1. What Is It, and What For?

Linear programming, surprisingly, is not directly related to computer programming. The term was introduced in the 1950s when computers were few and mostly top secret, and the word programming was a military term that, at that time, referred to plans or schedules for training, logistical supply, or deployment of men. The word linear suggests that feasible plans are restricted by linear constraints (inequalities), and also that the quality of the plan (e.g., costs or duration) is also measured by a linear function of the considered quantities. In a similar spirit, linear programming soon started to be used for planning all kinds of economic activities, such as transport of raw materials and products among factories, sowing various crop plants, or cutting paper rolls into shorter ones in sizes ordered by customers. The phrase "planning with linear constraints" would perhaps better capture this original meaning of linear programming. However, the term linear programming has been well established for many years, and at the same time, it has acquired a considerably broader meaning: Not only does it play a role only in mathematical economy, it appears frequently in computer science and in many other fields.

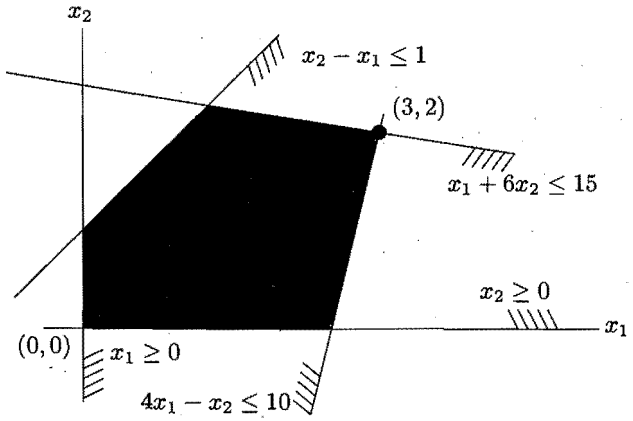
1.1 A Linear Program

We begin with a very simple linear programming problem (or **linear program** for short):

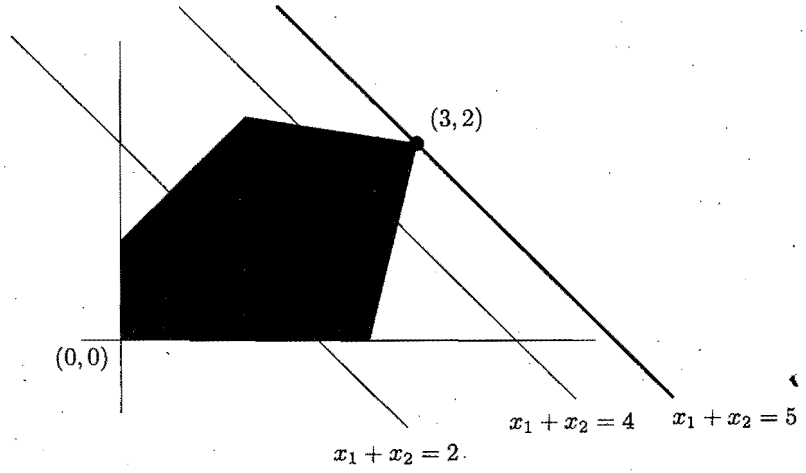
$$\begin{array}{ll} \text{Maximize the value} & x_1 + x_2 \\ \text{among all vectors } (x_1, x_2) \in \mathbb{R}^2 & \\ \text{satisfying the constraints} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_2 - x_1 \leq 1 \\ & x_1 + 6x_2 \leq 15 \\ & 4x_1 - x_2 \leq 10. \end{array}$$

For this linear program we can easily draw a picture. The set $\{\mathbf{x} \in \mathbb{R}^2 : x_2 - x_1 \leq 1\}$ is the half-plane lying below the line $x_2 = x_1 + 1$, and similarly,

each of the remaining four inequalities defines a half-plane. The set of all vectors satisfying the five constraints simultaneously is a convex polygon:



Which point of this polygon maximizes the value of $x_1 + x_2$? The one lying "farthest in the direction" of the vector $(1,1)$ drawn by the arrow; that is, the point $(3,2)$. The phrase "farthest in the direction" is in quotation marks since it is not quite precise. To make it more precise, we consider a line perpendicular to the arrow, and we think of translating it in the direction of the arrow. Then we are seeking a point where the moving line intersects our polygon for the last time. (Let us note that the function $x_1 + x_2$ is constant on each line perpendicular to the vector $(1,1)$, and as we move the line in the direction of that vector, the value of the function increases.) See the next illustration:



In a general linear program we want to find a vector $\mathbf{x}^* \in \mathbb{R}^n$ maximizing (or minimizing) the value of a given linear function among all vectors $\mathbf{x} \in \mathbb{R}^n$ that satisfy a given system of linear equations and inequalities. The linear function to be maximized, or sometimes minimized, is called the **objective function**. It has the form $\mathbf{c}^T\mathbf{x} = c_1x_1 + \cdots + c_nx_n$, where $\mathbf{c} \in \mathbb{R}^n$ is a given vector.[1]

The linear equations and inequalities in the linear program are called the **constraints**. It is customary to denote the number of constraints by m.

A linear program is often written using matrices and vectors, in a way similar to the notation $A\mathbf{x} = \mathbf{b}$ for a system of linear equations in linear algebra. To make such a notation simpler, we can replace each equation in the linear program by two opposite inequalities. For example, instead of the constraint $x_1 + 3x_2 = 7$ we can put the two constraints $x_1 + 3x_2 \le 7$ and $x_1 + 3x_2 \ge 7$. Moreover, the direction of the inequalities can be reversed by changing the signs: $x_1 + 3x_2 \ge 7$ is equivalent to $-x_1 - 3x_2 \le -7$, and thus we can assume that all inequality signs are "$\le$", say, with all variables appearing on the left-hand side. Finally, minimizing an objective function $\mathbf{c}^T\mathbf{x}$ is equivalent to maximizing $-\mathbf{c}^T\mathbf{x}$, and hence we can always pass to a maximization problem. After such modifications each linear program can be expressed as follows:

$$\begin{array}{ll} \text{Maximize the value of} & \mathbf{c}^T\mathbf{x} \\ \text{among all vectors } \mathbf{x} \in \mathbb{R}^n \text{ satisfying} & A\mathbf{x} \le \mathbf{b}, \end{array}$$

where A is a given $m \times n$ real matrix and $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ are given vectors. Here the relation $\le$ holds for two vectors of equal length if and only if it holds componentwise.

Any vector $\mathbf{x} \in \mathbb{R}^n$ satisfying all constraints of a given linear program is a **feasible solution**. Each $\mathbf{x}^* \in \mathbb{R}^n$ that gives the maximum possible value of $\mathbf{c}^T\mathbf{x}$ among all feasible $\mathbf{x}$ is called an **optimal solution**, or **optimum** for short. In our linear program above we have $n = 2$, $m = 5$, and $\mathbf{c} = (1,1)$. The only optimal solution is the vector $(3,2)$, while, for instance, $(2, \frac{3}{2})$ is a feasible solution that is not optimal.

