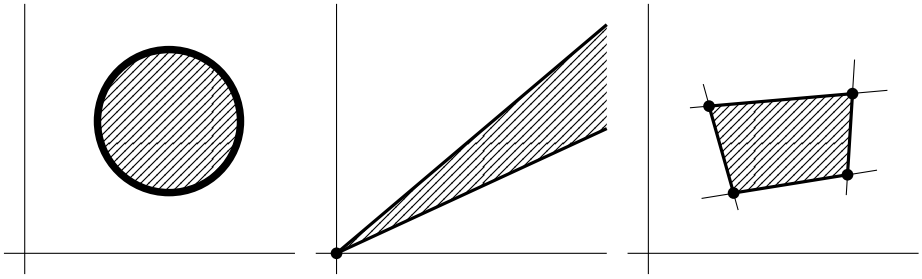

Figure 1.6: Left: a disk. Any point on the boundary of the disk is an extreme point. Middle: a cone. The only extreme point is indicated by the black point and corresponds to the corner point of the cone. Right: a polygon. The four corner points are extreme points.

Definition 5 (Extreme points of a set)**.** *Let $S \subset \mathbb{R}^n$ be a convex set. A point $\mathbf{z} \in S$ is called extreme point if there are no two points $\mathbf{x}$, $\mathbf{y} \in S$ different from $\mathbf{z}$ such that $\mathbf{z}$ belongs to the segment $\overline{\mathbf{xy}}$.*

Example 6 (Extreme points)**.** *We consider three examples corresponding to the three pictures given in Figure 1.6. In Figure 1.6 (left) we consider a disk. Any point on the boundary of the disk does not lie on a segment line connecting any other two points of the disk. Therefore, any point on the boundary of the disk is an extreme point.*

In Figure 1.6 (middle) we consider a cone. The only extreme point of this cone is the corner point. For any other point $\mathbf{z}$ in the cone (and on its boundary) one can always find two other points $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{z}$ lies on the segment $\overline{\mathbf{xy}}$.

In Figure 1.6 (right) we consider a polygon. Clearly, the extreme points of this polygon are the four corner points.

Remark 2.

- *A polyhedron has a finite number (zero is possible) of extreme points and they are called vertexes.*
- *Let P be a polytope. Every point $\mathbf{x} \in P$ is a convex combination of the vertexes of P[4], that is $\mathbf{x}$ can be written as*

$$\mathbf{x} = \sum_{j=1}^{k} \lambda_j \mathbf{v}_j,$$

where k is the number of vertexes of P, $\mathbf{v}_j$ are the vertexes of P for $j = 1, \ldots, k$, and λ_j are non-negative numbers such that $\sum_{j=1}^{k} \lambda_j = 1$.

[4]Notice that a polytope is a convex set (see Remark 1). Therefore this result follows from the famous Carathéodory's theorem; see, e.g., `https://en.wikipedia.org/wiki/Carath%C3%A9odory%27s_theorem_(convex_hull)`

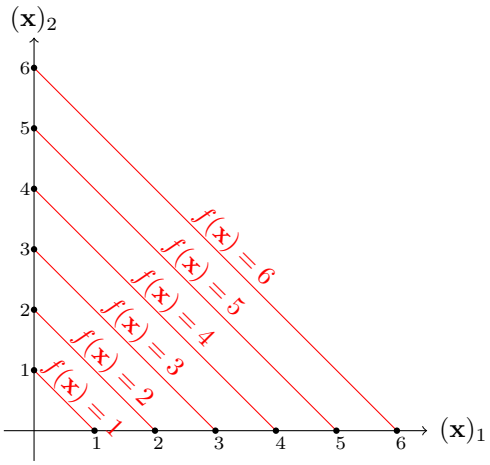

Figure 1.7: Isocost lines of $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$ with $\mathbf{c} = [1\ 1]^\top$.

1.3 Graphical solution of two-dimensional LP problems

In $\mathbb{R}^2$ one can solve a LP problem using a graphical method. To do so, we need the following definition.

Definition 6 (Isocost lines)**.** *Consider an LP problem in the form* (1.1)*. The isocost lines are the isolines (or the contour lines) of the cost function* $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$*. In particular, the isocost line corresponding to a value* $\alpha \in \mathbb{R}$ *is the set*

$$T_\alpha = \{\mathbf{x} \in \mathbb{R}^2 \,:\, f(\mathbf{x}) = \alpha\}.$$

Different values of α *induce different isocost lines.*

Let us give an example of isocost lines.

Example 7 (Isocost lines)**.** *Consider the cost function* $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$ *with* $\mathbf{c} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$ *(as in Example 1). The isocost line corresponding to a given* $\alpha \in \mathbb{R}$ *is the set of points* $I_\alpha \subset \mathbb{R}^2$ *such that* $f(\mathbf{x}) = \alpha$*, which is equivalent to*

$$\{\mathbf{x} \in \mathbb{R}^2 \,:\, (\mathbf{x})_2 = \alpha - (\mathbf{x})_1\}.$$

For example, for $\alpha = 1$*, the isoline* I_1 *is the line obtained by the equation* $(\mathbf{x})_2 = 1 - (\mathbf{x})_1\}$*, that is a straight line with negative slope that intersects the vertical axis at the point* $((\mathbf{x})_1, (\mathbf{x})_2) = (0, 1)$*. The isocost lines of* $f(\mathbf{x})$ *are depicted in Figure 1.7.*

The graphical method for solving a two-dimensional LP problem can be summarized by the following steps:

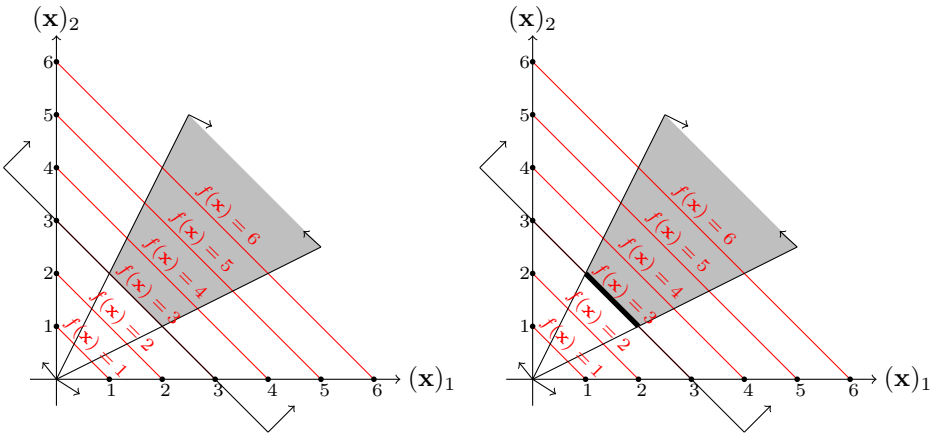

Figure 1.8: Step 2 (left) and Step 3 (right) of the graphical method. The gray region represents the feasible set F.

1. Draw the feasible set F.
2. Draw the isocost lines.
3. Identify the solution(s).

Example 8 (Graphical method)**.** *Consider the LP problem*

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b}, \end{aligned}$$

where $A = \begin{bmatrix} -2 & 1 \\ 1 & -2 \\ -1 & 1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ -3 \end{bmatrix}$ *and* $\mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. *To apply the graphical method, one must perform the Step 1 and draw the isocost lines for* $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. *This is exactly what we have done in Figure 1.7. The Step 2 consists in drawing on top of the isocost lines the feasible set* F. *This is done in Figure 1.8 (left). Once isocost lines and feasible set* F *are obtained, one can easily identify the minima of* $f(\mathbf{x})$ *on* F, *which correspond to the points where the feasible set* F *intersects the isocost line having minimal value. In our example, the segment depicted with a thick black line in Figure 1.8 (right) is the set of minima. The value of* $f(\mathbf{x})$ *corresponding to any minima is* $f(\mathbf{x}) = 3$.

1.4 Bases and basic feasible points

In what follows, we consider the LP problem (1.1) and assume that $m < n$ and that the matrix A is full row rank ($\operatorname{rank} A = m$), otherwise the system contains redundant rows, or is infeasible, or defines a unique point.

We shall see that the solutions to (1.1) (or at least some of them) can be characterized in terms of *basic feasible points*.

Definition 7 (Basic feasible point, basis, basis matrix)**.** *A vector $\mathbf{x} \in \mathbb{R}^n$ is a basic feasible point if it is feasible ($\mathbf{x} \in F$) and if there exists a subset S of the index set $\{1, 2, \ldots, n\} \subset \mathbb{N}$ such that*

- *S contains exactly m indexes,*
- *$j \notin S \Rightarrow (\mathbf{x})_j = 0$ and*
- *the matrix B defined as $B = [A_j]_{j \in S}$ is non-singular, where A is the j^{th} column of A.*

A set S satisfying these properties is called a basis and the corresponding matrix B is called basis matrix.

Example 9 (Basic feasible point)**.** *Consider the polyhedron $K = \{\mathbf{x} \in \mathbb{R}^3 : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$, where $A = [1 \quad 1 \quad -1]$ and $\mathbf{b} = 1$. Clearly $n = 3$ and $m = 1$. We wish to find the basic feasible points in K. Definition 7 says that, a subset S to be a basis must contain exactly m elements. Moreover, the corresponding vector can have non-zero components only if their index is an element in S. Therefore, there are only three vectors that are candidates to be basic feasible points:*

$$S = \{1\}, \mathbf{v} = \begin{bmatrix} \alpha_1 \\ 0 \\ 0 \end{bmatrix}, \quad S = \{2\}, \mathbf{w} = \begin{bmatrix} 0 \\ \alpha_2 \\ 0 \end{bmatrix}, \quad S = \{3\}, \mathbf{z} = \begin{bmatrix} 0 \\ 0 \\ \alpha_3 \end{bmatrix},$$

where α_1, α_2 and α_3 must be determined. Notice also that the matrix B corresponding to these three possibilities is a non-zero scalar, hence invertible. Since a basic feasible point must be feasible, we can determine the values α_j by requiring that $\mathbf{v}$, $\mathbf{w}$ and $\mathbf{z}$ satisfy the constraint $A\mathbf{x} = \mathbf{b}$:

$$A\mathbf{v} = \mathbf{b} \Rightarrow \alpha_1 = 1, \quad A\mathbf{w} = \mathbf{b} \Rightarrow \alpha_2 = 1, \quad A\mathbf{z} = \mathbf{b} \Rightarrow \alpha_3 = -1.$$

Hence, we have

$$\mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}.$$

Notice that $\mathbf{v} \geq 0$ and $\mathbf{w} \geq 0$, but $(\mathbf{z})_3 = -1 < 0$. Therefore, only $\mathbf{v}$ and $\mathbf{w}$ are feasible. We have obtained that the basic feasible points are $\mathbf{v}$ and $\mathbf{w}$. Let us draw the polyhedron K and the two points $\mathbf{v}$ and $\mathbf{w}$. To do so, we use the MATLAB *script*

```
n=3; m=1;
A=[1 1 -1];
b=ones(m,1);
polyh=Polyhedron('A',-eye(n),'b',0*ones(n,1),'Ae',A,'be',b);
figure();
```

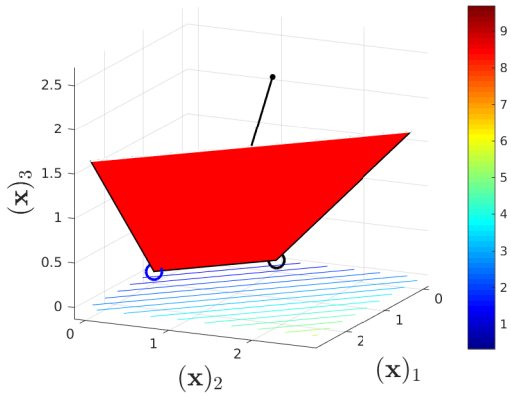

Figure 1.9: Polyhedron K considered in Example 9 and the two basic feasible points $\mathbf{v}$ (black circle) and $\mathbf{w}$ (blue circle).

```
set(gca,'FontSize',22);
plot(polyh);
hold on;
plot(1,0,'bo','LineWidth',2,'MarkerSize',12);
plot(0,1,'ko','LineWidth',2,'MarkerSize',12);
[X,Y] = meshgrid([0:.25:5]);
Z = X+Y;
contour(X,Y,Z,30);
colormap('jet');
colorbar;
xlabel('$$({\bf x})_1$$','interpreter','latex','FontSize',20);
ylabel('$$({\bf x})_2$$','interpreter','latex','FontSize',20);
zlabel('$$({\bf x})_3$$','interpreter','latex','FontSize',20);
view(115,15);
```

which produces the picture given in Figure 1.9. We can clearly see that the two basic feasible points $\mathbf{v}$ *and* $\mathbf{w}$ *are the two vertexes of the polyhedron* K*.*

Consider now the LP problem $\min_{\mathbf{x}\in K} f(\mathbf{x}) := \mathbf{c}^\top \mathbf{x}$ *with* $\mathbf{c}^\top = [1\,1\,0]$*. The function* $f(\mathbf{x})$ *is constant with respect to* $(\mathbf{x})_3$ *and its isolines are depicted in Figure 1.9. By comparing the set* K*, the two basic feasible points* $\mathbf{v}$ *and* $\mathbf{w}$*, and the isolines of* $f(\mathbf{x})$*, we observe that the two basic feasible points are two minima for the considered LP problem.*

In Example 9, we computed the basic feasible points of a given polyhedron K. We also noticed that these points are two solutions of an LP problem and vertexes of K. Is this a coincidence? The answer is given by the following theorems.

Theorem 1 (Fundamental theorem of linear programming)**.** *Let* A *be a full row rank matrix in* $\mathbb{R}^{m\times n}$ *(with* $m < n$*). It holds that*

- *If* (1.1) *has a non-empty feasible set, then there is at least one basic feasible point.*
- *If* (1.1) *has solutions, then at least one such solution is a basic feasible (optimal) point.*
- *If* (1.1) *is feasible and bounded, then it has an optimal solution.*

Proof. See [10, Theorem 13.2]. □

Theorem 1 says that, if (1.1) is feasible and bounded, then there exist solutions (minimizers) and among them at least one point is a basic feasible point. This is an important remark that will be used to define numerical algorithms for solving (1.1).

