# CHAPTER 12

# Nonlinear Programming

The fundamental role of linear programming in OR is accurately reflected by the fact that it is the focus of a *third* of this book. A key assumption of linear programming is that *all its functions* (objective function and constraint functions) are linear. Although this assumption essentially holds for many practical problems, it frequently does not hold. Therefore, it often is necessary to deal directly with nonlinear programming problems, so we turn our attention to this important area.

In one general form,[1] the *nonlinear programming problem* is to find $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ so as to

$$\text{Maximize} \quad f(\mathbf{x}),$$

subject to

$$g_i(\mathbf{x}) \le b_i, \qquad \text{for } i = 1, 2, \ldots, m,$$

and

$$\mathbf{x} \ge \mathbf{0},$$

where $f(\mathbf{x})$ and the $g_i(\mathbf{x})$ are given functions of the $n$ decision variables.[2]

There are many different types of nonlinear programming problems, depending on the characteristics of the $f(\mathbf{x})$ and $g_i(\mathbf{x})$ functions. Different algorithms are used for the different types. For certain types where the functions have simple forms, problems can be solved relatively efficiently. For some other types, solving even small problems is a real challenge.

Because of the many types and the many algorithms, nonlinear programming is a particularly large subject. We do not have the space to survey it completely. However, we do present a few sample applications and then introduce some of the basic ideas for solving certain important types of nonlinear programming problems.

Both Appendixes 2 and 3 provide useful background for this chapter, and we recommend that you review these appendixes as you study the next few sections.

[1] The other *legitimate forms* correspond to those for *linear programming* listed in Sec. 3.2. Section 4.6 describes how to convert these other forms to the form given here.

[2] For simplicity, we assume throughout the chapter that *all* these functions either are *differentiable* everywhere or are *piecewise linear functions* (discussed in Secs. 12.1 and 12.8).

## 12.1 SAMPLE APPLICATIONS

The following examples illustrate a few of the many important types of problems to which nonlinear programming has been applied.

### The Product-Mix Problem with Price Elasticity

In *product-mix* problems, such as the Wyndor Glass Co. problem of Sec. 3.1, the goal is to determine the optimal mix of production levels for a firm's products, given limitations on the resources needed to produce those products, in order to maximize the firm's total profit. In some cases, there is a fixed unit profit associated with each product, so the resulting objective function will be linear. However, in many product-mix problems, certain factors introduce *nonlinearities* into the objective function.

For example, a large manufacturer may encounter *price elasticity*, whereby the amount of a product that can be sold has an inverse relationship to the price charged. Thus, the *price-demand curve* for a typical product might look like the one shown in Fig. 12.1, where $p(x)$ is the price required in order to be able to sell $x$ units. The firm's profit from producing and selling $x$ units of the product then would be the sales revenue, $xp(x)$, minus the production and distribution costs. Therefore, if the unit cost for producing and distributing the product is fixed at $c$ (see the dashed line in Fig. 12.1), the firm's profit from producing and selling $x$ units is given by the nonlinear function

$$P(x) = xp(x) - cx,$$

as plotted in Fig. 12.2. If *each* of the firm's $n$ products has a similar profit function, say, $P_j(x_j)$ for producing and selling $x_j$ units of product $j$ ($j = 1, 2, \ldots, n$), then the overall objective function is

$$f(\mathbf{x}) = \sum_{j=1}^{n} P_j(x_j),$$

a sum of nonlinear functions.

Another reason that nonlinearities can arise in the objective function is the fact that the *marginal cost* of producing another unit of a given product varies with the production level. For example, the marginal cost may decrease when the production level is increased because of a *learning-curve effect* (more efficient production with more experience). On

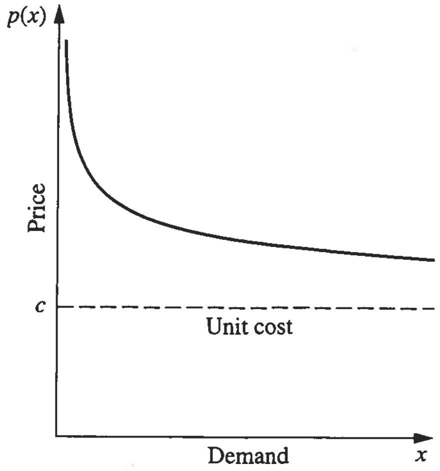


**FIGURE 12.1**
Price-demand curve.

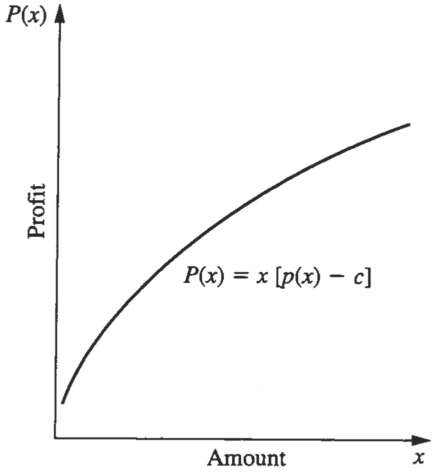


**FIGURE 12.2** Profit function.

the other hand, it may increase instead, because special measures such as overtime or more expensive production facilities may be needed to increase production further.

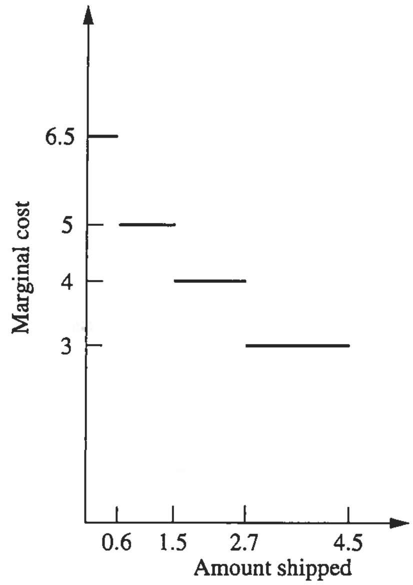
Nonlinearities also may arise in the $g_i(\mathbf{x})$ constraint functions in a similar fashion. For example, if there is a budget constraint on total production cost, the cost function will be nonlinear if the marginal cost of production varies as just described. For constraints on the other kinds of resources, $g_i(\mathbf{x})$ will be nonlinear whenever the use of the corresponding resource is not strictly proportional to the production levels of the respective products.

### The Transportation Problem with Volume Discounts on Shipping Costs

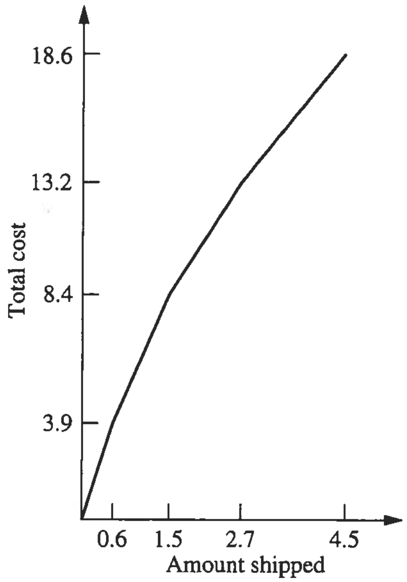
As illustrated by the P & T Company example in Sec. 8.1, a typical application of the transportation problem is to determine an optimal plan for shipping goods from various sources to various destinations, given supply and demand constraints, in order to minimize total shipping cost. It was assumed in Chap. 8 that the *cost per unit shipped* from a given source to a given destination is *fixed,* regardless of the amount shipped. In actuality, this cost may not be fixed. *Volume discounts* sometimes are available for large shipments, so that the *marginal cost* of shipping one more unit might follow a pattern like the one shown in Fig. 12.3. The resulting cost of shipping $x$ units then is given by a *nonlinear* function $C(x)$, which is a *piecewise linear function* with slope equal to the marginal cost, like the one shown in Fig. 12.4. [The function in Fig. 12.4 consists of a line segment with slope 6.5 from (0, 0) to (0.6, 3.9), a second line segment with slope 5 from (0.6, 3.9) to (1.5, 8.4), a third line segment with slope 4 from (1.5, 8.4) to (2.7, 13.2), and a fourth line segment with slope 3 from (2.7, 13.2) to (4.5, 18.6).] Consequently, if *each* combination of source and destination has a similar shipping cost function, so that the cost of shipping $x_{ij}$ units from source $i$ ($i = 1, 2, \ldots, m$) to destination $j$ ($j = 1, 2, \ldots, n$) is given by a nonlinear function $C_{ij}(x_{ij})$, then the overall objective function to be *minimized* is

$$f(\mathbf{x}) = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij}(x_{ij}).$$

Even with this nonlinear objective function, the constraints normally are still the special linear constraints that fit the transportation problem model in Sec. 8.1.



■ **FIGURE 12.3**
Marginal shipping cost.



■ **FIGURE 12.4**
Shipping cost function.

## Portfolio Selection with Risky Securities

It now is common practice for professional managers of large stock portfolios to use computer models based partially on nonlinear programming to guide them. Because investors are concerned about both the *expected return* (gain) and the *risk* associated with their investments, nonlinear programming is used to determine a portfolio that, under certain assumptions, provides an optimal trade-off between these two factors. This approach is based largely on path-breaking research done by Harry Markowitz and William Sharpe that helped them win the 1990 Nobel Prize in Economics.

A nonlinear programming model can be formulated for this problem as follows. Suppose that $n$ stocks (securities) are being considered for inclusion in the portfolio, and let the

## An Application Vignette

The **Bank Hapoalim Group** is Israel's largest banking group, providing services within Israel through a network of 327 branches, nine regional business centers, and various domestic subsidiaries. It also operates worldwide through 37 branches, offices, and subsidiaries in major financial centers in North and South America and Europe.

A major part of Bank Hapoalim's business involves providing investment advisors for its customers. To stay ahead of its competitors, management embarked on a restructuring program to provide these investment advisors with state-of-the-art methodology and technology. An OR team was formed to do this.

The team concluded that it needed to develop a flexible decision-support system for the investment advisors that could be tailored to meet the diverse needs of every customer. Each customer would be asked to provide extensive information about his or her needs, including choosing among various alternatives regarding his or her investment objectives, investment horizon, choice of an index to strive to exceed, preference with regard to liquidity and currency, etc. A series of questions also would be asked to ascertain the customer's risk-taking classification.

The natural choice of the model to drive the resulting decision-support system (called the *Opti-Money System*) was the *classical nonlinear programming model for portfolio selection* described in this section of the book, with modifications to incorporate all the information about the needs of the individual customer. This model generates an optimal weighting of 60 possible asset classes of equities and bonds in the portfolio, and the investment advisor then works with the customer to choose the specific equities and bonds within these classes.

In one recent year, the bank's investment advisors held some 133,000 consultation sessions with 63,000 customers while using this decision-support system. *The annual earnings* over benchmarks to customers who follow the investment advice provided by the system *total approximately* **US$244 million**, *while adding more than* **US$31 million** *to the bank's annual income.*

**Source:** M. Avriel, H. Pri-Zan, R. Meiri, and A. Peretz: "Opti-Money at Bank Hapoalim: A Model-Based Investment Decision-Support System for Individual Customers," *Interfaces*, **34**(1): 39–50, Jan.–Feb. 2004. (A link to this article is provided on our website, www.mhhe.com/hillier.)

decision variables $x_j$ ($j = 1, 2, \dots, n$) be the number of shares of stock $j$ to be included. Let $\mu_j$ and $\sigma_{jj}$ be the (estimated) *mean* and *variance*, respectively, of the return on each share of stock $j$, where $\sigma_{jj}$ measures the risk of this stock. For $i = 1, 2, \dots, n$ ($i \neq j$), let $\sigma_{ij}$ be the *covariance* of the return on one share each of stock $i$ and stock $j$. (Because it would be difficult to estimate all the $\sigma_{ij}$ values, the usual approach is to make certain assumptions about market behavior that enable us to calculate $\sigma_{ij}$ directly from $\sigma_{ii}$ and $\sigma_{jj}$.) Then the expected value $R(\mathbf{x})$ and the variance $V(\mathbf{x})$ of the total return from the entire portfolio are

$$R(\mathbf{x}) = \sum_{j=1}^{n} \mu_j x_j$$

and

$$V(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{ij} x_i x_j,$$

where $V(\mathbf{x})$ measures the risk associated with the portfolio. One way to consider the trade-off between these two factors is to use $V(\mathbf{x})$ as the objective function to be minimized and then impose the constraint that $R(\mathbf{x})$ must be no smaller than the minimum acceptable expected return. The complete nonlinear programming model then would be

$$\text{Minimize} \qquad V(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{ij} x_i x_j,$$

subject to

$$\sum_{j=1}^{n} \mu_j x_j \geq L$$

$$\sum_{j=1}^{n} P_j x_j \leq B$$

to use com-
se investors
'ith their in-
ıder certain
approach is
iam Sharpe

ɔllows. Sup-
, and let the

and

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

where $L$ is the minimum acceptable expected return, $P_j$ is the price for each share of stock $j$, and $B$ is the amount of money budgeted for the portfolio.

One drawback of this formulation is that it is relatively difficult to choose an appropriate value for $L$ for obtaining the best trade-off between $R(\mathbf{x})$ and $V(\mathbf{x})$. Therefore, rather than stopping with one choice of $L$, it is common to use a *parametric* (nonlinear) programming approach to generate the optimal solution as a function of $L$ over a wide range of values of $L$. The next step is to examine the values of $R(\mathbf{x})$ and $V(\mathbf{x})$ for these solutions that are optimal for some value of $L$ and then to choose the solution that seems to give the best trade-off between these two quantities. This procedure often is referred to as generating the solutions on the *efficient frontier* of the two-dimensional graph of $(R(\mathbf{x}), V(\mathbf{x}))$ points for feasible $\mathbf{x}$. The reason is that the $(R(\mathbf{x}), V(\mathbf{x}))$ point for an optimal $\mathbf{x}$ (for some $L$) lies on the *frontier* (boundary) of the feasible points. Furthermore, each optimal $\mathbf{x}$ is *efficient* in the sense that no other feasible solution is at least equally good with one measure ($R$ or $V$) and strictly better with the other measure (smaller $V$ or larger $R$).

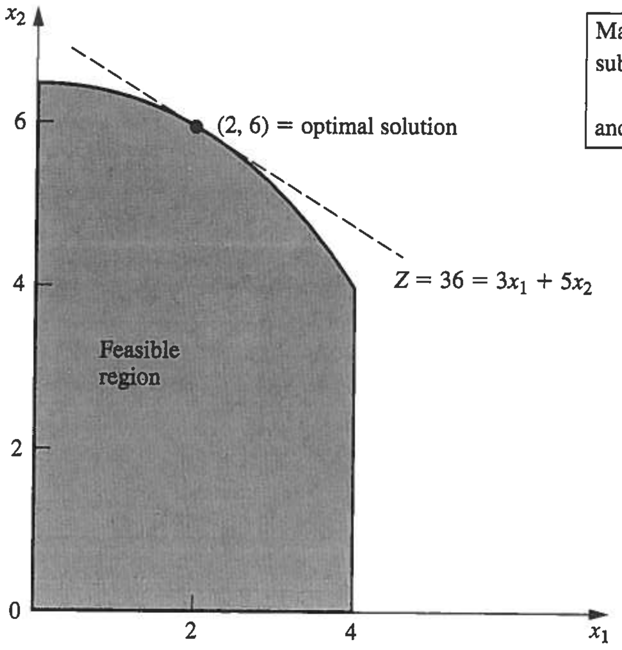
This application of nonlinear programming is a particularly important one. The use of nonlinear programming for portfolio optimization now lies at the center of modern financial analysis. (More broadly, the relatively new field of *financial engineering* has arisen to focus on the application of OR techniques such as nonlinear programming to various finance problems, including portfolio optimization.) As illustrated by the application vignette in this section, this kind of application of nonlinear programming is having a tremendous impact in practice. Much research also continues to be done on the properties and application of both the above model and related nonlinear programming models to sophisticated kinds of portfolio analysis.[3]

## 12.2 GRAPHICAL ILLUSTRATION OF NONLINEAR PROGRAMMING PROBLEMS

When a nonlinear programming problem has just one or two variables, it can be represented graphically much like the Wyndor Glass Co. example for linear programming in Sec. 3.1. Because such a graphical representation gives considerable insight into the properties of optimal solutions for linear and nonlinear programming, let us look at a few examples. To highlight the difference between linear and nonlinear programming, we shall use some *nonlinear* variations of the Wyndor Glass Co. problem.

Figure 12.5 shows what happens to this problem if the only changes in the model shown in Sec. 3.1 are that both the second and the third functional constraints are replaced by the single nonlinear constraint $9x_1^2 + 5x_2^2 \leq 216$. Compare Fig. 12.5 with Fig. 3.3. The optimal solution still happens to be $(x_1, x_2) = (2, 6)$. Furthermore, it still lies on the boundary of the feasible region. However, it is *not* a corner-point feasible (CPF) solution. The optimal solution could have been a CPF solution with a different objective function (check $Z = 3x_1 + x_2$), but the fact that it need not be one means that we no longer have the tremendous simplification used in linear programming of limiting the search for an optimal solution to just the CPF solutions.

[3] Important recent research includes the following papers. B. I. Jacobs, K. N. Levy, and H. M. Markowitz: "Portfolio Optimization with Factors, Scenarios, and Realistic Short Positions," *Operations Research*, **53**(4): 586–599, July–Aug. 2005; A. F. Siegel and A. Woodgate: "Performance of Portfolios Optimized with Estimation Error," *Management Science*, **53**(6): 1005–1015, June 2007; H. Konno and T. Koshizuka: "Mean-Absolute Deviation Model," *IIE Transactions*, **37**(10): 893–900, Oct. 2005.



| | |
|---|---|
| Maximize | $Z = 3x_1 + 5x_2,$ |
| subject to | $x_1 \leq 4$ |
| | $9x_1^2 + 5x_2^2 \leq 216$ |
| and | $x_1 \geq 0, \quad x_2 \geq 0$ |

**FIGURE 12.5** The Wyndor Glass Co. example with the nonlinear constraint $9x_1^2 + 5x_2^2 \leq 216$ replacing the original second and third functional constraints.