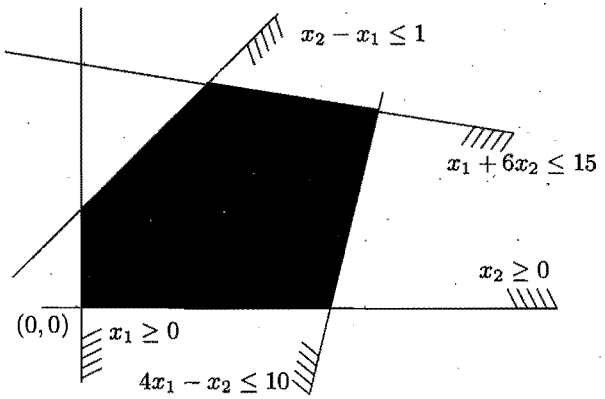
A linear program may in general have a single optimal solution, or infinitely many optimal solutions, or none at all.

We have seen a situation with a single optimal solution in the first example of a linear program. We will present examples of the other possible situations.

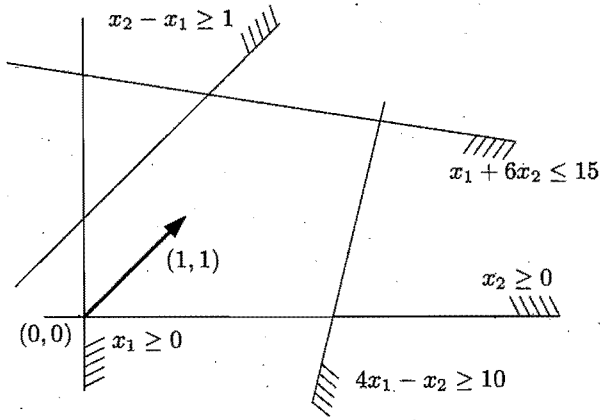
[1] Here we regard the vector $\mathbf{c}$ as an $n \times 1$ matrix, and so the expression $\mathbf{c}^T\mathbf{x}$ is a product of a $1 \times n$ matrix and an $n \times 1$ matrix. This product, formally speaking, should be a 1×1 matrix, but we regard it as a real number.

Some readers might wonder: If we consider $\mathbf{c}$ a column vector, why, in the example above, don't we write it as a column or as $(1,1)^T$? For us, a vector is an n-tuple of numbers, and when writing an explicit vector, we separate the numbers by commas, as in $\mathbf{c} = (1,1)$. Only if a vector appears in a context where one expects a matrix, that is, in a product of matrices, *then* it is regarded as (or "converted to") an $n \times 1$ matrix. (However, sometimes we declare a vector to be a row vector, and then it behaves as a $1 \times n$ matrix.)

If we change the vector $\mathbf{c}$ in the example to $(\frac{1}{6}, 1)$, all points on the side of the polygon drawn thick in the next picture are optimal solutions:

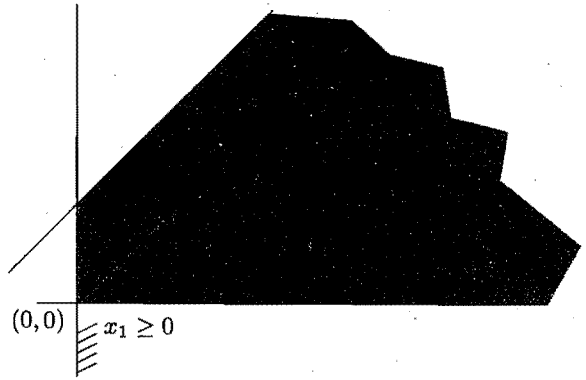


If we reverse the directions of the inequalities in the constraints $x_2 - x_1 \leq 1$ and $4x_1 - x_2 \leq 10$ in our first example, we obtain a linear program that has no feasible solution, and hence no optimal solution either:



Such a linear program is called **infeasible**.

Finally, an optimal solution need not exist even when there are feasible solutions. This happens when the objective function can attain arbitrarily large values (such a linear program is called **unbounded**). This is the case when we remove the constraints $4x_1 - x_2 \leq 10$ and $x_1 + 6x_2 \leq 15$ from the initial example, as shown in the next picture:



Let us summarize: We have seen that a linear program can have one or infinitely many optimal solutions, but it may also be unbounded or infeasible. Later we will prove that no other situations can occur.

We have solved the initial linear program graphically. It was easy since there are only two variables. However, for a linear program with four variables we won't even be able to make a picture, let alone find an optimal solution graphically. A substantial linear program in practice often has several thousand variables, rather than two or four. A graphical illustration is useful for understanding the notions and procedures of linear programming, but as a computational method it is worthless. Sometimes it may even be misleading, since objects in high dimension may behave in a way quite different from what the intuition gained in the plane or in three-dimensional space suggests.

One of the key pieces of knowledge about linear programming that one should remember forever is this:

> A linear program is efficiently solvable, both in theory and in practice.

- *In practice*, a number of software packages are available. They can handle inputs with several thousands of variables and constraints. Linear programs with a special structure, for example, with a small number of nonzero coefficients in each constraint, can often be managed even with a much larger number of variables and constraints.
- *In theory*, algorithms have been developed that provably solve each linear program in time bounded by a certain polynomial function of the input size. The input size is measured as the total number of bits needed to write down all coefficients in the objective function and in all the constraints.

These two statements summarize the results of long and strenuous research, and efficient methods for linear programming are not simple.

In order that the above piece of knowledge will also make sense forever, one should not forget what a linear program is, so we repeat it once again:

A linear program is the problem of maximizing a given linear function over the set of all vectors that satisfy a given system of linear equations and inequalities. Each linear program can easily be transformed to the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b}.$$

1.2 What Can Be Found in This Book

The rest of Chapter 1 briefly discusses the history and importance of linear programming and connects it to linear algebra.

For a large majority of readers it can be expected that whenever they encounter linear programming in practice or in research, they will be using it as a black box. From this point of view Chapter 2 is crucial, since it describes a number of algorithmic problems that can be solved via linear programming.

The closely related Chapter 3 discusses integer programming, in which one also optimizes a linear function over a set of vectors determined by linear constraints, but moreover, the variables must attain integer values. In this context we will see how linear programming can help in approximate solutions of hard computational problems.

Chapter 4 brings basic theoretical results on linear programming and on the geometric structure of the set of all feasible solutions. Notions introduced there, such as convexity and convex polyhedra, are important in many other branches of mathematics and computer science as well.

Chapter 5 covers the simplex method, which is a fundamental algorithm for linear programming. In full detail it is relatively complicated, and from the contemporary point of view it is not necessarily the central topic in a first course on linear programming. In contrast, some traditional introductions to linear programming are focused almost solely on the simplex method.

In Chapter 6 we will state and prove the duality theorem, which is one of the principal theoretical results in linear programming and an extremely useful tool for proofs.

Chapter 7 deals with two other important algorithmic approaches to linear programming: the ellipsoid method and the interior point method. Both of them are rather intricate and we omit some technical issues.

Chapter 8 collects several slightly more advanced applications of linear programming from various fields, each with motivation and some background material.

Chapter 9 contains remarks on software available for linear programming and on the literature.

Linear algebra is the main mathematical tool throughout the book. The needed linear-algebraic notions and results are summarized in an appendix.

The book concludes with a glossary of terms that are common in linear programming but do not appear in the main text. Some of them are listed to

ensure that our index can compete with those of thicker books, and others appear as background material for the advanced reader.

Two levels of text. This book should serve mainly as an introductory text for undergraduate and early graduate students, and so we do not want to assume previous knowledge beyond the usual basic undergraduate courses. However, many of the key results in linear programming, which would be a pity to omit, are not easy to prove, and sometimes they use mathematical methods whose knowledge cannot be expected at the undergraduate level. Consequently, the text is divided into two levels. On the basic level we are aiming at full and sufficiently detailed proofs.

