Chapter 1

Introduction

Contents

In this chapter, we introduce *linear programming*, the problem of minimizing a linear cost function subject to linear equality and inequality constraints. We consider a few equivalent forms and then present a number of examples to illustrate the applicability of linear programming to a wide variety of contexts. We also solve a few simple examples and obtain some basic geometric intuition on the nature of the problem. The chapter ends with a review of linear algebra and of the conventions used in describing the computational requirements (operation count) of algorithms.

1.1 Variants of the linear programming problem

In this section, we pose the linear programming problem, discuss a few special forms that it takes, and establish some standard notation that we will be using. Rather than starting abstractly, we first state a concrete example, which is meant to facilitate understanding of the formal definition that will follow. The example we give is devoid of any interpretation. Later on, in Section 1.2, we will have ample opportunity to develop examples that arise in practical settings.

Example 1.1 The following is a linear programming problem:

$$\begin{array}{lrcrcrcl} \text{minimize} & 2x_1 & - & x_2 & + & 4x_3 & & \\ \text{subject to} & x_1 & + & x_2 & & & + x_4 & \leq 2 \\ & & & 3x_2 & - & x_3 & & = 5 \\ & & & & & x_3 & + x_4 & \geq 3 \\ & x_1 & & & & & & \geq 0 \\ & & & & & x_3 & & \leq 0. \end{array}$$

Here x_1, x_2, x_3, and x_4 are variables whose values are to be chosen to minimize the linear cost function $2x_1 - x_2 + 4x_3$, subject to a set of linear equality and inequality constraints. Some of these constraints, such as $x_1 \geq 0$ and $x_3 \leq 0$, amount to simple restrictions on the sign of certain variables. The remaining constraints are of the form $\mathbf{a}'\mathbf{x} \leq b$, $\mathbf{a}'\mathbf{x} = b$, or $\mathbf{a}'\mathbf{x} \geq b$, where $\mathbf{a} = (a_1, a_2, a_3, a_4)$ is a given vector[1], $\mathbf{x} = (x_1, x_2, x_3, x_4)$ is the vector of decision variables, $\mathbf{a}'\mathbf{x}$ is their inner product $\sum_{i=1}^{4} a_i x_i$, and b is a given scalar. For example, in the first constraint, we have $\mathbf{a} = (1, 1, 0, 1)$ and $b = 2$.

We now generalize. In a *general* linear programming problem, we are given a cost vector $\mathbf{c} = (c_1, \ldots, c_n)$ and we seek to minimize a linear cost function $\mathbf{c}'\mathbf{x} = \sum_{i=1}^{n} c_i x_i$ over all n-dimensional vectors $\mathbf{x} = (x_1, \ldots, x_n)$,

[1] As discussed further in Section 1.5, all vectors are assumed to be column vectors, and are treated as such in matrix-vector products. Row vectors are indicated as transposes of (column) vectors. However, whenever we refer to a vector $\mathbf{x}$ inside the text, we use the more economical notation $\mathbf{x} = (x_1, \ldots, x_n)$, even though $\mathbf{x}$ is a column vector. The reader who is unfamiliar with our notation may wish to consult Section 1.5 before continuing.

subject to a set of linear equality and inequality constraints. In particular, let M_1, M_2, M_3 be some finite index sets, and suppose that for every i in any one of these sets, we are given an n-dimensional vector $\mathbf{a}_i$ and a scalar b_i, that will be used to form the ith constraint. Let also N_1 and N_2 be subsets of $\{1, \ldots, n\}$ that indicate which variables x_j are constrained to be nonnegative or nonpositive, respectively. We then consider the problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}'\mathbf{x} \\ \text{subject to} \quad & \mathbf{a}_i'\mathbf{x} \geq b_i, \qquad i \in M_1, \\ & \mathbf{a}_i'\mathbf{x} \leq b_i, \qquad i \in M_2, \\ & \mathbf{a}_i'\mathbf{x} = b_i, \qquad i \in M_3, \\ & x_j \geq 0, \qquad j \in N_1, \\ & x_j \leq 0, \qquad j \in N_2. \end{aligned} \tag{1.1}$$

The variables $x_1, \ldots, x_n$ are called *decision variables*, and a vector $\mathbf{x}$ satisfying all of the constraints is called a *feasible solution* or *feasible vector*. The set of all feasible solutions is called the *feasible set* or *feasible region*. If j is in neither N_1 nor N_2, there are no restrictions on the sign of x_j, in which case we say that x_j is a *free* or *unrestricted* variable. The function $\mathbf{c}'\mathbf{x}$ is called the *objective function* or *cost function*. A feasible solution $\mathbf{x}^*$ that minimizes the objective function (that is, $\mathbf{c}'\mathbf{x}^* \leq \mathbf{c}'\mathbf{x}$, for all feasible $\mathbf{x}$) is called an *optimal feasible solution* or, simply, an *optimal solution*. The value of $\mathbf{c}'\mathbf{x}^*$ is then called the *optimal cost*. On the other hand, if for every real number K we can find a feasible solution $\mathbf{x}$ whose cost is less than K, we say that the optimal cost is $-\infty$ or that the cost is *unbounded below*. (Sometimes, we will abuse terminology and say that the problem is *unbounded*.) We finally note that there is no need to study maximization problems separately, because maximizing $\mathbf{c}'\mathbf{x}$ is equivalent to minimizing the linear cost function $-\mathbf{c}'\mathbf{x}$.

An equality constraint $\mathbf{a}_i'\mathbf{x} = b_i$ is equivalent to the two constraints $\mathbf{a}_i'\mathbf{x} \leq b_i$ and $\mathbf{a}_i'\mathbf{x} \geq b_i$. In addition, any constraint of the form $\mathbf{a}_i'\mathbf{x} \leq b_i$ can be rewritten as $(-\mathbf{a}_i)'\mathbf{x} \geq -b_i$. Finally, constraints of the form $x_j \geq 0$ or $x_j \leq 0$ are special cases of constraints of the form $\mathbf{a}_i'\mathbf{x} \geq b_i$, where $\mathbf{a}_i$ is a unit vector and $b_i = 0$. We conclude that the feasible set in a general linear programming problem can be expressed exclusively in terms of inequality constraints of the form $\mathbf{a}_i'\mathbf{x} \geq b_i$. Suppose that there is a total of m such constraints, indexed by $i = 1, \ldots, m$, let $\mathbf{b} = (b_1, \ldots, b_m)$, and let $\mathbf{A}$ be the $m \times n$ matrix whose rows are the row vectors $\mathbf{a}_1', \ldots, \mathbf{a}_m'$, that is,

$$\mathbf{A} = \begin{bmatrix} — & \mathbf{a}_1' & — \\ & \vdots & \\ — & \mathbf{a}_m' & — \end{bmatrix}.$$

Then, the constraints $\mathbf{a}_i'\mathbf{x} \geq b_i$, $i = 1, \ldots, m$, can be expressed compactly in the form $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, and the linear programming problem can be written

as

$$\begin{array}{ll} \text{minimize} & \mathbf{c}'\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b}. \end{array} \tag{1.2}$$

Inequalities such as $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ will always be interpreted componentwise; that is, for every i, the ith component of the vector $\mathbf{A}\mathbf{x}$, which is $\mathbf{a}_i'\mathbf{x}$, is greater than or equal to the ith component b_i of the vector $\mathbf{b}$.

Example 1.2 The linear programming problem in Example 1.1 can be rewritten as

$$\begin{array}{lrcrcrcl} \text{minimize} & 2x_1 & - & x_2 & + & 4x_3 & & \\ \text{subject to} & -x_1 & - & x_2 & & & - x_4 & \geq -2 \\ & & & 3x_2 & - & x_3 & & \geq 5 \\ & & - & 3x_2 & + & x_3 & & \geq -5 \\ & & & & & x_3 & + x_4 & \geq 3 \\ & x_1 & & & & & & \geq 0 \\ & & & & - & x_3 & & \geq 0, \end{array}$$

which is of the same form as the problem (1.2), with $\mathbf{c} = (2, -1, 4, 0)$,

$$\mathbf{A} = \begin{bmatrix} -1 & -1 & 0 & -1 \\ 0 & 3 & -1 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

and $\mathbf{b} = (-2, 5, -5, 3, 0, 0)$.

Standard form problems

A linear programming problem of the form

$$\begin{array}{lrcl} \text{minimize} & \mathbf{c}'\mathbf{x} & & \\ \text{subject to} & \mathbf{A}\mathbf{x} & = & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{1.3}$$

is said to be in *standard form*. We provide an interpretation of problems in standard form. Suppose that $\mathbf{x}$ has dimension n and let $\mathbf{A}_1, \ldots, \mathbf{A}_n$ be the columns of $\mathbf{A}$. Then, the constraint $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be written in the form

$$\sum_{i=1}^{n} \mathbf{A}_i x_i = \mathbf{b}.$$

Intuitively, there are n available resource vectors $\mathbf{A}_1, \ldots, \mathbf{A}_n$, and a target vector $\mathbf{b}$. We wish to "synthesize" the target vector $\mathbf{b}$ by using a nonnegative amount x_i of each resource vector $\mathbf{A}_i$, while minimizing the cost $\sum_{i=1}^{n} c_i x_i$, where c_i is the unit cost of the ith resource. The following is a more concrete example.

Example 1.3 (The diet problem) Suppose that there are n different foods and m different nutrients, and that we are given the following table with the nutritional content of a unit of each food:

	food 1	$\cdots$	food n
nutrient 1	a_{11}	$\cdots$	a_{1n}
$\vdots$	$\vdots$		$\vdots$
nutrient m	a_{m1}	$\cdots$	a_{mn}

Let $\mathbf{A}$ be the $m \times n$ matrix with entries a_{ij}. Note that the jth column $\mathbf{A}_j$ of this matrix represents the nutritional content of the jth food. Let $\mathbf{b}$ be a vector with the requirements of an ideal diet or, equivalently, a specification of the nutritional contents of an "ideal food." We then interpret the standard form problem as the problem of mixing nonnegative quantities x_i of the available foods, to synthesize the ideal food at minimal cost. In a variant of this problem, the vector $\mathbf{b}$ specifies the *minimal* requirements of an adequate diet; in that case, the constraints $\mathbf{Ax} = \mathbf{b}$ are replaced by $\mathbf{Ax} \geq \mathbf{b}$, and the problem is not in standard form.

Reduction to standard form

As argued earlier, any linear programming problem, including the standard form problem (1.3), is a special case of the general form (1.1). We now argue that the converse is also true and that a general linear programming problem can be transformed into an equivalent problem in standard form. Here, when we say that the two problems are equivalent, we mean that given a feasible solution to one problem, we can construct a feasible solution to the other, with the same cost. In particular, the two problems have the same optimal cost and given an optimal solution to one problem, we can construct an optimal solution to the other. The problem transformation we have in mind involves two steps:

(a) *Elimination of free variables:* Given an unrestricted variable x_j in a problem in general form, we replace it by $x_j^+ - x_j^-$, where x_j^+ and x_j^- are new variables on which we impose the sign constraints $x_j^+ \geq 0$ and $x_j^- \geq 0$. The underlying idea is that any real number can be written as the difference of two nonnegative numbers.

(b) *Elimination of inequality constraints:* Given an inequality constraint of the form

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i,$$

we introduce a new variable s_i and the standard form constraints

$$\sum_{j=1}^{n} a_{ij}x_j + s_i = b_i,$$
$$s_i \geq 0.$$

Such a variable s_i is called a *slack* variable. Similarly, an inequality constraint $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ can be put in standard form by introducing a *surplus* variable s_i and the constraints $\sum_{j=1}^{n} a_{ij}x_j - s_i = b_i,\ s_i \geq 0$.

We conclude that a general problem can be brought into standard form and, therefore, we only need to develop methods that are capable of solving standard form problems.

