OPTIMISATION NOTES

A COMPILATION OF LECTURE NOTES
FROM GRADUATE-LEVEL
OPTIMISATION COURSES

WRITTEN BY
FABRICIO OLIVEIRA
ASSOCIATE PROFESSOR OF OPERATIONS RESEARCH
AALTO UNIVERSITY, SCHOOL OF SCIENCE

SOURCE CODE AVAILABLE AT
github.com/gamma-opt/optimisation-notes

Contents

Part I

Linear optimisation

Chapter 1

Introduction

1.1 What is optimisation?

An optimisation is one of these words that has many meanings, depending on the context you take as a reference. In the context of this book, optimisation refers to *mathematical optimisation*, which is a discipline of applied mathematics.

In mathematical optimisation, we build upon concepts and techniques from calculus, analysis, linear algebra, and other domains of mathematics to develop methods that allow us finding values for variables within a given domain that maximise (or minimise) the value of a function. In specific, we are trying to solve the following general problem:

$$\begin{aligned} &\text{min. } f(x) \\ &\text{s.t.: } x \in X. \end{aligned} \tag{1.1}$$

That is, we would like to find the solution x that *minimises* the value of the *objective function* f, such that (s.t.) x belongs to the *feasibility set* X. In a general sense, problems like this can be solved by employing the following strategy:

1. Analysing properties of functions under specific domains and deriving the conditions that must be satisfied such that a point x is a candidate optimal point.
2. Applying numerical methods that iteratively search for points satisfying these conditions.

This idea is central in several domains of knowledge, and is very often defined under area-specific nomenclature. Fields such as economics, engineering, statistics, machine learning and, perhaps more broadly, operations research, are intensive users and developers of optimisation theory and applications.

1.1.1 Mathematical programming and optimisation

Operations research and mathematical optimisation are somewhat intertwined, as they both were born around a similar circumstance.

I like to separate *mathematical programming* from (mathematical) *optimisation*. Mathematical programming is a modelling paradigm, in which we rely on (very powerful, I might add) analogies to model *real-world* problems. In that, we look at a given decision problem, considering that

- *variables* represent *decisions*, as in a business decision or a course of action. Examples include setting the parameters of (e.g., prediction) models, production systems layouts, geometries of structures, topologies of networks, and so forth;

- *domain* represents business rules or *constraints*, representing logic relations, design or engineering limitations, requirements, and such;
- *function* is an *objective function* that provides a measure of solution quality.

With these in mind, we can represent the decision problem as a *mathematical programming model* of the form of (1.1) that can be solved using *optimisation* methods. From now on, we will refer to this specific class of models as mathematical optimisation models, or optimisation models for short. We will also use the term *to solve the problem* to refer to the task of finding optimal solutions to optimisation models.

This book is mostly focused on the optimisation techniques employed to find optimal solutions for these models. As we will see, depending on the nature of the functions f and g that are used to formulate the model, some methods might be more or less appropriate. Further complicating the issue, for models of a given nature, there might be alternative algorithms that can be employed and with no generalised consense whether one method is generally better performing than another, which is one of the aspects that make optimisation so exciting and multifaceted when it comes to alternative approaches. I hope that this makes more sense as we progress through the chapters.

1.1.2 Types of mathematical optimisation models

In general, the simpler the assumptions on the parts forming the optimisation model, the more efficient are the methods to solve such problems.

Let us define some additional notation that we will use from now on. Consider a model in the general form

$$\begin{aligned} \text{min. } & f(x) \\ \text{s.t.: } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_i(x) = 0, i = 1, \dots, l \\ & x \in X, \end{aligned}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, $g : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a collection of m inequality constraints and $h : \mathbb{R}^n \mapsto \mathbb{R}^l$ is a collection of l equality constraints.

Remark: in fact, every inequality constraint can be represented by an equality constraint by making $h_i(x) = g_i(x) + x_{n+1}$ and augmenting the decision variable vector $x \in \mathbb{R}^n$ to include the slack variable x_{n+1}. However, since these constraints are of very different nature, we will explicitly represent both whenever necessary.

The most general types of models are the following. We also use this as an opportunity to define some (admittedly confusing) nomenclature from the field of operations research that we will be using in these notes.

1. *Unconstrained models:* in these, the set $X = \mathbb{R}^n$ and $m = l = 0$. These are prominent in, e.g., machine learning and statistics applications, where f represents a measure of model fitness or prediction error.
2. *Linear programming (LP):* presume a linear objective function $f(x) = c^\top x$ and affine constraints g and h, i.e., of the form $a_i^\top x - b_i$, with $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$. Normally, $X = \{x \in \mathbb{R}^n \mid x_j \geq 0, j = 1, \dots, n\}$ enforces that decision variables are constrained to be the non-negative orthant.

3. *Nonlinear programming (NLP):* some or all of the functions f, g, and h are nonlinear.
4. *Mixed-integer (linear) programming (MIP):* consist of an LP in which some (or all, being then simply integer programming) of the variables are constrained to be integers. In other words, $X \subseteq \mathbb{R}^k \times \mathbb{Z}^{n-k}$. Very frequently, the integer variables are constrained to be binary terms, i.e., $x_i \in \{0, 1\}$, for $i = 1, \ldots, n - k$ and are meant to represent true-or-false or yes-or-no conditions.
5. *Mixed-integer nonlinear programming (MINLP):* are the intersection of MIPs and NLPs.

Remark: notice that we use the vector notation $c^\top x = \sum_{j \in J} c_j x_j$, with $J = \{1, \ldots, N\}$. This is just a convenience for keeping the notation compact.

1.2 Linear programming applications

We will consider now a few examples of liner programming models with somewhat general structure. Many of these examples have features that can be combined into more general models.

1.2.1 Resource allocation

Most linear programming (LP) problems can be interpreted as a *resource allocation* problem. In that, we are interested in defining an optimal allocation of resources (i.e., a plan) that maximises return or minimise costs and satisfy allocation rules.

Specifically, let $I = \{1, \ldots, i, \ldots, M\}$ by a set of resources that can be combined to produce products in the set $J = \{1, \ldots, j, \ldots, N\}$. Assume that we are given a return per unit of product c_j, $\forall j \in J$, and a list of a_{ij}, $\forall i \in I, \forall j \in J$, describing which and how much of the resources $i \in I$ are required for making product $j \in J$. Assume that the availability of resource b_i, $\forall i \in I$, is known.

Our objective is to define the amounts x_j representing the production of $j \in J$. We would like to define those in a way that we optimise the resource allocation plan quality (in our case, maximise return from the production quantities x_j) while making sure the amount produced do not exceed the availability of resources. For that, we need to define:

The *objective function*, which measures the *quality* of a production plan. In this case, the total return for a given plan is given by:

$$\text{max.} \sum_{j \in J} c_j x_j \Rightarrow c^\top x,$$

where $c = [c_1, \ldots, c_N]^\top$ and $x = [x_1, \ldots, x_N]^\top$ are n-sized vectors. The transpose sign $^\top$ is meant to reinforce that we see our vectors as column vectors, unless otherwise stated.

Next, we need to define *constraints* that state the conditions for a plan to be *valid*. In this context, a valid plan is a plan that does not utilise more than the amount of resources b_i, $\forall i \in I$, available. This can be expressed as the collection (one for each $i \in I$) of linear inequalities

$$\text{s.t.:} \sum_{j \in J} a_{ij} x_j \leq b_i, \forall i \in I \Rightarrow Ax \leq b,$$

where a_{ij} are the components of the $M \times N$ matrix A and $b = [b_1, \ldots, b_M]^\top$. Furthermore, we also must require that $x_i \geq 0, \forall i \in I$.

Combining the above, we obtain the generic formulation that will be used throughout this text to represent linear programming models:

$$\text{max. } c^\top x \tag{1.2}$$
$$\text{s.t.: } Ax \leq b \tag{1.3}$$
$$x \geq 0. \tag{1.4}$$

Illustrative example: the paint factory problem

Let us work on a more specific example that will be useful for illustrating some important concepts related to the geometry of linear programming problems.

A paint factory produces *exterior* and *interior paint* from raw materials *M1* and *M2*. The *maximum demand* for interior paint is 2 tons/day. Moreover, the amount of interior paint produced *cannot exceed* that of exterior paint by more than 1 ton/day.

Our goal is to determine optimal paint production plan. Table 1.1 summarises the data to be considered. Notice the constraints that must be imposed to represent the daily availability of paint.

	material (ton)/paint (ton)		
	exterior paint	interior paint	daily availability (ton)
material M1	6	4	24
material M2	1	2	6
profit ($1000 /ton)	5	4	

Table 1.1: Paint factory problem data

Notice that the paint factory problem is an example of a resource allocation problem. Perhaps one aspect that is somewhat dissimilar is the constraint representing the production rules regarding the relative amounts of interior and exterior paint. Notice however, that this type of constraint also has the same format as the more straightforward resource allocation constraints.

Let x_1 be amount produced of exterior paints (in tons) and x_2 the amount of interior paints. The complete model that optimises the daily production plan of the paint factory is:

$$\text{max. } z = 5x_1 + 4x_2 \tag{1.5}$$
$$\text{s.t.: } 6x_1 + 4x_2 \leq 24 \tag{1.6}$$
$$x_1 + 2x_2 \leq 6 \tag{1.7}$$
$$x_2 - x_1 \leq 1 \tag{1.8}$$
$$x_2 \leq 2 \tag{1.9}$$
$$x_1, x_2 \geq 0 \tag{1.10}$$

Notice that paint factory model can also be *compactly represented* as in (1.2)–(1.4), where

$$c = [5, 4]^\top,\ x = [x_1, x_2]^\top,\ A = \begin{bmatrix} 6 & 4 \\ 1 & 2 \\ -1 & 1 \\ 0 & 1 \end{bmatrix},\ \text{and } b = [24, 6, 1, 2]^\top.$$

1.2.2 Transportation problem

Another important class of linear programming problems are those known as transportation problems. These problems are often modelled using the abstraction of graphs, since they consider a network of nodes and arcs through which some flow must be optimised. Transportation problems have several important characteristics that can be exploit to design specialised algorithms, the so-called *transportation simplex* method. Although we will not discuss these methods in this text, the simplex method (and its variant dual simplex) will be in the centre of our developments later on. Also, modern solvers have more and more relegated transport simplex methods in their development, as dual simplex has consistently shown to perform similarly in the context of transportation problems, despite being a far more general method.

The problem can be summarised as follows. We would like to plan production and distribution of a certain product, taking into account that the transportation cost is known (e.g., proportional to distance travelled), the factories (or source nodes) have a capacity limit and the clients (or demand nodes) have known demands. Figure 1.1 illustrate a small network with two factories, located in San Diego and Seattle, and three demand points, located in New York, Chicago, and Miami. Table 1.2 presents the data related to the problem.

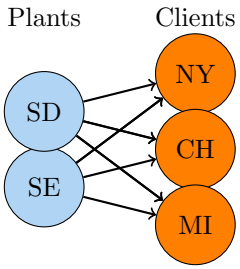


Figure 1.1: Schematic illustration of a network with two source nodes and three demand nodes

	Clients			
Factory	NY	Chicago	Miami	Capacity
Seattle	2.5	1.7	1.8	350
San Diego	3.5	1.8	1.4	600
Demands	325	300	275	-

Table 1.2: Problem data: unit transportation costs, demands and capacities

To formulate a linear programming model representing the transportation problem, let $i \in I = \{\text{Seattle}, \text{San Diego}\}$ be the index set representing factories. Similarly, let $j \in J = \{\text{New York}, \text{Chicago}, \text{Miami}\}$.

The decisions in this case are represented by x_{ij} be the amount produced in factory i and sent to client j. Such a distribution plan can then be assessed by its total transportation cost, which is given by

$$\text{min. } z = 2.5x_{11} + 1.7x_{12} + 1.8x_{13} + 3.5x_{21} + 1.9x_{22} + 1.4x_{23}.$$

The total transportation cost can be more generally represented as

$$\text{min. } z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$$

where c_{ij} is the unit transportation cost from i to j. The problem has two types of constraints that must be observed, relating to the supply capacity and demand requirements. These can be stated as the following linear constraints

$$\begin{aligned}
&x_{11} + x_{12} + x_{13} \leq 350 \text{ (capacity limit in Seattle)}\\
&x_{21} + x_{22} + x_{23} \leq 600 \text{ (capacity limit in San Diego)}\\
&x_{11} + x_{21} \geq 325 \text{ (demand in New York)}\\
&x_{12} + x_{22} \geq 300 \text{ (demand in Chicago)}\\
&x_{13} + x_{23} \geq 275 \text{ (demand in Miami).}
\end{aligned}$$

These constraints can be expressed in the more compact form

$$\sum_{j \in J} x_{ij} \leq C_i, \forall i \tag{1.11}$$

$$\sum_{i \in I} x_{ij} \geq D_j, \forall j, \tag{1.12}$$

where C_i is the production capacity of factory i and D_j is the demand of client j. Notice that the terms on the lefthand side in (1.11) accounts for the total production in each of the source nodes $i \in I$. Analogously, in constraint (1.12), the term on the left accounts for the total of the demand satisfied at the demand nodes $j \in J$. Using an optimality argument stating that any solution for which, for any $j \in J$, $\sum_{i \in I} x_{ij} > D_j$ can be improved by making $\sum_{i \in I} x_{ij} = D_j$. This show that his constraint under these conditions will always be satisfied as an equality constraint instead, and could be replaced like such.

The complete transportation model for the example above can be stated as

$$\begin{aligned}
\text{min. } z = {} & 2.5x_{11} + 1.7x_{12} + 1.8x_{13} + 3.5x_{21} + 1.9x_{22} + 1.4x_{23}\\
\text{s.t.: } & x_{11} + x_{12} + x_{13} \leq 350,\ x_{21} + x_{22} + x_{23} \leq 600\\
& x_{11} + x_{21} \geq 325,\ x_{12} + x_{22} \geq 300,\ x_{13} + x_{23} \geq 275\\
& x_{11}, \ldots, x_{23} \geq 0.
\end{aligned}$$

Or, more compactly, in the so called *algebraic (or symbolic) form*

$$\begin{aligned}
\text{min. } z = {} & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}\\
\text{s.t.: } & \sum_{j \in J} x_{ij} \leq C_i, \forall i\\
& \sum_{i \in I} x_{ij} \geq D_j, \forall j\\
& x_{ij} \geq 0, \forall i \in I, \forall j \in J.
\end{aligned}$$

One interesting aspect to notice regarding algebraic forms is that they allow to represent the main structure of the model while being independent of the instance being considered. For example, regardless of whether the instance would have 5 or 50 nodes, the algebraic formulation is the same, allowing for detaching the problem instance (in our case the 5 node network) from the model itself. Moreover, most computational tools for mathematical programming modelling (hereinafter referred to simply as modelling - like JuMP) empowers the user to define the optimisation model using this algebraic representation.

Algebraic forms are the main form in which we will specify optimisation models. This abstraction is a peculiar aspect of mathematical programming, and is perhaps one of its main features the fact that one must *formulate* models for each specific setting, which can be done in multiple ways and might have consequences for how well computationally an algorithm performs. Further in this text we will discuss this point in more detail.

1.2.3 Production planning (lot-sizing)

Production planning problems, commonly referred to as lot-sizing problems in contexts related to industrial engineering, consider problems where a planning horizon is taken into consideration. Differently from the previous examples, lot-sizing problems allow for the consideration of a time flow aspect, in which production that is made in the past can be "shifted" to a future point in time by means of inventories (i.e., stocks). Inventories are important because they allow for taking advantage of different prices at different time periods, circumventing production capacity limitations, or preparing against uncertainties in the future (e.g., uncertain demands).

The planning horizon is represented by a collection of chronologically ordered elements $t \in \{1, \ldots, T\}$ representing a set of uniformly-sized time periods (e.g., months or days). Then, let us define the decision variables p_t as the amount produced in period t and k_t the amount stored in period t, which is available for use in periods $t' > t$. These decisions are governed by two costs: P_t, $\forall t \in T$, representing the production cost in each time period t and the unit holding cost H, that is, how much it costs to hold one unit of product for one time period.

Our objective is to satisfy the demands D_t, $\forall t \in T$, at the minimum possible cost. Figure 1.2 provides a schematic representation of the process to be modelled. Notice that each node represents a material balance to be considered, that is, that at any period t, the total produced plus the amount held in inventory from the previous period $(t-1)$ must be the same as the amount used to satisfy the demand plus the amount held in inventory for the next period $(t+1)$.

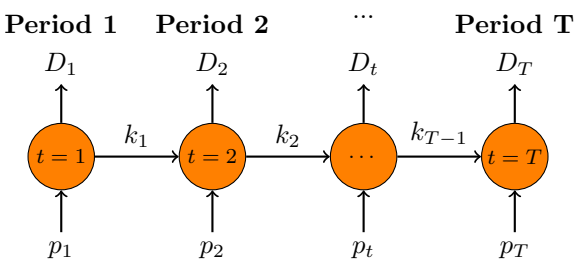


Figure 1.2: A schematic representation of the lot-sizing problem. Each node represents the material balance at each time period t.

The production planning problem can be formulated as

$$\begin{aligned} \text{min.} \quad & \sum_{t \in T} [C_t p_t + H s_t] \\ \text{s.t.:} \quad & p_t + s_{t-1} = D_t + k_t, \ \forall t \in T \\ & p_t, h_t \geq 0, \ \forall t \in T. \end{aligned}$$

A few points must be considered carefully when dealing with lot-sizing problems. First, one must carefully consider boundary condition, that is what the model is deciding in time periods $t = T$

and what is the initial initial inventory (carried from $t = 0$). While the former will be seen by the model as the "end of the world" and thus will realise that optimal inventory levels at period $|T|$ must be zero, the latter might considerable influence how much production is needed during the planning horizon T. These must be observed and handled accordingly.

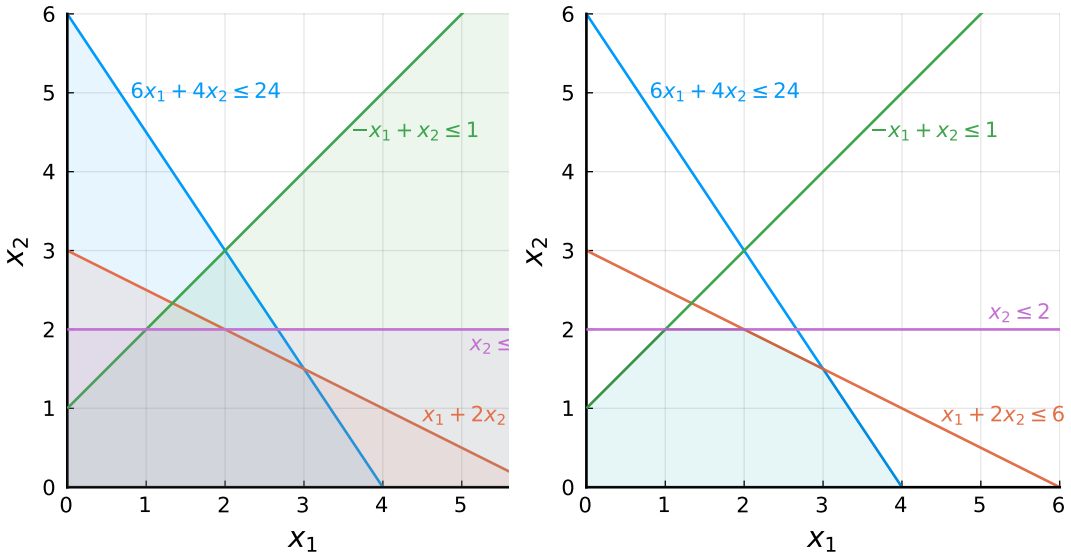
1.3 The geometry of LPs - graphical method

Let us now focus our attention to the geometry of linear programming (LP) models. As it will become evident later on, LP models have a very peculiar geometry that is exploited by one of the most widespread methods to solve them, the *simplex method*.

1.3.1 The graphical method

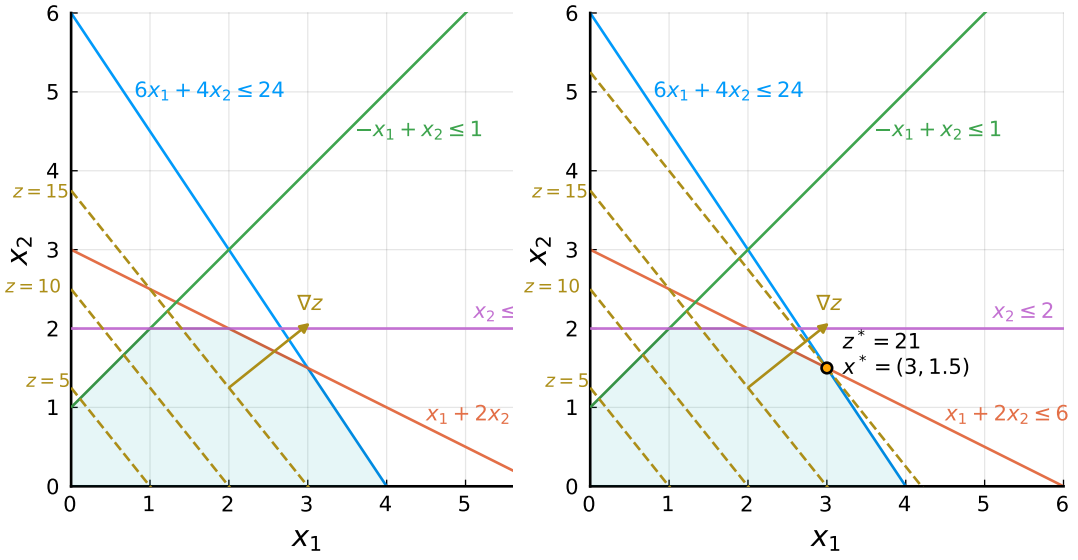
In order to create a geometric intuition, we will utilise a graphical representation of the resource allocation example (the paint factory problem). But first, recall the general LP formulation (1.2)–(1.4), where A is an $m \times n$ matrix, and b, c, and x have suitable dimensions. Let a_i be one of the m rows of A. Notice each constraint $a_i^\top x \leq b_i$ defines a closed half-space, with boundary defined by a hyperplane $a_i^\top x = b_i$, $\forall i \in I = \{1, \ldots, m\} \equiv [m]$ (we will return to these definitions in chapter 2; for now, just bear with e if these terms do not make sense to you). By plotting all of these closed half-spaces, we can see that their intersection will form the *feasible region* of the problem, that is, the (polyhedral) set of points that satisfy all constraints $Ax \leq b$.

Figure 1.3 provides a graphical representation of the feasible region of the paint factory problem.



(a) (b)

Figure 1.3: The feasible region of the paint factory problem (in Figure 1.3b), represented as the intersection of the four closed-half spaces formed by each of the constraints (as shown in Figure 1.3a). Notice how the feasible region is a polyhedral set in $\mathbb{R}^2$, as there are two decision variables (x_1 and x_2).



(a) (b)

Figure 1.4: Graphical representation of some of the level curves of the objective function $z = 5x_1 + 4x_2$. Notice that the constant gradient vector $\nabla z = (5, 4)^\top$ points to the direction in which the level curves increase in value. The optimal point is represented by $x^* = (3, 1.5)^\top$ with the furthermost level curve being that associated with the value $z^* = 21$

We can use this visual representation to find the optimal solution for the problem, that is, the point (x_1, x_2) within the feasible set such that the objective function value is minimal. For that, we must consider how the objective function $z = c^\top x$ can be represented in the (x_1, x_2)-plane. Notice that the objective function forms a hyperplane in $(x_1, x_2, z) \subset \mathbb{R}^3$, of which we can plot level curves (i.e., projections) onto the (x_1, x_2)-plane. Figure 1.4a shows the plotting of three level curves, for $z = 5, 10$, and 15. This observation provides us with a simple graphical method to find the optimal solution of linear problems. One must simply sweep the feasible region in the direction of the gradient $\nabla z = [\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}]^\top = [5, 4]^\top$ (or opposite to it, if minimising) until one last point (or edge) or contact remains, meaning that the whole of the feasible region is behind that furthermost level curve. Figure 1.4b illustrates the process of finding the optimal solution for the paint factory problem.

The graphical method is important because it allows to notice several key features that are going to be used later on when we analyse a method that can search for optimal solutions for LP problems. The first is related to the notion of active or inactive constraints. We say that a constraint is *active* if, at the optimum, the constraint is satisfied as an equality. For example, the constraints $6x_1 + 4x_2 \leq 24$ and $x_1 + 2x_2 \leq 6$ are active at the optimum $x^* = (3, 1.5)$, since $6(3) + 4(1.5) = 24$ and $3 + 2(1.5) = 6$. An active constraint indicates that the resource (or requirement) represented by that constraint is being fully depleted (or minimally satisfied).

Analogously, *inactive constraint* are constraints that are satisfied as strict inequalities at the optimum. For example, the constraint $-x_1 + x_2 \leq 1$ is inactive at the optimum, as $-(3) + 1.5 < 1$. In this case, an inactive constraint represents a resource (or requirement) that is not fully depleted (or is over satisfied).

1.3.2 Geometrical properties of LPs

One striking feature concerning the geometry of LPs that becomes evident when we analyse the graphical method is that the number of candidate solutions is not infinite, but yet, only *a finite set* of points are potential candidates for optimal solution. This is because the process of sweeping in the direction of the gradient of the (linear) objective function will, in general, lead to a unique solution that must lie on a vertex of the polyhedral feasible set. The only exceptions are either when the gradient ∇z happens to be perpendicular to a facet of the polyhedral set (and in the direction of the sweeping) or in case the sweeping direction is not bounded by some of the facets of the polyhedral set. These exceptional cases will be discussed in more detail later on, but the observation still holds.

In the graphical example (i.e., in $\mathbb{R}^2$), notice how making $n = 2$ constraints active out of $m = 4$ constraints *forms a vertex*. However, not all vertices are feasible. This allows us to devise a mechanism to describe vertices by activating n of the m constraints at a time, which we could exhaustively test and select the best. The issue however, is that the number of candidates increases *exponentially* with the number of constraints and variables of the problem, which indicates this would quickly become computationally infeasible. As we will see, it turns out that this search idea can be made considerably efficient and the underlying framework of the *simplex method*, however there are artificially engineered worst-case settings where the method does need to consider every single vertex.

The simplex method exploits the above idea to *heuristically* search for solutions by selecting n constraints to be active from the m constraints available. Starting from an initial selection of constraints to be active, it selects one inactive constraint to activate and one to deactivate in a way that improvement in the objective function can observed while feasibility is maintained. This process repeats until no improvement can be observed. When such is the case, the *geometry* of the problem guarantees *(global) optimality*. In the following chapters we will concentrate on defining algebraic objects that we will use to develop the simplex method.

Exercises

Exercise 1.1: Introduction to JuMP - part I

In the following optimisation problems use the Julia library JuMP.jl to find the optimal solution. For the two-dimensional problems use the library Plots.jl to illustrate the feasible region and the optimal point.

(a)
$$\begin{aligned} \max \quad & x_1 + 2x_2 + 5x_3 \\ \text{s.t.} \quad & \\ & x_1 - x_2 - 3x_3 \geq 5 \\ & x_1 + 3x_2 - 7x_3 \leq 10 \\ & x_1 \leq 10 \\ & x_i \geq 0 \end{aligned}$$

(b)

$$\begin{array}{ll} \max & 2x_1 + 4x_2 \\ \text{s.t.} & \\ & \begin{array}{rcl} x_1 + x_2 & \leq & 5 \\ -x_1 + 3x_2 & \leq & 1 \\ x_1 & \leq & 5 \\ x_2 & \leq & 5 \end{array} \\ & x_1, x_2 \geq 0 \end{array}$$

Exercise 1.2: Introduction to JuMP - part II

In the following optimisation problems use the Julia library JuMP.jl to find the optimal solution. For the two-dimensional problems use the library Plots.jl to illustrate the feasible region and the optimal point.

(a)

$$\begin{array}{ll} \min & -5x_1 + 10x_2 + x_3 + 2000x_4 \\ \text{s.t.} & \\ & \begin{array}{rcl} x_1 - x_2 & \leq & 1500 \\ 4x_2 - x_3 & \leq & 5000x_4 \\ x_1 + 3x_2 & \geq & 1000 \\ x_1 & \leq & 10000 \end{array} \\ & x_1 \geq 0,\ x_2 \text{ free},\ x_3 \leq 0,\ x_4 \text{ Binary} \end{array}$$

(b)

$$\begin{array}{ll} \max & 5x_1 + 3x_2 \\ \text{s.t.} & \\ & \begin{array}{rcl} x_1 + 5x_2 & \leq & 3 \\ 3x_1 - x_2 & \leq & 5 \\ x_1 & \leq & 2 \\ x_2 & \leq & 30 \end{array} \\ & x_1, x_2 \geq 0 \end{array}$$

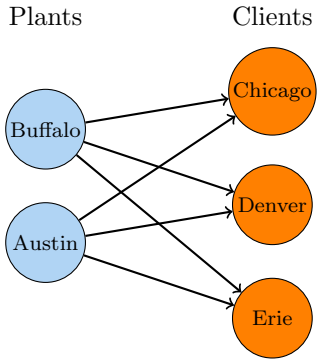
Exercise 1.3: Transportation problem

Source: Julia Programming for Operations Research 2nd edition (Online).
Available at: http://www.chkwon.net/julia - published by Changhyun Kwon

Consider the following network where the combined supplies from the Austin and Buffalo nodes need to meet the demands coming from Chicago, Denver, and Erie. The supplies available are represented by S and the demands by D, the costs of transporting in each arc connecting supply and demand nodes are shown as c.

	Clients			
Factory	Chicago	Denver	Eire	
Buffalo	4	9	8	25
Austin	10	7	9	600
Demands	15	12	13	-

Table 1.3: Problem data: unit transportation costs, demands and capacities



Solve the transportation problem to the minimum cost.

Exercise 1.4: Capacitated transportation

The Finnish company Next is involved with a logistics problem in which used oils serve as raw material to form a class of renewable diesel. The supply chain team need to organise, to a minimal cost, the acquisition of two types of used oils (products p1 and p2) from three suppliers (supply nodes s1, s2, and s3) to feed the line of three of their used oils processing factories (demand nodes d1, d2, and d3). As the used oils are the byproduct of the suppliers core activities, the only requirement is that Next need to fetch the amount of oil and pay the transportation costs alone.

The used oils have specific conditioning and handling requirements so transportation costs varies between p1 and p2. Additionally, not all the routes (arcs) between suppliers and factories are available as some distances are not economically feasible. Table 1.4a bring the volume requirement of the two types of oil from each supply and demand node, table 1.4b show the transportation costs for each oil type per L and the arc's capacity, arcs with "-" for costs are not available as transportation routes.

node	**p1**	**p2**
s1 / d1	80 / 60	400 / 300
s2 / d2	200 / 100	1500 / 1000
s3 / d3	200 / 200	300 / 500

(a) Supplies availability and demand per oil type [in L]

	d1 **p1/p2 (cap)**	**d2** **p1/p2 (cap)**	**d3** **p1/p2 (cap)**
s1	5/- (∞)	5/18 (300)	-/- (0)
s2	8/15 (300)	9/12 (700)	7/14 (600)
s3	-/- (0)	10/20 (∞)	8/- (∞)

(b) Arcs costs per oil type [in € per L] and arcs' capacities [in L]

Table 1.4: Supply chain data

Solve the transportation problem to the minimum cost.

