Chapter 1
Introduction

1.1 Optimization

The concept of optimization is now well rooted as a principle underlying the analysis of many complex decision or allocation problems. It offers a certain degree of philosophical elegance that is hard to dispute, and it often offers an indispensable degree of operational simplicity. Using this optimization philosophy, one approaches a complex decision problem, involving the selection of values for a number of interrelated variables, by focusing attention on a single objective designed to quantify performance and measure the quality of the decision. This one objective is maximized (or minimized, depending on the formulation) subject to the constraints that may limit the selection of decision variable values. If a suitable single aspect of a problem can be isolated and characterized by an objective, be it profit or loss in a business setting, speed or distance in a physical problem, expected return in the environment of risky investments, or social welfare in the context of government planning, optimization may provide a suitable framework for analysis.

It is, of course, a rare situation in which it is possible to fully represent all the complexities of variable interactions, constraints, and appropriate objectives when faced with a complex decision problem. Thus, as with all quantitative techniques of analysis, a particular optimization formulation should be regarded only as an approximation. Skill in modeling, to capture the essential elements of a problem, and good judgment in the interpretation of results are required to obtain meaningful conclusions. Optimization, then, should be regarded as a tool of conceptualization and analysis rather than as a principle yielding the philosophically correct solution.

Skill and good judgment, with respect to problem formulation and interpretation of results, is enhanced through concrete practical experience and a thorough understanding of relevant theory. Problem formulation itself always involves a tradeoff between the conflicting objectives of building a mathematical model sufficiently complex to accurately capture the problem description and building a model that is

D.G. Luenberger, Y. Ye, *Linear and Nonlinear Programming*, International Series in Operations Research & Management Science 228,
DOI 10.1007/978-3-319-18842-3_1

tractable. The expert model builder is facile with both aspects of this tradeoff. One aspiring to become such an expert must learn to identify and capture the important issues of a problem mainly through example and experience; one must learn to distinguish tractable models from nontractable ones through a study of available technique and theory and by nurturing the capability to extend existing theory to new situations.

This book is centered around a certain optimization structure—that characteristic of linear and nonlinear programming. Examples of situations leading to this structure are sprinkled throughout the book, and these examples should help to indicate how practical problems can be often fruitfully structured in this form. The book mainly, however, is concerned with the development, analysis, and comparison of algorithms for solving general subclasses of optimization problems. This is valuable not only for the algorithms themselves, which enable one to solve given problems, but also because identification of the collection of structures they most effectively solve can enhance one's ability to formulate problems.

1.2 Types of Problems

The content of this book is divided into three major parts: Linear Programming, Unconstrained Problems, and Constrained Problems. The last two parts together comprise the subject of nonlinear programming.

Linear Programming

Linear programming is without doubt the most natural mechanism for formulating a vast array of problems with modest effort. A linear programming problem is characterized, as the name implies, by linear functions of the unknowns; the objective is linear in the unknowns, and the constraints are linear equalities or linear inequalities in the unknowns. One familiar with other branches of linear mathematics might suspect, initially, that linear programming formulations are popular because the mathematics is nicer, the theory is richer, and the computation simpler for linear problems than for nonlinear ones. But, in fact, these are *not* the primary reasons. In terms of mathematical and computational properties, there are much broader classes of optimization problems than linear programming problems that have elegant and potent theories and for which effective algorithms are available. It seems that the popularity of linear programming lies primarily with the formulation phase of analysis rather than the solution phase—and for good cause. For one thing, a great number of constraints and objectives that arise in practice *are* indisputably linear. Thus, for example, if one formulates a problem with a budget constraint restricting the total amount of money to be allocated among two different commodities, the budget constraint takes the form $x_1 + x_2 \leq B$, where $x_j,\ i = 1, 2,$

is the amount allocated to activity i, and B is the budget. Similarly, if the objective is, for example, maximum weight, then it can be expressed as $w_1x_1 + w_2x_2$, where w_j, $i = 1, 2$, is the unit weight of the commodity i. The overall problem would be expressed as

$$\begin{aligned} \text{maximize} \quad & w_1x_1 + w_2x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq B, \\ & x_1 \geq 0, \; x_2 \geq 0, \end{aligned}$$

which is an elementary linear program. The linearity of the budget constraint is extremely natural in this case and does not represent simply an approximation to a more general functional form.