> The second, more advanced, and "edifying" level is typographically distinguished like this. In such parts, intended chiefly for mathematically more mature readers, say graduate or PhD students, we include sketches of proofs and somewhat imprecise formulations of more advanced results. Whoever finds these passages incomprehensible may freely ignore them; the basic text should also make sense without them.

1.3 Linear Programming and Linear Algebra

The basics of linear algebra can be regarded as a theory of systems of linear equations. Linear algebra considers many other things as well, but systems of linear equations are surely one of the core subjects. A key algorithm is Gaussian elimination, which efficiently finds a solution of such a system, and even a description of the set of all solutions. Geometrically, the solution set is an affine subspace of $\mathbb{R}^n$, which is an important linear-algebraic notion.[2]

In a similar spirit, the discipline of linear programming can be regarded as a theory of systems of linear *inequalities*.

> In a linear program this is somewhat obscured by the fact that we do not look for an arbitrary solution of the given system of inequalities, but rather a solution maximizing a given objective function. But it can be shown that finding an (arbitrary) feasible solution of a linear program, if one exists, is computationally almost equally difficult as finding an optimal solution. Let us outline how one can gain an optimal solution, provided that feasible solutions can be computed (a different and more elegant way will be described in Section 6.1). If we somehow know in advance that, for instance, the maximum value of the objective function in a given linear program lies between 0 and 100, we can first ask, whether there exists a feasible $\mathbf{x} \in \mathbb{R}^n$ for which the objective

[2] An affine subspace is a linear subspace translated by a fixed vector $\mathbf{x} \in \mathbb{R}^n$. For example, every point, every line, and $\mathbb{R}^2$ itself are the affine subspaces of $\mathbb{R}^2$.

function is at least 50. That is, we add to the existing constraints a new constraint requiring that the value of the objective function be at least 50, and we find out whether this auxiliary linear program has a feasible solution. If yes, we will further ask, by the same trick, whether the objective function can be at least 75, and if not, we will check whether it can be at least 25. A reader with computer-science-conditioned reflexes has probably already recognized the strategy of *binary search*, which allows us to quickly localize the maximum value of the objective function with great accuracy.

Geometrically, the set of all solutions of a system of linear inequalities is an intersection of finitely many half-spaces in $\mathbb{R}^n$. Such a set is called a *convex polyhedron*, and familiar examples of convex polyhedra in $\mathbb{R}^3$ are a cube, a rectangular box, a tetrahedron, and a regular dodecahedron. Convex polyhedra are mathematically much more complex objects than vector subspaces or affine subspaces (we will return to this later). So actually, we can be grateful for the objective function in a linear program: It is enough to compute a single point $\mathbf{x}^* \in \mathbb{R}^n$ as a solution and we need not worry about the whole polyhedron.

In linear programming, a role comparable to that of Gaussian elimination in linear algebra is played by the *simplex method*. It is an algorithm for solving linear programs, usually quite efficient, and it also allows one to prove theoretical results.

Let us summarize the analogies between linear algebra and linear programming in tabular form:

	Basic problem	Algorithm	Solution set
Linear algebra	system of linear equations	Gaussian elimination	affine subspace
Linear programming	system of linear inequalities	simplex method	convex polyhedron

1.4 Significance and History of Linear Programming

In a special issue of the journal *Computing in Science & Engineering*, the simplex method was included among "the ten algorithms with the greatest influence on the development and practice of science and engineering in the 20th century."[3] Although some may argue that the simplex method is only

[3] The remaining nine algorithms on this list are the Metropolis algorithm for Monte Carlo simulations, the Krylov subspace iteration methods, the decompositional approach to matrix computations, the Fortran optimizing compiler, the QR algorithm for computing eigenvalues, the Quicksort algorithm for sorting, the fast Fourier transform, the detection of integer relations, and the fast multipole method.

number fourteen, say, and although each such evaluation is necessarily subjective, the importance of linear programming can hardly be cast in doubt.

The simplex method was invented and developed by George Dantzig in 1947, based on his work for the U.S. Air Force. Even earlier, in 1939, Leonid Vitalyevich Kantorovich was charged with the reorganization of the timber industry in the U.S.S.R., and as a part of his task he formulated a restricted class of linear programs and a method for their solution. As happens under such regimes, his discoveries went almost unnoticed and nobody continued his work. Kantorovich together with Tjalling Koopmans received the Nobel Prize in Economics in 1975, for pioneering work in resource allocation. Somewhat ironically, Dantzig, whose contribution to linear programming is no doubt much more significant, was never awarded a Nobel Prize.

The discovery of the simplex method had a great impact on both theory and practice in economics. Linear programming was used to allocate resources, plan production, schedule workers, plan investment portfolios, and formulate marketing and military strategies. Even entrepreneurs and managers accustomed to relying on their experience and intuition were impressed when costs were cut by 20%, say, by a mere reorganization according to some mysterious calculation. Especially when such a feat was accomplished by someone who was not really familiar with the company, just on the basis of some numerical data. Suddenly, mathematical methods could no longer be ignored with impunity in a competitive environment.

Linear programming has evolved a great deal since the 1940s, and new types of applications have been found, by far not restricted to mathematical economics.

In theoretical computer science it has become one of the fundamental tools in algorithm design. For a number of computational problems the existence of an efficient (polynomial-time) algorithm was first established by general techniques based on linear programming.

For other problems, known to be computationally difficult (NP-hard, if this term tells the reader anything), finding an exact solution is often hopeless. One looks for approximate algorithms, and linear programming is a key component of the most powerful known methods.

Another surprising application of linear programming is theoretical: the *duality theorem*, which will be explained in Chapter 6, appears in proofs of numerous mathematical statements, most notably in combinatorics, and it provides a unifying abstract view of many seemingly unrelated results. The duality theorem is also significant algorithmically.

We will show examples of methods for constructing algorithms and proofs based on linear programming, but many other results of this kind are too advanced for a short introductory text like ours.

The theory of algorithms for linear programming itself has also grown considerably. As everybody knows, today's computers are many orders of magnitude faster than those of fifty years ago, and so it doesn't sound surprising

that much larger linear programs can be solved today. But it may be surprising that this enlargement of manageable problems probably owes more to theoretical progress in algorithms than to faster computers. On the one hand, the implementation of the simplex method has been refined considerably, and on the other hand, new computational methods based on completely different ideas have been developed. This latter development will be described in Chapter 7.

2. Examples

Linear programming is a wonderful tool. But in order to use it, one first has to start suspecting that the considered computational problem might be expressible by a linear program, and then one has to really express it that way. In other words, one has to see linear programming "behind the scenes."

One of the main goals of this book is to help the reader acquire skills in this direction. We believe that this is best done by studying diverse examples and by practice. In this chapter we present several basic cases from the wide spectrum of problems amenable to linear programming methods, and we demonstrate a few tricks for reformulating problems that do not look like linear programs at first sight. Further examples are covered in Chapter 3, and Chapter 8 includes more advanced applications.

Once we have a suitable linear programming formulation (a "model" in the mathematical programming parlance), we can employ general algorithms. From a programmer's point of view this is very convenient, since it suffices to input the appropriate objective function and constraints into general-purpose software.