Theorem 2 (Basic feasible points and vertexes)**.** *Let A be a full row rank matrix in $\mathbb{R}^{m\times n}$. All basic feasible points for* (1.1) *are vertexes of the feasible polyhedron*

$$K = \{\mathbf{x} \in \mathbb{R}^n \,:\, A\mathbf{x} = \mathbf{b},\, \mathbf{x} \geq 0\}$$

and viceversa.

Proof. See [10, Theorem 13.3]. □

An important property when studying the convergence of a numerical algorithm is the so-called *degeneracy* of a basic feasible point.

Definition 8 (Degeneracy)**.** *A basis S is said to be degenerate if $(\mathbf{x})_j = 0$ for some $j \in S$, where $\mathbf{x}$ is the basic feasible point corresponding to S. A linear program is said to be degenerate if it has at least one degenerate basis.*

Let us now give some further remarks about a basis and its corresponding feasible point. Consider a basis S and the corresponding basis matrix $B \in \mathbb{R}^{m\times m}$. By definition, the matrix B must be invertible. Therefore, the (unique) feasible point $\mathbf{x}$ corresponding to S can be computed by solving the linear system $B\mathbf{v} = \mathbf{b}$ and then setting for $j = 1, \ldots, n$

$$(\mathbf{x})_j = \begin{cases} 0 & \text{if } j \notin S, \\ (\mathbf{v})_i & \text{if } j \in S \text{ and } B_i = A_j, \end{cases} \tag{1.9}$$

where B_i and A_j denote the i^{th} and j^{th} columns of B and A. Notice that the vector $\mathbf{x}$ constructed as in (1.9) satisfies the constraint $A\mathbf{x} = \mathbf{b}$. To see it, let us define the set $S^c := \{1, \ldots, n\} \setminus S$, the vectors

$$\mathbf{x}_S = [(\mathbf{x})_j]_{j\in S} \in \mathbb{R}^m \text{ and } \mathbf{x}_N = [(\mathbf{x})_j]_{j\in S^c} \in \mathbb{R}^{n-m}$$

and the matrix

$$N = [A_j]_{j\in S^c} \in \mathbb{R}^{n\times(n-m)}.$$

Clearly, $\mathbf{x}_N = 0$ because of (1.9). We can then compute

$$\mathbf{b} = B\mathbf{x}_S = B\mathbf{x}_S + N\mathbf{x}_N = A\mathbf{x}.$$

Suppose to have a given subset S (of exactly m elements) of the index set $\{1, \dots, n\}$. How can we verify whether S is a basis or not? One can follow the simple procedure:

1. Construct the matrix B corresponding to S: $B = [A_j]_{j \in S}$.
2. Check the regularity of B (clearly if B is not invertible, then S is not a basis).
3. Solve the system $B\mathbf{v} = \mathbf{b}$.
4. Assemble the vector $\mathbf{x}$ as in (1.9).
5. Verify the non-negativity of $\mathbf{x}$ (clearly, if $\mathbf{x}$ has some negative elements, then S is not a basis).

Example 10 (Basic feasible point)**.** *Consider the matrix* $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{bmatrix}$ *and the vector* $\mathbf{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$*. Is the subset* $S = \{1, 3\}$ *of the index set* $\{1, 2, 3\}$ *a basis?*

Let us answer to this question by following the above procedure. The matrix B *corresponding to* S *is*

$$B = \begin{bmatrix} A_1 & A_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}.$$

This matrix is invertible ($\det B = 3$*). We can then solve the system* $B\mathbf{v} = \mathbf{b}$ *and get* $\mathbf{v} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$*. Now, we use* (1.9) *to define* $\mathbf{x}$*:*

$$\mathbf{x} = \begin{bmatrix} (\mathbf{x})_1 \\ (\mathbf{x})_2 \\ (\mathbf{x})_3 \end{bmatrix} = \begin{bmatrix} (\mathbf{v})_1 \\ 0 \\ (\mathbf{v})_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

where we used that the index 2 *is not in* S*. Notice that* $\mathbf{x} \geq 0$*, hence this is a basic feasible point. Moreover,* $(\mathbf{x})_1 = 0$ *with* $1 \in S$*, which implies that* S *is a degenerate basis.*

1.5 KKT system for LP problems

The *KKT system*[5] corresponding to the LP (1.1) is

$$A^\top \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c}, \tag{1.10}$$
$$A\mathbf{x} = \mathbf{b}, \tag{1.11}$$
$$\mathbf{x} \geq 0, \tag{1.12}$$
$$\mathbf{s} \geq 0, \tag{1.13}$$
$$(\mathbf{x})_j (\mathbf{s})_j = 0 \text{ for } j = 1, \dots, n, \tag{1.14}$$

[5] KKT stays for Karush-Kuhn-Tucker. These are the surnames of William Karush, Harold W. Kuhn, and Albert W. Tucker, who introduced this optimality system.

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\mathbf{s} \in \mathbb{R}^n$. The KKT system (1.10)-(1.14) is an optimality system. If $\mathbf{x}^\star \in \mathbb{R}^n$ is a solution to (1.1), then there exist two vectors $\mathbf{s}^\star \in \mathbb{R}^n$ and $\boldsymbol{\lambda}^\star \in \mathbb{R}^m$ such that the triple $(\mathbf{x}^\star, \boldsymbol{\lambda}^\star, \mathbf{s}^\star)$ solves the system (1.10)-(1.14). This is a necessary optimality condition.

The vectors $\mathbf{s}$ and $\boldsymbol{\lambda}$ are called *Lagrange multipliers*. In particular, $\mathbf{s}$ is the Lagrange multiplier corresponding to the constraint $\mathbf{x} \geq 0$, and $\boldsymbol{\lambda}$ is the Lagrange multiplier corresponding to $A\mathbf{x} = \mathbf{b}$. In general, the meaning of Lagrange multipliers can be briefly explain as follows. The value of each Lagrange multiplier tells us something about the sensitivity of the optimal value of the cost function $f(\mathbf{x})$ to the presence of the constraint. To put it another way, the value of the Lagrange multiplier indicates how hard the function $f(\mathbf{x})$ is "pushing" or "pulling" against the particular constraint; see [10].

The last condition (1.14) can be also written as

$$\mathbf{x}^\top \mathbf{s} = 0. \tag{1.15}$$

Notice that (1.15) and (1.14) are equivalent if (1.12) and (1.13) hold.

For general optimization problems the KKT system represents only a necessary optimality condition. However, it turns out that for LP problems, the KKT system is also a sufficient optimality condition.

Theorem 3 (Sufficient optimality condition)**.** *The KKT system* (1.10)-(1.14) *is a sufficient condition, in the sense that, if there exists a triple* $(\mathbf{x}^\star, \boldsymbol{\lambda}^\star, \mathbf{s}^\star) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ *that satisfies* (1.10)-(1.14)*, then* $\mathbf{x}^\star$ *is a global solution to* (1.1)*.*

Proof. We first compute

$$\begin{aligned} \mathbf{c}^\top \mathbf{x}^\star &= (A^\top \boldsymbol{\lambda}^\star + \mathbf{s}^\star)^\top \mathbf{x}^\star = (\boldsymbol{\lambda}^\star)^\top A \mathbf{x}^\star + (\mathbf{s}^\star)^\top \mathbf{x}^\star \\ &= \mathbf{b}^\top \boldsymbol{\lambda}^\star + (\mathbf{s}^\star)^\top \mathbf{x}^\star = \mathbf{b}^\top \boldsymbol{\lambda}^\star, \end{aligned}$$

where we used (1.10), (1.11) and (1.15). We have then obtained that

$$\mathbf{c}^\top \mathbf{x}^\star = \mathbf{b}^\top \boldsymbol{\lambda}^\star. \tag{1.16}$$

Now, we consider any other feasible point $\bar{\mathbf{x}} \in \mathbb{R}^n$ (which clearly satisfies $A\bar{\mathbf{x}} = \mathbf{b}$ and $\bar{\mathbf{x}} \geq 0$) and write

$$f(\bar{\mathbf{x}}) = \mathbf{c}^\top \bar{\mathbf{x}} = (A^\top \boldsymbol{\lambda}^\star + \mathbf{s}^\star)^\top \bar{\mathbf{x}} = \mathbf{b}^\top \boldsymbol{\lambda}^\star + \bar{\mathbf{x}}^\top \mathbf{s}^\star \geq \mathbf{b}^\top \boldsymbol{\lambda}^\star, \tag{1.17}$$

where (1.10), $A\bar{\mathbf{x}} = \mathbf{b}$, $\bar{\mathbf{x}} \geq 0$ and $\mathbf{s}^\star \geq 0$ are used. Using (1.17) and (1.16), we obtain

$$f(\bar{\mathbf{x}}) \geq \mathbf{b}^\top \boldsymbol{\lambda}^\star = \mathbf{c}^\top \mathbf{x}^\star = f(\mathbf{x}^\star).$$

This means that no feasible point can have a lower objective value than $f(\mathbf{x}^\star)$. Hence, $\mathbf{x}^\star$ is a global minimum for (1.1). □

Chapter 2

Numerical methods for LP problems

In Section 1.4, we studied the important results: Theorem 1 and Theorem 2. These say that in principle one could compute all the basic feasible points for the LP problem (1.1), evaluate the value of the cost function f at these points, and find a solution. This naive strategy is computationally unfeasible, because it would require the computation of all basic feasible points (all the vertexes of the polyhedron)! The number of basic feasible points is bounded from above by the quantity $\binom{n}{m} = \frac{n!}{m!(n-m)!}$, that is the total number of possible combinations of m (out of n) columns of A used to build possible basic matrices B. For this reason, we need numerical methods that are capable to solve LP problems more efficiently. In this chapter, some of the most famous algorithms for the solution of LP problems are described: simplex methods and interior-point methods.

2.1 The simplex method

The *simplex method*[1] was invented by George B. Dantzig in 1946; see, e.g., [3, 9, 10]. During 1946, G. B. Dantzig was working for the US Army Air Force when his colleagues challenged him to mechanize the planning process to distract him from taking another job. Dantzig himself explain it in [4]:

> In 1946 I was the Mathematical Advisor to the U.S. Air Force Comptroller. I had just formally completed my Ph.D. and was looking for an academic position. In order to entice me into not taking another job, colleagues challenged me to see what could be done to mechanize the planning process. I was asked to find a way to more rapidly compute a time-staged deployment, training and logistical supply program.

[1] The name of the algorithm is derived from the concept of a simplex and was suggested by T. S. Motzkin [9, Comment 2.2], even if simplices are not actually used in the method.

Figure 2.1: George Bernard Dantzig (Born: November 8, 1914 Portland, Oregon - Died: May 13, 2005 Stanford, California.

From this challenge, Dantzig created the probably most famous numerical algorithm for the solution of LP problems. For his work, he is sometimes considered the "father of linear programming" [1]. After Dantzig's work, several variants of the simplex method have been developed. The one that we consider in this section is sometimes known as the *revised simplex method*.

The main idea of the simplex method is that all its iterates are basic feasible points for (1.1) and therefore vertexes of the polyhedron $K = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$. The simplex method "explores" the set of vertexes of K while minimizing the cost function $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. More precisely: the simplex method creates a sequence $\{\mathbf{x}^k\}_{k\in\mathbb{N}}$ of vectors $\mathbf{x}^k \in \mathbb{R}^n$ that are basic feasible points for (1.1) and such that $f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k)$ for any k. Indeed the goal is to obtain strict inequality $f(\mathbf{x}^{k+1}) < f(\mathbf{x}^k)$ in most of the iterations. In other words, the simplex method generates a path on the polyhedron K while minimizing f along this path. Since each iterate $\mathbf{x}^k$ is a basic feasible point, it is natural to see that the simplex method works on the basis S: at each iteration an index is removed from S and another one is added to it in order to create a new basic feasible point $\mathbf{x}^{k+1}$ such that $f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k)$.

Let us describe the method in more details. Consider a basis S (see Definition 7), its complement

$$S^c := \{1, \ldots, n\} \setminus S$$

and the matrices

$$B := [A_j]_{j\in S} \quad \text{and} \quad N := [A_j]_{j\in S^c},$$

where A_j is the j^{th} column of the matrix A. Now, recalling the KKT system (1.10)-(1.15) and the Lagrange multiplier $\mathbf{s}$, we partition the vectors $\mathbf{x}$, $\mathbf{s}$ and $\mathbf{c}$

as

$$\begin{aligned} \mathbf{x}_B &:= [(\mathbf{x})_j]_{j\in S} & \mathbf{x}_N &:= [(\mathbf{x})_j]_{j\in S^c}, \\ \mathbf{s}_B &:= [(\mathbf{s})_j]_{j\in S} & \mathbf{s}_N &:= [(\mathbf{s})_j]_{j\in S^c}, \\ \mathbf{c}_B &:= [(\mathbf{c})_j]_{j\in S} & \mathbf{c}_N &:= [(\mathbf{c})_j]_{j\in S^c}. \end{aligned}$$