Now suppose that the linear constraints of Sec. 3.1 are kept unchanged, but the objective function is made nonlinear. For example, if

$$Z = 126x_1 - 9x_1^2 + 182x_2 - 13x_2^2,$$

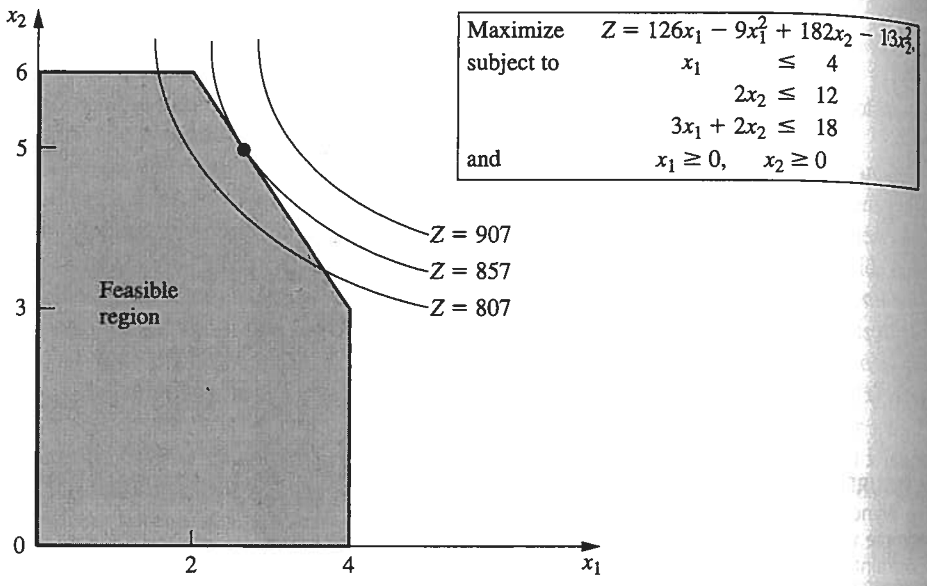
then the graphical representation in Fig. 12.6 indicates that the optimal solution is $x_1 = \frac{8}{3}$, $x_2 = 5$, which again lies on the boundary of the feasible region. (The value of $Z$ for this optimal solution is $Z = 857$, so Fig. 12.6 depicts the fact that the locus of all points with $Z = 857$ intersects the feasible region at just this one point, whereas the locus of points with any larger $Z$ does not intersect the feasible region at all.) On the other hand, if

$$Z = 54x_1 - 9x_1^2 + 78x_2 - 13x_2^2,$$

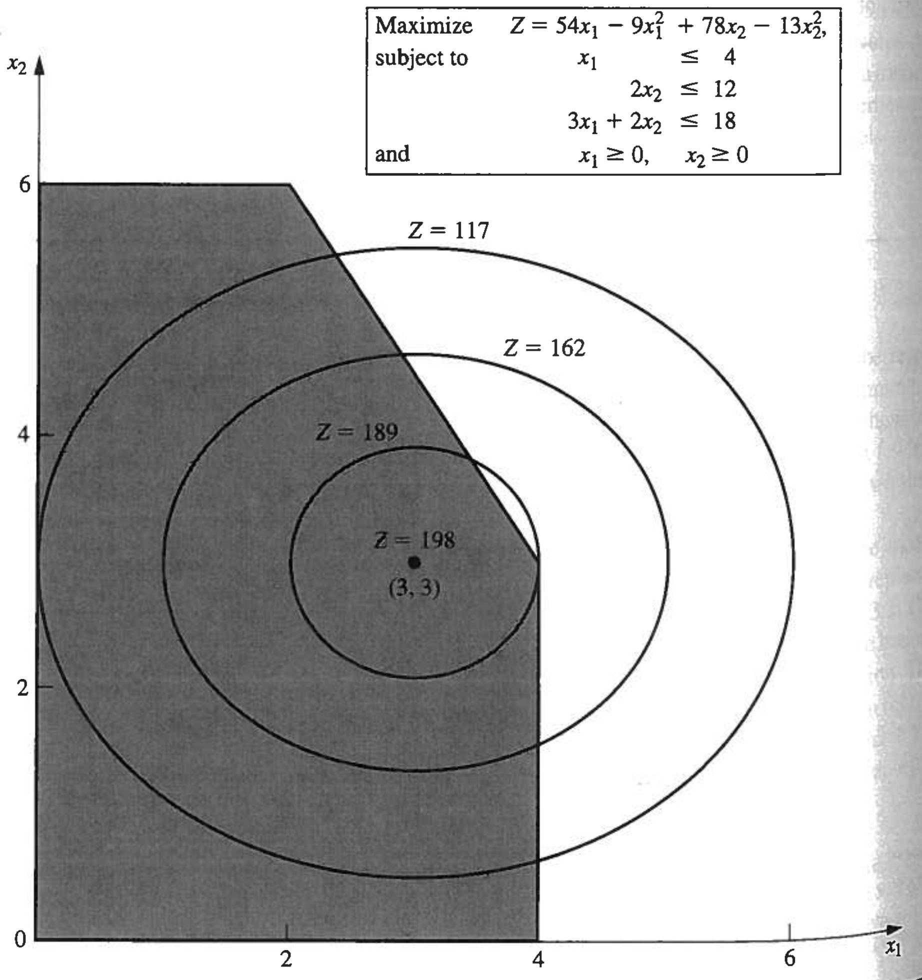
then Fig. 12.7 illustrates that the optimal solution turns out to be $(x_1, x_2) = (3, 3)$, which lies *inside* the boundary of the feasible region. (You can check that this solution is optimal by using calculus to derive it as the unconstrained global maximum; because it also satisfies the constraints, it must be optimal for the constrained problem.) Therefore, a general algorithm for solving similar problems needs to consider *all* solutions in the feasible region, not just those on the boundary.

Another complication that arises in nonlinear programming is that a *local* maximum need not be a *global* maximum (the overall optimal solution). For example, consider the function of a single variable plotted in Fig. 12.8. Over the interval $0 \leq x \leq 5$, this function has three local maxima—$x = 0$, $x = 2$, and $x = 4$—but only one of these—$x = 4$—is a *global maximum.* (Similarly, there are local minima at $x = 1$, 3, and 5, but only $x = 5$ is a *global minimum.*)

Nonlinear programming algorithms generally are unable to distinguish between a local maximum and a global maximum (except by finding another *better* local maximum). Therefore, it becomes crucial to know the conditions under which any local maximum is *guaranteed* to be a global maximum over the feasible region. You may recall from calculus that



■ **FIGURE 12.6**
The Wyndor Glass Co. example with the original feasible region but with the nonlinear objective function $Z = 126x_1 - 9x_1^2 + 182x_2 - 13x_2^2$ replacing the original objective function.



■ **FIGURE 12.7**
The Wyndor Glass Co. example with the original feasible region but with another nonlinear objective function, $Z = 54x_1 - 9x_1^2 + 78x_2 - 13x_2^2$, replacing the original objective function.

■ **FIGURE 12**
A function wit
maxima ($x$ = (
only $x = 4$ is a
maximum.

■ **FIGURE 12**
Examples of (
function and
function.

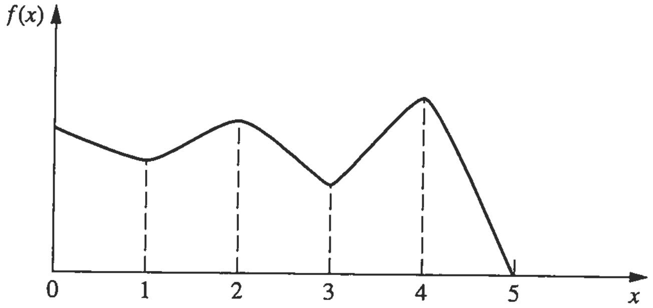


**FIGURE 12.8**
A function with several local maxima ($x = 0, 2, 4$), but only $x = 4$ is a global maximum.

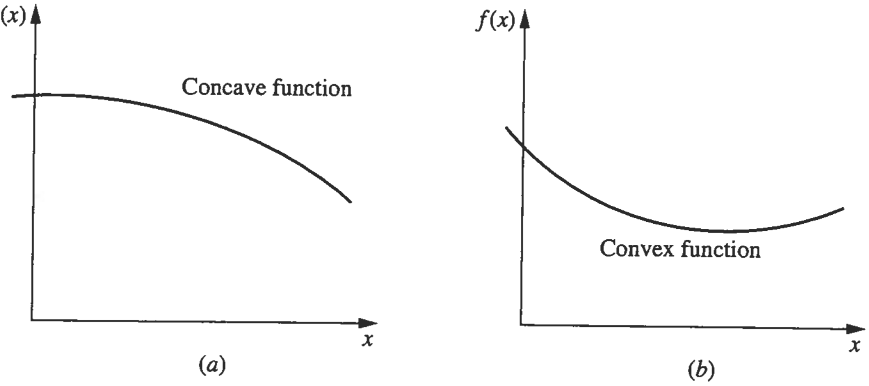


**FIGURE 12.9**
Examples of (*a*) a concave function and (*b*) a convex function.

when we maximize an ordinary (doubly differentiable) function of a single variable $f(x)$ without any constraints, this guarantee can be given when

$$\frac{\partial^2 f}{\partial x^2} \leq 0 \qquad \text{for all } x.$$

Such a function that is always "curving downward" (or not curving at all) is called a **concave** function.[4] Similarly, if $\leq$ is replaced by $\geq$, so that the function is always "curving upward" (or not curving at all), it is called a **convex** function.[5] (Thus, a *linear* function is both concave and convex.) See Fig. 12.9 for examples. Then note that Fig. 12.8 illustrates a function that is neither concave nor convex because it alternates between curving upward and curving downward.

Functions of multiple variables also can be characterized as concave or convex if they always curve downward or curve upward. These intuitive definitions are restated in precise terms, along with further elaboration on these concepts, in Appendix 2. (Concave and convex functions play a fundamental role in nonlinear programming, so if you are not very familiar with such functions, we suggest that you read further in Appendix 2.) Appendix 2 also provides a convenient test for checking whether a function of two variables is concave, convex, or neither.

Here is a convenient way of checking this for a function of more than two variables when the function consists of a *sum* of smaller functions of just one or two variables each.

[4] Concave functions sometimes are referred to as *concave downward.*
[5] Convex functions sometimes are referred to as *concave upward.*

If each smaller function is concave, then the overall function is concave. Similarly, the overall function is convex if each smaller function is convex.

To illustrate, consider the function

$$\begin{aligned} f(x_1, x_2, x_3) &= 4x_1 - x_1^2 - (x_2 - x_3)^2 \\ &= [4x_1 - x_1^2] + [-(x_2 - x_3)^2], \end{aligned}$$

which is the sum of the two smaller functions given in square brackets. The first smaller function $4x_1 - x_1^2$ is a function of the single variable $x_1$, so it can be found to be concave by noting that its second derivative is negative. The second smaller function $-(x_2 - x_3)^2$ is a function of just $x_2$ and $x_3$, so the test for functions of two variables given in Appendix 2 is applicable. In fact, Appendix 2 uses this particular function to illustrate the test and finds that the function is concave. Because both smaller functions are concave, the overall function $f(x_1, x_2, x_3)$ must be concave.

If a nonlinear programming problem has no constraints, the objective function being *concave* guarantees that a local maximum is a *global maximum.* (Similarly, the objective function being *convex* ensures that a local minimum is a *global minimum.*) If there are constraints, then one more condition will provide this guarantee, namely, that the *feasible region* is a *convex set.* For this reason, convex sets play a key role in nonlinear programming.

As discussed in Appendix 2, a **convex set** is simply a set of points such that, for each pair of points in the collection, the entire line segment joining these two points is also in the collection. Thus, the feasible region for the original Wyndor Glass Co. problem (see Fig. 12.6 or 12.7) is a convex set. In fact, the feasible region for *any* linear programming problem is a convex set. Similarly, the feasible region in Fig. 12.5 is a convex set.

In general, the feasible region for a nonlinear programming problem is a convex set whenever all the $g_i(\mathbf{x})$ [for the constraints $g_i(\mathbf{x}) \leq b_i$] are convex functions. For the example of Fig. 12.5, both of its $g_i(\mathbf{x})$ are convex functions, since $g_1(\mathbf{x}) = x_1$ (a linear function is automatically both concave and convex) and $g_2(\mathbf{x}) = 9x_1^2 + 5x_2^2$ (both $9x_1^2$ and $5x_2^2$ are convex functions so their sum is a convex function). These two convex $g_i(\mathbf{x})$ lead to the feasible region of Fig. 12.5 being a convex set.

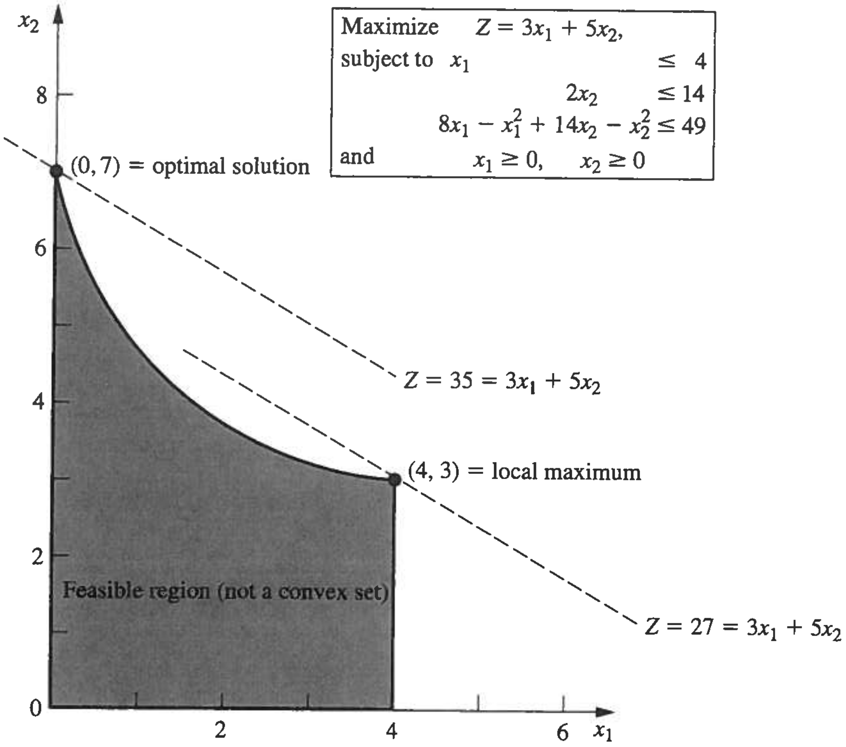
Now let's see what happens when just one of these $g_i(\mathbf{x})$ is a concave function instead. In particular, suppose that the only changes in the original Wyndor Glass Co. example are that the second and third functional constraints are replaced by $2x_2 \leq 14$ and $8x_1 - x_1^2 + 14x_2 - x_2^2 \leq 49$. Therefore, the new $g_3(\mathbf{x}) = 8x_1 - x_1^2 + 14x_2 - x_2^2$, which is a concave function since both $8x_1 - x_1^2$ and $14x_2 - x_2^2$ are concave functions. The new feasible region shown in Fig. 12.10 is *not* a convex set. Why? Because this feasible region contains pairs of points, for example, (0, 7) and (4, 3), such that part of the line segment joining these two points is not in the feasible region. Consequently, we cannot guarantee that a local maximum is a global maximum. In fact, this example has two local maxima, (0, 7) and (4, 3), but only (0, 7) is a global maximum.

Therefore, to guarantee that a local maximum is a global maximum for a nonlinear programming problem with constraints $g_i(\mathbf{x}) \leq b_i$ $(i = 1, 2, \ldots, m)$ and $\mathbf{x} \geq \mathbf{0}$, the objective function $f(\mathbf{x})$ must be a *concave* function and each $g_i(\mathbf{x})$ must be a *convex* function. Such a problem is called a *convex programming problem,* which is one of the key types of nonlinear programming problems discussed in Sec. 12.3.

## 12.3 TYPES OF NONLINEAR PROGRAMMING PROBLEMS

Nonlinear programming problems come in many different shapes and forms. Unlike the simplex method for linear programming, no single algorithm can solve all these different types of problems. Instead, algorithms have been developed for various individual

FIGURE 1
The Wyndor
example with
nonlinear con
$8x_1 - x_1^2 + 14$
replacing the
and third fun
constraints.



$$\begin{array}{ll} \text{Maximize} & Z = 3x_1 + 5x_2, \\ \text{subject to} & x_1 \le 4 \\ & 2x_2 \le 14 \\ & 8x_1 - x_1^2 + 14x_2 - x_2^2 \le 49 \\ \text{and} & x_1 \ge 0, \quad x_2 \ge 0 \end{array}$$

**FIGURE 12.10**
The Wyndor Glass Co. example with $2x_2 \le 14$ and a nonlinear constraint, $8x_1 - x_1^2 + 14x_2 - x_2^2 \le 49$, replacing the original second and third functional constraints.

*classes* (special types) of nonlinear programming problems. The most important classes are introduced briefly in this section. The subsequent sections then describe how some problems of these types can be solved. To simplify the discussion, we will assume throughout that the problems have been formulated (or reformulated) in the general form presented at the beginning of the chapter.