Exercise 1.5: The farmer's problem

Source: Section 1.1 of the book Birge, J. R., & Louveaux, F. (2011). Introduction to Stochastic Programming. New York, NY: Springer New York.

Consider a European farmer who specializes in raising wheat, corn, and sugar beets on his 500 acres of land. During the winter, "he" wants to decide how much land to devote to each crop. (We refer to the farmer as he for convenience and not to imply anything about the gender of European farmers.)

The farmer knows that at least 200 tons (T) of wheat and 240 T of corn are needed for cattle feed. These amounts can be raised on the farm or bought from a wholesaler. Any production in excess of the feeding requirement would be sold. Over the last decade, mean selling prices have been $ 170 and $ 150 per ton of wheat and corn, respectively. The purchase prices are 40 % more than this due to the wholesaler's margin and transportation costs.

Another profitable crop is sugar beet, which he expects to sell at $36/T; however, the European Commission imposes a quota on sugar beet production. Any amount in excess of the quota can be sold only at $10/T. The farmer's quota for next year is 6000 T.

Based on past experience, the farmer knows that the mean yield on his land is roughly 2.5 T, 3 T, and 20 T per acre for wheat, corn, and sugar beets, respectively.

Based on the data, build up a model to help the farmer allocate his farming area to each crop and how much to sell/buy of wheat, corn, and sugar beets considering the following cases.

(a) The predictions are 100% accurate and the mean yields are the only realization possible.

(b) There are three possible equiprobable scenarios (i.e, each one with a probability equal to $\frac{1}{3}$): a good, fair, and bad weather scenario. In the good weather, the yield is 20% better than the yield expected whereas in the bad weather scenario it is reduced 20% of the mean yield. In the regular weather scenario, the yield for each crop keeps the historical mean - 2.5T/acre, 3T/acre, and 20T/acre for wheat, corn, and sugar beets, respectively.

(c) What happens if we assume the same scenarios as item (b) but with probabilities 25%, 25%, and 50% for good, fair, and bad weather, respectively? How the production plan changes and why?

Exercise 1.6: Factory planning

Adapted from: Model Building in Mathematical Programming 5^{nd} edition. H. Paul Williams – London School of Economics, Wiley

An engineering factory makes seven products (PROD 1 to PROD 7) on the following machines: four grinders, two vertical drills, three horizontal drills, one borer and one planer. Each product yields a certain contribution to profit (defined as £/unit selling price minus cost of raw materials). These quantities (in £/unit) together with the unit production times (hours) required on each process are given in Table 1.5. A dash indicates that a product does not require a process. There are also marketing demand limitations on each product in each month. These are given in the Table 1.6.

	PROD1	**PROD2**	**PROD3**	**PROD4**	**PROD5**	**PROD6**	**PROD7**
Profit	10	6	8	4	11	9	3
Grinding	1.5	2.1	–	–	0.9	0.6	1.5
Vert. drilling	0.3	0.6	–	0.9	–	1.8	–
Horiz. drilling	0.6	–	2.4	–	–	–	1.8
Boring	0.15	0.09	–	0.21	0.3	–	0.24
Planing	–	–	0.03	–	0.15	–	0.15

Table 1.5: Products yield

	PROD1	**PROD2**	**PROD3**	**PROD4**	**PROD5**	**PROD6**	**PROD7**
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

Table 1.6: Maximum demand

It is possible to store up to 100 of each product at a time at a cost of £0.5 per unit per month. There are no stocks at present, but it is desired to have a stock of 50 of each type of product at the end of June. The factory works a six days a week with two shifts of 8 h each day. No sequencing problems need to be considered.

When and what should the factory make in order to maximise the total profit? Recommend any price increases and the value of acquiring any new machines. Note: it may be assumed that each month consists of only 24 working days.

Chapter 2

Basics of Linear Algebra

2.1 Basics of linear problems

As we have seen in the previous chapter, the feasible region of a linear programming problem can be represented as

$$Ax \leq b, \tag{2.1}$$

where A is a $m \times n$ matrix, x is a n-dimensional column vector (or more compactly, $x \in \mathbb{R}^n$), and b is an m-dimensional column vector ($b \in \mathbb{R}^m$). Notice that $\leq$ is considered component-wise.

Before introducing the simplex method, let us first revisit a few key elements and operations that we will use in the process. The first of them is presented in Definition 2.1.

Definition 2.1 (Matrix inversion)**.** *Let A be a square $n \times n$ matrix. A^{-1} is the* inverse matrix *of A if it exists and $AA^{-1} = I$, where I is the identity matrix.*

Matrix inversion is the "kingpin" of linear (and nonlinear) optimisation. As we will see later on, being able to perform efficient matrix inversion operations (in reality, operations that are equivalent to matrix inversion but that can exploit the matrix structure to be made faster) is of utmost importance for developing a linear optimisation solver.

Another important concept is the notion of *linear independence*. We formally state when a collection of vectors is said to be linearly independent (or dependent) in Definition 2.2.

Definition 2.2 (Linearly Independent vectors)**.** *The vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ are* linearly dependent *if there exist real numbers $\{a_i\}_{i=1}^k$ with $a_i \neq 0$ for at least one $i \in \{1, \ldots, k\}$ such that*

$$\sum_{i=1}^{k} a_i x_i = 0;$$

otherwise, $\{x_i\}_{i=1}^k$ are linearly independent.

In essence, for a collection of vectors to be linearly independent it must be so that none of the vectors in the collection can be expressed as a combination (that is multiplying the vectors by nonzero scalars and adding them) of the others. This is simpler to see in $\mathbb{R}^2$. Two vectors are linearly independent if one cannot obtain one by multiplying the other by a constant, which, effectively, means that they are not parallel. If the two vectors are not parallel, then one of them must have a component in a direction that the other cannot achieve. The same holds for any n-dimensional space. Also, that is why one can only have up to n independent vectors in $\mathbb{R}^n$. Figure 2.1 illustrates this effect.

Theorem 2.3 summarises results that we will utilise in the upcoming developments. These are classical results from linear algebra and are thus provided without a proof.

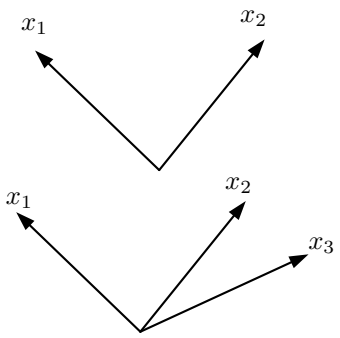


Figure 2.1: Linearly independent (top) and dependent (bottom) vectors in $\mathbb{R}^2$

Theorem 2.3 (Inverses, linear independence, and solving $Ax = b$)**.** *Let A be a $m \times m$ matrix. Then, the following statements are equivalent:*

1. *A is invertible*
2. *$A^\top$ is invertible*
3. *The determinant of A is nonzero*
4. *The rows of A are linearly independent*
5. *The columns of A are linearly independent*
6. *For every $b \in \mathbb{R}^m$, the linear system $Ax = b$ has a unique solution*
7. *There exists some $b \in \mathbb{R}^m$ such that $Ax = b$ has a unique solution.*

Notice that Theorem 2.3 establishes important relationships between the geometry of the matrix A (its rows and columns) and consequences it has to our ability to calculate its inverse A^{-1} and, consequently, solve the system $Ax = b$, to which the solution is obtained as $x = A^{-1}b$. This will turn out to be most important operation in the simplex method.

2.1.1 Subspaces and bases

Let us define some objects that we will frequently refer to. The first of them is the notion of a *subspace*. A subspace of $\mathbb{R}^n$ is a set comprising all linear combinations of its own elements. Specifically, if S is a subspace, then

$$S = \{ax + by : x, y \in S; a, b \in \mathbb{R}\}.$$

A related concept is the notion of a *span*. A span of a collection of vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ is the subspace of $\mathbb{R}^n$ formed by all linear combinations of such vectors, i.e.,

$$\mathbf{span}(x_1, \ldots, x_k) = \left\{ y = \sum_{i=1}^{k} a_i x_i : a_i \in \mathbb{R}, i \in \{1, \ldots, k\} \right\}.$$

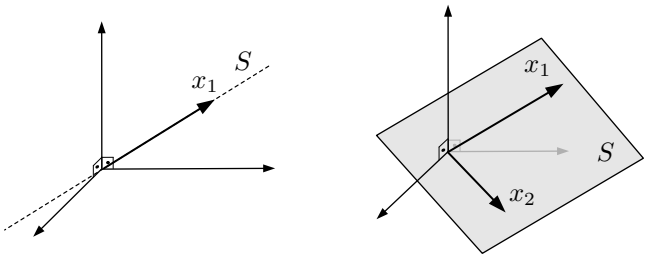


Figure 2.2: One- (left) and two-dimensional subspaces (right) in $\mathbb{R}^3$

Notice how the two concepts are related: subspaces can be characterised by a collection of vectors to which the span gives the said subspace. In other words, the span of a set of vectors is the subspace formed by all points we can represent by some linear combination of these vectors.

The missing part in this is the notion of a *basis*. A *basis* of the subspace $S \subseteq \mathbb{R}^n$ is a collection of vectors $\{x_i\}_{i=1}^k \in \mathbb{R}^n$ that are linearly independent such that $\mathbf{span}(x_1, \ldots, x_k) = S$.

Notice that a basis is a "minimal" set of vectors that form a subspace. You can think of it in light of the definition of linearly independent vectors; if a vector is linearly dependent to the others, it is not needed for characterising the subspace the vectors span, since it can be represented by a linear combination of the other vectors (and thus is in the subspace formed by the span of the other vectors).

The above leads us to some important realisations:

1. All bases of a given subspace S have the same dimension. Any extra vector would be linearly dependent to those vectors that span S. In that case, we say that the subspace has size (or dimension) k, the number of linearly independent vectors forming the basis of the subspace.

2. If the subspace $S \subset \mathbb{R}^n$ is formed by a basis of size $m < n$, we say that S is a proper subspace, because it is not the whole $\mathbb{R}^n$ itself, but a space contained within $\mathbb{R}^n$. For example, two linearly independent vectors form (i.e., span) a hyperplane in $\mathbb{R}^3$; this hyperplane is a proper subspace since $m = 2 < 3 = n$.

3. If a proper subspace has dimension $m < n$, then it means that there are $n-m$ directions in $\mathbb{R}^n$ that are perpendicular to the subspace and to each other. That is, there are nonzero vectors a_i that are orthogonal to each other and to S. Or, equivalently, $a_i^\top x = 0$ for $i = n-m+1, ..., n$. Referring to the $\mathbb{R}^3$, if $m = 2$, then there is a third direction that is perpendicular to (or not in) S. Figure 2.2 illustrates this.

Theorem 2.4 builds upon the previous points to guarantee the existence of bases and propose a procedure to form them.

Theorem 2.4 (Forming bases from linearly independent vectors)**.** *Suppose that* $S = \mathbf{span}(x_1, \ldots, x_k)$ *has dimension* $m \leq k$*. Then*

1. *There exists a basis of* S *consisting of* m *of the vectors* $x_1, \ldots, x_k$*.*

2. *If* $k' \leq m$ *and* $x_1, \ldots, x_{k'} \in S$ *are linearly independent, we can form a basis for* S *by starting with* $x_1, \ldots, x_{k'}$ *and choosing* $m - k'$ *additional vectors from* $x_1, \ldots, x_k$*.*

Our interest in subspaces and bases spans from (pun intended!) their usefulness in explaining how the simplex method works under a purely algebraic (as opposed to geometric) perspective. For now, we can use the opportunity to define some "famous" subspaces which will often appear in our derivations.

Let A be a $m \times n$ matrix as before. The *column space* of A consists of the subspace spanned by the n columns of A and has dimension m (recall that each column has as many components as the number of rows and is thus a m-dimensional vector). Likewise, the *row space* of A is the subspace in $\mathbb{R}^n$ spanned by the rows of A. Finally, the *null space* of A, often denoted as $\mathbf{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\}$, consist of the vectors that are perpendicular to the row space of A.

One important notion related to those subspaces is that of their size. Both the row and the column space have the same size, and that size is the *rank* of A. If A is *full rank*, than it means that

$$\mathbf{rank}(A) = \min\{m, n\}.$$

Finally, the size of the null space of A is given $n - \mathbf{rank}(A)$, which is in line with Theorem 2.4.

2.1.2 Affine subspaces

A related concept is that of an *affine subspace*. Differently from linear subspaces (to which we have been referring to simply as subspaces), affine subspaces encode some form of translation, such as

$$S = S_0 + x_0 = \{x + x_0 : x \in S_0\}.$$

Affine subspaces are not subspaces, because they do not contain the origin (recall that the definition of subspaces allows for a and b to be zero). Nevertheless, S has the *same dimension* as S_0.

Affine subspaces give a framework for representing linear programming problems algebraically. Specifically, let A be a $m \times n$ matrix with $m < n$ (which will always be the case in linear programming models in this form as we will see) and b a m-dimensional vector. Then, let

$$S = \{x \in \mathbb{R}^n : Ax = b\}. \tag{2.2}$$

As we will see, the feasible set of any linear programming problem can be represented as an equality-constrained equivalent of the form of (2.2) by means of adding slack variables to the inequality constraints. Now, assume that $x_0 \in \mathbb{R}^n$ is such that $Ax_0 = b$. Then, we have that

$$Ax = Ax_0 = b \Rightarrow A(x - x_0) = 0.$$

Thus, $x \in S$ if and only if the vector $(x - x_0)$ belongs to $\mathbf{null}(A)$, the nullspace of A. Notice that the feasible region S can be also defined as

$$S = \{x + x_0 : x \in \mathbf{null}(A)\},$$

being thus an affine subspace with dimension $n - m$, if A has m linearly independent rows (i.e., $\mathbf{rank}(A) = m$). This will have important implications in the way we can define multiple basis from the n vectors in the column space by choosing m to be removed and form a basis to $\mathbf{null}(A)$, and what this process means geometrically.

Figure 2.3 illustrates this concept for a single-row matrix a. For multiple rows, one would see S as being represented by the intersection of multiple hyperplanes.

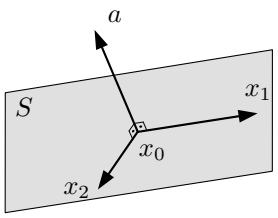


Figure 2.3: The affine subspace S generated by x_0 and $\mathbf{null}(a)$

2.2 Convex polyhedral set

The feasible region of any linear programming problem is a convex polyhedral set, which we will simply refer to as a polyhedral set. That is because we are interested in polyhedral sets that are formed by an intersection of a finite number of half-spaces, and can thus only be convex (as we will see in a moment), creating redundancy in our context, but maybe some confusion overall.

2.2.1 Hyperplanes, half-spaces and polyhedral sets

Definition 2.5 formally states the structure we refer to as polyhedral sets.

Definition 2.5 (Polyhedral set)**.** *A polyhedral set is a set that can be described as*

$$S = \{x \in \mathbb{R}^n : Ax \geq b\},$$

where A is an $m \times n$ matrix and b is a m-vector.

One important thing to notice is that polyhedral sets, as defined in Definition 2.5, as formed by the intersection multiple half-spaces. Specifically, let $\{a_i\}_{i=1}^m$ be the rows of A. Then, the set S can be described as

$$S = \left\{x \in \mathbb{R}^n : a_i^\top x \geq b_i, i = 1, \ldots, m\right\}, \tag{2.3}$$

which represents exactly the intersection of the half-spaces $a_i^\top x \geq b_i$. Furthermore, notice that the hyperplanes $a_i^\top x = b_i$, $\forall i \in \{1, \ldots, m\}$, are the boundaries of each hyperplane, and thus describe one of the faces of the polyhedral set, which is generally given by $\left\{x \in S : a_i^\top x \geq b_i\right\}$ for a given face $i \in \{1, \ldots, m\}$. Figure 2.4 illustrates a hyperplane forming two half-spaces (also polyhedral sets) and how the intersection of five half-spaces form a (bounded) polyhedral set.

You might finds authors referring to bounded polyhedral sets as polytopes, though this is not used consistently across references, sometimes with switched meanings (for example, using polytope to refer to a set defined as in Definition 2.5 and using polyhedron to refer to a bounded version of S). In this text, we will only use the term polyhedral set to refer to sets defined as in Definition 2.5 and use the term bounded whenever applicable.

2.2.2 Convexity of polyhedral sets

As will see in more details in Part 2 of this text, convexity plays a crucial role in optimisation, being the "watershed" between easy and hard optimisation problems. One of the main reasons why we can solve challenging linear programming problems is due to the inherent convexity of polyhedral sets.

Let us first define the notion of convexity for sets, which is stated in Definition 2.6

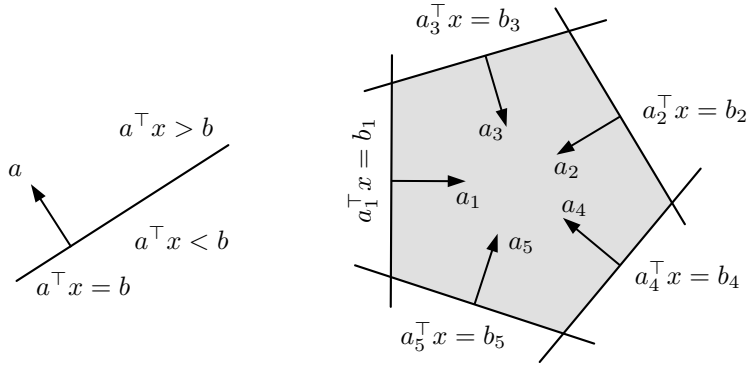


Figure 2.4: A hyperplane and its respective halfspaces (left) and the polyhedral set $\left\{x \in \mathbb{R}^2 : a_i^x \leq b_i, i = 1, \ldots, 5\right\}$ (right).

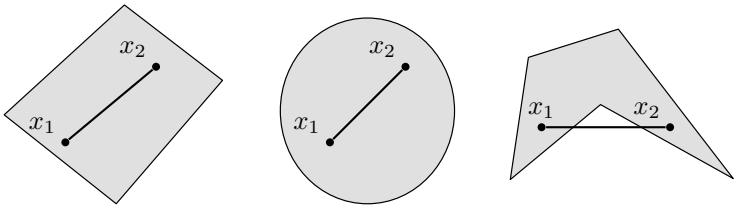


Figure 2.5: Two convex sets (left and middle) and one nonconvex set (right)

Definition 2.6 (Convex set)**.** *A set $S \subseteq \mathbb{R}^n$ is convex if, for any $x_1, x_2 \in S$ and any $\lambda \in [0, 1]$, we have that $\overline{x} = \lambda x_1 + (1 - \lambda)x_2 \in S$.*

Definition 2.6 leads to a simple geometrical intuition: for a set to be convex, the line segment connecting any two points within the set must lie within the set. This is illustrated in Figure 2.5.

Associated with the notion of convex sets are two important elements we will refer to later, when we discuss linear problems that embed *integrality requirements*. The first is the notion of a convex combination, which is already contained in Definition 2.6, but can be generalised for an arbitrary number of points. The second consists of *convex hulls*, which are sets formed by combining the convex combinations of all elements within a given set. As one might suspect, convex hulls are always convex sets, regardless whether the original set from which the points are drawn from is convex or not. These are formalised in Definition 2.7 and illustrated in Figure 2.6.

Definition 2.7 (Convex combinations and convex hulls)**.** *Let $x_1, \ldots, x_k \in \mathbb{R}^n$ and $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ such that $\lambda_i \geq 0$ for $i = 1, \ldots, k$ and $\sum_{i=1}^k \lambda_i = 1$. Then*

1. *$x = \sum_{i=1}^k \lambda_i x_i$ is a* convex combination *of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$.*
2. *The* convex hull *of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$, denoted $\mathbf{conv}(x_1, \ldots, x_k)$, is the set of all convex combinations of $\{x_i\}_{i=1}^k \in \mathbb{R}^n$.*

We are now ready to state the result that guarantees the convexity of polyhedral sets of the form

$$S = \{x \in \mathbb{R}^n : Ax \leq b\}.$$

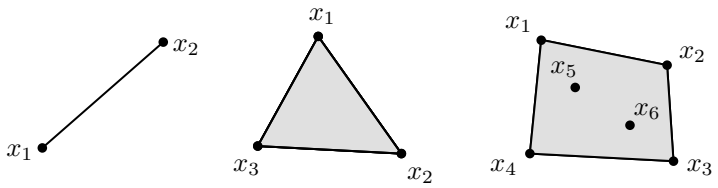


Figure 2.6: The convex hull of two points is the line segment connecting them (left); The convex hull of three (centre) and six (right) points in $\mathbb{R}^2$

Theorem 2.8 (Convexity of polyhedral sets)**.** *The following statements are true:*

1. *The intersection of convex sets is convex*
2. *Every polyhedral set is a convex set*
3. *A convex combination of a finite number of elements of a convex set also belongs to that set*
4. *The convex hull of a finite number of elements is a convex set.*

Proof. We provide proofs to each of the statements individually.

1. Let S_i, for $i \in I$, be convex sets and suppose that $x, y \in \bigcap_{i\in I} S_i$. Let $\lambda \in [0,1]$. Since S_i are convex and $x, y \in S_i$ for all $i \in I$, $\lambda x + (1-\lambda)y \in S_i$ for all $i \in I$ and, thus, $\lambda x + (1-\lambda)y \in \bigcap_{i\in I} S_i$.

2. Let $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Let $x, y \in \mathbb{R}^n$, such that $a^\top x \geq b$ and $a^\top y \geq b$. Let $\lambda \in [0,1]$. Then $a^\top(\lambda x + (1-\lambda)y) \geq \lambda b + (1-\lambda)b = b$, showing that half-spaces are convex. The result follows from combining this with (1).

3. By induction. Let S be a convex set and assume that the convex combination of $x_1, \ldots, x_k \in S$ also belongs to S. Consider $k+1$ elements $x_1, \ldots, x_{k+1} \in S$ and $\lambda_1, \ldots, \lambda_{k+1}$ with $\lambda_i \in [0,1]$ for $i = 1, \ldots, k+1$ and $\sum_{i=1}^{k+1} \lambda_i = 1$ and $\lambda_{k+1} \neq 1$ (without loss of generality). Then

$$\sum_{i=1}^{k+1} \lambda_i x_i = \lambda_{k+1} x_{k+1} + (1-\lambda_{k+1}) \sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} x_i. \tag{2.4}$$

   Notice that $\sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} = 1$. Thus, using the induction hypothesis, $\sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} x_i \in S$. Considering that S is convex and using (2.4), we conclude that $\sum_{i=1}^{k+1} \lambda_{k+1} x_{k+1} \in S$, completing the induction.

4. Let $S = \mathbf{conv}(x_1, \ldots, x_k)$. Let $y = \sum_{i=1}^{k} \alpha_i x_i$ and $z = \sum_{i=1}^{k} \beta_i x_i$ be such that $y, z \in S$, $\alpha_i, \beta_i \geq 0$, and $\sum_{i=1}^{k} \alpha_i = \sum_{i=1}^{k} \beta_i = 1$. Let $\lambda \in [0,1]$. Then

$$\lambda y + (1-\lambda)z = \lambda \sum_{i=1}^{k} \alpha_i x_i + (1-\lambda) \sum_{i=1}^{k} \beta_i x_i = \sum_{i=1}^{k} (\lambda\alpha_i + (1-\lambda)\beta_i) x_i. \tag{2.5}$$

   Since $\sum_{i=1}^{k} \lambda\alpha_i + (1-\lambda)\beta_i = 1$ and $\lambda\alpha_i + (1-\lambda)\beta_i \geq 0$ for $i = 1, \ldots, k$, $\lambda y + (1-\lambda)z$ is a convex combination of $x_1, \ldots, x_k$ and, thus, $\lambda y + (1-\lambda)z \in S$, showing the convexity of S. □

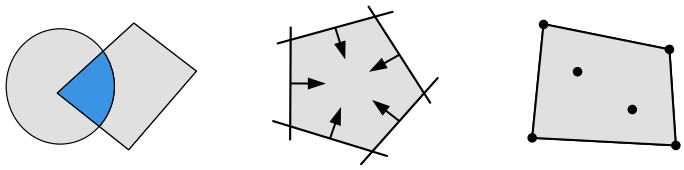


Figure 2.7: Illustration of statement 1 (left), 2 (centre), and 3 and 4 (right)

Figure 2.7 illustrates some of the statements represented in the proof. For example, the intersection of the convex sets is always a convex set. One should notice however that the same does not apply to the union of convex sets. Notice that statement 2 proves that polyhedral sets as defined according to Definition 2.5 are convex. Finally the third figure on the right illustrates the convex hull of four points as a convex polyhedral set containing the lines connecting any two points within the set.

We will halt our discussion about convexity for now and return to it in deeper details in Part 2. As it will become clearer then, the presence of convexity (which is a given in the context of linear programming, as we have just seen) is what allows us to conclude that the solutions returned by our optimisation algorithms are indeed optimal for the problem at hand.

2.3 Extreme points, vertices, and basic feasible solutions

Now we focus on the algebraic representation of the most relevant geometric elements in the optimisation of linear programming problems. As we have seen in the graphical example in the previous chapter, the optimum of linear programming problems is generally located at the vertices of the feasible set. Furthermore, such vertices are formed by the intersection of n constraints (in n-dimensional space, which comprises constraints that are active (or satisfied at the boundary of the half-space of said constraints).

First, let us formally define the notions of vertex and extreme point. Though in general these can refer to different objects, we will see that in the case of linear programming problems, if a point is a vertex, then it is an extreme point as well, being the converse also true.

Definition 2.9 (Vertex)**.** *Let P be a convex polyhedral set. The vector $x \in P$ is a vertex of P if there exists some c such that $c^\top x < c^\top y$ for all $y \in P$ with $y \neq x$.*

Definition 2.10 (Extreme points)**.** *Let P be a convex polyhedral set. The vector $x \in P$ is an extreme point of P if there are no two vectors $y, z \in P$ (different than x) such that $x = \lambda y + (1-\lambda)z$, for any $\lambda \in [0, 1]$.*

Figure 2.8 provides an illustration of the Definitions 2.9 and 2.10. Notice that, while the definition of a vertex involves an additional hyperplane that, once placed on a vertex point, strictly contains the whole polyhedral set in one of the half-spaces it defines, with the exception of the vertex itself. On the other hand, the definition of an extreme point only relies on convex combinations of elements in the set itself.

Now we focus on the description of active constraints under an algebraic standpoint. For that, let us first generalise our setting by considering all possible types of linear constraints. That is, let us

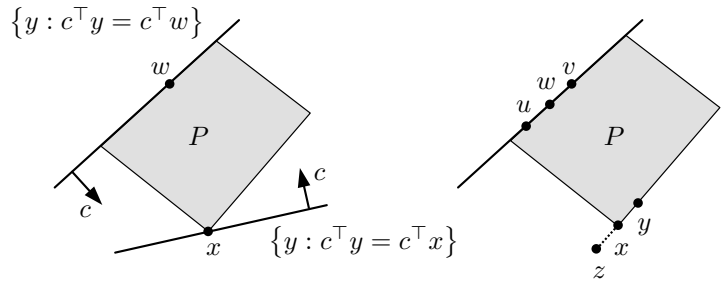


Figure 2.8: Representation of a vertex (left) and a extreme point (right)

consider the convex polyhedral set $P \subset \mathbb{R}^n$, formed by the set of inequalities and equalities:

$$a_i^\top x \geq b, i \in M_1,$$
$$a_i^\top x \leq b, i \in M_2,$$
$$a_i^\top x = b, i \in M_3.$$

Definition 2.11 formalises the notion of active constraints. This is illustrated in Figure 2.9, where the polyhedral set $P = \{x \in \mathbb{R}^3 : x_1 + x_2 + x_3 = 1, x_i \geq 0, i = 1, 2, 3\}$ is represented. Notice that, while points A, B, C and D have 3 active constraints, E only has 2 active constraints ($x_2 = 0$ and $x_1 + x_2 + x_3 = 1$).

Definition 2.11 (Active (or binding) constraints)**.** *If a vector $\overline{x}$ satisfies $a_i^\top \overline{x} = b_i$ for some $i \in M_1, M_2$, or M_3, we say that the corresponding constraints are active (or binding).*

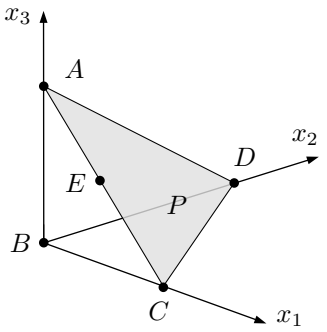


Figure 2.9: Representation of P in $\mathbb{R}^3$.

Theorem 2.12 sows a thread between having a collection of active constraints forming a vertex and being able to describe it as a basis of a subspace that is formed by the vectors a_i that form these constraints. This link is what will allow us to characterise vertices by their forming active constraints.

Theorem 2.12 (Properties of active constraints)**.** *Let $\overline{x} \in \mathbb{R}^n$ and $I = \{i \in M_1 \cup M_2 \cup M_3 \mid a_i^\top \overline{x} = b_i\}$. Then, the following are equivalent:*

1. *There exists n vectors in $\{a_i\}_{i \in I}$ that are linearly independent.*

2. *The* $\mathbf{span}(\{a_i\}_{i\in I})$ *spans* $\mathbb{R}^n$*. That is, every* $x \in \mathbb{R}^n$ *can be expressed as a linear combination of* $\{a_i\}_{i\in I}$*.*

3. *The system of equations* $\left\{a_i^\top x = b_i\right\}_{i\in I}$ *has a unique solution.*

Proof. Suppose that $\{a_i\}_{i\in I}$ spans $\mathbb{R}^n$, implying that the $\mathbf{span}(\{a_i\}_{i\in I})$ has dimension n. By Theorem 2.4 (part 1), n of these vectors form a basis for $\mathbb{R}^n$ and are, thus, linearly independent. Moreover, they must span $\mathbb{R}^n$ and therefore every $x \in \mathbb{R}^n$ can be expressed as a combination of $\{a_i\}_{i\in I}$. This connects (1) and (2).

Assume that the system of equations $\left\{a_i^\top x = b_i\right\}_{i\in I}$ has multiple solutions, say x_1 and x_2. Then, the nonzero vector $d = x_1 - x_2$ satisfies $a_i^\top d = 0$ for all $i \in I$. As d is orthogonal to every a_i, $i \in I$, d cannot be expressed as a combination of $\{a_i\}_{i\in I}$ and, thus, $\{a_i\}_{i\in I}$ do not span $\mathbb{R}^n$.

