Linear Programming: Foundations and Extensions

Robert J. Vanderbei

DEPARTMENT OF OPERATIONS RESEARCH AND FINANCIAL ENGINEERING, PRINCETON UNIVERSITY, PRINCETON, NJ 08544
E-mail address: rvdb@princeton.edu

LINEAR PROGRAMMING: FOUNDATIONS AND EXTENSIONS
Second Edition

ISBN 0-0000-0000-0

The text for this book was formated in Times-Roman and the mathematics was formated in Michael Spivak's Mathtimes using $\mathcal{AMS}$-LaTeX(which is a macro package for Leslie Lamport's LaTeX, which itself is a macro package for Donald Knuth's TeXtext formatting system) and converted from device-independent to postscript format using DVIPS. The figures were produced using SHOWCASE on a Silicon Graphics, Inc. workstation and were incorporated into the text as encapsulated postscript files with the macro package called PSFIG.TEX.

To my parents, Howard and Marilyn,

my dear wife, Krisadee,

and the babes, Marisa and Diana

Contents

Preface

This book is about constrained optimization. It begins with a thorough treatment of linear programming and proceeds to convex analysis, network flows, integer programming, quadratic programming, and convex optimization. Along the way, dynamic programming and the linear complementarity problem are touched on as well.

The book aims to be a first introduction to the subject. Specific examples and concrete algorithms precede more abstract topics. Nevertheless, topics covered are developed in some depth, a large number of numerical examples are worked out in detail, and many recent topics are included, most notably interior-point methods. The exercises at the end of each chapter both illustrate the theory and, in some cases, extend it.

Prerequisites. The book is divided into four parts. The first two parts assume a background only in linear algebra. For the last two parts, some knowledge of multivariate calculus is necessary. In particular, the student should know how to use Lagrange multipliers to solve simple calculus problems in 2 and 3 dimensions.

Associated software. It is good to be able to solve small problems by hand, but the problems one encounters in practice are large, requiring a computer for their solution. Therefore, to fully appreciate the subject, one needs to solve large (practical) problems on a computer. An important feature of this book is that it comes with software implementing the major algorithms described herein. At the time of writing, software for the following five algorithms is available:

- The two-phase simplex method as shown in Figure 6.1.
- The self-dual simplex method as shown in Figure 7.1.
- The path-following method as shown in Figure 17.1.
- The homogeneous self-dual method as shown in Figure 21.1.
- The long-step homogeneous self-dual method as described in Exercise 21.4.

The programs that implement these algorithms are written in C and can be easily compiled on most hardware platforms. Students/instructors are encouraged to install and compile these programs on their local hardware. Great pains have been taken to make the source code for these programs readable (see Appendix A). In particular, the names of the variables in the programs are consistent with the notation of this book.

There are two ways to run these programs. The first is to prepare the input in a standard computer-file format, called MPS format, and to run the program using such

a file as input. The advantage of this input format is that there is an archive of problems stored in this format, called the NETLIB suite, that one can download and use immediately (a link to the NETLIB suite can be found at the web site mentioned below). But, this format is somewhat archaic and, in particular, it is not easy to create these files by hand. Therefore, the programs can also be run from within a problem modeling system called AMPL. AMPL allows one to describe mathematical programming problems using an easy to read, yet concise, algebraic notation. To run the programs within AMPL, one simply tells AMPL the name of the solver-program before asking that a problem be solved. The text that describes AMPL, (Fourer et al. 1993), makes an excellent companion to this book. It includes a discussion of many practical linear programming problems. It also has lots of exercises to hone the modeling skills of the student.

Several interesting computer projects can be suggested. Here are a few suggestions regarding the simplex codes:

- Incorporate the partial pricing strategy (see Section 8.7) into the two-phase simplex method and compare it with full pricing.
- Incorporate the steepest-edge pivot rule (see Section 8.8) into the two-phase simplex method and compare it with the largest-coefficient rule.
- Modify the code for either variant of the simplex method so that it can treat bounds and ranges implicitly (see Chapter 9), and compare the performance with the explicit treatment of the supplied codes.
- Implement a "warm-start" capability so that the sensitivity analyses discussed in Chapter 7 can be done.
- Extend the simplex codes to be able to handle integer programming problems using the branch-and-bound method described in Chapter 22.

As for the interior-point codes, one could try some of the following projects:

- Modify the code for the path-following algorithm so that it implements the affine-scaling method (see Chapter 20), and then compare the two methods.
- Modify the code for the path-following method so that it can treat bounds and ranges implicitly (see Section 19.3), and compare the performance against the explicit treatment in the given code.
- Modify the code for the path-following method to implement the higher-order method described in Exercise 17.5. Compare.
- Extend the path-following code to solve quadratic programming problems using the algorithm shown in Figure 23.3.
- Further extend the code so that it can solve convex optimization problems using the algorithm shown in Figure 24.2.

And, perhaps the most interesting project of all:

- Compare the simplex codes against the interior-point code and decide for yourself which algorithm is better on specific families of problems.

The software implementing the various algorithms was developed using consistent data structures and so making fair comparisons should be straightforward. The software can be downloaded from the following web site:

http://www.princeton.edu/∼rvdb/LPbook/

If, in the future, further codes relating to this text are developed (for example, a self-dual network simplex code), they will be made available through this web site.

Features. Here are some other features that distinguish this book from others:

- The development of the simplex method leads to Dantzig's parametric self-dual method. A randomized variant of this method is shown to be immune to the travails of degeneracy.
- The book gives a balanced treatment to both the traditional simplex method and the newer interior-point methods. The notation and analysis is developed to be consistent across the methods. As a result, the self-dual simplex method emerges as the variant of the simplex method with most connections to interior-point methods.
- From the beginning and consistently throughout the book, linear programming problems are formulated in *symmetric form*. By highlighting symmetry throughout, it is hoped that the reader will more fully understand and appreciate duality theory.
- By slightly changing the right-hand side in the Klee–Minty problem, we are able to write down an explicit dictionary for each vertex of the Klee–Minty problem and thereby uncover (as a homework problem) a simple, elegant argument why the Klee-Minty problem requires $2^n - 1$ pivots to solve.
- The chapter on regression includes an analysis of the expected number of pivots required by the self-dual variant of the simplex method. This analysis is supported by an empirical study.
- There is an extensive treatment of modern interior-point methods, including the primal–dual method, the affine-scaling method, and the self-dual path-following method.
- In addition to the traditional applications, which come mostly from business and economics, the book features other important applications such as the optimal design of truss-like structures and L^1-regression.

Exercises on the Web. There is always a need for fresh exercises. Hence, I have created and plan to maintain a growing archive of exercises specifically created for use in conjunction with this book. This archive is accessible from the book's web site:

http://www.princeton.edu/∼rvdb/LPbook/

The problems in the archive are arranged according to the chapters of this book and use notation consistent with that developed herein.

Advice on solving the exercises. Some problems are routine while others are fairly challenging. Answers to some of the problems are given at the back of the book. In

general, the advice given to me by Leonard Gross (when I was a student) should help even on the hard problems: *follow your nose*.

Audience. This book evolved from lecture notes developed for my introductory graduate course in linear programming as well as my upper-level undergraduate course. A reasonable undergraduate syllabus would cover essentially all of Part 1 (Simplex Method and Duality), the first two chapters of Part 2 (Network Flows and Applications), and the first chapter of Part 4 (Integer Programming). At the graduate level, the syllabus should depend on the preparation of the students. For a well-prepared class, one could cover the material in Parts 1 and 2 fairly quickly and then spend more time on Parts 3 (Interior-Point Methods) and 4 (Extensions).

Dependencies. In general, Parts 2 and 3 are completely independent of each other. Both depend, however, on the material in Part 1. The first Chapter in Part 4 (Integer Programming) depends only on material from Part 1, whereas the remaining chapters build on Part 3 material.

Acknowledgments. My interest in linear programming was sparked by Robert Garfinkel when we shared an office at Bell Labs. I would like to thank him for his constant encouragement, advice, and support. This book benefited greatly from the thoughtful comments and suggestions of David Bernstein and Michael Todd. I would also like to thank the following colleagues for their help: Ronny Ben-Tal, Leslie Hall, Yoshi Ikura, Victor Klee, Irvin Lustig, Avi Mandelbaum, Marc Meketon, Narcis Nabona, James Orlin, Andrzej Ruszczynski, and Henry Wolkowicz. I would like to thank Gary Folven at Kluwer and Fred Hillier, the series editor, for encouraging me to undertake this project. I would like to thank my students for finding many typos and occasionally more serious errors: John Gilmartin, Jacinta Warnie, Stephen Woolbert, Lucia Wu, and Bing Yang My thanks to Erhan Çınlar for the many times he offered advice on questions of style. I hope this book reflects positively on his advice. Finally, I would like to acknowledge the support of the National Science Foundation and the Air Force Office of Scientific Research for supporting me while writing this book. In a time of declining resources, I am especially grateful for their support.

Robert J. Vanderbei

September, 1996

Preface to 2nd Edition

For the 2nd edition, many new exercises have been added. Also I have worked hard to develop online tools to aid in learning the simplex method and duality theory. These online tools can be found on the book's web page:

http://www.princeton.edu/∼rvdb/LPbook/

and are mentioned at appropriate places in the text. Besides the learning tools, I have created several online exercises. These exercises use randomly generated problems and therefore represent a virtually unlimited collection of "routine" exercises that can be used to test basic understanding. Pointers to these online exercises are included in the exercises sections at appropriate points.

Some other notable changes include:

- The chapter on network flows has been completely rewritten. Hopefully, the new version is an improvement on the original.
- Two different fonts are now used to distinguish between the set of basic indices and the basis matrix.
- The first edition placed great emphasis on the symmetry between the primal and the dual (the negative transpose property). The second edition carries this further with a discussion of the relationship between the basic and non-basic matrices B and N as they appear in the primal and in the dual. We show that, even though these matrices differ (they even have different dimensions), $B^{-1}N$ in the dual is the negative transpose of the corresponding matrix in the primal.
- In the chapters devoted to the simplex method in matrix notation, the collection of variables $z_1, z_2, \ldots, z_n, y_1, y_2, \ldots, y_m$ was replaced, in the first edition, with the single array of variables $y_1, y_2, \ldots, y_{n+m}$. This caused great confusion as the variable y_i in the original notation was changed to y_{n+i} in the new notation. For the second edition, I have changed the notation for the single array to $z_1, z_2, \ldots, z_{n+m}$.
- A number of figures have been added to the chapters on convex analysis and on network flow problems.

- The algorithm refered to as the primal–dual simplex method in the first edition has been renamed the parametric self-dual simplex method in accordance with prior standard usage.
- The last chapter, on convex optimization, has been extended with a discussion of merit functions and their use in shortenning steps to make some otherwise nonconvergent problems converge.

Acknowledgments. Many readers have sent corrections and suggestions for improvement. Many of the corrections were incorporated into earlier reprintings. Only those that affected pagination were accrued to this new edition. Even though I cannot now remember everyone who wrote, I am grateful to them all. Some sent comments that had significant impact. They were Hande Benson, Eric Denardo, Sudhakar Mandapati, Michael Overton, and Jos Sturm.

Robert J. Vanderbei

December, 2000

Part 1

Basic Theory—The Simplex Method and Duality

We all love to instruct, though we can teach only what is not worth knowing. — J. Austen

CHAPTER 1

Introduction

This book is mostly about a subject called Linear Programming. Before defining what we mean, in general, by a linear programming problem, let us describe a few practical real-world problems that serve to motivate and at least vaguely to define this subject.

1. Managing a Production Facility

Consider a production facility for a manufacturing company. The facility is capable of producing a variety of products that, for simplicity, we shall enumerate as $1, 2, \ldots, n$. These products are constructed/manufactured/produced out of certain raw materials. Let us assume that there are m different raw materials, which again we shall simply enumerate as $1, 2, \ldots, m$. The decisions involved in managing/operating this facility are complicated and arise dynamically as market conditions evolve around it. However, to describe a simple, fairly realistic optimization problem, we shall consider a particular snapshot of the dynamic evolution. At this specific point in time, the facility has, for each raw material $i = 1, 2, \ldots, m$, a known amount, say b_i, on hand. Furthermore, each raw material has at this moment in time a known unit market value. We shall denote the unit value of the ith raw material by ρ_i.

In addition, each product is made from known amounts of the various raw materials. That is, producing one unit of product j requires a certain known amount, say a_{ij} units, of raw material i. Also, the jth final product can be sold at the known prevailing market price of σ_j dollars per unit.

Throughout this section we make an important assumption:

> The production facility is small relative to the market as a whole and therefore cannot through its actions alter the prevailing market value of its raw materials, nor can it affect the prevailing market price for its products.

We shall consider two optimization problems related to the efficient operation of this facility (later, in Chapter 5, we shall see that these two problems are in fact closely related to each other).

1.1. Production Manager as Optimist. The first problem we wish to consider is the one faced by the company's production manager. It is the problem of how to use

the raw materials on hand. Let us assume that she decides to produce x_j units of the jth product, $j = 1, 2, \ldots, n$. The revenue associated with the production of one unit of product j is σ_j. But there is also a cost of raw materials that must be considered. The cost of producing one unit of product j is $\sum_{i=1}^{m} \rho_i a_{ij}$. Therefore, the net revenue associated with the production of one unit is the difference between the revenue and the cost. Since the net revenue plays an important role in our model, we introduce notation for it by setting

$$c_j = \sigma_j - \sum_{i=1}^{m} \rho_i a_{ij}, \qquad j = 1, 2, \ldots, n.$$

Now, the net revenue corresponding to the production of x_j units of product j is simply $c_j x_j$, and the total net revenue is

$$\sum_{j=1}^{n} c_j x_j. \tag{1.1}$$

The production planner's goal is to maximize this quantity. However, there are constraints on the production levels that she can assign. For example, each production quantity x_j must be nonnegative, and so she has the constraint

$$x_j \geq 0, \qquad j = 1, 2, \ldots, n. \tag{1.2}$$

Secondly, she can't produce more product than she has raw material for. The amount of raw material i consumed by a given production schedule is $\sum_{j=1}^{n} a_{ij} x_j$, and so she must adhere to the following constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, 2, \ldots, m. \tag{1.3}$$

To summarize, the production manager's job is to determine production values x_j, $j = 1, 2, \ldots, n$, so as to maximize (1.1) subject to the constraints given by (1.2) and (1.3). This optimization problem is an example of a linear programming problem. This particular example is often called the *resource allocation problem*.

1.2. Comptroller as Pessimist. In another office at the production facility sits an executive called the comptroller. The comptroller's problem (among others) is to assign a value to the raw materials on hand. These values are needed for accounting and planning purposes to determine the *cost of inventory*. There are rules about how these values can be set. The most important such rule (and the only one relevant to our discussion) is the following:

The company must be willing to sell the raw materials should an outside firm offer to buy them at a price consistent with these values.

Let w_i denote the assigned unit value of the ith raw material, $i = 1, 2, \ldots, m$. That is, these are the numbers that the comptroller must determine. The *lost opportunity cost* of having b_i units of raw material i on hand is $b_i w_i$, and so the total lost opportunity cost is

$$\sum_{i=1}^{m} b_i w_i. \tag{1.4}$$

The comptroller's goal is to minimize this lost opportunity cost (to make the financial statements look as good as possible). But again, there are constraints. First of all, each assigned unit value w_i must be no less than the prevailing unit market value ρ_i, since if it were less an outsider would buy the company's raw material at a price lower than ρ_i, contradicting the assumption that ρ_i is the prevailing market price. That is,

$$w_i \geq \rho_i, \qquad i = 1, 2, \ldots, m. \tag{1.5}$$

Similarly,

$$\sum_{i=1}^{m} w_i a_{ij} \geq \sigma_j, \qquad j = 1, 2, \ldots, n. \tag{1.6}$$

To see why, suppose that the opposite inequality holds for some specific product j. Then an outsider could buy raw materials from the company, produce product j, and sell it at a lower price than the prevailing market price. This contradicts the assumption that σ_j is the prevailing market price, which cannot be lowered by the actions of the company we are studying. Minimizing (1.4) subject to the constraints given by (1.5) and (1.6) is a linear programming problem. It takes on a slightly simpler form if we make a change of variables by letting

$$y_i = w_i - \rho_i, \qquad i = 1, 2, \ldots, m.$$

In words, y_i is the increase in the unit value of raw material i representing the "mark-up" the company would charge should it wish simply to act as a reseller and sell raw materials back to the market. In terms of these variables, the comptroller's problem is to minimize

$$\sum_{i=1}^{m} b_i y_i$$

subject to

$$\sum_{i=1}^{m} y_i a_{ij} \geq c_j, \qquad j = 1, 2, \ldots, n$$

and

$$y_i \geq 0, \qquad i = 1, 2, \ldots, m.$$

Note that we've dropped a term, $\sum_{i=1}^{m} b_i \rho_i$, from the objective. It is a constant (the market value of the raw materials), and so, while it affects the value of the function being minimized, it does not have any impact on the actual optimal values of the variables (whose determination is the comptroller's main interest).

2. The Linear Programming Problem

In the two examples given above, there have been variables whose values are to be decided in some optimal fashion. These variables are referred to as *decision variables*. They are usually written as

$$x_j, \qquad j = 1, 2, \ldots, n.$$

In linear programming, the objective is always to maximize or to minimize some linear function of these decision variables

$$\zeta = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n.$$

This function is called the *objective function*. It often seems that real-world problems are most naturally formulated as minimizations (since real-world planners always seem to be pessimists), but when discussing mathematics it is usually nicer to work with maximization problems. Of course, converting from one to the other is trivial both from the modeler's viewpoint (either minimize cost or maximize profit) and from the analyst's viewpoint (either maximize ζ or minimize $-\zeta$). Since this book is primarily about the mathematics of linear programming, we shall usually consider our aim one of maximizing the objective function.

In addition to the objective function, the examples also had constraints. Some of these constraints were really simple, such as the requirement that some decision variable be nonnegative. Others were more involved. But in all cases the constraints consisted of either an equality or an inequality associated with some linear combination of the decision variables:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b.$$

It is easy to convert constraints from one form to another. For example, an inequality constraint

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$$

can be converted to an equality constraint by adding a nonnegative variable, w, which we call a *slack variable*:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n + w = b, \qquad w \geq 0.$$

On the other hand, an equality constraint

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

can be converted to inequality form by introducing two inequality constraints:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$$
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b.$$

Hence, in some sense, there is no a priori preference for how one poses the constraints (as long as they are linear, of course). However, we shall also see that, from a mathematical point of view, there is a preferred presentation. It is to pose the inequalities as less-thans and to stipulate that all the decision variables be nonnegative. Hence, the linear programming problem as we shall study it can be formulated as follows:

$$\begin{array}{lrcl}
\text{maximize} & c_1x_1 + & c_2x_2 + \cdots + & c_nx_n \\
\text{subject to} & a_{11}x_1 + & a_{12}x_2 + \cdots + & a_{1n}x_n \leq b_1 \\
& a_{21}x_1 + & a_{22}x_2 + \cdots + & a_{2n}x_n \leq b_2 \\
& & \vdots & \\
& a_{m1}x_1 + & a_{m2}x_2 + \cdots + & a_{mn}x_n \leq b_m \\
& & x_1,\ x_2,\ \ldots\ x_n \geq & 0.
\end{array}$$

We shall refer to linear programs formulated this way as linear programs in *standard form*. We shall always use m to denote the number of constraints, and n to denote the number of decision variables.

A proposal of specific values for the decision variables is called a *solution*. A solution $(x_1, x_2, \ldots, x_n)$ is called *feasible* if it satisfies all of the constraints. It is called *optimal* if in addition it attains the desired maximum. Some problems are just

simply infeasible, as the following example illustrates:

$$\begin{array}{ll} \text{maximize} & 5x_1 + 4x_2 \\ \text{subject to} & x_1 + x_2 \leq 2 \\ & -2x_1 - 2x_2 \leq -9 \\ & x_1,\ x_2 \geq 0. \end{array}$$

Indeed, the second constraint implies that $x_1 + x_2 \geq 4.5$, which contradicts the first constraint. If a problem has no feasible solution, then the problem itself is called *infeasible*.

At the other extreme from infeasible problems, one finds unbounded problems. A problem is *unbounded* if it has feasible solutions with arbitrarily large objective values. For example, consider

$$\begin{array}{ll} \text{maximize} & x_1 - 4x_2 \\ \text{subject to} & -2x_1 + x_2 \leq -1 \\ & -x_1 - 2x_2 \leq -2 \\ & x_1,\ x_2 \geq 0. \end{array}$$

Here, we could set x_2 to zero and let x_1 be arbitrarily large. As long as x_1 is greater than 2 the solution will be feasible, and as it gets large the objective function does too. Hence, the problem is unbounded. In addition to finding optimal solutions to linear programming problems, we shall also be interested in detecting when a problem is infeasible or unbounded.

Exercises

1.1 A steel company must decide how to allocate next week's time on a rolling mill, which is a machine that takes unfinished slabs of steel as input and can produce either of two semi-finished products: bands and coils. The mill's two products come off the rolling line at different rates:

Bands	200 tons/hr
Coils	140 tons/hr .

They also produce different profits:

Bands	$ 25/ton
Coils	$ 30/ton .

Based on currently booked orders, the following upper bounds are placed on the amount of each product to produce:

Bands 6000 tons

Coils 4000 tons .

Given that there are 40 hours of production time available this week, the problem is to decide how many tons of bands and how many tons of coils should be produced to yield the greatest profit. Formulate this problem as a linear programming problem. Can you solve this problem by inspection?

1.2 A small airline, Ivy Air, flies between three cities: Ithaca, Newark, and Boston. They offer several flights but, for this problem, let us focus on the Friday afternoon flight that departs from Ithaca, stops in Newark, and continues to Boston. There are three types of passengers:

(a) Those traveling from Ithaca to Newark.
(b) Those traveling from Newark to Boston.
(c) Those traveling from Ithaca to Boston.

The aircraft is a small commuter plane that seats 30 passengers. The airline offers three fare classes:

(a) Y class: full coach.
(b) B class: nonrefundable.
(c) M class: nonrefundable, 3-week advanced purchase.

Ticket prices, which are largely determined by external influences (i.e., competitors), have been set and advertised as follows:

	Ithaca–Newark	Newark–Boston	Ithaca–Boston
Y	300	160	360
B	220	130	280
M	100	80	140

Based on past experience, demand forecasters at Ivy Air have determined the following upper bounds on the number of potential customers in each of the 9 possible origin-destination/fare-class combinations:

	Ithaca–Newark	Newark–Boston	Ithaca–Boston
Y	4	8	3
B	8	13	10
M	22	20	18

The goal is to decide how many tickets from each of the 9 origin/destination/fare-class combinations to sell. The constraints are that the plane cannot be overbooked on either of the two legs of the flight and that the number of tickets made available cannot exceed the forecasted maximum demand. The objective is to maximize the revenue. Formulate this problem as a linear programming problem.

1.3 Suppose that Y is a random variable taking on one of n known values:

$$a_1, a_2, \ldots, a_n.$$

Suppose we know that Y either has distribution p given by

$$\mathbb{P}(Y = a_j) = p_j$$

or it has distribution q given by

$$\mathbb{P}(Y = a_j) = q_j.$$

Of course, the numbers p_j, $j = 1, 2, \ldots, n$ are nonnegative and sum to one. The same is true for the q_j's. Based on a single observation of Y, we wish to guess whether it has distribution p or distribution q. That is, for each possible outcome a_j, we will assert with probability x_j that the distribution is p and with probability $1-x_j$ that the distribution is q. We wish to determine the probabilities x_j, $j = 1, 2, \ldots, n$, such that the probability of saying the distribution is p when in fact it is q has probability no larger than β, where β is some small positive value (such as 0.05). Furthermore, given this constraint, we wish to maximize the probability that we say the distribution is p when in fact it is p. Formulate this maximization problem as a linear programming problem.

Notes

The subject of linear programming has its roots in the study of linear inequalities, which can be traced as far back as 1826 to the work of Fourier. Since then, many mathematicians have proved special cases of the most important result in the subject—the *duality theorem*. The applied side of the subject got its start in 1939 when L.V. Kantorovich noted the practical importance of a certain class of linear programming problems and gave an algorithm for their solution—see Kantorovich (1960). Unfortunately, for several years, Kantorovich's work was unknown in the West and unnoticed in the East. The subject really took off in 1947 when G.B. Dantzig invented the *simplex method* for solving the linear programming problems that arose in U.S. Air Force planning problems. The earliest published accounts of Dantzig's work appeared in 1951 (Dantzig 1951*a*,*b*). His monograph (Dantzig 1963) remains an important reference. In the same year that Dantzig invented the simplex method, T.C. Koopmans showed that linear programming provided the appropriate model for the analysis of classical economic theories. In 1975, the Royal Swedish Academy of Sciences awarded the Nobel Prize in economic science to L.V. Kantorovich and T.C. Koopmans "for their contributions to the theory of optimum allocation of resources." Apparently the academy regarded Dantzig's work as too mathematical for the prize in economics (and there is no Nobel Prize in mathematics).

The textbooks by Bradley et al. (1977), Bazaraa et al. (1977), and Hillier & Lieberman (1977) are known for their extensive collections of interesting practical applications.

CHAPTER 2

The Simplex Method

In this chapter we present the simplex method as it applies to linear programming problems in standard form.

1. An Example

We first illustrate how the simplex method works on a specific example:

$$\begin{aligned} \text{maximize} \quad & 5x_1 + 4x_2 + 3x_3 \\ \text{subject to} \quad & 2x_1 + 3x_2 + x_3 \leq 5 \\ & 4x_1 + x_2 + 2x_3 \leq 11 \\ & 3x_1 + 4x_2 + 2x_3 \leq 8 \\ & x_1,\ x_2,\ x_3 \geq 0. \end{aligned} \tag{2.1}$$

We start by adding so-called *slack variables*. For each of the less-than inequalities in (2.1) we introduce a new variable that represents the difference between the right-hand side and the left-hand side. For example, for the first inequality,

$$2x_1 + 3x_2 + x_3 \leq 5,$$

we introduce the slack variable w_1 defined by

$$w_1 = 5 - 2x_1 - 3x_2 - x_3.$$

It is clear then that this definition of w_1, together with the stipulation that w_1 be non-negative, is equivalent to the original constraint. We carry out this procedure for each of the less-than constraints to get an equivalent representation of the problem:

$$\begin{aligned} \text{maximize} \quad & \zeta = 5x_1 + 4x_2 + 3x_3 \\ \text{subject to} \quad & w_1 = 5 - 2x_1 - 3x_2 - x_3 \\ & w_2 = 11 - 4x_1 - x_2 - 2x_3 \\ & w_3 = 8 - 3x_1 - 4x_2 - 2x_3 \\ & x_1,\ x_2,\ x_3,\ w_1,\ w_2,\ w_3 \geq 0. \end{aligned} \tag{2.2}$$

Note that we have included a notation, ζ, for the value of the objective function, $5x_1 + 4x_2 + 3x_3$.

The simplex method is an iterative process in which we start with a solution $x_1, x_2, \ldots, w_3$ that satisfies the equations and nonnegativities in (2.2) and then look for a new solution $\bar{x}_1, \bar{x}_2, \ldots, \bar{w}_3$, which is better in the sense that it has a larger objective function value:

$$5\bar{x}_1 + 4\bar{x}_2 + 3\bar{x}_3 > 5x_1 + 4x_2 + 3x_3.$$

We continue this process until we arrive at a solution that can't be improved. This final solution is then an optimal solution.

To start the iterative process, we need an initial feasible solution $x_1, x_2, \ldots, w_3$. For our example, this is easy. We simply set all the original variables to zero and use the defining equations to determine the slack variables:

$$x_1 = 0, \quad x_2 = 0, \quad x_3 = 0, \quad w_1 = 5, \quad w_2 = 11, \quad w_3 = 8.$$

The objective function value associated with this solution is $\zeta = 0$.

We now ask whether this solution can be improved. Since the coefficient of x_1 is positive, if we increase the value of x_1 from zero to some positive value, we will increase ζ. But as we change its value, the values of the slack variables will also change. We must make sure that we don't let any of them go negative. Since $x_2 = x_3 = 0$, we see that $w_1 = 5 - 2x_1$, and so keeping w_1 nonnegative imposes the restriction that x_1 must not exceed $5/2$. Similarly, the nonnegativity of w_2 imposes the bound that $x_1 \leq 11/4$, and the nonnegativity of w_3 introduces the bound that $x_1 \leq 8/3$. Since all of these conditions must be met, we see that x_1 cannot be made larger than the smallest of these bounds: $x_1 \leq 5/2$. Our new, improved solution then is

$$x_1 = \frac{5}{2}, \quad x_2 = 0, \quad x_3 = 0, \quad w_1 = 0, \quad w_2 = 1, \quad w_3 = \frac{1}{2}.$$

This first step was straightforward. It is less obvious how to proceed. What made the first step easy was the fact that we had one group of variables that were initially zero and we had the rest explicitly expressed in terms of these. This property can be arranged even for our new solution. Indeed, we simply must rewrite the equations in (2.2) in such a way that x_1, w_2, w_3, and ζ are expressed as functions of w_1, x_2, and x_3. That is, the roles of x_1 and w_1 must be swapped. To this end, we use the equation for w_1 in (2.2) to solve for x_1:

$$x_1 = \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3.$$

The equations for w_2, w_3, and ζ must also be doctored so that x_1 does not appear on the right. The easiest way to accomplish this is to do so-called *row operations* on the

equations in (2.2). For example, if we take the equation for w_2 and subtract two times the equation for w_1 and then bring the w_1 term to the right-hand side, we get

$$w_2 = 1 + 2w_1 + 5x_2.$$

Performing analogous row operations for w_3 and ζ, we can rewrite the equations in (2.2) as

$$\begin{aligned} \zeta &= 12.5 - 2.5w_1 - 3.5x_2 + 0.5x_3 \\ \hline x_1 &= \ 2.5 - 0.5w_1 - 1.5x_2 - 0.5x_3 \\ w_2 &= \quad 1 + \quad 2w_1 + \quad 5x_2 \\ w_3 &= \ 0.5 + 1.5w_1 + 0.5x_2 - 0.5x_3. \end{aligned} \tag{2.3}$$

Note that we can recover our current solution by setting the "independent" variables to zero and using the equations to read off the values for the "dependent" variables.

Now we see that increasing w_1 or x_2 will bring about a *decrease* in the objective function value, and so x_3, being the only variable with a positive coefficient, is the only independent variable that we can increase to obtain a further increase in the objective function. Again, we need to determine how much this variable can be increased without violating the requirement that all the dependent variables remain nonnegative. This time we see that the equation for w_2 is not affected by changes in x_3, but the equations for x_1 and w_3 do impose bounds, namely $x_3 \leq 5$ and $x_3 \leq 1$, respectively. The latter is the tighter bound, and so the new solution is

$$x_1 = 2, \quad x_2 = 0, \quad x_3 = 1, \quad w_1 = 0, \quad w_2 = 1, \quad w_3 = 0.$$

The corresponding objective function value is $\zeta = 13$.

Once again, we must determine whether it is possible to increase the objective function further and, if so, how. Therefore, we need to write our equations with ζ, x_1, w_2, and x_3 written as functions of w_1, x_2, and w_3. Solving the last equation in (2.3) for x_3, we get

$$x_3 = 1 + 3w_1 + x_2 - 2w_3.$$

Also, performing the appropriate row operations, we can eliminate x_3 from the other equations. The result of these operations is

$$\begin{aligned} \zeta &= 13 - \ w_1 - 3x_2 - \ w_3 \\ \hline x_1 &= \ 2 - 2w_1 - 2x_2 + \ w_3 \\ w_2 &= \ 1 + 2w_1 + 5x_2 \\ x_3 &= \ 1 + 3w_1 + \ x_2 - 2w_3. \end{aligned} \tag{2.4}$$

We are now ready to begin the third iteration. The first step is to identify an independent variable for which an increase in its value would produce a corresponding increase in ζ. But this time there is no such variable, since all the variables have negative coefficients in the expression for ζ. This fact not only brings the simplex method to a standstill but also proves that the current solution is optimal. The reason is quite simple. Since the equations in (2.4) are completely equivalent to those in (2.2) and, since all the variables must be nonnegative, it follows that $\zeta \leq 13$ for every feasible solution. Since our current solution attains the value of 13, we see that it is indeed optimal.

1.1. Dictionaries, Bases, Etc. The systems of equations (2.2), (2.3), and (2.4) that we have encountered along the way are called *dictionaries*. With the exception of ζ, the variables that appear on the left (i.e., the variables that we have been referring to as the dependent variables) are called *basic variables*. Those on the right (i.e., the independent variables) are called *nonbasic variables*. The solutions we have obtained by setting the nonbasic variables to zero are called *basic feasible solutions*.

2. The Simplex Method

Consider the general linear programming problem presented in standard form:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, 2, \ldots, m \\ & x_j \geq 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

Our first task is to introduce slack variables and a name for the objective function value:

$$\begin{aligned} \zeta = & \sum_{j=1}^{n} c_j x_j \\ w_i = b_i - & \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, \ldots, m. \end{aligned} \tag{2.5}$$

As we saw in our example, as the simplex method proceeds, the slack variables become intertwined with the original variables, and the whole collection is treated the same. Therefore, it is at times convenient to have a notation in which the slack variables are more or less indistinguishable from the original variables. So we simply add them to the end of the list of x-variables:

$$(x_1, \ldots, x_n, w_1, \ldots, w_m) = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}).$$

That is, we let $x_{n+i} = w_i$. With this notation, we can rewrite (2.5) as

$$\begin{aligned} \zeta &= \sum_{j=1}^{n} c_j x_j \\ x_{n+i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, \ldots, m. \end{aligned}$$

This is the starting dictionary. As the simplex method progresses, it moves from one dictionary to another in its search for an optimal solution. Each dictionary has m basic variables and n nonbasic variables. Let $\mathcal{B}$ denote the collection of indices from $\{1, 2, \ldots, n+m\}$ corresponding to the basic variables, and let $\mathcal{N}$ denote the indices corresponding to the nonbasic variables. Initially, we have $\mathcal{N} = \{1, 2, \ldots, n\}$ and $\mathcal{B} = \{n+1, n+2, \ldots, n+m\}$, but this of course changes after the first iteration. Down the road, the current dictionary will look like this:

$$\begin{aligned} \zeta &= \bar{\zeta} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j \\ x_i &= \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \qquad i \in \mathcal{B}. \end{aligned} \tag{2.6}$$

Note that we have put bars over the coefficients to indicate that they change as the algorithm progresses.

Within each iteration of the simplex method, exactly one variable goes from nonbasic to basic and exactly one variable goes from basic to nonbasic. We saw this process in our example, but let us now describe it in general.

The variable that goes from nonbasic to basic is called the *entering variable*. It is chosen with the aim of increasing ζ; that is, one whose coefficient is positive: *pick* k *from* $\{j \in \mathcal{N} : \bar{c}_j > 0\}$. Note that if this set is empty, then the current solution is optimal. If the set consists of more than one element (as is normally the case), then we have a choice of which element to pick. There are several possible selection criteria, some of which will be discussed in the next chapter. For now, suffice it to say that we usually pick an index k having the largest coefficient (which again could leave us with a choice).

The variable that goes from basic to nonbasic is called the *leaving variable*. It is chosen to preserve nonnegativity of the current basic variables. Once we have decided that x_k will be the entering variable, its value will be increased from zero to a positive value. This increase will change the values of the basic variables:

$$x_i = \bar{b}_i - \bar{a}_{ik} x_k, \qquad i \in \mathcal{B}.$$

We must ensure that each of these variables remains nonnegative. Hence, we require that

$$\bar{b}_i - \bar{a}_{ik} x_k \geq 0, \qquad i \in \mathcal{B}. \tag{2.7}$$

Of these expressions, the only ones that can go negative as x_k increases are those for which $\bar{a}_{ik}$ is positive; the rest remain fixed or increase. Hence, we can restrict our attention to those i's for which $\bar{a}_{ik}$ is positive. And for such an i, the value of x_k at which the expression becomes zero is

$$x_k = \bar{b}_i / \bar{a}_{ik}.$$

Since we don't want any of these to go negative, we must raise x_k only to the smallest of all of these values:

$$x_k = \min_{i \in \mathcal{B}: \bar{a}_{ik} > 0} \bar{b}_i / \bar{a}_{ik}.$$

Therefore, with a certain amount of latitude remaining, the rule for selecting the leaving variable is *pick l from $\{i \in \mathcal{B} : \bar{a}_{ik} > 0$ and $\bar{b}_i/\bar{a}_{ik}$ is minimal$\}$*.

The rule just given for selecting a leaving variable describes exactly the process by which we use the rule in practice. That is, we look only at those variables for which $\bar{a}_{ik}$ is positive and among those we select one with the smallest value of the ratio $\bar{b}_i/\bar{a}_{ik}$. There is, however, another, entirely equivalent, way to write this rule which we will often use. To derive this alternate expression we use the convention that $0/0 = 0$ and rewrite inequalities (2.7) as

$$\frac{1}{x_k} \geq \frac{\bar{a}_{ik}}{\bar{b}_i}, \qquad i \in \mathcal{B}$$

(we shall discuss shortly what happens when one of these ratios is an indeterminate form $0/0$ as well as what it means if none of the ratios are positive). Since we wish to take the largest possible increase in x_k, we see that

$$x_k = \left(\max_{i \in \mathcal{B}} \frac{\bar{a}_{ik}}{\bar{b}_i} \right)^{-1}.$$

Hence, the rule for selecting the leaving variable is as follows: *pick l from $\{i \in \mathcal{B} : \bar{a}_{ik}/\bar{b}_i$ is maximal$\}$*.

The main difference between these two ways of writing the rule is that in one we minimize the ratio of $\bar{a}_{ik}$ to $\bar{b}_i$ whereas in the other we maximize the reciprocal ratio. Of course, in the minimize formulation one must take care about the sign of the $\bar{a}_{ik}$'s. In the remainder of this book we will encounter these types of ratios often. We will always write them in the maximize form since that is shorter to write, acknowledging full well the fact that it is often more convenient, in practice, to do it the other way.

Once the leaving-basic and entering-nonbasic variables have been selected, the move from the current dictionary to the new dictionary involves appropriate row operations to achieve the interchange. This step from one dictionary to the next is called a *pivot*.

As mentioned above, there is often more than one choice for the entering and the leaving variables. Particular rules that make the choice unambiguous are called *pivot rules*.

3. Initialization

In the previous section, we presented the simplex method. However, we only considered problems for which the right-hand sides were all nonnegative. This ensured that the initial dictionary was feasible. In this section, we shall discuss what one needs to do when this is not the case.

Given a linear programming problem

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i = 1, 2, \ldots, m \\ & x_j \ge 0 \qquad j = 1, 2, \ldots, n, \end{aligned}$$

the initial dictionary that we introduced in the preceding section was

$$\begin{aligned} \zeta &= \sum_{j=1}^{n} c_j x_j \\ w_i &= b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, \ldots, m. \end{aligned}$$

The solution associated with this dictionary is obtained by setting each x_j to zero and setting each w_i equal to the corresponding b_i. This solution is feasible if and only if all the right-hand sides are nonnegative. But what if they are not? We handle this difficulty by introducing an *auxiliary problem* for which

(1) a feasible dictionary is easy to find and
(2) the optimal dictionary provides a feasible dictionary for the original problem.

The auxiliary problem is

$$\begin{aligned} \text{maximize} \quad & -x_0 \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij}x_j - x_0 \le b_i \qquad i = 1, 2, \ldots, m \\ & x_j \ge 0 \qquad j = 0, 1, \ldots, n. \end{aligned}$$

It is easy to give a feasible solution to this auxiliary problem. Indeed, we simply set $x_j = 0$, for $j = 1, \ldots, n$, and then pick x_0 sufficiently large. It is also easy to see that the original problem has a feasible solution if and only if the auxiliary problem has a feasible solution with $x_0 = 0$. In other words, the original problem has a feasible solution if and only if the optimal solution to the auxiliary problem has objective value zero.

Even though the auxiliary problem clearly has feasible solutions, we have not yet shown that it has an easily obtained feasible dictionary. It is best to illustrate how to obtain a feasible dictionary with an example:

$$\begin{aligned} \text{maximize} \quad & -2x_1 - x_2 \\ \text{subject to} \quad & -x_1 + x_2 \le -1 \\ & -x_1 - 2x_2 \le -2 \\ & x_2 \le 1 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

The auxiliary problem is

$$\begin{aligned} \text{maximize} \quad & -x_0 \\ \text{subject to} \quad & -x_1 + x_2 - x_0 \le -1 \\ & -x_1 - 2x_2 - x_0 \le -2 \\ & x_2 - x_0 \le 1 \\ & x_0,\ x_1,\ x_2 \ge 0. \end{aligned}$$

Next we introduce slack variables and write down an initial *infeasible dictionary*:

$$\begin{aligned} \xi &= -x_0 \\ \hline w_1 &= -1 + x_1 - x_2 + x_0 \\ w_2 &= -2 + x_1 + 2x_2 + x_0 \\ w_3 &= 1 - x_2 + x_0. \end{aligned}$$

This dictionary is infeasible, but it is easy to convert it into a feasible dictionary. In fact, all we need to do is one pivot with variable x_0 entering and the "most infeasible variable," w_2, leaving the basis:

$$\begin{aligned}\xi &= -2 + x_1 + 2x_2 - w_2\\ \hline w_1 &= 1 - 3x_2 + w_2\\ x_0 &= 2 - x_1 - 2x_2 + w_2\\ w_3 &= 3 - x_1 - 3x_2 + w_2.\end{aligned}$$

Note that we now have a feasible dictionary, so we can apply the simplex method as defined earlier in this chapter. For the first step, we pick x_2 to enter and w_1 to leave the basis:

$$\begin{aligned}\xi &= -1.33 + x_1 - 0.67w_1 - 0.33w_2\\ \hline x_2 &= 0.33 - 0.33w_1 + 0.33w_2\\ x_0 &= 1.33 - x_1 + 0.67w_1 + 0.33w_2\\ w_3 &= 2 - x_1 + w_1\,.\end{aligned}$$

Now, for the second step, we pick x_1 to enter and x_0 to leave the basis:

$$\begin{aligned}\xi &= 0 - x_0\\ \hline x_2 &= 0.33 - 0.33w_1 + 0.33w_2\\ x_1 &= 1.33 - x_0 + 0.67w_1 + 0.33w_2\\ w_3 &= 0.67 + x_0 + 0.33w_1 - 0.33w_2.\end{aligned}$$

This dictionary is optimal for the auxiliary problem. We now drop x_0 from the equations and reintroduce the original objective function:

$$\zeta = -2x_1 - x_2 = -3 - w_1 - w_2.$$

Hence, the starting feasible dictionary for the original problem is

$$\begin{aligned}\zeta &= -3 - w_1 - w_2\\ \hline x_2 &= 0.33 - 0.33w_1 + 0.33w_2\\ x_1 &= 1.33 + 0.67w_1 + 0.33w_2\\ w_3 &= 0.67 + 0.33w_1 - 0.33w_2.\end{aligned}$$

As it turns out, this dictionary is optimal for the original problem (since the coefficients of all the variables in the equation for ζ are negative), but we can't expect to be this lucky in general. All we normally can expect is that the dictionary so obtained will

be feasible for the original problem, at which point we continue to apply the simplex method until an optimal solution is reached.

The process of solving the auxiliary problem to find an initial feasible solution is often referred to as *Phase I*, whereas the process of going from a feasible solution to an optimal solution is called *Phase II*.

4. Unboundedness

In this section, we shall discuss how to detect when the objective function value is unbounded.

Let us now take a closer look at the "leaving variable" computation: *pick l from $\{i \in \mathcal{B} : \bar{a}_{ik}/\bar{b}_i$ is maximal$\}$*. We avoided the issue before, but now we must face what to do if a denominator in one of these ratios vanishes. If the numerator is nonzero, then it is easy to see that the ratio should be interpreted as plus or minus infinity depending on the sign of the numerator. For the case of $0/0$, the correct convention (as we'll see momentarily) is to take this as a zero.

What if all of the ratios, $\bar{a}_{ik}/\bar{b}_i$, are nonpositive? In that case, none of the basic variables will become zero as the entering variable increases. Hence, the entering variable can be increased indefinitely to produce an arbitrarily large objective value. In such situations, we say that the problem is *unbounded*. For example, consider the following dictionary:

$$\begin{aligned} \zeta &= 5 + x_3 - x_1 \\ \hline x_2 &= 5 + 2x_3 - 3x_1 \\ x_4 &= 7 - 4x_1 \\ x_5 &= x_1. \end{aligned}$$

The entering variable is x_3 and the ratios are

$$-2/5, \quad -0/7, \quad 0/0.$$

Since none of these ratios is positive, the problem is unbounded.

In the next chapter, we will investigate what happens when some of these ratios take the value $+\infty$.

5. Geometry

When the number of variables in a linear programming problem is three or less, we can graph the set of feasible solutions together with the level sets of the objective function. From this picture, it is usually a trivial matter to write down the optimal

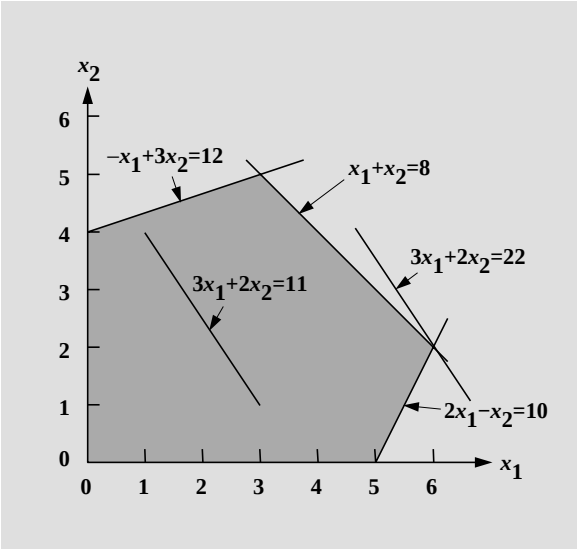


FIGURE 2.1. The set of feasible solutions together with level sets of the objective function.

solution. To illustrate, consider the following problem:

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 2x_2 \\ \text{subject to} \quad & -x_1 + 3x_2 \leq 12 \\ & x_1 + \ x_2 \leq \ 8 \\ & 2x_1 - \ x_2 \leq 10 \\ & x_1, \ x_2 \geq \ 0. \end{aligned}$$

Each constraint (including the nonnegativity constraints on the variables) is a half-plane. These half-planes can be determined by first graphing the equation one obtains by replacing the inequality with an equality and then asking whether or not some specific point that doesn't satisfy the equality (often $(0, 0)$ can be used) satisfies the inequality constraint. The set of feasible solutions is just the intersection of these half-planes. For the problem given above, this set is shown in Figure 2.1. Also shown are two level sets of the objective function. One of them indicates points at which the objective function value is 11. This level set passes through the middle of the set of feasible solutions. As the objective function value increases, the corresponding level set moves to the right. The level set corresponding to the case where the objective function equals 22 is the last level set that touches the set of feasible solutions.

Clearly, this is the maximum value of the objective function. The optimal solution is the intersection of this level set with the set of feasible solutions. Hence, we see from Figure 2.1 that the optimal solution is $(x_1, x_2) = (6, 2)$.

Exercises

Solve the following linear programming problems. If you wish, you may check your arithmetic by using the simple online pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/simple.html

2.1
$$\begin{aligned} \text{maximize } & 6x_1 + 8x_2 + 5x_3 + 9x_4 \\ \text{subject to } & 2x_1 + x_2 + x_3 + 3x_4 \le 5 \\ & x_1 + 3x_2 + x_3 + 2x_4 \le 3 \\ & x_1,\ x_2,\ x_3,\ x_4 \ge 0. \end{aligned}$$

2.2
$$\begin{aligned} \text{maximize } & 2x_1 + x_2 \\ \text{subject to } & 2x_1 + x_2 \le 4 \\ & 2x_1 + 3x_2 \le 3 \\ & 4x_1 + x_2 \le 5 \\ & x_1 + 5x_2 \le 1 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

2.3
$$\begin{aligned} \text{maximize } & 2x_1 - 6x_2 \\ \text{subject to } & -x_1 - x_2 - x_3 \le -2 \\ & 2x_1 - x_2 + x_3 \le 1 \\ & x_1,\ x_2,\ x_3 \ge 0. \end{aligned}$$

2.4
$$\begin{aligned} \text{maximize } & -x_1 - 3x_2 - x_3 \\ \text{subject to } & 2x_1 - 5x_2 + x_3 \le -5 \\ & 2x_1 - x_2 + 2x_3 \le 4 \\ & x_1,\ x_2,\ x_3 \ge 0. \end{aligned}$$

2.5
$$\begin{aligned} \text{maximize } & x_1 + 3x_2 \\ \text{subject to } & -x_1 - x_2 \le -3 \\ & -x_1 + x_2 \le -1 \\ & x_1 + 2x_2 \le 4 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

2.6
$$\begin{aligned} \text{maximize} \quad & x_1 + 3x_2 \\ \text{subject to} \quad & -x_1 - x_2 \le -3 \\ & -x_1 + x_2 \le -1 \\ & x_1 + 2x_2 \le 2 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

2.7
$$\begin{aligned} \text{maximize} \quad & x_1 + 3x_2 \\ \text{subject to} \quad & -x_1 - x_2 \le -3 \\ & -x_1 + x_2 \le -1 \\ & -x_1 + 2x_2 \le 2 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

2.8
$$\begin{aligned} \text{maximize} \quad & 3x_1 + 2x_2 \\ \text{subject to} \quad & x_1 - 2x_2 \le 1 \\ & x_1 - x_2 \le 2 \\ & 2x_1 - x_2 \le 6 \\ & x_1 \le 5 \\ & 2x_1 + x_2 \le 16 \\ & x_1 + x_2 \le 12 \\ & x_1 + 2x_2 \le 21 \\ & x_2 \le 10 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

2.9
$$\begin{aligned} \text{maximize} \quad & 2x_1 + 3x_2 + 4x_3 \\ \text{subject to} \quad & -2x_2 - 3x_3 \ge -5 \\ & x_1 + x_2 + 2x_3 \le 4 \\ & x_1 + 2x_2 + 3x_3 \le 7 \\ & x_1,\ x_2,\ x_3 \ge 0. \end{aligned}$$

2.10
$$\begin{aligned} \text{maximize} \quad & 6x_1 + 8x_2 + 5x_3 + 9x_4 \\ \text{subject to} \quad & x_1 + x_2 + x_3 + x_4 = 1 \\ & x_1,\ x_2,\ x_3,\ x_4 \ge 0. \end{aligned}$$

2.11
$$\begin{array}{lrcrcrcrcrcl}
\text{minimize} & x_{12} + & 8x_{13} + & 9x_{14} + & 2x_{23} + & 7x_{24} + & 3x_{34} & & & & & \\
\text{subject to} & x_{12} + & x_{13} + & x_{14} & & & & & & & & \geq 1 \\
& -x_{12} & & & + & x_{23} + & x_{24} & & & & & = 0 \\
& & -x_{13} & & - & x_{23} & & + & x_{34} & & & = 0 \\
& & & x_{14} & & & + & x_{24} + & x_{34} & & & \leq 1 \\
& & & & & & & & x_{12}, x_{13}, \ldots, x_{34} & & & \geq 0.
\end{array}$$

2.12 Using today's date (MMYY) for the seed value, solve 10 initially feasible problems using the online pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/primal.html .

2.13 Using today's date (MMYY) for the seed value, solve 10 not necessarily feasible problems using the online pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/primal_x0.html .

2.14 Consider the following dictionary:

$$\begin{aligned}
\zeta &= 3 + x_1 + 6x_2 \\
\hline
w_1 &= 1 + x_1 - x_2 \\
w_2 &= 5 - 2x_1 - 3x_2.
\end{aligned}$$

Using the largest coefficient rule to pick the entering variable, compute the dictionary that results after *one pivot*.

2.15 Show that the following dictionary cannot be the optimal dictionary for any linear programming problem in which w_1 and w_2 are the initial slack variables:

$$\begin{aligned}
\zeta &= 4 - w_1 - 2x_2 \\
\hline
x_1 &= 3 - 2x_2 \\
w_2 &= 1 + w_1 - x_2.
\end{aligned}$$

Hint: if it could, what was the original problem from whence it came?

2.16 Graph the feasible region for Exercise 2.8. Indicate on the graph the sequence of basic solutions produced by the simplex method.

2.17 Give an example showing that the variable that becomes basic in one iteration of the simplex method can become nonbasic in the next iteration.

2.18 Show that the variable that becomes nonbasic in one iteration of the simplex method cannot become basic in the next iteration.

2.19 Solve the following linear programming problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} q_j x_j \le \beta \\
& x_j \le 1 \qquad j = 1, 2, \ldots, n \\
& x_j \ge 0 \qquad j = 1, 2, \ldots, n.
\end{aligned}$$

Here, the numbers p_j, $j = 1, 2, \ldots, n$, are positive and sum to one. The same is true of the q_j's:

$$\sum_{j=1}^{n} q_j = 1$$

$$q_j > 0.$$

Furthermore (with only minor loss of generality), you may assume that

$$\frac{p_1}{q_1} < \frac{p_2}{q_2} < \cdots < \frac{p_n}{q_n}.$$

Finally, the parameter β is a small positive number. See Exercise 1.3 for the motivation for this problem.

Notes

The simplex method was invented by G.B. Dantzig in 1949. His monograph (Dantzig 1963) is the classical reference. Most texts describe the simplex method as a sequence of pivots on a table of numbers called the *simplex tableau*. Following Chvátal (1983), we have developed the algorithm using the more memorable dictionary notation.

CHAPTER 3

Degeneracy

In the previous chapter, we discussed what it means when the ratios computed to calculate the leaving variable are all nonpositive (the problem is unbounded). In this chapter, we take up the more delicate issue of what happens when some of the ratios are infinite (i.e., their denominators vanish).

1. Definition of Degeneracy

We say that a dictionary is *degenerate* if $\bar{b}_i$ vanishes for some $i \in \mathcal{B}$. A degenerate dictionary could cause difficulties for the simplex method, but it might not. For example, the dictionary we were discussing at the end of the last chapter,

$$\begin{aligned}
\zeta &= 5 + x_3 - x_1 \\
\hline
x_2 &= 5 + 2x_3 - 3x_1 \\
x_4 &= 7 - 4x_1 \\
x_5 &= x_1,
\end{aligned}$$

is degenerate, but it was clear that the problem was unbounded and therefore no more pivots were required. Furthermore, had the coefficient of x_3 in the equation for x_2 been -2 instead of 2, then the simplex method would have picked x_2 for the leaving variable and no difficulties would have been encountered.

Problems arise, however, when a degenerate dictionary produces degenerate pivots. We say that a pivot is a *degenerate pivot* if one of the ratios in the calculation of the leaving variable is $+\infty$; i.e., if the numerator is positive and the denominator vanishes. To see what happens, let's look at a few examples.

2. Two Examples of Degenerate Problems

Here is an example of a degenerate dictionary in which the pivot is also degenerate:

$$\begin{aligned}
\zeta &= 3 - 0.5x_1 + 2x_2 - 1.5w_1 \\
\hline
x_3 &= 1 - 0.5x_1 - 0.5w_1 \\
w_2 &= x_1 - x_2 + w_1.
\end{aligned} \tag{3.1}$$

For this dictionary, the entering variable is x_2 and the ratios computed to determine the leaving variable are 0 and $+\infty$. Hence, the leaving variable is w_2, and the fact that the ratio is infinite means that as soon as x_2 is increased from zero to a positive value, w_2 will go negative. Therefore, x_2 can't really increase. Nonetheless, it can be reclassified from nonbasic to basic (with w_2 going the other way). Let's look at the result of this degenerate pivot:

$$\begin{aligned} \zeta &= 3 + 1.5x_1 - 2w_2 + 0.5w_1 \\ \hline x_3 &= 1 - 0.5x_1 \qquad\quad - 0.5w_1 \\ x_2 &= \qquad x_1 - \;\; w_2 + \quad\; w_1. \end{aligned} \tag{3.2}$$

Note that $\bar{\zeta}$ remains unchanged at 3. Hence, this degenerate pivot has not produced any increase in the objective function value. Furthermore, the values of the variables haven't even changed: both before and after this degenerate pivot, we had

$$(x_1, x_2, x_3, w_1, w_2) = (0, 0, 1, 0, 0).$$

But we are now representing this solution in a new way, and perhaps the next pivot will make an improvement, or if not the next pivot perhaps the one after that. Let's see what happens for the problem at hand. The entering variable for the next iteration is x_1 and the leaving variable is x_3, producing a nondegenerate pivot that leads to

$$\begin{aligned} \zeta &= 6 - 3x_3 - 2w_2 - w_1 \\ \hline x_1 &= 2 - 2x_3 \qquad\quad - w_1 \\ x_2 &= 2 - 2x_3 - \;\; w_2 \;\; . \end{aligned}$$

These two pivots illustrate what typically happens. When one reaches a degenerate dictionary, it is usual that one or more of the subsequent pivots will be degenerate but that eventually a nondegenerate pivot will lead us away from these degenerate dictionaries. While it is typical for some pivot to "break away" from the degeneracy, the real danger is that the simplex method will make a sequence of degenerate pivots and eventually return to a dictionary that has appeared before, in which case the simplex method enters an infinite loop and never finds an optimal solution. This behavior is called *cycling*.

Unfortunately, under certain pivoting rules, cycling is possible. In fact, it is possible even when using one of the most popular pivoting rules:

- Choose the entering variable as the one with the largest coefficient in the ζ-row of the dictionary.
- When two or more variables compete for leaving the basis, use the one with the smallest subscript.

However, it is rare and exceedingly difficult to find examples of cycling. In fact, it has been shown that if a problem has an optimal solution but cycles off-optimum, then the problem must involve dictionaries with at least six variables and three constraints. Here is an example that cycles:

$$\begin{aligned}
\zeta &= 10x_1 - 57x_2 - 9x_3 - 24x_4 \\
\hline
w_1 &= -0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \\
w_2 &= -0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \\
w_3 &= 1 - x_1 .
\end{aligned}$$

And here is the sequence of dictionaries the above pivot rules produce. After the first iteration:

$$\begin{aligned}
\zeta &= -20w_1 + 53x_2 + 41x_3 - 204x_4 \\
\hline
x_1 &= -2w_1 + 11x_2 + 5x_3 - 18x_4 \\
w_2 &= w_1 - 4x_2 - 2x_3 + 8x_4 \\
w_3 &= 1 + 2w_1 - 11x_2 - 5x_3 + 18x_4 .
\end{aligned}$$

After the second iteration:

$$\begin{aligned}
\zeta &= -6.75w_1 - 13.25w_2 + 14.5x_3 - 98x_4 \\
\hline
x_1 &= 0.75w_1 - 2.75w_2 - 0.5x_3 + 4x_4 \\
x_2 &= 0.25w_1 - 0.25w_2 - 0.5x_3 + 2x_4 \\
w_3 &= 1 - 0.75w_1 - 13.25w_2 + 0.5x_3 - 4x_4 .
\end{aligned}$$

After the third iteration:

$$\begin{aligned}
\zeta &= 15w_1 - 93w_2 - 29x_1 + 18x_4 \\
\hline
x_3 &= 1.5w_1 - 5.5w_2 - 2x_1 + 8x_4 \\
x_2 &= -0.5w_1 + 2.5w_2 + x_1 - 2x_4 \\
w_3 &= 1 - x_1 .
\end{aligned}$$

After the fourth iteration:

$$\begin{aligned}
\zeta &= 10.5w_1 - 70.5w_2 - 20x_1 - 9x_2 \\
\hline
x_3 &= -0.5w_1 + 4.5w_2 + 2x_1 - 4x_2 \\
x_4 &= -0.25w_1 + 1.25w_2 + 0.5x_1 - 0.5x_2 \\
w_3 &= 1 - x_1 .
\end{aligned}$$

After the fifth iteration:

$$\begin{aligned}
\zeta = &- 21x_3 + 24w_2 + 22x_1 - 93x_2 \\
\hline
w_1 = &- 2x_3 + 9w_2 + 4x_1 - 8x_2 \\
x_4 = &\ 0.5x_3 - w_2 - 0.5x_1 + 1.5x_2 \\
w_3 = 1 &- x_1 .
\end{aligned}$$

After the sixth iteration:

$$\begin{aligned}
\zeta = &\ 10x_1 - 57x_2 - 9x_3 - 24x_4 \\
\hline
w_1 = &- 0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \\
w_2 = &- 0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \\
w_3 = 1 &- x_1 .
\end{aligned}$$

Note that we have come back to the original dictionary, and so from here on the simplex method simply cycles through these six dictionaries and never makes any further progress toward an optimal solution. As bad as cycling is, the following theorem tells us that nothing worse can happen:

THEOREM 3.1. *If the simplex method fails to terminate, then it must cycle.*

PROOF. A dictionary is completely determined by specifying which variables are basic and which are nonbasic. There are only

$$\binom{n+m}{m}$$

different possibilities. If the simplex method fails to terminate, it must visit some of these dictionaries more than once. Hence, the algorithm cycles. □

Note that, if the simplex method cycles, then all the pivots within the cycle must be degenerate. This is easy to see, since the objective function value never decreases. Hence, it follows that all the pivots within the cycle must have the same objective function value, i.e., all of the these pivots must be degenerate.

In practice, degeneracy is very common. But cycling is rare. In fact, it is so rare that most efficient implementations do not take precautions against it. Nonetheless, it is important to know that there are variants of the simplex method that do not cycle. This is the subject of the next two sections.

3. The Perturbation/Lexicographic Method

As we have seen, there is not just one algorithm called the simplex method. Instead, the simplex method is a whole family of related algorithms from which we can

pick a specific instance by specifying what we have been referring to as pivoting rules. We have also seen that, using the most natural pivoting rules, the simplex method can fail to converge to an optimal solution by occasionally cycling indefinitely through a sequence of degenerate pivots associated with a nonoptimal solution.

So this raises a natural question: are there pivoting rules for which the simplex method will definitely either reach an optimal solution or prove that no such solution exists? The answer to this question is yes, and we shall present two choices of such pivoting rules.

The first method is based on the observation that degeneracy is sort of an accident. That is, a dictionary is degenerate if one or more of the $\bar{b}_i$'s vanish. Our examples have generally used small integers for the data, and in this case it doesn't seem too surprising that sometimes cancellations occur and we end up with a degenerate dictionary. But each right-hand side could in fact be any real number, and in the world of real numbers the occurrence of any specific number, such as zero, seems to be quite unlikely. So how about perturbing a given problem by adding small random perturbations independently to each of the right-hand sides? If these perturbations are small enough, we can think of them as insignificant and hence not really changing the problem. If they are chosen independently, then the probability of an exact cancellation is zero.

Such random perturbation schemes are used in some implementations, but what we have in mind as we discuss perturbation methods is something a little bit different. Instead of using independent identically distributed random perturbations, let us consider using a fixed perturbation for each constraint, with the perturbation getting much smaller on each succeeding constraint. Indeed, we introduce a small positive number ϵ_1 for the first constraint and then a much smaller positive number ϵ_2 for the second constraint, etc. We write this as

$$0 < \epsilon_m \ll \cdots \ll \epsilon_2 \ll \epsilon_1 \ll \text{ all other data.}$$

The idea is that each ϵ_i acts on an entirely different scale from all the other ϵ_i's and the data for the problem. What we mean by this is that no linear combination of the ϵ_i's using coefficients that might arise in the course of the simplex method can ever produce a number whose size is of the same order as the data in the problem. Similarly, each of the "lower down" ϵ_i's can never "escalate" to a higher level. Hence, cancellations can only occur on a given scale. Of course, this complete isolation of scales can never be truly achieved in the real numbers, so instead of actually introducing specific values for the ϵ_i's, we simply treat them as abstract symbols having these scale properties.

To illustrate what we mean, let's look at a specific example. Consider the following degenerate dictionary:

$$\begin{array}{rl} \zeta = & 4 + 2x_1 - x_2 \\ \hline w_1 = & 0.5 \qquad - x_2 \\ w_2 = & -2x_1 + 4x_2 \\ w_3 = & x_1 - 3x_2. \end{array}$$

The first step is to introduce symbolic parameters

$$0 < \epsilon_3 \ll \epsilon_2 \ll \epsilon_1$$

to get a perturbed problem:

$$\begin{array}{rl} \zeta = & 4 \qquad\qquad + 2x_1 - x_2 \\ \hline w_1 = & 0.5 + \epsilon_1 \qquad - x_2 \\ w_2 = & \epsilon_2 \quad -2x_1 + 4x_2 \\ w_3 = & \epsilon_3 + x_1 - 3x_2. \end{array}$$

This dictionary is not degenerate. The entering variable is x_1 and the leaving variable is unambiguously w_2. The next dictionary is

$$\begin{array}{rl} \zeta = & 4 \quad + \quad \epsilon_2 \quad - \quad w_2 + 3x_2 \\ \hline w_1 = & 0.5 + \epsilon_1 \qquad - x_2 \\ x_1 = & 0.5\epsilon_2 \quad -0.5w_2 + 2x_2 \\ w_3 = & 0.5\epsilon_2 + \epsilon_3 - 0.5w_2 - x_2. \end{array}$$

For the next pivot, the entering variable is x_2 and the leaving variable is w_3. The new dictionary is

$$\begin{array}{rl} \zeta = & 4 \quad + 2.5\epsilon_2 + 3\epsilon_3 - 2.5w_2 - 3w_3 \\ \hline w_1 = & 0.5 + \epsilon_1 - 0.5\epsilon_2 - \epsilon_3 + 0.5w_2 + w_3 \\ x_1 = & 1.5\epsilon_2 + 2\epsilon_3 - 1.5w_2 - 2w_3 \\ x_2 = & 0.5\epsilon_2 + \epsilon_3 - 0.5w_2 - w_3. \end{array}$$

This last dictionary is optimal. At this point, we simply drop the symbolic ϵ_i parameters and get an optimal dictionary for the unperturbed problem:

$$\begin{aligned} \zeta = \quad & 4 - 2.5w_2 - 3w_3 \\ \hline w_1 = \, & 0.5 + 0.5w_2 + \quad w_3 \\ x_1 = \quad & -1.5w_2 - 2w_3 \\ x_2 = \quad & -0.5w_2 - \quad w_3. \end{aligned}$$

When treating the ϵ_i's as symbols, the method is called the *lexicographic method*. Note that the lexicographic method does not affect the choice of entering variable but does amount to a precise prescription for the choice of leaving variable.

It turns out that the lexicographic method produces a variant of the simplex method that never cycles:

THEOREM 3.2. *The simplex method always terminates provided that the leaving variable is selected by the lexicographic rule.*

PROOF. It suffices to show that no degenerate dictionary is ever produced. As we've discussed before, the ϵ_i's operate on different scales and hence can't cancel with each other. Therefore, we can think of the ϵ_i's as a collection of independent variables. Extracting the ϵ terms from the first dictionary, we see that we start with the following pattern:

$$\begin{aligned} \epsilon_1 & \\ & \epsilon_2 \\ & \quad \ddots \\ & \qquad \epsilon_m. \end{aligned}$$

And, after several pivots, the ϵ terms will form a system of linear combinations, say,

$$\begin{aligned} r_{11}\epsilon_1 \; + \; & r_{12}\epsilon_2 \; \ldots \; + \; r_{1m}\epsilon_m \\ r_{21}\epsilon_1 \; + \; & r_{22}\epsilon_2 \; \ldots \; + \; r_{2m}\epsilon_m \\ \vdots \qquad & \vdots \quad \ddots \qquad \vdots \\ r_{m1}\epsilon_1 + \, & r_{m2}\epsilon_2 \ldots + r_{mm}\epsilon_m. \end{aligned}$$

Since this system of linear combinations is obtained from the original system by pivot operations and, since pivot operations are reversible, it follows that the rank of the two systems must be the same. Since the original system had rank m, we see that every subsequent system must have rank m. This means that there must be at least one nonzero r_{ij} in every row i, which of course implies that none of the rows can be degenerate. Hence, no dictionary can be degenerate. □

4. Bland's Rule

The second pivoting rule we shall consider is called *Bland's rule*. It stipulates that both the entering and the leaving variable be selected from their respective sets of choices by choosing the variable x_k with the smallest index k.

THEOREM 3.3. *The simplex method always terminates provided that both the entering and the leaving variable are chosen according to Bland's rule.*

The proof may look rather involved, but the reader who spends the time to understand it will find the underlying elegance most rewarding.

PROOF. It suffices to show that such a variant of the simplex method never cycles. We prove this by assuming that cycling does occur and then showing that this assumption leads to a contradiction. So let's assume that cycling does occur. Without loss of generality, we may assume that it happens from the beginning. Let $D_0, D_1, \ldots, D_{k-1}$ denote the dictionaries through which the method cycles. That is, the simplex method produces the following sequence of dictionaries:

$$D_0, D_1, \ldots, D_{k-1}, D_0, D_1, \ldots.$$

We say that a variable is fickle if it is in some basis and not in some other basis. Let x_t be the fickle variable having the largest index and let D denote a dictionary in $D_0, D_1, \ldots, D_{k-1}$ in which x_t leaves the basis. Again, without loss of generality we may assume that $D = D_0$. Let x_s denote the corresponding entering variable. Suppose that D is recorded as follows:

$$\begin{aligned} \zeta &= v + \sum_{j \in \mathcal{N}} c_j x_j \\ x_i &= b_i - \sum_{j \in \mathcal{N}} a_{ij} x_j \qquad i \in \mathcal{B}. \end{aligned}$$

Since x_s is the entering variable and x_t is the leaving variable, we have that $s \in \mathcal{N}$ and $t \in \mathcal{B}$.

Now let D^* be a dictionary in $D_1, D_2, \ldots, D_{k-1}$ in which x_t enters the basis. Suppose that D^* is recorded as follows:

$$\begin{aligned} \zeta &= v^* + \sum_{j \in \mathcal{N}^*} c_j^* x_j \\ x_i &= b_i^* - \sum_{j \in \mathcal{N}^*} a_{ij}^* x_j \qquad i \in \mathcal{B}^*. \end{aligned} \tag{3.3}$$

Since all the dictionaries are degenerate, we have that $v^* = v$, and therefore we can write the objective function in (3.3) as

$$\zeta = v + \sum_{j=1}^{n+m} c_j^* x_j, \tag{3.4}$$

where we have extended the notation c_j^* to all variables (both original and slack) by setting $c_j^* = 0$ for $j \in \mathcal{B}^*$.

Ignoring for the moment the possibility that some variables could go negative, consider the solutions obtained by letting x_s increase while holding all other variables in $\mathcal{N}$ at zero:

$$\begin{aligned} x_s &= y, \\ x_j &= 0, && j \in \mathcal{N} \setminus \{s\}, \\ x_i &= b_i - a_{is} y, && i \in \mathcal{B}. \end{aligned}$$

The objective function at this point is given by

$$\zeta = v + c_s y.$$

However, using (3.4), we see that it is also given by

$$\zeta = v + c_s^* y + \sum_{i \in \mathcal{B}} c_i^* (b_i - a_{is} y).$$

Equating these two expressions for ζ, we see that

$$\left(c_s - c_s^* + \sum_{i \in \mathcal{B}} c_i^* a_{is} \right) y = \sum_{i \in \mathcal{B}} c_i^* b_i.$$

Since this equation must be an identity for every y, it follows that the coefficient multiplying y must vanish (as must the right-hand side):

$$c_s - c_s^* + \sum_{i \in \mathcal{B}} c_i^* a_{is} = 0.$$

Now, the fact that x_s is the entering variable in D implies that

$$c_s > 0.$$

Recall that x_t is the fickle variable with the largest index. Since x_s is also fickle, we see that $s < t$. Since x_s is not the entering variable in D^* (as x_t is), we see that

$$c_s^* \leq 0.$$

From these last three displayed equations, we get

$$\sum_{i \in \mathcal{B}} c_i^* a_{is} < 0.$$

Hence, there must exist an index $r \in \mathcal{B}$ for which

$$c_r^* a_{rs} < 0. \tag{3.5}$$

Consequently, $c_r^* \neq 0$ and $r \in \mathcal{N}^*$. Hence, x_r is fickle and therefore $r \leq t$. In fact, $r < t$, since $c_t^* a_{ts} > 0$. To see that this product is positive, note that both its factors are positive: c_t^* is positive, since x_t is the entering variable in D^*, and a_{ts} is positive, since x_t is the leaving variable in D.

The fact that $r < t$ implies that $c_r^* \leq 0$ (otherwise, according to the smallest index criteria, r would be the entering variable for D^*). Hence, (3.5) implies that

$$a_{rs} > 0.$$

Now, since each of the dictionaries in the cycle describe the same solution, it follows that every fickle variable is zero in all these dictionaries (since it is clearly zero in a dictionary in which it is nonbasic). In particular, $x_r = 0$. But in D, x_r is basic. Hence,

$$b_r = 0.$$

These last two displayed equations imply that x_r was a candidate to be the leaving variable in D, and since $r < t$, it should have been chosen over x_t. This is the contradiction we have been looking for. □

5. Fundamental Theorem of Linear Programming

Now that we have a Phase I algorithm and a variant of the simplex method that is guaranteed to terminate, we can summarize the main gist of this chapter in the following theorem:

THEOREM 3.4. *For an arbitrary linear program in standard form, the following statements are true:*

(1) *If there is no optimal solution, then the problem is either infeasible or unbounded.*
(2) *If a feasible solution exists, then a basic feasible solution exists.*
(3) *If an optimal solution exists, then a basic optimal solution exists.*

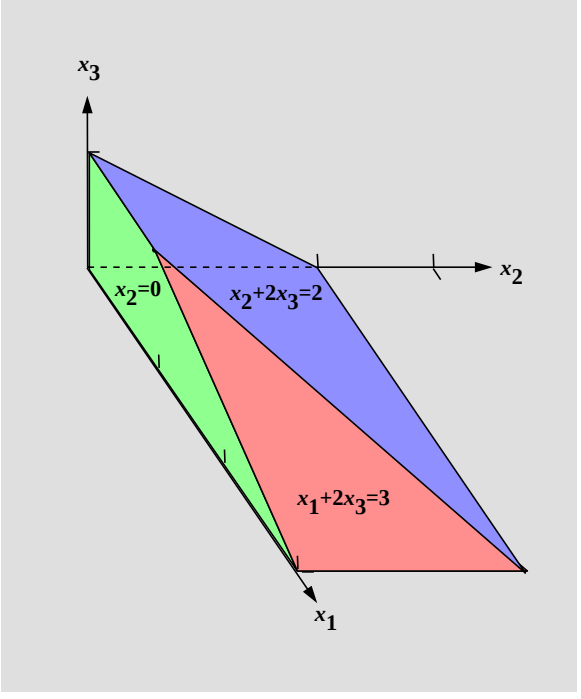


FIGURE 3.1. The set of feasible solutions for the problem given by (3.6).

PROOF. The Phase I algorithm either proves that the problem is infeasible or produces a basic feasible solution. The Phase II algorithm either discovers that the problem is unbounded or finds a basic optimal solution. These statements depend, of course, on applying a variant of the simplex method that does not cycle, which we now know to exist. □

6. Geometry

As we saw in the previous chapter, the set of feasible solutions for a problem in two dimensions is the intersection of a number of halfplanes, i.e., a polygon. In three dimensions, the situation is similar. Consider, for example, the following problem:

$$\begin{aligned} \text{maximize } & x_1 + 2x_2 + 3x_3 \\ \text{subject to } & x_1 \qquad + 2x_3 \leq 3 \\ & x_2 + 2x_3 \leq 2 \\ & x_1,\ x_2,\ x_3 \geq 0. \end{aligned} \tag{3.6}$$

The set of points satisfying $x_1 + 2x_3 = 3$ is a plane. The inequality $x_1 + 2x_3 \leq 3$ therefore consists of all points on one side of this plane; that is, it is a *halfspace*. The same is true for each of the other four inequalities. The feasible set consists of those points in space that satisfy all five inequalities, i.e., those points lying in the intersection of these halfspaces. This set is the *polyhedron* shown in Figure 3.1. This polyhedron is bordered by five *facets*, each facet being a portion of one of the planes that was defined by replacing a constraint inequality with an equation. For example, the "front" facet in the figure is a portion of the plane $x_1 + 2x_3 = 3$. The facets acquire a particularly simple description if we introduce slack variables into the problem:

$$\begin{aligned} w_1 &= 3 - x_1 \qquad - 2x_3 \\ w_2 &= 2 \qquad - x_2 - 2x_3. \end{aligned}$$

Indeed, each facet corresponds precisely to some variable (either original or slack) vanishing. For instance, the front facet in the figure corresponds to $w_1 = 0$ whereas the "left" facet corresponds to $x_2 = 0$.

The correspondences can be continued. Indeed, each *edge* of the polyhedron corresponds to a pair of variables vanishing. For example, the edge lying at the interface of the left and the front facets in the figure corresponds to both $w_1 = 0$ and $x_2 = 0$.

Going further yet, each *vertex* of the polyhedron corresponds to three variables vanishing. For instance, the vertex whose coordinates are $(1, 0, 1)$ corresponds to $w_1 = 0$, $x_2 = 0$, and $w_2 = 0$.

Now, let's think about applying the simplex method to this problem. Every basic feasible solution involves two basic variables and three nonbasic variables. Furthermore, the three nonbasic variables are, by definition, zero in the basic feasible solution. Therefore, for this example, the basic feasible solutions stand in one-to-one correspondence with the vertices of the polyhedron. In fact, applying the simplex method to this problem, one discovers that the sequence of vertices visited by the algorithm is

$$(0, 0, 0) \quad \longrightarrow \quad (0, 0, 1) \quad \longrightarrow \quad (1, 0, 1) \quad \longrightarrow \quad (3, 2, 0).$$

The example we've been considering has the nice property that every vertex is formed by the intersection of exactly three of the facets. But consider now the following problem:

$$\begin{aligned} &\text{maximize } x_1 + 2x_2 + 3x_3 \\ &\text{subject to } x_1 \qquad + 2x_3 \leq 2 \\ &\qquad\qquad\qquad\quad x_2 + 2x_3 \leq 2 \\ &\qquad\qquad\qquad\quad x_1,\ x_2,\ x_3 \geq 0. \end{aligned} \tag{3.7}$$

Algebraically, the only difference between this problem and the previous one is that the right-hand side of the first inequality is now a 2 instead of a 3. But look at the

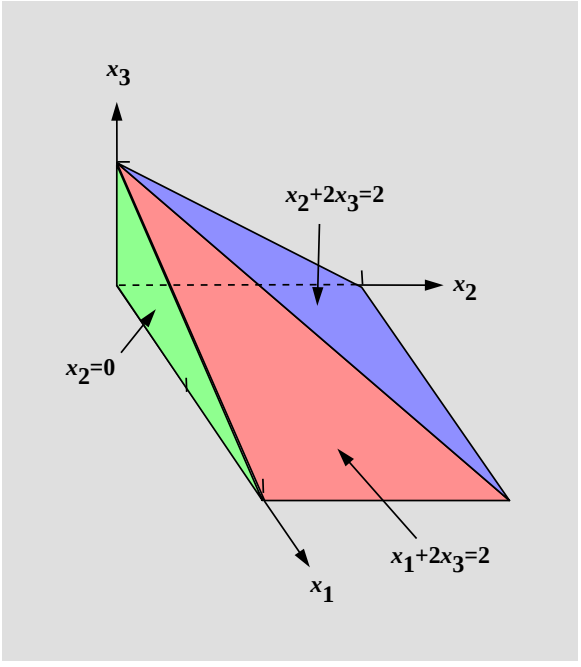


FIGURE 3.2. The set of feasible solutions for the (degenerate) problem given by (3.7).

polyhedron of feasible solutions shown in Figure 3.2. The vertex $(0, 0, 1)$ is at the intersection of four of the facets, not three as one would "normally" expect. This vertex does not correspond to one basic feasible solution; rather, there are four degenerate basic feasible solutions, each representing it. We've seen two of them before. Indeed, dictionaries (3.1) and (3.2) correspond to two of these degenerate dictionaries (in fact, dictionary (3.1) is the dictionary one obtains after one pivot of the simplex method applied to problem (3.7)).

We end by considering the geometric effect of the perturbation method for resolving degeneracy. By perturbing the right-hand sides, one moves the planes that determine the facets. If the moves are random or chosen with vastly different magnitudes (all small), then one would expect that each vertex in the perturbed problem would be determined by exactly three planes. That is, degenerate vertices from the original problem get split into multiple nearby vertices in the perturbed problem. For example, problem (3.6) can be thought of as a perturbation of degenerate problem (3.7) (the perturbation isn't small, but it also isn't so large as to obscure the effect). Note how the degenerate vertex in Figure 3.2 appears as two vertices in Figure 3.1.

Exercises

3.1 Solve the following linear program using the perturbation method to resolve degeneracy:

$$\begin{aligned}
\text{maximize} \quad & 10x_1 - 57x_2 - 9x_3 - 24x_4 \\
\text{subject to} \quad & 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 \le 0 \\
& 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 \le 0 \\
& x_1 \le 1 \\
& x_1,\ x_2,\ x_3,\ x_4 \ge 0.
\end{aligned}$$

Note: The simple pivot tool with the *Lexicographic* labels can be used to check your arithmetic:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/simple.html

3.2 Solve the following linear program using Bland's rule to resolve degeneracy:

$$\begin{aligned}
\text{maximize} \quad & 10x_1 - 57x_2 - 9x_3 - 24x_4 \\
\text{subject to} \quad & 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 \le 0 \\
& 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 \le 0 \\
& x_1 \le 1 \\
& x_1,\ x_2,\ x_3,\ x_4 \ge 0.
\end{aligned}$$

3.3 Using today's date (MMYY) for the seed value, solve 10 possibly degenerate problems using the online pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/lexico.html .

3.4 Consider the linear programming problems whose right-hand sides are identically zero:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \le 0 \qquad i = 1, 2, \ldots, m \\
& x_j \ge 0 \qquad j = 1, 2, \ldots, n.
\end{aligned}$$

Show that either $x_j = 0$ for all j is optimal or else the problem is unbounded.

3.5 Consider the following linear program:

$$\begin{array}{lrl} \text{maximize} & x_1 + 3x_2 & \\ \text{subject to} & -2x_1 & \leq -5 \\ & x_1 & \geq 0\,. \end{array}$$

Show that this problem has feasible solutions but no vertex solutions. How does this reconcile with the fundamental theorem of linear programming (Theorem 3.4)?

3.6 Suppose that a linear programming problem has the following property: its initial dictionary is not degenerate and, when solved by the simplex method, there is never a tie for the choice of leaving variable.

(a) Can such a problem have degenerate dictionaries? Explain.
(b) Can such a problem cycle? Explain.

3.7 Consider the following dictionary:

$$\begin{aligned} \zeta &= 5 + 2x_2 - 2x_3 + 3x_5 \\ \hline x_6 &= 4 - 2x_2 - x_3 + x_5 \\ x_4 &= 2 - x_2 + x_3 - x_5 \\ x_1 &= 6 - 2x_2 - 2x_3 - 3x_5. \end{aligned}$$

(a) List all pairs (x_r, x_s) such that x_r could be the entering variable and x_s could be the leaving variable.
(b) List all such pairs if the largest-coefficient rule for choosing the entering variable is used.
(c) List all such pairs if Bland's rule for choosing the entering and leaving variables is used.

Notes

The first example of cycling was given by Hoffman (1953). The fact that any linear programming problem that cycles must have at least six variables and three constraints was proved by Marshall & Suurballe (1969).

Early proofs of the fundamental theorem of linear programming (Theorem 3.4) were constructive, relying, as in our development, on the existence of a variant of the simplex method that works even in the presense of degeneracy. Hence, finding such variants occupied the attention of early researchers in linear programming. The *perturbation method* was first suggested by A. Orden and developed independently

by Charnes (1952). The essentially equivalent *lexicographic method* first appeared in Dantzig et al. (1955). Theorem 3.3 was proved by Bland (1977).

For an extensive treatment of degeneracy issues see Gal (1993).

CHAPTER 4

Efficiency of the Simplex Method

In the previous chapter, we saw that the simplex method (with appropriate pivoting rules to guarantee no cycling) will solve any linear programming problem for which an optimal solution exists. In this chapter, we investigate just how fast it will solve a problem of a given size.

1. Performance Measures

Performance measures can be broadly divided into two types:

- worst case
- average case.

As its name implies, a worst-case analysis looks at all problems of a given "size" and asks how much effort is needed to solve the hardest of these problems. Similarly, an average-case analysis looks at the average amount of effort, averaging over all problems of a given size. Worst-case analyses are generally easier than average-case analyses. The reason is that, for worst-case analyses, one simply needs to give an upper bound on how much effort is required and then exhibit a specific example that attains this bound. However, for average-case analyses, one must have a stochastic model of the space of "random linear programming problems" and then be able to say something about the solution effort averaged over all the problems in the sample space. There are two serious difficulties here. The first is that it is not clear at all how one should model the space of random problems. Secondly, given such a model, one must be able to evaluate the amount of effort required to solve every problem in the sample space.

Therefore, worst-case analysis is more tractable than average-case analysis, but it is also less relevant to a person who needs to solve real problems. In this chapter, we shall give a worst-case analysis of the simplex method. Later, in Chapter 12, we shall present results of empirical studies that indicate the average behavior over finite sets of real problems. Such studies act as a surrogate for a true average-case analysis.

2. Measuring the Size of a Problem

Before looking at worst cases, we must discuss two issues. First, how do we specify the size of a problem? Two parameters come naturally to mind: m and n.

Usually, we shall simply use these two numbers to characterize the size a problem. However, we should mention some drawbacks associated with this choice. First of all, it would be preferable to use only one number to indicate size. Since the data for a problem consist of the constraint coefficients together with the right-hand side and objective function coefficients, perhaps we should use the total number of data elements, which is roughly mn.

The product mn isn't bad, but what if many or even most of the data elements are zero? Wouldn't one expect such a problem to be easier to solve? Efficient implementations do indeed take advantage of the presence of lots of zeros, and so an analysis should also account for this. Hence, a good measure might be simply the number of nonzero data elements. This would definitely be an improvement, but one can go further. On a computer, floating-point numbers are all the same size and can be multiplied in the same amount of time. But if a person is to solve a problem by hand (or use unlimited precision computation on a computer), then certainly multiplying 23 by 7 is a lot easier than multiplying 23453.2352 by 86833.245643. So perhaps the best measure of a problem's size is not the number of data elements, but the actual number of bits needed to store all the data on a computer. This measure is popular among most computer scientists and is usually denoted by L.

However, with a little further abstraction, the size of the data, L, is seen to be ambiguous. As we saw in Chapter 1, real-world problems, while generally large and sparse, usually can be described quite simply and involve only a small amount of true input data that gets greatly expanded when setting the problem up with a constraint matrix, right-hand side, and objective function. So should L represent the number of bits needed to specify the nonzero constraint coefficients, objective coefficients, and right-hand sides, or should it be the number of bits in the original data set plus the number of bits in the description of how this data represents a linear programming problem? No one currently uses this last notion of problem size, but it seems fairly reasonable that they should (or at least that they should seriously consider it). Anyway, our purpose here is merely to mention that these important issues are lurking about, but, as stated above, we shall simply focus on m and n to characterize the size of a problem.

3. Measuring the Effort to Solve a Problem

The second issue to discuss is how one should measure the amount of work required to solve a problem. The best answer is the number of seconds of computer time required to solve the problem, using the computer sitting on one's desk. Unfortunately, there are (hopefully) many readers of this text, not all of whom use the exact same computer. Even if they did, computer technology changes rapidly, and a few years down the road everyone would be using something entirely different. It would be nice if the National Institute of Standards and Technology (the government organization in charge of setting standards, such as how many threads/inch a standard

light bulb should have) would identify a standard computer for the purpose of benchmarking algorithms, but, needless to say, this is not very likely. So the time needed to solve a problem, while the most desirable measure, is not the most practical one here. Fortunately, there is a fairly reasonable substitute. Algorithms are generally iterative processes, and the time to solve a problem can be factored into the number of iterations required to solve the problem times the amount of time required to do each iteration. The first factor, the number of iterations, does not depend on the computer and so is a reasonable surrogate for the actual time. This surrogate is useful when comparing various algorithms within the same general class of algorithms, in which the time per iteration can be expected to be about the same among the algorithms; however, it becomes meaningless when one wishes to compare two entirely different algorithms. For now, we shall measure the amount of effort to solve a linear programming problem by counting the number of iterations needed to solve it.

4. Worst-Case Analysis of the Simplex Method

How bad can the simplex method be in the worst case? Well, we have already seen that for some pivoting rules it can cycle, and hence the worst-case solution time for such variants is infinite. However, what about noncycling variants of the simplex method? Since the simplex method operates by moving from one basic feasible solution to another without ever returning to a previously visited solution, an upper bound on the number of iterations is simply the number of basic feasible solutions, of which there can be at most

$$\binom{n+m}{m}.$$

For a fixed value of the sum $n+m$, this expression is maximized when $m=n$. And how big is it? It is not hard to show that

$$\frac{1}{2n}2^{2n} \leq \binom{2n}{n} \leq 2^{2n}$$

(see Exercise 4.9). It should be noted that, even though typographically compact, the expression 2^n is huge even when n is not very big. For example, $2^{50} = 1.1259 \times 10^{15}$.

Our best chance for finding a bad example is to look at the case where $m=n$. We shall now give an example, first discovered by V. Klee and G.J. Minty in 1972, in which the simplex method using the largest coefficient rule requires $2^n - 1$ iterations

to solve. The example is quite simple to state:

$$
\begin{aligned}
&\text{maximize} && \sum_{j=1}^{n} 10^{n-j} x_j \\
&\text{subject to} && 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le 100^{i-1} && i = 1, 2, \ldots, n \\
& && x_j \ge 0 && j = 1, 2, \ldots, n.
\end{aligned} \tag{4.1}
$$

It is instructive to look more closely at the constraints. The first three constraints are

$$
\begin{aligned}
x_1 &\le 1 \\
20x_1 + x_2 &\le 100 \\
200x_1 + 20x_2 + x_3 &\le 10000.
\end{aligned}
$$

The first constraint simply says that x_1 is no bigger than one. With this in mind, the second constraint says that x_2 has an upper bound of about 100, depending on how big x_1 is. Similarly, the third constraint says that x_3 is roughly no bigger than 10,000 (again, this statement needs some adjustment depending on the sizes of x_1 and x_2). Therefore, the constraints are approximately just a set of upper bounds, which means that the feasible region is virtually an n-dimensional hypercube:

$$
\begin{aligned}
0 &\le x_1 \le 1 \\
0 &\le x_2 \le 100 \\
&\vdots \\
0 &\le x_n \le 100^{n-1}.
\end{aligned}
$$

For this reason, the feasible region for the Klee–Minty problem is often referred to as the Klee–Minty cube. An n-dimensional hypercube has 2^n vertices, and, as we shall see, the simplex method with the largest-coefficient rule will start at one of these vertices and visit every vertex before finally finding the optimal solution.

In order to understand the Klee–Minty problem, let us begin by replacing the specific right-hand sides, 100^{i-1}, with more generic values, b_i, with the property that

$$1 = b_1 \ll b_2 \ll \cdots \ll b_n.$$

As in the previous chapter, we use the expression $a \ll b$ to mean that a is so much smaller than b that no factors multiplying a and dividing b that arise in the course of applying the simplex method to the problem at hand can ever make the resulting a as large as the resulting b. Hence, we can think of the b_i's as independent variables

for now (specific values can be chosen later). Next, it is convenient to change each right-hand side replacing b_i with

$$\sum_{j=1}^{i-1} 10^{i-j} b_j + b_i.$$

Since the numbers b_j, $j = 1, 2, \ldots, i-1$ are "small potatoes" compared with b_i, this modification to the right-hand sides amounts to a very small perturbation. The right-hand sides still grow by huge amounts as i increases. Finally, we wish to add a constant to the objective function so that the Klee–Minty problem can finally be written as

$$\begin{aligned} &\text{maximize} && \sum_{j=1}^{n} 10^{n-j} x_j - \frac{1}{2} \sum_{j=1}^{n} 10^{n-j} b_j \\ &\text{subject to} && 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le \sum_{j=1}^{i-1} 10^{i-j} b_j + b_i \qquad i = 1, 2, \ldots, n \\ & && x_j \ge 0 \qquad\qquad\qquad\qquad\quad j = 1, 2, \ldots, n. \end{aligned} \tag{4.2}$$

In Exercise 4.7, you are asked to prove that this problem takes $2^n - 1$ iterations. To start to get a handle on the proof, here are the seven iterations that one gets with $n = 3$. The initial dictionary is

$$\begin{aligned} \zeta &= -\tfrac{100}{2} b_1 - \tfrac{10}{2} b_2 - \tfrac{1}{2} b_3 + 100 x_1 + 10 x_2 + x_3 \\ \hline w_1 &= \quad b_1 \qquad\qquad\quad - \quad x_1 \\ w_2 &= \quad 10 b_1 + \quad b_2 \qquad - \; 20 x_1 - \quad x_2 \\ w_3 &= \quad 100 b_1 + 10 b_2 + \quad b_3 - 200 x_1 - 20 x_2 - x_3, \end{aligned}$$

which is feasible. Using the largest coefficient rule, the entering variable is x_1. From the fact that each subsequent b_i is huge compared with its predecessor it follows that w_1 is the leaving variable. After the first iteration, the dictionary reads

$$\begin{aligned} \zeta &= \quad \tfrac{100}{2} b_1 - \tfrac{10}{2} b_2 - \tfrac{1}{2} b_3 - 100 w_1 + 10 x_2 + x_3 \\ \hline x_1 &= \quad b_1 \qquad\qquad\quad - \quad w_1 \\ w_2 &= -10 b_1 + \quad b_2 \qquad + \; 20 w_1 - \quad x_2 \\ w_3 &= -100 b_1 + 10 b_2 + \quad b_3 + 200 w_1 - 20 x_2 - x_3. \end{aligned}$$

Now, x_2 enters and w_2 leaves, so after the second iteration we get:

$$\begin{aligned}
\zeta &= -\tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 + 100w_1 - 10w_2 + x_3 \\
\hline
x_1 &= \quad b_1 \qquad\qquad - \quad w_1 \\
x_2 &= -10b_1 + \quad b_2 \qquad + \ 20w_1 - \quad w_2 \\
w_3 &= \ 100b_1 - 10b_2 + \ \ b_3 - 200w_1 + 20w_2 - x_3.
\end{aligned}$$

After the third iteration

$$\begin{aligned}
\zeta &= \ \ \tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 - 100x_1 - 10w_2 + x_3 \\
\hline
w_1 &= \quad b_1 \qquad\qquad - \quad x_1 \\
x_2 &= \ \ 10b_1 + \quad b_2 \qquad - \ 20x_1 - \quad w_2 \\
w_3 &= -100b_1 - 10b_2 + \ \ b_3 + 200x_1 + 20w_2 - x_3.
\end{aligned}$$

After the fourth iteration

$$\begin{aligned}
\zeta &= -\tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 + 100x_1 + 10w_2 - w_3 \\
\hline
w_1 &= \quad b_1 \qquad\qquad - \quad x_1 \\
x_2 &= \ \ 10b_1 + \quad b_2 \qquad - \ 20x_1 - \quad w_2 \\
x_3 &= -100b_1 - 10b_2 + \ \ b_3 + 200x_1 + 20w_2 - w_3.
\end{aligned}$$

After the fifth iteration

$$\begin{aligned}
\zeta &= \ \ \tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 - 100w_1 + 10w_2 - w_3 \\
\hline
x_1 &= \quad b_1 \qquad\qquad - \quad w_1 \\
x_2 &= -10b_1 + \quad b_2 \qquad + \ 20w_1 - \quad w_2 \\
x_3 &= \ 100b_1 - 10b_2 + \ \ b_3 - 200w_1 + 20w_2 - w_3.
\end{aligned}$$

After the sixth iteration

$$\begin{aligned}
\zeta &= -\tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 + 100w_1 - 10x_2 - w_3 \\
\hline
x_1 &= \quad b_1 \qquad\qquad - \quad w_1 \\
w_2 &= -10b_1 + \quad b_2 \qquad + \ 20w_1 - \quad x_2 \\
x_3 &= -100b_1 + 10b_2 + \ \ b_3 + 200w_1 - 20x_2 - w_3.
\end{aligned}$$

And, finally, after the seventh iteration, we get

$$\begin{aligned}
\zeta &= \tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 - 100x_1 - 10x_2 - w_3 \\
\hline
w_1 &= \quad b_1 \qquad\qquad\quad - \quad x_1 \\
w_2 &= 10b_1 + \quad b_2 \qquad - \; 20x_1 - \quad x_2 \\
x_3 &= 100b_1 + 10b_2 + \quad b_3 - 200x_1 - 20x_2 - w_3,
\end{aligned}$$

which is, of course, optimal.

A few observations should be made. First, every pivot is the swap of an x_j with the corresponding w_j. Second, every dictionary looks just like the first one with the exception that the w_i's and the x_i's have become intertwined and various signs have changed (see Exercise 4.6).

Also note that the final dictionary could have been reached from the initial dictionary in just one pivot if we had selected x_3 to be the entering variable. But the largest-coefficient rule dictated selecting x_1. It is natural to wonder whether the largest-coefficient rule could be replaced by some other pivot rule for which the worst-case behavior would be much better than the 2^n behavior of the largest-coefficient rule. So far no one has found such a pivot rule. However, no one has proved that such a rule does not exist either.

Finally, we mention that one desirable property of an algorithm is that it be scale invariant. This means that should the units in which one measures the decision variables in a problem be changed, the algorithm would still behave in exactly the same manner. The simplex method with the largest-coefficient rule is not scale invariant. To see this, consider changing variables in the Klee–Minty problem by putting

$$\bar{x}_j = 100^{j-1}x_j.$$

In the new variables, the initial dictionary for the $n = 3$ Klee–Minty problem becomes

$$\begin{aligned}
\zeta &= -\tfrac{100}{2}b_1 - \quad \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 + 100\bar{x}_1 + 1000\bar{x}_2 + 10000\bar{x}_3 \\
\hline
w_1 &= \quad b_1 \qquad\qquad\quad - \quad \bar{x}_1 \\
w_2 &= \quad 10b_1 + \quad b_2 \qquad - \; 20\bar{x}_1 - \quad \bar{x}_2 \\
w_3 &= \quad 100b_1 + 1000b_2 + \quad b_3 - 200\bar{x}_1 - 2000\bar{x}_2 - 10000\bar{x}_3.
\end{aligned}$$

Now, the largest-coefficient rule picks variable x_3 to enter. Variable w_3 leaves, and the method steps to the optimal solution in just one iteration. There exist pivot rules for the simplex method that are scale invariant. But Klee–Minty-like examples have been found for most proposed alternative pivot rules (whether scale invariant or not). In fact, it is an open question whether there exist pivot rules for which one can prove that no problem instance requires an exponential number of iterations (as a function of m or n).

Exercises

In solving the following problems, the simple pivot tool can be used to check your arithmetic:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/simple.html

4.1 Compare the performance of the largest-coefficient and the smallest-index pivoting rules on the following linear program:

$$\begin{aligned} \text{maximize} \quad & 4x_1 + 5x_2 \\ \text{subject to} \quad & 2x_1 + 2x_2 \leq 9 \\ & x_1 \leq 4 \\ & x_2 \leq 3 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

4.2 Compare the performance of the largest-coefficient and the smallest-index pivoting rules on the following linear program:

$$\begin{aligned} \text{maximize} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & 3x_1 + x_2 \leq 3 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

4.3 Compare the performance of the largest-coefficient and the smallest-index pivoting rules on the following linear program:

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 5x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 5 \\ & x_1 \leq 3 \\ & x_2 \leq 2 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

4.4 Solve the Klee–Minty problem (4.1) for $n = 3$.

4.5 Solve the 4 variable Klee-Minty problem using the online pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/kleeminty.html

4.6 Consider the dictionary

$$\begin{aligned}\zeta &= -\sum_{j=1}^{n} \epsilon_j 10^{n-j} \left(\frac{1}{2} b_j - x_j\right) \\ w_i &= \sum_{j=1}^{i-1} \epsilon_i \epsilon_j 10^{i-j} (b_j - 2x_j) + (b_i - x_i) \qquad i = 1, 2, \ldots, n,\end{aligned}$$

where the b_i's are as in the Klee–Minty problem (4.2) and where each ϵ_i is ± 1. Fix k and consider the pivot in which x_k enters the basis and w_k leaves the basis. Show that the resulting dictionary is of the same form as before. How are the new ϵ_i's related to the old ϵ_i's?

4.7 Use the result of the previous problem to show that the Klee–Minty problem (4.2) requires $2^n - 1$ iterations.

4.8 Consider the Klee–Minty problem (4.2). Suppose that $b_i = \beta^{i-1}$ for some $\beta > 1$. Find the greatest lower bound on the set of β's for which the this problem requires $2^n - 1$ iterations.

4.9 Show that, for any integer n,

$$\frac{1}{2n} 2^{2n} \leq \binom{2n}{n} \leq 2^{2n}.$$

4.10 Consider a linear programming problem that has an optimal dictionary in which exactly k of the original slack variables are nonbasic. Show that by ignoring feasibility preservation of intermediate dictionaries this dictionary can be arrived at in exactly k pivots. Don't forget to allow for the fact that some pivot elements might be zero. *Hint: see Exercise 2.15.*

Notes

The first example of a linear programming problem in n variables and n constraints taking $2^n - 1$ iterations to solve was published by Klee & Minty (1972). Several researchers, including Smale (1983), Borgwardt (1982), Borgwardt (1987*a*), Adler & Megiddo (1985), and Todd (1986), have studied the average number of iterations. For a survey of probabilistic methods, the reader should consult Borgwardt (1987*b*).

Roughly speaking, a class of problems is said to have *polynomial complexity* if there is a polynomial p for which every problem of "size" n in the class can be solved by some algorithm in at most $p(n)$ operations. For many years it was unknown whether linear programming had polynomial complexity. The Klee–Minty examples

show that, if linear programming is polynomial, then the simplex method is not the algorithm that gives the polynomial bound, since 2^n is not dominated by any polynomial. In 1979, Khachian (1979) gave a new algorithm for linear programming, called the *ellipsoid method*, which *is* polynomial and therefore established once and for all that linear programming has polynomial complexity. The collection of all problem classes having polynomial complexity is usually denoted by $\mathcal{P}$. A class of problems is said to belong to the class $\mathcal{NP}$ if, given a (proposed) solution, one can verify its optimality in a number of operations that is bounded by some polynomial in the "size" of the problem. Clearly, $\mathcal{P} \subset \mathcal{NP}$ (since, if we can solve from scratch in a polynomial amount of time, surely we can verify optimality at least that fast). An important problem in theoretical computer science is to determine whether or not $\mathcal{P}$ is a strict subset of $\mathcal{NP}$.

The study of how difficult it is to solve a class of problems is called *complexity theory*. Readers interested in pursuing this subject further should consult Garey & Johnson (1977).

CHAPTER 5

Duality Theory

Associated with every linear program is another called its dual. The dual of this dual linear program is the original linear program (which is then referred to as the primal linear program). Hence, linear programs come in primal/dual pairs. It turns out that every feasible solution for one of these two linear programs gives a bound on the optimal objective function value for the other. These ideas are important and form a subject called duality theory, which is the topic we shall study in this chapter.