Example 1.4 The problem

$$\begin{array}{rrcl} \text{minimize} & 2x_1 + 4x_2 & & \\ \text{subject to} & x_1 + x_2 & \geq & 3 \\ & 3x_1 + 2x_2 & = & 14 \\ & x_1 & \geq & 0, \end{array}$$

is equivalent to the standard form problem

$$\begin{array}{rrcl} \text{minimize} & 2x_1 + 4x_2^+ - 4x_2^- & & \\ \text{subject to} & x_1 + x_2^+ - x_2^- - x_3 & = & 3 \\ & 3x_1 + 2x_2^+ - 2x_2^- & = & 14 \\ & x_1, x_2^+, x_2^-, x_3 \geq 0. & & \end{array}$$

For example, given the feasible solution $(x_1, x_2) = (6, -2)$ to the original problem, we obtain the feasible solution $(x_1, x_2^+, x_2^-, x_3) = (6, 0, 2, 1)$ to the standard form problem, which has the same cost. Conversely, given the feasible solution $(x_1, x_2^+, x_2^-, x_3) = (8, 1, 6, 0)$ to the standard form problem, we obtain the feasible solution $(x_1, x_2) = (8, -5)$ to the original problem with the same cost.

In the sequel, we will often use the general form $\mathbf{Ax} \geq \mathbf{b}$ to develop the theory of linear programming. However, when it comes to algorithms, and especially the simplex and interior point methods, we will be focusing on the standard form $\mathbf{Ax} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}$, which is computationally more convenient.

1.2 Examples of linear programming problems

In this section, we discuss a number of examples of linear programming problems. One of our purposes is to indicate the vast range of situations to which linear programming can be applied. Another purpose is to develop some familiarity with the art of constructing mathematical formulations of loosely defined optimization problems.

A production problem

A firm produces n different goods using m different raw materials. Let b_i, $i = 1, \ldots, m$, be the available amount of the ith raw material. The jth good, $j = 1, \ldots, n$, requires a_{ij} units of the ith material and results in a revenue of c_j per unit produced. The firm faces the problem of deciding how much of each good to produce in order to maximize its total revenue.

In this example, the choice of the decision variables is simple. Let x_j, $j = 1, \ldots, n$, be the amount of the jth good. Then, the problem facing the firm can be formulated as follows:

$$\begin{array}{lrcl} \text{maximize} & c_1 x_1 + \cdots + c_n x_n & & \\ \text{subject to} & a_{i1} x_1 + \cdots + a_{in} x_n \leq b_i, & & i = 1, \ldots, m, \\ & x_j \geq 0, & & j = 1, \ldots, n. \end{array}$$

Production planning by a computer manufacturer

The example that we consider here is a problem that Digital Equipment Corporation (DEC) had faced in the fourth quarter of 1988. It illustrates the complexities and uncertainties of real world applications, as well as the usefulness of mathematical modeling for making important strategic decisions.

In the second quarter of 1988, DEC introduced a new family of (single CPU) computer systems and workstations: GP-1, GP-2, and GP-3, which are general purpose computer systems with different memory, disk storage, and expansion capabilities, as well as WS-1 and WS-2, which are workstations. In Table 1.1, we list the models, the list prices, the average disk usage per system, and the memory usage. For example, GP-1 uses four 256K memory boards, and 3 out of every 10 units are produced with a disk drive.

System	Price	# disk drives	# 256K boards
GP-1	\$60,000	0.3	4
GP-2	\$40,000	1.7	2
GP-3	\$30,000	0	2
WS-1	\$30,000	1.4	2
WS-2	\$15,000	0	1

Table 1.1: Features of the five different DEC systems.

Shipments of this new family of products started in the third quarter and ramped slowly during the fourth quarter. The following difficulties were anticipated for the next quarter:

(a) The in-house supplier of CPUs could provide at most 7,000 units, due to debugging problems.

(b) The supply of disk drives was uncertain and was estimated by the manufacturer to be in the range of 3,000 to 7,000 units.

(c) The supply of 256K memory boards was also limited in the range of 8,000 to 16,000 units.

On the demand side, the marketing department established that the maximum demand for the first quarter of 1989 would be 1,800 for GP-1 systems, 300 for GP-3 systems, 3,800 systems for the whole GP family, and 3,200 systems for the WS family. Included in these projections were 500 orders for GP-2, 500 orders for WS-1, and 400 orders for WS-2 that had already been received and had to be fulfilled in the next quarter.

In the previous quarters, in order to address the disk drive shortage, DEC had produced GP-1, GP-3, and WS-2 with no disk drive (although 3 out of 10 customers for GP-1 systems wanted a disk drive), and GP-2, WS-1 with one disk drive. We refer to this way of configuring the systems as the constrained mode of production.

In addition, DEC could address the shortage of 256K memory boards by using two alternative boards, instead of four 256K memory boards, in the GP-1 system. DEC could provide 4,000 alternative boards for the next quarter.

It was clear to the manufacturing staff that the problem had become complex, as revenue, profitability, and customer satisfaction were at risk. The following decisions needed to be made:

(a) The production plan for the first quarter of 1989.

(b) Concerning disk drive usage, should DEC continue to manufacture products in the constrained mode, or should it plan to satisfy customer preferences?

(c) Concerning memory boards, should DEC use alternative memory boards for its GP-1 systems?

(d) A final decision that had to be made was related to tradeoffs between shortages of disk drives and of 256K memory boards. The manufacturing staff would like to concentrate their efforts on either decreasing the shortage of disks or decreasing the shortage of 256K memory boards. Hence, they would like to know which alternative would have a larger effect on revenue.

In order to model the problem that DEC faced, we introduce variables x_1, x_2, x_3, x_4, x_5, that represent the number (in thousands) of GP-1, GP-2, GP-3, WS-1, and WS-2 systems, respectively, to be produced in the next quarter. Strictly speaking, since $1000x_i$ stands for number of units, it must be an integer. This can be accomplished by truncating each x_i after the third decimal point; given the size of the demand and the size of the

variables x_i, this has a negligible effect and the integrality constraint on $1000x_i$ can be ignored.

DEC had to make two distinct decisions: whether to use the constrained mode of production regarding disk drive usage, and whether to use alternative memory boards for the GP-1 system. As a result, there are four different combinations of possible choices.

We first develop a model for the case where alternative memory boards are not used and the constrained mode of production of disk drives is selected. The problem can be formulated as follows:

$$\text{maximize} \quad 60x_1 + 40x_2 + 30x_3 + 30x_4 + 15x_5 \qquad \text{(total revenue)}$$

subject to the following constraints:

$$\begin{aligned}
x_1 + x_2 + x_3 + x_4 + x_5 &\leq 7 && \text{(CPU availability)}\\
4x_1 + 2x_2 + 2x_3 + 2x_4 + x_5 &\leq 8 && \text{(256K availability)}\\
x_2 + x_4 &\leq 3 && \text{(disk drive availability)}\\
x_1 &\leq 1.8 && \text{(max demand for GP-1)}\\
x_3 &\leq 0.3 && \text{(max demand for GP-3)}\\
x_1 + x_2 + x_3 &\leq 3.8 && \text{(max demand for GP)}\\
x_4 + x_5 &\leq 3.2 && \text{(max demand for WS)}\\
x_2 &\geq 0.5 && \text{(min demand for GP-2)}\\
x_4 &\geq 0.5 && \text{(min demand for WS-1)}\\
x_5 &\geq 0.4 && \text{(min demand for WS-2)}\\
x_1, x_2, x_3, x_4, x_5 &\geq 0.
\end{aligned}$$

Notice that the objective function is in millions of dollars. In some respects, this is a pessimistic formulation, because the 256K memory and disk drive availability were set to 8 and 3, respectively, which is the lowest value in the range that was estimated. It is actually of interest to determine the solution to this problem as the 256K memory availability ranges from 8 to 16, and the disk drive availability ranges from 3 to 7, because this provides valuable information on the sensitivity of the optimal solution on availability. In another respect, the formulation is optimistic because, for example, it assumes that the revenue from GP-1 systems is $60x_1$ for any $x_1 \leq 1.8$, even though a demand for 1,800 GP-1 systems is not guaranteed.

In order to accommodate the other three choices that DEC had, some of the problem constraints have to be modified, as follows. If we use the unconstrained mode of production for disk drives, the constraint $x_2 + x_4 \leq 3$ is replaced by

$$0.3x_1 + 1.7x_2 + 1.4x_4 \leq 3.$$

Furthermore, if we wish to use alternative memory boards in GP-1 systems, we replace the constraint $4x_1 + 2x_2 + 2x_3 + 2x_4 + x_5 \leq 8$ by the two

constraints

$$\begin{aligned} 2x_1 &\leq 4, \\ 2x_2 + 2x_3 + 2x_4 + x_5 &\leq 8. \end{aligned}$$

The four combinations of choices lead to four different linear programming problems, each of which needs to be solved for a variety of parameter values because, as discussed earlier, the right-hand side of some of the constraints is only known to lie within a certain range. Methods for solving linear programming problems, when certain parameters are allowed to vary, will be studied in Chapter 5, where this case study is revisited.

Multiperiod planning of electric power capacity

A state wants to plan its electricity capacity for the next T years. The state has a forecast of d_t megawatts, presumed accurate, of the demand for electricity during year $t = 1, \ldots, T$. The existing capacity, which is in oil-fired plants, that will not be retired and will be available during year t, is e_t. There are two alternatives for expanding electric capacity: coal-fired or nuclear power plants. There is a capital cost of c_t per megawatt of coal-fired capacity that becomes operational at the beginning of year t. The corresponding capital cost for nuclear power plants is n_t. For various political and safety reasons, it has been decided that no more than 20% of the total capacity should ever be nuclear. Coal plants last for 20 years, while nuclear plants last for 15 years. A least cost capacity expansion plan is desired.

The first step in formulating this problem as a linear programming problem is to define the decision variables. Let x_t and y_t be the amount of coal (respectively, nuclear) capacity brought on line at the beginning of year t. Let w_t and z_t be the total coal (respectively, nuclear) capacity available in year t. The cost of a capacity expansion plan is therefore,

$$\sum_{t=1}^{T} (c_t x_t + n_t y_t).$$

Since coal-fired plants last for 20 years, we have

$$w_t = \sum_{s=\max\{1,t-19\}}^{t} x_s, \qquad t = 1, \ldots, T.$$

Similarly, for nuclear power plants,

$$z_t = \sum_{s=\max\{1,t-14\}}^{t} y_s, \qquad t = 1, \ldots, T.$$

Since the available capacity must meet the forecasted demand, we require

$$w_t + z_t + e_t \geq d_t, \qquad t = 1, \ldots, T.$$

Finally, since no more than 20% of the total capacity should ever be nuclear, we have

$$\frac{z_t}{w_t + z_t + e_t} \leq 0.2,$$

which can be written as

$$0.8z_t - 0.2w_t \leq 0.2e_t.$$

Summarizing, the capacity expansion problem is as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} (c_t x_t + n_t y_t) \\
\text{subject to} \quad & w_t - \sum_{s=\max\{1,t-19\}}^{t} x_s = 0, && t = 1, \ldots, T, \\
& z_t - \sum_{s=\max\{1,t-14\}}^{t} y_s = 0, && t = 1, \ldots, T, \\
& w_t + z_t \geq d_t - e_t, && t = 1, \ldots, T, \\
& 0.8z_t - 0.2w_t \leq 0.2e_t, && t = 1, \ldots, T, \\
& x_t, y_t, w_t, z_t \geq 0, && t = 1, \ldots, T.
\end{aligned}$$

We note that this formulation is not entirely realistic, because it disregards certain economies of scale that may favor larger plants. However, it can provide a ballpark estimate of the true cost.

A scheduling problem

In the previous examples, the choice of the decision variables was fairly straightforward. We now discuss an example where this choice is less obvious.

A hospital wants to make a weekly night shift (12pm-8am) schedule for its nurses. The demand for nurses for the night shift on day j is an integer d_j, $j = 1, \ldots, 7$. Every nurse works 5 days in a row on the night shift. The problem is to find the minimal number of nurses the hospital needs to hire.