If efficiency is a concern, this need not be the end of the story. Many problems have special features, and sometimes specialized algorithms are known, or can be constructed, that solve such problems substantially faster than a general approach based on linear programming. For example, the study of network flows, which we consider in Section 2.2, constitutes an extensive subfield of theoretical computer science, and fairly efficient algorithms have been developed. Computing a maximum flow via linear programming is thus not the best approach for large-scale instances.

However, even for problems where linear programming doesn't ultimately yield the most efficient available algorithm, starting with a linear programming formulation makes sense: for fast prototyping, case studies, and deciding whether developing problem-specific software is worth the effort.

2.1 Optimized Diet: Wholesome and Cheap?

... and when Rabbit said, "Honey or condensed milk with your bread?" he was so excited that he said, "Both," and then, so as not to seem greedy, he added, "But don't bother about the bread, please."

A. A. Milne, Winnie the Pooh

The Office of Nutrition Inspection of the EU recently found out that dishes served at the dining and beverage facility "Bullneck's," such as herring, hot dogs, and house-style hamburgers do not comport with the new nutritional regulations, and its report mentioned explicitly the lack of vitamins A and C and dietary fiber. The owner and operator of the aforementioned facility is attempting to rectify these shortcomings by augmenting the menu with vegetable side dishes, which he intends to create from white cabbage, carrots, and a stockpile of pickled cucumbers discovered in the cellar. The following table summarizes the numerical data: the prescribed amount of the vitamins and fiber per dish, their content in the foods, and the unit prices of the foods.[1]

Food	Carrot, Raw	White Cabbage, Raw	Cucumber, Pickled	Required per dish
Vitamin A [mg/kg]	35	0.5	0.5	0.5 mg
Vitamin C [mg/kg]	60	300	10	15 mg
Dietary Fiber [g/kg]	30	20	10	4 g
price [€/kg]	0.75	0.5	0.15*	—

*Residual accounting price of the inventory, most likely unsaleable.

At what minimum additional price per dish can the requirements of the Office of Nutrition Inspection be satisfied? This question can be expressed by the following linear program:

$$\begin{array}{ll} \text{Minimize} & 0.75x_1 + 0.5x_2 + 0.15x_3 \\ \text{subject to} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \\ & 35x_1 + 0.5x_2 + 0.5x_3 \geq 0.5 \\ & 60x_1 + 300x_2 + 10x_3 \geq 15 \\ & 30x_1 + 20x_2 + 10x_3 \geq 4. \end{array}$$

The variable x_1 specifies the amount of carrot (in kg) to be added to each dish, and similarly for x_2 (cabbage) and x_3 (cucumber). The objective function

[1] For those interested in healthy diet: The vitamin contents and other data are more or less realistic.

expresses the price of the combination. The amounts of carrot, cabbage, and cucumber are always nonnegative, which is captured by the conditions $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$ (if we didn't include them, an optimal solution might perhaps have the amount of carrot, say, negative, by which one would seemingly save money). Finally, the inequalities in the last three lines force the requirements on vitamins A and C and of dietary fiber.

The linear program can be solved by standard methods. The optimal solution yields the price of €0.07 with the following doses: carrot 9.5 g, cabbage 38 g, and pickled cucumber 290 g per dish (all rounded to two significant digits). This probably wouldn't pass another round of inspection. In reality one would have to add further constraints, for example, one on the maximum amount of pickled cucumber.

We have included this example so that our treatment doesn't look too revolutionary. It seems that all introductions to linear programming begin with various dietary problems, most likely because the first large-scale problem on which the simplex method was tested in 1947 was the determination of an adequate diet of least cost. Which foods should be combined and in what amounts so that the required amounts of all essential nutrients are satisfied and the daily ration is the cheapest possible. The linear program had 77 variables and 9 constraints, and its solution by the simplex method using hand-operated desk calculators took approximately 120 man-days.

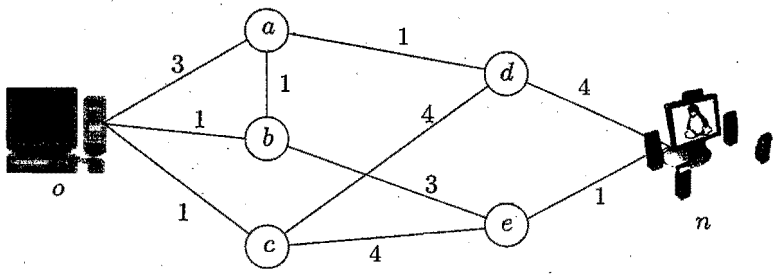
Later on, when George Dantzig had already gained access to an electronic computer, he tried to optimize his own diet as well. The optimal solution of the first linear program that he constructed recommended daily consumption of several liters of vinegar. When he removed vinegar from the next input, he obtained approximately 200 bouillon cubes as the basis of the daily diet. This story, whose truth is not entirely out of the question, doesn't diminish the power of linear programming in any way, but it illustrates how difficult it is to capture mathematically all the important aspects of real-life problems. In the realm of nutrition, for example, it is not clear even today what exactly the influence of various components of food is on the human body. (Although, of course, many things *are* clear, and hopes that the science of the future will recommend hamburgers as the main ingredient of a healthy diet will almost surely be disappointed.) Even if it were known perfectly, few people want and can formulate exactly what they expect from their diet—apparently, it is much easier to formulate such requirements for the diet of someone else. Moreover, there are nonlinear dependencies among the effects of various nutrients, and so the dietary problem can never be captured perfectly by linear programming.

There are many applications of linear programming in industry, agriculture, services, etc. that from an abstract point of view are variations of the diet problem and do not introduce substantially new mathematical tricks. It may still be challenging to design good models for real-life problems of this kind, but the challenges are not mathematical. We will not dwell on

such problems here (many examples can be found in Chvátal's book cited in Chapter 9), and we will present problems in which the use of linear programming has different flavors.

2.2 Flow in a Network

An administrator of a computer network convinced his employer to purchase a new computer with an improved sound system. He wants to transfer his music collection from an old computer to the new one, using a local network. The network looks like this:



What is the maximum transfer rate from computer o (old) to computer n (new)? The numbers near each data link specify the maximum transfer rate of that link (in Mbit/s, say). We assume that each link can transfer data in either direction, but not in both directions simultaneously. So, for example, through the link ab one can *either* send data from a to b at any rate from 0 up to 1 Mbit/s, *or* send data from b to a at any rate from 0 to 1 Mbit/s.

The nodes $a, b, \ldots, e$ are not suitable for storing substantial amounts of data, and hence all data entering them has to be sent further immediately. From this we can already see that the maximum transfer rate cannot be used on all links simultaneously (consider node a, for example). Thus we have to find an appropriate value of the data flow for each link so that the total transfer rate from o to n is maximum.

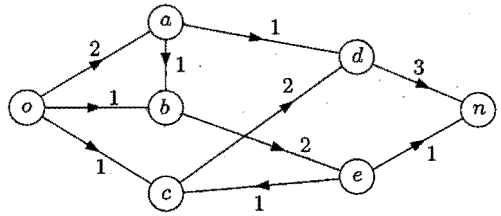
For every link in the network we introduce one variable. For example, x_{be} specifies the rate by which data is transfered from b to e. Here x_{be} can also be negative, which means that data flow in the opposite direction, from e to b. (And we thus do *not* introduce another variable x_{eb}, which would correspond to the transfer rate from e to b.) There are 10 variables: x_{oa}, x_{ob}, x_{oc}, x_{ab}, x_{ad}, x_{be}, x_{cd}, x_{ce}, x_{dn}, and x_{en}.