1. Motivation—Finding Upper Bounds

We begin with an example:

$$\begin{aligned} \text{maximize } & 4x_1 + x_2 + 3x_3 \\ \text{subject to } & x_1 + 4x_2 \leq 1 \\ & 3x_1 - x_2 + x_3 \leq 3 \\ & x_1,\ x_2,\ x_3 \geq 0. \end{aligned}$$

Our first observation is that every feasible solution provides a lower bound on the optimal objective function value, ζ^*. For example, the solution $(x_1, x_2, x_3) = (1, 0, 0)$ tells us that $\zeta^* \geq 4$. Using the feasible solution $(x_1, x_2, x_3) = (0, 0, 3)$, we see that $\zeta^* \geq 9$. But how good is this bound? Is it close to the optimal value? To answer, we need to give upper bounds, which we can find as follows. Let's multiply the first constraint by 2 and add that to 3 times the second constraint:

$$\begin{aligned} 2\ (x_1 + 4x_2 \quad\quad) &\leq 2(1) \\ +3\,(3x_1 - x_2 + x_3) &\leq 3(3) \\ \hline 11x_1 + 5x_2 + 3x_3 &\leq 11. \end{aligned}$$

Now, since each variable is nonnegative, we can compare the sum against the objective function and notice that

$$4x_1 + x_2 + 3x_3 \leq 11x_1 + 5x_2 + 3x_3 \leq 11.$$

Hence, $\zeta^* \leq 11$. We have localized the search to somewhere between 9 and 11. These bounds leave a gap (within which the optimal solution lies), but they are better than nothing. Furthermore, they can be improved. To get a better upper bound, we again apply the same upper bounding technique, but we replace the specific numbers we used before with variables and then try to find the values of those variables that give us the best upper bound. So we start by multiplying the two constraints by nonnegative numbers, y_1 and y_2, respectively. The fact that these numbers are nonnegative implies that they preserve the direction of the inequalities. Hence,

$$\begin{array}{rr} y_1(\ x_1 + 4x_2 \qquad) \leq & y_1 \\ +y_2(3x_1 - \ x_2 + x_3) \leq & 3y_2 \\ \hline (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + (y_2)x_3 \leq y_1 + 3y_2. & \end{array}$$

If we stipulate that each of the coefficients of the x_i's be at least as large as the corresponding coefficient in the objective function,

$$\begin{aligned} y_1 + 3y_2 &\geq 4 \\ 4y_1 - \ y_2 &\geq 1 \\ y_2 &\geq 3, \end{aligned}$$

then we can compare the objective function against this sum (and its bound):

$$\begin{aligned} \zeta &= 4x_1 + x_2 + 3x_3 \\ &\leq (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + (y_2)x_3 \\ &\leq y_1 + 3y_2. \end{aligned}$$

We now have an upper bound, $y_1 + 3y_2$, which we should minimize in our effort to obtain the best possible upper bound. Therefore, we are naturally led to the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & y_1 + 3y_2 \\ \text{subject to} \quad & y_1 + 3y_2 \geq 4 \\ & 4y_1 - \ y_2 \geq 1 \\ & y_2 \geq 3 \\ & y_1,\ y_2 \geq 0. \end{aligned}$$

This problem is called the dual linear programming problem associated with the given linear programming problem. In the next section, we will define the dual linear programming problem in general.

2. The Dual Problem

Given a linear programming problem in standard form,

$$\begin{aligned} &\text{maximize } \sum_{j=1}^{n} c_j x_j \\ &\text{subject to } \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i = 1, 2, \ldots, m \\ &\qquad\qquad x_j \ge 0 \qquad\quad j = 1, 2, \ldots, n, \end{aligned} \tag{5.1}$$

the associated *dual linear program* is given by

$$\begin{aligned} &\text{minimize } \sum_{i=1}^{m} b_i y_i \\ &\text{subject to } \sum_{i=1}^{m} y_i a_{ij} \ge c_j \qquad j = 1, 2, \ldots, n \\ &\qquad\qquad y_i \ge 0 \qquad\quad i = 1, 2, \ldots, m. \end{aligned}$$

Since we started with (5.1), it is called the *primal problem*. Our first order of business is to show that taking the dual of the dual returns us to the primal. To see this, we first must write the dual problem in standard form. That is, we must change the minimization into a maximization and we must change the first set of greater-than-or-equal-to constraints into less-than-or-equal-to. Of course, we must effect these changes without altering the problem. To change a minimization into a maximization, we note that to minimize something it is equivalent to maximize its negative and then negate the answer:

$$\min \sum_{i=1}^{m} b_i y_i = -\max \left(-\sum_{i=1}^{m} b_i y_i \right).$$

To change the direction of the inequalities, we simply multiply through by minus one. The resulting equivalent representation of the dual problem in standard form then is

$$\begin{aligned} &-\text{maximize } \sum_{i=1}^{m} (-b_i) y_i \\ &\text{subject to } \sum_{i=1}^{m} (-a_{ij}) y_i \le (-c_j) \qquad j = 1, 2, \ldots, n \\ &\qquad\qquad y_i \ge 0 \qquad\quad i = 1, 2, \ldots, m. \end{aligned}$$

Now we can take its dual:

$$\begin{aligned} -\text{minimize } & \sum_{j=1}^{n}(-c_j)x_j \\ \text{subject to } & \sum_{j=1}^{n}(-a_{ij})x_j \geq (-b_i) \qquad i = 1, 2, \ldots, m \\ & x_j \geq 0 \qquad\qquad\quad j = 1, 2, \ldots, n, \end{aligned}$$

which is clearly equivalent to the primal problem as formulated in (5.1).

3. The Weak Duality Theorem

As we saw in our example, the dual problem provides upper bounds for the primal objective function value. This result is true in general and is referred to as the *Weak Duality Theorem*:

THEOREM 5.1. *If $(x_1, x_2, \ldots, x_n)$ is feasible for the primal and $(y_1, y_2, \ldots, y_m)$ is feasible for the dual, then*

$$\sum_j c_j x_j \leq \sum_i b_i y_i.$$

PROOF. The proof is a simple chain of obvious inequalities:

$$\begin{aligned} \sum_j c_j x_j &\leq \sum_j \left(\sum_i y_i a_{ij} \right) x_j \\ &= \sum_{ij} y_i a_{ij} x_j \\ &= \sum_i \left(\sum_j a_{ij} x_j \right) y_i \\ &\leq \sum_i b_i y_i, \end{aligned}$$

where the first inequality follows from the fact that each x_j is nonnegative and each c_j is no larger than $\sum_i y_i a_{ij}$. The second inequality, of course, holds for similar reasons. □

Consider the subset of the real line consisting of all possible values for the primal objective function, and consider the analogous subset associated with the dual problem. The weak duality theorem tells us that the set of primal values lies entirely to

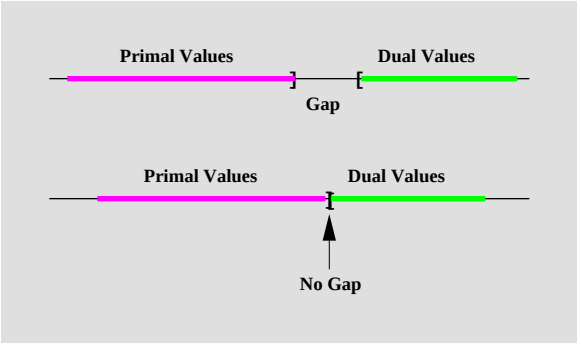


FIGURE 5.1. The primal objective values are all less than the dual objective values. An important question is whether or not there is a gap between the largest primal value and the smallest dual value.

the left of the set of dual values. As we shall see shortly, these sets are both closed intervals (perhaps of infinite extent), and the right endpoint of the primal set butts up against the left endpoint of the dual set (see Figure 5.1). That is, there is no gap between the optimal objective function value for the primal and for the dual. The lack of a gap between primal and dual objective values provides a convenient tool for verifying optimality. Indeed, if we can exhibit a feasible primal solution $(x_1^*, x_2^*, \ldots, x_n^*)$ and a feasible dual solution $(y_1^*, y_2^*, \ldots, y_m^*)$ for which

$$\sum_j c_j x_j^* = \sum_i b_i y_i^*,$$

then we may conclude that each of these solutions is optimal for its respective problem. To see that the primal solution is optimal, consider any other feasible solution $(x_1, x_2, \ldots, x_n)$. By the weak duality theorem, we have that

$$\sum_j c_j x_j \leq \sum_i b_i y_i^* = \sum_j c_j x_j^*.$$

Now, since $(x_1^*, x_2^*, \ldots, x_n^*)$ was assumed to be feasible, we see that it must be optimal. An analogous argument shows that the dual solution is also optimal for the dual problem. As an example, consider the solutions $x = (0, 0.25, 3.25)$ and $y = (1, 3)$ in our example. Both these solutions are feasible, and both yield an objective value of 10. Hence, the weak duality theorem says that these solutions are optimal.

4. The Strong Duality Theorem

The fact that for linear programming there is never a gap between the primal and the dual optimal objective values is usually referred to as the *Strong Duality Theorem*:

THEOREM 5.2. *If the primal problem has an optimal solution,*

$$x^* = (x_1^*, x_2^*, \ldots, x_n^*),$$

then the dual also has an optimal solution,

$$y^* = (y_1^*, y_2^*, \ldots, y_m^*),$$

such that

$$\sum_j c_j x_j^* = \sum_i b_i y_i^*. \tag{5.2}$$

Carefully written proofs, while attractive for their tightness, sometimes obfuscate the main idea. In such cases, it is better to illustrate the idea with a simple example. Anyone who has taken a course in linear algebra probably already appreciates such a statement. In any case, it is true here as we explain the strong duality theorem.

The main idea that we wish to illustrate here is that, as the simplex method solves the primal problem, it also implicitly solves the dual problem, and it does so in such a way that (5.2) holds.

To see what we mean, let us return to the example discussed in Section 5.1. We start by introducing variables w_i, $i = 1, 2$, for the primal slacks and z_j, $j = 1, 2, 3$, for the dual slacks. Since the inequality constraints in the dual problem are greater-than constraints, each dual slack is defined as a left-hand side minus the corresponding right-hand side. For example,

$$z_1 = y_1 + 3y_2 - 4.$$

Therefore, the primal and dual dictionaries are written as follows:

(P)
$$\begin{aligned} \zeta &= \quad 4x_1 + \ x_2 + 3x_3 \\ \hline w_1 &= 1 - \ x_1 - 4x_2 \\ w_2 &= 3 - 3x_1 + \ x_2 - \ x_3. \end{aligned}$$

(D)
$$\begin{aligned} -\xi &= \quad - \ y_1 - 3y_2 \\ \hline z_1 &= -4 + \ y_1 + 3y_2 \\ z_2 &= -1 + 4y_1 - \ y_2 \\ z_3 &= -3 \quad\quad + \ y_2. \end{aligned}$$

Note that we have recorded the negative of the dual objective function, since we prefer to maximize the objective function appearing in a dictionary. Also note that the numbers in the dual dictionary are simply the negative of the numbers in the primal dictionary arranged with the rows and columns interchanged. Indeed, stripping away everything but the numbers, we have

$$\begin{bmatrix} 0 & 4 & 1 & 3 \\ 1 & -1 & -4 & 0 \\ 3 & -3 & 1 & -1 \end{bmatrix} \overset{\text{neg.–transp.}}{\longleftrightarrow} \begin{bmatrix} 0 & -1 & -3 \\ -4 & 1 & 3 \\ -1 & 4 & -1 \\ -3 & 0 & 1 \end{bmatrix}.$$

That is, as a table of numbers, the dual dictionary is the *negative transpose* of the primal dictionary.

Our goal now is to apply the simplex method to the primal problem and at the same time perform the analogous pivots on the dual problem. We shall discover that the negative-transpose property persists throughout the iterations.

Since the primal dictionary is feasible, no Phase I procedure is necessary. For the first pivot, we pick x_3 as the entering variable (x_1 has the largest coefficient, but x_3 provides the greatest one-step increase in the objective). With this choice, the leaving variable must be w_2. Since the rows and columns are interchanged in the dual dictionary, we see that "column" x_3 in the primal dictionary corresponds to "row" z_3 in the dual dictionary. Similarly, row w_2 in the primal corresponds to column y_2 in the dual. Hence, to make an analogous pivot in the dual dictionary, we select y_2 as the entering variable and z_3 as the leaving variable. While this choice of entering and leaving variable may seem odd compared to how we have chosen entering and leaving variables before, we should note that our earlier choice was guided by the desire to increase the objective function while preserving feasibility. Here, the dual dictionary is not even feasible, and so such considerations are meaningless. Once we give up those rules for the choice of entering and leaving variables, it is easy to see that a pivot can be performed with any choice of entering and leaving variables provided only that the coefficient on the entering variable in the constraint of the leaving variables does not vanish. Such is the case with the current choice. Hence, we do the pivot in both

the primal and the dual. The result is

$$\text{(P)} \qquad \begin{aligned} \zeta &= 9 - 5x_1 + 4x_2 - 3w_2 \\ \hline w_1 &= 1 - x_1 - 4x_2 \\ x_3 &= 3 - 3x_1 + x_2 - w_2. \end{aligned}$$

$$\text{(D)} \qquad \begin{aligned} -\xi &= -9 - y_1 - 3z_3 \\ \hline z_1 &= 5 + y_1 + 3z_3 \\ z_2 &= -4 + 4y_1 - z_3 \\ y_2 &= 3 + z_3. \end{aligned}$$

Note that these two dictionaries still have the property of being negative-transposes of each other. For the next pivot, the entering variable in the primal dictionary is x_2 (this time there is no choice) and the leaving variable is w_1. In the dual dictionary, the corresponding entering variable is y_1 and the leaving variable is z_2. Doing the pivots, we get

$$\text{(P)} \qquad \begin{aligned} \zeta &= 10 - 6x_1 - w_1 - 3w_2 \\ \hline x_2 &= 0.25 - 0.25x_1 - 0.25w_1 \\ x_3 &= 3.25 - 3.25x_1 - 0.25w_1 - w_2. \end{aligned}$$

$$\text{(D)} \qquad \begin{aligned} -\xi &= -10 - 0.25z_2 - 3.25z_3 \\ \hline z_1 &= 6 + 0.25z_2 + 3.25z_3 \\ y_1 &= 1 + 0.25z_2 + 0.25z_3 \\ y_2 &= 3 + z_3. \end{aligned}$$

This primal dictionary is optimal, since the coefficients in the objective row are all negative. Looking at the dual dictionary, we see that it is now feasible for the analogous reason. In fact, it is optimal too. Finally, both the primal and dual objective function values are 10.

The situation should now be clear. Given a linear programming problem, which is assumed to possess an optimal solution, first apply the Phase I procedure to get a basic feasible starting dictionary for Phase II. Then apply the simplex method to find an optimal solution. Each primal dictionary generated by the simplex method implicitly defines a corresponding dual dictionary as follows: first write down the negative transpose and then replace each x_j with a z_j and each w_i with a y_i. As long as the primal dictionary is not optimal, the implicitly defined dual dictionary will be infeasible. But once an optimal primal dictionary is found, the corresponding dual dictionary will be feasible. Since its objective coefficients are always nonpositive, this feasible dual

dictionary is also optimal. Furthermore, at each iteration, the current primal objective function value coincides with the current dual objective function value.

To see why the negative transpose property is preserved from one dictionary to the next, let's observe the effect of one pivot. To keep notations uncluttered, we shall consider only four generic entries in a table of coefficients: the pivot element, which we shall denote by a, one other element on the pivot element's row, call it b, one other in its column, call it c, and a fourth element, denoted d, chosen to make these four entries into a rectangle. A little thought (and perhaps some staring at the examples above) reveals that a pivot produces the following changes:

- the pivot element gets replaced by its reciprocal;
- elements in the pivot row get negated and divided by the pivot element;
- elements in the pivot column get divided by the pivot element; and
- all other elements, such as d, get decreased by bc/a.

These effects can be summarized on our generic table as follows:

b	a	
d	c	

$\xrightarrow{\text{pivot}}$

$-\frac{b}{a}$	$\frac{1}{a}$	
$d-\frac{bc}{a}$	$\frac{c}{a}$	

Now, if we start with a dual dictionary that is the negative transpose of the primal and apply one pivot operation, we get

	$-b$	$-d$
	$-a$	$-c$

$\xrightarrow{\text{pivot}}$

	$\frac{b}{a}$	$-d+\frac{bc}{a}$
	$-\frac{1}{a}$	$-\frac{c}{a}$

.

Note that the resulting dual table is the negative transpose of the resulting primal table. By induction we then conclude that, if we start with this property, it will be preserved throughout the solution process.

Since the strong duality theorem is the most important theorem in this book, we present here a careful proof. Those readers who are satisfied with the above discussion may skip the proof.

PROOF OF THEOREM 5.2. It suffices to exhibit a dual feasible solution y^* satisfying (5.2). Suppose we apply the simplex method. We know that the simplex method produces an optimal solution whenever one exists, and we have assumed that one does indeed exist. Hence, the final dictionary will be an optimal dictionary for the primal problem. The objective function in this final dictionary is ordinarily written as

$$\zeta = \bar{\zeta} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j.$$

But, since this is the optimal dictionary and we prefer stars to bars for denoting optimal "stuff," let us write ζ^* instead of $\bar{\zeta}$. Also, the collection of nonbasic variables will generally consist of a combination of original variables as well as slack variables. Instead of using $\bar{c}_j$ for the coefficients of these variables, let us use c_j^* for the objective coefficients corresponding to original variables, and let us use d_i^* for the objective coefficients corresponding to slack variables. Also, for those original variables that are basic we put $c_j^* = 0$, and for those slack variables that are basic we put $d_i^* = 0$. With these new notations, we can rewrite the objective function as

$$\zeta = \zeta^* + \sum_{j=1}^{n} c_j^* x_j + \sum_{i=1}^{m} d_i^* w_i.$$

As we know, ζ^* is the objective function value corresponding to the optimal primal solution:

$$\zeta^* = \sum_{j=1}^{n} c_j x_j^*. \tag{5.3}$$

Now, put

$$y_i^* = -d_i^*, \qquad i = 1, 2, \ldots, m. \tag{5.4}$$

We shall show that $y^* = (y_1^*, y_2^*, \ldots, y_m^*)$ is feasible for the dual problem and satisfies (5.2). To this end, we write the objective function two ways:

$$\begin{aligned}\sum_{j=1}^{n} c_j x_j &= \zeta^* + \sum_{j=1}^{n} c_j^* x_j + \sum_{i=1}^{m} d_i^* w_i \\ &= \zeta^* + \sum_{j=1}^{n} c_j^* x_j + \sum_{i=1}^{m} (-y_i^*) \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right) \\ &= \zeta^* - \sum_{i=1}^{m} b_i y_i^* + \sum_{j=1}^{n} \left(c_j^* + \sum_{i=1}^{m} y_i^* a_{ij} \right) x_j.\end{aligned}$$

Since all these expressions are linear in the variables x_j, we can equate the coefficients of each variable appearing on the left-hand side with the corresponding coefficient appearing in the last expression on the right-hand side. We can also equate the constant terms on the two sides. Hence,

$$\zeta^* = \sum_{i=1}^{m} b_i y_i^* \tag{5.5}$$

$$c_j = c_j^* + \sum_{i=1}^{m} y_i^* a_{ij}, \qquad j = 1, 2, \ldots, n. \tag{5.6}$$

Combining (5.3) and (5.5), we get that (5.2) holds. Also, the optimality of the dictionary for the primal problem implies that each c_j^* is nonpositive, and hence we see from (5.6) that

$$\sum_{i=1}^{m} y_i^* a_{ij} \geq c_j, \qquad j = 1, 2, \ldots, n.$$

By the same reasoning, each d_i^* is nonpositive, and so we see from (5.4) that

$$y_i^* \geq 0, \qquad i = 1, 2, \ldots, m.$$

These last two sets of inequalities are precisely the conditions that guarantee dual feasibility. This completes the proof. □

The strong duality theorem tells us that, whenever the primal problem has an optimal solution, the dual problem has one also and there is no duality gap. But what if the primal problem does not have an optimal solution? For example, suppose that it is unbounded. The unboundedness of the primal together with the weak duality theorem tells us immediately that the dual problem must be infeasible. Similarly, if the dual problem is unbounded, then the primal problem must be infeasible. It is

natural to hope that these three cases are the only possibilities, because if they were we could then think of the strong duality theorem holding globally. That is, even if, say, the primal is unbounded, the fact that then the dual is infeasible is like saying that the primal and dual have a zero duality gap sitting out at $+\infty$. Similarly, an infeasible primal together with an unbounded dual could be viewed as a pair in which the gap is zero and sits at $-\infty$.

But it turns out that there is a fourth possibility that sometimes occurs—it can happen that both the primal and the dual problems are infeasible. For example, consider the following problem:

$$\begin{aligned} \text{maximize} \quad & 2x_1 - x_2 \\ \text{subject to} \quad & x_1 - x_2 \leq 1 \\ & -x_1 + x_2 \leq -2 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

It is easy to see that both this problem and its dual are infeasible. For these problems, one can think of there being a huge duality gap extending from $-\infty$ to $+\infty$.

Duality theory is often useful in that it provides a *certificate of optimality*. For example, suppose that you were asked to solve a really huge and difficult linear program. After spending weeks or months at the computer, you are finally able to get the simplex method to solve the problem, producing as it does an optimal dual solution y^* in addition to the optimal primal solution x^*. Now, how are you going to convince your boss that your solution is correct? Do you really want to ask her to verify the correctness of your computer programs? The answer is probably not. And in fact it is not necessary. All you need to do is supply the primal and the dual solution, and she only has to check that the primal solution is feasible for the primal problem (that's easy), the dual solution is feasible for the dual problem (that's just as easy), and the primal and dual objective values agree (and that's even easier). Certificates of optimality have also been known to dramatically reduce the amount of time certain underpaid professors have to devote to grading homework assignments!

As we've seen, the simplex method applied to a primal problem actually solves both the primal and the dual. Since the dual of the dual is the primal, applying the simplex method to the dual also solves both the primal and the dual problem. Sometimes it is easier to apply the simplex method to the dual, for example, if the dual has an obvious basic feasible solution but the primal does not. We take up this topic in the next chapter.

5. Complementary Slackness

Sometimes it is necessary to recover an optimal dual solution when only an optimal primal solution is known. The following theorem, known as the *Complementary Slackness Theorem*, can help in this regard.

THEOREM 5.3. *Suppose that $x = (x_1, x_2, \ldots, x_n)$ is primal feasible and that $y = (y_1, y_2, \ldots, y_m)$ is dual feasible. Let $(w_1, w_2, \ldots, w_m)$ denote the corresponding primal slack variables, and let $(z_1, z_2, \ldots, z_n)$ denote the corresponding dual slack variables. Then x and y are optimal for their respective problems if and only if*

$$x_j z_j = 0, \qquad \text{for } j = 1, 2, \ldots, n,$$

$$w_i y_i = 0, \qquad \text{for } i = 1, 2, \ldots, m. \tag{5.7}$$

PROOF. We begin by revisiting the chain of inequalities used to prove the weak duality theorem:

$$\sum_j c_j x_j \leq \sum_j \left(\sum_i y_i a_{ij} \right) x_j \tag{5.8}$$

$$= \sum_i \left(\sum_j a_{ij} x_j \right) y_i$$

$$\leq \sum_i b_i y_i. \tag{5.9}$$

Recall that the first inequality arises from the fact that each term in the left-hand sum is dominated by the corresponding term in the right-hand sum. Furthermore, this domination is a consequence of the fact that each x_j is nonnegative and

$$c_j \leq \sum_i y_i a_{ij}.$$

Hence, inequality (5.8) will be an equality if and only if, for every $j = 1, 2, \ldots, n$, either $x_j = 0$ or $c_j = \sum_i y_i a_{ij}$. But since

$$z_j = \sum_i y_i a_{ij} - c_j,$$

we see that the alternative to $x_j = 0$ is simply that $z_j = 0$. Of course, the statement that at least one of these two numbers vanishes can be succinctly expressed by saying that the product vanishes.

An analogous analysis of inequality (5.9) shows that it is an equality if and only if (5.7) holds. This then completes the proof. □

Suppose that we have a nondegenerate primal basic optimal solution

$$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$$

and we wish to find a corresponding optimal solution for the dual. Let

$$w^* = (w_1^*, w_2^*, \ldots, w_m^*)$$

denote the corresponding slack variables, which were probably given along with the x_j's but if not can be easily obtained from their definition as slack variables:

$$w_i^* = b_i - \sum_j a_{ij} x_j^*.$$

The dual constraints are

$$\sum_i y_i a_{ij} - z_j = c_j, \qquad j = 1, 2, \ldots, n, \tag{5.10}$$

where we have written the inequalities in equality form by introducing slack variables z_j, $j = 1, 2, \ldots, n$. These constraints form n equations in $m + n$ unknowns. But the basic optimal solution (x^*, w^*) is a collection of $n + m$ variables, many of which are positive. In fact, since the primal solution is assumed to be nondegenerate, it follows that the m basic variables will be strictly positive. The complementary slackness theorem then tells us that the corresponding dual variables must vanish. Hence, of the $m + n$ variables in (5.10), we can set m of them to zero. We are then left with just n equations in n unknowns, which we would expect to have a unique solution that can be solved for. If there is a unique solution, all the components should be nonnegative. If any are negative, this would stand in contradiction to the assumed optimality of x^*.

6. The Dual Simplex Method

In this section, we study what happens if we apply the simplex method to the dual problem. As we saw in our discussion of the strong duality theorem, one can actually apply the simplex method to the dual problem without ever writing down the dual problem or its dictionaries. Instead, the so-called dual simplex method is seen simply as a new way of picking the entering and leaving variables in a sequence of primal dictionaries.

We begin with an example:

$$\begin{aligned}
\text{maximize} \quad & -x_1 - x_2 \\
\text{subject to} \quad & -2x_1 - x_2 \leq 4 \\
& -2x_1 + 4x_2 \leq -8 \\
& -x_1 + 3x_2 \leq -7 \\
& x_1,\ x_2 \geq 0.
\end{aligned}$$

The dual of this problem is

$$\begin{aligned} \text{minimize} \quad & 4y_1 - 8y_2 - 7y_3 \\ \text{subject to} \quad & -2y_1 - 2y_2 - \ y_3 \geq -1 \\ & -y_1 + 4y_2 + 3y_3 \geq -1 \\ & y_1,\ y_2,\ y_3 \geq \ 0. \end{aligned}$$

Introducing variables w_i, $i = 1, 2, 3$, for the primal slacks and z_j, $j = 1, 2$, for the dual slacks, we can write down the initial primal and dual dictionaries:

$$\begin{aligned} \text{(P)} \qquad \zeta &= \ - \ x_1 - \ x_2 \\ \hline w_1 &= \ 4 + 2x_1 + \ x_2 \\ w_2 &= -8 + 2x_1 - 4x_2 \\ w_3 &= -7 + \ x_1 - 3x_2 \end{aligned}$$

$$\begin{aligned} \text{(D)} \qquad -\xi &= \ -4y_1 + 8y_2 + 7y_3 \\ \hline z_1 &= 1 - 2y_1 - 2y_2 - \ y_3 \\ z_2 &= 1 - \ y_1 + 4y_2 + 3y_3. \end{aligned}$$

As before, we have recorded the negative of the dual objective function, since we prefer to maximize the objective function appearing in a dictionary. More importantly, note that the dual dictionary is feasible, whereas the primal one is not. This suggests that it would be sensible to apply the simplex method to the dual. Let us do so, but as we go we shall keep track of the analogous pivots applied to the primal dictionary. For example, the entering variable in the initial dual dictionary is y_2, and the leaving variable then is z_1. Since w_2 is complementary to y_2 and x_1 is complementary to z_1, we will use w_2 and x_1 as the entering/leaving variables in the primal dictionary. Of course, since w_2 is basic and x_1 is nonbasic, w_2 must be the leaving variable and x_1 the entering variable—i.e., the reverse of what we have for the complementary variables

in the dual dictionary. The result of these pivots is

$$\begin{array}{ll} \text{(P)} & \zeta = -4 - 0.5w_2 - 3x_2 \\ \hline & w_1 = 12 + w_2 + 5x_2 \\ & x_1 = 4 + 0.5w_2 + 2x_2 \\ & w_3 = -3 + 0.5w_2 - x_2 \end{array}$$

$$\begin{array}{ll} \text{(D)} & -\xi = 4 - 12y_1 - 4z_1 + 3y_3 \\ \hline & y_2 = 0.5 - y_1 - 0.5z_1 - 0.5y_3 \\ & z_2 = 3 - 5y_1 - 2z_1 + y_3. \end{array}$$

Continuing to work on the dual, we now see that y_3 is the entering variable and y_2 leaves. Hence, for the primal we use w_3 and w_2 as the leaving and entering variable, respectively. After pivoting, we have

$$\begin{array}{ll} \text{(P)} & \zeta = -7 - w_3 - 4x_2 \\ \hline & w_1 = 18 + 2w_3 + 7x_2 \\ & x_1 = 7 + w_3 + 3x_2 \\ & w_2 = 6 + 2w_3 + 2x_2 \end{array}$$

$$\begin{array}{ll} \text{(D)} & -\xi = 7 - 18y_1 - 7z_1 - 6y_2 \\ \hline & y_3 = 1 - 2y_1 - z_1 - 2y_2 \\ & z_2 = 4 - 7y_1 - 3z_1 - 2y_2. \end{array}$$

Now we notice that both dictionaries are optimal.

Of course, in each of the above dictionaries, the table of numbers in each dual dictionary is the negative-transpose of the corresponding primal table. Therefore, we never need to write the dual dictionary; the dual simplex method can be entirely described in terms of the primal dictionaries. Indeed, first we note that the dictionary must be dual feasible. This means that all the coefficients of the nonbasic variables in the primal objective function must be nonpositive. Given this, we proceed as follows. First we select the leaving variable by picking that basic variable whose constant term in the dictionary is the most negative (if there are none, then the current dictionary is optimal). Then we pick the entering variable by scanning across this row of the dictionary and comparing ratios of the coefficients in this row to the corresponding coefficients in the objective row, looking for the largest negated ratio just as we did in the primal simplex method. Once the entering and leaving variable are identified, we pivot to the next dictionary and continue from there. The reader is encouraged to trace

the pivots in the above example, paying particular attention to how one determines the entering and leaving variables by looking only at the primal dictionary.

7. A Dual-Based Phase I Algorithm

The dual simplex method described in the previous section provides us with a new Phase I algorithm, which if nothing else is at least more elegant than the one we gave in Chapter 2. Let us illustrate it using an example:

$$\begin{aligned} \text{maximize} \quad & -x_1 + 4x_2 \\ \text{subject to} \quad & -2x_1 - x_2 \leq 4 \\ & -2x_1 + 4x_2 \leq -8 \\ & -x_1 + 3x_2 \leq -7 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

The primal dictionary for this problem is

$$\begin{array}{rl} \text{(P)} & \zeta = -x_1 + 4x_2 \\ \hline & w_1 = 4 + 2x_1 + x_2 \\ & w_2 = -8 + 2x_1 - 4x_2 \\ & w_3 = -7 + x_1 - 3x_2, \end{array}$$

and even though at this point we realize that we don't need to look at the dual dictionary, let's track it anyway:

$$\begin{array}{rl} \text{(D)} & -\xi = -4y_1 + 8y_2 + 7y_3 \\ \hline & z_1 = 1 - 2y_1 - 2y_2 - y_3 \\ & z_2 = -4 - y_1 + 4y_2 + 3y_3. \end{array}$$

Clearly, neither the primal nor the dual dictionary is feasible. But by changing the primal objective function, we can easily produce a dual feasible dictionary. For example, let us temporarily change the primal objective function to

$$\eta = -x_1 - x_2.$$

Then the corresponding initial dual dictionary is feasible. In fact, it coincides with the dual dictionary we considered in the previous section, so we already know the optimal

solution for this modified problem. The optimal primal dictionary is

$$\begin{aligned} \eta &= -7 - \ w_3 - 4x_2 \\ \hline w_1 &= \ 18 + 2w_3 + 7x_2 \\ x_1 &= \ \ 7 + \ w_3 + 3x_2 \\ w_2 &= \ \ 6 + 2w_3 + 2x_2. \end{aligned}$$

This primal dictionary is optimal for the modified problem but not for the original problem. However, it is feasible for the original problem, and we can now simply reinstate the intended objective function and continue with Phase II. Indeed,

$$\begin{aligned} \zeta &= -x_1 + 4x_2 \\ &= -(7 + w_3 + 3x_2) + 4x_2 \\ &= -7 - w_3 + x_2. \end{aligned}$$

Hence, the starting dictionary for Phase II is

$$\begin{aligned} \zeta &= -7 - \ w_3 + \ x_2 \\ \hline w_1 &= \ 18 + 2w_3 + 7x_2 \\ x_1 &= \ \ 7 + \ w_3 + 3x_2 \\ w_2 &= \ \ 6 + 2w_3 + 2x_2. \end{aligned}$$

The entering variable is x_2. Looking for a leaving variable, we discover that this problem is unbounded. Of course, more typically one would expect to have to do several iterations of Phase II to find the optimal solution (or show unboundedness). Here we just got lucky that the game ended so soon.

It is interesting to note how we detect infeasibility with this new Phase I algorithm. The modified problem is guaranteed always to be dual feasible. It is easy to see that the primal problem is infeasible if and only if the modified problem is dual unbounded (which the dual simplex method will detect just as the primal simplex method detects primal unboundedness).

The two-phase algorithm we have just presented can be thought of as a dual–primal algorithm, since we first apply the dual simplex method to a modified dual feasible problem and then finish off by applying the primal simplex method to the original problem, starting from the feasible dictionary produced by Phase I. One could consider turning this around and doing a primal–dual two-phase algorithm. Here, the right-hand side of the primal problem would be modified to produce an obvious primal feasible solution. The primal simplex method would then be applied. The optimal solution to this primal problem will then be feasible for the original dual problem but

will not be optimal for it. But then the dual simplex method can be applied, starting with this dual feasible basis until an optimal solution for the dual problem is obtained.

8. The Dual of a Problem in General Form

In Chapter 1, we saw that linear programming problems can be formulated in a variety of ways. In this section, we shall derive the form of the dual when the primal problem is not necessarily presented in standard form.

First, let us consider the case where the linear constraints are equalities (and the variables are nonnegative):

$$
\begin{aligned}
&\text{maximize} && \sum_{j=1}^{n} c_j x_j && \\
&\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j = b_i && i = 1, 2, \ldots, m \\
& && x_j \geq 0 && j = 1, 2, \ldots, n.
\end{aligned} \tag{5.11}
$$

As we mentioned in Chapter 1, this problem can be reformulated with inequality constraints by simply writing each equality as two inequalities: one greater-than-or-equal-to and one less-than-or-equal-to:

$$
\begin{aligned}
&\text{maximize} && \sum_{j=1}^{n} c_j x_j && \\
&\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \leq b_i && i = 1, 2, \ldots, m \\
& && \sum_{j=1}^{n} a_{ij} x_j \geq b_i && i = 1, 2, \ldots, m \\
& && x_j \geq 0 && j = 1, 2, \ldots, n.
\end{aligned}
$$

Then negating each greater-than-or-equal-to constraint, we can put the problem into standard form:

$$
\begin{aligned}
&\text{maximize} && \sum_{j=1}^{n} c_j x_j && \\
&\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \leq b_i && i = 1, 2, \ldots, m \\
& && \sum_{j=1}^{n} -a_{ij} x_j \leq -b_i && i = 1, 2, \ldots, m \\
& && x_j \geq 0 && j = 1, 2, \ldots, n.
\end{aligned}
$$

Now that the problem is in standard form, we can write down its dual. Since there are two sets of m inequality constraints, we need two sets of m dual variables. Let's denote the dual variables associated with the first set of m constraints by y_i^+, $i = 1, 2, \ldots, m$, and the remaining dual variables by y_i^-, $i = 1, 2, \ldots, m$. With these notations, the dual problem is

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} b_i y_i^+ - \sum_{i=1}^{m} b_i y_i^- \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i^+ a_{ij} - \sum_{i=1}^{m} y_i^- a_{ij} \geq c_j \qquad j = 1, 2, \ldots, n \\ & y_i^+, y_i^- \geq 0 \qquad i = 1, 2, \ldots, m. \end{aligned}$$

A moment's reflection reveals that we can simplify this problem. If we put

$$y_i = y_i^+ - y_i^-, \qquad i = 1, 2, \ldots, m,$$

the dual problem reduces to

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i a_{ij} \geq c_j \qquad j = 1, 2, \ldots, n. \end{aligned}$$

This problem is *the* dual associated with (5.11). Note what has changed from when we were considering problems in standard form: now the dual variables are not restricted to be nonnegative. And that is the message: *equality constraints in the primal yield unconstrained variables (also referred to as free variables) in the dual, whereas inequality constraints in the primal yield nonnegative variables in the dual.* Employing the symmetry between the primal and the dual, we can say more: *free variables in the primal yield equality constraints in the dual, whereas nonnegative variables in the primal yield inequality constraints in the dual.* These rules are summarized in Table 5.1.

9. Resource Allocation Problems

Let us return to the production facility problem studied in Chapter 1. Recall that this problem involves a production facility that can take a variety of raw materials (enumerated $i = 1, 2, \ldots, m$) and turn them into a variety of final products (enumerated $j = 1, 2, \ldots, n$). We assume as before that the current market value of a unit of the ith raw material is ρ_i, that the current market price for a unit of the jth product is σ_j, that producing one unit of product j requires a_{ij} units of raw material i, and that at the current moment in time the facility has on hand b_i units of the ith raw material.

Primal	Dual
Equality Constraint	Free Variable
Inequality Constraint	Nonnegative Variable
Free Variable	Equality Constraint
Nonnegative Variable	Inequality Constraint

TABLE 5.1. Rules for forming the dual.

The current market values/prices are, by definition, related to each other by the formulas

$$\sigma_j = \sum_i \rho_i a_{ij}, \qquad j = 1, 2, \ldots, n.$$

These equations hold whenever the market is in equilibrium. (Of course, it is crucial to assume here that the collection of "raw materials" appearing on the right-hand side is exhaustive, including such items as depreciation of fixed assets and physical labor.) In the real world, the market is always essentially in equilibrium. Nonetheless, it continually experiences small perturbations that ripple through it and move the equilibrium to new levels.

These perturbations can be from several causes, an important one being innovation. One possible innovation is to improve the production process. This means that the values of some of the a_{ij}'s are reduced. Now, suddenly there is a windfall profit for each unit of product j produced. This windfall profit is given by

$$c_j = \sigma_j - \sum_i \rho_i a_{ij}. \tag{5.12}$$

Of course, eventually most producers of these products will take advantage of the same innovation, and once the suppliers get wind of the profits being made, they will get in on the action by raising the price of the raw materials.[1] Nonetheless, there is always a time lag; it is during this time that fortunes are made.

To be concrete, let us assume that the time lag is about one month (depending on the industry, this lag time could be considered too short or too long). Suppose also that the production manager decides to produce x_j units of product j and that all units produced are sold immediately at their market value. Then the total revenue

[1]One could take the prices of raw materials as fixed and argue that the value of the final products will fall. It doesn't really matter which view one adopts, since prices are relative anyway. The point is simply that the difference between the price of the raw materials and the price of the final products must narrow due to this innovation.

during this month will be $\sum_j \sigma_j x_j$. The value of the raw materials on hand at the beginning of the month was $\sum_i \rho_i b_i$. Also, if we denote the new price levels for the raw materials at the end of the month by w_i, $i = 1, 2, \ldots, m$, then the value of any remaining inventory at the end of the month is given by

$$\sum_i w_i \left(b_i - \sum_j a_{ij} x_j \right)$$

(if any term is negative, then it represents the cost of purchasing additional raw materials to meet the month's production requirements—we assume that these additional purchases are made at the new, higher, end-of-month price). The total windfall, call it π, (over all products) for this month can now be written as

$$\pi = \sum_j \sigma_j x_j + \sum_i w_i \left(b_i - \sum_j a_{ij} x_j \right) - \sum_i \rho_i b_i. \tag{5.13}$$

Our aim is to choose production levels x_j, $j = 1, 2, \ldots, n$, that maximize this windfall. But our supplier's aim is to choose prices w_i, $i = 1, 2, \ldots, m$, so as to minimize our windfall. Before studying these optimizations, let us first rewrite the windfall in a more convenient form. As in Chapter 1, let y_i denote the increase in the price of raw material i. That is,

$$w_i = \rho_i + y_i. \tag{5.14}$$

Substituting (5.14) into (5.13) and then simplifying notations using (5.12), we see that

$$\pi = \sum_j c_j x_j + \sum_i y_i \left(b_i - \sum_j a_{ij} x_j \right). \tag{5.15}$$

To emphasize that π depends on each of the x_j's and on the y_i's, we sometimes write it as $\pi(x_1, \ldots, x_n, y_1, \ldots, y_m)$.

Now let us return to the competing optimizations. Given x_j for $j = 1, 2, \ldots, n$, the suppliers react to minimize $\pi(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Looking at (5.15), we see that for any resource i in short supply, that is,

$$b_i - \sum_j a_{ij} x_j < 0,$$

the suppliers will jack up the price immensely (i.e., $y_i = \infty$). To avoid this obviously bad situation, the production manager will be sure to set the production levels so that

$$\sum_j a_{ij}x_j \le b_i, \qquad i = 1, 2, \ldots, m.$$

On the other hand, for any resource i that is not exhausted during the windfall month, that is,

$$b_i - \sum_j a_{ij}x_j > 0,$$

the suppliers will have no incentive to change the prevailing market price (i.e., $y_i = 0$). Therefore, from the production manager's point of view, the problem reduces to one of maximizing

$$\sum_j c_j x_j$$

subject to the constraints that

$$\sum_j a_{ij}x_j \le b_i, \qquad i = 1, 2, \ldots, m,$$

$$x_j \ge 0, \qquad j = 1, 2, \ldots, n.$$

This is just our usual primal linear programming problem. This is the problem that the production manager needs to solve in anticipation of adversarial suppliers.

Now let us look at the problem from the suppliers' point of view. Rearranging the terms in (5.15) by writing

$$\pi = \sum_j \left(c_j - \sum_i y_i a_{ij} \right) x_j + \sum_i y_i b_i, \tag{5.16}$$

we see that if the suppliers set prices in such a manner that a windfall remains on the jth product even after the price adjustment, that is,

$$c_j - \sum_i y_i a_{ij} > 0,$$

then the production manager would be able to generate for the facility an arbitrarily large windfall by producing a huge amount of the jth product (i.e., $x_j = \infty$). We assume that this is unacceptable to the suppliers, and so they will determine their price increases so that

$$\sum_i y_i a_{ij} \ge c_j, \qquad j = 1, 2, \ldots, n.$$

Also, if the suppliers set the price increases too high so that the production facility will lose money by producing product j, that is,

$$c_j - \sum_i y_i a_{ij} < 0,$$

then the production manager would simply decide not to engage in that activity. That is, she would set $x_j = 0$. Hence, the first term in (5.16) will always be zero, and so the optimization problem faced by the suppliers is to minimize

$$\sum_i b_i y_i$$

subject to the constraints that

$$\sum_i y_i a_{ij} \geq c_j, \qquad j = 1, 2, \ldots, n,$$

$$y_i \geq 0, \qquad i = 1, 2, \ldots, m.$$

This is precisely the dual of the production manager's problem!

As we've seen earlier with the strong duality theorem, if the production manager's problem has an optimal solution, then so does the suppliers' problem, and the two objectives agree. This means than an equilibrium can be reestablished by setting the production levels and the price hikes according to the optimal solutions to these two linear programming problems.

10. Lagrangian Duality

The analysis of the preceding section is an example of a general technique that forms the foundation of a subject called *Lagrangian duality*, which we shall briefly describe.

Let us start by summarizing what we did. It was quite simple. The analysis revolved around a function

$$\pi(x_1, \ldots, x_n, y_1, \ldots, y_m) = \sum_j c_j x_j - \sum_i \sum_i y_i a_{ij} x_j + \sum_i y_i b_i.$$

To streamline notations, let x stand for the entire collection of variables $x_1, x_2, \ldots, x_n$ and let y stand for the collection of y_i's so that we can write $\pi(x, y)$ in place of $\pi(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Written with these notations, we showed in the previous section that

$$\max_{x \geq 0} \min_{y \geq 0} \pi(x, y) = \min_{y \geq 0} \max_{x \geq 0} \pi(x, y).$$

We also showed that the inner optimization could in both cases be solved explicitly, that the max–min problem reduced to a linear programming problem, and that the min–max problem reduced to the dual linear programming problem.

One could imagine trying to carry out the same program for functions π that don't necessarily have the form shown above. In the general case, one needs to consider each step carefully. The max–min problem is called the primal problem, and the min–max problem is called the dual problem. However, it may or may not be true that these two problems have the same optimal objective values. In fact, the subject is interesting because one can indeed state specific, verifyable conditions for which the two problems do agree. Also, one would like to be able to solve the inner optimizations explicitly so that the primal problem can be stated as a pure maximization problem and the dual can be stated as a pure minimization problem. This, too, is often doable. There are various ways in which one can extend the notions of duality beyond the context of linear programming. The one just described is referred to as Lagrangian duality. It is perhaps the most important such extension.

Exercises

In solving the following problems, the advanced pivot tool can be used to check your arithmetic:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/advanced.html

5.1 What is the dual of the following linear programming problem:

$$\begin{array}{ll} \text{maximize} & x_1 - 2x_2 \\ \text{subject to} & x_1 + 2x_2 - \ x_3 + \ x_4 \geq 0 \\ & 4x_1 + 3x_2 + 4x_3 - 2x_4 \leq 3 \\ & -x_1 - \ x_2 + 2x_3 + \ x_4 = 1 \\ & x_2,\ x_3 \geq \ \ 0\ . \end{array}$$

5.2 Illustrate Theorem 5.2 on the problem in Exercise 2.9.

5.3 Illustrate Theorem 5.2 on the problem in Exercise 2.1.

5.4 Illustrate Theorem 5.2 on the problem in Exercise 2.2.

5.5 Consider the following linear programming problem:

$$\begin{array}{lrcl}
\text{maximize} & 2x_1 + 8x_2 - x_3 - 2x_4 & & \\
\text{subject to} & 2x_1 + 3 + 6x_4 & \leq & 6 \\
& -2x_1 + 4x_2 + 3x_3 & \leq & 1.5 \\
& 3x_1 + 2x_2 - 2x_3 - x_4 & \leq & 4 \\
& x_1,\ x_2,\ x_3,\ x_4 & \geq & 0.
\end{array}$$

Suppose that, in solving this problem, you have arrived at the following dictionary:

$$\begin{aligned}
\zeta &= 3.5 - 0.25w_1 + 6.25x_2 - 0.5w_3 - 1.5x_4 \\
\hline
x_1 &= 3.0 - 0.5w_1 - 1.5x_2 - 3.0x_4 \\
w_2 &= 0.0 + 1.25w_1 - 3.25x_2 - 1.5w_3 + 13.5x_4 \\
x_3 &= 2.5 - 0.75w_1 - 1.25x_2 + 0.5w_3 - 6.5x_4.
\end{aligned}$$

(a) Write down the dual problem.
(b) In the dictionary shown above, which variables are basic? Which are nonbasic?
(c) Write down the primal solution corresponding to the given dictionary. Is it feasible? Is it degenerate?
(d) Write down the corresponding dual dictionary.
(e) Write down the dual solution. Is it feasible?
(f) Do the primal/dual solutions you wrote above satisfy the complementary slackness property?
(g) Is the current primal solution optimal?
(h) For the next (primal) pivot, which variable will enter if the largest coefficient rule is used? Which will leave? Will the pivot be degenerate?

5.6 Solve the following linear program:

$$\begin{array}{lrcl} \text{maximize} & -x_1 - 2x_2 & & \\ \text{subject to} & -2x_1 + 7x_2 & \leq & 6 \\ & -3x_1 + x_2 & \leq & -1 \\ & 9x_1 - 4x_2 & \leq & 6 \\ & x_1 - x_2 & \leq & 1 \\ & 7x_1 - 3x_2 & \leq & 6 \\ & -5x_1 + 2x_2 & \leq & -3 \\ & x_1,\ x_2 & \geq & 0. \end{array}$$

5.7 Solve the linear program given in Exercise 2.3 using the dual–primal two-phase algorithm.

5.8 Solve the linear program given in Exercise 2.4 using the dual–primal two-phase algorithm.

5.9 Solve the linear program given in Exercise 2.6 using the dual–primal two-phase algorithm.

5.10 Using today's date (MMYY) for the seed value, solve 10 problems using the dual phase I primal phase II simplex method:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/dp2phase.html .

5.11 Using today's date (MMYY) for the seed value, solve 10 problems using the primal phase I dual phase II simplex method:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/pd2phase.html

5.12 For x and y in $\mathbb{R}$, compute

$$\max_{x \geq 0} \min_{y \geq 0} (x - y) \quad \text{and} \quad \min_{y \geq 0} \max_{x \geq 0} (x - y)$$

and note whether or not they are equal.

5.13 Consider the following process. Starting with a linear programming problem in standard form,

$$\begin{aligned} &\text{maximize} && \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i = 1, 2, \ldots, m \\ & && x_j \ge 0 \qquad j = 1, 2, \ldots, n, \end{aligned}$$

first form its dual:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{m} b_i y_i \\ &\text{subject to} && \sum_{i=1}^{m} y_i a_{ij} \ge c_j \qquad j = 1, 2, \ldots, n \\ & && y_i \ge 0 \qquad i = 1, 2, \ldots, m. \end{aligned}$$

Then replace the minimization in the dual with a maximization to get a new linear programming problem, which we can write in standard form as follows:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{m} b_i y_i \\ &\text{subject to} && \sum_{i=1}^{m} -y_i a_{ij} \le -c_j \qquad j = 1, 2, \ldots, n \\ & && y_i \ge 0 \qquad i = 1, 2, \ldots, m. \end{aligned}$$

If we identify a linear programming problem with its data, (a_{ij}, b_i, c_j), the above process can be thought of as a transformation T on the space of data defined by

$$(a_{ij}, b_i, c_j) \xrightarrow{T} (-a_{ji}, -c_j, b_i).$$

Let $\zeta^*(a_{ij}, b_i, c_j)$ denote the optimal objective function value of the standard-form linear programming problem having data (a_{ij}, b_i, c_j). By strong duality together with the fact that a maximization dominates a minimization, it follows that

$$\zeta^*(a_{ij}, b_i, c_j) \le \zeta^*(-a_{ji}, -c_j, b_i).$$

Now if we repeat this process, we get

$$\begin{aligned}(a_{ij}, b_i, c_j) &\xrightarrow{T} (-a_{ji}, -c_j, b_i)\\ &\xrightarrow{T} (a_{ij}, -b_i, -c_j)\\ &\xrightarrow{T} (-a_{ji}, c_j, -b_i)\\ &\xrightarrow{T} (a_{ij}, b_i, c_j)\end{aligned}$$

and hence that

$$\begin{aligned}\zeta^*(a_{ij}, b_i, c_j) &\leq \zeta^*(-a_{ji}, -c_j, b_i)\\ &\leq \zeta^*(a_{ij}, -b_i, -c_j)\\ &\leq \zeta^*(-a_{ji}, c_j, -b_i)\\ &\leq \zeta^*(a_{ij}, b_i, c_j).\end{aligned}$$

But the first and the last entry in this chain of inequalities are equal. Therefore, all these inequalities would seem to be equalities. While this outcome could happen sometimes, it certainly isn't always true. What is the error in this logic? Can you state a (correct) nontrivial theorem that follows from this line of reasoning? Can you give an example where the four inequalities are indeed all equalities?

5.14 Consider the following variant of the resource allocation problem:

$$\begin{aligned}&\text{maximize } && c^T x\\ &\text{subject to } && Ax \leq b\\ &&& 0 \leq x \leq u.\end{aligned} \tag{5.17}$$

Here, c denotes the vector of unit prices for the products, b denotes the vector containing the number of units on hand of each raw material, and u denotes a vector of upper bounds on the number of units of each product that can be sold at the set price. Now, let's assume that the raw materials have not been purchased yet and it is part of the problem to determine b. Let p denote the vector of prices for each raw material. The problem then becomes an

optimization over both x and b:

$$\begin{aligned} \text{maximize} \quad & c^T x - p^T b \\ \text{subject to} \quad & Ax - b \leq 0 \\ & 0 \leq x \leq u \\ & b \geq 0. \end{aligned}$$

(a) Show that this problem always has an optimal solution.
(b) Let $y^*(b)$ denote optimal dual variables for the original resource allocation problem (5.17) (note that we've explicitly indicated that these dual variables depend on the vector b). Show that the optimal value of b, call it b^*, satisfies

$$y^*(b^*) = p.$$

5.15 Consider the following linear program:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} q_j x_j \leq \beta \\ & x_j \leq 1 \qquad j = 1, 2, \ldots, n \\ & x_j \geq 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

Here, the numbers p_j, $j = 1, 2, \ldots, n$ are positive and sum to one. The same is true of the q_j's:

$$\begin{aligned} \sum_{j=1}^{n} q_j &= 1 \\ q_j &> 0. \end{aligned}$$

Furthermore, assume that

$$\frac{p_1}{q_1} < \frac{p_2}{q_2} < \cdots < \frac{p_n}{q_n}$$

and that the parameter β is a small positive number. Let $k = \min\{j : q_{j+1} + \cdots + q_n \leq \beta\}$. Let y_0 denote the dual variable associated with the constraint involving β, and let y_j denote the dual variable associated with

the upper bound of 1 on variable x_j. Using duality theory, show that the optimal values of the primal and dual variables are given by

$$x_j = \begin{cases} 0 & j < k \\ \frac{\beta - q_{k+1} - \cdots - q_n}{q_k} & j = k \\ 1 & j > k \end{cases}$$

$$y_j = \begin{cases} \frac{p_k}{q_k} & j = 0 \\ 0 & 0 < j \leq k \\ q_j \left(\frac{p_j}{q_j} - \frac{p_k}{q_k} \right) & j > k \end{cases}$$

See Exercise 1.3 for the motivation for this problem.

5.16 *Diet Problem.* An MIT graduate student was trying to make ends meet on a very small stipend. He went to the library and looked up the National Research Council's publication entitled "Recommended Dietary Allowances" and was able to determine a minimum daily intake quantity of each essential nutrient for a male in his weight and age category. Let m denote the number of nutrients that he identified as important to his diet, and let b_i for $i = 1, 2, \ldots, m$ denote his personal minimum daily requirements. Next, he made a list of his favorite foods (which, except for pizza and due mostly to laziness and ineptitude in the kitchen, consisted almost entirely of frozen prepared meals). He then went to the local grocery store and made a list of the unit price for each of his favorite foods. Let us denote these prices as c_j for $j = 1, 2, \ldots, n$. In addition to prices, he also looked at the labels and collected information about how much of the critical nutrients are contained in one serving of each food. Let us denote by a_{ij} the amount of nutrient i contained in food j. (Fortunately, he was able to call his favorite pizza delivery service and get similar information from them.) In terms of this information, he formulated the following linear programming problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i = 1, 2, \ldots, m \\ & x_j \geq 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

Formulate the dual to this linear program. Can you introduce another person into the above story whose problem would naturally be to solve the dual?

5.17 *Saddle points.* A function $h(y)$ defined for $y \in \mathbb{R}$ is called *strongly convex* if

- $h''(y) > 0$ for all $y \in \mathbb{R}$,
- $\lim_{y\to-\infty} h'(y) = -\infty$, and
- $\lim_{y\to\infty} h'(y) = \infty$.

A function h is called *strongly concave* if $-h$ is strongly convex. Let $\pi(x, y)$, be a function defined for $(x, y) \in \mathbb{R}^2$ and having the following form

$$\pi(x, y) = f(x) - xy + g(y),$$

where f is strongly concave and g is strongly convex. Using elementary calculus

1. Show that there is one and only one point $(x^*, y^*) \in \mathbb{R}^2$ at which the gradient of π,
$$\nabla\pi = \begin{bmatrix} \partial\pi/\partial x \\ \partial\pi/\partial y \end{bmatrix},$$
vanishes. *Hint: From the two equations obtained by setting the derivatives to zero, derive two other relations having the form $x = \phi(x)$ and $y = \psi(y)$. Then study the functions ϕ and ψ to show that there is one and only one solution.*
2. Show that
$$\max_{x\in\mathbb{R}} \min_{y\in\mathbb{R}} \pi(x, y) = \pi(x^*, y^*) = \min_{y\in\mathbb{R}} \max_{x\in\mathbb{R}} \pi(x, y),$$
where (x^*, y^*) denotes the "critical point" identified in part 1 above. (Note: Be sure to check the signs of the second derivatives for both the inner and the outer optimizations.)

Associated with each strongly convex function h is another function, called the *Legendre transform* of h and denoted by L_h, defined by

$$L_h(x) = \max_{y\in\mathbb{R}}(xy - h(y)), \qquad x \in \mathbb{R}.$$

3. Using elementary calculus, show that L_h is strongly convex.
4. Show that
$$\max_{x\in\mathbb{R}} \min_{y\in\mathbb{R}} \pi(x, y) = \max_{x\in\mathbb{R}}(f(x) - L_g(x))$$
and that
$$\min_{y\in\mathbb{R}} \max_{x\in\mathbb{R}} \pi(x, y) = \min_{y\in\mathbb{R}}(g(y) + L_{-f}(-y)).$$

5. Show that the Legendre transform of the Legendre transform of a function is the function itself. That is,

$$L_{L_h}(z) = h(z) \qquad \text{for all } z \in \mathbb{R}.$$

Hint: This can be proved from scratch but it is easier to use the result of part 2 above.

Notes

The idea behind the strong duality theorem can be traced back to conversations between G.B. Dantzig and J. von Neumann in the fall of 1947, but an explicit statement did not surface until the paper of Gale et al. (1951). The term *primal problem* was coined by G.B. Dantzig's father, T. Dantzig. The dual simplex method was first proposed by Lemke (1954).

The solution to Exercise 5.13 (which is left to the reader to supply) suggests that a random linear programming problem is infeasible with probability $1/4$, unbounded with probability $1/4$, and has an optimal solution with probability $1/2$.

CHAPTER 6

The Simplex Method in Matrix Notation

So far, we have avoided using matrix notation to present linear programming problems and the simplex method. In this chapter, we shall recast everything into matrix notation. At the same time, we will emphasize the close relations between the primal and the dual problems.

1. Matrix Notation

As usual, we begin our discussion with the standard-form linear programming problem:

$$\begin{aligned} \text{maximize } & \sum_{j=1}^{n} c_j x_j \\ \text{subject to } & \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i = 1, 2, \ldots, m \\ & x_j \ge 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

In the past, we have generally denoted slack variables by w_i's but have noted that sometimes it is convenient just to string them onto the end of the list of original variables. Such is the case now, and so we introduce slack variables as follows:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, 2, \ldots, m.$$

With these slack variables, we now write our problem in matrix form:

$$\begin{aligned} \text{maximize } & c^T x \\ \text{subject to } & Ax = b \\ & x \ge 0, \end{aligned}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & 1 & & \\ a_{21} & a_{22} & \dots & a_{2n} & & 1 & \\ \vdots & \vdots & & \vdots & & & \ddots \\ a_{m1} & a_{m2} & \dots & a_{mn} & & & 1 \end{bmatrix}, \tag{6.1}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{and} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{n+m} \end{bmatrix}. \tag{6.2}$$

As we know, the simplex method is an iterative procedure in which each iteration is characterized by specifying which m of the $n + m$ variables are basic. As before, we denote by $\mathcal{B}$ the set of indices corresponding to the basic variables, and we denote by $\mathcal{N}$ the remaining nonbasic indices.

In component notation, the ith component of Ax can be broken up into a basic part and a nonbasic part:

$$\sum_{j=1}^{n+m} a_{ij}x_j = \sum_{j\in\mathcal{B}} a_{ij}x_j + \sum_{j\in\mathcal{N}} a_{ij}x_j. \tag{6.3}$$

We wish to introduce a notation for matrices that will allow us to break up the matrix product Ax analogously. To this end, let B denote an $m \times m$ matrix whose columns consist precisely of the m columns of A that are associated with the basic variables. Similarly, let N denote an $m \times n$ matrix whose columns are the n nonbasic columns of A. Then we write A in a partitioned-matrix form as follows:

$$A = \begin{bmatrix} B & N \end{bmatrix}$$

Strictly speaking, the matrix on the right does not equal the A matrix. Instead, it is the A matrix with its columns rearranged in such a manner that all the columns associated with basic variables are listed first followed by the nonbasic columns. Nonetheless, as

long as we are consistent and rearrange the rows of x in the same way, then no harm is done. Indeed, let us similarly rearrange the rows of x and write

$$x = \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix}.$$

Then the following separation of Ax into a sum of two terms is true and captures the same separation into basic and nonbasic parts as we had in (6.3):

$$Ax = \begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = Bx_{\mathcal{B}} + Nx_{\mathcal{N}}.$$

By similarly partitioning c, we can write

$$c^T x = \begin{bmatrix} c_{\mathcal{B}} \\ c_{\mathcal{N}} \end{bmatrix}^T \begin{bmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{bmatrix} = c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}}.$$

2. The Primal Simplex Method

A dictionary has the property that the basic variables are written as functions of the nonbasic variables. In matrix notation, we see that the constraint equations

$$Ax = b$$

can be written as

$$Bx_{\mathcal{B}} + Nx_{\mathcal{N}} = b.$$

The fact that the basic variables $x_{\mathcal{B}}$ can be written as a function of the nonbasic variables $x_{\mathcal{N}}$ is equivalent to the fact that the matrix B is invertible, and hence,

$$x_{\mathcal{B}} = B^{-1}b - B^{-1}Nx_{\mathcal{N}}. \tag{6.4}$$

(The fact that B is invertible means that its m column vectors are linearly independent and therefore form a basis for $\mathbb{R}^m$ — this is why the basic variables are called basic, in case you were wondering.) Similarly, the objective function can be written as

$$\begin{aligned} \zeta &= c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}} \\ &= c_{\mathcal{B}}^T \left(B^{-1}b - B^{-1}Nx_{\mathcal{N}} \right) + c_{\mathcal{N}}^T x_{\mathcal{N}} \\ &= c_{\mathcal{B}}^T B^{-1}b - \left((B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}} \right)^T x_{\mathcal{N}}. \end{aligned} \tag{6.5}$$

Combining (6.5) and (6.4), we see that we can write the dictionary associated with basis $\mathcal{B}$ as

$$\begin{aligned} \zeta &= c_{\mathcal{B}}^T B^{-1} b - \left((B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}} \right)^T x_{\mathcal{N}} \\ x_{\mathcal{B}} &= \quad B^{-1} b - B^{-1} N x_{\mathcal{N}}. \end{aligned} \tag{6.6}$$

Comparing against the component-form notation of Chapter 2 (see (2.6)), we make the following identifications:

$$\begin{aligned} c_{\mathcal{B}}^T B^{-1} b &= \bar{\zeta} \\ c_{\mathcal{N}} - (B^{-1}N)^T c_{\mathcal{B}} &= [\bar{c}_j] \\ B^{-1} b &= \left[\bar{b}_i\right] \\ B^{-1} N &= [\bar{a}_{ij}], \end{aligned}$$

where the bracketed expressions on the right denote vectors and matrices with the index i running over $\mathcal{B}$ and the index j running over $\mathcal{N}$. The basic solution associated with dictionary (6.6) is obtained by setting $x_{\mathcal{N}}$ equal to zero:

$$\begin{aligned} x_{\mathcal{N}}^* &= 0, \\ x_{\mathcal{B}}^* &= B^{-1} b. \end{aligned} \tag{6.7}$$

As we saw in the last chapter, associated with each primal dictionary there is a dual dictionary that is simply the negative-transpose of the primal. However, to have the negative-transpose property, it is important to correctly associate complementary pairs of variables. So first we recall that, for the current discussion, we have appended the primal slack variables to the end of the original variables:

$$(x_1, \ldots, x_n, w_1, \ldots, w_m) \longrightarrow (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}).$$

Also recall that the dual slack variables are complementary to the original primal variables and that the original dual variables are complementary to the primal slack variables. Therefore, to maintain the desired complementarity condition between like indices in the primal and the dual, we need to relabel the dual variables and append them to the end of the dual slacks:

$$(z_1, \ldots, z_n, y_1, \ldots, y_m) \longrightarrow (z_1, \ldots, z_n, z_{n+1}, \ldots, z_{n+m}).$$

With this relabeling of the dual variables, the dual dictionary corresponding to (6.6) is

$$\begin{aligned} -\xi &= \quad -c_{\mathcal{B}}^T B^{-1} b - (B^{-1}b)^T z_{\mathcal{B}} \\ z_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}} + (B^{-1}N)^T z_{\mathcal{B}}. \end{aligned}$$

The dual solution associated with this dictionary is obtained by setting $z_{\mathcal{B}}$ equal to zero:

$$\begin{aligned} z^*_{\mathcal{B}} &= 0, \\ z^*_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}. \end{aligned} \tag{6.8}$$

Using (6.7) and (6.8) and introducing the shorthand

$$\zeta^* = c_{\mathcal{B}}^T B^{-1} b, \tag{6.9}$$

we see that we can write the primal dictionary succinctly as

$$\begin{aligned} \zeta &= \zeta^* - {z^*_{\mathcal{N}}}^T x_{\mathcal{N}} \\ x_{\mathcal{B}} &= x^*_{\mathcal{B}} - B^{-1}N x_{\mathcal{N}}. \end{aligned} \tag{6.10}$$

The associated dual dictionary then has a very symmetric appearance:

$$\begin{aligned} -\xi &= -\zeta^* - (x^*_{\mathcal{B}})^T z_{\mathcal{B}} \\ z_{\mathcal{N}} &= \ z^*_{\mathcal{N}} + (B^{-1}N)^T z_{\mathcal{B}}. \end{aligned} \tag{6.11}$$

The (primal) simplex method can be described briefly as follows. The starting assumptions are that we are given

(1) a partition of the $n+m$ indices into a collection $\mathcal{B}$ of m basic indices and a collection $\mathcal{N}$ of n nonbasic ones with the property that the basis matrix B is invertible,
(2) an associated current primal solution $x^*_{\mathcal{B}} \geq 0$ (and $x^*_{\mathcal{N}} = 0$), and
(3) an associated current dual solution $z^*_{\mathcal{N}}$ (with $z^*_{\mathcal{B}} = 0$)

such that the dictionary given by (6.10) represents the primal objective function and the primal constraints. The simplex method then produces a sequence of steps to "adjacent" bases such that the current value ζ^* of the objective function ζ increases at each step (or, at least, would increase if the step size were positive), updating $x^*_{\mathcal{B}}$ and $z^*_{\mathcal{N}}$ along the way. Two bases are said to be adjacent to each other if they differ in only one index. That is, given a basis $\mathcal{B}$, an adjacent basis is determined by removing one basic index and replacing it with a nonbasic index. The index that gets removed corresponds to the leaving variable, whereas the index that gets added corresponds to the entering variable.

One step of the simplex method is called an iteration. We now elaborate further on the details by describing one iteration as a sequence of specific steps.

Step 1. Check for Optimality. If $z^*_{\mathcal{N}} \geq 0$, stop. The current solution is optimal. To see this, first note that the simplex method always maintains primal feasibility and complementarity. Indeed, the primal solution is feasible, since $x^*_{\mathcal{B}} \geq 0$ and $x_{\mathcal{N}} = 0$ and the dictionary embodies the primal constraints. Also, the fact that $x^*_{\mathcal{N}} = 0$ and

$z^*_{\mathcal{B}} = 0$ implies that the primal and dual solutions are complementary. Hence, all that is required for optimality is dual feasibility. But by looking at the associated dual dictionary (6.11), we see that the dual solution is feasible if and only if $z^*_{\mathcal{N}} \geq 0$.

Step 2. Select Entering Variable. Pick an index $j \in \mathcal{N}$ for which $z^*_j < 0$. Variable x_j is the *entering variable.*

Step 3. Compute Primal Step Direction $\Delta x_{\mathcal{B}}$. Having selected the entering variable, it is our intention to let its value increase from zero. Hence, we let

$$x_{\mathcal{N}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ t \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} = te_j, \qquad \begin{matrix} \\ \\ \\ \\ \nwarrow \\ \\ \\ \end{matrix} \quad j\text{th position}$$

where we follow the common convention of letting e_j denote the unit vector that is zero in every component except for a one in the position associated with index j (note that, because of our index rearrangement conventions, this is not generally the jth element of the vector). Then from (6.10), we have that

$$x_{\mathcal{B}} = x^*_{\mathcal{B}} - B^{-1}Nte_j.$$

Hence, we see that the step direction $\Delta x_{\mathcal{B}}$ for the primal basic variables is given by

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j.$$

Step 4. Compute Primal Step Length. We wish to pick the largest $t \geq 0$ for which every component of $x_{\mathcal{B}}$ remains nonnegative. That is, we wish to pick the largest t for which

$$x^*_{\mathcal{B}} \geq t\Delta x_{\mathcal{B}}.$$

Since, for each $i \in \mathcal{B}^*$, $x^*_i \geq 0$ and $t \geq 0$, we can divide both sides of the above inequality by these numbers and preserve the sense of the inequality. Therefore, doing this division, we get the requirement that

$$\frac{1}{t} \geq \frac{\Delta x_i}{x^*_i}, \qquad \text{for all } i \in \mathcal{B}.$$

We want to let t be as large as possible, and so $1/t$ should be made as small as possible. The smallest possible value for $1/t$ that satisfies all the required inequalities is

obviously

$$\frac{1}{t} = \max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*}.$$

Hence, the largest t for which all of the inequalities hold is given by

$$t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*} \right)^{-1}.$$

As always, the correct convention for $0/0$ is to set such ratios to zero. Also, if the maximum is less than or equal to zero, we can stop here—the primal is unbounded.

Step 5. Select Leaving Variable. The leaving variable is chosen as any variable x_i, $i \in \mathcal{B}$, for which the maximum in the calculation of t is obtained.

Step 6. Compute Dual Step Direction $\Delta z_{\mathcal{N}}$. Essentially all that remains is to explain how $z_{\mathcal{N}}^*$ changes. To see how, it is convenient to look at the dual dictionary. Since in that dictionary z_i is the entering variable, we see that

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i.$$

Step 7. Compute Dual Step Length. Since we know that z_j is the leaving variable in the dual dictionary, we see immediately that the step length for the dual variables is

$$s = \frac{z_j^*}{\Delta z_j}.$$

Step 8. Update Current Primal and Dual Solutions. We now have everything we need to update the data in the dictionary:

$$x_j^* \leftarrow t$$
$$x_{\mathcal{B}}^* \leftarrow x_{\mathcal{B}}^* - t\Delta x_{\mathcal{B}}$$

and

$$z_i^* \leftarrow s$$
$$z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}}.$$

Step 9. Update Basis. Finally, we update the basis:

$$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}.$$

We close this section with the important remark that the simplex method as presented here, while it may look different from the component-form presentation given in Chapter 2, is in fact mathematically identical to it. That is, given the same set of

pivoting rules and starting from the same primal dictionary, the two algorithms will generate exactly the same sequence of dictionaries.

3. An Example

In case the reader is feeling at this point that there are too many letters and not enough numbers, here is an example that illustrates the matrix approach to the simplex method. The problem we wish to solve is

$$\begin{aligned} \text{maximize} \quad & 4x_1 + 3x_2 \\ \text{subject to} \quad & x_1 - x_2 \le 1 \\ & 2x_1 - x_2 \le 3 \\ & x_2 \le 5 \\ & x_1,\ x_2 \ge 0. \end{aligned}$$

The matrix A is given by

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

(Note that some zeros have not been shown.) The initial sets of basic and nonbasic indices are

$$\mathcal{B} = \{3, 4, 5\} \qquad \text{and} \qquad \mathcal{N} = \{1, 2\}.$$

Corresponding to these sets, we have the submatrices of A:

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

From (6.7) we see that the initial values of the basic variables are given by

$$x_{\mathcal{B}}^* = b = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix},$$

and from (6.8) the initial nonbasic dual variables are simply

$$z_{\mathcal{N}}^* = -c_{\mathcal{N}} = \begin{bmatrix} -4 \\ -3 \end{bmatrix}.$$

Since $x^*_{\mathcal{B}} \geq 0$, the initial solution is primal feasible, and hence we can apply the simplex method without needing any Phase I procedure.

3.1. First Iteration. *Step 1.* Since $z^*_{\mathcal{N}}$ has some negative components, the current solution is not optimal.

Step 2. Since $z^*_1 = -4$ and this is the most negative of the two nonbasic dual variables, we see that the entering index is

$$j = 1.$$

Step 3.

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j = \begin{bmatrix} 1 & -1 \\ 2 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}.$$

Step 4.

$$t = \left(\max \left\{ \frac{1}{1}, \frac{2}{3}, \frac{0}{5} \right\} \right)^{-1} = 1.$$

Step 5. Since the ratio that achieved the maximum in Step 4 was the first ratio and this ratio corresponds to basis index 3, we see that

$$i = 3.$$

Step 6.

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i = - \begin{bmatrix} 1 & 2 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Step 7.

$$s = \frac{z^*_j}{\Delta z_j} = \frac{-4}{-1} = 4.$$

Step 8.

$$x_1^* = 1, \qquad x_{\mathcal{B}}^* = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} - 1 \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix},$$

$$z_3^* = 4, \qquad z_{\mathcal{N}}^* = \begin{bmatrix} -4 \\ -3 \end{bmatrix} - 4 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -7 \end{bmatrix}.$$

Step 9. The new sets of basic and nonbasic indices are

$$\mathcal{B} = \{1, 4, 5\} \qquad \text{and} \qquad \mathcal{N} = \{3, 2\}.$$

Corresponding to these sets, we have the new basic and nonbasic submatrices of A,

$$B = \begin{bmatrix} 1 & & \\ 2 & 1 & \\ 0 & & 1 \end{bmatrix} \qquad N = \begin{bmatrix} 1 & -1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix},$$

and the new basic primal variables and nonbasic dual variables:

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_1^* \\ x_4^* \\ x_5^* \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 5 \end{bmatrix} \qquad z_{\mathcal{N}}^* = \begin{bmatrix} z_3^* \\ z_2^* \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \end{bmatrix}.$$

3.2. Second Iteration. *Step 1.* Since $z_{\mathcal{N}}^*$ has some negative components, the current solution is not optimal.

Step 2. Since $z_2^* = -7$, we see that the entering index is

$$j = 2.$$

Step 3.

$$\Delta x_{\mathcal{B}} = B^{-1} N e_j = \begin{bmatrix} 1 & & \\ 2 & 1 & \\ 0 & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & -1 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}.$$

Step 4.

$$t = \left(\max \left\{ \frac{-1}{1}, \frac{1}{1}, \frac{1}{5} \right\} \right)^{-1} = 1.$$

Step 5. Since the ratio that achieved the maximum in Step 4 was the second ratio and this ratio corresponds to basis index 4, we see that

$$i = 4.$$

Step 6.

$$\begin{aligned}\Delta z_{\mathcal{N}} &= -(B^{-1}N)^T e_i \\ &= -\begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ & 1 & \\ & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}.\end{aligned}$$

Step 7.

$$s = \frac{z_j^*}{\Delta z_j} = \frac{-7}{-1} = 7.$$

Step 8.

$$x_2^* = 1, \qquad x_{\mathcal{B}}^* = \begin{bmatrix} 1 \\ 1 \\ 5 \end{bmatrix} - 1 \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 4 \end{bmatrix},$$

$$z_4^* = 7, \qquad z_{\mathcal{N}}^* = \begin{bmatrix} 4 \\ -7 \end{bmatrix} - 7 \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \end{bmatrix}.$$

Step 9. The new sets of basic and nonbasic indices are

$$\mathcal{B} = \{1, 2, 5\} \qquad \text{and} \qquad \mathcal{N} = \{3, 4\}.$$

Corresponding to these sets, we have the new basic and nonbasic submatrices of A,

$$B = \begin{bmatrix} 1 & -1 & 0 \\ 2 & -1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

and the new basic primal variables and nonbasic dual variables:

$$x_{\mathcal{B}}^* = \begin{bmatrix} x_1^* \\ x_2^* \\ x_5^* \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \qquad z_{\mathcal{N}}^* = \begin{bmatrix} z_3^* \\ z_4^* \end{bmatrix} = \begin{bmatrix} -10 \\ 7 \end{bmatrix}.$$

3.3. Third Iteration. *Step 1.* Since $z^*_{\mathcal{N}}$ has some negative components, the current solution is not optimal.

Step 2. Since $z^*_3 = -10$, we see that the entering index is

$$j = 3.$$

Step 3.

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j = \begin{bmatrix} 1 & -1 & 0 \\ 2 & -1 & 0 \\ 0 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 2 \end{bmatrix}.$$

Step 4.

$$t = \left(\max \left\{ \frac{-1}{2}, \frac{-2}{1}, \frac{2}{4} \right\} \right)^{-1} = 2.$$

Step 5. Since the ratio that achieved the maximum in Step 4 was the third ratio and this ratio corresponds to basis index 5, we see that

$$i = 5.$$

Step 6.

$$\begin{aligned} \Delta z_{\mathcal{N}} &= -(B^{-1}N)^T e_i \\ &= - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ -1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}. \end{aligned}$$

Step 7.

$$s = \frac{z^*_j}{\Delta z_j} = \frac{-10}{-2} = 5.$$

Step 8.

$$\begin{aligned} x^*_3 = 2, \qquad x^*_{\mathcal{B}} &= \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} - 2 \begin{bmatrix} -1 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 0 \end{bmatrix}, \\ z^*_5 = 5, \qquad z^*_{\mathcal{N}} &= \begin{bmatrix} -10 \\ 7 \end{bmatrix} - 5 \begin{bmatrix} -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}. \end{aligned}$$

Step 9. The new sets of basic and nonbasic indices are

$$\mathcal{B} = \{1, 2, 3\} \qquad \text{and} \qquad \mathcal{N} = \{5, 4\}.$$

Corresponding to these sets, we have the new basic and nonbasic submatrices of A,

$$B = \begin{bmatrix} 1 & -1 & 1 \\ 2 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad N = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix},$$

and the new basic primal variables and nonbasic dual variables:

$$x^*_{\mathcal{B}} = \begin{bmatrix} x^*_1 \\ x^*_2 \\ x^*_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 2 \end{bmatrix} \qquad z^*_{\mathcal{N}} = \begin{bmatrix} z^*_5 \\ z^*_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}.$$

3.4. Fourth Iteration. *Step 1.* Since $z^*_{\mathcal{N}}$ has all nonnegative components, the current solution is optimal. The optimal objective function value is

$$\zeta^* = 4x^*_1 + 3x^*_2 = 31.$$

It is undoubtedly clear at this point that the matrix approach, as we have presented it, is quite a bit more tedious than the dictionary manipulations with which we are quite familiar. The reason is that, with the dictionary approach, dictionary entries get updated from one iteration to the next and the updating process is fairly easy, whereas with the matrix approach, we continually compute everything from scratch and therefore end up solving many systems of equations. In the next chapter, we will deal with this issue and show that these systems of equations don't really have to be solved from scratch each time; instead, there is a certain updating that can be done that is quite analogous to the updating of a dictionary. However, before we take up such practical considerations, let us finish our general discussion of the simplex method by casting the dual simplex method into matrix notation and discussing some related issues.

4. The Dual Simplex Method

In the presentation of the primal simplex method given in the previous section, we tried to make the symmetry between the primal and the dual problems as evident as possible. One advantage of this approach is that we can now easily write down the dual simplex method. Instead of assuming that the primal dictionary is feasible ($x^*_{\mathcal{B}} \geq 0$), we now assume that the dual dictionary is feasible ($z^*_{\mathcal{N}} \geq 0$) and perform the analogous steps:

Step 1. Check for Optimality. If $x^*_{\mathcal{B}} \geq 0$, stop. The current solution is optimal. Note that for the dual simplex method, dual feasibility and complementarity are maintained from the beginning, and the algorithm terminates once a primal feasible solution is discovered.

Step 2. Select Entering Variable. Pick an index $i \in \mathcal{B}$ for which $x^*_i < 0$. Variable z_i is the *entering variable*.

Step 3. Compute Dual Step Direction $\Delta z_{\mathcal{N}}$. From the dual dictionary, we see that

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i.$$

Step 4. Compute Dual Step Length. We wish to pick the largest $s \geq 0$ for which every component of $z_{\mathcal{N}}$ remains nonnegative. As in the primal simplex method, this computation involves computing the maximum of some ratios:

$$s = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j} \right)^{-1}.$$

If s is not positive, then stop here—the dual is unbounded (implying, of course, that the primal is infeasible).

Step 5. Select Leaving Variable. The leaving variable is chosen as any variable z_j, $j \in \mathcal{N}$, for which the maximum in the calculation of s is obtained.

Step 6. Compute Primal Step Direction $\Delta x_{\mathcal{B}}$. To see how $x^*_{\mathcal{B}}$ changes in the dual dictionary, it is convenient to look at the primal dictionary. Since in that dictionary x_j is the entering variable, we see that

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j.$$

Step 7. Compute Primal Step Length. Since we know that x_i is the leaving variable in the primal dictionary, we see immediately that the step length for the primal variables is

$$t = \frac{x^*_i}{\Delta x_i}.$$

Step 8. Update Current Primal and Dual Solutions. We now have everything we need to update the data in the dictionary:

$$x^*_j \leftarrow t$$
$$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}.$$

and

$$z^*_i \leftarrow s$$
$$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$$

Primal Simplex	Dual Simplex
Suppose $x^*_{\mathcal{B}} \geq 0$	Suppose $z^*_{\mathcal{N}} \geq 0$
while $(z^*_{\mathcal{N}} \not\geq 0)$ {	while $(x^*_{\mathcal{B}} \not\geq 0)$ {
pick $j \in \{j \in \mathcal{N} : z^*_j < 0\}$	pick $i \in \{i \in \mathcal{B} : x^*_i < 0\}$
$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$	$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$
$t = \left(\max_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i} \right)^{-1}$	$s = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j} \right)^{-1}$
pick $i \in \operatorname{argmax}_{i \in \mathcal{B}} \frac{\Delta x_i}{x^*_i}$	pick $j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z^*_j}$
$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$	$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$
$s = \frac{z^*_j}{\Delta z_j}$	$t = \frac{x^*_i}{\Delta x_i}$
$x^*_j \leftarrow t$	$x^*_j \leftarrow t$
$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}$	$x^*_{\mathcal{B}} \leftarrow x^*_{\mathcal{B}} - t\Delta x_{\mathcal{B}}$
$z^*_i \leftarrow s$	$z^*_i \leftarrow s$
$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$	$z^*_{\mathcal{N}} \leftarrow z^*_{\mathcal{N}} - s\Delta z_{\mathcal{N}}$
$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$	$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$
}	}

FIGURE 6.1. The primal and the dual simplex methods.

Step 9. Update Basis. Finally, we update the basis:

$$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}.$$

To further emphasize the similarities between the primal and the dual simplex methods, Figure 6.1 shows the two algorithms side by side.

5. Two-Phase Methods

Let us summarize the algorithm obtained by applying the dual simplex method as a Phase I procedure followed by the primal simplex method as a Phase II. Initially, we set

$$\mathcal{B} = \{n+1, n+2, \ldots, n+m\} \qquad \text{and} \qquad \mathcal{N} = \{1, 2, \ldots, n\}.$$

Then from (6.1) we see that $A = \begin{bmatrix} N & B \end{bmatrix}$, where

$$N = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix},$$

and from (6.2) we have

$$c_{\mathcal{N}} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad \text{and} \quad c_{\mathcal{B}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Substituting these expressions into the definitions of $x^*_{\mathcal{B}}$, $z^*_{\mathcal{N}}$, and ζ^*, we find that

$$\begin{aligned} x^*_{\mathcal{B}} &= B^{-1}b = b \\ z^*_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}} = -c_{\mathcal{N}} \\ \zeta^* &= 0. \end{aligned}$$

Hence, the initial dictionary reads:

$$\begin{aligned} \zeta &= \quad c^T_{\mathcal{N}} x_{\mathcal{N}} \\ x_{\mathcal{B}} &= b - N x_{\mathcal{N}}. \end{aligned}$$

If b has all nonnegative components and $c_{\mathcal{N}}$ has all nonpositive components, then this dictionary is optimal—the problem was trivial. Suppose, however, that one of these two vectors (but not both) has components of the wrong sign. For example, suppose that b is okay (all nonnegative components) but $c_{\mathcal{N}}$ has some positive components. Then this dictionary is primal feasible, and we can start immediately with the

primal simplex method. On the other hand, suppose that $c_{\mathcal{N}}$ has all nonpositive components but b has some negative ones. Then the starting dictionary is dual feasible, and we can commence immediately with the dual simplex algorithm.

The last, and most common, case is where both b and $c_{\mathcal{N}}$ have components of the wrong sign. In this case, we must employ a two-phase procedure. There are two choices. We could temporarily replace $c_{\mathcal{N}}$ with another vector that is nonpositive. Then the modified problem is dual feasible, and so we can apply the dual simplex method to find an optimal solution of this modified problem. After that, the original objective function could be reinstated. With the original objective function, the optimal solution from Phase I is most likely not optimal, but it is feasible, and therefore the primal simplex method can be used to find the optimal solution to the original problem.

The other choice would be to modify b instead of $c_{\mathcal{N}}$, thereby obtaining a primal feasible solution to a modified problem. Then we would use the primal simplex method on the modified problem to obtain its optimal solution, which will then be dual feasible for the original problem, and so the dual simplex method can be used to finish the problem.

6. Negative Transpose Property

In our discussion of duality in Chapter 5, we emphasized the symmetry between the primal problem and its dual. This symmetry can be easily summarized by saying that the dual of a standard-form linear programming problem is the negative transpose of the primal problem. Now, in this chapter, the symmetry appears to have been lost. For example, the basis matrix is an $m \times m$ matrix. Why $m \times m$ and not $n \times n$? It seems strange. In fact, if we had started with the dual problem, added slack variables to it, and introduced a basis matrix on that side it would be an $n \times n$ matrix. How are these two basis matrices related? It turns out that they are not themselves related in any simple way, but the important matrix $B^{-1}N$ is still the negative transpose of the analogous dual construct. The purpose of this section is to make this connection clear.

Consider a standard-form linear programming problem

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \le b \\ & x \ge 0, \end{aligned}$$

and its dual

$$\begin{aligned} \text{minimize} \quad & b^T y \\ \text{subject to} \quad & A^T y \ge c \\ & y \ge 0. \end{aligned}$$

Let w be a vector containing the slack variables for the primal problem, let z be a slack vector for the dual problem, and write both problems in equality form:

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax + w = b \\ &\qquad\qquad x, w \geq 0, \end{aligned}$$

and

$$\begin{aligned} &\text{minimize } b^T y \\ &\text{subject to } A^T y - z = c \\ &\qquad\qquad y, z \geq 0. \end{aligned}$$

Introducing three new notations,

$$\bar{A} = \begin{bmatrix} A & I \end{bmatrix}, \qquad \bar{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}, \qquad \text{and} \qquad \bar{x} = \begin{bmatrix} x \\ w \end{bmatrix},$$

the primal problem can be rewritten succinctly as follows:

$$\begin{aligned} &\text{maximize } \bar{c}^T \bar{x} \\ &\text{subject to } \bar{A}\bar{x} = b \\ &\qquad\qquad \bar{x} \geq 0. \end{aligned}$$

Similarly, using "hats" for new notations on the dual side,

$$\hat{A} = \begin{bmatrix} -I & A^T \end{bmatrix}, \qquad \hat{b} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \qquad \text{and} \qquad \hat{y} = \begin{bmatrix} z \\ y \end{bmatrix},$$

the dual problem can be rewritten in this way:

$$\begin{aligned} &\text{minimize } \hat{b}^T \hat{y} \\ &\text{subject to } \hat{A}\hat{y} = c \\ &\qquad\qquad \hat{y} \geq 0. \end{aligned}$$

Note that the matrix $\bar{A} = [\, A \;\; I \,]$ is an $m \times (n+m)$ matrix. The first n columns of it are the initial nonbasic variables and the last m columns are the initial basic columns. After doing some simplex pivots, the basic and nonbasic columns get jumbled up but we can still write the equality

$$\begin{bmatrix} A & I \end{bmatrix} = \begin{bmatrix} \bar{N} & \bar{B} \end{bmatrix}$$

with the understanding that the equality only holds after rearranging the columns appropriately.

On the dual side, the matrix $\hat{A} = [\, -I \;\; A^T \,]$ is an $n \times (n+m)$ matrix. The first n columns of it are the initial basic variables (for the dual problem) and the last m columns are the initial nonbasic columns. If the same set of pivots that were applied to the primal problem are also applied to the dual, then the columns get rearranged in exactly the same way as they did for the primal and we can write

$$\left[\, -I \;\; A^T \,\right] = \left[\, \hat{B} \;\; \hat{N} \,\right]$$

again with the proviso that the columns of one matrix must be rearranged in a specific manner to bring it into exact equality with the other matrix.

Now, the primal dictionary involves the matrix $\bar{B}^{-1}\bar{N}$ whereas the dual dictionary involves the matrix $\hat{B}^{-1}\hat{N}$. It probably doesn't seem at all obvious that these two matrices are negative transposes of each other. To see that it is so, consider what happens when we multiply $\bar{A}$ by $\hat{A}^T$ in both the permuted notation and the unpermuted notation:

$$\bar{A}\hat{A}^T = \left[\, \bar{N} \;\; \bar{B} \,\right] \left[\begin{array}{c} \hat{B}^T \\ \hat{N}^T \end{array} \right] = \bar{N}\hat{B}^T + \bar{B}\hat{N}^T$$

and

$$\bar{A}\hat{A}^T = \left[\, A \;\; I \,\right] \left[\begin{array}{c} -I \\ A \end{array} \right] = -A + A = 0.$$

These two expressions obviously must agree so we see that

$$\bar{N}\hat{B}^T + \bar{B}\hat{N}^T = 0.$$

Putting the two terms on the opposite sides of the equality sign and multiplying on the right by the inverse of $\hat{B}^T$ and on the left by the inverse of $\bar{B}$, we get that

$$\bar{B}^{-1}\bar{N} = -\left(\hat{B}^{-1}\hat{N}\right)^T,$$

which is the property we wished to establish.

Exercises

6.1 Consider the following linear programming problem:

$$\begin{aligned} \text{maximize} \quad & -6x_1 + 32x_2 - 9x_3 \\ \text{subject to} \quad & -2x_1 + 10x_2 - 3x_3 \leq -6 \\ & x_1 - 7x_2 + 2x_3 \leq 4 \\ & x_1,\ x_2,\ x_3 \geq 0. \end{aligned}$$

Suppose that, in solving this problem, you have arrived at the following dictionary:

$$\begin{aligned} \zeta &= -18 - 3x_4 + 2x_2 \\ \hline x_3 &= 2 - x_4 + 4x_2 - 2x_5 \\ x_1 &= 2x_4 - x_2 + 3x_5. \end{aligned}$$

(a) Which variables are basic? Which are nonbasic?
(b) Write down the vector, $x^*_{\mathcal{B}}$, of current primal basic solution values.
(c) Write down the vector, $z^*_{\mathcal{N}}$, of current dual nonbasic solution values.
(d) Write down $B^{-1}N$.
(e) Is the primal solution associated with this dictionary feasible?
(f) Is it optimal?
(g) Is it degenerate?

6.2 Consider the following linear programming problem:

$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 + 4x_3 + 8x_4 + 16x_5 \\ \text{subject to} \quad & x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \leq 2 \\ & 7x_1 + 5x_2 - 3x_3 - 2x_4 \leq 0 \\ & x_1,\ x_2,\ x_3,\ x_4,\ x_5 \geq 0. \end{aligned}$$

Consider the situation in which x_3 and x_5 are basic and all other variables are nonbasic. Write down:
(a) B,
(b) N,
(c) b,
(d) $c_{\mathcal{B}}$,
(e) $c_{\mathcal{N}}$,
(f) $B^{-1}N$,
(g) $x^*_{\mathcal{B}} = B^{-1}b$,
(h) $\zeta^* = c^T_{\mathcal{B}} B^{-1} b$,
(i) $z^*_{\mathcal{N}} = (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}$,

(j) the dictionary corresponding to this basis.

6.3 Solve the problem in Exercise 2.1 using the matrix form of the primal simplex method.

6.4 Solve the problem in Exercise 2.4 using the matrix form of the dual simplex method.

6.5 Solve the problem in Exercise 2.3 using the two-phase approach in matrix form.

6.6 Find the dual of the following linear program:

$$\begin{array}{lrcl} \text{maximize} & & c^T x & \\ \text{subject to} & a \leq & Ax & \leq b \\ & l \leq & x & \leq u. \end{array}$$

6.7 (a) Let A be a given $m \times n$ matrix, c a given n-vector, and b a given m-vector. Consider the following max-min problem:

$$\max_{x \geq 0} \min_{y \geq 0} \left(c^T x - y^T A x + b^T y \right).$$

By noting that the inner optimization can be carried out explicitly, show that this problem can be reduced to a linear programming problem. Write it explicitly.

(b) What linear programming problem do you get if the min and max are interchanged?

Notes

In this chapter, we have accomplished two tasks: (1) we have expressed the simplex method in matrix notation, and (2) we have reduced the information we carry from iteration to iteration to simply the list of basic variables together with current values of the primal basic variables and the dual nonbasic variables. In particular, it is not necessary to calculate explicitly all the entries of the matrix $B^{-1}N$.

What's in a name? There are times when one thing has two names. So far in this book, we have discussed essentially only one algorithm: the simplex method (assuming, of course, that specific pivot rules have been settled on). But this one algorithm is sometimes referred to as the simplex method and at other times it is referred to as the *revised simplex method.* The distinction being made with this new name has nothing to do with the algorithm. Rather it refers to the specifics of an implementation. Indeed, an implementation of the simplex method that avoids explicit calculation of the matrix $B^{-1}N$ is referred to as an implementation of the revised simplex method. We shall see in Chapter 8 why it is beneficial to avoid computing $B^{-1}N$.

CHAPTER 7

Sensitivity and Parametric Analyses

In this chapter, we consider two related subjects. The first, called sensitivity analysis (or postoptimality analysis) addresses the following question: having found an optimal solution to a given linear programming problem, how much can we change the data and have the current partition into basic and nonbasic variables remain optimal? The second subject addresses situations in which one wishes to solve not just one linear program, but a whole family of problems parametrized by a single real variable.

We shall study parametric analysis in a very specific context in which we wish to find the optimal solution to a given linear programming problem by starting from a problem whose solution is trivially known and then deforming this problem back to the original problem, maintaining as we go optimality of the current solution. The result of this deformation approach to solving a linear programming problem is a new variant of the simplex method, which is called the parametric self-dual simplex method. We will see in later chapters that this variant of the simplex method resembles, in certain respects, the interior-point methods that we shall study.

1. Sensitivity Analysis

One often needs to solve not just one linear programming problem but several closely related problems. There are many reasons that this need might arise. For example, the data that define the problem may have been rather uncertain and one may wish to consider various possible data scenarios. Or perhaps the data are known accurately but change from day to day, and the problem must be resolved for each new day. Whatever the reason, this situation is quite common. So one is led to ask whether it is possible to exploit the knowledge of a previously obtained optimal solution to obtain more quickly the optimal solution to the problem at hand. Of course, the answer is often yes, and this is the subject of this section.

We shall treat a number of possible situations. All of them assume that a problem has been solved to optimality. This means that we have at our disposal the final, optimal dictionary:

$$\begin{aligned} \zeta &= \zeta^* - z_{\mathcal{N}}^{*T} x_{\mathcal{N}} \\ x_{\mathcal{B}} &= x_{\mathcal{B}}^* - B^{-1} N x_{\mathcal{N}}. \end{aligned}$$

Suppose we wish to change the objective coefficients from c to, say, $\tilde{c}$. It is natural to ask how the dictionary at hand could be adjusted to become a valid dictionary for the new problem. That is, we want to maintain the current classification of the variables into basic and nonbasic variables and simply adjust ζ^*, $z^*_{\mathcal{N}}$, and $x^*_{\mathcal{B}}$ appropriately. Recall from (6.7), (6.8), and (6.9) that

$$\begin{aligned} x^*_{\mathcal{B}} &= B^{-1}b, \\ z^*_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}, \\ \zeta^* &= c^T_{\mathcal{B}} B^{-1} b. \end{aligned}$$

Hence, the change from c to $\tilde{c}$ requires us to recompute $z^*_{\mathcal{N}}$ and ζ^*, but $x^*_{\mathcal{B}}$ remains unchanged. Therefore, after recomputing $z^*_{\mathcal{N}}$ and ζ^*, the new dictionary is still primal feasible, and so there is no need for a Phase I procedure: we can jump straight into the primal simplex method, and if $\tilde{c}$ is not too different from c, we can expect to get to the new optimal solution in a relatively small number of steps.

Now suppose that instead of changing c, we wish to change only the right-hand side b. In this case, we see that we need to recompute $x^*_{\mathcal{B}}$ and ζ^*, but $z^*_{\mathcal{N}}$ remains unchanged. Hence, the new dictionary will be dual feasible, and so we can apply the dual simplex method to arrive at the new optimal solution fairly directly.

Therefore, changing just the objective function or just the right-hand side results in a new dictionary having nice feasibility properties. What if we need/want to change some (or all) entries in both the objective function and the right-hand side and maybe even the constraint matrix too? In this case, everything changes: ζ^*, $z^*_{\mathcal{N}}$, $x^*_{\mathcal{B}}$. Even the entries in B and N change. Nonetheless, as long as the new basis matrix B is nonsingular, we can make a new dictionary that preserves the old classification into basic and nonbasic variables. The new dictionary will most likely be neither primal feasible nor dual feasible, but if the changes in the data are fairly small in magnitude, one would still expect that this starting dictionary will get us to an optimal solution in fewer iterations than simply starting from scratch. While there is no guarantee that any of these so-called warm-starts will end up in fewer iterations to optimality, extensive empirical evidence indicates that this procedure often makes a substantial improvement: sometimes the warm-started problems solve in as little as one percent of the time it takes to solve the original problem.

1.1. Ranging. Often one does not wish to solve a modification of the original problem, but instead just wants to ask a hypothetical question:

> *If I were to change the objective function by increasing or decreasing one of the objective coefficients a small amount, how much could I increase/decrease it without changing the optimality of my current basis?*

To study this question, let us suppose that c gets changed to $c + t\Delta c$, where t is a real number and Δc is a given vector (which is often all zeros except for a one in a single entry, but we don't need to restrict the discussion to this case). It is easy to see that $z^*_{\mathcal{N}}$ gets incremented by

$$t\Delta z_{\mathcal{N}},$$

where

$$\Delta z_{\mathcal{N}} = (B^{-1}N)^T \Delta c_{\mathcal{B}} - \Delta c_{\mathcal{N}}. \tag{7.1}$$

Hence, the current basis will remain dual feasible as long as

$$z^*_{\mathcal{N}} + t\Delta z_{\mathcal{N}} \geq 0. \tag{7.2}$$

We've manipulated this type of inequality many times before, and so it should be clear that, for $t > 0$, this inequality will remain valid as long as

$$t \leq \left(\max_{j \in \mathcal{N}} -\frac{\Delta z_j}{z^*_j} \right)^{-1}.$$

Similar manipulations show that, for $t < 0$, the lower bound is

$$t \geq \left(\min_{j \in \mathcal{N}} -\frac{\Delta z_j}{z^*_j} \right)^{-1}.$$

Combining these two inequalities, we see that t must lie in the interval

$$\left(\min_{j \in \mathcal{N}} -\frac{\Delta z_j}{z^*_j} \right)^{-1} \leq t \leq \left(\max_{j \in \mathcal{N}} -\frac{\Delta z_j}{z^*_j} \right)^{-1}.$$

Let us illustrate these calculations with an example. Consider the following linear programming problem:

$$\begin{aligned}
\text{maximize } & 5x_1 + 4x_2 + 3x_3 \\
\text{subject to } & 2x_1 + 3x_2 + x_3 \leq 5 \\
& 4x_1 + x_2 + 2x_3 \leq 11 \\
& 3x_1 + 4x_2 + 2x_3 \leq 8 \\
& x_1,\ x_2,\ x_3 \geq 0\ .
\end{aligned}$$

The optimal dictionary for this problem is given by

$$\begin{aligned} \xi &= 13 - 3x_2 - x_4 - x_6 \\ \hline x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\ x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\ x_5 &= 1 + 5x_2 + 2x_4 \ . \end{aligned}$$

The optimal basis is $\mathcal{B} = \{3,\ 1,\ 5\}$. Suppose we want to know how much the coefficient of 5 on x_1 in the objective function can change without altering the optimality of this basis. From the statement of the problem, we see that

$$c = \begin{bmatrix} 5 & 4 & 3 & 0 & 0 & 0 \end{bmatrix}^T .$$

Since we are interested in changes in the first coefficient, we put

$$\Delta c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T .$$

We partition c according to the final (optimal) basis. Hence, we have

$$\Delta c_{\mathcal{B}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad \Delta c_{\mathcal{N}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} .$$

Next, we need to compute $\Delta z_{\mathcal{N}}$ using (7.1). We could compute $B^{-1}N$ from scratch, but it is easier to extract it from the constraint coefficients in the final dictionary. Indeed,

$$-B^{-1}N = \begin{bmatrix} 1 & 3 & -2 \\ -2 & -2 & 1 \\ 5 & 2 & 0 \end{bmatrix} .$$

Hence, from (7.1) we see that

$$\Delta z_{\mathcal{N}} = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix} .$$

Now, (7.2) gives the condition on t. Writing it out componentwise, we get

$$3 + 2t \geq 0, \qquad 1 + 2t \geq 0, \qquad \text{and} \qquad 1 - t \geq 0.$$

These three inequalities, which must all hold, can be summarized by saying that

$$-\frac{1}{2} \leq t \leq 1$$

Hence, in terms of the coefficient on x_1, we finally see that it can range from 4.5 to 6.

Now suppose we change b to $b + t\Delta b$ and ask how much t can change before the current basis becomes nonoptimal. In this case, $z^*_{\mathcal{N}}$ does not change, but $x^*_{\mathcal{B}}$ gets incremented by $t\Delta x_{\mathcal{B}}$, where

$$\Delta x_{\mathcal{B}} = B^{-1}\Delta b.$$

Hence, the current basis will remain optimal as long as t lies in the interval

$$\left(\min_{i\in\mathcal{B}} -\frac{\Delta x_i}{x^*_i}\right)^{-1} \leq t \leq \left(\max_{i\in\mathcal{B}} -\frac{\Delta x_i}{x^*_i}\right)^{-1}.$$

2. Parametric Analysis and the Homotopy Method

In this section, we illustrate the notion of *parametric analysis* by applying a technique called the *homotopy method* to get a new algorithm for solving linear programming problems. The homotopy method is a general technique in which one creates a continuous deformation that changes a given difficult problem into a related but trivially solved problem and then attempts to work backwards from the trivial problem to the difficult problem by solving (hopefully without too much effort) all the problems in between. Of course, there is a continuum of problems between the hard one and the trivial one, and so we shouldn't expect that this technique will be effective in every situation; but for linear programming and for many other problem domains, it turns out to yield efficient algorithms.