One could try using a decision variable y_j equal to the number of nurses that work on day j. With this definition, however, we would not be able to capture the constraint that every nurse works 5 days in a row. For this reason, we choose the decision variables differently, and define x_j as

the number of nurses starting their week on day j. (For example, a nurse whose week starts on day 5 will work days $5, 6, 7, 1, 2$.) We then have the following problem formulation:

$$
\begin{array}{lll}
\text{minimize} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 & \\
\text{subject to} & x_1 \qquad\quad + x_4 + x_5 + x_6 + x_7 & \geq d_1 \\
& x_1 + x_2 \qquad\quad + x_5 + x_6 + x_7 & \geq d_2 \\
& x_1 + x_2 + x_3 \qquad\quad + x_6 + x_7 & \geq d_3 \\
& x_1 + x_2 + x_3 + x_4 \qquad\quad + x_7 & \geq d_4 \\
& x_1 + x_2 + x_3 + x_4 + x_5 \qquad\quad & \geq d_5 \\
& \quad x_2 + x_3 + x_4 + x_5 + x_6 \qquad & \geq d_6 \\
& \qquad\quad x_3 + x_4 + x_5 + x_6 + x_7 & \geq d_7 \\
& x_j \geq 0, \qquad x_j \text{ integer.} &
\end{array}
$$

This would be a linear programming problem, except for the constraint that each x_j must be an integer, and we actually have a linear *integer programming* problem. One way of dealing with this issue is to ignore ("relax") the integrality constraints and obtain the so-called *linear programming relaxation* of the original problem. Because the linear programming problem has fewer constraints, and therefore more options, the optimal cost will be less than or equal to the optimal cost of the original problem. If the optimal solution to the linear programming relaxation happens to be integer, then it is also an optimal solution to the original problem. If it is not integer, we can round each x_j upwards, thus obtaining a feasible, but not necessarily optimal, solution to the original problem. It turns out that for this particular problem, an optimal solution can be found without too much effort. However, this is the exception rather than the rule: finding optimal solutions to general integer programming problems is typically difficult; some methods will be discussed in Chapter 11.

Choosing paths in a communication network

Consider a communication network consisting of n nodes. Nodes are connected by communication links. A link allowing one-way transmission from node i to node j is described by an ordered pair (i, j). Let $\mathcal{A}$ be the set of all links. We assume that each link $(i, j) \in \mathcal{A}$ can carry up to u_{ij} bits per second. There is a positive charge c_{ij} per bit transmitted along that link. Each node k generates data, at the rate of $b^{k\ell}$ bits per second, that have to be transmitted to node ℓ, either through a direct link (k, ℓ) or by tracing a sequence of links. The problem is to choose paths along which all data reach their intended destinations, while minimizing the total cost. We allow the data with the same origin and destination to be split and be transmitted along different paths.

In order to formulate this problem as a linear programming problem, we introduce variables $x_{ij}^{k\ell}$ indicating the amount of data with origin k and

destination ℓ that traverse link (i,j). Let

$$b_i^{k\ell} = \begin{cases} b^{k\ell}, & \text{if } i = k, \\ -b^{k\ell}, & \text{if } i = \ell, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, $b_i^{k\ell}$ is the net inflow at node i, from outside the network, of data with origin k and destination ℓ. We then have the following formulation:

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} \sum_{k=1}^{n} \sum_{\ell=1}^{n} c_{ij} x_{ij}^{k\ell}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}^{k\ell} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji}^{k\ell} = b_i^{k\ell}, \qquad i, k, \ell = 1, \ldots, n,$$

$$\sum_{k=1}^{n} \sum_{\ell=1}^{n} x_{ij}^{k\ell} \leq u_{ij}, \qquad (i,j) \in \mathcal{A},$$

$$x_{ij}^{k\ell} \geq 0, \qquad (i,j) \in \mathcal{A}, \quad k, \ell = 1, \ldots, n.$$

The first constraint is a flow conservation constraint at node i for data with origin k and destination ℓ. The expression

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}^{k\ell}$$

represents the amount of data with origin and destination k and ℓ, respectively, that leave node i along some link. The expression

$$\sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji}^{k\ell}$$

represents the amount of data with the same origin and destination that enter node i through some link. Finally, $b_i^{k\ell}$ is the net amount of such data that enter node i from outside the network. The second constraint expresses the requirement that the total traffic through a link (i,j) cannot exceed the link's capacity.

This problem is known as the *multicommodity flow* problem, with the traffic corresponding to each origin-destination pair viewed as a different commodity. A mathematically similar problem arises when we consider a transportation company that wishes to transport several commodities from their origins to their destinations through a network. There is a version of this problem, known as the minimum cost *network flow* problem, in which we do not distinguish between different commodities. Instead, we are given the amount b_i of external supply or demand at each node i, and the objective is to transport material from the supply nodes to the demand nodes, at minimum cost. The network flow problem, which is the subject of Chapter 7, contains as special cases some important problems such as the shortest path problem, the maximum flow problem, and the assignment problem.

Pattern classification

We are given m examples of objects and for each one, say the ith one, a description of its features in terms of an n-dimensional vector $\mathbf{a}_i$. Objects belong to one of two classes, and for each example we are told the class that it belongs to.

More concretely, suppose that each object is an image of an apple or an orange (these are our two classes). In this context, we can use a three-dimensional feature vector $\mathbf{a}_i$ to summarize the contents of the ith image. The three components of $\mathbf{a}_i$ (the features) could be the ellipticity of the object, the length of its stem, and its color, as measured in some scale. We are interested in designing a *classifier* which, given a new object (other than the originally available examples), will figure out whether it is an image of an apple or of an orange.

A *linear classifier* is defined in terms of an n-dimensional vector $\mathbf{x}$ and a scalar x_{n+1}, and operates as follows. Given a new object with feature vector $\mathbf{a}$, the classifier declares it to be an object of the first class if

$$\mathbf{a}'\mathbf{x} \geq x_{n+1},$$

and of the second class if

$$\mathbf{a}'\mathbf{x} < x_{n+1}.$$

In words, a linear classifier makes decisions on the basis of a linear combination of the different features. Our objective is to use the available examples in order to design a "good" linear classifier.

There are many ways of approaching this problem, but a reasonable starting point could be the requirement that the classifier must give the correct answer for each one of the available examples. Let S be the set of examples of the first class. We are then looking for some $\mathbf{x}$ and x_{n+1} that satisfy the constraints

$$\begin{aligned} \mathbf{a}_i'\mathbf{x} &\geq x_{n+1}, \qquad && i \in S, \\ \mathbf{a}_i'\mathbf{x} &< x_{n+1}, \qquad && i \notin S. \end{aligned}$$

Note that the second set of constraints involves a strict inequality and is not quite of the form arising in linear programming. This issue can be bypassed by observing that if some choice of $\mathbf{x}$ and x_{n+1} satisfies all of the above constraints, then there exists some other choice (obtained by multiplying $\mathbf{x}$ and x_{n+1} by a suitably large positive scalar) that satisfies

$$\begin{aligned} \mathbf{a}_i'\mathbf{x} &\geq x_{n+1}, \qquad && i \in S, \\ \mathbf{a}_i'\mathbf{x} &\leq x_{n+1} - 1, \qquad && i \notin S. \end{aligned}$$

We conclude that the search for a linear classifier consistent with all available examples is a problem of finding a feasible solution to a linear programming problem.

1.3 Piecewise linear convex objective functions

All of the examples in the preceding section involved a *linear* objective function. However, there is an important class of optimization problems with a nonlinear objective function that can be cast as linear programming problems; these are examined next.

We first need some definitions:

Definition 1.1

(a) *A function* $f : \Re^n \mapsto \Re$ *is called* **convex** *if for every* $\mathbf{x}, \mathbf{y} \in \Re^n$, *and every* $\lambda \in [0,1]$, *we have*

$$f\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}).$$

(b) *A function* $f : \Re^n \mapsto \Re$ *is called* **concave** *if for every* $\mathbf{x}, \mathbf{y} \in \Re^n$, *and every* $\lambda \in [0,1]$, *we have*

$$f\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) \geq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}).$$

Note that if $\mathbf{x}$ and $\mathbf{y}$ are vectors in $\Re^n$ and if λ ranges in $[0,1]$, then points of the form $\lambda\mathbf{x} + (1-\lambda)\mathbf{y}$ belong to the line segment joining $\mathbf{x}$ and $\mathbf{y}$. The definition of a convex function refers to the values of f, as its argument traces this segment. If f were linear, the inequality in part (a) of the definition would hold with equality. The inequality therefore means that when we restrict attention to such a segment, the graph of the function lies no higher than the graph of a corresponding linear function; see Figure 1.1(a).

It is easily seen that a function f is convex if and only if the function $-f$ is concave. Note that a function of the form $f(\mathbf{x}) = a_0 + \sum_{i=1}^{n} a_i x_i$, where $a_0, \ldots, a_n$ are scalars, called an *affine* function, is both convex and concave. (It turns out that affine functions are the only functions that are both convex and concave.) Convex (as well as concave) functions play a central role in optimization.

We say that a vector $\mathbf{x}$ is a *local minimum* of f if $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{y}$ in the vicinity of $\mathbf{x}$. We also say that $\mathbf{x}$ is a *global minimum* if $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{y}$. A convex function cannot have local minima that fail to be global minima (see Figure 1.1), and this property is of great help in designing efficient optimization algorithms.

Let $\mathbf{c}_1, \ldots, \mathbf{c}_m$ be vectors in $\Re^n$, let $d_1, \ldots, d_m$ be scalars, and consider the function $f : \Re^n \mapsto \Re$ defined by

$$f(\mathbf{x}) = \max_{i=1,\ldots,m} (\mathbf{c}_i'\mathbf{x} + d_i)$$

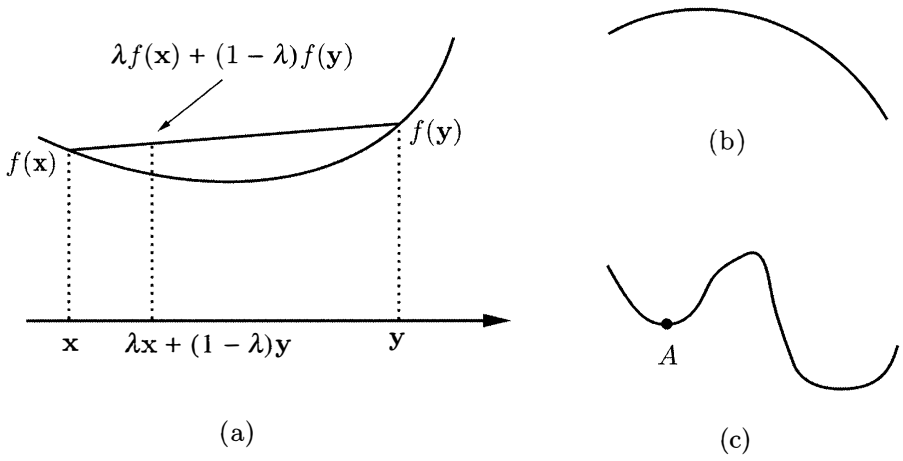

Figure 1.1: (a) Illustration of the definition of a convex function. (b) A concave function. (c) A function that is neither convex nor concave; note that A is a local, but not global, minimum.

[see Figure 1.2(a)]. Such a function is convex, as a consequence of the following result.

Theorem 1.1 *Let $f_1, \ldots, f_m : \Re^n \mapsto \Re$ be convex functions. Then, the function f defined by $f(\mathbf{x}) = \max_{i=1,\ldots,m} f_i(\mathbf{x})$ is also convex.*

Proof. Let $\mathbf{x}, \mathbf{y} \in \Re^n$ and let $\lambda \in [0,1]$. We have

$$\begin{aligned} f\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) &= \max_{i=1,\ldots,m} f_i\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) \\ &\leq \max_{i=1,\ldots,m} \big(\lambda f_i(\mathbf{x}) + (1-\lambda) f_i(\mathbf{y})\big) \\ &\leq \max_{i=1,\ldots,m} \lambda f_i(\mathbf{x}) + \max_{i=1,\ldots,m} (1-\lambda) f_i(\mathbf{y}) \\ &= \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}). \end{aligned}$$

□

A function of the form $\max_{i=1,\ldots,m}(\mathbf{c}_i'\mathbf{x}+d_i)$ is called a *piecewise linear convex* function. A simple example is the absolute value function defined by $f(x) = |x| = \max\{x, -x\}$. As illustrated in Figure 1.2(b), a piecewise linear convex function can be used to approximate a general convex function.