Another reason that linear forms for constraints and objectives are so popular in problem formulation is that they are often the least difficult to define. Thus, even if an objective function is not purely linear by virtue of its inherent definition (as in the above example), it is often far easier to define it as being linear than to decide on some other functional form and convince others that the more complex form is the best possible choice. Linearity, therefore, by virtue of its simplicity, often is selected as the easy way out or, when seeking generality, as the only functional form that will be equally applicable (or nonapplicable) in a class of similar problems.

Of course, the theoretical and computational aspects do take on a somewhat special character for linear programming problems—the most significant development being the simplex method. This algorithm is developed in Chaps. 2 and 3. More recent interior point methods are nonlinear in character and these are developed in Chap. 5.

Unconstrained Problems

It may seem that unconstrained optimization problems are so devoid of structural properties as to preclude their applicability as useful models of meaningful problems. Quite the contrary is true for two reasons. First, it can be argued, quite convincingly, that if the scope of a problem is broadened to the consideration of all relevant decision variables, there may then be no constraints—or put another way, constraints represent artificial delimitations of scope, and when the scope is broadened the constraints vanish. Thus, for example, it may be argued that a budget constraint is not characteristic of a meaningful problem formulation; since by borrowing at some interest rate it is always possible to obtain additional funds, and hence rather than introducing a budget constraint, a term reflecting the cost of funds should be incorporated into the objective. A similar argument applies to constraints describing the availability of other resources which at some cost (however great) could be supplemented.

The second reason that many important problems can be regarded as having no constraints is that constrained problems are sometimes easily converted to unconstrained problems. For instance, the sole effect of equality constraints is simply to limit the degrees of freedom, by essentially making some variables functions of others. These dependencies can sometimes be explicitly characterized, and a new problem having its number of variables equal to the true degree of freedom can be determined. As a simple specific example, a constraint of the form $x_1 + x_2 = B$ can be eliminated by substituting $x_2 = B - x_1$ everywhere else that x_2 appears in the problem.

Aside from representing a significant class of practical problems, the study of unconstrained problems, of course, provides a stepping stone toward the more general case of constrained problems. Many aspects of both theory and algorithms are most naturally motivated and verified for the unconstrained case before progressing to the constrained case.

Constrained Problems

In spite of the arguments given above, many problems met in practice are formulated as constrained problems. This is because in most instances a complex problem such as, for example, the detailed production policy of a giant corporation, the planning of a large government agency, or even the design of a complex device cannot be directly treated in its entirety accounting for all possible choices, but instead must be decomposed into separate subproblems—each subproblem having constraints that are imposed to restrict its scope. Thus, in a planning problem, budget constraints are commonly imposed in order to decouple that one problem from a more global one. Therefore, one frequently encounters general nonlinear constrained mathematical programming problems.

The general mathematical programming problem can be stated as

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & h_j(\mathbf{x}) = 0, \; i = 1, 2, \ldots, m \\ & g_j(\mathbf{x}) \leq 0, \; j = 1, 2, \; p \\ & \mathbf{x} \in S. \end{aligned}$$

In this formulation, $\mathbf{x}$ is an n-dimensional vector of unknowns, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and f, h_i, $i = 1, 2, \ldots, m$, and g_j, $j = 1, 2, \ldots, p$, are real-valued functions of the variables $x_1, x_2, \ldots, x_n$. The set S is a subset of n-dimensional space. The function f is the *objective function* of the problem and the equations, inequalities, and set restrictions are *constraints.*

Generally, in this book, additional assumptions are introduced in order to make the problem smooth in some suitable sense. For example, the functions in the problem are usually required to be continuous, or perhaps to have continuous derivatives. This ensures that small changes in $\mathbf{x}$ lead to small changes in other values associated with the problem. Also, the set S is not allowed to be arbitrary but usually is required to be a connected region of n-dimensional space, rather than, for example, a set of distinct isolated points. This ensures that small changes in $\mathbf{x}$ can be made. Indeed, in a majority of problems treated, the set S is taken to be the entire space; there is no set restriction.