Using the KKT condition (1.11), we write

$$A\mathbf{x} = B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b},$$

which allows us to compute $\mathbf{x}_B$ and $\mathbf{x}_N$ as

$$\mathbf{x}_B = B^{-1}\mathbf{b}, \quad \mathbf{x}_N = 0. \tag{2.1}$$

Since S is a basis, B is invertible and $\mathbf{x}_B \geq 0$. Therefore, the vectors $\mathbf{x}_B$ and $\mathbf{x}_N$ satisfy (1.11) and (1.12). Now, we choose $\mathbf{s}$ to satisfy (1.14) by setting

$$\mathbf{s}_B = 0, \tag{2.2}$$

which clearly implies that $\mathbf{x}^\top\mathbf{s} = \mathbf{x}_B^\top\mathbf{s}_B + \mathbf{x}_N^\top\mathbf{s}_N = 0$. The remaining vectors $\boldsymbol{\lambda}$ and $\mathbf{s}_N$ can be found using (1.10):

$$A^\top\boldsymbol{\lambda} + \mathbf{s} = \mathbf{c} \;\Leftrightarrow\; \begin{cases} B^\top\boldsymbol{\lambda} & = \mathbf{c}_B, \\ N^\top\boldsymbol{\lambda} + \mathbf{s}_N & = \mathbf{c}_N, \end{cases}$$

where we used that $\mathbf{s}_B = 0$. Since B is invertible we can compute

$$\boldsymbol{\lambda} = B^{-\top}\mathbf{c}_B \tag{2.3}$$

and

$$\mathbf{s}_N = \mathbf{c}_N - N^\top\boldsymbol{\lambda}. \tag{2.4}$$

The vectors $\mathbf{x}$, $\mathbf{s}$ and $\boldsymbol{\lambda}$ constructed using (2.1)-(2.4) satisfy (1.10), (1.11), (1.12) and (1.14), but not necessarily the non-negativity condition (1.13), that is $\mathbf{s} \geq 0$. Indeed, if $\mathbf{s}_N \geq 0$, then $\mathbf{s} \geq 0$. This means that we have found an optimal triple $(\mathbf{x}, \mathbf{s}, \boldsymbol{\lambda})$ and we can terminate our search. If this is not the case, we must update the basis S by choosing an *entering index* and a *leaving index*. Assume now that S is non-degenerate (see Definition 8). Entering and leaving index are chosen as follows. Since $\mathbf{s}_N$ is not non-negative there exists an index $q \in S^c$ such that $(\mathbf{s})_q < 0$. We wish to add this index to S and at the same time find a leaving index $p \in S$ such that the new basis $S^+ := (S \cup \{q\}) \setminus \{p\}$ corresponds to a new basic feasible point $\mathbf{x}^+$ satisfying $f(\mathbf{x}^+) \leq f(\mathbf{x})$. To do so, we consider the following partitioning of $\mathbf{x}^+$

$$\mathbf{x}_B^+ = [(\mathbf{x}^+)_j]_{j\in S}, \quad \mathbf{x}_{\widetilde{N}}^+ = [(\mathbf{x}^+)_j]_{j\in S^c\setminus\{q\}} \quad \text{and} \quad (\mathbf{x}^+)_q,$$

and the matrix $\widetilde{N} = [A_j]_{j\in S^c\setminus\{q\}}$. Now, we allow the q^{th} component of $\mathbf{x}^+$, that is $(\mathbf{x}^+)_q$, to increase from zero, while the components $\mathbf{x}_{\widetilde{N}}$ are fixed to zero. Since we wish that $\mathbf{x}^+$ remains feasible while increasing $(\mathbf{x}^+)_q$, we compute

$$\mathbf{b} = A\mathbf{x}^+ = B\mathbf{x}_B^+ + A_q(\mathbf{x}^+)_q + \widetilde{N}\mathbf{x}_{\widetilde{N}}^+ = B\mathbf{x}_B^+ + A_q(\mathbf{x}^+)_q.$$

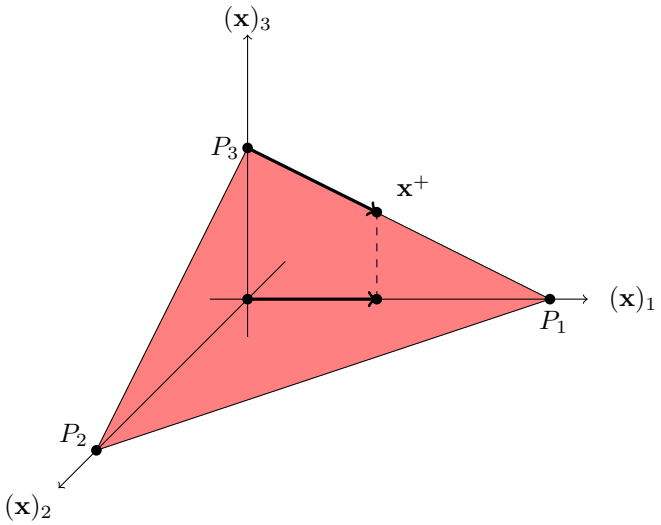

Figure 2.2: Polyhedron K of Example 11. The points $P_1 = [1\,0\,0]$, $P_2 = [0\,1\,0]$ and $P_3 = [0\,0\,\frac{1}{2}]$ are the three vertexes of K. The point $\mathbf{x}$ coincides with the vertex P_1. If one increases from zero the component $(\mathbf{x}^+)_1$, then $\mathbf{x}^+$ moves along the edge of K that is the segment line connecting P_1 with P_3.

Using this equality and $A\mathbf{x} = \mathbf{b}$ with $\mathbf{x}_N = 0$ ($\mathbf{x}$ is feasible), we obtain

$$B\mathbf{x}_B^+ + A_q(\mathbf{x}^+)_q = A\mathbf{x}^+ = \mathbf{b} = A\mathbf{x} = B\mathbf{x}_B.$$

By multiplying left- and right-hand sides by B, one gets

$$\mathbf{x}_B^+ = \mathbf{x}_B - (\mathbf{x}^+)_q B^{-1} A_q. \tag{2.5}$$

Geometrically, this means that we are increasing $(\mathbf{x}^+)_q$ and moving $\mathbf{x}^+$ along an edge of the polyhedron K toward another vertex (basic feasible point). An example of such situation is explained in the following example.

Example 11 (Moving a point from a vertex along an edge)**.** *Consider the polyhedron* $K = \{\mathbf{x} \in \mathbb{R}^3 : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$, *where* $A = [1\,1\,2]$ *and* $\mathbf{b} = 1$. *This polyhedron has three vertexes:* $P_1 = [1\,0\,0]$, $P_2 = [0\,1\,0]$ *and* $P_3 = [0\,0\,\frac{1}{2}]$; *see Figure 2.2. Consider the basis* $S = \{3\}$, *which corresponds to the basic feasible point* $\mathbf{x} = P_3$ *having component* $\mathbf{x}_B = (\mathbf{x})_3 = 1/2$, *and to the basis matrix* $B = A_3 = 2$ *(clearly invertible with* $B^{-1} = 1/2$*). Let us choose* $q = 1$ *as entering index. We have that* $A_q = A_1 = 1$ *and using* (2.5) *that*

$$(\mathbf{x}^+)_3 = \mathbf{x}_B^+ = \mathbf{x}_B - (\mathbf{x}^+)_q B^{-1} A_q = (\mathbf{x})_3 - (\mathbf{x}^+)_1 B^{-1} A_1 = 1/2 - (\mathbf{x}^+)_1/2.$$

Hence

$$\mathbf{x}^+ = \begin{bmatrix} (\mathbf{x}^+)_1 \\ 0 \\ 1/2 - (\mathbf{x}^+)_1/2 \end{bmatrix}.$$

As $(\mathbf{x}^+)_1$ *increases from zero, the point* $\mathbf{x}^+$ *moves along the edge* $\overline{P_1P_3}$ *toward the vertex* P_1*. Clearly, the maximum possible increase of* $(\mathbf{x}^+)_1$ *is equal to* 1*, because for* $(\mathbf{x}^+)_1 > 1$ *the component* $(\mathbf{x}^+)_3$ *is negative, hence not feasible.*

Let us now compute

$$\begin{aligned} f(\mathbf{x}^+) = \mathbf{c}^\top \mathbf{x}^+ &= \mathbf{c}_B^\top \mathbf{x}_B^+ + (\mathbf{c})_q (\mathbf{x}^+)_q \\ &= \mathbf{c}_B^\top \mathbf{x}_B - (\mathbf{x}^+)_q \mathbf{c}_B^\top B^{-1} A_q + (\mathbf{c})_q (\mathbf{x}^+)_q \\ &= \mathbf{c}_B^\top \mathbf{x}_B - (\mathbf{x}^+)_q \boldsymbol{\lambda}^\top A_q + (\mathbf{c})_q (\mathbf{x}^+)_q \\ &= \mathbf{c}_B^\top \mathbf{x}_B - (\mathbf{x}^+)_q ((\mathbf{c})_q - (\mathbf{s})_q) + (\mathbf{c})_q (\mathbf{x}^+)_q \\ &= \mathbf{c}_B^\top \mathbf{x}_B + (\mathbf{x}^+)_q (\mathbf{s})_q \\ &= \mathbf{c}^\top \mathbf{x} + (\mathbf{x}^+)_q (\mathbf{s})_q = f(\mathbf{x}) + (\mathbf{x}^+)_q (\mathbf{s})_q, \end{aligned}$$

where we used (2.5) to get the second line, (2.3) to get the third line, (2.4) to get the fourth line, and $\mathbf{x}_N = 0$ to get the sixth line. The previous equality gives us

$$f(\mathbf{x}^+) = f(\mathbf{x}) + (\mathbf{x}^+)_q (\mathbf{s})_q. \tag{2.6}$$

Since $(\mathbf{s})_q < 0$ and $(\mathbf{x}^+)_q \geq 0$, increasing $(\mathbf{x}^+)_q \geq 0$ means that the function $f(\mathbf{x}^+)$ decreases. Indeed, increasing $(\mathbf{x}^+)_q$ by keeping $\mathbf{x}_{\widetilde{N}}^+ = 0$ implies that some of the components of $\mathbf{x}_B^+$ decrease. How much should we increase $(\mathbf{x}^+)_q$? Clearly, Equation (2.6) says that the more $(\mathbf{x}^+)_q$ increases the more $f(\mathbf{x}^+)$ decreases. However, we want to keep $\mathbf{x}^+$ feasible. For this reason, we can increase $(\mathbf{x}^+)_q$ till one component of $\mathbf{x}_B^+$ becomes zero. A further increase of $(\mathbf{x}^+)_q$ would make $\mathbf{x}^+$ infeasible. In other words, we increase $(\mathbf{x}^+)_q$ and move $\mathbf{x}^+$ along an edge of K till another vertex is reached; see also Example 11 and Figure 2.2. Notice that this procedure is possible because we assumed S to be non-degenerate, otherwise some components of $\mathbf{x}_B$ could be zero ($\mathbf{x}$ would be a degenerate basic feasible point) and no increase from zero of $(\mathbf{x}^+)_q$ would be possible.

Now, if we denote by p the index of the component of $\mathbf{x}_B^+$ that became zero, then p is the leaving index. We can now update the basis S by adding q and removing p, and then repeat the procedure. The process of adding and removing indexes is sometimes called *pivoting*.

How to chose the entering index q if there are more components of $\mathbf{s}_N$ with negative value? There are different heuristic choices described in the literature; see, e.g., [10, 3]. For example, one can choose the index q corresponding to the most negative component of $\mathbf{s}_N$. Another possible choice is the minimal index i of $(\mathbf{x})_i$ that corresponds to a negative component $(\mathbf{s})_i$.

How to choose the leaving index p? Recalling (2.5) and defining $\mathbf{d} := B^{-1} A_q$, we obtain

$$\mathbf{x}_B^+ = \mathbf{x}_B - (\mathbf{x}^+)_q \mathbf{d},$$

which allows us to write that

$$\frac{(\mathbf{x}_B^+)_i}{(\mathbf{d})_i} = \frac{(\mathbf{x}_B)_i}{(\mathbf{d})_i} - (\mathbf{x}^+)_q \quad \text{for } i \text{ such that } (\mathbf{d})_i > 0.$$

Now, we choose the index p as

$$p = \arg\min_{i:(\mathbf{d})_i > 0} \frac{(\mathbf{x}_B)_i}{(\mathbf{d})_i}.$$

This choice corresponds to the maximum value of $(\mathbf{x}^+)_q$ that still guarantees the feasibility of $\mathbf{x}^+$. Notice that if $\mathbf{d} \leq 0$, then we have

$$\mathbf{x}^+ = \mathbf{x} - (\mathbf{x}^+)_q \mathbf{d} \geq 0 \quad \forall (\mathbf{x}^+)_q \geq 0.$$

Hence, we can arbitrarily increase $(\mathbf{x}^+)_q$ (hence decrease f) and move $\mathbf{x}^+$ along an edge of the polyhedron without encountering any new vertex! This means that the problem is unbounded (see Definition 1). For this reason, if $\mathbf{d} \leq 0$, then we stop the algorithm.

We are now ready to state the simplex algorithm:

Algorithm 1 (Simplex Algorithm)

Require: The matrix A and the vectors $\mathbf{b}$ and $\mathbf{c}$. A basis S and the corresponding basis matrix B. The complement S^c and the corresponding matrix N. A maximum number of iterations $k_{\max}$.

1: **for** $k = 1{:}k_{\max}$ **do**
2: Solve $B\mathbf{x}_B = \mathbf{b}$ and get $\mathbf{x}_B$.
3: Set $\mathbf{x}_N = 0$.
4: Solve $B^\top \boldsymbol{\lambda} = \mathbf{c}_B$ and get $\boldsymbol{\lambda}$.
5: Compute $\mathbf{s}_N = \mathbf{c}_N - N^\top \boldsymbol{\lambda}$.
6: If $\mathbf{s}_N \geq 0$, then break (optimal point found).
7: Select entering index $q \in S^c$ such that $(\mathbf{s})_q < 0$.
8: Solve $B\mathbf{d} = A_q$ and get $\mathbf{d}$.
9: If $\mathbf{d} \leq 0$, then break (the problem is unbounded).
10: Select leaving index p as $p = \arg\min_{i:(\mathbf{d})_i > 0} \frac{(\mathbf{x}_B)_i}{(\mathbf{d})_i}$.
11: Update S, S^c, B and N.
12: **end for**

One can prove that the simplex method converges.