We now consider a generalization of linear programming, where the objective function is piecewise linear and convex rather than linear:

$$\begin{aligned} \text{minimize} \quad & \max_{i=1,\ldots,m} (\mathbf{c}_i'\mathbf{x} + d_i) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}. \end{aligned}$$

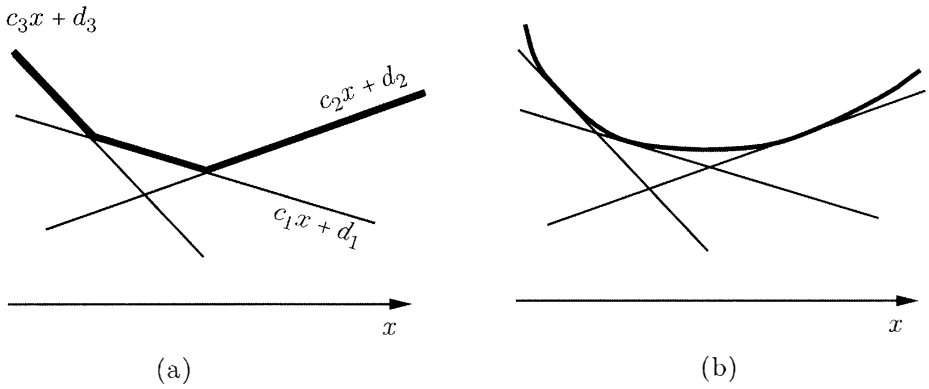

Figure 1.2: (a) A piecewise linear convex function of a single variable. (b) An approximation of a convex function by a piecewise linear convex function.

Note that $\max_{i=1,\ldots,m}(\mathbf{c}_i'\mathbf{x} + d_i)$ is equal to the smallest number z that satisfies $z \geq \mathbf{c}_i'\mathbf{x} + d_i$ for all i. For this reason, the optimization problem under consideration is equivalent to the linear programming problem

$$\begin{array}{lll} \text{minimize} & z & \\ \text{subject to} & z \geq \mathbf{c}_i'\mathbf{x} + d_i, & i = 1, \ldots, m, \\ & \mathbf{A}\mathbf{x} \geq \mathbf{b}, & \end{array}$$

where the decision variables are z and $\mathbf{x}$.

To summarize, linear programming can be used to solve problems with piecewise linear convex cost functions, and the latter class of functions can be used as an approximation of more general convex cost functions. On the other hand, such a piecewise linear approximation is not always a good idea because it can turn a smooth function into a nonsmooth one (piecewise linear functions have discontinuous derivatives).

We finally note that if we are given a constraint of the form $f(\mathbf{x}) \leq h$, where f is the piecewise linear convex function $f(\mathbf{x}) = \max_{i=1,\ldots,m}(\mathbf{f}_i'\mathbf{x}+g_i)$, such a constraint can be rewritten as

$$\mathbf{f}_i'\mathbf{x} + g_i \leq h, \qquad i = 1, \ldots, m,$$

and linear programming is again applicable.

Problems involving absolute values

Consider a problem of the form

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{n} c_i|x_i| \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b}, \end{array}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$, and where the cost coefficients c_i are assumed to be nonnegative. The cost criterion, being the sum of the piecewise linear convex functions $c_i|x_i|$ is easily shown to be piecewise linear and convex (Exercise 1.2). However, expressing this cost criterion in the form $\max_j(\mathbf{c}_j'\mathbf{x} + d_j)$ is a bit involved, and a more direct route is preferable. We observe that $|x_i|$ is the smallest number z_i that satisfies $x_i \leq z_i$ and $-x_i \leq z_i$, and we obtain the linear programming formulation

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i z_i \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & x_i \leq z_i, \qquad i = 1, \ldots, n, \\ & -x_i \leq z_i, \qquad i = 1, \ldots, n. \end{aligned}$$

An alternative method for dealing with absolute values is to introduce new variables x_i^+, x_i^-, constrained to be nonnegative, and let $x_i = x_i^+ - x_i^-$. (Our intention is to have $x_i = x_i^+$ or $x_i = -x_i^-$, depending on whether x_i is positive or negative.) We then replace every occurrence of $|x_i|$ with $x_i^+ + x_i^-$ and obtain the alternative formulation

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i(x_i^+ + x_i^-) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x}^+ - \mathbf{A}\mathbf{x}^- \geq \mathbf{b} \\ & \mathbf{x}^+, \mathbf{x}^- \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{x}^+ = (x_1^+, \ldots, x_n^+)$ and $\mathbf{x}^- = (x_1^-, \ldots, x_n^-)$.

The relations $x_i = x_i^+ - x_i^-$, $x_i^+ \geq 0$, $x_i^- \geq 0$, are not enough to guarantee that $|x_i| = x_i^+ + x_i^-$, and the validity of this reformulation may not be entirely obvious. Let us assume for simplicity that $c_i > 0$ for all i. At an optimal solution to the reformulated problem, and for each i, we must have either $x_i^+ = 0$ or $x_i^- = 0$, because otherwise we could reduce both x_i^+ and x_i^- by the same amount and preserve feasibility, while reducing the cost, in contradiction of optimality. Having guaranteed that either $x_i^+ = 0$ or $x_i^- = 0$, the desired relation $|x_i| = x_i^+ + x_i^-$ now follows.

The formal correctness of the two reformulations that have been presented here, and in a somewhat more general setting, is the subject of Exercise 1.5. We also note that the nonnegativity assumption on the cost coefficients c_i is crucial because, otherwise, the cost criterion is nonconvex.

Example 1.5 Consider the problem

$$\begin{aligned} \text{minimize} \quad & 2|x_1| + x_2 \\ \text{subject to} \quad & x_1 + x_2 \geq 4. \end{aligned}$$

Our first reformulation yields

$$\begin{array}{rl} \text{minimize} & 2z_1 + x_2 \\ \text{subject to} & x_1 + x_2 \geq 4 \\ & x_1 \leq z_1 \\ & -x_1 \leq z_1, \end{array}$$

while the second yields

$$\begin{array}{rl} \text{minimize} & 2x_1^+ + 2x_1^- + x_2 \\ \text{subject to} & x_1^+ - x_1^- + x_2 \geq 4 \\ & x_1^+ \geq 0 \\ & x_1^- \geq 0. \end{array}$$

We now continue with some applications involving piecewise linear convex objective functions.

Data fitting

We are given m data points of the form $(\mathbf{a}_i, b_i)$, $i = 1, \ldots, m$, where $\mathbf{a}_i \in \Re^n$ and $b_i \in \Re$, and wish to build a model that predicts the value of the variable b from knowledge of the vector $\mathbf{a}$. In such a situation, one often uses a linear model of the form $b = \mathbf{a}'\mathbf{x}$, where $\mathbf{x}$ is a parameter vector to be determined. Given a particular parameter vector $\mathbf{x}$, the *residual*, or prediction error, at the ith data point is defined as $|b_i - \mathbf{a}_i'\mathbf{x}|$. Given a choice between alternative models, one should choose a model that "explains" the available data as best as possible, i.e., a model that results in small residuals.

One possibility is to minimize the largest residual. This is the problem of minimizing

$$\max_i |b_i - \mathbf{a}_i'\mathbf{x}|,$$

with respect to $\mathbf{x}$, subject to no constraints. Note that we are dealing here with a piecewise linear convex cost criterion. The following is an equivalent linear programming formulation:

$$\begin{array}{rll} \text{minimize} & z & \\ \text{subject to} & b_i - \mathbf{a}_i'\mathbf{x} \leq z, & i = 1, \ldots, m, \\ - & b_i + \mathbf{a}_i'\mathbf{x} \leq z, & i = 1, \ldots, m, \end{array}$$

the decision variables being z and $\mathbf{x}$.

In an alternative formulation, we could adopt the cost criterion

$$\sum_{i=1}^{m} |b_i - \mathbf{a}_i'\mathbf{x}|.$$

Since $|b_i - \mathbf{a}_i'\mathbf{x}|$ is the smallest number z_i that satisfies $b_i - \mathbf{a}_i'\mathbf{x} \leq z_i$ and $-b_i + \mathbf{a}_i'\mathbf{x} \leq z_i$, we obtain the formulation

$$\begin{aligned} \text{minimize} \quad & z_1 + \cdots + z_m & \\ \text{subject to} \quad & b_i - \mathbf{a}_i'\mathbf{x} \leq z_i, & i = 1, \ldots, m, \\ - \; & b_i + \mathbf{a}_i'\mathbf{x} \leq z_i, & i = 1, \ldots, m. \end{aligned}$$

In practice, one may wish to use the quadratic cost criterion $\sum_{i=1}^{m}(b_i - \mathbf{a}_i'\mathbf{x})^2$, in order to obtain a "least squares fit." This is a problem which is easier than linear programming; it can be solved using calculus methods, but its discussion is outside the scope of this book.

Optimal control of linear systems

Consider a dynamical system that evolves according to a model of the form

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ y(t) &= \mathbf{c}'\mathbf{x}(t). \end{aligned}$$

Here $\mathbf{x}(t)$ is the state of the system at time t, $y(t)$ is the system output, assumed scalar, and $\mathbf{u}(t)$ is a control vector that we are free to choose subject to linear constraints of the form $\mathbf{D}\mathbf{u}(t) \leq \mathbf{d}$ [these might include saturation constraints, i.e., hard bounds on the magnitude of each component of $\mathbf{u}(t)$]. To mention some possible applications, this could be a model of an airplane, an engine, an electrical circuit, a mechanical system, a manufacturing system, or even a model of economic growth. We are also given the initial state $\mathbf{x}(0)$. In one possible problem, we are to choose the values of the control variables $\mathbf{u}(0), \ldots, \mathbf{u}(T-1)$ to drive the state $\mathbf{x}(T)$ to a target state, assumed for simplicity to be zero. In addition to zeroing the state, it is often desirable to keep the magnitude of the output small at all intermediate times, and we may wish to minimize

$$\max_{t=1,\ldots,T-1} |y(t)|.$$

We then obtain the following linear programming problem:

$$\begin{aligned} \text{minimize} \quad & z & \\ \text{subject to} \quad & -z \leq y(t) \leq z, & t = 1, \ldots, T-1, \\ & \mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), & t = 0, \ldots, T-1, \\ & y(t) = \mathbf{c}'\mathbf{x}(t), & t = 1, \ldots, T-1, \\ & \mathbf{D}\mathbf{u}(t) \leq \mathbf{d}, & t = 0, \ldots, T-1, \\ & \mathbf{x}(T) = \mathbf{0}, & \\ & \mathbf{x}(0) = \text{given}. & \end{aligned}$$

Additional linear constraints on the state vectors $\mathbf{x}(t)$, or a more general piecewise linear convex cost function of the state and the control, can also be incorporated.

Rocket control

Consider a rocket that travels along a straight path. Let x_t, v_t, and a_t be the position, velocity, and acceleration, respectively, of the rocket at time t. By discretizing time and by taking the time increment to be unity, we obtain an approximate discrete-time model of the form

$$\begin{aligned} x_{t+1} &= x_t + v_t, \\ v_{t+1} &= v_t + a_t. \end{aligned}$$

We assume that the acceleration a_t is under our control, as it is determined by the rocket thrust. In a rough model, the magnitude $|a_t|$ of the acceleration can be assumed to be proportional to the rate of fuel consumption at time t.

Suppose that the rocket is initially at rest at the origin, that is, $x_0 = 0$ and $v_0 = 0$. We wish the rocket to take off and "land softly" at unit distance from the origin after T time units, that is, $x_T = 1$ and $v_T = 0$. Furthermore, we wish to accomplish this in an economical fashion. One possibility is to minimize the total fuel $\sum_{t=0}^{T-1} |a_t|$ spent subject to the preceding constraints. Alternatively, we may wish to minimize the maximum thrust required, which is $\max_t |a_t|$. Under either alternative, the problem can be formulated as a linear programming problem (Exercise 1.6).

1.4 Graphical representation and solution

In this section, we consider a few simple examples that provide useful geometric insights into the nature of linear programming problems. Our first example involves the graphical solution of a linear programming problem with two variables.