We start with an example. Suppose we wish to solve the following linear programming problem:

$$\begin{aligned}
\text{maximize} \quad & -2x_1 + 3x_2 \\
\text{subject to} \quad & -x_1 + x_2 \leq -1 \\
& -x_1 - 2x_2 \leq -2 \\
& x_2 \leq 1 \\
& x_1,\ x_2 \geq 0.
\end{aligned}$$

The starting dictionary is

$$\begin{array}{rrrrr} \zeta = & & -2x_1 & -(-3)x_2 & \\ \hline x_3 = & -1 + & x_1 & - & x_2 \\ x_4 = & -2 + & x_1 & + & 2x_2 \\ x_5 = & 1 & & - & x_2\,. \end{array}$$

This dictionary is neither primal nor dual feasible. Let's perturb it by adding a positive real number μ to each right-hand side and subtracting it from each objective function coefficient. We now arrive at a family of dictionaries, parametrized by μ:

$$\begin{array}{rrrrr} \zeta = & & -(2+\mu)x_1 & -(-3+\mu)x_2 & \\ \hline x_3 = & -1+\mu + & x_1 & - & x_2 \\ x_4 = & -2+\mu + & x_1 & + & 2x_2 \\ x_5 = & 1+\mu & & - & x_2\,. \end{array} \tag{7.3}$$

Clearly, for μ sufficiently large, specifically $\mu \geq 3$, this dictionary is both primal and dual feasible. Hence, the associated solution $x = [0, 0, -1+\mu, -2+\mu, 1+\mu]$ is optimal. Starting with μ large, we reduce it as much as we can while keeping dictionary (7.3) optimal. This dictionary will become nonoptimal as soon as $\mu < 3$, since the associated dual variable $y_2^* = -3+\mu$ will become negative. In other words, the coefficient of x_2, which is $3-\mu$, will become positive. This change of sign on the coefficient of x_2 suggests that we make a primal pivot in which x_2 enters the basis. The usual ratio test (using the specific value of $\mu = 3$) indicates that x_3 must be the leaving variable. Making the pivot, we get

$$\begin{array}{rrrrr} \zeta = & -3+4\mu-\mu^2 & -(-1+2\mu)x_1 & -(3-\mu)x_3 & \\ \hline x_2 = & -1+\mu & + & x_1 - & x_3 \\ x_4 = & -4+3\mu & + & 3x_1 - & 2x_3 \\ x_5 = & 2 & + & x_1 + & x_3\,. \end{array}$$

This dictionary is optimal as long as

$$\begin{aligned} -1+2\mu \geq 0, \qquad & 3-\mu \geq 0, \\ -1+\mu \geq 0, \qquad & -4+3\mu \geq 0. \end{aligned}$$

These inequalities reduce to

$$\frac{4}{3} \leq \mu \leq 3.$$

So now we can reduce μ from its current value of 3 down to $4/3$. If we reduce it below $4/3$, the primal feasibility inequality, $-4 + 3\mu \geq 0$, becomes violated. This violation suggests that we perform a dual pivot with x_4 serving as the leaving variable. The usual (dual) ratio test (with $\mu = 4/3$) then tells us that x_1 must be the entering variable. Doing the pivot, we get

$$\begin{array}{rrrrr}
\zeta = & -\frac{5}{3} + \frac{1}{3}\mu + \mu^2 & - & (-\frac{1}{3} + \frac{2}{3}\mu)x_4 & - (\frac{7}{3} + \frac{1}{3}\mu)x_3 \\
\hline
x_2 = & \frac{1}{3} & + & \frac{1}{3}x_4 - & \frac{1}{3}x_3 \\
x_1 = & \frac{4}{3} - \mu & + & \frac{1}{3}x_4 + & \frac{2}{3}x_3 \\
x_5 = & \frac{2}{3} + \mu & - & \frac{1}{3}x_4 + & \frac{1}{3}x_3 \,.
\end{array}$$

Now the conditions for optimality are

$$-\frac{1}{3} + \frac{2}{3}\mu \geq 0, \qquad \frac{7}{3} + \frac{1}{3}\mu \geq 0,$$
$$\frac{4}{3} - \mu \geq 0, \qquad \frac{2}{3} + \mu \geq 0,$$

which reduce to

$$\frac{1}{2} \leq \mu \leq \frac{4}{3}.$$

For the next iteration, we reduce μ to $1/2$ and see that the inequality that becomes binding is the dual feasibility inequality

$$-\frac{1}{3} + \frac{2}{3}\mu \geq 0.$$

Hence, we do a primal pivot with x_4 entering the basis. The leaving variable is x_5, and the new dictionary is

$$\begin{array}{rrrrr}
\zeta = & -1 & - \mu^2 - (1 - 2\mu)x_5 & - (2 + \mu)x_3 \\
\hline
x_2 = & 1 + \mu & - & x_5 & \\
x_1 = & 2 & - & x_5 + & x_3 \\
x_4 = & 2 + 3\mu & - & 3x_5 + & x_3 \,.
\end{array}$$

For this dictionary, the range of optimality is given by

$$1 - 2\mu \geq 0, \qquad 2 + \mu \geq 0,$$
$$1 + \mu \geq 0, \qquad 2 + 3\mu \geq 0,$$

which reduces to

$$-\frac{2}{3} \le \mu \le \frac{1}{2}.$$

This range covers $\mu = 0$, and so now we can set μ to 0 and get an optimal dictionary for our original problem:

$$\begin{aligned} \zeta &= -1 - x_5 - 2x_3 \\ \hline x_2 &= 1 - x_5 \\ x_1 &= 2 - x_5 + x_3 \\ x_4 &= 2 - 3x_5 + x_3\,. \end{aligned}$$

The algorithm we have just illustrated is called the *parametric self-dual simplex method*.[1] We shall often refer to it more simply as the self-dual simplex method. It has some attractive features. First, in contrast to the methods presented earlier, this algorithm does not require a separate Phase I procedure. It starts with any problem, be it primal infeasible, dual infeasible, or both, and it systematically performs pivots (whether primal or dual) until it finds an optimal solution.

A second feature is that a trivial modification of the algorithm can avoid entirely ever encountering a degenerate dictionary. Indeed, suppose that, instead of adding/subtracting μ from each of the right-hand sides and objective coefficients, we add/subtract a positive constant times μ. Suppose further that the positive constant is different in each addition/subtraction. In fact, suppose that they are chosen independently from, say, a uniform distribution on $[1/2, 3/2]$. Then with probability one, the algorithm will produce no primal degenerate or dual degenerate dictionary in any iteration. In Chapter 3, we discussed perturbing the right-hand side of a linear programming problem to avoid degeneracy in the primal simplex method, but back then the perturbation changed the problem. The present perturbation does not in any way affect the problem that is solved.

With the above randomization trick to resolve the degeneracy issue, the analysis of the convergence of the algorithm is straightforward. Indeed, let us consider a problem that is feasible and bounded (the questions regarding feasibility and boundedness are addressed in Exercise 7.10). For each nondegenerate pivot, the next value of μ will be strictly less than the current value. Since each of these μ values is determined by a partition of the variables into basics and nonbasics and there are only a finite number of such partitions, it follows that the method must reach a partition with a negative μ value in a finite number of steps.

[1] In the first edition, this method was called the primal–dual simplex method.

3. The Parametric Self-Dual Simplex Method

In the previous section, we illustrated on an example a new algorithm for solving linear programming problems, called the parametric self-dual simplex method. In this section, we shall lay out the algorithm in matrix notation.

Our starting point is an initial dictionary as written in (6.10) and transcribed here for convenience:

$$\begin{aligned}\zeta &= \zeta^* - {z^*_{\mathcal{N}}}^T x_{\mathcal{N}}\\ x_{\mathcal{B}} &= x^*_{\mathcal{B}} - B^{-1}Nx_{\mathcal{N}},\end{aligned}$$

where

$$\begin{aligned}x^*_{\mathcal{B}} &= B^{-1}b\\ z^*_{\mathcal{N}} &= (B^{-1}N)^T c_{\mathcal{B}} - c_{\mathcal{N}}\\ \zeta^* &= c^T_{\mathcal{B}} x^*_{\mathcal{B}} = c^T_{\mathcal{B}} B^{-1} b.\end{aligned}$$

Generally speaking, we don't expect this dictionary to be either primal or dual feasible. So we perturb it by adding essentially arbitrary perturbations $\bar{x}_{\mathcal{B}}$ and $\bar{z}_{\mathcal{N}}$ to $x^*_{\mathcal{B}}$ and $z^*_{\mathcal{N}}$, respectively:

$$\begin{aligned}\zeta &= \zeta^* - (z^*_{\mathcal{N}} + \mu\bar{z}_{\mathcal{N}})^T x_{\mathcal{N}}\\ x_{\mathcal{B}} &= (x^*_{\mathcal{B}} + \mu\bar{x}_{\mathcal{B}}) - B^{-1}Nx_{\mathcal{N}}.\end{aligned}$$

We assume that the perturbations are all strictly positive,

$$\bar{x}_{\mathcal{B}} > 0 \qquad \text{and} \qquad \bar{z}_{\mathcal{N}} > 0,$$

so that by taking μ sufficiently large the perturbed dictionary will be optimal. (Actually, to guarantee optimality for large μ, we only need to perturb those primal and dual variables that are negative in the initial dictionary.)

The parametric self-dual simplex method generates a sequence of dictionaries having the same form as the initial one—except, of course, the basis $\mathcal{B}$ will change, and hence all the data vectors ($z^*_{\mathcal{N}}$, $\bar{z}_{\mathcal{N}}$, $x^*_{\mathcal{B}}$, and $\bar{x}_{\mathcal{B}}$) will change too. Additionally, the current value of the objective function ζ^* will, with the exception of the first dictionary, depend on μ.

One step of the self-dual simplex method can be described as follows. First, we compute the smallest value of μ for which the current dictionary is optimal. Letting μ^* denote this value, we see that

$$\mu^* = \min\{\mu : z^*_{\mathcal{N}} + \mu\bar{z}_{\mathcal{N}} \geq 0 \text{ and } x^*_{\mathcal{B}} + \mu\bar{x}_{\mathcal{B}} \geq 0\}.$$

There is either a $j \in \mathcal{N}$ for which $z^*_j + \mu^*\bar{z}_j = 0$ or an $i \in \mathcal{B}$ for which $x^*_i + \mu^*\bar{x}_i = 0$ (if there are multiple choices, an arbitrary selection is made). If the blocking constraint

corresponds to a nonbasic index $j \in \mathcal{N}$, then we do one step of the primal simplex method. If, on the other hand, it corresponds to a basic index $i \in \mathcal{B}$, then we do one step of the dual simplex method.

Suppose, for definiteness, that the blocking constraint corresponds to an index $j \in \mathcal{N}$. Then, to do a primal pivot, we declare x_j to be the entering variable, and we compute the step direction for the primal basic variables as the jth column of the dictionary. That is,

$$\Delta x_{\mathcal{B}} = B^{-1} N e_j.$$

Using this step direction, we find an index $i \in \mathcal{B}$ that achieves the maximal value of $\Delta x_i/(x_i^* + \mu^* \bar{x}_i)$. Variable x_i is the leaving variable. After figuring out the leaving variable, the step direction vector for the dual nonbasic variables is just the negative of the ith row of the dictionary

$$\Delta z_{\mathcal{N}} = -(B^{-1} N)^T e_i.$$

After computing the primal and dual step directions, it is easy to see that the step length adjustments are given by

$$t = \frac{x_i^*}{\Delta x_i}, \qquad \bar{t} = \frac{\bar{x}_i}{\Delta x_i},$$

$$s = \frac{z_j^*}{\Delta z_j}, \qquad \bar{s} = \frac{\bar{z}_j}{\Delta z_j}.$$

And from these, it is easy to write down the new solution vectors:

$$x_j^* \leftarrow t, \quad \bar{x}_j \leftarrow \bar{t}, \quad z_i^* \leftarrow s, \quad \bar{z}_i \leftarrow \bar{s},$$

$$x_{\mathcal{B}}^* \leftarrow x_{\mathcal{B}}^* - t\Delta x_{\mathcal{B}}, \qquad \bar{x}_{\mathcal{B}} \leftarrow \bar{x}_{\mathcal{B}} - \bar{t}\Delta x_{\mathcal{B}},$$

$$z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}}, \qquad \bar{z}_{\mathcal{N}} \leftarrow \bar{z}_{\mathcal{N}} - \bar{s}\Delta z_{\mathcal{N}}.$$

Finally, the basis is updated by adding the entering variable and removing the leaving variable

$$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}.$$

The algorithm is summarized in Figure 7.1.

Exercises

In solving the following problems, the advanced pivot tool can be used to check your arithmetic:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/advanced.html

Compute $\mu^* = \max\left(\max_{\substack{j\in\mathcal{N}\\ \bar{z}_j>0}} -\frac{z_j^*}{\bar{z}_j}, \max_{\substack{i\in\mathcal{B}\\ \bar{x}_i>0}} -\frac{x_i^*}{\bar{x}_i}\right)$

While $(\mu^* > 0)$ {

If max is achieved by

$j \in \mathcal{N}$:	$i \in \mathcal{B}$:
$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$	$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$
pick $i \in \text{argmax}_{i\in\mathcal{B}} \frac{\Delta x_i}{x_i^* + \mu^* \bar{x}_i}$	pick $j \in \text{argmax}_{j\in\mathcal{N}} \frac{\Delta z_j}{z_j^* + \mu^* \bar{z}_j}$
$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i$	$\Delta x_{\mathcal{B}} = B^{-1}Ne_j$

$$t = \frac{x_i^*}{\Delta x_i} \quad \bar{t} = \frac{\bar{x}_i}{\Delta x_i}$$
$$s = \frac{z_j^*}{\Delta z_j} \quad \bar{s} = \frac{\bar{z}_j}{\Delta z_j}$$

$$x_j^* \leftarrow t \quad \bar{x}_j \leftarrow \bar{t}$$
$$z_i^* \leftarrow s \quad \bar{z}_i \leftarrow \bar{s}$$

$$x_{\mathcal{B}}^* \leftarrow x_{\mathcal{B}}^* - t\Delta x_{\mathcal{B}} \quad \bar{x}_{\mathcal{B}} \leftarrow \bar{x}_{\mathcal{B}} - \bar{t}\Delta x_{\mathcal{B}}$$
$$z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}} \quad \bar{z}_{\mathcal{N}} \leftarrow \bar{z}_{\mathcal{N}} - \bar{s}\Delta z_{\mathcal{N}}$$

$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$

Recompute μ^* as above

}

FIGURE 7.1. The parametric self-dual simplex method.

7.1 The final dictionary for

$$\begin{array}{lrcl}
\text{maximize} & x_1 + 2x_2 + x_3 + x_4 & & \\
\text{subject to} & 2x_1 + x_2 + 5x_3 + x_4 & \leq & 8 \\
& 2x_1 + 2x_2 + 4x_4 & \leq & 12 \\
& 3x_1 + x_2 + 2x_3 & \leq & 18 \\
& x_1,\ x_2,\ x_3,\ x_4 & \geq & 0
\end{array}$$

is

$$\begin{array}{rl}
\zeta = & 12.4 - 1.2x_1 - 0.2x_5 - 0.9x_6 - 2.8x_4 \\
\hline
x_2 = & 6 - x_1 - 0.5x_6 - 2x_4 \\
x_3 = & 0.4 - 0.2x_1 - 0.2x_5 + 0.1x_6 + 0.2x_4 \\
x_7 = & 11.2 - 1.6x_1 + 0.4x_5 + 0.3x_6 + 1.6x_4.
\end{array}$$

(the last three variables are the slack variables).

(a) What will be an optimal solution to the problem if the objective function is changed to

$$3x_1 + 2x_2 + x_3 + x_4?$$

(b) What will be an optimal solution to the problem if the objective function is changed to

$$x_1 + 2x_2 + 0.5x_3 + x_4?$$

(c) What will be an optimal solution to the problem if the second constraint's right-hand side is changed to 26?

7.2 For each of the objective coefficients in the problem in Exercise 7.1, find the range of values for which the final dictionary will remain optimal.

7.3 Consider the following dictionary which arises in solving a problem using the self-dual simplex method:

$$\begin{array}{rl}
\zeta = & -3 - (-1 + 2\mu)x_1 - (3 - \mu)x_3 \\
\hline
x_2 = & -1 + \mu + x_1 - x_3 \\
x_4 = & -4 + 3\mu + 3x_1 - 2x_3 \\
x_5 = & 2 + x_1 + x_3\,.
\end{array}$$

(a) For which values of μ is the current dictionary optimal?

(b) For the next pivot in the self-dual simplex method, identify the entering and the leaving variable.

7.4 Solve the linear program given in Exercise 2.3 using the self-dual simplex method. *Hint: It is easier to use dictionary notation than matrix notation.*

7.5 Solve the linear program given in Exercise 2.4 using the self-dual simplex method. *Hint: It is easier to use dictionary notation than matrix notation.*

7.6 Solve the linear program given in Exercise 2.6 using the self-dual simplex method. *Hint: It is easier to use dictionary notation than matrix notation.*

7.7 Using today's date (MMYY) for the seed value, solve 10 problems using the self-dual simplex method:

campuscgi.princeton.edu/∼rvdb/JAVA/pivot/pd1phase.html .

7.8 Use the self-dual simplex method to solve the following problem:

$$\begin{aligned} \text{maximize} \quad & 3x_1 - x_2 \\ \text{subject to} \quad & x_1 - x_2 \leq 1 \\ & -x_1 + x_2 \leq -4 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

7.9 Let P_μ denote the perturbed primal problem (with perturbation μ). Show that if P_μ is infeasible, then $P_{\mu'}$ is infeasible for every $\mu' \leq \mu$. State and prove an analogous result for the perturbed dual problem.

7.10 Using the notation of Figure 7.1 state precise conditions for detecting infeasibility and/or unboundedness in the self-dual simplex method.

7.11 Consider the following one parameter family of linear programming problems (parametrized by μ):

$$\begin{aligned} \max \quad & (4-4\mu)x_0 - 2x_1 - 2x_2 - 2x_3 - 2x_4 \\ \text{s.t.} \quad & x_0 - x_1 \leq 1 \\ & x_0 - x_2 \leq 2 \\ & x_0 - x_3 \leq 4 \\ & x_0 - x_4 \leq 8 \\ & x_0,\ x_1,\ x_2,\ x_3,\ x_4 \geq 0. \end{aligned}$$

Starting from $\mu = \infty$, use the parametric simplex method to decrease μ until you get to $\mu = -\infty$. *Hint: the pivots are straight forward and, after the first couple, a clear pattern should emerge which will make the subsequent*

pivots easy. Clearly indicate the range of μ values for which each dictionary is optimal.

Notes

Parametric analysis has its roots in Gass & Saaty (1955). G.B. Dantzig's classic book (Dantzig 1963) describes the self-dual simplex method under the name of the *self-dual parametric simplex method.* It is a special case of "Lemke's algorithm" for the linear complementarity problem (Lemke 1965) (see Exercise 17.7). Smale (1983) and Borgwardt (1982) were first to realize that the parametric self-dual simplex method is amenable to probabilistic analysis. For a more recent discussion of homotopy methods and the parametric self-dual simplex method, see Nazareth (1986) and Nazareth (1987).

CHAPTER 8

Implementation Issues

In the previous chapter, we rewrote the simplex method using matrix notation. This is the first step toward our aim of describing the simplex method as one would implement it as a computer program. In this chapter, we shall continue in this direction by addressing some important implementation issues.

The most time-consuming steps in the simplex method are the computations

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j \qquad \text{and} \qquad \Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i,$$

and the difficulty in these steps arises from the B^{-1}. Of course, we don't ever actually compute the inverse of the basis matrix. Instead, we calculate, say, $\Delta x_{\mathcal{B}}$ by solving the following system of equations:

$$B\Delta x_{\mathcal{B}} = a_j, \tag{8.1}$$

where

$$a_j = Ne_j$$

is the column of N associated with nonbasic variable x_j.

Similarly, the calculation of $\Delta z_{\mathcal{N}}$ is also broken into two steps:

$$\begin{aligned} B^T v &= e_i, \\ \Delta z_{\mathcal{N}} &= -N^T v. \end{aligned} \tag{8.2}$$

Here, the first step is the solution of a large system of equations, this time involving B^T instead of B, and the second step is the comparatively trivial task of multiplying a vector on the left by the matrix $-N^T$.

Solving the systems of equations (8.1) and (8.2) is where most of the complexity of a simplex iteration lies. We discuss solving such systems in the first two sections. In the second section, we look at the effect of sparsity on these systems. The next few sections explain how to reuse and/or update the computations of one iteration in subsequent iterations. In the final sections, we address a few other issues that affect the efficiency of an implementation.

1. Solving Systems of Equations: LU-Factorization

In this section, we discuss solving systems of equations of the form

$$Bx = b,$$

where B is an invertible $m \times m$ matrix and b is an arbitrary m-vector. (Analysis of the transpose $B^T x = b$ is left to Exercise 8.4.) Our first thought is to use Gaussian elimination. This idea is correct, but to explain how Gaussian elimination is actually implemented, we need to take a fresh look at how it works. To explain, let us consider an example:

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

(Note that, to emphasize the importance of sparsity, zero entries are simply left blank.) In Gaussian elimination, one begins by subtracting appropriate multiples of the first row from each subsequent row to get zeros in the first column below the diagonal. For our specific example, we subtract $3/2$ times the first row from the second row and we subtract $-1/2$ times the first row from the third row. The result is

$$\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix}.$$

Shortly, we will want to remember the values of the nonzero elements in the first column. Therefore, let us agree to do the row operations that are required to eliminate nonzeros, but when we write down the result of the elimination, we will leave the nonzeros there. With this convention, the result of the elimination of the first column can be written as

$$\left[\begin{array}{cccccc} 2 & & 4 & & -2 \\ \hline 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{array}\right].$$

Note that we have drawn a line to separate the eliminated top/left parts of the matrix from the uneliminated lower-right part.

Next, we eliminate the nonzeros below the second diagonal (there's only one) by subtracting an appropriate multiple of the second row from each subsequent row. Again, we write the answer without zeroing out the eliminated elements:

$$\left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -3 \\ & & 1 & & 4 \end{array}\right].$$

After eliminating the third column, we get

$$\left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -21 \\ & & 1 & & 7 \end{array}\right].$$

Now, the remaining uneliminated part is already an upper triangular matrix, and hence no more elimination is required.

At this point, you are probably wondering how this strangely produced matrix is related to the original matrix B. The answer is both simple and elegant. First, take the final matrix and split it into three matrices: the matrix consisting of all elements on or below the diagonal, the matrix consisting of just the diagonal elements, and the matrix consisting of all elements on or above the diagonal. It is amazing but true that B is simply the product of the resulting lower triangular matrix times the inverse of the diagonal matrix times the upper triangular matrix:

$$B = \left[\begin{array}{ccccc} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{array}\right] \left[\begin{array}{ccccc} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{array}\right]^{-1} \left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{array}\right].$$

(If you don't believe it, multiply them and see.) Normally, the product of the lower triangular matrix and the diagonal matrix is denoted by L,

$$L = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix},$$

and the upper triangular matrix is denote by U:

$$U = \begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}.$$

The resulting representation,

$$B = LU,$$

is called an LU-*factorization* of B. Finding an LU-factorization is equivalent to Gaussian elimination in the sense that multiplying B on the left by L^{-1} has the effect of applying row operations to B to put it into upper-triangular form U.

The value of an LU-factorization is that it can be used to solve systems of equations. For example, suppose that we wish to solve Equation (8.1), where B is as above and

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}. \tag{8.3}$$

First, we substitute LU for B so that the system becomes

$$LU\Delta x_{\mathcal{B}} = a_j.$$

Now, if we let $y = U\Delta x_{\mathcal{B}}$, then we can solve

$$Ly = b$$

for y, and once y is known, we can solve

$$U\Delta x_{\mathcal{B}} = y$$

for $\Delta x_{\mathcal{B}}$. Because L is lower triangular, solving $Ly = b$ is easy. Indeed, writing the system out,

$$\begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix},$$

we notice immediately that $y_1 = 7$. Then, given y_1, it becomes clear from the second equation that $y_2 = -2 - (3/2)y_1 = -25/2$. Continuing in this way, we find that

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ -\frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}.$$

The process of successively solving for the elements of the vector y starting with the first and proceeding to the last is called *forward substitution*.

Of course, solving $U\Delta x_{\mathcal{B}} = y$ is easy too, since U is upper triangular. The system to solve is given by

$$\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ -\frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

(note that, to keep notations simple, we are assuming that the basic indices are 1 through 5 so that $\Delta x_{\mathcal{B}} = (\Delta x_1, \Delta x_2, \Delta x_3, \Delta x_4, \Delta x_5)$). This time we start with the last equation and see that $\Delta x_5 = -1/2$. Then the second to last equation tells us that

$\Delta x_4 = 23/2 + 21(\Delta x_5) = 1$. After working our way to the first equation, we have

$$\Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}.$$

This process of working from the last element of $\Delta x_{\mathcal{B}}$ back to the first is called *backward substitution.*

2. Exploiting Sparsity

In the previous section, we took a specific matrix B and constructed an LU factorization of it. However, with that example we were lucky in that every diagonal element was nonzero at the moment it was used to eliminate the nonzeros below it. Had we encountered a zero diagonal element, we would have been forced to rearrange the columns and/or the rows of the matrix to put a nonzero element in this position. For a random matrix (whatever that means), the odds of encountering a zero are nil, but a basis matrix can be expected to have plenty of zeros in it, since, for example, it is likely to contain columns associated with slack variables, which are all zero except for one 1. A matrix that contains zeros is called a *sparse matrix*.

When a sparse matrix has lots of zeros, two things happen. First, the chances of being required to make row and/or column permutations is high. Second, additional computational efficiency can be obtained by making further row and/or column permutations with the aim of keeping L and/or U as sparse as possible.

The problem of finding the "best" permutation is, in itself, harder than the linear programming problem that we ultimately wish to solve. But there are simple heuristics that help to preserve sparsity in L and U. We shall focus on just one such heuristic, called the *minimum-degree* ordering heuristic, which is describe as follows:

> *Before eliminating the nonzeros below a diagonal "pivot" element, scan all uneliminated rows and select the sparsest row, i.e., that row having the fewest nonzeros in its uneliminated part (ties can be broken arbitrarily). Swap this row with the pivot row. Then scan the uneliminated nonzeros in this row and select that one whose column has the fewest nonzeros in its uneliminated part. Swap this column with the pivot column so that this nonzero becomes the pivot element. (Of course, provisions should be made to reject such a pivot element if its value is close to zero.)*

As a matter of terminology, the number of nonzeros in the uneliminated part of a row/column is called the *degree* of the row/column. Hence, the name of the heuristic.

Let's apply the minimum-degree heuristic to the LU-factorization of the matrix B studied in the previous section. To keep track of the row and column permutations, we will indicate original row indices on the left and original column indices across the top. Hence, we start with:

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix} \end{array}.$$

To begin, row 4 has the fewest nonzeros, and within row 4, the -1 in column 2 belongs to the column with the fewest nonzeros. Hence, we swap rows 1 and 4 and we swap columns 1 and 2 to rewrite B as

$$B = \begin{array}{c} \\ 4 \\ 2 \\ 3 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & & & & -6 \\ 1 & 3 & & 1 & \\ & -1 & -1 & & -2 \\ & 2 & 4 & & -2 \\ & & 1 & & 4 \end{bmatrix} \end{array}.$$

Now, we eliminate the nonzeros under the first diagonal element (and, as before, we leave the eliminated nonzeros as they were). The result is

$$\begin{array}{c} \\ 4 \\ 2 \\ 3 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ \hline 1 & 3 & & 1 & -6 \\ & -1 & -1 & & -2 \\ & 2 & 4 & & -2 \\ & & 1 & & 4 \end{array}\right] \end{array}.$$

Before doing the elimination associated with the second diagonal element, we note that row 5 is the row with minimum degree, and within row 5, the element 1 in column 3 has minimum column degree. Hence, we swap rows 2 and 5 and we swap

columns 1 and 3 to get

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & -2 \\ & 4 & 2 & & -2 \\ 1 & & 3 & 1 & -6 \end{array}\right] \end{array}.$$

Now we eliminate the nonzeros under the second diagonal element to get

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & 2 \\ & 4 & 2 & & -18 \\ 1 & & 3 & 1 & -6 \end{array}\right] \end{array}.$$

For the third stage of elimination, note that row 3 is a minimum-degree row and that, among the nonzero elements of that row, the -1 is in a minimum-degree column. Hence, for this stage no permutations are needed. The result of the elimination is

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & 2 \\ & 4 & 2 & & -14 \\ 1 & & 3 & 1 & \end{array}\right] \end{array}.$$

For the next stage of the elimination, both of the remaining two rows have the same degree, and hence we don't need to swap rows. But we do need to swap columns

5 and 4 to put the -14 into the diagonal position. The result of the swap is

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & \\ & 1 & & 4 & \\ & -1 & -1 & 2 & \\ & 4 & 2 & -14 & \\ 1 & & 3 & & 1 \end{array}\right] \end{array}.$$

At this point, we notice that the remaining 2×2 uneliminated part of the matrix is already upper triangular (in fact, diagonal), and hence no more elimination is needed.

With the elimination completed, we can extract the matrices L and U in the usual way:

$$L = \begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \\ \left[\begin{array}{ccccc} -1 & & & & \\ & 1 & & & \\ & -1 & -1 & & \\ & 4 & 2 & -14 & \\ 1 & & 3 & & 1 \end{array}\right] \end{array} \begin{array}{c} \\ \left[\begin{array}{ccccc} -1 & & & & \\ & 1 & & & \\ & & -1 & & \\ & & & -\frac{1}{14} & \\ & & & & 1 \end{array}\right] \end{array}$$

$$= \begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \\ \left[\begin{array}{ccccc} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{array}\right] \end{array},$$

and

$$U = \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{array}\right] \end{array}.$$

(Note that the columns of L and the rows of U do not have any "original" indices associated with them, and so no permutation is indicated across the top of L or down the left side of U.)

This LU-factorization has five off-diagonal nonzeros in L and three off-diagonal nonzeros in U for a total of eight off-diagonal nonzeros. In contrast, the LU factorization from the previous section had a total of 12 off-diagonal nonzeros. Hence, the minimum-degree ordering heuristic paid off for this example by reducing the number of nonzeros by 33%. While such a reduction may not seem like a big deal for small matrices such as our 5×5 example, for large matrices the difference can be dramatic.

The fact that we have permuted the rows and columns to get this factorization has only a small impact on how one uses the factorization to solve systems of equations. To illustrate, let us solve the same system that we considered before: $B\Delta x_{\mathcal{B}} = a_j$, where a_j is given by (8.3). The first step in solving this system is to permute the rows of a_j so that they agree with the rows of L and then to use forward substitution to solve the system $Ly = a_j$. Writing it out, the system looks like this:

$$\begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ -2 \end{bmatrix}$$

The result of the forward substitution is that

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{bmatrix}. \tag{8.4}$$

The next step is to solve the system $U\Delta x_{\mathcal{B}} = y$. Writing this system out, we get

$$\begin{array}{c} \begin{matrix} 2 & 3 & 1 & 5 & 4 \end{matrix} \\ \begin{bmatrix} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{bmatrix} \end{array} \begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array} \begin{bmatrix} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{bmatrix}.$$

Using backward substitution, we see that

$$\begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array} \begin{bmatrix} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ -1 \\ -\frac{1}{2} \\ 1 \end{bmatrix}.$$

Finally, we rewrite the solution listing the elements of $\Delta x_{\mathcal{B}}$ in their original order:

$$\Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}.$$

Of course, the answer obtained here agrees with the one obtained at the end of the previous section.

Even with good fill-in minimizing heuristics such as minimum-degree, the LU-factorization remains a significant computational bottleneck. To see why, consider for the moment dense matrices. If we were to write a subroutine to carry out an LU-factorization, we would find that the main body of the routine would have a big triply nested loop:

```
for each column index j {
    for each remaining row index i {
        for each remaining column index k {
            update the (i,k) entry in accordance with
            the aim to make the (i,j) entry be zero
        }
    }
}
```

Since each of these loops involves approximately m steps, the LU-factorization routine requires about m^3 operations and hence is called an order m^3 algorithm. Similar considerations tell us that the forward and backward substitutions are both order m^2 algorithms. This means that forward and backward substitution can be done much faster than LU-factorization. Indeed, if $m = 5000$, then factorization takes a couple of thousand times longer than a forward or backward substitution. Of course, this argument is for dense matrices. But for sparse matrices a similar, if less dramatic, effect is seen. Typically, for sparse matrices, one expects that factorization will take from 10 to 100 times longer than substitution. Therefore, it is important to perform as few LU-factorizations as possible. This is the subject of the next section.

3. Reusing a Factorization

In the previous two sections, we showed how to use an LU-factorization of B to solve the system of equations

$$B\Delta x_{\mathcal{B}} = a_j$$

for the primal step direction $\Delta x_{\mathcal{B}}$. Since the basis matrix doesn't change much from one iteration of the simplex method to the next (columns get replaced by new ones one at a time), we ask whether the LU-factorization of B from the current iteration might somehow be used again to solve the systems of equations that arise in the next iteration (or even the next several iterations).

Let B denote the current basis (for which a factorization has already been computed) and let $\tilde{B}$ denote the basis of the next iteration. Then $\tilde{B}$ is simply B with the column that holds the column vector a_i associated with the leaving variable x_i replaced by a new column vector a_j associated with the entering variable x_j. This verbal description can be converted into a formula:

$$\tilde{B} = B + (a_j - a_i)e_i^T. \tag{8.5}$$

Here, as before, e_i denotes the vector that is all zeros except for a one in the position associated with index i—to be definite, let us say that this position is the pth position in the vector. To see why this formula is correct, it is helpful to realize that a column vector, say a, times e_i^T produces a matrix that is all zero except for the pth column, which contains the column vector a.

Since the basis B is invertible, (8.5) can be rewritten as

$$\tilde{B} = B\left(I + B^{-1}(a_j - a_i)e_i^T\right).$$

Denote the matrix in parentheses by E. Recall that $a_j = Ne_j$, since it is the column vector from A associated with the entering variable x_j. Hence,

$$B^{-1}a_j = B^{-1}Ne_j = \Delta x_{\mathcal{B}},$$

which is a vector we need to compute in the current iteration anyway. Also,

$$B^{-1}a_i = e_i,$$

since a_i is the column of B associated with the leaving variable x_i. Therefore, we can write E more simply as

$$E = I + (\Delta x_{\mathcal{B}} - e_i)e_i^T.$$

Now, if E has a simple inverse, then we can use it together with the LU-factorization of B to provide an efficient means of solving systems of equations involving $\tilde{B}$. The following proposition shows that E does indeed have a simple inverse.

PROPOSITION 8.1. *Given two column vectors u and v for which $1 + v^T u \neq 0$,*

$$(I + uv^T)^{-1} = I - \frac{uv^T}{1 + v^T u}.$$

PROOF. The proof is trivial. We simply multiply the matrix by its supposed inverse and check that we get the identity:

$$\begin{aligned}\left(I + uv^T\right)\left(I - \frac{uv^T}{1 + v^T u}\right) &= I + uv^T - \frac{uv^T}{1 + v^T u} - \frac{uv^T uv^T}{1 + v^T u} \\ &= I + uv^T\left(1 - \frac{1}{1 + v^T u} - \frac{v^T u}{1 + v^T u}\right) \\ &= I,\end{aligned}$$

where the last equality follows from the observation that the parenthesized expression vanishes. □

The identity in Proposition 8.1 may seem mysterious, but in fact it has a simple derivation based on the explicit formula for the sum of a geometric series:

$$\sum_{j=0}^{\infty} \xi^j = \frac{1}{1 - \xi}, \qquad \text{for } |\xi| < 1.$$

This is an identity for real numbers, but it also holds for matrices:

$$\sum_{j=0}^{\infty} X^j = (I - X)^{-1},$$

provided that the absolute value of each of the eigenvalues of X is less than one (we don't prove this here, since it's just for motivation). Assuming, for the moment, that the absolute values of the eigenvalues of uv^T are less than one (actually, all but one of them are zero), we can expand $(I + uv^T)^{-1}$ in a geometric series, reassociate products,

and collapse the resulting geometric series to get

$$\begin{aligned}\left(I+uv^T\right)^{-1} &= I - uv^T + (uv^T)(uv^T) - (uv^T)(uv^T)(uv^T) + \cdots \\ &= I - uv^T + u(v^Tu)v^T - u(v^Tu)(v^Tu)v^T + \cdots \\ &= I - u\left(1 - v^Tu + (v^Tu)^2 - \cdots\right)v^T \\ &= I - u\frac{1}{1+v^Tu}v^T \\ &= I - \frac{uv^T}{1+v^Tu},\end{aligned}$$

where the last equality follows from the fact that $1/(1+v^Tu)$ is a scalar and therefore can be pulled out of the vector/matrix calculation.

Applying Proposition 8.1 to matrix E, we see that

$$\begin{aligned}E^{-1} &= I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{1 + e_i^T(\Delta x_{\mathcal{B}} - e_i)} \\ &= I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{\Delta x_i} \\ &= \left[\begin{array}{ccc|c|ccc} 1 & & & -\frac{\Delta x_{j_1}}{\Delta x_i} & & & \\ & \ddots & & & & & \\ & & 1 & -\frac{\Delta x_{j_{p-1}}}{\Delta x_i} & & & \\ \hline & & & \frac{1}{\Delta x_i} & & & \\ \hline & & & -\frac{\Delta x_{j_{p+1}}}{\Delta x_i} & 1 & & \\ & & & & & \ddots & \\ & & & -\frac{\Delta x_{j_m}}{\Delta x_i} & & & 1 \end{array}\right].\end{aligned}$$

Now, let's look at the systems of equations that need to be solved in the next iteration. Using tildes to denote items associated with the next iteration, we see that we need to solve

$$\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j \qquad \text{and} \qquad \tilde{B}^T\tilde{v} = \tilde{e}_i$$

(actually, we should probably put the tilde on the j instead of the a_j and on the i instead of the e_i, but doing so seems less aesthetically appealing, even though it's more correct). Recalling that $\tilde{B} = BE$, we see that the first system is equivalent to

$$BE\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j,$$

which can be solved in two stages:

$$\begin{aligned}Bu &= \tilde{a}_j, \\ E\Delta\tilde{x}_{\mathcal{B}} &= u.\end{aligned}$$

Of course, the second system (involving E) is trivial, since we have an explicit formula for the inverse of E:

$$\begin{aligned}\Delta\tilde{x}_{\mathcal{B}} &= E^{-1}u \\ &= u - \frac{u_i}{\Delta x_i}(\Delta x_{\mathcal{B}} - e_i)\end{aligned}$$

(where, in keeping with our tradition, we have used u_i to denote the element of u associated with the basic variable x_i—that is, u_i is the pth entry of u).

The system involving $\tilde{B}^T$ is handled in the same manner. Indeed, first we rewrite it as

$$E^T B^T \tilde{v} = \tilde{e}_i$$

and then observe that it too can be solved in two steps:

$$\begin{aligned}E^T u &= \tilde{e}_i, \\ B^T \tilde{v} &= u.\end{aligned}$$

This time, the first step is the trivial one[1]:

$$\begin{aligned}u &= E^{-T}\tilde{e}_i \\ &= \tilde{e}_i - e_i \frac{(\Delta x_{\mathcal{B}} - e_i)^T \tilde{e}_i}{\Delta x_i}.\end{aligned}$$

Note that the fraction in the preceding equation is a scalar, and so this final expression for u shows that it is a vector with at most two nonzeros—that is, the result is utterly trivial even if the formula looks a little bit cumbersome.

We end this section by returning briefly to our example. Suppose that $\tilde{B}$ is B with column 3 replaced by the vector a_j given in (8.3). Suppose that

$$\tilde{a}_j = \begin{bmatrix} 5 & 0 & 0 & 0 & -1 \end{bmatrix}^T.$$

To solve $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$, we first solve $Bu = \tilde{a}_j$ using our LU-factorization of B. The result of the forward and backward substitutions is

$$u = \begin{bmatrix} 0 & 3 & 1 & -3 & -\frac{1}{2} \end{bmatrix}^T.$$

[1]Occasionally we use the superscript $-T$ for the transpose of the inverse of a matrix. Hence, $E^{-T} = (E^{-1})^T$.

Next, we solve for $\Delta\tilde{x}_{\mathcal{B}} = E^{-1}u$:

$$\Delta\tilde{x}_{\mathcal{B}} = u - \frac{u_3}{\Delta x_3}(\Delta x_{\mathcal{B}} - e_3) = \begin{bmatrix} 0 \\ 3 \\ 1 \\ -3 \\ -\frac{1}{2} \end{bmatrix} - \frac{1}{2}\left(\begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} \frac{1}{2} \\ 3 \\ \frac{1}{2} \\ -\frac{7}{2} \\ -\frac{1}{4} \end{bmatrix}.$$

Of course, it is easy to check that we have gotten the correct answer: simply multiply $\tilde{B}$ times $\Delta\tilde{x}_{\mathcal{B}}$ and check that it equals $\tilde{a}_j$. It does.

4. Performance Tradeoffs

The idea of writing the next basis as a product of the current basis times an easily invertible matrix can be extended over several iterations. For example, if we look k iterations out, we can write

$$B_k = B_0 E_0 E_1 \cdots E_{k-1}.$$

If we have an LU-factorization of B_0 and we have saved enough information to reconstruct each E_j, then we can use this product to solve systems of equations involving B_k.

Note that in order to reconstruct E_j, all we need to save is the primal step direction vector $\Delta x^j_{\mathcal{B}}$ (and an integer telling which column it goes in). In actual implementations, these vectors are stored in lists. For historical reasons, this list is called an *eta-file* (and the matrices E_j are called *eta matrices*). Given the LU-factorization of B_0 and the eta-file, it is an easy matter to solve systems of equations involving either B or B^T. However, as k gets large, the amount of work required to go through the entire eta-file begins to dominate the amount of work that would be required to simply form a new LU-factorization of the current basis. Hence, the best strategy is to use an eta-file but with periodic refactorization of the basis (and accompanied purging of the eta-file).

The question then becomes: how often should one recompute a factorization of the current basis? To answer this question, suppose that we know that it takes F arithmetic operations to form an LU-factorization (of a typical basis for the problem at hand), S operations to do one forward/backward substitution, and E operations to multiply by the inverse of one eta-matrix. Then the number of operations for the initial iteration of the simplex method is $F + 2S$ (since we need to do an LU-factorization and two forward/backward substitutions—one for the system involving the basis and the other for the system involving its transpose). Then, in the next iteration, we need to do two forward/backward substitutions and two eta-inverse calculations. Each subsequent iteration is the same as the previous, except that there are two extra eta-inverse

calculations. Hence, the average number of arithmetic operations per iteration if we refactorize after every K iterations is

$$\begin{aligned} T(K) &= \frac{1}{K}\left((F+2S)+2(S+E)+2(S+2E)\right. \\ &\qquad \left.+\cdots+2(S+(K-1)E)\right) \\ &= \frac{1}{K}F+2S+(K-1)E. \end{aligned}$$

Treating K as a real variable for the moment, we can differentiate this expression with respect to K, set the derivative equal to zero, and solve for K to get an estimate for the optimal choice of K:

$$K=\sqrt{\frac{F}{E}}.$$

As should be clear from our earlier discussions, E is of order m and, if the basis matrix is dense, F is of order m^3. Hence, for dense matrices, our estimates would indicate that refactorizations should take place every m iterations or so. However, for sparse matrices, F will be substantially less that m^3—more like a constant times m^2—which would indicate that refactorizations should occur on the order of every $\sqrt{m}$ iterations. In practice, one typically allows the value of K to be a user-settable parameter whose default value is set to something like 100.

5. Updating a Factorization

There is an important alternative to the eta-matrix method for reusing an LU-factorization, which we shall describe in this section and the next. As always, it is easiest to work with an example, so let's continue with the same example we've been using throughout this chapter.

Recall that the matrix $\tilde{B}$ is simply B with its third column replaced by the vector a_j given in (8.3):

$$\tilde{B}=\begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 2 & & 7 & & -2 \\ 3 & 1 & -2 & 1 & \\ -1 & & & & -2 \\ & -1 & 3 & & -6 \\ & & & & 4 \end{bmatrix} \end{array} = \begin{array}{c} \\ 4\\5\\3\\1\\2 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \begin{bmatrix} -1 & 3 & & -6 & \\ & & & 4 & \\ & & -1 & -2 & \\ & 7 & 2 & -2 & \\ 1 & -2 & 3 & & 1 \end{bmatrix} \end{array}.$$

(Note that we've highlighted the new column by putting a box around it.)

Since $L^{-1}B=U$ and $\tilde{B}$ differs from B in only one column, it follows that $L^{-1}\tilde{B}$ coincides with U except for the column that got changed. And, since this column got

replaced by a_j, it follows that this column of $L^{-1}\tilde{B}$ contains $L^{-1}a_j$, which we've already computed and found to be given by (8.4). Hence,

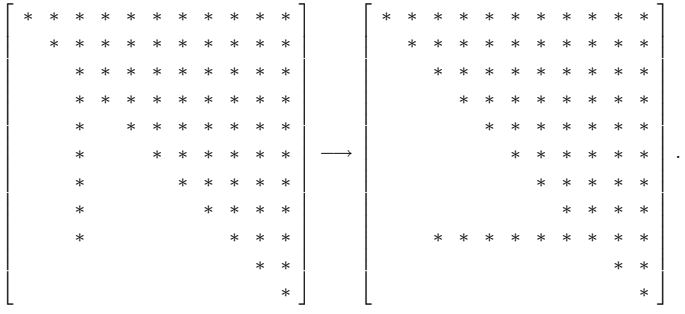
$$L^{-1}\tilde{B} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & \boxed{3} & & -6 & \\ & & & 4 & \\ & & -1 & 2 & \\ & 7 & & -14 & \\ & 1 & & & 1 \end{array}\right] \end{array} . \tag{8.6}$$

As we saw before, the columns of L have no "original" indices to relate back to, and so one can simply take them to be numbered in natural order. The same is then true for the rows of L^{-1} and hence for the rows of $L^{-1}\tilde{B}$. That is why the rows shown above are numbered as they are. We've shown these numbers explicitly, since they are about to get permuted.

The boxed column in (8.6) is called a *spike*, since it has nonzeros below the diagonal. The 4×4 submatrix constructed from rows 2 through 5 and columns 3, 1, 5, and 4 is called the *bump*. To get this matrix back into upper-triangular form, one could do row operations to eliminate the nonzeros in the spike that lie below the diagonal. But such row operations could create fill-in anywhere in the bump. Such fill-in would be more than one would like to encounter. However, consider what happens if the spike column is moved to the rightmost column of the bump, shifting the other columns left one position in the process, and if the top row of the bump (i.e., row 2) is moved to the bottom of the bump, shifting the other bump rows up by one. The result of these permutations is

$$L^{-1}\tilde{B} = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & -6 & & 3 \\ & \boxed{-1} & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & 4 & & \end{array}\right] \end{array} .$$

(For future reference, we've boxed the bump.) In general, the effect of this permutation is that the column spike gets replaced by a row spike along the bottom row of the bump:



Now any fill-in produced by row operations is confined to the spike row. In our example, there is only one nonzero in the spike row, and to eliminate it we need to add $2/7$ times row 4 to it. This row operation can be represented algebraically as multiplication on the left by the matrix

$$\tilde{E} = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & \frac{2}{7} & & 1 & \end{bmatrix}.$$

That is,

$$\tilde{E}L^{-1}\tilde{B} = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \begin{bmatrix} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & & & 2 \end{bmatrix} \end{array}.$$

If we denote the new upper triangular matrix by $\tilde{U}$, then, solving for $\tilde{B}$, we get the following factorization of $\tilde{B}$:

$$\tilde{B} = L\tilde{E}^{-1}\tilde{U}.$$

We can use this new factorization to solve systems of equations. For example, to solve

$$\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j,$$

we first solve

$$Ly = \tilde{a}_j \tag{8.7}$$

for y. Then, given y, we compute

$$z = \tilde{E}y,$$

and finally we solve

$$\tilde{U}\Delta\tilde{x}_{\mathcal{B}} = z$$

for $\Delta\tilde{x}_{\mathcal{B}}$. It is clear from the following chain of equalities that these three steps compute $\Delta\tilde{x}_{\mathcal{B}}$:

$$\Delta\tilde{x}_{\mathcal{B}} = \tilde{U}^{-1}z = \tilde{U}^{-1}\tilde{E}y = \tilde{U}^{-1}\tilde{E}L^{-1}\tilde{a}_j = \tilde{B}^{-1}\tilde{a}_j.$$

For our example, we use forward substitution to solve (8.7) for y. The result is

$$\begin{array}{rcc} 1 & \begin{bmatrix} 0 \\ -1 \\ -1 \\ 7 \\ -3 \end{bmatrix} & \begin{array}{c} 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \\ y = 2 & & = \\ 3 & & \\ 4 & & \\ 5 & & \end{array} \begin{bmatrix} 0 \\ -1 \\ 7 \\ -3 \\ -1 \end{bmatrix}.$$

Next, we apply the row operations (in our case, there is only one) to get z:

$$z = \tilde{E}y = \begin{array}{c} 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{bmatrix} 0 \\ -1 \\ 7 \\ -3 \\ 1 \end{bmatrix}.$$

Finally, backward substitution using $\tilde{U}$ is performed to compute $\Delta\tilde{x}_{\mathcal{B}}$:

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \begin{bmatrix} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & & & 2 \end{bmatrix} \end{array} \begin{array}{c} 2 \\ 1 \\ 5 \\ 4 \\ 3 \end{array} \begin{bmatrix} \\ \\ ? \\ \\ \end{bmatrix} = \begin{array}{c} 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{bmatrix} 0 \\ -1 \\ 7 \\ -3 \\ 1 \end{bmatrix}.$$

The result of the backward substitution is

$$\Delta\tilde{x}_{\mathcal{B}} = \begin{array}{c} 2 \\ 1 \\ 5 \\ 4 \\ 3 \end{array} \begin{bmatrix} 3 \\ \frac{1}{2} \\ -\frac{1}{4} \\ -\frac{7}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} \frac{1}{2} \\ 3 \\ \frac{1}{2} \\ -\frac{7}{2} \\ -\frac{1}{4} \end{bmatrix},$$

which agrees with the solution we got before using eta-matrices.

6. Shrinking the Bump

There is an important enhancement to the factorization updating technique described in the previous section. After permuting rows and columns converting the spike column into a spike row, we can exploit the fact that the spike row is often very sparse (coming as it does from what was originally the top row of the bump) and do further row and column permutations to reduce the size of the bump. To see what we mean, let's look at our example. First, we note that the leftmost element of the spike row is zero (and hence that the left column of the bump is a singleton column). Therefore, we can simply declare that this column and the corresponding top row do not belong to the bump. That is, we can immediately reduce the size of the bump by one:

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & 4 & & \end{array}\right] \end{array}.$$

This idea can be extended to any column that has a single nonzero in the bump. For example, column 4 is a singleton column too. The trick now is to move this column to the leftmost column of the bump, pushing the intermediate columns to the right, and to apply the same permutation to the rows. After permuting the rows and columns like this, the bump can be reduced in size again:

$$\begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 4 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 4 & 5 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & 3 \\ & -1 & & 2 & \\ & & 1 & & 1 \\ & & & -14 & 7 \\ & & & 4 & \end{array}\right] \end{array}.$$

Furthermore, this reduction in the bump causes a new singleton column to appear (since a singleton need only be a singleton within the bump), namely, column 3. Hence, we permute once again. This time just the column gets permuted, since the singleton is already in the correct row. The bump gets reduced in size once again, now

to a 1×1 bump, which is not really a bump at all:

$$\begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 4 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 4 & 3 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & 3 & -6 \\ & -1 & & & 2 \\ & & 1 & 1 & \\ & & & 7 & -14 \\ & & & & \boxed{4} \end{array}\right] \end{array}.$$

Note that we have restored upper triangularity using only permutations; no row operations were needed. While this doesn't always happen, it is a common and certainly welcome event.

Our example, being fairly small, doesn't exhibit all the possible bump-reducing permutations. In addition to looking for singleton columns, one can also look for singleton rows. Each singleton row can be moved to the bottom of the bump. At the same time, the associated column is moved to the right-hand column of the bump. After this permutation, the right-hand column and the bottom row can be removed from the bump.

Before closing this section, we reiterate a few important points. First, as the bump gets smaller, the chances of finding further singletons increases. Also, with the exception of the lower-right diagonal element of the bump, all other diagonal elements are guaranteed to be nonzero, since the matrix U from which $\tilde{U}$ is derived has this property. Therefore, most bump reductions apply the same permutation to the rows as to the columns. Finally, we have illustrated how to update the factorization once, but this technique can, of course, be applied over and over. Eventually, however, it becomes more efficient to refactorize the basis from scratch.

7. Partial Pricing

In many real-world problems, the number of constraints m is small compared with the number of variables n. Looking over the steps of the primal simplex method, we see that the only steps involving n-vectors are Step 2, in which we pick a nonbasic variable to be the entering variable,

$$\text{pick } j \in \{j \in \mathcal{N} : z_j^* < 0\};$$

Step 6, in which we compute the step direction for the dual variables,

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i;$$

and Step 8, in which we update the dual variables,

$$z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}}.$$

Scanning all the nonbasic indices in Step 2 requires looking at n candidates. When n is huge, this step is likely to be a bottleneck step for the algorithm. However, there is

no requirement that all indices be scanned. We could simply scan from the beginning and stop at the first index j for which z_j^* is negative (as in Bland's rule). However, in practice, it is felt that picking an index j corresponding to a very negative z_j^* produces an algorithm that is likely to reach optimality faster. Therefore, the following scheme, referred to as *partial pricing* is often employed. Initially, scan only a fraction of the indices (say $n/3$), and set aside a handful of good ones (say, the 40 or so having the most negative z_j^*). Then use only these 40 in Steps 2, 6, and 8 for subsequent iterations until less than a certain fraction (say, $1/2$) of them remain eligible. At this point, use (6.8) to compute the current values of a new batch of $n/3$ nonbasic dual variables, and go back to the beginning of this partial pricing process by setting aside the best 40. In this way, most of the iterations look like they only have 40 nonbasic variables. Only occasionly does the grim reality of the full huge number of nonbasic variables surface.

Looking at the dual simplex method (Figure 6.1), we see that we aren't so lucky. In it, vectors of length n arise in the max-ratio test:

$$t = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*} \right)^{-1}$$

$$\text{pick } j \in \text{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*}.$$

Here, the entire collection of nonbasic indices must be checked; otherwise, dual feasibility will be lost and the algorithm will fail. Therefore, in cases where n is huge relative to m and partial pricing is used, it is important not to use the dual simplex method as a Phase I procedure. Instead, one should use the technique of adding artificial variables as we did in Chapter 2 to force an initial feasible solution.

8. Steepest Edge

In Chapter 4, we saw that one of the drawbacks of the largest-coefficient rule is its sensitivity to the scale in which variables are quantified. In this section, we shall discuss a pivot rule that partially remedies this problem. Recall that each step of the simplex method is a step along an edge of the feasible region from one vertex to an adjacent vertex. The largest coefficient rule picks the variable that gives the largest rate of increase of the objective function. However, this rate of increase is measured in the "space of nonbasic variables" (we view the basic variables simply as dependent variables). Also, this space changes from one iteration to the next. Hence, in a certain respect, it would seem wiser to measure the rate of increase in the larger space consisting of all the variables, both basic and nonbasic. When the rate of increase is gauged in this larger space, the pivot rule is called the *steepest-edge* rule. It is the subject of this section.

Fix a nonbasic index $j \in \mathcal{N}$. We wish to consider whether x_j should be the entering variable. If it were, the step direction vector would be

$$\Delta x = \begin{bmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} -B^{-1}Ne_j \\ e_j \end{bmatrix}.$$

This vector points out along the edge corresponding to the pivot that would result by letting x_j enter the basis. As we know, the objective function is

$$f(x) = c^T x = c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}}.$$

The derivative of $f(x)$ in the direction of Δx is given by

$$\frac{\partial f}{\partial \Delta x} = c^T \frac{\Delta x}{\|\Delta x\|} = \frac{c_{\mathcal{B}}^T \Delta x_{\mathcal{B}} + c_{\mathcal{N}}^T \Delta x_{\mathcal{N}}}{\|\Delta x\|}.$$

The numerator is easy (and familiar):

$$\begin{aligned} c_{\mathcal{B}}^T \Delta x_{\mathcal{B}} + c_{\mathcal{N}}^T \Delta x_{\mathcal{N}} &= c_j - c_{\mathcal{B}}^T B^{-1} N e_j \\ &= \left(c_{\mathcal{N}} - (B^{-1}N)^T c_{\mathcal{B}}\right)_j \\ &= -z_j^*. \end{aligned}$$

The denominator is more troublesome:

$$\|\Delta x\|^2 = \|\Delta x_{\mathcal{B}}\|^2 + 1 = \|B^{-1}Ne_j\|^2 + 1.$$

To calculate $B^{-1}Ne_j$ for every $j \in \mathcal{N}$ is exactly the same as computing the matrix $B^{-1}N$, which (as we've discussed before) is time consuming and therefore a computation we wish to avoid. But it turns out that we can compute $B^{-1}Ne_j$ for every $j \in \mathcal{N}$ once at the start (when B is essentially, if not identically, an identity matrix) and then update the norms of these vectors using a simple formula, which we shall now derive.

Let

$$\nu_k = \|B^{-1}Ne_k\|^2, \qquad k \in \mathcal{N}.$$

Suppose that we know these numbers, we use them to perform one step of the simplex method, and we are now at the beginning of the next iteration. As usual, let us denote quantities in this next iteration by putting tildes on them. For example, $\tilde{B}$ denotes the new basis matrix. As we've seen before, $\tilde{B}$ is related to B by the equation $\tilde{B} = BE$, where

$$E^{-1} = I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{\Delta x_i}.$$

Now, let's compute the new ν values:

$$\begin{aligned} \tilde{\nu}_k &= a_k^T \tilde{B}^{-T} \tilde{B}^{-1} a_k \\ &= a_k^T B^{-T} E^{-T} E^{-1} B^{-1} a_k \\ &= a_k^T B^{-T} \left(I - \frac{e_i(\Delta x_{\mathcal{B}} - e_i)^T}{\Delta x_i}\right)\left(I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{\Delta x_i}\right) B^{-1} a_k. \end{aligned} \tag{8.8}$$

Recall from (8.2) that we must compute

$$v = B^{-T} e_i$$

in the course of the old iteration. If, in addition, we compute

$$w = B^{-T} \Delta x_{\mathcal{B}}$$

then, expanding out the product in (8.8) and expressing the resulting terms using v and w, we get the following formula for quickly updating the old ν's to the new ν's:

$$\tilde{\nu}_k = \nu_k - 2\frac{a_k^T v (w - v)^T a_k}{\Delta x_i} + (a_k^T v)^2 \frac{\|\Delta x_{\mathcal{B}} - e_i\|^2}{(\Delta x_i)^2}.$$

Recent computational studies using this update formula have shown that the steepest-edge rule for choosing the entering variable is competitive against, if not superior to, other pivot rules.

Exercises

8.1 (a) Without permuting rows or columns, compute the LU-factorization of

(8.9)
$$B = \begin{bmatrix} 2 & 5 & & 6 & \\ 1 & 1 & 3 & 9 & 6 \\ & & 2 & 6 & 4 \\ & & & 4 & 1 \\ & & -1 & -3 & -1 \end{bmatrix}.$$

(b) Solve the system $B\Delta x_{\mathcal{B}} = a_j$ where

$$a_j = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 3 \\ 0 \end{bmatrix}.$$

(c) Suppose that $\tilde{B}$ is B with its second column replaced by a_j. Solve the system $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$ where

$$\tilde{a}_j = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

using the eta-matrix method.

(d) Solve the system $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$ again, this time using the factorization updating method.

8.2 Use the minimum-degree ordering heuristic to find an LU-factorization of the matrix B given by (8.9).

8.3 A *permutation matrix* is a matrix of zeros and ones for which each row has one 1 and each column has one 1.

(a) Let B be an $m \times m$ matrix, and let P be a permutation matrix. Show that PB is a matrix obtained by permuting the rows of B and that BP is a matrix obtained by permuting the columns of B. Are the two permutations the same?

(b) Show that every permutation of the rows of a matrix B corresponds to multiplying B on the left by a permutation matrix.

(c) Show that for any permutation matrix P,

$$P^{-1} = P^T.$$

8.4 Explain how to use the factorization $B = LU$ to solve

$$B^T x = b.$$

Notes

Techniques for exploiting sparsity in matrix factorization have their roots in the paper by Markowitz (1957). A few standard references on matrix factorization are the books of Duff et al. (1986), Golub & VanLoan (1989), and Gill et al. (1991). The eta-matrix technique given in Section 8.3 for using an old basis to solve systems of equations involving the current basis was first described by Dantzig & Orchard-Hayes (1954). The factorization updating technique described in Section 8.5 is the method given by Forrest & Tomlin (1972). The bump reduction techniques of Section 8.6 were first introduced by Saunders (1973) and Reid (1982). The steepest-edge pivoting rule is due to Goldfarb & Reid (1977). A similar rule, known as *Devex*, was given by Harris (1973).

CHAPTER 9

Problems in General Form

Up until now, we have always considered our problems to be given in standard form. However, for real-world problems it is often convenient to formulate problems in the following form:

$$\begin{array}{lll} \text{maximize} & & c^T x \\ \text{subject to} & a \leq & Ax \leq b \\ & l \leq & x \leq u. \end{array} \tag{9.1}$$

Two-sided constraints such as those given here are called constraints with *ranges*. The vector l is called the vector of *lower bounds*, and u is the vector of *upper bounds*. We allow some of the data to take infinite values; that is, for each $i = 1, 2, \ldots, m$,

$$-\infty \leq a_i \leq b_i \leq \infty,$$

and, for each $j = 1, 2, \ldots, n$,

$$-\infty \leq l_j \leq u_j \leq \infty.$$

In this chapter, we shall show how to modify the simplex method to handle problems presented in this form.

1. The Primal Simplex Method

It is easiest to illustrate the ideas with an example:

$$\begin{array}{lrcl} \text{maximize} & & 3x_1 - x_2 & \\ \text{subject to} & 1 \leq & -x_1 + x_2 \leq & 5 \\ & 2 \leq & -3x_1 + 2x_2 \leq & 10 \\ & & 2x_1 - x_2 \leq & 0 \\ & -2 \leq & x_1 & \\ & 0 \leq & x_2 \leq & 6. \end{array}$$

With this formulation, zero no longer plays the special role it once did. Instead, that role is replaced by the notion of a variable or a constraint being at its upper or lower

bound. Therefore, instead of defining slack variables for each constraint, we use w_i simply to denote the value of the ith constraint:

$$\begin{aligned} w_1 &= -x_1 + x_2 \\ w_2 &= -3x_1 + 2x_2 \\ w_3 &= 2x_1 - x_2. \end{aligned}$$

The constraints can then be interpreted as upper and lower bounds on these variables. Now when we record our problem in a dictionary, we will have to keep explicit track of the upper and lower bound on the original x_j variables and the new w_i variables. Also, the value of a nonbasic variable is no longer implicit; it could be at either its upper or its lower bound. Hence, we shall indicate which is the case by putting a box around the relevant bound. Finally, we need to keep track of the values of the basic variables. Hence, we shall write our dictionary as follows:

l		$\boxed{-2}$	$\boxed{0}$	
u		∞	6	
	$\zeta =$	$3x_1 -$	$x_2 =$	-6
$1 \quad 5$	$w_1 =$	$-x_1 +$	$x_2 =$	2
$2 \quad 10$	$w_2 =$	$-3x_1 + 2x_2 =$		6
$-\infty \quad 0$	$w_3 =$	$2x_1 -$	$x_2 =$	-4.

Since all the w_i's are between their upper and lower bounds, this dictionary is feasible. But it is not optimal, since x_1 could be increased from its present value at the lower bound, thereby increasing the objective function's value. Hence, x_1 shall be the entering variable for the first iteration. Looking at w_1, we see that x_1 can be raised only 1 unit before w_1 hits its lower bound. Similarly, x_1 can be raised by $4/3$ units, at which point w_2 hits its lower bound. Finally, if x_1 were raised 2 units, then w_3 would hit its upper bound. The tightest of these constraints is the one on w_1, and so w_1 becomes the leaving variable—which, in the next iteration, will then be at its lower bound. Performing the usual row operations, we get

l		$\boxed{1}$	$\boxed{0}$	
u		5	6	
	$\zeta =$	$-3w_1 + 2x_2 =$		-3
$-2 \quad \infty$	$x_1 =$	$-w_1 +$	$x_2 =$	-1
$2 \quad 10$	$w_2 =$	$3w_1 -$	$x_2 =$	3
$-\infty \quad 0$	$w_3 =$	$-2w_1 +$	$x_2 =$	-2.

Note, of course, that the objective function value has increased (from -6 to -3). For the second iteration, raising x_2 from its lower bound will produce an increase in ζ.

Hence, x_2 is the entering variable. Looking at the basic variables (x_1, w_2, and w_3), we see that w_2 will be the first variable to hit a bound, namely, its lower bound. Hence, w_2 is the leaving variable, which will become nonbasic at its lower bound:

$$\begin{array}{rr|l}
l & & \boxed{1} & \boxed{2} \\
 & u & 5 & 10 \\
\hline
 & & \zeta = 3w_1 - 2w_2 = -1 \\
\hline
-2 & \infty & x_1 = 2w_1 - w_2 = 0 \\
0 & 6 & x_2 = 3w_1 - w_2 = 1 \\
-\infty & 0 & w_3 = w_1 - w_2 = -1\,.
\end{array}$$

For the third iteration, w_1 is the entering variable, and w_3 is the leaving variable, since it hits its upper bound before any other basic variables hit a bound. The result is

$$\begin{array}{rr|l}
l & & -\infty & \boxed{2} \\
 & u & \boxed{0} & 10 \\
\hline
 & & \zeta = 3w_3 + w_2 = 2 \\
\hline
-2 & \infty & x_1 = 2w_3 + w_2 = 2 \\
0 & 6 & x_2 = 3w_3 + 2w_2 = 4 \\
1 & 5 & w_1 = w_3 + w_2 = 2\,.
\end{array}$$

Now for the next iteration, note that the coefficients on both w_3 and w_2 are positive. But w_3 is at its upper bound, and so if it were to change, it would have to decrease. However, this would mean a decrease in the objective function. Hence, only w_2 can enter the basis, in which case x_2 is the leaving variable getting set to its upper bound:

$$\begin{array}{rr|l}
l & & -\infty & 0 \\
 & u & \boxed{0} & \boxed{6} \\
\hline
 & & \zeta = 1.5w_3 + 0.5x_2 = 3 \\
\hline
-2 & \infty & x_1 = 0.5w_3 + 0.5x_2 = 3 \\
2 & 10 & w_2 = -1.5w_3 + 0.5x_2 = 3 \\
1 & 5 & w_1 = -0.5w_3 + 0.5x_2 = 3\,.
\end{array}$$

For this dictionary, both w_3 and x_2 are at their upper bounds and have positive coefficients in the formula for ζ. Hence, neither can be moved off from its bound to increase the objective function. Therefore, the current solution is optimal.

2. The Dual Simplex Method

The problem considered in the previous section had an initial dictionary that was feasible. But as always, we must address the case where the initial dictionary is not

feasible. That is, we must define a Phase I algorithm. Following the ideas presented in Chapter 5, we base our Phase I algorithm on a dual simplex method. To this end, we need to introduce the dual of (9.1). So first we rewrite (9.1) as

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & -Ax \leq -a \\ & x \leq u \\ & -x \leq -l, \end{aligned}$$

and adding slack variables, we have

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax + f = b \\ & -Ax + p = -a \\ & x + t = u \\ & -x + g = -l \\ & f,\ p,\ t,\ g \geq 0. \end{aligned}$$

We see immediately from the inequality form of the primal that the dual can be written as

$$\begin{aligned} \text{minimize} \quad & b^T v - a^T q + u^T s - l^T h \\ \text{subject to} \quad & A^T(v-q) - (h-s) = c \\ & v,\ q,\ s,\ h \geq 0. \end{aligned} \tag{9.2}$$

Furthermore, at optimality, the dual variables are complementary to the corresponding primal slack variables:

$$\begin{aligned} f_i v_i &= 0 \qquad i = 1, 2, \ldots, m, \\ p_i q_i &= 0 \qquad i = 1, 2, \ldots, m, \\ t_j s_j &= 0 \qquad j = 1, 2, \ldots, n, \\ g_j h_j &= 0 \qquad j = 1, 2, \ldots, n. \end{aligned} \tag{9.3}$$

Note that for each i, if $b_i > a_i$, then at optimality v_i and q_i must be complementary to each other. Indeed, if both were positive, then they could be reduced by an equal amount without destroying feasibility, and the objective function value would strictly decrease, thereby implying that the supposedly optimal solution is not optimal. Similarly, if for some i, $b_i = a_i$, then it is no longer required that v_i and q_i be complementary at optimality; but, given an optimal solution for which both v_i and q_i are positive, we can decrease both these values at the same rate until the smaller of the two reaches zero, all the while preserving feasibility of the solution and not changing

the objective function value. Hence, there always exists an optimal solution in which every component of v is complementary to the corresponding component of q. The same argument shows that if there exists an optimal solution, then there exists one in which all the components of h and s are complementary to each other as well.

For a real variable ξ, its positive part ξ^+ is defined as

$$\xi^+ = \max\{\xi, 0\}$$

and its negative part ξ^- is defined similarly as

$$\xi^- = \max\{-\xi, 0\}.$$

Clearly, both ξ^+ and ξ^- are nonnegative. Furthermore, they are complementary,

$$\xi^+ = 0 \qquad \text{or} \qquad \xi^- = 0,$$

and their difference represents ξ:

$$\xi = \xi^+ - \xi^-.$$

From the complementarity of the components of v against the components of q, we can think of them as the positive and negative parts of the components of just one vector y. So let us write:

$$v = y^+ \qquad \text{and} \qquad q = y^-.$$

Similarly, let us write

$$h = z^+ \qquad \text{and} \qquad s = z^-.$$

If we impose these complementarity conditions not just at optimality but also from the start, then we can eliminate v, q, s, and h from the dual and write it simply as

$$\begin{aligned} &\text{minimize} \quad b^T y^+ - a^T y^- + u^T z^+ - l^T z^- \\ &\text{subject to} \quad A^T y - z = c, \end{aligned} \tag{9.4}$$

where the notation y^+ denotes the componentwise positive part of y, etc. This problem is an example from the class of problems called piecewise linear programs. Usually, piecewise linear programs are solved by converting them into linear programs. Here, however, we wish to go in the other direction. We shall present an algorithm for (9.4) that will serve as an algorithm for (9.2). We will call this algorithm the *dual simplex method* for problems in general form.

To economize on the presentation, we shall present the dual simplex method in the context of a Phase I algorithm for linear programs in general form. Also, to avoid

cumbersome notations, we shall present the algorithm with the following example:

$$
\begin{array}{lrcl}
\text{maximize} & & 2x_1 - x_2 & \\
\text{subject to} & 0 \le & x_1 + x_2 & \le 6 \\
& 2 \le & -x_1 + 2x_2 & \le 10 \\
& & x_1 - x_2 & \le 0 \\
& -2 \le & x_1 & \\
& 1 \le & x_2 & \le 5.
\end{array}
\tag{9.5}
$$

The piecewise linear formulation of the dual is

$$
\begin{array}{lrcl}
\text{minimize} & 6y_1^+ + 10y_2^+ & + 2z_1^+ - z_2^+ & \\
& - 2y_2^- + \infty y_3^- + \infty z_1^- + 5z_2^- & & \\
\text{subject to} & y_1 - y_2 + y_3 - z_1 & = & 2 \\
& y_1 + 2y_2 - y_3 - & z_2 & = -1.
\end{array}
$$

Note that the objective function has coefficients that are infinite. The correct convention is that infinity times a variable is plus infinity if the variable is positive, zero if the variable is zero, and minus infinity if the variable is negative.

Since the objective function is nonlinear (taking positive and negative parts of variables is certainly a nonlinear operation), we will not be able to do the usual row operations on the objective function. Therefore, in each iteration, we simply study it as is. But as usual, we prefer to think in terms of maximization, and so we record the negative of the objective function:

$$
\begin{aligned}
-\xi = -6y_1^+ - 10y_2^+ & - 2z_1^+ + z_2^+ \\
+ 2y_2^- - \infty y_3^- - \infty z_1^- - 5z_2^-&.
\end{aligned}
\tag{9.6}
$$

We can of course perform row operations on the two constraints, so we set up the usual sort of dictionary for them:

$$
\begin{aligned}
z_1 &= -2 + y_1 - y_2 + y_3 \\
z_2 &= 1 + y_1 + 2y_2 - y_3.
\end{aligned}
\tag{9.7}
$$

For the dual problem, all the action takes place at zero. That is, slopes in the objective function change when a variable goes from negative to positive. Since nonbasic variable are supposed to be set where the action is, we associate a current solution with each dictionary by setting the nonbasic variables to zero. Hence, the solution associated with the initial dictionary is

$$(y_1, y_2, y_3, z_1, z_2) = (0, 0, 0, -2, 1).$$

The fact that z_1 is negative implies that z_1^- is a positive number and hence that the objective function value associated with this solution is minus infinity. Whenever the

objective function value is minus infinity, we say that the solution is *infeasible*. We also refer to the associated dictionary as infeasible. Hence, the initial dictionary given in (9.7) is infeasible.

The dual simplex method must start with a dual feasible solution. But since we intend to use the dual simplex method simply to find a feasible solution for (9.5), we are free to change the objective function in (9.5) any way we please. In particular, we can change it from

$$\zeta = 2x_1 - x_2$$

to

$$\eta = -2x_1 - x_2.$$

Making that change to the primal leaves the dual objective function unchanged, but produces a feasible dual dictionary:

$$\begin{aligned} z_1 &= 2 + y_1 - \ \ y_2 + y_3 \\ z_2 &= 1 + y_1 + 2y_2 - y_3. \end{aligned} \tag{9.8}$$

For comparison purposes, let us also record the corresponding primal dictionary. It is easy to write down the equations defining the w_i's, but how do we know whether the x_j's are supposed to be at their upper or their lower bounds? The answer comes from the requirement that the primal and dual satisfy the complementarity conditions given in (9.3). Indeed, from the dual dictionary we see that $z_1 = 1$. Hence, $z_1^+ = 1$. But since z_1^+ is just a surrogate for h_1, we see that h_1 is positive and hence that g_1 must be zero. This means that x_1 must be at its lower bound. Similarly, for the sake of complementarity, x_2 must also be at its lower bound. Hence, the primal dictionary is

$$\begin{array}{rr|rrrrr} l & & & \boxed{-2} & & \boxed{1} & \\ & u & & \infty & & 5 & \\ \hline & & \eta = & -x_1 & - & x_2 & = 1 \\ \hline 0 & 6 & w_1 = & x_1 & + & x_2 & = -1 \\ 2 & 10 & w_2 = & -x_1 & + & 2x_2 & = 4 \\ -\infty & 0 & w_3 = & x_1 & - & x_2 & = -3\,. \end{array}$$

Note that it is infeasible, since w_1 is not between its upper and lower bounds.

We are now ready to describe the first iteration of the dual simplex method. To this end, we ask whether we can improve the dual objective function value by moving one of the nonbasic variables (y_1, y_2, or y_3) away from zero. Of course, each of these three variables can be moved either to the positive or the negative side of zero; we must analyze these six cases individually. First of all, note that since z_1 is positive at the current solution, it follows that $z_1^+ = z_1$ and $z_1^- = 0$ in a neighborhood of the current solution. A similar statement can be made for z_2, and so we can rewrite (9.6)

locally around the current solution as

$$\begin{aligned} -\xi = -6y_1^+ - 10y_2^+ \qquad\qquad & - 2z_1 + z_2 \\ + \;\; 2y_2^- - \infty y_3^- & . \end{aligned}$$

Now, as y_1 is increased from zero, the rate of increase of $-\xi$ is simply the derivative of the right-hand side with respect to y_1, where we must keep in mind that z_1 and z_2 are functions of y_1 via the dictionary (9.8). Hence, the rate of increase is $-6-2+1 = -7$; i.e., the objective function decreases at a rate of 7 units per unit increase of y_1. If, on the other hand, y_2 is decreased from zero into negative territory, then the rate of increase of $-\xi$ is the negative of the derivative of the right-hand side. In this case we get no contribution from y_1^- but we do get something from z_1 and z_2 for a total of $2 - 1 = 1$. Hence, the rate of increase as we move in this direction is one unit increase per unit move. We can analyze changes to y_2 and y_3. The entire situation can be summarized as follows:

$$\begin{aligned} y_1 \nearrow \;\; -6-2+1 &= \;\; -7 \\ y_1 \searrow \;\;\;\; 0+2-1 &= \;\;\;\; 1 \\ y_2 \nearrow -10+2+2 &= \;\; -6 \\ y_2 \searrow \;\;\;\; 2-2-2 &= \;\; -2 \\ y_3 \nearrow \;\;\;\; 0-2-1 &= \;\; -3 \\ y_3 \searrow -\infty+2+1 &= -\infty. \end{aligned}$$

Of these six cases, the only one that brings about an increase in $-\xi$ is the one in which y_1 is sent negative. Hence, y_1 shall be our entering variable, and it will go negative. To find the leaving variable, we must ask: as y_1 goes negative, which of z_1 and z_2 will hit zero first? For the current dictionary, z_2 gets to zero first and so becomes the leaving variable. Performing the usual row operations, the new dictionary for the dual problem is

$$\begin{aligned} z_1 = \;\; 1 + z_2 - 3y_2 + 2y_3 \\ y_1 = -1 + z_2 - 2y_2 - \;\; y_3. \end{aligned}$$

Let us have a look at the new primal dictionary. The fact that y_1 was the entering variable in the dual dictionary implies that w_1 is the leaving variable in the primal. Furthermore, the fact that y_1 has gone negative implies that y_1^- is now positive, and so complementarity then demands that q_1 be zero; i.e., w_1 should go to its lower bound. The fact that z_2 was the leaving variable in the dual dictionary implies that x_2 is the

entering variable in the primal. Hence, the new primal dictionary is

$$\begin{array}{rr|rrrr}
l & & \boxed{-2} & & \boxed{0} & \\
u & & \infty & & 6 & \\
\hline
 & & \eta = & -x_1 - & w_1 = & 2 \\
\hline
1 & 5 & x_2 = & -x_1 + & w_1 = & 2 \\
2 & 10 & w_2 = & -3x_1 + & 2w_1 = & 6 \\
-\infty & 0 & w_3 = & 2x_1 - & w_1 = & -4\,.
\end{array}$$

We are now ready to begin the second iteration. Therefore, we ask which nonbasic variable should be moved away from zero (and in which direction). As before, we first note that z_1 positive implies that $z_1^+ = z_1$ and $z_1^- = 0$ and that y_1 negative implies that $y_1^+ = 0$ and $y_1^- = -y_1$. Hence, the objective function can be written locally around the current solution as

$$\begin{aligned}
-\xi = &-10y_2^+ && -2z_1 + z_2^+ \\
&+ 2y_2^- - \infty y_3^- && -5z_2^-\,.
\end{aligned}$$

We now summarize the possibilities in a small table:

$$\begin{array}{rr}
z_2 \nearrow & 1-2 = -1 \\
z_2 \searrow & -5+2 = -3 \\
y_2 \nearrow & -10+6 = -4 \\
y_2 \searrow & 2-6 = -4 \\
y_3 \nearrow & 0-4 = -4 \\
y_3 \searrow & -\infty+4 = -\infty.
\end{array}$$

Note that all the changes are negative, meaning that there are no possibilities to increase the objective function any further. That is, the current dual solution is optimal. Of course, this also could have been deduced by observing that the primal dictionary is feasible (which is what we are looking for, after all).

Even though this example of the dual simplex method has terminated after only one iteration, it should be clear how to proceed had it not terminated.

Now that we have a feasible solution for the primal, we could solve the problem to optimality by simply reinstating the original objective function and proceeding by applying the primal simplex method in a Phase II procedure to find the optimal solution. Since the primal simplex method has already been discussed, we stop here on this problem.

Exercises

Solve the following linear programming problems:

9.1

$$\begin{aligned}
\text{maximize}\quad & -x_1 + x_2 \\
\text{subject to}\quad & -x_1 + x_2 \le 5 \\
& x_1 - 2x_2 \le 9 \\
& 0 \le x_1 \le 6 \\
& 0 \le x_2 \le 8.
\end{aligned}$$

9.2

$$\begin{aligned}
\text{maximize}\quad & -3x_1 - x_2 + x_3 + 2x_4 - x_5 + x_6 - x_7 - 4x_8 \\
\text{subject to}\quad & x_1 + 4x_3 + x_4 - 5x_5 - 2x_6 + 3x_7 - 6x_8 = 7 \\
& x_2 - 3x_3 - x_4 + 4x_5 + x_6 - 2x_7 + 5x_8 = -3 \\
& 0 \le x_1 \le 8 \\
& 0 \le x_2 \le 6 \\
& 0 \le x_3 \le 10 \\
& 0 \le x_4 \le 15 \\
& 0 \le x_5 \le 2 \\
& 0 \le x_6 \le 10 \\
& 0 \le x_7 \le 4 \\
& 0 \le x_8 \le 3.
\end{aligned}$$

Notes

Dantzig (1955) was the first to consider variants of the simplex method that handle bounds and ranges implicitly.

CHAPTER 10

Convex Analysis

This book is mostly about linear programming. However, this subject, important as it is, is just a subset of a larger subject called convex analysis. In this chapter, we shall give a brief introduction to this broader subject. In particular, we shall prove a few of the fundamental results of convex analysis and see that their proofs depend on some of the theory of linear programming that we have already developed.

1. Convex Sets

Given a finite set of points, $z_1, z_2, \ldots, z_n$, in $\mathbb{R}^m$, a point z in $\mathbb{R}^m$ is called a *convex combination* of these points if[1]

$$z = \sum_{j=1}^{n} t_j z_j,$$

where $t_j \geq 0$ for each j and $\sum_{j=1}^{n} t_j = 1$. It is called a *strict convex combination* if none of the t_j's vanish. For $n = 2$, the set of all convex combinations of two points is simply the line segment connecting them.

A subset S of $\mathbb{R}^m$ is called *convex* if, for every x and y in S, S also contains all points on the line segment connecting x and y. That is, $tx + (1-t)y \in S$, for every $0 < t < 1$. See Figure 10.1.

Certain elementary properties of convex sets are trivial to prove. For example, the intersection of an arbitrary collection of convex sets is convex. Indeed, let S_α, $\alpha \in I$, denote a collection of convex sets indexed by some set I. Then the claim is that $\cap_{\alpha \in I} S_\alpha$ is convex. To see this, consider an arbitrary pair of points x and y in the intersection. It follows that x and y are in each S_α. By the convexity of S_α it follows that S_α contains the line segment connecting x and y. Since each of these sets contains the line segment, so does their intersection. Hence, the intersection is convex.

Here is another easy one:

THEOREM 10.1. *A set C is convex if and only if it contains all convex combinations of points in C.*

[1]Until now we've used subscripts for the components of a vector. In this Chapter, subscripts will be used to list sequences of vectors. Hopefully, this will cause no confusion.

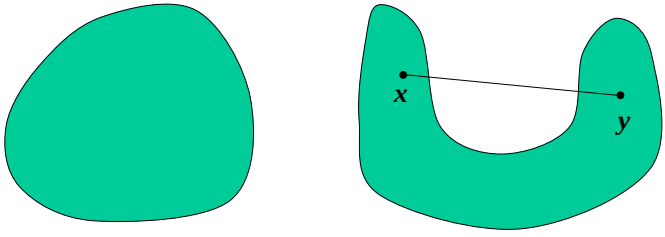


FIGURE 10.1. The set on the left is *convex*—for any pair of points in the set, the line segment connecting the two points is also contained in the set. The set on the right is *not convex*—there exists pairs of points, such as the x and y shown, for which the connecting line segment is not entirely in the set.

PROOF. Let C be a convex set. By definition, C contains all convex combinations of pairs of points in C. The first nontrivial step is to show that C contains all convex combinations of triples of points in C. To see this, fix z_1, z_2, and z_3 in C and consider

$$z = t_1 z_1 + t_2 z_2 + t_3 z_3,$$

where $t_j \geq 0$ for each j and $\sum_{j=1}^{3} t_j = 1$. If any of the t_j's vanish, then z is really just a convex combination of two points and so belongs to C. Hence, suppose that each of the t_j's is strictly positive. Rewrite z as follows:

$$\begin{aligned} z &= (1 - t_3)\left(\frac{t_1}{1 - t_3} z_1 + \frac{t_2}{1 - t_3} z_2\right) + t_3 z_3 \\ &= (1 - t_3)\left(\frac{t_1}{t_1 + t_2} z_1 + \frac{t_2}{t_1 + t_2} z_2\right) + t_3 z_3. \end{aligned}$$

Since C contains all convex combinations of pairs of points, it follows that

$$\frac{t_1}{t_1 + t_2} z_1 + \frac{t_2}{t_1 + t_2} z_2 \in C.$$

Now, since z is a convex combination of the two points $\frac{t_1}{t_1+t_2} z_1 + \frac{t_2}{t_1+t_2} z_2$ and z_3, both of which belong to C, it follows that z is in C. It is easy to see (pun intended) that this argument can be extended to an inductive proof that C contains all convex combinations of finite collections of points in C. Indeed, one must simply show that the fact that C contains all convex combinations of n points from C implies that it contains all convex combinations of $n + 1$ points from C. We leave the details to the reader.

Of course, proving that a set is convex if it contains every convex combination of its points is trivial: simply take convex combinations of pairs to get that it is convex. □

For each set S in $\mathbb{R}^m$ (not necessarily convex), there exists a smallest convex set, which we shall denote by $\text{conv}(S)$, containing S. It is defined, quite simply, as the intersection of all convex sets containing S. From our discussion about intersections, it follows that this set is convex. The set $\text{conv}(S)$ is called the *convex hull* of S. This definition can be thought of as a definition from the "outside," since it involves forming the intersection of a collection of sets that contain S. Our next theorem gives a characterization of convex sets from the "inside":

THEOREM 10.2. *The convex hull* $\text{conv}(S)$ *of a set* S *in* $\mathbb{R}^m$ *consists precisely of the set of all convex combinations of finite collections of points from* S.

PROOF. Let H denote the set of all convex combinations of finite sets of points in S:

$$H = \left\{ z = \sum_{j=1}^{n} t_j z_j : n \geq 1, z_j \in S \text{ and } t_j > 0 \text{ for all } j, \text{ and } \sum_{j=1}^{n} t_j = 1 \right\}.$$

It suffices to show that (1) H contains S, (2) H is convex, and (3) every convex set containing S also contains H.

To see that H contains S, just take $n = 1$ in the definition of H.

To see that H is convex, fix two points x and y in H and a real number $0 < t < 1$. We must show that $z = tx + (1-t)y \in H$. The fact that $x \in H$ implies that $x = \sum_{j=1}^{r} p_j x_j$, for some $r \geq 1$, where $p_j > 0$ for $j = 1, 2, \ldots, r$, $\sum_{j=1}^{r} p_j = 1$, and $x_j \in S$ for $j = 1, 2, \ldots, r$. Similarly, the fact that y is in H implies that $y = \sum_{j=1}^{s} q_j y_j$, for some $s \geq 1$, where $q_j > 0$ for $j = 1, 2, \ldots, s$, $\sum_{j=1}^{s} q_j = 1$, and $y_j \in S$ for $j = 1, 2, \ldots, s$. Hence,

$$z = tx + (1-t)y = \sum_{j=1}^{r} t p_j x_j + \sum_{j=1}^{s} (1-t) q_j y_j.$$

Since the coefficients $(tp_1, \ldots, tp_r, (1-t)q_1, \ldots, (1-t)q_s)$ are all positive and sum to one, it follows that this last expression for z is a convex combination of $r+s$ points from S. Hence, z is in H. Since x and y were arbitrary points in H and t was an arbitrary real number between zero and one, the fact that $z \in H$ implies that H is convex.

It remains simply to show that H is contained in every convex set containing S. Let C be such a set (i.e., convex and containing S). From Theorem 10.1 and the fact that C contains S, it follows that C contains all convex combinations of points in S. Hence, C contains H. □

2. Carathéodory's Theorem

In the previous section, we showed that the convex hull of a set S can be constructed by forming all convex combinations of finite sets of points from S. In 1907,

Carathéodory showed that it is not necessary to use all finite sets. Instead, $m+1$ points suffice:

THEOREM 10.3. *The convex hull* $\text{conv}(S)$ *of a set* S *in* $\mathbb{R}^m$ *consists of all convex combinations of* $m+1$ *points from* S*:*

$$\text{conv}(S) = \left\{ z = \sum_{j=1}^{m+1} t_j z_j : z_j \in S \text{ and } t_j \geq 0 \text{ for all } j, \text{ and } \sum_j t_j = 1 \right\}.$$

PROOF. Let H denote the set on the right. From Theorem 10.2, we see that H is contained in $\text{conv}(S)$. Therefore, it suffices to show that every point in $\text{conv}(S)$ belongs to H. To this end, fix a point z in $\text{conv}(S)$. By Theorem 10.2, there exists a collection of, say, n points $z_1, z_2, \ldots, z_n$ in S and associated nonnegative multipliers $t_1, t_2, \ldots, t_n$ summing to one such that

$$z = \sum_{j=1}^{n} t_j z_j. \tag{10.1}$$

Let A denote the matrix consisting of the points $z_1, z_2, \ldots, z_n$ as the columns of A:

$$A = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}.$$

Also, let x^* denote the vector consisting of the multipliers $t_1, t_2, \ldots, t_n$:

$$x^* = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}.$$

Finally, let $b = z$. Then from (10.1), we see that x^* is feasible for the following linear programming problem:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & e^T x = 1 \\ & x \geq 0. \end{aligned} \tag{10.2}$$

The fundamental theorem of linear programming (Theorem 3.4) tells us that every feasible linear program has a basic feasible solution. For such a solution, only the basic variables can be nonzero. The number of basic variables in (10.2) coincides with the number of equality constraints; that is, there are at most $m+1$ variables that are nonzero. Hence, this basic feasible solution corresponds to a convex combination of just $m+1$ of the original n points. (See Exercise 10.5.) □

It is easy to see that the number $m+1$ is the best possible. For example, the point $(1/(m+1), 1/(m+1), \ldots, 1/(m+1))$ in $\mathbb{R}^m$ belongs to the convex hull of the $m+1$ points $e_1, e_2, \ldots, e_m, 0$ but is not a convex combination of any subset of them.

3. The Separation Theorem

We shall define a *halfspace* of $\mathbb{R}^n$ to be any set given by a single (nontrivial) linear inequality:

$$\{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \leq b\}, \qquad (a_1, a_2, \ldots, a_n) \neq 0. \tag{10.3}$$

Every halfspace is convex. To see this, suppose that $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ both satisfy the linear inequality in (10.3). Fix t between zero and one. Then both t and $1-t$ are nonnegative, and so multiplying by them preserves the direction of inequality. Therefore, multiplying $\sum_j a_j x_j \leq b$ by t and $\sum_j a_j y_j \leq b$ by $1-t$ and then adding, we get

$$\sum_j a_j \left(t x_j + (1-t) y_j\right) \leq b.$$

That is, $tx + (1-t)y$ also satisfies the inequality defining the halfspace.

If we allow the vector of coefficients $(a_1, a_2, \ldots, a_n)$ in the definition of a halfspace to vanish, then we call the set so defined a *generalized halfspace*. It is easy to see that every generalized halfspace is simply a halfspace, all of $\mathbb{R}^n$, or the empty set. Also, every generalized halfspace is clearly convex.

A *polyhedron* is defined as the intersection of a finite collection of generalized halfspaces. That is, a polyhedron is any set of the form

$$\{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, 2, \ldots, m\}.$$

Every polyhedron, being the intersection of a collection of convex sets, is convex.

The following theorem is called the *Separation Theorem* for polyhedra.

THEOREM 10.4. *Let P and $\tilde{P}$ be two disjoint nonempty polyhedra in $\mathbb{R}^n$. Then there exist disjoint halfspaces H and $\tilde{H}$ such that $P \subset H$ and $\tilde{P} \subset \tilde{H}$.*

PROOF. Suppose that P and $\tilde{P}$ are given by the following systems of inequalities:

$$P = \{x : Ax \leq b\},$$
$$\tilde{P} = \{x : \tilde{A}x \leq \tilde{b}\}.$$

The disjointness of P and $\tilde{P}$ implies that there is no solution to the system

$$\begin{bmatrix} A \\ \tilde{A} \end{bmatrix} x \leq \begin{bmatrix} b \\ \tilde{b} \end{bmatrix}. \tag{10.4}$$

To continue the proof, we need a result known as Farkas' Lemma, which says that $Ax \leq b$ has no solutions if and only if there is an m-vector y such that

$$\begin{aligned} A^T y &= 0 \\ y &\geq 0 \\ b^T y &< 0. \end{aligned}$$

We shall prove this result in the next section. For now, let us apply Farkas' Lemma to the situation at hand. Indeed, the fact that there are no solutions to (10.4) implies that there exists a vector, which we shall write in block form as

$$\begin{bmatrix} y \\ \tilde{y} \end{bmatrix},$$

such that

$$\begin{bmatrix} A^T & \tilde{A}^T \end{bmatrix} \begin{bmatrix} y \\ \tilde{y} \end{bmatrix} = A^T y + \tilde{A}^T \tilde{y} = 0 \tag{10.5}$$

$$\begin{bmatrix} y \\ \tilde{y} \end{bmatrix} \geq 0 \tag{10.6}$$

$$\begin{bmatrix} b^T & \tilde{b}^T \end{bmatrix} \begin{bmatrix} y \\ \tilde{y} \end{bmatrix} = b^T y + \tilde{b}^T \tilde{y} < 0. \tag{10.7}$$

From the last condition, we see that either $b^T y < 0$ or $\tilde{b}^T \tilde{y} < 0$ (or both). Without loss of generality, we may assume that

$$b^T y < 0.$$

Farkas' Lemma (this time applied in the other direction) together with the nonemptiness of P now implies that

$$A^T y \neq 0.$$

Put

$$H = \left\{ x : (A^T y)^T x \leq b^T y \right\} \qquad \text{and} \qquad \tilde{H} = \left\{ x : (A^T y)^T x \geq -\tilde{b}^T \tilde{y} \right\}.$$

These sets are clearly halfspaces. To finish the proof, we must show that they are disjoint and contain their corresponding polyhedra.

First of all, it follows from (10.7) that H and $\tilde{H}$ are disjoint. Indeed, suppose that $x \in H$. Then $(A^T y)^T x \leq b^T y < -\tilde{b}^T \tilde{y}$, which implies that x is not in $\tilde{H}$.

To show that $P \subset H$, fix x in P. Then $Ax \leq b$. Since $y \geq 0$ (as we know from (10.6)), it follows then that $y^T Ax \leq y^T b$. But this is exactly the condition that says that x belongs to H. Since x was an arbitrary point in P, it follows that $P \subset H$.

Showing that $\tilde{P}$ is a subset of $\tilde{H}$ is similar. Indeed, suppose that $x \in \tilde{P}$. Then $\tilde{A}x \leq \tilde{b}$. Multiplying on the left by $-\tilde{y}^T$ and noting that $\tilde{y} \geq 0$, we see that $-\tilde{y}^T \tilde{A}x \geq$

$-\tilde{y}^T\tilde{b}$. But from (10.5) we see that $-\tilde{y}^T\tilde{A}x = y^TAx$, and so this last inequality is exactly the condition that $x \in \tilde{H}$. Again, the arbitrariness of $x \in \tilde{P}$ implies that $\tilde{P} \subset \tilde{H}$, and the proof is complete. □

4. Farkas' Lemma

The following result, known as Farkas' Lemma, played a fundamental role in the proof of the separation theorem of the preceding section (Theorem 10.4). In this section, we state it formally as a lemma and give its proof.

LEMMA 10.5. *The system $Ax \leq b$ has no solutions if and only if there is a y such that*

$$\begin{aligned} A^Ty &= 0 \\ y &\geq 0 \\ b^Ty &< 0. \end{aligned} \tag{10.8}$$

PROOF. Consider the linear program

$$\begin{aligned} \text{maximize} \quad & 0 \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$

and its dual

$$\begin{aligned} \text{minimize} \quad & b^Ty \\ \text{subject to} \quad & A^Ty = 0 \\ & y \geq 0. \end{aligned}$$

Clearly, the dual is feasible (just take $y = 0$). So if the primal is feasible, then the dual is bounded. Also, if the primal is infeasible, the dual must be unbounded. That is, the primal is infeasible if and only if the dual is unbounded. To finish the proof, we claim that the dual is unbounded if and only if there exists a solution to (10.8). Indeed, suppose that the dual is unbounded. The dual simplex method is guaranteed to prove that it is unbounded, and it does so as follows. At the last iteration, a step direction Δy is computed that preserves feasibility, i.e.,

$$A^T\Delta y = 0,$$

is a descent direction for the objective function, i.e.,

$$b^T\Delta y < 0,$$

and is a direction for which the step length is unbounded, i.e.,

$$\Delta y \geq 0.$$

But these three properties show that Δy is the solution to (10.8) that we were looking for. Conversely, suppose that there is a solution to (10.8). Call it Δy. It is easy to see that starting from $y = 0$, this step direction provides an unbounded decrease in the objective function. This completes the proof. □

5. Strict Complementarity

In this section, we consider the usual inequality-form linear programming problem, which we write with its slack variables shown explicitly:

$$\begin{aligned} &\text{maximize } && c^T x \\ &\text{subject to } && Ax + w = b \\ &&& x, w \geq 0. \end{aligned} \tag{10.9}$$

As we know, the dual can be written as follows:

$$\begin{aligned} &\text{minimize } && b^T y \\ &\text{subject to } && A^T y - z = c \\ &&& y, z \geq 0. \end{aligned} \tag{10.10}$$

In our study of duality theory in Chapter 5, we saw that every pair of optimal solutions to these two problems possesses a property called complementary slackness. If (x^*, w^*) denotes an optimal solution to the primal and (y^*, z^*) denotes an optimal solution to the dual, then the complementary slackness theorem says that, for each $j = 1, 2, \ldots, n$, either $x_j^* = 0$ or $z_j^* = 0$ (or both) and, for each $i = 1, 2, \ldots, m$, either $y_i^* = 0$ or $w_i^* = 0$ (or both). In this section, we shall prove that there are optimal pairs of solutions for which the parenthetical "or both" statements don't happen. That is, there are optimal solutions for which exactly one member of each pair (x_j^*, z_j^*) vanishes and exactly one member from each pair (y_i^*, w_i^*) vanishes. In such cases, we say that the optimal solutions are *strictly complementary* to each other. The strictness of the complementary slackness is often expressed by saying that $x^* + z^* > 0$ and $y^* + w^* > 0$[2].

As a warm-up, we prove the following theorem.

THEOREM 10.6. *If both the primal and the dual have feasible solutions, then there exists a primal feasible solution* $(\bar{x}, \bar{w})$ *and a dual feasible solution* $(\bar{y}, \bar{z})$ *such that* $\bar{x} + \bar{z} > 0$ *and* $\bar{y} + \bar{w} > 0$.

PROOF. If there is a feasible primal solution $\bar{x}$ for which $\bar{x}_j > 0$, then it doesn't matter whether there is a feasible dual solution whose jth slack variable is strictly positive. But what about indices j for which $x_j = 0$ for every feasible solution? Let j be such an index. Consider the following linear programming problem:

$$\begin{aligned} &\text{maximize } && x_j \\ &\text{subject to } && Ax \leq b \\ &&& x \geq 0. \end{aligned} \tag{10.11}$$

[2]Given any vector ξ, we use the notation $\xi > 0$ to indicate that every component of ξ is strictly positive: $\xi_j > 0$ for all j

This problem is feasible, since its constraints are the same as for the original primal problem (10.9). Furthermore, it has an optimal solution (the corresponding objective function value is zero). The dual of (10.11) is:

$$\begin{aligned} \text{minimize} \quad & b^T y \\ \text{subject to} \quad & A^T y \geq e_j \\ & y \geq 0. \end{aligned}$$

By the strong duality theorem, the dual has an optimal solution, say y'. Letting z' denote the corresponding slack variable, we have that

$$\begin{aligned} A^T y' - z' &= e_j \\ y', z' &\geq 0. \end{aligned}$$

Now, let y be any feasible solution to (10.10) and let z be the corresponding slack variable. Then the above properties of y' and z' imply that $y + y'$ is feasible for (10.10) and its slack is $z + z' + e_j$. Clearly, for this dual feasible solution we have that the jth component of its vector of slack variables is at least 1. To summarize, we have shown that, for each j, there exists a primal feasible solution, call it $(x^{(j)}, w^{(j)})$, and a dual feasible solution, call it $(y^{(j)}, z^{(j)})$, such that $x_j^{(j)} + z_j^{(j)} > 0$. In the same way, one can exhibit primal and dual feasible solutions for which each individual dual variable and its corresponding primal slack add to a positive number. To complete the proof, we now form a strict convex combination of these $n + m$ feasible solutions. Since the feasible region for a linear programming problem is convex, these convex combinations preserve primal and dual feasibility. Since the convex combination is strict, it follows that every primal variable and its dual slack add to a strictly positive number as does every dual variable and its primal slack. □

A variable x_j that must vanish in order for a linear programming problem to be feasible is called a *null variable*. The previous theorem says that if a variable is null, then its dual slack is not null.

The following theorem is called the *Strict Complementary Slackness Theorem*

THEOREM 10.7. *If a linear programming problem has an optimal solution, then there is an optimal solution* (x^*, w^*) *and an optimal dual solution* (y^*, z^*) *such that* $x^* + z^* > 0$ *and* $y^* + w^* > 0$.

We already know from the complementary slackness theorem (Theorem 5.1) that x^* and z^* are complementary to each other as are y^* and w^*. This theorem then asserts that the complementary slackness is strict.

PROOF. The proof is much the same as the proof of Theorem 10.6 except this time we look at an index j for which x_j vanishes in every *optimal* solution. We then

consider the following problem:

$$\begin{aligned} \text{maximize} \quad & x_j \\ \text{subject to} \quad & Ax \leq b \\ & c^T x \geq \zeta^* \\ & x \geq 0, \end{aligned} \tag{10.12}$$

where ζ^* denotes the objective value of the optimal solution to the original problem. In addition to the dual variables y corresponding to the $Ax \leq b$ constraints, there is one more dual variable, call it t, associated with the constraint $c^T x \geq \zeta^*$. The analysis of problem (10.12) is similar to the analysis given in Theorem 10.6 except that one must now consider two cases: (a) the optimal value of t is strictly positive and (b) the optimal value of t vanishes. The details are left as an exercise (see Exercise 10.6). □

Exercises

10.1 Is $\mathbb{R}^n$ a polyhedron?

10.2 For each $b \in \mathbb{R}^m$, let $\xi^*(b)$ denote the optimal objective function value for the following linear program:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0. \end{aligned}$$

Suppose that $\xi^*(b) < \infty$ for all b. Show that the function $\xi^*(b)$ is concave (a function f on $\mathbb{R}^m$ is called *concave* if $f(tx+(1-t)y) \geq tf(x)+(1-t)f(y)$ for all x and y in $\mathbb{R}^m$ and all $0 < t < 1$). *Hint: Consider the dual problem.*

10.3 Describe how one needs to modify the proof of Theorem 10.4 to get a proof of the following result:

> Let P and $\tilde{P}$ be two disjoint polyhedra in $\mathbb{R}^n$. Then there exist disjoint generalized halfspaces H and $\tilde{H}$ such that $P \subset H$ and $\tilde{P} \subset \tilde{H}$.

10.4 Find a strictly complementary solution to the following linear programming problem and its dual:

$$\begin{aligned} \text{maximize} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & 4x_1 + 2x_2 \leq 6 \\ & x_2 \leq 1 \\ & 2x_1 + x_2 \leq 3 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

10.5 There is a slight oversimplification in the proof of Theorem 10.3. Can you spot it? Can you fix it?

10.6 Complete the proof of Theorem 10.7.

10.7 *Interior solutions.* Prove the following statement: If a linear programming problem has feasible solutions and the set of feasible solutions is bounded, then there is a strictly positive dual feasible solution: $y > 0$ and $z > 0$. *Hint. It is easier to prove the equivalent statement: if a linear programming problem has feasible solutions and the dual has null variables, then the set of primal feasible solutions is an unbounded set.*

Notes

Carathéodory (1907) proved Theorem 10.3. Farkas (1902) proved Lemma 10.5. Several similar results were discovered by many others, including Gordan (1873), Stiemke (1915), Ville (1938), and Tucker (1956). The standard reference on convex analysis is Rockafellar (1970).

CHAPTER 11

Game Theory

In this chapter, we shall study if not the most practical then certainly an elegant application of linear programming. The subject is called game theory, and we shall focus on the simplest type of game, called the *finite two-person zero-sum game*, or just *matrix game* for short. Our primary goal shall be to prove the famous Minimax Theorem, which was first discovered and proved by John von Neumann in 1928. His original proof of this theorem was rather involved and depended on another beautiful theorem from mathematics, the Brouwer Fixed-Point Theorem. However, it eventually became clear that the solution of matrix games could be found by solving a certain linear programming problem and that the Minimax Theorem is just a fairly straightforward consequence of the Duality Theorem.