We set up the following linear program:

$$
\begin{array}{ll}
\text{Maximize} & x_{oa} + x_{ob} + x_{oc} \\
\text{subject to} & -3 \le x_{oa} \le 3, \quad -1 \le x_{ob} \le 1, \quad -1 \le x_{oc} \le 1 \\
& -1 \le x_{ab} \le 1, \quad -1 \le x_{ad} \le 1, \quad -3 \le x_{be} \le 3 \\
& -4 \le x_{cd} \le 4, \quad -4 \le x_{ce} \le 4, \quad -4 \le x_{dn} \le 4 \\
& -1 \le x_{en} \le 1 \\
& \qquad x_{oa} = x_{ab} + x_{ad} \\
& x_{ob} + x_{ab} = x_{be} \\
& \qquad x_{oc} = x_{cd} + x_{ce} \\
& x_{ad} + x_{cd} = x_{dn} \\
& x_{be} + x_{ce} = x_{en}.
\end{array}
$$

The objective function $x_{oa} + x_{ob} + x_{oc}$ expresses the total rate by which data is sent out from computer o. Since it is neither stored nor lost (hopefully) anywhere, it has to be received at n at the same rate. The next 10 constraints, $-3 \le x_{oa} \le 3$ through $-1 \le x_{en} \le 1$, restrict the transfer rates along the individual links. The remaining constraints say that whatever enters each of the nodes a through e has to leave immediately.

The optimal solution of this linear program is depicted below:



The number near each link is the transfer rate on that link, and the arrow determines the direction of the data flow. Note that between c and e data has to be sent in the direction from e to c, and hence $x_{ce} = -1$. The optimum value of the objective function is 4, and this is the desired maximum transfer rate.

In this example it is easy to see that the transfer rate cannot be larger, since the total capacity of all links connecting the computers o and a to the rest of the network equals 4. This is a special case of a remarkable theorem on maximum flow and minimum cut, which is usually discussed in courses on graph algorithms (see also Section 8.2).

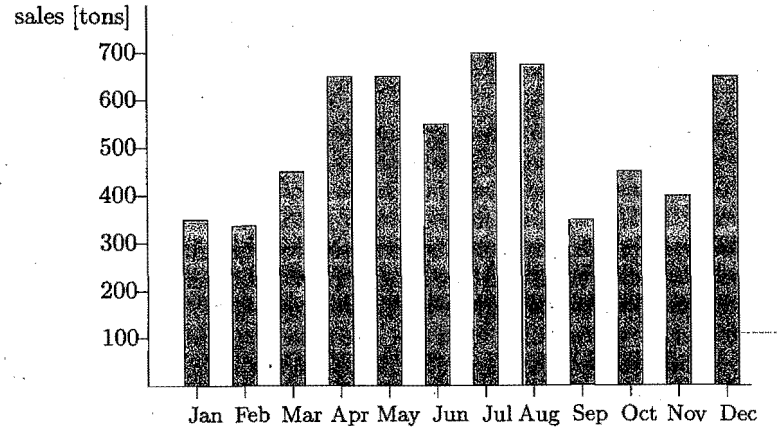
Our example of data flow in a network is small and simple. In practice, however, flows are considered in intricate networks, sometimes even with many source nodes and sink nodes. These can be electrical networks (current flows), road or railroad networks (cars or trains flow), telephone networks (voice or data signals flow), financial (money flows), and so on. There are also many less-obvious applications of network flows—for example, in image processing.

Historically, the network flow problem was first formulated by American military experts in search of efficient ways of disrupting the railway system of the Soviet block; see

A. Schrijver: On the history of the transportation and maximum flow problems, *Math. Programming Ser. B* 91(2002) 437–445.

2.3 Ice Cream All Year Round

The next application of linear programming again concerns food (which should not be surprising, given the importance of food in life and the difficulties in optimizing sleep or love). The ice cream manufacturer Icicle Works Ltd.[2] needs to set up a production plan for the next year. Based on history, extensive surveys, and bird observations, the marketing department has come up with the following prediction of monthly sales of ice cream in the next year:



Now Icicle Works Ltd. needs to set up a production schedule to meet these demands.

A simple solution would be to produce "just in time," meaning that all the ice cream needed in month i is also produced in month i, $i = 1, 2, \ldots, 12$. However, this means that the produced amount would vary greatly from month to month, and a change in the produced amount has significant costs: Temporary workers have to be hired or laid off, machines have to be adjusted,

[2] Not to be confused with a rock group of the same name. The name comes from a science fiction story by Frederik Pohl.

and so on. So it would be better to spread the production more evenly over the year: In months with low demand, the idle capacities of the factory could be used to build up a stock of ice cream for the months with high demand.

So another simple solution might be a completely "flat" production schedule, with the same amount produced every month. Some thought reveals that such a schedule need not be feasible if we want to end up with zero surplus at the end of the year. But even if it is feasible, it need not be ideal either, since storing ice cream incurs a nontrivial cost. It seems likely that the best production schedule should be somewhere between these two extremes (production following demand and constant production). We want a compromise minimizing the total cost resulting both from changes in production and from storage of surpluses.

To formalize this problem, let us denote the demand in month i by $d_i \geq 0$ (in tons). Then we introduce a nonnegative variable x_i for the production in month i and another nonnegative variable s_i for the total surplus in store at the end of month i. To meet the demand in month i, we may use the production in month i and the surplus at the end of month $i-1$:

$$x_i + s_{i-1} \geq d_i \quad \text{for } i = 1, 2, \ldots, 12.$$

The quantity $x_i + s_{i-1} - d_i$ is exactly the surplus after month i, and thus we have

$$x_i + s_{i-1} - s_i = d_i \quad \text{for } i = 1, 2, \ldots, 12.$$

Assuming that initially there is no surplus, we set $s_0 = 0$ (if we took the production history into account, s_0 would be the surplus at the end of the previous year). We also set $s_{12} = 0$, unless we want to plan for another year.

Among all nonnegative solutions to these equations, we are looking for one that minimizes the total cost. Let us assume that changing the production by 1 ton from month $i-1$ to month i costs €50, and that storage facilities for 1 ton of ice cream cost €20 per month. Then the total cost is expressed by the function

$$50 \sum_{i=1}^{12} |x_i - x_{i-1}| + 20 \sum_{i=1}^{12} s_i,$$

where we set $x_0 = 0$ (again, history can easily be taken into account).

Unfortunately, this cost function is not linear. Fortunately, there is a simple but important trick that allows us to make it linear, at the price of introducing extra variables.