Example 1.6 Consider the problem

$$\begin{array}{rrcl} \text{minimize} & -x_1 - x_2 & & \\ \text{subject to} & x_1 + 2x_2 & \leq & 3 \\ & 2x_1 + x_2 & \leq & 3 \\ & x_1, x_2 \geq 0. & & \end{array}$$

The feasible set is the shaded region in Figure 1.3. In order to find an optimal solution, we proceed as follows. For any given scalar z, we consider the set of all points whose cost $\mathbf{c}'\mathbf{x}$ is equal to z; this is the line described by the equation $-x_1 - x_2 = z$. Note that this line is perpendicular to the vector $\mathbf{c} = (-1, -1)$. Different values of z lead to different lines, all of them parallel to each other. In

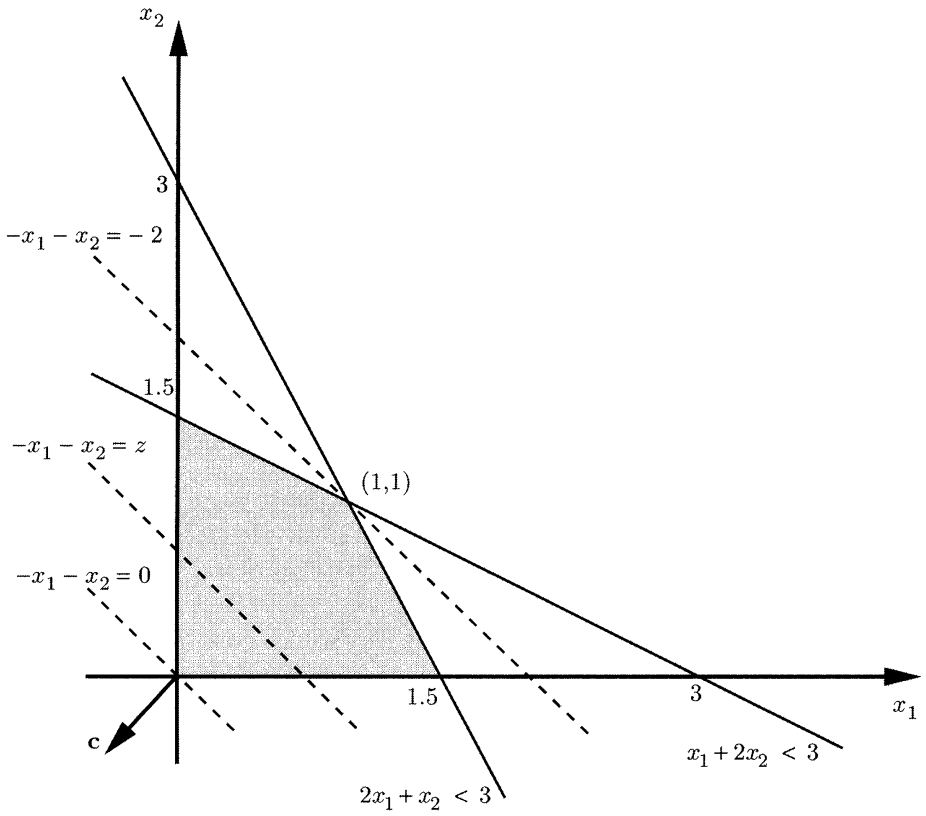

Figure 1.3: Graphical solution of the problem in Example 1.6.

particular, increasing z corresponds to moving the line $z = -x_1 - x_2$ along the direction of the vector $\mathbf{c}$. Since we are interested in minimizing z, we would like to move the line as much as possible in the direction of $-\mathbf{c}$, as long as we do not leave the feasible region. The best we can do is $z = -2$ (see Figure 1.3), and the vector $\mathbf{x} = (1, 1)$ is an optimal solution. Note that this is a corner of the feasible set. (The concept of a "corner" will be defined formally in Chapter 2.)

For a problem in three dimensions, the same approach can be used except that the set of points with the same value of $\mathbf{c}'\mathbf{x}$ is a plane, instead of a line. This plane is again perpendicular to the vector $\mathbf{c}$, and the objective is to slide this plane as much as possible in the direction of $-\mathbf{c}$, as long as we do not leave the feasible set.

Example 1.7 Suppose that the feasible set is the unit cube, described by the constraints $0 \leq x_i \leq 1$, $i = 1, 2, 3$, and that $\mathbf{c} = (-1, -1, -1)$. Then, the vector $\mathbf{x} = (1, 1, 1)$ is an optimal solution. Once more, the optimal solution happens to be a corner of the feasible set (Figure 1.4).

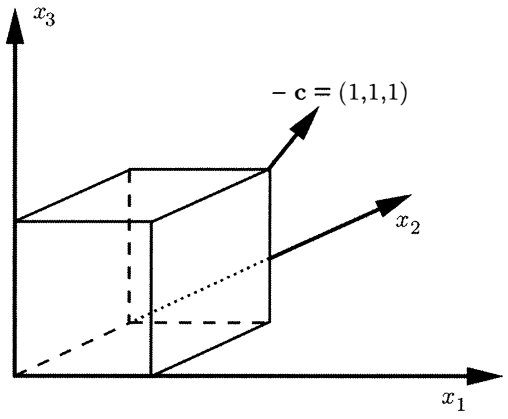

Figure 1.4: The three-dimensional linear programming problem in Example 1.7.

In both of the preceding examples, the feasible set is bounded (does not extend to infinity), and the problem has a unique optimal solution. This is not always the case and we have some additional possibilities that are illustrated by the example that follows.

Example 1.8 Consider the feasible set in $\Re^2$ defined by the constraints

$$\begin{aligned} -x_1 + x_2 &\leq 1 \\ x_1 &\geq 0 \\ x_2 &\geq 0, \end{aligned}$$

which is shown in Figure 1.5.

(a) For the cost vector $\mathbf{c} = (1, 1)$, it is clear that $\mathbf{x} = (0, 0)$ is the unique optimal solution.

(b) For the cost vector $\mathbf{c} = (1, 0)$, there are multiple optimal solutions, namely, every vector $\mathbf{x}$ of the form $\mathbf{x} = (0, x_2)$, with $0 \leq x_2 \leq 1$, is optimal. Note that the set of optimal solutions is bounded.

(c) For the cost vector $\mathbf{c} = (0, 1)$, there are multiple optimal solutions, namely, every vector $\mathbf{x}$ of the form $\mathbf{x} = (x_1, 0)$, with $x_1 \geq 0$, is optimal. In this case, the set of optimal solutions is unbounded (contains vectors of arbitrarily large magnitude).

(d) Consider the cost vector $\mathbf{c} = (-1, -1)$. For any feasible solution (x_1, x_2), we can always produce another feasible solution with less cost, by increasing the value of x_1. Therefore, no feasible solution is optimal. Furthermore, by considering vectors (x_1, x_2) with ever increasing values of x_1 and x_2, we can obtain a sequence of feasible solutions whose cost converges to $-\infty$. We therefore say that the optimal cost is $-\infty$.

(e) If we impose an additional constraint of the form $x_1 + x_2 \leq -2$, it is evident that no feasible solution exists.

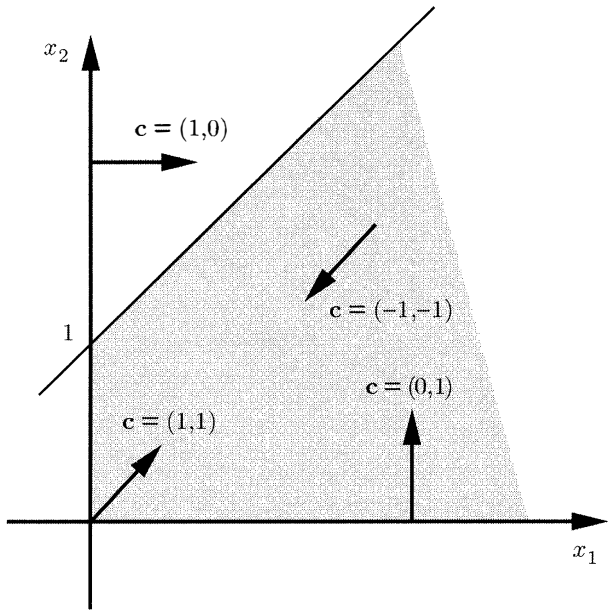

Figure 1.5: The feasible set in Example 1.8. For each choice of $\mathbf{c}$, an optimal solution is obtained by moving as much as possible in the direction of $-\mathbf{c}$.

To summarize the insights obtained from Example 1.8, we have the following possibilities:

(a) There exists a unique optimal solution.

(b) There exist multiple optimal solutions; in this case, the set of optimal solutions can be either bounded or unbounded.

(c) The optimal cost is $-\infty$, and no feasible solution is optimal.

(d) The feasible set is empty.

In principle, there is an additional possibility: an optimal solution does not exist even though the problem is feasible and the optimal cost is not $-\infty$; this is the case, for example, in the problem of minimizing $1/x$ subject to $x > 0$ (for every feasible solution, there exists another with less cost, but the optimal cost is not $-\infty$). We will see later in this book that this possibility never arises in linear programming.

In the examples that we have considered, if the problem has at least one optimal solution, then an optimal solution can be found among the corners of the feasible set. In Chapter 2, we will show that this is a general feature of linear programming problems, as long as the feasible set has at least one corner.

Visualizing standard form problems

We now discuss a method that allows us to visualize standard form problems even if the dimension n of the vector $\mathbf{x}$ is greater than three. The reason for wishing to do so is that when $n \leq 3$, the feasible set of a standard form problem does not have much variety and does not provide enough insight into the general case. (In contrast, if the feasible set is described by constraints of the form $\mathbf{Ax} \geq \mathbf{b}$, enough variety is obtained even if $\mathbf{x}$ has dimension three.)

Suppose that we have a standard form problem, and that the matrix $\mathbf{A}$ has dimensions $m \times n$. In particular, the decision vector $\mathbf{x}$ is of dimension n and we have m equality constraints. We assume that $m \leq n$ and that the constraints $\mathbf{Ax} = \mathbf{b}$ force $\mathbf{x}$ to lie on an $(n - m)$-dimensional set. (Intuitively, each constraint removes one of the "degrees of freedom" of $\mathbf{x}$.) If we "stand" on that $(n-m)$-dimensional set and ignore the m dimensions orthogonal to it, the feasible set is only constrained by the linear inequality constraints $x_i \geq 0$, $i = 1, \ldots, n$. In particular, if $n - m = 2$, the feasible set can be drawn as a two-dimensional set defined by n linear inequality constraints.

To illustrate this approach, consider the feasible set in $\Re^3$ defined by the constraints $x_1 + x_2 + x_3 = 1$ and $x_1, x_2, x_3 \geq 0$ [Figure 1.6(a)], and note that $n = 3$ and $m = 1$. If we stand on the plane defined by the constraint $x_1 + x_2 + x_3 = 1$, then the feasible set has the appearance of a triangle in two-dimensional space. Furthermore, each edge of the triangle corresponds to one of the constraints $x_1, x_2, x_3 \geq 0$; see Figure 1.6(b).

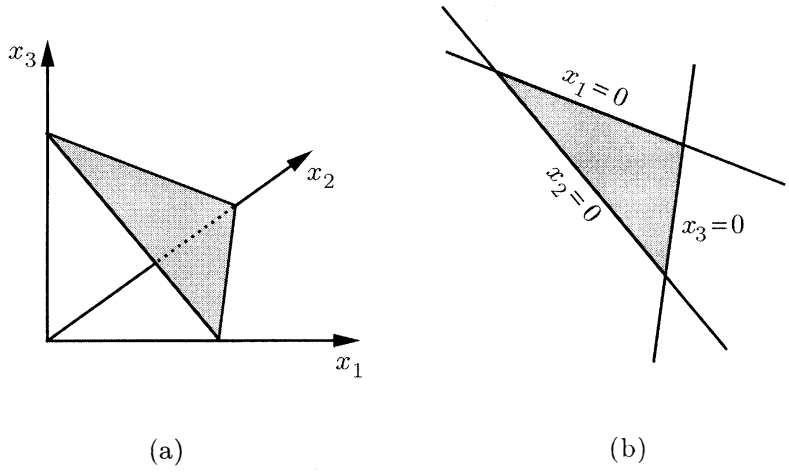

Figure 1.6: (a) An n-dimensional view of the feasible set. (b) An $(n - m)$-dimensional view of the same set.

1.5 Linear algebra background and notation

This section provides a summary of the main notational conventions that we will be employing. It also contains a brief review of those results from linear algebra that are used in the sequel.

Set theoretic notation

If S is a set and x is an element of S, we write $x \in S$. A set can be specified in the form $S = \{x \mid x \text{ satisfies } P\}$, as the set of all elements having property P. The cardinality of a finite set S is denoted by $|S|$. The union of two sets S and T is denoted by $S \cup T$, and their intersection by $S \cap T$. We use $S \backslash T$ to denote the set of all elements of S that do not belong to T. The notation $S \subset T$ means that S is a subset of T, i.e., every element of S is also an element of T; in particular, S could be equal to T. If in addition $S \neq T$, we say that S is a *proper* subset of T. We use $\emptyset$ to denote the empty set. The symbols $\exists$ and $\forall$ have the meanings "there exists" and "for all," respectively.