## Unconstrained Optimization

Unconstrained optimization problems have *no* constraints, so the objective is simply to

$$\text{Maximize} \quad f(\mathbf{x})$$

over *all* values of $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. As reviewed in Appendix 3, the *necessary* condition that a particular solution $\mathbf{x} = \mathbf{x}^*$ be optimal when $f(\mathbf{x})$ is a differentiable function is

$$\frac{\partial f}{\partial x_j} = 0 \qquad \text{at } \mathbf{x} = \mathbf{x}^*, \text{ for } j = 1, 2, \ldots, n.$$

When $f(\mathbf{x})$ is a *concave* function, this condition also is *sufficient,* so then solving for $\mathbf{x}^*$ reduces to solving the system of $n$ equations obtained by setting the $n$ partial derivatives equal to zero. Unfortunately, for *nonlinear* functions $f(\mathbf{x})$, these equations often are going to be *nonlinear* as well, in which case you are unlikely to be able to solve analytically for their simultaneous solution. What then? Sections 12.4 and 12.5 describe *algorithmic search procedures* for finding $\mathbf{x}^*$, first for $n = 1$ and then for $n > 1$. These procedures also play an important role in solving many of the problem types described next, where there are constraints. The reason is that many algorithms for *constrained* problems are designed so that they can focus on an *unconstrained* version of the problem during a portion of each iteration.

When a variable $x_j$ does have a nonnegativity constraint $x_j \geq 0$, the preceding necessary and (perhaps) sufficient condition changes slightly to

$$\frac{\partial f}{\partial x_j} \begin{cases} \leq 0 & \text{at } \mathbf{x} = \mathbf{x}^*, \quad \text{if } x_j^* = 0 \\ = 0 & \text{at } \mathbf{x} = \mathbf{x}^*, \quad \text{if } x_j^* > 0 \end{cases}$$

for each such $j$. This condition is illustrated in Fig. 12.11, where the optimal solution for a problem with a single variable is at $x = 0$ even though the derivative there is negative rather than zero. Because this example has a concave function to be maximized subject to a nonnegativity constraint, having the derivative less than or equal to 0 at $x = 0$ is both a necessary and sufficient condition for $x = 0$ to be optimal.

A problem that has some nonnegativity constraints but no functional constraints is one special case ($m = 0$) of the next class of problems.

### Linearly Constrained Optimization

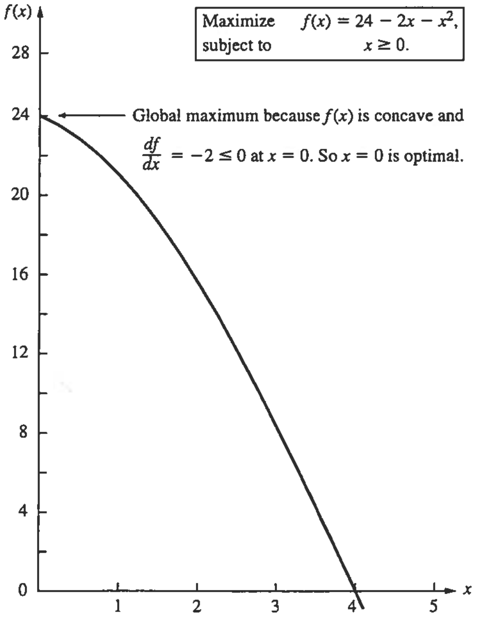
Linearly constrained optimization problems are characterized by constraints that completely fit linear programming, so that *all* the $g_i(\mathbf{x})$ constraint functions are linear, but the objective function $f(\mathbf{x})$ is nonlinear. The problem is considerably simplified by having just one nonlinear function to take into account, along with a linear programming feasible region. A number of special algorithms based upon *extending* the simplex method to consider the nonlinear objective function have been developed.

One important special case, which we consider next, is quadratic programming.

### Quadratic Programming

Quadratic programming problems again have linear constraints, but now the objective function $f(\mathbf{x})$ must be *quadratic*. Thus, the only difference between such a problem and a linear programming problem is that some of the terms in the objective function involve the *square* of a variable or the *product* of two variables.

**FIGURE 12.11**
An example that illustrates how an optimal solution can lie at a point where a derivative is negative instead of zero, because that point lies at the boundary of a nonnegativity constraint.



Many algorithms have been developed for this case under the additional assumption that $f(\mathbf{x})$ is a concave function. Section 12.7 presents an algorithm that involves a direct extension of the simplex method.

Quadratic programming is very important, partially because such formulations arise naturally in many applications. For example, the problem of portfolio selection with risky securities described in Sec. 12.1 fits into this format. However, another major reason for its importance is that a common approach to solving general linearly constrained optimization problems is to solve a sequence of quadratic programming approximations.

### Convex Programming

*Convex programming* covers a broad class of problems that actually encompasses as special cases all the preceding types when $f(\mathbf{x})$ is a concave function to be maximized. Continuing to assume the general problem form (including maximization) presented at the beginning of the chapter, the assumptions are that

1. $f(\mathbf{x})$ is a concave function.
2. Each $g_i(\mathbf{x})$ is a convex function.

As discussed at the end of Sec. 12.2, these assumptions are enough to ensure that a local maximum is a global maximum. (If the objective were to *minimize* $f(\mathbf{x})$ instead, subject to either $g_i(\mathbf{x}) \leq b_i$ or $-g_i(\mathbf{x}) \geq b_i$ for $i = 1, 2, \ldots, m$, the first assumption would change to requiring that $f(\mathbf{x})$ must be a *convex* function, since this is what is needed to ensure that a local minimum is a global minimum.) You will see in Sec. 12.6 that the necessary and sufficient conditions for such an optimal solution are a natural generalization of the conditions just given for *unconstrained optimization* and its extension to include *nonnegativity constraints.* Section 12.9 then describes algorithmic approaches to solving convex programming problems.

### Separable Programming

*Separable programming* is a special case of convex programming, where the one additional assumption is that

3. All the $f(\mathbf{x})$ and $g_i(\mathbf{x})$ functions are separable functions.

A **separable function** is a function where *each term* involves just a *single variable,* so that the function is separable into a sum of functions of individual variables. For example, if $f(\mathbf{x})$ is a separable function, it can be expressed as

$$f(\mathbf{x}) = \sum_{j=1}^{n} f_j(x_j),$$

where each $f_j(x_j)$ function includes only the terms involving just $x_j$. In the terminology of linear programming (see Sec. 3.3), separable programming problems satisfy the assumption of additivity but violate the assumption of proportionality when any of the $f_j(x_j)$ functions are nonlinear functions.

To illustrate, the objective function considered in Fig. 12.6,

$$f(x_1, x_2) = 126x_1 - 9x_1^2 + 182x_2 - 13x_2^2$$

is a separable function because it can be expressed as

$$f(x_1, x_2) = f_1(x_1) + f_2(x_2)$$

where $f_1(x_1) = 126x_1 - 9x_1^2$ and $f_2(x_2) = 182x_2 - 13x_2^2$ are each a function of a single variable—$x_1$ and $x_2$, respectively. By the same reasoning, you can verify that the objective function considered in Fig. 12.7 also is a separable function.

It is important to distinguish separable programming problems from other convex programming problems, because any such problem can be closely approximated by a linear programming problem so that the extremely efficient simplex method can be used. This approach is described in Sec. 12.8. (For simplicity, we focus there on the *linearly constrained* case where the special approach is needed only on the objective function.)

### Nonconvex Programming

*Nonconvex programming* encompasses all nonlinear programming problems that do not satisfy the assumptions of convex programming. Now, even if you are successful in finding a *local maximum*, there is no assurance that it also will be a *global maximum*. Therefore, there is no algorithm that will find an optimal solution for all such problems. However, there do exist some algorithms that are relatively well suited for exploring various parts of the feasible region and perhaps finding a global maximum in the process. We describe this approach in Sec. 12.10. Section 12.10 also will introduce two global optimizers (available with LINGO and MPL) for finding an optimal solution for nonconvex programming problems of moderate size, as well as a search procedure that generally will find a near-optimal solution for rather large problems.

Certain specific types of nonconvex programming problems can be solved without great difficulty by special methods. Two especially important such types are discussed briefly next.

### Geometric Programming

When we apply nonlinear programming to engineering design problems, as well as certain economics and statistics problems, the objective function and the constraint functions frequently take the form

$$g(\mathbf{x}) = \sum_{i=1}^{N} c_i P_i(\mathbf{x}),$$

where

$$P_i(\mathbf{x}) = x_1^{a_{i1}} x_2^{a_{i2}} \cdots x_n^{a_{in}}, \qquad \text{for } i = 1, 2, \ldots, N.$$

In such cases, the $c_i$ and $a_{ij}$ typically represent physical constants, and the $x_j$ are design variables. These functions generally are neither convex nor concave, so the techniques of convex programming cannot be applied directly to these *geometric programming* problems. However, there is one important case where the problem can be transformed to an equivalent convex programming problem. This case is where *all* the $c_i$ coefficients in each function are strictly positive, so that the functions are *generalized positive polynomials* now called **posynomials** and the objective function is to be minimized. The equivalent convex programming problem with decision variables $y_1, y_2, \ldots, y_n$ is then obtained by setting

$$x_j = e^{y_j}, \qquad \text{for } j = 1, 2, \ldots, n$$

throughout the original model, so now a convex programming algorithm can be applied. Alternative solution procedures also have been developed for solving these *posynomial programming* problems, as well as for geometric programming problems of other types.

### Fractional Programming

Suppose that the objective function is in the form of a *fraction*, i.e., the ratio of two functions.

$$\text{Maximize} \qquad f(\mathbf{x}) = \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})}.$$

Such *fractional programming* problems arise, e.g., when one is maximizing the ratio of output to person-hours expended (productivity), or profit to capital expended (rate of return), or expected value to standard deviation of some measure of performance for an investment portfolio (return/risk). Some special solution procedures have been developed for certain forms of $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$.

When it can be done, the most straightforward approach to solving a fractional programming problem is to transform it to an equivalent problem of a standard type for which effective solution procedures already are available. To illustrate, suppose that $f(\mathbf{x})$ is of the *linear fractional programming* form

$$f(\mathbf{x}) = \frac{\mathbf{cx} + c_0}{\mathbf{dx} + d_0},$$

where $\mathbf{c}$ and $\mathbf{d}$ are row vectors, $\mathbf{x}$ is a column vector, and $c_0$ and $d_0$ are scalars. Also assume that the constraint functions $g_i(\mathbf{x})$ are linear, so that the constraints in matrix form are $\mathbf{Ax} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$.

Under mild additional assumptions, we can transform the problem to an equivalent *linear programming* problem by letting

$$\mathbf{y} = \frac{\mathbf{x}}{\mathbf{dx} + d_0} \qquad \text{and} \qquad t = \frac{1}{\mathbf{dx} + d_0},$$

so that $\mathbf{x} = \mathbf{y}/t$. This result yields

$$\text{Maximize} \qquad Z = \mathbf{cy} + c_0 t,$$

subject to

$$\mathbf{Ay} - \mathbf{b}t \le \mathbf{0},$$
$$\mathbf{dy} + d_0 t = 1,$$

and

$$\mathbf{y} \ge \mathbf{0}, \qquad t \ge 0,$$

which can be solved by the simplex method. More generally, the same kind of transformation can be used to convert a fractional programming problem with concave $f_1(\mathbf{x})$, convex $f_2(\mathbf{x})$, and convex $g_i(\mathbf{x})$ to an equivalent convex programming problem.

### The Complementarity Problem

When we deal with quadratic programming in Sec. 12.7, you will see one example of how solving certain nonlinear programming problems can be reduced to solving the complementarity problem. Given variables $w_1, w_2, \ldots, w_p$ and $z_1, z_2, \ldots, z_p$, the **complementarity problem** is to find a *feasible* solution for the set of constraints

$$\mathbf{w} = F(\mathbf{z}), \qquad \mathbf{w} \ge \mathbf{0}, \qquad \mathbf{z} \ge \mathbf{0}$$

that also satisfies the **complementarity contraint**

$$\mathbf{w}^T\mathbf{z} = 0.$$

Here, $\mathbf{w}$ and $\mathbf{z}$ are column vectors, $F$ is a given vector-valued function, and the superscript $T$ denotes the transpose (see Appendix 4). The problem has no objective function, so technically it is not a full-fledged nonlinear programming problem. It is called the complementarity problem because of the complementary relationships that either

$$w_i = 0 \qquad \text{or} \qquad z_i = 0 \qquad \text{(or both)} \qquad \text{for each } i = 1, 2, \ldots, p.$$

An important special case is the **linear complementarity problem,** where

$$F(\mathbf{z}) = \mathbf{q} + \mathbf{Mz},$$

where **q** is a given column vector and **M** is a given $p \times p$ matrix. Efficient algorithms have been developed for solving this problem under suitable assumptions[6] about the properties of the matrix **M.** One type involves pivoting from one basic feasible (BF) solution to the next, much like the simplex method for linear programming.

In addition to having applications in nonlinear programming, complementarity problems have applications in game theory, economic equilibrium problems, and engineering equilibrium problems.

## 12.4 ONE-VARIABLE UNCONSTRAINED OPTIMIZATION

We now begin discussing how to solve some of the types of problems just described by considering the simplest case—*unconstrained optimization* with just a single variable $x$ ($n = 1$), where the differentiable function $f(x)$ to be maximized is *concave.*[7] Thus, the *necessary and sufficient condition* for a particular solution $x = x^*$ to be optimal (a global maximum) is

$$\frac{df}{dx} = 0 \qquad \text{at } x = x^*,$$

as depicted in Fig. 12.12. If this equation can be solved directly for $x^*$, you are done. However, if $f(x)$ is not a particularly simple function, so the derivative is not just a linear or quadratic function, you may not be able to solve the equation *analytically.* If not, a number of *search procedures* are available for solving the problem *numerically.*

The approach with any of these search procedures is to find a sequence of *trial solutions* that leads toward an optimal solution. At each iteration, you begin at the current trial solution to conduct a systematic search that culminates by identifying a new *improved*

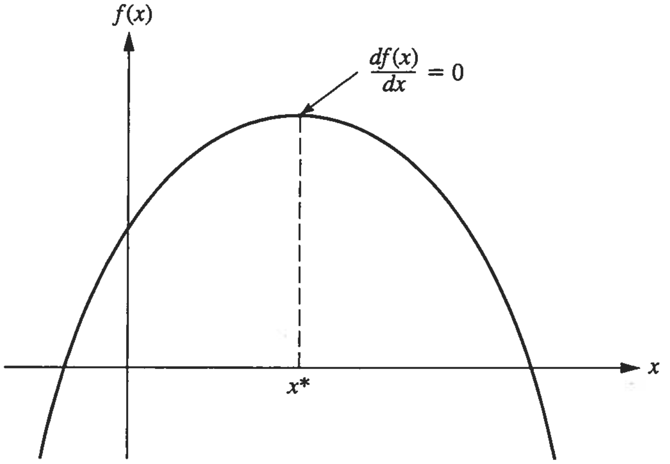


**FIGURE 12.12**
The one-variable unconstrained optimization problem when the function is concave.

[6] See R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem,* Academic Press, Boston, 1992.

[7] See the beginning of Appendix 3 for a review of the corresponding case when $f(x)$ is not concave.

trial solution. The procedure is continued until the trial solutions have converged to an optimal solution, assuming that one exists.

We now will describe two common search procedures. The first one (the *bisection method*) was chosen because it is such an intuitive and straightforward procedure. The second one (*Newton's method*) is included because it plays a fundamental role in nonlinear programming in general.

## The Bisection Method

This search procedure always can be applied when $f(x)$ is concave (so that the second derivative is negative or zero for all $x$) as depicted in Fig. 12.12. It also can be used for certain other functions as well. In particular, if $x^*$ denotes the optimal solution, all that is needed[8] is that

$$\frac{df(x)}{dx} > 0 \quad \text{if } x < x^*,$$

$$\frac{df(x)}{dx} = 0 \quad \text{if } x = x^*,$$

$$\frac{df(x)}{dx} < 0 \quad \text{if } x > x^*.$$

These conditions automatically hold when $f(x)$ is concave, but they also can hold when the second derivative is positive for some (but not all) values of $x$.

The idea behind the bisection method is a very intuitive one, namely, that whether the slope (derivative) is positive or negative at a trial solution definitely indicates whether improvement lies immediately to the right or left, respectively. Thus, if the derivative evaluated at a particular value of $x$ is *positive,* then $x^*$ must be larger than this $x$ (see Fig. 12.12), so this $x$ becomes a *lower bound* on the trial solutions that need to be considered thereafter. Conversely, if the derivative is *negative,* then $x^*$ must be *smaller* than this $x$, so $x$ would become an *upper bound.* Therefore, after both types of bounds have been identified, each new trial solution selected between the current bounds provides a new tighter bound of one type, thereby narrowing the search further. As long as a reasonable rule is used to select each trial solution in this way, the resulting *sequence* of trial solutions must *converge* to $x^*$. In practice, this means continuing the sequence until the distance between the bounds is sufficiently small that the next trial solution must be within a prespecified *error tolerance* of $x^*$.

This entire process is summarized next, given the notation

$x'$ = current trial solution,

$\underline{x}$ = current lower bound on $x^*$,

$\bar{x}$ = current upper bound on $x^*$,

$\epsilon$ = error tolerance for $x^*$.

Although there are several reasonable rules for selecting each new trial solution, the one used in the bisection method is the **midpoint rule** (traditionally called the *Bolzano search plan*), which says simply to select the midpoint between the two current bounds.

[8] Another possibility is that the graph of $f(x)$ is flat at the top so that $x$ is optimal over some interval $[a, b]$. In this case, the procedure still will converge to one of these optimal solutions as long as the derivative is positive for $x < a$ and negative for $x > b$.

### Summary of the Bisection Method

*Initialization:* Select $\epsilon$. Find an initial $\underline{x}$ and $\overline{x}$ by inspection (or by respectively finding any value of $x$ at which the derivative is positive and then negative). Select an initial trial solution

$$x' = \frac{\underline{x} + \overline{x}}{2}.$$

*Iteration:*

1. Evaluate $\dfrac{df(x)}{dx}$ at $x = x'$.
2. If $\dfrac{df(x)}{dx} \geq 0$, reset $\underline{x} = x'$.
3. If $\dfrac{df(x)}{dx} \leq 0$, reset $\overline{x} = x'$.
4. Select a new $x' = \dfrac{\underline{x} + \overline{x}}{2}$.

*Stopping rule:* If $\overline{x} - \underline{x} \leq 2\epsilon$, so that the new $x'$ must be within $\epsilon$ of $x^*$, stop. Otherwise, perform another iteration.

We shall now illustrate the bisection method by applying it to the following example.