1. Matrix Games

A *matrix game* is a two-person game defined as follows. Each person first selects, independently of the other, an action from a finite set of choices (the two players in general will be confronted with different sets of actions from which to choose). Then both reveal to each other their choice. If we let i denote the first player's choice and j denote the second player's choice, then the rules of the game stipulate that the first player will pay the second player a_{ij} dollars. The array of possible payments

$$A = [a_{ij}]$$

is presumed known to both players before the game begins. Of course, if a_{ij} is negative for some pair (i, j), then the payment goes in the reverse direction — from the second player to the first. For obvious reasons, we shall refer to the first player as the *row player* and the second player as the *column player*. Since we have assumed that the row player has only a finite number of actions from which to choose, we can enumerate these actions and assume without loss of generality that i is simply an integer selected from 1 to m. Similarly, we can assume that j is simply an index ranging from 1 to n (in its real-world interpretation, row action 3 will generally have nothing to do with column action 3—the number 3 simply indicates that it is the third action in the enumerated list of choices).

Let us look at a specific familiar example. Namely, consider the game every child knows, called Paper–Scissors–Rock. To refresh the memory of older readers,

this is a two-person game in which at the count of three each player declares either Paper, Scissors, or Rock. If both players declare the same object, then the round is a draw. But Paper loses to Scissors (since scissors can cut a piece of paper), Scissors loses to Rock (since a rock can dull scissors), and finally Rock loses to Paper (since a piece of paper can cover up a rock—it's a weak argument but that's the way the game is defined). Clearly, for this game, if we enumerate the actions of declaring Paper, Scissors, or Rock as 1, 2, 3, respectively, then the payoff matrix is

$$\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}.$$

With this matrix, neither player has an obvious (i.e., deterministic) winning strategy. If the column player were always to declare Paper (hoping that the row player will declare Rock), then the row player could counter by always declaring Scissors and guaranteeing herself a winning of one dollar in every round. In fact, if the column player were to stick to any specific declaration, then the row player would eventually get wise to it and respond appropriately to guarantee that she wins. Of course, the same logic applies to the row player. Hence, neither player should employ the same declaration over and over. Instead, they should randomize their declarations. In fact, due to the symmetry of this particular game, both players should make each of the three possible declarations with equal likelihood.

But what about less trivial games? For example, suppose that the payoffs in the Paper–Scissors–Rock game are altered so that the payoff matrix becomes

$$A = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix}.$$

This new game still has the property that every deterministic strategy can be foiled by an intelligent opponent. Hence, randomized behavior remains appropriate. But the best probabilities are no longer uniformly $1/3$. Also, who has the edge in this game? Since the total of the payoffs that go from the row player to the column player is 10 whereas the total of the payoffs that go to the row player is 11, we suspect that the row player might have the edge. But this is just a guess. Is it correct? If it is correct, how much can the row player expect to win on average in each round? If the row player knows this number accurately and the column player does not, then the row player could offer to pay the column player a small fee for playing each round. If the fee is smaller than the expected winnings, then the row player can still be confident that over time she will make a nice profit. The purpose of this chapter is to answer these questions precisely.

Let us return now to the general setup. Consider the row player. By a *randomized strategy*, we mean that, at each play of the game, it appears (from the column player's viewpoint) that the row player is making her choices at random according to some fixed probability distribution. Let y_i denote the probability that the row player selects action i. The vector y composed of these probabilities is called a *stochastic vector*. Mathematically, a vector is a stochastic vector if it has nonnegative components that sum up to one:

$$y \geq 0 \qquad \text{and} \qquad e^T y = 1,$$

where e denotes the vector consisting of all ones. Of course, the column player must also adopt a randomized strategy. Let x_j denote the probability that the column player selects action j, and let x denote the stochastic vector composed of these probabilities.

The expected payoff to the column player is computed by summing over all possible outcomes the payoff associated with that outcome times the probability of the outcome. The set of possible outcomes is simply the set of pairs (i, j) as i ranges over the row indices $(1, 2, \ldots, m)$ and j ranges over the column indices $(1, 2, \ldots, n)$. For outcome (i, j) the payoff is a_{ij}, and, assuming that the row and column players behave independently, the probability of this outcome is simply $y_i x_j$. Hence, the expected payoff to the column player is

$$\sum_{i,j} y_i a_{ij} x_j = y^T A x.$$

2. Optimal Strategies

Suppose that the column player adopts strategy x (i.e., decides to play in accordance with the stochastic vector x). Then the row player's best defense is to use the strategy y^* that achieves the following minimum:

$$\begin{aligned} &\text{minimize} && y^T A x \\ &\text{subject to} && e^T y = 1 \\ &&& y \geq 0. \end{aligned} \tag{11.1}$$

From the fundamental theorem of linear programming, we know that this problem has a basic optimal solution. For this problem, the basic solutions are simply y vectors that are zero in every component except for one, which is one. That is, the basic optimal solutions correspond to deterministic strategies. This is fairly obvious if we look again at our example. Suppose that

$$x = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}.$$

Then

$$Ax = \begin{bmatrix} -1/3 \\ 1/3 \\ -1/3 \end{bmatrix},$$

and so the row player's best choice is to select either $i = 1$ (Paper) or $i = 3$ (Rock) or any combination thereof. That is, an optimal solution is $y^* = (1, 0, 0)$ (it is not unique).

Since for any given x the row player will adopt the strategy that achieves the minimum in (11.1), it follows that the column player should employ a strategy x^* that attains the following maximum:

$$\max_x \min_y \; y^T A x, \tag{11.2}$$

where the max and the min are over all stochastic vectors (of the appropriate dimension).

The question then becomes: how do we solve (11.2)? It turns out that this problem can be reformulated as a linear programming problem. Indeed, we have already seen that the inner optimization (the minimization) can be taken over just the deterministic strategies:

$$\min_y y^T A x = \min_i e_i^T A x,$$

where we have used e_i to denote the vector of all zeros except for a one in position i. Hence, the max-min problem given in (11.2) can be rewritten as

$$\begin{aligned} &\text{maximize} \;\; \left(\min_i e_i^T A x\right) \\ &\text{subject to} \;\; \sum_{j=1}^{n} x_j = 1 \\ &\qquad\qquad x_j \geq 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

Now, if we introduce a new variable, v, representing a lower bound on the $e_i^T A x$'s, then we see that the problem can be recast as a linear program:

$$\begin{aligned} &\text{maximize} \;\; v \\ &\text{subject to} \qquad v \leq e_i^T A x \qquad i = 1, 2, \ldots, m \\ &\qquad\qquad \sum_{j=1}^{n} x_j = 1 \\ &\qquad\qquad x_j \geq 0 \qquad\qquad j = 1, 2, \ldots, n. \end{aligned}$$

Switching back to vector notation, the problem can be written as

$$\begin{aligned} &\text{maximize } v \\ &\text{subject to } ve - Ax \leq 0 \\ &\qquad\qquad e^T x = 1 \\ &\qquad\qquad x \geq 0. \end{aligned}$$

Finally, writing in block-matrix form, we get

$$\begin{aligned} &\text{maximize } \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} \\ &\text{subject to } \begin{bmatrix} -A & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} \leq \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &\qquad\qquad\qquad\qquad\qquad\qquad x \geq 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad v \text{ free.} \end{aligned} \tag{11.3}$$

Now let's turn it around. By symmetry, the row player seeks a strategy y^* that attains optimality in the following min-max problem:

$$\min_y \max_x y^T A x,$$

which can be reformulated as the following linear program:

$$\begin{aligned} &\text{minimize } u \\ &\text{subject to } ue - A^T y \geq 0 \\ &\qquad\qquad e^T y = 1 \\ &\qquad\qquad y \geq 0. \end{aligned}$$

Writing in block-matrix form, we get

$$\begin{aligned} &\text{minimize } \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} \\ &\text{subject to } \begin{bmatrix} -A^T & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} \geq \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &\qquad\qquad\qquad\qquad\qquad\qquad y \geq 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad u \text{ free.} \end{aligned} \tag{11.4}$$

3. The Minimax Theorem

Having reduced the computation of the optimal strategies x^* and y^* to the solution of linear programs, it is now a simple matter to show that they are consistent with each

other. The next theorem, which establishes this consistency, is called the Minimax Theorem:

THEOREM 11.1. *There exist stochastic vectors x^* and y^* for which*

$$\max_x \; {y^*}^T Ax = \min_y \; y^T Ax^*.$$

PROOF. The proof follows trivially from the observation that (11.4) is the dual of (11.3). Therefore, $v^* = u^*$. Furthermore,

$$v^* = \min_i e_i^T Ax^* = \min_y y^T Ax^*,$$

and similarly,

$$u^* = \max_j e_j^T A^T y^* = \max_x x^T A^T y^* = \max_x {y^*}^T Ax.$$

□

The common optimal value $v^* = u^*$ of the primal and dual linear programs is called the *value* of the game. From the Minimax Theorem, we see that, by adopting strategy y^*, the row player assures herself of losing no more than v units per round on the average. Similarly, the column player can assure himself of winning at least v units per round on the average by adopting strategy x^*. A game whose value is zero is therefore a *fair game*. Games where the roles of the two players are interchangeable are clearly fair. Such games are called *symmetric*. They are characterized by payoff matrices having the property that $a_{ij} = -a_{ji}$ for all i and j (in particular, m must equal n and the diagonal must vanish).

For the Paper–Scissors–Rock game, the linear programming problem that the column player needs to solve is

$$\begin{array}{ll} \text{maximize} & w \\ \text{subject to} & \begin{bmatrix} 0 & -1 & 2 & 1 \\ 3 & 0 & -4 & 1 \\ -5 & 6 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ w \end{bmatrix} \begin{matrix} \leq \\ \leq \\ \leq \\ = \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ & x_1, x_2, x_3 \geq 0 \\ & w \text{ free.} \end{array}$$

In nonmatrix notation, it looks like this:

$$\begin{array}{ll} \text{maximize} & w \\ \text{subject to} & -x_2 + 2x_3 + w \le 0 \\ & 3x_1 \quad - 4x_3 + w \le 0 \\ & -5x_1 + 6x_2 \quad + w \le 0 \\ & x_1 + x_2 + x_3 \quad = 1 \\ & x_1,\ x_2,\ x_3 \ge 0. \end{array}$$

This linear programming problem deviates from our standard inequality form in two respects: (1) it has an equality constraint and (2) it has a free variable. There are several ways in which one can convert this problem into standard form. The most compact conversion is as follows. First, use the equality constraint to solve explicitly for one of the x_j's, say x_3:

$$x_3 = 1 - x_1 - x_2.$$

Then eliminate this variable from the remaining equations to get

$$\begin{array}{ll} \text{maximize} & w \\ \text{subject to} & -2x_1 - 3x_2 + w \le -2 \\ & 7x_1 + 4x_2 + w \le 4 \\ & -5x_1 + 6x_2 + w \le 0 \\ & x_1 + x_2 \le 1 \\ & x_1,\ x_2 \ge 0. \end{array}$$

The elimination of x_3 has changed the last constraint from an equality into an inequality.

The next step is to write down a starting dictionary. To do this, we need to introduce slack variables for each of these constraints. It is natural (and desirable) to denote the slack variable for the last constraint by x_3. In fact, doing this, we get the following starting dictionary:

$$\begin{array}{rl} \xi = & w \\ \hline x_4 = & -2 + 2x_1 + 3x_2 - w \\ x_5 = & 4 - 7x_1 - 4x_2 - w \\ x_6 = & 5x_1 - 6x_2 - w \\ x_3 = & 1 - x_1 - x_2\ . \end{array}$$

The variable w is not constrained to be nonnegative. Therefore, there is no reason for it to be nonbasic. Let us do an arbitrary pivot with w as the entering variable and any basic variable as the leaving variable (well, not exactly any—we must make sure that

it causes no division by 0, so therefore x_3 is not a candidate). Picking x_4 to leave, we get

$$\begin{aligned}\xi &= -2 + 2x_1 + 3x_2 - x_4\\ \hline w &= -2 + 2x_1 + 3x_2 - x_4\\ x_5 &= 6 - 9x_1 - 7x_2 + x_4\\ x_6 &= 2 + 3x_1 - 9x_2 + x_4\\ x_3 &= 1 - x_1 - x_2\ .\end{aligned}$$

Since w is free of sign constraints, it will never leave the basis (since a leaving variable is, by definition, a variable that hits its lower bound—w has no such bound). Therefore, we may as well remove it from the dictionary altogether; it can always be computed at the end. Hence, we note that

$$w = -2 + 2x_1 + 3x_2 - x_4,$$

or better yet that

$$w = \xi,$$

and the dictionary now becomes

$$\begin{aligned}\xi &= -2 + 2x_1 + 3x_2 - x_4\\ \hline x_5 &= 6 - 9x_1 - 7x_2 + x_4\\ x_6 &= 2 + 3x_1 - 9x_2 + x_4\\ x_3 &= 1 - x_1 - x_2\ .\end{aligned}$$

At last, we are in a position to apply the simplex method. Two (tedious) iterations bring us to the optimal dictionary. Since it involves fractions, we multiply each equation by an integer to make each number in the dictionary an integer. Indeed, after multiplying by 102, the optimal dictionary is given by

$$\begin{aligned}102\xi &= -16 - 27x_5 - 13x_6 - 62x_4\\ \hline 102x_1 &= 40 + 9x_5 + 7x_6 + 2x_4\\ 102x_2 &= 36 - 3x_5 - 9x_6 + 12x_4\\ 102x_3 &= 26 + 12x_5 + 2x_6 - 14x_4.\end{aligned}$$

From this dictionary, it is easy to read off the optimal primal solution:

$$x^* = \begin{bmatrix} 40/102 \\ 36/102 \\ 26/102 \end{bmatrix}.$$

Also, since x_4, x_5, and x_6 are complementary to y_1, y_2, and y_3 in the dual problem, the optimal dual solution is

$$y^* = \begin{bmatrix} 62/102 \\ 27/102 \\ 13/102 \end{bmatrix}.$$

Finally, the value of the game is

$$w^* = \xi^* = -16/102 = -0.15686275,$$

which indicates that the row player does indeed have an advantage and can expect to make on the average close to 16 cents per round.

4. Poker

Some card games such as poker involve a round of bidding in which the players at times *bluff* by increasing their bid in an attempt to coerce their opponents into backing down, even though if the challenge is accepted they will surely lose. Similarly, they will sometimes *underbid* to give their opponents false hope. In this section, we shall study a simplified version of poker (the real game is too hard to analyze) to see if bluffing and underbidding are justified bidding strategies.

Simplified poker involves two players, A and B, and a deck having three cards, 1, 2, and 3. At the beginning of a round, each player "antes up" \$1 and is dealt one card from the deck. A bidding session follows in which each player in turn, starting with A, either (a) *bets* and adds \$1 to the "kitty" or (b) *passes*. Bidding terminates when

a bet is followed by a bet,
a pass is followed by a pass, or
a bet is followed by a pass.

In the first two cases, the winner of the round is decided by comparing cards, and the kitty goes to the player with the higher card. In the third case, bet followed by pass, the player who bet wins the round independently of who had the higher card (in real poker, the player who passes is said to *fold*).

With these simplified betting rules, there are only five possible betting scenarios:

A passes,	B passes:		\$1 to holder of higher card
A passes,	B bets,	A passes:	\$1 to B
A passes,	B bets,	A bets:	\$2 to holder of higher card
A bets,	B passes:		\$1 to A
A bets,	B bets:		\$2 to holder of higher card

After being dealt a card, player A will decide to bet along one of three lines:

1. Pass. If B bets, pass again.
2. Pass. If B bets, bet.
3. Bet.

Similarly, after being dealt a card, player B can bet along one of four lines:

1. Pass no matter what.
2. If A passes, pass, but if A bets, bet.
3. If A passes, bet, but if A bets, pass.
4. Bet no matter what.

To model the situation as a matrix game, we must identify each player's pure strategies. A pure strategy is a statement of what line of betting a player intends to follow for each possible card that the player is dealt. Hence, the players' pure strategies can be denoted by triples (y_1, y_2, y_3), where y_i is the line of betting that the player will use when holding card i. (For player A, the y_i's can take values 1, 2, and 3, whereas for player B, they can take values 1, 2, 3, and 4.)

Given a pure strategy for both players, one can compute the average payment from, say, A to B. For example, suppose that player A adopts strategy $(3, 1, 2)$ and player B adopts strategy $(3, 2, 4)$. There are six ways in which the cards can be dealt, and we can analyze each of them as follows:

card dealt		betting session			payment
A	B				A to B
1	2	A bets,	B bets		2
1	3	A bets,	B bets		2
2	1	A passes,	B bets,	A passes	1
2	3	A passes,	B bets,	A passes	1
3	1	A passes,	B bets,	A bets	-2
3	2	A passes,	B passes		-1

Since each of the six deals are equally likely, the average payment from A to B is $(2 + 2 + 1 + 1 - 2 - 1)/6 = 0.5$

The calculation of the average payment must be carried out for every combination of pairs of strategies. How many are there? Player A has $3 \times 3 \times 3 = 27$ pure strategies and player B has $4 \times 4 \times 4 = 64$ pure strategies. Hence, there are $27 \times 64 = 1728$ pairs.

Calculating the average payment for all these pairs is a daunting task. Fortunately, we can reduce the number of pure strategies (and hence the number of pairs) that need to be considered by making a few simple observations.

The first observation is that a player holding a 1 should never answer a bet with a bet, since the player will lose regardless of the answering bet and will lose less by passing. This logic implies that, when holding a 1,

player A should refrain from betting along line 2;
player B should refrain from betting along lines 2 and 4.

More clearly improvable strategies can be ruled out when holding the highest card. For example, a player holding a 3 should never answer a bet with a pass, since by passing the player will lose, but by betting the player will win. Furthermore, when holding a 3, a player should always answer a pass with a bet, since in either case the player is going to win, but answering with a bet opens the possibility of the opponent betting again and thereby increasing the size of the win for the player holding the 3. Hence, when holding a 3,

player A should refrain from betting along line 1;
player B should refrain from betting along lines 1, 2, and 3.

Eliminating from consideration the above lines of betting, we see that player A now has $2 \times 3 \times 2 = 12$ pure strategies and player B has $2 \times 4 \times 1 = 8$ pure strategies. The number of pairs has therefore dropped to 96—a significant reduction. Not only do we eliminate these "bad" strategies from the mathematical model but also we assume that both players know that these bad strategies will not be used. That is, player A can assume that player B will play intelligently, and player B can assume the same of A. This knowledge then leads to further reductions. For example, when holding a 2, player A should refrain from betting along line 3. To reach this conclusion, we must carefully enumerate possibilities. Since player A holds the 2, player B holds either the 1 or the 3. But we've already determined what player B will do in both of those cases. Using this knowledge, it is not hard to see that player A would be unwise to bet along line 3. A similar analysis reveals that, when holding a 2, player B should refrain from lines 3 and 4. Therefore, player A now has only $2 \times 2 \times 2 = 8$ pure strategies and player B has only $2 \times 2 \times 1 = 4$ pure strategies.

At this point, no further reductions are possible. Computing the payoff matrix, we get

$$A = \begin{array}{c} \\ (1,1,2) \\ (1,1,3) \\ (1,2,2) \\ (1,2,3) \\ (3,1,2) \\ (3,1,3) \\ (3,2,2) \\ (3,2,3) \end{array} \begin{array}{c} \begin{array}{cccc} (1,1,4) & (1,2,4) & (3,1,4) & (3,2,4) \end{array} \\ \begin{bmatrix} & & \frac{1}{6} & \frac{1}{6} \\ & -\frac{1}{6} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ \frac{1}{6} & & & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{3} & & \frac{1}{2} \\ -\frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \\ & \frac{1}{2} & -\frac{1}{3} & \frac{1}{6} \\ & \frac{1}{3} & -\frac{1}{6} & \frac{1}{6} \end{bmatrix} \end{array}.$$

Solving the matrix game, we find that

$$y^* = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{6} \end{bmatrix}^T$$

and

$$x^* = \begin{bmatrix} \frac{2}{3} & 0 & 0 & \frac{1}{3} \end{bmatrix}^T.$$

These stochastic vectors can be summarized as simple statements of the optimal randomized strategies for the two players. Indeed, player A's optimal strategy is as follows:

> when holding 1, mix lines 1 and 3 in 5:1 proportion;
> when holding 2, mix lines 1 and 2 in 1:1 proportion;
> when holding 3, mix lines 2 and 3 in 1:1 proportion.

Similarly, player B's optimal strategy can be described as

> when holding 1, mix lines 1 and 3 in 2:1 proportion;
> when holding 2, mix lines 1 and 2 in 2:1 proportion;
> when holding 3, use line 4.

Note that it is optimal for player A to use line 3 when holding a 1 at least some of the time. Since line 3 says to bet, this bet is a bluff. Player B also bluffs sometimes, since betting line 3 is sometimes used when holding a 1. Clearly, the optimal strategies also exhibit some underbidding.

Exercises

11.1 Players A and B each hide a nickel or a dime. If the hidden coins match, player A gets both; if they don't match, then B gets both. Find the optimal

strategies. Which player has the advantage? Solve the problem for arbitrary denominations a and b.

11.2 Players A and B each pick a number between 1 and 100. The game is a draw if both players pick the same number. Otherwise, the player who picks the smaller number wins unless that smaller number is one less than the opponent's number, in which case the opponent wins. Find the optimal strategy for this game.

11.3 We say that row r *dominates* row s if $a_{rj} \geq a_{sj}$ for all $j = 1, 2, \ldots, n$. Similarly, column r is said to dominate column s if $a_{ir} \geq a_{is}$ for all $i = 1, 2, \ldots, m$. Show that

(a) If a row (say, r) dominates another row, then the row player has an optimal strategy y^* in which $y_r^* = 0$.

(b) If a column (say, s) is dominated by another column, then the column player has an optimal strategy x^* in which $x_s^* = 0$.

Use these results to reduce the following payoff matrix to a 2×2 matrix:

$$\begin{bmatrix} -6 & 2 & -4 & -7 & -5 \\ 0 & 4 & -2 & -9 & -1 \\ -7 & 3 & -3 & -8 & -2 \\ 2 & -3 & 6 & 0 & 3 \end{bmatrix}.$$

11.4 Solve simplified poker assuming that antes are \$2 and bets are \$1.

11.5 Give necessary and sufficient conditions for the rth pure strategy of the row and the sth pure strategy of the column player to be simultaneously optimal.

11.6 Use the Minimax Theorem to show that

$$\max_x \min_y \; y^T A x = \min_y \max_x \; y^T A x.$$

11.7 *Bimatrix Games.* Consider the following two-person game defined in terms of a pair of $m \times n$ matrices A and B: if the row player selects row index i and the column player selects column index j, then the row player pays a_{ij} dollars and the column player pays b_{ij} dollars. Stochastic vectors x^* and y^* are said to form a *Nash equilibrium* if

$$\begin{aligned} {y^*}^T A x^* &\leq y^T A x^* \qquad \text{for all } y \\ {y^*}^T B x^* &\leq {y^*}^T B x \qquad \text{for all } x. \end{aligned}$$

The purpose of this exercise is to relate Nash equilibria to the problem of finding vectors x and y that satisfy

$$\begin{bmatrix} 0 & -A \\ -B^T & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} + \begin{bmatrix} w \\ z \end{bmatrix} = \begin{bmatrix} -e \\ -e \end{bmatrix},$$

$$y_i w_i = 0, \qquad \text{for all } i, \tag{11.5}$$

$$x_j z_j = 0, \qquad \text{for all } j,$$

$$x, w, y, z \geq 0$$

(vectors w and z can be thought of as being defined by the matrix equality). Problem (11.5) is called a *linear complementarity problem*.

(a) Show that there is no loss in generality in assuming that A and B have all positive entries.

(b) Assuming that A and B have all positive entries, show that, if (x^*, y^*) is a Nash equilibrium, then

$$x' = \frac{x^*}{y^{*T} A x^*}, \qquad y' = \frac{y^*}{y^{*T} B x^*}$$

solves the linear complementarity problem (11.5).

(c) Show that, if (x', y') solves the linear complementarity problem (11.5), then

$$x^* = \frac{x'}{e^T x'}, \qquad y^* = \frac{y'}{e^T y'}$$

is a Nash equilibrium.

(An algorithm for solving the linear complementarity problem is developed in Exercise 17.7.)

11.8 *The Game of Morra.* Two players simultaneously throw out one or two fingers and call out their guess as to what the total sum of the outstretched fingers will be. If a player guesses right, but his opponent does not, he receives payment equal to his guess. In all other cases, it is a draw.

(a) List the pure strategies for this game.

(b) Write down the payoff matrix for this game.

(c) Formulate the row player's problem as a linear programming problem. (*Hint: Recall that the row player's problem is to minimize the maximum expected payout.*)

(d) What is the value of this game?

(e) Find the optimal randomized strategy.

11.9 *Heads I Win—Tails You Lose.* In the classical coin-tossing game, player A tosses a fair coin. If it comes up heads player B pays player A \$2 but if it comes up tails player A pays player B \$2. As a two-person zero-sum game, this game is rather trivial since neither player has anything to *decide* (after agreeing to play the game). In fact, the matrix for this game is a 1×1 matrix

with only a zero in it, which represents the expected payoff from player A to B.

Now consider the same game with the following twist. Player A is allowed to peek at the outcome and then decide either to stay in the game or to bow out. If player A bows out, then he automatically loses but only has to pay player B $1. Of course, player A must inform player B of his decision. If his decision is to stay in the game, then player B has the option either to stay in the game or not. If she decides to get out, then she loses $1 to player A. If both players stay in the game, then the rules are as in the classical game: heads means player A wins, tails means player B wins.

(a) List the strategies for each player in this game. (Hint: Don't forget that a strategy is something that a player has control over.)
(b) Write down the payoff matrix.
(c) A few of player A's strategies are uniformly inferior to others. These strategies can be ruled out. Which of player A's strategies can be ruled out?
(d) Formulate the row player's problem as a linear programming problem. (*Hints: (1) Recall that the row player's problem is to minimize the maximum expected payout. (2) Don't include rows that you ruled out in the previous part.*)
(e) Find the optimal randomized strategy.
(f) Discuss whether this game is interesting or not.

Notes

The Minimax Theorem was proved by von Neumann (1928). Important references include Gale et al. (1951), von Neumann & Morgenstern (1947), Karlin (1959), and Dresher (1961). Simplified poker was invented and analyzed by Kuhn (1950). Exercises 11.1 and 11.2 are borrowed from Chvátal (1983).

CHAPTER 12

Regression

In this chapter, we shall study an application of linear programming to an area of statistics called regression. As a specific example, we shall use size and iteration-count data collected from a standard suite of linear programming problems to derive a regression estimate of the number of iterations needed to solve problems of a given size.

1. Measures of Mediocrity

We begin our discussion with an example. Here are the midterm exam scores for a linear programming course:

$$28, 62, 80, 84, 86, 86, 92, 95, 98.$$

Let m denote the number of exam scores (i.e., $m = 9$) and let b_i, $i = 1, 2, \ldots, m$, denote the actual scores (arranged in increasing order as above). The most naive measure of the "average" score is just the *mean* value, $\bar{x}$, defined by

$$\bar{x} = \frac{1}{m} \sum_{i=1}^{m} b_i = 79.0.$$

This is an example of a statistic, which, by definition, is a function of a set of data. Statistics are computed so that one does not need to bother with reporting large tables of raw numbers. (Admittedly, the task of reporting the above list of 9 exam scores is not very onerous, but this is just an example.) Now, suppose the professor in question did not report the scores but instead just gave summary statistics. Consider the student who got an 80 on the exam. This student surely didn't feel great about this score but might have thought that at least it was better than average. However, as the raw data makes clear, this student really did worse than average[1] on the exam (the professor confesses that the exam was rather easy). In fact, out of the nine students, the one who got an 80 scored third from the bottom of the class. Furthermore, the student who scored worst on the exam did so badly that one might expect this student to drop the course, thereby making the 80 look even worse.

[1] "Average" is usually taken as synonymous with "mean" but in this section we shall use it in an imprecise sense, employing other technically defined terms for specific meanings.

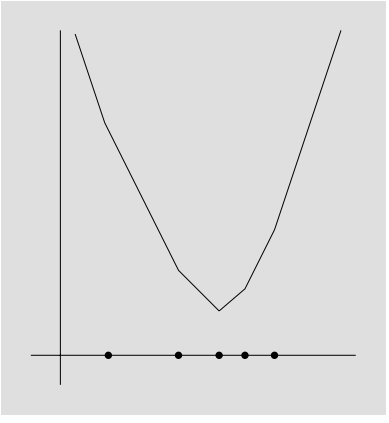


FIGURE 12.1. The objective function whose minimum occurs at the median.

Any statistician would, of course, immediately suggest that we report the median score instead of the mean. The *median* score is, by definition, that score which is worse than half of the other scores and better than the other half. In other words, the median $\hat{x}$ is defined as

$$\hat{x} = b_{(m+1)/2} = 86.$$

(Here and in various places in this chapter, we shall assume that m is odd so that certain formulas such as this one remain fairly simple.) Clearly, the 86 gives a more accurate indication of what the average score on the exam was.

There is a close connection between these statistical concepts and optimization. For example, the mean $\bar{x}$ minimizes, over all real numbers x, the sum of the squared deviations between the data points and x itself. That is,

$$\bar{x} = \text{argmin}_{x \in \mathbb{R}} \sum_{i=1}^{m} (x - b_i)^2.$$

To verify this claim, we let $f(x) = \sum_{i=1}^{m} (x - b_i)^2$, differentiate with respect to x, and set the derivative to zero to get

$$f'(x) = \sum_{i=1}^{m} 2(x - b_i) = 0.$$

Solving this equation for the critical point[2] x, we see that

$$x = \frac{1}{m}\sum_{i=1}^{m} b_i = \bar{x}.$$

The fact that this critical point is a minimum rather than a maximum (or a saddle point) follows from the fact that $f''(x) > 0$ for all $x \in \mathbb{R}$.

The median $\hat{x}$ also enjoys a close connection with optimization. Indeed, it is the point that minimizes the sum of the absolute values of the difference between each data point and itself. That is,

$$\hat{x} = \text{argmin}_{x\in\mathbb{R}} \sum_{i=1}^{m} |x - b_i|.$$

To see that this is correct, we again use calculus. Let

$$f(x) = \sum_{i=1}^{m} |x - b_i|.$$

This function is continuous, piecewise linear, and convex (see Figure 12.1). However, it is not differentiable at the data points. Nonetheless, we can look at its derivative at other points to see where it jumps across zero. The derivative, for $x \notin \{b_1, b_2, \ldots, b_m\}$, is

$$f'(x) = \sum_{i=1}^{m} \text{sgn}(x - b_i),$$

where

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases}$$

Hence, we see that the derivative at x is just the number of data points to the left of x minus the number of data points to the right. Clearly, this derivative jumps across zero at the median, implying that the median is the minimum.

In this chapter, we shall discuss certain generalizations of means and medians called regressions. At the end, we will consider a specific example that is of particular interest to us: the empirical average performance of the simplex method.

2. Multidimensional Measures: Regression Analysis

The analysis of the previous section can be recast as follows. Given a "random" observation b, we assume that it consists of two parts: a fixed, but unknown, part

[2]Recall from calculus that a *critical point* is any point at which the derivative vanishes or fails to exist.

denoted by x and a random fluctuation about this fixed part, which we denote by ϵ. Hence,

$$b = x + \epsilon.$$

Now, if we take several observations and index them as $i = 1, 2, \ldots, m$, the b's and the ϵ's will vary, but x is assumed to be the same for all observations. Therefore, we can summarize the situation by writing

$$b_i = x + \epsilon_i, \qquad i = 1, 2, \ldots, m.$$

We now see that the mean is simply the value of x that minimizes the sum of the squares of the ϵ_i'is. Similarly, the median is the value of x that minimizes the sum of the absolute values of the ϵ_i's.

Sometimes one wishes to do more than merely identify some sort of "average." For example, a medical researcher might collect blood pressure data on thousands of patients with the aim of identifying how blood pressure depends on age, obesity (defined as weight over height), sex, etc. So associated with each observation b of a blood pressure are values of these *control* variables. Let's denote by a_1 the age of a person, a_2 the obesity, a_3 the sex, etc. Let n denote the number of different control variables being considered by the researcher. In (linear) regression analysis, we assume that the *response* b depends linearly on the control variables. Hence, we assume that there are (unknown) numbers x_j, $j = 1, 2, \ldots, n$, such that

$$b = \sum_{j=1}^{n} a_j x_j + \epsilon.$$

This equation is referred to as the *regression model*. Of course, the researcher collects data from thousands of patients, and so the data items, b and the a_j's, must be indexed over these patients. That is,

$$b_i = \sum_{j=1}^{n} a_{ij} x_j + \epsilon_i, \qquad i = 1, 2, \ldots, m.$$

If we let b denote the vector of observations, ϵ the vector of random fluctuations, and A the matrix whose ith row consists of the values of the control variables for the ith patient, then the regression model can be expressed in matrix notation as

$$b = Ax + \epsilon. \tag{12.1}$$

In regression analysis, the goal is to find the vector x that best explains the observations b. Hence, we wish to pick values that minimize, in some sense, the vector ϵ's. Just as for the mean and median, we can consider minimizing either the sum of the squares of the ϵ_i's or the sum of the absolute values of the ϵ_i's. There are even other possibilities. In the next two sections, we will discuss the range of possibilities and then give specifics for the two mentioned above.

3. L^2-Regression

There are several notions of the size of a vector. The most familiar one is the Euclidean length

$$\|y\|_2 = (\sum_i y_i^2)^{1/2}.$$

This notion of length corresponds to our physical notion (at least when the dimension is low, such as 1, 2, or 3). However, one can use any power inside the sum as long as the corresponding root accompanies it on the outside of the sum. For $1 \leq p < \infty$, we get then the so-called *L^p-norm* of a vector y

$$\|y\|_p = (\sum_i y_i^p)^{1/p}.$$

Other than $p = 2$, the second most important case is $p = 1$ (and the third most important case corresponds to the limit as p tends to infinity).

Measuring the size of ϵ in (12.1) using the L^2-norm, we arrive at the *L^2-regression* problem, which is to find $\bar{x}$ that attains the minimum L^2-norm for the difference between b and Ax. Of course, it is entirely equivalent to minimize the square of the L^2-norm, and so we get

$$\bar{x} = \text{argmin}_x \|b - Ax\|_2^2.$$

Just as for the mean, there is an explicit formula for $\bar{x}$. To find it, we again rely on elementary calculus. Indeed, let

$$f(x) = \|b - Ax\|_2^2 = \sum_i \left(b_i - \sum_j a_{ij} x_j \right)^2.$$

In this multidimensional setting, a critical point is defined as a point at which the derivative with respect to every variable vanishes. So if we denote a critical point by $\bar{x}$, we see that it must satisfy the following equations:

$$\frac{\partial f}{\partial x_k}(\bar{x}) = \sum_i 2 \left(b_i - \sum_j a_{ij} \bar{x}_j \right) (-a_{ik}) = 0, \qquad k = 1, 2, \ldots, n.$$

Simplifying these equations, we get

$$\sum_i a_{ik} b_i = \sum_i \sum_j a_{ik} a_{ij} \bar{x}_j, \qquad k = 1, 2, \ldots, n.$$

In matrix notation, these equations can be summarized as follows:

$$A^T b = A^T A \bar{x}.$$

In other words, assuming that $A^T A$ is invertible, we get

$$\bar{x} = (A^T A)^{-1} A^T b. \tag{12.2}$$

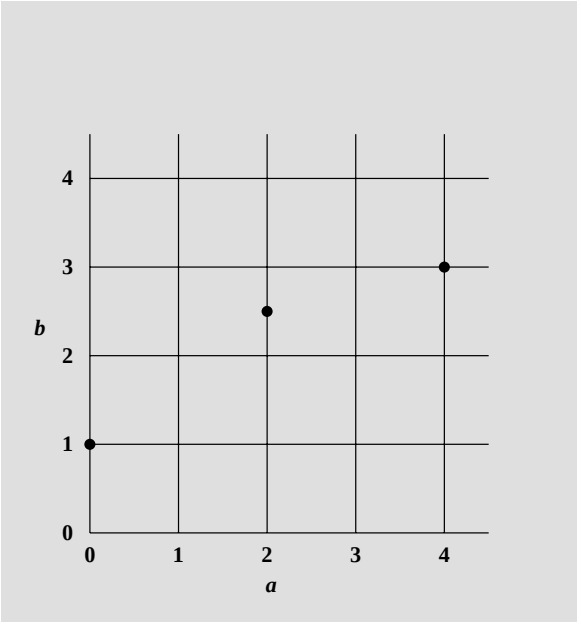


FIGURE 12.2. Three data points for a linear regression.

This is the formula for L^2-regression. It is also commonly called *least squares regression.* In Section 12.6, we will use this formula to solve a specific regression problem.

Example. The simplest and most common regression model arises when one wishes to describe a response variable b as a linear function of a single input variable a. In this case, the model is

$$b = ax_1 + x_2.$$

The unknowns here are the slope x_1 and the intercept x_2. Figure 12.2 shows a plot of three pairs (a, b) through which we want to draw the "best" straight line. At first glance, this model does not seem to fit the regression paradigm, since regression models (as we've defined them) do not involve a term for a nonzero intercept. But the model here can be made to fit by introducing a new control variable, say, a_2, which is always set to 1. While we're at it, let's change our notation for a to a_1 so that the model can now be written as

$$b = a_1x_1 + a_2x_2.$$

The three data points can then be summarized in matrix notation as

$$\begin{bmatrix} 1 \\ 2.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix}.$$

For this problem,

$$A^T A = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 2 & 1 \\ 4 & 1 \end{bmatrix} = \begin{bmatrix} 20 & 6 \\ 6 & 3 \end{bmatrix}$$

and

$$A^T b = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 17 \\ 6.5 \end{bmatrix}.$$

Hence,

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} = \frac{1}{24} \begin{bmatrix} 3 & -6 \\ -6 & 20 \end{bmatrix} \begin{bmatrix} 17 \\ 6.5 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 7/6 \end{bmatrix}.$$

4. L^1-Regression

Just as the median gives a more robust estimate of the "average value" of a collection of numbers than the mean, L^1-regression is less sensitive to outliers than least squares regression is. It is defined by minimizing the L^1-norm of the deviation vector in (12.1). That is, the problem is to find $\hat{x}$ as follows:

$$\hat{x} = \text{argmin}_x \|b - Ax\|_1.$$

Unlike for least squares regression, there is no explicit formula for the solution to the L^1-regression problem. However, the problem can be reformulated as a linear programming problem. Indeed, it is easy to see that the L^1-regression problem,

$$\text{minimize} \sum_i \left| b_i - \sum_j a_{ij} x_j \right|,$$

can be rewritten as

$$\begin{array}{ll} \text{minimize} & \sum_i t_i \\ \text{subject to} & t_i - \left| b_i - \sum_j a_{ij} x_j \right| = 0, \quad i = 1, 2, \ldots, m, \end{array}$$

which is equivalent to the following linear programming problem:

$$\begin{array}{ll} \text{minimize} & \sum_i t_i \\ \text{subject to} & -t_i \le b_i - \sum_j a_{ij}x_j \le t_i, \quad i = 1, 2, \ldots, m. \end{array} \tag{12.3}$$

Hence, to solve the L^1-regression problem, it suffices to solve this linear programming problem. In the next section, we shall present an alternative algorithm for computing the solution to an L^1-regression problem.

Example. Returning to the example of the last section, the L^1-regression problem is solved by finding the optimal solution to the following linear programming problem:

$$\begin{array}{lrcl} \text{minimize} & t_1 + t_2 + t_3 & & \\ \text{subject to} & -x_2 - t_1 & \le & -1 \\ & -2x_1 - x_2 - t_2 & \le & -2.5 \\ & -4x_1 - x_2 - t_3 & \le & -3 \\ & x_2 - t_1 & \le & 1 \\ & 2x_1 + x_2 - t_2 & \le & 2.5 \\ & 4x_1 + x_2 - t_3 & \le & 3 \\ & t_1,\ t_2,\ t_3 & \ge & 0. \end{array}$$

The solution to this linear programming problem is

$$\hat{x} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix},$$

which clearly indicates that the point $(2, 2.5)$ is viewed by the L^1-regression as an outlier, since the regression line passes exactly through the other two points.

5. Iteratively Reweighted Least Squares

Even though calculus cannot be used to obtain an explicit formula for the solution to the L^1-regression problem, it can be used to obtain an iterative procedure that, when properly initialized, converges to the solution of the L^1-regression problem. The resulting iterative process is called *iteratively reweighted least squares*. In this section, we briefly discuss this method. We start by considering the objective function for L^1-regression:

$$\begin{aligned} f(x) &= \|b - Ax\|_1 \\ &= \sum_i \left| b_i - \sum_j a_{ij}x_j \right|. \end{aligned}$$

Differentiating this objective function is a problem, since it involves absolute values. However, the absolute value function

$$g(z) = |z|$$

is differentiable everywhere except at one point: $z = 0$. Furthermore, we can use the following simple formula for the derivative, where it exists:

$$g'(z) = \frac{z}{|z|}.$$

Using this formula to differentiate f with respect to each variable, and setting the derivatives to zero, we get the following equations for critical points:

$$\frac{\partial f}{\partial x_k} = \sum_i \frac{b_i - \sum_j a_{ij}x_j}{|b_i - \sum_j a_{ij}x_j|}(-a_{ik}) = 0, \quad k = 1, 2, \ldots, n. \tag{12.4}$$

If we introduce the following shorthand notation for the deviations,

$$\epsilon_i(x) = \left| b_i - \sum_j a_{ij}x_j \right|,$$

we see that we can rewrite (12.4) as

$$\sum_i \frac{a_{ik}b_i}{\epsilon_i(x)} = \sum_i \sum_j \frac{a_{ik}a_{ij}x_j}{\epsilon_i(x)}, \qquad k = 1, 2, \ldots, n.$$

Now, if we let E_x denote the diagonal matrix containing the vector $\epsilon(x)$ on the diagonal, we can write these equations in matrix notation as follows:

$$A^T E_x^{-1} b = A^T E_x^{-1} A x.$$

This equation can't be solved for x as we were able to do in L^2-regression because of the dependence of the diagonal matrix on x. But let us rearrange this system of equations by multiplying both sides by the inverse of $A^T E_x^{-1} A$. The result is

$$x = \left(A^T E_x^{-1} A\right)^{-1} A^T E_x^{-1} b.$$

This formula suggests an iterative scheme that hopefully converges to a solution. Indeed, we start by initializing x^0 arbitrarily and then use the above formula to successively compute new approximations. If we let x^k denote the approximation at the kth iteration, then the update formula can be expressed as

$$x^{k+1} = \left(A^T E_{x^k}^{-1} A\right)^{-1} A^T E_{x^k}^{-1} b.$$

Assuming only that the matrix inverse exists at every iteration, one can show that this iteration scheme converges to a solution to the L^1-regression problem.

6. An Example: How Fast is the Simplex Method?

In Chapter 4, we discussed the worst-case behavior of the simplex method and studied the Klee–Minty problem that achieves the worst case. We also discussed the importance of empirical studies of algorithm performance. In this section, we shall introduce a model that allows us to summarize the results of these empirical studies.

We wish to relate the number of simplex iterations T required to solve a linear programming problem to the number of constraints m and/or the number of variables n in the problem (or some combination of the two). As any statistician will report, the first step is to introduce an appropriate model.[3] Hence, we begin by asking: *how many iterations, on average, do we expect the simplex method to take if the problem has m constraints and n variables?* To propose an answer to this question, consider the initial dictionary associated with a given problem. This dictionary involves m values, $x^*_{\mathcal{B}}$, for the primal basic variables, and n values, $y^*_{\mathcal{N}}$, for the dual nonbasic variables. We would like each of these $m + n$ variables to have nonnegative values, since that would indicate optimality. If we assume that the initial dictionary is nondegenerate, then one would expect on the average that $(m + n)/2$ of the values would be positive and the remaining $(m + n)/2$ values would be negative.

Now let's look at the dynamics of the simplex method. Each iteration focuses on exactly one of the negative values. Suppose, for the sake of discussion, that the negative value corresponds to a dual nonbasic variable, that is, one of the coefficients in the objective row of the dictionary. Then the simplex method selects the corresponding primal nonbasic variable to enter the basis, and a leaving variable is chosen by a ratio test. After the pivot, the variable that exited now appears as a nonbasic variable in the same position that the entering variable held before. Furthermore, the coefficient on this variable is guaranteed to be positive (since we've assumed nondegeneracy). Hence, the effect of one pivot of the simplex method is to correct the sign of one of the negative values from the list of $m + n$ values of interest. Of course, the pivot also affects all the other values, but there seems no reason to assume that the situation relative to them will have any tendency to get better or worse, on the average. Therefore, we can think of the simplex method as statistically reducing the number of negative values by one at each iteration.

Since we expect on the average that an initial dictionary will have $(m+n)/2$ negative values, it follows that the simplex method should take $(m + n)/2$ iterations, on average. Of course, these expectations are predicated on the assumption that degenerate dictionaries don't arise. As we saw in Section 7.2, the self-dual simplex method initialized with random perturbations will, with probability one, never encounter a degenerate dictionary. Hence, we hypothesize that this variant of the simplex method will, on average, take $(m + n)/2$ iterations. It is important to note the main point of

[3] In the social sciences, a fundamental difficulty is the lack of specific arguments validating the appropriateness of the models commonly introduced.

our hypothesis; namely, that the number of iterations is *linear* in $m + n$ as opposed, say, to quadratic or cubic.

We can test our hypothesis by first supposing that T can be approximated by a function of the form

$$2^{\alpha}(m+n)^{\beta}$$

for a pair of real numbers α and β. Our goal then is to find the value for these parameters that best fits the data obtained from a set of empirical observations. (We've written the leading constant as 2^{α} simply for symmetry with the other factor—there is no fundamental need to do this.) This multiplicative representation of the number of iterations can be converted into an additive (in α and β) representation by taking logarithms. Introducing an ϵ to represent the difference between the model's prediction and the true number of iterations, we see that the model can be written as

$$\log T = \alpha \log 2 + \beta \log(m+n) + \epsilon.$$

Now, suppose that several observations are made. Using subscripts to distinguish the various observations, we get the following equations:

$$\begin{bmatrix} \log T_1 \\ \log T_2 \\ \vdots \\ \log T_k \end{bmatrix} = \begin{bmatrix} \log 2 & \log(m_1+n_1) \\ \log 2 & \log(m_2+n_2) \\ \vdots & \vdots \\ \log 2 & \log(m_k+n_k) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_k \end{bmatrix}.$$

If we let b denote the vector on the left, A the matrix on the right, x the vector multiplied by A, and ϵ the vector of deviations, then the model can be expressed as

$$b = Ax + \epsilon,$$

where A and b are given. As we've seen, this is just a regression model, which we can solve as an L^1-regression or as an L^2-regression.

Given real data, we shall solve this model both ways. Table 12.1 shows specific data obtained by running the self-dual simplex method described in Chapter 7 (with randomized initial perturbations) against most of the problems in a standard suite of test problems (called the NETLIB suite (Gay 1985)). Some problems were too big to run on the workstation used for this experiment, and others were formulated with free variables that the code was not equipped to handle.

Using (12.2) to solve the problem as an L^2-regression, we get

$$\begin{bmatrix} \bar{\alpha} \\ \bar{\beta} \end{bmatrix} = \begin{bmatrix} -1.03561 \\ 1.05152 \end{bmatrix}.$$

Or, in other words,

$$T \approx 0.488(m+n)^{1.052}.$$

Name	m	n	iters	Name	m	n	iters
25fv47	777	1545	5089	nesm	646	2740	5829
80bau3b	2021	9195	10514	recipe	74	136	80
adlittle	53	96	141	sc105	104	103	92
afiro	25	32	16	sc205	203	202	191
agg2	481	301	204	sc50a	49	48	46
agg3	481	301	193	sc50b	48	48	53
bandm	224	379	1139	scagr25	347	499	1336
beaconfd	111	172	113	scagr7	95	139	339
blend	72	83	117	scfxm1	282	439	531
bnl1	564	1113	2580	scfxm2	564	878	1197
bnl2	1874	3134	6381	scfxm3	846	1317	1886
boeing1	298	373	619	scorpion	292	331	411
boeing2	125	143	168	scrs8	447	1131	783
bore3d	138	188	227	scsd1	77	760	172
brandy	123	205	585	scsd6	147	1350	494
czprob	689	2770	2635	scsd8	397	2750	1548
d6cube	403	6183	5883	sctap1	284	480	643
degen2	444	534	1421	sctap2	1033	1880	1037
degen3	1503	1818	6398	sctap3	1408	2480	1339
e226	162	260	598	seba	449	896	766
etamacro	334	542	1580	share1b	107	217	404
fffff800	476	817	1029	share2b	93	79	189
finnis	398	541	680	shell	487	1476	1155
fit1d	24	1026	925	ship04l	317	1915	597
fit1p	627	1677	15284	ship04s	241	1291	560
forplan	133	415	576	ship08l	520	3149	1091
ganges	1121	1493	2716	ship08s	326	1632	897
greenbea	1948	4131	21476	ship12l	687	4224	1654
grow15	300	645	681	ship12s	417	1996	1360
grow22	440	946	999	sierra	1212	2016	793
grow7	140	301	322	standata	301	1038	74
israel	163	142	209	standmps	409	1038	295
kb2	43	41	63	stocfor1	98	100	81
lotfi	134	300	242	stocfor2	2129	2015	2127
maros	680	1062	2998				

TABLE 12.1. Number of iterations for the self-dual simplex method.

This is amazingly close to our hypothesized formula, $(m+n)/2$. Figure 12.3 shows a log–log plot of T vs. $m+n$ with the L^2-regression line drawn through it. It is clear from this graph that a straight line (in the log–log plot) is a good model for fitting this data.

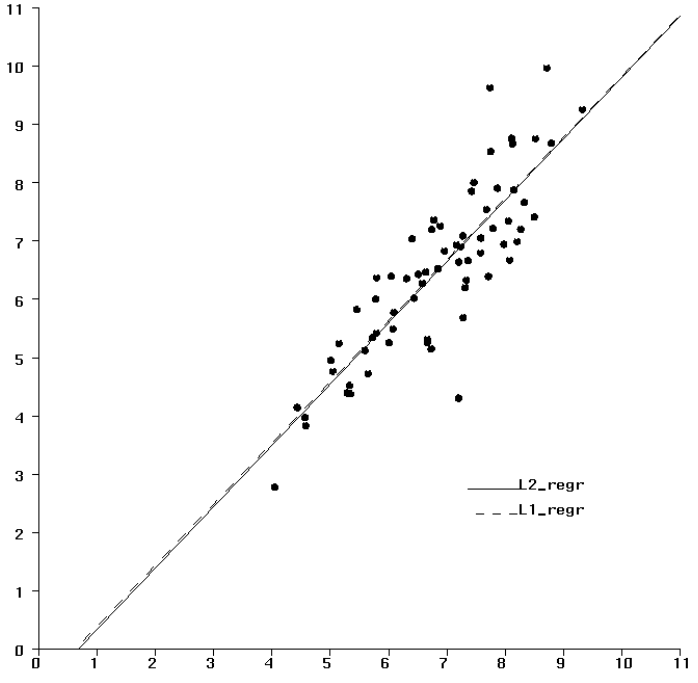


FIGURE 12.3. A log–log plot of T vs. $m+n$ and the L^1 and L^2 regression lines.

Using (12.3) to solving the problem, we get

$$\begin{bmatrix} \bar{\alpha} \\ \bar{\beta} \end{bmatrix} = \begin{bmatrix} -0.9508 \\ 1.0491 \end{bmatrix}.$$

In other words,

$$T \approx 0.517(m+n)^{1.049}.$$

The fact that this regression formula agrees closely with the L^2-regression indicates that the data set contains no outliers. In Section 12.7, we shall see an example in which outliers do indeed cause the L^1 and L^2 regression lines to be significantly different from each other.

7. Which Variant of the Simplex Method is Best?

As we saw in the previous section, if the simplex method does not encounter degenerate dictionaries along its path to optimality, then we can expect that on the average it will converge in $(m+n)/2$ iterations. Also, we've verified this hypothesis running the self-dual simplex method (with randomized initial perturbations) against actual data. The other variants of the simplex method that we have studied do encounter degenerate dictionaries and hence we expect them to take more iterations.

To test this hypothesis, we looked at the two-phase simplex method using a dual Phase I followed by a primal Phase II. The temporary objective function chosen for Phase I was generated randomly, and hence the Phase I portion of the algorithm encountered no degeneracy. However, the Phase II dictionaries were often degenerate using this method. Figure 12.4 shows a log–log plot of iterations vs. $m+n$.

The particular code that was used to produce these numbers has not been carefully tuned, and hence, due to minor numerical problems, the code failed to find an optimal solution for several of the test problems (even though all the problems in the test suite are known to have optimal solutions). For these bad problems, one would expect that the number of iterations to solve the problem is in fact larger (perhaps significantly) than the number obtained. Nonetheless, Figure 12.4 shows all the results, even those corresponding to bad problems. The bad data points are shown with open circles.

The figure shows two regression lines. The lower one is the L^2-regression line. Note that it is pulled down significantly by the single outlier at the bottom of the graph. The upper line is the L^1 regression line. Visually it appears to capture better the trend exhibited by most points. The equation for the L^1-regression is

$$T \approx 0.877(m+n)^{0.994}.$$

Again, the hypothesis that the number of iterations is linear in $m+n$ is strongly supported. But this time the coefficient is obviously not close to $1/2$. Instead, it is almost twice as large. Hence, this method seems to be worse than the self-dual simplex method.

To reiterate, our conclusion is that degeneracy is bad for iteration counts and any version of the simplex method that can avoid it will likely do well. To see this effect in a dramatic setting, let us consider one instance of a large problem that is known to be highly degenerate. The problem we shall consider is called an *assignment problem*. This class of linear programming problem is discussed in detail in Chapter 14. For now, it suffices to say simply that problems in this class are always degenerate. We generated one large ($m = 600$, $n =$ 90,000) random assignment problem and let the two codes described above solve it. The two-phase code[4] took 9951 iterations, whereas the self-dual code took only 1655—a substantial improvement.

[4]Unlike the earlier experiments, here the Phase I objective was initialized, without the benefit of randomization, by setting the jth coefficient to $\max(c_j, 1)$ where c_j is the Phase II objective coefficient.

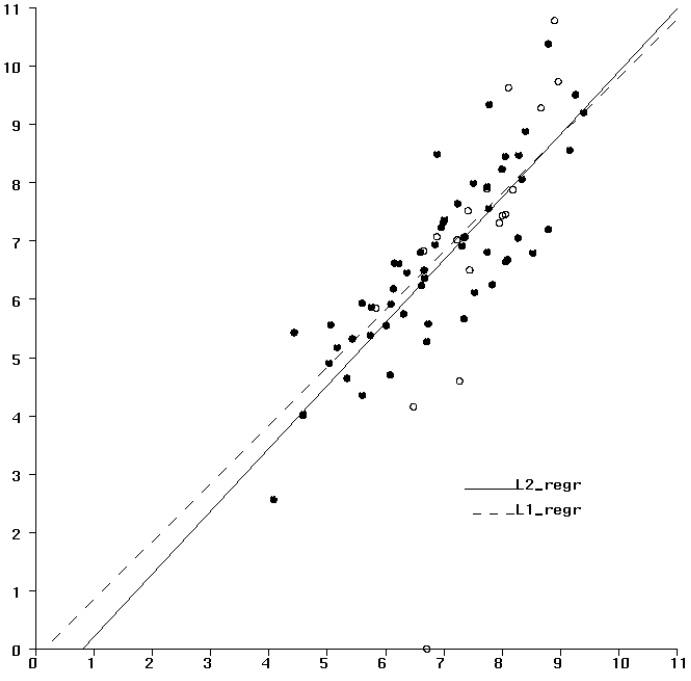


FIGURE 12.4. A log–log plot of T vs. $m + n$ for the two-phase simplex method. The upper line is the L^1-regression line, and the lower line is the L^2-regression line. The open circles show data points for which the code did not find an optimal solution (even though one exists).

Exercises

12.1 Find the L^2-regression line for the data shown in Figure 12.5.

12.2 Find the L^1-regression line for the data shown in Figure 12.5.

12.3 *Midrange.* Given a sorted set of real numbers, $\{b_1, b_2, \ldots, b_m\}$, show that the midrange, $\tilde{x} = (b_1 + b_m)/2$, minimizes the maximum deviation from the set of observations. That is,

$$\frac{1}{2}(b_1 + b_m) = \text{argmin}_{x \in \mathbb{R}} \max_i |x - b_i|.$$

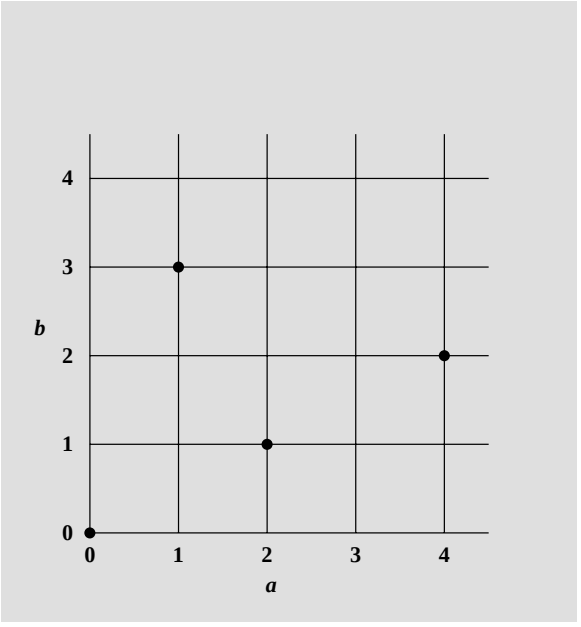


FIGURE 12.5. Four data points for a linear regression.

12.4 *Centroid.* Given a set of points $\{b_1, b_2, \ldots, b_m\}$ on the plane $\mathbb{R}^2$, show that the centroid

$$\bar{x} = \frac{1}{m} \sum_{i=1}^{m} b_i$$

minimizes the sum of the squares of the distance to each point in the set. That is, $\bar{x}$ solves the following optimization problem:

$$\text{minimize} \sum_{i=1}^{m} \|x - b_i\|_2^2$$

Note: Each data point b_i is a vector in $\mathbb{R}^2$ whose components are denoted, say, by b_{i1} and b_{i2}, and, as usual, the subscript 2 on the norm denotes the Euclidean norm. Hence,

$$\|x - b_i\|_2 = \sqrt{(x_1 - b_{i1})^2 + (x_2 - b_{i2})^2}.$$

12.5 *Facility Location.* A common problem is to determine where to locate a facility so that the distance from its customers is minimized. That is, given a set of points $\{b_1, b_2, \ldots, b_m\}$ on the plane $\mathbb{R}^2$, the problem is to find $\hat{x} =$

Jan	390	May	310	Sep	550
Feb	420	Jun	590	Oct	360
Mar	340	Jul	340	Nov	420
Apr	320	Aug	580	Dec	600 .

TABLE 12.2. Projected labor hours by month.

$(\hat{x}_1, \hat{x}_2)$ that solves the following optimization problem:

$$\text{minimize} \sum_{i=1}^{m} \|x - b_i\|_2.$$

As for L^1-regression, there is no explicit formula for $\hat{x}$, but an iterative scheme can be derived along the same lines as in Section 12.5. Derive an explicit formula for this iteration scheme.

12.6 *A Simple Steiner Tree.* Suppose there are only three customers in the facility location problem of the previous exercise. Suppose that the triangle formed by b_1, b_2, and b_3 has no angles greater than 120 degrees. Show that the solution $\hat{x}$ to the facility location problem is the unique point in the triangle from whose perspective the three customers are 120 degrees apart from each other. What is the solution if one of the angles, say, at vertex b_1, is more than 120 degrees?

12.7 *Sales Force Planning.* A distributor of office equipment finds that the business has seasonal peaks and valleys. The company uses two types of sales persons: (a) regular employees who are employed year-round and cost the company \$17.50/hr (fully loaded for benefits and taxes) and (b) temporary employees supplied by an outside agency at a cost of \$25/hr. Projections for the number of hours of labor by month for the following year are shown in Table 12.2. Let a_i denote the number of hours of labor needed for month i and let x denote the number of hours per month of labor that will be handled by regular employees. To minimize total labor costs, one needs to solve the following optimization problem:

$$\text{minimize} \sum_{i} (25 \max(a_i - x, 0) + 17.50x).$$

(a) Show how to reformulate this problem as a linear programming problem.
(b) Solve the problem for the specific data given above.
(c) Use calculus to find a formula giving the optimal value for x.

12.8 *Acceleration Due to Gravity.* The law of gravity from classical physics says that an object dropped from a tall building will, in the absence of air resistance, have a constant rate of acceleration g so that the height x, as a function of time t, is given by

$$x(t) = -\frac{1}{2}gt^2.$$

Unfortunately, the effects of air resistance cannot be ignored. To include them, we assume that the object experiences a retarding force that is directly proportional to its speed. Letting $v(t)$ denote the velocity of the object at time t, the equations that describe the motion are then given by

$$\begin{aligned} x'(t) &= v(t), & t > 0, \qquad & x(0) = 0, \\ v'(t) &= -g - fv(t), & t > 0, \qquad & v(0) = 0 \end{aligned}$$

(f is the unknown constant of proportionality from the air resistance). These equations can be solved explicitly for x as a function of t:

$$\begin{aligned} x(t) &= -\frac{g}{f^2}\left(e^{-ft} - 1 + ft\right) \\ v(t) &= -\frac{g}{f}\left(1 - e^{-ft}\right). \end{aligned}$$

It is clear from the equation for the velocity that the *terminal velocity* is g/f. It would be nice to be able to compute g by measuring this velocity, but this is not possible, since the terminal velocity involves both f and g. However, we can use the formula for $x(t)$ to get a two-parameter model from which we can compute both f and g. Indeed, if we assume that all measurements are taken after terminal velocity has been "reached" (i.e., when e^{-ft} is much smaller than 1), then we can write a simple linear expression relating position to time:

$$x = \frac{g}{f^2} - \frac{g}{f}t.$$

Now, in our experiments we shall set values of x (corresponding to specific positions below the drop point) and measure the time at which the object passes these positions. Since we prefer to write regression models with the observed variable expressed as a linear function of the control variables, let us rearrange the above expression so that t appears as a function of x:

$$t = \frac{1}{f} - \frac{f}{g}x.$$

Using this regression model and the data shown in Table 12.3, do an L^2-regression to compute estimates for $1/f$ and $-f/g$. From these estimates derive an estimate for g. If you have access to linear programming software, solve the problem using an L^1-regression and compare your answers.

Obs. Number	Position (meters)	Time (secs)
1	-10	3.72
2	-20	7.06
3	-30	10.46
4	-10	3.71
5	-20	7.00
6	-30	10.48
7	-10	3.67
8	-20	7.08
9	-30	10.33

TABLE 12.3. Time at which a falling object passes certain points.

12.9 *Iteratively Reweighted Least Squares.* Show that the sequence of iterates in the iteratively reweighted least squares algorithm produces a monotonically decreasing sequence of objective function values by filling in the details in the following outline. First, recall that the objective function for L^1-regression is given by

$$f(x) = \|b - Ax\|_1 = \sum_{i=1}^{m} \epsilon_i(x),$$

where

$$\epsilon_i(x) = \left| b_i - \sum_{j=1}^{n} a_{ij} x_j \right|.$$

Also, the function that defines the iterative scheme is given by

$$T(x) = \left(A^T E_x^{-1} A\right)^{-1} A^T E_x^{-1} b,$$

where E_x denotes the diagonal matrix with the vector $\epsilon(x)$ on its diagonal. Our aim is to show that

$$f(T(x)) < f(x).$$

In order to prove this inequality, let

$$g_x(z) = \sum_{i=1}^{m} \frac{\epsilon_i^2(z)}{\epsilon_i(x)} = \|E_x^{-1/2}(b - Az)\|_2^2.$$

(a) Use calculus to show that, for each x, $T(x)$ is a global minimum of g_x.
(b) Show that $g_x(x) = f(x)$.
(c) By writing

$$\epsilon_i(T(x)) = \epsilon_i(x) + (\epsilon_i(T(x)) - \epsilon_i(x))$$

and then substituting the right-hand expression into the definition of $g_x(T(x))$, show that

$$g_x(T(x)) \geq 2f(T(x)) - f(x).$$

(d) Combine the three steps above to finish the proof.

12.10 In our study of means and medians, we showed that the median of a collection of numbers, $b_1, b_2, \ldots, b_n$, is the number $\hat{x}$ that minimizes $\sum_j |x - b_j|$. Let μ be a real parameter.

(a) Give a statistical interpretation to the following optimization problem:

$$\text{minimize} \sum_j \left(|x - b_j| + \mu(x - b_j)\right).$$

Hint: the special cases $\mu = 0, \pm 1/2, \pm 1$ might help clarify the general situation.

(b) Express the above problem as a linear programming problem.
(c) The parametric simplex method can be used to solve families of linear programming problems indexed by a parameter μ (such as we have here). Starting at $\mu = \infty$ and proceeding to $\mu = -\infty$ one solves all of the linear programs with just a finite number of pivots. Use the parametric simplex method to solve the problems of the previous part in the case where $n = 4$ and $b_1 = 1$, $b_2 = 2$, $b_3 = 4$, and $b_4 = 8$.
(d) Now consider the general case. Write down the dictionary that appears in the k-th iteration and show by induction that it is correct.

12.11 Show that the L^∞-norm is just the maximum of the absolute values. That is,

$$\lim_{p \to \infty} \|x\|_p = \max_i |x_i|.$$

Notes

Gonin & Money (1989) and Dodge (1987) are two references on regression that include discussion of both L^2 and L^1 regression. The standard reference on L^1 regression is Bloomfield & Steiger (1983).

Several researchers, including Smale (1983), Borgwardt (1982), Borgwardt (1987*a*), Adler & Megiddo (1985), and Todd (1986), have studied the average number of iterations of the simplex method as a function of m and/or n. The model discussed in

this chapter is similar to the sign-invariant model introduced by Adler & Berenguer (1981).

Part 2

Network-Type Problems

Allow me to say, . . . , that the arguments with which you have supported this extraordinary application have been as frivolous as the application was ill-judged. — J. Austen

CHAPTER 13

Network Flow Problems

Many linear programming problems can be viewed as a problem of minimizing the "transportation" cost of moving materials through a network to meet demands for material at various locations given sources of material at other locations. Such problems are called *network flow problems*. They form the most important special class of linear programming problems. Transportation, electric, and communication networks provide obvious examples of application areas. Less obvious, but just as important, are applications in facilities location, resource management, financial planning, and others.

In this chapter we shall formulate a special type of linear programming problem called the *minimum-cost network flow problem*. It turns out that the simplex method when applied to this problem has a very simple description and some important special properties. Implementations that exploit these properties benefit dramatically.

1. Networks

A network consists of two types of objects: nodes and arcs. We shall let $\mathcal{N}$ denote the set of *nodes*. We let m denote the number of nodes (i.e., the cardinality of the set $\mathcal{N}$).

The nodes are connected by *arcs*. Arcs are assumed to be directed. This means that an arc connecting node i to node j is not the same as an arc connecting node j to node i. For this reason, we denote arcs using the standard mathematical notation for ordered pairs. That is, the arc connecting node i to node j is denoted simply as (i, j). We let $\mathcal{A}$ denote the set of all arcs in the network. This set is a subset of the set of all possible arcs:

$$\mathcal{A} \subset \{(i, j) : i, j \in \mathcal{N}, i \neq j\}.$$

In typical networks, the set $\mathcal{A}$ is much smaller than the set of all arcs. In fact, usually each node is only connected to a handful of "nearby" nodes.

The pair $(\mathcal{N}, \mathcal{A})$ is called a *network*. It is also sometimes called a *graph* or a *digraph* (to emphasize the fact that the arcs are directed). Figure 13.1 shows a network having 7 nodes and 14 arcs.

To specify a network flow problem, we need to indicate the supply of (or demand for) material at each node. So, for each $i \in \mathcal{N}$, let b_i denote the amount of material being supplied to the network at node i. We shall use the convention that negative

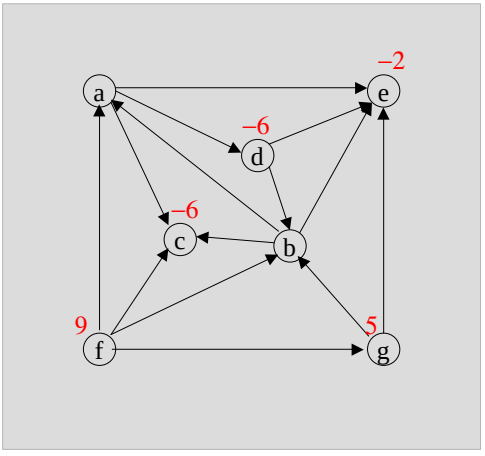


FIGURE 13.1. A network having 7 nodes and 14 arcs. The numbers written next to the nodes denote the supply at the node (negative values indicate demands; missing values indicate no supply or demand).

supplies are in fact demands. Hence, our problem will be to move the material that sits at the supply nodes over to the demand nodes. The movements must be along the arcs of the network (and adhering to the directions of the arcs). Since, except for the supply and demand, there is no other way for material to enter or leave the system, it follows that the total supply must equal the total demand for the problem to have a feasible solution. Hence, we shall always assume that

$$\sum_{i \in \mathcal{N}} b_i = 0.$$

To help us decide the paths along which materials should move, we assume that each arc, say, (i, j), has associated with it a cost c_{ij} that represents the cost of shipping one unit from i to j directly along arc (i, j). The decision variables then are how much material to ship along each arc. That is, for each $(i, j) \in \mathcal{A}$, x_{ij} will denote the quantity shipped directly from i to j along arc (i, j). The objective is to minimize the total cost of moving the supply to meet the demand:

$$\text{minimize} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}.$$

As we mentioned before, the constraints on the decision variables are that they ensure flow balance at each node. Let us consider a fixed node, say, $k \in \mathcal{N}$. The total

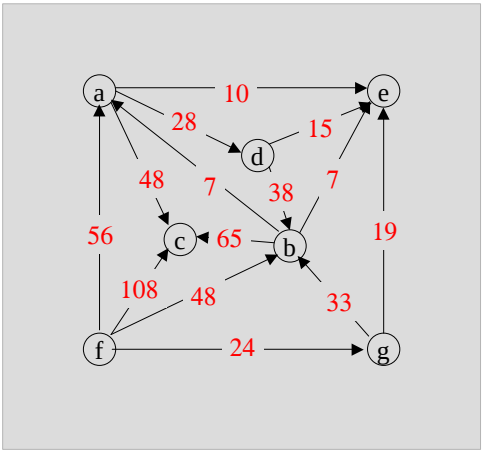


FIGURE 13.2. The costs on the arcs for the network in Figure 13.1.

flow into node k is given by

$$\sum_{\substack{i: \\ (i,k)\in\mathcal{A}}} x_{ik}.$$

Similarly, the total flow out from node k is

$$\sum_{\substack{j: \\ (k,j)\in\mathcal{A}}} x_{kj}.$$

The difference between these two quantities is the net inflow, which must be equal to the demand at the node. Hence, the *flow balance constraints* can be written as

$$\sum_{\substack{i: \\ (i,k)\in\mathcal{A}}} x_{ik} - \sum_{\substack{j: \\ (k,j)\in\mathcal{A}}} x_{kj} = -b_k, \qquad k \in \mathcal{N}.$$

Finally, the flow on each arc must be nonnegative (otherwise it would be going in the wrong direction):

$$x_{ij} \geq 0, \qquad (i,j) \in \mathcal{A}.$$

Figure 13.2 shows cost information for the network shown in Figure 13.1. In matrix notation, the problem can be written as follows:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax = -b \\ & && x \geq 0, \end{aligned} \tag{13.1}$$

where

$$x^T = \begin{bmatrix} x_{ac} & x_{ad} & x_{ae} & x_{ba} & x_{bc} & x_{be} & x_{db} & x_{de} & x_{fa} & x_{fb} & x_{fc} & x_{fg} & x_{gb} & x_{ge} \end{bmatrix},$$

$$A = \begin{bmatrix} -1 & -1 & -1 & 1 & & & & & 1 & & & & & \\ & & & -1 & -1 & -1 & 1 & & & 1 & & & 1 & \\ 1 & & & & 1 & & & & & & 1 & & & \\ & 1 & & & & & -1 & -1 & & & & & & \\ & & 1 & & & 1 & & 1 & & & & & & 1 \\ & & & & & & & & -1 & -1 & -1 & -1 & & \\ & & & & & & & & & & & 1 & -1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ -6 \\ -6 \\ -2 \\ 9 \\ 5 \end{bmatrix},$$

$$c^T = \begin{bmatrix} 48 & 28 & 10 & 7 & 65 & 7 & 38 & 15 & 56 & 48 & 108 & 24 & 33 & 19 \end{bmatrix}.$$

In network flow problems, the constraint matrix A is called the *node–arc incidence matrix*.

The network flow problem differs from our usual standard form linear programming problem in two respects: (1) it is a minimization instead of a maximization and (2) the constraints are equalities instead of inequalities. Nonetheless, we have studied before how duality applies to problems in nonstandard form. The dual of (13.1) is

$$\begin{aligned} \text{maximize} \quad & -b^T y \\ \text{subject to} \quad & A^T y + z = c \\ & z \geq 0. \end{aligned}$$

Written in network notation, the dual is

$$\begin{aligned} \text{maximize} \quad & -\sum_{i \in \mathcal{N}} b_i y_i \\ \text{subject to} \quad & y_j - y_i + z_{ij} = c_{ij}, \qquad (i,j) \in \mathcal{A} \\ & z_{ij} \geq 0, \qquad (i,j) \in \mathcal{A}. \end{aligned}$$

Finally, it is not hard to check that the complementarity conditions (to be satisfied by an optimal primal–dual solution pair) are

$$x_{ij} z_{ij} = 0, \qquad (i,j) \in \mathcal{A}.$$

We shall often refer to the primal variables as *primal flows*.

2. Spanning Trees and Bases

Network flow problems can be solved efficiently because the basis matrices have a special structure that can be described nicely in terms of the network. In order to explain this structure, we need to introduce a number of definitions.

First of all, an ordered list of nodes $(n_1, n_2, \ldots, n_k)$ is called a *path* in the network if each adjacent pair of nodes in the list is connected by an arc in the network. It is

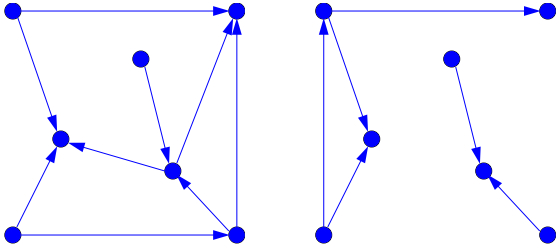


FIGURE 13.3. The network on the left is connected whereas the one on the right is not.

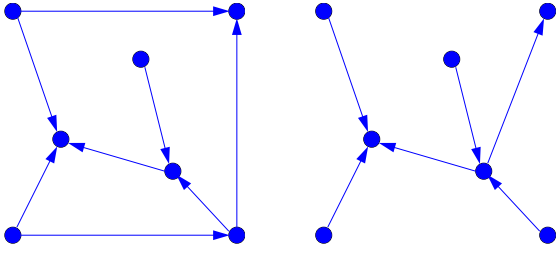


FIGURE 13.4. The network on the left contains a cycle whereas the one on the right is acyclic.

important to note that we do not assume that the arcs point in any particular direction. For example, for nodes n_i and n_{i+1}, there must be an arc in the network. It could run either from n_i to n_{i+1} or from n_{i+1} to n_i. (One should think about one-way roads—even though cars can only go one way, pedestrians are allowed to walk along the path of the road in either direction.) A network is called *connected* if there is a path connecting every pair of nodes (see Figure 13.3). For the remainder of this chapter, we make the following assumption:

Assumption. The network is connected.

For any arc (i, j), we refer to i as its *tail* and j as its *head*.

A *cycle* is a path in which the last node coincides with the first node. A network is called *acyclic* if it does not contain any cycles (see Figure 13.4).

A network is a *tree* if it is connected and acyclic (see Figure 13.5). A network $(\tilde{\mathcal{N}}, \tilde{\mathcal{A}})$ is called a *subnetwork* of $(\mathcal{N}, \mathcal{A})$ if $\tilde{\mathcal{N}} \subset \mathcal{N}$ and $\tilde{\mathcal{A}} \subset \mathcal{A}$. A subnetwork $(\tilde{\mathcal{N}}, \tilde{\mathcal{A}})$ is a *spanning tree* if it is a tree and $\tilde{\mathcal{N}} = \mathcal{N}$. Since a spanning tree's node set coincides with the node set of the underlying network, it suffices to refer to a spanning tree by simply giving its arc set.

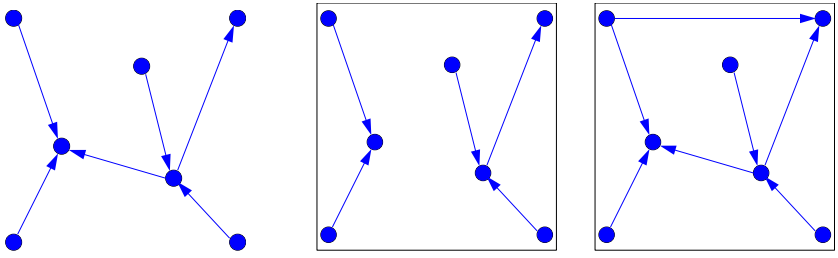


FIGURE 13.5. The network on the left is a tree whereas the two on the right not—they fail in the first case by being disconnected and in the second by containing a cycle.

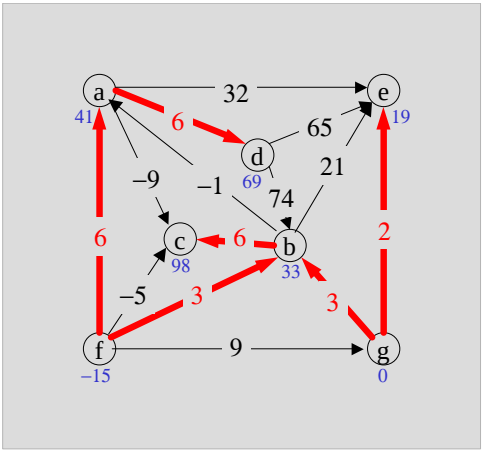


FIGURE 13.6. The fat arcs show a spanning tree for the network in Figure 13.1. The numbers shown on the arcs of the spanning tree are the primal flows, the numbers shown next to the nodes are the dual variables, and the numbers shown on the arcs not belonging to the spanning tree are the dual slacks.

Given a network flow problem, any selection of primal flow values that satisfies the balance equations at every node will be called a *balanced flow*. It is important to note that we do not require the flows to be nonnegative to be a balanced flow. That is, we allow flows to go in the wrong direction. If all the flows are nonnegative, then a balanced flow is called a *feasible flow*. Given a spanning tree, a balanced flow that assigns zero flow to every arc not on the spanning tree will be called a *tree solution*. Consider, for example, the tree shown in Figure 13.6. The numbers shown on the

arcs of the spanning tree give the tree solution corresponding to the supplies/demands shown in Figure 13.1. They were obtained by starting at the "leaves" of the tree and working "inward." For instance, the flows could be solved for successively as follows:

$$\begin{aligned}
&\text{flow bal at d:} && x_{\text{ad}} = 6, && \\
&\text{flow bal at a:} && x_{\text{fa}} - x_{\text{ad}} = 0 && \Longrightarrow \quad x_{\text{fa}} = 6, \\
&\text{flow bal at f:} && -x_{\text{fa}} - x_{\text{fb}} = -9 && \Longrightarrow \quad x_{\text{fb}} = 3, \\
&\text{flow bal at c:} && x_{\text{bc}} = 6, && \\
&\text{flow bal at b:} && x_{\text{fb}} + x_{\text{gb}} - x_{\text{bc}} = 0 && \Longrightarrow \quad x_{\text{gb}} = 3, \\
&\text{flow bal at e:} && x_{\text{ge}} = 2. &&
\end{aligned}$$

It is easy to see that this process always works. The reason is that every tree must have at least one leaf node, and deleting a leaf node together with the edge leading into it produces a subtree.

The above computation suggests that spanning trees are related to bases in the simplex method. Let us pursue this idea. Normally, a basis is an invertible square submatrix of the constraint matrix. But for incidence matrices, no such submatrix exists. To see why, note that if we sum together all the rows of A, we get a row vector of all zeros (since each column of A has exactly one $+1$ and one -1). Of course, every square submatrix of A has this same property and so is singular. In fact, we shall show in a moment that for a connected network, there is exactly one redundant equation (i.e., the rank of A is exactly $m - 1$).

Let us select some node, say, the last one, and delete the flow-balance constraint associated with this node from the constraints defining the problem (since it is redundant anyway). Let's call this node the *root node*. Let $\tilde{A}$ denote the incidence matrix A without the row corresponding to the root node (i.e., the last row), and let $\tilde{b}$ denote the supply/demand vector with the last entry deleted. The most important property of network flow problems is summarized in the following theorem:

THEOREM 13.1. *A square submatrix of $\tilde{A}$ is a basis if and only if the arcs to which its columns correspond form a spanning tree.*

Rather than presenting a formal proof of this theorem, it is more instructive to explain the idea using the example we've been studying. Therefore, consider the spanning tree shown in Figure 13.6, and let B denote the square submatrix of $\tilde{A}$ corresponding to this tree. The matrix B is invertible if and only if every system of equations of the form

$$Bu = \beta$$

has a unique solution. This is exactly the type of equation that we already solved to find the tree solution associated with the spanning tree:

$$Bx_{\mathcal{B}} = -\tilde{b}.$$

We solved this system of equations by looking at the spanning tree and realizing that we could work our way to a solution by starting with the leaves and working inward. This process amounts to a permutation of the rows and columns of B to get a lower triangular matrix. Indeed, for the calculations given above, we have permuted the rows by P and the columns by Q to get

$$PBQ^T = \begin{array}{c} \\ \text{d} \\ \text{a} \\ \text{f} \\ \text{c} \\ \text{b} \\ \text{e} \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{(a,d)} & \text{(f,a)} & \text{(f,b)} & \text{(b,c)} & \text{(g,b)} & \text{(g,e)} \end{array} \\ \begin{bmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ & -1 & -1 & & & \\ & & & 1 & & \\ & & 1 & -1 & 1 & \\ & & & & & 1 \end{bmatrix} \end{array}.$$

The fact that B is invertible is now immediately apparent from the fact that the permuted matrix is lower triangular. In fact, it has only $+1$'s and -1's on the diagonal. Therefore, we can solve systems of equations involving B without ever having to do any divisions. Also, since the off-diagonal entries are also ± 1's, it follows that we don't need to do any multiplications either. Every system of equations involving the matrix B can be solved by a simple sequence of additions and subtractions.

We have shown that, given a spanning tree, the submatrix of $\tilde{A}$ consisting of the columns corresponding to the arcs in the spanning tree is a basis. The converse direction (which is less important to our analysis) is relegated to an exercise (see Exercise 13.12).

Not only is there a primal solution associated with any basis but also there is a dual solution. Hence, corresponding to any spanning tree there is a dual solution. The dual solution consists of two types of variables: the y_i's and the z_{ij}'s. These variables must satisfy the dual feasibility conditions:

$$y_j - y_i + z_{ij} = c_{ij}, \qquad (i,j) \in \mathcal{A}.$$

By complementarity, $z_{ij} = 0$ for each (i,j) in the spanning tree $\mathcal{T}$. Hence,

$$y_j - y_i = c_{ij}, \qquad (i,j) \in \mathcal{T}.$$

Since a spanning tree on m nodes has $m-1$ arcs (why?), these equations define a system of $m-1$ equations in m unknowns. But don't forget that there was a redundant equation in the primal problem, which we associated with a specific node called the root node. Removing that equation and then looking at the dual, we see that there is not really a dual variable associated with the root node. Or equivalently, we can just say that the dual variable for the root node is zero. Making this assignment, we get m equations in m unknowns. These equations can be solved by starting at the root node and working down the tree.

For example, let node "g" be the root node in the spanning tree in Figure 13.6. Starting with it, we compute the dual variables as follows:

$$
\begin{aligned}
& & y_{\mathrm{g}} &= 0, & & \\
\text{across arc (g,e):} & & y_{\mathrm{e}} - y_{\mathrm{g}} &= 19 & \Longrightarrow \quad & y_{\mathrm{e}} = 19, \\
\text{across arc (g,b):} & & y_{\mathrm{b}} - y_{\mathrm{g}} &= 33 & \Longrightarrow \quad & y_{\mathrm{b}} = 33, \\
\text{across arc (b,c):} & & y_{\mathrm{c}} - y_{\mathrm{b}} &= 65 & \Longrightarrow \quad & y_{\mathrm{c}} = 98, \\
\text{across arc (f,b):} & & y_{\mathrm{b}} - y_{\mathrm{f}} &= 48 & \Longrightarrow \quad & y_{\mathrm{f}} = -15, \\
\text{across arc (f,a):} & & y_{\mathrm{a}} - y_{\mathrm{f}} &= 56 & \Longrightarrow \quad & y_{\mathrm{a}} = 41, \\
\text{across arc (a,d):} & & y_{\mathrm{d}} - y_{\mathrm{a}} &= 28 & \Longrightarrow \quad & y_{\mathrm{d}} = 69.
\end{aligned}
$$

Now that we know the dual variables, the dual slacks for the arcs not in the spanning tree $\mathcal{T}$ can be computed using

$$z_{ij} = y_i + c_{ij} - y_j, \qquad (i,j) \notin \mathcal{T}$$

(which is just the dual feasibility condition solved for z_{ij}). These values are shown on the nontree arcs in Figure 13.6.

From duality theory, we know that the current tree solution is optimal if all the flows are nonnegative and if all the dual slacks are nonnegative. The tree solution shown in Figure 13.6 satisfies the first condition but not the second. That is, it is primal feasible but not dual feasible. Hence, we can apply the primal simplex method to move from this solution to an optimal one. We take up this task in the next section.

3. The Primal Network Simplex Method

Each of the variants of the simplex method presented in earlier chapters of this book can be applied to network flow problems. It would be overkill to describe them all here in the context of networks. However, they are all built on two simple algorithms: the primal simplex method (for problems that are primal feasible) and the dual simplex method (for problems that are dual feasible). We discuss them both in detail.

We shall describe the primal network simplex method by continuing with our example. As mentioned above, the tree shown in Figure 13.6 is primal feasible but not dual feasible. The basic idea that defines the primal simplex method is to pick a nontree arc that is dual infeasible and let it enter the tree (i.e., become basic) and then readjust everything so that we still have a tree solution.

The First Iteration. For our first pivot, we let arc (a,c) enter the tree using a *primal pivot*. In a primal pivot, we add flow to the entering variable, keeping all other nontree flows set to zero and adjusting the tree flows appropriately to maintain flow balance. Given any spanning tree, adding an extra arc must create a cycle (why?). Hence, the current spanning tree together with the entering arc must contain a cycle. The flows on the cycle must change to accommodate the increasing flow on the entering arc. The flows on the other tree arcs remain unchanged. In our example, the cycle is: "a", "c",

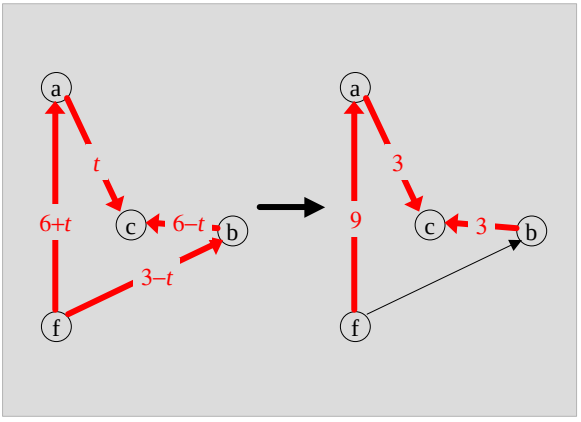


FIGURE 13.7. The cycle produced by including the entering arc with the spanning tree. As the flow t on the entering arc increases, eventually the flow on arc (f,b) becomes zero (when $t = 3$). Hence, arc (f,b) is the leaving arc.

"b", "f". This cycle is shown in Figure 13.7 with flows adjusted to take into account a flow of t on the entering arc. As t increases, eventually the flow on arc (f,b) decreases to zero. Hence, arc (f,b) is the leaving arc. Updating the flows is easy; just take $t = 3$ and adjust the flows appropriately.

With a little thought, one realizes that the selection rule for the leaving arc in a primal pivot is as follows:

Leaving arc selection rule:

- the leaving arc must be oriented along the cycle in the reverse direction from the entering arc, and
- among all such arcs, it must have the smallest flow.

Also, the flows on the cycle get updated as follows:

Primal flows update:

- Flows oriented in the same direction as the leaving arc are decreased by the amount of flow that was on the leaving arc whereas flows in the opposite direction are increased by this amount.

The next issue is how to update the dual variables. To this end, note that if we delete the leaving arc from the spanning tree (without concurrently adding the entering arc), we disconnect it into two disjoint trees. In our example, one tree contains nodes "a", "d" and "f" while the second tree contains the other nodes. Figure 13.8 shows the two disjoint trees. Recalling that the dual variables are calculated starting with the root

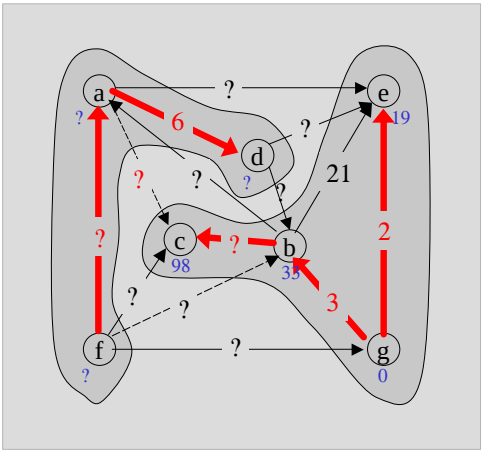


FIGURE 13.8. The two disjoint trees. Primal and dual values that remained unchanged are shown, whereas those that need to be updated are shown as question marks.

node and working up the spanning tree, it is clear that the dual variables on the subtree containing the root node remain unchanged, whereas those on the other subtree must change. For the current pivot, the other subtree consists of nodes "a", "d", and "f". They all get incremented by the same fixed amount, since the only change is that the arc by which we bridged from the root-containing tree to this other tree has changed from the leaving arc to the entering arc. Looking at node "a" and using tildes to denote values after being changed, we see that

$$\begin{aligned}\tilde{y}_{\mathrm{a}} &= \tilde{y}_{\mathrm{c}} - c_{\mathrm{ac}} \\ &= y_{\mathrm{c}} - c_{\mathrm{ac}},\end{aligned}$$

whereas

$$z_{\mathrm{ac}} = y_{\mathrm{a}} + c_{\mathrm{ac}} - y_{\mathrm{c}}.$$

Combining these two equations, we get

$$\tilde{y}_{\mathrm{a}} = y_{\mathrm{a}} - z_{\mathrm{ac}}.$$

That is, the dual variable at node "a" gets decremented by $z_{\mathrm{ac}} = -9$. Of course, all of the dual variables on this subtree get decremented by this same amount. In general, the dual variable update rule can be stated as follows:

Dual variables update:

- If the entering arc crosses from the root-containing tree to the non-root-containing tree, then increase all dual variables on

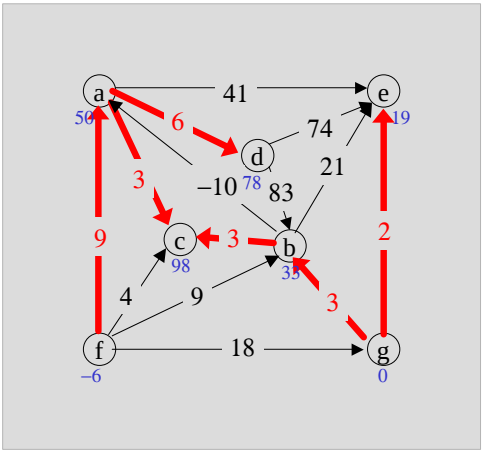


FIGURE 13.9. The tree solution at the end of the first iteration.

the non-root-containing tree by the dual slack of the entering arc.

- Otherwise, decrease these dual variables by this amount.

Finally, we must update the dual slacks. The only dual slacks that change are those that span across the two trees since, for these nodes, either the head or the tail dual variable changes, while the other does not. Those that span the two subtrees in the same direction as the entering arc must be decreased by z_{ac}, whereas those that bridge the two trees in the opposite direction must be increased by this amount. For our example, six nontree arcs, (f,g), (f,b), (f,c), (d,b), (d,e), and (a,e), span in the same direction as the entering arc. They all must be decreased by -9. That is, they must be increased by 9. For example, the dual slack on arc (f,c) changes from -5 to 4. Only one arc, (b,a), spans in the other direction. It must be decreased by 9. The updated solution is shown in Figure 13.9. The general rule for updating the dual slacks is as follows:

Dual slacks update:

- The dual slacks corresponding to those arcs that bridge in the same direction as the entering arc get decremented by the old dual slack on the entering arc, whereas those that correspond to arcs bridging in the opposite direction get incremented by this amount.

The Second Iteration. The tree solution shown in Figure 13.9 has only one remaining infeasibility: $z_{ba} = -10$. Arc (b,a) must therefore enter the spanning tree. Adding it, we create a cycle consisting of nodes “a”, “b”, and “c”. The leaving arc

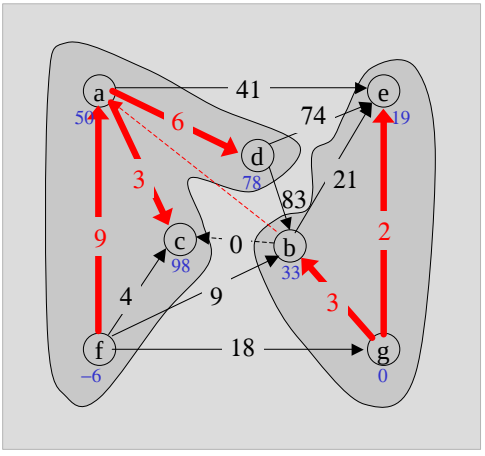


FIGURE 13.10. The two disjoint subtrees arising in the second iteration.

must be pointing in the opposite direction from the entering arc. Here, there is only one such arc, (b,c). It must be the leaving arc. The leaving arc's flow decreases from 3 to 0. The flow on the other two cycle arcs must increase by 3 to preserve flow balance.

The two subtrees formed by removing the leaving arc are shown in Figure 13.10 The dual variables on the non-root-containing subtree get incremented by the dual slack on the entering arc $z_{ba} = -10$. The dual slacks for the spanning arcs also change by 10 either up or down depending on which way they bridge the two subtrees. The resulting tree solution is shown in Figure 13.11.

The Third and Final Iteration. The tree solution shown in Figure 13.11 has one infeasibility: $z_{fb} = -1$. Hence, arc (f,c) must enter the spanning tree. The leaving arc must be (f,a). Leaving the details of updating to the reader, the resulting tree solution is shown in Figure 13.12. It is both primal and dual feasible—hence optimal.

4. The Dual Network Simplex Method

In the previous section, we developed simple rules for the primal network simplex method, which is used in situations where the tree solution is primal feasible but not dual feasible. When a tree solution is dual feasible but not primal feasible, then the dual network simplex method can be used. We shall define this method now. Consider the tree solution shown in Figure 13.13. It is dual feasible but not primal feasible (since $x_{db} < 0$). The basic idea that defines the dual simplex method is to pick a tree arc that is primal infeasible and let it leave the spanning tree (i.e., become nonbasic) and then readjust everything to preserve dual feasibility.

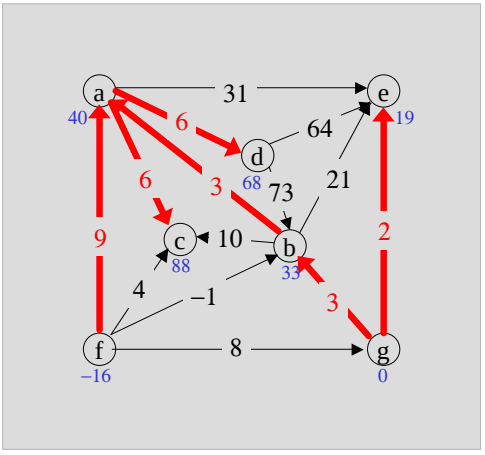


FIGURE 13.11. The tree solution at the end of the second iteration. To get from the spanning tree in Figure 13.9 to here, we let arc (b,a) enter and arc (b,c) leave.

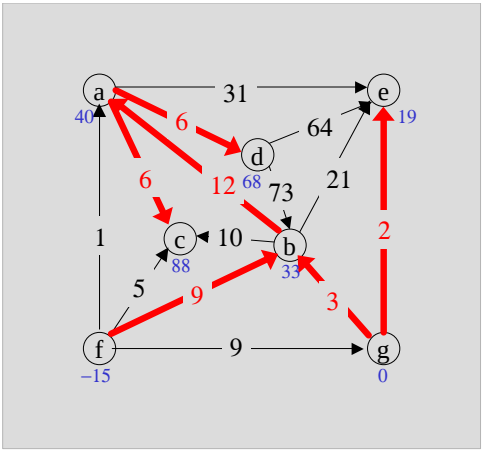


FIGURE 13.12. The tree solution at the end of the third iteration. To get from the spanning tree in Figure 13.11 to here, we let arc (f,b) enter and arc (f,a) leave. This tree solution is the *optimal* solution to the problem.

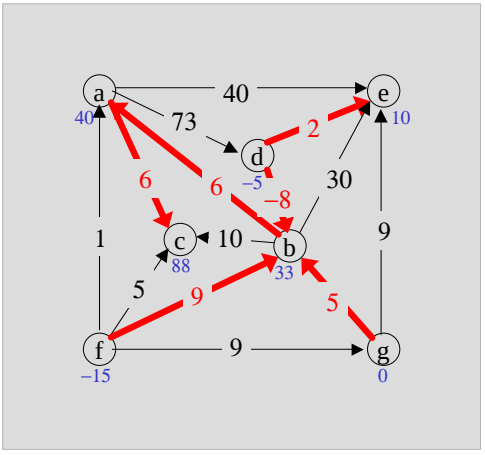


FIGURE 13.13. A tree solution that is dual feasible but not primal feasible.

The First Iteration. For the first iteration, we need to let arc (d,b) leave the spanning tree using a *dual pivot*, which is defined as follows. Removing arc (d,b) disconnects the spanning tree into two disjoint subtrees. The entering arc must be one of the arcs that spans across the two subtrees so that it can reconnect them into a spanning tree. That is, it must be one of

(a,e), (a,d), (b,e), or (g,e).

See Figure 13.14. To see how to decide which it must be, we need to consider carefully the impact of each possible choice.

To this end, let us consider the general situation. As mentioned above, the spanning tree with the leaving arc removed consists of two disjoint trees. The entering arc must reconnect these two trees.

First, consider a reconnecting arc that connects in the same direction as the leaving arc. When we add flow to this prospective entering arc, we will have to decrease flow on the leaving arc to maintain flow balance. Therefore, the leaving arc's flow, which is currently negative, can't be raised to zero. That is, the leaving arc can't leave. This is no good.

Now suppose that the reconnecting arc connects in the opposite direction. If it were to be the entering arc, then its dual slack would drop to zero. All other reconnecting arcs pointing in the same direction would drop by the same amount. To maintain nonnegativity of all the others, we must pick the one that drops the least. We can summarize the rule as follows:

Entering arc selection rule:

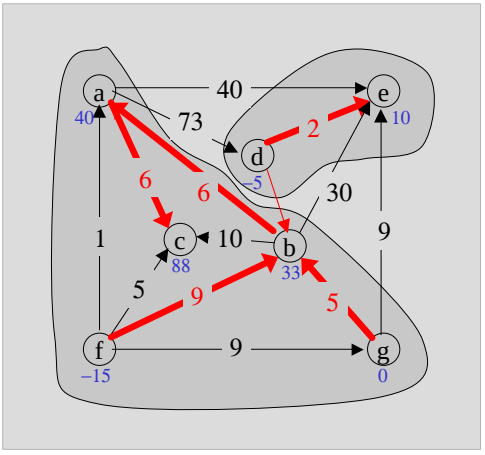


FIGURE 13.14. The two subtrees for the first pivot of the dual simplex method.

- the entering arc must bridge the two subtrees in the opposite direction from the leaving arc, and
- among all such arcs, it must have the smallest dual slack.

In our example, all bridging arcs point in the opposite direction from the leaving arc. The one with the smallest dual slack is (g,e) whose slack is $z_{ge} = 9$. This arc must be the entering arc.

We have now determined both the entering and leaving arcs. Hence, the new spanning tree is determined and therefore, in principle, all the variables associated with this new spanning tree can be computed. Furthermore, the rules for determining the new values by updating from the previous ones are the same as in the primal network simplex method. The resulting tree solution is shown in Figure 13.15.

The Second Iteration. For the second pivot, there are two choices for the leaving arc: (g,b) and (d,e). Using the most infeasible, we choose (d,e). We remove this arc from the spanning tree to produce two subtrees. One of the subtrees consists of just the node "d" all by itself while the other subtree consists of the rest of the nodes. Remembering that the reconnecting arc must bridge the two subtrees in the opposite direction, the only choice is (a,d). So this arc is the entering arc. Making the pivot, we arrive at the optimal tree solution shown in Figure 13.12.

5. Putting It All Together

As we saw in Chapter 5, for linear programming the primal and the dual simplex methods form the foundation on which one can build a few different variants of the simplex method. The same is true here in the context of network flows.

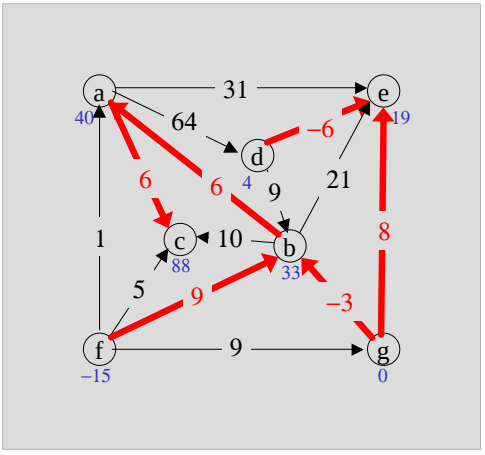


FIGURE 13.15. The tree solution after the first pivot.

For example, one can build a two-phased procedure in which one first uses the dual network simplex method (with costs artificially and temporarily altered to ensure dual feasibility of an initial tree solution) to find a primal feasible solution and then uses the primal network simplex method to move from the feasible solution to an optimal one.

Alternatively, one can use the primal network simplex method (with supplies temporarily altered to ensure primal feasibility of an initial tree solution) to find a dual feasible solution and then use the dual network simplex method (with the original supplies) to move from the dual feasible solution to an optimal one.

Finally, as described for linear programming in Chapter 7, one can define a parametric self-dual method in which primal pivots and dual pivots are intermingled as needed so as to reduce a perturbation parameter μ from ∞ to zero.

Since there is nothing new in how one builds the network versions of these algorithms from the basic primal and dual simplex pivots, we don't go through any examples here. Instead, we just mention one final observation about the dual variables, the y_i's. Namely, they are not needed anywhere in the performance of a primal or a dual pivot. Hence, their calculation is entirely optional and can be skipped altogether or simply defered to the end.

For completeness, we end this section by giving a step-by-step description of the *self-dual network simplex method*. The steps are as follows:

(1) *Identify a spanning tree*—any one will do (see Exercise 13.14). Also identify a root node.

(2) Compute *initial primal flows* on the tree arcs by assuming that nontree arcs have zero flow and the total flow at each node must be balanced. For this

calculation, the computed primal flows may be negative. In this case, the initial primal solution is not feasible. The calculation is performed working from leaf nodes inward.