Theorem 4 (Convergence of the simplex method)**.** *If the LP problem is bounded and non-degenerate, then the simplex method terminates at a basic feasible point.*

Proof. See [10, Theorem 13.4]. □

A MATLAB implementation of the simplex algorithm is the following.

```
function [x,x_iter,S] = simplex(A,b,c,indB,maxit)
ind = 1:length(c); % construct initial basis matrix
indN = ind;
B = A(:,indB);
indN(indB) = [];
N = A(:,indN);
x_iter=[];
```

```
for it=1:maxit
    cB=c(indB); cN=c(indN); % extract the two components of c
    [L,U,P]=LUdec(B); % compute LU-factorization of B
    xB=U\(L\(P*b)); % construct the vector xB
    x_iter=[x_iter,zeros(length(c),1)];
    x_iter(indB,it)=xB;
    f = c'*x_iter(:,it);
    fprintf('it= %3.0d | f= %5.3e \n',it, f);
    lam = P'*(L'\(U'\cB)); % compute the Lagrange multiplier
    sN = cN-N'*lam; % pricing step
    if sN >=0 % check optimality
        break;
    end
    msN=min(sN); % pivoting: select entering index
    enter=min(indN(sN==msN));
    Aq=A(:,enter); % compute d
    d=U\(L\(P*Aq));
    if d <= 0  % check unboundedness
        disp('Warning: the problem is unbounded!');
        break;
    end
    ratio=(xB./d).*(d>0)+1e8*ones(size(xB)).*(d<=0); % pivoting: select exiting index
    xq=min(ratio);
    exitt=min(indB(ratio==xq));  % the "min" is used in case the basis is degenerate
    indB(indB==exitt)=enter; % update the basis
    indN(indN==enter)=exitt;
    B=A(:,indB); N=A(:,indN);
end
x=x_iter(:,end);
S=indB;
end
```

Here, the function `[L,U,P]=LUdec(B)` is given in Section 2.3. To test this algorithm, one can run the following MATLAB script.

```
% Example Nocedal-Wright (page 371)
A = [ 1 1 1 0 ; 2 0.5 0 1 ];
b = [ 5 ; 8 ];
c = [ -4 ; -2 ; 0 ; 0 ];
n = length(c);
ind = 1:n; indN = ind;
indB = [3,4]; % initial basis
[x,x_iter,S] = simplex(A,b,c,indB,10); % simplex method ...
% plot ...
polyh=Polyhedron('A',-eye(n),'b',zeros(n,1),'Ae',A,'be',b);
figure();
[X,Y] = meshgrid([0:.25:5]);
Z = c(1)*X+c(2)*Y;
contour(X,Y,Z,30); colorbar;
hold on;
plot(polyh.projection(1:3));
```

```
view(70,25);
xlabel('x_1'); ylabel('x_2'); zlabel('x_3');
hold on
plot3(x(1),x(2),x(3),'*k');
for j=1:size(x_iter,2)
    plot3(x_iter(1,j),x_iter(2,j),x_iter(3,j),'ob');
    txt = ['it=' num2str(j)];
    text(x_iter(1,j)+0.2,x_iter(2,j)+0.2,x_iter(3,j)+0.2,txt,'FontSize',14);
end
figure();
contour(X,Y,Z,30); colorbar;
hold on;
plot(polyh.projection([1,2,4]));
view(50,50);
xlabel('x_1'); ylabel('x_2'); zlabel('x_4');
hold on;
plot3(x(1),x(2),x(4),'*k');
for j=1:size(x_iter,2)
    plot3(x_iter(1,j),x_iter(2,j),x_iter(4,j),'ob');
    txt = ['it=' num2str(j)];
    text(x_iter(1,j)+0.2,x_iter(2,j)+0.2,x_iter(4,j)+0.2,txt,'FontSize',14);
end
figure();
plot(polyh.projection([1,2]));
grid on;
xlabel('x_1'); ylabel('x_2');
contour(X,Y,Z,30); colorbar;
hold on;
plot(x(1),x(2),'*k');
for j=1:size(x_iter,2)
    plot(x_iter(1,j),x_iter(2,j),'ob');
    txt = ['it=' num2str(j)];
    text(x_iter(1,j)-0.15,x_iter(2,j)+0.25,txt,'FontSize',14);
end
```

This script solves a simple LP problem in 3 iterations and plots the pictures shown in Figure 2.3. These pictures show the three iterations performed by the simplex algorithm. Each iteration is a vertex of the polyhedron (represented in red) and the solution is marked with a '$*$' marker. Each figure shows the projection of the 5-dimensional geometry onto different subspaces. The colored lines are the isolines of the function $f(\mathbf{x})$. It is clear that the simplex method generates a sequence of vertexes (basic feasible points) that converges to the solution to the problem.

An important question arises: how should we choose an initial basis/basic feasible point to start the simplex algorithm? The answer is in Section 2.2.

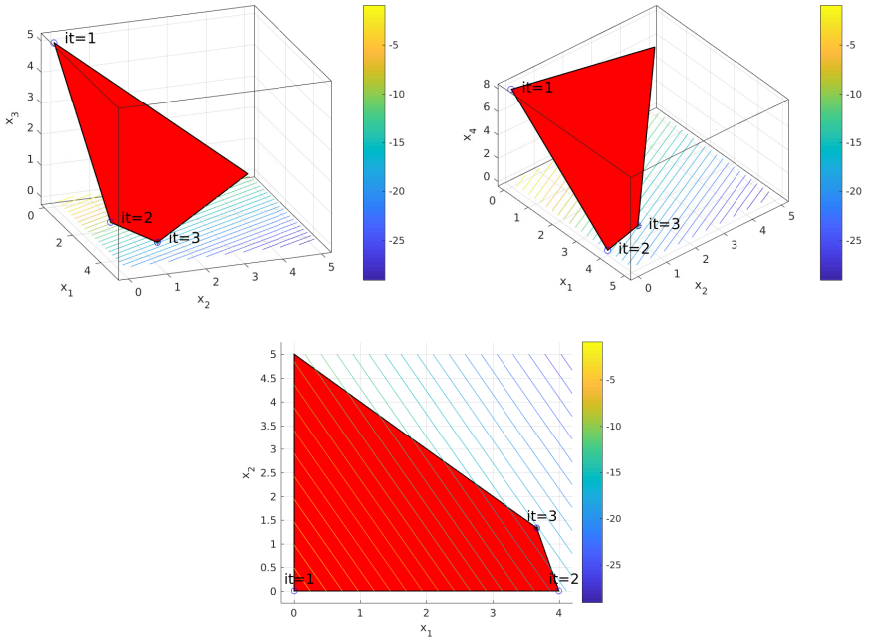

Figure 2.3: Example of iterations of the simplex algorithm. At each iteration the algorithm visits a vertex of the polyhedron. The minimum is computed at iteration 3. The three figures represent the projections of the 5-dimensional geometry onto different subspaces. Top left: projection onto the coordinates $(\mathbf{x})_1$, $(\mathbf{x})_2$ and $(\mathbf{x})_3$. Top right: projection onto the coordinates $(\mathbf{x})_1$, $(\mathbf{x})_2$ and $(\mathbf{x})_4$. Bottom: projection onto the coordinates $(\mathbf{x})_1$ and $(\mathbf{x})_2$.

2.2 The two-phase strategy

The problem of finding an initial point and a basis to start the simplex method is not trivial and can be quite expensive. One of the most used strategies to tackle this problem is the so-called *two-phase strategy* that we describe in this section.

The two-phase strategy solves the LP problem (1.1) in two phases. In Phase 1, an auxiliary LP problem is set up based on the data of (1.1) and solved using the simplex algorithm of Section 2.1. The auxiliary problem is designed in a way that an initial basis is trivial to find and the solution provides a basic feasible point for (1.1) (or for an equivalent problem). In Phase 2, a second LP problem similar to (1.1) is solved using the solution to the Phase-1 problem as initial guess for a possibly modified simplex algorithm. A solution to the Phase-2 problem will coincide with a solution to the original problem (1.1).

2.2.1 Phase 1

We introduce the auxiliary problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n,\, \mathbf{z}\in\mathbb{R}^m} \quad & \mathbf{e}^\top \mathbf{z} \\ \text{s.t.} \quad & A\mathbf{x} + E\mathbf{z} = \mathbf{b} \\ & \mathbf{x} \geq 0 \\ & \mathbf{z} \geq 0, \end{aligned} \tag{2.7}$$

where $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are the same as in (1.1), $\mathbf{z} \in \mathbb{R}^m$ is an artificial vector of variables, $\mathbf{e} = [1 \ \dots \ 1]^\top \in \mathbb{R}^m$, and $E \in \mathbb{R}^{m\times m}$ is a diagonal matrix whose diagonal entries are

$$E_{jj} = \begin{cases} +1 & \text{if } (\mathbf{b})_j \geq 0, \\ -1 & \text{if } (\mathbf{b})_j < 0, \end{cases}$$

for $j = 1, \dots, m$. Notice that the matrix E is invertible and the point $\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} \in \mathbb{R}^{n+m}$ with $\mathbf{x} = 0$ and $(\mathbf{z})_j = |(\mathbf{b})_j|$, for $j = 1, \dots, m$, is a basic feasible point for (2.7). This point corresponds to the basis $S = \{n+1, n+2, \dots, n+m\} \subset \{1, 2, \dots, n+m\}$, which contains only indexes associated to the artificial variable $\mathbf{z}$. The corresponding basis matrix is $B = E$. Using the basis S one can initialize the simplex algorithm of Section 2.1 and solve (2.7).

Notice that since $\mathbf{e} \geq 0$ and $\mathbf{x} \geq 0$, any global solution to (2.7) must correspond to $f(\mathbf{x}) \geq 0$ (and clearly satisfy the KKT system (1.10)-(1.15)). We have the following result.

Theorem 5 (Solutions of the auxiliary problem)**.** *The auxiliary problem* (2.7) *has a global solution* $(\mathbf{x}^\star, \mathbf{z}^\star)$ *corresponding to an optimal objective value of zero* ($\mathbf{e}^\top \mathbf{z}^\star = 0$) *if and only if the original problem* (1.1) *is feasible.*

Proof. Assume that there is a pair $(\widehat{\mathbf{x}}, \widehat{\mathbf{z}})$ that is feasible for (2.7) and such that $\mathbf{e}^\top \widehat{\mathbf{z}} = 0$. We must have that $\widehat{\mathbf{z}} = 0$ (since $\mathbf{e}^\top \widehat{\mathbf{z}} = 0$) and $\widehat{\mathbf{x}} \geq 0$, and hence $\mathbf{b} = A\widehat{\mathbf{x}} + E\widehat{\mathbf{z}} = A\widehat{\mathbf{x}}$. Therefore, $\widehat{\mathbf{x}}$ is a feasible point for (1.1).

Assume that $\widehat{\mathbf{x}}$ is a feasible point for (1.1). The pair $(\widehat{\mathbf{x}}, 0)$ is feasible for (2.7) and optimal with an objective value of zero. □

Theorem 5 allows us to study the solution that we obtain by solving (2.7) using the simplex method. The simplex method finds a solution $((\mathbf{x}^\star, \mathbf{z}^\star), \boldsymbol{\lambda}^\star, \mathbf{s}^\star)$ to the KKT system (1.10)-(1.15). Theorem 3 guarantees that $(\mathbf{x}^\star, \mathbf{z}^\star)$ is a global solution to (2.7). Therefore, this solution must correspond to an optimal objective value of zero, which is only possible if $\mathbf{z}^\star = 0$. This argument says that, if the simplex method finds a solution to (2.7) with a non-zero $\mathbf{z}$, then the global solutions to (2.7) correspond to a non-zero objective value. Hence, Theorem 5 guarantees that in this case the original problem (1.1) is not feasible! If this unfavorable situation occurs, then we must stop our numerical procedure, because

our original problem is not well defined. In case this unfavorable situation does not occur, we have obtained an initialization for the Phase 2.

Let us discuss a final issue before describing the Phase 2. Assume that the simplex method found a solution to (2.7) corresponding to a zero objective value and hence to a pair $\widehat{\mathbf{y}} = (\widehat{\mathbf{x}}, \widehat{\mathbf{z}})$ with $\widehat{\mathbf{z}} = 0$. This optimal solution is associated to a basis S (recall that the simplex algorithm actually iterates on the basis S). Even if $\widehat{\mathbf{z}} = 0$, the optimal basis can still contain some indexes that correspond to the variable $\mathbf{z}$! This is shown in Example 12.

Example 12 (Degenerate Phase-1 problem)**.** *Consider problem* (1.1) *with*

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix},$$

and some vector $\mathbf{c}$*. The corresponding Phase-1 problem* (2.7) *is obtained by introducing the artificial variable* $\mathbf{z} \in \mathbb{R}^2$ *and the matrix*

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

In this case, if we introduce the variable $\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix}$*, the matrix* $\widetilde{A} = [A\, E]$ *and the vector* $\widetilde{\mathbf{c}} = \begin{bmatrix} 0 \\ \mathbf{e} \end{bmatrix} \in \mathbb{R}^{n+m}$*, the auxiliary problem* (2.7) *is equivalent to*

$$\begin{aligned} \min_{\mathbf{y} \in \mathbb{R}^{n+m}} \quad & \widetilde{\mathbf{c}}^\top \mathbf{y} \\ \text{s.t.} \quad & \widetilde{A}\mathbf{y} = \mathbf{b} \\ & \mathbf{y} \geq 0. \end{aligned} \tag{2.8}$$

Notice that this problem is degenerate. To see the degeneracy, consider the subset $S = \{3, 5\}$ *of the index set* $\{1, 2, 3, 4, 5\}$*. The matrix* $B = \begin{bmatrix} \widetilde{A}_3\, \widetilde{A}_5 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}$ *is invertible and allows us to compute* $\mathbf{y}_B = B^{-1}\mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$*. Therefore, the vector* $\mathbf{y} = [0\,0\,1\,0\,0]^\top$ *satisfies* $\widetilde{A}\mathbf{y} = \mathbf{b}$ *and* $\mathbf{y} \geq 0$*. This shows that* $\mathbf{y}$ *is a basic feasible point and* S *a degenerate basis. Notice that* S *contains the index* $j = 5$*, which corresponds to an index of the artificial variable* $\mathbf{z}$*:* $(\mathbf{y})_5 = (\mathbf{z})_2$*.*

One can solve (2.8) *using the simplex algorithm of Section 2.1 with the initialization*

$$S = \{4, 5\}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}.$$

To do that, one can run the following MATLAB *script.*

```
% example of degenerate Phase-1 problem
A = [ 1 1 2 ; 0 1 3 ];
b = [ 2 ; 3 ];
```

```
E = [ 1 0 ; 0 1 ];
At = [ A , E ];
ct = [ 0; 0; 0; 1; 1 ];
indB = [4,5]; % initial basis
[x,x_iter,S] = simplex(At,b,ct,indB,10);
S
```

which produces the following result:

```
it=   1 | f= 5.000e+00
it=   2 | f= -1.665e-16

S =

     3     5
```

These results mean that the simplex algorithm found a global solution in two iterations. The optimal objective value is zero (hence $\mathbf{z} = 0$*) and corresponds to the optimal basis* $S = \{3, 5\}$*. Clearly, even if the optimal solution satisfies* $\mathbf{z} = 0$*, the optimal basis contains the index* $j = 5$*, which corresponds to* $(\mathbf{z})_2$*.*

2.2.2 Phase 2

The Phase 2 consists in solving a second auxiliary problem, that is

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{z}\in\mathbb{R}^m} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} + \mathbf{z} = \mathbf{b} \\ & \mathbf{x} \geq 0 \\ & 0 \leq \mathbf{z} \leq 0. \end{aligned} \tag{2.9}$$

Notice that a solution to (2.9) must have $\mathbf{z} = 0$ and that (2.9) is equivalent to (1.1). Clearly, a solution to the Phase-1 problem (2.7) is a basic feasible point for (2.9). This Phase-1 solution is used to initialize a simplex algorithm for the solution of (2.9). Due to the two-side constraint $0 \leq \mathbf{z} \leq 0$, one can not use directly the algorithm described in Section 2.1. Luckily, it is possible to slightly modify the simplex method of Section 2.1 to obtain an algorithm that is capable to take into a account general *two-side constraint* $\mathbf{z}_\ell \leq \mathbf{z} \leq \mathbf{z}_u$, where $\mathbf{z}_\ell$ and $\mathbf{z}_u$ are lower- and upper-bound vectors, and solve (2.9); see, e.g., [2, Section 5.2].