**Example.** Suppose that the function to be maximized is

$$f(x) = 12x - 3x^4 - 2x^6,$$

as plotted in Fig. 12.13. Its first two derivatives are

$$\frac{df(x)}{dx} = 12(1 - x^3 - x^5),$$

$$\frac{d^2f(x)}{dx^2} = -12(3x^2 + 5x^4).$$

**FIGURE 12.13**
Example for the bisection method.

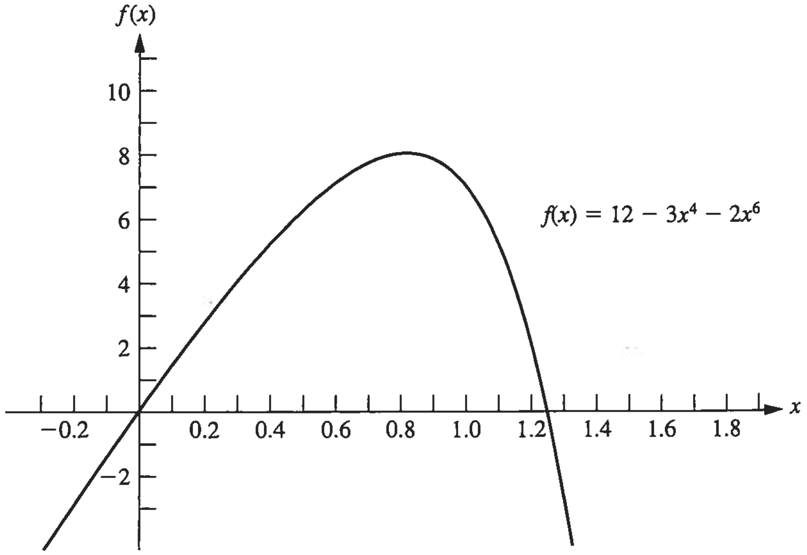


**TABLE 12.1** Application of the bisection method to the example

| Iteration | $\frac{df(x)}{dx}$ | $\underline{x}$ | $\bar{x}$ | New $x'$ | $f(x')$ |
|---|---|---|---|---|---|
| 0 | | 0 | 2 | 1 | 7.0000 |
| 1 | −12 | 0 | 1 | 0.5 | 5.7812 |
| 2 | +10.12 | 0.5 | 1 | 0.75 | 7.6948 |
| 3 | +4.09 | 0.75 | 1 | 0.875 | 7.8439 |
| 4 | −2.19 | 0.75 | 0.875 | 0.8125 | 7.8672 |
| 5 | +1.31 | 0.8125 | 0.875 | 0.84375 | 7.8829 |
| 6 | −0.34 | 0.8125 | 0.84375 | 0.828125 | 7.8815 |
| 7 | +0.51 | 0.828125 | 0.84375 | 0.8359375 | 7.8839 |
| Stop | | | | | |

Because the second derivative is nonpositive everywhere, $f(x)$ is a concave function, so the bisection method can be safely applied to find its global maximum (assuming a global maximum exists).

A quick inspection of this function (without even constructing its graph as shown in Fig. 12.13) indicates that $f(x)$ is positive for small positive values of $x$, but it is negative for $x < 0$ or $x > 2$. Therefore, $\underline{x} = 0$ and $\bar{x} = 2$ can be used as the initial bounds, with their midpoint, $x' = 1$, as the initial trial solution. Let $\epsilon = 0.01$ be the error tolerance for $x^*$ in the stopping rule, so the final $(\bar{x} - \underline{x}) \leq 0.02$ with the final $x'$ at the midpoint.

Applying the bisection method then yields the sequence of results shown in Table 12.1. [This table includes both the function and derivative values for your information, where the derivative is evaluated at the trial solution generated at the *preceding* iteration. However, note that the algorithm actually doesn't need to calculate $f(x')$ at all and that it only needs to calculate the derivative far enough to determine its sign.] The conclusion is that

$$x^* \approx 0.836,$$
$$0.828125 < x^* < 0.84375.$$

Your IOR Tutorial includes an interactive procedure for executing the bisection method.

### Newton's Method

Although the bisection method is an intuitive and straightforward procedure, it has the disadvantage of converging relatively slowly toward an optimal solution. Each iteration only decreases the difference between the bounds by one-half. Therefore, even with the fairly simple function being considered in Table 12.1, seven iterations were required to reduce the error tolerance for $x^*$ to less than 0.01. Another seven iterations would be needed to reduce this error tolerance to less than 0.0001.

The basic reason for this slow convergence is that the only information about $f(x)$ being used is the value of the first derivative $f'(x)$ at the respective trial values of $x$. Additional helpful information can be obtained by considering the second derivative $f''(x)$ as well. This is what *Newton's method*[9] does.

[9] This method is due to the great 17th-century mathematician and physicist, Sir Isaac Newton. While a young student at the University of Cambridge (England), Newton took advantage of the university being closed for two years (due to the bubonic plague that devastated Europe in 1664–65) to discover the law of universal gravitation and invent calculus (among other achievements). His development of calculus led to this method.

The basic idea behind Newton's method is to approximate $f(x)$ within the neighborhood of the current trial solution by a quadratic function and then to maximize (or minimize) the approximate function exactly to obtain the new trial solution to start the next iteration. (This idea of working with a **quadratic approximation** of the objective function has since been made a key feature of many algorithms for more general kinds of nonlinear programming problems.) This approximating quadratic function is obtained by truncating the Taylor series after the second derivative term. In particular, by letting $x_{i+1}$ be the trial solution generated at iteration $i$ to start iteration $i + 1$ (so $x_1$ is the initial trial solution provided by the user to begin iteration 1), the truncated Taylor series for $x_{i+1}$ is

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2}(x_{i+1} - x_i)^2.$$

Having fixed $x_i$ at the beginning of iteration $i$, note that $f(x_i), f'(x_i)$, and $f''(x_i)$ also are fixed constants in this approximating function on the right. Thus, this approximating function is just a quadratic function of $x_{i+1}$. Furthermore, this quadratic function is such a good approximation of $f(x_{i+1})$ in the neighborhood of $x_i$ that their values and their first and second derivatives are exactly the same when $x_{i+1} = x_i$.

This quadratic function now can be maximized in the usual way by setting its first derivative to zero and solving for $x_{i+1}$. (Remember that we are assuming that $f(x)$ is concave, which implies that this quadratic function is concave, so the solution when setting the first derivative to zero will be a global maximum.) This first derivative is

$$f'(x_{i+1}) \approx f'(x_i) + f''(x_i)(x_{i+1} - x_i)$$

since $x_i$, $f(x_i)$, $f'(x_i)$, and $f''(x_i)$ are constants. Setting the first derivative on the right to zero yields

$$f'(x_{i+1}) + f''(x_i)(x_{i+1} - x_i) = 0,$$

which directly leads algebraically to the solution,

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}.$$

This is the key formula that is used at each iteration $i$ to calculate the next trial solution $x_{i+1}$ after obtaining the trial solution $x_i$ to begin iteration $i$ and then calculating the first and second derivatives at $x_i$. (The same formula is used when minimizing a convex function.)

Iterations generating new trial solutions in this way would continue until these solutions have essentially converged. One criterion for convergence is that $|x_{i+1} - x_i|$ has become sufficiently small. Another is that $f'(x)$ is sufficiently close to zero. Still another is that $|f(x_{i+1}) - f(x_i)|$ is sufficiently small. Choosing the first criterion, define $\epsilon$ as the value such that the algorithm is stopped when $|x_{i+1} - x_i| \leq \epsilon$.

Here is a complete description of the algorithm.

### Summary of Newton's Method

*Initialization:* Select $\epsilon$. Find an initial trial solution $x_i$ by inspection. Set $i = 1$.

*Iteration i:*

**1.** Calculate $f'(x_i)$ and $f''(x_i)$. [Calculating $f(x_i)$ is optional.]

**2.** Set $x_{i+1} = x_i - \dfrac{f'(x_i)}{f''(x_i)}$.

*Stopping Rule:* If $|x_{i+1} - x_i| \leq \epsilon$, stop; $x_{i+1}$ is essentially the optimal solution. Otherwise, reset $i = i + 1$ and perform another iteration.

**TABLE 12.2** Application of Newton's method to the example

| Iteration $i$ | $x_i$ | $f(x_i)$ | $f'(x_i)$ | $f''(x_i)$ | $x_{i+1}$ |
|---|---|---|---|---|---|
| 1 | 1 | 7 | −12 | −96 | 0.875 |
| 2 | 0.875 | 7.8439 | −2.1940 | −62.733 | 0.84003 |
| 3 | 0.84003 | 7.8838 | −0.1325 | −55.279 | 0.83763 |
| 4 | 0.83763 | 7.8839 | −0.0006 | −54.790 | 0.83762 |

**Example.** We now will apply Newton's method to the same example used for the bisection method. As depicted in Fig. 12.13, the function to be maximized is

$$f(x) = 12x - 3x^4 - 2x^6.$$

Thus, the formula for calculating the new trial solution ($x_{i+1}$) from the current one ($x_i$) is

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} = x_i - \frac{12(1 - x^3 - x^5)}{-12(3x^2 + 5x^4)} = x_i + \frac{1 - x^3 - x^5}{3x^2 + 5x^4}.$$

After selecting $\epsilon = 0.00001$ and choosing $x_1 = 1$ as the initial trial solution, Table 12.2 shows the results from applying Newton's method to this example. After just four iterations, this method has converged to $x = 0.83762$ as the optimal solution with a very high degree of precision.

A comparison of this table with Table 12.1 illustrates how much more rapidly Newton's method converges than the bisection method. Nearly 20 iterations would be required for the bisection method to converge with the same degree of precision that Newton's method achieved after only four iterations.

Although this rapid convergence is fairly typical of Newton's method, its performance does vary from problem to problem. Since the method is based on using a quadratic approximation of $f(x)$, its performance is affected by the degree of accuracy of the approximation.

## 12.5 MULTIVARIABLE UNCONSTRAINED OPTIMIZATION

Now consider the problem of maximizing a *concave* function $f(\mathbf{x})$ of *multiple* variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ when there are no constraints on the feasible values. Suppose again that the necessary and sufficient condition for optimality, given by the system of equations obtained by setting the respective partial derivatives equal to zero (see Sec. 12.3), cannot be solved analytically, so that a numerical search procedure must be used.

As for the one-variable case, a number of search procedures are available for solving such a problem numerically. One of these (the *gradient search procedure*) is an especially important one because it identifies and uses the direction of movement from the current trial solution that maximizes the rate at which $f(\mathbf{x})$ is increased. This is one of the key ideas of nonlinear programming. Adaptations of this same idea to take constraints into account are a central feature of many algorithms for *constrained* optimization as well.

After discussing this procedure in some detail, we will briefly describe how Newton's method is extended to the multivariable case.

### The Gradient Search Procedure

In Sec. 12.4, the value of the ordinary derivative was used by the bisection method to select one of just two possible directions (increase $x$ or decrease $x$) in which to move from the current trial solution to the next one. The goal was to reach a point eventually where this derivative is (essentially) 0. Now, there are *innumerable* possible directions in which to move; they correspond to the possible *proportional rates* at which the respective variables can be changed. The goal is to reach a point eventually where all the partial derivatives are (essentially) 0. Therefore, a natural approach is to use the values of the *partial* derivatives to select the specific direction in which to move. This selection involves using the gradient of the objective function, as described next.

Because the objective function $f(\mathbf{x})$ is assumed to be differentiable, it possesses a gradient, denoted by $\nabla f(\mathbf{x})$, at each point $\mathbf{x}$. In particular, the **gradient** at a specific point $\mathbf{x} = \mathbf{x}'$ is the *vector* whose elements are the respective *partial derivatives* evaluated at $\mathbf{x} = \mathbf{x}'$, so that

$$\nabla f(\mathbf{x}') = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right) \qquad \text{at } \mathbf{x} = \mathbf{x}'.$$

The significance of the gradient is that the (infinitesimal) change in $\mathbf{x}$ that *maximizes* the rate at which $f(\mathbf{x})$ increases is the change that is *proportional* to $\nabla f(\mathbf{x})$. To express this idea geometrically, the "direction" of the gradient $\nabla f(\mathbf{x}')$ is interpreted as the *direction* of the directed line segment (arrow) from the origin $(0, 0, \ldots, 0)$ to the point $(\partial f/\partial x_1, \partial f/\partial x_2, \ldots, \partial f/\partial x_n)$, where $\partial f/\partial x_j$ is evaluated at $x_j = x_j'$. Therefore, it may be said that the rate at which $f(\mathbf{x})$ increases is maximized if (infinitesimal) changes in $\mathbf{x}$ are in the *direction* of the gradient $\nabla f(\mathbf{x})$. Because the objective is to find the feasible solution maximizing $f(\mathbf{x})$, it would seem expedient to attempt to move in the direction of the gradient as much as possible.

Because the current problem has no constraints, this interpretation of the gradient suggests that an efficient search procedure should keep moving in the direction of the gradient until it (essentially) reaches an optimal solution $\mathbf{x}^*$, where $\nabla f(\mathbf{x}^*) = \mathbf{0}$. However, normally it would not be practical to change $\mathbf{x}$ *continuously* in the direction of $\nabla f(\mathbf{x})$, because this series of changes would require continuously *reevaluating* the $\partial f/\partial x_j$ and changing the direction of the path. Therefore, a better approach is to keep moving in a *fixed* direction from the current trial solution, not stopping until $f(\mathbf{x})$ stops increasing. This stopping point would be the next trial solution, so the gradient then would be recalculated to determine the new direction in which to move. With this approach, each iteration involves changing the current trial solution $\mathbf{x}'$ as follows:

$$\text{Reset} \qquad \mathbf{x}' = \mathbf{x}' + t^* \, \nabla f(\mathbf{x}'),$$

where $t^*$ is the positive value of $t$ that *maximizes* $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$; that is,

$$f(\mathbf{x}' + t^*\,\nabla f(\mathbf{x}')) = \max_{t \geq 0} f(\mathbf{x}' + t\,\nabla f(\mathbf{x}')).$$

[Note that $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ is simply $f(\mathbf{x})$ where

$$x_j = x_j' + t\left(\frac{\partial f}{\partial x_j}\right)_{\mathbf{x}=\mathbf{x}'}, \qquad \text{for } j = 1, 2, \ldots, n,$$

and that these expressions for the $x_j$ involve only constants and $t$, so $f(\mathbf{x})$ becomes a function of just the single variable $t$.] The iterations of this gradient search procedure continue until $\nabla f(\mathbf{x}) = 0$ within a small tolerance $\epsilon$, that is, until

$$\left|\frac{\partial f}{\partial x_j}\right| \leq \epsilon \qquad \text{for } j = 1, 2, \ldots, n.$$[10]

[10] This stopping rule generally will provide a solution $\mathbf{x}$ that is close to an optimal solution $\mathbf{x}^*$, with a value of $f(\mathbf{x})$ that is very close to $f(\mathbf{x}^*)$. However, this cannot be guaranteed, since it is possible that the function maintains a very small positive slope ($\leq \epsilon$) over a great distance from $\mathbf{x}$ to $\mathbf{x}^*$.

An analogy may help to clarify this procedure. Suppose that you need to climb to the top of a hill. You are nearsighted, so you cannot see the top of the hill in order to walk directly in that direction. However, when you stand still, you can see the ground around your feet well enough to determine the direction in which the hill is sloping upward most sharply. You are able to walk in a straight line. While walking, you also are able to tell when you stop climbing (zero slope in your direction). Assuming that the hill is *concave,* you now can use the *gradient search procedure* for climbing to the top efficiently. This problem is a *two-variable problem,* where $(x_1, x_2)$ represents the coordinates (ignoring height) of your current location. The function $f(x_1, x_2)$ gives the height of the hill at $(x_1, x_2)$. You start each iteration at your current location (current trial solution) by determining the direction [in the $(x_1, x_2)$ coordinate system] in which the hill is sloping upward most sharply (the direction of the gradient) at this point. You then begin walking in this fixed direction and continue as long as you still are climbing. You eventually stop at a new trial location (solution) when the hill becomes level in your direction, at which point you prepare to do another iteration in another direction. You continue these iterations, following a zigzag path up the hill, until you reach a trial location where the slope is essentially zero in all directions. Under the assumption that the hill $[f(x_1, x_2)]$ is concave, you must then be essentially at the top of the hill.

The most difficult part of the gradient search procedure usually is to find $t^*$, the value of $t$ that maximizes $f$ in the direction of the gradient, at each iteration. Because $\mathbf{x}$ and $\nabla f(\mathbf{x})$ have fixed values for the maximization, and because $f(\mathbf{x})$ is concave, this problem should be viewed as maximizing a *concave* function of a *single variable t.* Therefore, it can be solved by the kind of search procedures for one-variable unconstrained optimization that are described in Sec. 12.4 (while considering only nonnegative values of $t$ because of the $t \geq 0$ constraint). Alternatively, if $f$ is a simple function, it may be possible to obtain an analytical solution by setting the derivative with respect to $t$ equal to zero and solving.

### Summary of the Gradient Search Procedure

*Initialization:* Select $\epsilon$ and any initial trial solution $x'$. Go first to the stopping rule.
*Iteration:*

1. Express $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ as a function of $t$ by setting

$$x_j = x_j' + t\left(\frac{\partial f}{\partial x_j}\right)_{\mathbf{x}=\mathbf{x}'}, \qquad \text{for } j = 1, 2, \ldots, n,$$

and then substituting these expressions into $f(\mathbf{x})$.

2. Use a search procedure for one-variable unconstrained optimization (or calculus) to find $t = t^*$ that maximizes $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ over $t \geq 0$.
3. Reset $\mathbf{x}' = \mathbf{x}' + t^*\,\nabla f(\mathbf{x}')$. Then go to the stopping rule.

*Stopping rule:* Evaluate $\nabla f(\mathbf{x}')$ at $\mathbf{x} = \mathbf{x}'$. Check if

$$\left|\frac{\partial f}{\partial x_j}\right| \leq \epsilon \qquad \text{for all } j = 1, 2, \ldots, n.$$

If so, stop with the current $\mathbf{x}'$ as the desired approximation of an optimal solution $\mathbf{x}^*$. Otherwise, perform another iteration.