(3) Compute *initial dual values* by working out from the root node along tree arcs using the formula

$$y_j - y_i = c_{ij},$$

which is valid on tree arcs, since the dual slacks vanish on these arcs.

(4) Compute *initial dual slacks* on each nontree arc using the formula

$$z_{ij} = y_i + c_{ij} - y_j.$$

Again, some of the z_{ij}'s might be nonnegative. This is okay (for now), but it is important that they satisfy the above equality.

(5) *Perturb* each primal flow and each dual slack that has a negative initial value by adding a positive scalar μ to each such value.

(6) *Identify a range* $\mu_{\text{MIN}} \leq \mu \leq \mu_{\text{MAX}}$ over which the current solution is optimal (on the first iteration, μ_{MAX} will be infinite).

(7) *Check the stopping rule:* if $\mu_{\text{MIN}} \leq 0$, then set $\mu = 0$ to recover an optimal solution. While not optimal, perform each of the remaining steps and then return to recheck this condition.

(8) *Select an arc* associated with the inequality $\mu_{\text{MIN}} \leq \mu$ (if there are several, pick one arbitrarily). If this arc is a nontree arc, then the current pivot is a *primal pivot.* If, on the other hand, it is a tree arc, then the pivot is a *dual pivot.*

   (a) If the pivot is a primal pivot, the arc identified above is the *entering arc*. Identify the associated *leaving arc* as follows. First, add the entering arc to the tree. With this arc added, there must be a cycle consisting of the entering arc and other tree arcs. The leaving arc is chosen from those arcs on the cycle that go in the opposite direction from the entering arc and having the smallest flow among all such arcs (evaluated at $\mu = \mu_{\text{MIN}}$).

   (b) If the pivot is a dual pivot, the arc identified above is the *leaving arc*. Identify the associated *entering arc* as follows. First, delete the leaving arc from the tree. This deletion splits the tree into two subtrees. The entering arc must bridge these two trees in the opposite direction to the leaving arc, and, among such arcs, it must be the one with the smallest dual slack (evaluated at $\mu = \mu_{\text{MIN}}$).

(9) *Update primal flows* as follows. Add the entering arc to the tree. This addition creates a cycle containing both the entering and leaving arcs. Adjust the flow on the leaving arc to zero, and then adjust the flows on each of the other cycle arcs as necessary to maintain flow balance.

(10) *Update dual variables* as follows. Delete the leaving arc from the old tree. This deletion splits the old tree into two subtrees. Let $\mathcal{T}_u$ denote the subtree containing the tail of the entering arc, and let $\mathcal{T}_v$ denote the subtree containing its head. The dual variables for nodes in $\mathcal{T}_u$ remain unchanged, but the dual variables for nodes in $\mathcal{T}_v$ get incremented by the old dual slack on the entering arc.

(11) *Update dual slacks* as follows. All dual slacks remain unchanged except for those associated with nontree arcs that bridge the two subtrees $\mathcal{T}_u$ and $\mathcal{T}_v$. The dual slacks corresponding to those arcs that bridge in the same direction as the entering arc get decremented by the old dual slack on the entering arc, whereas those that correspond to arcs bridging in the opposite direction get incremented by this amount.

As was said before and should now be clear, there is no need to update the dual variables from one iteration to the next; that is, step 10 can be skipped.

6. The Integrality Theorem

In this section, we consider network flow problems for which all the supplies and demands are integers. Such problems are called *network flow problems with integer data*. As we explained in Section 13.2, for network flow problems, basic primal solutions are computed without any multiplication or division. The following important theorem follows immediately from this property:

THEOREM 13.2. Integrality Theorem. *For network flow problems with integer data, every basic feasible solution and, in particular, every basic optimal solution assigns integer flow to every arc.*

This theorem is important because many real-world network flow problems have integral supplies/demands and require their solutions to be integral too. This integrality restriction typically occurs when one is shipping indivisible units through a network. For example, it wouldn't make sense to ship one third of a car from an automobile assembly plant to one dealership with the other two thirds going to another dealership.

Problems that are linear programming problems with the additional stipulation that the optimal solution values must be integers are called *integer programming problems*. Generally speaking, these problems are much harder to solve than linear programming problems (see Chapter 22). However, if the problem is a network flow problem with integer data, it can be solved efficiently using the simplex method to compute a basic optimal solution, which the integrality theorem tells us will be integer valued.

6.1. König's Theorem. In addition to its importance in real-world optimization problems, the integrality theorem also has many applications to the branch of mathematics called combinatorics. We illustrate with just one example.

THEOREM 13.3. König's Theorem. *Suppose that there are n girls and n boys, that every girl knows exactly k boys, and that every boy knows exactly k girls. Then n marriages can be arranged with everybody knowing his or her spouse.*

Before proving this theorem it is important to clarify its statement by saying that the property of "knowing" is symmetric (for example, knowing in the biblical sense). That is, if a certain girl knows a certain boy, then this boy also knows this girl.

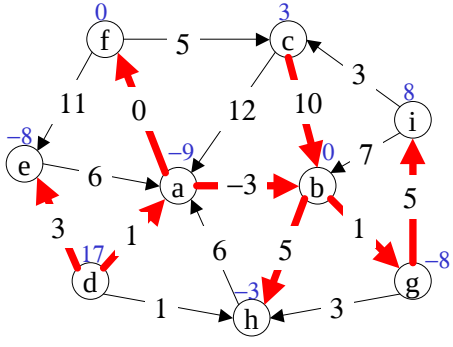
PROOF. Consider a network with nodes $g_1, g_2, \ldots, g_n, b_1, b_2, \ldots, b_n$ and an arc from g_i to b_j if girl i and boy j know each other. Assign one unit of supply to each girl node and a unit of demand to each boy node. Assign arbitrary objective coefficients to create a well-defined network flow problem. The problem is guaranteed to be feasible: just put a flow of $1/k$ on each arc (the polygamists in the group might prefer this nonintegral solution). By the integrality theorem, the problem has an integer-valued solution. Clearly, the flow on each arc must be either zero or one. Also, each girl node is the tail of exactly one arc having a flow of one. This arc points to her intended mate. □

Exercises

In solving the following problems, the network pivot tool can be used to check your arithmetic:

campuscgi.princeton.edu/∼rvdb/JAVA/network/nettool/netsimp.html

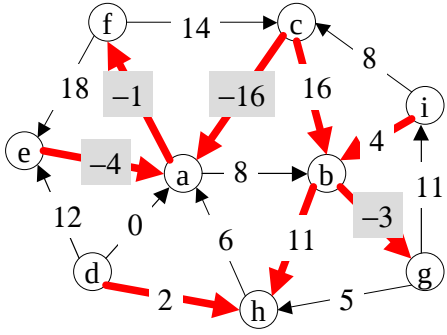
13.1 Consider the following network flow problem:



Numbers shown above the nodes are supplies (negative values represent demands) and numbers shown above the arcs are unit shipping costs. The darkened arcs form a spanning tree.

(a) Compute primal flows for each tree arc.
(b) Compute dual variables for each node.
(c) Compute dual slacks for each nontree arc.

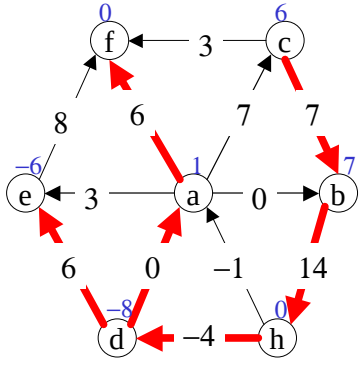
13.2 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.

(a) Using the largest–coefficient rule in the dual network simplex method, what is the leaving arc?
(b) What is the entering arc?
(c) After *one* pivot, what is the new tree solution?

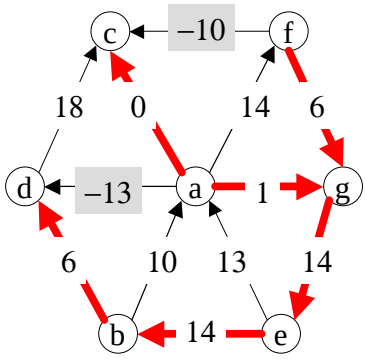
13.3 Consider the following network flow problem:



The numbers above the nodes are supplies (negative values represent demands) and numbers shown above the arcs are unit shipping costs. The darkened arcs form a spanning tree.

(a) Compute primal flows for each tree arc.
(b) Compute dual variables for each node.
(c) Compute dual slacks for each nontree arc.

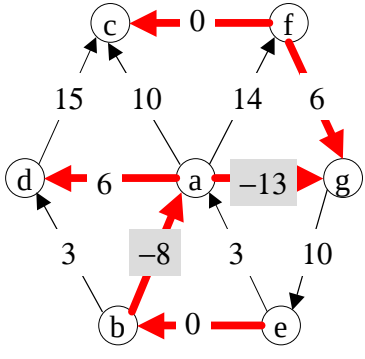
13.4 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.

(a) Using the largest–coefficient rule in the primal network simplex method, what is the entering arc?
(b) What is the leaving arc?
(c) After *one* pivot, what is the new tree solution?

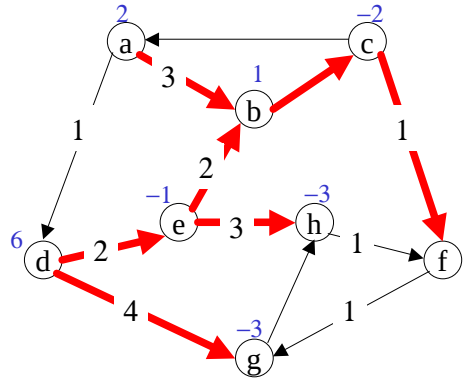
13.5 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.

(a) Using the largest–coefficient rule in the dual network simplex method, what is the leaving arc?
(b) What is the entering arc?
(c) After *one* pivot, what is the new tree solution?

13.6 Solve the following network flow problem starting with the spanning tree shown.



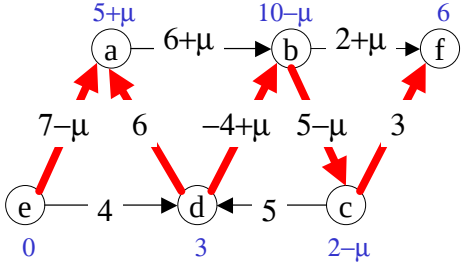
The numbers displayed next to nodes are supplies(+)/demands(−). Numbers on arcs are costs. Missing data should be assumed to be zero. The bold arcs represent an initial spanning tree.

13.7 Solve Exercise 2.11 using the self-dual network simplex method.

13.8 Using today's date (MMYY) for the seed value, solve 10 problems using the network simplex pivot tool:

campuscgi.princeton.edu/∼rvdb/JAVA/network/challenge/netsimp.html .

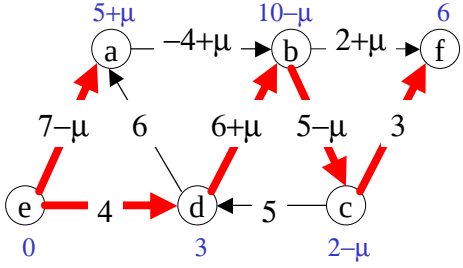
13.9 Consider the following tree solution for a minimum cost network flow problem:



As usual, bold arcs represent arcs on the spanning tree, numbers next to the bold arcs are primal flows, numbers next to non-bold arcs are dual slacks, and numbers next to nodes are dual variables.

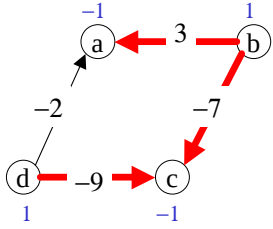
(a) For what values of μ is this tree solution optimal?
(b) What are the entering and leaving arcs?
(c) After *one* pivot, what is the new tree solution?
(d) For what values of μ is the new tree solution optimal?

13.10 Consider the following tree solution for a minimum cost network flow problem:



(a) For what values of μ is this tree solution optimal?
(b) What are the entering and leaving arcs?
(c) After *one* pivot, what is the new tree solution?
(d) For what values of μ is the new tree solution optimal?

13.11 Consider the following minimum cost network flow problem



As usual, the numbers on the arcs represent the flow costs and numbers at the nodes represent supplies (demands are shown as negative supplies). The arcs shown in bold represent a spanning tree. If the solution corresponding to this spanning tree is optimal prove it, otherwise find an optimal solution using this tree as the initial spanning tree.

13.12 Suppose that a square submatrix of $\tilde{A}$ is invertible. Show that the arcs corresponding to the columns of this submatrix form a spanning tree.

13.13 Show that a spanning tree on m nodes must have exactly $m - 1$ arcs.

13.14 Define an algorithm that takes as input a network and either finds a spanning tree or proves that the network is not connected.

13.15 Give an example of a minimum-cost network flow problem with all arc costs positive and the following counterintuitive property: if the supply at a particular source node and the demand at a particular sink node are simultaneously reduced by one unit, then the optimal cost increases.

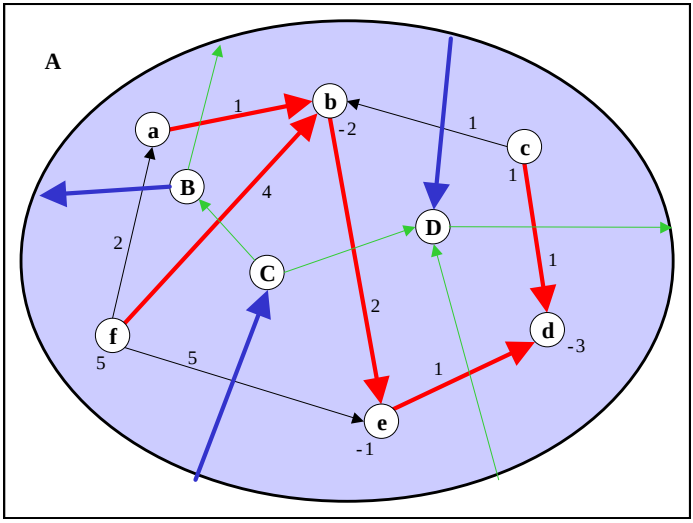


FIGURE 13.16. The primal network has nodes "a" through "f". The corresponding dual network has nodes "A" through "D" (node "A" is "at infinity"). A primal spanning tree is shown. It consists of five arcs: (a,b), (f,b), (b,e), (e,d), and (c,d). The corresponding dual spanning tree consists of three arcs: (B,A), (A,C), and (D,A). Primal costs are shown along the primal arcs and supplies/demands are shown at the primal nodes.

13.16 Consider a possibly disconnected network $(\mathcal{N}, \mathcal{A})$. Two nodes i and j in $\mathcal{N}$ are said to be *connected* if there is a path from i to j (recall that paths can traverse arcs backwards or forwards). We write $i \sim j$ if i and j are connected.

(a) Show that "$\sim$" defines an *equivalence relation*. That is, it has the following three properties:

(i) (reflexivity) for all $i \in \mathcal{N}$, $i \sim i$;

(ii) (symmetry) for all $i, j \in \mathcal{N}$, $i \sim j$ implies that $j \sim i$;

(iii) (transitivity) for all $i, j, k \in \mathcal{N}$, $i \sim j$ and $j \sim k$ implies that $i \sim k$.

Using the equivalence relation, we can partition $\mathcal{N}$ into a collection of subsets of *equivalence classes* $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ such that two nodes are connected if and only if they belong to the same subset. The number k is called the number of *connected components*.

(b) Show that the rank of the node–arc incidence matrix A is exactly $m - k$ (recall that m is the number of rows of A).

13.17 One may assume without loss of generality that every node in a minimum cost network flow problem has at least two arcs associated with it. Why?

13.18 The sum of the dual slacks around any cycle is a constant. What is that constant?

13.19 *Planar Networks.* A network is called *planar* if the nodes and arcs can be layed out on the two-dimensional plane in such a manner that no two arcs cross each other (it is allowed to draw the arcs as curves if necessary). All of the networks encountered so far in this chapter have been planar. Associated with each planar network is a geometrically defined dual network. The purpose of this problem is to establish the following interesting fact:

A dual network simplex pivot is precisely a primal network simplex method applied to the dual network.

Viewed geometrically, the nodes of a planar graph are called *vertices* and the arcs are called *edges*. Consider a specific connected planar network. If one were to delete the vertices and the edges from the plane, one would be left with a disjoint collection of subsets of the plane. These subsets are called *faces*. Note that there is one unbounded face. It is a face just like the other bounded ones. An example of a connected planar network with its faces labeled A through D is shown in Figure 13.16.

Dual nodes. Associated with each connected planar network is a *dual network* defined by interchanging vertices and faces. That is, place a dual vertex in the center of each primal face. Note: the dual vertex corresponding to the unbounded primal face could be placed anywhere in the unbounded face but we choose to put it *at infinity*. In this way, dual edges (defined next) that have a head or a tail at this node can run off to infinity in any direction.

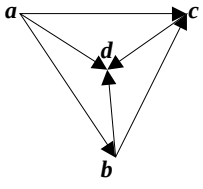
Dual arcs. Connect with a dual edge any pair of dual nodes whose corresponding primal faces share an edge. Each dual edge crosses exactly one primal edge. The directionality of the dual edge is determined as follows: first, place a vector along the corresponding primal edge pointing in the direction of the primal arc, and then rotate it counterclockwise until it is tangent to the dual edge. The vector now defines the direction for the dual arc.

Dual spanning tree. Consider a spanning tree on the primal network and suppose that a primal–dual tree solution is given. We define a *spanning tree* on the dual network as follows. A dual edge is on the dual network's spanning tree if and only if the corresponding primal edge is not on the primal network's spanning tree.

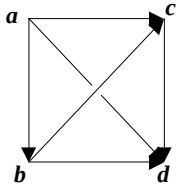
Dual flows and dual dual-slacks. The numerical arc data for the dual network is inherited directly from the primal. That is, flows on the dual tree arcs are exactly equal to the dual slacks on the associated primal nontree

arcs. And, the dual slacks on the the dual nontree arcs are exactly equal to the primal flows on the associated primal tree arcs. Having specified numerical data on the arcs of the dual network, it is fairly straightforward to determine values for supplies/demands at the nodes and shipping costs along the arcs that are consistent with these numerical values.

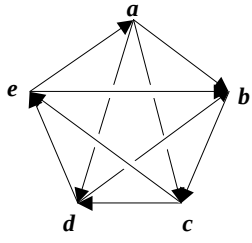
(a) Which of the following networks are planar:



(a)



(b)



(c)

(b) A network is called *complete* if there is an arc between every pair of nodes. If a complete network with m nodes is planar, then every network with m nodes is planar. Prove it.

(c) Show that a nonplanar network must have 5 or more nodes.

(d) As always, let m denote the number of nodes and let n denote the number of arcs in a network. Let f denote the number of faces in a planar network. Show by induction on f that $m = n - f + 2$.

(e) Show that the dual spanning tree defined above is in fact a spanning tree.

(f) Show that a dual pivot for a minimum cost network flow problem defined on the primal network is precisely the same as a primal pivot for the corresponding network flow problem on the dual network.

(g) Using the cost and supply/demand information given for the primal problem in Figure 13.16, write down the primal problem as a linear programming problem.

(h) Write down the dual linear programming problem that one derives algebraically from the primal linear programming problem.

(i) Using the spanning tree shown in Figure 13.16, compute the primal flows, dual variables, and dual slacks for the network flow problem associated with the primal network.

(j) Write down the flow and slacks for the network flow problem associated with the dual network.

(k) Find arc costs and node supplies/demands for the dual network that are consistent with the flows and slacks just computed.

(l) Write down the linear programming problem associated with the network flow problem on the dual network.

Notes

The classical reference is Ford & Fulkerson (1962). More recent works include the books by Christofides (1975), Lawler (1976), Bazaraa et al. (1977), Kennington & Helgason (1980), Jensen & Barnes (1980), Bertsekas (1991), and Ahuja et al. (1993).

The two "original" algorithms for solving minimum-cost network flow problems are the *network simplex method* developed by Dantzig (1951*a*) and the *primal–dual method* developed by Ford & Fulkerson (1958). The self-dual algorithm described in this chapter is neither of these. In fact, it resembles the "out-of-kilter" method described by Ford & Fulkerson (1962).

CHAPTER 14

Applications

In this chapter, we discuss briefly the most important applications of network flow problems.

1. The Transportation Problem

The network flow problem, when thought of as representing the shipment of goods along a transportation network, is called the *transshipment problem*. An important special case is when the set of nodes $\mathcal{N}$ can be partitioned into two sets $\mathcal{S}$ and $\mathcal{D}$,

$$\mathcal{N} = \mathcal{S} \cup \mathcal{D}, \qquad \mathcal{S} \cap \mathcal{D} = \emptyset,$$

such that every arc in $\mathcal{A}$ has its tail in $\mathcal{S}$ and its head in $\mathcal{D}$. The nodes in $\mathcal{S}$ are called *source (or supply) nodes*, while those in $\mathcal{D}$ are called *destination (or demand) nodes*. Such graphs are called *bipartite graphs* (see Figure 14.1). A network flow problem on such a bipartite graph is called a *transportation problem*. In order for a transportation

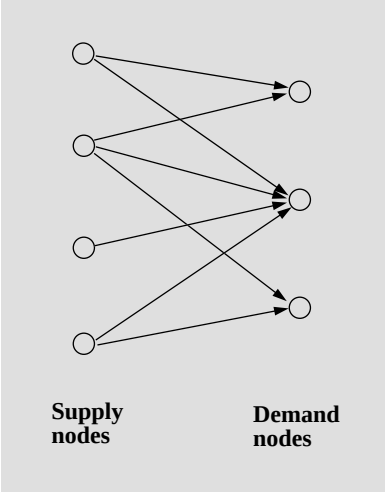


FIGURE 14.1. A bipartite graph—the network for a transportation problem.

problem to be feasible, the supply must be nonnegative at every supply node, and the demand must be nonnegative at every demand node. That is,

$$b_i \geq 0 \quad \text{for } i \in \mathcal{S},$$
$$b_i \leq 0 \quad \text{for } i \in \mathcal{D}.$$

When put on paper, a bipartite graph has the annoying property that the arcs tend to cross each other many times. This makes such a representation inconvenient for carrying out the steps of the network simplex method. But there is a nice, uncluttered, tabular representation of a bipartite graph that one can use when applying the simplex method. To discover this tabular representation, first suppose that the graph is laid out as shown in Figure 14.2. Now if we place the supplies and demands on the nodes and the costs at the kinks in the arcs, then we get, for example, the following simple tabular representation of a transportation problem:

(14.1)

	-10	-23	-15
7	5	6	$*$
11	8	4	3
18	$*$	9	$*$
12	$*$	3	6

(the asterisks represent nonexistent arcs). The iterations of the simplex method can be written in this tabular format by simply placing the dual variables where the supplies and demands are and by placing the primal flows and dual slacks where the arc costs are. Of course, some notation needs to be introduced to indicate which cells are part of the current spanning tree. For example, the tree could be indicated by putting a box around the primal flow values. Here is a (nonoptimal) tree solution for the data given above:

(14.2)

	5	1	4
0	$\boxed{7}$	5	$*$
-3	$\boxed{3}$	$\boxed{8}$	-4
-8	$*$	$\boxed{18}$	$*$
-2	$*$	$\boxed{-3}$	$\boxed{15}$

(the solution to this problem is left as an exercise).

In the case where every supply node is connected to every demand node, the problem is called the *Hitchcock Transportation Problem*. In this case, the equations defining the problem are especially simple. Indeed, if we denote the supplies at the supply nodes by r_i, $i \in \mathcal{S}$, and if we denote the demands at the demand nodes by s_j,

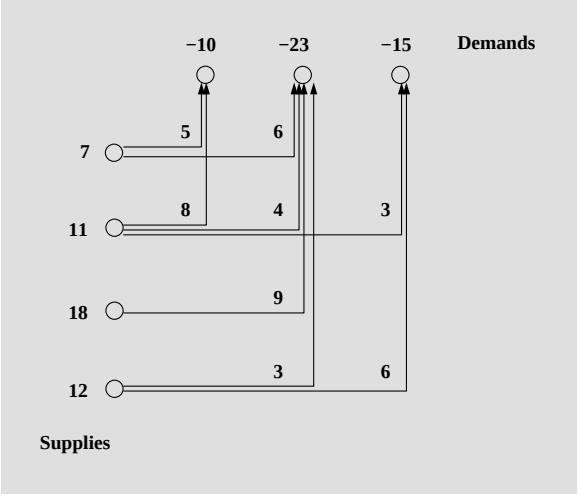


FIGURE 14.2. The bipartite graph from Figure 14.1 laid out in a rectangular fashion, with supplies and demands given at the nodes, and with costs given on the arcs.

$j \in \mathcal{D}$, then we can write the problem as

$$\begin{aligned} \text{minimize} \quad & \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{D}} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{j\in\mathcal{D}} x_{ij} = r_i \qquad i \in \mathcal{S} \\ & \sum_{i\in\mathcal{S}} x_{ij} = s_j \qquad j \in \mathcal{D} \\ & x_{ij} \geq 0 \qquad i \in \mathcal{S}, j \in \mathcal{D}. \end{aligned}$$

2. The Assignment Problem

Given a set $\mathcal{S}$ of m people, a set $\mathcal{D}$ of m tasks, and for each $i \in \mathcal{S}$, $j \in \mathcal{D}$ a cost c_{ij} associated with assigning person i to task j, the *assignment problem* is to assign each person to one and only one task in such a manner that each task gets covered by someone and the total cost of the assignments is minimized. If we let

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned task } j, \\ 0 & \text{otherwise,} \end{cases}$$

then the objective function can be written as

$$\text{minimize} \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij}.$$

The constraint that each person is assigned exactly one task can be expressed simply as

$$\sum_{j \in \mathcal{D}} x_{ij} = 1, \qquad \text{for all } i \in \mathcal{S}.$$

Also, the constraint that every task gets covered by someone is just

$$\sum_{i \in \mathcal{S}} x_{ij} = 1, \qquad \text{for all } j \in \mathcal{D}.$$

Except for the assumed integrality of the decision variables, x_{ij}, the assignment problem is just a Hitchcock transportation problem in which the supply at every supply node (person) is one and the demand at every demand node (task) is also one. This Hitchcock transportation problem therefore is called the *LP-relaxation* of the assignment problem. It is easy to see that every feasible solution to the assignment problem is a feasible solution for its LP-relaxation. Furthermore, every integral feasible solution to the LP-relaxation is a feasible solution to the assignment problem. Since the network simplex method applied to the LP-relaxation produces an integral solution, it therefore follows that the method solves not only the LP-relaxation but also the assignment problem itself. We should note that this is a very special and important feature of the network simplex method. For example, had we used the primal–dual interior-point method to solve the LP-relaxation, there would be no guarantee that the solution obtained would be integral (unless the problem has a unique optimal solution, in which case any LP solver would find the same, integral answer—but typical assignment problems have alternate optimal solutions, and an interior-point method will report a convex combination of all of them).

3. The Shortest-Path Problem

Roughly speaking, the shortest-path problem is to find, well, the shortest path from one specific node to another in a network $(\mathcal{N}, \mathcal{A})$. In contrast to earlier usage, the arcs connecting successive nodes on a path must point in the direction of travel. Such paths are sometimes referred to as *directed paths*. To determine a shortest path, we assume that we are given the length of each arc. To be consistent with earlier notations, let us assume that the length of arc (i, j) is denoted by c_{ij}. Naturally, we assume that these lengths are nonnegative.

To find the shortest path from one node (say, s) to another (say, r), we will see that it is necessary to compute the shortest path from many, perhaps all, other nodes to r. Hence, we define the *shortest-path problem* as the problem of finding the shortest path from every node in $\mathcal{N}$ to a specific node $r \in \mathcal{N}$. The destination node r is called the *root node*.

3.1. Network Flow Formulation. The shortest-path problem can be formulated as a network flow problem. Indeed, put a supply of one unit at each nonroot node, and put the appropriate amount of demand at the root (to meet the total supply). The cost on each arc is just the length of the arc. Suppose that we've solved this network flow problem. Then the shortest path from a node i to r can be found by simply following the arcs from i to r on the optimal spanning tree. Also, the length of the shortest path is $y_r^* - y_i^*$.

While the network simplex method can be used to solve the shortest-path problem, there are faster algorithms designed especially for it. To describe these algorithms, let us denote the distance from i to r by v_i. These distances (or approximations thereof) are called *labels* in the networks literature. Some algorithms compute these distances systematically in a certain order. These algorithms are called *label-setting algorithms*. Other algorithms start with an estimate for these labels and then iteratively correct the estimates until the optimal values are found. Such algorithms are called *label-correcting algorithms*.

Note that if we set y_r^* to zero in the network flow solution, then the labels are simply the negative of the optimal dual variables. In the following subsections, we shall describe simple examples of label-setting and label-correcting algorithms.

3.2. A Label-Correcting Algorithm. To describe a label-correcting algorithm, we need to identify a system of equations that characterize the shortest-path distances. First of all, clearly

$$v_r = 0.$$

What can we say about the labels at other nodes, say, node i? Suppose that we select an arc (i, j) that leaves node i. If we were to travel along this arc and then, from node j, travel along the shortest path to r, then the distance to the root would be $c_{ij} + v_j$. So, from node i, we should select the arc that minimizes these distances. This selection will then give the shortest distance from i to r. That is,

$$v_i = \min\{c_{ij} + v_j : (i, j) \in \mathcal{A}\}, \qquad i \neq r. \tag{14.3}$$

The argument we have just made is called the *principle of dynamic programming*, and equation (14.3) is called *Bellman's equation*. Dynamic programming is a whole subject of its own—we shall only illustrate some of its basic ideas by our study of the shortest-path problem. In the dynamic programming literature, the set of v_i's viewed as a function defined on the nodes is called the *value function* (hence the notation).

From Bellman's equation, it is easy to identify the arcs one would travel on in a shortest-path route to the root. Indeed, these arcs are given by

$$\mathcal{T} = \{(i, j) \in \mathcal{A} : v_i = c_{ij} + v_j\}.$$

This set of arcs may contain alternate shortest paths to the root, and so the set is not necessarily a tree. Nonetheless, any path that follows these arcs will get to the root on a shortest-path route.

3.2.1. *Method of Successive Approximation.* Bellman's equation is an implicit system of equations for the values v_i, $i \in \mathcal{N}$. Implicit equations such as this arise frequently and beg to be solved by starting with a guess at the solution, using this guess in the right-hand side, and computing a new guess by evaluating the right-hand side. This approach is called the *method of successive approximations*. To apply it to the shortest-path problem, we initialize the labels as follows:

$$v_i^{(0)} = \begin{cases} 0 & i = r \\ \infty & i \neq r. \end{cases}$$

Then the updates are computed using Bellman's equation:

$$v_i^{(k+1)} = \begin{cases} 0 & i = r \\ \min\{c_{ij} + v_j^{(k)} : (i,j) \in \mathcal{A}\} & i \neq r. \end{cases}$$

3.2.2. *Efficiency.* The algorithm stops when an update leaves all the v_i's unchanged. It turns out that the algorithm is guaranteed to stop in no more than m iterations. To see why, it suffices to note that $v_i^{(k)}$ has a very simple description: it is the length of the shortest path from i to r that has k or fewer arcs in the path. (It is not hard to convince yourself with an induction on k that this is correct, but a pedantic proof requires introducing a significant amount of added notation that we wish to avoid.) Hence, the label-correcting algorithm cannot take more than m iterations, since every shortest path can visit each node at most once. Since each iteration involves looking at every arc of the network, it follows that the number of additions/comparisons needed to solve a shortest-path problem using the label-correcting algorithm is about nm.

3.3. A Label-Setting Algorithm. In this section, we describe *Dijkstra's algorithm* for solving shortest-path problems. The data structures that are carried from one iteration to the next are a set $\mathcal{F}$ of *finished* nodes and two arrays indexed by the nodes of the graph. The first array, v_j, $j \in \mathcal{N}$, is just the array of labels. The second array, h_i, $i \in \mathcal{N}$, indicates the next node to visit from node i in a shortest path. As the algorithm proceeds, the set $\mathcal{F}$ contains those nodes for which the shortest path has already been found. This set starts out empty. Each iteration of the algorithm adds one node to it. This is why the algorithm is called a label-setting algorithm, since each iteration sets one label to its optimal value. For finished nodes, the labels are fixed at their optimal values. For each unfinished node, the label has a temporary value, which represents the length of the shortest path from that node to the root, subject to the condition that all intermediate nodes on the path must be finished nodes. At those nodes for which no such path exists, the temporary label is set to infinity (or, in practice, a large positive number).

The algorithm is initialized by setting all the labels to infinity except for the root node, whose label is set to 0. Also, the set of finished nodes is initialized to the empty set. Then, as long as there remain unfinished nodes, the algorithm selects an unfinished

Initialize:
$\mathcal{F} = \emptyset$
$v_j = \begin{cases} 0 & j = r, \\ \infty & j \neq r. \end{cases}$
while $(|\mathcal{F}^c| > 0)\{$
$j = \text{argmin}\{v_k : k \notin \mathcal{F}\}$
$\mathcal{F} \leftarrow \mathcal{F} \cup \{j\}$
for each i for which $(i, j) \in \mathcal{A}$ and $i \notin \mathcal{F}$ {
if $(c_{ij} + v_j < v_i)$ {
$v_i = c_{ij} + v_j$
$h_i = j$
}
}
}

FIGURE 14.3. Dijkstra's shortest-path algorithm.

node j having the smallest temporary label, adds it to the set of finished nodes, and then updates each unfinished "upstream" neighbor i by setting its label to $c_{ij} + v_j$ if this value is smaller than the current value v_i. For each neighbor i whose label gets changed, h_i is set to j. The algorithm is summarized in Figure 14.3.

4. Upper-Bounded Network Flow Problems

Some real-world network flow problems involve upper bounds on the amount of flow that an arc can handle. There are modified versions of the network simplex method that allow one to handle such upper bounds implicitly, but we shall simply show how to reduce an upper-bounded network flow problem to one without upper bounds.

Let us consider just one arc, (i, j), in a network flow problem. Suppose that there is an upper bound of u_{ij} on the amount of flow that this arc can handle. We can express

this bound as an extra constraint:

$$0 \le x_{ij} \le u_{ij}.$$

Introducing a slack variable, t_{ij}, we can rewrite these bound constraints as

$$\begin{aligned} x_{ij} + t_{ij} &= u_{ij} \\ x_{ij},\ t_{ij} &\ge 0. \end{aligned}$$

If we look at the flow balance constraints and focus our attention on the variables x_{ij} and t_{ij}, we see that they appear in only three constraints: the flow balance constraints for nodes i and j and the upper bound constraint,

$$\begin{array}{rcl} \cdots\ -x_{ij}\ \cdots & & = -b_i \\ \cdots\quad x_{ij}\ \cdots & & = -b_j \\ x_{ij} & +t_{ij} & = u_{ij}. \end{array}$$

If we subtract the last constraint from the second one, we get

$$\begin{array}{rcl} \cdots\ -x_{ij}\ \cdots & = & -b_i \\ \cdots \quad \cdots\ -t_{ij} & = & -b_j - u_{ij} \\ x_{ij} \quad +t_{ij} & = & u_{ij}. \end{array}$$

Note that we have restored a network structure in the sense that each column again has one $+1$ and one -1 coefficient. To make a network picture, we need to create a new node (corresponding to the third row). Let us call this node k. The network transformation is shown in Figure 14.4.

We can use the above transformation to derive optimality conditions for upper-bounded network flow problems. Indeed, let us consider an optimal solution to the transformed problem. Clearly, if x_{ik} is zero, then the corresponding dual slack $z_{ik} = y_i + c_{ij} - y_k$ is nonnegative:

$$y_i + c_{ij} - y_k \ge 0. \tag{14.4}$$

Furthermore, the back-flow x_{jk} must be at the upper bound rate:

$$x_{jk} = u_{ij}.$$

Hence, by complementarity, the corresponding dual slack must vanish:

$$z_{jk} = y_j - y_k = 0. \tag{14.5}$$

Combining (14.4) with (14.5), we see that

$$y_i + c_{ij} \ge y_j.$$

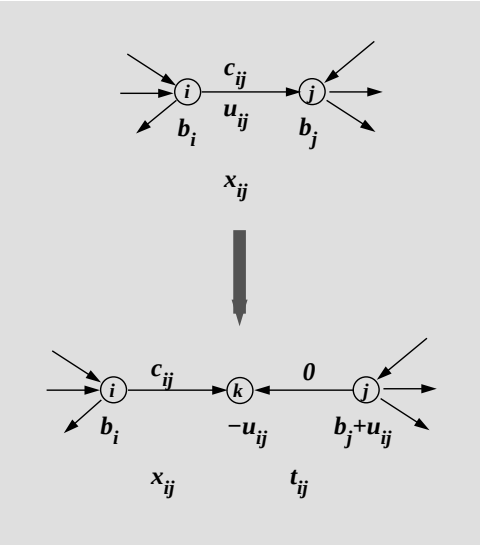


FIGURE 14.4. Adding a new node, k, to accommodate an arc (i, j) having an upper bound u_{ij} on its flow capacity.

On the other hand, if the flow on arc (i, k) is at the capacity value, then the back-flow on arc (j, k) must vanish. The complementarity conditions then say that

$$z_{ik} = y_i + c_{ij} - y_k = 0$$
$$z_{jk} = y_j - y_k \geq 0.$$

Combining these two statements, we get

$$y_i + c_{ij} \leq y_j.$$

Finally, if $0 < x_{ij} < u_{ij}$, then both slack variables vanish, and this implies that

$$y_i + c_{ij} = y_j.$$

These properties can then be summarized as follows:

$$\begin{aligned} x_{ij} = 0 &\Longrightarrow y_i + c_{ij} \geq y_j \\ x_{ij} = u_{ij} &\Longrightarrow y_i + c_{ij} \leq y_j \\ 0 < x_{ij} < u_{ij} &\Longrightarrow y_i + c_{ij} = y_j. \end{aligned} \tag{14.6}$$

While upper-bounded network flow problems have important applications, we admit that our main interest in them is more narrowly focused. It stems from their relation to an important theorem called the Max-Flow Min-Cut Theorem. We shall state and prove this theorem in the next section. The only tool we need to prove

this theorem is the above result giving the complementarity conditions when there are upper bounds on arcs. So on with the show.

5. The Maximum-Flow Problem

The subject of this section is the class of problems called *maximum-flow problems*. These problems form an important topic in the theory of network flows. There are very efficient algorithms for solving them, and they appear as subproblems in many algorithms for the general network flow problem. However, our aim is rather modest. We wish only to expose the reader to one important theorem in this subject, which is called the Max-Flow Min-Cut Theorem.

Before we can state this theorem we need to set up the situation. Suppose that we are given a network $(\mathcal{N}, \mathcal{A})$, a distinguished node $s \in \mathcal{N}$ called the *source node*, a distinguished node $t \in \mathcal{N}$ called the *sink node*, and upper bounds on the arcs of the network u_{ij}, $(i, j) \in \mathcal{A}$. For simplicity, we shall assume that the upper bounds are all finite (although this is not really necessary). The objective is to "push" as much flow from s to t as possible.

To solve this problem, we can convert it to an upper-bounded network flow problem as follows. First, let $c_{ij} = 0$ for all arcs $(i, j) \in \mathcal{A}$, and let $b_i = 0$ for every node $i \in \mathcal{N}$. Then add one extra arc (t, s) connecting the sink node t back to the source node s, put a negative cost on this arc (say, $c_{ts} = -1$), and let it have infinite capacity $u_{ts} = \infty$. Since the only nonzero cost is actually negative, it follows that we shall actually make a profit by letting more and more flow circulate through the network. But the upper bound on the arc capacities limits the amount of flow that it is possible to push through.

In order to state the Max-Flow Min-Cut Theorem, we must define what we mean by a cut. A *cut*, C, is a set of nodes that contains the source node but does not contain the sink node (see Figure 14.5). The *capacity* of a cut is defined as

$$\kappa(C) = \sum_{\substack{i \in C \\ j \notin C}} u_{ij}.$$

Note that here and elsewhere in this section, the summations are over "original" arcs that satisfy the indicated set membership conditions. That is, they don't include the arc that we added connecting from t back to s. (If it did, the capacity of every cut would be infinite—which is clearly not our intention.)

Flow balance tells us that the total flow along original arcs connecting the cut set C to its complement minus the total flow along original arcs that span these two sets in the opposite direction must equal the amount of flow on the artificial arc (t, s). That is,

$$x_{ts} = \sum_{\substack{i \in C \\ j \notin C}} x_{ij} - \sum_{\substack{i \notin C \\ j \in C}} x_{ij}. \tag{14.7}$$

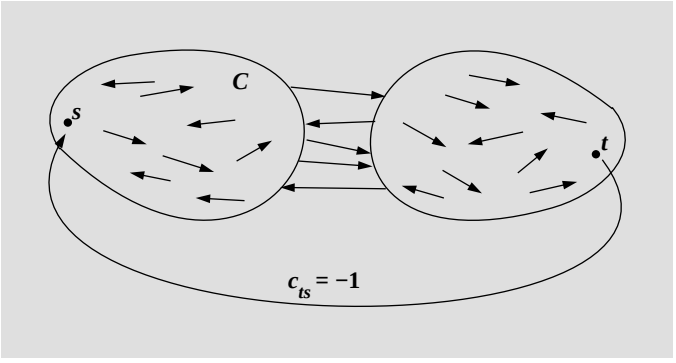


FIGURE 14.5. A cut set C for a maximum flow problem.

We are now ready to state the Max-Flow Min-Cut Theorem.

THEOREM 14.1. *The maximum value of x_{ts} equals the minimum value of $\kappa(C)$.*

PROOF. The proof follows the usual sort of pattern common in subjects where there is a sort of duality theory. First of all, we note that it follows from (14.7) that

$$x_{ts} \leq \kappa(C) \tag{14.8}$$

for every feasible flow and every cut set C. Then all that is required is to exhibit a feasible flow and a cut set for which this inequality is an equality.

Let x^*_{ij}, $(i,j) \in \mathcal{A}$, denote the optimal values of the primal variables, and let y^*_i, $i \in \mathcal{N}$, denote the optimal values of the dual variables. Then the complementarity conditions (14.6) imply that

$$x^*_{ij} = 0 \text{ whenever } y^*_i + c_{ij} > y^*_j \tag{14.9}$$

$$x^*_{ij} = u_{ij} \text{ whenever } y^*_i + c_{ij} < y^*_j. \tag{14.10}$$

In particular,

$$y^*_t - 1 \geq y^*_s$$

(since $u_{ts} = \infty$). Put $C^* = \{k : y^*_k \leq y^*_s\}$. Clearly, C^* is a cut.

Consider an arc having its tail in C^* and its head in the complement of C^*. It follows from the definition of C^* that $y^*_i \leq y^*_s < y^*_j$. Since c_{ij} is zero, we see from (14.10) that $x^*_{ij} = u_{ij}$.

Now consider an original arc having its tail in the complement of C^* and its head in C^* (i.e., bridging the two sets in the opposite direction). It follows then that $y^*_j \leq y^*_s < y^*_i$. Hence, we see from (14.9) that $x^*_{ij} = 0$.

Combining the observations of the last two paragraphs with (14.7), we see that

$$x^*_{ts} = \sum_{\substack{i \in C \\ j \notin C}} u_{ij} = \kappa(C^*).$$

In light of (14.8), the proof is complete. □

Exercises

14.1 Solve the transportation problem given in (14.1), using (14.2) for the starting tree solution.

14.2 Solve the following linear programming problem:

$$\begin{array}{lrcl}
\text{maximize} & 7x_1 - 3x_2 + 9x_3 + 2x_4 & & \\
\text{subject to} & x_1 + x_2 & \leq & 1 \\
& x_3 + x_4 & \leq & 1 \\
& x_1 + x_3 & \geq & 1 \\
& x_2 + x_4 & \geq & 1 \\
& x_1,\ x_2,\ x_3,\ x_4 & \geq & 0.
\end{array}$$

(Note: there are two greater-than-or-equal-to constraints.)

14.3 Bob, Carol, David, and Alice are stranded on a desert island. Bob and David each would like to give their affection to Carol or to Alice. Food is the currency of trade for this starving foursome. Bob is willing to pay Carol 7 clams if she will accept his affection. David is even more keen and is willing to give Carol 9 clams if she will accept it. Both Bob and David prefer Carol to Alice (sorry Alice). To quantify this preference, David is willing to pay Alice only 2 clams for his affection. Bob is even more averse: he says that Alice would have to pay him for it. In fact, she'd have to pay him 3 clams for his affection. Carol and Alice, being proper young women, will accept affection from one and only one of the two guys. Between the two of them they have decided to share the clams equally between them and hence their objective is simply to maximize the total number of clams they will receive. Formulate this problem as a transportation problem. Solve it.

14.4 *Project Scheduling.* This problem deals with the creation of a project schedule; specifically, the project of building a house. The project has been divided into a set of jobs. The problem is to schedule the time at which each of these jobs should start and also to predict how long the project will take. Naturally, the objective is to complete the project as quickly as possible (time is money!). Over the duration of the project, some of the jobs can be

done concurrently. But, as the following table shows, certain jobs definitely can't start until others are completed.

Job	Duration (weeks)	Must be Preceeded by
0. Sign Contract with Buyer	0	–
1. Framing	2	0
2. Roofing	1	1
3. Siding	3	1
4. Windows	2.5	3
5. Plumbing	1.5	3
6. Electical	2	2,4
7. Inside Finishing	4	5,6
8. Outside Painting	3	2,4
9. Complete the Sale to Buyer	0	7,8

One possible schedule is the following:

Job	Start Time
0. Sign Contract with Buyer	0
1. Framing	1
2. Roofing	4
3. Siding	6
4. Windows	10
5. Plumbing	9
6. Electical	13
7. Inside Finishing	16
8. Outside Painting	14
9. Complete the Sale to Buyer	21

With this schedule, the project duration is 21 weeks (*the difference between the start times of jobs 9 and 0*).

To model the problem as a linear program, introduce the following decision variables:

$$t_j = \text{the start time of job } j.$$

(a) Write an expression for the objective function, which is to minimize the project duration.
(b) For each job j, write a constraint for each job i that must preceed j; the constraint should ensure that job j doesn't start until job i is finished. These are called *precedence constraints*.

14.5 *Continuation.* This problem generalizes the specific example of the previous problem. A project consists of a set of jobs $\mathcal{J}$. For each job $j \in \mathcal{J}$ there is a certain set $\mathcal{P}_j$ of other jobs that must be completed before job j can be started. (This is called the set of *predecessors* of job j.) One of the jobs, say s, is the starting job; it has no predecessors. Another job, say t, is the final (or terminal) job; it is not the predecessor of any other job. The time it will take to do job j is denoted d_j (the *duration* of the job).

The problem is to decide what time each job should begin so that no job begins before its predecessors are finished, and the duration of the entire project is minimized. Using the notations introduced above, write out a complete description of this linear programming problem.

14.6 *Continuation.* Let x_{ij} denote the dual variable corresponding to the precedence constraint that ensures job j doesn't start until job i finishes.
(a) Write out the dual to the specific linear program in Problem 14.4.
(b) Write out the dual to the general linear program in Problem 14.5.
(c) Describe how the optimal value of the dual variable x_{ij} can be interpreted.

14.7 *Continuation.* The project scheduling problem can be represented on a directed graph with arc weights as follows. The nodes of the graph correspond to the jobs. The arcs correspond to the precedence relations. That is, if job i must be completed before job j, then there is an arc pointing from node i to node j. The weight on this arc is d_i.
(a) Draw the directed graph associated with the example in Problem 14.4, being sure to label the nodes and write the weights beside the arcs.
(b) Return to the formulation of the dual from Problem 14.6(a). Give an interpretation of that dual problem in terms of the directed graph drawn in Part (a).
(c) Explain why there is always an optimal solution to the dual problem in which each variable x_{ij} is either 0 or 1.
(d) Write out the complementary slackness condition corresponding to dual variable x_{26}.

(e) Describe the dual problem in the language of the original project scheduling model.

14.8 *Continuation.* Here is an algorithm for computing optimal start times t_j:

1. List the jobs so that the predecessors of each job come before it in the list.
2. Put $t_0 = 0$.
3. Go down the list of jobs and for job j put $t_j = \max\{t_i + d_i : i \text{ is a predecessor of } j\}$.

(a) Apply this algorithm to the specific instance from Problem 14.4. What are the start times of each of the jobs? What is the project duration?

(b) Prove that the solution found in Part (a) is optimal by exhibiting a corresponding dual solution and checking the usual conditions for optimality (*Hint: The complementary slackness conditions may help you find a dual solution.*).

14.9 *Currency Arbitrage.* Consider the world's currency market. Given two currencies, say the Japanese Yen and the US Dollar, there is an exchange rate between them (currently about 110 Yen to the Dollar). It is always true that, if you convert money from one currency to another and then back, you will end up with less than you started with. That is, the product of the exchange rates between any pair of countries is always less than one. However, it sometimes happens that a longer chain of conversions results in a gain. Such a lucky situation is called an *arbitrage*. One can use a linear programming model to find such situations when they exist.

Consider the following table of exchange rates (which is actual data from the Wall Street Journal on Nov 10, 1996):

```
param rate:
      USD     Yen     Mark    Franc   :=
USD   .       111.52  1.4987  5.0852
Yen   .008966 .       .013493 .045593
Mark  .6659   73.964  .       3.3823
Franc .1966   21.933  .29507  .
;
```

It is not obvious, but the USD→Yen→Mark→USD conversion actually makes $0.002 on each initial dollar.

To look for arbitrage possibilities, one can make a *generalized network model*, which is a network flow model with the unusual twist that a unit of flow that leaves one node arrives at the next node multiplied by a scale factor—in our example, the currency conversion rate. For us, each currency is represented by a node. There is an arc from each node to every other node. A flow of one unit out of one node becomes a flow of a different magnitude at the head node. For example, one dollar flowing out of the USD node arrives at the Franc node as 5.0852 Francs.

Let x_{ij} denote the flow from node (i.e. currency) i to node j. This flow is measured in the currency of node i.

One node is special; it is the *home* node, say the US Dollars (USD) node. At all other nodes, there must be flow balance.

(a) Write down the flow balance constraints at the 3 non-home nodes (Franc, Yen, and Mark).

At the home node, we assume that there is a supply of one unit (to get things started). Furthermore, at this node, flow balance will not be satisfied. Instead one expects a net inflow. If it is possible to make this inflow greater than one, then an arbitrage has been found. Let f be a variable that represents this inflow.

(b) Using variable f to represent net inflow to the home node, write a flow balance equation for the home node.

Of course, the primal objective is to maximize f.

(c) Using y_i to represent the dual variable associated with the primal constraint for currency i, write down the dual linear program. (Regard the primal variable f as a free variable.)

Now consider the general case, which might involve hundreds of currencies worldwide.

(d) Write down the model mathematically using x_{ij} for the flow leaving node i heading for node j (measured in the currency of node i), r_{ij} for the exchange rate when converting from currency i to currency j, and f for the net inflow at the home node i^*.
(e) Write down the dual problem.

(f) Can you give an interpretation for the dual variables? Hint: It might be helpful to think about the case where $r_{ji} = 1/r_{ij}$ for all i, j.

(g) Comment on the conditions under which your model will be unbounded and/or infeasible.

Notes

The Hitchcock problem was introduced by Hitchcock (1941). Dijkstra's algorithm was discovered by Dijkstra (1959).

The Max-Flow Min-Cut Theorem was proved independently by Elias et al. (1956), by Ford & Fulkerson (1956) and, in the restricted case where the upper bounds are all integers, by Kotzig (1956). Fulkerson & Dantzig (1955) also proved the Max-Flow Min-Cut Theorem. Their proof uses duality, which is particularly relevant to this chapter.

The classic references for dynamic programming are the books by Bellman (1957) and Howard (1960). Further discussion of label-setting and label-correcting algorithms can be found in the book by Ahuja et al. (1993).

CHAPTER 15

Structural Optimization

This final chapter on network-type problems deals with finding the best design of a structure to support a specified load at a fixed set of points. The *topology* of the problem is described by a graph where each node represents a *joint* in the structure and each arc represents a potential *member*.[1] We shall formulate this problem as a linear programming problem whose solution determines which of the potential members to include in the structure and how thick each included member must be to handle the load. The optimization criterion is to find a minimal weight structure. As we shall see, the problem bears a striking resemblance to the minimum-cost network flow problem that we studied in Chapter 13.

1. An Example

We begin with an example. Consider the graph shown in Figure 15.1. This graph represents a structure consisting of five joints and eight possible members connecting the joints. The five joints and their coordinates are given as follows:

Joint	Coordinates
1	(0.0 , 0.0)
2	(6.0 , 0.0)
3	(0.0 , 8.0)
4	(6.0 , 8.0)
5	(3.0 , 12.0)

Since joints are analogous to nodes in a network, we shall denote the set of joints by $\mathcal{N}$ and denote by m the number of joints. Also, since members are analogous to arcs in network flows, we shall denote the set of them by $\mathcal{A}$. For the structure shown in Figure 15.1, the set of members is

$$\mathcal{A} = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},\{3,5\},\{4,5\}\}\,.$$

[1] Civil engineers refer to beams as members.

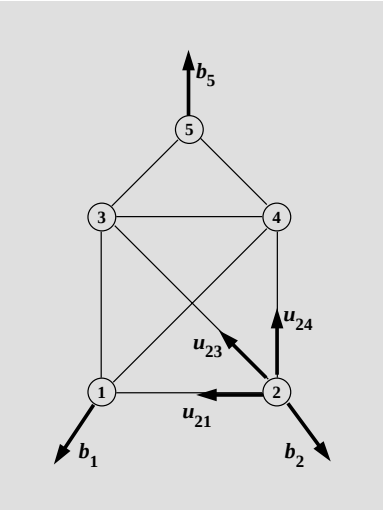


FIGURE 15.1. Sample topology for a two-dimensional structure.

Note that we enclosed the pairs of endjoints in braces to emphasize that their order is irrelevant. For example, $\{2, 3\}$ and $\{3, 2\}$ refer to one and the same member spanning between joints 2 and 3. In network flows, the graphs we considered were directed graphs. Here, they are undirected. Also, the graphs here are embedded in a d-dimensional Euclidean space (meaning that every node comes with a set of coordinates indicating its location in d-dimensional space). No such embedding was imposed before in our study of network flows, even though real-world network flow problems often possess such an embedding.

Following the standard convention of using braces to denote sets, we ought to let $x_{\{i,j\}}$ denote the force exerted by member $\{i, j\}$ on its endjoints. But the braces are cumbersome. Hence, we shall write this force simply as x_{ij}, with the understanding that x_{ij} and x_{ji} denote one and the same variable.

We shall assume that a positive force represents tension in the member (i.e., the member is pulling "in" on its two endjoints) and that a negative value represents compression (i.e., the member is pushing "out" on its two endjoints).

If the structure is to be in equilibrium (i.e., not accelerating in some direction), then forces must be balanced at each joint. Of course, we assume that there may be a nonzero external load at each joint (this is the analogue of the external supply/demand in the minimum-cost network flow problem). Hence, for each node i, let b_i denote the externally applied load. Note that each b_i is a vector whose dimension equals the dimension of the space in which the structure lies. For our example, this dimension is 2. In general, we shall denote the spatial dimension by d.

Force balance imposes a number of constraints on the member forces. For example, the force balance equations for joint 2 can be written as follows:

$$x_{12}\begin{bmatrix}-1\\0\end{bmatrix}+x_{23}\begin{bmatrix}-0.6\\0.8\end{bmatrix}+x_{24}\begin{bmatrix}0\\1\end{bmatrix}=-\begin{bmatrix}b_2^1\\b_2^2\end{bmatrix},$$

where b_2^1 and b_2^2 denote the components of b_2. Note that the three vectors appearing on the left are unit vectors pointing out from joint 2 along each of the corresponding members.

2. Incidence Matrices

If, for each joint i, we let p_i denote its position vector, then the unit vectors pointing along the arcs can be written as follows:

$$u_{ij}=\frac{p_j-p_i}{\|p_j-p_i\|},\qquad \{i,j\}\in\mathcal{A}.$$

It is important to note that $u_{ji}=-u_{ij}$, since the first vector points from j towards i, whereas the second points from i towards j. In terms of these notations, the force balance equations can be expressed succinctly as

$$\sum_{\substack{j:\\ \{i,j\}\in\mathcal{A}}} u_{ij}x_{ij}=-b_i\qquad i=1,2,\ldots,m. \tag{15.1}$$

These equations can be written in matrix form as

$$Ax=-b \tag{15.2}$$

where x denotes the vector consisting of the member forces, b denotes the vector whose elements are the applied load vectors, and A is a matrix containing the unit vectors pointing along the appropriate arcs. For our example, we have

$$x^T=\begin{bmatrix}x_{12} & x_{13} & x_{14} & x_{23} & x_{24} & x_{34} & x_{35} & x_{45}\end{bmatrix}$$

$$A=\begin{array}{c}\\1\\2\\3\\4\\5\end{array}\begin{bmatrix}
\begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}0\\1\end{bmatrix} & \begin{bmatrix}.6\\.8\end{bmatrix} & & & & & \\
\begin{bmatrix}-1\\0\end{bmatrix} & & & \begin{bmatrix}-.6\\.8\end{bmatrix} & \begin{bmatrix}0\\1\end{bmatrix} & & & \\
& \begin{bmatrix}0\\-1\end{bmatrix} & & \begin{bmatrix}.6\\-.8\end{bmatrix} & & \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}.6\\.8\end{bmatrix} & \\
& & \begin{bmatrix}-.6\\-.8\end{bmatrix} & & \begin{bmatrix}0\\-1\end{bmatrix} & \begin{bmatrix}-1\\0\end{bmatrix} & & \begin{bmatrix}-.6\\.8\end{bmatrix} \\
& & & & & & \begin{bmatrix}-.6\\-.8\end{bmatrix} & \begin{bmatrix}.6\\-.8\end{bmatrix}
\end{bmatrix},\qquad b=\begin{bmatrix}b_1^1\\b_1^2\\b_2^1\\b_2^2\\b_3^1\\b_3^2\\b_4^1\\b_4^2\\b_5^1\\b_5^2\end{bmatrix}.$$

Note that we have written A as a matrix of 2-vectors by putting "inner" brackets around appropriate pairs of entries. These inner brackets could of course be dropped—they are included simply to show how the constraints match up with (15.1).

In network flows, an incidence matrix is characterized by the property that every column of the matrix has exactly two nonzero entries, one $+1$ and one -1. Here, the matrix A is characterized by the property that, when viewed as a matrix of d-vectors, every column has two nonzero entries that are unit vectors pointing in opposite directions from each other. Generically, matrix A can be written as follows:

$$A = \begin{array}{c} \\ i \to \\ \\ j \to \\ \\ \end{array} \begin{array}{c} \{i,j\} \\ \downarrow \\ \left[\begin{array}{c} \\ u_{ij} \\ \\ u_{ji} \\ \\ \end{array}\right. \end{array}$$

where, for each $\{i,j\} \in \mathcal{A}$,

$$u_{ij}, u_{ji} \in \mathbb{R}^d,$$

$$u_{ij} = -u_{ji},$$

and

$$\|u_{ij}\| = \|u_{ji}\| = 1.$$

By analogy with network flows, the matrix A is called an *incidence matrix*. This definition is a strict generalization of the definition we had before, since, for $d = 1$, the current notion reduces to the network flows notion. Incidence matrices for network flows enjoy many useful properties. In the following sections, we shall investigate the extent to which these properties carry over to our generalized notion.

3. Stability

Recall that for network flows, the sum of the rows of the incidence matrix vanishes and that if the network is connected, this is the only redundancy. For $d > 1$, the situation is similar. Clearly, the sum of the rows vanishes. But is this the only redundancy? To answer this question, we need to look for nonzero row vectors y^T for which $y^T A = 0$. The set of all such row vectors is a subspace of the set of all row vectors. Our aim is to find a basis for this subspace and, in particular, to identify its dimension. To this end, first write y in component form as $y^T = [y_1^T \cdots y_m^T]$ where each of the entries y_i, $i = 1, 2, \ldots, m$, are d-vectors (transposed to make them into row vectors).

Multiplying this row vector against each column of A, we see that $y^T A = 0$ if and only if

$$y_i^T u_{ij} + y_j^T u_{ji} = 0, \qquad \text{for all } \{i,j\} \in \mathcal{A}. \tag{15.3}$$

There are many choices of y that yield a zero row combination. For example, we can take any vector $v \in \mathbb{R}^d$ and put

$$y_i = v, \qquad \text{for every } i \in \mathcal{N}.$$

Substituting this choice of y_i's into the left-hand side of (15.3), we get

$$y_i^T u_{ij} + y_j^T u_{ji} = v^T u_{ij} + v^T u_{ji} = v^T u_{ij} - v^T u_{ij} = 0.$$

This set of choices shows that the subspace is at least d-dimensional.

But there are more! They are defined in terms of skew symmetric matrices. A matrix R is called *skew symmetric* if $R^T = -R$. A simple but important property of skew symmetric matrices is that, for every vector ξ,

$$\xi^T R \xi = 0 \tag{15.4}$$

(see Exercise 15.1). We shall make use of this property shortly. Now, to give more choices of y, let R be a $d \times d$ skew symmetric matrix and put

$$y_i = R p_i, \qquad \text{for every } i \in \mathcal{N}$$

(recall that p_i denotes the position vector for joint i). We need to check (15.3). Substituting this definition of the y's into the left-hand side in (15.3), we see that

$$\begin{aligned} y_i^T u_{ij} + y_j^T u_{ji} &= p_i^T R^T u_{ij} + p_j^T R^T u_{ji} \\ &= -p_i^T R u_{ij} - p_j^T R u_{ji} \\ &= (p_j - p_i)^T R u_{ij}. \end{aligned}$$

Now substituting in the definition of u_{ij}, we get

$$(p_j - p_i)^T R u_{ij} = \frac{(p_j - p_i)^T R (p_j - p_i)}{\|p_j - p_i\|}.$$

Finally, by putting $\xi = p_j - p_i$ and using property (15.4) of skew symmetric matrices, we see that the numerator on the right vanishes. Hence, (15.3) holds.

How many redundancies have we found? For $d = 2$, there are two independent v-type redundancies and one more R-type. The following two vectors and a matrix can be taken as a basis for these redundancies

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

For $d = 3$, there are three independent v-type redundancies and three R-type. Here are three vectors and three matrices that can be taken as a basis for the space of redundancies:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{15.5}$$

In general, there are $d + d(d-1)/2 = d(d+1)/2$ independent redundancies. There could be more. But just as for network flows, where we showed that there is one redundancy if and and only if the network is connected, further redundancies represent a defect in the underlying graph structure. In fact, we say that the graph is *stable* if the rank of the incidence matrix A is exactly $md - d(d+1)/2$, that is, if and only if the above redundancies account for the entire rank deficiency of A.

4. Conservation Laws

Recall that for network flows, not all choices of supplies/demands yield feasible flows. For connected networks, it is necessary and sufficient that the total supply equals the total demand. The situation is similar here. The analogous question is: which external loads give rise to solutions to (15.2)? We have already identified several row vectors y^T for which $y^T A = 0$. Clearly, in order to have a solution to (15.2), it is necessary that $y^T b = 0$ for all these row vectors. In particular for every $v \in \mathbb{R}^d$, we see that b must satisfy the following condition:

$$\sum_i v^T b_i = 0.$$

Bringing the sum inside of the product, we get

$$v^T \left(\sum_i b_i \right) = 0.$$

Since this must hold for every d-vector v, it follows that

$$\sum_i b_i = 0.$$

This condition has a simple physical interpretation: the loads, taken in total, must balance.

What about the choices of y^T arising from skew symmetric matrices? We shall show that these choices impose the conditions necessary to prevent the structure from

spinning around some axis of rotation. To show that this is so, let us first consider the two-dimensional case. For every 2×2 skew symmetric matrix R, the load vectors b_i, $i \in \mathcal{N}$, must satisfy

$$\sum_i (Rp_i)^T b_i = 0. \tag{15.6}$$

This expression is a sum of terms of the form $(Rp)^T b$, where p is the position vector of a point and b is a force applied at this point. We claim that $(Rp)^T b$ is precisely the torque about the origin created by applying force b at location p. To see this connection between the algebraic expression and its physical interpretation, first decompose p into the product of its length r times a unit vector v pointing in the same direction and rewrite the algebraic expression as

$$(Rp)^T b = r(Rv)^T b.$$

Now, without loss of generality, we may assume that R is the "basis" matrix for the space of skew symmetric matrices,

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

This matrix has the additional property that its two columns are unit vectors that are orthogonal to each other. That is, $R^T R = I$. Hence,

$$\|Rv\|^2 = \|v\|^2 = 1.$$

Furthermore, property (15.4) tells us that Rv is orthogonal to v. Therefore, the product $(Rv)^T b$ is the length of the projection of b in the direction of Rv, and so $r(Rv)^T b$ is the distance from the origin (of the coordinate system) to p, which is called the *moment arm*, times the component of the force that is orthogonal to the moment arm in the direction of Rv (see Figure 15.2). This interpretation for each summand in (15.6) shows that it is exactly the torque around the rotation axis passing through the origin of the coordinate system caused by the force b_i applied to joint i. In $d = 2$, there is only one rotation around the origin. This fact corresponds to the fact that the dimension of the space of skew symmetric matrices in two dimensions is 1. Also, stipulating that the total torque about the origin vanishes implies that the total torque around any point other point also vanishes—see Exercise 15.4.

The situation for $d > 2$, in particular for $d = 3$, is slightly more complicated. Algebraically, the complications arise because the basic skew symmetric matrices no longer satisfy $R^T R = I$. Physically, the complications stem from the fact that in two dimensions rotation takes place around a point, whereas in three dimensions it takes place around an axis. We shall explain how to resolve the complications for $d = 3$. The extension to higher dimensions is straightforward (and perhaps not so important). The basic conclusion that we wish to derive is the same, namely that for basic skew symmetric matrices, the expression $(Rp)^T b$ represents the torque generated

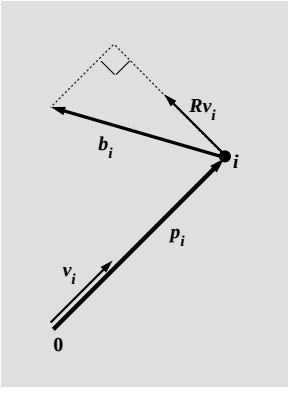


FIGURE 15.2. The ith summand in (15.6) is the length of p_i times the length of the projection of b_i onto the direction given by Rv_i. This is precisely the torque around an axis at 0 caused by the force b_i applied at joint i.

by applying a force b at point p. Recall that there are just three basic skew symmetric matrices, and they are given by (15.5). To be specific, let us just study the first one:

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

This matrix can be decomposed into the product of two matrices:

$$R = UP$$

where

$$U = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The matrix U has the property that R had before, namely,

$$U^T U = I.$$

Such matrices are called *unitary*. The matrix P is a projection matrix. If we let

$$q = Pp,$$
$$v = \frac{q}{\|q\|},$$

and

$$r = \|q\|,$$

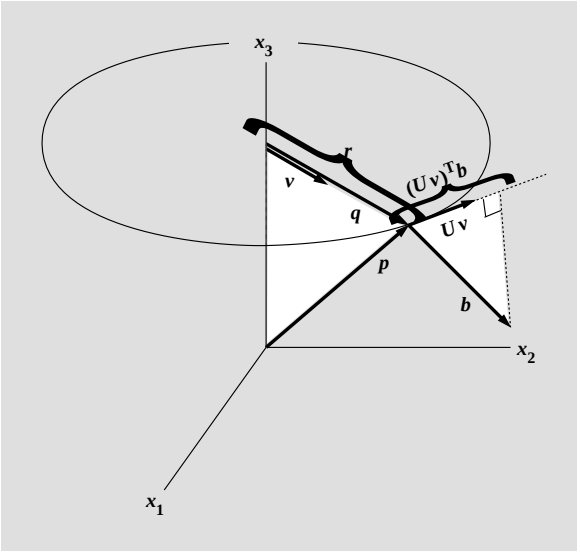


FIGURE 15.3. The decomposition of $(Rp)^T b$ into the product of a moment arm r times the component of b in the direction Uv shows that it is precisely the torque around the third axis.

then we can rewrite $(Rp)^T b$ as

$$(Rp)^T b = r(Uv)^T b.$$

Since v is a unit vector and U is unitary, it follows that Uv is a unit vector. Hence, $(Uv)^T b$ represents the scalar projection of b onto the direction determined by Uv. Also, it is easy to check that Uv is orthogonal to v. At this point, we can consult Figure 15.3 to see that r is the moment arm for the torque around the third coordinate axis and $(Uv)^T b$ is the component of force in the direction of rotation around this axis. Therefore, the product is precisely the torque around this axis. As we know, for $d = 3$, there are three independent axes of rotation, namely, *pitch*, *roll*, and *yaw*. These axes correspond to the three basis matrices for the space of 3×3 skew symmetric matrices (the one we have just studied corresponds to the yaw axis).

Finally, we note that (15.6) simply states that the total torque around each axis of rotation must vanish. This means that the forces cannot be chosen to make the system spin.

5. Minimum-Weight Structural Design

For a structure with m nodes, the system of force balance equations (15.2) has md equations. But, as we now know, if the structure is stable, there are exactly $d(d+1)/2$

redundant equations. That is, the rank of A is $md - d(d+1)/2$. Clearly, the structure must contain at least this many members. We say that the structure is a *truss* if it is stable and has exactly $md - d(d+1)/2$ members. In this case, the force balance equations have a unique solution (assuming, of course, that the total applied force and the total applied torque around each axis vanish). From an optimization point of view, trusses are not interesting because they leave nothing to optimize—one only needs to calculate.

To obtain an interesting optimization problem, we assume that the proposed structure has more members than the minimum required to form a truss. In this setting, we introduce an optimization criterion to pick that solution (whether a truss or otherwise) that minimizes the criterion. For us, we shall attempt to minimize total weight. To keep things simple, we assume that the weight of a member is directly proportional to its volume and that the constant of proportionality (the *density* of the material) is the same for each member. (These assumptions are purely for notational convenience—a real engineer would certainly include these constants and let them vary from one member to the next). Hence, it suffices to minimize the total volume. The volume of one member, say, $\{i,j\}$, is its length $l_{ij} = \|p_j - p_i\|$ times its cross-sectional area. Again, to keep things as simple as possible, we assume that the cross-sectional area must be proportional to the tension/compression carried by the member (members carrying big loads must be "fat"—otherwise they might break). Let's set the constant of proportionality arbitrarily to one. Then the function that we should minimize is just the sum over all members of $l_{ij}|x_{ij}|$. Hence, our optimization problem can be written as follows:

$$\begin{aligned} &\text{minimize} && \sum_{\{i,j\}\in\mathcal{A}} l_{ij}|x_{ij}| \\ &\text{subject to} && \sum_{\substack{j:\\ \{i,j\}\in\mathcal{A}}} u_{ij}x_{ij} = -b_i \qquad i = 1,2,\ldots,m. \end{aligned}$$

This problem is not a linear programming problem: the constraints are linear, but the objective function involves the absolute value of each variable. We can, however, convert this problem to a linear programming problem with the following trick. For each $\{i,j\} \in \mathcal{A}$, write x_{ij} as the difference between two nonnegative variables:

$$x_{ij} = x_{ij}^+ - x_{ij}^-, \qquad x_{ij}^+,\ x_{ij}^- \geq 0.$$

Think of x_{ij}^+ as the tension part of x_{ij} and x_{ij}^- as the compression part. The absolute value can then be modeled as the sum of these components

$$|x_{ij}| = x_{ij}^+ + x_{ij}^-.$$

We allow both components to be positive at the same time, but no minimum-weight solution will have any member with both components positive, since if there were

such a member, the tension component and the compression component could be decreased simultaneously at the same rate without changing the force balance equations but reducing the weight. This reduction contradicts the minimum-weight assumption.

We can now state the linear programming formulation of the minimum weight structural design problem as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{\{i,j\}\in\mathcal{A}} (l_{ij}x_{ij}^{+} + l_{ij}x_{ij}^{-}) \\ \text{subject to} \quad & \sum_{\substack{j: \\ \{i,j\}\in\mathcal{A}}} (u_{ij}x_{ij}^{+} - u_{ij}x_{ij}^{-}) = -b_i \qquad i = 1, 2, \ldots, m \\ & x_{ij}^{+},\ x_{ij}^{-} \geq 0 \qquad \{i,j\} \in \mathcal{A}. \end{aligned}$$

In terms of the incidence matrix, each column must now be written down twice, once as before and once as the negative of before.

6. Anchors Away

So far we have considered structures that are free floating in the sense that even though loads are applied at various joints, we have not assumed that any of the joints are anchored to a large object such as the Earth. This setup is fine for structures intended for a rocket or a space station, but for Earth-bound applications it is generally desired to anchor some joints. It is trivial to modify the formulation we have already given to cover the situation where some of the joints are anchored. Indeed, the d force balance equations associated with an anchored joint are simply dropped as constraints, since the Earth supplies whatever counterbalancing force is needed. Of course, one can consider dropping only some of the d force balance equations associated with a particular joint. In this case, the physical interpretation is quite simple. For example, in two dimensions it simply means that the joint is allowed to roll on a track that is aligned with one of the coordinate directions but is not allowed to move off the track.

If enough "independent" constraints are dropped (at least three in two dimensions and at least six in three dimensions), then there are no longer any limitations on the applied loads—the structure will be sufficiently well anchored so that the Earth will apply whatever forces are needed to prevent the structure from moving. This is the most typical scenario under which these problems are solved. It makes setting up the problem much easier, since one no longer needs to worry about supplying loads that can't be balanced.

We end this chapter with one realistic example. Suppose the need exists to design a bracket to support a hanging load at a fixed distance from a wall. This bracket will be molded out of plastic, which means that the problem of finding an optimal design belongs to the realm of continuum mechanics. However, we can get an idea of the optimal shape by modeling the problem discretely (don't tell anyone). That is, we define a lattice of joints as shown in Figure 15.4 and introduce a set of members

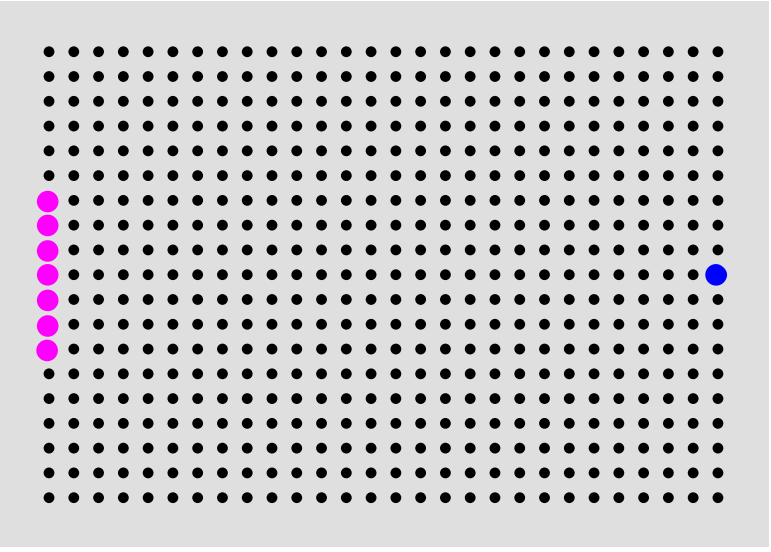


FIGURE 15.4. The set of joints used for the discrete approximation to the bracket design problem. The highlighted joints on the left are anchored to the wall, and the highlighted joint on the right must support the hanging load.

from which the bracket can be constructed. Each joint has members connecting it to several nearby joints. Figure 15.5 shows the members connected to one specific joint. Each joint in the structure has this connection "topology" with, of course, the understanding that joints close to the boundary do not have any member for which the intended connecting joint does not exist. The highlighted joints on the left side in Figure 15.4 are the anchored joints, and the highlighted joint on the right side is the joint to which the hanging load is applied (by "hanging," we mean that the applied load points downward). The optimal solution is shown in Figure 15.6. The thickness of each member is drawn in proportion to the square root of the tension/compression in the member (since if the structure actually exists in three dimensions, the diameter of a member would be proportional to the square root of the cross-sectional area). Also, those members under compression are drawn in dark gray, whereas those under tension are drawn in light gray. Note that the compression members appear to cross the tension members at right angles. These curves are called *principle stresses*. It is a fundamental result in continuum mechanics that the principle tension stresses cross the principle compression stresses at right angles. We have discovered this result using optimization.

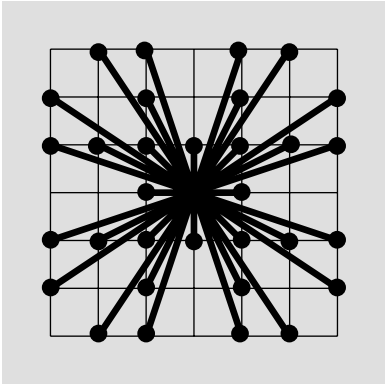


FIGURE 15.5. The members connected to a single interior joint.

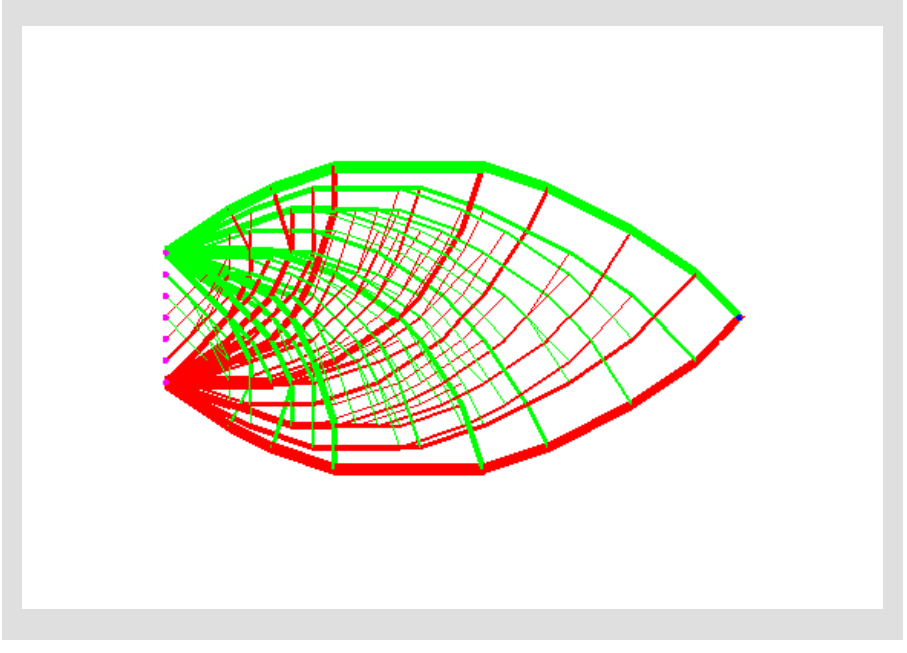


FIGURE 15.6. The minimum weight bracket.

Most nonexperts find the solution to this problem to be quite surprising, since it covers such a large area. Yet it is indeed optimal. Also, one can see that the continuum solution should be roughly in the shape of a leaf.

Exercises

15.1 Show that a matrix R is skew symmetric if and only if

$$\xi^T R\xi = 0, \qquad \text{for every vector } \xi.$$

15.2 Which of the structures shown in Figure 15.7 is stable? (Note: each structure is shown embedded in a convenient coordinate system.)

15.3 Which of the structures shown in Figure 15.7 is a truss?

15.4 Assuming that the total applied force vanishes, show that total torque is translation invariant. That is, for any vector $\xi \in \mathbb{R}^d$,

$$\sum_i (R(p_i - \xi))^T b_i = \sum_i (Rp_i)^T b_i.$$

15.5 In 3-dimensions there are 5 regular (Platonic) solids. They are shown in Figure 15.8 and have the following number of vertices and edges:

	vertices	edges
tetrahedron	4	6
cube	8	12
octahedron	6	12
dodecahedron	20	30
icosahedron	12	30

If one were to construct pin-jointed wire-frame models of these solids, which ones would be stable?

Notes

Structural optimization has its roots in Michell (1904). The first paper in which truss design was formulated as a linear programming problem is Dorn et al. (1964). A few general references on the subject include Hemp (1973), Rozvany (1989), Bendsøe et al. (1994), and Recski (1989).

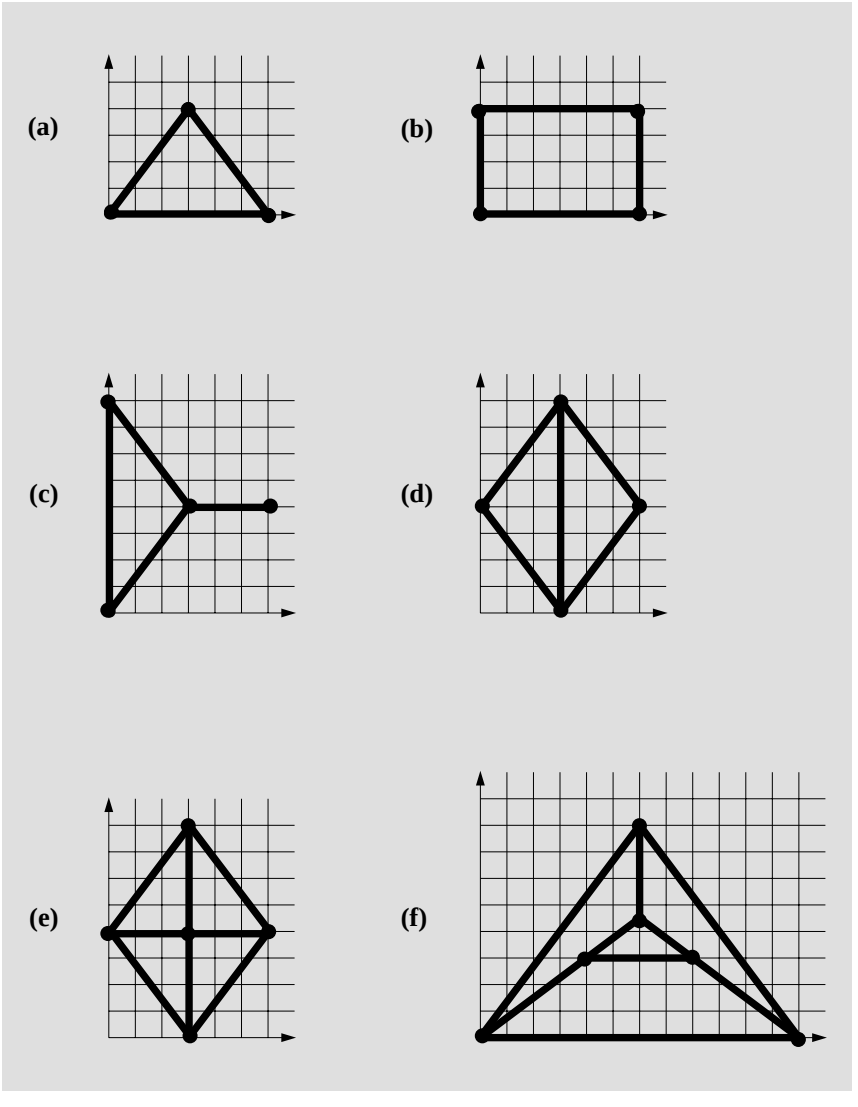


FIGURE 15.7. Structures for Exercises 15.2 and 15.3.

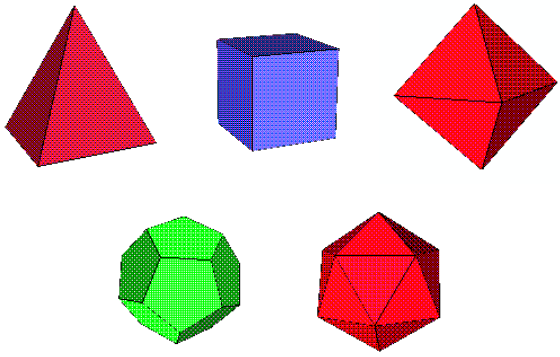


FIGURE 15.8. The five regular solids.

Part 3

Interior-Point Methods

There is, I believe, in every disposition a tendency to some particular evil — a natural defect, which not even the best education can overcome. — J. Austen

CHAPTER 16

The Central Path

In this chapter, we begin our study of an alternative to the simplex method for solving linear programming problems. The algorithm we are going to introduce is called a *path-following method*. It belongs to a class of methods called *interior-point methods*. The path-following method seems to be the simplest and most natural of all the methods in this class, so in this book we focus primarily on it. Before we can introduce this method, we must define the path that appears in the name of the method. This path is called the *central path* and is the subject of this chapter. Before discussing the central path, we must lay some groundwork by analyzing a nonlinear problem, called the *barrier problem*, associated with the linear programming problem that we wish to solve.

Warning: Nonstandard Notation Ahead

Starting with this chapter, given a lower-case letter denoting a vector quantity, we shall use the upper-case form of the same letter to denote the diagonal matrix whose diagonal entries are those of the corresponding vector. For example,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \Longrightarrow \quad X = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{bmatrix}.$$

This notation is nonstandard in mathematics at large, but has achieved a certain amount of acceptance in the interior-point-methods community.

1. The Barrier Problem

In this chapter, we consider the linear programming problem expressed, as usual, with inequality constraints and nonnegative variables:

$$\begin{aligned} &\text{maximize} && c^T x \\ &\text{subject to} && Ax \leq b \\ & && x \geq 0. \end{aligned}$$

The corresponding dual problem is

$$\begin{aligned} &\text{minimize } && b^T y \\ &\text{subject to } && A^T y \geq c \\ & && y \geq 0. \end{aligned}$$

As usual, we add slack variables to convert both problems to equality form:

$$\begin{aligned} &\text{maximize } && c^T x \\ &\text{subject to } && Ax + w = b \\ & && x, w \geq 0 \end{aligned} \tag{16.1}$$

and

$$\begin{aligned} &\text{minimize } && b^T y \\ &\text{subject to } && A^T y - z = c \\ & && y, z \geq 0. \end{aligned}$$

Given a constrained maximization problem where some of the constraints are inequalities (such as our primal linear programming problem), one can consider replacing any inequality constraint with an extra term in the objective function. For example, in (16.1) we could remove the constraint that a specific variable, say, x_j, is nonnegative by adding to the objective function a term that is negative infinity when x_j is negative and is zero otherwise. This reformulation doesn't seem to be particularly helpful, since this new objective function has an abrupt discontinuity that, for example, prevents us from using calculus to study it. However, suppose we replace this discontinuous function with another function that is negative infinity when x_j is negative but is finite for x_j positive and approaches negative infinity as x_j approaches zero. In some sense this smooths out the discontinuity and perhaps improves our ability to apply calculus to its study. The simplest such function is the logarithm. Hence, for each variable, we introduce a new term in the objective function that is just a constant times the logarithm of the variable:

$$\begin{aligned} &\text{maximize } && c^T x + \mu \textstyle\sum_j \log x_j + \mu \sum_i \log w_i \\ &\text{subject to } && Ax + w = b. \end{aligned} \tag{16.2}$$

This problem, while not equivalent to our original problem, seems not too different either. In fact, as the parameter μ, which we assume to be positive, gets small, it appears that (16.2) becomes a better and better stand-in for (16.1). Problem (16.2) is called the *barrier problem* associated with (16.1). Note that it is not really one problem, but rather a whole family of problems indexed by the parameter μ. Each of these problems is a nonlinear programming problem because the objective function is nonlinear. This nonlinear objective function is called a *barrier function* or, more specifically, a *logarithmic barrier function*.

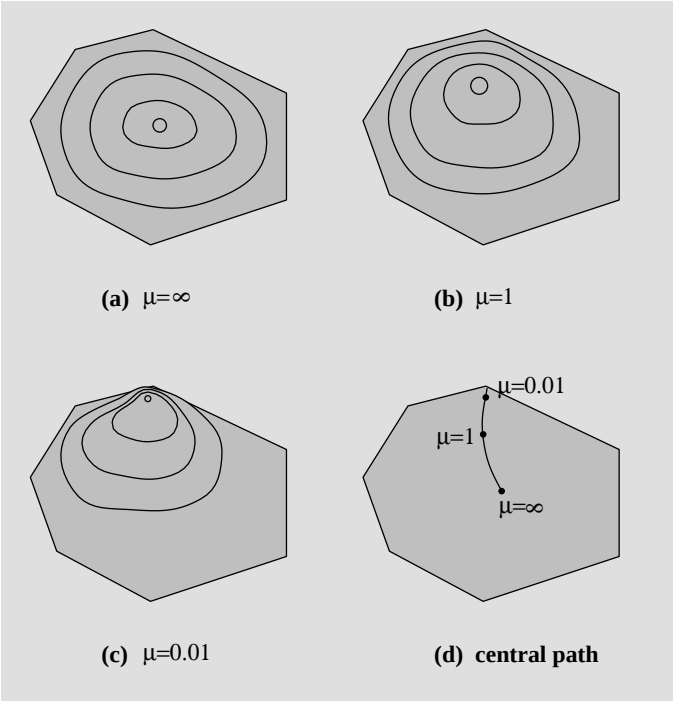


FIGURE 16.1. Parts (a) through (c) show level sets of the barrier function for three values of μ. For each value of μ, four level sets are shown. The maximum value of the barrier function is attained inside the innermost level set. The drawing in part (d) shows the central path.

It is instructive to have in mind a geometric picture of the barrier function. Recall that, for problems expressed in standard form, the set of feasible solutions is a polyhedron with each face being characterized by the property that one of the variables is zero. Hence, the barrier function is minus infinity on each face of the polyhedron. Furthermore, it is finite in the interior of the polyhedron, and it approaches minus infinity as the boundary is approached. Figure 16.1 shows some level sets for the barrier function for a specific problem and a few different choices of μ. Notice that, for each μ, the maximum is attained at an interior point, and as μ gets closer to zero this interior point moves closer to the optimal solution of the original linear programming problem (which is at the top vertex). Viewed as a function of μ, the set of optimal solutions to the barrier problems forms a path through the interior of the polyhedron of feasible

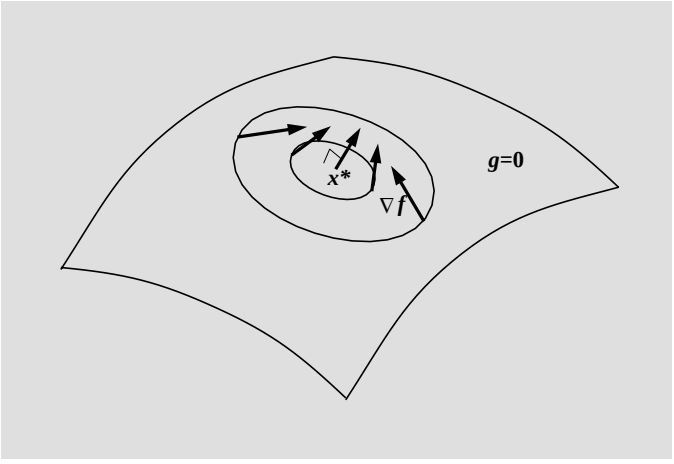


FIGURE 16.2. The concentric rings illustrate a few level sets of f. Clearly, at the optimal solution, x^*, the gradient must be perpendicular to the feasible set.

solutions. This path is called the *central path*. Our aim is to study this central path. To this end, we need to develop some machinery, referred to as *Lagrange multipliers*.

2. Lagrange Multipliers

We wish to discuss briefly the general problem of maximizing a function subject to one or more equality constraints. Here, the functions are permitted to be nonlinear, but are assumed to be smooth, say, twice differentiable.

For the moment, suppose that there is a single constraint equation so that the problem can be formally stated as

$$\begin{aligned} &\text{maximize } \ f(x) \\ &\text{subject to } \ g(x) = 0. \end{aligned}$$

In this case, the geometry behind the problem is compelling (see Figure 16.2). The gradient of f, denoted ∇f, is a vector that points in the direction of most rapid increase of f. For unconstrained optimization, we would simply set this vector equal to zero to determine the so-called *critical points* of f, and the maximum, if it exists, would have to be included in this set. However, given the constraint, $g(x) = 0$, it is no longer correct to look at points where the gradient vanishes. Instead, the gradient must be orthogonal to the set of feasible solutions $\{x : g(x) = 0\}$. Of course, at each point x in the feasible set, $\nabla g(x)$, is a vector that is orthogonal to the feasible set at this point x. Hence, our new requirement for a point x^* to be a critical point is that it is feasible

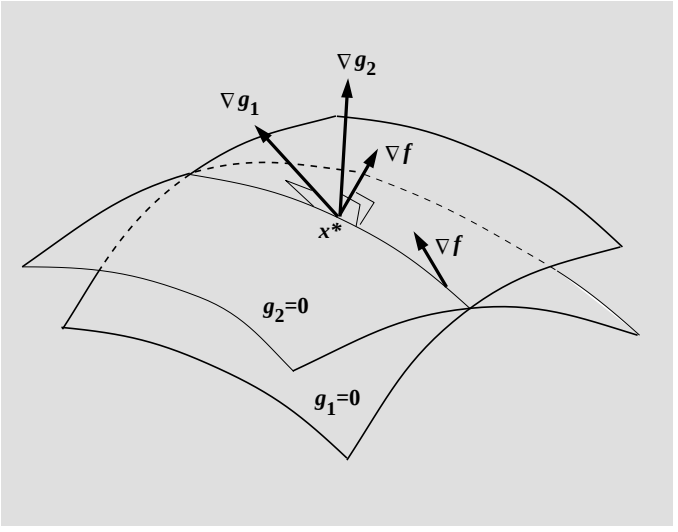


FIGURE 16.3. The feasible set is the curve formed by the intersection of $g_1 = 0$ and $g_2 = 0$. The point x^* is optimal, since the gradient of f at that point is perpendicular to the feasible set.

and that $\nabla f(x^*)$ be proportional to $\nabla g(x^*)$. Writing this out as a system of equations, we have

$$\begin{aligned} g(x^*) &= 0 \\ \nabla f(x^*) &= y \nabla g(x^*). \end{aligned}$$

Here, y is the proportionality constant. Note that it can be any real number, either positive, negative, or zero. This proportionality constant is called a *Lagrange multiplier*.

Now consider several constraints:

$$\begin{aligned} \text{maximize} \quad & f(x) \\ \text{subject to} \quad & g_1(x) = 0 \\ & g_2(x) = 0 \\ & \vdots \\ & g_m(x) = 0. \end{aligned}$$

In this case, the feasible region is the intersection of m hypersurfaces (see Figure 16.3). The space orthogonal to the feasible set at a point x is no longer a one-dimensional set determined by a single gradient, but is instead a higher-dimensional space (typically m), given by the span of the gradients. Hence, we require that $\nabla f(x^*)$ lie in this span.

This yields the following set of equations for a critical point:

$$g(x^*) = 0$$
$$\nabla f(x^*) = \sum_{i=1}^{m} y_i \nabla g(x^*). \tag{16.3}$$

The derivation of these equations has been entirely geometric, but there is also a simple algebraic formalism that yields the same equations. The idea is to introduce the so-called *Lagrangian* function

$$L(x, y) = f(x) - \sum_i y_i g_i(x)$$

and to look for its critical points over both x and y. Since this is now an unconstrained optimization problem, the critical points are determined by simply setting all the first derivatives to zero:

$$\frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j} - \sum_i y_i \frac{\partial g_i}{\partial x_j} \quad = 0, \qquad j = 1, 2, \ldots, n,$$
$$\frac{\partial L}{\partial y_i} = -g_i \qquad\qquad\qquad\quad = 0, \qquad i = 1, 2, \ldots, m.$$

Writing these equations in vector notation, we see that they are exactly the same as those derived using the geometric approach. These equations are usually referred to as the *first-order optimality conditions*.

Determining whether a solution to the first-order optimality conditions is indeed a global maximum as desired can be difficult. However, if the constraints are all linear, the first step (which is often sufficient) is to look at the matrix of second derivatives:

$$Hf(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right].$$

This matrix is called the *Hessian* of f at x. We have

THEOREM 16.1. *If the constraints are linear, a critical point x^* is a local maximum if*

$$\xi^T Hf(x^*)\xi < 0 \tag{16.4}$$

for each $\xi \neq 0$ satisfying

$$\xi^T \nabla g_i(x^*) = 0, \qquad i = 1, 2, \ldots, m. \tag{16.5}$$

PROOF. We start with the two-term Taylor series expansion of f about x^*:

$$f(x^* + \xi) = f(x^*) + \nabla f(x^*)^T \xi + \frac{1}{2} \xi^T Hf(x^*)\xi + o(\|\xi\|^2).$$

The vector ξ represents a displacement from the current point x^*. The only displacements that are relevant are those that lie in the feasible set. Hence, let ξ be a direction vector satisfying (16.5). From (16.3) and (16.5), we see that $\nabla f(x^*)^T\xi = 0$, and so

$$f(x^* + \xi) = f(x^*) + \frac{1}{2}\xi^T Hf(x^*)\xi + o(\|\xi\|^2).$$

Employing (16.4) finishes the proof. □

It is worth remarking that if (16.4) is satisfied not just at x^* but at all x, then x^* is a unique global maximum.

In the next section, we shall use Lagrange multipliers to study the central path defined by the barrier problem.

3. Lagrange Multipliers Applied to the Barrier Problem

In this section, we shall use the machinery of Lagrange multipliers to study the solution to the barrier problem. In particular, we will show that (subject to some mild assumptions) for each value of the barrier parameter μ, there is a unique solution to the barrier problem. We will also show that as μ tends to zero, the solution to the barrier problem tends to the solution to the original linear programming problem. In the course of our study, we will stumble naturally upon the central path for the dual problem. Taken together, the equations defining the primal and the dual central paths play an important role, and so we will introduce the notion of a primal–dual central path.

We begin by recalling the barrier problem:

$$\begin{aligned} &\text{maximize } c^T x + \mu \textstyle\sum_j \log x_j + \mu \sum_i \log w_i \\ &\text{subject to } Ax + w = b. \end{aligned}$$

This is an equality-constrained optimization problem, and so it is a problem to which we can apply the Lagrange multiplier tools developed in the previous section. The Lagrangian for this problem is

$$L(x, w, y) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i + y^T(b - Ax - w).$$

Taking derivatives with respect to each variable and setting them to zero, we get the first-order optimality conditions:

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= c_j + \mu\frac{1}{x_j} - \sum_i y_i a_{ij} &&= 0, \quad j = 1, 2, \ldots, n, \\ \frac{\partial L}{\partial w_i} &= \mu\frac{1}{w_i} - y_i &&= 0, \quad i = 1, 2, \ldots, m. \\ \frac{\partial L}{\partial y_i} &= b_i - \sum_j a_{ij}x_j - w_i &&= 0, \quad i = 1, 2, \ldots, m. \end{aligned}$$

Writing these equations in matrix form, we get

$$\begin{aligned} A^T y - \mu X^{-1} e &= c \\ y &= \mu W^{-1} e \\ Ax + w &= b. \end{aligned}$$

Here, as warned at the beginning of the chapter, X denotes the diagonal matrix whose diagonal entries are the components of x, and similarly for W. Also, recall that we use e to denote the vector of all ones.

Introducing an extra vector defined as $z = \mu X^{-1} e$, we can rewrite the first-order optimality conditions like this:

$$\begin{aligned} Ax + w &= b. \\ A^T y - z &= c \\ z &= \mu X^{-1} e \\ y &= \mu W^{-1} e. \end{aligned}$$

Finally, if we multiply the third equation through by X and the fourth equation by W, we arrive at a primal–dual symmetric form for writing these equations:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e. \end{aligned} \tag{16.6}$$

Note that the first equation is the equality constraint that appears in the primal problem, while the second equation is the equality constraint for the dual problem. Furthermore, writing the third and fourth equations out componentwise,

$$\begin{aligned} x_j z_j &= \mu \qquad \text{j=1,2,...,n} \\ y_i w_i &= \mu \qquad \text{i=1,2,...,m,} \end{aligned}$$

we see that they are closely related to our old friend: complementarity. In fact, if we set μ to zero, then they are exactly the usual complementarity conditions that must be satisfied at optimality. For this reason, we call these last two equations the μ-*complementarity* conditions.

The first-order optimality conditions, as written in (16.6), give us $2n + 2m$ equations in $2n + 2m$ unknowns. If these equations were linear, they could be solved using Gaussian elimination, and the entire subject of linear programming would be no more difficult than solving systems of linear equations. But alas, they are nonlinear—but just barely. The only nonlinear expressions in these equations are simple multiplications such as $x_j z_j$. This is about the closest to being linear that one could imagine. Yet, it is this nonlinearity that makes the subject of linear programming nontrivial.

We must ask both whether a solution to (16.6) exists and if so is it unique. We address these questions in reverse order.

4. Second-Order Information

To show that the solution, if it is exists, must be unique, we use second-order information on the barrier function:

$$f(x,w) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i. \tag{16.7}$$

The first derivatives are

$$\begin{aligned} \frac{\partial f}{\partial x_j} &= c_j + \frac{\mu}{x_j}, \qquad && j = 1, 2, \ldots, n, \\ \frac{\partial f}{\partial w_i} &= \frac{\mu}{w_i}, && i = 1, 2, \ldots, m, \end{aligned}$$

and the pure second derivatives are

$$\begin{aligned} \frac{\partial^2 f}{\partial x_j^2} &= -\frac{\mu}{x_j^2}, \qquad && j = 1, 2, \ldots, n, \\ \frac{\partial^2 f}{\partial w_i^2} &= -\frac{\mu}{w_i^2}, && i = 1, 2, \ldots, m. \end{aligned}$$

All the mixed second derivatives vanish. Therefore, the Hessian is a diagonal matrix with strictly negative entries. Hence, by Theorem 16.1, there can be at most one critical point and, if it exists, it is a global maximum.

5. Existence

So, does a solution to the barrier problem always exist? It might not. Consider, for example, the following trivial optimization problem on the nonnegative half-line:

$$\begin{aligned} &\text{maximize } 0 \\ &\text{subject to } x \geq 0. \end{aligned}$$

For this problem, the barrier function is

$$f(x) = \mu \log x,$$

which doesn't have a maximum (or, less precisely, the maximum is infinity which is attained at $x = \infty$). However, such examples are rare. For example, consider modifying the objective function in this example to make $x = 0$ the unique optimal solution:

$$\begin{aligned} &\text{maximize } -x \\ &\text{subject to } x \geq 0. \end{aligned}$$

In this case, the barrier function is

$$f(x) = -x + \mu \log x,$$

which is a function whose maximum is attained at $x = \mu$.

In general, we have the following result:

THEOREM 16.2. *There exists a solution to the barrier problem if and only if both the primal and the dual feasible regions have nonempty interior.*

PROOF. The "only if" part is trivial and less important to us. Therefore, we only prove the "if" part. To this end, suppose that both the primal and the dual feasible regions have nonempty interior. This means that there exists a primal feasible point $(\bar{x}, \bar{w})$ with $\bar{x} > 0$ and $\bar{w} > 0$ and there exists a dual feasible point $(\bar{y}, \bar{z})$ with $\bar{y} > 0$ and $\bar{z} > 0$.[1] Now, given any primal feasible point (x, w), consider the expression $\bar{z}^T x + \bar{y}^T w$. Replacing the primal and dual slack variables with their definitions, we can rewrite this expression as follows:

$$\begin{aligned} \bar{z}^T x + \bar{y}^T w &= \left(A^T \bar{y} - c\right)^T x + \bar{y}^T (b - Ax) \\ &= b^T \bar{y} - c^T x. \end{aligned}$$

Solving this equation for the primal objective function $c^T x$, we get that

$$c^T x = -\bar{z}^T x - \bar{y}^T w + b^T \bar{y}.$$

Therefore, the barrier function f defined in equation (16.7) can be written as follows:

$$\begin{aligned} f(x, w) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ = \sum_j (-\bar{z}_j x_j + \mu \log x_j) \\ + \sum_i (-\bar{y}_i w_i + \mu \log w_i) \\ + b^T \bar{y}. \end{aligned}$$

Note that the last term is just a constant. Also, each summand in the two sums is a function of just one variable. These functions all have the following general form:

$$h(\xi) = -a\xi + \mu \log \xi, \qquad 0 < \xi < \infty,$$

where $a > 0$. Such functions have a unique maximum (at μ/a) and tend to $-\infty$ as ξ tends to ∞. From these observations, it is easy to see that, for every constant c, the set

$$\left\{(x, w) \in \mathbb{R}^{n+m} : f(x, w) \geq c\right\}$$

is bounded.