We use $\Re$ to denote the set of real numbers. For any real numbers a and b, we define the closed and open intervals $[a, b]$ and (a, b), respectively, by

$$[a, b] = \{x \in \Re \mid a \leq x \leq b\},$$

and

$$(a, b) = \{x \in \Re \mid a < x < b\}.$$

Vectors and matrices

A *matrix* of dimensions $m \times n$ is an array of real numbers a_{ij}:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Matrices will be always denoted by upper case boldface characters. If $\mathbf{A}$ is a matrix, we use the notation a_{ij} or $[\mathbf{A}]_{ij}$ to refer to its (i, j)th entry. A *row vector* is a matrix with $m = 1$ and a *column vector* is a matrix with $n = 1$. The word *vector* will always mean *column vector* unless the contrary is explicitly stated. Vectors will be usually denoted by lower case boldface characters. We use the notation $\Re^n$ to indicate the set of all n-dimensional vectors. For any vector $\mathbf{x} \in \Re^n$, we use $x_1, x_2, \ldots, x_n$ to

indicate its components. Thus,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

The more economical notation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ will also be used even if we are referring to column vectors. We use $\mathbf{0}$ to denote the vector with all components equal to zero. The ith *unit vector* $\mathbf{e}_i$ is the vector with all components equal to zero except for the ith component which is equal to one.

The *transpose* $\mathbf{A}'$ of an $m \times n$ matrix $\mathbf{A}$ is the $n \times m$ matrix

$$\mathbf{A}' = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix};$$

that is, $[\mathbf{A}']_{ij} = [\mathbf{A}]_{ji}$. Similarly, if $\mathbf{x}$ is a vector in $\Re^n$, its transpose $\mathbf{x}'$ is the row vector with the same entries.

If $\mathbf{x}$ and $\mathbf{y}$ are two vectors in $\Re^n$, then

$$\mathbf{x}'\mathbf{y} = \mathbf{y}'\mathbf{x} = \sum_{i=1}^{n} x_i y_i.$$

This quantity is called the *inner product* of $\mathbf{x}$ and $\mathbf{y}$. Two vectors are called *orthogonal* if their inner product is zero. Note that $\mathbf{x}'\mathbf{x} \geq 0$ for every vector $\mathbf{x}$, with equality holding if and only if $\mathbf{x} = \mathbf{0}$. The expression $\sqrt{\mathbf{x}'\mathbf{x}}$ is the *Euclidean norm* of $\mathbf{x}$ and is denoted by $\|\mathbf{x}\|$. The *Schwartz inequality* asserts that for any two vectors of the same dimension, we have

$$|\mathbf{x}'\mathbf{y}| \leq \|\mathbf{x}\| \cdot \|\mathbf{y}\|,$$

with equality holding if and only if one of the two vectors is a scalar multiple of the other.

If $\mathbf{A}$ is an $m \times n$ matrix, we use $\mathbf{A}_j$ to denote its jth column, that is, $\mathbf{A}_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$. (This is our only exception to the rule of using lower case characters to represent vectors.) We also use $\mathbf{a}_i$ to denote the vector formed by the entries of the ith row, that is, $\mathbf{a}_i = (a_{i1}, a_{i2}, \ldots, a_{in})$. Thus,

$$\mathbf{A} = \begin{bmatrix} | & | & & | \\ \mathbf{A}_1 & \mathbf{A}_2 & \cdots & \mathbf{A}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} — & \mathbf{a}_1' & — \\ & \vdots & \\ — & \mathbf{a}_m' & — \end{bmatrix}.$$

Given two matrices $\mathbf{A}$, $\mathbf{B}$ of dimensions $m \times n$ and $n \times k$, respectively, their *product* $\mathbf{AB}$ is a matrix of dimensions $m \times k$ whose entries are given by

$$[\mathbf{AB}]_{ij} = \sum_{\ell=1}^{n} [\mathbf{A}]_{i\ell}[\mathbf{B}]_{\ell j} = \mathbf{a}_i' \mathbf{B}_j,$$

where $\mathbf{a}_i'$ is the ith row of $\mathbf{A}$, and $\mathbf{B}_j$ is the jth column of $\mathbf{B}$. Matrix multiplication is associative, i.e., $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$, but, in general, it is not commutative, that is, the equality $\mathbf{AB} = \mathbf{BA}$ is not always true. We also have $(\mathbf{AB})' = \mathbf{B}'\mathbf{A}'$.

Let $\mathbf{A}$ be an $m \times n$ matrix with columns $\mathbf{A}_i$. We then have $\mathbf{A}\mathbf{e}_i = \mathbf{A}_i$. Any vector $\mathbf{x} \in \Re^n$ can be written in the form $\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{e}_i$, which leads to

$$\mathbf{Ax} = \mathbf{A} \sum_{i=1}^{n} \mathbf{e}_i x_i = \sum_{i=1}^{n} \mathbf{A}\mathbf{e}_i x_i = \sum_{i=1}^{n} \mathbf{A}_i x_i.$$

A different representation of the matrix-vector product $\mathbf{Ax}$ is provided by the formula

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{a}_1'\mathbf{x} \\ \mathbf{a}_2'\mathbf{x} \\ \vdots \\ \mathbf{a}_m'\mathbf{x} \end{bmatrix},$$

where $\mathbf{a}_1', \ldots, \mathbf{a}_m'$ are the rows of $\mathbf{A}$.

A matrix is called *square* if the number m of its rows is equal to the number n of its columns. We use $\mathbf{I}$ to denote the *identity* matrix, which is a square matrix whose diagonal entries are equal to one and its off-diagonal entries are equal to zero. The identity matrix satisfies $\mathbf{IA} = \mathbf{A}$ and $\mathbf{BI} = \mathbf{B}$ for any matrices $\mathbf{A}$, $\mathbf{B}$ of dimensions compatible with those of $\mathbf{I}$.

If $\mathbf{x}$ is a vector, the notation $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{x} > \mathbf{0}$ means that every component of $\mathbf{x}$ is nonnegative (respectively, positive). If $\mathbf{A}$ is a matrix, the inequalities $\mathbf{A} \geq \mathbf{0}$ and $\mathbf{A} > \mathbf{0}$ have a similar meaning.

Matrix inversion

Let $\mathbf{A}$ be a square matrix. If there exists a square matrix $\mathbf{B}$ of the same dimensions satisfying $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$, we say that $\mathbf{A}$ is *invertible* or *nonsingular*. Such a matrix $\mathbf{B}$, called the *inverse* of $\mathbf{A}$, is unique and is denoted by $\mathbf{A}^{-1}$. We note that $(\mathbf{A}')^{-1} = (\mathbf{A}^{-1})'$. Also, if $\mathbf{A}$ and $\mathbf{B}$ are invertible matrices of the same dimensions, then $\mathbf{AB}$ is also invertible and $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$.

Given a finite collection of vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K \in \Re^n$, we say that they are *linearly dependent* if there exist real numbers $a_1, \ldots, a_K$, not all of them zero, such that $\sum_{k=1}^{K} a_k \mathbf{x}^k = \mathbf{0}$; otherwise, they are called *linearly independent*. An equivalent definition of linear independence requires that

none of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K$ is a linear combination of the remaining vectors (Exercise 1.18). We have the following result.

Theorem 1.2 *Let* $\mathbf{A}$ *be a square matrix. Then, the following statements are equivalent:*

(a) *The matrix* $\mathbf{A}$ *is invertible.*

(b) *The matrix* $\mathbf{A}'$ *is invertible.*

(c) *The determinant of* $\mathbf{A}$ *is nonzero.*

(d) *The rows of* $\mathbf{A}$ *are linearly independent.*

(e) *The columns of* $\mathbf{A}$ *are linearly independent.*

(f) *For every vector* $\mathbf{b}$*, the linear system* $\mathbf{Ax} = \mathbf{b}$ *has a unique solution.*

(g) *There exists some vector* $\mathbf{b}$ *such that the linear system* $\mathbf{Ax} = \mathbf{b}$ *has a unique solution.*

Assuming that $\mathbf{A}$ is an invertible square matrix, an explicit formula for the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ of the system $\mathbf{Ax} = \mathbf{b}$, is given by *Cramer's rule*. Specifically, the jth component of $\mathbf{x}$ is given by

$$x_j = \frac{\det(\mathbf{A}^j)}{\det(\mathbf{A})},$$

where $\mathbf{A}^j$ is the same matrix as $\mathbf{A}$, except that its jth column is replaced by $\mathbf{b}$. Here, as well as later, the notation $\det(\mathbf{A})$ is used to denote the *determinant* of a square matrix $\mathbf{A}$.

Subspaces and bases

A nonempty subset S of $\Re^n$ is called a *subspace* of $\Re^n$ if $a\mathbf{x} + b\mathbf{y} \in S$ for every $\mathbf{x}, \mathbf{y} \in S$ and every $a, b \in \Re$. If, in addition, $S \neq \Re^n$, we say that S is a *proper* subspace. Note that every subspace must contain the zero vector.

The *span* of a finite number of vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K$ in $\Re^n$ is the subspace of $\Re^n$ defined as the set of all vectors $\mathbf{y}$ of the form $\mathbf{y} = \sum_{k=1}^{K} a_k \mathbf{x}^k$, where each a_k is a real number. Any such vector $\mathbf{y}$ is called a *linear combination* of $\mathbf{x}^1, \ldots, \mathbf{x}^K$.

Given a subspace S of $\Re^n$, with $S \neq \{\mathbf{0}\}$, a *basis* of S is a collection of vectors that are linearly independent and whose span is equal to S. Every basis of a given subspace has the same number of vectors and this number is called the *dimension* of the subspace. In particular, the dimension of $\Re^n$ is equal to n and every proper subspace of $\Re^n$ has dimension smaller than n. Note that one-dimensional subspaces are lines through the origin; two-dimensional subspaces are planes through the origin. Finally, the set $\{\mathbf{0}\}$ is a subspace and its dimension is defined to be zero.

If S is a proper subspace of $\Re^n$, then there exists a nonzero vector $\mathbf{a}$ which is orthogonal to S, that is, $\mathbf{a}'\mathbf{x} = 0$ for every $\mathbf{x} \in S$. More generally, if S has dimension $m < n$, there exist $n - m$ linearly independent vectors that are orthogonal to S.

The result that follows provides some important facts regarding bases and linear independence.

Theorem 1.3 *Suppose that the span S of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K$ has dimension m. Then:*

(a) *There exists a basis of S consisting of m of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K$.*

(b) *If $k \leq m$ and $\mathbf{x}^1, \ldots, \mathbf{x}^k$ are linearly independent, we can form a basis of S by starting with $\mathbf{x}^1, \ldots, \mathbf{x}^k$, and choosing $m - k$ of the vectors $\mathbf{x}^{k+1}, \ldots, \mathbf{x}^K$.*

Proof. We only prove part (b), because (a) is the special case of part (b) with $k = 0$. If every vector $\mathbf{x}^{k+1}, \ldots, \mathbf{x}^K$ can be expressed as a linear combination of $\mathbf{x}^1, \ldots, \mathbf{x}^k$, then every vector in the span of $\mathbf{x}^1, \ldots, \mathbf{x}^K$ is also a linear combination of $\mathbf{x}^1, \ldots, \mathbf{x}^k$, and the latter vectors form a basis. (In particular, $m = k$.) Otherwise, at least one of the vectors $\mathbf{x}^{k+1}, \ldots, \mathbf{x}^K$ is linearly independent from $\mathbf{x}^1, \ldots, \mathbf{x}^k$. By picking one such vector, we now have $k+1$ of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^K$ that are linearly independent. By repeating this process $m - k$ times, we end up with the desired basis of S. □

Let $\mathbf{A}$ be a matrix of dimensions $m \times n$. The *column space* of $\mathbf{A}$ is the subspace of $\Re^m$ spanned by the columns of $\mathbf{A}$. The *row space* of $\mathbf{A}$ is the subspace of $\Re^n$ spanned by the rows of $\mathbf{A}$. The dimension of the column space is always equal to the dimension of the row space, and this number is called the *rank* of $\mathbf{A}$. Clearly, $\text{rank}(\mathbf{A}) \leq \min\{m, n\}$. The matrix $\mathbf{A}$ is said to have *full rank* if $\text{rank}(\mathbf{A}) = \min\{m, n\}$. Finally, the set $\{\mathbf{x} \in \Re^n \mid \mathbf{A}\mathbf{x} = \mathbf{0}\}$ is called the *nullspace* of $\mathbf{A}$; it is a subspace of $\Re^n$ and its dimension is equal to $n - \text{rank}(\mathbf{A})$.