Now let us illustrate this procedure.

**Example.** Consider the following two-variable problem:

$$\text{Maximize} \qquad f(\mathbf{x}) = 2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2.$$

Thus,

$$\frac{\partial f}{\partial x_1} = 2x_2 - 2x_1,$$
$$\frac{\partial f}{\partial x_2} = 2x_1 + 2 - 4x_2.$$

We also can verify (see Appendix 2) that $f(\mathbf{x})$ is concave.

To begin the gradient search procedure, after choosing a suitably small value of $\epsilon$ (normally well under 0.1) suppose that $\mathbf{x} = (0, 0)$ is selected as the initial trial solution. Because the respective partial derivatives are 0 and 2 at this point, the gradient is

$$\nabla f(0, 0) = (0, 2).$$

With $\epsilon < 2$, the stopping rule then says to perform an iteration.

*Iteration 1:* With values of 0 and 2 for the respective partial derivatives, the first iteration begins by setting

$$x_1 = 0 + t(0) = 0,$$
$$x_2 = 0 + t(2) = 2t,$$

and then substituting these expressions into $f(\mathbf{x})$ to obtain

$$\begin{aligned} f(\mathbf{x}' + t\,\nabla f(\mathbf{x}')) &= f(0, 2t) \\ &= 2(0)(2t) + 2(2t) - 0^2 - 2(2t)^2 \\ &= 4t - 8t^2. \end{aligned}$$

Because

$$f(0, 2t^*) = \max_{t \geq 0} f(0, 2t) = \max_{t \geq 0} \{4t - 8t^2\}$$

and

$$\frac{d}{dt}(4t - 8t^2) = 4 - 16t = 0,$$

it follows that

$$t^* = \frac{1}{4},$$

so

$$\text{Reset} \quad \mathbf{x}' = (0, 0) + \frac{1}{4}(0, 2) = \left(0, \frac{1}{2}\right).$$

This completes the first iteration. For this new trial solution, the gradient is

$$\nabla f\left(0, \frac{1}{2}\right) = (1, 0).$$

With $\epsilon < 1$, the stopping rule now says to perform another iteration.

*Iteration 2:* To begin the second iteration, use the values of 1 and 0 for the respective partial derivatives to set

$$\mathbf{x} = \left(0, \frac{1}{2}\right) + t(1, 0) = \left(t, \frac{1}{2}\right),$$

so

$$f(\mathbf{x}' + t\,\nabla f(\mathbf{x}')) = f\left(0 + t, \frac{1}{2} + 0t\right) = f\left(t, \frac{1}{2}\right)$$
$$= (2t)\left(\frac{1}{2}\right) + 2\left(\frac{1}{2}\right) - t^2 - 2\left(\frac{1}{2}\right)^2$$
$$= t - t^2 + \frac{1}{2}.$$

Because

$$f\left(t^*, \frac{1}{2}\right) = \max_{t \geq 0} f\left(t, \frac{1}{2}\right) = \max_{t \geq 0} \left\{t - t^2 + \frac{1}{2}\right\}$$

and

$$\frac{d}{dt}\left(t - t^2 + \frac{1}{2}\right) = 1 - 2t = 0,$$

then

$$t^* = \frac{1}{2},$$

so

$$\text{Reset} \quad \mathbf{x}' = \left(0, \frac{1}{2}\right) + \frac{1}{2}(1, 0) = \left(\frac{1}{2}, \frac{1}{2}\right).$$

This completes the second iteration. With a typically small value of $\epsilon$, the procedure now would continue on to several more iterations in a similar fashion. (We will forego the details.)

A nice way of organizing this work is to write out a table such as Table 12.3 which summarizes the preceding two iterations. At each iteration, the second column shows the current trial solution, and the rightmost column shows the eventual new trial solution, which then is carried down into the second column for the next iteration. The fourth column gives the expressions for the $x_j$ in terms of $t$ that need to be substituted into $f(\mathbf{x})$ to give the fifth column.

By continuing in this fashion, the subsequent trial solutions would be $(\frac{1}{2}, \frac{3}{4})$, $(\frac{3}{4}, \frac{3}{4})$, $(\frac{3}{4}, \frac{7}{8})$, $(\frac{7}{8}, \frac{7}{8})$, . . . , as shown in Fig. 12.14. Because these points are converging to $\mathbf{x}^* = (1, 1)$, this solution is the optimal solution, as verified by the fact that

$$\nabla f(1, 1) = (0, 0).$$

However, because this converging sequence of trial solutions never reaches its limit, the procedure actually will stop somewhere (depending on $\epsilon$) slightly below (1, 1) as its final approximation of $\mathbf{x}^*$.

As Fig. 12.14 suggests, the gradient search procedure zigzags to the optimal solution rather than moving in a straight line. Some modifications of the procedure have been

■ **TABLE 12.3 Application of the gradient search procedure to the example**

| Iteration | $\mathbf{x}'$ | $\nabla f(\mathbf{x}')$ | $\mathbf{x}' + t\,\nabla f(\mathbf{x}')$ | $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ | $t^*$ | $\mathbf{x}' + t^*\,\nabla f(\mathbf{x}')$ |
|---|---|---|---|---|---|---|
| 1 | (0, 0) | (0, 2) | (0, 2$t$) | $4t - 8t^2$ | $\frac{1}{4}$ | $\left(0, \frac{1}{2}\right)$ |
| 2 | $\left(0, \frac{1}{2}\right)$ | (1, 0) | $\left(t, \frac{1}{2}\right)$ | $t - t^2 + \frac{1}{2}$ | $\frac{1}{2}$ | $\left(\frac{1}{2}, \frac{1}{2}\right)$ |

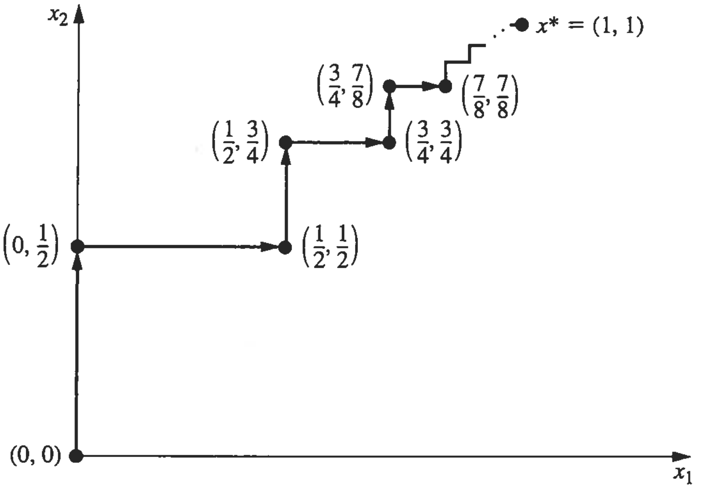

$x_2$
$x^* = (1, 1)$
$\left(\frac{3}{4}, \frac{7}{8}\right)$
$\left(\frac{7}{8}, \frac{7}{8}\right)$
$\left(\frac{1}{2}, \frac{3}{4}\right)$
$\left(\frac{3}{4}, \frac{3}{4}\right)$
$\left(0, \frac{1}{2}\right)$
$\left(\frac{1}{2}, \frac{1}{2}\right)$
(0, 0)
$x_1$

**FIGURE 12.14**
Illustration of the gradient search procedure when $f(x_1, x_2) = 2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2$.

developed that *accelerate* movement toward the optimal solution by taking this zigzag behavior into account.

If $f(\mathbf{x})$ were *not* a concave function, the gradient search procedure still would converge to a *local* maximum. The only change in the description of the procedure for this case is that $t^*$ now would correspond to the *first local maximum* of $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ as $t$ is increased from 0.

If the objective were to *minimize* $f(\mathbf{x})$ instead, one change in the procedure would be to move in the *opposite* direction of the gradient at each iteration. In other words, the rule for obtaining the next point would be

$$\text{Reset} \quad \mathbf{x}' = \mathbf{x}' - t^*\,\nabla f(\mathbf{x}').$$

The only other change is that $t^*$ now would be the nonnegative value of $t$ that *minimizes* $f(\mathbf{x}' - t\,\nabla f(\mathbf{x}'))$; that is,

$$f(\mathbf{x}' - t^*\,\nabla f(\mathbf{x}')) = \min_{t \geq 0} f(\mathbf{x}' - t\,\nabla f(\mathbf{x}')).$$

**Additional examples** of the application of the gradient search procedure are included in both the Worked Examples section of the book's website and your OR Tutor. The IOR Tutorial includes both an interactive procedure and an automatic procedure for applying this algorithm.

## Newton's Method

Section 12.4 describes how Newton's method would be used to solve *one-variable* unconstrained optimization problems. The general version of Newton's method actually is designed to solve *multivariable* unconstrained optimization problems. The basic idea is the same as described in Sec. 12.4, namely, work with a *quadratic approximation* of the objective function $f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in this case. This approximating quadratic function is obtained by truncating the Taylor series around the current trial solution after the second derivative term. This approximate function then is maximized (or minimized) exactly to obtain the new trial solution to start the next iteration.

When the objective function is concave and both the current trial solution $\mathbf{x}$ and its gradient $\nabla f(\mathbf{x})$ are written as *column vectors*, the solution $\mathbf{x}'$ that maximizes the approximating quadratic function has the form,

$$\mathbf{x}' = \mathbf{x} - \left[\nabla^2 f(\mathbf{x})\right]^{-1} \nabla f(\mathbf{x}),$$

where $\nabla^2 f(\mathbf{x})$ is the $n \times n$ matrix (called the *Hessian matrix*) of the second partial derivatives of $f(\mathbf{x})$ evaluated at the current trial solution $\mathbf{x}$ and $\left[\nabla^2 f(\mathbf{x})\right]^{-1}$ is the *inverse* of this Hessian matrix.

Nonlinear programming algorithms that employ Newton's method (including those that adapt it to help deal with *constrained* optimization problems) commonly approximate the inverse of the Hessian matrix in various ways. These approximations of Newton's method are referred to as **quasi-Newton methods** (or *variable metric methods*). We will comment further on the important role of these methods in nonlinear programming in Sec. 12.9.

Further description of these methods is beyond the scope of this book, but further details can be found in books devoted to nonlinear programming.

## 12.6 THE KARUSH-KUHN-TUCKER (KKT) CONDITIONS FOR CONSTRAINED OPTIMIZATION

We now focus on the question of how to recognize an *optimal solution* for a nonlinear programming problem (with differentiable functions). What are the necessary and (perhaps) sufficient conditions that such a solution must satisfy?

In the preceding sections we already noted these conditions for *unconstrained optimization*, as summarized in the first two rows of Table 12.4. Early in Sec. 12.3 we also gave these conditions for the slight *extension* of unconstrained optimization where the *only* constraints are nonnegativity constraints. These conditions are shown in the third row of Table 12.4. As indicated in the last row of the table, the conditions for the general case are called the **Karush-Kuhn-Tucker conditions** (or **KKT conditions**), because they were derived independently by Karush[11] and by Kuhn and Tucker.[12] Their basic result is embodied in the following theorem.

**TABLE 12.4** Necessary and sufficient conditions for optimality

| Problem | Necessary Conditions for Optimality | Also Sufficient If: |
|---|---|---|
| One-variable unconstrained | $\frac{df}{dx} = 0$ | $f(x)$ concave |
| Multivariable unconstrained | $\frac{\partial f}{\partial x_j} = 0 \quad (j = 1, 2, \ldots, n)$ | $f(\mathbf{x})$ concave |
| Constrained, nonnegativity constraints only | $\frac{\partial f}{\partial x_j} = 0 \quad (j = 1, 2, \ldots, n)$<br>(or $\leq 0$ if $x_j = 0$) | $f(\mathbf{x})$ concave |
| General constrained problem | Karush-Kuhn-Tucker conditions | $f(\mathbf{x})$ concave and $g_i(\mathbf{x})$ convex<br>$(i = 1, 2, \ldots, m)$ |

[11] W. Karush, "Minima of Functions of Several Variables with Inequalities as Side Conditions," M.S. thesis, Department of Mathematics, University of Chicago, 1939.

[12] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming," in Jerzy Neyman (ed.), *Proceedings of the Second Berkeley Symposium*, University of California Press, Berkeley, 1951, pp. 481–492.

**Theorem.** Assume that $f(\mathbf{x})$, $g_1(\mathbf{x})$, $g_2(\mathbf{x})$, . . . , $g_m(\mathbf{x})$ are *differentiable* functions satisfying certain regularity conditions.[13] Then

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

can be an *optimal solution* for the nonlinear programming problem only if there exist $m$ numbers $u_1, u_2, \ldots, u_m$ such that *all* the following *KKT conditions* are satisfied:

$$\left.\begin{array}{l} \textbf{1. } \dfrac{\partial f}{\partial x_j} - \displaystyle\sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} \le 0 \\[2em] \textbf{2. } x_j^* \left(\dfrac{\partial f}{\partial x_j} - \displaystyle\sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j}\right) = 0 \end{array}\right\} \quad \text{at } \mathbf{x} = \mathbf{x}^*, \text{ for } j = 1, 2, \ldots, n.$$

$$\left.\begin{array}{l} \textbf{3. } g_i(\mathbf{x}^*) - b_i \le 0 \\ \textbf{4. } u_i[g_i(\mathbf{x}^*) - b_i] = 0 \end{array}\right\} \quad \text{for } i = 1, 2, \ldots, m.$$

$$\begin{array}{ll} \textbf{5. } x_j^* \ge 0, & \text{for } j = 1, 2, \ldots, n. \\ \textbf{6. } u_i \ge 0, & \text{for } i = 1, 2, \ldots, m. \end{array}$$

Note that both conditions 2 and 4 require that the product of two quantities be zero. Therefore, each of these conditions really is saying that at least one of the two quantities must be zero. Consequently, condition 4 can be combined with condition 3 to express them in another equivalent form as

$$(3, 4) \qquad g_i(\mathbf{x}^*) - b_i = 0 \qquad (\text{or} \le 0 \quad \text{if } u_i = 0), \qquad \text{for } i = 1, 2, \ldots, m.$$

Similarly, condition 2 can be combined with condition 1 as

$$(1, 2) \qquad \frac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x_j} = 0 \qquad (\text{or} \le 0 \quad \text{if } x_j^* = 0), \qquad \text{for } j = 1, 2, \ldots, n.$$

When $m = 0$ (no functional constraints), this summation drops out and the combined condition (1, 2) reduces to the condition given in the third row of Table 12.4. Thus, for $m > 0$, each term in the summation modifies the $m = 0$ condition to incorporate the effect of the corresponding functional constraint.

In conditions 1, 2, 4, and 6, the $u_i$ correspond to the *dual variables* of linear programming (we expand on this correspondence at the end of the section), and they have a comparable economic interpretation. However, the $u_i$ actually arose in the mathematical derivation as *Lagrange multipliers* (discussed in Appendix 3). Conditions 3 and 5 do nothing more than ensure the feasibility of the solution. The other conditions eliminate most of the feasible solutions as possible candidates for an optimal solution.

However, note that satisfying these conditions does not guarantee that the solution is optimal. As summarized in the rightmost column of Table 12.4, certain additional *convexity* assumptions are needed to obtain this guarantee. These assumptions are spelled out in the following extension of the theorem.

**Corollary.** Assume that $f(\mathbf{x})$ is a *concave* function and that $g_1(\mathbf{x})$, $g_2(\mathbf{x})$, . . . , $g_m(\mathbf{x})$ are *convex* functions (i.e., this problem is a convex programming problem), where all these functions satisfy the regularity conditions. Then $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an *optimal solution* if and only if all the conditions of the theorem are satisfied.

[13] Ibid., p. 483.

**Example.** To illustrate the formulation and application of the *KKT conditions,* we consider the following two-variable nonlinear programming problem:

$$\text{Maximize} \quad f(\mathbf{x}) = \ln(x_1 + 1) + x_2,$$

subject to

$$2x_1 + x_2 \leq 3$$

and

$$x_1 \geq 0, \qquad x_2 \geq 0,$$

where ln denotes the natural logarithm. Thus, $m = 1$ (one functional constraint) and $g_1(\mathbf{x}) = 2x_1 + x_2$, so $g_1(\mathbf{x})$ is convex. Furthermore, it can be easily verified (see Appendix 2) that $f(\mathbf{x})$ is concave. Hence, the corollary applies, so any solution that satisfies the KKT conditions will definitely be an optimal solution. Applying the formulas given in the theorem yields the following KKT conditions for this example:

**1($j$ = 1).** $\dfrac{1}{x_1 + 1} - 2u_1 \leq 0.$

**2($j$ = 1).** $x_1\left(\dfrac{1}{x_1 + 1} - 2u_1\right) = 0.$

**1($j$ = 2).** $1 - u_1 \leq 0.$

**2($j$ = 2).** $x_2(1 - u_1) = 0.$

**3.** $2x_1 + x_2 - 3 \leq 0.$

**4.** $u_1(2x_1 + x_2 - 3) = 0.$

**5.** $x_1 \geq 0,\ x_2 \geq 0.$

**6.** $u_1 \geq 0.$

The steps in solving the KKT conditions for this particular example are outlined below.

1. $u_1 \geq 1$, from condition 1($j$ = 2).
   $x_1 \geq 0$, from condition 5.
2. Therefore, $\dfrac{1}{x_1 + 1} - 2u_1 < 0.$
3. Therefore, $x_1 = 0$, from condition 2($j$ = 1).
4. $u_1 \neq 0$ implies that $2x_1 + x_2 - 3 = 0$, from condition 4.
5. Steps 3 and 4 imply that $x_2 = 3$.
6. $x_2 \neq 0$ implies that $u_1 = 1$, from condition 2($j$ = 2).
7. No conditions are violated by $x_1 = 0$, $x_2 = 3$, $u_1 = 1$.