Conversely, if $\{a_i\}_{i\in I}$ do not span $\mathbb{R}^n$, choose $d \in \mathbb{R}^n$ that is orthogonal to $\mathbf{span}(\{a_i\}_{i\in I})$. If x satisfies $\left\{a_i^\top x = b_i\right\}_{i\in I}$, so does $\left\{a_i^\top (x + d) = b_i\right\}_{i\in I}$, thus yielding multiple solutions. This connects (2) and (3). □

Notice that Theorem 2.12 implies that there are (at least) n *active constraints* (a_i) that are *linearly independent* at $\overline{x}$. This is the reason why we will refer to $\overline{x}$ and any vertex forming solution a *basic solution*, of which we will be interested in those that are feasible, i.e., that satisfy all constraints $i \in M_1 \cup M_2 \cup M_3$. Definition 2.13 provides a formal definition of these concepts.

Definition 2.13 (Basic feasible solution (BFS))**.** *Consider a convex polyhedral set* $P \subset \mathbb{R}^n$ *defined by linear equality and inequality constraints, and let* $\overline{x} \in \mathbb{R}^n$*.*

1. $\overline{x}$ *is a* basic solution *if*

   (a) *All equality constraints are active and,*

   (b) *Out of the constraints active at* $\overline{x}$*,* n *of them are linearly independent.*

2. *if* $\overline{x}$ *is a basic solution satisfying all constraints, we say* $\overline{x}$ *is a basic feasible solution.*

Figure 2.10 provides an illustration of the notion of basic solutions, and show how only a subset of the basic solutions are feasible. As one might infer, these will be the points of interest in out future developments, as these are the candidates for optimal solution.

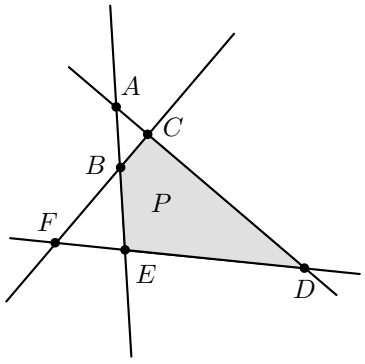


Figure 2.10: Points A to F are basic solutions; B,C,D, and E are BFS.

We finalise stating the main result of this chapter, which formally confirms the intuition we developed so far. That is, for convex polyhedral sets, the notion of vertices and extreme points coincide and these points can be represented as basic feasible solutions. This is precisely the link that allows to treat the feasible region of linear programming problems under a purely algebraic characterisation of the candidates for optimal solutions, those described uniquely by a subset of constraints of the problem.

Theorem 2.14 (BFS, extreme points and vertices)**.** *Let $P \subset \mathbb{R}^n$ be a convex polyhedral set and let $\overline{x} \in P$. Then, the following are equivalent*

$$\overline{x} \text{ is a vertex} \iff \overline{x} \text{ is an extreme point} \iff \overline{x} \text{ is a BFS.}$$

Proof. Let $P = \{x \in \mathbb{R}^n \mid a_i^\top x \geq b_i, i \in M_1, a_i^\top x = b_i, i \in M_2\}$, and $I = \{i \in M_1 \cup M_2 \mid a_i^\top x = b_i\}$.

1. (Vertex $\Rightarrow$ Extreme point) Suppose $\overline{x}$ is a vertex. Then, there exists some $c \in \mathbb{R}^n$ such that $c^\top \overline{x} < c^\top x$, for every $x \in P$ with $x \neq \overline{x}$ (cf. Definition 2.9). Take $y, z \in P$ with $y, z \neq \overline{x}$. Thus $c^\top \overline{x} < c^\top y$ and $c^\top \overline{x} < c^\top z$. For $\lambda \in [0,1]$, $c^\top \overline{x} < c^\top(\lambda y + (1-\lambda)z)$ implying that $\overline{x} \neq \lambda y + (1-\lambda)z$, and is thus an extreme point (cf. Definition 2.10).

2. (Extreme point $\Rightarrow$ BFS) suppose $\overline{x} \in P$ is not a BFS. Then, there are no n linearly independent vectors within $\{a_i\}_{i \in I}$. Thus $\{a_i\}_{i \in I}$ lie in a proper subspace of $\mathbb{R}^n$. Let the nonzero vector $d \in \mathbb{R}^n$ be such that $a_i^\top d = 0$, for all $i \in I$.

   Let $\epsilon > 0$, $y = \overline{x} + \epsilon d$, and $z = \overline{x} - \epsilon d$. Notice that $a_i^\top y = a_i^\top z = b_i$, for all $i \in I$. Moreover, for $i \neq I$, $a_i^\top x > b$ and, provided that ϵ is sufficiently small (such that $\epsilon |a_i^\top d| < a_i^\top \overline{x} - b_i$), we have that $a_i^\top x \geq b_i$. Thus $y \in P$, and by a similar argument, $z \in P$. Now, by noticing that $\overline{x} = \frac{1}{2}y + \frac{1}{2}z$, we see that $\overline{x}$ is not an extreme point.

3. (BFS $\Rightarrow$ Vertex) Let $\overline{x}$ be a BFS. Define $c = \sum_{i \in I} a_i$. Then

   $$c^\top \overline{x} = \sum_{i \in I} a_i^\top \overline{x} = \sum_{i \in I} b_i.$$

   Also, for any $x \in P$, we have that

   $$c^\top x = \sum_{i \in I} a_i^\top x \geq \sum_{i \in I} b_i,$$

   since $a_i^\top x \geq b_i$ for $i \in M_1 \cup M_2$. Thus, for any $x \in P$, $c^\top \overline{x} \leq c^\top x$, making $\overline{x}$ a vertex (cf. Definition 2.9). □

Some interesting insights emerge from the proof of Theorem 2.14, upon which we will build our next developments. First, notice how the definition of vertex encodes a concept of optimality, since it implies that $\overline{x}$ is a unique minimiser (i.e., all other feasible points y are such that $c^\top \overline{x} < c^\top y$). Also, once the relationship between being a vertex/ extreme point and a BFS is made, it means that $\overline{x}$ can be recovered as the unique solution of a system of linear equations, these equations being the active constraints at that vertex. This means that the list of all candidate points for optimal solution can be obtained by simply looking at all possible combinations of n active constraints, discarding those that are infeasible. This means that the number of candidates for optimal solution is *finite* and can be bounded by $\binom{m}{n}$, where $m = |M_1 \cup M_2|$.

Exercises

Exercise 2.1: Polyhedral sets

Which of the following sets are polyhedral?

(a) $\{(x, y) \in \mathbb{R}^2 \mid x \cos\theta + y \sin\theta \leq 1, \theta \in [0, \pi/2], x \geq 0, y \geq 0\}$

(b) $\{x \in \mathbb{R} \mid x^2 - 8x + 15 \leq 0\}$

(c) The empty set ($\emptyset$).

Exercise 2.2: Convexity of polyhedral sets

Prove the following theorem.

Theorem (Convexity of polyhedral sets)**.** The following convexity properties about convex sets can be said:

1. *The intersection of convex sets is convex*
2. *Every polyhedral set is a convex set*
3. *A convex combination of a finite number of elements of a convex set is also belongs to that set*
4. *The convex hull of a finite number of elements is a convex set.*

Note: the theorem is proved in the notes. Use this as an opportunity to revisit the proof carefully, and try to take as many steps without consulting the text as you can. This is a great exercise to help you internalise the proof and its importance in the context of the material. I strongly advise against blindly memorising it, as I suspect you will never (in my courses, at least) be requested to recite the proof literally.

Exercise 2.3: Properties of active constraints

Let us consider the convex polyhedral set $P \subset \mathbb{R}^n$, formed by the set of equalities and inequalities:

$$a_i^\top x \geq b, i \in M_1,$$
$$a_i^\top x \leq b, i \in M_2,$$
$$a_i^\top x = b, i \in M_3.$$

Prove the following result.

Theorem (Properties of active constraints)**.** *Let $\overline{x} \in \mathbb{R}^n$ and $I = \left\{i \in M_1 \cup M_2 \cup M_3 \mid a_i^\top \overline{x} = b_i\right\}$. Then, the following are equivalent:*

1. *There exists n vectors in $\{a_i\}_{i \in I}$ that are linearly independent.*
2. *The* **span**$(\{a_i\}_{i \in I})$ *spans $\mathbb{R}^n$. That is, every $x \in \mathbb{R}^n$ can be expressed as a combination of $\{a_i\}_{i \in I}$.*
3. *The system of equations $\left\{a_i^\top x = b_i\right\}_{i \in I}$ has a unique solution.*

Note: see Exercise 2.2.

Exercise 2.4: Vertex, extreme points, and BFSs

Prove the following result.

Theorem (BFS, extreme points and vertices)**.** *Let $P \subset \mathbb{R}^n$ be a convex polyhedral set and let $\overline{x} \in P$. Then, the following are equivalent*

1. *$\overline{x}$ is a vertex;*
2. *$\overline{x}$ is a extreme point;*
3. *$\overline{x}$ is a BFS;*

Note: see Exercise 2.2.

Exercise 2.5: Binding constraints applications

Given the linear program defined by the system of inequalities below,

$$
\begin{array}{lrcl}
\max & 2x_1 + x_2 & & \\
\text{s.t.} & & & \\
& 2x_1 + 2x_2 & \leq & 9 \\
& 2x_1 - x_2 & \leq & 3 \\
& x_1 - x_2 & \leq & 1 \\
& x_1 & \leq & 2.5 \\
& x_2 & \leq & 4 \\
\\
& x_1, x_2 \geq 0 & &
\end{array}
$$

assess the following points relative to the polyhedron defined in $\mathbb{R}^2$ by this system and classify them as in (i) belonging to which active constraint(s), (ii) being a non-feasible/basic/basic feasible solution, and (iii) being an extreme point, vertex, or outside the polyhedron. Use Theorem 2.12 and Theorem 2.14 to check if your classification is correct.

(a) $(1.5, 0)$

(b) $(1, 0)$

(c) $(2, 1)$

(d) $(1.5, 3)$

Chapter 3

Basis, Extreme Points and Optimality in Linear Programming

3.1 Polyhedral sets in standard form

In the context of linear programming problems, we will often consider problems written in the so-called *standard form*. The standard form can be understood as posing the linear programming problem as an underdetermined system of equations (that is, with less equations than variables). Then, we will work selecting subset of the variables to be set to zero, on a way that the number of remaining variables is the same as that of equations, making the system solvable.

A key point in this chapter will be devising how we relate this process of selecting variables with that of selecting a subset of active constraints (forming a vertex, as we seen in the previous chapter) that will eventually lead to an optimal solution.

3.1.1 The standard form of linear programming problems

First, let us formally define the notion of a standard-form polyhedral set. Let A is a $m \times n$ matrix and $b \in \mathbb{R}^m$. The *standard form* polyhedral set P is given by

$$P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}.$$

We assume that the m equality constraints are linearly independent, i.e., A is full (row) rank ($m \leq n$). We know that a basic solution can be obtained from a collection of n active constraints, since the problem is defined in $\mathbb{R}^n$.

One important point, to which we will return later is that *any* linear programming problem can be represented in the standard form. This is achieved by means of including nonnegative *slack variables*. Thus, a feasibility set that is, say, originally represented as

$$P = \{x \in \mathbb{R}^n : A_1 x \leq b_1, A_2 x \geq b_2, x \geq 0\}$$

can be equivalently represented as a standard-form polyhedral set. For that, it must be modified to consider slack variables $s_1 \geq 0$ and $s_2 \geq 0$ such that

$$P = \left\{(x, s_1, s_2) \in \mathbb{R}^{(n+|b_1|+|b_2|)} : A_1 x + s_1 = b_1, A_2 x - s_2 = b_2, (x, s_1, s_2) \geq 0\right\},$$

where $|u|$ represents the cardinality of the vector u. This transformation, as we will see, will be a requirement for employing the simplex method to solve linear programming problems with inequality constraints and, inevitably, will always render standard form linear programming problems with more variables than constraints, or $m < n$.

The standard-form polyhedral set P always has by definition m active constraints, because they are its forming equality constraints. To reach the total of n active constraints, $n-m$ of the remaining constraints $x_i \geq 0$, $i = 1, \ldots, n$, must be activated, and this is achieved by selecting $n-m$ of those to be set as $x_i = 0$. These n active constraints (the original m plus the $n-m$ variables set to zero) form a basic solution, as we seen in the last chapter. If it happens that the m equalities can still be satisfied while the constraints $x_i \geq 0$, $i = 1, \ldots, n$, are satisfied, then we have a basic feasible solution (BFS). Theorem 3.1 summarises this process, guaranteeing that the setting of $n-m$ variables to zero will render a basic solution.

Theorem 3.1 (Linear independence and basic solutions)**.** *Consider the constraints $Ax = b$ and $x \geq 0$, and assume that A has m linearly independent (LI) rows $M = \{1, \ldots, m\}$. A vector $\overline{x} \in \mathbb{R}^n$ is a basic solution if and only if we have that $A\overline{x} = b$ and there exists indices $B(1), \ldots, B(m)$ such that*

1. *The columns $A_{B(1)}, \ldots, A_{B(m)}$ of A are LI*
2. *If $j \neq B(1), \ldots, B(m)$, then $\overline{x}_j = 0$.*

Proof. Assume that (1) and (2) are satisfied. Then the active constraints $\overline{x}_j = 0$ for $j \notin \{B(1), \ldots, B(m)\}$ and $Ax = b$ imply that

$$\sum_{i=1}^{m} A_{B(i)}\overline{x}_{B(i)} = \sum_{j=1}^{n} A_j\overline{x}_j = A\overline{x} = b.$$

Since the columns $\left\{A_{B(i)}\right\}_{i\in M}$ are LI, $\left\{\overline{x}_{B(i)}\right\}_{i\in M}$ are uniquely determined and thus $A\overline{x} = b$ has a unique solution, implying that $\overline{x}$ is a basic solution (cf. Theorem 2.14).

Conversely, assume that $\overline{x}$ is a basic solution. Let $\overline{x}_{B(1)}, \ldots, \overline{x}_{B(k)}$ be the nonzero components of $\overline{x}$. Thus, the system

$$\sum_{i=1}^{n} A_i\overline{x}_i = b \text{ and } \{\overline{x}_i = 0\}_{i\notin\{B(1),\ldots,B(k)\}}$$

has a unique solution, and so does $\sum_{i=1}^{k} A_{B(i)}\overline{x}_{B(i)} = b$, implying that the columns $A_{B(1)}, \ldots, A_{B(k)}$ are LI. Otherwise, there would be scalars $\lambda_1, \ldots, \lambda_k$, not all zeros, for which $\sum_{i=1}^{k} A_{B(i)}\lambda_i = 0$; this would imply that $\sum_{i=1}^{k} A_{B(i)}(\overline{x}_{B(i)} + \lambda_i) = b$, contradicting the uniqueness of $\overline{x}$. Since $A_{B(1)}, \ldots, A_{B(k)}$ are LI, $k \leq m$. Also, since A has m LI rows, it must have m LI columns spanning $\mathbb{R}^m$. Using Theorem 2.4, we can obtain $m-k$ additional columns $A_{B(k+1)}, \ldots, A_{B(m)}$ so that $A_{B(1)}, \ldots, A_{B(m)}$ are LI.

Finally, since $k \leq m$, $\{\overline{x}_j = 0\}_{i\notin\{B(1),\ldots,B(m)\}} \subset \{\overline{x}_j = 0\}_{i\notin\{B(1),\ldots,B(k)\}}$, satisfying (1) and (2). □

This proof highlights an important aspect in the process of generating basic solutions. Notice that, once we set $n-m$ variables to be zero, the system of equations forming P becomes uniquely determined, i.e.,

$$\sum_{i=1}^{m} A_{B(i)}\overline{x}_{B(i)} = \sum_{j=1}^{n} A_j\overline{x}_j = A\overline{x} = b.$$

3.1.2 Forming basis for standard-form linear programming problems

Theorem 3.1 allows to develop a simple procedure to generate all basic solutions of a linear programming problem in standard form.

1. Choose m LI columns $A_{B(1)}, \ldots, A_{B(m)}$;
2. Let $x_j = 0$ for all $j \notin \{B(1), \ldots, B(m)\}$;
3. Solve the system $Ax = b$ to obtain $x_{B(1)}, \ldots, x_{B(m)}$.

You might have notice that in the proof of Theorem 3.1, the focus shifted to the columns of A rather than its rows. The reason for that is because, when we think of solving the system $Ax = b$, what we are truly doing is finding a vector x representing the linear combination of the columns of A that yield the vector b. This creates an association between the columns of A and the components of x (i.e., the variables).

You will notice that from here onwards we will implicitly refer to the columns of A as variables. Then, when we say that we are setting some $(n - m)$ of the variables to be zero, it means that we are ignoring the respective columns of A (the mapping between variables and columns being their indices: x_1 referring to the first column, x_2 to the second, and so forth), while using the remainder to form a (unique) combination that yields the vector b, being the weights of this combination precisely the solution x, which in turn represent the coordinates in $\mathbb{R}^n$ of the vertex formed by the n (m equality constrains plus $n - m$ variables set to zero) active constraints.

As we will see, this procedure will be at the core of the simplex method. Since we will be often referring to elements associated with this procedure, it will be useful to define some nomenclature.

We say that $B = \left\{A_{B(i)}\right\}_{i \in I_B}$ is a *basis* (or, perhaps more precisely, a basic matrix) with basic indices $I_B = \{B(1), \ldots, B(m)\}$. Consequently, we say that the variables x_j, for x_j, for $j \in I_B$, are *basic variables*. Somewhat analogously, we say that the variables chosen to be set to zero are the *nonbasic variables* x_j, for $j \in I_N$, where $I_N = J \setminus I_B$, with $J = \{1, \ldots, n\}$ being the indices of all variables (and all columns of A).

Remark: The basic matrix B is invertible, since its columns are LI (c.f. Theorem 2.4). For $x_B = (x_{B(1)}, \ldots, x_{B(m)})$, the *unique solution* for $Bx_B = b$ is

$$x_B = B^{-1}b, \text{ where } B = \begin{bmatrix} | & | & | \\ A_{B(1)} & \ldots & A_{B(m)} \\ | & | & | \end{bmatrix} \text{ and } x_B = \begin{bmatrix} x_{B(1)} \\ \vdots \\ x_{B(m)} \end{bmatrix}.$$

Let us consider the following numerical example. If we consider the following set P

$$P = \left\{ x \in \mathbb{R}^7 : \begin{array}{r} x_1 + x_2 + 2x_3 + 1x_4 = 8 \\ x_2 + 6x_3 + x_5 = 12 \\ x_1 + x_6 = 4 \\ x_2 + x_7 = 6 \\ x_1, \ldots, x_7 \geq 0, \end{array} \right\} \tag{3.1}$$

the system $Ax = b$ can be represented as

$$\begin{bmatrix} 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 8 \\ 12 \\ 4 \\ 6 \end{bmatrix}.$$

We can make an arbitrary selections of variables to be set to zero (i.e., to be nonbasic) so we can calculate the value of the remaining (basic) variables. For example:

- Let $I_B = \{4, 5, 6, 7\}$; in that case $x_B = (8, 12, 4, 6)$ and $x = (0, 0, 0, 8, 12, 4, 6)$, which is a BFS, as $x \geq 0$.
- For $I_B = \{3, 5, 6, 7\}$, $x_B = (4, -12, 4, 6)$, which is basic but not feasible, since $x_5 < 0$.

3.1.3 Adjacent basic solutions

Now that we know how a solution can be recovered, the next important concept that we need to define is how we, from one basic solution, move to an *adjacent* solution. This will be the mechanism that the simplex will utilise to move from one solution to the next in the search for the optimal solution.

Let us start formally defining the notion of a adjacent basic solution.

Definition 3.2 (Adjacent basic solutions)**.** *Two basic solutions are adjacent if they share $n-1$ LI active constraints. Alternatively, two basis B_1 and B_2 are adjacent if all but one of their columns are the same.*

For example, consider the set polyhedral set P defined in (3.1). Our first BFS was defined by making $x_1 = x_2 = x_3 = 0$ (nonbasic index set $I_N = \{1, 2, 3\}$). Thus, our basis was $I_B = \{4, 5, 6, 7\}$. An adjacent basis was then formed, by replacing the basic variable x_4 with the nonbasic variable x_3, rendering the new (not feasible) basis $I_B = \{3, 5, 6, 7\}$.

Notice that the process of moving between adjacent basis has a simple geometrical interpretation. Since adjacent bases share all but one basic element, this means that the two must be connected by a (projected) line segment (in the case of the example, it would be the segment between $(0, 8)$ and $(4, 0)$, projected onto $(x_3, x_4) \in \mathbb{R}^2$ (recall that the basic point where $(0, 0, 0, 8, 12, 4, 6)$ and $(0, 0, 4, 0, -12, 4, 6)$, respectively). This will become clearer when we analyse the simplex method in further detail in Chapter 4.

3.1.4 Redundancy and degeneracy

An important underlying assumption in Theorem 3.1 is that the matrix A in the definition of the polyhedral set P is full (row) rank, that is, there are m linearly independent rows. Theorem 3.3 shows that this assumption can actually be made without loss of generality.

Theorem 3.3 (Redundant constraints)**.** *Let $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$, where A is $m \times n$ matrix with rows $\{a_i\}_{i \in M}$ and $M = \{1, \ldots, m\}$. Suppose that* $\mathbf{rank}(A) = k < m$ *and that the rows $a_{i_1}, \ldots, a_{i_k}$ are LI. Then P is the same set as $Q = \left\{x \in \mathbb{R}^n : a_{i_1}^\top x = b_{i_1}, \ldots, a_{i_k}^\top x = b_{i_k}, x \geq 0\right\}$.*

Proof. Assume, without loss of generality, that $i_1 = 1$ and $i_k = k$. Clearly, $P \subset Q$, since a solution satisfying the constraints forming P also satisfy those forming Q.

As $\mathbf{rank}(A) = k$, the rows $a_{i_1}, \ldots, a_{i_k}$ form a basis in the row space of A and any row a_i, $i \in M$, can be expressed as $a_i^\top = \sum_{j=1}^k \lambda_{ij} a_j^\top$ for $\lambda_{ij} \in \mathbb{R}$.

For $y \in Q$ and $i \in M$, we have $a_i^\top y = \sum_{j=1}^k \lambda_{ij} a_j^\top y = \sum_{j=1}^k \lambda_{ij} b_j = b_i$, which implies that $y \in P$ and that $Q \subset P$. Consequently, $P = Q$. □

Theorem 3.3 implies that any linear programming problem in standard form can be reduced to an equivalent problem with linearly independent constraints. It turns out that, in practice, most professional-grade solvers (i.e., software that implements solution methods and can be used to find

optimal solutions to mathematical programming models) have *preprocessing* routines to remove redundant constraints. This means that the problem is automatically treated to become smaller by not incorporating unnecessary constraints.

Degeneracy is somewhat related to the notion of redundant constraints. We say that a given vertex is a *degenerate* basic solution if it is formed by the intersection of more than n active constraints (in $\mathbb{R}^n$). Effectively, this means that more than $n - m$ variables (i.e, some of the basic variables) are set to zero, which is the main way to identify a degenerate BFS. Figure 3.1 illustrates a case in which degeneracy is present.

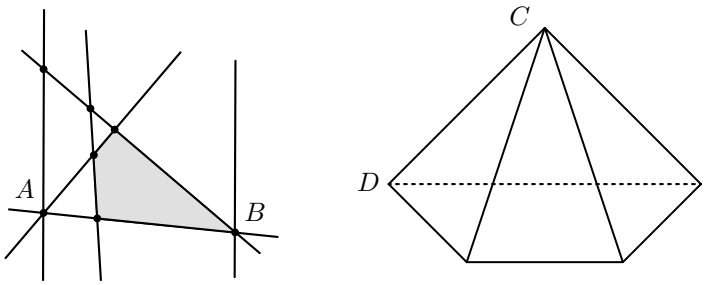


Figure 3.1: A is a degenerate basic solution, B and C are degenerate BFS, and D is a BFS.

Notice that, while in the figure on the left, the constraint causing degeneracy is redundant, that is not the case on the figure on the righthand side. That is, redundant constraints may cause degeneracy, but not all constraints causing degeneracy are redundant.

In practice, degeneracy might cause issues related to the way we identify vertices. Because more than n active constraints form the vertex, and yet, we identify vertices by groups of n constraints to be active, it means that we might be have a collection of adjacent bases that, in fact, are representing the same vertex in space, meaning that the method might be "stuck" for a while in the same position. The numerical example below illustrates this phenomenon.

Let us consider again the same example as before.

$$\begin{bmatrix} 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 8 \\ 12 \\ 4 \\ 6 \end{bmatrix}$$

Observe the following,

- let $I_B = \{1, 2, 3, 7\}$; $x = (4, 0, 2, 0, 0, 0, 6)$. There are 4 zeros (instead of $n - m = 3$) in x, which indicates degeneracy.
- Now, let $= I_B = \{1, 3, 4, 7\}$. Again, $x = (4, 0, 2, 0, 0, 0, 6)$. The two bases are adjacent, yet represent the same point in $\mathbb{R}^7$.

As we will see, there are mechanisms that prevent the simplex method from being stuck on such vertices forever, an issue that is referred to as *cycling*. One final point to observe about degeneracy is that it can be caused by the chosen representation of the problem. For example, consider the two equivalent sets:

$$P_1 = \{(x_1, x_2, x_3) : x_1 - x_2 = 0, x_1 + x_2 + 2x_3 = 2, x_1, x_3 \geq 0\} \text{ and } P_2 = P_1 \cap \{x_2 \geq 0\}.$$

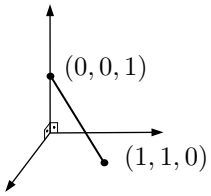


Figure 3.2: $(0, 0, 1)$ is degenerate if you add the constraint $x_2 \geq 0$

The polyhedral set P_2 is equivalent to P_1 since $x_2 \geq 0$ is a redundant constraint. In that case, one can see that, while the point $(0, 0, 1)$ is not degenerate in P_1, it is in P_2, which illustrates the (weak but existent) relationship between redundancy and degeneracy. This is illustrated in Figure 3.2.

3.2 Optimality of extreme points

Now that we have discussed how to algebraic represent extreme points and have seen a simple mechanism to iterate among their adjacent neighbours, the final element missing for us to be able to devise an optimisation method is to define the optimality conditions we wish to satisfy.

3.2.1 The existence of extreme points

First, let us define the condition that guarantee the existence of extreme points is a polyhedral set, otherwise, there is no hope hoping of finding an optimal solution.

Definition 3.4 (Existence of extreme point)**.** *A polyhedral set $P \subset \mathbb{R}^n$ contains a line if $P \neq \emptyset$ and there exists a nonzero vector $d \in \mathbb{R}^n$ such that $x + \lambda d \in P$ for all $\lambda \in \mathbb{R}$.*

Figure 3.3 illustrates the notion of containing a line and the existence of extreme points. Notice that if a set would have any sort of "corner", then that would imply that a line is contained between the edges that form that corner and, therefore, that corner would be an extreme point.

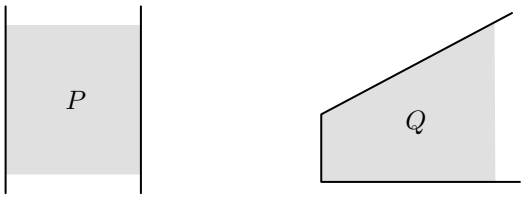


Figure 3.3: P contains a line (left) and Q does not contain a line (right)

We are now ready to pose the result that utilises Definition 3.4 to provide the conditions for the existence of extreme points.

Theorem 3.5 (Existence of extreme points)**.** *Let $P = \left\{x \in \mathbb{R}^n \mid a_i^\top x \geq b_i, i = 1, \ldots, m\right\} \neq \emptyset$ be a polyhedral set. Then the following are equivalent:*

1. *P has at least one extreme point;*
2. *P does not contain a line;*
3. *There exists n LI vectors within $\{a_i\}_{i=1}^m$.*

It tuns out that linear programming problems in the standard form do not contain a line, meaning that they will always provide at least one extreme point (or a basic feasible solution). More generally, bounded polyhedral sets do not contain a line, and neither does the positive orthant.

We are now to state the result that proves the intuition we had when analysing the plots in Chapter 1, which states that if a polyhedral set has at least one extreme point and at least one optimal solution, then there must be an optimal solution that is an extreme point.

Theorem 3.6 (Optimality of extreme points)**.** *Let $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ be a polyhedral set and $c \in \mathbb{R}^n$. Consider the problem*

$$z = min.\ \left\{c^\top x : x \in P\right\}.$$

Suppose that P has at least one extreme point and that there exists an optimal solution. Then, there exists an optimal solution that is an extreme point of P.

Proof. Let $Q = \left\{x \in \mathbb{R}^n \mid Ax \geq b, c^\top x = z\right\}$ be the (nonempty) polyhedral set of all optimal solutions. Since $Q \subset P$ and P contains no line (cf. Theorem 3.5), Q contains no line either, and thus has an extreme point.

Let $\overline{x}$ be an extreme point of Q. By contradiction, assume that $\overline{x}$ is not an extreme point of P. Then, there exist $y \neq \overline{x}$, $w \neq \overline{x}$, and $\lambda \in [0, 1]$ such that $\overline{x} = \lambda y + (1 - \lambda)w$. Then, $c^\top \overline{x} = \lambda c^\top y + (1 - \lambda)c^\top w$. As $c^\top \overline{x} = z$ is optimal, we have that $z \leq c^\top y$ and $z \leq c^\top w$, and thus $z = c^\top y = c^\top w$.

Thus, $w \in Q$ and $y \in Q$, contradicting that $\overline{x}$ is an extreme point. Thus, $\overline{x}$ must be an extreme point and, since we established that $\overline{x} \in Q$, it is also optimal. □

Theorem 3.6 is posed in a somewhat general way, which might be a source for confusion. First, recall that in the example in Chapter 1, we considered the possibility of the objective function level curve associated with the optimal value to be parallel to one of the edges of the feasible region, meaning that instead of a single optimal solution (a vertex), we would observe a line segment containing an infinite number of optimal solutions, of which exactly two would be extreme points.

In a more general case (with $n > 2$) it might be so that a whole facet of optimal solutions is obtained. That is precisely the polyhedral set of all optimal solutions Q in the proof. Clearly, this polyhedral set will not contain a line and, therefore (cf. Theorem 3.5), have at least one extreme point.

This is important because we intend to design an algorithm that only inspect extreme points. This discussion guarantees that, even for the cases in which a whole set of optimal solution exists, some elements in that set will be extreme points anyway, and thus identifiable by our method.

3.2.2 Finding optimal solutions

We now focus on the issue of being able to find and recognise extreme points as optimal solutions. In generally, optimisation methods iterate the following steps:

1. Start from an initial (often feasible) solution;

2. Find a nearby solution with better value;
3. If none are available, return the best known solution.

This very simple procedure happens to be the core idea of the majority of the optimisation methods known. We will concentrate how to identify directions of improvement and, as consequence of their absence, how to identify optimality.

Starting from a point $x \in P$, we would like to move in a direction d that yields improvement. Definition 3.7 provides a formalisation of this idea.