The change in production is either an increase or a decrease. Let us introduce a nonnegative variable y_i for the increase from month $i-1$ to month i, and a nonnegative variable z_i for the decrease. Then

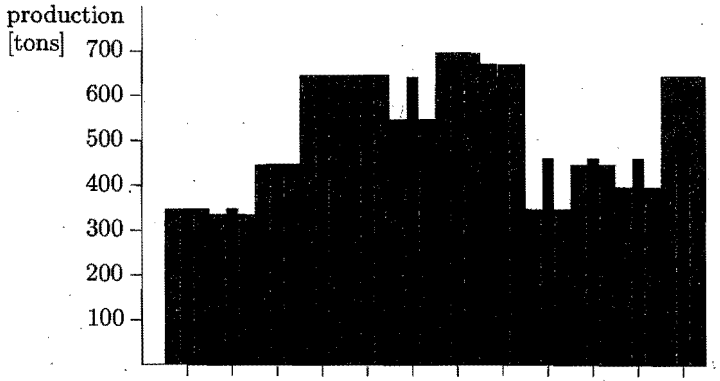
$$x_i - x_{i-1} = y_i - z_i \text{ and } |x_i - x_{i-1}| = y_i + z_i.$$

A production schedule of minimum total cost is given by an optimal solution of the following linear program:

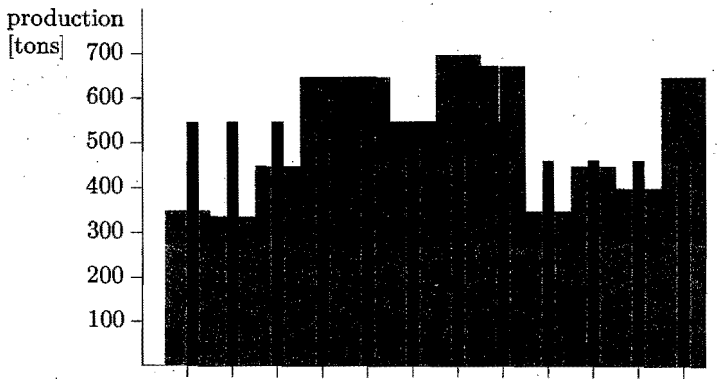
$$
\begin{array}{ll}
\text{Minimize} & 50\sum_{i=1}^{12} y_i + 50\sum_{i=1}^{12} z_i + 20\sum_{i=1}^{12} s_i \\
\text{subject to} & x_i + s_{i-1} - s_i = d_i \text{ for } i = 1, 2, \ldots, 12 \\
& x_i - x_{i-1} = y_i - z_i \text{ for } i = 1, 2, \ldots, 12 \\
& x_0 = 0 \\
& s_0 = 0 \\
& s_{12} = 0 \\
& x_i, s_i, y_i, z_i \geq 0 \text{ for } i = 1, 2, \ldots, 12.
\end{array}
$$

To see that an optimal solution $(\mathbf{s}^*, \mathbf{y}^*, \mathbf{z}^*)$ of this linear program indeed defines a schedule, we need to note that one of y_i^* and z_i^* has to be zero for all i, for otherwise, we could decrease both and obtain a better solution. This means that $y_i^* + z_i^*$ indeed equals the change in production from month $i-1$ to month i, as required.

In the Icicle Works example above, this linear program yields the following production schedule (shown with black bars; the gray background graph represents the demands).



Below is the schedule we would get with zero storage costs (that is, after replacing the "20" by "0" in the above linear program).

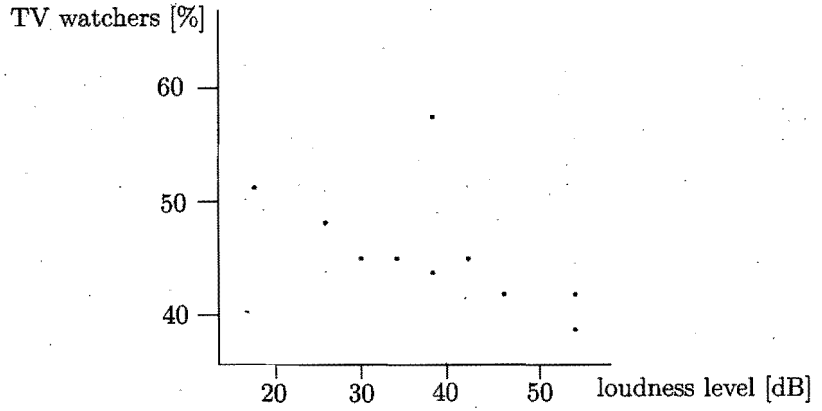


The pattern of this example is quite general, and many problems of optimal control can be solved via linear programming in a similar manner. A neat example is "Moon Rocket Landing," a once-popular game for programmable calculators (probably not sophisticated enough to survive in today's competition). A lunar module with limited fuel supply is descending vertically to the lunar surface under the influence of gravitation, and at chosen time intervals it can flash its rockets to slow down the descent (or even to start flying upward). The goal is to land on the surface with (almost) zero speed before exhausting all of the fuel. The reader is invited to formulate an appropriate linear program for determining the minimum amount of fuel necessary for landing, given the appropriate input data. For the linear programming formulation, we have to discretize time first (in the game this was done anyway), but with short enough time steps this doesn't make a difference in practice.

Let us remark that this particular problem can be solved analytically, with some calculus (or even mathematical control theory). But in even slightly more complicated situations, an analytic solution is out of reach.

2.4 Fitting a Line

The loudness level of nightingale singing was measured every evening for a number of days in a row, and the percentage of people watching the principal TV news was surveyed by questionnaires. The following diagram plots the measured values by points in the plane:



The simplest dependencies are linear, and many dependencies can be well approximated by a linear function. We thus want to find a line that best fits the measured points. (Readers feeling that this example is not sufficiently realistic can recall some measurements in physics labs, where the measured quantities should actually obey an exact linear dependence.)

How can one formulate mathematically that a given line "best fits" the points? There is no unique way, and several different criteria are commonly used for line fitting in practice.

The most popular one is the method of *least squares*, which for given points $(x_1, y_1), \ldots, (x_n, y_n)$ seeks a line with equation $y = ax + b$ minimizing the expression

$$\sum_{i=1}^{n} (ax_i + b - y_i)^2. \tag{2.1}$$

In words, for every point we take its vertical distance from the line, square it, and sum these "squares of errors."

This method need not always be the most suitable. For instance, if a few exceptional points are measured with very large error, they can influence the resulting line a great deal. An alternative method, less sensitive to a small number of "outliers," is to minimize the sum of absolute values of all errors:

$$\sum_{i=1}^{n} |ax_i + b - y_i|. \tag{2.2}$$

By a trick similar to the one we have seen in Section 2.3, this apparently nonlinear optimization problem can be captured by a linear program:

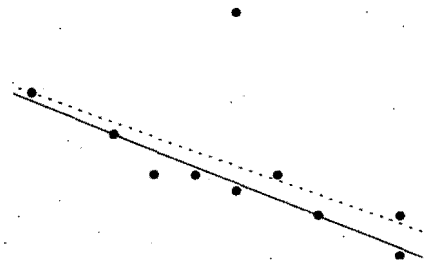
$$\begin{array}{lll} \text{Minimize} & e_1 + e_2 + \cdots + e_n & \\ \text{subject to} & e_i \geq ax_i + b - y_i & \text{for } i = 1, 2, \ldots, n \\ & e_i \geq -(ax_i + b - y_i) & \text{for } i = 1, 2, \ldots, n. \end{array}$$

The variables are a, b, and $e_1, e_2, \ldots, e_n$ (while $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are given numbers). Each e_i is an auxiliary variable standing for the error at the ith point. The constraints guarantee that

$$e_i \geq \max\Big(ax_i + b - y_i, -(ax_i + b - y_i)\Big) = |ax_i + b - y_i|.$$

In an optimal solution each of these inequalities has to be satisfied with equality, for otherwise, we could decrease the corresponding e_i. Thus, an optimal solution yields a line minimizing the expression (2.2).