Therefore, there exists a number $u_1 = 1$ such that $x_1 = 0$, $x_2 = 3$, and $u_1 = 1$ satisfy all the conditions. Consequently, $\mathbf{x}^* = (0, 3)$ is an optimal solution for this problem.

This particular problem was relatively easy to solve because the first two steps above quickly led to the remaining conclusions. It often is more difficult to see how to get started. The particular progression of steps needed to solve the KKT conditions will differ from one problem to the next. When the logic is not apparent, it is sometimes helpful to consider separately the different cases where each $x_j$ and $u_i$ are specified to be either equal to or greater than 0 and then trying each case until one leads to a solution.

To illustrate, suppose this approach of considering the different cases separately had been applied to the above example instead of using the logic involved in the above seven steps. For this example, eight cases need to be considered. These cases correspond to the eight combinations of $x_1 = 0$ versus $x_1 > 0$, $x_2 = 0$ versus $x_2 > 0$, and $u_1 = 0$ versus $u_1 > 0$. Each case leads to a simpler statement and analysis of the conditions. To illustrate, consider first the case shown next, where $x_1 = 0$, $x_2 = 0$, and $u_1 = 0$.

*KKT Conditions for the Case* $x_1 = 0,\ x_2 = 0,\ u_1 = 0$

**1(*j* = 1).** $\dfrac{1}{0+1} \le 0.$ Contradiction.

**1(*j* = 2).** $1 - 0 \le 0.$ Contradiction.

**3.** $0 + 0 \le 3.$

(All the other conditions are redundant.)

As listed below, the other three cases where $u_1 = 0$ also give immediate contradictions in a similar way, so no solution is available.

Case $x_1 = 0, x_2 > 0, u_1 = 0$ contradicts conditions $1(j = 1)$, $1(j = 2)$, and $2(j = 2)$.
Case $x_1 > 0, x_2 = 0, u_1 = 0$ contradicts conditions $1(j = 1)$, $2(j = 1)$, and $1(j = 2)$.
Case $x_1 > 0, x_2 > 0, u_1 = 0$ contradicts conditions $1(j = 1)$, $2(j = 1)$, $1(j = 2)$, and $2(j = 2)$.

The case $x_1 > 0, x_2 > 0, u_1 > 0$ enables one to delete these nonzero multipliers from conditions $2(j = 1)$, $2(j = 2)$, and 4, which then enables deletion of conditions $1(j = 1)$, $1(j = 2)$, and 3 as redundant, as summarized next.

*KKT Conditions for the Case* $x_1 > 0,\ x_2 > 0,\ u_1 > 0$

**1(*j* = 1).** $\dfrac{1}{x_1+1} - 2u_1 = 0.$

**2(*j* = 2).** $1 - u_1 = 0.$

**4.** $2x_1 + x_2 - 3 = 0.$

(All the other conditions are redundant.)

Therefore, $u_1 = 1$, so $x_1 = -\frac{1}{2}$, which contradicts $x_1 > 0$.

Now suppose that the case $x_1 = 0, x_2 > 0, u_1 > 0$ is tried next.

*KKT Conditions for the Case* $x_1 = 0,\ x_2 > 0,\ u_1 > 0$

**1(*j* = 1).** $\dfrac{1}{0+1} - 2u_1 = 0.$

**2(*j* = 2).** $1 - u_1 = 0.$

**4.** $0 + x_2 - 3 = 0.$

(All the other conditions are redundant.)

Therefore, $x_1 = 0,\ x_2 = 3,\ u_1 = 1$. Having found a solution, we know that no additional cases need be considered.

If you would like to see **another example** of using the KKT conditions to solve for an optimal solution, one is provided in the Worked Examples section of the book's website.

For problems more complicated than the above example, it may be difficult, if not essentially impossible, to derive an optimal solution *directly* from the KKT conditions. Nevertheless, these conditions still provide valuable clues as to the identity of an optimal solution, and they also permit us to check whether a proposed solution may be optimal.

There also are many valuable *indirect* applications of the KKT conditions. One of these applications arises in the *duality theory* that has been developed for nonlinear programming to parallel the duality theory for linear programming presented in Chap. 6. In particular, for any given constrained maximization problem (call it the *primal problem*), the KKT conditions can be used to define a closely associated dual problem that is a constrained minimization problem. The variables in the dual problem consist of both the Lagrange multipliers $u_i$ ($i = 1, 2, \ldots, m$) and the primal variables $x_j$ ($j = 1, 2, \ldots, n$).

In the special case where the primal problem is a linear programming problem, the $x_j$ variables drop out of the dual problem and it becomes the familiar dual problem of linear programming (where the $u_i$ variables here correspond to the $y_i$ variables in Chap. 6). When the primal problem is a convex programming problem, it is possible to establish relationships between the primal problem and the dual problem that are similar to those for linear programming. For example, the *strong duality property* of Sec. 6.1, which states that the optimal objective function values of the two problems are equal, also holds here. Furthermore, the values of the $u_i$ variables in an optimal solution for the dual problem can again be interpreted as *shadow prices* (see Secs. 4.7 and 6.2); i.e., they give the rate at which the optimal objective function value for the primal problem could be increased by (slightly) increasing the right-hand side of the corresponding constraint. Because duality theory for nonlinear programming is a relatively advanced topic, the interested reader is referred elsewhere for further information.[14]

You will see another indirect application of the KKT conditions in the next section.

## 12.7 QUADRATIC PROGRAMMING

As indicated in Sec. 12.3, the quadratic programming problem differs from the linear programming problem only in that the objective function also includes $x_j^2$ and $x_i x_j$ $(i \neq j)$ terms. Thus, if we use matrix notation like that introduced at the beginning of Sec. 5.2, the problem is to find $\mathbf{x}$ so as to

$$\text{Maximize} \qquad f(\mathbf{x}) = \mathbf{cx} - \frac{1}{2}\mathbf{x}^T\mathbf{Qx},$$

subject to

$$\mathbf{Ax} \leq \mathbf{b} \qquad \text{and} \qquad \mathbf{x} \geq \mathbf{0},$$

where $\mathbf{c}$ is a row vector, $\mathbf{x}$ and $\mathbf{b}$ are column vectors, $\mathbf{Q}$ and $\mathbf{A}$ are matrices, and the superscript $T$ denotes the transpose (see Appendix 4). The $q_{ij}$ (elements of $Q$) are given constants such that $q_{ij} = q_{ji}$ (which is the reason for the factor of $\frac{1}{2}$ in the objective function). By performing the indicated vector and matrix multiplications, the objective function then is expressed in terms of these $q_{ij}$, the $c_j$ (elements of $\mathbf{c}$), and the variables as follows:

$$f(\mathbf{x}) = \mathbf{cx} - \frac{1}{2}\mathbf{x}^T\mathbf{Qx} = \sum_{j=1}^{n} c_j x_j - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} x_i x_j.$$

For each term where $i = j$ in this double summation, $x_i x_j = x_j^2$, so $-\frac{1}{2}q_{jj}$ is the coefficient of $x_j^2$. When $i \neq j$, then $-\frac{1}{2}(q_{ij}x_i x_j + q_{ji}x_j x_i) = -q_{ij}x_i x_j$, so $-q_{ij}$ is the total coefficient for the product of $x_i$ and $x_j$.

To illustrate this notation, consider the following example of a quadratic programming problem.

$$\text{Maximize} \qquad f(x_1, x_2) = 15x_1 + 30x_2 + 4x_1x_2 - 2x_1^2 - 4x_2^2,$$

subject to

$$x_1 + 2x_2 \leq 30$$

and

$$x_1 \geq 0, \qquad x_2 \geq 0.$$

[14] For a unified survey of various approaches to duality in nonlinear programming, see A. M. Geoffrion, "Duality in Nonlinear Programming: A Simplified Applications-Oriented Development," *SIAM Review*, **13**: 1–37, 1971.

In this case,

$$\mathbf{c} = [15 \quad 30], \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix},$$

$$\mathbf{A} = [1 \quad 2], \qquad \mathbf{b} = [30].$$

Note that

$$\begin{aligned} \mathbf{x}^T\mathbf{Q}\mathbf{x} &= [x_1 \quad x_2] \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= [(4x_1 - 4x_2) \quad (-4x_1 + 8x_2)] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 4x_1^2 - 4x_2x_1 - 4x_1x_2 + 8x_2^2 \\ &= q_{11}x_1^2 + q_{21}x_2x_1 + q_{12}x_1x_2 + q_{22}x_2^2. \end{aligned}$$

Multiplying through by $-\frac{1}{2}$ gives

$$-\frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} = -2x_1^2 + 4x_1x_2 - 4x_2^2,$$

which is the nonlinear portion of the objective function for this example. Since $q_{11} = 4$ and $q_{22} = 8$, the example illustrates that $-\frac{1}{2}q_{jj}$ is the coefficient of $x_j^2$ in the objective function. The fact that $q_{12} = q_{21} = -4$ illustrates that both $-q_{ij}$ and $-q_{ji}$ give the total coefficient of the product of $x_i$ and $x_j$.

Several algorithms have been developed for the special case of the quadratic programming problem where the objective function is a *concave* function. (A way to verify that the objective function is concave is to verify the equivalent condition that

$$\mathbf{x}^T\mathbf{Q}\mathbf{x} \geq \mathbf{0}$$

for all **x**, that is, **Q** is a *positive semidefinite* matrix.) We shall describe one[15] of these algorithms, the *modified simplex method*, that has been quite popular because it requires using only the simplex method with a slight modification. The key to this approach is to construct the KKT conditions from the preceding section and then to reexpress these conditions in a convenient form that closely resembles linear programming. Therefore, before describing the algorithm, we shall develop this convenient form.

## The KKT Conditions for Quadratic Programming

For concreteness, let us first consider the above example. Starting with the form given in the preceding section, its KKT conditions are the following.

**1(j = 1).** $15 + 4x_2 - 4x_1 - u_1 \leq 0.$
**2(j = 1).** $x_1(15 + 4x_2 - 4x_1 - u_1) = 0.$
**1(j = 2).** $30 + 4x_1 - 8x_2 - 2u_1 \leq 0.$
**2(j = 2).** $x_2(30 + 4x_1 - 8x_2 - 2u_1) = 0.$
**3.** $x_1 + 2x_2 - 30 \leq 0.$
**4.** $u_1(x_1 + 2x_2 - 30) = 0.$
**5.** $x_1 \geq 0, \qquad x_2 \geq 0.$
**6.** $u_1 \geq 0.$

To begin reexpressing these conditions in a more convenient form, we move the constants in conditions 1(j = 1), 1(j = 2), and 3 to the right-hand side and then introduce

[15] P. Wolfe, "The Simplex Method for Quadratic Programming," *Econometrics*, **27:** 382–398, 1959. This paper develops both a short form and a long form of the algorithm. We present a version of the *short form*, which assumes further that *either* $\mathbf{c} = \mathbf{0}$ *or* the objective function is *strictly* concave.

nonnegative *slack variables* (denoted by $y_1$, $y_2$, and $v_1$, respectively) to convert these inequalities to equations.

$$\begin{aligned}
&\mathbf{1(j=1).} && -4x_1 + 4x_2 - u_1 + y_1 && = -15\\
&\mathbf{1(j=2).} && 4x_1 - 8x_2 - 2u_1 + y_2 && = -30\\
&\mathbf{3.} && x_1 + 2x_2 + v_1 && = 30
\end{aligned}$$

Note that condition $2(j = 1)$ can now be reexpressed as simply requiring that either $x_1 = 0$ or $y_1 = 0$; that is,

$$\mathbf{2(j=1).}\ x_1y_1 = 0.$$

In just the same way, conditions $2(j = 2)$ and 4 can be replaced by

$$\begin{aligned}
&\mathbf{2(j=2).} && x_2y_2 = 0,\\
&\mathbf{4.} && u_1v_1 = 0.
\end{aligned}$$

For each of these three pairs—$(x_1, y_1)$, $(x_2, y_2)$, $(u_1, v_1)$—the two variables are called **complementary variables,** because only one of the two variables can be nonzero. These new forms of conditions $2(j = 1)$, $2(j = 2)$, and 4 can be combined into one constraint,

$$x_1y_1 + x_2y_2 + u_1v_1 = 0,$$

called the **complementarity constraint.**

After multiplying through the equations for conditions $1(j = 1)$ and $1(j = 2)$ by $-1$ to obtain nonnegative right-hand sides, we now have the desired convenient form for the entire set of conditions shown here:

$$\begin{aligned}
4x_1 - 4x_2 + u_1 - y_1 &= 15\\
-4x_1 + 8x_2 + 2u_1 - y_2 &= 30\\
x_1 + 2x_2 + v_1 &= 30\\
x_1 \geq 0, \quad x_2 \geq 0, \quad u_1 \geq 0, \quad y_1 \geq 0, &\quad y_2 \geq 0, \quad v_1 \geq 0\\
x_1y_1 + x_2y_2 + u_1v_1 &= 0
\end{aligned}$$

This form is particularly convenient because, except for the complementarity constraint, these conditions are *linear programming constraints.*

For *any* quadratic programming problem, its KKT conditions can be reduced to this same convenient form containing just linear programming constraints plus one complementarity constraint. In matrix notation again, this general form is

$$\begin{aligned}
&\mathbf{Qx} + \mathbf{A}^T\mathbf{u} - \mathbf{y} = \mathbf{c}^T,\\
&\qquad \mathbf{Ax} + \mathbf{v} = \mathbf{b},\\
&\mathbf{x} \geq \mathbf{0}, \quad \mathbf{u} \geq \mathbf{0}, \quad \mathbf{y} \geq \mathbf{0}, \quad \mathbf{v} \geq \mathbf{0},\\
&\mathbf{x}^T\mathbf{y} + \mathbf{u}^T\mathbf{v} = 0,
\end{aligned}$$

where the elements of the column vector **u** are the $u_i$ of the preceding section and the elements of the column vectors **y** and **v** are slack variables.

Because the objective function of the original problem is assumed to be concave and because the constraint functions are linear and therefore convex, the corollary to the theorem of Sec. 12.6 applies. Thus, **x** is *optimal* if and only if there exist values of **y**, **u**, and **v** such that all four vectors together satisfy all these conditions. The original problem is thereby reduced to the equivalent problem of finding a *feasible solution* to these *constraints.*

It is of interest to note that this equivalent problem is one example of the *linear complementarity problem* introduced in Sec. 12.3 (see Prob. 12.3-6), and that a key constraint for the linear complementarity problem is its *complementarity constraint.*

### The Modified Simplex Method

The *modified simplex method* exploits the key fact that, with the exception of the complementarity constraint, the KKT conditions in the convenient form obtained above are nothing more than linear programming constraints. Furthermore, the complementarity constraint simply implies that it is not permissible for *both* complementary variables of any pair to be (nondegenerate) basic variables (the only variables > 0) when (nondegenerate) BF solutions are considered. Therefore, the problem reduces to finding an initial BF solution to any linear programming problem that has these constraints, subject to this additional restriction on the identity of the basic variables. (This initial BF solution may be the only feasible solution in this case.)

As we discussed in Sec. 4.6, finding such an initial BF solution is relatively straightforward. In the simple case where $\mathbf{c}^T \leq \mathbf{0}$ (unlikely) and $\mathbf{b} \geq \mathbf{0}$, the initial basic variables are the elements of **y** and **v** (multiply through the first set of equations by $-1$), so that the desired solution is $\mathbf{x} = \mathbf{0}$, $\mathbf{u} = \mathbf{0}$, $\mathbf{y} = -\mathbf{c}^T$, $\mathbf{v} = \mathbf{b}$. Otherwise, you need to revise the problem by introducing an *artificial variable* into each of the equations where $c_j > 0$ (add the variable on the left) or $b_i < 0$ (subtract the variable on the left and then multiply through by $-1$) in order to use these artificial variables (call them $z_1$, $z_2$, and so on) as initial basic variables for the revised problem. (Note that this choice of initial basic variables satisfies the complementarity constraint, because as nonbasic variables $\mathbf{x} = \mathbf{0}$ and $\mathbf{u} = \mathbf{0}$ automatically.)

Next, use phase 1 of the *two-phase method* (see Sec. 4.6) to find a BF solution for the real problem; i.e., apply the simplex method (with one modification) to the following linear programming problem

$$\text{Minimize} \qquad Z = \sum_j z_j,$$

subject to the linear programming constraints obtained from the KKT conditions, but with these artificial variables included.

The one modification in the simplex method is the following change in the procedure for selecting an entering basic variable.

> **Restricted-Entry Rule:** When you are choosing an entering basic variable, exclude from consideration any nonbasic variable whose *complementary variable* already is a basic variable; the choice should be made from the other nonbasic variables according to the usual criterion for the simplex method.

This rule keeps the complementarity constraint satisfied throughout the course of the algorithm. When an optimal solution

$$\mathbf{x}^*, \mathbf{u}^*, \mathbf{y}^*, \mathbf{v}^*, z_1 = 0, \ldots, z_n = 0$$

is obtained for the phase 1 problem, $\mathbf{x}^*$ is the desired optimal solution for the original quadratic programming problem. Phase 2 of the two-phase method is not needed.

**Example.** We shall now illustrate this approach on the example given at the beginning of the section. As can be verified from the results in Appendix 2 (see Prob. 12.7-1*a*), $f(x_1, x_2)$ is *strictly concave;* i.e.,

$$\mathbf{Q} = \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix}$$

is positive definite, so the algorithm can be applied.