Definition 3.7 (Feasible directions)**.** *Let $x \in P$, where $P \subset \mathbb{R}^n$ is a polyhedral set. A vector $d \in \mathbb{R}^n$ is a feasible direction at x if there exists $\theta > 0$ for which $x + \theta d \in P$.*

Figure 3.4 provides an illustration of the concept. Notice that at extreme points, the relevant feasible directions are those along the edges of the polyhedral set, since those are the directions that can lead to other extreme points.

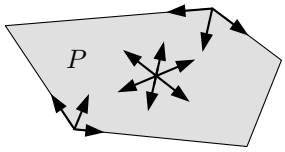


Figure 3.4: Feasible directions at different points of P

Let us now devise a way of identifying feasible directions algebraically. For that, let A be a $m \times n$ matrix, $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$. Consider the problem

$$\text{min. } \left\{c^\top x : Ax = b, x \geq 0\right\}.$$

Let x be a basic feasible solution (BFS) with basis $B = [A_{B(1)}, \ldots, A_{B(m)}]$. Recall that the basic variables x_B are given by

$$x_B = (x_{B(i)})_{i \in I_B} = B^{-1}b, \text{ with } I_B = \{B(1), \ldots, B(m)\} \subset J,$$

and that the remaining nonbasic variables x_N are such that $x_N = (x_j)_{j \in I_N} = 0$, with $I_N = J \setminus I_B$.

Moving to a neighbouring solution can be achieved by simply moving between adjacent basis, which can be accomplished with little computational burden. This entail first selecting a nonbasic variable x_j, $j \in I_N$, and increase it to a positive value θ.

Equivalently, we can define a *feasible direction* $d = [d_N, d_B]$, where d_N represent the components associated with nonbasic variables and d_B those associated with basic variables, and move from the point x to the point $x + \theta d$. The components d_N associated with the nonbasic variables are thus simply defined as

$$d = \begin{cases} d_j = 1 \\ d_{j'} = 0, \text{for all } j' \neq j, \end{cases}$$

with $j, j' \in I_N$. Notice that, geometrically, we are moving along a line in the dimension represented by the nonbasic variable x_j.

Now, feasibility might become an issue, if we are not careful to retain feasibility conditions. To retain feasibility, we must observe that $A(x + \theta d) = b$, implying that $Ad = 0$. This allows us to

define the components d_B of the direction vector d that is associated with the basic variables x_j, $j \in I_B$, since

$$0 = Ad = \sum_{j=1}^{n} A_j d_j = \sum_{i=1}^{m} A_{B(i)} d_{B(i)} + A_j = Bd_B + A_j$$

and thus $d_B = -B^{-1}A_j$ is the *basic direction* implied by the choice of the nonbasic variable x_j, $j \in I_N$ to become basic. The vector d_B can be though as the adjustments that must be made in the value of the other basic variables to accommodate the new variable becoming basic in order to retain feasibility.

Figure 3.5 provides a schematic representation of this process, showing how the change between adjacent basis can be seen as a movement between adjacent extreme points. Notice that it conveys a schematic representation of a $n = 5$ dimensional problem, in which we ignore all the m dimensions and concentrate on the $n - m$ dimensional projection of the feasibility set. This implies that the only constraints left are those associated with the nonnegativity of the variables $x \geq 0$, each associated with an edge of this alternative representation. Thus, when we set $n - m$ (nonbasic variables) to be zero, we identify an associated extreme point. As $n - m = 2$, we can plot this alternative representation on $\mathbb{R}^2$.

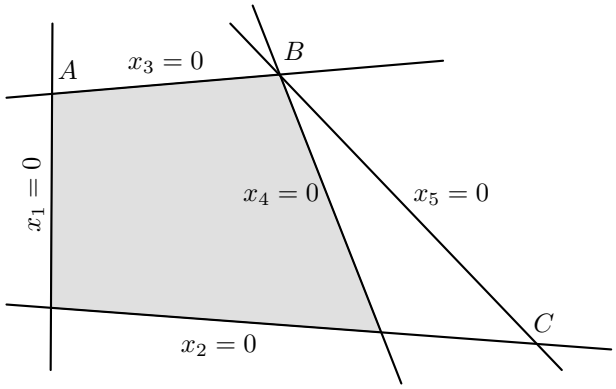


Figure 3.5: Example: $n = 5$ and $n - m = 2$. At A, $x_1 = x_3 = 0$ and $x_2, x_4, x_5 \geq 0$. Increasing x_1 while keeping x_3 zero leads to B. At B, suppose $I_N = \{3, 5\}$; by increasing x_3 while keeping x_5 zero would leads to C.

Clearly, overall feasibility, i.e., ensuring that $x \geq 0$ can only be retained if $\theta > 0$ is chosen appropriately small. This can be achieve if the following is observed:

1. All the other nonbasic variables remain valued at zero, that is, $x_{j'} = 0$ for $j' \in I_N \setminus \{j\}$.

2. if x is a *nondegenerate* extreme point, then all $x_B > 0$ and thus $x_B + \theta d_B \geq 0$ for appropriately small $\theta > 0$.

3. if x is a *degenerate* extreme point: $d_{B(i)}$ might not be feasible since, for some $B(i)$, if $d_{B(i)} < 0$ and $x_{B(i)} = 0$, any $\theta > 0$ will make $x_{B(i)} < 0$.

We will see later that we can devise a simply rule to define a value for θ that guarantees the above will be always observed. For now, we will put this discussion on hold, and focus on the issue on how to guide the choice of which nonbasic variable x_j, $j \in I_N$, to select to become basic.

3.2.3 Moving towards improved solutions

A simple yet efficient way of deciding which nonbasic component $j \in I_N$ to make basic is to consider the immediate potential benefit that it would have for the objective function value.

Specifically, if we move along the feasible direction d as previously defined, we have that the objective function value changes by

$$c^\top d = c_B^\top d_B + c_j = c_j - c_B B^{-1} A_j,$$

where $c_B = (c_{B(1)}, \ldots, c_{B(m)})$. The quantity $c_j - c_B B^{-1} A_j$ can be used, for example, in a greedy fashion, meaning that we choose the nonbasic variable index $j \in I_N$ with greatest *potential of improvement*.

First, let us formaly define this quantity, which is know as the *reduced cost*.

Definition 3.8 (Reduced cost)**.** *Let x be a basic solution associated with the basis B and objective value vector c_B. For each nonbasic variable x_j, with $j \in I_N$, we define the reduced cost $\bar{c}_j$ as*

$$\bar{c}_j = c_j - c_B^\top B^{-1} A_j.$$

The name reduced cost is motivated by the fact that it quantifies a cost change onto the reduced space of the basic variables. In fact, the reduced cost is calculating the change in the objective function caused by the increase in one unit of the nonbasic variable x_j elected to become basic (represented by the c_j component) and the associated change caused by the accommodation in the basic variable values to retain feasibility, as discussed in the previous section. Therefore, the reduced cost can be understood as a *marginal value* of change in the objective function value associated with each nonbasic variable.

Let us demonstrate the this with a numerical example. Consider the following linear programming problem

$$\begin{aligned} \text{min. } & c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 \\ \text{s.t.: } & x_1 + x_2 + x_3 + x_4 = 2 \\ & 2x_1 + 3x_3 + 4x_4 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Let $I_B = \{1, 2\}$, yielding

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

Thus, $x_3 = x_4 = 0$ and $x = (1, 1, 0, 0)$. The basic direction d_B for x_3 is given by

$$d_B = -B^{-1} A_3 = -\begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} -3/2 \\ 1/2 \end{bmatrix}.$$

The "cost" of moving along this direction is

$$c^\top d = -3c_1/2 + c_2/2 + c_3 = c_3 - (c_1, c_2)^\top d_B.$$

Clearly, the willingness for choosing x_3 as the variable to become basic will depend on whether the scalar $c_3 - (c_1, c_2)^\top d_B$ is negative (recall that we want to minimise the problem, so the smaller the total cost, the better). Another point is how large in module the reduced cost is. Recall that

the reduced is in fact a measure of the marginal value associated with the increase in value of the nonbasic variable, and thus the higher (in module) it his, the quicker the objective function value will decrease per unit of increase of the nonbasic variable value.

One interesting thing to notice is what is the reduced cost associated with basic variables. Recall that $B = [A_{B(1)}, \ldots, A_{B(m)}]$ and thus $B^{-1}[A_{B(1)}, \ldots, A_{B(m)}] = I$. Therefore $B^{-1}A_{B(i)}$ is the i^{th} column of I, denoted e_i, implying that

$$\overline{c}_{B(i)} = c_{B(i)} - c_B^\top B^{-1} A_{B(i)} = c_{B(i)} - c_B^\top e_i = c_{B(i)} - c_{B(i)} = 0.$$

3.2.4 Optimality conditions

Now that we have seen how to identify promising directions for improvement, we have incidentally developed a framework to identifying optimality of a given basic feasible solution (BFS). That is, a BFS that from which no direction of improvement can be found must be locally optimal. And, since local optimality implies global optimality in the presence of convexity (the convexity of linear programming problems was attested in Chapter 2; the global optimality in the presence of convexity is a results discussed in detail in **??**), we can declare this BFS as an optimal solution.

Theorem 3.9 establishes the optimality of a BFS from which no improving feasible direction can be found without relying on the notion of convexity.

Theorem 3.9 (Optimality conditions)**.** *Consider the problem P : min. $\{c^\top x : Ax = b, x \geq 0\}$. Let x be the BFS associated with a basis B and let $\overline{c}$ be the corresponding vector of reduced costs.*

1. *if $\overline{c} \geq 0$, then x is optimal.*
2. *if x is optimal and nondegenerate, then $\overline{c} \geq 0$.*

Proof. To prove (1), assume that $\overline{c}_j \geq 0$, let y be a feasible solution to P, and $d = y - x$. We have that $Ax = Ay = b$ and thus $Ad = 0$. Equivalently:

$$\begin{aligned} &Bd_B + \sum_{j \in I_N} A_j d_j = 0 \Rightarrow d_B = -\sum_{j \in I_N} B^{-1} A_j d_j, \text{ implying that} \\ &c^\top d = c_B^\top d_B + \sum_{j \in I_N} c_j d_j = \sum_{j \in I_N} (c_j - c_B^\top B^{-1} A_j) d_j = \sum_{j \in I_N} \overline{c}_j d_j \end{aligned} \tag{3.2}$$

We have that $x_j = 0$ and $y_j \geq 0$ for $j \in I_N$. Thus, $d_j \geq 0$ and $\overline{c}_j d_j \geq 0$ for $j \in I_N$, which implies that $c^\top d \geq 0$ (cf. (3.2)). Consequently,

$$c^\top d \geq 0 \Rightarrow c^\top (y - x) \geq 0 \Rightarrow c^\top y \geq c^\top x, \text{ i.e., } x \text{ is optimal.}$$

To prove (2) by contradiction, assume that x is optimal with $\overline{c}_j < 0$ for some $j \in I_N$. Thus, we could improve on x moving along this j^{th} direction d, contradicting the optimality of x. □

A couple of remarks are worth making at this point. First, notice that, in the presence of degeneracy, it might be that x is optimal with $\overline{c}_j < 0$ for some $j \in I_N$. Luckily, the simplex method manages to get around this issue in an effective manner, as we will see in the next chapter. Another point to notice is that, if $\overline{c}_j > 0$, $\forall j \in I_N$, then x is a *unique optimal*. Analogously, if $\overline{c} \geq 0$ with $c_j = 0$ for some $j \in I_N$, then it means that moving in that direction will cause no change in the objective function value, implying that both BFS are "equally optimal" and that the problem has multiple optimal solutions.

Exercises

Exercise 3.1: Properties of basic solutions

Prove the following theorem:

Theorem (Linear independence and basic solutions)**.** *Consider the constraints $Ax = b$ and $x \geq 0$, and assume that A has m LI rows $M = \{1, \ldots, m\}$. A vector $\overline{x} \in \mathbb{R}^n$ is a* basic solution *if and only if we have that $A\overline{x} = b$ and there exists indices $B(1), \ldots, B(m)$ such that*

1. *The columns $A_{B(1)}, \ldots, A_{B(m)}$ are LI*
2. *If $j \neq B(1), \ldots, B(m)$, then $\overline{x}_j = 0$.*

Note: the theorem is proved in the notes. Use this as an opportunity to revisit the proof carefully, and try to take as many steps without consulting the text as you can. This is a great exercise to help you internalise the proof and its importance in the context of the material. I strongly advise against blindly memorising it, as I suspect you will never (in my courses, at least) be requested to recite the proof literally.

Exercise 3.2: Basic solutions and extreme points

Consider the set $P = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 + x_2 \leq 6,\ x_2 \leq 3,\ x_1, x_2 \geq 0\}$.

(a) Enumerate all basic solutions, and identify those that are basic feasible solutions.

(b) Draw the feasible region, and identify the extreme point associated with each basic feasible solution.

(c) Consider a minimization problem with the cost vector $\mathbf{c}' = (c_1, c_2, c_3, c_4) = (-2, \frac{1}{2}, 0, 0)$. Compute the basic directions and the corresponding reduced costs of the nonbasic variables at the basic solution $\mathbf{x}' = (3, 3, 0, 0)$ with $\mathbf{x}'_B = (x_1, x_2)$ and $\mathbf{x}'_N = (x_3, x_4)$; either verify that $\mathbf{x}'$ is optimal, or move along a basic direction which leads to a better solution.

Exercise 3.3: Degeneracy - part 1

Given the linear program given below,

$$
\begin{array}{lrcl}
\max & 2x_1 + x_2 & & \\
\text{s.t.} & & & \\
& 2x_1 + 2x_2 & \leq & 9 \\
& 2x_1 - x_2 & \leq & 3 \\
& x_1 - x_2 & \leq & 1 \\
& 4x_1 - 3x_2 & \leq & 5 \\
& x_1 & \leq & 2.5 \\
& x_2 & \leq & 4 \\
& & & \\
& x_1, x_2 \geq 0 & &
\end{array}
$$

(a) Solve the problem and find the optimal point.

(b) What are the degenerated solutions and the bases forming them?

(c) Plot the constraints and degenerated points and determine why those are degenerated.

Exercise 3.4: Feasible direction

Let $\mathbf{x}$ be a point in a polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$. Show that a vector $\mathbf{d} \in \mathbb{R}^n$ is a feasible direction at $\mathbf{x} \in P$ if and only if $\mathbf{A}\mathbf{d} = 0$ and $d_i \geq 0$ for all i for which $x_i = 0$. A feasible direction of P at point $\mathbf{x}$ is a vector $\mathbf{d} \neq \mathbf{0}$ such that $\mathbf{x} + \theta\mathbf{d} \in P$ for some $\theta > 0$.

Exercise 3.5: Optimality of extreme points

Prove the following theorem.

Theorem (Optimality of extreme points)**.** *Let $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a polyhedral set and $c \in \mathbb{R}^n$. Consider the problem*

$$z = min. \left\{ c^\top x : x \in P \right\}.$$

Suppose that P has at least one extreme point and that there exists an optimal solution. Then, there exists an optimal solution that is an extreme point of P.

Note: see Exercise 3.1.

Chapter 4

The simplex method

4.1 Developing the simplex method

In Chapter 3, we discussed all the necessary theoretical aspects required for the development of the simplex method. In this chapter, we will concentrate on operationalising the method under a computational standpoint.

4.1.1 Calculating step sizes

One discussion that we purposely delayed was that of how to define the value of the step size θ taken in the feasible direction d. Let $c \in \mathbb{R}^n$, A be a $m \times n$ full-rank matrix, b a nonnegative m-sized vector (notice that this can be assumed without loss of generality, by a trivial transformation that does not change the constraint) and $J = \{1, \ldots, n\}$. Consider the linear programming problem P in the standard form

$$(P) : \text{min. } \left\{c^\top x : Ax = b, x \geq 0\right\}.$$

Building upon the elements we defined in Chapter 3, employing the simplex method to solve P comprise the following set of steps:

1. Start from a nondegenerate basic feasible solution (BFS)
2. Find a negative reduced cost component $\overline{c}_j$. If $\overline{c} \geq 0$, return the current solution.
3. Move along the feasible direction $d = (d_B, d_N)$, where $d_j = 1$, $d_{N\setminus\{j\}} = 0$ and $d_B = -B^{-1}A_j$.

Moving along the feasible direction d towards $x + \theta d$ (with $\theta > 0$) makes $x_j > 0$ (i.e., $j \in I_N$ enter the basis), while reducing the objective value at a rate of $\overline{c}_j$. Thus, one should move as furthest as possible (say, $\overline{\theta}$) along the direction d, which incurs on an objective value change of $\overline{\theta}(c^\top d) = \overline{\theta}\overline{c}_j$.

Moving as far along the feasible direction d as possible while observing that feasibility is retained is equivalent to setting $\overline{\theta}$ as

$$\overline{\theta} = \max\left\{\theta \geq 0 : A(x + \overline{\theta}d) = b,\ x + \overline{\theta}d \geq 0\right\}.$$

Recall that, by construction of the feasible direction d, we have that $Ad = 0$ and thus $A(x + \overline{\theta}d) = Ax = b$. Therefore, the only feasibility condition that can be violated when setting $\overline{\theta}$ too large is the nonnegativity of all variables, i.e., $x + \overline{\theta}d \geq 0$.

To avoid this to be the case, all basic variables $i \in I_B$ for which the component in the basic direction vector d_B is negative must be guaranteed to retain

$$x_i + \overline{\theta}d_i \geq 0 \Rightarrow \overline{\theta} \leq -\frac{x_i}{d_i}$$

Therefore, the maximum value $\overline{\theta}$ is that that can be increased until the first component of x_B turns zero. Or, more precisely put,

$$\overline{\theta} = \min_{i \in I_B : d_i < 0} \left\{ -\frac{x_{B(i)}}{d_{B(i)}} \right\}.$$

Notice that we only need to consider those basic variables with component d_i, $i \in I_B$, that are negative. This is because, if $d_i \geq 0$, then $x_i + \overline{\theta} d_i \geq 0$ holds for any value of $\overline{\theta} > 0$. This means that the constraints associated with theses basic variables (referring to the representation in Figure 3.5) do not limit the increase in value of the select nonbasic variable. Notice that this can lead to a pathological case in which none of the constraints limit the increase in value of the nonbasic variable, which is an indication that the problem has an unbounded direction of decrease for the objective function. In this case, we simply say that the problem is *unbounded*.

Another important point is the assumption of a nondegenerate BFS. The nondegeneracy of the BFS implies that $x_{B(i)} > 0$, $\forall i \in I_B$ and, thus, $\overline{\theta} > 0$. In the presence of degeneracy, one can infer that the definition of the step size $\overline{\theta}$ must be done more carefully.

Let us consider a numerical example, the same we used in Chapter 3.

$$\begin{aligned} \text{min. } & c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 \\ \text{s.t.: } & x_1 + x_2 + x_3 + x_4 = 2 \\ & 2x_1 + 3x_3 + 4x_4 = 2 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Let $c = (2, 0, 0, 0)$ and $I_B = \{1, 2\}$. The reduced costs of the nonbasic variable x_3 is

$$\overline{c}_3 = c_3 - (c_1, c_2)^\top [-3/2, 1/2] = -3.$$

where $d_B = [-3/2, 1/2]$. As x_3 increases in value, only x_1 decreases, since $d_1 < 0$. Therefore, the largest $\overline{\theta}$ for which $x_1 \geq 0$ is $-(x_1/d_1) = 2/3$. Notice that this is precisely the value that makes $x_1 = 0$, i.e., nonbasic. The new basic variable is now $x_3 = 2/3$, and the new (adjacent, as we will see next) extreme point is

$$\overline{x} = x + \theta d = (1, 1, 0, 0) + (2/3)(-3/2, 1/2, 1, 0) = (0, 4/3, 2/3, 0).$$

4.1.2 Moving between adjacent bases

Once we have defined the optimal step size $\overline{\theta}$, we move to a new BFS $\overline{x}$. That new solution is such that, for the nonbasic variable $j \in I_N$ selected to be basic, we observe that $\overline{x}_j = \theta$. Now, let us define as l the index of the basic variable that first becomes zero, that is, the variable that defines the value of $\overline{\theta}$. More precisely put, let

$$l = \operatorname*{argmin}_{i \in I_B : d_i < 0} \left\{ -\frac{x_{B(i)}}{d_{B(i)}} \right\} \text{ and, thus, } \overline{\theta} = \left\{ -\frac{x_{B(l)}}{d_{B(l)}} \right\}. \tag{4.1}$$

By moving to the BFS $\overline{x}$ by making $\overline{x} = x + \overline{\theta} d$, we are in fact moving from the basis B to an adjacent basis $\overline{B}$, defined as

$$\overline{B} = \begin{cases} \overline{B}(i) = B(i), \text{ for } i \in I_B \setminus \{l\} \\ \overline{B}(i) = j, \text{ for } i = l. \end{cases}$$

Notice that the new basis $\overline{B}$ only has one pair of variables swapped between basic and nonbasic when compared against B. Analogously, the basic matrix associated with $\overline{B}$ is given by

$$\begin{bmatrix} | & & | & | & | & & | \\ A_{B(1)} & \dots & A_{B(l-1)} & A_j & A_{B(l+1)} & \dots & A_{B(m)} \\ | & & | & | & | & & | \end{bmatrix},$$

where the middle column representing that the column $A_{B(l)}$ has been replaced with A_j.

Theorem 4.1 provide the technical result that formalises our developments so far.

Theorem 4.1 (Adjacent bases)**.** *Let A_j be the column of the matrix A associated with the selected nonbasic variable index $j \in I_N$. And let l be defined as* (4.1)*, with $A_{B(i)}$ being its respective column in A. Then*

1. *The columns $A_{B(i)}$ and A_j are linearly independent. Thus, $\overline{B}$ is a basic matrix;*
2. *The vector $\overline{x} = x + \overline{\theta} d$ is a BFS associated with $\overline{B}$.*

Proof. We start by proving (1). By contradiction, assume that $\left\{A_{B(i)}\right\}_{i \in I_B \setminus \{l\}}$ and A_j are not linearly independent. Thus, there exist $\{\lambda_i\}_{i=1}^m$ (not all zeros) such that

$$\sum_{i=1}^{m} \lambda_i A_{\overline{B}(i)} = 0 \Rightarrow \sum_{i=1}^{m} \lambda_i B^{-1} A_{\overline{B}(i)} = 0,$$

making $B^{-1}A_{\overline{B}(i)}$ not linearly independent. However, $B^{-1}A_{\overline{B}(i)} = B^{-1}A_{B(i)}$ for $i \in I_B \setminus \{l\}$ and thus are all unit vectors e_i with the l^{th} component zero.

Now, $B^{-1}A_j = -d_B$, with component $d_{B(l)} \neq 0$, is linearly indepedent from $B^{-1}A_{B(i)} = B^{-1}A_{\overline{B}(i)}$. Thus, $\left\{A_{\overline{B}(i)}\right\}_{i \in I_{\overline{B}}} = \left\{A_{B(i)}\right\}_{i \in B \setminus \{l\}} \cup \{A_j\}$ are linearly independent, forming the contradiction.

Now we focus on proving (2). We have that $\overline{x} \geq 0$, $A\overline{x} = b$ and $\overline{x}_j = 0$ for $j \in I_{\overline{N}} = J \setminus I_{\overline{B}}$. This combined with $\left\{\overline{B}(i)\right\}_{i \in I_{\overline{B}}}$ being linearly independent (cf. 1) completes the proof. □

We have finally compiled all the elements we need to state the simplex method pseudocode, presented in Algorithm 1. One minor detail in the presentation of the algorithm is use of the auxiliary vector u, which simply allows for the precalculation of the components of $d_B = -B^{-1}A_j$ (notice the changed signed) to test for unboundedness, that is, the lack of a constraint (and associated basic variable) that can limit the increase of the chosen nonbasic variable.

The last missing element is proving that Algorithm 1 eventually converges to an optimal solution, should one exists. This is formally state in Theorem 4.2.

Theorem 4.2 (Convergence of the simplex method)**.** *Assume that P has at least one feasible solution and that all BFS are nondegenerate. Then, the simplex method terminates after a finite number of iterations, in one of the following states:*

1. *The basis B and the associated BFS are optimal; or*
2. *d is such that $Ad = 0$, $d \geq 0$, and $c^\top d < 0$, with optimal value $-\infty$.*

Proof. If the condition in Line 3 of Algorithm 1 is not met, then B and associated BFS are optimal, cf. Theorem 3.9. Otherwise, if Line 5 condition is met, then d is such that $Ad = 0$ and $d \geq 0$, implying that $x + \theta d \in P$ for all $\theta > 0$, and a value reduction $\theta \overline{c}$ of $-\infty$.

Finally, notice that $\overline{\theta} > 0$ step sizes are taken along d satisfy $c^\top d < 0$. Thus, the value of successive solutions is strictly decreasing and no BFS can be visited twice. As there is a finite number of BFS, the algorithm must eventually terminate. □

Algorithm 1 Simplex method

initialise. Initial basis B, associated BFS x, and reduced costs $\overline{c}$.
while $\overline{c}_j < 0$ for some $j \in N$ **do**
    Choose some j for which $\overline{c_j} < 0$. Calculate $u = B^{-1}A_j$.
    if $u \leq 0$ **then**
        return $z = -\infty$.
    else
        $\overline{\theta} = \min_{i \in I_B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$ and $l = \operatorname{argmin}_{i \in B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$
        Set $x_j = \overline{\theta}$ and $x_B = x - \overline{\theta} u$. Form new basis $B = B \setminus \{l\} \cup \{j\}$.
        Calculate $\overline{c}_j = c_j - c_B^\top B^{-1} A_j$ for all $j \in N$.
    end if
end while
return optimal basis B and optimal solution x.

4.1.3 A remark on degeneracy

We now return to the issue related with degeneracy. As we discussed earlier, degeneracy is an important pitfall for the simplex method. To recognise that the method arrived at a degenerate BFS, one must observe how the values of the basic variables are changing. If, for $\overline{\theta}$ more than one basic variable become zero at $\overline{x} = x + \overline{\theta} d$, then $\overline{B}$ degenerate.

Basically, if the current BFS is degenerate, $\overline{\theta} = 0$ can happen when $x_{B(l)} = 0$ and the component $d_{B(l)} < 0$. Notice that a step size of $\overline{\theta} = 0$ is the only option to prevent infeasibility in this case. Nevertheless, a new basis can still be defined by replacing $A_{B(l)}$ with A_j in B, however $\overline{x} = x + \overline{\theta} d = x$. Sometimes, though staying on the same extreme point, these changing of basis on a degenerate solution might eventually expose a direction of improvement, a phenomenon that is called *stalling*. In an extreme case, it might be so that the selection of the next basic variable is such that the same extreme point is recovered over and over again, which is called *cycling*. The latter can be prevented by a specific technique for carefully selecting the variable that will enter the basis.

Figure 4.1 illustrates an example of stalling. Notice that, from the first basis ($I_B = \{1, 2, 3\}$) to the second ($I_B = \{1, 3, 4\}$) the extreme point is the same, but the new basis expose the possibility of moving to the nondegenerate basis $I_B = \{3, 4, 5\}$.

4.2 Implementing the simplex method

We now focus on some specific details related to alternative simplex method implementations. In a general sense, implementations of the simplex method vary in terms of how the selection of the nonbasic variables with $\overline{c}_j < 0$ that enters the basis is made. Also, the basic variable $l = \operatorname{argmin}_{i \in B | d_i < 0} \left\{ -x_{B(i)} / d_{B(i)} \right\}$ to leave the basis in case of ties might be of interest, specially in the attempt of preventing cycling.

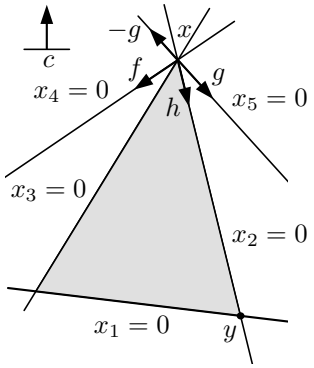


Figure 4.1: $I_N = \{4, 5\}$ for x; f ($x_5 > 0$) and g ($x_4 > 0$) are basic directions. Making $I_N = \{2, 5\}$ lead to new basic directions h ($x_4 > 0$) and $-g$ ($x_2 > 0$).

Another important aspect related to implementations of the simplex method is how matrices are represented and its consequences to memory utilisation, and how the operations related to matrix inversion are carried out.

4.2.1 Pivot or variable selection

The rules utilised for making choices in terms of entering and leaving variables are generally referred to as *pivoting rules*, though the term most commonly used to refer to the selection of nonbasic variables to enter the basis is *(simplex) pricing rules*.

- *Greedy selection* (or Dantzig's rule): choose x_j, $j \in I_N$, with largest $|\overline{c}_j|$. Prone to cycling.
- *Index-based order* (or Bland's rule): choose x_j, $j \in I_N$, with smallest j. Prevents cycling but is computationally inefficient.
- *Reduced cost pricing*: calculate $\overline{\theta}$ for all (or some) $j \in N$ and pick smallest $\overline{\theta}\overline{c}_j$. Calculating the actual observed change for all nonbasic variables is too computationally expensive. Partial pricing refers to the idea of only considering a subset of the nonbasic variables to calculate $\overline{\theta}\overline{c}_j$.
- *Devex*[1] and *steepest-edge* [2]: most commonly used by modern implementations of the simplex method available in professional-grade solvers.

4.2.2 The revised simplex method

The central element in the simplex method is the calculation of the matrix $B^{-1}A_j$, from which the reduced cost vector $\overline{c}_j$, $j \in I_N$, the basic feasible direction vector d_B and the step size $\overline{\theta}$ can be easily computed.

First, let us consider a more natural way of implementing the simplex method, so then we can point out how the method can be revised to be more computationally efficient. We will refer to this

[1] P. M. J. Harris (1973), Pivot Selection Methods in the Devex LP Code, Math. Prog., 57, 341–374.
[2] J. Forrest & D. Goldfarb (1992), Steepest-Edge Simplex Algorithms for LP, Math. Prog., 5, 1–28.

version as the naive simplex. The main differences between the naive and its revised version will be how $B^{-1}A_j$ is computed and the amount of information being carried over between iterations.

A somewhat natural way to implement the simplex method would be to first store in an auxiliary variable the term $p^\top = c_B B^{-1}$, by solving the linear system $p^\top B = c_B^\top$. These terms have an importance on themselves, as we will see later, and they are often referred to as the "simplex multipliers".

Once the simplex multipliers p are available, the reduced cost c_j associated with the nonbasic variable index $j \in I_N$ is simply

$$\overline{c}_j = c_j - p^\top A_j.$$

Once the column A_j is selected, we can then solve a second linear system $Bu = A_j$ to determine $u = B^{-1}A_j$.

The key point that can yield computational savings is the solution of the two linear systems. As one can notice, there is a common term between the two, the inverse of the basic matrix B^{-1}. If this matrix can be made available at the beginning of each iteration, then the terms $c_B^\top B^{-1}$ and $B^{-1}A_j$ can be easily and more cheaply (computationally) obtained.