The following picture shows a line fitted by this method (solid) and a line fitted using least squares (dotted):



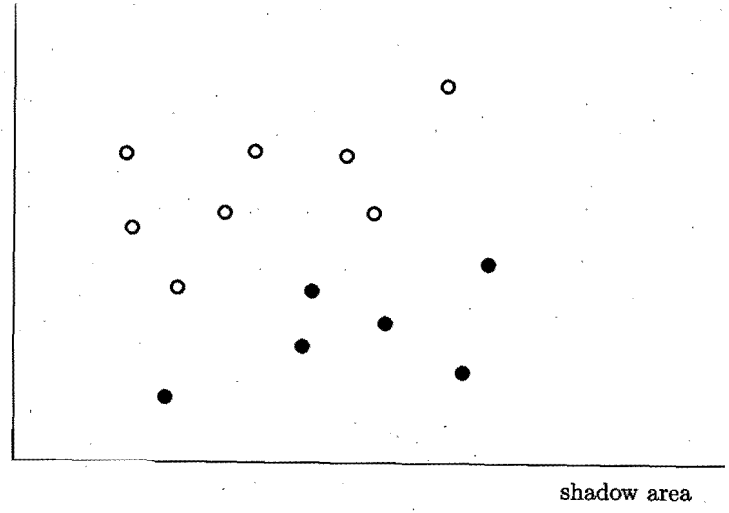
In conclusion, let us recall the useful trick we have learned here and in the previous section:

> Objective functions or constraints involving absolute values can often be handled via linear programming by introducing extra variables or extra constraints.

2.5 Separation of Points

A computer-controlled rabbit trap "Gromit RT 2.1" should be programmed so that it catches rabbits, but if a weasel wanders in, it is released. The trap can weigh the animal inside and also can determine the area of its shadow.

These two parameters were collected for a number of specimens of rabbits and weasels, as depicted in the following graph:



(empty circles represent rabbits and full circles weasels).

Apparently, neither weight alone nor shadow area alone can be used to tell a rabbit from a weasel. One of the next-simplest things would be a linear criterion distinguishing them. That is, geometrically, we would like to separate the black points from the white points by a straight line if possible. Mathematically speaking, we are given m white points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$

and n black points $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ in the plane, and we would like to find out whether there exists a line having all white points on one side and all black points on the other side (none of the points should lie on the line).

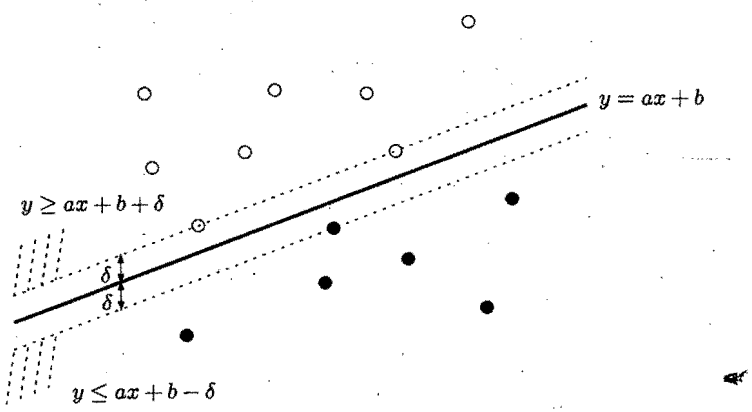
In a solution of this problem by linear programming we distinguish three cases. First we test whether there exists a *vertical* line with the required property. This case needs neither linear programming nor particular cleverness.

The next case is the existence of a line that is not vertical and that has all black points below it and all white points above it. Let us write the equation of such a line as $y = ax + b$, where a and b are some yet unknown real numbers. A point $\mathbf{r}$ with coordinates $x(\mathbf{r})$ and $y(\mathbf{r})$ lies above this line if $y(\mathbf{r}) > ax(\mathbf{r}) + b$, and it lies below it if $y(\mathbf{r}) < ax(\mathbf{r}) + b$. So a suitable line exists if and only if the following system of inequalities with variables a and b has a solution:

$$\begin{aligned} y(\mathbf{p}_i) &> ax(\mathbf{p}_i) + b \qquad \text{for } i = 1, 2, \ldots, m \\ y(\mathbf{q}_j) &< ax(\mathbf{q}_j) + b \qquad \text{for } j = 1, 2, \ldots, n. \end{aligned}$$

We haven't yet mentioned strict inequalities in connection with linear programming, and actually, they are not allowed in linear programs. But here we can get around this issue by a small trick: We introduce a new variable δ, which stands for the "gap" between the left and right sides of each strict inequality. Then we try to make the gap as large as possible:

$$\begin{array}{lll} \text{Maximize} & \delta & \\ \text{subject to} & y(\mathbf{p}_i) \geq ax(\mathbf{p}_i) + b + \delta & \text{for } i = 1, 2, \ldots, m \\ & y(\mathbf{q}_j) \leq ax(\mathbf{q}_j) + b - \delta & \text{for } j = 1, 2, \ldots, n. \end{array}$$



This linear program has three variables: a, b, and δ. The optimal δ is positive exactly if the preceding system of strict inequalities has a solution, and the latter happens exactly if a nonvertical line exists with all black points below and all white points above.

Similarly, we can deal with the third case, namely the existence of a nonvertical line having all black points above it and all white points below it. This completes the description of an algorithm for the line separation problem.

A plane separating two point sets in $\mathbb{R}^3$ can be computed by the same approach, and we can also solve the analogous problem in higher dimensions. So we could try to distinguish rabbits from weasels based on more than two measured parameters.

Here is another, perhaps more surprising, extension. Let us imagine that separating rabbits from weasels by a straight line proved impossible. Then we could try, for instance, separating them by a graph of a quadratic function (a parabola), of the form $ax^2 + bx + c$. So given m white points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ and n black points $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ in the plane, we now ask, are there coefficients $a, b, c \in \mathbb{R}$ such that the graph of $f(x) = ax^2 + bx + c$ has all white points above it and all black points below? This leads to the inequality system

$$\begin{aligned} y(\mathbf{p}_i) &> ax(\mathbf{p}_i)^2 + bx(\mathbf{p}_i) + c \qquad \text{for } i = 1, 2, \ldots, m \\ y(\mathbf{q}_j) &< ax(\mathbf{q}_j)^2 + bx(\mathbf{q}_j) + c \qquad \text{for } j = 1, 2, \ldots, n. \end{aligned}$$

By introducing a gap variable δ as before, this can be written as the following linear program in the variables a, b, c, and δ:

$$\begin{array}{lll} \text{Maximize} & \delta & \\ \text{subject to} & y(\mathbf{p}_i) \geq ax(\mathbf{p}_i)^2 + bx(\mathbf{p}_i) + c + \delta & \text{for } i = 1, 2, \ldots, m \\ & y(\mathbf{q}_j) \leq ax(\mathbf{q}_j)^2 + bx(\mathbf{q}_j) + c - \delta & \text{for } j = 1, 2, \ldots, n. \end{array}$$

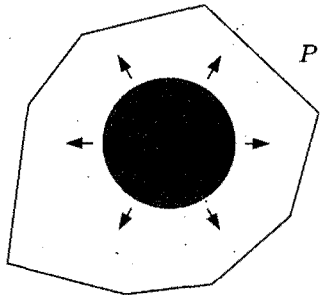
In this linear program the quadratic terms are coefficients and therefore they cause no harm.