What is the reason for the "weird" constraint $0 \leq \mathbf{z} \leq 0$? At a first look this constraint seems meaningless. If the solution to the Phase-1 problem is a basic feasible point with basis S that contains indexes corresponding only to components of the $\mathbf{x}$ variable, then the constraint $0 \leq \mathbf{z} \leq 0$ is meaningless and one can remove the variable $\mathbf{z}$ from (2.9), which becomes then equal to (1.1). However, if the solution to the Phase-1 problem is a basic feasible point with basis S that contains at least one index corresponding to the artificial variable $\mathbf{z}$ (see Example 12), then this artificial variable must be retained to use the Phase-1 solution as initial guess for Phase 2; see, e.g., [10] for further details.

2.3 Solving linear systems in the simplex algorithm

Each iteration of the simplex algorithm requires the solution of three linear systems:

$$B\mathbf{x}_B = \mathbf{b}, \quad B^\top \boldsymbol{\lambda} = \mathbf{c}_B \quad \text{and} \quad B\mathbf{d} = A_q.$$

How should one solve these problems? There are many different algorithms for the efficient solution of linear systems; see, e.g., [5] and references therein. In the simplex algorithm one computes at the first iteration the LU-decomposition of the matrix B and use it to solve the three above linear systems. Once the first iteration is completed, the basis matrix B is updated and the column that corresponds to the leaving index is replaced by the column of A corresponding to the entering index. Therefore, the new basis matrix B^+ differs from the old one by only one column. This allows one to perform only an update of the LU-decomposition, which must not to be entirely recomputed at each iteration; see [10] and references therein. To better understand this procedure, let us first recall the LU-decomposition.

2.3.1 LU-decomposition

Consider a linear system $B\mathbf{x}_B = \mathbf{b}$ and assume that the *LU-decomposition* $B = LU$ is known, where L is a lower-triangular matrix and U is an upper-triangular matrix. Using this factorization, one can compute the solution $\mathbf{x}_B$ by first solving the triangular system $L\mathbf{y} = \mathbf{b}$ by forward substitution, and then solving the triangular system $U\mathbf{x}_B = \mathbf{y}$ by backward substitution. How to compute the LU-decomposition?

Let us consider a simple example of a 4×4 matrix B:

$$B = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix},$$

where the symbol "$\times$" indicates a possibly non-zero entry. The idea is to find some matrices $L_j \in \mathbb{R}^{4\times 4}$ that when multiplied with B are capable to cancel the elements below the diagonal of B. To be more precise, we wish to find three matrices L_1, L_2 and L_3 in $\mathbb{R}^{4\times 4}$ such that

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{B^{(0)}:=B} \xrightarrow{L_1} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{B^{(1)}:=L_1B^{(0)}} \xrightarrow{L_2} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}}_{B^{(2)}:=L_2B^{(1)}} \xrightarrow{L_3} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}}_{B^{(3)}:=L_3B^{(2)}}.$$

Now, we set $U := B^{(3)}$ and notice that

$$U = L_3L_2L_1B \tag{2.10}$$

is an upper-triangular matrix. Which are the matrices L_j that allow us to perform the above transformation? In $\mathbb{R}^{m\times m}$ these matrices are given by

$$L_j = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & -\ell_{j+1,j} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -\ell_{m,j} & & & 1 \end{bmatrix}, \tag{2.11}$$

which is an identity matrix up to the j^{th} column, where the entries $\ell_{k,j}$ are given by

$$\ell_{k,j} = \frac{B_{k,j}^{(j-1)}}{B_{j,j}^{(j-1)}},$$

for $k = j+1, \ldots, m$ (assuming that $B_{j,j}^{(j-1)} \neq 0$). For example, the L_2 matrix in our 4×4 example is

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\ell_{3,2} & 1 & 0 \\ 0 & -\ell_{4,2} & 0 & 1 \end{bmatrix}.$$

The matrices L_j have the following important properties.

Theorem 6 (Properties of the matrices L_j)**.** *For $j = 1, \ldots, m-1$, it holds that*

$$L_j^{-1} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & \ell_{j+1,j} & 1 & \\ & & & \vdots & & \ddots \\ & & & \ell_{m,j} & & 1 \end{bmatrix}, \quad L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1} = \begin{bmatrix} 1 & & & \\ \ell_{2,1} & 1 & & \\ \vdots & \ddots & \ddots & \\ \ell_{m,1} & \cdots & \ell_{m,m-1} & 1 \end{bmatrix}.$$

Proof. See, e.g., [11]. □

This theorem says that the inverses L_j^{-1} are very easy to compute and that the product $L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1}$ is a lower-triangular matrix. If we use this result for our example, define $L = L_1^{-1}L_2^{-1}L_3^{-1}$, and recall (2.10), we obtain

$$L_3L_2L_1B = U \Leftrightarrow B = L_1^{-1}L_2^{-1}L_3^{-1}U = LU,$$

which is the LU-decomposition of the matrix B. This result can be generalized to get an LU-decomposition of a matrix $B \in \mathbb{R}^{m\times m}$:

$$B = L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1}U = LU.$$

The existence of an LU-decomposition of a matrix B (with invertible factors L and U) is guaranteed if some fair assumptions are satisfied; see, e.g., [6, Corollary 3.4.5].

In the previous procedure, we tacitly assumed that $B_{j,j}^{(j-1)} \neq 0$. Unfortunately, this condition is not always satisfied and if it happens that $B_{j,j}^{(j-1)} = 0$, then we must introduce extra steps in the above process. The idea is to permute two rows of the matrix $B^{(j-1)}$ before multiplying it by a matrix L_j:

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{B^{(1)}} \xrightarrow{P_1} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{\widetilde{B}^{(1)}:=P_1B^{(1)}} \xrightarrow{L_2} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}}_{B^{(2)}:=L_2\widetilde{B}^{(1)}},$$

where P_1 is the permutation matrix that swaps second and third rows. The idea of this permutation is to obtain a matrix $\widetilde{B}^{j-1}$ whose diagonal element $\widetilde{B}_{j,j}^{j-1}$ is non-zero. This permutation process is also known as *pivoting*. Indeed, the rows to be permuted can be chosen in several efficient ways; see, e.g., [5, 11]. If the previous process is repeated till an upper-triangular matrix U is obtained, one gets

$$U = L_{n-1}P_{n-1}L_{n-2}P_{n-2}\cdots L_1P_1B.$$

A direct calculation shows that

$$U = \widetilde{L}_{n-1}\widetilde{L}_{n-2}\cdots\widetilde{L}_1P_{n-1}P_{n-1}\cdots P_1B,$$

where

$$\begin{aligned} \widetilde{L}_{n-1} &= L_{n-1}, \\ \widetilde{L}_{n-2} &= P_{n-1}L_{n-2}P_{n-1}^{-1}, \\ \widetilde{L}_{n-2} &= P_{n-1}P_{n-2}L_{n-3}P_{n-2}^{-1}P_{n-1}^{-1}, \\ &\vdots \end{aligned}$$

It is possible to show that the matrices $\widetilde{L}_j$ have the same (triangular) structure of L_j and the matrix $L := (\widetilde{L}_{n-1}\widetilde{L}_{n-2}\cdots\widetilde{L}_1)^{-1}$ is lower triangular. Hence, by defining $P := P_{n-1}P_{n-1}\cdots P_1$ we obtain the decomposition

$$PB = LU.$$

A MATLAB function that allows one to compute this decomposition is the following.

```
function [L,U,P] = LUdec(A)
% LUdec  : LU-decomposition of a square matrix A: PA=LU
% Input : A - a square matrix (of size n)
% Output: L - lower-triangular matrix of size n
```

```
%         U - upper-triangular matrix of size n
%         P - permutations matrix of size n
n = size(A,1); % size of A
P = eye(n);  % initialize P
L = zeros(n); % initialize L
for k=1:n
    j=(k-1)+find(abs(A(k:n,k))==max(abs(A(k:n,k)))); % find the pivot
    if A(j,k)==0
        disp('the matrix A is singular!');
        break
    elseif j~=k
        A([j k],:)=A([k j],:); % pivot swap
        L([j k],:)=L([k j],:);
        P([j k],:)=P([k j],:);
    end
    L(k,k)=1;
    for p=k+1:n
        L(p,k)=A(p,k)/A(k,k);
        for q=k+1:n
            A(p,q)=A(p,q)-L(p,k)*A(k,q);
        end
    end
end
U = triu(A); % U is the upper triangular part of A
end
```

Clearly, if some permutations are needed and the decomposition of B is $B = P^\top LU$, then the solution to the linear system $B\mathbf{x}_B = \mathbf{b}$ is achieved by first computing $\mathbf{z} = P\mathbf{b}$ and then solving $L\mathbf{y} = \mathbf{z}$ and $U\mathbf{x}_B = \mathbf{y}$.

2.3.2 The LU-decomposition in the simplex algorithm

Each iteration of the simplex algorithm requires the solution of three linear systems characterized by the same basis matrix B. Moreover, this matrix changes from one iteration to another by only one column. This fact is essential for an efficient numerical implementation of the simplex algorithm: one computes the LU-decomposition of B only at the first iteration of the simplex procedure, and then only updates the decomposition in the subsequent iterations.

Assume that the LU-decomposition of B is known, that is we have computed the two factors L and U such that $B = LU$ (we assume for simplicity that $P = I$). Consider the updated matrix B^+ that differs from B by only one column:

$$B = [B_1 \cdots B_j \cdots B_m] \quad B^+ = [B_1 \cdots A_q \cdots B_m].$$

Here the column B_j is replaced by the column A_q. The key step is to notice that

$$U = L^{-1}B = [L^{-1}B_1 \cdots L^{-1}B_j \cdots L^{-1}B_m]$$

is an upper-triangular matrix, while the matrix

$$\widetilde{U} = L^{-1}B^{+} = [L^{-1}B_1 \cdots L^{-1}A_q \cdots L^{-1}B_m]$$

has the form

$$\widetilde{U} = \begin{bmatrix} \times & \cdots & \times & \cdots & \times \\ & \ddots & \vdots & & \vdots \\ & & \times & \cdots & \times \\ & & \vdots & \ddots & \vdots \\ & & \times & & \times \end{bmatrix},$$

which is not upper triangular only because of the j^{th} column given by $L^{-1}A_q$. However, to transform $\widetilde{U}$ into an upper-triangular matrix is easy and computationally cheap. To do so, one can perform a cyclic permutation by moving the j^{th} column of $\widetilde{U}$ to the last column position and moving the columns $j+1, j+2, \ldots, m$ one position to the left. Formally, this is possible via the action of a permutation matrix P:

$$\begin{aligned} \widetilde{U}P^{\top} &= [L^{-1}B_1, \cdots L^{-1}B_{j-1}, L^{-1}A_q, L^{-1}B_{j+1}, \cdots L^{-1}B_m]P^{\top} \\ &= [L^{-1}B_1, \cdots L^{-1}B_{j-1}, L^{-1}B_{j+1}, \cdots L^{-1}B_m, L^{-1}A_q]. \end{aligned}$$

If we apply the same permutations to the rows of this matrix, we obtain a matrix of the form

$$P\widetilde{U}P^{\top} = \begin{bmatrix} \times & \cdots & \times & \cdots & \times \\ & \ddots & \vdots & & \vdots \\ & & \times & \cdots & \times \\ & & & \ddots & \vdots \\ & & \times & \cdots & \times \end{bmatrix},$$

which is not upper triangular only because of the last row. Now, the LU-decomposition of the matrix B^{+} can be easily achieved by a matrix $\widetilde{L}$ which is lower triangular and differs form the identity only in the last row. We have

$$\widetilde{L}PL^{-1}B^{+}P^{\top} = U^{+} \Rightarrow B^{+} = LP^{\top}\widetilde{L}^{-1}U^{+}P.$$

We wish to remark that this factorization does not require the explicit calculation of the product $LP^{\top}\widetilde{L}^{-1}U^{+}P$, but the factors are used one after the other in the solution process of the linear system $B^{+}\mathbf{x}_B^{+} = \mathbf{b}$. For more details see, e.g., [10] and references therein.

2.4 Interior-point methods

The simplex method is very efficient in many practical cases. However, there are problems on which this algorithm performs very poorly. It can happen that it visits every single vertex of the polyhedron before reaching an optimal point!

An example of such pathological situations was presented by Klee and Minty in [7].

A class of more efficient methods was then developed. These are the so-called *interior-point methods*. In this chapter, we will focus on a particular subclass of interior-point strategies, the so-called *primal-dual methods*.

Interior-point methods are completely different from the simplex method. Each interior-point iteration is in general expensive to compute and can make significant progress toward a solution, while the simplex method usually requires a larger number of inexpensive iterations. Geometrically speaking, the simplex method works on the boundary of the feasible polyhedron, testing a sequence of vertexes until a minimum is found. Interior-point methods approach the boundary only in the limit. They may approach a solution either from the interior or from the exterior of the feasible region, but they never lie on the boundary of this region.

In what follows, we first describe the general idea of a primal-dual method and then focus on *feasible primal-dual methods*, that are interior-point method generating feasible iterates, and *infeasible primal-dual methods*, that correspond to infeasible iterates.

To describe a general primal-dual method, we consider the LP problem (1.1) and the corresponding KKT system (1.10)-(1.14). In what follows, we assume that the matrix A is full row rank (as we did for the simplex method).