Affine subspaces

Let S_0 be a subspace of $\Re^n$ and let $\mathbf{x}^0$ be some vector. If we add $\mathbf{x}^0$ to every element of S_0, this amounts to translating S_0 by $\mathbf{x}^0$. The resulting set S can be defined formally by

$$S = S_0 + \mathbf{x}^0 = \{\mathbf{x} + \mathbf{x}^0 \mid \mathbf{x} \in S_0\}.$$

In general, S is not a subspace, because it does not necessarily contain the zero vector, and it is called an *affine subspace*. The *dimension* of S is defined to be equal to the dimension of the underlying subspace S_0.

As an example, let $\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^k$ be some vectors in $\Re^n$, and consider the set S of all vectors of the form

$$\mathbf{x}^0 + \lambda_1 \mathbf{x}^1 + \cdots + \lambda_k \mathbf{x}^k,$$

where $\lambda_1, \ldots, \lambda_k$ are arbitrary scalars. For this case, S_0 can be identified with the span of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^k$, and S is an affine subspace. If the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^k$ are linearly independent, their span has dimension k, and the affine subspace S also has dimension k.

For a second example, we are given an $m \times n$ matrix $\mathbf{A}$ and a vector $\mathbf{b} \in \Re^m$, and we consider the set

$$S = \{\mathbf{x} \in \Re^n \mid \mathbf{A}\mathbf{x} = \mathbf{b}\},$$

which we assume to be nonempty. Let us fix some $\mathbf{x}^0$ such that $\mathbf{A}\mathbf{x}^0 = \mathbf{b}$. An arbitrary vector $\mathbf{x}$ belongs to S if and only if $\mathbf{A}\mathbf{x} = \mathbf{b} = \mathbf{A}\mathbf{x}^0$, or $\mathbf{A}(\mathbf{x} - \mathbf{x}^0) = \mathbf{0}$. Hence, $\mathbf{x} \in S$ if and only if $\mathbf{x} - \mathbf{x}^0$ belongs to the subspace $S_0 = \{\mathbf{y} \mid \mathbf{A}\mathbf{y} = \mathbf{0}\}$. We conclude that $S = \{\mathbf{y} + \mathbf{x}^0 \mid \mathbf{y} \in S_0\}$, and S is an affine subspace of $\Re^n$. If $\mathbf{A}$ has m linearly independent rows, its nullspace S_0 has dimension $n - m$. Hence, the affine subspace S also has dimension $n - m$. Intuitively, if $\mathbf{a}_i'$ are the rows of $\mathbf{A}$, each one of the constraints $\mathbf{a}_i'\mathbf{x} = b_i$ removes one degree of freedom from $\mathbf{x}$, thus reducing the dimension from n to $n - m$; see Figure 1.7 for an illustration.

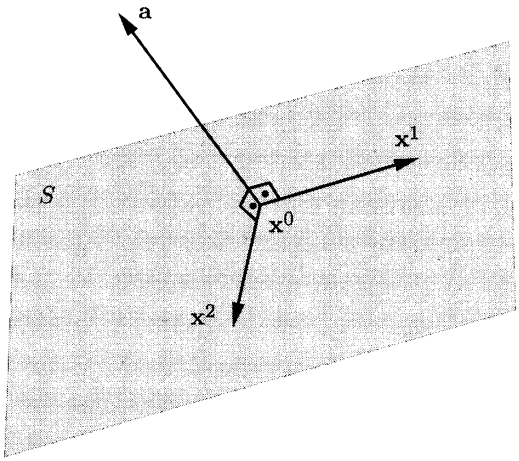

Figure 1.7: Consider a set S in $\Re^3$ defined by a single equality constraint $\mathbf{a}'\mathbf{x} = b$. Let $\mathbf{x}^0$ be an element of S. The vector $\mathbf{a}$ is perpendicular to S. If $\mathbf{x}^1$ and $\mathbf{x}^2$ are linearly independent vectors that are orthogonal to $\mathbf{a}$, then every $\mathbf{x} \in S$ is of the form $\mathbf{x} = \mathbf{x}^0 + \lambda_1 \mathbf{x}^1 + \lambda_2 \mathbf{x}^2$. In particular, S is a two-dimensional affine subspace.

1.6 Algorithms and operation counts

Optimization problems such as linear programming and, more generally, all computational problems are solved by *algorithms*. Loosely speaking, an algorithm is a finite set of instructions of the type used in common programming languages (arithmetic operations, conditional statements, read and write statements, etc.). Although the running time of an algorithm may depend substantially on clever programming or on the computer hardware available, we are interested in comparing algorithms without having to examine the details of a particular implementation. As a first approximation, this can be accomplished by counting the number of arithmetic operations (additions, multiplications, divisions, comparisons) required by an algorithm. This approach is often adequate even though it ignores the fact that adding or multiplying large integers or high-precision floating point numbers is more demanding than adding or multiplying single-digit integers. A more refined approach will be discussed briefly in Chapter 8.

Example 1.9

(a) Let $\mathbf{a}$ and $\mathbf{b}$ be vectors in $\Re^n$. The natural algorithm for computing $\mathbf{a}'\mathbf{b}$ requires n multiplications and $n-1$ additions, for a total of $2n-1$ arithmetic operations.

(b) Let $\mathbf{A}$ and $\mathbf{B}$ be matrices of dimensions $n \times n$. The traditional way of computing $\mathbf{AB}$ forms the inner product of a row of $\mathbf{A}$ and a column of $\mathbf{B}$ to obtain an entry of $\mathbf{AB}$. Since there are n^2 entries to be evaluated, a total of $(2n-1)n^2$ arithmetic operations are involved.

In Example 1.9, an exact operation count was possible. However, for more complicated problems and algorithms, an exact count is usually very difficult. For this reason, we will settle for an estimate of the rate of growth of the number of arithmetic operations, as a function of the problem parameters. Thus, in Example 1.9, we might be content to say that the number of operations in the computation of an inner product increases linearly with n, and the number of operations in matrix multiplication increases cubically with n. This leads us to the order of magnitude notation that we define next.

Definition 1.2 *Let f and g be functions that map positive numbers to positive numbers.*

(a) *We write $f(n) = O\big(g(n)\big)$ if there exist positive numbers n_0 and c such that $f(n) \leq cg(n)$ for all $n \geq n_0$.*

(b) *We write $f(n) = \Omega\big(g(n)\big)$ if there exist positive numbers n_0 and c such that $f(n) \geq cg(n)$ for all $n \geq n_0$.*

(c) *We write $f(n) = \Theta\big(g(n)\big)$ if both $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$ hold.*

For example, we have $3n^3 + n^2 + 10 = \Theta(n^3)$, $n \log n = O(n^2)$, and $n \log n = \Omega(n)$.

While the running time of the algorithms considered in Example 1.9 is predictable, the running time of more complicated algorithms often depends on the numerical values of the input data. In such cases, instead of trying to estimate the running time for each possible choice of the input, it is customary to estimate the running time for the *worst possible input data* of a given "size." For example, if we have an algorithm for linear programming, we might be interested in estimating its worst-case running time over all problems with a given number of variables and constraints. This emphasis on the worst case is somewhat conservative and, in practice, the "average" running time of an algorithm might be more relevant. However, the average running time is much more difficult to estimate, or even to define, and for this reason, the worst-case approach is widely used.

Example 1.10 (Operation count of linear system solvers and matrix inversion) Consider the problem of solving a system of n linear equations in n unknowns. The classical method that eliminates one variable at a time (Gaussian elimination) is known to require $O(n^3)$ arithmetic operations in order to either compute a solution or to decide that no solution exists. Practical methods for matrix inversion also require $O(n^3)$ arithmetic operations. These facts will be of use later on.

Is the $O(n^3)$ running time of Gaussian elimination good or bad? Some perspective into this question is provided by the following observation: each time that technological advances lead to computer hardware that is faster by a factor of 8 (presumably every few years), we can solve problems of twice the size than earlier possible. A similar argument applies to algorithms whose running time is $O(n^k)$ for some positive integer k. Such algorithms are said to run in *polynomial time*.

Algorithms also exist whose running time is $\Omega(2^{cn})$, where n is a parameter representing problem size and c is a constant; these are said to take at least *exponential time*. For such algorithms and if $c = 1$, each time that computer hardware becomes faster by a factor of 2, we can increase the value of n that we can handle only by 1. It is then reasonable to expect that no matter how much technology improves, problems with truly large values of n will always be difficult to handle.

Example 1.11 Suppose that we have a choice of two algorithms. The running time of the first is $10^n/100$ (exponential) and the running time of the second is $10n^3$ (polynomial). For very small n, e.g., for $n = 3$, the exponential time algorithm is preferable. To gain some perspective as to what happens for larger n, suppose that we have access to a workstation that can execute 10^7 arithmetic operations per second and that we are willing to let it run for 1000 seconds. Let us figure out what size problems can each algorithm handle within this time frame. The equation $10^n/100 = 10^7 \times 1000$ yields $n = 12$, whereas the equation

$10n^3 = 10^7 \times 1000$ yields $n = 1000$, indicating that the polynomial time algorithm allows us to solve much larger problems.

The point of view emerging from the above discussion is that, as a first cut, it is useful to juxtapose polynomial and exponential time algorithms, the former being viewed as relatively fast and efficient, and the latter as relatively slow. This point of view is justified in many – but not all – contexts and we will be returning to it later in this book.

1.7 Exercises

Exercise 1.1* Suppose that a function $f : \Re^n \mapsto \Re$ is both concave and convex. Prove that f is an affine function.

Exercise 1.2 Suppose that $f_1, \ldots, f_m$ are convex functions from $\Re^n$ into $\Re$ and let $f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$.

(a) Show that if each f_i is convex, so is f.

(b) Show that if each f_i is piecewise linear and convex, so is f.

Exercise 1.3 Consider the problem of minimizing a cost function of the form $\mathbf{c}'\mathbf{x} + f(\mathbf{d}'\mathbf{x})$, subject to the linear constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$. Here, $\mathbf{d}$ is a given vector and the function $f : \Re \mapsto \Re$ is as specified in Figure 1.8. Provide a linear programming formulation of this problem.

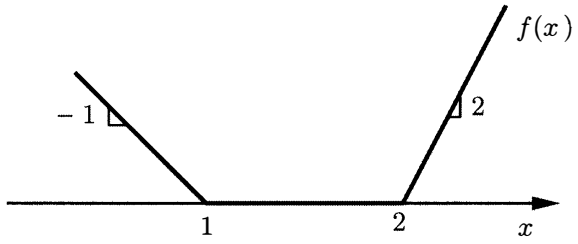

Figure 1.8: The function f of Exercise 1.3.

Exercise 1.4 Consider the problem

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3|x_2 - 10| \\ \text{subject to} \quad & |x_1 + 2| + |x_2| \leq 5, \end{aligned}$$

and reformulate it as a linear programming problem.

Exercise 1.5 Consider a linear optimization problem, with absolute values, of the following form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b} \\ & y_i = |x_i|, \qquad \forall\, i. \end{aligned}$$

Assume that all entries of $\mathbf{B}$ and $\mathbf{d}$ are nonnegative.

(a) Provide two different linear programming formulations, along the lines discussed in Section 1.3.

(b) Show that the original problem and the two reformulations are equivalent in the sense that either all three are infeasible, or all three have the same optimal cost.

(c) Provide an example to show that if $\mathbf{B}$ has negative entries, the problem may have a local minimum that is not a global minimum. (It will be seen in Chapter 2 that this is never the case in linear programming problems. Hence, in the presence of such negative entries, a linear programming reformulation is implausible.)

Exercise 1.6 Provide linear programming formulations of the two variants of the rocket control problem discussed at the end of Section 1.3.

Exercise 1.7 **(The moment problem)** Suppose that Z is a random variable taking values in the set $0, 1, \ldots, K$, with probabilities $p_0, p_1, \ldots, p_K$, respectively. We are given the values of the first two moments $E[Z] = \sum_{k=0}^{K} k p_k$ and $E[Z^2] = \sum_{k=0}^{K} k^2 p_k$ of Z and we would like to obtain upper and lower bounds on the value of the fourth moment $E[Z^4] = \sum_{k=0}^{K} k^4 p_k$ of Z. Show how linear programming can be used to approach this problem.