For that to be possible, we need an efficient manner to update the matrix B^{-1} after each iteration. To see how that can be accomplished, recall that

$$\begin{aligned} B &= [A_{B(1)}, \ldots, A_{B(m)}], \text{ and} \\ \overline{B} &= [A_{B(1)}, \ldots, A_{B(l-1)}, A_j, A_{B(l+1)} \ldots, A_{B(m)}], \end{aligned}$$

where the l^{th} column $A_{B(l)}$ is precisely how the adjacent bases B and $\overline{B}$ differ, with A_j replacing $A_{B(l)}$ in $\overline{B}$.

We can devise an efficient manner to update B^{-1} into $\overline{B}^{-1}$ utilising elementary row operations. First, let us formally define the concept.

Definition 4.3 (Elementary row operations)**.** *Adding a constant multiple of one row to the same or another row is called an elementary row operation.*

Defining elementary row operations is the same of devising a matrix $Q = I + D_{ij}$ to premultiply B, where

$$D = \begin{cases} D_{ij} = \beta, \\ D_{i'j'} = 0 \text{ for all } i', j' \neq i, j. \end{cases}$$

Calculating $QB = (I + D)B$ is the same as having the j^{th} row of B multiplied by a scalar β and then having the resulting j^{th} row added to the i^{th} row of B. Before we continue, let us utilise a numerical example to clarify this procedure. Let

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

and suppose we would like to multiply the third row by 2 and have it then added to the first row. That means that $D_{13} = 2$ and that $Q = I + D$ would be

$$Q = \begin{bmatrix} 1 & & 2 \\ & 1 & \\ & & 1 \end{bmatrix}$$

Then premultiplying B by Q yields

$$QB = \begin{bmatrix} 11 & 14 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}.$$

As a side notice, we have that Q^{-1} exists since $\mathbf{det}(Q) = 1$. Furthermore, sequential elementary row operations $\{1, 2, \ldots, k\}$ can be represented as $Q = Q_1 Q_2, \ldots Q_k$.

Going back to the purpose of updating B^{-1} into $\overline{B}^{-1}$, notice the following. Since $B^{-1}B = I$, each term $B^{-1}A_{B(i)}$ is the i^{th} unit vector e_i (the i^{th} column of the identity matrix). That is,

$$B^{-1}\overline{B} = \begin{bmatrix} | & & | & | & | & & | \\ e_1 & \ldots & e_{l-1} & u & e_{l+1} & \ldots & e_m \\ | & & | & | & | & & | \end{bmatrix} = \begin{bmatrix} 1 & & u_1 & & \\ & \ddots & \vdots & & \\ & & u_l & & \\ & & \vdots & \ddots & \\ & & u_m & & 1 \end{bmatrix},$$

where $u = B^{-1}A_j$. We want to define an elementary row operation matrix Q such that $QB^{-1} = \overline{B}^{-1}$, or $QB^{-1}\overline{B} = I$. Therefore Q will be such that the elementary row operations turn $B^{-1}\overline{B}$ into an identity matrix, i.e., that turn the vector u into the unit vector e_l.

The main trick is that we do not need matrix multiplication to achieve it, saving considerably in computational resources. Instead, we can simply apply the elementary row operations, focusing only on the column u and the operations required to turn it into the unit vector e_l. This can be achieved by:

1. for each $i \neq l$, multiply the l^{th} row by $-\frac{u_i}{u_l}$ and add to the i^{th} row. This replaces u_i with zero for all $i \in I \setminus \{l\}$.
2. Divide the l^{th} row by u_l. This replaces u_l with one.

We can restate the simplex method in its revised form. This is presented in Algorithm 2.

Notice that in Algorithm 2, apart from the initialisation step, no linear systems are directly solved. Instead, elementary row operations (ERO) are performed, leading to computational savings.

The main aspect that actually make the revised simplex method "revised" is a matter of representation and, thus, memory allocation savings. Algorithm 2 only require the maintenance of a matrix of the form

$$\begin{bmatrix} p & | & p^\top b \\ B^{-1} & | & u \end{bmatrix}$$

which, after each series of elementary row operations yield a not only $\overline{B}^{-1}$, but also the updated simplex multipliers $\overline{p}$ and $overline p^\top b = c_B \overline{B}^{-1} b = c_B^\top \overline{x}_B$, which represents the objective function value of the new basic feasible solution $\overline{x} = [\overline{x_B}, \overline{x}_N]$. This bookkeeping savings will become obvious once we discuss the tabular (or non-revised) version of the simplex method.

Three main issues arise when considering the efficiency of implementations of the simplex method, namely, matrix (re)inversion, representation in memory, and the use of matrix decomposition strategies.

- *Reinversion:* localised updates have the side effect of accumulating truncation and round-off error. In order to correct this, solvers typically rely on periodically recalculating B^{-1} which although costly, can avoid numerical issues.

Algorithm 2 Revised simplex method

initialise. Initial basis B, associated BFS x, and B^{-1} are available.
Calculate $p^\top = c_B^\top B^{-1}$ and $\overline{c_j} = c_j - pA_j$ for $j \in N$.
while $\overline{c}_j < 0$ for some $j \in N$ **do**
    Choose some j for which $\overline{c_j} < 0$. Calculate $u = B^{-1}A_j$.
    if $u \leq 0$ **then**
        return $z = -\infty$.
    else
        $\overline{\theta} = \min_{i \in I_B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$ and $l = \operatorname{argmin}_{i \in B : u_i > 0} \left\{ \frac{x_{B(i)}}{u_i} \right\}$
        Set $x_j = \overline{\theta}$ and $x_B = x - \overline{\theta} u$.
        Form the matrix $[B^{-1} \,|\, u]$ and perform EROs to convert it to $[\overline{B}^{-1} \,|\, e_l]$.
        Make $B = B \setminus \{l\} \cup \{j\}$ and $B^{-1} = \overline{B}^{-1}$.
        Calculate $p^\top = c_B^\top B^{-1}$ and $\overline{c}_j = c_j - pA_j$ for all $j \in I_N = J \setminus I_B$.
    end if
end while
return optimal basis B and optimal solution x.

- *Representation:* A sparse representation of $Q_n = Q_1 Q_2 \dots Q_{k-1}$ can be kept instead of updating B^{-1}. Recall that $u = \overline{B}^{-1} A_j = Q B^{-1} A_j$. For larger problems, that means that a trade-off between memory allocation and the number of matrix-matrix multiplications.
- *Decomposition:* Decomposed (e.g., LU decomposition) forms of B are used to improve efficiency in storage and the solution of the linear systems to exploit the typical sparsity of linear programming problems.

4.2.3 Tableau representation

The tableau representation of the simplex method is useful as a concept presentation tool. It consist of a naive memory-space intensive representation of the problem elements as they are iterated between each basic feasible solution. However, it is a helpful representation under a pedagogical standpoint, and will be useful for explaining further concepts in the upcoming chapters.

In contrast to the revised simplex method, instead of updating only B^{-1}, we consider the complete matrix

$$B^{-1}[A \mid b] = \left[B^{-1}A_1, \dots, B^{-1}A_n \mid B^{-1}b \right].$$

Furthermore, we adjoint a row representing the reduced cost vector $\overline{c}^\top = c^\top - c_B^\top B^{-1} A$ and the negative of the objective function value for the current basis, $-c_B x_B = -c_B^\top B^{-1} b$, a row often referred to as the *zeroth row*. The reason why we consider the negative sign is that it allows for a simple updating rule for the zeroth row, by performing elementary row operations to make zero the j^{th} element associated with the nonbasic variable in B that becomes basic in $\overline{B}$.

The full tableau representation is given by

$$\begin{array}{|c|c|}\hline c^\top - c_B^\top B^{-1} A & -c_B^\top B^{-1} b \\ \hline B^{-1}A & B^{-1}b \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|c|}\hline \overline{c}_1 & \cdots & \overline{c}_n & -c_B x_B \\ \hline | & & | & x_{B(1)} \\ B^{-1}A_1 & \cdots & B^{-1}A_n & \vdots \\ | & & | & x_{B(m)} \\ \hline \end{array}$$

In this representation, we say that the j^{th} column associated with the nonbasic variable to become basic is the *pivot column* u. Notice that, since the tableau exposes the reduced costs $\overline{c}_j$, $j \in I_N$, it allows for trivially applying the greedy pricing strategy (by simply choosing the variables with a negative reduced cost of largest module).

The l^{th} row associated with the basic variable selected to leave the basis is the *pivot row*. Again, the tableau representation facilitates the calculation of the ratios used in choosing the basic variable l to leave the basis, since it amount to simply calculating the ratios between the elements on the rightmost column and those in the pivot column, disregarding those that present entries less or equal than zero and the zeroth row. The row with the minimal ratio will be that associated with the current basic variable leaving the basis.

Once a pivot column and a pivot row have been defined, it is a matter of performing elemental row operations utilising the pivot row to turn the pivot column into the unit vector e_l and turn to zero the respective zeroth element (recall that basic variables have zero reduced costs). This is the same as using elementary row operations using the pivot row to turn all elements in the pivot column zero, with exception of the *pivot element* u_l, which is the intersection of the pivot row and the pivot column, that must be turned into one. The above highlights the main purpose of the tableau representation, which is to facilitate hand calculation.

Notice that, as we seem before, performing elementary row operations to convert the pivot column u into e_l converts $B^{-1}[A \mid b]$ into $\overline{B}^{-1}[A \mid b]$. Analogously, by turning the entry associated with the pivot column u in the zeroth row to zero converts $[c^\top - c_B^\top B^{-1}A \mid -c_B^\top B^{-1}b]$ into $[c^\top - c_{\overline{B}}^\top \overline{B}^{-1}A \mid -c_{\overline{B}}^\top \overline{B}^{-1}b]$.

4.2.4 Generating initial feasible solutions

We now consider the issue of converting general linear programming problems into the standard form they are assumed to be for the simplex method. As we mentioned before, problems with constraints of the form $Ax \leq b$ can be converted by simply adding nonnegative *slack variables* $s \geq 0$, and trivially obtain an initial basic feasible solution (BFS) with $(x, s) = (0, b)$, with $B = I$, as

$$Ax \leq b \Rightarrow Ax + s = b.$$

Notice that this is equivalent to assume all original variables of the problem (i.e., those that are not slack variables) to be initialised as zero (i.e., nonbasic), since this is a trivially available initial feasible solution. However, this method does not work for constraints of the form $Ax \geq b$, as in this case, the transformation would take the form

$$Ax \geq b \Rightarrow Ax - s = b \Rightarrow Ax - s + y = b.$$

Notice that making the respective slack variable basic would yield an initial value of $-b$ (recall that $b \geq 0$ can be assumed without loss of generality), making the basic solution not feasible.

For more general problems, however, this might not be possible since simply setting the original problem variables to zero might not yield a feasible solution that can be used as a BFS. To circumvent that, we rely on *artificial variables* to obtain a BFS.

Let $P : \min. \left\{c^\top x : Ax = b, x \geq 0\right\}$, which can be achieved with appropriate transformation and assumed (without loss of generality) to have $b \geq 0$. Then, finding a BFS for P amounts to finding

a zero-valued optimal solution to the *auxiliary problem*

$$\begin{aligned}(AUX): \text{min. } & \sum_{i=1}^{m} y_i \\ \text{s.t.: } & Ax + y = b \\ & x, y \geq 0.\end{aligned}$$

The auxiliary problem AUX is formed by including one artificial variable for each constraint in P, represented by the vector y. Notice that the problem is represented in a somewhat compact notation, in which we assume that all slack variables used to convert inequalities into equalities have been already incorporated in the vector x and matrix A, with the artificial variables y playing the role of "slacks" in AUX that can be assumed to be basic and trivially yield an initial BFS for AUX. In principle, one does not need artificial variables for the rows in which there is a positive signed slack variable (i.e., an originally less-or-equal-than constraint), but this representation allows for compactness.

Solving AUX to optimality consists of trying to find a BFS in which the value of the artificial variables are zero, since, in practice, the value of the artificial variables measures a degree of infeasibility of the current basis in the context of the original problem P. This means that BFS in which the artificial variable play no roles was found and that can be used as an initial BFS for solving P. On the other hand, if the optimal for AUX is such that some of the artificial variables are nonzero, then this implies that there is no BFS for AUX in which the artificial variables are all zero, or, more specifically, there is no BFS for P, indicating that the problem P is *infeasible*.

Assuming that P is feasible and $\overline{y} = 0$, two scenarios can arise. The first is when the optimal basis B for AUX is composed only by columns A_j of the original matrix A, with no columns associated with the artificial variables. Then B can be used as an initial starting basis without any issues.

The second scenario is somewhat more complicated. Often, AUX is a degenerate problem and the optimal basis B may contain some of the artificial variables y. This then requires an additional preprocessing step, which consists of the following:

1. Let $A_{B(1)}, \ldots, A_{B(k)}$ be the columns A in B, which are linearly indepedent. Being A full-rank, we can choose additional columns $A_{B(k+1)}, \ldots, A_{B(m)}$ that will span $\mathbb{R}^m$.

2. Select the l^{th} artificial variable $y_l = 0$ and select a component j in the l^{th} row with nonzero $B^{-1}A_j$ and use elementary row operations to include A_j in the basis. Repeat this $m - k$ times.

The procedure is based on several ideas we have seem before. Since $\sum_{i=1}^{m} y_i$ is zero at the optimal, there must be a BFS in which the artificial variables are nonbasic (which is what (1) is referring to). Thus, step (2) can be repeated until a basis B is formed including none of the artificial variables.

Some interesting points are worth highlighting. First, notice that $B^{-1}A_{B(i)} = e_i$, $i = 1, \ldots, k$. Since $k < l$, the l^{th} component of each of these vectors is zero, and will remain so after performing the elementary row operations. In turn, the l^{th} entry of $B^{-1}A_j$ is not zero, and thus A_j is linearly independent to $A_{B(1)}, \ldots, A_{B(k)}$.

However, it might be so that we find zero elements in the l^{th} row. Let g be the l^{th} row of $B^{-1}A$ (i.e., the entries in the tableau associated with the original problem variables). If g is the null vector, then g_l is zero and the procedure fails. However, note that $g^\top A = 0 = g^\top Ax = g^\top b$, implying that $g^\top Ax = g^\top b$ is *redundant* can be removed altogether.

This process of generating initial BFS is often referred to as *Phase I* of the two-phase simplex method. Phase II consists of employing the simplex method as we developed it utilising the BFS found in the Phase I as a starting basis.

4.3 Column geometry of the simplex method

Now, let us try to develop a geometrical intuition on why it is so that the simplex method is remarkably efficient in practice. As we have seen in Theorem 4.2, althought the simplex method is guaranteed to converge, the total number of steps the algorithm might need to take before convergence grows exponentially with the number of variables and constraints, since the number of steps depends on the number of vertices of the feasible region. However, it turns out that, in practice, the simplex method typically requires $O(m)$ iterations, being one of the reasons why it has experienced tremendous success and is by far the most mature and reliable technology when it comes to optimisation.

In order to develop a geometrical intuition on why this is the case, let us first consider an equivalently reformulated problem P:

$$\begin{aligned} P: \text{min. } & z \\ \text{s.t.: } & Ax = b \\ & c^\top x = z \\ & \sum_{j=1}^{n} x_j = 1 \\ & x \geq 0 \end{aligned}$$

In this reformulation, we make the objective function an auxiliary variable, so it can be easily represented on a real line, at the expense of adding an additional constraint $c^\top x = z$. Furthermore, we normalise the decision variables, so they add to one (notice that this implies a bounded feasible set). Notice that problem P can be equivalently represented as

$$\begin{aligned} P: \text{min. } & z \\ \text{s.t.: } & x_1 \begin{bmatrix} A_1 \\ c_1 \end{bmatrix} + x_2 \begin{bmatrix} A_2 \\ c_2 \end{bmatrix} + \cdots + x_n \begin{bmatrix} A_n \\ c_n \end{bmatrix} = \begin{bmatrix} b \\ z \end{bmatrix} \\ & \sum_{j=1}^{n} x_j = 1 \\ & x \geq 0. \end{aligned}$$

This second formulation exposes one interesting interpretation of the problem. Solving P is akin to finding a set of weights x that makes a convex combination (cf. Definition 2.7) of the columns of A such that it constructs (or matches) b in a way that the resulting combination of the respective components of the vector c is minimised. Now, let us define some nomenclature that will be useful in what follows.

Definition 4.4 (k-dimensional simplex)**.** *A collection of vectors $y_1, \ldots, y_{k+1}$ are affinely independent if $k \leq n$ and $y_1 - y_{k+1}, \ldots, y_k - y_{k+1}$ are linearly indepedent. The convex hull of $k+1$ affinely independent vectors is a k-dimensional simplex.*

Definition 4.4 is precisely the inspiration for the name of the simplex method. We know that only $m+1$ components of x will be different than zero, since that is the number of constraints we have and, thus, the size of a basis in this case. Thus, a BFS is formed by $m+1$ $(A_i, 1)$ columns, which in turn are associated with (A_i, c_i) basic points.

Figure 4.2 provides an illustration of the concept. In this, we have that $m = 2$, so each column A_j can be represented in a two-dimensional plane. Notice that a basis require three points (A_i, c_i) and forms a 3-simplex. A BFS is a selection of three points (A_i, c_i) such that b, also illustrated in the picture, can be formed by a convex combination of the (A_i, c_i) forming the basis. This will be possible if b happens to be inside the 3-simplex formed by these points. For example, in Figure 4.2, the basis formed by columns $\{2, 3, 4\}$ is a BFS, while the basis $\{1, 2, 3\}$ is not.

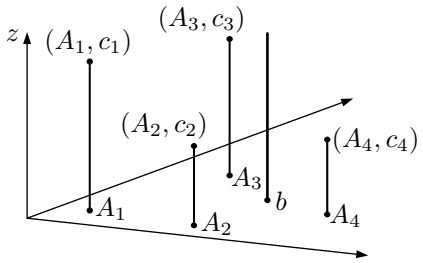


Figure 4.2: A solution x is a convex combinations of (A_i, c_i) such that $Ax = b$.

We now can add a third dimension to the analysis representing the value of z. For that, we will use Figure 4.3. As can be seen, each selection of basis creates a tilt in the three-dimensional simplex, such that the point b is met precisely at the high corresponding to its value in the z axis. This allows to compare bases according to their objective function value. And, since we are minimising, we wold like to find the basis that has its respective simplex crossing b at the lowermost point.

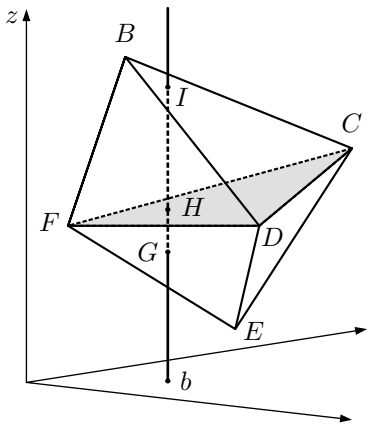


Figure 4.3: A solution x is a convex combinations of (A_i, c_i) such that $Ax = b$.

Notice that in Figure 4.3, although each facet is a basic simplex, only three are feasible (BCD, CDF, and DEF). We can also see what one iteration of the simplex method does under this geometrical interpretation. Moving between adjacent basis means that we are replacing one vertex (say, C) with another (say, E) considering the potential for decrease in value in the z axis (rep-

resented by the difference between points H and G onto the z axis). You can also see the notion of pivoting: since we are moving between adjacent bases, two successive simplexes share an edge in common, consequently, they pivot around that edge (think about the movement of the edge C moving to the point E while the edge $\overline{DF}$ remains fixed).

Now we are ready to provide an insight on why the simplex method is often so efficient. The main reason is associated with the ability that the method possess of skipping bases in favour of those with most promising improvement. To see that, consider Figure 4.4, which is a 2-dimensional schematic projection of Figure 4.3. By using the reduced costs to guide the choice of the next basis, we tend to choose the steepest of the simplexes that can provide reductions in the objective function value, which has the side effect of allowing for skipping several basis that would have to be otherwise considered. This creates a "sweeping effect", in which allows the method to find optimal solutions in fewer pivots than vertices. Clearly, this can be engineered to be prevented, as there are examples purposely constructed to force the method to consider every single vertex, but the situation illustrated in Figure 4.4 is by far the most common in practice.

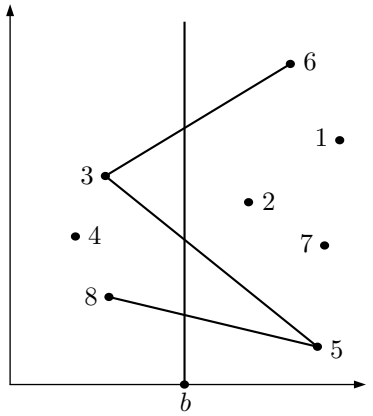


Figure 4.4: Pivots from initial basis $[A_3, A_6]$ to $[A_3, A_5]$ and to the optimal basis $[A_8, A_5]$

Exercises

Exercise 4.1: Properties of the simplex algorithms

Consider the simplex method applied to a standard form minimization problem, and assume that the rows of the matrix A are linearly independent. For each of the statements that follow, give either a proof or a counter example.

(a) An iteration of the simplex method might change the feasible solution while leaving the cost unchanged.

(b) A variable that has just entered the basis cannot leave in the very next iteration.

(c) If there is a non-degenerate optimal basis, then there exists a unique optimal basis.

Exercise 4.2: The simplex method

Consider the problem

$$\begin{aligned} \text{max. } & 40x_1 + 60x_2 \\ \text{s.t.: } & 2x_1 + x_2 \leq 7 \\ & x_1 + x_2 \leq 4 \\ & x_1 + 3x_2 \leq 9 \\ & x_1, x_2 \geq 0. \end{aligned}$$

A feasible point for this problem is $(x_1, x_2) = (0, 3)$. Formulate the problem as a minimisation problem in standard form and verify whether or not this point is optimal. If not, solve the problem by using the simplex method.

Exercise 4.3: Solving a tableau

Consider a linear programming problem in standard form, described in terms of the following tableau:

		x_1	x_2	x_3	x_4	x_5	x_6	x_7
	0	0	0	0	δ	3	γ	ξ
$x_2 =$	β	0	1	0	α	1	0	3
$x_3 =$	2	0	0	1	-2	2	η	-1
$x_1 =$	3	1	0	0	0	-1	2	1

The entries α, β, γ, δ, η and ξ in the tableau are unknown parameters. Furthermore, let $\mathbf{B}$ be the basis matrix corresponding to having x_2, x_3, and x_1 (in that order) be the basic variables. For each one of the following statements, find the ranges of values of the various parameters that will make the statement to be true.

(a) Phase II of the Simplex method can be applied using this as an initial tableau.

(b) The corresponding basic solution is feasible, but we do not have an optimal basis.

(c) The corresponding basic solution is feasible and the first Simplex iteration indicates that the optimal cost is $-\infty$.

(d) The corresponding basic solution is feasible, x_6 is candidate for entering the basis, and when x_6 is the entering variable, x_3 leaves the basis.

(e) The corresponding basic solution is feasible, x_7 is candidate for entering the basis, but if it does, the objective value remains unchanged.

Exercise 4.4: Two-phase simplex method

Solve the problem below using the two-phase simplex method. What is your conclusion about the feasibility of the problem? Verify your results by drawing the feasible region.

$$
\begin{aligned}
\text{max. } & 5x_1 + x_2 \\
\text{s.t.: } & 2x_1 + x_2 \geq 5 \\
& x_2 \geq 1 \\
& 2x_1 + 3x_2 \leq 12 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

Chapter 5

Linear Programming Duality - part I

5.1 Formulating duals

In this chapter, we will discuss the notion of duality, under the context of linear programming problem. Duality can be understood as a property that makes available a collection of tools and features that can be exploited to both further understand characteristics of the optimal solution and to devise specialised methods for solving large-scale linear programming problems.

5.1.1 Motivation

Let us define the notation we will be using throughout the next chapters. As before, let $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and P be the standard form linear programming problem

$$\begin{aligned} (P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

which we will refer to as the *primal* problem. In mathematical programming, we say that a constraint has been *relaxed* if it has been removed from the set of constraints. With that in mind, let us consider a *relaxed* version of P, where $Ax = b$ is replaced with a *violation penalty* term $p^\top(b - Ax)$. This lead to the following problem:

$$g(p) = \min_{x \geq 0} \left\{ c^\top x + p^\top (b - Ax) \right\},$$

which has the benefit of not having equality constraints explicitly represented, but only implicit by means of a penalty element that can be used to penalise infeasibility of the constraints in the attempt to steer the solution of this relaxed problems towards the solution to P. Recalling that our main objective is to solve P, we are interested in the values (or prices, as they are often called) for $p \in \mathbb{R}^m$ that make P and $g(p)$ equivalent.

Let $\overline{x}$ be the optimal solution to P. Notice that, for any $p \in \mathbb{R}^m$, we have that

$$g(p) = \min_{x \geq 0} \left\{ c^\top x + p^\top (b - Ax) \right\} \leq c^\top \overline{x} + p^\top (b - A\overline{x}) = c^\top \overline{x},$$

i.e., $g(p)$ is a *lower bound* on the optimal value $c^\top \overline{x}$. The lefthand-side inequality holds because, although $\overline{x}$ is optimal for P, it might not be optimal for $g(p)$ for an arbitrary vector p. The second inequality is a consequence of $\overline{x} \in P$, i.e., the feasibility of $\overline{x}$ implies that $A\overline{x} = b$.

We can use an optimisation-based approach to try to find an optimal lower bound, i.e., the tightest possible lower bound for P. this can be achieved by solving the *dual problem* D formulated as

$$(D): \max_{p}. \; g(p).$$

Notice that D is an unconstrained problem, to which a solution proves the *tightest* lower bound on P (say, at $\overline{p}$). Also, notice how the function $g(p): \mathbb{R}^m \mapsto \mathbb{R}$ has embedded on its evaluation the solution of a linear programming problem with $x \in \mathbb{R}^n$ as decision variables for a fixed p, which is the argument given to the function g. This is a new concept at this point that often is a source of confusion.

We will proceed in this chapter developing the analytical framework that allows use to pose the key result in duality theory, which is that stating that

$$g(\overline{p}) = c^\top \overline{x}.$$

That is, we will next develop the results that guarantee the equivalence between primal and dual representations. This will be useful for interpreting properties associated with the optimal primal solution $\overline{x}$ from the associated optimal prices $\overline{p}$. Furthermore, we will see in later chapters that linear programming duality can be used as a framework for replacing constraints with equivalent representations, which is a useful procedure in many settings, including for developing alternative solution strategies also based in linear programming.

5.1.2 General form of duals

Now, let us focus on developing a formulation for dual problems that is based on linear programming as well. Using the definition of D, we notice that

$$\begin{aligned} g(p) &= \min_{x \geq 0} \; \left\{ c^\top x + p^\top (b - Ax) \right\} \\ &= p^\top b + \min_{x \geq 0} \; \left\{ c^\top x - p^\top Ax \right\} \\ &= p^\top b + \min_{x \geq 0} \; \left\{ (c^\top - p^\top A)x \right\}. \end{aligned}$$

As $x \geq 0$, the rightmost problem can only be bounded if $(c^\top - p^\top A) \geq 0$. This gives us a linear constraint that can be used to enforce the existence of a solution for

$$\min_{x \geq 0} \; \left\{ (c^\top - p^\top A)x \right\}.$$

With that in mind, we can equivalently reformulate D as

$$\begin{aligned} (D): \max. \;\; & p^\top b \\ \text{s.t.:} \;\; & p^\top A \leq c^\top. \end{aligned}$$

Notice that D is a linear programming problem with m variables (one per constraint of the primal problem P) and n constraints (one per variable of P). As you might suspect, if you were to repeat the analysis, looking at D as the "primal", you would end with a dual that is exactly P. For this to become more apparent, let us fist define more generally the rules that dictate what kind of dual formulations are obtained for different types of primal problems in terms of its original variables and constraints.

In the more general case, let P be defined as

$$\begin{aligned}(P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax \geq b.\end{aligned}$$

Notice that the problem P can be equivalently reformulated as

$$\begin{aligned}(P): \text{min.} \quad & c^\top x \\ \text{s.t.:} \quad & Ax - s = b \\ & s \geq 0.\end{aligned}$$

Let us focus on the constraints in the reformulated version of P, which can be written as

$$[A \mid -I] \begin{bmatrix} x \\ s \end{bmatrix} = b.$$

We will apply the same procedure as before, being our constraint matrix $[A \mid -I]$ in place of A and $[x \mid s]^\top$ our vector of variables, in place of x. Using analogous arguments, we now require $c^\top - p^\top A = 0$ so $g(p)$ is finite. Notice that this is a slight deviation from before, but is this case we have that $x \in \mathbb{R}^n$, so $c^\top - p^\top A = 0$ is the only condition that allow the inner problem in $g(p)$ to have a finite solution. Then, we obtain the following dual linear programming formulation

$$\begin{aligned} & p^\top [A \mid -I] \leq [c^\top \mid 0^\top] \\ & c^\top - p^\top A = 0, \end{aligned}$$

or simply

$$\begin{aligned}(D): \text{max.} \quad & p^\top b \\ \text{s.t.:} \quad & p^\top A = c^\top \\ & p \geq 0.\end{aligned}$$

Notice how the change in the type of constraints in the primal problem P lead to additional nonnegative constraints in the dual variables p. Similarly, the absence of explicit nonnegativity constraints in the primal variables x lead to equality constraints in the dual problem D, as opposed to inequalities.

Table 5.1 provides a summary which allows one to identify the resulting formulation of the dual problem based on the primal formulation, in particular regarding its type (minimisation or maximisation), constraint types and variable domains.

For converting a minimisation primal problem into a (maximisation) dual, one must read the table from left to right. That is, the independent terms (b) become the objective function coefficients, greater or equal constraints become nonnegative variables, and so forth. However, if the primal problem is a maximisation problem, then the table must be read from right to left. For example, in this case, less-or-equal-than constraints would be come nonnegative variables instead, and so forth. It takes a little practice to familiarise yourself with this table, but it turns to be a really useful resource to obtain dual formulations from primal problems.

One remark to be made at this point is that, as is hopefully clearer now, the conversion of primal problems into duals is symmetric, meaning that reapplying the rules in Table 5.1 would take you from the obtained dual back to the original primal. This is a property of linear programming

Primal (dual)	**Dual (primal)**
minimise	maximise
Independent terms	Obj. function coef.
Obj. function coef.	Independent terms
i-th row of constraint coef.	i-th column of constraint coef.
i-th column of constraint coef.	i-th row of constraint coef.
Constraints	**Variables**
$\geq$	≥ 0
$\leq$	≤ 0
$=$	$\in \mathbb{R}$
Variables	**Constraints**
≥ 0	$\leq$
≤ 0	$\geq$
$\in \mathbb{R}$	$=$

Table 5.1: Primal-dual conversion table

problems and is called being *self dual*. Another remark is that equivalent reformulations made in the primal lead to equivalent duals. Specifically, transformations that replace variables $x \in \mathbb{R}$ with $x^+ - x^-$, with $x^+, x^- \geq 0$, introduce of nonnegative slack variables, or remove redundant constraints lead to equivalent duals.