Primal-dual methods find solutions $(\mathbf{x}^\star, \boldsymbol{\lambda}^\star, \mathbf{s}^\star)$ to (1.10)-(1.14) by applying variants of Newton's method to the three equalities of the KKT system and modifying the approximation obtained at each iteration in order to satisfy strictly the inequality conditions $\mathbf{x} \geq 0$ and $\mathbf{s} \geq 0$.

Let us rewrite (1.10)(1.14) as

$$G(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) := \begin{bmatrix} A^\top \boldsymbol{\lambda} + \mathbf{s} - \mathbf{c} \\ A\mathbf{x} - \mathbf{b} \\ XS\mathbf{e} \end{bmatrix} = 0$$
$$\mathbf{x} \geq 0, \ \mathbf{s} \geq 0,$$

where

$$X := \operatorname{diag}\big((\mathbf{x})_1, \ldots, (\mathbf{x})_n\big) \in \mathbb{R}^{n \times n}, \quad S := \operatorname{diag}\big((\mathbf{s})_1, \ldots, (\mathbf{s})_n\big) \in \mathbb{R}^{n \times n},$$

and $\mathbf{e} = [1 \ \ldots \ 1]^\top \in \mathbb{R}^n$. To apply *Newton's method* to the root problem

$$G(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = 0,$$

given an approximation $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ we must solve the Newton linear system

$$J(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = -G(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) \tag{2.12}$$

to compute the direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$, and then update the approximation $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ as

$$(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+) = (\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) + (\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}). \tag{2.13}$$

In (2.12), $J(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ is the Jacobian matrix of $G : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ at $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$, that is

$$J(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = \begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \in \mathbb{R}^{(2n+m)\times(2n+m)},$$

with I the $n \times n$ identity. In (2.12), $G(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ represents the residual of the root problem at $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$:

$$G(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) = \begin{bmatrix} \mathbf{r}_c \\ \mathbf{r}_b \\ X_0 S_0 \mathbf{e} \end{bmatrix},$$

where

$$\mathbf{r}_c = A^\top \boldsymbol{\lambda}^0 + \mathbf{s}^0 - \mathbf{c}, \quad \mathbf{r}_b = A\mathbf{x}^0 - \mathbf{b}.$$

Each iteration of Newton's method consists in solving the linear system (2.12) and updating the current approximation as in (2.13). This is a full Newton step.

Usually, a full Newton step would violate the bounds $\mathbf{x} \geq 0$ and $\mathbf{s} \geq 0$. For this reason, interior-point methods perform a so-called *line-search* along the Newton direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$ to obtain a step-length $\alpha \in (0, 1]$ and compute the new approximation as

$$(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+) = (\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) + \alpha(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}).$$

The parameter α must be computed so that $\mathbf{x}^+ > 0$ and $\mathbf{s}^+ > 0$. This property is the origin of the term *interior-point*. Clearly, if $\mathbf{x}^0 > 0$ and $\mathbf{s}^0 > 0$, then one can find a sufficiently small α in $(0, 1]$ such that $\mathbf{x}^+ > 0$ and $\mathbf{s}^+ > 0$. On the other hand, we would like to have α as big as possible to produce a significant improvement toward a solution. The easiest way to compute α is to perform a so-called *backtracking*: one starts with $\alpha = 1$, computes $(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+)$ and verify the conditions $\mathbf{x}^+ > 0$ and $\mathbf{s}^+ > 0$. If these are satisfied, then the value of α is accepted, if not then the value of α is reduced (e.g., divided by 2). This procedure is repeated until a value of α such that $\mathbf{x}^+ > 0$ and $\mathbf{s}^+ > 0$ is found.

Often, the "pure" Newton direction, that is the solution to the Newton system

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c \\ -\mathbf{r}_b \\ -XS\mathbf{e} \end{bmatrix}, \tag{2.14}$$

does not allow one to make much progress toward a solution, because the resulting step-length α is very small. For this reason, most primal-dual methods use a less aggressive Newton direction that does not solve (2.14) exactly. Let us introduce the so-called *duality measure*

$$\mu = \frac{1}{n}\mathbf{x}^\top \mathbf{s},$$

where n is the size of the vectors $\mathbf{x}$ and $\mathbf{s}$, and the following modified Newton system

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c \\ -\mathbf{r}_b \\ -XS\mathbf{e} + \sigma\mu\mathbf{e} \end{bmatrix}, \tag{2.15}$$

where $\sigma \in [0, 1]$ is called *centering parameter*. When $\sigma > 0$, it is usually possible to take a larger step α along the direction defined by (2.15).

We are now ready to sketch a general primal-dual algorithm given by Algorithm 2. Notice that we denoted σ and α by σ_k and α_k to indicate that their

Algorithm 2 (General Primal-Dual Algorithm)

Require: The matrix A and the vectors $\mathbf{b}$ and $\mathbf{c}$. An initial guess $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ such that $\mathbf{x}^0 > 0$ and $\mathbf{s}^0 > 0$. A maximum number of iterations $k_{\max}$.

1: **for** $k = 0{:}k_{\max}$ **do**
2: Choose $\sigma_k \in [0, 1]$.
3: Compute $\mu_k = \frac{1}{n}(\mathbf{x}^k)^\top \mathbf{s}^k$.
4: Solve the linear system (2.15):

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^k \\ \Delta\boldsymbol{\lambda}^k \\ \Delta\mathbf{s}^k \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c^k \\ -\mathbf{r}_b^k \\ -X_k S_k \mathbf{e} + \sigma_k \mu_k \mathbf{e} \end{bmatrix}.$$

5: Compute a step-length α_k and the new approximation

$$(\mathbf{x}^{k+1}, \boldsymbol{\lambda}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{x}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k) + \alpha_k(\Delta\mathbf{x}^k, \Delta\boldsymbol{\lambda}^k, \Delta\mathbf{s}^k)$$

such that $\mathbf{x}^{k+1} > 0$ and $\mathbf{s}^{k+1} > 0$.
6: If some stopping criterion is satisfied, then break.
7: **end for**

values can change in the iterations. The choice of σ_k and α_k influences strongly the performance of the method; see, e.g., [10] for further details.

An example of MATLAB implementation of Algorithm 2 is given in the following script.

```
% Example Nocedal-Wright (page 371)
A = [ 1 1 1 0 ; 2 0.5 0 1 ];
b = [ 5 ; 8 ];
c = [ -4 ; -2 ; 0 ; 0 ];
n = length(c);
m = length(b);
e = ones(n,1);
x0 = 4*ones(n,1);
s0 = ones(n,1);
l0 = 4*ones(m,1);
sigma = 0.01;
M = zeros(2*n+m);
M(1:n,n+1:n+m) = A';
M(1:n,n+m+1:2*n+m) = eye(n);
```

```
M(n+1:n+m,1:n) = A;
x = x0; s = s0; l = l0;
mu = (x'*s)/n;
v = [x;l;s];
x_iter = x;
for j=1:30
    M(n+m+1:2*n+m,1:n) = diag(s);
    M(n+m+1:2*n+m,n+m+1:2*n+m) = diag(x);
    rhs = - [ A'*l+s-c ; A*x-b; diag(s)*x - sigma*mu*e ];
    d = M\rhs;
    alpha = 1;
    for k=1:30 % backtracking for computing alpha
        v_new = v + alpha*d;
        if min(v_new(1:n))>0 && min(v_new(n+m+1:2*n+m))>0
            break
        end
        alpha = alpha/2;
    end
    v = v + alpha*d;
    x = v(1:n); l = v(n+1:n+m); s = v(n+m+1:2*n+m);
    x_iter = [x_iter,x];
    mu = (x'*s)/n; f = c'*x;
    fprintf('j= %3.0d | mu= %5.3e | f= %5.3e | alpha= %5.3e |\n',j, mu, f, alpha);
    if mu<1e-6
        break
    end
end
```

This script solves the same LP problem that we have solved using the simplex method in Section 2.1 and Figure 2.3. The primal-dual method needs 11 iterations to converge. Notice that, for this very small-dimensional example, the primal-dual method performs less good than the simplex method. However, as we will see in the next sections, this is not in general the case for higher-dimensional problems, on which primal-dual methods outperform the simplex method.

In Figure 2.4, the first 4 iterations and the last iteration are depicted. It is clear that the method generates a sequence that converges by approaching the boundary from the exterior of the feasible polyhedron. If the same experiment is repeated using the initial guess

```
x0 = 4*ones(n,1);
s0 = 4*ones(n,1);
l0 = 4*ones(m,1);
```

then one would obtain the results depicted in Figure 2.5, which shows a completely different behavior. Already at iteration 2, the method computes an approximation that lies in the feasible polyhedron, and the sequence approaches the minimum from the interior of the feasible set.

The two different behaviors shown in Figures 2.4 and 2.5 suggest that two different kinds of primal-dual methods can be developed using Algorithm 2.

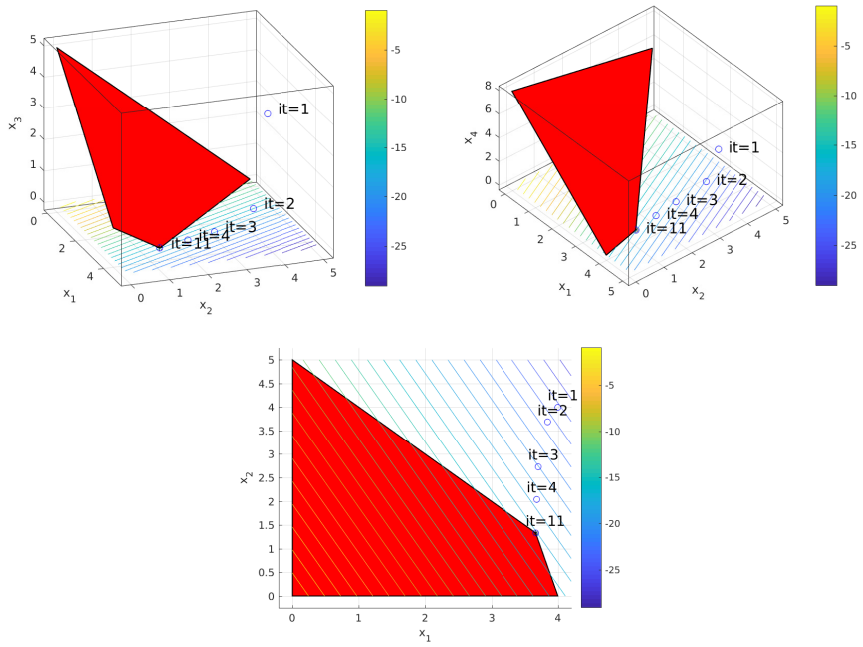

Figure 2.4: Example of iterations of the interior-point algorithm. This is the same example described in Figure 2.3. Notice that at each iteration the algorithm corresponds to a point outside the feasible region (red area). The boundary of the polyhedron is approached only in the limit.

These are described in the next two subsections.

2.4.1 Feasible primal-dual methods

Although practical implementations of interior-point methods do not require the initial guess to be a feasible point, most of the historical development of theory and algorithms assumed that these conditions are satisfied [10]. In general, the requirement for a feasible initial guess could be quite restrictive, but it allows the development of efficient feasible methods.

In order to define feasible primal-dual methods, we introduce the *primal-dual strictly feasible set*

$$F^0 := \{(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \ : \ A\mathbf{x} = \mathbf{b},\ A^\top \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c},\ \mathbf{x} > 0,\ \mathbf{s} > 0\},$$

and the set

$$N_{-\infty}(\gamma, \mu) := \{(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) \in F^0 \ : \ (\mathbf{x})_i (\mathbf{s})_i \geq \gamma \mu,\ i = 1, 2, \ldots, n\}$$

for some $\gamma \in (0, 1]$. Consider an initial guess

$$(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) \in N_{-\infty}(\gamma, \mu_0),$$

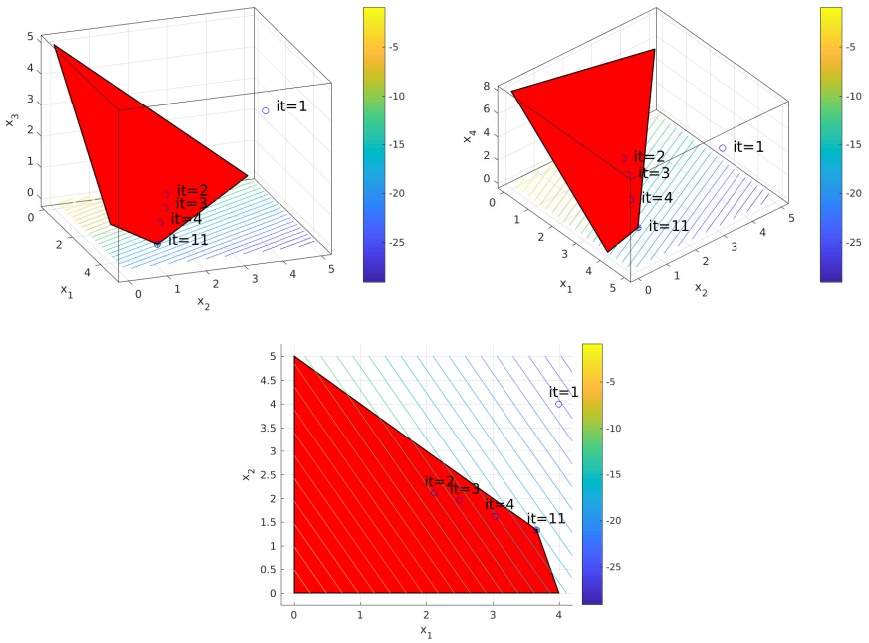

Figure 2.5: Example of iterations of the interior-point algorithm. This is the same example described in Figure 2.4, but it shows the behavior of the Algorithm 2 initialized with a different initial guess. The approximations generated by the primal-dual method approach the minimum from the interior of the feasible polyhedron.

where $\mu_0 = \frac{1}{n}(\mathbf{x}^0)^\top \mathbf{s}^0$. Since this triple is feasible, we clearly have that

$$\mathbf{r}_b = A\mathbf{x}^0 - \mathbf{b} = 0 \quad \text{and} \quad \mathbf{r}_c = A^\top \boldsymbol{\lambda}^0 + \mathbf{s}^0 - \mathbf{c} = 0,$$

which implies that the Newton linear system (2.15) becomes

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XS\mathbf{e} + \sigma\mu\mathbf{e} \end{bmatrix}. \tag{2.16}$$

If we solve this system and then compute a new approximation by

$$(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+) = (\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) + \alpha(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}), \tag{2.17}$$

we observe that

$$A\mathbf{x}^+ = A\mathbf{x}^0 + \alpha A\Delta\mathbf{x} = A\mathbf{x}^0 = \mathbf{b},$$

where we used that $A\Delta\mathbf{x} = 0$ and that $\mathbf{x}^0$ is feasible, and

$$A^\top \boldsymbol{\lambda}^+ + \mathbf{s}^+ = A^\top \boldsymbol{\lambda}^0 + \mathbf{s}^0 + \alpha(A^\top \Delta\boldsymbol{\lambda} + \Delta\mathbf{s}) = A^\top \boldsymbol{\lambda}^0 + \mathbf{s}^0 = \mathbf{c},$$

where we used that $A^\top \Delta\boldsymbol{\lambda} + \Delta\mathbf{s} = 0$ and that $\boldsymbol{\lambda}^0$ and $\mathbf{s}^0$ are feasible. Moreover, it is possible to show that one can always find a small enough α such that $(\mathbf{x}^+)_i(\mathbf{s}^+)_i \geq \gamma\mu^+$, where $\mu^+ = \frac{1}{n}(\mathbf{x}^+)^\top\mathbf{s}^+$ (see proof of Theorem 14.3 in [10]). Therefore, the new triple $(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+)$ lies in $N_{-\infty}(\gamma, \mu^+)$ and hence is feasible. Proceeding iteratively by solving (2.16) and updating as in (2.17), we obtain the feasible primal-dual Algorithm 3.