The starting point for solving this example is its KKT conditions in the convenient form obtained earlier in the section. After the needed artificial variables are introduced, the linear programming problem to be addressed explicitly by the modified simplex method then is

$$\text{Minimize} \qquad Z = z_1 + z_2,$$

subject to

$$\begin{array}{rcrcrcrcrcrcrcrcl} 4x_1 & - & 4x_2 & + & u_1 & - & y_1 & & & & & + & z_1 & & & = & 15 \\ -4x_1 & + & 8x_2 & + & 2u_1 & & & - & y_2 & & & & & + & z_2 & = & 30 \\ x_1 & + & 2x_2 & & & & & & & + & v_1 & & & & & = & 30 \end{array}$$

and

$$x_1 \geq 0, \quad x_2 \geq 0, \quad u_1 \geq 0, \quad y_1 \geq 0, \quad y_2 \geq 0, \quad v_1 \geq 0,$$
$$z_1 \geq 0, \quad z_2 \geq 0.$$

The additional complementarity constraint

$$x_1y_1 + x_2y_2 + u_1v_1 = 0,$$

is not included explicitly, because the algorithm automatically enforces this constraint because of the *restricted-entry rule.* In particular, for each of the three pairs of complementary variables—$(x_1, y_1)$, $(x_2, y_2)$, $(u_1,v_1)$—whenever one of the two variables already is a basic variable, the other variable is *excluded* as a candidate for the entering basic variable. Remember that the only *nonzero* variables are basic variables. Because the initial set of basic variables for the linear programming problem—$z_1$, $z_2$, $v_1$—gives an initial BF solution that satisfies the complementarity constraint, there is no way that this constraint can be violated by any subsequent BF solution.

Table 12.5 shows the results of applying the modified simplex method to this problem. The first simplex tableau exhibits the initial system of equations *after* converting from minimizing $Z$ to maximizing $-Z$ *and* algebraically eliminating the initial basic variables from Eq. (0), just as was done for the radiation therapy example in Sec. 4.6. The three iterations proceed just as for the regular simplex method, *except* for eliminating certain candidates for the entering basic variable because of the restricted-entry rule. In the first tableau, $u_1$ is eliminated as a candidate because its complementary variable ($v_1$) already is a basic variable (but $x_2$ would have been chosen anyway because $-4 < -3$). In the second tableau, both $u_1$ and $y_2$ are eliminated as candidates (because $v_1$ and $x_2$ are basic variables), so $x_1$ automatically is chosen as the only candidate with a negative coefficient in row 0 (whereas the *regular* simplex method would have permitted choosing *either* $x_1$ or $u_1$ because they are tied for having the largest negative coefficient). In the third tableau, both $y_1$ and $y_2$ are eliminated (because $x_1$ and $x_2$ are basic variables). However, $u_1$ is *not* eliminated because $v_1$ no longer is a basic variable, so $u_1$ is chosen as the entering basic variable in the usual way.

The resulting optimal solution for this phase 1 problem is $x_1 = 12$, $x_2 = 9$, $u_1 = 3$, with the rest of the variables zero. (Problem 12.7-1$c$ asks you to verify that this solution is optimal by showing that $x_1 = 12$, $x_2 = 9$, $u_1 = 3$ satisfy the KKT conditions for the original problem when they are written in the form given in Sec. 12.6.) Therefore, the optimal solution for the quadratic programming problem (which includes only the $x_1$ and $x_2$ variables) is $(x_1, x_2) = (12, 9)$.

**TABLE 12.5** Application of the modified simplex method to the quadratic programming example

| Iteration | Basic Variable | Eq. | $Z$ | $x_1$ | $x_2$ | $u_1$ | $y_1$ | $y_2$ | $v_1$ | $z_1$ | $z_2$ | Right Side |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $Z$ | (0) | $-1$ | 0 | $-4$ | $-3$ | 1 | 1 | 0 | 0 | 0 | $-45$ |
| | $z_1$ | (1) | 0 | 4 | $-4$ | 1 | $-1$ | 0 | 0 | 1 | 0 | 15 |
| | $z_2$ | (2) | 0 | $-4$ | 8 | 2 | 0 | $-1$ | 0 | 0 | 1 | 30 |
| | $v_1$ | (3) | 0 | 1 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 30 |
| 1 | $Z$ | (0) | $-1$ | $-2$ | 0 | $-2$ | 1 | $\frac{1}{2}$ | 0 | 0 | $\frac{1}{2}$ | $-30$ |
| | $z_1$ | (1) | 0 | 2 | 0 | 2 | $-1$ | $-\frac{1}{2}$ | 0 | 1 | $\frac{1}{2}$ | 30 |
| | $x_2$ | (2) | 0 | $-\frac{1}{2}$ | 1 | $\frac{1}{4}$ | 0 | $-\frac{1}{8}$ | 0 | 0 | $\frac{1}{8}$ | $3\frac{3}{4}$ |
| | $v_1$ | (3) | 0 | 2 | 0 | $-\frac{1}{2}$ | 0 | $\frac{1}{4}$ | 1 | 0 | $-\frac{1}{4}$ | $22\frac{1}{2}$ |
| 2 | $Z$ | (0) | $-1$ | 0 | 0 | $-\frac{5}{2}$ | 1 | $\frac{3}{4}$ | 1 | 0 | $\frac{1}{4}$ | $-7\frac{1}{2}$ |
| | $z_1$ | (1) | 0 | 0 | 0 | $\frac{5}{2}$ | $-1$ | $-\frac{3}{4}$ | $-1$ | 1 | $\frac{3}{4}$ | $7\frac{1}{2}$ |
| | $x_2$ | (2) | 0 | 0 | 1 | $\frac{1}{8}$ | 0 | $-\frac{1}{16}$ | $\frac{1}{4}$ | 0 | $\frac{1}{16}$ | $9\frac{3}{8}$ |
| | $x_1$ | (3) | 0 | 1 | 0 | $-\frac{1}{4}$ | 0 | $\frac{1}{8}$ | $\frac{1}{2}$ | 0 | $-\frac{1}{8}$ | $11\frac{1}{4}$ |
| 3 | $Z$ | (0) | $-1$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | $u_1$ | (1) | 0 | 0 | 0 | 1 | $-\frac{2}{5}$ | $-\frac{3}{10}$ | $-\frac{2}{5}$ | $\frac{2}{5}$ | $\frac{3}{10}$ | 3 |
| | $x_2$ | (2) | 0 | 0 | 1 | 0 | $\frac{1}{20}$ | $-\frac{1}{40}$ | $\frac{3}{10}$ | $-\frac{1}{20}$ | $\frac{1}{40}$ | 9 |
| | $x_1$ | (3) | 0 | 1 | 0 | 0 | $-\frac{1}{10}$ | $\frac{1}{20}$ | $\frac{2}{5}$ | $\frac{1}{10}$ | $-\frac{1}{20}$ | 12 |

The Worked Examples section of the book's website include **another example** that illustrates the application of the modified simplex method to a quadratic programming problem. The KKT conditions also are applied to this example.

## Some Software Options

Your IOR Tutorial includes an interactive procedure for the modified simplex method to help you learn this algorithm efficiently. In addition, Excel, LINGO, LINDO, and MPL/CPLEX all can solve quadratic programming problems.

The procedure for using Excel is almost the same as with linear programming. The one crucial difference is that the equation entered for the cell that contains the value of the objective function now needs to be a quadratic equation. To illustrate, consider again the example introduced at the beginning of the section, which has the objective function

$$f(x_1, x_2) = 15x_1 + 30x_2 + 4x_1x_2 - 2x_1^2 - 4x_2^2.$$

Suppose that the values of $x_1$ and $x_2$ are in cells B4 and C4 of the Excel spreadsheet, and that the value of the objective function is in cell F4. Then the equation for cell F4 needs to be

F4 = 15*B4 + 30*C4 + 4*B4*C4 − 2*(B4^2) − 4*(C4^2), where the symbol ^2 indicates an exponent of 2. Before solving the model, you should click on the Option button and make sure that the *Assume Linear Model* option is *not* selected (since this is not a *linear* programming model).

When using MPL/CPLEX, you should set the model type to Quadratic by adding the following statement at the beginning of the model file.

```
OPTIONS
   ModelType = Quadratic
```

(Alternatively, you can select the Quadratic Models option from the MPL Language option dialogue box, but then you will need to remember to change the setting when dealing with linear programming problems again.) Otherwise, the procedure is the same as with linear programming except that the expression for the objective function now is a quadratic function. Thus, for the example, the objective function would be expressed as

```
15x1 + 30x2 + 4x1*x2 − 2(x1^2) − 4(x2^2).
```

Nothing more needs to be done when calling CPLEX, since it will automatically recognize the model as being a quadratic programming problem.

This objective function would be expressed in this same way for a LINGO model. LINGO then will automatically call its nonlinear solver to solve the model.

In fact, the Excel, MPL/CPLEX, and LINGO/LINDO files for this chapter in your OR Courseware all demonstrate their procedures by showing the details for how these software packages set up and solve this example.

Some of these software packages also can be applied to more complicated kinds of nonlinear programming problems than quadratic programming. Although CPLEX cannot, the professional version of MPL does support some other solvers that can. The student version of MPL on the book's website includes one such solver called CONOPT (a product of ARKI Consulting) that is designed for solving convex programming problems. It can be used by adding the following statement at the beginning of the model file.

```
OPTIONS
   ModelType = Nonlinear
```

Both Excel and LINGO include versatile nonlinear solvers. However, be aware that the Excel Solver is not guaranteed to find an optimal solution for complicated problems, especially nonconvex programming problems (the subject of Sec. 12.10). On the other hand, LINGO contains a *global optimizer* that will find a globally optimal solution for sufficiently small nonconvex programming problems. MPL also supports a global optimizer called LGO as one of its solvers provided on the book's website.

## 12.8 SEPARABLE PROGRAMMING

The preceding section showed how one class of nonlinear programming problems can be solved by an extension of the simplex method. We now consider another class, called *separable programming,* that actually can be solved by the simplex method itself, because any such problem can be approximated as closely as desired by a linear programming problem with a larger number of variables.

As indicated in Sec. 12.3, in separable programming it is assumed that the objective function $f(\mathbf{x})$ is concave, that each of the constraint functions $g_i(\mathbf{x})$ is convex, and that all these functions are separable functions (functions where each term involves just a single variable). However, to simplify the discussion, we focus here on the special case where the convex and separable $g_i(\mathbf{x})$ are, in fact, *linear functions,* just as for linear programming. (We will turn to the general case briefly at the end of this section.) Thus, only the objective function requires special treatment for this special case.

Under the preceding assumptions, the objective function can be expressed as a sum of concave functions of individual variables

$$f(\mathbf{x}) = \sum_{j=1}^{n} f_j(x_j),$$

so that each $f_j(x_j)$ has a shape[16] such as the one shown in Fig. 12.15 (either case) over the feasible range of values of $x_j$. Because $f(\mathbf{x})$ represents the measure of performance (say, profit) for all the activities together, $f_j(x_j)$ represents the *contribution to profit* from activity $j$ when it is conducted at level $x_j$. The condition of $f(\mathbf{x})$ being separable simply implies additivity (see Sec. 3.3); i.e., there are no interactions between the activities (no cross-product terms) that affect total profit beyond their independent contributions. The assumption that each $f_j(x_j)$ is concave says that the *marginal profitability* (slope of the profit curve) either stays the same or decreases (*never* increases) as $x_j$ is increased.

Concave profit curves occur quite frequently. For example, it may be possible to sell a limited amount of some product at a certain price, then a further amount at a lower price, and perhaps finally a further amount at a still lower price. Similarly, it may be necessary to purchase raw materials from increasingly expensive sources. In another common situation, a more expensive production process must be used (e.g., overtime rather than regular-time work) to increase the production rate beyond a certain point.

These kinds of situations can lead to either type of profit curve shown in Fig. 12.15. In case 1, the slope decreases only at certain *breakpoints,* so that $f_j(x_j)$ is a *piecewise linear function* (a sequence of connected line segments). For case 2, the slope may decrease continuously as $x_j$ increases, so that $f_j(x_j)$ is a general concave function. Any such function can be approximated as closely as desired by a piecewise linear function, and this kind of approximation is used as needed for separable programming problems. (Figure 12.15 shows an approximating function that consists of just three line segments, but the approximation can be made even better just by introducing additional breakpoints.) This approximation is very convenient because a piecewise linear function of a single variable can be rewritten as a *linear function* of several variables, with one special restriction on the values of these variables, as described next.

## Reformulation as a Linear Programming Problem

The key to rewriting a piecewise linear function as a linear function is to use a separate variable for each line segment. To illustrate, consider the piecewise linear function $f_j(x_j)$ shown in Fig. 12.15, case 1 (or the approximating piecewise linear function for case 2), which has three line segments over the feasible range of values of $x_j$. Introduce the three new variables $x_{j1}$, $x_{j2}$, and $x_{j3}$ and set

$$x_j = x_{j1} + x_{j2} + x_{j3},$$

where

$$0 \le x_{j1} \le u_{j1}, \qquad 0 \le x_{j2} \le u_{j2}, \qquad 0 \le x_{j3} \le u_{j3}.$$

[16] $f(\mathbf{x})$ is concave if and only if *every* $f_j(x_j)$ is concave.

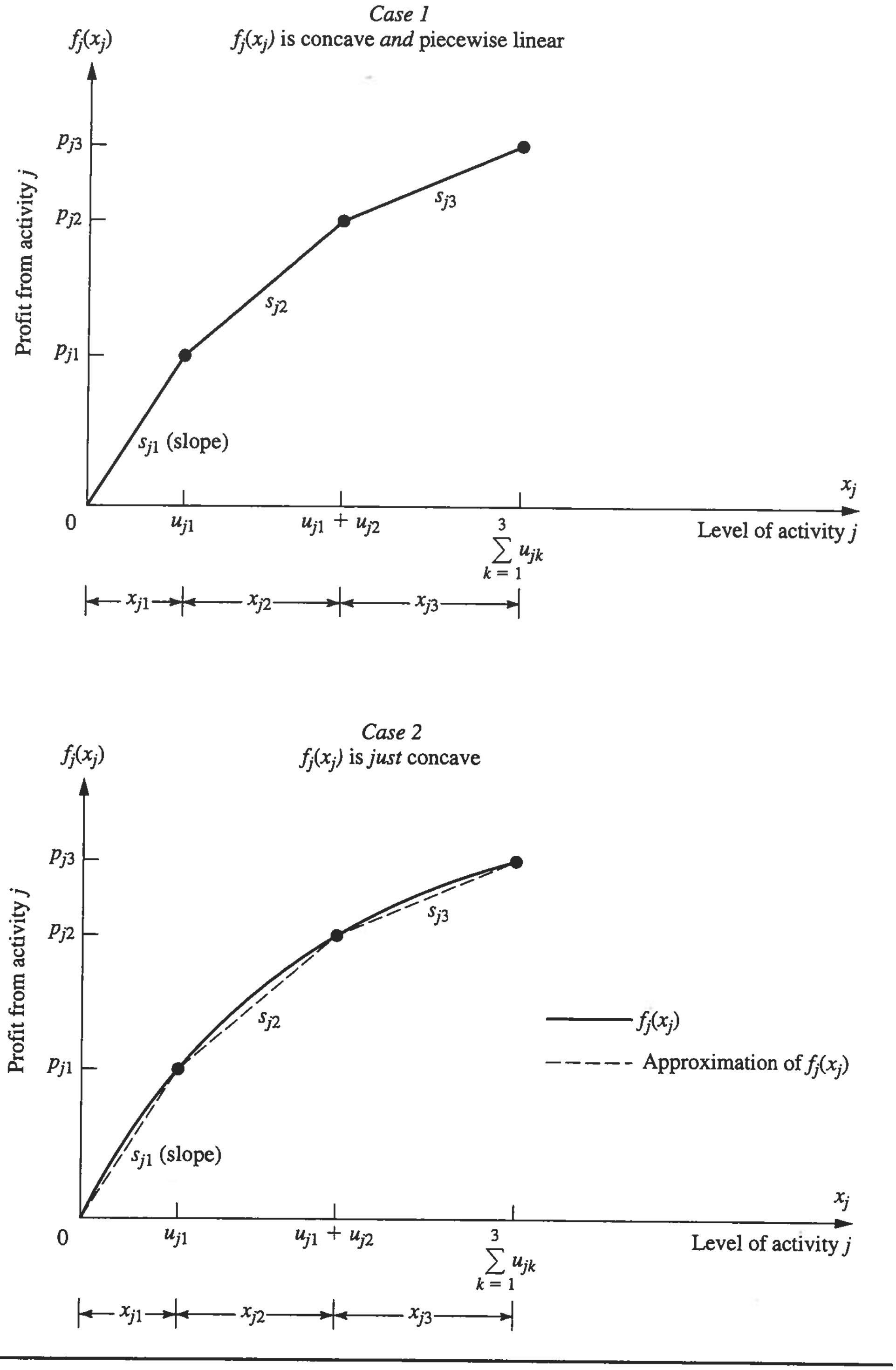


**FIGURE 12.15** Shape of profit curves for separable programming.

Then use the slopes $s_{j1}$, $s_{j2}$, and $s_{j3}$ to rewrite $f_j(x_j)$ as

$$f_j(x_j) = s_{j1}x_{j1} + s_{j2}x_{j2} + s_{j3}x_{j3},$$

with the **special restriction** that

$$\begin{aligned} x_{j2} &= 0 \quad \text{whenever} \quad x_{j1} < u_{j1}, \\ x_{j3} &= 0 \quad \text{whenever} \quad x_{j2} < u_{j2}. \end{aligned}$$

To see why this special restriction is required, suppose that $x_j = 1$, where $u_{jk} > 1$ ($k = 1, 2, 3$), so that $f_j(1) = s_{j1}$. Note that

$$x_{j1} + x_{j2} + x_{j3} = 1$$

permits

$$\begin{aligned} x_{j1} &= 1, \quad x_{j2} = 0, \quad x_{j3} = 0 \quad \Rightarrow \quad f_j(1) = s_{j1}, \\ x_{j1} &= 0, \quad x_{j2} = 1, \quad x_{j3} = 0 \quad \Rightarrow \quad f_j(1) = s_{j2}, \\ x_{j1} &= 0, \quad x_{j2} = 0, \quad x_{j3} = 1 \quad \Rightarrow \quad f_j(1) = s_{j3}, \end{aligned}$$

and so on, where

$$s_{j1} > s_{j2} > s_{j3}.$$

However, the special restriction permits only the first possibility, which is the only one giving the correct value for $f_j(1)$.