For example, recall that the dual formulation for the primal problem

$$\begin{aligned} (P): \min.\ & c^\top x \\ \text{s.t.:}\ & Ax \geq b \\ & x \in \mathbb{R}^n \end{aligned}$$

is given by

$$\begin{aligned} (D): \max.\ & p^\top b \\ \text{s.t.:}\ & p \geq 0 \\ & p^\top A = c^\top. \end{aligned}$$

Now suppose we equivalently reformulate the primal problem to become

$$\begin{aligned} (P'): \min.\ & c^\top x + 0^\top s \\ \text{s.t.:}\ & Ax - s = b \\ & x \in \mathbb{R}^n, s \geq 0. \end{aligned}$$

Then, using Table 5.1, we would obtain the following dual formulation, which is equivalent to D

$$\begin{aligned} (D'): \max.\ & p^\top b \\ \text{s.t.:}\ & p \in \mathbb{R}^m \\ & p^\top A = c^\top \\ & -p \leq 0. \end{aligned}$$

Analogously, suppose we were to equivalently reformulate P as

$$\begin{aligned}(P'') : \text{min.} \quad & c^\top x^+ - c^\top x^- \\ \text{s.t.:} \quad & Ax^+ - Ax^- \geq b \\ & x^+ \geq 0, x^- \geq 0.\end{aligned}$$

Then, the dual formulation for P'' would be

$$\begin{aligned}(D'') : \text{max.} \quad & p^\top b \\ \text{s.t.:} \quad & p \geq 0 \\ & p^\top A \leq c \\ & -p^\top A \leq -c^\top,\end{aligned}$$

which is also equivalent to D.

5.2 Duality theory

We now will develop the technical results associated with duality that will be the kingpin for its use as a framework for devising solution methods and interpreting optimal solution properties.

5.2.1 Weak duality

Weak duality is the property associated with the bounding nature of dual feasible solutions. This is stated in Theorem 5.1.

Theorem 5.1 (Weak duality)**.** *Let x be a feasible solution to $(P) : min.\ \{c^\top x : Ax = b, x \geq 0\}$ and p be a feasible solution to $(D) : max.\ \{p^\top b : p^\top A \leq c^\top\}$, the dual problem of P. Then $c^\top x \geq p^\top b$.*

Proof. Let $I = \{i\}_{i=1}^m$ and $J = \{j\}_{j=1}^n$. For any x and p, define

$$u_i = p_i(a_i^\top x - b_i) \text{ and } v_j = (c_j - p^\top A_j)x_j.$$

Notice that $u_i \geq 0$ for $i \in I$ and $v_j \geq 0$ for $j \in J$, since each pair of terms will have the same sign. Thus, we have that

$$0 \leq \sum_{i \in I} u_i + \sum_{j \in J} v_j = [p^\top Ax - p^\top b] + [c^\top x - p^\top Ax] = c^\top x - p^\top b. \qquad \square$$

Let us also pose some results that are direct consequences of Theorem 5.1, which are summarised in Corollary 5.2.

Corollary 5.2 (Consequences of weak duality)**.** *The following are immediate consequences of Theorem 5.1:*

1. *If the optimal value of P is $-\infty$ (i.e., P is unbounded), then D must be infeasible;*
2. *if the optimal value of D is ∞ (i.e., D is unbounded), then P must be infeasible;*
3. *let x and p be feasible to P and D, respectively. Suppose that $p^\top b = c^\top x$. Then x is optimal to P and p is optimal to D.*

Proof. By contradiction, suppose that P has optimal value $-\infty$ and that D has a feasible solution p. By weak duality, $p^\top b \leq c^\top x = -\infty$, i.e., a contradiction. Part (2) follows a symmetric argument.

Part (3): let $\overline{x}$ be an alternative feasible solution to P. From weak duality, we have $c^\top \overline{x} \geq p^\top b = c^\top x$, which proves the optimality of x. The optimality of p follows a symmetric argument. □

Notice that Theorem 5.1 provides with a bounding technique for any linear programming problem. That is, for a given pair of primal and dual feasible solutions, $\overline{x}$ and $\overline{p}$, respectively, we have that

$$\overline{p}^\top b \leq c^\top x^* \leq c^\top \overline{x},$$

where $c^\top x^*$ is the optimal objective function value.

Corollary 5.2 also provides an alternative way of identifying infeasibility by means of linear programming duals. One can always use the unboundedness of a given element of a primal-dual pair to state the infeasibility of the other element in the pair. That is, an unbounded dual (primal) implies an infeasible primal (dual). However, the reverse statement is not as conclusive. Specifically, an infeasible primal (dual) does not necessarily imply that the dual (primal) is unbounded, but does imply it to be *either* infeasible or unbounded.

5.2.2 Strong duality

This bounding property can also be used as a certificate of optimality, in case they match in value. This is precisely the notion of *strong duality*, a property that is inherent to linear programming problems. This is stated in Theorem 5.3.

Theorem 5.3 (Strong duality)**.** *If* $(P) : min. \left\{c^\top x : Ax = b, x \geq 0\right\}$ *has an optimal solution, so does its dual* $(D) : max. \left\{p^\top b : p^\top A \leq c^\top\right\}$ *and their respective optimal values are equal.*

Proof. Assume P is solved to optimality, with optimal solution x and basis B. Let $x_B = B^{-1}b$. At the optimal, the reduced costs are $c^\top - c_B^\top B^{-1}A \geq 0$. Let $p = c_B^\top B^{-1}$. We then have $p^\top A \leq c^\top$, which shows that p is feasible to D. Moreover,

$$p^\top b = c_B^\top B^{-1} b = c_B^\top x_B = c^\top x, \tag{5.1}$$

which, in turn, implies the optimality of p (cf. Corollary 5.2 (3)). □

The proof of Theorem 5.3 reveals something remarkable relating the simplex method and the dual variables p. As can be seen in (5.1), for any primal feasible solution x, an associated dual (not necessarily) feasible solution p can be immediately recovered. In case the associated solution is also feasible, then Theorem 5.3 guarantees that optimality ensues.

This means that we can interchangeably solve either a primal or a dual form of a given problem, taking into account aspects related to convenience and computational ease. This is particularly useful in the context of the simplex method, since the prices p are readily available as the algorithm progresses. In the next section, we will discuss several practical uses of this relationship in more detail. For now, let us halt this discussion for a moment and consider a geometrical interpretation of duality in the context of linear programming.

5.3 Geometric interpretation of duality

Linear programming duality has a interesting geometric interpretation that stems from the much more general framework of Lagrangian duality (of which linear programming duality is a special case) and its connection to optimality conditions, topics that will be further explored in Part **??**. For now, let us focus on how linear programming duality can be interpreted in the context of balancing forces.

First, let $\overline{x}$ be the optimal solution of primal problem P in the form

$$\begin{aligned}(P): \min.\ & c^\top x \\ \text{s.t.: } & a_i^\top x \geq b_i,\ \forall i \in I.\end{aligned}$$

Imagine that there is a particle within the polyhedral set representing the feasible region of P and that this particle is subjected to a force represented by the vector $-c$. Notice that this is equivalent to minimising the function $z = c^\top x$ within the polyhedral set $\left\{a_i^\top x \geq b_i\right\}_{i\in I}$ representing the feasible set of P. Assuming that the feasible set of P is bounded in the direction $-c$, this particle will eventually come to a halt after hitting the "walls" of the feasible set, at a point where the pulling force $-c$ and the reaction of these walls reach an equilibrium. We can think of $\overline{x}$ as this stopping point. This is illustrated in Figure 5.1.

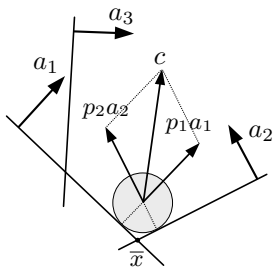


Figure 5.1: A geometric representation of duality for linear programming problems

We can then think of the dual variables p as the multipliers applied to the normal vectors associated with the hyperplanes (i.e., the walls) that are in contact with the particle to achieve this equilibrium. Hence, these multipliers p will be such that

$$c = \sum_{i\in I} p_i a_i,\ \text{for some } p_i \geq 0, i \in I,$$

which is precisely the dual feasibility condition (i.e., constraint) associated with the dual of P, given by

$$D: \max.\ \left\{p^\top b : p^\top A = c, p \geq 0\right\}.$$

And, dual feasibility, as we seen before, implies the optimality of $\overline{x}$.

5.3.1 Complementary slackness

One point that must be noticed is that, for the constraints that are not active at the optimal point $\overline{x}$ (i.e., the walls that are not exerting resistance to the particle at the equilibrium point),

the multipliers p must be set to zero. That is, we have that

$$p^\top b = \sum_{i \in I} p_i b_i = \sum_{i \in I} p_i (a_i^\top \overline{x}) = c^\top \overline{x},$$

which again implies the optimality of p (cf. Corollary 5.2 (3)). This geometrical insight leads to another key result for linear programming duality, which is the notion of *complementary slackness*.

Theorem 5.4 (Complementary slackness)**.** *Let x be a feasible solution for*

$$(P) : \textit{min.} \ \left\{ c^\top x : Ax = b, x \geq 0 \right\}$$

and p be a feasible solution for

$$(D) : \textit{max.} \ \left\{ p^\top b : p^\top A \leq c^\top \right\}.$$

The vectors x and p are optimal solutions to P and D, respectively, if and only if $p_i(a_i^\top x - b_i) = 0, \forall i \in I$, and $(c_j - p^\top A_j)x_j = 0$, $\forall j \in J$.

Proof. From the proof of Theorem 5.1 and with Theorem 5.3 holding, we have that

$$p_i(a_i^\top x - b_i) = 0, \forall i \in I, \text{ and } (c_j - p^\top A_j)x_j = 0, \ \forall j \in J.$$

In turn, if these hold, then x and p are optimal (cf. Corollary 5.2 (3)). □

For nondegenerate basic feasible solutions (BFS) (i.e., $x_j > 0$, $\forall j \in I_B$, where I_B is the set of basic variable indices), complementary slackness determine a *unique* dual solution. That is

$$(c_j - p^\top A_j)x_j = 0, \text{ which yields } c_j = p^\top A_j, \ \forall j \in I_B,$$

which has a unique solution $p^\top = c_B^\top B^{-1}$, as the columns A_j of be are assumed to be linearly indepedent. In the presence of degeneracy, this is not the case anymore, typically implying that a degenerate optimal BFS will multiple feasible dual variable.

5.3.2 Dual feasibility and optimality

Combining what we have seen so far, the conditions for a *primal-dual pair* (x, p) to be optimal to their respective primal (P) and dual (D) problems are given by

$$a_i^\top x \geq b_i, \ \forall i \in I \qquad \text{(primal feasibility)} \tag{5.2}$$

$$p_i = 0, \ \forall i \notin I^0 \qquad \text{(complementary conditions)} \tag{5.3}$$

$$\sum_{i \in I} p_i^\top a_i = c \qquad \text{(dual feasibility I)} \tag{5.4}$$

$$p_i \geq 0, \qquad \text{(dual feasibility II)} \tag{5.5}$$

where $I^0 = \left\{ i \in I, a_i^\top x = b_i \right\}$ are the active constraints. From (5.2)–(5.5), we see that the optimality of the primal-dual pair has two main requirements. The first is that x must be (primal) feasible. The second, expressed as

$$\sum_{i \in I^0} p_i a_i = c, \ p_i \geq 0,$$

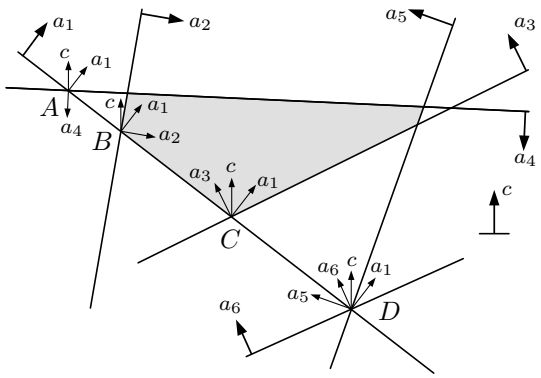


Figure 5.2: A is both primal and dual infeasible; B is primal feasible and dual infeasible; C is primal and dual feasible; D is degenerate.

is equivalent to requiring c to be expressed as a nonnegative linear combination (also known as a conic combination) of the active constraints. This has a nice geometrical interpretation: dual feasibility can be seen as having the vector c inside the "cone" formed by the normal vectors of the active constraints, which in turn is a necessary condition for the existence of an equilibrium, as described in Figure 5.1. Figure 5.2 illustrates this fact.

Notice how in Figure 5.2 neither points A or B are dual feasible, while C represents a dual feasible point, being thus the optimal for the problem depicted. One interesting point to notice is D. Although not feasible, it allows us to see an important effect that degeneracy may cause. Assume for a moment that D is feasible. Then, dual feasibility becomes dependent on the basis representing the vertex. That is, while the bases $I_B = \{1, 5\}$ and $I_B = \{1, 6\}$ are dual feasible, the basis $I_B = \{5, 6\}$ is not. As we will see, just as it is the case with the simplex method, the dual simplex method, which we will discuss in the next section, might be subject to stalling and cycling from the presence of primal degeneracy, which in turn may also leads to multiple dual feasible (primal optimal) solutions.

5.4 Practical uses of duality

We now consider practical uses for the properties we have discussed in the previous section. The most direct application of duality in linear programming problems is the interpretation of dual variable values as marginal values associated with constraints, with important economical implications.

Another important employment of duality consists of the development of an alternative variant of the simplex method, the *dual simplex method*, which is particularly useful in several contexts.

5.4.1 Optimal dual variables as marginal costs

The optimal dual variable values associated with an optimal BFS have an important practical interpretation. To see that, let $\overline{x}$ be a nondegenerate optimal BFS with corresponding basis B. Thus, $\overline{x}_B = B^{-1}b > 0$.

Now, assume that we cause a marginal perturbation on the vector b, represented by a vector d.

That is, assume that we have $B^{-1}(b+d) > 0$, noticing that nondegenerate feasibility holds.

Recall that the optimality condition $\overline{c} = c^\top - c_B^\top B^{-1}A \geq 0$ is not influenced by such a marginal perturbation. That is, for a small change d, the optimal basis (i.e., the selection of which the are basic variables) is not disturbed. On the other hand, the optimal value of the basic variables is, and consequently, so is the optimal value, which becomes

$$c_B^\top B^{-1}(b+d) = p^\top(b+d).$$

Notice that $p^\top = c_B^\top B^{-1}$ is optimal for the (respective) dual problem. Thus, a change d causes a change of $p^\top d$ in the optimal value, meaning that the components p_i represent a marginal value/cost associated with the independent term b_i, for $i \in I$.

This has important implications in practice, as it allows for pricing the the values of the resources associated with constraints. For example, if the dual value (or price) p_i is associated with a resource whose requirement is given by b_i, this means that any opportunity of removing any units of b_i for less than p_i should be seized, since it costs p_i to satisfy any additional unit in the requirement b_i. A similar interpretation can be made in the context of less-or-equal-than constraints, in which p_i would indicate benefits (or loses, if $p_i > 0$) in increasing the availability of b_i.

5.4.2 The dual simplex method

In general, solution methods in mathematical programming can be either *primal methods*, in which primal feasibility of an initial solution is maintained while seeking for dual feasibility (i.e., primal optimality); or *dual methods*, where dual feasibility is maintained while seeking for primal feasibility (i.e., dual optimality).

As we have seen in Chapter 4, the original (or primal) simplex method iterated from an initial basic feasible solution (BFS) until the optimality condition

$$\overline{c} = c^\top - c_B B^{-1}A \geq 0$$

was observed. Notice that this condition is precisely the dual feasibility condition $p^\top A \leq c$.

Being a dual method, the dual version of the simplex method, or the *dual simplex method*, consider conditions in a reverse order. That is, it starts from an initial dual feasible solution and iterates in a manner that the primal feasibility condition $B^{-1}b \geq 0$ is *sought* for, while $\overline{c} \geq 0$, or equivalently, $p^\top A \leq c$, is maintained.

To achieve that, one must revise the pivoting of the primal simplex method such that the variable to leave the basis is some $i \in I_B$, with $x_{B(i)} < 0$, while the variable chosen to enter the basis is some $j \in I_N$, such that $\overline{c} \geq 0$ is maintained.

Consider the l^{th} simplex tableau row for which $x_{B(i)} < 0$ of the form $[v_1, \ldots, v_n, x_{B(i)}]$; i.e., v_j is the l^{th} component of $B^{-1}A_j$.

For each $j \in I_N$ for which $v_j < 0$, we pick

$$j' = \arg\min_{j \in I_N : v_j < 0} \frac{\overline{c}_j}{|v_j|}.$$

Pivoting is performed by employing elemental row operations to replace $A_{B(i)}$ with A_j in the basis. This implies that $\overline{c}_j \geq 0$ is maintained, since

$$\frac{\overline{c}_j}{|v_j|} \geq \frac{\overline{c}_{j'}}{|v_{j'}|} \Rightarrow \overline{c}_j - |v_j|\frac{\overline{c}_{j'}}{|v_{j'}|} \geq 0 \Rightarrow \overline{c}_j + v_j\frac{\overline{c}_{j'}}{|v_{j'}|} \geq 0, \forall j \in J.$$

Notice that it also justifies why we must only consider for entering the basis those variables for which $v_j < 0$. Analogously to the case in the primal simplex method, if we observe that $v_j \geq 0$ for all $j \in J$, then no limiting condition is imposed in terms the increase in the nonbasic variable (i.e., an unbounded dual, which, according to Corollary 5.2 (2), implies the original problem is infeasible).

Assuming that the dual is not unbounded, the termination of the dual simplex method is observed when $B^{-1}b \geq 0$ is achieved, and primal-dual optimal solutions have been found, with $x = (x_B, x_N) = (B^{-1}b, 0)$ (i.e., the primal solution) and $p = (p_B, p_N) = (0, c_B^\top B^{-1})$ (dual). Algorithm 3 presents a pseudocode for the dual simplex method.

Algorithm 3 Dual simplex method

initialise. Initial basis B and associated basic solution x.
while $x_B = B^{-1}b < 0$ for some component $i \in I_B$ **do**
    Choose some l for which $x_{B(l)} < 0$. Calculate $u = B^{-1}A_j$.
    if $u \geq 0$ **then**
        return $z = +\infty$.
    else
        Form new basis $B = B \setminus \{l\} \cup \{j'\}$ where $j' = \arg\min_{j \in I_N : u_j < 0} \frac{\bar{c}_j}{|u_j|}$
        Calculate $x_B = B^{-1}b$.
    end if
end while
return optimal basis B and optimal solution x.

To clarify some of the previous points, let us consider a numerical example. Consider the problem

$$\begin{aligned} \text{min. } & x_1 + x_2 \\ \text{s.t.: } & x_1 + 2x_2 \geq 2 \\ & x_1 \geq 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

The first thing we must do is to convert the greater-or-equal-than inequalities into less-or-equal-than inequalities and add the respective slack variables. This allows us to avoid the inclusion of artificial variables, which are not required anymore since we can allow for primal infeasibility. This leads to the equivalent standard form problem

$$\begin{aligned} \text{min. } & x_1 + x_2 \\ \text{s.t.: } & -x_1 - 2x_2 + x_3 = -2 \\ & -x_1 + x_4 = -1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

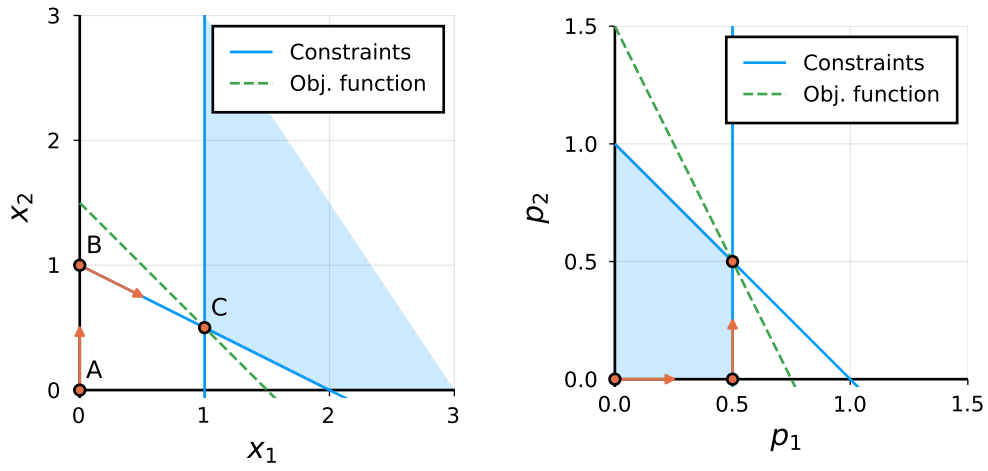
Below is the sequence of tableaus after applying the dual simplex method to solve the problem. The terms in bold font represent the pivot element (i.e., the intersection between the pivot row and pivot column).

	x_1	x_2	x_3	x_4	RHS
z	1	1	0	0	0
x_3	-1	**-2**	1	0	-2
x_4	-1	0	0	1	-1

	x_1	x_2	x_3	x_4	RHS
z	1/2	0	1/2	0	-1
x_2	1/2	1	-1/2	0	1
x_4	**-1**	0	0	1	-1

	x_1	x_2	x_3	x_4	RHS
z	0	0	1/2	1/2	-3/2
x_2	0	1	-1/2	1/2	1/2
x_1	1	0	0	-1	1

Figure 5.3 illustrates the progress of the algorithm both in the primal (Figure 5.3a) and in the dual (Figure 5.3b) variable space. Notice how in the primal space the solution remains primal infeasible until a primal feasible solution is reached, that being the optimal for the problem. Also, notice that the coordinates of the dual variables can be extracted from the zeroth row of the simplex tableau.



(a) The primal-variable space

(b) The dual-variable space

Figure 5.3: The progress of the dual simplex method in the primal and dual space.

Some interesting features related to the progress of the dual simplex algorithm are worth highlighting. First, notice that the objective function is monotonically increasing in this case, since $x_{B(l)}\frac{\bar{c}_{j'}}{|v_{j'}|}$ is added to $-c_B B^{-1}b$ and $x_{B(l)} < 0$, meaning that the dual cost increases (recall the convention of having a minus sign so that the zeroth row correctly represent the objective function value, given by the negative of the value displayed in the rightmost column). This illustrates the gradual loss of optimality in the search for (primal) feasibility. For a nondegerate problem, this can also be used as an argument for eventual convergence, since the dual objective value can only increase and is bounded by the primal optimal value. However, in the presence of dual degeneracy, that is, $\bar{c}_j = 0$ for some $j \in I_N$ in the optimal solution, the algorithm can suffer from cycling. As we seen before, that is an indication that the primal problem has multiple optima.

The dual simplex method is often the best choice of algorithm, because it typically precludes the need of a Phase I type of method as it is often trivial to find initial dual feasible solutions (the origin, for example, is typically dual feasible in minimisation problems with nonnegative coefficients; similar trivial cases are also well known).

Moreover, dual simplex is the algorithm of choice for resolving a linear programming problem when after finding an optimal solution, you modify the feasible region. Turns out that this procedure is in the core of the methods used to solve integer programming problems, as well as in the Benders decomposition, both topics we will explore later on. The dual simplex method is also more successful than its primal counterpart in combinatorial optimisation problems, which are typically plagued with degeneracy. As we have seen, primal degeneracy simply means multiple dual optima, which are far less problematic under an algorithmic standpoint.

Most professional implementations of the simplex method use by default the dual simplex version. This has several computational reasons, in particular related to more effective Phase I and pricing methods for the dual counterpart.

Exercises

Exercise 5.1: Duality in the transportation problem

Recall Exercise 1.4 in which we solved a capacitated transportation problem, answer the following questions based on the dual prices interpretation. The tables with the arc capacities and supply/demand settings are once more presented in Tables 5.2a and 5.2b.

node	p1	p2
s1 / d1	80 / 60	400 / 300
s2 / d2	200 / 100	1500 / 1000
s3 / d3	200 / 200	300 / 500

(a) Supplies availability and demand per oil type [in L]

	d1 p1/p2 (cap)	d2 p1/p2 (cap)	d3 p1/p2 (cap)
s1	5/- (∞)	5/18 (300)	-/- (0)
s2	8/15 (300)	9/12 (700)	7/14 (600)
s3	-/- (0)	10/20 (∞)	8/- (∞)

(b) Arcs costs per oil type [in € per L] and arcs‘ capacities [in L]

Table 5.2: Supply chain data

(a) What would be the price the company would be willing to pay for raising in one unit any of the supplies availability?

(b) What would be the price the company would be willing to pay for raising in one unit any of the arcs capacity?

Exercise 5.2: Dual simplex

(a) Solve the problem below by using the dual Simplex method. Report both the primal and dual optimal solutions x and p associated with the optimal basis.

(b) Write the dual and use duality to verify that x and p are optimal.

$$
\begin{aligned}
\text{min. } & 2x_1 + x_3 \\
\text{s.t.: } & -1/4x_1 - 1/2x_2 \leq -3/4 \\
& 8x_1 + 12x_2 \leq 20 \\
& x_1 + 1/2x_2 - x_3 \leq -1/2 \\
& 9x_1 + 3x_2 \geq -6 \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}
$$

Exercise 5.3: Unboundedness and duality

Consider the LP:

$$
\begin{aligned}
(P): \text{min. } & c^\top x \\
\text{s.t.: } & Ax = b \\
& x \geq 0
\end{aligned}
$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Show that if P has a finite optimal solution, then the new problem $\overline{P}$ obtained from P by replacing the right hand side vector b with another one $\bar{b} \in \mathbb{R}^m$ cannot be unbounded no matter what value the components of $\bar{b}$ can take.

Exercise 5.4: Dual in matrix form

Consider the LP:

$$
\begin{aligned}
(P): \text{min. } & {c^1}^\top x^1 + {c^2}^\top x^2 + {c^3}^\top x^3 \\
\text{s.t.} & \\
& A^1x^1 + A^2x^2 + A^3x^3 \leq b^1 \quad (y^1) \\
& A^4x^1 + A^5x^2 + A^6x^3 \leq b^2 \quad (y^2) \\
& A^7x^1 + A^8x^2 + A^9x^3 \leq b^3 \quad (y^3) \\
& x^1 \leq 0 \\
& x^2 \geq 0 \\
& x^3 \in \mathbb{R}^{|x^3|}
\end{aligned}
$$

where $A^{1,\ldots,9}$ are matrices, $b^{1,\ldots,3}$, $c^{1,\ldots,3}$ are column vectors, and $y^{1,\ldots,3}$ are the dual variables associated to each constraint.

(a) Write the dual problem in matrix form.

(b) Compute the dual optimum for the case in which

$A^1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$; $A^2 = \begin{bmatrix} 5 & 1 \\ 0 & 0 \end{bmatrix}$; $A^3 = \begin{bmatrix} 6 \\ 0 \end{bmatrix}$; $A_4 = \begin{bmatrix} 1 & 1 \end{bmatrix}$; $A^5 = \begin{bmatrix} 0 & 1 \end{bmatrix}$; $A^6 = \begin{bmatrix} 1 \end{bmatrix}$;
$A^7 = \begin{bmatrix} 0 & 2 \end{bmatrix}$; $A^8 = \begin{bmatrix} 0 & 0 \end{bmatrix}$; $A^9 = \begin{bmatrix} 3 \end{bmatrix}$; $c^1 = \begin{bmatrix} 3 \\ 9 \end{bmatrix}$; $c^2 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$; $c^3 = \begin{bmatrix} 1 \end{bmatrix}$; $b^1 = \begin{bmatrix} 5 \\ 10 \end{bmatrix}$;
$b^2 = \begin{bmatrix} 3 \end{bmatrix}$; $b^3 = \begin{bmatrix} 6 \end{bmatrix}$.

Exercise 5.5: Primal-dual conversion and complementary slackness

Recall the paint factory problem introduced in Section 1.2.1.

(a) Construct the dual of the problem below and solve both the original problem and its dual.

(b) Use complementary slackness to verify that the primal and dual solutions are optimal.

$$
\begin{aligned}
\text{max. } & z = 5x_1 + 4x_2 \\
\text{s.t.: } & 6x_1 + 4x_2 \leq 24 \\
& x_1 + 2x_2 \leq 6 \\
& x_2 - x_1 \leq 1 \\
& x_2 \leq 2 \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

Chapter 6

Linear Programming Duality - part II

6.1 Sensitivity analysis

We consider now further developments arising from the notion of duality in linear programming problems. We begin by focusing on the employment of the dual simplex method and the interpretation of the dual multipliers, as discussed in Chapter 5.

Specifically, assume that we have solved to optimality the problem P given as

$$\begin{aligned} P: \text{min. } & c^\top x \\ \text{s.t.: } & Ax = b \\ & x \geq 0. \end{aligned}$$

As we have seen in Chapter 4, the optimal solution $\overline{x}$ with associated basis B satisfies the following optimality conditions: it is a *basic feasible solution* and, therefore (i) $B^{-1}b \geq 0$; and (ii) all reduced costs are nonnegative, that is $c^\top - c_B^\top B^{-1}b \geq 0$.

We are interested in analysing aspects associated with the *stability* of the optimal solution $\overline{x}$, in terms of how it changes in face of the inclusion of new decision variables and constraints, or in face of changes in the input data. Both cases are somewhat motivated by the realisation that problems typically emerge from dynamic settings and thus one must assess how *stable* a given plan (represented by $\overline{x}$) is, or how it can be adapted, in face of changes in the original problem setting. This kind of analysis is generally referred to as *sensitivity analysis* in the context of linear programming.

First, we will consider the inclusion of new variables or new constraints *after* the optimal solution $\overline{x}$ is obtained. This setting represents, for example, the inclusion of a new product or a new production plant (referring to the context of resource allocation and transportation problems, as discussed in Chapter 1) or the consideration of additional constraints imposing new (or previously disregarded) requirements or conditions. The techniques we will consider here will also be relevant in the next chapters, when we discuss specialised methods for large-scale problems and solution techniques for integer programming problems, both topics that heavily rely on the idea of iteratively incrementing linear programming problems with additional constraints (or variables).

The second group of cases relate to changes in the input data. When utilising linear programming models to optimise systems performance, one must bear in mind that there is inherent uncertainty associated with the input data. Be it due to measurement errors, or due to lack of complete knowledge about the future, one must accept that the input data of these models will, by definition, embed some measure of error. One way of taking this into account is to try to understand what

would be the consequences to the optimality of $\overline{x}$ in terms of eventual changes in the input data, represented by the matrix A, and the vectors c and b. We will achieve this by studying the ranges within which variations in this terms do not compromise the optimality of $\overline{x}$.