Algorithm 3 (Feasible (Path-following) Primal-Dual Algorithm)

Require: The matrix A and the vectors $\mathbf{b}$ and $\mathbf{c}$. An initial guess $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) \in N_{-\infty}(\gamma, \mu^0)$. Parameters $\gamma \in (0,1)$ and $\sigma_{\max}, \sigma_{\min} \in (0,1)$ such that $0 < \sigma_{\min} \leq \sigma_{\max} < 1$. A maximum number of iterations $k_{\max}$.

1: **for** $k = 0{:}\,k_{\max}$ **do**
2: Choose $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$.
3: Compute $\mu_k = \frac{1}{n}(\mathbf{x}^k)^\top\mathbf{s}^k$.
4: Solve the linear system (2.16).
5: Compute the largest step-length α_k such that the new approximation

$$(\mathbf{x}^{k+1}, \boldsymbol{\lambda}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{x}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k) + \alpha_k(\Delta\mathbf{x}_k, \Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k)$$

lies in $N_{-\infty}(\gamma, \mu^{k+1})$.
6: If some stopping criterion is satisfied, then break.
7: **end for**

It is possible to prove that this algorithm converges to a solution of the KKT system; see, e.g., [10, Theorem 14.3]. Moreover, this algorithm can perform very efficiently if σ_k is chose in an "appropriate" way. However, a primal-dual strictly feasible initial guess is required! This is the main drawback of this method and the reason for the large use of infeasible primal-dual methods in practical implementations.

A MATLAB implementation of Algorithm 3 is the following.

```
function [x,x_iter] = IP_feasible(A,b,c,x0,s0,l0,tol,maxit)
n = length(c);
m = length(b);
e = ones(n,1);
sigma = 0.1;
gamma = 1e-3;
M = zeros(2*n+m);
M(1:n,n+1:n+m) = A';
M(1:n,n+m+1:2*n+m) = eye(n);
M(n+1:n+m,1:n) = A;
rhs = zeros(2*n+m,1);
x = x0; s = s0; l = l0;
mu = (x'*s)/n;
v = [x;l;s];
x_iter = x;
for it=1:maxit
    M(n+m+1:2*n+m,1:n) = diag(s);
    M(n+m+1:2*n+m,n+m+1:2*n+m) = diag(x);
```

```
    rhs(n+m+1:2*n+m) = sigma*mu*e - diag(s)*x;
    d = M\rhs;
    alpha = 1;
    for k=1:30 % backtracking for computing alpha
        v_new = v + alpha*d;
        mu_new = (v_new(1:n)'*v_new(n+m+1:2*n+m))/n;
        if min(v_new(1:n).*v_new(n+m+1:2*n+m))>mu_new*gamma
            break
        end
        alpha = alpha*0.5;
    end
    v = v + alpha*d;
    x = v(1:n); l = v(n+1:n+m); s = v(n+m+1:2*n+m);
    x_iter = [x_iter,x];
    mu = (x'*s)/n;
    rc = A'*l+s-c;
    rb = A*x-b;
    opt= norm(rc)+norm(rb)+mu;
    f = c'*x;
    fprintf('it= %3.0d | mu= %5.3e | f= %5.3e | opt= %5.3e | alpha= %5.3e |\n',it, mu, f, opt, alpha);
    if abs(opt)<tol
        break
    end
end
end
```

We can test this function for our usual example by running the following MATLAB script.

```
% Example Nocedal-Wright (page 371)
A = [ 1 1 1 0 ; 2 0.5 0 1 ];
b = [ 5 ; 8 ];
c = [ -4 ; -2 ; 0 ; 0 ];
n = length(c);
m = length(b);
x0 = [ 1 ; 1 ; 3 ; 5.5 ]; % primal-dual strictly feasible initial guess
l0 = -2*ones(2,1);
s0 = c-A'*l0;
[x,x_iter] = IP_feasible(A,b,c,x0,s0,l0,1e-6,30);
```

This script shows that the algorithm converges in 9 iterations. In Figure 2.6 the first 5 iterates are shown. The method is clearly converging to the solution and each iterate lies in the feasible set.

2.4.2 Infeasible primal-dual methods

In this section, we describe the infeasible primal-dual method that was introduced by S. Mehrotra in 1992 [8][2]. Mehrotra's algorithm is a primal-dual

[2]Notice that the interior-point method implemented in the MATLAB function `linprog` is based on a variant of Mehrotra's algorithm; see [13].

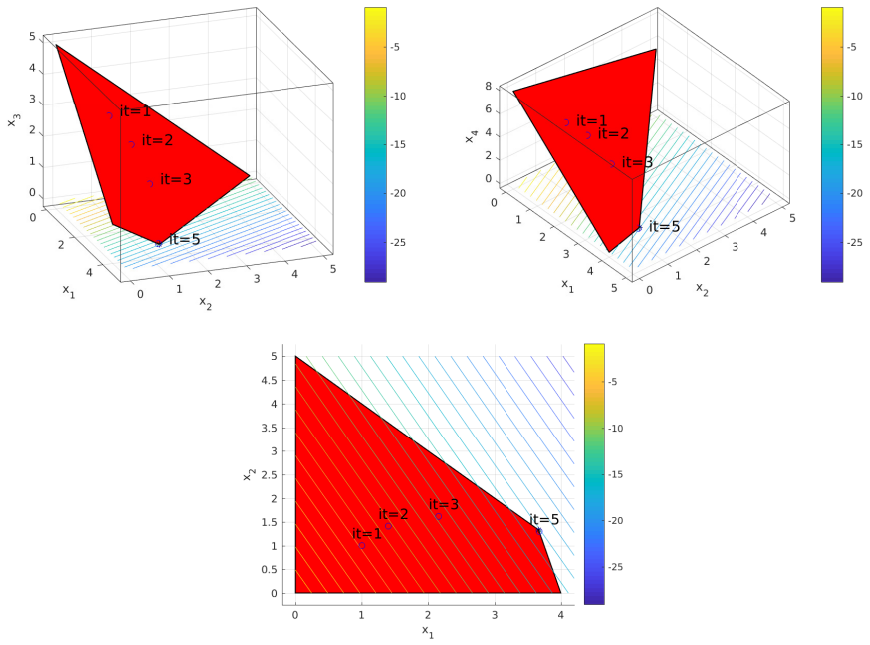

Figure 2.6: Example of iterations of the feasible primal dual algorithm. This is the same example described in Figure 2.4. The approximations generated by the feasible primal-dual method approach the minimum from the interior of the feasible polyhedron.

method that performs at each iteration a prediction step and a correction step. For this reason, this is also called *predictor-corrector primal-dual method*; see, e.g., [10].

Given an approximation $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$, Mehrotra's method performs a prediction step by solving the Newton system

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S_0 & 0 & X_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{pre} \\ \Delta\boldsymbol{\lambda}^{pre} \\ \Delta\mathbf{s}^{pre} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c^0 \\ -\mathbf{r}_b^0 \\ -X_0 S_0 \mathbf{e} \end{bmatrix} \tag{2.18}$$

and then computing

$$(\mathbf{x}^{pre}, \boldsymbol{\lambda}^{pre}, \mathbf{s}^{pre}) = (\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0) + (\Delta\mathbf{x}^{pre}, \Delta\boldsymbol{\lambda}^{pre}, \Delta\mathbf{s}^{pre}).$$

By the linearity of the first two equations of $G(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = 0$, this prediction satisfies the equalities $A\mathbf{x}^{pre} = \mathbf{b}$ and $A^\top\boldsymbol{\lambda}^{pre} + \mathbf{s}^{pre} = \mathbf{c}$. However, the KKT condition (1.14) (the third equation of $G(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = 0$) is not satisfied, and we

can compute the corresponding residual:

$$\begin{aligned}[(\mathbf{x}^0)_j + (\Delta\mathbf{x}^{pre})_j][(\mathbf{s}^0)_j + (\Delta\mathbf{s}^{pre})_j] &= (\mathbf{x}^0)_j(\mathbf{s}^0)_j + (\mathbf{x}^0)_j(\Delta\mathbf{s}^{pre})_j \\ &\quad + (\mathbf{s}^0)_j(\Delta\mathbf{x}^{pre})_j + (\Delta\mathbf{x}^{pre})_j(\Delta\mathbf{s}^{pre})_j \\ &= (\Delta\mathbf{x}^{pre})_j(\Delta\mathbf{s}^{pre})_j\end{aligned}$$

for $j = 1, \ldots, n$, where we used the last (block) equation of (2.18). Hence, the updated value $(\mathbf{x}^{pre})_j(\mathbf{s}^{pre})_j$ is $(\Delta\mathbf{x}^{pre})_j(\Delta\mathbf{s}^{pre})_j$ rather than the ideal value zero. Now, the idea is to attempt to correct this value by solving another Newton-type system:

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S_0 & 0 & X_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{cor} \\ \Delta\boldsymbol{\lambda}^{cor} \\ \Delta\mathbf{s}^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Delta X_{pre}\Delta S_{pre}\mathbf{e} \end{bmatrix}. \tag{2.19}$$

The new direction is then

$$(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}) = (\Delta\mathbf{x}^{pre}, \Delta\boldsymbol{\lambda}^{pre}, \Delta\mathbf{s}^{pre}) + (\Delta\mathbf{x}^{cor}, \Delta\boldsymbol{\lambda}^{cor}, \Delta\mathbf{s}^{cor}).$$

In many practical cases this predictor-corrector step performs very well. As for feasible methods, also Mehrotra's algorithm uses duality measure and centering parameter. The centering parameter is chosen as

$$\sigma = \Big(\frac{\mu_{pre}}{\mu_0}\Big)^3, \tag{2.20}$$

where $\mu_0 = \frac{1}{n}(\mathbf{x}^0)^\top\mathbf{s}^0$ is the duality measure and μ_{pre} is the predicted duality measure given by

$$\mu_{pre} = \frac{1}{n}(\mathbf{x}^0 + \alpha^p_{pre}\Delta\mathbf{x}^{pre})^\top(\mathbf{s}^0 + \alpha^d_{pre}\Delta\mathbf{s}^{pre}). \tag{2.21}$$

Here we used two different step-length parameters: α^p_{pre} for the primal variable $\mathbf{x}$ and α^d_{pre} for the dual variable $\mathbf{s}$. These are computed as

$$\begin{aligned}\alpha^p_{pre} &:= \min\Big(1, \min_{j\,:\,(\Delta\mathbf{x}^{pre})_j<0} -\frac{(\mathbf{x}^0)_j}{(\Delta\mathbf{x}^{pre})_j}\Big), \\ \alpha^d_{pre} &:= \min\Big(1, \min_{j\,:\,(\Delta\mathbf{s}^{pre})_j<0} -\frac{(\mathbf{s}^0)_j}{(\Delta\mathbf{s}^{pre})_j}\Big),\end{aligned} \tag{2.22}$$

and represent the maximum allowable step lengths along the directions $\Delta\mathbf{x}^{pre}$ and $\Delta\mathbf{s}^{pre}$ (larger steps would violate the constraints $\mathbf{x}^{pre} > 0$ and $\mathbf{s}^{pre} > 0$).

To summarize, Mehrotra's predictor-corrector strategy performs at each iteration the following key steps:

(a) Solve (2.18) to obtain the prediction direction $(\Delta\mathbf{x}^{pre}, \Delta\boldsymbol{\lambda}^{pre}, \Delta\mathbf{s}^{pre})$.

(b) Compute the centering parameter σ using (2.20).

(c) Solve the linear system

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S_0 & 0 & X_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c^0 \\ -\mathbf{r}_b^0 \\ -X_0 S_0 \mathbf{e} - \Delta X_{pre} \Delta S_{pre} \mathbf{e} + \sigma\mu_0 \mathbf{e} \end{bmatrix} \tag{2.23}$$

to obtain the predictor-corrector direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$.

Once the direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$ is obtained, the algorithm computes primal step length $\alpha^p_{\max}$ and dual step length $\alpha^d_{\max}$ in $(0, 1]$ that are as big as possible and that allow us to guarantee that the new iterate $(\mathbf{x}^+, \boldsymbol{\lambda}^+, \mathbf{s}^+)$ satisfies $\mathbf{x}^+ > 0$ and $\mathbf{s}^+ > 0$. These step lengths are computed as

$$\alpha^p_{\max} := \min_{j\,:\,(\Delta\mathbf{x}^{pre})_j < 0} -\frac{(\mathbf{x}^0)_j}{(\Delta\mathbf{x}^{pre})_j},$$
$$\alpha^d_{\max} := \min_{j\,:\,(\Delta\mathbf{s}^{pre})_j < 0} -\frac{(\mathbf{s}^0)_j}{(\Delta\mathbf{s}^{pre})_j}.$$

In practical implementations, these values are modified:

$$\alpha^p = \min(1, \eta\alpha^p_{\max}), \quad \alpha^d = \min(1, \eta\alpha^d_{\max}), \tag{2.24}$$

where $\eta \in [0.9, 1)$ and $\eta \to 1$ as the iteration count increases. Using these step lengths, the new approximation is computed as

$$\mathbf{x}^+ = \mathbf{x}^0 + \alpha^p \Delta\mathbf{x},$$
$$\boldsymbol{\lambda}^+ = \boldsymbol{\lambda}^0 + \alpha^d \Delta\boldsymbol{\lambda},$$
$$\mathbf{s}^+ = \mathbf{s}^0 + \alpha^d \Delta\mathbf{s}.$$

Once the new approximation is computed, the predictor-corrector step is repeated iteratively. Mehrotra's strategy is summarized in Algorithm 4.