[1] Recall that we write $\xi > 0$ to mean that $\xi_j > 0$ for all j.

Put

$$\bar{f} = f(\bar{x}, \bar{w})$$

and let

$$\bar{P} = \{(x, w) : Ax + w = b, x \geq 0, w \geq 0, f(x, w) \geq \bar{f}\}.$$

Clearly, $\bar{P}$ is nonempty, since it contains $(\bar{x}, \bar{w})$. From the discussion above, we see that $\bar{P}$ is a bounded set.

This set is also closed. To see this, note that it is the intersection of three sets,

$$\{(x, w) : Ax + w = b\} \cap \{(x, w) : x \geq 0, w \geq 0\} \cap \{(x, w) : f(x, w) \geq \bar{f}\}.$$

The first two of these sets are obviously closed. The third set is closed because it is the inverse image of a closed set, $[\bar{f}, \infty]$, under a continuous mapping f. Finally, the intersection of three closed sets is closed.

In Euclidean spaces, a closed bounded set is called compact. A well-known theorem from real analysis about compact sets is that a continuous function on a nonempty compact set attains its maximum. This means that there exists a point in the compact set at which the function hits its maximum. Applying this theorem to f on $\bar{P}$, we see that f does indeed attain its maximum on $\bar{P}$, and this implies it attains its maximum on all of $\{(x, w) : x > 0, w > 0\}$, since $\bar{P}$ was by definition that part of this domain on which f takes large values (bigger than $\bar{f}$, anyway). This completes the proof. □

We summarize our main result in the following corollary:

COROLLARY 16.3. *If a primal feasible set (or, for that matter, its dual) has a nonempty interior and is bounded, then for each $\mu > 0$ there exists a unique solution*

$$(x_\mu, w_\mu, y_\mu, z_\mu)$$

to (16.6).

PROOF. Follows immediately from the previous theorem and Exercise 10.7. □

The path $\{(x_\mu, w_\mu, y_\mu, z_\mu) : \mu > 0\}$ is called the *primal–dual central path*. It plays a fundamental role in interior-point methods for linear programming. In the next chapter, we define the simplest interior-point method. It is an iterative procedure that at each iteration attempts to move toward a point on the central path that is closer to optimality than the current point.

Exercises

16.1 Compute and graph the central trajectory for the following problem:

$$\begin{aligned} \text{maximize } & -x_1 + x_2 \\ \text{subject to } & x_2 \leq 1 \\ & -x_1 \leq -1 \\ & x_1,\ x_2 \geq 0. \end{aligned}$$

Hint: The primal and dual problems are the same — exploit this symmetry.

16.2 Let θ be a fixed parameter, $0 \leq \theta \leq \frac{\pi}{2}$, and consider the following problem:

$$\begin{aligned} \text{maximize} \quad & (\cos\theta)x_1 + (\sin\theta)x_2 \\ \text{subject to} \quad & x_1 \leq 1 \\ & x_2 \leq 1 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Compute an explicit formula for the central path $(x_\mu, w_\mu, y_\mu, z_\mu)$, and evaluate $\lim_{\mu\to\infty} x_\mu$ and $\lim_{\mu\to 0} x_\mu$.

16.3 Suppose that $\{x : Ax \leq b, x \geq 0\}$ is bounded. Let $r \in \mathbb{R}^n$ and $s \in \mathbb{R}^m$ be vectors with positive elements. By studying an appropriate barrier function, show that there exists a unique solution to the following nonlinear system:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= r \\ YWe &= s \\ x, y, z, w &> 0. \end{aligned}$$

16.4 Consider the linear programming problem in equality form:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_j c_j x_j \\ \text{subject to} \quad & \displaystyle\sum_j a_j x_j = b \\ & x_j \geq 0, \qquad j = 1, 2, \ldots, n, \end{aligned} \tag{16.8}$$

where each a_j is a vector in $\mathbb{R}^m$, as is b. Consider the change of variables,

$$x_j = \xi_j^2,$$

and the associated maximization problem:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_j c_j \xi_j^2 \\ \text{subject to} \quad & \textstyle\sum_j a_j \xi_j^2 = b \end{aligned} \tag{16.9}$$

(note that the nonnegativity constraints are no longer needed). Let V denote the set of basic feasible solutions to (16.8), and let W denote the set of points $(\xi_1^2, \xi_2^2, \ldots, \xi_n^2)$ in $\mathbb{R}^n$ for which $(\xi_1, \xi_2, \ldots, \xi_n)$ is a solution to the first-order optimality conditions for (16.9). Show that $V \subset W$. What does this say about the possibility of using (16.9) as a vehicle to solve (16.8)?

Notes

Research into interior-point methods has its roots in the work of Fiacco & McCormick (1968). Interest in these methods exploded after the appearance of the seminal paper Karmarkar (1984). Karmarkar's paper uses clever ideas from *projective geometry.* It doesn't mention anything about central paths, which have become fundamental to the theory of interior-point methods. The discovery that Karmarkar's algorithm has connections with the primal–dual central path introduced in this chapter can be traced to Megiddo (1989). The notion of central points can be traced to pre-Karmarkar times with the work of Huard (1967). D.A. Bayer and J.C. Lagarias, in a pair of papers (Bayer & Lagarias 1989*a*,*b*), give an in-depth study of the central path.

Deriving optimality conditions and giving conditions under which they are necessary and sufficient to guarantee optimality is one of the main goals of *nonlinear programming*. Standard texts on this subject include the books by Luenberger (1984), Bertsekas (1995), and Nash & Sofer (1996).

CHAPTER 17

A Path-Following Method

In this chapter, we define an interior-point method for linear programming that is called a path-following method. Recall that for the simplex method we required a two-phase solution procedure. The path-following method is a one-phase method. This means that the method can begin from a point that is neither primal nor dual feasible and it will proceed from there directly to the optimal solution. Hence, we start with an arbitrary choice of strictly positive values for all the primal and dual variables, i.e., $(x, w, y, z) > 0$, and then iteratively update these values as follows:

(1) Estimate an appropriate value for μ (i.e., smaller than the "current" value but not too small).
(2) Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$ pointing approximately at the point $(x_\mu, w_\mu, y_\mu, z_\mu)$ on the central path.
(3) Compute a step length parameter θ such that the new point

$$\begin{aligned} \tilde{x} &= x + \theta \Delta x, \qquad & \tilde{y} &= y + \theta \Delta y, \\ \tilde{w} &= w + \theta \Delta w, \qquad & \tilde{z} &= z + \theta \Delta z \end{aligned}$$

continues to have strictly positive components.
(4) Replace (x, w, y, z) with the new solution $(\tilde{x}, \tilde{w}, \tilde{y}, \tilde{z})$.

To fully define the path-following method, it suffices to make each of these four steps precise. Since the second step is in some sense the most fundamental, we start by describing that one after which we turn our attention to the others.

1. Computing Step Directions

Our aim is to find $(\Delta x, \Delta w, \Delta y, \Delta z)$ such that the new point $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ lies approximately on the primal–dual central path at the point $(x_\mu, w_\mu, y_\mu, z_\mu)$. Recalling the defining equations for this point on the central path,

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e, \end{aligned}$$

we see that the new point $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$, if it were to lie exactly on the central path at μ, would be defined by

$$\begin{aligned} A(x + \Delta x) + (w + \Delta w) &= b \\ A^T(y + \Delta y) - (z + \Delta z) &= c \\ (X + \Delta X)(Z + \Delta Z)e &= \mu e \\ (Y + \Delta Y)(W + \Delta W)e &= \mu e. \end{aligned}$$

Thinking of (x, w, y, z) as data and $(\Delta x, \Delta w, \Delta y, \Delta z)$ as unknowns, we rewrite these equations with the unknowns on the left and the data on the right:

$$\begin{aligned} A\Delta x + \Delta w &= b - Ax - w \ =: \rho \\ A^T\Delta y - \Delta z &= c - A^T y + z =: \sigma \\ Z\Delta x + X\Delta z + \Delta X\Delta Z e &= \mu e - XZe \\ W\Delta y + Y\Delta w + \Delta Y\Delta W e &= \mu e - YWe. \end{aligned}$$

Note that we have introduced abbreviated notations, ρ and σ, for the first two right-hand sides. These two vectors represent the *primal infeasibility* and the *dual infeasibility*, respectively.

Now, just as before, these equations form a system of nonlinear equations (this time for the "delta" variables). We want to have a linear system, so at this point we simply drop the nonlinear terms to arrive at the following linear system:

$$A\Delta x + \Delta w = \rho \tag{17.1}$$

$$A^T\Delta y - \Delta z = \sigma \tag{17.2}$$

$$Z\Delta x + X\Delta z = \mu e - XZe \tag{17.3}$$

$$W\Delta y + Y\Delta w = \mu e - YWe. \tag{17.4}$$

This system of equations is a linear system of $2n+2m$ equations in $2n+2m$ unknowns. We will show later that this system is nonsingular (under the mild assumption that A has full rank) and therefore that it has a unique solution that defines the step directions for the path-following method. Chapters 18 and 19 are devoted to studying methods for efficiently solving systems of this type.

If the business of dropping the nonlinear "delta" terms strikes you as bold, let us remark that this is the most common approach to solving nonlinear systems of equations. The method is called Newton's method. It is described briefly in the next section.

2. Newton's Method

Given a function

$$F(\xi) = \begin{bmatrix} F_1(\xi) \\ F_2(\xi) \\ \vdots \\ F_N(\xi) \end{bmatrix}, \qquad \xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix},$$

from $\mathbb{R}^N$ into $\mathbb{R}^N$, a common problem is to find a point $\xi^* \in \mathbb{R}^N$ for which $F(\xi^*) = 0$. Such a point is called a *root* of F. Newton's method is an iterative method for solving this problem. One step of the method is defined as follows. Given any point $\xi \in \mathbb{R}^N$, the goal is to find a *step direction* $\Delta\xi$ for which $F(\xi + \Delta\xi) = 0$. Of course, for a nonlinear F it is not possible to find such a step direction. Hence, it is approximated by the first two terms of its Taylor's series expansion,

$$F(\xi + \Delta\xi) \approx F(\xi) + F'(\xi)\Delta\xi,$$

where

$$F'(\xi) = \begin{bmatrix} \frac{\partial F_1}{\partial \xi_1} & \frac{\partial F_1}{\partial \xi_2} & \cdots & \frac{\partial F_1}{\partial \xi_N} \\ \frac{\partial F_2}{\partial \xi_1} & \frac{\partial F_2}{\partial \xi_2} & \cdots & \frac{\partial F_2}{\partial \xi_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_N}{\partial \xi_1} & \frac{\partial F_N}{\partial \xi_2} & \cdots & \frac{\partial F_N}{\partial \xi_N} \end{bmatrix}.$$

The approximation is linear in $\Delta\xi$. Hence, equating it to zero gives a linear system to solve for the step direction:

$$F'(\xi)\Delta\xi = -F(\xi).$$

Given $\Delta\xi$, Newton's method updates the current solution ξ by replacing it with $\xi + \Delta\xi$. The process continues until the current solution is approximately a root (i.e., $F(\xi) \approx 0$). Simple one-dimensional examples given in every elementary calculus text illustrate that this method works well, when it works, but it can fail if F is not well behaved and the initial point is too far from a solution.

Let's return now to the problem of finding a point on the central path. Letting

$$\xi = \begin{bmatrix} x \\ y \\ w \\ z \end{bmatrix}$$

and

$$F(\xi) = \begin{bmatrix} Ax + w - b \\ A^T y - z - c \\ XZe - \mu e \\ YWe - \mu e \end{bmatrix},$$

we see that the set of equations defining $(x_\mu, y_\mu, w_\mu, z_\mu)$ is a root of F. The matrix of derivatives of F is given by

$$F'(\xi) = \begin{bmatrix} A & 0 & I & 0 \\ 0 & A^T & 0 & -I \\ Z & 0 & 0 & X \\ 0 & W & Y & 0 \end{bmatrix}.$$

Noting that

$$\Delta\xi = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta w \\ \Delta z \end{bmatrix},$$

it is easy to see that the Newton direction coincides with the direction obtained by solving equations (17.1)–(17.4).

3. Estimating an Appropriate Value for the Barrier Parameter

We need to say how to pick μ. If μ is chosen to be too large, then the sequence could converge to the analytic center of the feasible set, which is not our intention. If, on the other hand, μ is chosen to be too small, then the sequence could stray too far from the central path and the algorithm could *jam* into the boundary of the feasible set at a place that is suboptimal. The trick is to find a reasonable compromise between these two extremes. To do this, we first figure out a value that represents, in some sense, the current value of μ and we then choose something smaller than that, say a fixed fraction of it.

We are given a point (x, w, y, z) that is almost certainly off the central path. If it were on the central path, then there are several formulas by which we could recover the corresponding value of μ. For example, we could just compute $z_j x_j$ for any fixed index j. Or we could compute $y_i w_i$ for any fixed i. Or, perverse as it may seem, we could average all these values:

$$\mu = \frac{z^T x + y^T w}{n + m}. \tag{17.5}$$

This formula gives us exactly the value of μ whenever it is known that (x, w, y, z) lies on the central path. The key point here then is that we will use this formula to produce an estimate for μ even when the current solution (x, w, y, z) does not lie on the central path. Of course, the algorithm needs a value of μ that represents a point closer to optimality than the current solution. Hence, the algorithm takes this "par" value and reduces it by a certain fraction:

$$\mu = \delta \frac{z^T x + y^T w}{n + m},$$

where δ is a number between zero and one. In practice, one finds that setting δ to approximately $1/10$ works quite well, but for the sake of discussion we will always leave it as a parameter.

4. Choosing the Step Length Parameter

The step directions, which were determined using Newton's method, were determined under the assumption that the step length parameter θ would be equal to one (i.e., $\tilde{x} = x + \Delta x$, etc.). But taking such a step might cause the new solution to violate the property that every component of all the primal and the dual variables must remain positive. Hence, we may need to use a smaller value for θ. We need to guarantee, for example, that

$$x_j + \theta \Delta x_j > 0, \qquad j = 1, 2, \ldots, n.$$

Moving the Δx_j term to the other side and then dividing through by θ and x_j, both of which are positive, we see that θ must satisfy

$$\frac{1}{\theta} > -\frac{\Delta x_j}{x_j}, \qquad j = 1, 2, \ldots, n.$$

Of course, a similar inequality must be satisfied for the w, y, and z variables too. Putting it all together, the largest value of θ would be given by

$$\frac{1}{\theta} = \max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\},$$

where we have abused notation slightly by using the $\max_{ij}$ to denote the maximum of all the ratios in the indicated set. However, this choice of θ will not guarantee strict inequality, so we introduce a parameter r, which is a number close to but strictly less than one, and we set[1]

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1. \tag{17.6}$$

This formula may look messy, and no one should actually do it by hand, but it is trivial to program a computer to do it. Such a subroutine will be really fast (requiring only on the order of $2n + 2m$ operations).

[1]For compactness, we use the notation $a \wedge b$ to represent the minimum of the two numbers a and b.

initialize $(x, w, y, z) > 0$

while (not optimal) {

$$\rho = b - Ax - w$$

$$\sigma = c - A^T y + z$$

$$\gamma = z^T x + y^T w$$

$$\mu = \delta \frac{\gamma}{n+m}$$

solve:

$$\begin{aligned} A\Delta x + \Delta w &= \rho \\ A^T \Delta y - \Delta z &= \sigma \\ Z\Delta x + X\Delta z &= \mu e - XZe \\ W\Delta y + Y\Delta w &= \mu e - YWe \end{aligned}$$

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$$

$$x \leftarrow x + \theta \Delta x, \qquad w \leftarrow w + \theta \Delta w$$

$$y \leftarrow y + \theta \Delta y, \qquad z \leftarrow z + \theta \Delta z$$

}

FIGURE 17.1. The path-following method.

A summary of the algorithm is shown in Figure 17.1. In the next section, we investigate whether this algorithm actually converges to an optimal solution.

5. Convergence Analysis

In this section, we investigate the convergence properties of the path-following algorithm. Recall that the simplex method is a finite algorithm (assuming that steps are taken to guard against cycling). For interior-point methods, the situation is different. Every solution produced has all variables strictly positive. Yet for a solution to be optimal generally requires many variables to vanish. This vanishing can only happen "in the limit." This raises questions, the most fundamental of which are these: does the sequence of solutions produced by the path-following method converge? If so, is

the limit optimal? How fast is the convergence? In particular, if we set "optimality tolerances," how many iterations will it take to achieve these tolerances? We will address these questions in this section.

In this section, we will need to measure the size of various vectors. There are many choices. For example, for each $1 \leq p < \infty$, we can define the so-called *p-norm* of a vector x as

$$\|x\|_p = \left(\sum_j |x_j|^p \right)^{\frac{1}{p}}.$$

The limit as p tends to infinity is also well defined, and it simplifies to the so-called *sup-norm*:

$$\|x\|_\infty = \max_j |x_j|.$$

5.1. Measures of Progress. Recall from duality theory that there are three criteria that must be met in order that a primal–dual solution be optimal:

(1) Primal feasibility,
(2) Dual feasibility, and
(3) Complementarity.

For each of these criteria, we introduce a measure of the extent to which they fail to be met.

For the primal feasibility criterion, we use the 1-norm of the primal infeasibility vector

$$\rho = b - Ax - w.$$

For the dual feasibility criterion, we use the 1-norm of the dual infeasibility vector

$$\sigma = c - A^T y + z.$$

For complementarity, we use

$$\gamma = z^T x + y^T w.$$

5.2. Progress in One Iteration. For the analysis in the section, we prefer to modify the algorithm slightly by having it take shorter steps than specified before. Indeed, we let

$$\theta = r \left(\max_{i,j} \left\{ \left| \frac{\Delta x_j}{x_j} \right|, \left| \frac{\Delta w_i}{w_i} \right|, \left| \frac{\Delta y_i}{y_i} \right|, \left| \frac{\Delta z_j}{z_j} \right| \right\} \right)^{-1} \wedge 1$$

$$= \frac{r}{\max(\|X^{-1}\Delta x\|_\infty, \ldots, \|Z^{-1}\Delta z\|_\infty)} \wedge 1. \tag{17.7}$$

Note that the only change has been to replace the negative ratios with the absolute value of the same ratios. Since the maximum of the absolute values can be larger than the maximum of the ratios themselves, this formula produces a smaller value for θ. In this section, let x, y, etc., denote quantities from one iteration of the algorithm, and

put a tilde on the same letters to denote the same quantity at the next iteration of the algorithm. Hence,

$$\begin{aligned} \tilde{x} &= x + \theta\Delta x, \qquad & \tilde{y} &= y + \theta\Delta y, \\ \tilde{w} &= w + \theta\Delta w, \qquad & \tilde{z} &= z + \theta\Delta z. \end{aligned}$$

Now let's compute some of the other quantities. We begin with the primal infeasibility:

$$\begin{aligned} \tilde{\rho} &= b - A\tilde{x} - \tilde{w} \\ &= b - Ax - w - \theta(A\Delta x + \Delta w). \end{aligned}$$

But $b - Ax - w$ equals the primal infeasibility ρ (by definition) and $A\Delta x + \Delta w$ also equals ρ, since this is precisely the first equation in the system that defines the "delta" variables. Hence,

$$\tilde{\rho} = (1 - \theta)\rho. \tag{17.8}$$

Similarly,

$$\begin{aligned} \tilde{\sigma} &= c - A^T\tilde{y} + \tilde{z} \\ &= c - A^T y + z - \theta(A\Delta y - \Delta z) \\ &= (1 - \theta)\sigma. \end{aligned} \tag{17.9}$$

Since θ is a number between zero and one, it follows that each iteration produces a decrease in both the primal and the dual infeasibility and that this decrease is better the closer θ is to one.

The analysis of the complementarity is a little more complicated (after all, this is the part of the system where the linearization took place):

$$\begin{aligned} \tilde{\gamma} &= \tilde{z}^T\tilde{x} + \tilde{y}^T\tilde{w} \\ &= (z + \theta\Delta z)^T(x + \theta\Delta x) + (y + \theta\Delta y)^T(w + \theta\Delta w) \\ &= z^T x + y^T w \\ &\quad + \theta(z^T\Delta x + \Delta z^T x + y^T\Delta w + \Delta y^T w) \\ &\quad + \theta^2(\Delta z^T\Delta x + \Delta y^T\Delta w). \end{aligned}$$

We need to analyze each of the θ terms separately. From (17.3), we see that

$$\begin{aligned} z^T\Delta x + \Delta z^T x &= e^T\left(Z\Delta x + X\Delta z\right) \\ &= e^T\left(\mu e - ZXe\right) \\ &= \mu n - z^T x. \end{aligned}$$

Similarly, from (17.4), we have

$$\begin{aligned} y^T \Delta w + \Delta y^T w &= e^T \left(Y \Delta w + W \Delta y \right) \\ &= e^T \left(\mu e - YWe \right) \\ &= \mu m - y^T w. \end{aligned}$$

Finally, (17.1) and (17.2) imply that

$$\begin{aligned} \Delta z^T \Delta x + \Delta y^T \Delta w &= \left(A^T \Delta y - \sigma \right)^T \Delta x + \Delta y^T \left(\rho - A \Delta x \right) \\ &= \Delta y^T \rho - \sigma^T \Delta x. \end{aligned}$$

Substituting these expressions into the last expression for $\tilde{\gamma}$, we get

$$\begin{aligned} \tilde{\gamma} = z^T x + y^T w & \\ + \theta \left(\mu(n+m) - (z^T x + y^T w) \right) & \\ + \theta^2 \left(\Delta y^T \rho - \sigma^T \Delta x \right). & \end{aligned}$$

At this point, we recognize that $z^T x + y^T w = \gamma$ and that $\mu(n+m) = \delta\gamma$. Hence,

$$\tilde{\gamma} = \left(1 - (1-\delta)\theta \right) \gamma + \theta^2 \left(\Delta y^T \rho - \sigma^T \Delta x \right).$$

We must now abandon equalities and work with estimates. Our favorite tool for estimation is the following inequality:

$$\begin{aligned} |v^T w| &= |\sum_j v_j w_j| \\ &\leq \sum_j |v_j||w_j| \\ &\leq (\max_j |v_j|)(\sum_j |w_j|) \\ &= \|v\|_\infty \|w\|_1. \end{aligned}$$

This inequality is the trivial case of *Hölder's inequality*. From Hölder's inequality, we see that

$$|\Delta y^T \rho| \leq \|\rho\|_1 \|\Delta y\|_\infty \qquad \text{and} \qquad |\sigma^T \Delta x| \leq \|\sigma\|_1 \|\Delta x\|_\infty.$$

Hence,

$$\tilde{\gamma} \leq \left(1 - (1-\delta)\theta \right) \gamma + \theta \left(\|\rho\|_1 \|\theta \Delta y\|_\infty + \|\sigma\|_1 \|\theta \Delta x\|_\infty \right).$$

Next, we use the specific choice of step length θ to get a bound on $\|\theta \Delta y\|_\infty$ and $\|\theta \Delta x\|_\infty$. Indeed, (17.7) implies that

$$\theta \leq \frac{r}{\|X^{-1} \Delta x\|_\infty} \leq \frac{x_j}{|\Delta x_j|} \qquad \text{for all } j.$$

Hence,

$$\|\theta \Delta x\|_\infty \leq \|x\|_\infty.$$

Similarly,

$$\|\theta\Delta y\|_\infty \le \|y\|_\infty.$$

If we now assume that, along the sequence of points x and y visited by the algorithm, $\|x\|_\infty$ and $\|y\|_\infty$ are bounded by a large real number M, then we can estimate the new complementarity as

$$\tilde{\gamma} \le (1 - (1-\delta)\theta)\,\gamma + M\|\rho\|_1 + M\|\sigma\|_1. \tag{17.10}$$

5.3. Stopping Rule. Let $\epsilon > 0$ be a small positive tolerance, and let $M < \infty$ be a large finite tolerance. If $\|x\|_\infty$ gets larger than M, then we stop and declare the problem primal unbounded. If $\|y\|_\infty$ gets larger than M, then we stop and declare the problem dual unbounded. Finally, if $\|\rho\|_1 < \epsilon$, $\|\sigma\|_1 < \epsilon$, and $\gamma < \epsilon$, then we stop and declare the current solution to be optimal (at least within this small tolerance).

Since γ is a measure of complementarity and complementarity is related to the duality gap, one would expect that a small value of γ should translate into a small duality gap. This turns out to be true. Indeed, from the definitions of γ, σ, and ρ, we can write

$$\begin{aligned}\gamma &= z^T x + y^T w \\ &= (\sigma + A^T y - c)^T x + y^T(b - Ax - \rho) \\ &= b^T y - c^T x + \sigma^T x - \rho^T y.\end{aligned}$$

At this point, we use Hölder's inequality to bound some of these terms to get an estimate on the duality gap:

$$\begin{aligned}|b^T y - c^T x| &\le \gamma + |\sigma^T x| + |y^T \rho| \\ &\le \gamma + \|\sigma\|_1\|x\|_\infty + \|\rho\|_1\|y\|_\infty.\end{aligned}$$

Now, if γ, $\|\sigma\|_1$, and $\|\rho\|_1$ are all small (and $\|x\|_\infty$ and $\|y\|_\infty$ are not too big), then the duality gap will be small. This estimate shows that one shouldn't expect the duality gap to get small until the primal and the dual are very nearly feasible. Actual implementations confirm this expectation.

5.4. Progress Over Several Iterations. Now let $\rho^{(k)}$, $\sigma^{(k)}$, $\gamma^{(k)}$, $\theta^{(k)}$, etc., denote the values of these quantities at the kth iteration. We have the following result about the overall performance of the algorithm:

THEOREM 17.1. *Suppose there is a real number $t > 0$, a real number $M < \infty$, and an integer K such that for all $k \le K$,*

$$\begin{aligned}\theta^{(k)} &\ge t, \\ \|x^{(k)}\|_\infty &\le M, \\ \|y^{(k)}\|_\infty &\le M.\end{aligned}$$

Then there exists a constant $\bar{M} < \infty$ such that

$$\|\rho^{(k)}\|_1 \leq (1-t)^k \|\rho^{(0)}\|_1,$$
$$\|\sigma^{(k)}\|_1 \leq (1-t)^k \|\sigma^{(0)}\|_1,$$
$$\gamma^{(k)} \leq (1-\tilde{t})^k \bar{M},$$

for all $k \leq K$ where

$$\tilde{t} = t(1-\delta).$$

PROOF. From (17.8) and the bound on $\theta^{(k)}$, it follows that

$$\|\rho^{(k)}\|_1 \leq (1-t)\|\rho^{(k-1)}\|_1 \leq \cdots \leq (1-t)^k \|\rho^{(0)}\|_1.$$

Similarly, from (17.9), it follows that

$$\|\sigma^{(k)}\|_1 \leq (1-t)\|\sigma^{(k-1)}\|_1 \leq \cdots \leq (1-t)^k \|\sigma^{(0)}\|_1.$$

As usual, $\gamma^{(k)}$ is harder to estimate. From (17.10) and the previous two estimates, we see that

$$\begin{aligned}\gamma^{(k)} &\leq (1-t(1-\delta))\,\gamma^{(k-1)} \\ &\quad + M(1-t)^{k-1}\left(\|\rho^{(0)}\|_1 + \|\sigma^{(0)}\|_1\right) \\ &= (1-\tilde{t})\gamma^{(k-1)} + \tilde{M}(1-t)^{k-1}, \end{aligned} \tag{17.11}$$

where $\tilde{M} = M\left(\|\rho^{(0)}\|_1 + \|\sigma^{(0)}\|_1\right)$. Since an analogous inequality relates $\gamma^{(k-1)}$ to $\gamma^{(k-2)}$, we can substitute this analogous inequality into (17.11) to get

$$\begin{aligned}\gamma^{(k)} &\leq (1-\tilde{t})\left[(1-\tilde{t})\gamma^{(k-2)} + \tilde{M}(1-t)^{k-2}\right] + \tilde{M}(1-t)^{k-1} \\ &= (1-\tilde{t})^2\gamma^{(k-2)} + \tilde{M}(1-t)^{k-1}\left[\frac{1-\tilde{t}}{1-t} + 1\right].\end{aligned}$$

Continuing in this manner, we see that

$$\begin{aligned}\gamma^{(k)} &\leq (1-\tilde{t})^2\left[(1-\tilde{t})\gamma^{(k-3)} + \tilde{M}(1-t)^{k-3}\right] \\ &\qquad + \tilde{M}(1-t)^{k-1}\left[\frac{1-\tilde{t}}{1-t} + 1\right] \\ &= (1-\tilde{t})^3\gamma^{(k-3)} + \tilde{M}(1-t)^{k-1}\left[\left(\frac{1-\tilde{t}}{1-t}\right)^2 + \frac{1-\tilde{t}}{1-t} + 1\right] \\ &\leq \cdots \leq \\ &\leq (1-\tilde{t})^k\gamma^{(0)} + \tilde{M}(1-t)^{k-1}\left[\left(\frac{1-\tilde{t}}{1-t}\right)^{k-1} + \cdots + \frac{1-\tilde{t}}{1-t} + 1\right].\end{aligned}$$

Now we sum the bracketed partial sum of a geometric series to get

$$(1-t)^{k-1}\left[\left(\frac{1-\tilde{t}}{1-t}\right)^{k-1}+\cdots+\frac{1-\tilde{t}}{1-t}+1\right]=(1-t)^{k-1}\frac{1-\left(\frac{1-\tilde{t}}{1-t}\right)^{k}}{1-\frac{1-\tilde{t}}{1-t}}$$
$$=\frac{(1-\tilde{t})^k-(1-t)^k}{t-\tilde{t}}.$$

Recalling that $\tilde{t}=t(1-\delta)$ and dropping the second term in the numerator, we get

$$\frac{(1-\tilde{t})^k-(1-t)^k}{t-\tilde{t}}\leq\frac{(1-\tilde{t})^k}{\delta t}.$$

Putting this all together, we see that

$$\gamma^{(k)}\leq(1-\tilde{t})^k\left(\gamma^{(0)}+\frac{\tilde{M}}{\delta t}\right).$$

Denoting the parenthesized expression by $\bar{M}$ completes the proof. □

Theorem 17.1 is only a partial convergence result because it depends on the assumption that the step lengths remain bounded away from zero. To show that the step lengths do indeed have this property requires that the algorithm be modified and that the starting point be carefully selected. The details are rather technical and hence omitted (see the Notes at the end of the chapter for references).

Also, before we leave this topic, note that the primal and dual infeasibilities go down by a factor of $1-t$ at each iteration, whereas the duality gap goes down by a smaller amount $1-\tilde{t}$. The fact that the duality gap converges more slowly that the infeasibilities is also readily observed in practice.

Exercises

17.1 Starting from $(x,w,y,z)=(e,e,e,e)$, and using $\delta=1/10$, and $r=9/10$, compute (x,w,y,z) after one step of the path-following method for the problem given in
(a) Exercise 2.3.
(b) Exercise 2.4.
(c) Exercise 2.5.
(d) Exercise 2.10.

17.2 Let $\{(x_\mu,w_\mu,y_\mu,z_\mu):\mu\geq 0\}$ denote the central trajectory. Show that

$$\lim_{\mu\to\infty} b^Ty_\mu-c^Tx_\mu=\infty.$$

Hint: look at (17.5).

17.3 Consider a linear programming problem whose feasible region is bounded and has nonempty interior. Use the result of Exercise 17.2 to show that the dual problem's feasible set is unbounded.

17.4 *Scale invariance.* Consider a linear program and its dual:

$$(P)\quad \begin{aligned} &\max\ c^T x \\ &\text{s.t. } Ax + w = b \\ &\qquad x, w \geq 0 \end{aligned} \qquad (D)\quad \begin{aligned} &\min\ b^T y \\ &\text{s.t. } A^T y - z = c \\ &\qquad y, z \geq 0. \end{aligned}$$

Let R and S be two given diagonal matrices having positive entries along their diagonals. Consider the *scaled* reformulation of the original problem and its dual:

$$(\bar{P})\quad \begin{aligned} &\max\ (Sc)^T \bar{x} \\ &\text{s.t. } RAS\bar{x} + \bar{w} = Rb \\ &\qquad \bar{x}, \bar{w} \geq 0 \end{aligned} \qquad (\bar{D})\quad \begin{aligned} &\min\ (Rb)^T \bar{y} \\ &\text{s.t. } SA^T R\bar{y} - \bar{z} = Sc \\ &\qquad \bar{y}, \bar{z} \geq 0. \end{aligned}$$

Let (x^k, w^k, y^k, z^k) denote the sequence of solutions generated by the primal–dual interior-point method applied to (P)–(D). Similarly, let $(\bar{x}^k, \bar{w}^k, \bar{y}^k, \bar{z}^k)$ denote the sequence of solutions generated by the primal–dual interior-point method applied to $(\bar{P})$–$(\bar{D})$. Suppose that we have the following relations among the starting points:

$$\bar{x}^0 = S^{-1}x^0, \quad \bar{w}^0 = Rw^0, \quad \bar{y}^0 = R^{-1}y^0, \quad \bar{z}^0 = Sz^0.$$

Show that these relations then persist. That is, for each $k \geq 1$,

$$\bar{x}^k = S^{-1}x^k, \quad \bar{w}^k = Rw^k, \quad \bar{y}^k = R^{-1}y^k, \quad \bar{z}^k = Sz^k.$$

17.5 *Homotopy method.* Let $\bar{x}$, $\bar{y}$, $\bar{z}$, and $\bar{w}$ be given componentwise positive "initial" values for x, y, z, and w, respectively. Let t be a parameter between 0 and 1. Consider the following nonlinear system:

$$\begin{aligned} Ax + w &= tb + (1-t)(A\bar{x} + \bar{w}) \\ A^T y - z &= tc + (1-t)(A^T\bar{y} - \bar{z}) \\ XZe &= (1-t)\bar{X}\bar{Z}e \\ YWe &= (1-t)\bar{Y}\bar{W}e \\ x, y, z, w &> 0. \end{aligned} \tag{17.12}$$

(a) Use Exercise 16.3 to show that this nonlinear system has a unique solution for each $0 \leq t < 1$. Denote it by $(x(t), y(t), z(t), w(t))$.

(b) Show that $(x(0), y(0), z(0), w(0)) = (\bar{x}, \bar{y}, \bar{z}, \bar{w})$.

(c) Assuming that the limit

$$(x(1), y(1), z(1), w(1)) = \lim_{t \to 1} (x(t), y(t), z(t), w(t))$$

exists, show that it solves the standard-form linear programming problem.

(d) The family of solutions $(x(t), y(t), z(t), w(t))$, $0 \le t < 1$, describes a curve in "primal–dual" space. Show that the tangent to this curve at $t = 0$ coincides with the path-following step direction at $(\bar{x}, \bar{y}, \bar{z}, \bar{w})$ computed with $\mu = 0$; that is,

$$\left(\frac{dx}{dt}(0), \frac{dy}{dt}(0), \frac{dz}{dt}(0), \frac{dw}{dt}(0)\right) = (\Delta x, \Delta y, \Delta z, \Delta w),$$

where $(\Delta x, \Delta y, \Delta z, \Delta w)$ is the solution to (17.1)–(17.4).

17.6 *Higher-order methods.* The previous exercise shows that the path-following step direction can be thought of as the direction one gets by approximating a homotopy path with its tangent line:

$$x(t) \approx x(0) + \frac{dx}{dt}(0)t.$$

By using more terms of the Taylor's series expansion, one can get a better approximation:

$$x(t) \approx x(0) + \frac{dx}{dt}(0)t + \frac{1}{2}\frac{d^2x}{dt^2}(0)t^2 + \cdots + \frac{1}{k!}\frac{d^kx}{dt^k}(0)t^k.$$

(a) Differentiating the equations in (17.12) twice, derive a linear system for $(d^2x/dt^2(0), d^2y/dt^2(0), d^2z/dt^2(0), d^2w/dt^2(0))$.

(b) Can the same technique be applied to derive linear systems for the higher-order derivatives?

17.7 *Linear Complementarity Problem.* Given a $k \times k$ matrix M and a k-vector q, a vector x is said to solve the linear complementarity problem if

$$\begin{aligned} -Mx + z &= q \\ XZe &= 0 \\ x, z &\ge 0 \end{aligned}$$

(note that the first equation can be taken as the definition of z).

(a) Show that the optimality conditions for linear programming can be expressed as a linear complementarity problem with

$$M = \begin{bmatrix} 0 & -A \\ A^T & 0 \end{bmatrix}.$$

(b) The path-following method introduced in this chapter can be extended to cover linear complementarity problems. The main step in the derivation is to replace the complementarity condition $XZe = 0$ with a μ-complementarity condition $XZe = \mu e$ and then to use Newton's

method to derive step directions Δx and Δz. Carry out this procedure and indicate the system of equations that define Δx and Δz.

(c) Give conditions under which the system derived above is guaranteed to have a unique solution.

(d) Write down the steps of the path-following method for the linear complementarity problem.

(e) Study the convergence of this algorithm by adapting the analysis given in Section 17.5.

17.8 Consider again the L^1-regression problem:

$$\text{minimize } \|b - Ax\|_1.$$

Complete the following steps to derive the step direction vector Δx associated with the primal-dual affine-scaling method for solving this problem.

(a) Show that the L^1-regression problem is equivalent to the following linear programming problem:

$$\begin{aligned} &\text{minimize } && e^T(t_+ + t_-) \\ &\text{subject to } && Ax + t_+ - t_- = b \\ &&& t_+, t_- \geq 0. \end{aligned} \tag{17.13}$$

(b) Write down the dual of (17.13).

(c) Add slack and/or surplus variables as necessary to reformulate the dual so that all inequalities are simple nonnegativities of variables.

(d) Identify all primal-dual pairs of complementary variables.

(e) Write down the nonlinear system of equations consisting of: (1) the primal equality constraints, (2) the dual equality constraints, (3) all complementarity conditions (using $\mu = 0$ since we are looking for an affine-scaling algorithm).

(f) Apply Newton's method to the nonlinear system to obtain a linear system for step directions for all of the primal and dual variables.

(g) We may assume without loss of generality that both the initial primal solution and the initial dual solution are feasible. Explain why.

(h) The linear system derived above is a 6×6 block matrix system. But it is easy to solve most of it by hand. First eliminate those step direction associated with the nonnegative variables to arrive at a 2×2 block matrix system.

(i) Next, solve the 2×2 system. Give an explicit formula for Δx.

(j) How does this primal-dual affine-scaling algorithm compare with the iteratively reweighted least squares algorithm defined in Section 12.5?

17.9

(a) Let ξ_j, $j = 1, 2, \ldots$, denote a sequence of real numbers between zero and one. Show that $\prod_j (1 - \xi_j) = 0$ if $\sum_j \xi_j = \infty$.

(b) Use the result of part a to prove the following convergence result: if the sequences $\|x^{(k)}\|_\infty$, $k = 1, 2, \ldots$, and $\|y^{(k)}\|_\infty$, $k = 1, 2, \ldots$, are bounded and $\sum_k \theta^{(k)} = \infty$, then

$$\lim_{k \to \infty} \|\rho^{(k)}\|_1 = 0$$

$$\lim_{k \to \infty} \|\sigma^{(k)}\|_1 = 0$$

$$\lim_{k \to \infty} \gamma^{(k)} = 0.$$

Notes

The path-following algorithm introduced in this chapter has its origins in a paper by Kojima et al. (1989). Their paper assumed an initial feasible solution and therefore was a true interior-point method. The method given in this chapter does not assume the initial solution is feasible—it is a one-phase algorithm. The simple yet beautiful idea of modifying the Kojima–Mizuno–Yoshise primal–dual algorithm to make it into a one-phase algorithm is due to Lustig (1990).

Of the thousands of papers on interior-point methods that have appeared in the last decade, the majority have included convergence proofs for some version of an interior-point method. Here, we only mention a few of the important papers. The first polynomial-time algorithm for linear programming was discovered by Khachian (1979). Khachian's algorithm is fundamentally different from any algorithm presented in this book. Paradoxically, it proved in practice to be inferior to the simplex method. N.K. Karmarkar's pathbreaking paper (Karmarkar 1984) contained a detailed convergence analysis. His claims, based on preliminary testing, that his algorithm is uniformly substantially faster than the simplex method sparked a revolution in linear programming. Unfortunately, his claims proved to be exaggerated, but nonetheless interior-point methods have been shown to be competitive with the simplex method and usually superior on very large problems. The convergence proof for a primal–dual interior-point method was given by Kojima et al. (1989). Shortly thereafter, Monteiro & Adler (1989) improved on the convergence analysis. Two recent survey papers, Todd (1995) and Anstreicher (1996), give nice overviews of the current state of the art. Also, a soon-to-be-published book by Wright (1996) should prove to be a valuable reference to the reader wishing more information on convergence properties of these algorithms.

The homotopy method outlined in Exercise 17.5 is described in Nazareth (1986) and Nazareth (1996). Higher-order path-following methods are described (differently) in Carpenter et al. (1993).

CHAPTER 18

The KKT System

The most time-consuming aspect of each iteration of the path-following method is solving the system of equations that defines the step direction vectors Δx, Δy, Δw, and Δz:

$$A\Delta x + \Delta w = \rho \tag{18.1}$$

$$A^T\Delta y - \Delta z = \sigma \tag{18.2}$$

$$Z\Delta x + X\Delta z = \mu e - XZe \tag{18.3}$$

$$W\Delta y + Y\Delta w = \mu e - YWe. \tag{18.4}$$

After minor manipulation, these equations can be written in block matrix form as follows:

$$\left[\begin{array}{cc|cc} -XZ^{-1} & & -I & \\ & & A & I \\ \hline -I & A^T & & \\ & I & & YW^{-1} \end{array}\right] \left[\begin{array}{c} \Delta z \\ \Delta y \\ \hline \Delta x \\ \Delta w \end{array}\right] = \left[\begin{array}{c} -\mu Z^{-1}e + x \\ \rho \\ \hline \sigma \\ \mu W^{-1}e - y \end{array}\right]. \tag{18.5}$$

This system is called the *Karush–Kuhn–Tucker system*, or simply the KKT system. It is a symmetric linear system of $2n + 2m$ equations in $2n + 2m$ unknowns. One could, of course, perform a factorization of this large system and then follow that with a forward and backward substitution to solve the system. However, it is better to do part of this calculation "by hand" first and only use a factorization routine to help solve a smaller system. There are two stages of reductions that one could apply. After the first stage, the remaining system is called the reduced KKT system, and after the second stage it is called the system of normal equations. We shall discuss these two systems in the next two sections.

1. The Reduced KKT System

Equations (18.3) and (18.4) are trivial (in the sense that they only involve diagonal matrices), and so it seems sensible to eliminate them right from the start. To preserve the symmetry that we saw in (18.5), we should solve them for Δz and Δw,

respectively:

$$\Delta z = X^{-1}(\mu e - XZe - Z\Delta x)$$
$$\Delta w = Y^{-1}(\mu e - YWe - W\Delta y).$$

Substituting these formulas into (18.1) and (18.2), we get the so-called *reduced KKT system*:

$$A\Delta x - Y^{-1}W\Delta y = \rho - \mu Y^{-1}e + w \tag{18.6}$$
$$A^T\Delta y + X^{-1}Z\Delta x = \sigma + \mu X^{-1}e - z. \tag{18.7}$$

Substituting in the definitions of ρ and σ and writing the system in matrix notation, we get

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} b - Ax - \mu Y^{-1}e \\ c - A^T y + \mu X^{-1}e \end{bmatrix}.$$

Note that the reduced KKT matrix is again a symmetric matrix. Also, the right-hand side displays symmetry between the primal and the dual. To reduce the system any further, one needs to break the symmetry that we have carefully preserved up to this point. Nonetheless, we forge ahead.

2. The Normal Equations

For the second stage of reduction, there are two choices: we could either (1) solve (18.6) for Δy and eliminate it from (18.7) or (2) solve (18.7) for Δx and eliminate it from (18.6). For the moment, let us assume that we follow the latter approach. In this case, we get from (18.7) that

$$\Delta x = XZ^{-1}(c - A^T y + \mu X^{-1}e - A^T\Delta y), \tag{18.8}$$

which we use to eliminate Δx from (18.6) to get

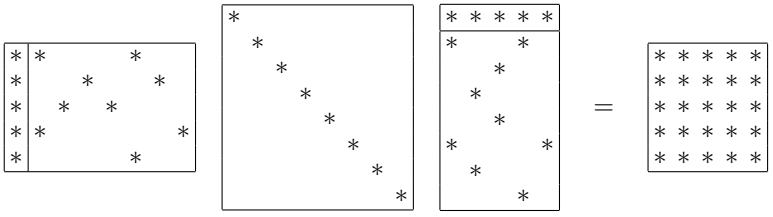
$$\begin{aligned} -(Y^{-1}W + AXZ^{-1}A^T)\Delta y = {} & b - Ax - \mu Y^{-1}e \\ & -AXZ^{-1}(c - A^T y + \mu X^{-1}e). \end{aligned} \tag{18.9}$$

This last system is a system of m equations in m unknowns. It is called the *system of normal equations in primal form*. It is a system of equations involving the matrix $Y^{-1}W + AXZ^{-1}A^T$. The $Y^{-1}W$ term is simply a diagonal matrix, and so the real meat of this matrix is contained in the $AXZ^{-1}A^T$ term.

Given that A is sparse (which is generally the case in real-world linear programs), one would expect the matrix $AXZ^{-1}A^T$ to be likewise sparse. However, we need to investigate the sparsity of $AXZ^{-1}A^T$ (or lack thereof) more closely. Note that the (i,j)th element of this matrix is given by

$$(AXZ^{-1}A^T)_{ij} = \sum_{k=1}^{n} a_{ik}\frac{x_k}{z_k}a_{jk}.$$

That is, the (i,j)th element is simply a weighted inner product of rows i and j of the A matrix. If these rows have disjoint nonzero patterns, then this inner product is guaranteed to be zero, but otherwise it must be treated as a potential nonzero. This is bad news if A is generally sparse but has, say, one dense column:



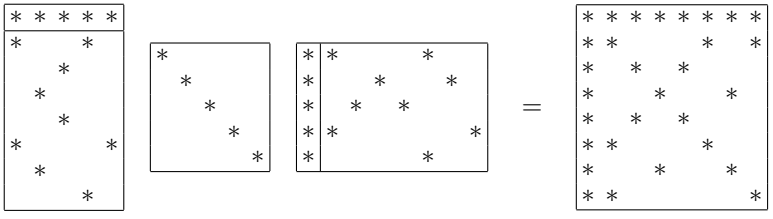
But don't forget that we didn't have to go the primal normal equations route. Instead, we could have chosen the other alternative of solving (18.6) for Δy,

$$\Delta y = -YW^{-1}(b - Ax - \mu Y^{-1}e - A\Delta x),$$

and eliminating it from (18.7):

$$(A^TYW^{-1}A + X^{-1}Z)\Delta x = c - A^Ty + \mu X^{-1}e + A^TYW^{-1}(b - Ax - \mu Y^{-1}e). \tag{18.10}$$

The system defined by (18.10) is a system of n equations in n unknowns. It is called the system of *normal equations in dual form*. Note that dense columns do not pose a problem for these equations. Indeed, for the example given above, we now get



While this system is larger than the one before, it is also sparse, and sparsity almost always is more important than matrix dimensions. In this example, the dense matrix associated with the primal normal equations requires 65 arithmetic operations to factor, whereas the larger, sparser matrix associated with the dual normal equations requires just 60. This is a small difference, but these are small matrices. As the matrices involved get large, factoring a dense matrix takes on the order of n^3 operations, whereas a very sparse matrix might take only on the order of n operations. Clearly, as n gets large, the difference between these becomes quite significant.

It would be great if we could say that it is always best to solve the primal normal equations or the dual normal equations. But as we've just seen, dense columns in A are bad for the primal normal equations and, of course, it follows that dense rows are bad for the dual normal equations. Even worse, some problems have constraint matrices

A that are overall very sparse but contain some dense rows and some dense columns. Such problems are sure to run into trouble with either sets of normal equations. For these reasons, it is best to factor the matrix in the reduced KKT system directly. Then it is possible to find pivot orders that circumvent the difficulties posed by both dense columns and dense rows.

3. Step Direction Decomposition

In the next chapter, we shall discuss factorization techniques for symmetric matrices (along with other implementation issues). However, before we embark on that discussion, we end this chapter by taking a closer look at the formulas for the step direction vectors. To be specific, let us look at Δx. From the primal normal equations (18.9), we can solve for Δy and then substitute the solution into (18.8) to get an explicit formula for Δx:

$$\begin{aligned}(18.11)\quad \Delta x = &\left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)(c - A^Ty + \mu X^{-1}e)\\ &+ D^2A^T(E^{-2} + AD^2A^T)^{-1}(b - Ax - \mu Y^{-1}e),\end{aligned}$$

where we have denoted by D the positive diagonal matrix defined by

$$D^2 = XZ^{-1}$$

and we have denoted by E the positive diagonal matrix defined by

$$E^2 = YW^{-1}$$

(defining these matrices by their squares is possible, since the squares have positive diagonal entries). However, using the dual normal equations, we get

$$\begin{aligned}(18.12)\qquad \Delta x = &\left(A^TE^2A + D^{-2}\right)^{-1}(c - A^Ty + \mu X^{-1}e)\\ &+ \left(A^TE^2A + D^{-2}\right)^{-1}A^TE^2(b - Ax - \mu Y^{-1}e).\end{aligned}$$

These two expressions for Δx look entirely different, but they must be the same, since we know that Δx is uniquely defined by the reduced KKT system. They are indeed the same, as can be shown directly by establishing a certain matrix identity. This is the subject of Exercise 18.1. There are a surprising number of published research papers on interior-point methods that present supposedly new algorithms that are in fact identical to existing methods. These papers get published because the equivalence is not immediately obvious (such as the one we just established).

We can gain further insight into the path-following method by looking more closely at the primal step direction vector. Formula (18.11) can be rearranged as follows:

$$\begin{aligned}\Delta x = &\left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)c \\ &+ \mu\left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)X^{-1}e \\ &- \mu D^2A^T(E^{-2} + AD^2A^T)^{-1}Y^{-1}e \\ &+ D^2A^T(E^{-2} + AD^2A^T)^{-1}(b - Ax) \\ &- D^2A^T\left(I - (E^{-2} + AD^2A^T)^{-1}AD^2A^T\right)y.\end{aligned}$$

For comparison purposes down the road, we prefer to write the $Y^{-1}e$ that appears in the second term containing μ as $E^{-2}W^{-1}e$. Also, using the result of Exercise 18.2, we can rewrite the bulk of the last line as follows:

$$\begin{aligned}\left(I - (E^{-2} + AD^2A^T)^{-1}AD^2A^T\right)y &= (E^{-2} + AD^2A^T)^{-1}E^{-2}y \\ &= (E^{-2} + AD^2A^T)^{-1}w.\end{aligned}$$

Putting this all together, we get

$$\begin{aligned}\Delta x = &\left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)c \\ &+ \mu\left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)X^{-1}e \\ &- \mu D^2A^T(E^{-2} + AD^2A^T)^{-1}E^{-2}W^{-1}e \\ &+ D^2A^T(E^{-2} + AD^2A^T)^{-1}\rho \\ =&\Delta x_{\text{OPT}} + \mu\Delta x_{\text{CTR}} + \Delta x_{\text{FEAS}},\end{aligned}$$

where

$$\begin{aligned}\Delta x_{\text{OPT}} &= \left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)c, \\ \Delta x_{\text{CTR}} &= \left(D^2 - D^2A^T(E^{-2} + AD^2A^T)^{-1}AD^2\right)X^{-1}e \\ &\qquad - D^2A^T(E^{-2} + AD^2A^T)^{-1}E^{-2}W^{-1}e,\end{aligned}$$

and

$$\Delta x_{\text{FEAS}} = D^2A^T(E^{-2} + AD^2A^T)^{-1}\rho.$$

In Chapter 20, we shall show that these components of Δx have important connections to the step directions that arise in a related interior-point method called the affine-scaling method. For now, we simply establish some of their properties as they relate to the path-following method. Our first result is that Δx_{OPT} is an ascent direction.

THEOREM 18.1. $c^T\Delta x_{\text{OPT}} \geq 0$.

PROOF. We use the result of Exercise 18.1 (with the roles of E and D switched) to see that

$$\Delta x_{\text{OPT}} = (A^TE^2A + D^{-2})^{-1}c.$$

Hence,

$$c^T \Delta x_{\text{OPT}} = c^T (A^T E^2 A + D^{-2})^{-1} c.$$

We claim that the right-hand side is obviously nonnegative, since the matrix sandwiched between c and its transpose is positive semidefinite.[1] Indeed, the claim follows from the definition of positive semidefiniteness: a matrix B is *positive semidefinite* if $\xi^T B \xi \geq 0$ for all vectors ξ. To see that the matrix in question is in fact positive semidefinite, we first note that $A^T E^2 A$ and D^{-2} are positive semidefinite:

$$\xi^T A^T E^2 A \xi = \|EA\xi\|^2 \geq 0 \qquad \text{and} \qquad \xi^T D^{-2} \xi = \|D^{-1}\xi\|^2 \geq 0.$$

Then we show that the sum of two positive semidefinite matrices is positive semidefinite and finally that the inverse of a symmetric positive semidefinite matrix is positive semidefinite. To verify closure under summation, suppose that $B^{(1)}$ and $B^{(2)}$ are positive semidefinite, and then compute

$$\xi^T (B^{(1)} + B^{(2)}) \xi = \xi^T B^{(1)} \xi + \xi^T B^{(2)} \xi \geq 0.$$

To verify closure under forming inverses of symmetric positive semidefinite matrices, suppose that B is symmetric and positive semidefinite. Then

$$\xi^T B^{-1} \xi = \xi^T B^{-1} B B^{-1} \xi = (B^{-1}\xi)^T B (B^{-1}\xi) \geq 0,$$

where the inequality follows from the fact that B is positive semidefinite and $B^{-1}\xi$ is simply any old vector. This completes the proof. □

The theorem just proved justifies our referring to Δx_{OPT} as a *step-toward-optimality* direction. We next show that Δx_{FEAS} is in fact a *step-toward-feasibility*.

In Exercise 18.3, you are asked to find the formulas for the primal slack vector's step directions, Δw_{OPT}, Δw_{CTR}, and Δw_{FEAS}. It is easy to verify from these formulas that the pairs $(\Delta x_{\text{OPT}}, \Delta w_{\text{OPT}})$ and $(\Delta x_{\text{CTR}}, \Delta w_{\text{CTR}})$ preserve the current level of infeasibility. That is,

$$A \Delta x_{\text{OPT}} + \Delta w_{\text{OPT}} = 0$$

and

$$A \Delta x_{\text{CTR}} + \Delta w_{\text{CTR}} = 0.$$

Hence, only the "feasibility" directions can improve the degree of feasibility. Indeed, it is easy to check that

$$A \Delta x_{\text{FEAS}} + \Delta w_{\text{FEAS}} = \rho.$$

Finally, we consider Δx_{CTR}. If the objective function were zero (i.e., $c = 0$) and if the current point were feasible, then steps toward optimality and feasibility would vanish and we would be left with just Δx_{CTR}. Since our step directions were derived in an effort to move toward a point on the central path parametrized by μ, we now see that Δx_{CTR} plays the role of a *step-toward-centrality*.

[1] In fact, it's positive definite, but we don't need this stronger property here.

Exercises

18.1 *Sherman–Morrison–Woodbury Formula.* Assuming that all the inverses below exist, show that the following identity is true:

$$(E^{-1} + ADA^T)^{-1} = E - EA(A^T EA + D^{-1})^{-1} A^T E.$$

Use this identity to verify directly the equivalence of the expressions given for Δx in (18.11) and (18.12).

18.2 Assuming that all the inverses exist, show that the following identity holds:

$$I - (E + ADA^T)^{-1} ADA^T = (E + ADA^T)^{-1} E.$$

18.3 Show that

$$\Delta w = \Delta w_{\text{OPT}} + \mu \Delta w_{\text{CTR}} + \Delta w_{\text{FEAS}},$$

where

$$\begin{aligned}
\Delta w_{\text{OPT}} &= -A \left(D^2 - D^2 A^T (E^{-2} + AD^2 A^T)^{-1} AD^2 \right) c, \\
\Delta w_{\text{CTR}} &= -A \left(D^2 - D^2 A^T (E^{-2} + AD^2 A^T)^{-1} AD^2 \right) X^{-1} e \\
&\quad + AD^2 A^T (E^{-2} + AD^2 A^T)^{-1} E^{-2} W^{-1} e,
\end{aligned}$$

and

$$\Delta w_{\text{FEAS}} = \rho - AD^2 A^T (E^{-2} + AD^2 A^T)^{-1} \rho.$$

Notes

The KKT system for general inequality constrained optimization problems was derived by Kuhn & Tucker (1951). It was later discovered that W. Karush had proven the same result in his 1939 master's thesis at the University of Chicago (Karush 1939). John (1948) was also an early contributor to inequality-constrained optimization. Kuhn's survey paper (Kuhn 1976) gives a historical account of the development of the subject.

CHAPTER 19

Implementation Issues

In this chapter, we discuss implementation issues that arise in connection with the path-following method.

The most important issue is the efficient solution of the systems of equations discussed in the previous chapter. As we saw, there are basically three choices, involving either the reduced KKT matrix,

$$B = \begin{bmatrix} -E^{-2} & A \\ A^T & D^{-2} \end{bmatrix}, \tag{19.1}$$

or one of the two matrices associated with the normal equations:

$$AD^2A^T + E^{-2} \tag{19.2}$$

or

$$A^TE^2A + D^{-2}. \tag{19.3}$$

(Here, $E^{-2} = Y^{-1}W$ and $D^{-2} = X^{-1}Z$.)

In the previous chapter, we explained that dense columns/rows are bad for the normal equations and that therefore one might be better off solving the system involving the reduced KKT matrix. But there is also a reason one might prefer to work with one of the systems of normal equations. The reason is that these matrices are positive definite. We shall show in the first section that there are important advantages in working with positive definite matrices. In the second section, we shall consider the reduced KKT matrix and see to what extent the nice properties possessed by positive definite matrices carry over to these matrices.

After finishing our investigations into numerical factorization, we shall take up a few other relevant tasks, such as how one extends the path-following algorithm to handle problems with bounds and ranges.

1. Factoring Positive Definite Matrices

As we saw in the proof of Theorem 18.1, the matrix (19.2) appearing in the primal normal equations is positive semidefinite (and so is (19.3), of course). In fact, it is even better—it's positive definite. A matrix B is *positive definite* if $\xi^TB\xi > 0$ for all vectors $\xi \neq 0$. In this section, we will show that, if we restrict our row/column

reordering to symmetric reorderings, that is, reorderings where the rows and columns undergo the same permutation, then there is no danger of encountering a pivot element whose value is zero. Hence, the row/column permutation can be selected ahead of time based only on the aim of maintaining sparsity.

If we restrict ourselves to symmetric permutations, each pivot element is a diagonal element of the matrix. The following result shows that we can start by picking an arbitrary diagonal element as the first pivot element:

THEOREM 19.1. *If B is positive definite, then $b_{ii} > 0$ for all i.*

The proof follows trivially from the definition of positive definiteness:

$$b_{ii} = e_i^T B e_i > 0.$$

The next step is to show that after each stage of the elimination process, the remaining uneliminated matrix is positive definite. Let us illustrate by breaking out the first row/column of the matrix and looking at what the first step of the elimination process does. Breaking out the first row/column, we write

$$B = \begin{bmatrix} a & b^T \\ b & C \end{bmatrix}.$$

Here, a is the first diagonal element (a scalar), b is the column below a, and C is the matrix consisting of all of B except the first row/column. One step of elimination (as described in Chapter 8) transforms B into

$$\begin{bmatrix} a & b^T \\ b & \boxed{C - \frac{bb^T}{a}} \end{bmatrix}.$$

The following theorem tells us that the uneliminated part is positive definite:

THEOREM 19.2. *If B is positive definite, then so is $C - bb^T/a$.*

PROOF. The fact that B is positive definite implies that

$$\begin{bmatrix} x & y^T \end{bmatrix} \begin{bmatrix} a & b^T \\ b & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = ax^2 + 2y^T bx + y^T Cy \tag{19.4}$$

is positive whenever the scalar x or the vector y is nonzero (or both). Fix a vector $y \neq 0$, and put $x = -\frac{1}{a} b^T y$. Using these choices in (19.4), we get

$$0 < \frac{1}{a} y^T bb^T y - 2\frac{1}{a} y^T bb^T y + y^T Cy = y^T \left(C - \frac{bb^T}{a} \right) y.$$

Since y was an arbitrary nonzero vector, it follows that $C - bb^T/a$ is positive definite. □

Hence, after one step of the elimination, the uneliminated part is positive definite. It follows by induction then that the uneliminated part is positive definite at every step of the elimination.

Here's an example:

$$B = \begin{bmatrix} 2 & -1 & & & -1 \\ -1 & 3 & -1 & -1 & \\ & -1 & 2 & -1 & \\ & -1 & -1 & 3 & -1 \\ -1 & & & -1 & 3 \end{bmatrix}.$$

At the end of the four steps of the elimination (without permutations), we end up with

$$\begin{bmatrix} 2 & -1 & & & -1 \\ -1 & \frac{5}{2} & -1 & -1 & -\frac{1}{2} \\ & -1 & \frac{8}{5} & -\frac{7}{5} & -\frac{1}{5} \\ & -1 & -\frac{7}{5} & \frac{11}{8} & -\frac{11}{8} \\ -1 & -\frac{1}{2} & -\frac{1}{5} & -\frac{11}{8} & 1 \end{bmatrix}.$$

From this eliminated form, we extract the lower triangular matrix, the diagonal matrix, and the upper triangular matrix to write B as

$$B = \begin{bmatrix} 2 & & & & \\ -1 & \frac{5}{2} & & & \\ & -1 & \frac{8}{5} & & \\ & -1 & -\frac{7}{5} & \frac{11}{8} & \\ -1 & -\frac{1}{2} & -\frac{1}{5} & -\frac{11}{8} & 1 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & \frac{5}{2} & & & \\ & & \frac{8}{5} & & \\ & & & \frac{11}{8} & \\ & & & & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & -1 & & & -1 \\ & \frac{5}{2} & -1 & -1 & -\frac{1}{2} \\ & & \frac{8}{5} & -\frac{7}{5} & -\frac{1}{5} \\ & & & \frac{11}{8} & -\frac{11}{8} \\ & & & & 1 \end{bmatrix}.$$

As we saw in Chapter 8, it is convenient to combine the lower triangular matrix with the diagonal matrix to get a new lower triangular matrix with ones on the diagonal. But the current lower triangular matrix is exactly the transpose of the upper triangular matrix. Hence, to preserve symmetry, we should combine the diagonal matrix with both the lower and the upper triangular matrices. Since it only appears once, we must multiply and divide by it (in the middle of the product). Doing this, we get

$$B = \begin{bmatrix} 1 & & & & \\ -\frac{1}{2} & 1 & & & \\ & -\frac{2}{5} & 1 & & \\ & -\frac{2}{5} & -\frac{7}{8} & 1 & \\ -\frac{1}{2} & -\frac{1}{5} & -\frac{1}{8} & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & \frac{5}{2} & & & \\ & & \frac{8}{5} & & \\ & & & \frac{11}{8} & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & & & -\frac{1}{2} \\ & 1 & -\frac{2}{5} & -\frac{2}{5} & -\frac{1}{5} \\ & & 1 & -\frac{7}{8} & -\frac{1}{8} \\ & & & 1 & -1 \\ & & & & 1 \end{bmatrix}.$$

The lower triangular matrix in this representation is usually denoted by L and the diagonal matrix by D (not to be confused with the D at the beginning of the chapter).

Hence, this factorization can be summarized as

$$B = LDL^T$$

and is referred to as an LDL^T*-factorization*. Of course, once a factorization is found, it is easy to solve systems of equations using forward and backward substitution as discussed in Chapter 8.

1.1. Stability. We began our discussion of factoring positive definite matrices with the comment that a symmetric permutation can be chosen purely with the aim of preserving sparsity, since it is guaranteed that no pivot element will ever vanish. However, the situation is even better than that—we can show that whenever a pivot element is small, so is every other nonzero in the uneliminated part of the same row/column. Before saying why, we need to set down a few technical results.

THEOREM 19.3. *If $\bar{b}_{ii}$ denotes a diagonal element in the uneliminated submatrix at some stage of an elimination and b_{ii} denotes the original value of that diagonal element, then $0 < \bar{b}_{ii} \leq b_{ii}$.*

PROOF. The positivity of $\bar{b}_{ii}$ follows from the fact the uneliminated submatrix is positive definite. The fact that it is bounded above by b_{ii} follows from the fact that each step of the elimination can only decrease diagonal elements, which can be seen by looking at the first step of the elimination. Using the notation introduced just after Theorem 19.1,

$$c_{ii} - \frac{b_i^2}{a} \leq c_{ii}.$$

□

THEOREM 19.4. *If B is symmetric and positive definite, then $|b_{ij}| < \sqrt{b_{ii}b_{jj}}$ for all $i \neq j$.*

PROOF. Fix $i \neq j$ and let $\xi = re_i + e_j$. That is, ξ is the vector that's all zero except for the ith and jth position, where it's r and 1, respectively. Then,

$$0 < \xi^T B \xi = b_{ii}r^2 + 2b_{ij}r + b_{jj},$$

for all $r \in \mathbb{R}$. This quadratic expression is positive for all values of r if and only if it is positive at its minimum, and it's easy to check that it is positive at that point if and only if $|b_{ij}| < \sqrt{b_{ii}b_{jj}}$. □

These two theorems, together with the fact that the uneliminated submatrix is symmetric and positive definite, give us bounds on the off-diagonal elements. Indeed, consider the situation after a number of steps of the elimination. Using bars to denote matrix elements in the uneliminated submatrix and letting M denote an upper bound on the diagonal elements before the elimination process began (which, without loss of generality, could be taken as 1), we see that, if $\bar{b}_{jj} < \epsilon$, then

$$\bar{b}_{ij} < \sqrt{\epsilon M}. \tag{19.5}$$

This bound is exceedingly important and is special to positive definite matrices.

2. Quasidefinite Matrices

In this section, we shall study factorization techniques for the reduced KKT matrix (19.1). The reduced KKT matrix is an example of a quasidefinite matrix. A symmetric matrix is called *quasidefinite* if it can be written (perhaps after a symmetric permutation) as

$$B = \begin{bmatrix} -E & A \\ A^T & D \end{bmatrix},$$

where E and D are positive definite matrices. Quasidefinite matrices inherit some of the nice properties of positive definite matrices. In particular, one can perform an arbitrary symmetric permutation of the rows/columns and still be able to form a factorization of the permuted matrix.

The idea is that, after each step of the elimination, the remaining uneliminated part of the matrix is still quasidefinite. To see why, let's break out the first row/column of the matrix and look at the first step of the elimination process. Breaking out the first row/column of B, we write

$$B = \begin{bmatrix} -a & -b^T & f^T \\ -b & -C & G \\ f & G^T & D \end{bmatrix},$$

where a is a scalar, b and f are vectors, and C, D, and G are matrices (of the appropriate dimensions). One step of the elimination process transforms B into

$$\left[\begin{array}{c|cc} -a & -b^T & f^T \\ \hline -b & -\left(C - \frac{bb^T}{a}\right) & G + \frac{bf^T}{a} \\ f & G^T + \frac{fb^T}{a} & D + \frac{ff^T}{a} \end{array}\right].$$

The uneliminated part is

$$\begin{bmatrix} -\left(C - \frac{bb^T}{a}\right) & G + \frac{bf^T}{a} \\ G^T + \frac{fb^T}{a} & D + \frac{ff^T}{a} \end{bmatrix}.$$

Clearly, the lower-left and upper-right blocks are transposes of each other. Also, the upper-left and lower-right blocks are symmetric, since C and D are. Therefore, the whole matrix is symmetric. Theorem 19.2 tells us that $C - bb^T/a$ is positive definite and $D + ff^T/a$ is positive definite, since the sum of a positive definite matrix and a positive semidefinite matrix is positive definite (see Exercise 19.2). Therefore, the uneliminated part is indeed quasidefinite.

Of course, had the first pivot element been selected from the submatrix D instead of E, perhaps the story would be different. But it is easy to check that it's the same.

Hence, no matter which diagonal element is selected to be the first pivot element, the resulting uneliminated part is quasidefinite. Now, by induction it follows that every step of the elimination process involves choosing a pivot element from the diagonals of a quasidefinite matrix. Since these diagonals come from either a positive definite submatrix or the negative of such a matrix, it follows that they are always nonzero (but many of them will be negative). Therefore, just as for positive definite matrices, an arbitrary symmetric permutation of a quasidefinite matrix can be factored without any risk of encountering a zero pivot element.

Here's an example:

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & & & -2 & 1 \\ & -2 & & & 2 \\ & & -3 & 1 & \\ -2 & & 1 & 2 & \\ 1 & 2 & & & 1 \end{bmatrix} \end{array}. \tag{19.6}$$

(The blocks are easy to pick out, since the negative diagonals must be from $-E$, whereas the positive ones are from D.) Let's eliminate by picking the diagonals in the order $1, 5, 2, 4, 3$. No permutations are needed in preparation for the first step of the elimination. After this step, we have

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -2 & 1 \\ \hline & -2 & & & 2 \\ & & -3 & 1 & \\ -2 & & 1 & 6 & -2 \\ 1 & 2 & & -2 & 2 \end{array}\right] \end{array}.$$

Now, we move row/column 5 to the pivot position, slide the other rows/columns down/over, and eliminate to get

$$\begin{array}{c} \\ 1 \\ 5 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 5 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & 1 & & & -2 \\ 1 & 2 & 2 & & -2 \\ & 2 & -4 & & 2 \\ & & & -3 & 1 \\ -2 & -2 & 2 & 1 & 4 \end{array}\right] \end{array}.$$

Row/column 2 is in the correct position for the third step of the elimination, and therefore, without further ado, we do the next step in the elimination:

$$\begin{array}{c} \\ 1 \\ 5 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 5 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & 1 & & & -2 \\ 1 & 2 & 2 & & -2 \\ & 2 & -4 & & 2 \\ & & & -3 & 1 \\ -2 & -2 & 2 & 1 & 5 \end{array}\right] \end{array}.$$

Finally, we interchange rows/columns 3 and 4 and do the last elimination step to get

$$\begin{array}{c} \\ 1 \\ 5 \\ 2 \\ 4 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 5 & 2 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & 1 & & -2 & \\ 1 & 2 & 2 & -2 & \\ & 2 & -4 & 2 & \\ -2 & -2 & 2 & 5 & 1 \\ & & & 1 & -\frac{16}{5} \end{array}\right] \end{array}.$$

From this final matrix, it is easy to extract the LDL^T-factorization of the permutation of B:

$$B = \begin{array}{c} \\ 1 \\ 5 \\ 2 \\ 4 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 5 & 2 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & 1 & & -2 & \\ 1 & 1 & 2 & & \\ & 2 & -2 & & \\ -2 & & & 2 & 1 \\ & & & 1 & -3 \end{array}\right] \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 5 \\ 2 \\ 4 \\ 3 \end{array} \begin{array}{c} \\ \left[\begin{array}{ccccc} 1 & & & & \\ -1 & 1 & & & \\ & 1 & 1 & & \\ 2 & -1 & -\frac{1}{2} & 1 & \\ & & & \frac{1}{5} & 1 \end{array}\right] \end{array} \begin{array}{c} \\ \left[\begin{array}{ccccc} -1 & & & & \\ & 2 & & & \\ & & -4 & & \\ & & & 5 & \\ & & & & -\frac{16}{5} \end{array}\right] \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 5 & 2 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} 1 & -1 & & 2 & \\ & 1 & 1 & -1 & \\ & & 1 & -\frac{1}{2} & \\ & & & 1 & \frac{1}{5} \\ & & & & 1 \end{array}\right] \end{array}.$$

As always, the factorization is computed so that one can solve systems of equations. Given the factorization, all that is required is a forward substitution, a backward substitution, a scaling, and two permutations.

2.1. Instability. We have shown that an arbitrary symmetric permutation of the rows/columns of a quasidefinite matrix can be factored into LDL^T. That is, mathematically none of the pivot elements will be zero. But they can be small, and their smallness can cause troubles not encountered with positive definite matrices. To explain, let's look at an example. Consider the linear programming problem

$$\begin{aligned} \text{maximize} \quad & x_1 + \ x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \le 1 \\ & 2x_1 + \ x_2 \le 1 \\ & x_1, \ x_2 \ge 0 \end{aligned}$$

and its dual

$$\begin{aligned} \text{minimize} \quad & y_1 + \ y_2 \\ \text{subject to} \quad & y_1 + 2y_2 \ge 1 \\ & 2y_1 + \ y_2 \ge 1 \\ & y_1, \ y_2 \ge 0. \end{aligned}$$

Drawing pictures, it is easy to see that the optimal solution is

$$\begin{aligned} x_1^* = x_2^* = y_1^* = y_2^* = \frac{1}{3} \\ z_1^* = z_2^* = w_1^* = w_2^* = 0. \end{aligned}$$

Therefore, as the optimal solution is approached, the diagonal elements in $X^{-1}Z$ and $Y^{-1}W$ approach zero. Therefore, at late stages in the path-following method, one is faced with the problem of factoring the following reduced KKT matrix:

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} -\epsilon_1 & & 1 & 2 \\ & -\epsilon_2 & 2 & 1 \\ 1 & 2 & \delta_1 & \\ 2 & 1 & & \delta_2 \end{bmatrix} \end{array},$$

where ϵ_1, ϵ_2, δ_1, and δ_2 are small positive numbers. Consider what happens if we permute the rows/columns so that the original diagonal elements appear in the following order: $1, 3, 4, 2$. The permuted matrix is seen to be

$$B = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 3 & 4 & 2 \end{array} \\ \begin{bmatrix} -\epsilon_1 & 1 & 2 & \\ 1 & \delta_1 & & 2 \\ 2 & & \delta_2 & 1 \\ & 2 & 1 & -\epsilon_2 \end{bmatrix} \end{array}.$$

After the first two steps of the elimination, we have

$$\left[\begin{array}{ccc|cc} -\epsilon_1 & 1 & & 2 & \\ 1 & \delta_1+\frac{1}{\epsilon_1} & & \frac{2}{\epsilon_1} & 2 \\ 2 & \frac{2}{\epsilon_1} & & (\delta_2+\frac{4}{\epsilon_1})-\frac{4/\epsilon_1^2}{(\delta_1+\frac{1}{\epsilon_1})} & 1-\frac{4/\epsilon_1}{(\delta_1+\frac{1}{\epsilon_1})} \\ & 2 & & 1-\frac{4/\epsilon_1}{(\delta_1+\frac{1}{\epsilon_1})} & -\epsilon_2-\frac{4}{(\delta_1+\frac{1}{\epsilon_1})} \end{array}\right]. \tag{19.7}$$

Using exact arithmetic, the uneliminated part simplifies to

$$\left[\begin{array}{cc} \delta_2+\frac{4\delta_1}{1+\epsilon_1\delta_1} & 1-\frac{4}{1+\epsilon_1\delta_1} \\ 1-\frac{4}{1+\epsilon_1\delta_1} & -\epsilon_2-\frac{4\epsilon_1}{1+\epsilon_1\delta_1} \end{array}\right].$$

Here, the diagonal elements are small but not zero and the off-diagonal elements are close to -3. But on a computer that stores numbers with finite precision, the computation comes out quite differently when the ϵ_i's and the δ_i's are smaller than the square root of the machine precision (i.e., about 10^{-8} for double-precision floating-point numbers). In the elimination process, the parenthesized expressions in (19.7) are evaluated before the other operations. Hence, in finite precision arithmetic, these expressions produce

$$\delta_2+\frac{4}{\epsilon_1}=\frac{4}{\epsilon_1} \qquad \text{and} \qquad \delta_1+\frac{1}{\epsilon_1}=\frac{1}{\epsilon_1},$$

and so (19.7) becomes

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 3 & 4 & 2 \end{array} \\ \left[\begin{array}{cccc} -\epsilon_1 & 1 & 2 & \\ 1 & \frac{1}{\epsilon_1} & \frac{2}{\epsilon_1} & 2 \\ 2 & \frac{2}{\epsilon_1} & 0 & -3 \\ & 2 & -3 & -4\epsilon_1 \end{array}\right] \end{array},$$

which clearly presents a problem for the next step in the elimination process.

Now let's consider what happens if the elimination process is applied directly to B without permuting the rows/columns. Sparing the reader the tedious details, the end result of the elimination is the following matrix:

$$\left[\begin{array}{cccc} -\epsilon_1 & & 1 & 2 \\ & -\epsilon_2 & 2 & 1 \\ 1 & 2 & \delta_1+\frac{1}{\epsilon_1}+\frac{4}{\epsilon_2} & \frac{2}{\epsilon_1}+\frac{2}{\epsilon_2} \\ 2 & 1 & \frac{2}{\epsilon_1}+\frac{2}{\epsilon_2} & \delta_2+\frac{4}{\epsilon_1}+\frac{1}{\epsilon_2}-\frac{(\frac{2}{\epsilon_1}+\frac{2}{\epsilon_2})^2}{(\delta_1+\frac{1}{\epsilon_1}+\frac{4}{\epsilon_2})} \end{array}\right]. \tag{19.8}$$

As before, in finite precision arithmetic, certain small numbers get lost:

$$\delta_2 + \frac{4}{\epsilon_1} = \frac{4}{\epsilon_1} \qquad \text{and} \qquad \delta_1 + \frac{1}{\epsilon_1} = \frac{1}{\epsilon_1}.$$

Making these substitutions in (19.8), we see that the final matrix produced by the elimination process using finite precision arithmetic is

$$\begin{bmatrix} -\epsilon_1 & & 1 & 2 \\ & -\epsilon_2 & 2 & 1 \\ 1 & 2 & \frac{1}{\epsilon_1} + \frac{4}{\epsilon_2} & \frac{2}{\epsilon_1} + \frac{2}{\epsilon_2} \\ 2 & 1 & \frac{2}{\epsilon_1} + \frac{2}{\epsilon_2} & 0 \end{bmatrix}.$$

Just as before, the fact that small numbers got lost has resulted in a zero appearing on the diagonal where a small but nonzero (in this case positive) number belongs. However, the situation is fundamentally different this time. With the first ordering, a -3 remained to be eliminated under the zero diagonal element, whereas with the second ordering, this did not happen. Of course, it didn't happen in this particular example because the 0 appeared as the last pivot element, which has no elements below it to be eliminated. But that is not the general reason why the second ordering does not produce nonzeros under zero pivot elements. In general, a zero (which should be a small positive) can appear anywhere in the lower-right block (relative to the original quasidefinite partitioning). But once the elimination process gets to this block, the remaining uneliminated part of the matrix is positive definite. Hence, the estimate in (19.5) can be used to tell us that all the nonzeros below a zero diagonal are in fact small themselves. A zero appearing on the diagonal only presents a problem if there are nonzeros below it that need to be eliminated. If there are none, then the elimination can simply proceed to the next pivot element (see Exercise 19.1).

Let's summarize the situation. We showed in the last chapter that the possibility of dense rows/columns makes it unattractive to work strictly with the normal equations. Yet, although the quasidefinite reduced KKT system can be used, it is numerically less stable. A compromise solution seems to be suggested. One could take a structured approach to the reordering heuristic. In the structured approach, one decides first whether it seems better to begin pivoting with elements from the upper-left block or from the lower-right block. Once this decision is made, one should pivot out all the diagonal elements from this block before working on the other block, with the exception that pivots involving dense rows/columns be deferred to the end of the elimination process. If no dense columns are identified, this strategy mimics the normal equations approach. Indeed, after eliminating all the diagonal elements in the upper-left block, the remaining uneliminated lower-right block contains exactly the matrix for the system of dual normal equations. Similarly, had the initial choice been to pivot out all the diagonal elements from the lower-right block, then the remaining uneliminated upper-left block becomes the matrix for the system of primal normal equations.

With this structured approach, if no dense rows/columns are identified and deferred, then the elimination process is numerically stable. If, on the other hand, some dense rows/columns are deferred, then the factorization is less stable. But in practice, this approach seems to work well. Of course, one could be more careful and monitor the diagonal elements. If a diagonal element gets small (relative to the other uneliminated nonzeros in the same row/column), then one could flag it and then calculate a new ordering in which such pivot elements are deferred to the end of the elimination process.

3. Problems in General Form

In this section, we describe how to adapt the path-following algorithm to solving problems presented in the following general form:

$$\begin{array}{lrcl} \text{maximize} & & c^T x & \\ \text{subject to} & a \leq & Ax & \leq b \\ & l \leq & x & \leq u. \end{array} \tag{19.9}$$

As in Chapter 9, some of the data elements are allowed to take on infinite values. However, let us consider first the case where all the components of a, b, l, and u are finite. Infinities require special treatment, which shall be discussed shortly.

Following the derivation of the path-following method that we introduced in Chapter 17, the first step is to introduce slack variables as appropriate to replace all inequality constraints with simple nonnegativity constraints. Hence, we rewrite the primal problem (19.9) as follows:

$$\begin{array}{lrcl} \text{maximize} & c^T x & & \\ \text{subject to} & Ax + f & = & b \\ & -Ax + p & = & -a \\ & x + t & = & u \\ & -x + g & = & -l \\ & f,\ p,\ t,\ g & \geq & 0. \end{array}$$

In Chapter 9, we showed that the dual problem is given by

$$\begin{array}{lrcl} \text{minimize} & b^T v - a^T q + u^T s - l^T h & & \\ \text{subject to} & A^T(v-q) - (h-s) & = & c \\ & v,\ q,\ s,\ h & \geq & 0, \end{array}$$

and the corresponding complementarity conditions are given by

$$\begin{aligned} f_i v_i &= 0 \qquad i = 1, 2, \ldots, m, \\ p_i q_i &= 0 \qquad i = 1, 2, \ldots, m, \\ t_j s_j &= 0 \qquad j = 1, 2, \ldots, n, \\ g_j h_j &= 0 \qquad j = 1, 2, \ldots, n. \end{aligned}$$

The next step in the derivation is to introduce the primal–dual central path, which we parametrize as usual by a positive real parameter μ. Indeed, for each $\mu > 0$, we define the associated central-path point in primal–dual space as the unique point that simultaneously satisfies the conditions of primal feasibility, dual feasibility, and μ-complementarity. Ignoring nonnegativity (which is enforced separately), these conditions are

$$\begin{aligned} Ax + f &= b \\ f + p &= b - a \\ x + t &= u \\ -x + g &= -l \end{aligned}$$

$$\begin{aligned} A^T y + s - h &= c \\ y + q - v &= 0 \end{aligned}$$

$$\begin{aligned} FVe &= \mu e \\ PQe &= \mu e \\ TSe &= \mu e \\ GHe &= \mu e. \end{aligned}$$

Note that we have replaced the primal feasibility condition, $-Ax + p = -a$, with the equivalent condition that $f + p = b - a$, and we have introduced into the dual problem new variables y defined by $y = v - q$. The reason for these changes is to put the system of equations into a form in which A and A^T appear as little as possible (so that solving the system of equations for the step direction will be as efficient as possible).

The last four equations are the μ-complementarity conditions. As usual, each upper case letter that appears on the left in these equations denotes the diagonal matrix having the components of the corresponding lower-case vector on its diagonal. The system is a nonlinear system of $5n + 5m$ equations in $5n + 5m$ unknowns. It has a unique solution in the strict interior of the following subset of primal–dual space:

$$\{(x, f, p, t, g, y, v, q, s, h) : f, p, t, g, v, q, s, h \geq 0\}. \tag{19.10}$$

This fact can be seen by noting that these equations are the first-order optimality conditions for an associated strictly convex barrier problem.

As μ tends to zero, the central path converges to the optimal solution to both the primal and dual problems. The path-following algorithm is defined as an iterative process that starts from a point in the strict interior of (19.10), estimates at each iteration a value of μ representing a point on the central path that is in some sense closer to the optimal solution than the current point, and then attempts to step toward this central-path point, making sure that the new point remains in the strict interior of the the set given in (19.10).

Suppose for the moment that we have already decided on the target value for μ. Let $(x, \ldots, h)$ denote the current point in the strict interior of (19.10), and let $(x + \Delta x, \ldots, h + \Delta h)$ denote the point on the central path corresponding to the target value of μ. The defining equations for the point on the central path can be written as

$$
\begin{aligned}
A\Delta x + \Delta f &= b - Ax - f && =: \rho \\
\Delta f + \Delta p &= b - a - f - p && =: \alpha \\
\Delta x + \Delta t &= u - x - t && =: \tau \\
-\Delta x + \Delta g &= -l + x - g && =: \nu \\
A^T\Delta y + \Delta s - \Delta h &= c - A^T y - s + h && =: \sigma \\
\Delta y + \Delta q - \Delta v &= -y - q + v && =: \beta \\
FV^{-1}\Delta v + \Delta f &= \mu V^{-1}e - f - V^{-1}\Delta V\Delta f && =: \gamma_f \\
QP^{-1}\Delta p + \Delta q &= \mu P^{-1}e - q - P^{-1}\Delta P\Delta q && =: \gamma_q \\
ST^{-1}\Delta t + \Delta s &= \mu T^{-1}e - s - T^{-1}\Delta T\Delta s && =: \gamma_s \\
HG^{-1}\Delta g + \Delta h &= \mu G^{-1}e - h - G^{-1}\Delta G\Delta h && =: \gamma_h,
\end{aligned}
$$

where we have introduced notations $\rho, \ldots, \gamma_h$ as shorthands for the right-hand side expressions. This is almost a linear system for the direction vectors $(\Delta x, \ldots, \Delta h)$. The only nonlinearities appear on the right-hand sides of the complementarity equations (i.e., in $\gamma_f, \ldots, \gamma_h$). As we saw before, Newton's method suggests that we simply drop these nonlinear terms to obtain a linear system for the "delta" variables.

Clearly, the main computational burden is to solve the system shown above. It is important to note that this is a large, sparse, indefinite, linear system. It is also

symmetric if one negates certain rows and rearranges rows and columns appropriately:

$$\left[\begin{array}{cccccc|cccccc}
-FV^{-1} & & & & & & & -I & & & & \\
 & & & & & & I & & & & & I \\
 & & & & & & -I & & & & I & \\
 & & & & & & & I & I & & & \\
 & & & & & & A & I & & & & \\
\hline
 & I & -I & & A^T & & & & & & & \\
 -I & & & I & I & & & & & & & \\
 & & & I & & & & QP^{-1} & & & & \\
 & & I & & & & & & HG^{-1} & & & \\
 & I & & & & & & & & ST^{-1} & &
\end{array}\right]
\left[\begin{array}{c}
\Delta v \\ \Delta s \\ \Delta h \\ \Delta q \\ \Delta y \\ \hline \Delta x \\ \Delta f \\ \Delta p \\ \Delta g \\ \Delta t
\end{array}\right]
=
\left[\begin{array}{c}
-\gamma_f \\ \tau \\ \nu \\ \alpha \\ \rho \\ \hline \sigma \\ \beta \\ \gamma_q \\ \gamma_h \\ \gamma_s
\end{array}\right].$$

Because this system is symmetric, we look for symmetric pivots to solve it. That is, we choose our pivots only from the diagonal elements. It turns out that we can eliminate Δv, Δp, Δg, and Δt using the nonzero diagonals $-FV^{-1}$, QP^{-1}, HG^{-1}, and ST^{-1}, respectively, in any order without causing any nondiagonal fill-in. Indeed, the equations for Δv, Δp, Δg, and Δt are

$$\begin{aligned}
\Delta v &= VF^{-1}(\gamma_f - \Delta f) \\
\Delta p &= PQ^{-1}(\gamma_q - \Delta q) \\
\Delta g &= GH^{-1}(\gamma_h - \Delta h) \\
\Delta t &= TS^{-1}(\gamma_s - \Delta s),
\end{aligned} \tag{19.11}$$

and after elimination from the system, we get

$$\left[\begin{array}{cccc|cc}
-TS^{-1} & & & & I & \\
 & -GH^{-1} & & & -I & \\
 & & -PQ^{-1} & & & I \\
 & & & & A & I \\
\hline
 I & -I & & A^T & & \\
 & & I & I & & VF^{-1}
\end{array}\right]
\left[\begin{array}{c}
\Delta s \\ \Delta h \\ \Delta q \\ \Delta y \\ \hline \Delta x \\ \Delta f
\end{array}\right]
=
\left[\begin{array}{c}
\hat{\tau} \\ \hat{\nu} \\ \hat{\alpha} \\ \rho \\ \hline \sigma \\ \hat{\beta}
\end{array}\right],$$

where again we have introduced abbreviated notations for the components of the right-hand side:

$$\begin{aligned}\hat{\tau} &= \tau - TS^{-1}\gamma_s \\ \hat{\nu} &= \nu - GH^{-1}\gamma_h \\ \hat{\alpha} &= \alpha - PQ^{-1}\gamma_q \\ \hat{\beta} &= \beta + VF^{-1}\gamma_f.\end{aligned}$$

Next we use the pivot elements $-TS^{-1}$, $-GH^{-1}$, and $-PQ^{-1}$ to solve for Δs, Δh, and Δq, respectively:

$$\begin{aligned}\Delta s &= -ST^{-1}(\hat{\tau} - \Delta x) \\ \Delta h &= -HG^{-1}(\hat{\nu} + \Delta x) \\ \Delta q &= -QP^{-1}(\hat{\alpha} - \Delta f).\end{aligned} \tag{19.12}$$

After eliminating these variables, the system simplifies to

$$\left[\begin{array}{c|cc} & A & I \\ \hline A^T & D & \\ I & & E \end{array}\right] \left[\begin{array}{c} \Delta y \\ \hline \Delta x \\ \Delta f \end{array}\right] = \left[\begin{array}{c} \rho \\ \hline \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \\ \hat{\beta} + QP^{-1}\hat{\alpha}. \end{array}\right],$$

where

$$D = ST^{-1} + HG^{-1}$$

and

$$E = VF^{-1} + QP^{-1}.$$

Finally, we use the pivot element E to solve for Δf,

$$\Delta f = E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha} - \Delta y), \tag{19.13}$$

which brings us to the reduced KKT equations:

$$\left[\begin{array}{c|c} -E^{-1} & A \\ \hline A^T & D \end{array}\right] \left[\begin{array}{c} \Delta y \\ \hline \Delta x \end{array}\right] = \left[\begin{array}{c} \rho - E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha}) \\ \hline \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \end{array}\right]. \tag{19.14}$$

Up to this point, none of the eliminations have produced any off-diagonal fill-in. Also, the matrix for system given in (19.14) is a symmetric quasidefinite matrix. Hence, the techniques given in Section 18.2 for solving such systems can be used. The algorithm is summarized in 19.1.

initialize $(x, f, p, t, g, y, v, q, s, h)$ such that $f, p, t, g, v, q, s, h > 0$

while (not optimal) {

$$\rho = b - Ax - w$$
$$\sigma = c - A^T y + z$$
$$\gamma = f^T v + p^T q + t^T s + g^T h$$
$$\mu = \delta \frac{\gamma}{n+m}$$
$$\gamma_f = \mu V^{-1} e - f$$
$$\gamma_q = \mu P^{-1} e - q$$
$$\gamma_s = \mu T^{-1} e - s$$
$$\gamma_h = \mu G^{-1} e - h$$
$$\hat{\tau} = u - x - t - TS^{-1}\gamma_s$$
$$\hat{\nu} = -l + x - g - GH^{-1}\gamma_h$$
$$\hat{\alpha} = b - a - f - p - PQ^{-1}\gamma_q$$
$$\hat{\beta} = -y - q + v + VF^{-1}\gamma_f$$
$$D = ST^{-1} + HG^{-1}$$
$$E = VF^{-1} + QP^{-1}$$

solve: $\begin{bmatrix} -E^{-1} & A \\ A^T & D \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} \rho - E^{-1}(\hat{\beta} + QP^{-1}\hat{\alpha}) \\ \sigma + ST^{-1}\hat{\tau} - HG^{-1}\hat{\nu} \end{bmatrix}$

compute: Δf using (19.13), Δs, Δh, Δq using (19.12),
and Δv, Δp, Δg, Δt using (19.11)

$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta f_j}{f_j}, -\frac{\Delta p_i}{p_i}, -\frac{\Delta t_i}{t_i}, -\frac{\Delta g_j}{g_j}, -\frac{\Delta v_j}{v_j}, -\frac{\Delta q_i}{q_i}, -\frac{\Delta s_i}{s_i}, -\frac{\Delta h_j}{h_j} \right\} \right)^{-1} \wedge 1$$

$$x \leftarrow x + \theta \Delta x, \quad y \leftarrow y + \theta \Delta y, \quad f \leftarrow f + \theta \Delta f, \quad v \leftarrow v + \theta \Delta v$$
$$p \leftarrow p + \theta \Delta p, \quad q \leftarrow q + \theta \Delta q, \quad t \leftarrow t + \theta \Delta t, \quad s \leftarrow s + \theta \Delta s$$
$$g \leftarrow g + \theta \Delta g, \quad h \leftarrow h + \theta \Delta h$$

}

FIGURE 19.1. The path-following method—general form.

Exercises

19.1 The matrix

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

is not positive definite but is positive semidefinite. Find a factorization $B = LDL^T$, where L is lower triangular with ones on the diagonal and D is a diagonal matrix with nonnegative diagonal elements. If such a factorization exists for every symmetric positive semidefinite matrix, explain why. If not, give a counterexample.

19.2 Show that the sum of a positive definite matrix and a positive semidefinite matrix is positive definite.

19.3 Permute the rows/columns of the matrix B given in (19.6) so that the diagonal elements from B appear in the order $2, 3, 4, 5, 1$. Compute an LDL^T-factorization of this matrix.

19.4 Show that, if B is symmetric and positive semidefinite, then $|b_{ij}| \leq \sqrt{b_{ii} b_{jj}}$ for all i, j.

Notes

Most implementations of interior-point methods assume the problem to be formulated with equality constraints. In this formulation, Lustig et al. (1994) give a good overview of the performance of interior-point algorithms compared with the simplex method.

The suggestion that it is better to solve equations in the KKT form instead of normal form was offered independently by a number of researchers (Gill et al. 1992, Turner 1991, Fourer & Mehrotra 1991, Vanderbei & Carpenter 1993).

The advantages of the primal–dual symmetric formulation were first reported in Vanderbei (1994). The basic properties of quasidefinite matrices were first given in Vanderbei (1995).

CHAPTER 20

The Affine-Scaling Method

In the previous chapter, we showed that the step direction for the path-following method can be decomposed into a linear combination of three directions: a direction toward optimality, a direction toward feasibility, and a direction toward centrality. It turns out that these directions, or minor variants of them, arise in all interior-point methods.

Historically, one of the first interior-point methods to be invented, analyzed, and implemented was a two-phase method in which Phase I uses only the feasibility direction and Phase II uses only the optimality direction. This method is called *affine scaling*. While it is no longer considered the method of choice for practical implementations, it remains important because its derivation provides valuable insight into the nature of the three basic directions mentioned above.

In this chapter, we shall explain the affine-scaling principle and use it to derive the step toward optimality and step toward feasibility directions. As always, our main interest lies in problems presented in standard form. But for affine scaling, it is easier to start by considering problems in equality form. Hence, we begin by assuming that the linear programming problem is given as

$$\begin{aligned} &\text{maximize } && c^T x \\ &\text{subject to } && Ax = b \\ &&& x \geq 0. \end{aligned} \tag{20.1}$$

We shall begin with the Phase II algorithm. Hence, we assume that we have a feasible initial starting point, x^0. For the affine-scaling method, it is important that this starting point lie in the strict interior of the feasible set. That is, we assume that

$$Ax^0 = b \quad \text{and} \quad x^0 > 0.$$

1. The Steepest Ascent Direction

Since the affine-scaling principle is fundamentally geometric, it is useful to keep a picture in mind. A typical picture for $m = 1$ and $n = 3$ is shown in Figure 20.1. The ultimate goal is, of course, to move from x^0 to the optimal solution x^*. However, the short-term goal is to move from x^0 in some direction Δx that improves the objective function. Such a direction is called an *ascent direction*. You probably recall from

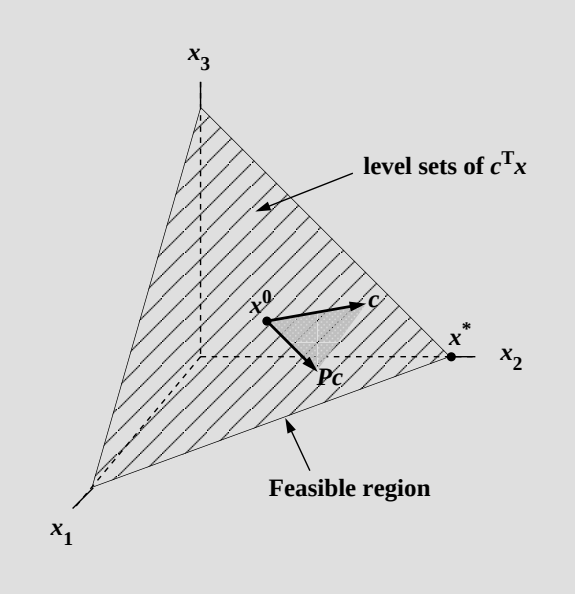


FIGURE 20.1. A typical feasible region when the problem is in equality form, $m = 1$, and $n = 3$. The lines drawn on the feasible set represent level sets of the objective function, and x^0 represents the starting point for the affine-scaling method.

elementary calculus that the best, i.e., steepest, ascent direction is given by the gradient of the objective function. However, as we see in Figure 20.1, there is no reason a priori for the gradient to "lie in" the feasible region. Hence, the steepest ascent direction will almost surely cause a move to infeasible points. This is also clear algebraically. Indeed,

$$A(x^0 + \Delta x) = Ax^0 + A\Delta x = b + Ac \neq b$$

(unless Ac = 0 which is not likely).

To see how to find a better direction, let us first review in what sense the gradient is the steepest ascent direction. The *steepest ascent direction* is defined to be the direction that gives the greatest increase in the objective function subject to the constraint that the displacement vector has unit length. That is, the steepest ascent direction is the solution to the following optimization problem:

$$\begin{aligned} &\text{maximize } c^T(x^0 + \Delta x) \\ &\text{subject to } \|\Delta x\|^2 = 1. \end{aligned} \tag{20.2}$$

We can solve this problem using Lagrange multipliers. Indeed, if we let λ denote the Lagrange multiplier for the constraint, the problem becomes

$$\max_{\Delta x, \lambda} \; c^T(x^0 + \Delta x) - \lambda(\Delta x^T \Delta x - 1).$$

Differentiating with respect to Δx and setting the derivative to zero, we get

$$c - 2\lambda \Delta x = 0,$$

which implies that

$$\Delta x = \frac{1}{2\lambda} c \propto c.$$

Then differentiating the Lagrangian with respect to λ and setting that derivative to zero, we see that

$$\|\Delta x\|^2 - 1 = 0,$$

which implies that

$$\|\Delta x\| = \pm 1.$$

Hence, the steepest ascent direction points in the direction of either c or its negative. Since the negative is easily seen not to be an ascent direction at all, it follows that the steepest ascent direction points in the direction of c.

2. The Projected Gradient Direction

The problem with the steepest ascent direction is that it fails to preserve feasibility. That is, it fails to preserve the equality constraints $Ax = b$. To remedy this problem, let's add these constraints to (20.2) so that we get the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & c^T(x^0 + \Delta x) \\ \text{subject to} \quad & \|\Delta x\|^2 = 1 \\ & A(x^0 + \Delta x) = b. \end{aligned}$$

Again, the method of Lagrange multipliers is the appropriate tool. As before, let λ denote the Lagrange multiplier for the norm constraint, and now introduce a vector y containing the Lagrange multipliers for the equality constraints. The resulting unconstrained optimization problem is

$$\max_{\Delta x, \lambda, y} \; c^T(x^0 + \Delta x) - \lambda(\Delta x^T \Delta x - 1) - y^T(A(x^0 + \Delta x) - b).$$

Differentiating this Lagrangian with respect to Δx, λ, and y and setting these derivatives to zero, we get

$$\begin{aligned} c - 2\lambda \Delta x - A^T y &= 0 \\ \|\Delta x\|^2 - 1 &= 0 \\ A(x^0 + \Delta x) - b &= 0. \end{aligned}$$

The second equation tells us that the length of Δx is one. Since we are interested in the direction of Δx and are not concerned about its length, we ignore this second

equation. The first equation tells us that Δx is proportional to $c - A^T y$, and again, since we aren't concerned about lengths, we put $\lambda = 1/2$ so that the first equation reduces to

$$\Delta x = c - A^T y. \tag{20.3}$$

Since $Ax^0 = b$, the third equation says that

$$A\Delta x = 0.$$

Substituting (20.3) into this equation, we get

$$Ac - AA^T y = 0,$$

which, assuming that AA^T has full rank (as it should), can be solved for y to get

$$y = (AA^T)^{-1}Ac.$$

Now, substituting this expression into (20.3), we see that

$$\Delta x = c - A^T(AA^T)^{-1}Ac.$$

It is convenient to let P be the matrix defined by

$$P = I - A^T(AA^T)^{-1}A.$$

With this definition, Δx can be expressed succinctly as

$$\Delta x = Pc.$$

We claim that P is the matrix that maps any vector, such as c, to its orthogonal projection onto the null space of A. To justify this claim, we first need to define some of the terms we've used. The *null space* of A is defined as $\{d \in \mathbb{R}^n : Ad = 0\}$. We shall denote the null space of A by $N(A)$. A vector $\tilde{c}$ is the *orthogonal projection* of c onto $N(A)$ if it lies in the null space,

$$\tilde{c} \in N(A),$$

and if the difference between it and c is orthogonal to every other vector in $N(A)$. That is,

$$d^T(c - \tilde{c}) = 0, \qquad \text{for all } d \in N(A).$$

Hence, to show that Pc is the orthogonal projection of c onto the null space of A, we simply check these two conditions. Checking the first, we see that

$$APc = Ac - AA^T(AA^T)^{-1}Ac,$$

which clearly vanishes. To check the second condition, let d be an arbitrary vector in the null space, and compute

$$d^T(c - Pc) = d^T A^T(AA^T)^{-1}Ac,$$

which also vanishes, since $d^T A^T = (Ad)^T = 0$. The orthogonal projection Pc is shown in Figure 20.1.

3. The Projected Gradient Direction with Scaling

The orthogonal projection of the gradient gives a good step direction in the sense that among all feasibility-preserving directions, it gives the largest rate of increase of $c^T x$ per unit step length. This property is nice as long as the current point x^0 is well inside the feasible set. But if it is close to a "wall," the overall increase in one step will be small, since even though the rate is large the actual step length will be small, yielding a small overall increase. In fact, the increase will become arbitrarily small as the point x^0 is taken closer and closer to a "wall." Hence, to get a reasonable algorithm, we need to find a formula for computing step directions that steers away from walls as they get close.

The affine-scaling algorithm achieves this affect as follows: *scale the variables in the problem so that the current feasible solution is far from the walls, compute the step direction as the projected gradient in the scaled problem, and then translate this direction back into the original system.* The idea of scaling seems too simple to do any good, and this is true if one tries the most naive scaling—just multiplying every variable by one large number (such as the reciprocal of the smallest component of x^0). Such a uniform scaling does not change the picture in any way. For example, Figure 20.1, which doesn't show specific scales on the coordinate axes, would not change at all. Hence, whether distance is measured in miles or in feet, the property of being close to a wall remains unchanged.