Exercise 1.8 **(Road lighting)** Consider a road divided into n segments that is illuminated by m lamps. Let p_j be the power of the jth lamp. The illumination I_i of the ith segment is assumed to be $\sum_{j=1}^{m} a_{ij} p_j$, where a_{ij} are known coefficients. Let I_i^* be the desired illumination of road i.

We are interested in choosing the lamp powers p_j so that the illuminations I_i are close to the desired illuminations I_i^*. Provide a reasonable linear programming formulation of this problem. Note that the wording of the problem is loose and there is more than one possible formulation.

Exercise 1.9 Consider a school district with I neighborhoods, J schools, and G grades at each school. Each school j has a capacity of C_{jg} for grade g. In each neighborhood i, the student population of grade i is S_{ig}. Finally, the distance of school j from neighborhood i is d_{ij}. Formulate a linear programming problem whose objective is to assign all students to schools, while minimizing the total distance traveled by all students. (You may ignore the fact that numbers of students must be integer.)

Exercise 1.10 **(Production and inventory planning)** A company must deliver d_i units of its product at the end of the ith month. Material produced during

a month can be delivered either at the end of the same month or can be stored as inventory and delivered at the end of a subsequent month; however, there is a storage cost of c_1 dollars per month for each unit of product held in inventory. The year begins with zero inventory. If the company produces x_i units in month i and x_{i+1} units in month $i+1$, it incurs a cost of $c_2|x_{i+1} - x_i|$ dollars, reflecting the cost of switching to a new production level. Formulate a linear programming problem whose objective is to minimize the total cost of the production and inventory schedule over a period of twelve months. Assume that inventory left at the end of the year has no value and does not incur any storage costs.

Exercise 1.11 (Optimal currency conversion) Suppose that there are N available currencies, and assume that one unit of currency i can be exchanged for r_{ij} units of currency j. (Naturally, we assume that $r_{ij} > 0$.) There also certain regulations that impose a limit u_i on the total amount of currency i that can be exchanged on any given day. Suppose that we start with B units of currency 1 and that we would like to maximize the number of units of currency N that we end up with at the end of the day, through a sequence of currency transactions. Provide a linear programming formulation of this problem. Assume that for any sequence $i_1, \ldots, i_k$ of currencies, we have $r_{i_1 i_2} r_{i_2 i_3} \cdots r_{i_{k-1} i_k} r_{i_k i_1} \leq 1$, which means that wealth cannot be multiplied by going through a cycle of currencies.

Exercise 1.12 (Chebychev center) Consider a set P described by linear inequality constraints, that is, $P = \{\mathbf{x} \in \Re^n \mid \mathbf{a}_i'\mathbf{x} \leq b_i,\ i = 1, \ldots, m\}$. A ball with center $\mathbf{y}$ and radius r is defined as the set of all points within (Euclidean) distance r from $\mathbf{y}$. We are interested in finding a ball with the largest possible radius, which is entirely contained within the set P. (The center of such a ball is called the *Chebychev center* of P.) Provide a linear programming formulation of this problem.

Exercise 1.13 (Linear fractional programming) Consider the problem

$$\begin{array}{ll} \text{minimize} & \dfrac{\mathbf{c}'\mathbf{x} + d}{\mathbf{f}'\mathbf{x} + g} \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{f}'\mathbf{x} + g > 0. \end{array}$$

Suppose that we have some prior knowledge that the optimal cost belongs to an interval $[K, L]$. Provide a procedure, that uses linear programming as a subroutine, and that allows us to compute the optimal cost within any desired accuracy. *Hint:* Consider the problem of deciding whether the optimal cost is less than or equal to a certain number.

Exercise 1.14 A company produces and sells two different products. The demand for each product is unlimited, but the company is constrained by cash availability and machine capacity.

Each unit of the first and second product requires 3 and 4 machine hours, respectively. There are 20,000 machine hours available in the current production period. The production costs are \$3 and \$2 per unit of the first and second product, respectively. The selling prices of the first and second product are \$6 and \$5.40 per unit, respectively. The available cash is \$4,000; furthermore, 45%

of the sales revenues from the first product and 30% of the sales revenues from the second product will be made available to finance operations during the current period.

(a) Formulate a linear programming problem that aims at maximizing net income subject to the cash availability and machine capacity limitations.

(b) Solve the problem graphically to obtain an optimal solution.

(c) Suppose that the company could increase its available machine hours by 2,000, after spending \$400 for certain repairs. Should the investment be made?

Exercise 1.15 A company produces two kinds of products. A product of the first type requires 1/4 hours of assembly labor, 1/8 hours of testing, and \$1.2 worth of raw materials. A product of the second type requires 1/3 hours of assembly, 1/3 hours of testing, and \$0.9 worth of raw materials. Given the current personnel of the company, there can be at most 90 hours of assembly labor and 80 hours of testing, each day. Products of the first and second type have a market value of \$9 and \$8, respectively.

(a) Formulate a linear programming problem that can be used to maximize the daily profit of the company.

(b) Consider the following two modifications to the original problem:

(i) Suppose that up to 50 hours of overtime assembly labor can be scheduled, at a cost of \$7 per hour.

(ii) Suppose that the raw material supplier provides a 10% discount if the daily bill is above \$300.

Which of the above two elements can be easily incorporated into the linear programming formulation and how? If one or both are not easy to incorporate, indicate how you might nevertheless solve the problem.

Exercise 1.16 A manager of an oil refinery has 8 million barrels of crude oil A and 5 million barrels of crude oil B allocated for production during the coming month. These resources can be used to make either gasoline, which sells for \$38 per barrel, or home heating oil, which sells for \$33 per barrel. There are three production processes with the following characteristics:

	Process 1	Process 2	Process 3
Input crude A	3	1	5
Input crude B	5	1	3
Output gasoline	4	1	3
Output heating oil	3	1	4
Cost	\$51	\$11	\$40

All quantities are in barrels. For example, with the first process, 3 barrels of crude A and 5 barrels of crude B are used to produce 4 barrels of gasoline and

3 barrels of heating oil. The costs in this table refer to variable and allocated overhead costs, and there are no separate cost items for the cost of the crudes. Formulate a linear programming problem that would help the manager maximize net revenue over the next month.

Exercise 1.17 (Investment under taxation) An investor has a portfolio of n different stocks. He has bought s_i shares of stock i at price p_i, $i = 1, \ldots, n$. The current price of one share of stock i is q_i. The investor expects that the price of one share of stock i in one year will be r_i. If he sells shares, the investor pays transaction costs at the rate of 1% of the amount transacted. In addition, the investor pays taxes at the rate of 30% on capital gains. For example, suppose that the investor sells 1,000 shares of a stock at \$50 per share. He has bought these shares at \$30 per share. He receives \$50,000. However, he owes $0.30 \times (50{,}000 - 30{,}000) = \$6{,}000$ on capital gain taxes and $0.01 \times (50{,}000) = \$500$ on transaction costs. So, by selling 1,000 shares of this stock he nets $50{,}000 - 6{,}000 - 500 = \$43{,}500$. Formulate the problem of selecting how many shares the investor needs to sell in order to raise an amount of money K, net of capital gains and transaction costs, while maximizing the expected value of his portfolio next year.

Exercise 1.18 Show that the vectors in a given finite collection are linearly independent if and only if none of the vectors can be expressed as a linear combination of the others.

Exercise 1.19 Suppose that we are given a set of vectors in $\Re^n$ that form a basis, and let $\mathbf{y}$ be an arbitrary vector in $\Re^n$. We wish to express $\mathbf{y}$ as a linear combination of the basis vectors. How can this be accomplished?

Exercise 1.20

(a) Let $S = \{\mathbf{A}\mathbf{x} \mid \mathbf{x} \in \Re^n\}$, where $\mathbf{A}$ is a given matrix. Show that S is a subspace of $\Re^n$.

(b) Assume that S is a proper subspace of $\Re^n$. Show that there exists a matrix $\mathbf{B}$ such that $S = \{\mathbf{y} \in \Re^n \mid \mathbf{B}\mathbf{y} = \mathbf{0}\}$. *Hint:* Use vectors that are orthogonal to S to form the matrix $\mathbf{B}$.

(c) Suppose that V is an m-dimensional affine subspace of $\Re^n$, with $m < n$. Show that there exist linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_{n-m}$, and scalars $b_1, \ldots, b_{n-m}$, such that

$$V = \left\{\mathbf{y} \mid \mathbf{a}_i'\mathbf{y} = b_i,\ i = 1, \ldots, n-m\right\}.$$

1.8 History, notes, and sources

The word "programming" has been used traditionally by planners to describe the process of operations planning and resource allocation. In the 1940s, it was realized that this process could often be aided by solving optimization problems involving linear constraints and linear objectives. The term "linear programming" then emerged. The initial impetus came in the aftermath of World War II, within the context of military planning problems. In 1947, Dantzig proposed an algorithm, the *simplex method*, which

made the solution of linear programming problems practical. There followed a period of intense activity during which many important problems in transportation, economics, military operations, scheduling, etc., were cast in this framework. Since then, computer technology has advanced rapidly, the range of applications has expanded, new powerful methods have been discovered, and the underlying mathematical understanding has become deeper and more comprehensive. Today, linear programming is a routinely used tool that can be found in some spreadsheet software packages.

Dantzig's development of the simplex method has been a defining moment in the history of the field, because it came at a time of growing practical needs and of advances in computing technology. But, as is the case with most "scientific revolutions," the history of the field is much richer. Early work goes back to Fourier, who in 1824 developed an algorithm for solving systems of linear inequalities. Fourier's method is far less efficient than the simplex method, but this issue was not relevant at the time. In 1910, de la Vallée Poussin developed a method, similar to the simplex method, for minimizing $\max_i |b_i - \mathbf{a}_i'\mathbf{x}|$, a problem that we discussed in Section 1.3.

In the late 1930s, the Soviet mathematician Kantorovich became interested in problems of optimal resource allocation in a centrally planned economy, for which he gave linear programming formulations. He also provided a solution method, but his work did not become widely known at the time. Around the same time, several models arising in classical, Walrasian, economics were studied and refined, and led to formulations closely related to linear programming. Koopmans, an economist, played an important role and eventually (in 1975) shared the Nobel Prize in economic science with Kantorovich.

On the theoretical front, the mathematical structures that underlie linear programming were independently studied, in the period 1870-1930, by many prominent mathematicians, such as Farkas, Minkowski, Carathéodory, and others. Also, in 1928, von Neumann developed an important result in game theory that would later prove to have strong connections with the deeper structure of linear programming.

Subsequent to Dantzig's work, there has been much and important research in areas such as large scale optimization, network optimization, interior point methods, integer programming, and complexity theory. We defer the discussion of this research to the notes and sources sections of later chapters. For a more detailed account of the history of linear programming, the reader is referred to Schrijver (1986), Orden (1993), and the volume edited by Lenstra, Rinnooy Kan, and Schrijver (1991) (see especially the article by Dantzig in that volume).

There are several texts that cover the general subject of linear programming, starting with a comprehensive one by Dantzig (1963). Some more recent texts are Papadimitriou and Steiglitz (1982), Chvátal (1983), Murty (1983), Luenberger (1984), Bazaraa, Jarvis, and Sherali (1990). Fi-

nally, Schrijver (1986) is a comprehensive, but more advanced reference on the subject.

1.1. The formulation of the diet problem is due to Stigler (1945).

1.2. The case study on DEC's production planning was developed by Freund and Shannahan (1992). Methods for dealing with the nurse scheduling and other cyclic problems are studied by Bartholdi, Orlin, and Ratliff (1980). More information on pattern classification can be found in Duda and Hart (1973), or Haykin (1994).

1.3. A deep and comprehensive treatment of convex functions and their properties is provided by Rockafellar (1970). Linear programming arises in control problems, in ways that are more sophisticated than what is described here; see, e.g., Dahleh and Diaz-Bobillo (1995).

1.5. For an introduction to linear algebra, see Strang (1988).

1.6. For a more detailed treatment of algorithms and their computational requirements, see Lewis and Papadimitriou (1981), Papadimitriou and Steiglitz (1982), or Cormen, Leiserson, and Rivest (1990).

1.7. Exercise 1.8 is adapted from Boyd and Vandenberghe (1995). Exercises 1.9 and 1.14 are adapted from Bradley, Hax, and Magnanti (1977). Exercise 1.11 is adapted from Ahuja, Magnanti, and Orlin (1993).