In view of these smoothness assumptions, one might characterize the problems treated in this book as *continuous variable programming*, since we generally discuss problems where all variables and function values can be varied continuously. In fact, this assumption forms the basis of many of the algorithms discussed, which operate essentially by making a series of small movements in the unknown $\mathbf{x}$ vector.

1.3 Size of Problems

One obvious measure of the complexity of a programming problem is its size, measured in terms of the number of unknown variables or the number of constraints. As might be expected, the size of problems that can be effectively solved has been increasing with advancing computing technology and with advancing theory. Today, with present computing capabilities, however, it is reasonable to distinguish three classes of problems: *small-scale problems* having about five or fewer unknowns and constraints; *intermediate-scale problems* having from about five to a hundred or a thousand variables; and *large-scale problems* having perhaps thousands or even millions of variables and constraints. This classification is not entirely rigid, but it reflects at least roughly not only size but the basic differences in approach that accompany different size problems. As a rough rule, small-scale problems can be solved by hand or by a small computer. Intermediate-scale problems can be solved on a personal computer with general purpose mathematical programming codes. Large-scale problems require sophisticated codes that exploit special structure and usually require large computers.

Much of the basic theory associated with optimization, particularly in nonlinear programming, is directed at obtaining necessary and sufficient conditions satisfied by a solution point, rather than at questions of computation. This theory involves mainly the study of Lagrange multipliers, including the Karush-Kuhn-Tucker Theorem and its extensions. It tremendously enhances insight into the philosophy of constrained optimization and provides satisfactory basic foundations for other important disciplines, such as the theory of the firm, consumer economics, and optimal control theory. The interpretation of Lagrange multipliers that accompanies this theory is valuable in virtually every optimization setting. As a basis for computing numerical solutions to optimization, however, this theory is far from adequate, since it does not consider the difficulties associated with solving the equations resulting from the necessary conditions.

If it is acknowledged from the outset that a given problem is too large and too complex to be efficiently solved by hand (and hence it is acknowledged that a computer solution is desirable), then one's theory should be directed toward development of procedures that exploit the efficiencies of computers. In most cases this leads to the abandonment of the idea of solving the set of necessary conditions in favor of the more direct procedure of searching through the space (in an intelligent manner) for ever-improving points.

Today, search techniques can be effectively applied to more or less general nonlinear programming problems. Problems of great size, *large-scale programming* problems, can be solved if they possess special structural characteristics, especially sparsity, that can be exploited by a solution method. Today linear programming software packages are capable of automatically identifying sparse structure within the input data and taking advantage of this sparsity in numerical computation. It is now not uncommon to solve linear programs of up to a million variables and constraints, as long as the structure is sparse. Problem-dependent methods, where the structure is not automatically identified, are largely directed to transportation and network flow problems as discussed in the book.

This book focuses on the aspects of general theory that are most fruitful for computation in the widest class of problems. While necessary and sufficient conditions are examined and their application to small-scale problems is illustrated, our primary interest in such conditions is in their role as the core of a broader theory applicable to the solution of larger problems. At the other extreme, although some instances of structure exploitation are discussed, we focus primarily on the general continuous variable programming problem rather than on special techniques for special structures.