Fortunately, the scaling we have in mind for the affine-scaling algorithm is just slightly fancier. Indeed, the idea is to scale each variable in such a manner that its initial value gets mapped to 1. That is, for each $j = 1, 2, \ldots, n$, we introduce new variables given by

$$\xi_j = \frac{x_j}{x_j^0}.$$

Of course, this change of variable is trivial to undo:

$$x_j = x_j^0 \xi_j.$$

In matrix notation, the change of variables can be written as

$$x = X^0 \xi. \tag{20.4}$$

Note that we are employing our usual convention of letting an upper-case letter stand for a diagonal matrix whose diagonal elements come from the components of the vector denoted by the corresponding lower-case letter. Clearly, under this change of variables, the initial solution x^0 gets mapped to the vector e of all ones, which is at least one unit away from each wall. Figure 20.2 shows an example of this scaling transformation. Note that, unlike the trivial scaling mentioned above, this scaling changes the way the level sets of the objective cut across the feasible region.

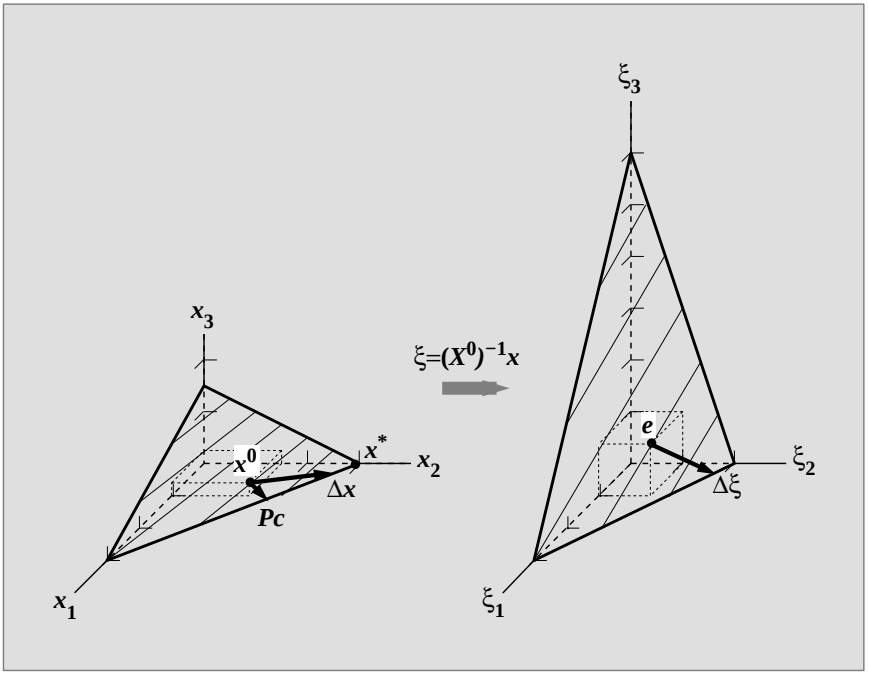


FIGURE 20.2. The effect of affine scaling on projected gradients.

Making the change of variables given by (20.4) in (20.1), we find that the problem in the scaled space is

$$\begin{aligned} \text{maximize} \quad & c^T X^0 \xi \\ \text{subject to} \quad & AX^0 \xi = b \\ & \xi \geq 0. \end{aligned}$$

Clearly, it is a linear programming problem in standard form with constraint matrix AX^0 and vector of objective function coefficients $(c^T X^0)^T = X^0 c$. Letting $\Delta\xi$ denote the projected gradient of the objective function in this scaled problem, we see that

$$\Delta\xi = \left(I - X^0 A^T (AX^{0^2} A^T)^{-1} AX^0\right) X^0 c.$$

Ignore step length worries for the moment, and consider moving from the current solution $\xi^0 = e$ to the following new point in the scaled problem:

$$\xi^1 = \xi^0 + \Delta\xi.$$

Then transforming this new point back into the original unscaled variables, we get a new point x^1 given by

$$x^1 = X^0\xi^1 = X^0(e + \Delta\xi) = x^0 + X^0\Delta\xi.$$

Of course, the difference between x^1 and x^0 is the step direction in the original variables. Denoting this difference by Δx, we see that

$$\begin{aligned}\Delta x &= X^0\left(I - X^0A^T(AX^{0^2}A^T)^{-1}AX^0\right)X^0c \\ &= \left(D - DA^T(ADA^T)^{-1}AD\right)c, \end{aligned} \tag{20.5}$$

where

$$D = X^{0^2}.$$

The expression for Δx given by (20.5) is called the *affine-scaling step direction*. Of course, to construct the affine-scaling algorithm out of this formula for a step direction, one simply needs to choose step lengths in such a manner as to ensure "strict" feasibility of each iterate.

We end this section by illustrating some of the calculations on the following trivial example:

$$\begin{aligned} \text{maximize}\ \ & 2x_1 + 3x_2 + 2x_3 \\ \text{subject to}\ \ & x_1 + \ x_2 + 2x_3 = 3 \\ & x_1,\ x_2,\ x_3 \geq 0. \end{aligned}$$

This is precisely the problem shown in Figure 20.2. As in the figure, let us assume that

$$x^0 = \begin{bmatrix} 1 \\ \frac{3}{2} \\ \frac{1}{4} \end{bmatrix}.$$

For later comparison, we compute the projected gradient (without scaling):

$$\begin{aligned} Pc &= c - A^T(AA^T)^{-1}Ac \\ &= \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \left(\begin{bmatrix} 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \frac{9}{6} \\ &= \begin{bmatrix} \frac{1}{2} \\ \frac{3}{2} \\ -1 \end{bmatrix}. \end{aligned}$$

Now, in the scaled coordinate system, the gradient of the objective function is

$$X^0 c = \begin{bmatrix} 1 & & \\ & \frac{3}{2} & \\ & & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{9}{2} \\ \frac{1}{2} \end{bmatrix}$$

and the constraint matrix is given by

$$AX^0 = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & & \\ & \frac{3}{2} & \\ & & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} 1 & \frac{3}{2} & \frac{1}{2} \end{bmatrix}.$$

Using these, we compute $\Delta\xi$ as follows:

$$\begin{aligned} \Delta\xi &= \begin{bmatrix} 2 \\ \frac{9}{2} \\ \frac{1}{2} \end{bmatrix} - \begin{bmatrix} 1 \\ \frac{3}{2} \\ \frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} 1 & \frac{3}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ \frac{3}{2} \\ \frac{1}{2} \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & \frac{3}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 \\ \frac{9}{2} \\ \frac{1}{2} \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ \frac{9}{2} \\ \frac{1}{2} \end{bmatrix} - \begin{bmatrix} 1 \\ \frac{3}{2} \\ \frac{1}{2} \end{bmatrix} \frac{2}{7}9 \\ &= \begin{bmatrix} -\frac{4}{7} \\ \frac{9}{14} \\ -\frac{11}{14} \end{bmatrix}. \end{aligned}$$

Finally, Δx is obtained by the inverse scaling:

$$\begin{aligned}\Delta x &= X^0 \Delta\xi \\ &= \begin{bmatrix} 1 & & \\ & \frac{3}{2} & \\ & & \frac{1}{4} \end{bmatrix} \begin{bmatrix} -\frac{4}{7} \\ \frac{9}{14} \\ -\frac{11}{14} \end{bmatrix} \\ &= \begin{bmatrix} -\frac{4}{7} \\ \frac{27}{28} \\ -\frac{11}{56} \end{bmatrix}.\end{aligned}$$

In the next section, we discuss the convergence properties of the affine-scaling algorithm.

4. Convergence

In the previous section, we derived the affine-scaling step direction Δx. To make an algorithm, we need to introduce an associated step length. If the step were chosen so that the new point were to lie exactly on the boundary of the feasible region, then the multiplier for Δx, which as usual we denote by θ, would be given by

$$\theta = \left(\max_j \left\{ -\frac{\Delta x_j}{x_j} \right\} \right)^{-1}.$$

But as always, we need to shorten the step by introducing a parameter $0 < r < 1$ and setting

$$\theta = r \left(\max_j \left\{ -\frac{\Delta x_j}{x_j} \right\} \right)^{-1}.$$

With this choice of θ, the iterations of the affine-scaling algorithm are defined by

$$x \leftarrow x + \theta \Delta x.$$

It turns out that the analysis of the affine-scaling algorithm is more delicate than the analysis of the path-following algorithm that we discussed in Chapter 17. Hence, we simply state without proof the main results.

THEOREM 20.1.

(a) *If the problem and its dual are nondegenerate, then for every $r < 1$, the sequence generated by the algorithm converges to the optimal solution.*
(b) *For $r \leq 2/3$, the sequence generated by the algorithm converges to an optimal solution (regardless of degeneracy).*
(c) *There exists an example and an associated $r < 1$ for which the algorithm converges to a nonoptimal solution.*

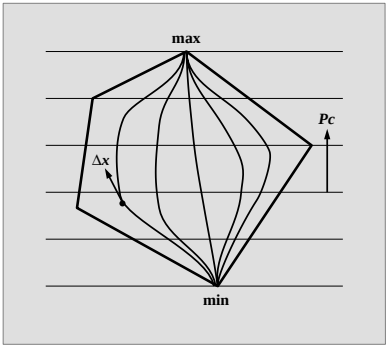


FIGURE 20.3. A few continuous paths of the affine-scaling algorithm. At every point, the continuous path is tangent to the step direction Δx.

There is only one example currently known for which the affine-scaling algorithm fails by converging to a nonoptimal solution. For this example, the failure occurs only for all $r > 0.995$. It is not known whether there are examples of the algorithm failing for all $r > 2/3$, although such a worst-case example seems likely to exist.

Convergence is only the first question. Once convergence is established, the follow-up question is: how fast? For example, given a fixed tolerance, does the affine-scaling algorithm produce a solution within this tolerance of optimality in a number of iterations that is bounded by a polynomial in n? Some variants of the path-following method have this desirable property, so one would hope that the affine-scaling method would share it. Unfortunately, while no one has written down a detailed example yet, there is strong evidence that the affine-scaling method does not have this property.

To explain the evidence, consider letting the step lengths in the affine-scaling algorithm be extremely short, even infinitesimally short. In this case, the algorithm no longer generates a sequence of points moving toward the optimal solution but rather makes a smooth curve connecting the starting point to the optimal solution. If we let the starting point vary, then we get a family of curves filling out the entire interior of the feasible region and connecting each interior point to the optimal solution. Figure 20.3 shows an example of a feasible region and some of the *continuous paths*. Studying the continuous paths gives information about the discrete step algorithm, since, for each point x, the step direction Δx at x is tangent to the continuous path through x. The important property that the continuous paths illustrate is that as one gets close to a face of the feasible polytope, the continuous path becomes tangent to the face (see Exercise 20.1 for an algebraic verification of this statement). This tangency holds for faces of all dimensions. In particular, it is true for edges. Hence, if one starts close to an edge, then one gets a step that looks a lot like a step of the

simplex method. Therefore, it is felt that if one were to take a problem that is bad for the simplex method, such as the Klee–Minty problem, and start the affine-scaling algorithm in just the right place, then it would mimic the steps of the simplex method and therefore take 2^n iterations to get close to the optimal solution. This is the idea, but as noted above, no one has carried out the calculations.

5. Feasibility Direction

To derive a Phase I procedure for the affine-scaling algorithm, we consider a starting point x^0 that has strictly positive components but does not necessarily satisfy the equality constraints $Ax = b$. We then let

$$\rho = b - Ax^0$$

denote the vector of infeasibilities. With these definitions under our belt, we introduce the following auxiliary problem involving one extra variable, which we shall denote by x_0 (not to be confused with the initial solution vector x^0):

$$\begin{aligned} \text{maximize} \quad & -x_0 \\ \text{subject to} \quad & Ax + x_0\rho = b \\ & x \geq 0,\ x_0 \geq 0. \end{aligned}$$

Clearly, the vector

$$\begin{bmatrix} x^0 \\ 1 \end{bmatrix}$$

is a strictly positive feasible solution to the auxiliary problem. Hence, we can apply the affine-scaling algorithm to the auxiliary problem. If the optimal solution has $x_0^* > 0$, then the original problem is infeasible. If, on the other hand, the optimal solution has $x_0^* = 0$, then the optimal solution to the auxiliary problem provides a feasible starting solution to the original problem (it may not be a strictly interior feasible solution, but we shall ignore such technicalities in the present discussion).

Let us now derive a specific formula for the step direction vector in the auxiliary problem. The vector of objective coefficients is

$$\begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

the constraint matrix is

$$\begin{bmatrix} A & \rho \end{bmatrix},$$

and the "current" solution can be denoted as

$$\begin{bmatrix} x \\ x_0 \end{bmatrix}.$$

Substituting these three objects appropriately into (20.5), we get

$$\begin{bmatrix} \Delta x \\ \Delta x_0 \end{bmatrix} = \left(\begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} - \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \begin{bmatrix} A^T \\ \rho^T \end{bmatrix} \left(\begin{bmatrix} A & \rho \end{bmatrix} \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \begin{bmatrix} A^T \\ \rho^T \end{bmatrix} \right)^{-1} \begin{bmatrix} A & \rho \end{bmatrix} \begin{bmatrix} X^2 & \\ & x_0^2 \end{bmatrix} \right) \begin{bmatrix} 0 \\ -1 \end{bmatrix}.$$

Exploiting heavily the fact that all the coefficients in the objective vector are zero except for the last, we can write a fairly simple expression for Δx:

$$\Delta x = X^2 A^T \left(A X^2 A^T + x_0^2 \rho \rho^T \right)^{-1} \rho x_0.$$

The final simplification comes from applying the Sherman–Morrison–Woodbury formula (see Exercise 18.1) to the inverted expression above to discover that the vector $(AX^2A^T + x_0^2\rho\rho^T)^{-1}\rho$ points in the same direction as $(AX^2A^T)^{-1}\rho$. That is, there is a positive scalar α such that

$$(AX^2A^T + x_0^2\rho\rho^T)^{-1}\rho = \alpha(AX^2A^T)^{-1}\rho$$

(see Exercise 20.3). Since vector lengths are irrelevant, we can define the affine-scaling feasibility step direction as

$$\Delta x = X^2 A^T \left(A X^2 A^T \right)^{-1} \rho. \tag{20.6}$$

6. Problems in Standard Form

We return now to problems in standard form:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0. \end{aligned}$$

Introducing slack variables w, we can write the problem equivalently as

$$\begin{aligned} \text{maximize} \quad & \begin{bmatrix} c \\ 0 \end{bmatrix}^T \begin{bmatrix} x \\ w \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} = b \\ & \begin{bmatrix} x \\ w \end{bmatrix} \geq 0. \end{aligned} \tag{20.7}$$

Writing down the affine-scaling step direction for this problem, we get

$$\begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} = \left(\begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} - \begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} \begin{bmatrix} A^T \\ I \end{bmatrix} \left(\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} \begin{bmatrix} A^T \\ I \end{bmatrix} \right)^{-1} \begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} \right) \begin{bmatrix} c \\ 0 \end{bmatrix},$$

which simplifies to

$$\begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} = \begin{bmatrix} X^2 c \\ 0 \end{bmatrix} - \begin{bmatrix} X^2 A^T \\ W^2 \end{bmatrix} \left(A X^2 A^T + W^2 \right)^{-1} A X^2 c.$$

Therefore, in particular

$$\Delta x = X^2 c - X^2 A^T (A X^2 A^T + W^2)^{-1} A X^2 c.$$

Note that this formula for Δx matches the formula for Δx_{OPT} given in Section 18.3, except that the diagonal matrix X^2 replaces XZ^{-1} and W^2 replaces WY^{-1}. These diagonal matrices are referred to as *scaling matrices*. Hence, the formula for Δx given above is often called the *affine-scaling step direction with primal scaling*, whereas Δx_{OPT} is referred to as the *affine-scaling step direction with primal–dual scaling*.

Similar connections can be established between the Phase I step direction derived in this section and Δx_{FEAS} from Section 18.3. Indeed, from (20.6), we see that the feasibility step direction for (20.7) is

$$\begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} = \begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} \begin{bmatrix} A^T \\ I \end{bmatrix} \left(\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} X^2 & \\ & W^2 \end{bmatrix} \begin{bmatrix} A^T \\ I \end{bmatrix} \right)^{-1} \left(b - \begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} \right).$$

Again, looking just at the formula for Δx, we see that

$$\Delta x = X^2 A^T (A X^2 A^T + W^2)^{-1} (b - Ax - w),$$

which coincides with Δx_{FEAS} except that X^2 replaces XZ^{-1} and W^2 replaces WY^{-1}.

Exercises

20.1 *Step direction becomes tangent to each facet.* Let Δx denote the affine-scaling step direction given by

$$\Delta x = \left(X^2 - X^2 A^T (A X^2 A^T)^{-1} A X^2 \right) c.$$

This step direction is clearly a function of x. Fix j. Show that the limit as x_j tends to zero of Δx is a vector whose jth component vanishes. That is,

$$\lim_{x_j \to 0} \Delta x_j = 0.$$

20.2 *Dual Estimates.* Consider the following function, defined in the interior of the polytope of feasible solutions $\{x : Ax = b,\ x > 0\}$ by

$$y(x) = (AX^2A^T)^{-1}AX^2c.$$

Consider a partition of the columns of $A = \begin{bmatrix} B & N \end{bmatrix}$ into a basic part B and a nonbasic part N, and, as we did in our study of the simplex method, partition the n-vectors analogously. Show that

$$\lim_{x_{\mathcal{N}} \to 0} y(x) = (B^T)^{-1}c_{\mathcal{B}}.$$

20.3 Let A be an $m \times n$ matrix having rank m, and let ρ be an arbitrary m-vector. Use the identity proved in Exercise 18.1 to show that there exists a scalar α such that

$$(AA^T + \rho\rho^T)^{-1}\rho = \alpha(AA^T)^{-1}\rho.$$

Hint: Be mindful of which matrices are invertible.

20.4 *(So-called) Dual Affine-Scaling Method.* Compute the affine-scaling step-direction vector Δx for problems in the following form:

$$\begin{aligned} &\text{maximize } c^Tx \\ &\text{subject to } Ax \le b. \end{aligned}$$

Notes

The affine-scaling algorithm was first suggested by Dikin (1967). He subsequently published a convergence analysis in Dikin (1974). Dikin's work went largely unnoticed for many years until several researchers independently rediscovered the affine-scaling algorithm as a simple variant of Karmarkar's algorithm (Karmarkar 1984). Of these independent rediscoveries, only two papers offered a convergence analysis: one by Barnes (1986) and the other by Vanderbei et al. (1986). It is interesting to note that Karmarkar himself was one of the independent rediscoverers, but he mistakenly believed that the algorithm enjoyed the same convergence properties as his algorithm (i.e., that it would get within any fixed tolerance of optimality within a specific number of iterations bounded by a polynomial in n).

Theorem 20.1(a) was proved by Vanderbei et al. (1986). Part (b) of the Theorem was proved by Tsuchiya & Muramatsu (1992) who also show that the result is sharp. A sharper sharpness result can be found in Hall & Vanderbei (1993). Part (c) of the Theorem was established by Mascarenhas (1997).

The first derivation of the affine-scaling feasibility step direction was given by Vanderbei (1989). The simple derivation given in Section 20.5 is due to M. Meketon.

A recent book by Saigal (1995) contains an extensive treatment of affine-scaling methods.

CHAPTER 21

The Homogeneous Self-Dual Method

In Chapter 17, we described and analyzed an interior-point method called the path-following algorithm. This algorithm is essentially what one implements in practice but as we saw in the section on convergence analysis, it is not easy (and perhaps not possible) to give a complete proof that the method converges to an optimal solution. If convergence were completely established, the question would still remain as to how fast is the convergence. In this chapter, we shall present a similar algorithm for which a complete convergence analysis can be given.

1. From Standard Form to Self-Dual Form

As always, we are interested in a linear programming problem given in standard form

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \le b \\ &\qquad\qquad x \ge 0 \end{aligned} \tag{21.1}$$

and its dual

$$\begin{aligned} &\text{minimize } b^T y \\ &\text{subject to } A^T y \ge c \\ &\qquad\qquad y \ge 0. \end{aligned} \tag{21.2}$$

As we shall show, these two problems can be solved by solving the following problem, which essentially combines the primal and dual problems into one problem:

$$\begin{aligned} \text{maximize} \quad & 0 \\ \text{subject to} \quad & -A^T y + c\phi \le 0, \\ & Ax - b\phi \le 0, \\ & -c^T x + b^T y \le 0, \\ & x,\ y,\ \phi \ge 0. \end{aligned} \tag{21.3}$$

Note that, beyond combining the primal and dual into one big problem, one new variable (ϕ) and one new constraint have been added. Hence, the total number of variables

in (21.3) is $n+m+1$ and the total number of constraints is $n+m+1$. Furthermore, the objective function and the right-hand sides all vanish. Problems with such right-hand sides are called *homogeneous*. Also, the constraint matrix for problem (21.3) is skew symmetric. That is, it is equal to the negative of its transpose. Homogeneous linear programming problems having a skew symmetric constraint matrix are called *self-dual*.

In the next section, we shall give an algorithm for the solution of homogeneous self-dual linear programming problems. But first, let's note that a solution to (21.3) in which $\phi > 0$ can be converted into solutions for (21.1) and (21.2). Indeed, let $(\bar{x}, \bar{y}, \bar{\phi})$ be an optimal solution to problem (21.3). Suppose that $\bar{\phi} > 0$. (The algorithm given in the next section will guarantee that this property is satisfied whenever (21.1) and (21.2) have optimal solutions[1]). Put

$$x^* = \bar{x}/\bar{\phi} \quad \text{and} \quad y^* = \bar{y}/\bar{\phi}.$$

Then the constraints in (21.3) say that

$$\begin{aligned} -A^T y^* + c &\le 0, \\ Ax^* - b &\le 0, \\ -c^T x^* + b^T y^* &\le 0. \end{aligned}$$

Also, x^* and y^* are both nonnegative. Therefore, x^* is feasible for (21.1) and y^* is feasible for (21.2). From the weak duality theorem together with the third inequality above, we get

$$c^T x^* = b^T y^*.$$

Therefore, x^* is optimal for the primal problem (21.1) and y^* is optimal for the dual problem (21.2). As we will see later, the case where $\bar{\phi} = 0$ corresponds to infeasibility of either the primal or the dual problem (or both).

2. Homogeneous Self-Dual Problems

Consider a linear programming problem in standard form

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \le b \\ &\qquad\qquad x \ge 0 \end{aligned}$$

and its dual

$$\begin{aligned} &\text{minimize } b^T y \\ &\text{subject to } A^T y \ge c \\ &\qquad\qquad y \ge 0. \end{aligned}$$

[1]The astute reader might notice that setting all variables to 0 produces an optimal solution.

Such a linear programming problem is called *self-dual* if $m = n$, $A = -A^T$, and $b = -c$. The reason for the name is that the dual of such a problem is the same as the primal. To see this, rewrite the constraints as less-thans and then use the defining properties for self-duality to get

$$A^T y \geq c \quad \Leftrightarrow \quad -A^T y \leq -c \quad \Leftrightarrow \quad Ay \leq b.$$

Similarly, writing the objective function as a maximization, we get

$$\min b^T y \quad = \quad -\max -b^T y \quad = \quad -\max c^T y.$$

Hence, ignoring the (irrelevant) fact that the dual records the negative of the objective function, the primal and the dual are seen to be the same. A linear programming problem in which the right-hand side vanishes is called a *homogeneous* problem. It follows that if a problem is homogeneous and self-dual, then its objective function must vanish too.

For the remainder of this section, we assume that the problem under consideration is homogeneous and self-dual. Since the case $m = n = 1$ is trivial ($A = 0$ in this case), we assume throughout this section that $n \geq 2$. Also, since the dual is the same problem as the primal, we prefer to use the letter z for the primal slacks (instead of the usual w). Hence, the primal can be written as

$$\begin{aligned} &\text{maximize } 0 \\ &\text{subject to } Ax + z = 0 \\ &\qquad\qquad\quad x, z \geq 0. \end{aligned} \tag{21.4}$$

The following theorem establishes some of the important properties of homogeneous self-dual problems.

THEOREM 21.1. *For homogeneous self-dual problem* (21.4)*, the following statements hold:*

(1) *It has feasible solutions and every feasible solution is optimal.*
(2) *The set of feasible solutions has empty interior. In fact, if* (x, z) *is feasible, then* $z^T x = 0$.

PROOF. (1) The trivial solution, $(x, z) = (0, 0)$, is feasible. Since the objective function is zero, every feasible solution is optimal.

(2) Suppose that (x, z) is feasible for (21.4). The fact that A is skew symmetric implies that $\xi^T A \xi = 0$ for every vector ξ (see Exercise 15.1). In particular, $x^T Ax = 0$. Therefore, multiplying $Ax + z = 0$ on the left by x^T, we get $0 = x^T Ax + x^T z = x^T z$. This completes the proof. □

Part (2) of the previous Theorem tells us that homogeneous self-dual problems do not have central paths.

2.1. Step Directions. As usual, the interior-point method we shall derive will have the property that the intermediate solutions it produces will be infeasible. Hence, let

$$\rho(x,z) = Ax + z$$

denote the infeasibility of a solution (x,z). Also, let

$$\mu(x,z) = \frac{1}{n}x^T z.$$

The number $\mu(x,z)$ measures the degree of noncomplementarity between x and z. When x and z are clear from context, we shall simply write ρ for $\rho(x,z)$ and μ for $\mu(x,z)$.

Step directions $(\Delta x, \Delta z)$ are chosen to reduce the infeasibility and noncomplementarity of the current solution by a given factor δ, $0 \le \delta \le 1$. Hence, we consider the nonlinear system that would make the infeasibility and noncomplementarity of $(x+\Delta x, z+\Delta z)$ be δ times that of (x,z):

$$A(x+\Delta x) + (z+\Delta z) = \delta(Ax+z),$$
$$(X+\Delta X)(Z+\Delta Z)e = \delta\mu(x,z)e.$$

As usual, this system is nonlinear in the "delta" variables. Dropping the nonlinear term (appearing only in the second equation), we get the following linear system of equations for the step directions:

$$A\Delta x + \Delta z = -(1-\delta)\rho(x,z), \tag{21.5}$$

$$Z\Delta x + X\Delta z = \delta\mu(x,z)e - XZe. \tag{21.6}$$

With these step directions, we pick a step length θ and step to a new point:

$$\bar{x} = x + \theta\Delta x, \qquad \bar{z} = z + \theta\Delta z.$$

We denote the new ρ-vector by $\bar{\rho}$ and the new μ-value by $\bar{\mu}$:

$$\bar{\rho} = \rho(\bar{x},\bar{z}) \quad \text{and} \quad \bar{\mu} = \mu(\bar{x},\bar{z}).$$

The following theorem establishes some of the properties of these step directions.

THEOREM 21.2. *The following relations hold:*

(1) $\Delta z^T \Delta x = 0$.
(2) $\bar{\rho} = (1-\theta+\theta\delta)\rho$.
(3) $\bar{\mu} = (1-\theta+\theta\delta)\mu$.
(4) $\bar{X}\bar{Z}e - \bar{\mu}e = (1-\theta)(XZe-\mu e) + \theta^2\Delta X\Delta Z e$.

PROOF. (1) We start by multiplying both sides of (21.5) on the left by Δx^T:

$$\Delta x^T A\Delta x + \Delta x^T\Delta z = -(1-\delta)\Delta x^T\rho. \tag{21.7}$$

The skew symmetry of A (i.e., $A = -A^T$) implies that $\Delta x^T A\Delta x = 0$ (see Exercise 15.1). Hence, the left-hand side of (21.7) simplifies nicely:

$$\Delta x^T A\Delta x + \Delta x^T\Delta z = \Delta x^T\Delta z.$$

Substituting the definition of ρ into the right-hand side of (21.7), we get

$$-(1-\delta)\Delta x^T\rho = -(1-\delta)\Delta x^T(Ax+z).$$

Next, we use the skew symmetry of A to rewrite $\Delta x^T Ax$ as follows:

$$\Delta x^T Ax = (Ax)^T\Delta x = x^T A^T\Delta x = -x^T A\Delta x.$$

Assembling what we have so far, we see that

$$\Delta x^T\Delta z = -(1-\delta)(-x^T A\Delta x + z^T\Delta x). \tag{21.8}$$

To proceed, we use (21.5) to replace $A\Delta x$ with $-(1-\delta)\rho - \Delta z$. Therefore,

$$\begin{aligned} -x^T A\Delta x + z^T\Delta x &= x^T\left((1-\delta)\rho + \Delta z\right) + z^T\Delta x \\ &= (1-\delta)x^T\rho + x^T\Delta z + z^T\Delta x. \end{aligned} \tag{21.9}$$

Again using the definition of ρ and the skew symmetry of A, we see that

$$x^T\rho = x^T(Ax+z) = x^T z.$$

The last two terms in (21.9) can be simplified by multiplying both sides of (21.6) on the left by e^T and then using the definition of μ to see that

$$z^T\Delta x + x^T\Delta z = \delta\mu n - x^T z = (\delta-1)x^T z..$$

Making these substitutions in (21.9), we get

$$-x^T A\Delta x + z^T\Delta x = (1-\delta)x^T z + (\delta-1)x^T z = 0.$$

Hence, from (21.8), we see that $\Delta x^T\Delta z$ vanishes as claimed.

(2) From the definitions of $\bar{x}$ and $\bar{z}$, we see that

$$\begin{aligned} \bar{\rho} &= A(x+\theta\Delta x) + (z+\theta\Delta z) \\ &= Ax + z + \theta(A\Delta x + \Delta z) \\ &= (1-\theta+\theta\delta)\rho. \end{aligned}$$

(3) From the definitions of $\bar{x}$ and $\bar{z}$, we see that

$$\begin{aligned} \bar{x}^T\bar{z} &= (x+\theta\Delta x)^T(z+\theta\Delta z) \\ &= x^T z + \theta(z^T\Delta x + x^T\Delta z) + \theta^2\Delta z^T\Delta x. \end{aligned}$$

From part (1) and (21.6), we then get

$$\bar{x}^T\bar{z} = x^T z + \theta(\delta\mu n - x^T z).$$

Therefore,

$$\bar{\mu} = \frac{1}{n}\bar{x}^T\bar{z} = (1-\theta)\mu + \theta\delta\mu.$$

(4) From the definitions of $\bar{x}$ and $\bar{z}$ together with part (3), we see that

$$\begin{aligned} \bar{X}\bar{Z}e - \bar{\mu}e &= (X+\theta\Delta X)(Z+\theta\Delta Z)e - (1-\theta+\theta\delta)\mu e \\ &= XZe + \theta(Z\Delta x + X\Delta z) + \theta^2\Delta X\Delta Z e - (1-\theta+\theta\delta)\mu e. \end{aligned}$$

Substituting (21.6) into the second term on the right and recollecting terms, we get the desired expression. □

2.2. Predictor-Corrector Algorithm. With the preliminaries behind us, we are now ready to describe an algorithm. We shall be more conservative than we were in Chapter 17 and define the algorithm in such a way that it keeps the components of XZe close to each other. Indeed, for each $0 \leq \beta \leq 1$, let

$$\mathcal{N}(\beta) = \{(x, z) > 0 : \|XZe - \mu(x, z)e\| \leq \beta\mu(x, z)\}.$$

Shortly, we will only deal with $\mathcal{N}(1/4)$ and $\mathcal{N}(1/2)$ but first let us note generally that $\beta < \beta'$ implies that $\mathcal{N}(\beta) \subset \mathcal{N}(\beta')$. Hence, as a function of β, the $\mathcal{N}(\beta)$'s form an increasing family of sets. Also, $\mathcal{N}(0)$ is precisely the set of points (x, z) for which XZe has all equal components.

The algorithm alternates between two types of steps. On the first iteration and subsequently on every other iteration, the algorithm performs a *predictor step*. Before a predictor step, one assumes that

$$(x, z) \in \mathcal{N}(1/4).$$

Then step directions are computed using $\delta = 0$ (i.e., with no centering) and the step length is calculated so as not to go outside of $\mathcal{N}(1/2)$:

$$\theta = \max\{t : (x + t\Delta x, z + t\Delta z) \in \mathcal{N}(1/2)\}. \tag{21.10}$$

On the even iterations, the algorithm performs a *corrector step*. Before a corrector step, one assumes that

$$(x, z) \in \mathcal{N}(1/2)$$

(as is guaranteed by the predictor step's step length). Then step directions are computed using $\delta = 1$ (i.e., pure centering) and the step length parameter θ is set to 1.

The following theorem shows that the result of each step satisfies the precondition for the next step of the algorithm and that μ decreases on predictor steps while it stays the same on corrector steps.

THEOREM 21.3. *The following statements are true:*

(1) *After a predictor step,* $(\bar{x}, \bar{z}) \in \mathcal{N}(1/2)$ *and* $\bar{\mu} = (1 - \theta)\mu$.
(2) *After a corrector step,* $(\bar{x}, \bar{z}) \in \mathcal{N}(1/4)$ *and* $\bar{\mu} = \mu$.

PROOF OF PART (1). The formula for $\bar{\mu}$ follows from part (3) of Theorem 21.2 by putting $\delta = 0$. The fact that $(\bar{x}, \bar{z}) \in \mathcal{N}(1/2)$ is an immediate consequence of the choice of θ. □

Before proving Part (2) of the Theorem, we need to introduce some notation and prove a few technical results. Let

$$\begin{aligned} p &= X^{-1/2}Z^{1/2}\Delta x, \\ q &= X^{1/2}Z^{-1/2}\Delta z, \\ r &= p + q \\ &= X^{-1/2}Z^{-1/2}(Z\Delta x + X\Delta z) \\ &= X^{-1/2}Z^{-1/2}(\delta\mu e - XZe). \end{aligned} \tag{21.11}$$

The technical results are summarized in the following lemma.

LEMMA 21.4. *The following statements are true:*

(1) $\|PQe\| \leq \frac{1}{2}\|r\|^2$.
(2) *If* $\delta = 0$, *then* $\|r\|^2 = n\mu$.
(3) *If* $\delta = 1$ *and* $(x, z) \in \mathcal{N}(\beta)$, *then* $\|r\|^2 \leq \beta^2\mu/(1-\beta)$.

PROOF. (1) First note that $p^T q = \Delta x^T \Delta z = 0$ by Theorem 21.2(1). Hence,

$$\|r\|^2 = \|p+q\|^2 = p^T p + 2p^T q + q^T q = \sum_j (p_j^2 + q_j^2).$$

Therefore,

$$\begin{aligned} \|r\|^4 &= \left(\sum_j (p_j^2 + q_j^2)\right)^2 \\ &\geq \sum_j (p_j^2 + q_j^2)^2 \\ &= \sum_j \left((p_j^2 - q_j^2)^2 + 4p_j^2 q_j^2\right) \\ &\geq 4\sum_j p_j^2 q_j^2 \\ &= 4\|PQe\|^2. \end{aligned}$$

Taking square roots yields the desired inequality.

(2) Putting $\delta = 0$ in (21.11), we see that $r = -X^{1/2}Z^{1/2}e$. Therefore, $\|r\|^2 = z^T x = n\mu$.

(3) Suppose that $(x, z) \in \mathcal{N}(\beta)$. Whenever the norm of a vector is smaller than some number, the magnitude of each component of the vector must also be smaller than this number. Hence, $|x_j z_j - \mu| \leq \beta\mu$. It is easy to see that this inequality is equivalent to

$$(1-\beta)\mu \leq x_j z_j \leq (1+\beta)\mu. \tag{21.12}$$

Now putting $\delta = 1$ in (21.11), we get

$$\|r\|^2 = \sum_j \frac{(x_j z_j - \mu)^2}{x_j z_j}.$$

Therefore, using the lower bound given in (21.12), we get the following upper bound:

$$\|r\|^2 \le \frac{1}{(1-\beta)\mu} \sum_j (x_j z_j - \mu)^2.$$

Finally, since $(x, z) \in \mathcal{N}(\beta)$, we see that the above sum is bounded by $\beta^2\mu^2$. This gives the claimed inequality. □

PROOF OF THEOREM 21.3(2). Since $\theta = 1$ in a corrector step, it follows from Theorem 21.2(4) that $\bar{X}\bar{Z}e - \bar{\mu}e = \Delta X \Delta Z e = PQe$. Therefore, parts (1) and (3) of Lemma 21.4 imply that

$$\begin{aligned} \|\bar{X}\bar{Z}e - \bar{\mu}e\| &= \|PQe\| \\ &\le \frac{1}{2}\|r\|^2 \\ &\le \frac{1}{2}\frac{(1/2)^2}{1-1/2}\mu \\ &= \frac{1}{4}\mu. \end{aligned} \tag{21.13}$$

We also need to show that $(\bar{x}, \bar{z}) > 0$. For $0 \le t \le 1$, let

$$x(t) = x + t\Delta x, \quad z(t) = z + t\Delta z, \quad \text{and} \quad \mu(t) = \mu(x(t), z(t)).$$

Then from part (4) of Theorem 21.2, we have

$$X(t)Z(t)e - \mu(t)e = (1-t)(XZe - \mu e) + t^2 \Delta X \Delta Z e.$$

The right-hand side is the sum of two vectors. Since the length of the sum of two vectors is less than the sum of the lengths (i.e., by the *triangle inequality*), it follows that

$$\|X(t)Z(t)e - \mu(t)e\| \le (1-t)\|XZe - \mu e\| + t^2\|\Delta X \Delta Z e\| \tag{21.14}$$

(note that we've pulled the scalars out of the norms). Now, since $(x, z) \in \mathcal{N}(1/2)$, we have $\|XZe - \mu e\| \le \mu/2$. Furthermore, from (21.13) we have that $\|\Delta X \Delta Z e\| = \|PQe\| \le \mu/4$. Replacing the norms in (21.14) with these upper bounds, we get the following bound:

$$\|X(t)Z(t)e - \mu(t)e\| \le (1-t)\frac{\mu}{2} + t^2\frac{\mu}{4} \le \frac{\mu}{2} \tag{21.15}$$

(the second inequality follows from the obvious facts that $t^2 \le t$ and $\mu/4 \le \mu/2$).

Now, consider a specific component j. It follows from (21.15) that

$$x_j(t)z_j(t) - \mu(t) \ge -\frac{\mu}{2}.$$

Since $\delta = 1$, part (3) of Theorem 21.2 tells us that $\mu(t) = \mu$ for all t. Therefore the previous inequality can be written as

$$x_j(t)z_j(t) \geq \frac{\mu}{2} > 0. \tag{21.16}$$

This inequality then implies that $x_j(t) > 0$ and $z_j(t) > 0$ for all $0 \leq t \leq 1$ (since they could only become negative by passing through 0, which is ruled out by (21.16)). Putting $t = 1$, we get that $\bar{x}_j > 0$ and $\bar{z}_j > 0$. Since the component j was arbitrary, it follows that $(\bar{x}, \bar{z}) > 0$. Therefore $(\bar{x}, \bar{z}) \in \mathcal{N}(1/4)$. □

2.3. Convergence Analysis. The previous theorem showed that the predictor-corrector algorithm is well defined. The next theorem gives us a lower bound on the progress made by each predictor step.

THEOREM 21.5. *In each predictor step,* $\theta \geq \frac{1}{2\sqrt{n}}$.

PROOF. Using the same notation as in the proof of Theorem 21.3, we have the inequality:

$$\begin{aligned}\|X(t)Z(t)e - \mu(t)e\| \leq (1-t)\|XZe - \mu e\| \\ + t^2\|\Delta X \Delta Z e\|.\end{aligned} \tag{21.17}$$

This time, however, $(x, z) \in \mathcal{N}(1/4)$ and $\delta = 0$. Hence,

$$\|XZe - \mu e\| \leq \frac{\mu}{4}$$

and, from parts (1) and (2) of Lemma 21.4,

$$\|\Delta X \Delta Z e\| = \|PQe\| \leq \frac{1}{2}\|r\|^2 = \frac{1}{2}n\mu.$$

Using these two bounds in (21.17), we get the following bound:

$$\|X(t)Z(t)e - \mu(t)e\| \leq (1-t)\frac{\mu}{4} + t^2\frac{n\mu}{2}.$$

Now, fix a $t \leq (2\sqrt{n})^{-1}$. For such a t, we have $t^2 n/2 \leq 1/8$. Therefore, using the fact that $t \leq 1/2$ for $n \geq 2$, we get

$$\begin{aligned}\|X(t)Z(t)e - \mu(t)e\| &\leq (1-t)\frac{\mu}{4} + \frac{\mu}{8} \\ &\leq (1-t)\frac{\mu}{4} + (1-t)\frac{\mu}{4} \\ &= (1-t)\frac{\mu}{2} \\ &= \frac{\mu(t)}{2}.\end{aligned}$$

Hence, as in the previous theorem, $(x(t), z(t)) \in \mathcal{N}(1/2)$. Since t was an arbitrary number less than $(2\sqrt{n})^{-1}$, it follows that $\theta \geq (2\sqrt{n})^{-1}$. □

Let $(x^{(k)}, z^{(k)})$ denote the solution after the kth iteration and let

$$\rho^{(k)} = \rho(x^{(k)}, z^{(k)}) \quad \text{and} \quad \mu^{(k)} = \mu(x^{(k)}, z^{(k)}).$$

The algorithm starts with $x^{(0)} = z^{(0)} = e$. Therefore, $\mu^{(0)} = 1$. Our aim is to show that $\mu^{(k)}$ and $\rho^{(k)}$ tend to zero as k tends to infinity. The previous theorem together with Theorem 21.3 implies that, after an even number of iterations, say $2k$, the following inequality holds:

$$\mu^{(2k)} \leq \left(1 - \frac{1}{2\sqrt{n}}\right)^k.$$

Also, since the corrector steps don't change the value of μ, it follows that

$$\mu^{(2k-1)} = \mu^{(2k)}.$$

From these two statements, we see that

$$\lim_{k\to\infty} \mu^{(k)} = 0.$$

Now, consider $\rho^{(k)}$. It follows from parts (2) and (3) of Theorem 21.2 that the reduction in infeasibility tracks the reduction in noncomplementarity. Hence,

$$\rho^{(k)} = \mu^{(k)}\rho^{(0)}.$$

Therefore, the fact that $\mu^{(k)}$ tends to zero implies the same for $\rho^{(k)}$.

In fact, more can be said:

THEOREM 21.6. *The limits $x^* = \lim_{k\to\infty} x^{(k)}$ and $z^* = \lim_{k\to\infty} z^{(k)}$ exist and (x^*, z^*) is optimal. Furthermore, the vectors x^* and z^* are strictly complementary to each other. That is, for each j, $x_j^* z_j^* = 0$ but either $x_j^* > 0$ or $z_j^* > 0$.*

The proof is fairly technical, and so instead of proving it, we prove the following theorem, which captures the main idea.

THEOREM 21.7. *Suppose that $(x, z) \in \mathcal{N}(\beta)$ and that $\rho(x, z) = \mu(x, z)\rho^{(0)}$. Then there exist positive constants $c_1, c_2, \ldots, c_n$ such that $x_j + z_j \geq c_j > 0$ for each $j = 1, 2, \ldots, n$.*

PROOF. Put $\mu = \mu(x, z)$ and $\rho = \rho(x, z)$. Let (x^*, z^*) be a strictly complementary feasible solution (the existence of which is guaranteed by Theorem 10.6). We begin by studying the expression $z^T x^* + x^T z^*$. Since $Ax^* + z^* = 0$, we have that

$$\begin{aligned} z^T x^* + x^T z^* &= z^T x^* - x^T A x^* \\ &= (-A^T x + z)^T x^*. \end{aligned}$$

By the skew-symmetry of A, we see that $-A^T x + z = Ax + z = \rho$. And, since $\rho = \mu\rho^{(0)}$, we get

$$z^T x^* + x^T z^* = \mu {\rho^{(0)}}^T x^*. \tag{21.18}$$

The factor ${\rho^{(0)}}^T x^*$ is a constant (i.e., it does not depend on x or z). Let us denote it by M. Since all the terms in the two products on the left in (21.18) are nonnegative, it follows that each one is bounded by the right-hand side. So if we focus on a particular index j, we get the following bounds:

$$z_j x_j^* \le \mu M \quad \text{and} \quad x_j z_j^* \le \mu M. \tag{21.19}$$

Now, we use the assumption that $(x, z) \in \mathcal{N}(\beta)$ to see that

$$x_j z_j \ge (1-\beta)\mu.$$

In other words, $\mu \le x_j z_j/(1-\beta)$, and so the inequalities in (21.19) become

$$z_j x_j^* \le \frac{M}{1-\beta} z_j x_j \quad \text{and} \quad x_j z_j^* \le \frac{M}{1-\beta} x_j z_j.$$

Since x_j and z_j are strictly positive, we can divide by them (and the constants) to get

$$\frac{1-\beta}{M} x_j^* \le x_j \quad \text{and} \quad \frac{1-\beta}{M} z_j^* \le z_j.$$

Putting

$$c_j = \frac{1-\beta}{M}(x_j^* + z_j^*),$$

we get the desired lower bound on $x_j + z_j$. □

2.4. Complexity of the Predictor-Corrector Algorithm. Of course, in practice we don't run an infinite number of iterations. Instead, we set a priori a threshold and stop when $\mu^{(k)}$ falls below it. The threshold is usually denoted by 2^{-L} where L is some number. Typically, we want the threshold to be about 10^{-8}, which corresponds to $L \approx 26$.

As we saw before, after an even number of iterations, say $2k$, the μ-value is bounded by the following inequality:

$$\mu^{(2k)} \le \left(1 - \frac{1}{2\sqrt{n}}\right)^k.$$

Hence, it suffices to pick a k big enough to have

$$\left(1 - \frac{1}{2\sqrt{n}}\right)^k \le 2^{-L}.$$

Taking logarithms of both sides and solving for k, we see that any

$$k \ge \frac{L}{-\log(1 - \frac{1}{2\sqrt{n}})}$$

will do. Since $-\log(1-x) \ge x$, we get

$$2L\sqrt{n} \ge \frac{L}{-\log(1 - \frac{1}{2\sqrt{n}})}.$$

Therefore, any $k \geq 2L\sqrt{n}$ will do. In particular, $k = 2L\sqrt{n}$ rounded up to the nearest integer will suffice. Since k represents half the number of iterations, it follows that it will take at most $4L\sqrt{n}$ iterations for the μ-value to fall below the threshold of 2^{-L}. This bound implies that the method is a *polynomial algorithm*, since it says that any desired precision can be obtained in a number of iterations that is bounded above by a polynomial in n (here, $4L\sqrt{n}$ is not itself a polynomial but is bounded above by say a linear function in n for $n \geq 2$).

2.5. The KKT System. We end this section on homogeneous self-dual problems by briefly discussing the KKT system (21.5)–(21.6). Solving this system of equations is the most time consuming step within each iteration of the predictor-corrector algorithm. There are several ways in which one can organize the computation. The approach that most parallels what we have done before is first to solve (21.6) for Δz,

$$\begin{aligned} \Delta z &= X^{-1}(-Z\Delta x + \delta\mu e - XZe) \\ &= -X^{-1}Z\Delta x + \delta\mu X^{-1}e - z, \end{aligned} \tag{21.20}$$

and then to eliminate it from (21.5) to get the following *reduced KKT system*:

$$(A - X^{-1}Z)\Delta x = -(1-\delta)\rho + z - \delta\mu X^{-1}e.$$

In the next section, we apply the algorithm developed in this section to the homogeneous self-dual problem given by (21.3).

3. Back to Standard Form

We return now to the setup in Section 1. Let z, w, and ψ denote the slack variables for the constraints in problem (21.3):

$$\begin{array}{lrcl} \text{maximize} & & & 0 \\ \text{subject to} & -A^T y + c\phi + z & = & 0, \\ & Ax - b\phi + w & = & 0, \\ & -c^T x + b^T y + \psi & = & 0, \\ & x,\ y,\ \phi,\ z,\ w,\ \psi & \geq & 0. \end{array} \tag{21.21}$$

We say that a feasible solution $(\bar{x}, \bar{y}, \bar{\phi}, \bar{z}, \bar{w}, \bar{\psi})$ is *strictly complementary* if $\bar{x}_j + \bar{z}_j > 0$ for all j, $\bar{y}_i + \bar{w}_i > 0$ for all i, and $\bar{\phi} + \bar{\psi} > 0$. Theorem 10.6 ensures the existence of such a solution (why?).

The following theorem summarizes and extends the motivating discussion given in Section 1.

THEOREM 21.8. *Suppose that $(\bar{x}, \bar{y}, \bar{\phi}, \bar{z}, \bar{w}, \bar{\psi})$ is a strictly complementary feasible (hence, optimal) solution to* (21.21).

(1) *If $\bar{\phi} > 0$, then $x^* = \bar{x}/\bar{\phi}$ is optimal for the primal problem* (21.1) *and $y^* = \bar{y}/\bar{\phi}$ is optimal for its dual* (21.2).
(2) *If $\bar{\phi} = 0$, then either $c^T\bar{x} > 0$ or $b^T\bar{y} < 0$.*

(a) *If $c^T\bar{x} > 0$, then the dual problem is infeasible.*
(b) *If $b^T\bar{y} < 0$, then the primal problem is infeasible.*

PROOF. Part (1) was proved in Section 1. For part (2), suppose that $\bar{\phi} = 0$. By strict complementarity, $\bar{\psi} > 0$. Hence, $\bar{x}$ and $\bar{y}$ satisfy

$$\begin{aligned} A^T\bar{y} &\geq 0, \\ A\bar{x} &\leq 0, \\ b^T\bar{y} &< c^T\bar{x}. \end{aligned} \tag{21.22}$$

From the last inequality, we see that it is impossible to have $b^T\bar{y} \geq 0$ and $c^T\bar{x} \leq 0$. That is, either $c^T\bar{x} > 0$ or $b^T\bar{y} < 0$ (or both). Suppose, without loss of generality, that $c^T\bar{x} > 0$. We will prove by contradiction that the dual problem is infeasible. To this end, suppose that there exists a vector $y^0 \geq 0$ such that

$$A^T y^0 \geq c. \tag{21.23}$$

Since $\bar{x} \geq 0$, we can multiply by it without changing the direction of an inequality. So multiplying (21.23) on the left by $\bar{x}^T$, we get

$$\bar{x}^T A^T y^0 \geq \bar{x}^T c.$$

Now, the right-hand side is strictly positive. But inequality (21.22) together with the nonnegativity of y^0 implies that the left-hand side is nonpositive:

$$\bar{x}^T A^T y^0 = (A\bar{x})^T y^0 \leq 0.$$

This is a contradiction and therefore the dual must be infeasible. □

3.1. The Reduced KKT System. The right-hand side in the reduced KKT system involves the vector of infeasibilities. We partition this vector into three parts as follows:

$$\begin{bmatrix} \sigma \\ \rho \\ \gamma \end{bmatrix} = \begin{bmatrix} -A^T & c \\ A & -b \\ -c^T & b^T \end{bmatrix} \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} + \begin{bmatrix} z \\ w \\ \psi \end{bmatrix} = \begin{bmatrix} -A^T y + c\phi + z \\ Ax - b\phi + w \\ -c^T x + b^T y + \phi \end{bmatrix}.$$

The reduced KKT system for (21.3) is given by

$$\begin{bmatrix} -X^{-1}Z & -A^T & c \\ A & -Y^{-1}W & -b \\ -c^T & b^T & -\psi/\phi \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\phi \end{bmatrix} = \begin{bmatrix} \hat{\sigma} \\ \hat{\rho} \\ \hat{\gamma} \end{bmatrix}, \tag{21.24}$$

where

$$\begin{bmatrix} \hat{\sigma} \\ \hat{\rho} \\ \hat{\gamma} \end{bmatrix} = \begin{bmatrix} -(1-\delta)\sigma + z - \delta\mu X^{-1}e \\ -(1-\delta)\rho + w - \delta\mu Y^{-1}e \\ -(1-\delta)\gamma + \psi - \delta\mu/\phi \end{bmatrix}.$$

This system is not symmetric. One could use a general purpose equation solver to solve it, but its special structure would be mostly ignored by such a solver. To exploit the structure, we solve this system in two stages. We start by using the first two equations to solve simultaneously for Δx and Δy in terms of $\Delta\phi$:

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \begin{bmatrix}-X^{-1}Z & -A^T\\ A & -Y^{-1}W\end{bmatrix}^{-1}\left(\begin{bmatrix}\hat\sigma\\ \hat\rho\end{bmatrix} - \begin{bmatrix}c\\ -b\end{bmatrix}\Delta\phi\right).$$

Introducing abbreviating notations, we can write

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \begin{bmatrix}f_x\\ f_y\end{bmatrix} - \begin{bmatrix}g_x\\ g_y\end{bmatrix}\Delta\phi, \tag{21.25}$$

where the vectors

$$f = \begin{bmatrix}f_x\\ f_y\end{bmatrix} \quad \text{and} \quad g = \begin{bmatrix}g_x\\ g_y\end{bmatrix}$$

are found by solving the following two systems of equations:

$$\begin{bmatrix}-X^{-1}Z & -A^T\\ A & -Y^{-1}W\end{bmatrix}\begin{bmatrix}f_x\\ f_y\end{bmatrix} = \begin{bmatrix}\hat\sigma\\ \hat\rho\end{bmatrix}$$

and

$$\begin{bmatrix}-X^{-1}Z & -A^T\\ A & -Y^{-1}W\end{bmatrix}\begin{bmatrix}g_x\\ g_y\end{bmatrix} = \begin{bmatrix}c\\ -b\end{bmatrix}.$$

Then we use (21.25) to eliminate Δx and Δy from the last equation in (21.24):

$$\begin{bmatrix}-c^T & b^T\end{bmatrix}\left(\begin{bmatrix}f_x\\ f_y\end{bmatrix} - \begin{bmatrix}g_x\\ g_y\end{bmatrix}\Delta\phi\right) - \frac{\psi}{\phi}\Delta\phi = \hat\gamma.$$

We then solve for $\Delta\phi$:

$$\Delta\phi = \frac{c^T f_x - b^T f_y + \hat\gamma}{c^T g_x - b^T g_y - \psi/\phi}.$$

Given $\Delta\phi$, (21.25) determines Δx and Δy. Once these vectors are known, (21.20) is used to compute the step directions for the slack variables:

$$\begin{aligned}\Delta z &= -X^{-1}Z\Delta x + \delta\mu X^{-1}e - z\\ \Delta w &= -Y^{-1}W\Delta y + \delta\mu Y^{-1}e - w\\ \Delta\psi &= -\frac{\psi}{\phi}\Delta\phi + \delta\mu/\phi - \psi.\end{aligned}$$

We now see that the reduced KKT system can be solved by solving two systems of equations for f and g. These two systems both involve the same matrix. Furthermore, these systems can be formulated as quasidefinite systems by negating the first equation

and then reordering the equations appropriately. For example, the quasidefinite system for g is

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} g_y \\ g_x \end{bmatrix} = \begin{bmatrix} -b \\ -c \end{bmatrix}.$$

Therefore, the techniques developed in Chapter 19 can be used to solve these systems of equations. In particular, to solve the two systems, the quasidefinite matrix only needs to be factored once. Then the two systems can be solved by doing two forward and two backward substitutions. Since factorization requires more computation than forward and backward substitutions, one would expect to be able to solve these two systems in much less time than if they were each being solved from scratch. In fact, it is often the case that one can solve two systems involving the same quasidefinite matrix in little more time than is required to solve just one such system.

The full homogeneous self-dual method is summarized in Figure 21.1.

4. Simplex Method vs Interior-Point Methods

Finally, we compare the performance of interior-point methods with the simplex method. For this comparison, we have chosen the homogeneous self-dual method described in this chapter and the self-dual simplex method (see Figure 7.1). In the interest of efficiency certain liberties have been taken with the implementations. For example, in the homogeneous self-dual method, (17.6) is used to compute "long" step lengths instead of the more conservative "short" step lengths in (21.10). The code fragments implementing each of these two algorithms are shown in Appendix A.

A standard collection of test problems, the so-called NETLIB suite, were used in the comparison. Problems in this collection are formulated with bounds and ranges:

$$\begin{array}{lrcl} \text{minimize} & c^T x & & \\ \text{subject to} & b \leq & Ax & \leq b + r \\ & l \leq & x & \leq u. \end{array}$$

However, to keep the algorithms as simple as possible, they were implemented only for problems in our standard inequality form. Therefore, the problems from the NETLIB suite were converted to standard form as follows:

$$\begin{array}{lrcl} - \text{ maximize} & -c^T x - c^T l & & \\ \text{subject to} & -Ax & \leq & -b + Al \\ & Ax & \leq & b + r - Al \\ & x & \leq & u - l \\ & x & \geq & 0. \end{array}$$

initialize

$(x, y, \phi, z, w, \psi) = (e, e, 1, e, e, 1)$

while (not optimal) {

$\mu = (z^T x + w^T y + \psi\phi)/(n + m + 1)$

$\delta = \begin{cases} 0, & \text{on odd iterations} \\ 1, & \text{on even iterations} \end{cases}$

$\hat{\rho} = -(1 - \delta)(Ax - b\phi + w) + w - \delta\mu Y^{-1} e$

$\hat{\sigma} = -(1 - \delta)(-A^T y + c\phi + z) + z - \delta\mu X^{-1} e$

$\hat{\gamma} = -(1 - \delta)(b^T y - c^T x + \psi) + \psi - \delta\mu/\phi$

solve the two $(n + m) \times (n + m)$ quasidefinite systems:

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} f_y \\ f_x \end{bmatrix} = \begin{bmatrix} \hat{\rho} \\ -\hat{\sigma} \end{bmatrix}$$

and

$$\begin{bmatrix} -Y^{-1}W & A \\ A^T & X^{-1}Z \end{bmatrix} \begin{bmatrix} g_y \\ g_x \end{bmatrix} = \begin{bmatrix} -b \\ -c \end{bmatrix}$$

$$\Delta\phi = \frac{c^T f_x - b^T f_y + \hat{\gamma}}{c^T g_x - b^T g_y - \psi/\phi}$$

$$\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} - \begin{bmatrix} g_x \\ g_y \end{bmatrix} \Delta\phi$$

$\Delta z = -X^{-1} Z \Delta x + \delta\mu X^{-1} e - z$

$\Delta w = -Y^{-1} W \Delta y + \delta\mu Y^{-1} e - w$

$\Delta\psi = -\frac{\psi}{\phi}\Delta\phi + \delta\mu/\phi - \psi$

$\theta = \begin{cases} \max\{t : (x(t), \ldots, \psi(t)) \in \mathcal{N}(1/2)\}, & \text{on odd iterations} \\ 1, & \text{on even iterations} \end{cases}$

$x \leftarrow x + \theta\Delta x, \qquad z \leftarrow z + \theta\Delta z$

$y \leftarrow y + \theta\Delta y, \qquad w \leftarrow w + \theta\Delta w$

$\phi \leftarrow \phi + \theta\Delta\phi, \qquad \psi \leftarrow \psi + \theta\Delta\psi$

}

FIGURE 21.1. The Homogeneous Self-Dual Method.

Of course, this transformation is invalid when any of the lower bounds are infinite. Therefore, such problems have been dropped in our experiment. Also, this transformation introduces significant computational inefficiency but, since it was applied equally to the problems presented to both methods, the comparison remains valid.

The results of our experiment are shown in Table 21.1. The most obvious observation is that the simplex method is generally faster and that, for many problems, the slower method is not more than 3 or 4 times slower. For problems in this suite, these results are consistent with results reported in the literature. However, it must be noted that the problems in this suite range only from small to medium in size. The largest problem, fit2p, has about 3000 constraints and about 14000 variables. By today's standards, this problem is considered of medium size. For larger problems, reports in the literature indicate that interior point methods tend to be superior although the results are very much dependent on the specific class of problems. In the remaining chapters of this book we shall consider various extensions of the linear programming model. We shall see that the simplex method is particularly well suited for solving integer programming problems studied in Chapter 22 whereas interior point methods are more appropriate for extensions into the quadratic and convex programming problems studied in Chapters 23 and 24. These considerations are often more important than speed. There are, of course, exceptions. For example, the interior-point method is about 900 times faster than the simplex method on problem fit2p. Such a difference cannot be ignored.

When comparing algorithms it is always tempting to look for ways to improve the slower method. There are obvious enhancements to the interior-point method used in this implementation. For example, one could use the same LDL^T factorization to compute both the predictor and the corrector directions. When implemented properly, this enhancement alone can almost halve the times for this method.

Of course, the winning algorithm can also be improved (but, significant overall improvements such as the one just mentioned for the interior-point method are not at all obvious). Looking at the table, we note that the interior-point method solved both fit2p and fit2d in roughly the same amount of time. These two problems are duals of each other and hence any algorithm that treats the primal and the dual symmetrically should take about the same time to solve them. Now, look at the simplex method's performance on these two problems. There is a factor of 36 difference between them. The reason is that, even though we have religiously adhered to primal-dual symmetry in our development of the simplex method, an asymmetry did creep in. To see it, note that the basic matrix is always a square submatrix of $\left[\begin{array}{cc} A & I \end{array}\right]$. That is, it is an $m \times m$ matrix. If we apply the algorithm to the dual problem, then the basis matrix is $n \times n$. Hence, even though the sequence of iterates generated should be identical with the two problems, the computations involved in each iteration can be very different if m and n are not about the same. This is the case for the fit2p/fit2d pair. Of course, one can easily think up schemes to overcome this difficulty. But even if the performance of the simplex method on fit2p can be brought in line with its performance on fit2d, it will still be about 25 times slower than the interior-point on this problem – a difference that remains significant.

Name	Time		Name	Time	
	Simplex Method	Interior Point		Simplex Method	Interior Point
25fv47	2m55.70s	3m14.82s	maros	1m0.87s	3m19.43s
80bau3b	7m59.57s	2m34.84s	nesm	1m40.78s	6m21.28s
adlittle	0m0.26s	0m0.47s	pilot87	*	*
afiro	0m0.03s	0m0.11s	pilotnov	*	4m15.31s
agg	0m1.09s	0m4.59s	pilots	*	32m48.15s
agg2	0m1.64s	0m21.42s	recipe	0m0.21s	0m1.04s
agg3	0m1.72s	0m26.52s	sc105	0m0.28s	0m0.37s
bandm	0m15.87s	0m9.01s	sc205	0m1.30s	0m0.84s
beaconfd	0m0.67s	0m6.42s	sc50a	0m0.09s	0m0.17s
blend	0m0.40s	0m0.56s	sc50b	0m0.12s	0m0.15s
bnl1	0m38.38s	0m46.09s	scagr25	0m12.93s	0m4.44s
bnl2	3m54.52s	10m19.04s	scagr7	0m1.16s	0m1.05s
boeing1	0m5.56s	0m9.14s	scfxm1	0m4.44s	0m7.80s
boeing2	0m0.80s	0m1.72s	scfxm2	0m14.33s	0m18.84s
bore3d	0m1.17s	0m3.97s	scfxm3	0m28.92s	0m28.92s
brandy	0m5.33s	0m8.44s	scorpion	0m3.38s	0m2.64s
czprob	0m50.14s	0m41.77s	scrs8	0m7.15s	0m9.53s
d2q06c	*	1h11m1.93s	scsd1	0m0.86s	0m3.88s
d6cube	2m46.71s	13m44.52s	scsd6	0m2.89s	0m9.31s
degen2	0m17.28s	0m17.02s	scsd8	0m28.87s	0m16.82s
degen3	5m55.52s	3m36.73s	sctap1	0m2.98s	0m3.08s
dfl001	8h55m33.05s	**	sctap2	0m7.41s	0m12.03s
e226	0m4.76s	0m6.65s	sctap3	0m11.70s	0m17.18s
etamacro	0m17.94s	0m43.40s	seba	0m27.25s	0m11.90s
fffff800	0m10.07s	1m9.15s	share1b	0m2.07s	0m10.90s
finnis	0m4.76s	0m6.17s	share2b	0m0.47s	0m0.71s
fit1d	0m18.15s	0m11.63s	shell	0m16.12s	0m29.45s
fit1p	7m10.86s	0m16.47s	ship04l	0m3.82s	0m13.60s
fit2d	1h3m14.37s	4m27.66s	ship04s	0m3.48s	0m10.81s
fit2p	36h31m31.80s	2m35.67s	ship08l	0m17.83s	0m39.06s
forplan	0m3.99s	*	ship08s	0m8.85s	0m19.64s
ganges	0m44.27s	0m34.89s	ship12l	0m26.55s	1m8.62s
gfrdpnc	0m11.51s	0m8.46s	ship12s	0m16.75s	0m30.33s
greenbea	22m45.49s	43m4.32s	sierra	0m10.88s	0m42.89s
grow15	0m8.55s	0m58.26s	standata	0m0.57s	0m6.60s
grow22	0m11.79s	2m0.53s	standmps	0m2.41s	0m13.44s
grow7	0m3.61s	0m13.57s	stocfor1	0m0.22s	0m0.92s
israel	0m1.83s	0m2.66s	stocfor2	0m45.15s	0m40.43s
kb2	0m0.15s	0m0.34s	wood1p	0m14.15s	7m18.47s
lotfi	0m0.81s	0m3.36s	woodw	1m48.14s	8m53.92s
maros-r7	*	1h31m12.06s			

TABLE 21.1. Comparison between the self-dual simplex method and the homogeneous self-dual interior-point method. (*) denotes numerical difficulties. (**) denotes insufficient memory.

Exercises

21.1 When $n = 1$, the set $\mathcal{N}(\beta)$ is a subset of $\mathbb{R}^2$. Graph it.

21.2 Suppose there is an algorithm for which one can prove that

$$\mu^{(k)} \leq \left(1 - \frac{a}{f(n)}\right)^k,$$

for every $k \geq 1$, where $f(n)$ denotes a specific function of n, such as $f(n) = n^2$, and a is a constant. In terms of a and f and the "precision" L, give a (tight) upper bound on the number of iterations that would be sufficient to guarantee that

$$\mu^{(k)} \leq 2^{-L}.$$

21.3 In Section 3 of Chapter 19, we extended the primal-dual path-following method to treat problems in general form. Extend the homogeneous self-dual method in the same way.

21.4 *Long-step variant.* Let

$$\mathcal{M}(\beta) = \{(x, z) : \min XZe \geq (1 - \beta)\mu(x, z)\}.$$

(The notation $\min XZe$ denotes the scalar that is the minimum of all the components of the vector XZe. Throughout this problem, given any vector v, the notation $\min v$ ($\max v$) will denote the minimum (maximum) of the components of v.) Fix $\frac{1}{2} < \beta < 1$ (say $\beta = 0.95$). A *long-step* variant of the homogeneous self-dual method starts with an initial $(x, z) \in \mathcal{M}(\beta)$ and in every iteration uses

$$\delta = 2(1 - \beta)$$

and

$$\theta = \max\{t : (x + t\Delta x, z + t\Delta z) \in \mathcal{M}(\beta)\}.$$

The goal of this exercise is to analyze the complexity of this algorithm by completing the following steps.

(a) Show that $\mathcal{N}(\beta) \subset \mathcal{M}(\beta) \subset \mathcal{M}(1) = \{(x, z) > 0\}$.

(b) Show that $\max(-PQe) \leq \|r\|^2/4$. *Hint: Start by writing*

$$p_j q_j \geq \sum_{i:p_i q_i < 0} p_i q_i$$

and then use the facts that $p^T q = 0$, $p_i + q_i = r_i$, and that for any two real numbers a and b, $(a + b)^2 \geq 4ab$ (prove this).

(c) Show that if $(x, z) \in \mathcal{M}(\beta)$, then $\|r\|^2 \leq n\mu$. *Hint: Use* (21.11) *to write $\|r\|^2 = \sum_j (x_j z_j - \delta\mu)^2/x_j z_j$. Expand the numerator, use the definitions of μ and δ to simplify, and then use the assumption that $(x, z) \in \mathcal{M}(\beta)$ to take care of the remaining denominator.*

(d) Show that if $(x, z) \in \mathcal{M}(\beta)$, then

$$\theta \geq \min\{1, -\frac{\beta\delta\mu}{\min PQe}\} \geq \frac{4\beta\delta}{n}.$$

Hint: Using the same notation as in the proof of Theorem 21.3, fix $t \leq \min\{1, -\beta\delta\mu/\min PQe\}$*, write*

$$x_j(t)z_j(t) - \mu(t) = (1-t)(x_jz_j - \mu) + t^2\Delta x_j\Delta z_j,$$

and then replace the right-hand side by a lower bound that is independent of j*. From there, follow your nose until you get the first inequality. The second inequality follows from parts* (b) *and* (c).

(e) As usual letting $\mu^{(k)}$ denote the μ value associated with the solution on the kth iteration of the algorithm, show that

$$\mu^{(k)} \leq \left(1 - \frac{4\beta\delta}{n}(1-\delta)\right)^k.$$

(f) Give an upper bound on the number of iterations required to get $\mu^{(k)} \leq 2^{-L}$.

(g) Show that θ can be computed by solving n (univariate) quadratic equations.

(h) A robust implementation of a quadratic equation solver uses the formula

$$x = \begin{cases} \frac{-b-\sqrt{b^2-4ac}}{2a}, & b \geq 0, \\ \frac{2c}{-b+\sqrt{b^2-4ac}}, & b < 0, \end{cases}$$

for one of the two roots to $ax^2 + bx + c = 0$ (a similar formula is used for the other one). Show that the two expressions on the right are mathematically equal and suggest a reason to prefer one over the other in the particular cases indicated.

Notes

The first study of homogeneous self-dual problems appeared in Tucker (1956). This chapter is based on the papers Mizuno et al. (1993), Ye et al. (1994), and Xu et al. (1993). The step length formula (21.10) forces the algorithm studied in this chapter to take much shorter steps than those in Chapter 17. In general, algorithms that are based on steps that confine the iterates to $\mathcal{N}(\beta)$ are called *short-step methods*. A *long-step* variant of the algorithm can be obtained by enlarging the set $\mathcal{N}(\beta)$. Such a variant is the subject of Exercise 21.4. For this method, a worst case analysis shows that it takes on the order of n steps to achieve a given level of precision. Xu et al. (1993) describes an efficient implementation of the long-step variant.

The predictor–corrector method is a standard technique used in the numerical solution of ordinary differential equations. Mehrotra (1992) (see also Mehrotra (1989))

was the first to apply this technique in the context of interior-point methods, although the related notion of forming power series approximations was suggested earlier by N.K. Karmarkar and is described in Adler et al. (1989).

Part 4

Extensions

It’s hard. But it’s harder to ignore it. — C. Stevens

CHAPTER 22

Integer Programming

Many real-world problems could be modeled as linear programs except that some or all of the variables are constrained to be integers. Such problems are called *integer programming problems*. One might think that these problems wouldn't be much harder than linear programming problems. For example, we saw in Chapter 13 that for network flow problems with integer data, the simplex method automatically produces integer solutions. But that was just luck. In general, one can't expect to get integer solutions; in fact, as we shall see in this chapter, integer programming problems turn out to be generally much harder to crack than linear ones.

There are many important real-world problems that can be formulated as integer programming problems. The subject is so important that several monographs are devoted entirely to it. In this chapter, we shall just present a few favorite applications that can be modeled as integer programming problems and then we will discuss one technique for solving problems in this class, called *branch-and-bound*.

1. Scheduling Problems

There are many problems that can be classified as *scheduling problems*. We shall consider just two related problems of this type: the equipment scheduling and crew scheduling problems faced by large airlines. Airlines determine how to route their planes as follows. First, a number of specific flight *legs* are defined based on market demand. A leg is by definition one flight taking off from somewhere at some time and landing somewhere else (hopefully). For example, a leg could be a flight from New York directly to Chicago departing at 7:30 A.M. Another might be a flight from Chicago to San Francisco departing at 1:00 P.M. The important point is that these legs are defined by market demand, and it is therefore not clear a priori how to put these legs together in such a way that the available aircraft can cover all of them. That is, for each airplane, the airline must put together a *route* that it will fly. A route, by definition, consists of a sequence of flight legs for which the destination of one leg is the origin of the next (and, of course, the final destination must be the origin of the first leg, forming a closed loop).

The airline scheduling problems are generally tackled in two stages. First, reasonable routes are identified that meet various regulatory and temporal constraints (you can't leave somewhere before you've arrived there—time also must be reserved for

dropping off and taking on passengers). This route-identification problem is by no means trivial, but it isn't our main interest here, so we shall simply assume that a collection of reasonable routes has already been identified. Given the potential routes, the second stage is to select a subset of them with the property that each leg is covered by exactly one route. If the collection of potential routes is sufficiently rich, we would expect there to be several feasible solutions. Therefore, as always, our goal is to pick an optimal one, which in this case we define as one that minimizes the total cost. To formulate this problem as an integer program, let

$$x_j = \begin{cases} 1 & \text{if route } j \text{ is selected,} \\ 0 & \text{otherwise,} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if leg } i \text{ is part of route } j, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$c_j = \text{cost of using route } j.$$

With these notations, the *equipment scheduling problem* is to

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j = 1 \qquad i = 1, 2, \ldots, m, \\ & x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n. \end{aligned}$$

This model is often called a *set-partitioning problem*, since the set of legs gets divided, or partitioned, among the various routes.

The flight crews do not necessarily follow the same aircraft around a route. The main reason is that the constraints that apply to flight crews differ from those for the aircraft (for example, flight crews need to sleep occasionally). Hence, the problem has a different set of potential routes. Also, it is sometimes reasonable to allow crews to ride as passengers on some legs with the aim of getting in position for a subsequent flight. With these changes, the *crew scheduling problem* is

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, 2, \ldots, m, \\ & x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n. \end{aligned}$$

This model is often referred to as a *set-covering problem*, since the crews are assigned so as to cover each leg.

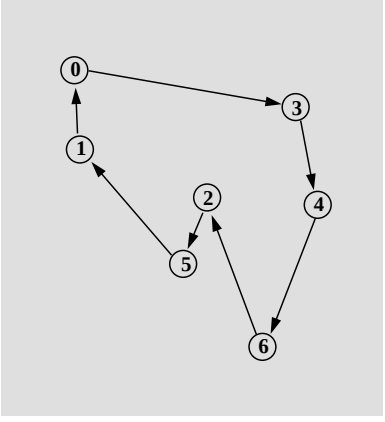


FIGURE 22.1. A feasible tour in a seven-city traveling salesman problem.

2. The Traveling Salesman Problem

Consider a salesman who needs to visit each of n cities, which we shall enumerate as $0, 1, \ldots, n-1$. His goal is to start from his home city, 0, and make a tour visiting each of the remaining cities once and only once and then returning to his home. We assume that the "distance" between each pair of cities, c_{ij}, is known (distance does not necessarily have to be distance—it could be travel time or, even better, the cost of travel) and that the salesman wants to make the tour that minimizes the total distance. This problem is called the *traveling salesman problem*. Figure 22.1 shows an example with seven cities. Clearly, a tour is determined by listing the cities in the order in which they will be visited. If we let s_i denote the ith city visited, then the tour can be described simply as

$$s_0 = 0, s_1, s_2, \ldots, s_{n-1}.$$

The total number of possible tours is equal to the number of ways one can permute the $n-1$ cities, i.e., $(n-1)!$. Factorials are huge even for small n (for example, $50! = 3.041 \times 10^{64}$). Hence, enumeration is out of the question. Our aim is to formulate this problem as an integer program that can be solved more quickly than by using enumeration.

It seems reasonable to introduce for each (i, j) a decision variable x_{ij} that will be equal to one if the tour visits city j immediately after visiting city i; otherwise, it will be equal to zero. In terms of these variables, the objective function is easy to write:

$$\text{minimize } \sum_i \sum_j c_{ij} x_{ij}.$$

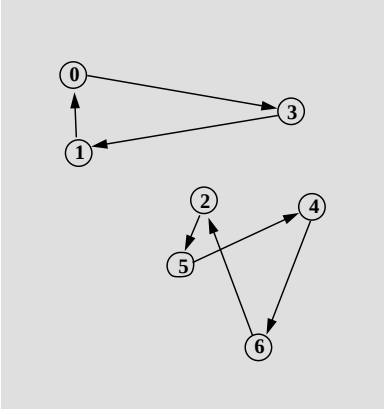


FIGURE 22.2. Two disjoint subtours in a seven-city traveling salesman problem.

The tricky part is to formulate constraints to guarantee that the set of nonzero x_{ij}'s corresponds exactly to a bonafide tour. Some of the constraints are fairly obvious. For example, after the salesman visits city i, he must go to one and only one city next. We can write these constraints as

$$\sum_j x_{ij} = 1, \qquad i = 0, 1, \ldots, n-1 \tag{22.1}$$

(we call them the *go-to constraints*). Similarly, when the salesman visits a city, he must have come from one and only one prior city. That is,

$$\sum_i x_{ij} = 1, \qquad j = 0, 1, \ldots, n-1 \tag{22.2}$$

(by analogy we call these the *come-from constraints*). If the go-to and the come-from constraints are sufficient to ensure that the decision variables represent a tour, the traveling salesman problem would be quite easy to solve because it would just be an assignment problem, which can be solved efficiently by the simplex method. But unfortunately, these constraints are not sufficient, since they do not rule out the possibility of forming disjoint subtours. An example is shown in Figure 22.2.

We need to introduce more constraints to guarantee connectivity of the graph that the tour represents. To see how to do this, consider a specific tour

$$s_0 = 0, s_1, s_2, \ldots, s_{n-1}.$$

Let t_i for $i = 0, 1, \ldots, n$ be defined as the number of the stop along the tour at which city i is visited; i.e., "when" city i is visited along the tour. From this definition, we

see that $t_0 = 0,\ t_{s_1} = 1,\ t_{s_2} = 2$, etc. In general,

$$t_{s_i} = i, \qquad i = 0, 1, \ldots, n-1,$$

so that we can think of the t_j's as being the inverse of the s_i's. For a bonafide tour,

$$t_j = t_i + 1, \qquad \text{if } x_{ij} = 1.$$

Also, each t_i is an integer between 0 and $n-1$, inclusive. Hence, t_j satisfies the following constraints:

$$t_j \geq \begin{cases} t_i + 1 - n & \text{if } x_{ij} = 0, \\ t_i + 1 & \text{if } x_{ij} = 1. \end{cases}$$

(Note that by subtracting n in the $x_{ij} = 0$ case, we have effectively made the condition always hold.) These constraints can be written succinctly as

$$t_j \geq t_i + 1 - n(1 - x_{ij}), \qquad i \geq 0, j \geq 1, i \neq j. \tag{22.3}$$

Now, these constraints were derived based on conditions that a bonafide tour satisfies. It turns out that they also force a solution to be a bonafide tour. That is, they rule out subtours. To see this, suppose to the contrary that there exists a solution to (22.1), (22.2), and (22.3) that consists of at least two subtours. Consider a subtour that does not include city 0. Let r denote the number of legs on this subtour. Clearly, $r \geq 2$. Now, sum (22.3) over all arcs on this subtour. On the left, we get the sum of the t_j's over each city visited by the subtour. On the right, we get the same sum plus r. Cancelling the sums from the two sides, we get that

$$0 \geq r,$$

which is a contradiction. Hence, the traveling salesman problem can be formulated as the following integer programming problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, && i = 0, 1, \ldots, n-1, \\
& \sum_{i=1}^{n} x_{ij} = 1, && j = 0, 1, \ldots, n-1, \\
& t_j \geq t_i + 1 - n(1 - x_{ij}), && i \geq 0, j \geq 1, i \neq j, \\
& t_0 = 0, \\
& x_{ij} \in \{0, 1\}, \\
& t_i \in \{0, 1, 2, \ldots\}.
\end{aligned}$$

Note that, for the n-city problem, there are $n^2 + n$ variables in this formulation.

3. Fixed Costs

The terms in an objective function often represent costs associated with engaging in an activity. Until now, we've always assumed that each of these terms is a linear function such as cx. However, it is sometimes more realistic to assume that there is a fixed cost for engaging in the activity plus a linear variable cost. That is, one such term might have the form

$$c(x) = \begin{cases} 0 & \text{if } x = 0 \\ K + cx & \text{if } x > 0. \end{cases}$$

If we assume that there is an upper bound on the size of x, then it turns out that such a function can be equivalently modeled using strictly linear functions at the expense of introducing one integer-valued variable. Indeed, suppose that u is an upper bound on the x variable. Let y denote a $\{0, 1\}$-valued variable that is one when and only when $x > 0$. Then

$$c(x) = Ky + cx.$$

Also, the condition that y is one exactly when $x > 0$ can be guaranteed by introducing the following constraints:

$$\begin{aligned} x &\leq uy \\ x &\geq 0 \\ y &\in \{0, 1\}. \end{aligned}$$

Of course, if the objective function has several terms with associated fixed costs, then this trick must be used on each of these terms.

4. Nonlinear Objective Functions

Sometimes the terms in the objective function are not linear at all. For example, one such term could look like the function shown in Figure 22.3. In Chapter 24, we will discuss efficient algorithms that can be used in the presence of nonlinear objective functions—at least when they have appropriate convexity/concavity properties. In this section, we will show how to formulate an integer programming approximation to a general nonlinear term in the objective function. The first step is to approximate the nonlinear function by a continuous piecewise linear function.

The second step is to introduce integer variables that allow us to represent the piecewise linear function using linear relations. To see how to do this, first we decompose the variable x into a sum,

$$x = x_1 + x_2 + \cdots + x_k,$$

where x_i denotes how much of the interval $[0, x]$ is contained in the ith linear segment of the piecewise linear function (see Figure 22.4). Of course, some of the initial segments will lie entirely within the interval $[0, x]$, one segment will lie partially in and partially out, and then the subsequent segments will lie entirely outside of the interval.

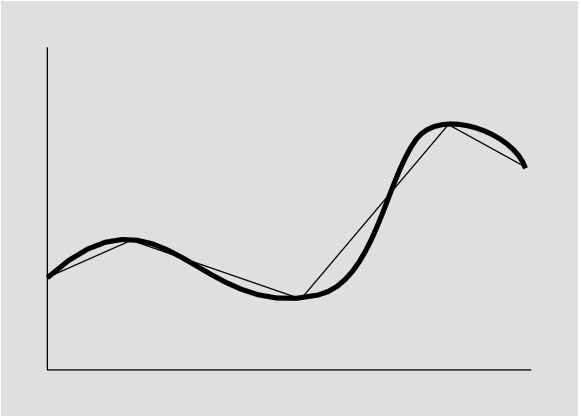


FIGURE 22.3. A nonlinear function and a piecewise linear approximation to it.

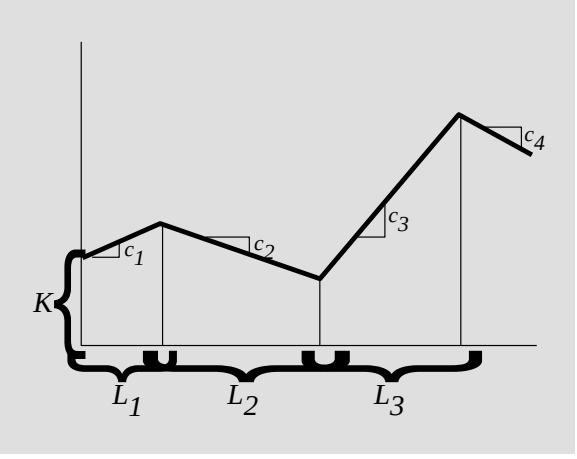


FIGURE 22.4. A piecewise linear function.

Hence, we need to introduce constraints to guarantee that the initial x_i's are equal to the length of their respective segments and that after the straddling segment the subsequent x_i's are all zero. A little thought reveals that the following constraints do the

trick:

$$\begin{aligned} L_j w_j \le x_j \le L_j w_{j-1} \qquad & j = 1, 2, \ldots, k \\ w_0 = 1 & \\ w_j \in \{0, 1\} \qquad & j = 1, 2, \ldots, k \\ x_j \ge 0 \qquad & j = 1, 2, \ldots, k. \end{aligned}$$

Indeed, it follows from these constraints that $w_j \le w_{j-1}$ for $j = 1, 2, \ldots, k$. This inequality implies that once one of the w_j's is zero, then all the subsequent ones must be zero. If $w_j = w_{j-1} = 1$, the two-sided inequality on x_j reduces to $L_j \le x_j \le L_j$. That is, $x_j = L_j$. Similarly, if $w_j = w_{j-1} = 0$, then the two-sided inequality reduces to $x_j = 0$. The only other case is when $w_j = 0$ but $w_{j-1} = 1$. In this case, the two-sided inequality becomes $0 \le x_j \le L_j$. Therefore, in all cases, we get what we want. Now with this decomposition we can write the piecewise linear function as

$$K + c_1 x_1 + c_2 x_2 + \cdots + c_k x_k.$$

5. Branch-and-Bound

In the previous sections, we presented a variety of problems that can be formulated as integer programming problems. As it happens, all of them had the property that the integer variables took just one of two values, namely, zero or one. However, there are other integer programming problems in which the integer variables can be any nonnegative integer. Hence, we define the standard *integer programming problem* as follows:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \le b \\ & x \ge 0 \\ & x \text{ has integer components.} \end{aligned}$$

In this section, we shall present an algorithm for solving these problems. The algorithm is called *branch-and-bound*. It involves solving a (potentially) large number of (related) linear programming problems in its search for an optimal integer solution. The algorithm starts out with the following wishful approach: *first ignore the constraint that the components of x be integers, solve the resulting linear programming problem, and hope that the solution vector has all integer components*. Of course, hopes are almost always unfulfilled, and so a backup strategy is needed. The simplest strategy would be to round each resulting solution value to its nearest integer value. Unfortunately, this naive strategy can be quite bad. In fact, the integer solution so obtained might not even be feasible, which shouldn't be surprising, since we know that the solution to a linear programming problem is at a vertex of the feasible set and so it is quite expected that naive movement will go outside of the feasible set.

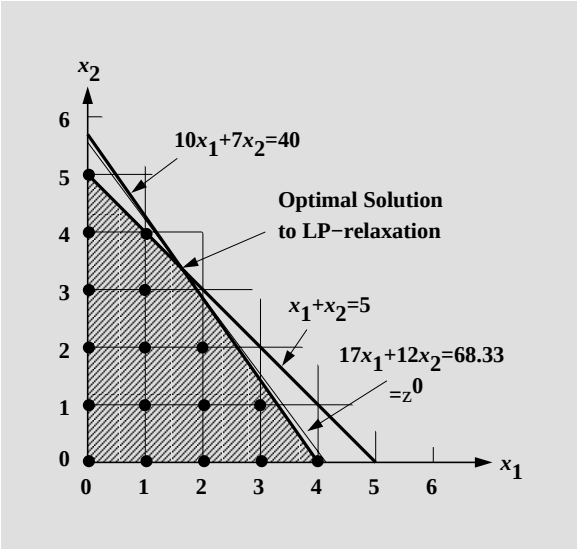


FIGURE 22.5. An integer programming problem. The dots represent the feasible integer points, and the shaded region shows the feasible region for the LP-relaxation.

To be concrete, consider the following example:

$$\begin{aligned} \text{maximize } & 17x_1 + 12x_2 \\ \text{subject to } & 10x_1 + \ 7x_2 \leq 40 \\ & x_1 + \ \ x_2 \leq \ 5 \\ & x_1,\ x_2 \geq \ 0 \\ & x_1,\ x_2 \text{ integers.} \end{aligned}$$

The linear programming problem obtained by dropping the integrality constraint is called the *LP-relaxation*. Since it has fewer constraints, its optimal solution provides an upper bound ζ^0 on the the optimal solution ζ^* to the integer programming problem. Figure 22.5 shows the feasible points for the integer programming problem as well as the feasible polytope for its LP-relaxation. The solution to the LP-relaxation is at $(x_1, x_2) = (5/3, 10/3)$, and the optimal objective value is $205/3 = 68.33$. Rounding each component of this solution to the nearest integer, we get $(2, 3)$, which is not even feasible. The feasible integer solution that is closest to the LP-optimal solution is $(1, 3)$, but we can see from Figure 22.5 that this solution is not the optimal solution to the integer programming problem. In fact, it is easy to see from the figure that

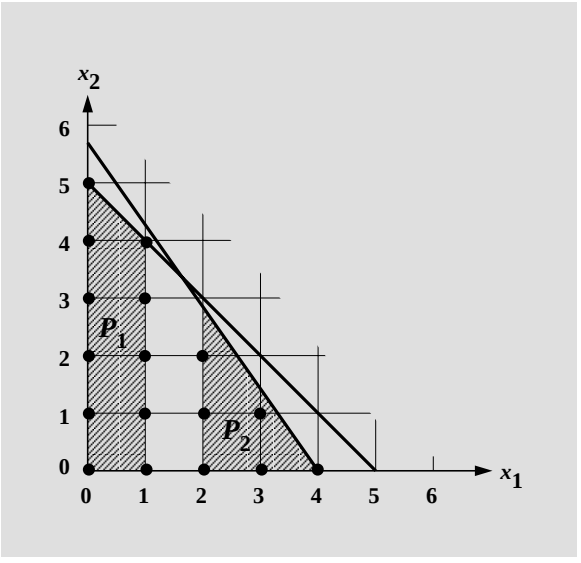


FIGURE 22.6. The feasible subregions formed by the first branch.

the optimal integer solution is either $(1, 4)$ or $(4, 0)$. To make the problem interesting, we've chosen the objective function to make the more distant point $(4, 0)$ be the optimal solution.

Of course, we can solve only very small problems by the graphical method: to solve larger problems, an algorithm is required, which we now describe. Consider variable x_1 in the optimal solution to the LP-relaxation. Its value is $5/3$. In the optimal solution to the integer programming problem, it will be an integer. Hence, it will satisfy either $x_1 \leq 1$ or $x_1 \geq 2$. We consider these two cases separately. Let P_1 denote the linear programming problem obtained by adding the constraint $x_1 \leq 1$ to the LP-relaxation, and let P_2 denote the problem obtained by including the other possibility, $x_1 \geq 2$. The feasible regions for P_1 and P_2 are shown in Figure 22.6. Let us study P_1 first. It is clear from Figure 22.6 that the optimal solution is at $(x_1, x_2) = (1, 4)$ with an objective value of 65. Our algorithm has found its first feasible solution to the integer programming problem. We record this solution as the *best-so-far*. Of course, better ones may (in this case, will) come along later.

Now let's consider P_2. Looking at Figure 22.6 and doing a small amount of calculation, we see that the optimal solution is at $(x_1, x_2) = (2, 20/7)$. In this case, the objective function value is $478/7 = 68.29$. Now if this value had turned out to be less than the best-so-far value, then we'd be done, since any integer solution that lies within the feasible region for P_2 would have a smaller value yet. But this is not the

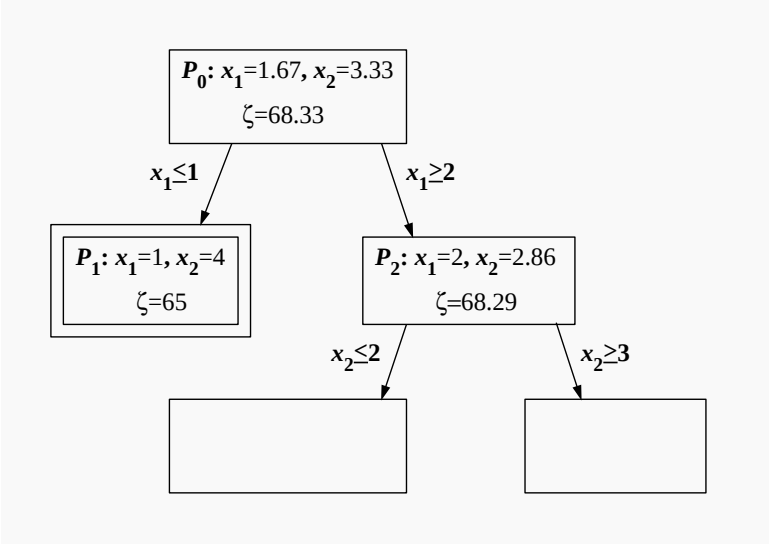


FIGURE 22.7. The beginnings of the enumeration tree.

case, and so we must continue our systematic search. Since $x_2 = 20/7 = 2.86$, we divide P_2 into two subproblems, one in which the constraint $x_2 \leq 2$ is added and one with $x_2 \geq 3$ added.

Before considering these two new cases, note that we are starting to develop a tree of linear programming subproblems. This tree is called the *enumeration tree*. The tree as far as we have investigated is shown in Figure 22.7. The double box around P_1 indicates that that part of the tree is done: i.e., there are no branches emanating from P_1—it is a leaf node. The two empty boxes below P_2 indicate two subproblems that have yet to be studied. Let's proceed by looking at the left branch, which corresponds to adding the constraint $x_2 \leq 2$ to what we had before. We denote this subproblem by P_3. Its feasible region is shown in Figure 22.8, from which we see that the optimal solution is at $(2.6, 2)$. The associated optimal objective value is 68.2. Again, the solution is fractional. Hence, the process of subdividing must continue. This time we subdivide based on the values of x_1. Indeed, we consider two cases: either $x_1 \leq 2$ or $x_1 \geq 3$.

Figure 22.9 shows the enumeration tree as it now stands. At this juncture, there are three directions in which we could proceed. We could either study the other branch under P_2 or work on one of the two branches sitting under P_3. If we were to systematically solve all the problems on a given level of the tree before going deeper, we would be performing what is referred to as a *breadth-first search*. On the other hand, going deep before going wide is called a *depth-first search*. For reasons that we shall

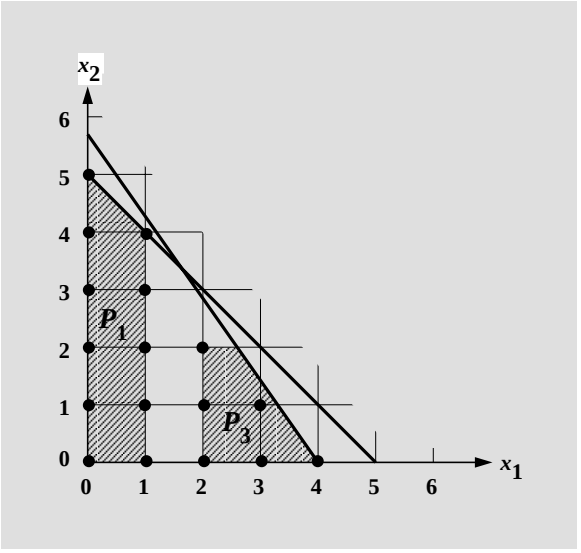


FIGURE 22.8. The refinement of P_2 to P_3.

explain later, it turns out to be better to do a depth-first search. And, to be specific, let us always choose the left branch before the right branch (in practice, there are much better rules that one can employ here). So our next linear programming problem is the one that we get by adding the constraint that $x_1 \leq 2$ to the constraints that defined P_3. Let us call this new problem P_4. Its feasible region is shown in Figure 22.10. It is easy to see that the optimal solution to this problem is $(2, 2)$, with an objective value of 58. This solution is an integer solution, so it is feasible for the integer programming problem. But it is not better than our best-so-far. Nonetheless, we do not need to consider any further subproblems below this one in the enumeration tree.

Since problem P_4 is a leaf in the enumeration tree, we need to work back up the tree looking for the first node that has an unsolved problem sitting under it. For the case at hand, the unsolved problem is on the right branch underneath P_3. Let us call this problem P_5. It too is depicted in Figure 22.10. The optimal solution is $(3, 1.43)$, with an optimal objective function value of 68.14. Since this objective function value is larger than the value of the best-so-far integer solution, we must further investigate by dividing into two possibilities, either $x_2 \leq 1$ or $x_2 \geq 2$. At this point, the enumeration tree looks like that shown in Figure 22.11.

Let P_6 denote the linear programming problem that we get on the left branch under P_5. Its feasible region is shown in Figure 22.12. The optimal solution is $(3.3, 1)$, and the associated objective value is 68.1. Again, the solution is fractional and has a

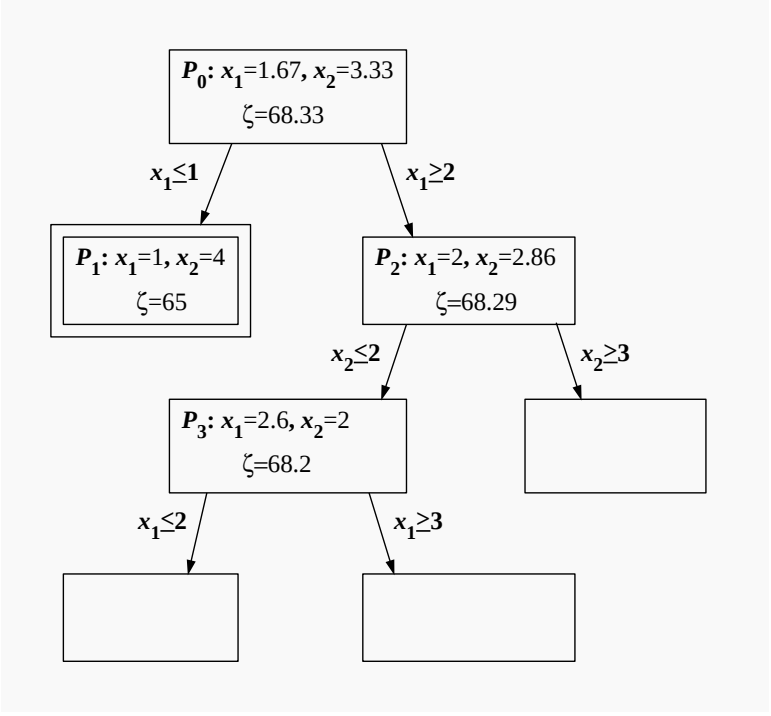


FIGURE 22.9. The enumeration tree after solving P_3.

higher objective value than the best-so-far integer solution. Hence, it must be subdivided based on $x_1 \leq 3$ as opposed to $x_1 \geq 4$. Denoting these two problems by P_7 and P_8, their feasible regions are as depicted in Figure 22.13. The solution to P_7 is $(3, 1)$, and the objective value is 63. This is an integer solution, but it is not better than the best-so-far. Nonetheless, the node becomes a leaf, since the solution is integral. Hence, we move on to P_8. The solution to this problem is also integral, $(4, 0)$. Also, the objective value associated with this solution is 68, which is a new record for feasible integer solutions. Hence, this solution becomes our best-so-far. The enumeration tree at this point is shown in Figure 22.14.

Now we need to go back and solve the problems under P_5 and P_2 (and any subproblems thereof). It turns out that both these subproblems are infeasible, and so no more subdivisions are needed. The enumeration tree is now completely fathomed and is shown in Figure 22.15. We can now assert that the optimal solution to the original integer programming problem was found in problem P_8. The solution is $(x_1, x_2) = (4, 0)$, and the associated objective function value is 68.

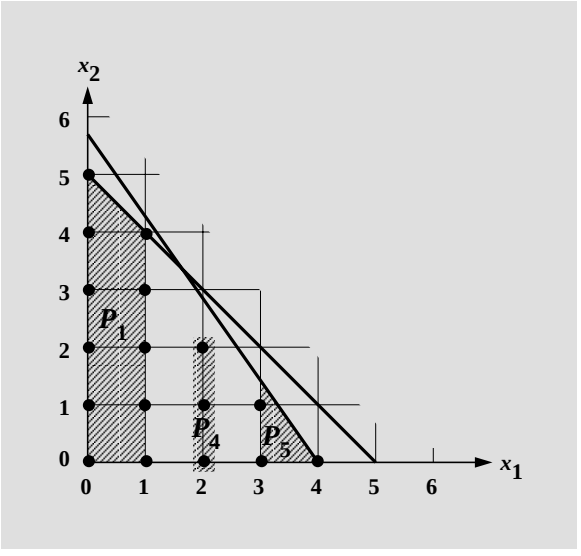


FIGURE 22.10. The refinement of P_3 to P_4.

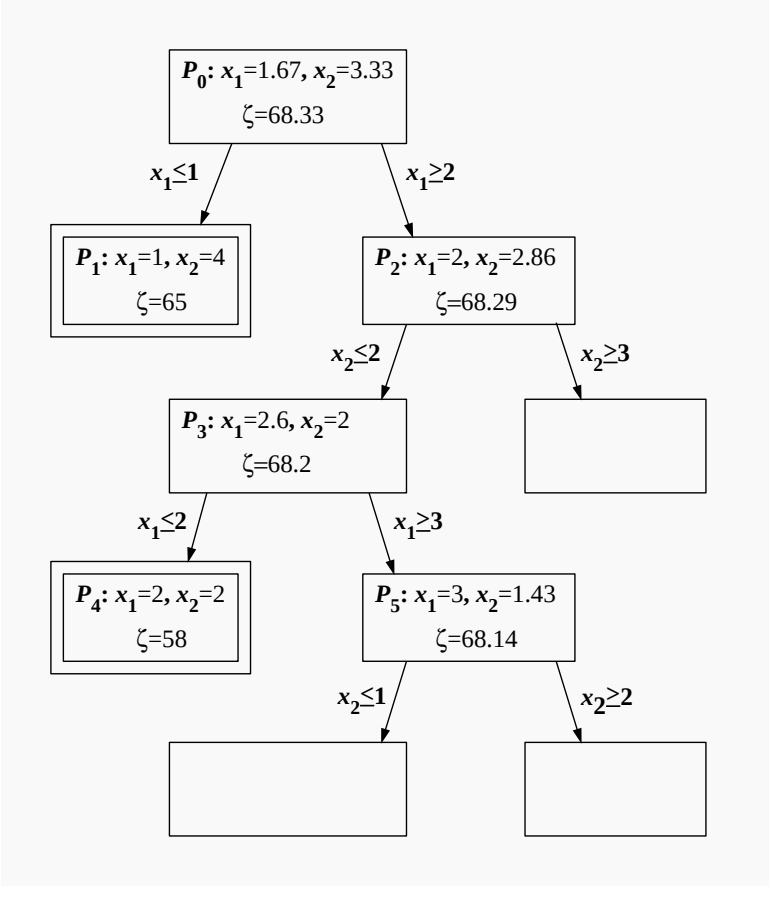


FIGURE 22.11. The enumeration tree after solving P_5. The double box around P_4 indicates that it is a leaf in the tree.

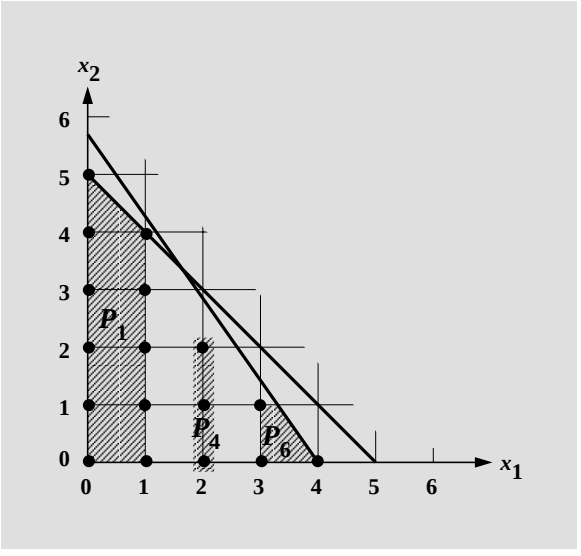


FIGURE 22.12. The refinement of P_5 to P_6.

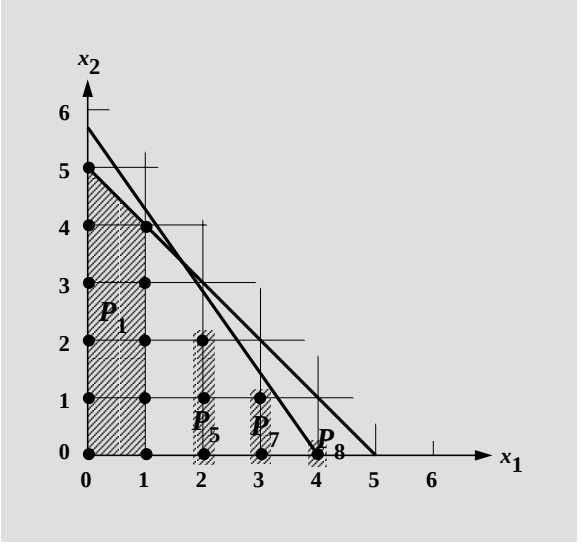


FIGURE 22.13. The refinement of P_6 to P_7 and P_8.