6.1.1 Adding a new variable

Let us consider that a new variable x_{n+1} with associated column (that is, respective objective function and constraint coefficients) (c_{n+1}, A_{n+1}) is added to P. This leads to a new augmented problem P' of the form

$$\begin{aligned} P' : \text{min. } & c^\top x + c_{n+1}x_{n+1} \\ \text{s.t.: } & Ax + A_{n+1}x_{n+1} = b \\ & x \geq 0, x_{n+1} \geq 0. \end{aligned}$$

We need to determine if, after the inclusion of this new variable, the current basis B is still optimal. Clearly, making the newly added variable nonbasic yields the basic feasible solution (BFS) $x = (\overline{x}, 0)$. Moreover, we know that the optimality condition $c^\top - c_B^\top B^{-1} b \geq 0$ held before the inclusion of the variable, so we know that all the other reduced costs associated with the nonbasic variables $j \in I_N$ were nonnegative.

Therefore, the only check that needs to be done is whether the reduced cost associated with x_{n+1} also satisfy the optimality condition, i.e., if

$$\overline{c}_{n+1} = c_{n+1} - c_B^\top B^{-1} A_{n+1} \geq 0.$$

If the optimality condition is satisfied, this means that the new variable does not change the optimal basis and the solution $x = (\overline{x}, 0)$ is optimal. Otherwise, one must perform a new simplex iteration, using B as a starting BFS. Notice that in this case primal feasibility is trivially satisfied, while dual feasibility is not observed (that is, $\overline{c}_{n+1} < 0$). Therefore, primal simplex can be employed, *warm started* by B. This is often a far more efficient strategy than resolving P' from scratch.

Let us consider a numerical example. Consider the problem

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 3x_2 + x_4 = 16 \\ & x_1, \ldots, x_4 \geq 0. \end{aligned}$$

The tableaus associated with its optimal solution is given by

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

Suppose we include a variable x_5, for which $c_5 = -1$ and $A_5 = (1, 1)$. The modified problem then becomes

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 - x_5 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 + x_5 = 10 \\ & 5x_1 + 3x_2 + x_4 + x_5 = 16 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

We have that the reduced cost of the new variable is given by $\bar{c}_5 = c_5 - c_B^\top B^{-1} A_5 = -4$ and $B^{-1}A_5 = (-1, 2)$. The tableau for the optimal basis B considering the new column associated with x_5 is thus

	x_1	x_2	x_3	x_4	x_5	RHS
z	0	0	2	7	-4	12
x_1	1	0	-3	2	-1	2
x_2	0	1	5	-3	2	2

Notice that this tableau now shows a primal feasible solution that is not optimal and can be further iterated using primal simplex.

6.1.2 Adding a new constraint

We now focus on the inclusion of additional constraints. Let us assume that a general constraint of the form ${a_{m+1}}^\top x \geq b_{m+1}$ is added to P. We assume it to be an inequality that has been applied to problem P, which was originally in the standard form, after it was solved.

The first thing to notice is that, if the optimal solution $\overline{x}$ to P satisfy ${a_{m+1}}^\top \overline{x} \geq b_{m+1}$, then nothing changes. Otherwise, we need to rewrite the new constraint accordingly by including a slack variable, obtaining

$$ {a_{m+1}}^\top x - x_{n+1} = b_{m+1}. $$

Notice that doing so changes the matrix A of the original problem P, which becomes

$$ \overline{A} = \begin{bmatrix} A & 0 \\ a_{m+1}^\top & -1 \end{bmatrix}. $$

We can reuse the optimal basis B to form a new basis $\overline{B}$ for the problem. This will have the form

$$ \overline{B} = \begin{bmatrix} B & 0 \\ a^\top & -1 \end{bmatrix}, $$

where a are the respective components of a_{m+1} associated with the columns from A that formed B. Now, since we have that ${\overline{B}}^{-1}\overline{B} = I$, we must have that

$$ {\overline{B}}^{-1} = \begin{bmatrix} B^{-1} & 0 \\ a^\top B^{-1} & -1 \end{bmatrix}. $$

Notice however, that the basic solution $(\overline{x}, {a_{m+1}}^\top \overline{x} - b_{m+1})$ associated with $\overline{B}$ is not feasible, since we assumed that $\overline{x}$ did not satisfy the newly added constraint, i.e., ${a_{m+1}}^\top \overline{x} < b_{m+1}$.

The reduced costs considering the new basis $\overline{B}$ then becomes

$$ [c^\top \;\; 0] - [c_B^\top \;\; 0] \begin{bmatrix} B^{-1} & 0 \\ a^\top B^{-1} & -1 \end{bmatrix} \begin{bmatrix} A & 0 \\ a_{m+1}^\top & -1 \end{bmatrix} = [c^\top - c_B^\top B^{-1} A \;\; 0]. $$

Notice that the new slack variable has a null component as a reduced cost, meaning that it does not violated dual feasibility conditions. Thus, after adding a constraint that makes $\overline{x}$ infeasible, we still have a dual feasible solution that can be immediately used by the dual simplex method, again allowing for warm starting the solution of the new problem.

To build an initial solution in terms of the tableau representation of the simplex method, we must simply add an additional row, which leads to a new tableau with the following structure

$$\overline{B}^{-1}A = \begin{bmatrix} B^{-1}A & 0 \\ a^\top B^{-1}A - a_{m+1}^\top & 1 \end{bmatrix}.$$

Let us consider again a numerical example. Consider the same problem as the previous example, but that we instead include the additional constraint $x_1 + x_2 \geq 5$, which is violated by the optimal solution $(2, 2, 0, 0)$. In this case, we have that $a_{m+1} = (1, 1, 0, 0)$ and $a^\top B^{-1}A - a_{m+1}^\top = [0, 0, 2, -1]$. This modified problem then looks like

$$\begin{aligned} \text{min. } & -5x_1 - x_2 + 12x_3 \\ \text{s.t.: } & 3x_1 + 2x_2 + x_3 = 10 \\ & 5x_1 + 3x_2 + x_4 = 16 \\ & -x_1 - x_2 + x_5 = -5 \\ & x_1, \ldots, x_4 \geq 0 \end{aligned}$$

with associated tableau

	x_1	x_2	x_3	x_4	x_5	RHS
z	0	0	2	7	0	12
x_1	1	0	-3	2	0	2
x_2	0	1	5	-3	0	2
x_5	0	0	2	-1	1	-1

Notice that this tableau indicate that we have a dual feasible solution that is not primal feasible, and thus suitable to be solved using dual simplex.

A final point to note is that these operations are related to each other in terms of equivalent primal-dual formulations. That is, consider dual of P, which is given by

$$\begin{aligned} D: \text{max. } & p^\top b \\ \text{s.t.: } & p^\top A \leq c. \end{aligned}$$

Then, adding a constraint of the form $p^\top A_{n+1} \leq c_{n+1}$ is equivalent to adding a variable to P, exactly as discussed in Section 6.1.1.

6.1.3 Changing input data

We now consider how changes in the input data can influence the optimality of a given basis. Specifically, we consider how to predict whether changes in the vector of independent terms b and objective coefficients c will affect the optimality of the problem. Notice that variations in the coefficient matrix A are left aside.

Changes in the vector b

Suppose that some component b_i changes and becomes $b_i + \delta$, with $\delta \in \mathbb{R}$. We are interested in the range for δ within which the basis B remains optimal.

First, we must notice that optimality conditions $\overline{c} = c^\top - c_B^\top B^{-1}A \geq 0$ are not directly affected by variation in the vector b. That means that the choice of variable indices $j \in I_B$ to form the basis B will in principle be stable, unless the change in b_i is such that B is rendered *infeasible*. Thus, we need to study the conditions in which feasibility is retained, or, more specifically, if

$$B^{-1}(b + \delta e_i) \geq 0.$$

Let $g = (g_{1i}, \ldots, g_{mi})$ be the i^{th} column of B^{-1}. Thus

$$B^{-1}(b + \delta e_i) \geq 0 \Rightarrow x_B + \delta g \geq 0 \Rightarrow x_{B(j)} + \delta g_{ji} \geq 0, j = 1, \ldots, m.$$

Notice that this is equivalent to having δ within the range

$$\max_{j:g_{ji}>0} \left(-\frac{x_{B(j)}}{g_{ji}} \right) \leq \delta \leq \min_{j:g_{ji}<0} \left(-\frac{x_{B(j)}}{g_{ji}} \right).$$

In other words, changing b_i will incurr changes in the value of the basic variables and, thus, we are analysing what is the range within which all basic variables remain nonnegative (i.e., feasible).

Let us consider a numerical example. Once again, consider the problem from Section 6.1.1. The optimal tableau was given by

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

Suppose that b_1 will change by δ in the constraint $3x_1 + 2x_2 + x_3 = 10$. Notice that the first column of B^{-1} can be directly extracted from the optimal tableau, and is given by $(-3, 5)$. The optimal basis will remain feasible if $2 - 3\delta \geq 0$ and $2 + 5\delta \geq 0$, and thus $-2/5 \leq \delta \leq 2/3$.

Notice that this means that we can calculate what would be the change in the objective function value as a function of $\delta \in [-2/5, 2/3]$. Within this range, the optimal cost changes as

$$c_B^\top(b + \delta e_i) = p^\top b + \delta p_i,$$

where $p^\top = c_B^\top B^{-1}$ is the optimal dual solution. In case the variation falls outside that range, this means that some of the basic variables will become negative. However, since the dual feasibility conditions are not affected by changes in b_i, one can still reutilise the basis B using dual simplex.

Changes in the vector c

We now consider the case where variations are expected in the objective function coefficients. Suppose that some component c_j becomes $c_j + \delta$. In this case, optimality conditions become a concern. Two scenarios can occur. First, it might be that the changing coefficient is associated with a variable $j \in J$ that happens to be nonbasic ($j \in I_N$) in the optimal solution. In this case, we have that optimality will be retained as long as the nonbasic variable remains "not attractive", i.e., the reduced cost associated with j remains nonnegative. More precisely put, the basis B will remain optimal if

$$(c_j + \delta) - c_B B^{-1} A_j \geq 0 \Rightarrow \delta \geq -\overline{c}_j.$$

The second scenario concern changes in variables that are basic in the optimal solution, i.e., $j \in I_B$. In that case, the optimality conditions are directly affected, meaning that we have to analyse the range of variation for δ within which the optimality conditions are maintained, i.e., the reduced costs remain nonnegative.

Let c_j is the coefficient of the l^{th} basic variable, that is $j = B(l)$. In this case, c_B becomes $c_B + \delta e_l$, meaning that all of the optimality conditions are simultaneously affected. Thus, we have to define a range for δ in which the condition

$$(c_B + \delta e_i)^\top B^{-1} A_i \leq c_i, \forall i \neq j$$

holds. Notice that we do not need to consider j, since x_j is a basic variable and thus, its reduced costs is assumed to stay at zero.

Considering the tableau representation, we can use the l^{th} row and examine the conditions for which $\delta q_{li} \leq \bar{c}_i, \forall i \neq j$, where q_{li} is the l^{th} entry of $B^{-1}A_i$.

Let us once again consider the previous example, with optimal tableau

	x_1	x_2	x_3	x_4	RHS
z	0	0	2	7	12
x_1	1	0	-3	2	2
x_2	0	1	5	-3	2

First, let us consider variations in the the objective function coefficients of variables x_3 and x_4. Since both variables are nonbasic in the optimal basis, the allowed variation for them is given by

$$\delta_3 \geq -\bar{c}_3 = -2 \text{ and } \delta_4 \geq -\bar{c}_4 = -7.$$

Two points to notice. First, notice that both intervals are one-sided. This means that one should only be concerned with variations that decrease the reduced cost value, since increases in their value can never cause any changes in the optimality conditions. Second, notice that the allowed variation is trivially the negative value of the reduced cost. For variations that turn the reduced costs negative, the current basis can be utilised as a starting point for the primal simplex.

Now, let us consider a variation in the basic variable x_1. Notice that in this case we have to analyse the impact in all reduced costs, with exception of x_1 itself. Using the tableau, we have that $q_l = [1, 0, -3, 2]$ and thus

$$\begin{aligned}
&\delta_1 q_{12} \leq \bar{c}_2 \Rightarrow 0 \leq 0 \\
&\delta_1 q_{13} \leq \bar{c}_3 \Rightarrow \delta_1 \geq -2/3 \\
&\delta_1 q_{14} \leq \bar{c}_4 \Rightarrow \delta_1 \leq 7/2,
\end{aligned}$$

implying that $-2/3 \leq \delta_1 \leq 7/2$. Like before, for a change outside this range, primal simplex can be readily employed.

6.2 Cones and extreme rays

We now change the course of our discussion towards some results that will be useful in identifying two non-ordinary situations when employing the simplex method: unboundedness and infeasibility. Typically, these are consequences of issues related with the data and/or with modelling assumptions

and are challenging, in that they prevent us from obtaining a solution from the model. As we will see, they also rely on duality in their derivations, and are all connected by the notion of cones, which we formally give in Definition 6.1.

Definition 6.1 (Cones)**.** *A set $C \subset \mathbb{R}$ is a cone if $\lambda x \in C$ for all $\lambda \geq 0$ and all $x \in C$.*

A cone C can be understood as a set formed by the nonnegative scaling of a collection of vectors $x \in C$. Notice that it implies that $0 \in C$. Often, it will be the case that $0 \in C$ is an extreme point of C and in that case, we say that C is *pointed*. As one might suspect, in the context of linear programming, we will be mostly interested in a specific type of cone, those known are *polyhedral cones*. Polyhedral cones are sets of the form

$$P = \{x \in \mathbb{R}^n : Ax \geq 0\}$$

Figure 6.1 illustrates a polyhedral cone in $\mathbb{R}^3$ formed by the intersection of three half-spaces.

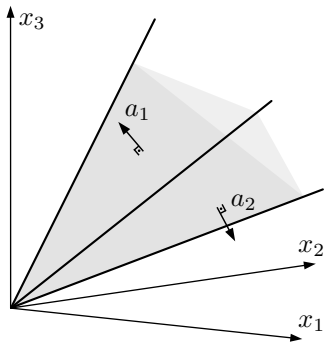


Figure 6.1: A polyhedral cone in $\mathbb{R}^3$ formed by 3 half-space

Some interesting properties can be immediately concluded regarding polyhedral cones. First, they are convex sets, since they are polyhedral sets (cf. Theorem 2.8). Also, the origin is an extreme point and thus, polyhedral cones are always pointed. Furthermore, just like general polyhedral sets, a cone $C \in \mathbb{R}^n$ will always be associated with a collection of n linearly independent vectors. Corollary 6.2 summarises these points. Notice we pose it as a Corollary, because these are immediate consequences of Theorem 3.5.

Corollary 6.2. *Let $C \subset \mathbb{R}^n$ be a polyhedral cone defined by constraints $\left\{a_i^\top x \geq 0\right\}_{i=1,\ldots,m}$. Then the following are equivalent*

1. *0 is an extreme point of C;*
2. *C does not contain a line;*
3. *There exists n vectors in $a_1, \ldots, a_m$ that are LI.*

Notice that $0 \in C$ is the unique extreme point of the polyhedra cone C. To see that, let $0 \neq x \in C$, $x_1 = (1/2)x$ and $x_2 = (3/2)x$. Note that $x_1, x_2 \in C$, and $x \neq x_1 \neq x_2$. Setting $\lambda_1 = \lambda_2 = 1/2$, we have that $\lambda_1 x_1 + \lambda_2 x_2 = x$ and thus, x is not an extreme point (cf. Definition 2.10).

6.2.1 Recession cones and extreme rays

We now focus on a specific type of cone, called *recession cones*. In the context of linear optimisation, recession cones are useful for helping us identifying directions of unboundedness. Let us first formally define the concept.

Definition 6.3 (Recession cone)**.** *Consider the polyhedral set $P = \{x \in \mathbb{R}^n : Ax \geq b\}$. The recession cone at $\overline{x} \in P$, denoted* $\mathbf{recc}(P)$*, is defined as*

$$\mathbf{recc}(P) = \{d \in \mathbb{R}^n : A(\overline{x} + \lambda d) \geq b, \lambda \geq 0\} \text{ or } \{d \in \mathbb{R}^n : Ad \geq 0\}.$$

Notice that the definition states that a recession cone comprises all directions d along which one can move from $\overline{x} \in P$ without ever leaving P. However, notice that the definition do not depend on $\overline{x}$, meaning that the recession cone is unique for the polyhedral set P, regardless of its "origin". Furthermore, notice that Definition 6.3 implies that recession cones of polyhedral sets are polyhedral cones.

We say that any directions $d \in \mathbf{recc}(P)$ is a *ray*. Thus, another way of thinking of bounded polyhedra is to think of polyhedral sets that do not contain rays.

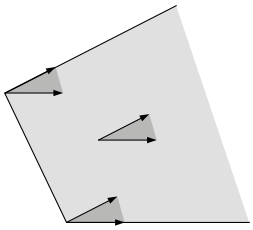


Figure 6.2: Representation of the recession cone of a polyhedral set

Figure 6.2 illustrates the concept of recession cones. Notice that it is purposely placed in several places to illustrate the independence of the point $\overline{x} \in P$.

Finally, the recession cone for a standard form polyhedral set $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ is given by

$$\mathbf{recc}(P) = \{d \in \mathbb{R}^n : Ad = 0, d \geq 0\}.$$

6.2.2 Unbounded problems

To identify unboundedness in linear programming problems, we must check for the existence of *extreme rays*. Extreme rays are the analogous of an extreme point, but defined with a "loose" degree of freedom. Definition 6.4 provide a technical definition of extreme rays.

Definition 6.4 (Extreme ray)**.** *Let $C \subset \mathbb{R}^n$ be a nonempty polyhedral cone. A nonzero $x \in C$ is an extreme ray if there are $n-1$ linearly independent active constraints at x.*

Notice that we are interested in extreme rays of the recession cone $\mathbf{recc}(P)$ of the polyhedral set P. However, it is typical to say that they are extreme rays of P. Figure 6.3 illustrates the concept of extreme rays in polyhedral cones.

Notice that, just like extreme points, the number of extreme rays is finite, by definition. In fact, we say that two extreme rays are equivalent if they are positive multiples corresponding to the same $n-1$ linearly independent active constraints.

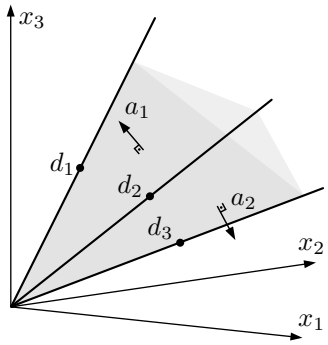


Figure 6.3: A polyhedral cone formed by the intersection of three half-spaces (the normal vector a_3 is perpendicular to the plane of the picture and cannot be seen). Directions d_1, d_2, and d_3 represent extreme rays.

The existence of extreme rays can be used to verify for unboundedness in linear programming problems. The mere existence of extreme rays does not suffice, since unboundedness is a consequence of the extreme ray being a direction of improvement for the objective function. To demonstrate this, let us first describe unboundedness in polyhedral cones, which we can then use to show the unboundedness in polyhedral sets.

Theorem 6.5 (Unboundedness in polyhedral cones)**.** *Let* (P) : *min.* $\{c^\top x : x \in C\}$*, with* $C = \{a_i^\top x \geq 0, i = 1, \ldots, m\}$*. The optimal value is equal to* $-\infty$ *if and only if some extreme ray* $d \in C$ *satisfies* $c^\top d < 0$*.*

Proof. If $c^\top d < 0$, then P is unbounded, since $c^\top x \to -\infty$ along d. Also, there exists some $x \in C$ for which $c^\top x < 0$ that can be scaled to -1.

Let $P = \{x \in \mathbb{R}^n : a_i^\top x \geq 0, i = 1, \ldots, m, c^\top x = -1\}$. Since $0 \in C$, $\{a_i\}_{i=1}^m$ span $\mathbb{R}^n$ and thus P has at least one extreme point (cf. Theorem 3.5). Let d be one of those. As we have n linearly-independent active constraints at d, $n-1$ of the constraints $\{a_i^\top x \geq 0\}_{i=1}^m$ must be active (plus $c^\top x = -1$), and thus d is an extreme ray. □

We can now expand the result to general polyhedral sets.

Theorem 6.6 (Unboundedness in polyhedral sets)**.** *Let* (P) : *min.* $\{c^\top x : x \in X\}$ *with* $X = \{x \in \mathbb{R}^n : Ax \geq b\}$ *and assume that the feasible set has at least one extreme point. Then, the optimal value is* $-\infty$ *if and only if* $c^\top d < 0$*.*

Proof. As before, if $c^\top d < 0$, then P is unbounded, since $c^\top x \to -\infty$ along d. Now, let (D) : max. $\{p^\top b : p^\top A = c^\top, p \geq 0\}$. Recall that, if P is unbounded, then D is infeasible, and so must be (D^0) : max. $\{p^\top 0 : p^\top A = c^\top, p \geq 0\}$. This implies that the primal (P^0) : min. $\{c^\top x : Ax \geq 0\}$ is unbounded (as 0 is feasible).

The existence of at least one extreme point for P implies that the rows $\{a_i\}_{i=1,\ldots,m}$ of A span $\mathbb{R}^n$ and $\mathbf{recc}(X) = \{x \in \mathbb{R}^n : Ax \geq 0\}$ is pointed. Thus, by Theorem 6.5 there exists d such that $c^\top d < 0$. □

We now focus on the question of how this can be utilised in the context of the simplex method. It turns out that, once unboundedness is identified in the simplex method, one can extract the

extreme ray causing the said unboundedness. In fact, most professional-grade solvers are capable of returning extreme (or unbounded) rays, which is helpful in the process of understanding the causes for unboundedness in the model. We will also see in the next chapter that this extreme rays are also used in the context of specialised solution methods.

To see that is possible, let (P) : min. $\left\{c^\top x : x \in X\right\}$ with $X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ and assume that, for a given basis B, we conclude that the optimal value is $-\infty$, that is, the problem is unbounded. In the context of the simplex method, this implies that we found a nonbasic variable x_j for which the reduced cost $\bar{c}_j < 0$ and the j^{th} column of $B^{-1}A_j$ has no positive coefficient. Nevertheless, we can still form the feasible direction $d = [d_B \; d_N]$ as before, with

$$d_B = -B^{-1}A_j \text{ and } d_N = \begin{cases} d_j = 1 \\ d_i = 0, \forall i \in I_N \setminus \{j\}. \end{cases}$$

This direction d is precisely an extreme ray for P. To see that, first, notice that $Ad = 0$ and $d \geq 0$, thus $d \in \mathbf{recc}(X)$. Moreover, there are $n-1$ active constraints at d: m in $Ad = 0$ and $n - m - 1$ in $d_i = 0$ for $i \in I_N \setminus \{j\}$. The last thing to notice is that $\bar{c}_j = c^\top d < 0$, which attests the unboundedness in the direction d.

6.2.3 Farkas' lemma

We now focus on the idea of generating certificates of infeasibility for linear programming problems. That is, we show that if a problem P is infeasible, then there is a structure that can be identified to attest the infeasibility. To see how this works, consider the the two polyhedral sets

$$X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\} \text{ and}$$
$$Y = \left\{p \in \mathbb{R}^m : p^\top Ax \geq 0, p^\top b < 0\right\}.$$

If there exists any $p \in Y$, then there is no $x \in X$ for which $p^\top Ax = p^\top b$, (and in turn $Ax = b$), holds. Thus, X must be empty. Notice that this can be used to infer that a problem P with a feasibility set represented by X prior to solving P itself, by means of solving the *feasibility problem* of finding a vector $p \in Y$.

We now pose this relationship more formally, via a result that is generally known as the Farkas' lemma.

Theorem 6.7 (Farkas' lemma)**.** *Let A be a $m \times n$ matrix and $b \in \mathbb{R}^m$. Then, exactly one of the following statements hold*

1. *There exists some $x \geq 0$ such that $Ax = b$;*
2. *there exists some vector p such that $p^\top A \geq 0$, $p^\top b < 0$.*

Proof. Assume that (1) is satisfied. If $p^\top A \geq 0$, then $p^\top b = p^\top Ax \geq 0$, which violates (2).

Now, consider the primal-dual pair (P) : min. $\left\{0^\top x : Ax = b, x \geq 0\right\}$ and (D) : max. $\left\{p^\top b : p^\top A \geq 0\right\}$. Being P infeasible, D must be unbounded (instead of infeasible), since $p = 0$ is feasible for D. Thus, $p^\top b < 0$ for some $p \neq 0$. □

The Farkas' lemma has a nice geometrical interpretation that represents the mutually exclusive relationship between the two sets. For that, notice that we can think of b as being a conic combination of the columns A_j of A, for some $x \geq 0$. If that cannot be the case, then there exists a

hyperplane that separates b and the cone formed by the columns of A, $C = \{y \in \mathbb{R}^m \mid y = Ax\}$. This is illustrated in Figure 6.4. Notice that the separation caused by such a hyperplane with normal vector p implies that $p^\top Ax \geq 0$ and $p^\top b < 0$, i.e., Ax and b are in the opposite sides of the hyperplane.

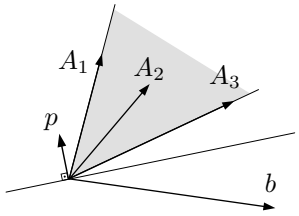


Figure 6.4: Since $b \notin X$, $p^\top x = 0$ separates them

6.3 Resolution theorem

We finalise with a result that will be the foundation for what we will discuss in the next chapter. Specifically, we will see how we can use a presentation of the polyhedral set P purely based on extreme points and extreme rays to devise solution strategies that can be more efficient for large scale linear programming problems.

For now, let us concentrate on this alternative representation. Basically, polyhedral sets in standard form can be represented in two manners: either by (i) a finite set of linear constraints; or (ii) by combinations of its extreme points and extreme rays.

Clearly, the first representation is far more practical than the second. That is, the second representation is an *explicit* representation that would require knowing beforehand each extreme point and extreme ray forming the polyhedral set. Notice that the first representation, which we have relied on so far, have extreme points and extreme rays only implicitly represented. However, we will see that this explicit representation has an important application in the devising of alternative solution method for large-scale linear programming problems.

This fundamental result is stated in Theorem 6.8.

Theorem 6.8 (Resolution theorem)**.** *Let $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ be a nonempty polyhedral set with at least one extreme point. Let $\{x_i\}_{i=1}^k$ be the extreme points and $\{w\}_{j=1}^r$ be a complete set of extreme rays. Then $P = Q$, where*

$$Q = \left\{ \sum_{i=1}^{k} \lambda_i x^i + \sum_{j=1}^{r} \theta_j w^j : \lambda_i \geq 0,\ \theta_j \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

Theorem 6.8 has an important consequence, as it states that bounded polyhedra, i.e., polyhedral sets that have no extreme rays, can be represented by the convex hull of its extreme points. For now, let us look at an example that illustrates the concept.

Consider the polyhedral set P given by

$$P = \{x_1 - x_2 \geq -2; x_1 + x_2 \geq 1, x_1, x_2 \geq 0\}.$$

The recession cone $C = \mathbf{recc}(P)$ is described by $d_1 - d_2 \geq 0$, $d_1 + d_2 \geq 0$ (from $Ad = 0$), and $d_1, d_2 \geq 0$, which can be simplified as

$$C = \left\{(d_1, d_2) \in \mathbb{R}^2 : 0 \leq d_2 \leq d_1\right\}.$$

We can then conclude that the two vectors $w^1 = (1, 1)$ and $w_2 = (1, 0)$ are extreme rays of P. Moreover, P has three extreme points: $x_1 = (0, 2)$, $x_2 = (0, 1)$, and $x_3 = (1, 0)$.

Figure 6.5 illustrates what is stated in Theorem 6.8. For example, a representation for the point $y = (2, 2) \in P$ is given by

$$y = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

that is, $y = x^2 + w^1 + w^2$. Notice however, that y could also be represented as

$$y = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{3}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

with then $y == \frac{1}{2}x^2 + \frac{1}{2}x^3 + \frac{3}{2}w^1$. Notice that this imply that the representation of each point is not unique.

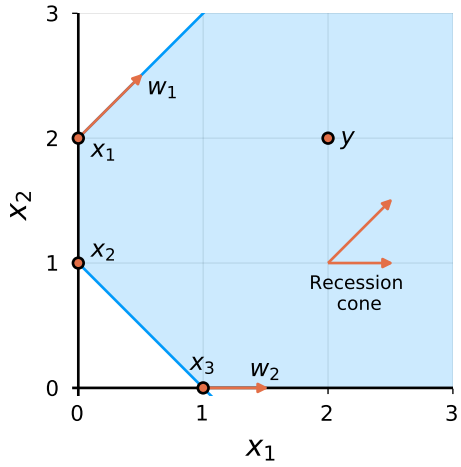


Figure 6.5: Example showing that every point of $P = \{x_1 - x_2 \geq -2; x_1 + x_2 \geq 1, x_1, x_2 \geq 0\}$ can be represented as a convex combination of its extreme point and a linear combination of its extreme rays

Exercises

Exercise 6.1: Sensitivity analysis in the RHS

Consider the following linear programming problem and its optimal tableau below:

$$\begin{aligned} \text{min. } & -2x_1 - x_2 + x_3 \\ \text{s.t.: } & x_1 + 2x_2 + x_3 \leq 8 \\ & -x_1 + x_2 - 2x_3 \leq 4 \\ & 3x_1 + x_2 \leq 10 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	0	1.2	0.2	0	0.6	-7.6
x_1	1	0	-0.2	-0.2	0	0.4	2.4
x_2	0	1	0.6	0.6	0	-0.2	2.8
x_5	0	0	-2.8	-0.8	1	0.6	3.6

(a) If you were to choose between increasing in 1 unit the right hand side of any constraints, which one would you choose, and why? What is the effect of the increase on the optimal cost?

(b) Perform a sensitivity analysis on the model to discover what is the range of alteration in the RHS in which the same effect calculated in item (a) can be expected. *HINT*: JuMP (from version 0.21.6) includes the function "lp_sensitivity_report()" that you can use to help performing the analysis.

Exercise 6.2: Extreme points and extreme rays

(a) Let $P = \{(x_1, x_2) : x_1 - x_2 = 0,\ x_1 + x_2 = 0\}$. What are the extreme points and the extreme rays of P?

(b) Let $P = \{(x_1, x_2) : 4x_1 + 2x_2 \geq 0,\ 2x_1 + x_2 \leq 1\}$. What are the extreme points and the extreme rays of P?

(c) For the polyhedron of part (b), is it possible to express each one of its elements as a convex combination of its extreme points plus a nonnegative linear combination of its extreme rays? Is this compatible with the Resolution Theorem?

Exercise 6.3: Unboundedness and duality

Use the Farkas' lemma to prove the duality theorem for a linear programming problem involving constraints of the form $a_i'x = b_i, a_i'x \geq b_i$, and nonegativity constraints for some of the variables x_j. *Hint*: Start by deriving the form of the set of feasible directions at an optimal solution.

Exercise 6.4: Dual in matrix form

Consider the the linear programming problem below with optimal basis $[x_1, x_2, x_5, x_6]$ and dual variables $p_1, \ldots, p_4$.

$$\begin{aligned} \text{max. } & 2x_1 + x_2 \\ \text{s.t.: } & 2x_1 + 2x_2 \leq 9 \quad (p_1) \\ & 2x_1 - x_2 \leq 3 \quad (p_2) \\ & x_1 \leq 3 \quad (p_3) \\ & x_2 \leq 4 \quad (p_4) \\ & x_1, x_2 \geq 0. \end{aligned}$$

(a) Find the primal and dual optimal solutions. *HINT*: You can use complementary slackness, once having the primal optimum, to finding the dual optimal solution.