The same approach also allows us to test whether two point sets in the plane, or in higher dimensions, can be separated by a function of the form $f(\mathbf{x}) = a_1\varphi_1(\mathbf{x}) + a_2\varphi_2(\mathbf{x}) + \cdots + a_k\varphi_k(\mathbf{x})$, where $\varphi_1, \ldots, \varphi_k$ are given functions (possibly nonlinear) and $a_1, a_2, \ldots, a_k$ are real coefficients, in the sense that $f(\mathbf{p}_i) > 0$ for every white point $\mathbf{p}_i$ and $f(\mathbf{q}_j) < 0$ for every black point $\mathbf{q}_j$.

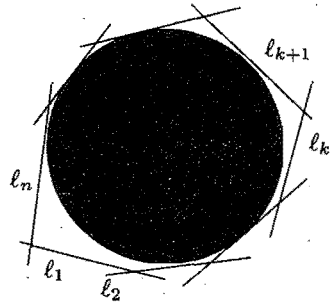
2.6 Largest Disk in a Convex Polygon

Here we will encounter another problem that may look nonlinear at first sight but can be transformed to a linear program. It is a simple instance of a geometric *packing problem:* Given a container, in our case a convex polygon, we want to fit as large an object as possible into it, in our case a disk of the largest possible radius.

Let us call the given convex polygon P, and let us assume that it has n sides. As we said, we want to find the largest circular disk contained in P.



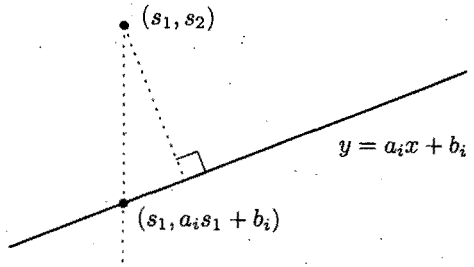
For simplicity let us assume that none of the sides of P is vertical. Let the ith side of P lie on a line ℓ_i with equation $y = a_i x + b_i$, $i = 1, 2, \ldots, n$, and let us choose the numbering of the sides in such a way that the first, second, up to the kth side bound P from below, while the $(k+1)$st through nth side bound it from above.



Let us now ask, under what conditions does a circle with center $\mathbf{s} = (s_1, s_2)$ and radius r lie completely inside P? This is the case if and only if the point $\mathbf{s}$ has distance at least r from each of the lines $\ell_1, \ldots, \ell_n$, lies above the lines $\ell_1, \ldots, \ell_k$, and lies below the lines $\ell_{k+1}, \ldots, \ell_n$. We compute the distance of $\mathbf{s}$ from ℓ_i. A simple calculation using similarity of triangles and the Pythagorean theorem shows that this distance equals the absolute value of the expression

$$\frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}}.$$

Moreover, the expression is positive if $\mathbf{s}$ lies above ℓ_i, and it is negative if $\mathbf{s}$ lies below ℓ_i:



The disk of radius r centered at $\mathbf{s}$ thus lies inside P exactly if the following system of inequalities is satisfied:

$$\frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} \geq r, \qquad i = 1, 2, \ldots, k$$
$$\frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} \leq -r, \qquad i = k+1, k+2, \ldots, n.$$

Therefore, we want to find the largest r such that there exist s_1 and s_2 so that all the constraints are satisfied. This yields a linear program! (Some might be frightened by the square roots, but these can be computed in advance, since all the a_i are concrete numbers.)

$$\begin{aligned} &\text{Maximize} \quad r \\ &\text{subject to} \quad \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} \geq r \quad \text{for } i = 1, 2, \ldots, k \\ &\qquad\qquad \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} \leq -r \quad \text{for } i = k+1, k+2, \ldots, n. \end{aligned}$$

There are three variables: s_1, s_2, and r. An optimal solution yields the desired largest disk contained in P.

A similar problem in higher dimension can be solved analogously. For example, in three-dimensional space we can ask for the largest ball that can be placed into the intersection of n given half-spaces.

Interestingly, another similar-looking problem, namely, finding the smallest disk containing a given convex n-gon in the plane, cannot be expressed by a linear program and has to be solved differently; see Section 8.7.

Both in practice and in theory, one usually encounters geometric packing problems that are more complicated than the one considered in this section and not so easily solved by linear programming. Often we have a fixed collection of objects and we want to pack as many of them as possible into a given container (or several containers). Such problems are encountered by confectioners when cutting cookies from a piece of dough, by tailors or clothing

manufacturers when making as many trousers, say, as possible from a large piece of cloth, and so on. Typically, these problems are computationally hard, but linear programming can sometimes help in devising heuristics or approximate algorithms.

2.7 Cutting Paper Rolls

Here we have another industrial problem, and the application of linear programming is quite nonobvious. Moreover, we will naturally encounter an integrality constraint, which will bring us to the topic of the next chapter.

A paper mill manufactures rolls of paper of a standard width 3 meters. But customers want to buy paper rolls of shorter width, and the mill has to cut such rolls from the 3 m rolls. One 3 m roll can be cut, for instance, into two rolls 93 cm wide, one roll of width 108 cm, and a rest of 6 cm (which goes to waste).

Let us consider an order of

- 97 rolls of width 135 cm,
- 610 rolls of width 108 cm,
- 395 rolls of width 93 cm, and
- 211 rolls of width 42 cm.

What is the smallest number of 3 m rolls that have to be cut in order to satisfy this order, and how should they be cut?

In order to engage linear programming one has to be generous in introducing variables. We write down all of the requested widths: 135 cm, 108 cm, 93 cm, and 42 cm. Then we list all possibilities of cutting a 3 m paper roll into rolls of some of these widths (we need to consider only possibilities for which the wasted piece is shorter than 42 cm):

P1:	2×135	P7:	$108 + 93 + 2 \times 42$
P2:	$135 + 108 + 42$	P8:	$108 + 4 \times 42$
P3:	$135 + 93 + 42$	P9:	3×93
P4:	$135 + 3 \times 42$	P10:	$2 \times 93 + 2 \times 42$
P5:	$2 \times 108 + 2 \times 42$	P11:	$93 + 4 \times 42$
P6:	$108 + 2 \times 93$	P12:	7×42

For each possibility Pj on the list we introduce a variable $x_j \geq 0$ representing the number of rolls cut according to that possibility. We want to minimize the total number of rolls cut, i.e., $\sum_{j=1}^{12} x_j$, in such a way that the customers are satisfied. For example, to satisfy the demand for 395 rolls of width 93 cm we require

$$x_3 + 2x_6 + x_7 + 3x_9 + 2x_{10} + x_{11} \geq 395.$$

For each of the widths we obtain one constraint.

For a more complicated order, the list of possibilities would most likely be produced by computer. We would be in a quite typical situation in which a linear program is not entered "by hand," but rather is generated by some computer program. More-advanced techniques even generate the possibilities "on the fly," during the solution of the linear program, which may save time and memory considerably. See the entry "column generation" in the glossary or Chvátal's book cited in Chapter 9, from which this example is taken.

The optimal solution of the resulting linear program has $x_1 = 48.5$, $x_5 = 206.25$, $x_6 = 197.5$, and all other components 0. In order to cut 48.5 rolls according to the possibility P1, one has to unwind half of a roll. Here we need more information about the technical possibilities of the paper mill: Is cutting a fraction of a roll technically and economically feasible? If yes, we have solved the problem optimally. If not, we have to work further and somehow take into account the restriction that only feasible solutions of the linear program with *integral* x_i are of interest. This is not at all easy in general, and it is the subject of Chapter 3.