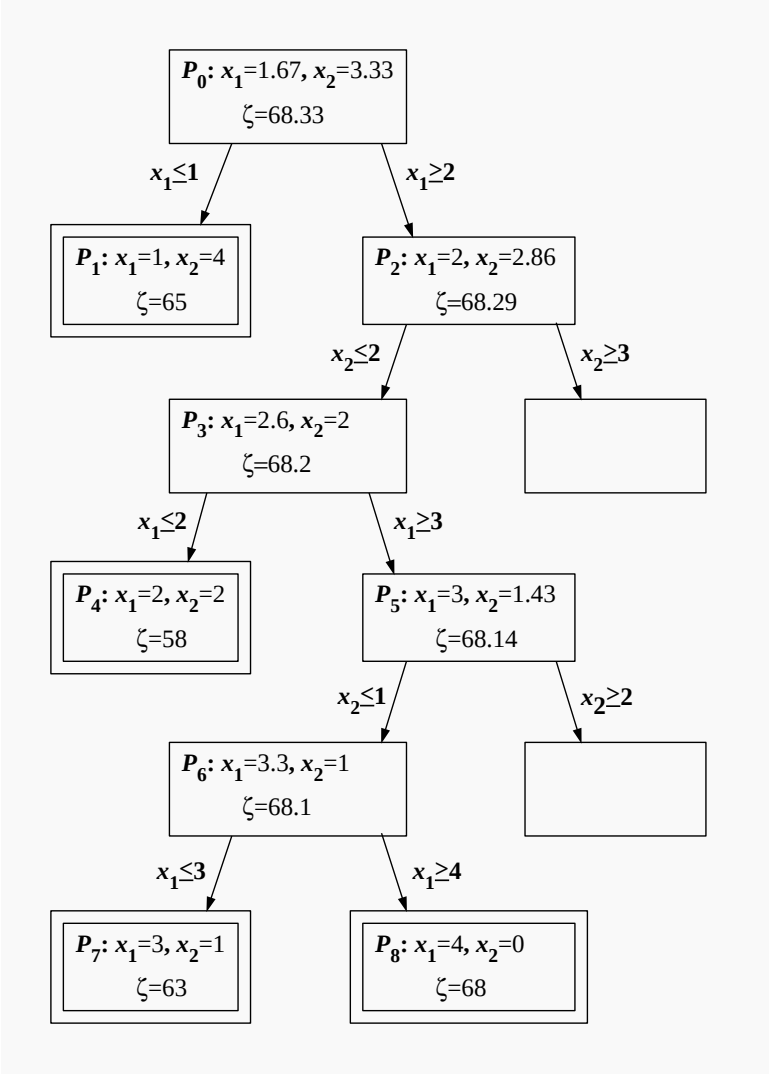


FIGURE 22.14. The enumeration tree after solving P_6, P_7, and P_8.

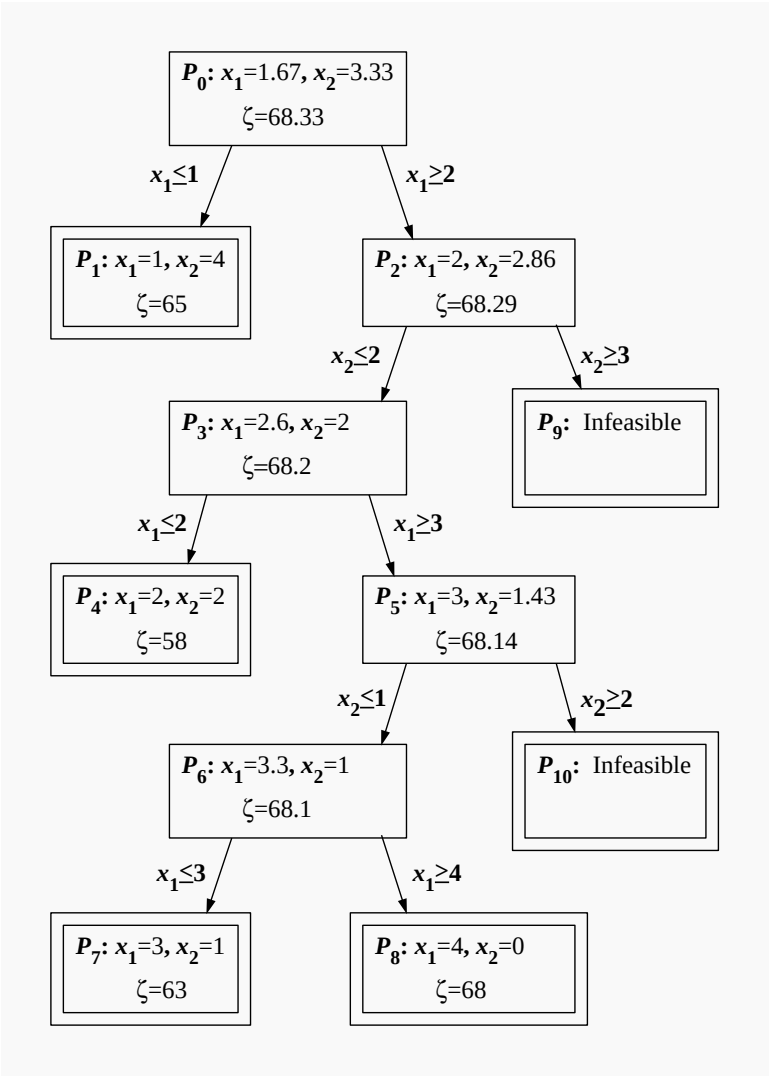


FIGURE 22.15. The complete enumeration tree.

There are three reasons why depth-first search is generally the preferred order in which to fathom the enumeration tree. The first is based on the observation that most integer solutions lie deep in the tree. There are two advantages to finding integer feasible solutions early. The first is simply the fact that it is better to have a feasible solution than nothing in case one wishes to abort the solution process early. But more importantly, identifying an feasible integer solution can result in subsequent nodes of the enumeration tree being made into leaves simply because the optimal objective function associated with that node is lower than the best-so-far integer solution. Making such nodes into leaves is called *pruning* the tree and can account for tremendous gains in efficiency.

A second reason to favor depth-first search is the simple fact that it is very easy to code the algorithm as a recursively defined function. This may seem trite, but one shouldn't underestimate the value of code simplicity when implementing algorithms that are otherwise quite sophisticated, such as the one we are currently describing.

The third reason to favor depth-first search is perhaps the most important. It is based on the observation that as one moves deeper in the enumeration tree, each subsequent linear programming problem is obtained from the preceding one by simply adding (or refining) an upper/lower bound on one specific variable. To see why this is an advantage, consider for example problem P_2, which is a refinement of P_0. The optimal dictionary for problem P_0 is recorded as

$$\begin{aligned} \zeta &= \tfrac{205}{3} - \tfrac{5}{3}w_1 - \tfrac{1}{3}w_2 \\ \hline x_1 &= \tfrac{5}{3} - \tfrac{1}{3}w_1 + \tfrac{7}{3}w_2 \\ x_2 &= \tfrac{10}{3} + \tfrac{1}{3}w_1 - \tfrac{10}{3}w_2. \end{aligned}$$

Problem P_2 is obtained from P_0 by adding the constraint that $x_1 \geq 2$. Introducing a variable, g_1, to stand for the difference between x_1 and this lower bound and using the dictionary above to write x_1 in terms of the nonbasic variables, we get

$$g_1 = x_1 - 2 = -\frac{1}{3} - \frac{1}{3}w_1 + \frac{7}{3}w_2.$$

Therefore, we can use the following dictionary as a starting point for the solution of P_2:

$$\begin{aligned} \zeta &= \tfrac{205}{3} - \tfrac{5}{3}w_1 - \tfrac{1}{3}w_2 \\ \hline x_1 &= \tfrac{5}{3} - \tfrac{1}{3}w_1 + \tfrac{7}{3}w_2 \\ x_2 &= \tfrac{10}{3} + \tfrac{1}{3}w_1 - \tfrac{10}{3}w_2 \\ g_1 &= -\tfrac{1}{3} - \tfrac{1}{3}w_1 + \tfrac{7}{3}w_2. \end{aligned}$$

This dictionary is dual feasible but primal infeasible. Therefore, the dual simplex method is likely to find a new optimal solution in very few iterations. According to the dual simplex method, variable g_1 is the leaving variable and w_2 is the corresponding

entering variable. Making the pivot, we get the following dictionary:

$$\begin{aligned}
\zeta &= \tfrac{478}{7} - \tfrac{12}{7}w_1 - \tfrac{1}{7}g_1 \\
\hline
x_1 &= \quad 2 \qquad\quad + \quad g_1 \\
x_2 &= \tfrac{20}{7} - \tfrac{1}{7}w_1 - \tfrac{10}{7}g_1 \\
w_2 &= \tfrac{1}{7} + \tfrac{1}{7}w_1 + \tfrac{3}{7}g_1.
\end{aligned}$$

This dictionary is optimal for P_2. In general, the dual simplex method will take more than one iteration to reoptimize, but nonetheless, one does expect it to get to a new optimal solution quickly.

We end this chapter by remarking that many real problems have the property that some variables must be integers but others can be real valued. Such problems are called *mixed integer programming problems*. It should be easy to see how to modify the branch-and-bound technique to handle such problems as well.

Exercises

22.1 *Knapsack Problem.* Consider a picnicker who will be carrying a knapsack that holds a maximum amount b of "stuff." Suppose that our picnicker must decide what to take and what to leave behind. The jth thing that might be taken occupies a_j units of space in the knapsack and will bring c_j amount of "enjoyment." The knapsack problem then is to maximize enjoyment subject to the constraint that the stuff brought must fit into the knapsack:

$$\begin{aligned}
&\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \\
&\text{subject to} \quad \sum_{j=1}^{n} a_j x_j \le b \\
&\qquad\qquad x_j \in \{0,1\} \qquad j = 1, 2, \ldots, n.
\end{aligned}$$

This apparently simple problem has proved difficult for general-purpose branch-and-bound algorithms. To see why, analyze the special case in which each thing contributes the same amount of enjoyment, i.e., $c_j = c$ for all j, and takes up exactly two units of space, i.e., $a_j = 2$ for all j. Suppose also that the knapsack holds n units of stuff.

(a) What is the optimal solution when n is even? when n is odd?

(b) How many subproblems must the branch-and-bound algorithm consider when n is odd?

22.2 *Vehicle Routing.* Consider the dispatching of delivery vehicles (for example, mail trucks, fuel-oil trucks, newspaper delivery trucks, etc.). Typically, there is a fleet of vehicles that must be routed to deliver goods from a depot to a given set of n drop-points. Given a set of feasible delivery routes and

the cost associated with each one, explain how to formulate the problem of minimizing the total delivery cost as a set-partitioning problem.

22.3 Explain how to modify the integer programming reformulation of continuous piecewise linear functions so that it covers piecewise linear functions having discontinuities at the junctions of the linear segments. Can fixed costs be handled with this approach?

Notes

Standard references for integer programming include the classic text by Garfinkel & Nemhauser (1972) and the more recent text by Nemhauser & Wolsey (1988).

CHAPTER 23

Quadratic Programming

In Chapter 22, we studied a generalization of the linear programming problem in which variables were constrained to take on integer values. In this chapter, we consider a generalization of a different kind. Namely, we shall study the class of problems that would be linear programs except that the objective function is permitted to include terms involving products of pairs of variables. Such terms are called *quadratic terms*, and the problems we shall study are called *quadratic programming* problems.

We have two reasons for being interested in quadratic programming problems. First, on the practical side, an important model for portfolio optimization in finance requires one to be able to solve quadratic programming problems. We shall discuss this model in detail. The second reason for our interest is that the quadratic programming problem forms a bridge to the much broader subject of convex programming that we shall take up in Chapter 24.

1. The Markowitz Model

Harry Markowitz received the 1990 Nobel Prize in Economics for his portfolio optimization model in which the tradeoff between risk and reward is explicitly treated. We shall briefly describe this model in its simplest form. Given a collection of potential investments (indexed, say, from 1 to n), let R_j denote the return in the next time period on investment j, $j = 1, \ldots, n$. In general, R_j is a random variable, although some investments may be essentially deterministic.

A *portfolio* is determined by specifying what fraction of one's assets to put into each investment. That is, a portfolio is a collection of nonnegative numbers x_j, $j = 1, \ldots, n$, that sum to one. The return (on each dollar) one would obtain using a given portfolio is given by

$$R = \sum_j x_j R_j.$$

The *reward* associated with such a portfolio is defined as the expected return:

$$\mathbb{E}R = \sum_j x_j \mathbb{E}R_j.$$

If reward were the only issue, then the problem would be trivial: simply put everything in the investment with the highest expected return. But unfortunately, investments

with high reward typically also carry a high level of risk. That is, even though they are expected to do very well in the long run, they also tend to be erratic in the short term. Markowitz defined the *risk* associated with an investment (or, for that matter, a portfolio of investments) to be the variance of the return:

$$\begin{aligned}\mathrm{Var}(R) &= \mathbb{E}(R - \mathbb{E}R)^2 \\ &= \mathbb{E}\left(\sum_j x_j(R_j - \mathbb{E}R_j)\right)^2 \\ &= \mathbb{E}\left(\sum_j x_j \tilde{R}_j\right)^2,\end{aligned}$$

where $\tilde{R}_j = R_j - \mathbb{E}R_j$. One would like to maximize the reward while at the same time not incurring excessive risk. In the Markowitz model, one forms a linear combination of the mean and the variance (parametrized here by μ) and minimizes that:

$$\begin{aligned}&\text{minimize} && -\sum_j x_j \mathbb{E}R_j + \mu\mathbb{E}\left(\sum_j x_j \tilde{R}_j\right)^2 \\ &\text{subject to} && \sum_j x_j = 1 \\ & && x_j \geq 0 \qquad j = 1, 2, \ldots, n.\end{aligned} \tag{23.1}$$

Here, μ is a positive parameter that represents the importance of risk relative to reward: high values of μ tend to minimize risk at the expense of reward, whereas low values put more weight on reward.

It is important to note that by diversifying (that is, not putting everything into one investment), it might be possible to reduce the risk without reducing the reward. To see how this can happen, consider a hypothetical situation involving two investments A and B. Each year, investment A either goes up 20% or goes down 10%, but unfortunately, the ups and downs are unpredictable (that is, each year is independent of the previous years and is an up year with probability $1/2$). Investment B is also highly volatile. In fact, in any year in which A goes up 20%, investment B goes down 10%, and in the years in which A goes down 10%, B goes up 20%. Clearly, by putting half of our portfolio into A and half into B, we can create a portfolio that goes up 5% every year without fail. The act of identifying investments that are negatively correlated with each other (such as A and B) and dividing the portfolio among these investments is called *hedging*. Unfortunately, it is fairly difficult to find pairs of investments with strong negative correlations.

Solving problem (23.1) requires knowledge of the joint distribution of the R_j's. However, this distribution is not known theoretically but instead must be estimated by looking at historical data. For example, Table 23.1 shows annual returns from

Year	US 3-Month T-Bills	US Gov. Long Bonds	S&P 500	Wilshire 5000	NASDAQ Composite	Lehman Bros. Corp. Bonds	EAFE	Gold
1973	1.075	0.942	0.852	0.815	0.698	1.023	0.851	1.677
1974	1.084	1.020	0.735	0.716	0.662	1.002	0.768	1.722
1975	1.061	1.056	1.371	1.385	1.318	1.123	1.354	0.760
1976	1.052	1.175	1.236	1.266	1.280	1.156	1.025	0.960
1977	1.055	1.002	0.926	0.974	1.093	1.030	1.181	1.200
1978	1.077	0.982	1.064	1.093	1.146	1.012	1.326	1.295
1979	1.109	0.978	1.184	1.256	1.307	1.023	1.048	2.212
1980	1.127	0.947	1.323	1.337	1.367	1.031	1.226	1.296
1981	1.156	1.003	0.949	0.963	0.990	1.073	0.977	0.688
1982	1.117	1.465	1.215	1.187	1.213	1.311	0.981	1.084
1983	1.092	0.985	1.224	1.235	1.217	1.080	1.237	0.872
1984	1.103	1.159	1.061	1.030	0.903	1.150	1.074	0.825
1985	1.080	1.366	1.316	1.326	1.333	1.213	1.562	1.006
1986	1.063	1.309	1.186	1.161	1.086	1.156	1.694	1.216
1987	1.061	0.925	1.052	1.023	0.959	1.023	1.246	1.244
1988	1.071	1.086	1.165	1.179	1.165	1.076	1.283	0.861
1989	1.087	1.212	1.316	1.292	1.204	1.142	1.105	0.977
1990	1.080	1.054	0.968	0.938	0.830	1.083	0.766	0.922
1991	1.057	1.193	1.304	1.342	1.594	1.161	1.121	0.958
1992	1.036	1.079	1.076	1.090	1.174	1.076	0.878	0.926
1993	1.031	1.217	1.100	1.113	1.162	1.110	1.326	1.146
1994	1.045	0.889	1.012	0.999	0.968	0.965	1.078	0.990

TABLE 23.1. Returns per dollar for each of eight investments over several years. That is, \$1 invested in US 3-Month T-Bills on January 1, 1973, was worth $\$1.075$ on December 31, 1973.

1973 to 1994 for eight different possible investments: U.S. Three-Month T-Bills, U.S. Government Long Bonds, S&P 500, Wilshire 5000 (a collection of small company stocks), NASDAQ Composite, Lehman Brothers Corporate Bonds Index, EAFE (a securities index for Europe, Asia, and the Far East), and Gold. Let $R_j(t)$ denote the return on investment j in year $1972 + t$. One way to estimate the mean $\mathbb{E}R_j$ is simply to take the average of the historical returns:

$$\mathbb{E}R_j = \frac{1}{T}\sum_{t=1}^{T} R_j(t).$$

There are two drawbacks to this simple formula. First, whatever happened in 1973 certainly has less bearing on the future than what happened in 1994. Hence, giving all the past returns equal weight puts too much emphasis on the distant past at the expense

of the recent past. A better estimate is obtained by using a discounted sum:

$$\mathbb{E}R_j = \frac{\sum_{t=1}^{T} p^{T-t} R_j(t)}{\sum_{t=1}^{T} p^{T-t}}.$$

Here, p is a discount factor. Putting $p = 0.9$ gives a weighted average that puts more weight on the most recent years. To see the effect of discounting the past, consider the Gold investment. The unweighted average return is 1.129, whereas the weighted average is 1.053. Most experts in 1995 felt that a 5.3% return represented a more realistic expectation than a 12.9% return. In the results that follow, all expectations are estimated by computing weighted averages using $p = 0.9$.

The second issue concerns the estimation of means (not variances). An investment that returns 1.1 one year and 0.9 the next has an (unweighted) average return of 1, that is, no gain or loss. However, one dollar invested will actually be worth $(1.1)(0.9) = 0.99$ at the end of the second year. While a 1% error is fairly small, consider what happens if the return is 2.0 one year and then 0.5 the next. Clearly, the value of one dollar at the end of the two years is $(2.0)(0.5) = 1$, but the average of the two returns is $(2.0 + 0.5)/2 = 1.25$. There is a very significant difference between an investment that is flat and one that yields a 25% return in two years. This is obviously an effect for which a correction is required. We need to average 2.0 and 0.5 in such a way that they cancel out—and this cancellation must work not only for 2.0 and 0.5 but for every positive number and its reciprocal. The trick is to average the logarithm of the returns (and then exponentiate the average). The logarithm has the correct effect of cancelling a return r and its reciprocal:

$$\log r + \log \frac{1}{r} = 0.$$

Hence, we estimate means from Table 23.1 using

$$\mathbb{E}R_j = \exp\left(\frac{\sum_{t=1}^{T} p^{T-t} \log R_j(t)}{\sum_{t=1}^{T} p^{T-t}}\right).$$

This estimate for Gold gives an estimate of its return at 2.9%, which is much more in line with the beliefs of experts (at least in 1995).

Table 23.2 shows the optimal portfolios for several choices of μ. The corresponding optimal values for the mean and standard deviation (which is defined as the square root of the variance) are plotted in Figure 23.1. Letting μ vary continuously generates a curve of optimal solutions. This curve is called the *efficient frontier*. Any portfolio that produces a mean–variance combination that does not lie on the efficient frontier can be improved either by increasing its mean without changing the variance or by decreasing the variance without changing the mean. Hence, one should only invest in portfolios that lie on the efficient frontier.

Of course, the optimal portfolios shown in Table 23.2 were obtained by solving (23.1). The rest of this chapter is devoted to describing an algorithm for solving quadratic programs such as this one.

μ	Gold	US 3-Month T-Bills	Lehman Bros. Corp. Bonds	NASDAQ Composite	S&P 500	EAFE	Mean	Std. Dev.
0.0						1.000	1.122	0.227
0.1					0.603	0.397	1.121	0.147
1.0					0.876	0.124	1.120	0.133
2.0		0.036	0.322		0.549	0.092	1.108	0.102
4.0		0.487	0.189		0.261	0.062	1.089	0.057
8.0		0.713	0.123		0.117	0.047	1.079	0.037
1024.0	0.008	0.933	0.022	0.016		0.022	1.070	0.028

TABLE 23.2. Optimal portfolios for several choices of μ.

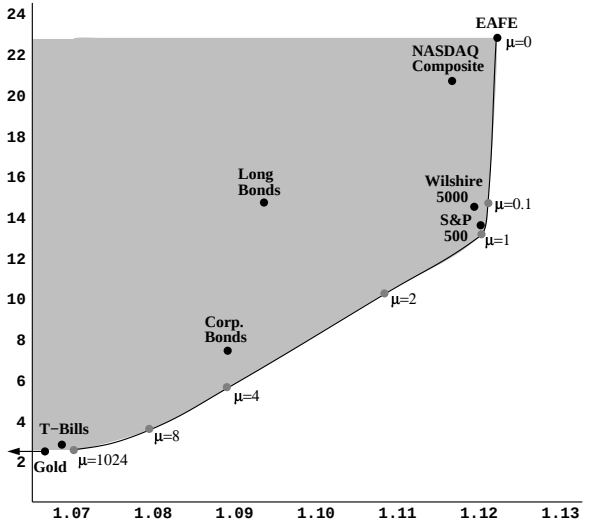


FIGURE 23.1. The efficient frontier.

2. The Dual

We have seen that duality plays a fundamental role in our understanding and derivation of algorithms for linear programming problems. The same is true for quadratic programming. Hence, our first goal is to figure out what the dual of a quadratic programming problem should be.

Quadratic programming problems are usually formulated as minimizations. Therefore, we shall consider problems given in the following form:

$$\begin{aligned} &\text{minimize} && c^T x + \tfrac{1}{2} x^T Q x \\ &\text{subject to} && Ax \geq b \\ & && x \geq 0. \end{aligned} \tag{23.2}$$

Of course, we may (and do) assume that the matrix Q is symmetric (see Exercise 23.2). Note that we have also changed the sense of the inequality constraints from our usual less-than to greater-than. This change is not particularly important—its only purpose is to maintain a certain level of parallelism with past formulations (that is, minimizations have always gone hand-in-hand with greater-than constraints, while maximizations have been associated with less-than constraints).

In Chapter 5, we derived the dual problem by looking for tight bounds on the optimal solution to the primal problem. This approach could be followed here, but it seems less compelling in the context of quadratic programming. A more direct approach stems from the connection between duality and the first-order optimality conditions for the barrier problem that we examined in Chapter 16. Indeed, let us start by writing down the barrier problem associated with (23.2). To this end, we introduce a nonnegative vector w of surplus variables and then subtract a barrier term for each nonnegative variable to get the following barrier problem:

$$\begin{aligned} &\text{minimize} && c^T x + \tfrac{1}{2} x^T Q x - \mu \textstyle\sum_j \log x_j - \mu \sum_i \log w_i \\ &\text{subject to} && Ax - w = b. \end{aligned}$$

Next, we introduce the Lagrangian:

$$\begin{aligned} f(x, w, y) = c^T x + \frac{1}{2} x^T Q x - \mu \sum_j \log x_j - \mu \sum_i \log w_i \\ + y^T (b - Ax + w). \end{aligned}$$

The first-order optimality conditions for the barrier problem are obtained by differentiating the Lagrangian with respect to each of its variables and setting these derivatives to zero. In vector notation, setting to zero the derivative with respect to the x variables gives

$$c + Qx - \mu X^{-1} e - A^T y = 0.$$

Similarly, setting to zero the derivatives with respect to the w and y variables gives

$$\begin{aligned} -\mu W^{-1} e + y &= 0 \\ b - Ax + w &= 0, \end{aligned}$$

respectively. As we did in our study of linear programming problems, we now introduce a new vector z given by

$$z = \mu X^{-1} e.$$

With this definition, the first-order optimality conditions can be summarized as

$$\begin{aligned} A^T y + z - Qx &= c \\ Ax - w &= b \\ XZe &= \mu e \\ YWe &= \mu e. \end{aligned}$$

From the last two conditions, we see that the dual problem involves an n-vector of variables z that are complementary to the primal variables x and an m-vector of variables y that are complementary to the primal slack variables w. Because of these complementarity conditions, we expect that the variables y and z are constrained to be nonnegative in the dual problem. Also, to establish the proper connection between the first-order optimality conditions and the dual problem, we must recognize the first condition as a dual constraint. Hence, the constraints for the dual problem are

$$\begin{aligned} A^T y + z - Qx &= c \\ y, z &\geq 0. \end{aligned}$$

It is interesting to note that the dual constraints involve an n-vector x that seems as if it should belong to the primal problem. This may seem odd, but when understood properly it turns out to be entirely harmless. The correct interpretation is that the variable x appearing in the dual has, in principle, no connection to the variable x appearing in the primal (except that, as we shall soon see, at optimality they will be equal).

The barrier problem has helped us write down the dual constraints, but it does not shed any light on the dual objective function. To see what the dual objective function should be, we look at what it needs to be for the weak duality theorem to hold true. In the weak duality theorem, we assume that we have a primal feasible solution (x, w) and a dual feasible solution (x, y, z). We then follow the obvious chains of equalities:

$$y^T(Ax) = y^T(b + w) = b^T y + y^T w$$

and

$$(A^T y)^T x = (c - z + Qx)^T x = c^T x - z^T x + x^T Qx.$$

Now, since $y^T(Ax) = (A^T y)^T x$, we see that

$$\begin{aligned} 0 \leq y^T w + z^T x &= c^T x + x^T Qx - b^T y \\ &= (c^T x + \frac{1}{2} x^T Qx) - (b^T y - \frac{1}{2} x^T Qx). \end{aligned}$$

From this inequality, we see that the dual objective function is $b^T y - \frac{1}{2} x^T Q x$. Hence, the dual problem can be stated now as

$$\begin{aligned} &\text{maximize } b^T y - \tfrac{1}{2} x^T Q x \\ &\text{subject to } A^T y + z - Qx = c \\ &\qquad\qquad\qquad\qquad y, z \geq 0. \end{aligned}$$

For linear programming, the fundamental connection between the primal and dual problems is summarized in the Complementary Slackness Theorem. In the next section, we shall derive a version of this theorem for quadratic programming.

3. Convexity and Complexity

In linear programming, the dual problem is important because it provides a certificate of optimality as manifest in the Complementary Slackness Theorem. Under certain conditions, the same is true here. Let us start by deriving the analogue of the Complementary Slackness Theorem. The derivation begins with a reiteration of the derivation of the Weak Duality Theorem. Indeed, let (x, w) denote a feasible solution to the primal problem and let $(\bar{x}, y, z)$ denote a feasible solution to the dual problem (we have put a bar on the dual x to distinguish it from the one appearing in the primal). The chain of equalities that form the backbone of the proof of the Weak Duality Theorem are, as always, obtained by writing $y^T Ax$ two ways, namely,

$$y^T (Ax) = (A^T y)^T x,$$

and then producing the obvious substitutions

$$y^T (Ax) = y^T (b + w) = b^T y + y^T w$$

and

$$(A^T y)^T x = (c - z + Q\bar{x})^T x = c^T x - z^T x + \bar{x}^T Q x.$$

Comparing the ends of these two chains and using the fact that both $y^T w$ and $z^T x$ are nonnegative, we see that

$$0 \leq y^T w + z^T x = c^T x + \bar{x}^T Q x - b^T y. \tag{23.3}$$

So far, so good.

Now, what about the Complementary Slackness Theorem? In the present context, we expect this theorem to say roughly the following: given a solution (x^*, w^*) that is feasible for the primal and a solution (x^*, y^*, z^*) that is feasible for the dual, if these solutions make inequality (23.3) into an equality, then the primal solution is optimal for the primal problem and the dual solution is optimal for the dual problem.

Let's try to prove this. Let (x, w) be an arbitrary primal feasible solution. Weak duality applied to (x, w) on the primal side and (x^*, y^*, z^*) on the dual side says that

$$c^T x + {x^*}^T Q x - b^T y^* \geq 0.$$

But for the specific primal feasible solution (x^*, w^*), this inequality is an equality:

$$c^T x^* + {x^*}^T Q x^* - b^T y^* = 0.$$

Combining these, we get

$$c^T x^* + {x^*}^T Q x^* \leq c^T x + {x^*}^T Q x.$$

This is close to what we want, but not quite it. Recall that our aim is to show that the primal objective function evaluated at x^* is no larger than its value at x. That is,

$$c^T x^* + \frac{1}{2}{x^*}^T Q x^* \leq c^T x + \frac{1}{2}x^T Q x.$$

It is easy to get from the one to the other. Starting from the desired left-hand side, we compute as follows:

$$\begin{aligned} c^T x^* + \frac{1}{2}{x^*}^T Q x^* &= c^T x^* + {x^*}^T Q x^* - \frac{1}{2}{x^*}^T Q x^* \\ &\leq c^T x + {x^*}^T Q x - \frac{1}{2}{x^*}^T Q x^* \\ &= c^T x + \frac{1}{2}x^T Q x - \frac{1}{2}x^T Q x + {x^*}^T Q x - \frac{1}{2}{x^*}^T Q x^* \\ &= c^T x + \frac{1}{2}x^T Q x - \frac{1}{2}(x - x^*)^T Q (x - x^*). \end{aligned}$$

The last step in the derivation is to drop the subtracted term on the right-hand side of the last expression. We can do this if the quantity being subtracted is nonnegative. But is it? In general, the answer is no. For example, if Q were the negative of the identity matrix, then the expression $(x - x^*)^T Q (x - x^*)$ would be negative rather than nonnegative.

So it is here that we must impose a restriction on the class of quadratic programming problems that we study. The correct assumption is that Q is positive semidefinite. Recall from Chapter 18 that a matrix Q is *positive semidefinite* if

$$\xi^T Q \xi \geq 0 \qquad \text{for all } \xi \in \mathbb{R}^n.$$

With this assumption, we can finish the chain of inequalities and conclude that

$$c^T x^* + \frac{1}{2}{x^*}^T Q x^* \leq c^T x + \frac{1}{2}x^T Q x.$$

Since x was an arbitrary primal feasible point, it follows that x^* (together with w^*) is optimal for the primal problem. A similar analysis shows that y^* (together with x^* and z^*) is optimal for the dual problem (see Exercise 23.4).

A quadratic programming problem of the form (23.2) in which the matrix Q is positive semidefinite is called a *convex quadratic programming problem*. The discussion given above can be summarized in the following theorem:

THEOREM 23.1. *For convex quadratic programming problems, given a solution* (x^*, w^*) *that is feasible for the primal and a solution* (x^*, y^*, z^*) *that is feasible for the dual, if these solutions make inequality* (23.3) *into an equality, then the primal solution is optimal for the primal problem and the dual solution is optimal for the dual problem.*

To see how bad things are when Q is not positive semidefinite, consider the following example:

$$\begin{aligned} &\text{minimize } \textstyle\sum_j x_j(1-x_j) + \sum_j c_j x_j \\ &\text{subject to } 0 \le x_j \le 1, \qquad j = 1, 2, \ldots, n. \end{aligned} \tag{23.4}$$

We assume that the coefficients, c_j, $j = 1, 2, \ldots, n$, are small. To be precise, we assume that

$$|c_j| < 1, \qquad j = 1, 2, \ldots, n.$$

Let $f(x)$ denote the value of the objective function at point x. Setting the gradient to zero,

$$\nabla f(x) = e - 2x + c = 0,$$

we see that there is one interior critical point. It is given by

$$x = (e + c)/2$$

(the assumption that c is small guarantees that this x lies in the interior of the feasible set: $0 < x < 1$). However, this critical point is a local maximum, since the matrix of second derivatives is $-2I$. The algebraic details are tedious, but if we look at Figure 23.2, it is easy to be convinced that every vertex of the feasible set is a local minimum. While this particular problem is easy to solve explicitly, it does indicate the essential difficulty associated with nonconvex quadratic programming problems—namely, for such problems one may need to check every vertex individually, and there may be an exponential number of such vertices.

The situation for convex quadratic programming problems is much better, since they inherit most of the properties that made linear programs efficiently solvable. Indeed, in the next section, we derive an interior-point method for quadratic programming problems.

4. Solution Via Interior-Point Methods

In this section, we derive an interior-point method for quadratic programming problems. We start from the first-order optimality conditions, which we saw in the last section are given by

$$\begin{aligned} A^T y + z - Qx &= c \\ Ax - w &= b \\ XZe &= \mu e \\ YWe &= \mu e. \end{aligned}$$

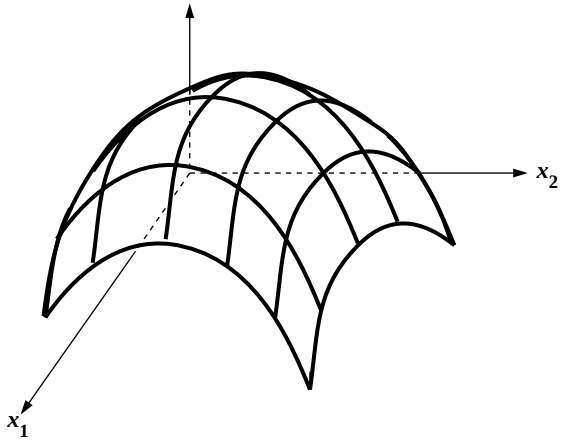


FIGURE 23.2. The objective function for (23.4) in the case where $n = 2$.

Following the derivation given in Chapter 17, we replace (x, w, y, z) with $(x+\Delta x, w+\Delta w, y+\Delta y, z+\Delta z)$ to get the following nonlinear system in $(\Delta x, \Delta w, \Delta y, \Delta z)$:

$$\begin{aligned} A^T\Delta y + \Delta z - Q\Delta x &= c - A^T y - z + Qx =: \sigma \\ A\Delta x - \Delta w &= b - Ax + w \qquad =: \rho \\ Z\Delta x + X\Delta z + \Delta X\Delta Z e &= \mu e - XZe \\ W\Delta y + Y\Delta w + \Delta Y\Delta W e &= \mu e - YWe. \end{aligned}$$

Next, we drop the nonlinear terms to get the following linear system for the step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$:

$$\begin{aligned} A^T\Delta y + \Delta z - Q\Delta x &= \sigma \\ A\Delta x - \Delta w &= \rho \\ Z\Delta x + X\Delta z &= \mu e - XZe \\ W\Delta y + Y\Delta w &= \mu e - YWe. \end{aligned}$$

Following the reductions of Chapter 18, we use the last two equations to solve for Δz and Δw to get

$$\begin{aligned} \Delta z &= X^{-1}(\mu e - XZe - Z\Delta x) \\ \Delta w &= Y^{-1}(\mu e - YWe - W\Delta y). \end{aligned}$$

We then use these expressions to eliminate Δz and Δw from the remaining two equations in the system. After elimination, we arrive at the following *reduced KKT system*:

$$A^T\Delta y - (X^{-1}Z + Q)\Delta x = \sigma - \mu X^{-1}e + z \tag{23.5}$$

$$A\Delta x + Y^{-1}W\Delta y = \rho + \mu Y^{-1}e - w. \tag{23.6}$$

Substituting in the definitions of ρ and σ and writing the system in matrix notation, we get

$$\begin{bmatrix} -(X^{-1}Z + Q) & A^T \\ A & Y^{-1}W \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - A^T y - \mu X^{-1}e + Qx \\ b - Ax + \mu Y^{-1}e \end{bmatrix}.$$

A summary of the algorithm is shown in Figure 23.3. It should be clear that the quadratic term in the objective function plays a fairly small role. In fact, the convergence analysis given in Chapter 17 can be easily adapted to yield analogous results for quadratic programming problems (see Exercise 23.6).

5. Practical Considerations

For practical implementations of interior-point algorithms, we saw in Chapter 18 that the difficulties created by dense rows/columns suggest that we solve the reduced KKT system using an equation solver that can handle symmetric indefinite systems (such as those described in Chapter 19). Quadratic programming problems give us even more reason to prefer the reduced KKT system. To see why, let us reduce the system further to get a feel for the normal equations for quadratic programming.

If we use (23.5) to solve for Δx and then eliminate it from (23.6), we get

$$\Delta x = -(X^{-1}Z + Q)^{-1}\left(c - A^T y + Qx - \mu X^{-1}e - A^T\Delta y\right)$$

and the associated system of normal equations (in primal form):

$$\begin{aligned}\left(A(X^{-1}Z + Q)^{-1}A^T + Y^{-1}W\right)\Delta y = {} & b - Ax + \mu Y^{-1}e \\ & + A(X^{-1}Z + Q)^{-1}\left(c - A^T y + Qx - \mu X^{-1}e\right).\end{aligned}$$

As we saw in Chapter 18, the most significant disadvantage of the normal equations is that they could involve a dense matrix even when the original constraint matrix is sparse. For quadratic programming, this disadvantage is even more pronounced. Now the matrix of normal equations has the nonzero pattern of $A(D + Q)^{-1}A^T$, where D is a diagonal matrix. If Q is a diagonal matrix, then this matrix appearing between A and A^T is diagonal, and the system has the same structure as we saw for linear programming. But if Q is not a diagonal matrix, then all hope for any sparsity in $A(D + Q)^{-1}A^T$ is lost.

Fortunately, however, the dual form of the normal equations is likely to retain some sparsity. Indeed, to derive the dual form, we use (23.6) to solve for Δy and then eliminate it from (23.5). The result is

$$\Delta y = YW^{-1}\left(b - Ax + \mu Y^{-1}e - A\Delta x\right)$$

initialize $(x, w, y, z) > 0$

while (not optimal) {

$\rho = b - Ax + w$

$\sigma = c - A^T y - z + Qx$

$\gamma = z^T x + y^T w$

$\mu = \delta \dfrac{\gamma}{n+m}$

solve:

$$\begin{bmatrix} -(X^{-1}Z + Q) & A^T \\ A & Y^{-1}W \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - A^T y - \mu X^{-1} e + Qx \\ b - Ax + \mu Y^{-1} e \end{bmatrix}.$$

$\Delta z = X^{-1}(\mu e - XZe - Z\Delta x)$

$\Delta w = Y^{-1}(\mu e - YWe - W\Delta y)$

$\theta = r \left(\max_{ij} \left\{ -\dfrac{\Delta x_j}{x_j}, -\dfrac{\Delta w_i}{w_i}, -\dfrac{\Delta y_i}{y_i}, -\dfrac{\Delta z_j}{z_j} \right\} \right)^{-1} \wedge 1$

$x \leftarrow x + \theta \Delta x, \qquad w \leftarrow w + \theta \Delta w$

$y \leftarrow y + \theta \Delta y, \qquad z \leftarrow z + \theta \Delta z$

}

FIGURE 23.3. The path-following method for quadratic programming problems.

and

$$-\left(X^{-1}Z + Q + A^T Y W^{-1} A\right) \Delta x = c - A^T y + Qx - \mu X^{-1} e - A^T Y W^{-1} \left(b - Ax + \mu Y^{-1} e\right).$$

Now the matrix has a nonzero pattern of $A^T A + Q$. This pattern is much more likely to be sparse than the pattern we had above.

As mentioned earlier, there is significantly less risk of fill-in if Q is diagonal. A quadratic programming problem for which Q is diagonal is called a *separable quadratic programming problem*. It turns out that every nonseparable quadratic programming problem can be replaced by an equivalent separable version, and sometimes this

replacement results in a problem that can be solved dramatically faster than the original nonseparable problem. The trick reveals itself when we remind ourselves that the problems we are studying are convex quadratic programs, and so we ask the question: how do we know that the matrix Q is positive semidefinite? Or, more to the point, how does the creator of the model know that Q is positive semidefinite? There are many equivalent characterizations of positive semidefiniteness, but the one that is easiest to check is the one that says that Q is positive semidefinite if and only if it can be factored as follows:

$$Q = F^T D F.$$

Here F is a $k \times n$ matrix and D is a $k \times k$ diagonal matrix having all nonnegative diagonal entries. In fact, the model creator often started with F and D and then formed Q by multiplying. In these cases, the matrix F will generally be less dense than Q. And if k is substantially less than n, then the following substitution is almost guaranteed to dramatically improve the solution time. Introduce new variables y by setting

$$y = Fx.$$

With this definition, the nonseparable quadratic programming problem (23.2) can be replaced by the following equivalent separable one:

$$\begin{aligned} \text{minimize} \quad & c^T x + \tfrac{1}{2} y^T D y \\ \text{subject to} \quad & Ax \geq b \\ & Fx - y = 0 \\ & x \geq 0. \end{aligned}$$

The cost of separation is the addition of k new constraints. As we said before, if k is small and/or F is sparse, then we can expect this formulation to be solved more efficiently.

To illustrate this trick, let us return to the Markowitz model. Recall that the quadratic terms in this model come from the variance of the portfolio's return, which is given by

$$\begin{aligned} \text{Var}(R) &= \mathbb{E}(\sum_j x_j \tilde{R}_j)^2 \\ &= \sum_{t=1}^{T} p(t) \left(\sum_j x_j \tilde{R}_j(t) \right)^2 . \end{aligned}$$

Here,

$$p(t) = \frac{p^{T-t}}{\sum_{s=1}^{T} p^{T-s}}$$

for $t = 1, 2, \ldots, T$, and

$$\tilde{R}_j(t) = R_j(t) - \sum_{t=1}^{T} p(t) R_j(t).$$

If we introduce the variables,

$$y(t) = \sum_j x_j \tilde{R}_j(t), \qquad t = 1, 2, \ldots, T,$$

then we get the following separable version of the Markowitz model:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_j x_j \mathbb{E} R_j - \mu \sum_{t=1}^{T} p(t) y(t)^2 & \\ \text{subject to} & \displaystyle\sum_j x_j = 1 & \\ & y(t) = \sum_j x_j \tilde{R}_j(t), & t = 1, 2, \ldots, T, \\ & x_j \geq 0 & j = 1, 2, \ldots, n. \end{array}$$

Using specific data involving 500 possible investments and 20 historical time periods, the separable version solves 60 times faster than the nonseparable version using a QP-solver called LOQO.

Exercises

23.1 Show that the gradient of the function

$$f(x) = \frac{1}{2} x^T Q x$$

is given by

$$\nabla f(x) = Qx.$$

23.2 Suppose that Q is an $n \times n$ matrix that is not necessarily symmetric. Let $\tilde{Q} = \frac{1}{2}(Q + Q^T)$. Show that

(a) $x^T Q x = x^T \tilde{Q} x$, for every $x \in \mathbb{R}^n$, and

(b) $\tilde{Q}$ is symmetric.

23.3 *Penalty Methods.*

(a) Consider the following problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^T Q x \\ \text{subject to} & Ax = b, \end{array}$$

where Q is symmetric, positive semidefinite, and invertible (these last two conditions are equivalent to saying that Q is positive definite). By solving the first-order optimality conditions, give an explicit formula for the solution to this problem.

(b) Each equality constraint in the above problem can be replaced by a *penalty term* added to the objective function. Penalty terms should be small when the associated constraint is satisfied and become rapidly larger as it becomes more and more violated. One choice of penalty function is the quadratic function. The *quadratic penalty problem* is defined as follows:

$$\text{minimize } \frac{1}{2}x^T Qx + \frac{\lambda}{2}(b - Ax)^T(b - Ax),$$

where λ is a large real-valued parameter. Derive an explicit formula for the solution to this problem.

(c) Show that, in the limit as λ tends to infinity, the solution to the quadratic penalty problem converges to the solution to the original problem.

23.4 Consider a convex quadratic programming problem. Suppose that (x^*, w^*) is a feasible solution for the primal and that (x^*, y^*, z^*) is a feasible solution for the dual. Suppose further that these solutions make inequality (23.3) into an equality. Show that the dual solution is optimal for the dual problem.

23.5 A real-valued function f defined on $\mathbb{R}^n$ is called *convex* if, for every $x, y \in \mathbb{R}^n$, and for every $0 < t < 1$,

$$f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y).$$

Show that the function

$$f(x) = c^T x + \frac{1}{2}x^T Qx, \qquad x \in \mathbb{R}^n,$$

is convex if Q is positive semidefinite.

23.6 Extend the convergence analysis given in Chapter 17 so that it applies to convex quadratic programming problems, and identify in particular any steps that depend on Q being positive semidefinite.

23.7 Consider the quadratic programming problem given in the following form:

$$\begin{aligned} &\text{minimize } \; c^T x + \tfrac{1}{2}x^T Qx \\ &\text{subject to } \; Ax \geq b, \end{aligned}$$

(i.e., without assuming nonnegativity of the x vector). Show that the formulas for the step directions Δx and Δy are given by the following reduced KKT system:

$$\begin{bmatrix} -Q & A^T \\ A & WY^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - A^T y + Qx \\ b - Ax + \mu Y^{-1} e \end{bmatrix}. \tag{23.7}$$

Notes

The portfolio optimization model presented in Section 23.1 was first introduced by Markowitz (1959). He received the 1990 Nobel Prize in Economics for this work.

Quadratic programming is the simplest class of problems from the subject called nonlinear programming. Two excellent recent texts that cover nonlinear programming are those by Bertsekas (1995) and Nash & Sofer (1996). The first paper that extended the path-following method to quadratic programming was Monteiro & Adler (1989). The presentation given here follows Vanderbei (1999).

CHAPTER 24

Convex Programming

In the last chapter, we saw that small modifications to the primal–dual interior-point algorithm allow it to be applied to quadratic programming problems as long as the quadratic objective function is convex. In this chapter, we shall go further and allow the objective function to be a general (smooth) convex function. In addition, we shall allow the feasible region to be any convex set given by a finite collection of convex inequalities.

1. Differentiable Functions and Taylor Approximations

In this chapter, all nonlinear functions will be assumed to be twice differentiable, and the second derivatives will be assumed continuous. We begin by reiterating a few definitions and results that were briefly touched on in Chapter 16. First of all, given a real-valued function f defined on a domain in $\mathbb{R}^n$, the vector

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}$$

is called the *gradient* of f at x. The matrix

$$Hf(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \cdots \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{bmatrix}$$

is called the *Hessian* of f at x. In dimensions greater than one, the gradient and the Hessian are the analogues of the first and second derivatives of a function in one dimension. In particular, they appear in the three-term Taylor series expansion of f about the point x:

$$f(x + \Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T Hf(x) \Delta x + r_x(\Delta x).$$

The last term is called the remainder term. The value of this expansion lies in the fact that this remainder is small when Δx is small. To be precise, the remainder has the following property:

$$\lim_{\Delta x \to 0} \frac{r_x(\Delta x)}{\|\Delta x\|^2} = 0.$$

This result follows immediately from the one-dimensional three-term Taylor series expansion applied to $g(t) = f(x + t\Delta x)$ and the chain rule (see Exercise 24.8).

2. Convex and Concave Functions

There are several equivalent definitions of convexity of a function. The definition that is most expedient for our purposes is the multidimensional generalization of the statement that a function is convex if its second derivative is nonnegative. Hence, we say that a real-valued function defined on a domain in $\mathbb{R}^n$ is *convex* if its Hessian is positive semidefinite everywhere in its domain. A function is called *concave* if its negation is convex.

3. Problem Formulation

We shall study convex optimization problems posed in the following form:

$$\begin{aligned} &\text{minimize} && c(x) \\ &\text{subject to} && a_i(x) \geq b_i, \qquad i = 1, 2, \ldots, m. \end{aligned}$$

Here, the real-valued function $c(\cdot)$ is assumed to be convex, and the m real-valued functions $a_i(\cdot)$ are assumed to be concave. This formulation is the natural extension of the convex quadratic programming problem studied in the previous chapter, except that we have omitted the nonnegativity constraints on the variables. This omission is only a matter of convenience since, if a given problem involves nonnegative variables, the assertion of their nonnegativity can be incorporated as part of the m nonlinear inequality constraints. Also note that once we allow for general concave inequality constraints, we can take the right-hand sides to be zero by simply incorporating appropriate shifts into the nonlinear constraint functions. Hence, many texts on convex optimization prefer to formulate the constraints in the form $a_i(x) \geq 0$. We have left the constants b_i on the right-hand side for later comparisons with the quadratic programming problem of the previous chapter. Finally, note that many convex and concave functions become infinite in places and therefore have a natural domain that is a strict subset of $\mathbb{R}^n$. This issue is important to address when solving practical problems, but since this chapter is just an introduction to convex optimization, we shall assume that all functions are finite on all of $\mathbb{R}^n$.

At times it will be convenient to use vector notation to consolidate the m constraints into a single inequality. Hence, we sometimes express the problem as

$$\begin{aligned} &\text{minimize} && c(x) \\ &\text{subject to} && A(x) \geq b, \end{aligned}$$

where $A(\cdot)$ is a function from $\mathbb{R}^n$ into $\mathbb{R}^m$ and b is a vector in $\mathbb{R}^m$. As usual, we let w denote the slack variables that convert the inequality constraints to equalities:

$$\begin{aligned} &\text{minimize} && c(x) \\ &\text{subject to} && A(x) - w = b \\ & && w \geq 0. \end{aligned}$$

4. Solution Via Interior-Point Methods

In this section, we derive an interior-point method for convex programming problems. We start by introducing the associated barrier problem:

$$\begin{aligned} &\text{minimize} && c(x) - \mu \textstyle\sum_i \log w_i \\ &\text{subject to} && a_i(x) - w_i = b_i, \qquad i = 1, 2, \ldots, m. \end{aligned}$$

The Lagrangian for this problem is given by

$$L(x, w, y) = c(x) - \mu \sum_i \log w_i + \sum_i y_i (b_i - a_i(x) + w_i).$$

Equating to zero the derivative of L with respect to each of its variables, we get the following set of first-order optimality conditions:

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= \frac{\partial c}{\partial x_j}(x) - \sum_i y_i \frac{\partial a_i}{\partial x_j}(x) &&= 0, \qquad j = 1, 2, \ldots, n, \\ \frac{\partial L}{\partial w_i} &= -\frac{\mu}{w_i} + y_i &&= 0, \qquad i = 1, 2, \ldots, m, \\ \frac{\partial L}{\partial y_i} &= b_i - a_i(x) + w_i &&= 0, \qquad i = 1, 2, \ldots, m. \end{aligned}$$

The next step is to multiply the ith equation in the middle set by w_i and then replace x with $x + \Delta x$, y by $y + \Delta y$, and w by $w + \Delta w$ to get the following system:

$$\begin{aligned} \frac{\partial c}{\partial x_j}(x + \Delta x) - \sum_i (y_i + \Delta y_i) \frac{\partial a_i}{\partial x_j}(x + \Delta x) &= 0, \qquad j = 1, 2, \ldots, n, \\ -\mu + (w_i + \Delta w_i)(y_i + \Delta y_i) &= 0, \qquad i = 1, 2, \ldots, m, \\ b_i - a_i(x + \Delta x) + w_i + \Delta w_i &= 0, \qquad i = 1, 2, \ldots, m. \end{aligned}$$

Now we view this set of equations as a nonlinear system in the "delta" variables and linearize it by replacing each nonlinear function with its two-term Taylor series approximation. For example, $\partial c/\partial x_j(x+\Delta x)$ gets replaced with

$$\frac{\partial c}{\partial x_j}(x+\Delta x) \approx \frac{\partial c}{\partial x_j}(x) + \sum_k \frac{\partial^2 c}{\partial x_j \partial x_k}(x)\Delta x_k.$$

Similarly, $\partial a_i/\partial x_j(x+\Delta x)$ gets replaced with

$$\frac{\partial a_i}{\partial x_j}(x+\Delta x) \approx \frac{\partial a_i}{\partial x_j}(x) + \sum_k \frac{\partial^2 a_i}{\partial x_j \partial x_k}(x)\Delta x_k.$$

Writing the resulting linear system with the delta-variable terms on the left and everything else on the right, we get

$$\begin{aligned}
\sum_k \left(-\frac{\partial^2 c}{\partial x_j \partial x_k} + \sum_i y_i \frac{\partial^2 a_i}{\partial x_j \partial x_k} \right) \Delta x_k + \sum_i \frac{\partial a_i}{\partial x_j}\Delta y_i &= \frac{\partial c}{\partial x_j} - \sum_i y_i \frac{\partial a_i}{\partial x_j} \\
y_i \Delta w_i + w_i \Delta y_i &= \mu - w_i y_i \\
\sum_k \frac{\partial a_i}{\partial x_k}\Delta x_k - \Delta w_i &= b_i - a_i + w_i.
\end{aligned}$$

(Note that we have omitted the indication that the functions c, a_i, and their derivatives are to be evaluated at x.)

As usual, the next step is to solve the middle set of equations for the Δw_i's and then to eliminate them from the system. The reduced system then becomes

$$\begin{aligned}
\sum_k \left(-\frac{\partial^2 c}{\partial x_j \partial x_k} + \sum_i y_i \frac{\partial^2 a_i}{\partial x_j \partial x_k} \right) \Delta x_k + \sum_i \frac{\partial a_i}{\partial x_j}\Delta y_i &= \frac{\partial c}{\partial x_j} - \sum_i y_i \frac{\partial a_i}{\partial x_j} \\
\sum_k \frac{\partial a_i}{\partial x_k}\Delta x_k + \frac{w_i}{y_i}\Delta y_i &= b_i - a_i + \frac{\mu}{y_i},
\end{aligned}$$

and the equations for the Δw_i's are

$$\Delta w_i = -\frac{w_i}{y_i}\Delta y_i + \frac{\mu}{y_i} - w_i, \qquad i = 1, 2, \ldots, m.$$

At this point it is convenient to put the equations into matrix form. If we generalize our familiar gradient notation by letting $\nabla A(x)$ denote the $m \times n$ matrix whose (i,j)th entry is $\partial a_i/\partial x_j(x)$, then we can write the above system succinctly as follows:

(24.1)
$$\begin{bmatrix} -Hc(x) + \sum_i y_i Ha_i(x) & \nabla A(x)^T \\ \nabla A(x) & WY^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \nabla c(x) - \nabla A(x)^T y \\ b - A(x) + \mu Y^{-1} e \end{bmatrix}.$$

Now that we have step directions, the algorithm is easy to describe—just compute step lengths that preserve strict positivity of the w_i's and the y_i's, step to a new point, and iterate.

5. Successive Quadratic Approximations

It is instructive to notice the similarity between the system given above and the analogous system for the quadratic programming problem posed in the analogous form (see Exercise 23.7). Indeed, a careful matching of terms reveals that the step directions derived here are exactly those that would be obtained if one were to form a certain quadratic approximation at the beginning of each iteration of the interior-point algorithm. Hence, the interior-point method can be thought of as a *successive quadratic programming algorithm*. In order to write this quadratic approximation neatly, let $\bar{x}$ and $\bar{y}$ denote the current primal and dual variables, respectively. Then the quadratic approximation can be written as

$$\begin{aligned} \text{minimize } & c(\bar{x}) + \nabla c(\bar{x})^T(x-\bar{x}) + \tfrac{1}{2}(x-\bar{x})^T Hc(\bar{x})(x-\bar{x}) \\ & \qquad\qquad - \tfrac{1}{2}(x-\bar{x})^T \left(\textstyle\sum_i y_i Ha_i(\bar{x})\right)(x-\bar{x}) \\ \text{subject to } & A(\bar{x}) + \nabla A(\bar{x})(x-\bar{x}) \geq b. \end{aligned}$$

To verify the equivalence, we first observe that this problem is a quadratic program whose linear objective coefficients are given by

$$\nabla c(\bar{x}) - Hc(\bar{x})\bar{x} + \left(\sum_i y_i Ha_i(\bar{x})\right)\bar{x},$$

whose quadratic objective coefficients are given by

$$Hc(\bar{x}) - \sum_i y_i Ha_i(\bar{x}),$$

and whose right-hand side vector is given by

$$b - A(\bar{x}) + \nabla A(\bar{x})\bar{x}.$$

Substituting these expressions into the appropriate places in (23.7), we get (24.1).

Looking at the quadratic terms in the objective of the quadratic programming approximation, we see that the objective is convex, since we assumed at that start that c is convex, each a_i is concave, and the dual variables multiplying the Hessians of the constraint functions are all strictly positive.

6. Merit Functions

It is perhaps a little late to bring this up, but here's a small piece of advice: *always test your knowledge on the simplest possible example.* With that in mind, consider the following trivial convex optimization problem:

$$\text{minimize } \sqrt{1+x^2}.$$

This problem has no constraints. Looking at the graph of the objective function, which looks like a smoothed out version of $|x|$, we see that the optimal solution is $x^* = 0$.

What could be easier! There are no y_i's nor any w_i's and equation (24.1) becomes just

$$-Hc(x)\Delta x = \nabla c(x),$$

where $c(x) = \sqrt{1+x^2}$. Taking the first and second derivatives, we get

$$\nabla c(x) = \frac{x}{\sqrt{1+x^2}} \quad \text{and} \quad Hc(x) = \frac{1}{(1+x^2)^{3/2}}.$$

Substituting these expressions into the equation for Δx and simplifying, we get that

$$\Delta x = -x(1+x^2).$$

Since there are no nonnegative variables that need to be kept positive, we can take unshortened steps. Hence, letting $x^{(k)}$ denote our current point and $x^{(k+1)}$ denote the next point, we have that

$$x^{(k+1)} = x^{(k)} + \Delta x = -(x^{(k)})^3.$$

That is, the algorithm says to start at any point $x^{(0)}$ and then replace this point with the negative of its cube, replace that with the negative of its cube, and so on.

The question is: does this sequence converge to zero? It is easy to see that the answer is yes if $|x^{(0)}| < 1$ but no otherwise. For example, if we start with $x^{(0)} = 1/2$, then the sequence of iterates is

k	$x^{(k)}$
0	0.50000000
1	-0.12500000
2	0.00195313
3	-0.00000001

If, on the other hand, we start at $x^{(0)} = 2$, then we get the following wildly divergent sequence:

k	$x^{(k)}$
0	2
1	-8
2	512
3	-134,217,728

Here is what goes wrong in this example. For problems without constraints, our algorithm has an especially simple description:

> From the current point, use the first three terms of a Taylor series expansion to make a quadratic approximation to the objective function. The next point is the minimum of this quadratic approximation function.

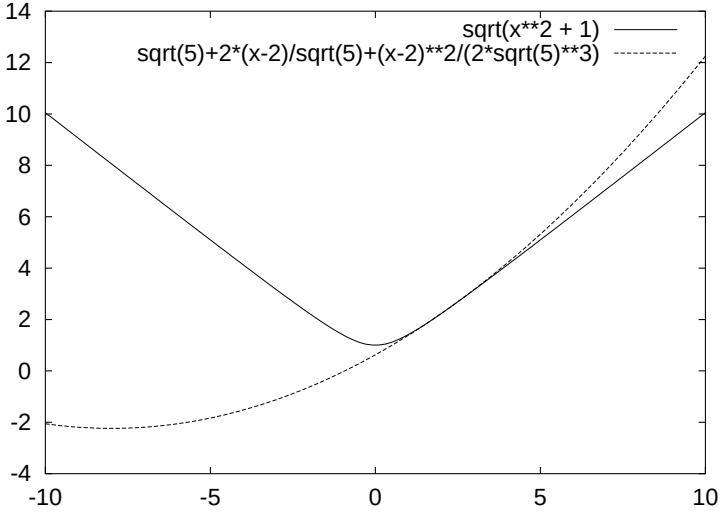


FIGURE 24.1. The function $c(x) = \sqrt{1 + x^2}$ and its quadratic approximation at $x = 2$.

Figure 24.1 shows a graph of the objective function together with the quadratic approximation at $x^{(0)} = 2$. It is easy to see that the next iterate is at -8. Also, the further from zero that one starts, the more the function looks like a straight line and hence the further the minimum will be to the other side.

How do we remedy this nonconvergence? The key insight is the observation that the steps are always in the correct direction (i.e, a descent direction) but they are too long—we need to shorten them. A standard technique for shortening steps in situations like this is to introduce a function called a *merit function* and to shorten steps as needed to ensure that this merit function is always monotonically decreasing. For the example above, and in fact for any unconstrained optimization problem, we can use the objective function itself as the merit function. But, for problems with constraints, one needs to use something a little different from just the objective function. For example, one can use the logarithmic barrier function plus a constant times the square of the Euclidean norm of the infeasibility vector:

$$\Psi(x, w) := c(x) - \sum_i \log(w_i) + \beta \|b - A(x) + w\|^2.$$

Here, β is a positive real number. One can show that for β sufficiently large the step directions are always descent directions for this merit function.

A summary of the algorithm is shown in Figure 24.2.

initialize (x, w, y) so that $(w, y) > 0$
while (not optimal) {
set up QP subproblem:

$$A = \nabla A(x)$$
$$b = b - A(x) + \nabla A(x)x$$
$$c = \nabla c(x) - Hc(x)x + \left(\sum_i y_i Ha_i(x)\right) x$$
$$Q = Hc(x) - \sum_i y_i Ha_i(x)$$

$$\rho = b - Ax + w$$
$$\sigma = c - A^T y + Qx$$
$$\gamma = y^T w$$
$$\mu = \delta \frac{\gamma}{n+m}$$

solve:

$$\begin{bmatrix} -Q & A^T \\ A & Y^{-1}W \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - A^T y + Qx \\ b - Ax - \mu Y^{-1} e \end{bmatrix}$$

$$\Delta w = Y^{-1}(\mu e - YWe - W\Delta y)$$
$$\theta = r \left(\max_{ij} \left\{ -\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i} \right\} \right)^{-1} \wedge 1$$

do {

$$x^{\text{new}} = x + \theta \Delta x,$$
$$w^{\text{new}} = w + \theta \Delta w$$
$$y^{\text{new}} = y + \theta \Delta y$$
$$\theta \leftarrow \theta/2$$

}while ($\Psi(x^{\text{new}}, w^{\text{new}}) \geq \Psi(x, w)$)
}

FIGURE 24.2. The path-following method for convex programming problems.

7. Parting Words

A story is never over, but every book must have an end. So, we stop here mindful of the fact that there are many interesting things left unsaid and topics unexplored. We hope we have motivated the reader to pursue the story further without our assistance—by reading other books and/or research papers and even perhaps making his or her own contributions. Cheers.

Exercises

24.1 *Piecewise Linear Approximation.* Given real numbers $b_1 < b_2 < \cdots < b_k$, let f be a continuous function on $\mathbb{R}$ that is linear on each interval $[b_i, b_{i+1}]$, $i = 0, 1, \ldots, k$ (for convenience we let $b_0 = -\infty$ and $b_{k+1} = \infty$). Such a function is called *piecewise linear* and the numbers b_i are called *breakpoints*. Piecewise linear functions are often used to approximate (continuous) nonlinear functions. The purpose of this exercise is to show how and why.

(a) Every piecewise linear function can be written as a sum of a constant plus a linear term plus a sum of absolute value terms:

$$f(x) = d + a_0 + \sum_{i=1}^{k} a_i |x - b_i|.$$

Let c_i denote the slope of f on the interval $[b_i, b_{i+1}]$. Derive an explicit expression for each of the a_j's (including a_0) in terms of the c_i's.

(b) In terms of the c_i's, give necessary and sufficient conditions for f to be convex.

(c) In terms of the a_j's, give necessary and sufficient conditions for f to be convex.

(d) Assuming that f is convex and is a term in the objective function for a linearly constrained optimization problem, derive an equivalent linear programming formulation involving at most k extra variables and constraints.

(e) Repeat the first four parts of this problem using $\max(x - b_i, 0)$ in place of $|x - b_i|$.

24.2 Let f be the function of 2 real variables defined by

$$f(x, y) = x^2 - 2xy + y^2.$$

Show that f is convex.

24.3 A function f of 2 real variables is called a *monomial* if it has the form

$$f(x, y) = x^m y^n$$

for some nonnegative integers m and n. Which monomials are convex?

24.4 Let ϕ be a convex function of a single real variable. Let f be a function defined on $\mathbb{R}^n$ by the formula

$$f(x) = \phi(a^T x + b),$$

where a is an n-vector and b is a scalar. Show that f is convex.

24.5 Which of the following functions are convex (assume that the domain of the function is all of $\mathbb{R}^n$ unless specified otherwise)?

(a) $4x^2 - 12xy + 9y^2$
(b) $x^2 + 2xy + y^2$
(c) $x^2 y^2$
(d) $x^2 - y^2$
(e) e^{x-y}
(f) $e^{x^2-y^2}$
(g) $\frac{x^2}{y}$ on $\{(x, y) : y > 0\}$

24.6 Given a symmetric square matrix A, the quadratic form $x^T Ax = \sum_{i,j} a_{ij} x_i x_j$ generalizes the notion of the square of a variable. The generalization of the notion of the fourth power of a variable is an expression of the form

$$f(x) = \sum_{i,j,k,l} a_{ijkl} x_i x_j x_k x_l.$$

The four-dimensional array of numbers $A = \{a_{ijkl} : 1 \le i \le n, 1 \le j \le n, 1 \le k \le n, 1 \le l \le n\}$ is called a *4-tensor*. As with quadratic expressions, we may assume that A is symmetric:

$$a_{ijkl} = a_{jkli} = \cdots = a_{lkij}$$

(i.e., given i, j, k, l, all $4! = 24$ premutations must give the same value for the tensor).

(a) Give conditions on the 4-tensor A to guarantee that f is convex.
(b) Suppose that some variables, say y_i's, are related to some other variables, say x_j's, in a linear fashion:

$$y_i = \sum_j f_{ij} x_j.$$

Express $\sum_i y_i^4$ in terms of the x_j's. In particular, give an explicit expression for the 4-tensor and show that it satisfies the conditions derived in part (a).

24.7 Consider the problem

$$\begin{aligned} &\text{minimize } \ ax_1 + x_2 \\ &\text{subject to } \ \sqrt{\epsilon^2 + x_1^2} \le x_2. \end{aligned}$$

where $-1 < a < 1$.

(a) Graph the feasible set: $\left\{(x_1, x_2) : \sqrt{\epsilon^2 + x_1^2} \leq x_2\right\}$. Is the problem convex?

(b) Following the steps in the middle of p. 391 of the text, write down the first-order optimality conditions for the barrier problem associated with barrier parameter $\mu > 0$.

(c) Solve explicitly the first-order optimality conditions. Let $(x_1(\mu), x_2(\mu))$ denote the solution.

(d) Graph the central path, $(x_1(\mu), x_2(\mu))$, as μ varies from 0 to ∞.

24.8 *Multidimensional Taylor's series expansion.* Given a function $g(t)$ defined for real values of t, the three-term Taylor's series expansion with remainder is

$$g(t + \Delta t) = g(t) + g'(t)\Delta t + \frac{1}{2}g''(t)\Delta t^2 + r_t(\Delta t).$$

The remainder term satisfies

$$\lim_{\Delta t \to 0} \frac{r_t(\Delta t)}{\Delta t^2} = 0.$$

Let f be a smooth function defined on $\mathbb{R}^n$. Apply the three-term Taylor's series expansion to $g(t) = f(x + t\Delta x)$ to show that

$$f(x + \Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2}\Delta x^T H f(x) \Delta x + r_x(\Delta x).$$

24.9 Consider the following convex programming problem:

$$\begin{aligned} &\text{minimize} && x_2 \\ &\text{subject to} && x_1^2 + x_2^2 \leq 1. \end{aligned}$$

(a) Find the quadratic subproblem if the current primal solution is $(\bar{x}_1, \bar{x}_2) = (1/2, -2/3)$ and the current dual solution is $\bar{y} = 2$.

(b) Show that for arbitrary current primal and dual solutions, the feasible set for the convex programming problem is contained within the feasible set for the quadratic approximation.

Notes

Interior-point methods for nonlinear programming can be traced back to the pioneering work of Fiacco & McCormick (1968). For more on interior-point methods for convex programming, see Nesterov & Nemirovsky (1993) or den Hertog (1994).

The fact that the step directions are descent directions for the merit function Ψ is proved in Vanderbei & Shanno (1999).

APPENDIX A

Source Listings

The algorithms presented in this book have all been implemented and are publicly available from the author's web site:

`http://www.princeton.edu/~rvdb/LPbook/`

There are two variants of the simplex method: the two-phase method as shown in Figure 6.1 and the self-dual method as shown in Figure 7.1. The simplex codes require software for efficiently solving basis systems. There are two options: the eta-matrix approach described in Section 8.3 and the refactorization approach described in Section 8.5. Each of these "engines" can be used with either simplex method. Hence, there are in total four possible simplex codes that one can experiment with.

There are three variants of interior-point methods: the path-following method as shown in Figure 17.1, the homogeneous self-dual method shown in Figure 21.1 (modified to take long steps), and the long-step homogeneous self-dual method described in Exercise 21.4 of Chapter 21.

The source code that implements the algorithms mentioned above share as much common code as possible. For example, they all share the same input and output routines (the input routine, by itself, is a substantial piece of code). They also share code for the common linear algebra functions. Therefore, the difference between two methods is limited primarily to the specific function that implements the method itself.

The total number of lines of code used to implement all of the algorithms is about 9000. That is too many lines to reproduce all of the code here. But the routines that actually lay out the particular algorithms are fairly short, only about 300 lines each. The relevant part of the self-dual simplex method is shown starting on the next page. It is followed by a listing of the relevant part of the homogeneous self-dual method.

1. The Self-Dual Simplex Method

```
/*************************************************************
*   Main loop                                                *
*************************************************************/

for (iter=0; iter<MAX_ITER; iter++) {

  /*********************************************************
  * STEP 1: Find mu                                        *
  *********************************************************/

  mu = -HUGE_VAL;
  col_in  = -1;
  for (j=0; j<n; j++) {
            if (zbar_N[j] > EPS2) {
                    if ( mu < -z_N[j]/zbar_N[j] ) {
                         mu = -z_N[j]/zbar_N[j];
                         col_in  = j;
                    }
            }
  }
  col_out = -1;
  for (i=0; i<m; i++) {
            if (xbar_B[i] > EPS2) {
                    if ( mu < -x_B[i]/xbar_B[i] ) {
                         mu = -x_B[i]/xbar_B[i];
                         col_out = i;
                         col_in  = -1;
                    }
            }
  }
  if ( mu <= EPS3 ) {       /* OPTIMAL */
      status = 0;
      break;
  }

  if ( col_out >= 0 ) {

    /*******************************************************
    *                          -1  T                       *
    * STEP 2: Compute dz  = -(B  N) e                      *
    *                  N           i                       *
    *         where i = col_out                            *
    *******************************************************/

    vec[0] = -1.0;
    ivec[0] = col_out;
    nvec = 1;

    btsolve( m, vec, ivec, &nvec );

    Nt_times_z( N, at, iat, kat, basicflag, vec, ivec, nvec,
                dz_N, idz_N, &ndz_N );

    /*******************************************************
    * STEP 3: Ratio test to find entering column           *
    *******************************************************/

    col_in = ratio_test( dz_N, idz_N, ndz_N, z_N, zbar_N, mu );

    if (col_in == -1) {     /* INFEASIBLE */
        status = 2;
        break;
    }

    /*******************************************************
    *                       -1                             *
    * STEP 4: Compute dx  = B  N e                         *
```

```
  *                   B         j                              *
  *                                                            *
  ************************************************************/

  j = nonbasics[col_in];
  for (i=0, k=ka[j]; k<ka[j+1]; i++, k++) {
       dx_B[i] =  a[k];
      idx_B[i] = ia[k];
  }
  ndx_B = i;

  bsolve( m, dx_B, idx_B, &ndx_B );

} else {

  /************************************************************
  *                         -1                                 *
  * STEP 2: Compute dx  = B  N e                               *
  *                   B         j                              *
  ************************************************************/

  j = nonbasics[col_in];
  for (i=0, k=ka[j]; k<ka[j+1]; i++, k++) {
       dx_B[i] =  a[k];
      idx_B[i] = ia[k];
  }
  ndx_B = i;

  bsolve( m, dx_B, idx_B, &ndx_B );

  /************************************************************
  * STEP 3: Ratio test to find leaving column                  *
  ************************************************************/

  col_out = ratio_test( dx_B, idx_B, ndx_B, x_B, xbar_B, mu );

  if (col_out == -1) {    /* UNBOUNDED */
      status = 1;
      break;
  }

  /************************************************************
  *                           -1  T                            *
  * STEP 4: Compute dz  = -(B  N) e                            *
  *                   N           i                            *
  *                                                            *
  ************************************************************/

   vec[0] = -1.0;
  ivec[0] = col_out;
  nvec = 1;

  btsolve( m, vec, ivec, &nvec );

  Nt_times_z( N, at, iat, kat, basicflag, vec, ivec, nvec,
              dz_N, idz_N, &ndz_N );

}

/************************************************************
*                                                           *
* STEP 5: Put       t = x /dx                               *
*                       i   i                               *
*                   _   _                                   *
*                   t = x /dx                               *
*                       i   i                               *
*                   s = z /dz                               *
*                       j   j                               *
*                   _   _                                   *
*                   s = z /dz                               *
*                       j   j                               *
```

```
    ************************************************************/

    for (k=0; k<ndx_B; k++) if (idx_B[k] == col_out) break;

    t    =    x_B[col_out]/dx_B[k];
    tbar = xbar_B[col_out]/dx_B[k];

    for (k=0; k<ndz_N; k++) if (idz_N[k] == col_in) break;

    s    =    z_N[col_in]/dz_N[k];
    sbar = zbar_N[col_in]/dz_N[k];

    /************************************************************
    *                                                           *
    * STEP 7: Set z  = z  - s dz     z  = z  - s dz             *
    *             N    N       N      N    N       N           *
    *                                                           *
    *            z  = s             z  = s                      *
    *             i                  i                        *
    *                                                         *
    *            x  = x  - t dx     x  = x  - t dx             *
    *             B    B       B     B    B       B           *
    *                                                         *
    *            x  = t             x  = t                      *
    *             j                  j                        *
    ************************************************************/

    for (k=0; k<ndz_N; k++) {
              j = idz_N[k];
              z_N[j]    -= s   *dz_N[k];
              zbar_N[j] -= sbar*dz_N[k];
    }

    z_N[col_in]    = s;
    zbar_N[col_in] = sbar;

    for (k=0; k<ndx_B; k++) {
              i = idx_B[k];
              x_B[i]    -= t   *dx_B[k];
              xbar_B[i] -= tbar*dx_B[k];
    }

    x_B[col_out]     = t;
    xbar_B[col_out]  = tbar;

    /************************************************************
    * STEP 8: Update basis                                      *
    ************************************************************/

    i =    basics[col_out];
    j = nonbasics[col_in];
    basics[col_out]   = j;
    nonbasics[col_in] = i;
    basicflag[i] = -col_in-1;
    basicflag[j] = col_out;

    /************************************************************
    * STEP 9: Refactor basis and print statistics               *
    ************************************************************/

    from_scratch = refactor( m, ka, ia, a, basics, col_out, v );

    if (from_scratch) {
        primal_obj = sdotprod(c,x_B,basics,m) + f;
        printf("%8d   %14.7e %9.2e \n", iter, primal_obj, mu );
        fflush(stdout);
    }
}
```

2. The Homogeneous Self-Dual Method

```
/*************************************************************
*   Main loop                                                *
*************************************************************/

for (iter=0; iter<MAX_ITER; iter++) {

    /*************************************************************
    * STEP 1: Compute mu and centering parameter delta.
    *************************************************************/

    mu = (dotprod(z,x,n)+dotprod(w,y,m)+phi*psi) / (n+m+1);

    if (iter%2 == 0) {
        delta = 0.0;
    } else {
        delta = 1.0;
    }

    /*************************************************************
    * STEP 1: Compute primal and dual objective function values.
    *************************************************************/

    primal_obj = dotprod(c,x,n);
    dual_obj   = dotprod(b,y,m);

    /*************************************************************
    * STEP 2: Check stopping rule.
    *************************************************************/

    if ( mu < EPS ) {
        if ( phi > EPS ) {
            status = 0;
            break;       /* OPTIMAL */
        }
        else
        if ( dual_obj < 0.0) {
            status = 2;
            break;       /* PRIMAL INFEASIBLE */
        }
        else
        if ( primal_obj > 0.0) {
            status = 4;
            break;       /* DUAL INFEASIBLE */
        }
        else
        {
            status = 7;  /* NUMERICAL TROUBLE */
            break;
        }
    }

    /*************************************************************
    * STEP 3: Compute infeasibilities.
    *************************************************************/

    smx(m,n,A,kA,iA,x,rho);
    for (i=0; i<m; i++) {
        rho[i] = rho[i] - b[i]*phi + w[i];
    }
    normr = sqrt( dotprod(rho,rho,m) )/phi;
    for (i=0; i<m; i++) {
        rho[i] = -(1-delta)*rho[i] + w[i] - delta*mu/y[i];
    }

    smx(n,m,At,kAt,iAt,y,sigma);
    for (j=0; j<n; j++) {
        sigma[j] = -sigma[j] + c[j]*phi + z[j];
    }
```

```
norms = sqrt( dotprod(sigma,sigma,n) )/phi;
for (j=0; j<n; j++) {
    sigma[j] = -(1-delta)*sigma[j] + z[j] - delta*mu/x[j];
}

gamma = -(1-delta)*(dual_obj - primal_obj + psi) + psi - delta*mu/phi;

/*************************************************************
* Print statistics.
*************************************************************/

printf("%8d   %14.7e  %8.1e    %14.7e  %8.1e  %8.1e \n",
        iter, primal_obj/phi+f, normr,
              dual_obj/phi+f,   norms, mu );
fflush(stdout);

/*************************************************************
* STEP 4: Compute step directions.
*************************************************************/

for (j=0; j<n; j++) { D[j] = z[j]/x[j]; }
for (i=0; i<m; i++) { E[i] = w[i]/y[i]; }

ldltfac(n, m, kAt, iAt, At, E, D, kA, iA, A, v);

for (j=0; j<n; j++) { fx[j] = -sigma[j]; }
for (i=0; i<m; i++) { fy[i] =  rho[i]; }

forwardbackward(E, D, fy, fx);

for (j=0; j<n; j++) { gx[j] = -c[j]; }
for (i=0; i<m; i++) { gy[i] = -b[i]; }

forwardbackward(E, D, gy, gx);

dphi = (dotprod(c,fx,n)-dotprod(b,fy,m)+gamma)/
       (dotprod(c,gx,n)-dotprod(b,gy,m)-psi/phi);

for (j=0; j<n; j++) { dx[j] = fx[j] - gx[j]*dphi; }
for (i=0; i<m; i++) { dy[i] = fy[i] - gy[i]*dphi; }

for (j=0; j<n; j++) { dz[j] = delta*mu/x[j] - z[j] - D[j]*dx[j]; }
for (i=0; i<m; i++) { dw[i] = delta*mu/y[i] - w[i] - E[i]*dy[i]; }
dpsi = delta*mu/phi - psi - (psi/phi)*dphi;

/*************************************************************
* STEP 5: Compute step length (long steps).
*************************************************************/

theta = 0.0;
for (j=0; j<n; j++) {
    if (theta < -dx[j]/x[j]) { theta = -dx[j]/x[j]; }
    if (theta < -dz[j]/z[j]) { theta = -dz[j]/z[j]; }
}
for (i=0; i<m; i++) {
    if (theta < -dy[i]/y[i]) { theta = -dy[i]/y[i]; }
    if (theta < -dw[i]/w[i]) { theta = -dw[i]/w[i]; }
}
theta = MIN( 0.95/theta, 1.0 );

/*************************************************************
* STEP 6: Step to new point
*************************************************************/

for (j=0; j<n; j++) {
    x[j] = x[j] + theta*dx[j];
    z[j] = z[j] + theta*dz[j];
}
for (i=0; i<m; i++) {
    y[i] = y[i] + theta*dy[i];
    w[i] = w[i] + theta*dw[i];
```

```
    }
    phi = phi + theta*dphi;
    psi = psi + theta*dpsi;
}
```

Answers to Selected Exercises

1.3: See Exercise 2.19.
2.1: $(x_1, x_2, x_3, x_4) = (2, 0, 1, 0)$, $\zeta = 17$.
2.2: $(x_1, x_2) = (1, 0)$, $\zeta = 2$.
2.3: $(x_1, x_2, x_3) = (0, 0.5, 1.5)$, $\zeta = -3$.
2.4: $(x_1, x_2, x_3) = (0, 1, 0)$, $\zeta = -3$.
2.5: $(x_1, x_2) = (2, 1)$, $\zeta = 5$.
2.6: Infeasible.
2.7: Unbounded.
2.8: $(x_1, x_2) = (4, 8)$, $\zeta = 28$.
2.9: $(x_1, x_2, x_3) = (1.5, 2.5, 0)$, $\zeta = 10.5$.
2.10: $(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$, $\zeta = 9$.
2.11: $(x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34}) = (1, 0, 0, 1, 0, 1)$, $\zeta = 6$.
7.1: (1) $x^* = (2, 4, 0, 0, 0, 0, 8)$, $\xi^* = 14$. (2) x^* unchanged, $\xi^* = 12.2$. (3) $x^* = (0, 8, 0, 0, 0, 10, 10)$, $\xi^* = 16$.
7.2: $\Delta c_1 \in (-\infty, 1.2]$, $\Delta c_2 \in [-1.2, \infty)$, $\Delta c_3 \in [-1, 9]$, $\Delta c_4 \in (-\infty, 2.8]$.
9.1: $(x_1, x_2) = (0, 5)$.
9.2: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (0, 6, 1, 15, 2, 1, 0, 0)$.
10.5: The fundamental theorem was proved only for problems in standard form. The LP here can be reduced to standard form.
11.1: A should hide a or b with probabilities $b/(a+b)$ and $a/(a+b)$, respectively. B should hide a or b with equal probability.
11.3:

$$\begin{bmatrix} 2 & -4 \\ -3 & 6 \end{bmatrix}$$

12.1: Slope = $2/7$, intercept = 1.
12.2: Slope = $1/2$, intercept = 0.
12.7: (2) 340. (3) x^* is chosen so that the number of months in which extra workers will be used is equal to the number of months in the cycle (12) times the inhouse employee cost ($17.50) divided by the outhouse employee cost ($25) rounded down to the nearest integer.
12.8: Using L^1, $g = 8.976$. With L^2, $g = 8.924$.

13.6: The optimal spanning tree consists of the following arcs:

$$\{(a,b),(b,c),(c,f),(f,g),(d,g),(d,e),(g,h)\}.$$

The solution is not unique.

16.1: $x_1 = \left(1+2\mu+\sqrt{1+4\mu^2}\right)/2,$

$x_2 = \left(1-2\mu+\sqrt{1+4\mu^2}\right)/2$

$= 2\mu/\left(-(1-2\mu)+\sqrt{1+4\mu^2}\right).$

16.2: Let $c = \cos\theta$. If $c \neq 0$, then $x_1 = \left(c-2\mu+\sqrt{c^2+4\mu^2}\right)/2c$, else $x_1 = 1/2$. Formula for x_2 is the same except that $\cos\theta$ is replaced by $\sin\theta$.

16.3: $\max\{c^T x + \sum_j r_j \log x_j + \sum_i s_i \log w_i : Ax \le b, x \ge 0\}$.

17.1: Using $\delta = 1/10$ and $r = 9/10$:

(1) $x = (545, 302, 644)/680$, $y = (986, 1049)/680$,
$w = (131, 68)/680$, $z = (572, 815, 473)/680$.

(2) $x = (3107, 5114, 4763)/4250$, $y = (4016, 425)/4250$,
$w = (2783, 6374)/4250$, $z = (3692, 1685, 2036)/4250$.

(3) $x = (443, 296)/290$, $y = (263, 275, 347)/290$,
$w = (209, 197, 125)/290$, $z = (29, 176)/290$.

(4) $x = (9, 12, 8, 14)/10$, $y = (18)/10$,
$w = (1)/10$, $z = (9, 6, 11, 5)/10$.

19.1:

$$L = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ -1 & & 1 & & \\ & -1 & & 1 & \\ & & & -1 & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 0 & & \\ & & & 1 & \\ & & & & 0 \end{bmatrix}.$$

19.3:

$$L = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -\frac{1}{3} & 1 & & \\ -1 & & & 1 & \\ & & -\frac{6}{7} & \frac{1}{3} & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} -2 & & & & \\ & -3 & & & \\ & & \frac{7}{3} & & \\ & & & 3 & \\ & & & & -\frac{64}{21} \end{bmatrix}.$$

Bibliography

Adler, I. & Berenguer, S. (1981), Random linear programs, Technical Report 81-4, Operations Research Center Report, U.C. Berkeley. 209

Adler, I. & Megiddo, N. (1985), 'A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension', *Journal of the ACM* **32**, 871–895. 53, 208

Adler, I., Karmarkar, N., Resende, M. & Veiga, G. (1989), 'An implementation of Karmarkar's algorithm for linear programming', *Mathematical Programming* **44**, 297–335. 369

Ahuja, R., Magnanti, T. & Orlin, J. (1993), *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ. 240, 257

Anstreicher, K. (1996), Potential Reduction Algorithms, Technical report, Department of Management Sciences, University of Iowa. 306

Barnes, E. (1986), 'A variation on Karmarkar's algorithm for solving linear programming problems', *Mathematical Programming* **36**, 174–182. 346

Bayer, D. & Lagarias, J. (1989*a*), 'The nonlinear geometry of linear programming. I. affine and projective scaling trajectories', *Transactions of the AMS* **314**, 499–525. 289

Bayer, D. & Lagarias, J. (1989*b*), 'The nonlinear geometry of linear programming. II. Legendre transform coordinates and central trajectories', *Transactions of the AMS* **314**, 527–581. 289

Bazaraa, M., Jarvis, J. & Sherali, H. (1977), *Linear Programming and Network Flows*, 2 edn, Wiley, New York. 11, 240

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton, NJ. 257

Bendsøe, M., Ben-Tal, A. & Zowe, J. (1994), 'Optimization methods for truss geometry and topology design', *Structural Optimization* **7**, 141–159. 272

Bertsekas, D. (1991), *Linear Network Optimization*, The MIT Press, Cambridge, MA. 240

Bertsekas, D. (1995), *Nonlinear Programming*, Athena Scientific, Belmont MA. 289, 411

Bland, R. (1977), 'New finite pivoting rules for the simplex method', *Mathematics of Operations Research* **2**, 103–107. 44

Bloomfield, P. & Steiger, W. (1983), *Least Absolute Deviations: Theory, Applications, and Algorithms*, Birkhäuser, Boston. 208

Borgwardt, K.-H. (1982), 'The average number of pivot steps required by the simplex-method is polynomial', *Zeitschrift für Operations Research* **26**, 157–177. 53, 124, 208

Borgwardt, K.-H. (1987*a*), Probabilistic analysis of the simplex method, *in* 'Operations Research Proceedings, 16th DGOR Meeting', pp. 564–576. 53, 208

Borgwardt, K.-H. (1987*b*), *The Simplex Method—A Probabilistic Approach*, Springer-Verlag, Berlin-Heidelberg-New York. 53

Bradley, S., Hax, A. & Magnanti, T. (1977), *Applied Mathematical Programming*, Addison Wesley, Reading, MA. 11

Carathéodory, C. (1907), 'Über den Variabilitätsbereich der Koeffizienten von Potenzreihen, die gegebene Werte nicht annehmen', *Mathematische Annalen* **64**, 95–115. 171

Carpenter, T., Lustig, I., Mulvey, J. & Shanno, D. (1993), 'Higher order predictor-corrector interior point methods with application to quadratic objectives', *SIAM Journal on Optimization* **3**, 696–725. 306

Charnes, A. (1952), 'Optimality and degeneracy in linear programming', *Econometrica* **20**, 160–170. 44

Christofides, N. (1975), *Graph Theory: An Algorithmic Approach*, Academic Press, New York. 240

Chvátal, V. (1983), *Linear Programming*, Freeman, New York. 27, 187

Dantzig, G. (1951*a*), Application of the simplex method to a transportation problem, *in* T. Koopmans, ed., 'Activity Analysis of Production and Allocation', John Wiley and Sons, New York, pp. 359–373. 10, 240

Dantzig, G. (1951*b*), A proof of the equivalence of the programming problem and the game problem, *in* T. Koopmans, ed., 'Activity Analysis of Production and Allocation', John Wiley and Sons, New York, pp. 330–335. 10

Dantzig, G. (1955), 'Upper bounds, secondary constraints, and block triangularity in linear programming', *Econometrica* **23**, 174–183. 160

Dantzig, G. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ. 10, 27, 124

Dantzig, G. & Orchard-Hayes, W. (1954), 'The product form for the inverse in the simplex method', *Mathematical Tables and Other Aids to Computation* **8**, 64–67. 150

Dantzig, G., Orden, A. & Wolfe, P. (1955), 'The generalized simplex method for minimizing a linear form under linear inequality constraints', *Pacific Journal of Mathematics* **5**, 183–195. 44

den Hertog, D. (1994), *Interior Point Approach to Linear, Quadratic, and Convex Programming*, Kluwer Academic Publishers, Dordrecht. 423

Dijkstra, E. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**, 269–271. 257

Dikin, I. (1967), 'Iterative solution of problems of linear and quadratic programming', *Soviet Mathematics Doklady* **8**, 674–675. 346

Dikin, I. (1974), 'On the speed of an iterative process', *Upravlyaemye Sistemi* **12**, 54–60. 346

Dodge, Y., ed. (1987), *Statistical Data Analysis Based on The L^1-Norm and Related Methods*, North-Holland, Amsterdam. 208

Dorn, W., Gomory, R. & Greenberg, H. (1964), 'Automatic design of optimal structures', *J. de Mécanique* **3**, 25–52. 272

Dresher, M. (1961), *Games of Strategy: Theory and Application*, Prentice-Hall, Englewood Cliffs, NJ. 187

Duff, I., Erisman, A. & Reid, J. (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford. 150

Elias, P., Feinstein, A. & Shannon, C. (1956), 'Note on maximum flow through a network', *IRE Transactions on Information Theory* **IT-2**, 117–119. 257

Farkas, J. (1902), 'Theorie der einfachen Ungleichungen', *Journal für die reine und angewandte Mathematik* **124**, 1–27. 171

Fiacco, A. & McCormick, G. (1968), *Nonlinear Programming: Sequential Unconstrainted Minimization Techniques*, Research Analysis Corporation, McLean Virginia. Republished in 1990 by SIAM, Philadelphia. 289, 423

Ford, L. & Fulkerson, D. (1956), 'Maximal flow through a network', *Canadian Journal of Mathematics* **8**, 399–404. 257

Ford, L. & Fulkerson, D. (1958), 'Constructing maximal dynamic flows from static flows', *Operations Research* **6**, 419–433. 240

Ford, L. & Fulkerson, D. (1962), *Flows in Networks*, Princeton University Press, Princeton, NJ. 240

Forrest, J. & Tomlin, J. (1972), 'Updating triangular factors of the basis to maintain sparsity in the product form simplex method', *Mathematical Programming* **2**, 263–278. 150

Fourer, R. & Mehrotra, S. (1991), 'Solving symmetric indefinite systems in an interior point method for linear programming', *Mathematical Programming* **62**, 15–40. 331

Fourer, R., Gay, D. & Kernighan, B. (1993), *AMPL: A Modeling Language for Mathematical Programming*, Scientific Press. xiv

Fulkerson, D. & Dantzig, G. (1955), 'Computation of maximum flow in networks', *Naval Research Logistics Quarterly* **2**, 277–283. 257

Gal, T., ed. (1993), *Degeneracy in Optimization Problems*, Vol. 46/47 of *Annals of Operations Research*, J.C. Baltzer AG. 44

Gale, D., Kuhn, H. & Tucker, A. (1951), Linear programming and the theory of games, *in* T. Koopmans, ed., 'Activity Analysis of Production and Allocation', John Wiley and Sons, New York, pp. 317–329. 87, 187

Garey, M. & Johnson, D. (1977), *Computers and Intractability*, W.H. Freeman and Company, San Francisco. 54

Garfinkel, R. & Nemhauser, G. (1972), *Integer Programming*, John Wiley and Sons, New York. 393

Gass, S. & Saaty, T. (1955), 'The computational algorithm for the parametric objective function', *Naval Research Logistics Quarterly* **2**, 39–45. 124

Gay, D. (1985), 'Electronic mail distribution of linear programming test problems', *Mathematical Programming Society COAL Newslettter* **13**, 10–12. 199

Gill, P., Murray, W. & Wright, M. (1991), *Numerical Linear Algebra and Optimization*, Vol. 1, Addison-Wesley, Redwood City, CA. 150

Gill, P., Murray, W., Ponceleón, D. & Saunders, M. (1992), 'Preconditioners for indefinite systems arising in optimization', *SIAM Journal on Matrix Analysis and Applications* **13**(1), 292–311. 331

Goldfarb, D. & Reid, J. (1977), 'A practicable steepest-edge simplex algorithm', *Mathematical Programming* **12**, 361–371. 150

Golub, G. & VanLoan, C. (1989), *Matrix Computations*, 2 edn, The Johns Hopkins University Press, Baltimore, MD. 150

Gonin, R. & Money, A. (1989), *Nonlinear L_p-Norm Estimation*, Marcel Dekker, New York-Basel. 208

Gordan, P. (1873), 'Über die Auflösung linearer Gleichungen mit reelen Coefficienten', *Mathematische Annalen* **6**, 23–28. 171

Hall, L. & Vanderbei, R. (1993), 'Two-thirds is sharp for affine scaling', *OR Letters* **13**, 197–201. 346

Harris, P. (1973), 'Pivot selection methods of the Devex LP code', *Mathematical Programming* **5**, 1–28. 150

Hemp, W. (1973), *Optimum Structures*, Clarendon Press, Oxford. 272

Hillier, F. & Lieberman, G. (1977), *Introduction to Mathematical Programming*, 2 edn, McGraw-Hill, New York. 11

Hitchcock, F. (1941), 'The distribution of a produce from several sources to numerous localities', *Journal of Mathematical Physics* **20**, 224–230. 257

Hoffman, A. (1953), Cycling in the simplex algorithm, Technical Report Report 2974, National Bureau of Standards. 43

Howard, R. (1960), *Dynamic Programming and Markov Processes*, John Wiley and Sons, New York. 257

Huard, P. (1967), Resolution of mathematical programming with nonlinear constraints by the method of centers, *in* J. Abadie, ed., 'Nonlinear Programming', North-Holland, Amsterdam, pp. 209–219. 289

Jensen, P. & Barnes, J. (1980), *Network Flow Programming*, John Wiley and Sons, New York. 240

John, F. (1948), Extremum problems with inequalities as subsidiary conditions, *in* K. Fredrichs, O. Neugebauer & J. Stoker, eds, 'Studies and Essays: Courant Anniversary Volume', Wiley Interscience, New York, pp. 187–204. 313

Kantorovich, L. (1960), 'Mathematical methods in the organization and planning of production', *Management Science* **6**, 550–559. Original Russian version appeared

in 1939. 10

Karlin, S. (1959), *Mathematical Methods and Theory in Games, Programming, and Economics*, Vol. 1 and 2, Addison-Wesley, Reading, MA. 187

Karmarkar, N. (1984), 'A new polynomial time algorithm for linear programming', *Combinatorica* **4**, 373–395. 289, 306, 346

Karush, W. (1939), Minima of functions of several variables with inequalities as side conditions, Technical report, M.S. Thesis, Department of Mathematics, University of Chicago. 313

Kennington, J. & Helgason, R. (1980), *Algorithms for Network Programming*, John Wiley and Sons, New York. 240

Khachian, L. (1979), 'A polynomial algorithm in linear programming', *Doklady Academiia Nauk SSSR* **244**, 191–194. In Russian. English Translation: *Soviet Mathematics Doklady* **20**: 191-194. 54, 306

Klee, V. & Minty, G. (1972), How good is the simplex algorithm?, *in* O. Shisha, ed., 'Inequalities–III', Academic Press, New York, pp. 159–175. 53

Kojima, M., Mizuno, S. & Yoshise, A. (1989), A primal-dual interior point algorithm for linear programming, *in* N. Megiddo, ed., 'Progress in Mathematical Programming', Springer-Verlag, New York, pp. 29–47. 306

Kotzig, A. (1956), Súvislost' a Pravideliná Súvislots' Konečných Grafov, Technical report, Bratislava: Vysoká Škola Ekonomická. 257

Kuhn, H. (1950), 'A simplified two-person poker', *Annals of Mathematics Studies* **24**, 97–103. 187

Kuhn, H. (1976), Nonlinear prgramming: A historical view, *in* R. Cottle & C. Lemke, eds, 'Nonlinear Programming, SIAM-AMS Proceedings', Vol. 9, American Mathetical Society, Providence, RI, pp. 1–26. 313

Kuhn, H. & Tucker, A. (1951), Nonlinear prgramming, *in* J. Neyman, ed., 'Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability', University of California Press, Berkeley, CA, pp. 481–492. 313

Lawler, E. (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York. 240

Lemke, C. (1954), 'The dual method of solving the linear programming problem', *Naval Research Logistics Quarterly* **1**, 36–47. 87

Lemke, C. (1965), 'Bimatrix equilibrium points and mathematical programming', *Management Science* **11**, 681–689. 124

Luenberger, D. (1984), *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading MA. 289

Lustig, I. (1990), 'Feasibility issues in a primal-dual interior-point method for linear programming', *Mathematical Programming* **49**(2), 145–162. 306

Lustig, I., Marsten, R. & Shanno, D. (1994), 'Interior point methods for linear programming: computational state of the art', *ORSA J. on Computing* **6**, 1–14. 331

Markowitz, H. (1957), 'The elimination form of the inverse and its application to linear programming', *Management Science* **3**, 255–269. 150

Markowitz, H. (1959), *Portfolio Selection: Efficient Diversification of Investments*, Wiley, New York. 411

Marshall, K. & Suurballe, J. (1969), 'A note on cycling in the simplex method', *Naval Research Logistics Quarterly* **16**, 121–137. 43

Mascarenhas, W. (1997), 'The affine scaling algorithm fails for $\lambda = 0.999$', *SIAM J. Optimization* **7**, 34–46. 346

Megiddo, N. (1989), Pathways to the optimal set in linear programming, *in* N. Megiddo, ed., 'Progress in Mathematical Programming', Springer-Verlag, New York, pp. 131–158. 289

Mehrotra, S. (1989), Higher order methods and their performance, Technical Report TR 90-16R1, Department of Ind. Eng. and Mgmt. Sci., Northwestern University, Evanston, IL. Revised July, 1991. 368

Mehrotra, S. (1992), 'On the implementation of a (primal-dual) interior point method', *SIAM Journal on Optimization* **2**, 575–601. 368

Michell, A. (1904), 'The limits of economy of material in frame structures', *Phil. Mag.* **8**, 589–597. 272

Mizuno, S., Todd, M. & Ye, Y. (1993), 'On adaptive-step primal-dual interior-point algorithms for linear programming', *Mathematics of Operations Research* **18**, 964–981. 368

Monteiro, R. & Adler, I. (1989), 'Interior path following primal-dual algorithms: Part i: Linear programming', *Mathematical Programming* **44**, 27–41. 306, 411

Nash, S. & Sofer, A. (1996), *Linear and Nonlinear Programming*, McGraw-Hill, New York. 289, 411

Nazareth, J. (1986), 'Homotopy techniques in linear programming', *Algorithmica* **1**, 529–535. 124, 306

Nazareth, J. (1987), *Computer Solutions of Linear Programs*, Oxford University Press, Oxford. 124

Nazareth, J. (1996), 'The implementation of linear programming algorithms based on homotopies', *Algorithmica* **15**, 332–350. 306

Nemhauser, G. & Wolsey, L. (1988), *Integer and Combinatorial Optimization*, Wiley, New York. 393

Nesterov, Y. & Nemirovsky, A. (1993), *Interior Point Polynomial Methods in Convex Programming : Theory and Algorithms*, SIAM Publications, Philadelphia. 423

Recski, A. (1989), *Matroid Theory and its Applications in Electric Network Theory and in Statics*, Springer-Verlag, Berlin-Heidelberg-New York. 272

Reid, J. (1982), 'A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases', *Mathematical Programming* **24**, 55–69. 150

Rockafellar, R. (1970), *Convex Analysis*, Princeton University Press, Princeton, NJ. 171

Rozvany, G. (1989), *Structural Design via Optimality Criteria*, Kluwer, Dordrecht. 272

Saigal, R. (1995), *Linear Programming*, Kluwer Academic Publishers, Boston. 347

Saunders, M. (1973), The complexity of LU updating in the simplex method, *in* R. Andersen & R. Brent, eds, 'The complexity of computational problem solving', University Press, Queensland, pp. 214–230. 150

Smale, S. (1983), 'On the average number of steps of the simplex method of linear programming', *Mathematical Programming* **27**, 241–262. 53, 124, 208

Stiemke, E. (1915), 'Über positive Lösungen homogener linearer Gleichungen', *Mathematische Annalen* **76**, 340–342. 171

Todd, M. (1986), 'Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming', *Mathematical Programming* **35**, 173–192. 53, 208

Todd, M. (1995), Potential-reduction methods in mathematical programming, Technical Report 1112, SORIE, Cornell University, Ithaca, NY. 306

Tsuchiya, T. & Muramatsu, M. (1992), 'Global convergence of a long-step affine scaling algorithm for degenerate linear programming problems', *SIAM J. Optimization* **5**(3), 525–551. 346

Tucker, A. (1956), 'Dual systems of homogeneous linear equations', *Annals of Mathematics Studies* **38**, 3–18. 171, 368

Turner, K. (1991), 'Computing projections for the Karmarkar algorithm', *Linear Algebra and Its Applications* **152**, 141–154. 331

Vanderbei, R. (1989), 'Affine scaling and linear programs with free variables', *Mathematical Programming* **39**, 31–44. 347

Vanderbei, R. (1994), 'Interior-point methods: algorithms and formulations', *ORSA J. on Computing* **6**, 32–34. 331

Vanderbei, R. (1995), 'Symmetric quasi-definite matrices', *SIAM Journal on Optimization* **5**(1), 100–113. 331

Vanderbei, R. (1999), 'LOQO: An interior point code for quadratic programming', *Optimization Methods and Software* **12**, 451–484. 411

Vanderbei, R. & Carpenter, T. (1993), 'Symmetric indefinite systems for interior-point methods', *Mathematical Programming* **58**, 1–32. 331

Vanderbei, R. & Shanno, D. (1999), 'An interior-point algorithm for nonconvex nonlinear programming', *Computational Optimization and Applications* **13**, 231–252. 423

Vanderbei, R., Meketon, M. & Freedman, B. (1986), 'A modification of Karmarkar's linear programming algorithm', *Algorithmica* **1**, 395–407. 346

Ville, J. (1938), Sur la théorie général des jeux ou intervient l'habileté des jouers, *in* E. Borel, ed., 'Traité du Calcul des Probabilités et des ses Applications', Paris, Gauthiers-Villars. 171

von Neumann, J. (1928), 'Zur Theorie der Gesselschaftschpiele', *Mathematische Annalen* **100**, 295–320. 187

von Neumann, J. & Morgenstern, O. (1947), *Theory of Games and Economic Behavior*, 2nd edn, Princeton University Press, Princeton, NJ. 187

Wright, S. (1996), *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, USA. 306

Xu, X., Hung, P. & Ye, Y. (1993), A simplified homogeneous and self-dual linear programming algorithm and its implementation, Technical report, College of Business Administration, University of Iowa. To appear in Annals of Operations Research. 368

Ye, Y., Todd, M. & Mizuno, S. (1994), 'An $o(\sqrt{nl})$-iteration homogeneous and self-dual linear programming algorithm', *Mathematics of Operations Research* **19**, 53–67. 368

Index