1.4 Iterative Algorithms and Convergence

The most important characteristic of a high-speed computer is its ability to perform repetitive operations efficiently, and in order to exploit this basic characteristic, most algorithms designed to solve large optimization problems are iterative in nature. Typically, in seeking a vector that solves the programming problem, an initial vector $\mathbf{x}_0$ is selected and the algorithm generates an improved vector $\mathbf{x}_1$. The process is repeated and a still better solution $\mathbf{x}_2$ is found. Continuing in this fashion, a sequence of ever-improving points $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots$, is found that approaches a solution point $\mathbf{x}^*$. For linear programming problems solved by the simplex method, the generated sequence is of finite length, reaching the solution point exactly after a finite (although initially unspecified) number of steps. For nonlinear programming problems or interior-point methods, the sequence generally does not ever exactly reach the solution point, but converges toward it. In operation, the process is terminated when a point sufficiently close to the solution point, for practical purposes, is obtained.

The theory of iterative algorithms can be divided into three (somewhat overlapping) aspects. The first is concerned with the creation of the algorithms themselves. Algorithms are not conceived arbitrarily, but are based on a creative examination of the programming problem, its inherent structure, and the efficiencies of digital computers. The second aspect is the verification that a given algorithm will in fact generate a sequence that converges to a solution point. This aspect is referred to as *global convergence analysis*, since it addresses the important question of whether the algorithm, when initiated far from the solution point, will eventually converge to it. The third aspect is referred to as *local convergence analysis* or *complexity analysis* and is concerned with the rate at which the generated sequence of points converges to the solution. One cannot regard a problem as solved simply because an algorithm is known which will converge to the solution, since it may require an exorbitant amount of time to reduce the error to an acceptable tolerance. It is essential when prescribing algorithms that some estimate of the time required be available. It is the convergence-rate aspect of the theory that allows some quantitative evaluation and comparison of different algorithms, and at least crudely, assigns a measure of tractability to a problem, as discussed in Sect. 1.1.

A modern-day technical version of Confucius' most famous saying, and one which represents an underlying philosophy of this book, might be, "One good theory is worth a thousand computer runs." Thus, the convergence properties of an iterative algorithm can be estimated with confidence either by performing numerous computer experiments on different problems or by a simple well-directed theoretical analysis. A simple theory, of course, provides invaluable insight as well as the desired estimate.

For linear programming using the simplex method, solid theoretical statements on the speed of convergence were elusive, because the method actually converges to an exact solution in a finite number of steps. The question is how many steps might be required. This question was finally resolved when it was shown that it was possible for the number of steps to be exponential in the size of the program. The situation is different for interior point algorithms, which essentially treat the problem by introducing nonlinear terms, and which therefore do not generally obtain a solution in a finite number of steps but instead converge toward a solution.

For nonlinear programs, including interior point methods applied to linear programs, it is meaningful to consider the speed of convergence. There are many different classes of nonlinear programming algorithms, each with its own convergence characteristics. However, in many cases the convergence properties can be deduced analytically by fairly simple means, and this analysis is substantiated by computational experience. Presentation of convergence analysis, which seems to be the natural focal point of a theory directed at obtaining specific answers, is a unique feature of this book.

There are in fact two aspects of convergence-rate theory. The first is generally known as *complexity analysis* and focuses on how fast the method converges overall, distinguishing between polynomial-time algorithms and non-polynomial-time algorithms. The second aspect provides more detailed analysis of how fast the method converges in the final stages, and can provide comparisons between different algorithms. Both of these are treated in this book.

The convergence-rate theory presented has two somewhat surprising but definitely pleasing aspects. First, the theory is, for the most part, extremely simple in nature. Although initially one might fear that a theory aimed at predicting the speed of convergence of a complex algorithm might itself be doubly complex, in fact the associated convergence analysis often turns out to be exceedingly elementary, requiring only a line or two of calculation. Second, a large class of seemingly distinct algorithms turns out to have a common convergence rate. Indeed, as emphasized in the later chapters of the book, there is a *canonical rate* associated with a given programming problem that seems to govern the speed of convergence of many algorithms when applied to that problem. It is this fact that underlies the potency of the theory, allowing definitive comparisons among algorithms to be made even without detailed knowledge of the problems to which they will be applied. Together these two properties, simplicity and potency, assure convergence analysis a permanent position of major importance in mathematical programming theory.