A MATLAB implementation of Algorithm 4 is the following.

```
function [x,l,s] = IP_Mehrotra(A,b,c,x0,s0,l0,tol,maxit)
n = length(c);
m = length(b);
e = ones(n,1);
sigma = 0.1;
M = zeros(2*n+m);
M(1:n,n+1:n+m) = A';
M(1:n,n+m+1:2*n+m) = eye(n);
M(n+1:n+m,1:n) = A;
x = x0; s = s0; l = l0;
rc = A'*l+s-c;
rb = A*x-b;
mu = (x'*s)/n;
for it=1:maxit
    M(n+m+1:2*n+m,1:n) = diag(s);
    M(n+m+1:2*n+m,n+m+1:2*n+m) = diag(x);
```

Algorithm 4 (Mehrotra's Primal-Dual Algorithm)

Require: The matrix A and the vectors $\mathbf{b}$ and $\mathbf{c}$. An initial guess $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ such that $(\mathbf{x}^0, \mathbf{s}^0) > 0$. A maximum number of iterations $k_{\max}$.

1: **for** $k = 0{:}k_{\max}$ **do**

2: Set $(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = (\mathbf{x}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k)$ and solve (2.19) to get $(\Delta\mathbf{x}^{pre}, \Delta\boldsymbol{\lambda}^{pre}, \Delta\mathbf{s}^{pre})$.

3: Compute α^p_{pre}, α^d_{pre}, μ_k, μ_{pre} and σ using (2.22), (2.21) and (2.20).

4: Solve the linear system

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_c^k \\ -\mathbf{r}_b^k \\ -X_k S_k \mathbf{e} - \Delta X_{pre} \Delta S_{pre} \mathbf{e} + \sigma\mu_k \mathbf{e} \end{bmatrix}$$

to obtain the predictor-corrector direction $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$.

5: Compute α^p and α^d using (2.24).

6: Compute the new updates as

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha^p \Delta\mathbf{x}, \\ \boldsymbol{\lambda}^{k+1} &= \boldsymbol{\lambda}^k + \alpha^d \Delta\boldsymbol{\lambda}, \\ \mathbf{s}^{k+1} &= \mathbf{s}^k + \alpha^d \Delta\mathbf{s}. \end{aligned}$$

7: If some stopping criterion is satisfied, then break.

8: **end for**

```
rhs = - [ rc ; rb; s.*x ]; % affine step
d_aff = M\rhs;
ratio=-(x./d_aff(1:n)).*(d_aff(1:n)<0) ...
      +1e8*ones(size(x)).*(d_aff(1:n)>=0);
alpha_p_m=min(ratio);
ratio=-(s./d_aff(n+m+1:2*n+m)).*(d_aff(n+m+1:2*n+m)<0) ...
      +1e8*ones(size(s)).*(d_aff(n+m+1:2*n+m)>=0);
alpha_d_m=min(ratio);
alpha_aff_p=min(1,alpha_p_m);
alpha_aff_d=min(1,alpha_d_m);
mu_aff = ((x+alpha_aff_p*d_aff(1:n))'*(s+alpha_aff_d*d_aff(n+m+1:2*n+m)))/n;
sigma = (mu_aff/mu)^3; % centering parameter ...
rhs(n+m+1:2*n+m) = rhs(n+m+1:2*n+m)-d_aff(1:n).*d_aff(n+m+1:2*n+m)+sigma*mu*e;
d = M\rhs;     % corrector step
ratio=-(x./d(1:n)).*(d(1:n)<0)+1e8*ones(size(x)).*(d(1:n)>=0);
alpha_p_m=min(ratio);
ratio=-(s./d(n+m+1:2*n+m)).*(d(n+m+1:2*n+m)<0)+1e8*ones(size(s)).*(d(n+m+1:2*n+m)>=0);
alpha_d_m=min(ratio);
eta = min(1,0.9+0.1*it/10);
alpha_p = min(1,eta*alpha_p_m); % compute primal and dual step length
alpha_d = min(1,eta*alpha_d_m);
x = x + alpha_p*d(1:n); % update
l = l + alpha_d*d(n+1:n+m);
s = s + alpha_d*d(n+m+1:2*n+m);
mu = (x'*s)/n;
```

```
    rc = A'*l+s-c; rb = A*x-b;
    opt= norm(rc)+norm(rb)+mu;
    f = c'*x;
    fprintf('it= %3.0d | mu= %5.3e | f= %5.3e | opt= %5.3e | eta= %5.3e | alpha= %5.3e |\n',it,
    if abs(opt)<tol
        break
    end
end
end
```

Notice that no convergence analysis is available for Mehrotra's algorithm; see, e.g., [10]. There are examples on which the algorithm diverges, but these failures are rare and sometimes "mitigated" by detailed modifications of the algorithm.

The performance of primal-dual algorithms is strongly affected by the choice of the initial guess. For infeasible algorithm, like Mehrotra's method, there are several heuristic choices described in the literature; see, e.g., [10, 12]. In what follows, we describe one of these.

The first step toward the computation of a warm start is to get a triple $(\widehat{\mathbf{x}}, \widehat{\boldsymbol{\lambda}}, \widehat{\mathbf{s}})$ such that $\widehat{\mathbf{x}}$ satisfies $A\mathbf{x} = \mathbf{b}$ and has minimum norm, and $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\mathbf{s}}$ satisfy $A^\top \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c}$ and $\widehat{\mathbf{s}}$ has minimum norm. Such a triple solves the two problems

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\mathbf{x}^\top\mathbf{x} && \text{s.t. } A\mathbf{x} = \mathbf{b}, \\ &\min_{\mathbf{s}\in\mathbb{R}^n,\, \boldsymbol{\lambda}\in\mathbb{R}^m} \frac{1}{2}\mathbf{s}^\top\mathbf{s} && \text{s.t. } A^\top\boldsymbol{\lambda} + \mathbf{s} = \mathbf{c}. \end{aligned}$$

It is not difficult to show that their solutions are

$$\widehat{\mathbf{x}} = A^\top(AA^\top)^{-1}\mathbf{b}, \quad \widehat{\boldsymbol{\lambda}} = (AA^\top)^{-1}A\mathbf{c}, \quad \widehat{\mathbf{s}} = \mathbf{c} - A^\top\widehat{\boldsymbol{\lambda}}.$$

In general, $\widehat{\mathbf{x}}$ and $\widehat{\mathbf{s}}$ can have negative components, which means that they are not suitable as starting point. For this reason, we correct them:

$$\widetilde{\mathbf{x}} = \widehat{\mathbf{x}} + \delta_x\mathbf{e}, \quad \widetilde{\mathbf{s}} = \widehat{\mathbf{s}} + \delta_s\mathbf{e},$$

where $\mathbf{e} = [1 \ \dots \ 1]^\top \in \mathbb{R}^n$ and

$$\delta_x = \max\Big(-(3/2)\min_j(\widehat{\mathbf{x}})_j, 0\Big), \quad \delta_s = \max\Big(-(3/2)\min_j(\widehat{\mathbf{s}})_j, 0\Big).$$

Clearly $\widetilde{\mathbf{x}} > 0$ and $\widetilde{\mathbf{s}} > 0$. Finally, to ensure that the components of $\mathbf{x}^0$ and $\mathbf{s}^0$ are not too close to zero and not too dissimilar, we introduce two more scalars:

$$\widehat{\delta}_x = \frac{1}{2}\frac{\widetilde{\mathbf{x}}^\top\widetilde{\mathbf{s}}}{\mathbf{e}^\top\widetilde{\mathbf{s}}}, \quad \widehat{\delta}_s = \frac{1}{2}\frac{\widetilde{\mathbf{x}}^\top\widetilde{\mathbf{s}}}{\mathbf{e}^\top\widetilde{\mathbf{x}}}.$$

Notice that $\widehat{\delta}_x$ is the average size of the components of $\widetilde{\mathbf{x}}$, weighted by the corresponding components of $\widetilde{\mathbf{s}}$; similarly for $\widehat{\delta}_s$. We can finally compute the initial guess

$$\mathbf{x}^0 = \widetilde{\mathbf{x}} + \widehat{\delta}_x \mathbf{e}, \quad \boldsymbol{\lambda}^0 = \widehat{\boldsymbol{\lambda}}, \quad \mathbf{s}^0 = \widetilde{\mathbf{s}} + \widehat{\delta}_s \mathbf{e}.$$

To compute this initial guess, it is possible to use the following MATLAB function.

```
function [x0,l0,s0] = warm_start(A,b,c)
e = ones(length(c),1);
x0 = A'*((A*A')\b);
l0 = (A*A')\(A*c);
s0 = c - A'*l0;
dx = max(-(3/2)*min(x0),0);
ds = max(-(3/2)*min(s0),0);
x0 = x0+dx*e; s0 = s0+ds*e;
dx = 0.5*(x0'*s0)/(e'*s0);
ds = 0.5*(x0'*s0)/(e'*x0);
x0 = x0+dx*e; s0 = s0+ds*e;
end
```

Now, we wish to test Mehrotra's method and compare it with the simplex method of Section 2.1. To do so, we use the following MATLAB script.

```
m = 100; n = 2*m;
A = rand(m,2*m);
indB = 1:m;
x0 = [rand(m,1);zeros(m,1)];
b = A*x0;
c = rand(2*m,1);
disp('Simplex Iterations')
[x_simplex,sN] = simplex(A,b,c,indB,200);
disp('Simplex Iterations')
[x0,l0,s0] = warm_start(A,b,c);
[x,l,s] = IP_Mehrotra(A,b,c,x0,s0,l0,1e-8,200);
norm(x-x_simplex)
```

This script tests for a random problem of dimensions $m = 100$ and $n = 200$. The results are similar (due to randomness) to the following.

```
Simplex Iterations
it=   1 | f= 2.326e+01
it=   2 | f= 2.324e+01
it=   3 | f= 2.318e+01
it=   4 | f= 2.313e+01
it=   5 | f= 2.301e+01
it=   6 | f= 2.271e+01
it=   7 | f= 2.268e+01
it=   8 | f= 2.261e+01
it=   9 | f= 2.258e+01
it=  10 | f= 2.258e+01
...
```

```
it=  83 | f= 1.862e+01
it=  84 | f= 1.861e+01
it=  85 | f= 1.860e+01
it=  86 | f= 1.859e+01
it=  87 | f= 1.859e+01
Simplex Iterations
it=   1 | mu= 1.839e-01 | f= 3.439e+01 | opt= 1.263e+02 | eta= 9.100e-01 | alpha= 8.028e-01 |
it=   2 | mu= 4.066e-02 | f= 2.258e+01 | opt= 2.181e+01 | eta= 9.200e-01 | alpha= 7.869e-01 |
it=   3 | mu= 1.338e-02 | f= 1.994e+01 | opt= 6.707e+00 | eta= 9.300e-01 | alpha= 6.740e-01 |
it=   4 | mu= 4.810e-03 | f= 1.909e+01 | opt= 2.192e+00 | eta= 9.400e-01 | alpha= 6.732e-01 |
it=   5 | mu= 1.681e-03 | f= 1.876e+01 | opt= 6.764e-01 | eta= 9.500e-01 | alpha= 6.664e-01 |
it=   6 | mu= 4.315e-04 | f= 1.862e+01 | opt= 1.196e-01 | eta= 9.600e-01 | alpha= 7.231e-01 |
it=   7 | mu= 8.515e-05 | f= 1.860e+01 | opt= 3.314e-02 | eta= 9.700e-01 | alpha= 7.210e-01 |
it=   8 | mu= 1.817e-05 | f= 1.859e+01 | opt= 3.511e-03 | eta= 9.800e-01 | alpha= 7.032e-01 |
it=   9 | mu= 1.985e-06 | f= 1.859e+01 | opt= 3.786e-05 | eta= 9.900e-01 | alpha= 8.406e-01 |
it=  10 | mu= 1.121e-07 | f= 1.859e+01 | opt= 1.313e-06 | eta= 1.000e+00 | alpha= 9.618e-01 |
it=  11 | mu= 1.446e-08 | f= 1.859e+01 | opt= 1.446e-08 | eta= 1.000e+00 | alpha= 1.000e+00 |
it=  12 | mu= 4.697e-15 | f= 1.859e+01 | opt= 1.583e-13 | eta= 1.000e+00 | alpha= 1.000e+00 |

ans =

   3.7583e-12
```

The simplex algorithm requires about 90 iterations to converge, while Mehrotra's algorithm converges within a tolerance of 10^{-8} to a solution in only 12 iterations. The corresponding solution differs in norm from the simplex solution of around 10^{-12}. This simple experiment shows the higher efficiency of primal-dual methods for higher-dimensional problem.

Bibliography

[1] D. J. Albers and C. Reid. An interview with George B. Dantzig: The father of linear programming. *The College Mathematics Journal*, 17(4):292–314, 1986.

[2] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows (Third Edition)*. Wiley-Interscience, Hoboken, New Jersey, USA, 2010.

[3] G. B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, 1963.

[4] G. B. Dantzig. Reminiscences about the origins of linear programming. *Oper. Res. Lett.*, 1(2):43–48, 1982.

[5] W. Gander, M. J. Gander, and F. Kwok. *Scientific computing-An introduction using Maple and MATLAB*, volume 11. Springer Science & Business, 2014.

[6] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.

[7] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972.

[8] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[9] K. G. Murty. *Linear Programming*. Wiley, 1983.

[10] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[11] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[12] E. A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM J. on Optimization*, 12(3):782–810, 2002.

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

Index