(b) Suppose we add a new constraint $6x_1 - x_2 \leq 6$, classify the primal and dual former optimal points saying if they: (i) keep being optimal ; (ii) turn to be feasible but not optimal ; or (iii) turn infeasible.

(c) Consider the new problem from item (b), find the new dual optimal point through one dual simplex iteration. After that, find the primal optimum.

Chapter 7

Decomposition methods

7.1 Large-scale problems

We now consider the notion of *decomposition*, which consists of a general term used in the context of mathematical programming that refers to solution methods that utilise some separability mechanism to more efficiently solve large-scale problems.

In general, decomposition methods are based on the premise that it is more efficient, under a computational resource standpoint, to repeatedly resolve a (collection of) smaller instances of a linear programming problem than to solve the full-scale original problem. More recently, with the widespread adoption of multithreaded processors and computing clusters with multiple nodes, further perspectives have become attractive by means of parallelisation strategies, which can yield considerable computational savings.

There a mainly two-classes of decomposition methods. The first class utilises the explicit representation of polyhedral sets, as stated in Theorem 6.8, to iteratively reconstruct the full-scale problem, in the hope that the structure containing the optimal vertex will be successfully reconstructed at some point before all of the problem is reconstructed. It turns out that this is the case in many applications, which is precisely the feature that render these methods very efficient in some contexts. This is the class of methods we are going to analyse in this chapter, first the Dantzig-Wolfe decomposition and related column generation, and then its equivalent dual method, generally known as Benders' decomposition.

The second class of methods utilises Lagrangian duality for obtaining separability. We will delay the presentation of this sort of approach to Part **??**, when we discuss Lagrangian duality under the more general context of nonlinear programming problems.

In either case, decomposition methods are designed in a way that they seek to break problems into easier parts by removing linking elements. Specifically, let

$$(P) : \max. \ \left\{ c^\top x : x \in X \right\},$$

where $X = \bigcap_{k=1}^{K} X_k$, for some $K > 0$, and

$$X_k = \left\{ x^k \in \mathbb{R}^{n_k}_+ : D_k x_k = d_k \right\}, \forall k \in \{1, \ldots, K\}.$$

That is, X is the intersection of K standard-form polyhedral sets. Our objective is to devise a way to break into K separable parts that can be solved separately and recombined as a solution for P. In this case, this can be straightforwardly achieved by noticing that P can be equivalently stated as

$$\begin{array}{rccccc} (P): \max._{x} & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & & \\ \text{s.t.:} & A_1 x_1 & +\cdots+ & A_K x_K & = b & \\ & D_1 x_1 & & & = d_1 & \\ & & \ddots & & \vdots & \\ & & & D_K x_K & = d_K & \\ & x_1, & \ldots, & x_K & \in \mathbb{R}^n_+ & \end{array}$$

has a structure that immediately allows for separation. That is, P could be solved as K independent problems

$$P_k : \max. \left\{ c_k^\top x_k : x_k \in X_k \right\},$$

in parallel and then combine their individual solutions onto a solution for P, simply by making $\overline{x} = [x_k]_{k=1}^K$ and $c^\top \overline{x} = \sum_{i=1}^K c_k^\top \overline{x}_k$. Notice that, if we were to assume that the solution time scales linearly (it does not,; it grows faster than linear) and $K = 10$, then solving P like this would be ten fold faster (that is not true; there are bottlenecks and other considerations to take into account, but the point stands).

Unfortunately, *complicating structures* often compromise this natural separability, preventing one from being able to directly exploit this idea. Specifically, two types of complicating structures can be observed. The first is of the form of *complicating constraint*. That is, we observe that a constraint is such that it connects variables from multiple of the subsets X_k. In this case, we would notice that P has an additional constraint of the form

$$\begin{array}{rccccc} P' : \max._{x} & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & & \\ \text{s.t.:} & A_1 x_1 & +\cdots+ & A_K x_K & = b & \\ & D_1 x_1 & & & = d_1 & \\ & & \ddots & & \vdots & \\ & & & D_K x_K & = d_K & \\ & x_1, & \ldots, & x_K & \in \mathbb{R}^n_+. & \end{array}$$

The other type of complicating structure is the case in which the same set of decision variable is present in multiple constraints, or multiple subsets X_k. in this case, we observe that problem P takes the form of

$$\begin{array}{rccccc} P'' : \max._{x} & c_0^\top x_0 + & c_1^\top x_1 & +\cdots+ & c_K^\top x_K & \\ \text{s.t.:} & A_1 x_0 + & D_1 x_1 & & & = d_1 \\ & \vdots & & \ddots & & \vdots \\ & A_K x_0 + & & & D_K x_K & = d_K \\ & x_0, & x_1, & \ldots, & x_K & \in \mathbb{R}^n_+. \end{array}$$

The challenging aspect is that, depending on the type of complicating structure, a specific method becomes more suitable. Therefore, being able to identify these structures is one of the key success factors in terms of the chosen method performance. As a general rule, problems with complicating constraints (as P') are suitable to be solved by a delayed variable generation method such as column generation. Analogously, problem with complicating variables (P'')are better suited for employing delayed constraint generation methods such as Benders decomposition.

The development of professional-grade code employing decomposition methods is a somewhat recent occurrence. The commercial solver CPLEX offer a Benders decomposition implementation

that requires the user to specify the separable structure. On the other hand, although there are some available frameworks for implementing column generation-based methods, these tend to be more ad hoc occurrences, yet reaping impressive results.

7.2 Dantzig-Wolfe decomposition and column generation

We start with the Dantzig-Wolfe decomposition, which consists of an alternative approach for reducing memory requirements when solving large-scale linear programming problems. Then, we show how this can be expanded further with the notion of delayed generation to yield a truly decomposed problem.

7.2.1 Dantzig-Wolfe decomposition

As before, let $P_k = \{x_k \geq 0 : D_k x_k = d\}$, with $P_k \neq \emptyset$ for $k \in \{1, \ldots, K\}$. Then, the problem P can be reformulated as:

$$\begin{aligned} \text{min.} \quad & \sum_{k=1}^{K} c_k^\top x_k \\ \text{s.t.:} \quad & \sum_{k=1}^{K} A_k x_k = b \\ & x_k \in P_k, \ \forall k \in \{1, \ldots, K\}. \end{aligned}$$

Notice that P has a complicating constraint structure, due to the constraints $\sum_{k=1}^{K} A_k x_k = b$. In order to devise a decomposition method for this setting, let us first assume that we have available for each of the set P_k, $k \in K$, (i) all extreme points, represented by x_k^j, $\forall j \in J_k$; and (ii) all extreme rays w_k^r, $\forall r \in R_k$. As one might suspect, this is in principle a demanding assumption, but one that we will be able to drop later on.

Using the Resolution theorem (Theorem 6.8), we know that any element of P_k can be represented as

$$x_k = \sum_{j \in J_k} \lambda_k^j x_k^j + \sum_{r \in R_k} \theta_k^r w_k^r, \tag{7.1}$$

where $\lambda_k^j \geq 0$, $\forall j \in J_k$, are the coefficients of the convex combination of extreme points, meaning that we also observe $\sum_{j \in J_k} \lambda_k^j = 1$, and $\theta_k^r \geq 0$, $\forall r \in R_k$, are the coefficients of the conic combination of the extreme rays.

Using the identity represented in (7.1), we can reformulate P onto the *main problem* P_M as follows.

$$\begin{aligned} (P_M) : \text{min.} \quad & \sum_{k=1}^{K} \left(\sum_{j \in J_k} \lambda_k^j c_k^\top x_k^j + \sum_{r \in R_k} \theta_k^r c_k^\top w_k^r \right) \\ \text{s.t.:} \quad & \sum_{k=1}^{K} \left(\sum_{j \in J_k} \lambda_k^j A_k x_k^j + \sum_{r \in R_k} \theta_k^r A_k w_k^r \right) = b \qquad (7.2) \\ & \sum_{j \in J_k} \lambda_k^j = 1, \ \forall k \in \{1, \ldots, K\} \qquad (7.3) \\ & \lambda_k^j \geq 0, \theta_k^r \geq 0, \forall j \in J_k, r \in R_k, k \in \{k\}_{k=1}^{K}. \end{aligned}$$

Notice that (7.2) and (7.3) can be equivalently represented as

$$\sum_{k\in K}\left(\sum_{j\in J_k}\lambda_k^j\begin{bmatrix}A_k x_k^j\\ e_k\end{bmatrix}+\sum_{r\in R_k}\theta_k^r\begin{bmatrix}A_k w_k^r\\ 0\end{bmatrix}\right)=\begin{bmatrix}b\\ 1\end{bmatrix},$$

where e_k is the unit vector (i.e., with 1 in the k^{th} component, and 0 otherwise). Notice that P_M has as many variables as the number of extreme points and extreme rays of P, which is likely to be prohibitively large.

However, we can still solve it use a slightly modified version of the revised simplex method. To see that, let us consider that b is a m-dimensional vector. Then, basis for P_M would be of size $m+K$, since we have the original m constraints plus one for each convex combination (arising from each subproblem $k \in K$). This means that we are effectively working with $(m+K)\times(m+K)$ matrices, i.e., the basic matrix B and its inverse B^{-1}. Another element we need is the vector of simplex multipliers p, which is a vector of dimension $(m+K)$.

The issue with the representation adopted in P_M arises when we are required to calculate the reduced costs of *all* the nonbasic variables, since this is the critical issue for its tractability. That is where the method provides a clever solution. To see that, notice that the vector p is formed by components $p^\top = (q, r_1, \ldots, r_K)^\top$, where q represent the m dual variables associated with (7.2), and r_k, $\forall k \in \{1,\ldots,K\}$, are the dual variables associated with (7.3).

The reduced costs associated with the extreme-point variables λ_k^j, $j \in J_K$, is given by

$$c_k^\top x_k^j - [q^\top\ r_1\ \ldots\ r_K]\begin{bmatrix}A_k x_k^j\\ e_k\end{bmatrix} = (c_k^\top - q^\top A_k)x_k^j - r_k. \tag{7.4}$$

Analogously, the reduced cost associated with extreme-ray variables θ_k^r, $r \in R_k$, is

$$c_k^\top w_k^r - [q^\top\ r_1\ \ldots\ r_K]\begin{bmatrix}A_k w_k^r\\ 0\end{bmatrix} = (c_k^\top - q^\top A_k)w_k^r. \tag{7.5}$$

The main difference is how we assess the reduced costs of the non-basic variables. Instead of explicitly calculating the reduced costs of all variables, we instead rely on an optimisation-based approach to consider them only implicitly. For that, we can use the subproblem

$$\begin{aligned}(S_k): \min_x.\ & \bar{c}_k = (c_k^\top - q^\top A_k)x_k\\ \text{s.t.: } & x_k \in P_k,\end{aligned}$$

which can be solved in parallel for each subproblem $k \in \{1,\ldots,K\}$. The subproblem S_k is known as the *pricing problem*. For each subproblem $k = 1,\ldots,K$, we have the following cases.

We might observe that $\bar{c}_k = -\infty$. In this case, we have found an *extreme ray* w_k^r satisfying $(c_k^\top - q^\top A_k)w_k^r < 0$. Thus, the reduced cost of the associated extreme-ray variable θ_k^r is negative. If that is the case, we must generate the column

$$\begin{pmatrix}A_k w_k^r\\ 0\end{pmatrix}$$

associated with θ_k^r and make it enter the basis.

Otherwise, being S_k bounded, i.e., $\bar{c}_k < \infty$, two other cases can occur. The first is the case in which $\bar{c}_k < r_k$. Therefore, we found an extreme point x_k^j satisfying $(c_k^\top - q^\top A_k)x_k^j - r_k < 0$. Thus,

the reduced cost associated with the extreme-point variable λ_k^j is negative and, analogously, we must generate the column

$$\begin{pmatrix} A_k x_k^j \\ e_k \end{pmatrix}$$

associated with λ_k^j and make it enter the basis.

The last possible case is when we observe that $r_k < \overline{c}_k < \infty$. In this case, the pricing problem could not identify beneficial, and therefore there is not one, extreme point or ray with negative reduced cost for subproblem k. If this condition holds for all $k = 1, dots, K$, then all necessary extreme points and rays to characterise the region where the optimal extreme point (or one of the extreme points, in the case of multiple solutions) lies and the optimal solution can be recovered.

Algorithm 4 Dantzig-Wolfe decomposition

initialise. Let B be a BFS for P_M and set $l \leftarrow 0$.
repeat
  for $k \in \{1, \ldots, K\}$ **do**
    solve S_k and let $\overline{c}_k = \min_x \{S_k\}$
    if $\overline{c}_k = -\infty$ **then**
      obtain extreme ray w_k^r and make $R_k^l = R_k^l \cup \{w_k^r\}$.
      generate column $(A_k w_k^r, 0)$ to become basic.
    else if $\overline{c}_k < r_k < \infty$ **then**
      obtain extreme point x_k^j and make $J_k^l = J_k^l \cup \left\{x_k^j\right\}$.
      generate column $(A_k x_k^j, e_k)$ to become basic.
    end if
  end for
  select one of the generated columns to replace one of the columns of B and update B accordingly.
  $l \leftarrow l + 1$.
until $\overline{c}_k > r_k$ for all $k \in \{1, \ldots, K\}$
return B

Algorithm 4 summarises the Dantzig-Wolfe method. The two most remarkable features of the method are (i) the fact that columns are not explicitly represented, but generated "on demand" and (ii) the fact that the pricing problem requires the solution of another linear programming problem. Analogously to the simplex method, it might be necessary to employ a "Phase 1" approach to obtain an initial basis.

Under a computational standpoint, the use of the Dantzig-Wole method is typically not faster than the revised simplex method. There are however two settings where the decomposition is most favourable. The first, consists of applications in which the pricing problem can be solved in a closed-form, without invoking a method to solve an additional linear programming subproblem. There are a few examples in which this happens to be the case and certainly many others yet to be discovered.

Secondly, the memory requirements of the Dantzig-Wolfe decomposition makes it an interesting approach for very large-scale problems. The original simplex method requires an amount of memory space that is $O((m + K \times m_0)^2)$, where m_0 is the number of rows of D_k, for $\forall k \in \{1, \ldots, K\}$. This is essentially the size of the inverse basic matrix B^{-1}. In contrast, the Dantzig-Wolfe reformulation requires $O((m + K)^2) + K \times O(m_0^2)$ or memory space, being the first term referring to the main problem inverse basic matrix and the second to the pricing problems basic matrices. For

example, for a problem in which $m = m_0$ and much larger than, say, $K = 10$, this implies that the space memory space required by the Dantzig-Wolfe reformulation is 100 times smaller, which can substantially enlarge the range of large-scale problems that can be solved.

7.2.2 Delayed column generation

The term column generation can also refer to a related, and perhaps more wider known, variant of the Dantzig-Wolfe decomposition. In that, the main problem P_M is also repeatedly solved, each time being incremented by one additional variable, that associated with the column(s) identified with negative reduced costs in the pricing problem S_k, $k \in \{1, \ldots, K\}$. This is particularly useful for problems exponentially increasing number of variables, or with a large number of variables associated with the complicating constraints (i.e., when m is a large number).

The (delayed) column generation method is presented in Algorithm 5. Notice in Line 6 the step that is generating new columns in the main problem P_M, represented in the statement $\tilde{X}_k \leftarrow \tilde{X}_k \cup \{\overline{x}_k^{*l}\}$. That is precisely when a new variable λ_k^t is introduced in the P_M with coefficients represented by the column

$$\begin{pmatrix} c_k \overline{x}_k^{*l} \\ A_k \overline{x}_k^{*l} \\ e_k \end{pmatrix}.$$

Algorithm 5 Column generation algorithm

1: **initialise.** Let $\tilde{X}_k \subset X_k$, for $k \in \{1, \ldots, K\}$, and set $l \leftarrow 0$.
2: **repeat**
3: solve P_M^l to obtain $\lambda^{*l} = (\lambda_1^{*l}, \ldots, \lambda_K^{*l})$ and duals $(q^{*l}, \{r_k^{*l}\}_{k=1}^K)$.
4: **for** $k \in \{1, \ldots, K\}$ **do**
5: solve the pricing problem

$$\overline{x}_k^{*l} \leftarrow \operatorname{argmin} \{c_k^\top x_k - q^{*l}(A_k x_k) - r_k^{*l} : x_k \in P_k\}.$$

6: **if** $\overline{c}_k = c_k^\top \overline{x}_k^{*l} - q^{*l}(A_k \overline{x}_k^{*l}) < r_k^{*l}$ **then** $\tilde{X}_k \leftarrow \tilde{X}_k \cup \{\overline{x}_k^{*l}\}$
7: **end if**
8: **end for**
9: $l \leftarrow l + 1$.
10: **until** $\overline{c}_k > r_k$ for all $k \in \{1, \ldots, K\}$
11: **return** λ^{*l}.

Notice that the unbounded case is not treated to simplify the pseudocode, but could be trivially adapted to return extreme rays to be used in P_M, like the previous variant. Also, notice that the method is assumed to be initialised with a collection of columns (i.e., extreme points) $\tilde{X}_k$, which can normally be obtained from inspection or using a specialised method

We finalise showing that the Dantzig-Wolfe and column generation methods can provide information related to its own convergence. This means that we have access to an optimality bound tat can be used to monitor the convergence of the method and allow for a preemptive termination of the method given an acceptable tolerance. This bounding property is stated in Theorem 7.1.

Theorem 7.1. *Suppose P_M is feasible with finite optimal value $\overline{z}$. Let z be the cost associated with P_M at a given iteration l of the DW method. Also, let r_k be the dual variable associated with*

the convex combination of the k^{th} subproblem and z_k its optimal cost. Then

$$z + \sum_{k=1}^{K}(z_k - r_k) \leq \overline{z} \leq z.$$

Proof. $\overline{z} \leq z$ is because a solution for P_M is primal feasible and thus feasible for P.

Now, consider the dual of P_M

$$\begin{aligned}(D_M): \quad \max. \quad & q^\top b + \sum_{k=1}^{K} r_k \\ \text{s.t.:} \quad & q^\top A_k x_k^j + r_k \leq c_k^\top x_k^j, \ \forall j \in J_k, \forall k \in \{1,\dots,K\} \\ & q^\top A_k w_k^r \leq c_k^\top w_k^r, \ \forall r \in R_k, \forall k \in \{1,\dots,K\}\end{aligned}$$

We know that strong duality holds, and thus $z = q^\top b + \sum_{k=1}^{K} r_k$ for dual variables $(q, r_1, \dots, r_K)$. Now, since z_k are finite, we have $\min_{j \in J_k}(c_k^\top x_k^j - q^\top A_k x_k^j) = z_k$ and $\min_{r \in R_k}(c_k^\top w_k^r - q^\top D_k w_k^r) \geq 0$, meaning that $(q, z_1, \dots, z_K)$ is feasible to D_M. By weak duality, we have that

$$\overline{z} \geq q^\top b + \sum_{k=1}^{K} z_k = q^\top b + \sum_{k=1}^{K} r_k + \sum_{k=1}^{K}(z_k - r_k) = z + \sum_{k=1}^{K}(z_k - r_k). \qquad \square$$

7.3 Benders decomposition

Benders decomposition is an alternative decomposition method, suitable for settings in which one observe the presence of *complicating variables*. Differently than the Dantzig-Wolfe decomposition, the method presume from the get-go the employment of delayed constraint generation.

Benders decomposition has made significant inroads into practical applications. Not only it is extensively used in problems related to multi-period decision making under uncertainty, but it is also available in the commercial solver CPLEX to be used directly, only requiring the user to indicate (or annotate) which are the complicating variables.

Once again, let the problem P be defined as

$$\begin{aligned}(P): \min_{x,y} \ & c^\top x + \sum_{k=1}^{K} f_k^\top y_k \\ & Ax = b \\ & C_k x + D_k y_k = e_k, && k = 1,\dots,K \\ & x \geq 0, y_k \geq 0, && k = 1,\dots,K.\end{aligned}$$

Notice that this is equivalent to the problem P' presented in Section 7.1, but with a notation modified to make it easier to track how the terms are separated in the process. Again, we can see that P has a set of complicating variables x, which becomes obvious when we recast the problem as

$$
\begin{array}{ccccccccccc}
c^\top x & + & f_1^\top y_1 & + & f_2^\top y_2 & + & \dots & + & f_k^\top y_k & & \\
Ax & & & & & & & & & = & b \\
C_1 x & + & D_1 y_1 & & & & & & & = & e_1 \\
C_2 x & & & + & D_2 y_2 & & & & & = & e_2 \\
\vdots & & \vdots & & & & \ddots & & & & \vdots \\
C_K x & & & & & & & + & D_k y_k & = & e_K \\
x & & y_1 & & y_2 & & \dots & & y_k & \geq & 0
\end{array}
$$

This structure is sometimes referred to as block-angular, referring to the initial block of columns on the left (as many as there are components in x) and the diagonal structure representing the elements associated with the variables y. In this case, notice that if the variable x were to be removed, or fixed to a value $x = \overline{x}$, the problem becomes separable in K independent parts

$$
\begin{aligned}
(S_k) : \min_y \ & f_k^\top y_k \\
\text{s.t.: } & D_k y_k = e_k - C_k \overline{x} \\
& y_k \geq 0.
\end{aligned}
$$

Notice that these subproblems $k \in \{1, \dots, K\}$ can be solved in parallel and, in certain contexts, might even have analytical closed-form solutions. The part missing is the development of a coordination mechanism that would allow for iteratively update the solution $\overline{x}$ based on information emerging from the solution of the subproblems $k \in \{1, \dots, K\}$.

To see how that can be achieved, let us reformulate P as

$$
\begin{aligned}
(P_R) : \min_x \ & c^\top x + \sum_{k=1}^{K} z_k(x) \\
\text{s.t.: } & Ax = b \\
& x \geq 0.
\end{aligned}
$$

where, for $k \in \{1, \dots, K\}$,

$$
z_k(x) = \min_y \ \left\{ f_k^\top y_k : D_k y_k = e_k - C_k x \right\}.
$$

Note that obtaining $z_k(x)$ requires solving S_k, which, in turn, depends on x. To get around this difficulty, we can rely on linear programming duality, combined, once again, with the resolution theorem (Theorem 6.8).

First, let us consider the dual formulation of the subproblems $k \in \{1, \dots, K\}$, which is given by

$$
\begin{aligned}
(S_k^D) : z_k^D = \max. \ & p_k^\top (e_k - C_k x) \\
\text{s.t.: } & p_k^\top D_k \leq f_k.
\end{aligned}
$$

The main advantage of utilising the equivalent dual formulation is to "move" the original decision variable x to the objective function, a trick that will present its benefits shortly. Next, let

$$
P_k = \left\{ p : p^\top D_k \leq f_k \right\}, \forall k \in \{1, \dots, K\}, \tag{7.6}
$$

and assume that each $P_k \neq \emptyset$ with at least one extreme point. Relying on the representation theorem, we can define the sets of extreme points of P_k, given by p_k^i, $i \in I_k$, and extreme rays w_k^r, $r \in R_k$.

As we assume that $P_k \neq \emptyset$, two cases can occur when we solve S_k^D, $k \in \{1, \ldots, K\}$. Either S_k^D is unbounded, meaning that the relative primal subproblem is infeasible, or S_k^D is bounded, meaning that $z_k^D < \infty$.

From the first case, we can use Theorem 6.6 to conclude that primal feasibility (or a bounded dual value $z_k^D < \infty$) can only be attained if and only if

$$(w_k^r)^\top (e_k - C_k x) \leq 0, \ \forall r \in R_k. \tag{7.7}$$

Furthermore, we know that if S_k^D has a solution, that must lie on a vertex of P_k. So, having the available the set of all extreme vertices p_k^i, $i \in I_k$, we have that if one can solve S_k^D, it can be equivalently represented as

$$(S_k^D) : z_k(x) = \max_{i \in I_k} (p_k^i)^\top (e_k - C_k x),$$

which can be equivalently reformulated as

$$\text{min.} \ \ \theta_k \tag{7.8}$$

$$\text{s.t.:} \ \theta_k \geq (p_k^i)^\top (e_k - C_k x), \ \forall i \in I_k. \tag{7.9}$$

Combining (7.7)–(7.9), we can reformulate P_R into a single-level equivalent form

$$\begin{aligned}
(P_R) : \min_x \ & c^\top x + \sum_{k=1}^{K} \theta_k \\
\text{s.t.:} \ & Ax = b \\
& (p_k^i)^\top (e_k - C_k x) \leq \theta_k, \ \forall i \in I_k, \forall k \in \{1, \ldots, K\} && (7.10) \\
& (w_k^r)^\top (e_k - C_k x) \leq 0, \ \forall r \in R_k, \forall k \in \{1, \ldots, K\} && (7.11) \\
& x \geq 0.
\end{aligned}$$

Notice that, just like the reformulation used for the Dantzig-Wolfe method presented in Section 7.2, the formulation of P_R is of little practical use, since it requires the complete enumeration of (a typically prohibitive) number of extreme points and rays and is likely to be computationally intractable due to the large number of associated constraints. To address this issue, we can employ delayed constraint generation and iteratively generate only the constraints we observe to be violated.

By doing so, at a given iteration l, we have at hand a *relaxed main problem* P_M^l, which comprises only some of the extreme point and rays obtained until iteration l. The relaxed main problem can be stated as

$$\begin{aligned}
(P_M^l) : z_{P_M}^l = \min_x \ & c^\top x + \sum_{k=1}^{K} \theta_k \\
\text{s.t.:} \ & Ax = b \\
& (p_k^i)^\top (e_k - C_k x) \leq \theta_k, \ \forall i \in I_k^l, \forall k \in \{1, \ldots, K\} \\
& (w_k^r)^\top (e_k - C_k x) \leq 0, \ \forall r \in R_k^l, \forall k \in \{1, \ldots, K\} \\
& x \geq 0,
\end{aligned}$$

where $I_k^l \subseteq I_k$ represent subsets of extreme points p_k^i of P_k, and $R_k^l \subset R_k$ subsets of extreme rays w_k^r of P_k.

One can iteratively obtain these extreme points and rays from the subproblems S_k, $k \in \{1, \ldots, K\}$. To see that, let us fist define that, at iteration l, we solve the the main problem P_M^l and obtain a solution

$$\underset{x,\theta}{\operatorname{argmin}} \left\{ P_M^l \right\} = (\overline{x}^l, \left\{ \overline{\theta}_k^l \right\}_{k=1}^K)$$

We can then solve the subproblems S_k^l, $k \in \{1, \ldots, K\}$, for that fixed solution $\overline{x}^l$ and then observe if we can find additional constraints that were to be violated if they had been in the relaxed main problem in the first place. In other words, we can identify if the solution $\overline{x}^l$ allows for identifying additional extreme points p_k^i or extreme rays w_k^r of P_k that were not yet included in P_M^l.

To identify those, first recall that the subproblem (in its primal form), is given by

$$\begin{aligned} (S_k^l) : \min. \quad & f^\top y \\ \text{s.t.: } & D_k y_k = e_k - C_k \overline{x}^l \\ & y_k \geq 0. \end{aligned}$$

Then, two cases can lead to generating violated constraints that must be added to the relaxed primal problem to form P_M^{l+1}. The first is when S_k^l is feasible. In that case, a dual optimal basic feasible solution p_k^{il} is obtained. If $(p_k^{il})^\top (e_k - C_k \overline{x}^l) > \overline{\theta}_k^l$, then we can conclude that we just formed a violated constraint of the form of (7.10). The second case is when S_k^l is infeasible, then an extreme ray w_k^{rl} of P_k is available, such that $(w_k^{rl})^\top (e_k - C_k \overline{x}^l) > 0$, violating (7.11).

Notice that the above can also be accomplished by solving the dual subproblems S_k^D, $k \in \{1, \ldots, K\}$, instead. In that case, the extreme point p_k^{il} is immediately available and so are the extreme rays w_k^{rl} in case of unboundedness.

Algorithm 6 presents a pseudocode for the Benders decomposition. Notice that the method can benefit in terms of efficiency from the use of dual simplex, since we are iteratively adding violated constraints to the relaxed main problem P_M^l. Likewise, the subproblem S_k^l has only the independent terms being modified at each iteration and, in light of the discussion in Section 6.1.3, can also benefit from the use of dual simplex. Furthermore, the loop represented by Line 4 can be parallelised to provide further computational performance improvements.

Algorithm 6 Benders decomposition

1: **initialise.** Let $P_i^l = W_j^l = \emptyset$, for $k \in \{1, \ldots, K\}$, and set $l \leftarrow 0$.
2: **repeat**
3: solve P_M^l to obtain $\left(\overline{x}^l, \left\{ \overline{\theta}_k^l \right\}_{k=1}^K \right)$.
4: **for** $k \in \{1, \ldots, K\}$ **do**
5: solve S_k^l.
6: **if** S_k^l is infeasible **then**
7: obtain extreme ray w_j^k and make $W^l = W^l \cup \left\{ w_j^k \right\}$.
8: **else**
9: obtain extreme point p_i^k and $P^l = P^l \cup \left\{ p_i^k \right\}$
10: **end if**
11: **end for**
12: $l = l + 1$.
13: **until** $(p_k^i)^\top (e_k - C_k \overline{x}) \leq \overline{\theta}_k, \ \forall k \in \{1, \ldots, K\}$
14: **return** $\left(\overline{x}^l, \left\{ \overline{\theta}_k^l \right\}_{k=1}^K \right)$

Notice that the algorithm terminates if no violated constraint is found. This in practice implies that $(p_k^i)^\top(e_k - C_k\overline{x}) \leq \overline{\theta}_k$ for all $k \in K$, and thus $(\overline{x}, \left\{\overline{\theta}_k\right\}_{k=1}^K)$ is optimal for P. In a way, if one consider the dual version subproblem, S_k^D, one can notice that it is acting as an implicit search for values of p_k^i that can be larger than $overline\theta_k$, meaning that the current solution $\overline{x}$ violates 7.10 and is thus not feasible to P.

Also, every time one solves P_M^l, a dual (lower for minimisation) bound $LB^l = z_{P_M}^l$ is obtained. This is simply because the relaxed main problem is a relaxation of the P, i.e., contains less constraints than the original problem P. A primal (upper) bound can also be calculated at every iteration, which allows for keeping track of the progress of the algorithm in terms of convergence and preemptively terminate it at any arbitrary optimality tolerance. That can be achieved by setting

$$\begin{aligned} UB^l &= \min\left\{z_P^{k-1}, c^\top\overline{x}^l + \sum_{k=1}^K f^\top\overline{y}_k^l\right\} \\ &= \min\left\{z_P^{k-1}, z_{P_M}^l - \sum_{k=1}^K \theta_k + \sum_{k=1}^K z_D^{kl}\right\}, \end{aligned}$$

where $\overline{y}_k^l = \text{argmin}_y\left\{S_k^l\right\}$.

Part II

Nonlinear optimisation