Unfortunately, the special restriction does not fit into the required format for linear programming constraints, so *some* piecewise linear functions cannot be rewritten in a linear programming format. However, *our* $f_j(x_j)$ are assumed to be concave, so $s_{j1} > s_{j2} > \cdots$, so that an algorithm for maximizing $f(\mathbf{x})$ *automatically* gives the highest priority to using $x_{j1}$ when (in effect) increasing $x_j$ from zero, the next highest priority to using $x_{j2}$, and so on, without even including the special restriction explicitly in the model. This observation leads to the following key property.

**Key Property of Separable Programming.** When $f(\mathbf{x})$ and the $g_i(\mathbf{x})$ satisfy the assumptions of separable programming, and when the resulting piecewise linear functions are rewritten as linear functions, deleting the *special restriction* gives a *linear programming model* whose optimal solution automatically satisfies the special restriction.

We shall elaborate further on the logic behind this key property later in this section in the context of a specific example. (Also see Prob. 12.8-6*a*).

To write down the complete linear programming model in the above notation, let $n_j$ be the number of line segments in $f_j(x_j)$ (or the piecewise linear function approximating it), so that

$$x_j = \sum_{k=1}^{n_j} x_{jk}$$

would be substituted throughout the original model and

$$f_j(x_j) = \sum_{k=1}^{n_j} s_{jk}x_{jk}$$

would be substituted[17] into the objective function for $j = 1, 2, \ldots, n$. The resulting model is

$$\text{Maximize} \qquad Z = \sum_{j=1}^{n} \left( \sum_{k=1}^{n_j} s_{jk}x_{jk} \right),$$

[17] If one or more of the $f_j(x_j)$ already are *linear* functions $f_j(x_j) = c_jx_j$, then $n_j = 1$ so neither of these substitutions will be made for $j$.

subject to

$$\sum_{j=1}^{n} a_{ij}\left(\sum_{k=1}^{n_j} x_{jk}\right) \le b_i, \qquad \text{for } i = 1, 2, \ldots, m$$

$$x_{jk} \le u_{jk}, \qquad \text{for } k = 1, 2, \ldots, n_j;\ j = 1, 2, \ldots, n$$

and

$$x_{jk} \ge 0, \qquad \text{for} \qquad k = 1, 2, \ldots, n_j;\ j = 1, 2, \ldots, n.$$

(The $\Sigma_{k=1}^{n_j} x_{jk} \ge 0$ constraints are deleted because they are ensured by the $x_{jk} \ge 0$ constraints.) If some original variable $x_j$ has no upper bound, then $u_{jn_j} = \infty$, so the constraint involving this quantity will be deleted.

An efficient way of solving this model[18] is to use the streamlined version of the simplex method for dealing with upper bound constraints (described in Sec. 7.3). After obtaining an optimal solution for this model, you then would calculate

$$x_j = \sum_{k=1}^{n_j} x_{jk},$$

for $j = 1, 2, \ldots, n$ in order to identify an optimal solution for the original separable programming problem (or its piecewise linear approximation).

**Example.** The Wyndor Glass Co. (see Sec. 3.1) has received a special order for handcrafted goods to be made in Plants 1 and 2 throughout the next 4 months. Filling this order will require borrowing certain employees from the work crews for the regular products, so the remaining workers will need to work overtime to utilize the full production capacity of the plant's machinery and equipment for these regular products. In particular, for the two new regular products discussed in Sec. 3.1, overtime will be required to utilize the last 25 percent of the production capacity available in Plant 1 for product 1 and for the last 50 percent of the capacity available in Plant 2 for product 2. The additional cost of using overtime work will reduce the profit for each unit involved from \$3 to \$2 for product 1 and from \$5 to \$1 for product 2, giving the *profit curves* of Fig. 12.16, both of which fit the form for case 1 of Fig. 12.15.

Management has decided to go ahead and use overtime work rather than hire additional workers during this temporary situation. However, it does insist that the work crew for each product be fully utilized on regular time before any overtime is used. Furthermore, it feels that the current production rates ($x_1 = 2$ for product 1 and $x_2 = 6$ for product 2) should be changed temporarily if this would improve overall profitability. Therefore, it has instructed the OR team to review products 1 and 2 again to determine the most profitable product mix during the next 4 months.

**Formulation.** To refresh your memory, the linear programming model for the original Wyndor Glass Co. problem in Sec. 3.1 is

$$\text{Maximize} \qquad Z = 3x_1 + 5x_2,$$

subject to

$$\begin{aligned} x_1 \qquad &\le 4 \\ 2x_2 &\le 12 \\ 3x_1 + 2x_2 &\le 18 \end{aligned}$$

[18] For a specialized algorithm for solving this model very efficiently, see R. Fourer, "A Specialized Algorithm for Piecewise-Linear Programming III: Computational Analysis and Applications," *Mathematical Programming,* **53:** 213–235, 1992. Also see A. M. Geoffrion, "Objective Function Approximations in Mathematical Programming," *Mathematical Programming,* **13:** 23–37, 1977.

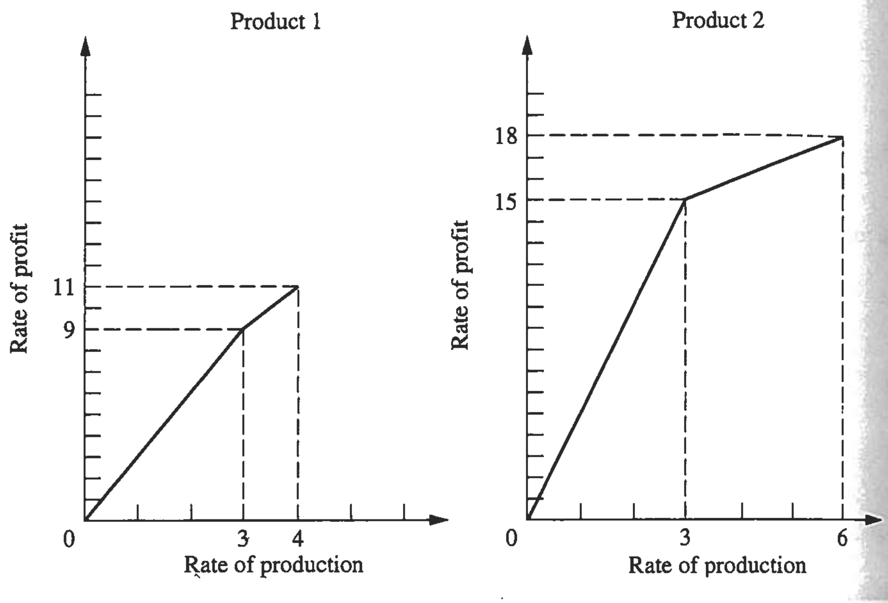


**FIGURE 12.16**
Profit data during the next 4 months for the Wyndor Glass Co.

and

$$x_1 \geq 0, \qquad x_2 \geq 0.$$

We now need to modify this model to fit the new situation described above. For this purpose, let the production rate for product 1 be $x_1 = x_{1R} + x_{1O}$, where $x_{1R}$ is the production rate achieved on regular time and $x_{1O}$ is the incremental production rate from using overtime. Define $x_2 = x_{2R} + x_{2O}$ in the same way for product 2. Thus, in the notation of the general linear programming model for separable programming given just before this example, $n = 2$, $n_1 = 2$, and $n_2 = 2$. Plugging the data given in Fig. 12.16 (including maximum rates of production on regular time and on overtime) into this general model gives the specific model for this application. In particular, the new linear programming problem is to determine the values of $x_{1R}$, $x_{1O}$, $x_{2R}$, and $x_{2O}$ so as to

$$\text{Maximize} \qquad Z = 3x_{1R} + 2x_{1O} + 5x_{2R} + x_{2O},$$

subject to

$$\begin{aligned}
x_{1R} + x_{1O} \qquad\qquad\qquad\qquad &\leq 4\\
2(x_{2R} + x_{2O}) &\leq 12\\
3(x_{1R} + x_{1O}) + 2(x_{2R} + x_{2O}) &\leq 18\\
x_{1R} \leq 3, \qquad x_{1O} \leq 1, \qquad x_{2R} \leq 3, \qquad & x_{2O} \leq 3
\end{aligned}$$

and

$$x_{1R} \geq 0, \qquad x_{1O} \geq 0, \qquad x_{2R} \geq 0, \qquad x_{2O} \geq 0.$$

(Note that the upper bound constraints in the next-to-last row of the model make the first two functional constraints *redundant,* so these two functional constraints can be deleted.)

However, there is one important factor that is not taken into account explicitly in this formulation. Specifically, there is nothing in the model that requires all available regular time for a product to be fully utilized before any overtime is used for that product. In other words, it may be feasible to have $x_{1O} > 0$ even when $x_{1R} < 3$ and to have $x_{2O} > 0$

even when $x_{2R} < 3$. Such solutions would not, however, be acceptable to management. (Prohibiting such solutions is the *special restriction* discussed earlier in this section.)

Now we come to the *key property of separable programming.* Even though the model does not take this factor into account explicitly, the model does take it into account implicitly! Despite the model's having excess "feasible" solutions that actually are unacceptable, any *optimal* solution for the model is *guaranteed* to be a legitimate one that does not replace any available regular-time work with overtime work. (The reasoning here is analogous to that for the Big $M$ method discussed in Sec. 4.6, where excess feasible but *nonoptimal* solutions also were allowed in the model as a matter of convenience.) Therefore, the simplex method can be safely applied to this model to find the most profitable acceptable product mix. The reasons are twofold. First, the two decision variables for each product *always* appear together as a *sum,* $x_{1R} + x_{1O}$ or $x_{2R} + x_{2O}$, in *each* functional constraint other than the upper bound constraints on individual variables. Therefore, it *always* is possible to convert an unacceptable feasible solution to an acceptable one having the same total production rates, $x_1 = x_{1R} + x_{1O}$ and $x_2 = x_{2R} + x_{2O}$, merely by replacing overtime production by regular-time production as much as possible. Second, overtime production is less profitable than regular-time production (i.e., the slope of each profit curve in Fig. 12.16 is a monotonic *decreasing* function of the rate of production), so converting an unacceptable feasible solution to an acceptable one in this way *must* increase the total rate of profit $Z$. Consequently, any feasible solution that uses overtime production for a product when regular-time production is still available *cannot* be optimal with respect to the model.

For example, consider the unacceptable feasible solution $x_{1R} = 1$, $x_{1O} = 1$, $x_{2R} = 1$, $x_{2O} = 3$, which yields a total rate of profit $Z = 13$. The acceptable way of achieving the same total production rates $x_1 = 2$ and $x_2 = 4$ is $x_{1R} = 2$, $x_{1O} = 0$, $x_{2R} = 3$, $x_{2O} = 1$. This latter solution is still feasible, but it also increases $Z$ by $(3 - 2)(1) + (5 - 1)(2) = 9$ to a total rate of profit $Z = 22$.

Similarly, the optimal solution for this model turns out to be $x_{1R} = 3$, $x_{1O} = 1$, $x_{2R} = 3$, $x_{2O} = 0$, which is an acceptable feasible solution.

**Another example** that illustrates the application of separable programming is included in the Worked Examples section of the book's website.

### Extensions

Thus far we have focused on the special case of separable programming where the only nonlinear function is the objective function $f(\mathbf{x})$. Now consider briefly the general case where the constraint functions $g_i(\mathbf{x})$ need not be linear but are convex and separable, so that each $g_i(\mathbf{x})$ can be expressed as a sum of functions of individual variables

$$g_i(\mathbf{x}) = \sum_{j=1}^{n} g_{ij}(x_j),$$

where each $g_{ij}(x_j)$ is a *convex* function. Once again, each of these new functions may be approximated as closely as desired by a *piecewise linear* function (if it is not already in that form). The one new restriction is that for each variable $x_j$ ($j = 1, 2, \ldots, n$), all the piecewise linear approximations of the functions of this variable $[f_j(x_j), g_{1j}(x_j), \ldots, g_{mj}(x_j)]$ must have the *same* breakpoints so that the same new variables $(x_{j1}, x_{j2}, \ldots, x_{jn_j})$ can be used for all these piecewise linear functions. This formulation leads to a linear programming model just like the one given for the special case except that for each $i$ and $j$, the $x_{jk}$ variables now have different coefficients in constraint $i$ [where these coefficients are the corresponding slopes of the piecewise linear function approximating $g_{ij}(x_j)$]. Because the

$g_{ij}(x_j)$ are required to be convex, essentially the same logic as before implies that the key property of separable programming still must hold. (See Prob. 12.8-6*b*.)

One drawback of approximating functions by piecewise linear functions as described in this section is that achieving a close approximation requires a large number of line segments (variables), whereas such a fine grid for the breakpoints is needed only in the immediate neighborhood of an optimal solution. Therefore, more sophisticated approaches that use a succession of *two-segment* piecewise linear functions have been developed[19] to obtain *successively closer approximations* within this immediate neighborhood. This kind of approach tends to be both faster and more accurate in closely approximating an optimal solution.

## 12.9 CONVEX PROGRAMMING

We already have discussed some special cases of convex programming in Secs. 12.4 and 12.5 (unconstrained problems), 12.7 (quadratic objective function with linear constraints), and 12.8 (separable functions). You also have seen some theory for the general case (necessary and sufficient conditions for optimality) in Sec. 12.6. In this section, we briefly discuss some types of approaches used to solve the general convex programming problem [where the objective function $f(\mathbf{x})$ to be maximized is concave and the $g_i(\mathbf{x})$ constraint functions are convex], and then we present one example of an algorithm for convex programming.

There is no single standard algorithm that always is used to solve convex programming problems. Many different algorithms have been developed, each with its own advantages and disadvantages, and research continues to be active in this area. Roughly speaking, most of these algorithms fall into one of the following three categories.

The first category is **gradient algorithms,** where the gradient search procedure of Sec. 12.5 is modified in some way to keep the search path from penetrating any constraint boundary. For example, one popular gradient method is the *generalized reduced gradient* (GRG) method. The Excel Solver uses the GRG method for solving convex programming problems. (As discussed in the next section, Premium Solver also includes an Evolutionary Solver option that is well suited for dealing with *nonconvex* programming problems.)

The second category—**sequential unconstrained algorithms**—includes *penalty function* and *barrier function* methods. These algorithms convert the original constrained optimization problem to a sequence of *unconstrained optimization* problems whose optimal solutions converge to the optimal solution for the original problem. Each of these unconstrained optimization problems can be solved by the kinds of procedures described in Sec. 12.5. This conversion is accomplished by incorporating the constraints into a penalty function (or barrier function) that is subtracted from the objective function in order to impose large penalties for violating constraints (or even being near constraint boundaries). In the latter part of this section, we will describe an algorithm from the 1960s, called the **sequential unconstrained minimization technique** (or **SUMT** for short), that pioneered this category of algorithms. (SUMT also helped to motivate some of the *interior-point methods* for linear programming.)

The third category—**sequential-approximation algorithms**—includes *linear approximation* and *quadratic approximation* methods. These algorithms replace the nonlinear objective function by a succession of linear or quadratic approximations. For linearly constrained optimization problems, these approximations allow repeated application of linear or quadratic programming algorithms. This work is accompanied by other analysis that yields a sequence of solutions that converges to an optimal solution for the original problem. Although these algorithms are particularly suitable for linearly constrained optimization problems,

[19] R. R. Meyer, "Two-Segment Separable Programming," *Management Science*, **25:** 385–395, 1979.

some also can be extended to problems with nonlinear constraint functions by the use of appropriate linear approximations.

As one example of a *sequential-approximation* algorithm, we present here the **Frank-Wolfe algorithm**[20] for the case of *linearly constrained* convex programming (so the constraints are $\mathbf{Ax} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$ in matrix form). This procedure is particularly straightforward; it combines *linear* approximations of the objective function (enabling us to use the simplex method) with a procedure for one-variable unconstrained optimization (such as described in Sec. 12.4).

### A Sequential Linear Approximation Algorithm (Frank-Wolfe)

Given a feasible trial solution $\mathbf{x}'$, the linear approximation used for the objective function $f(\mathbf{x})$ is the first-order Taylor series expansion of $f(\mathbf{x})$ around $\mathbf{x} = \mathbf{x}'$, namely,

$$f(\mathbf{x}') \approx f(\mathbf{x}') + \sum_{j=1}^{n} \frac{\partial f(\mathbf{x}')}{\partial x_j}(x_j - x_j') = f(\mathbf{x}') + \nabla f(\mathbf{x}')(\mathbf{x} - \mathbf{x}'),$$

where these partial derivatives are evaluated at $\mathbf{x} = \mathbf{x}'$. Because $f(\mathbf{x}')$ and $\nabla f(\mathbf{x}')\mathbf{x}'$ have fixed values, they can be dropped to give an equivalent linear objective function

$$g(\mathbf{x}) = \nabla f(\mathbf{x}')\mathbf{x} = \sum_{j=1}^{n} c_j x_j, \qquad \text{where } c_j = \frac{\partial f(\mathbf{x})}{\partial x_j} \qquad \text{at } \mathbf{x} = \mathbf{x}'.$$

The simplex method (or the graphical procedure if $n = 2$) then is applied to the resulting linear programming problem [maximize $g(\mathbf{x})$ subject to the original constraints, $\mathbf{Ax} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$] to find *its* optimal solution $x_{\text{LP}}$. Note that the linear objective function necessarily increases steadily as one moves along the line segment from $\mathbf{x}'$ to $\mathbf{x}_{\text{LP}}$ (which is on the boundary of the feasible region). However, the linear approximation may not be a particularly close one for $\mathbf{x}$ far from $\mathbf{x}'$, so the *nonlinear* objective function may not continue to increase all the way from $\mathbf{x}'$ to $\mathbf{x}_{\text{LP}}$. Therefore, rather than just accepting $\mathbf{x}_{\text{LP}}$ as the next trial solution, we choose the point that maximizes the nonlinear objective function along this line segment. This point may be found by conducting a procedure for one-variable unconstrained optimization of the kind presented in Sec. 12.4, where the one variable for purposes of this search is the fraction $t$ of the total distance from $\mathbf{x}'$ to $\mathbf{x}_{\text{LP}}$. This point then becomes the new trial solution for initiating the next iteration of the algorithm, as just described. The sequence of trial solutions generated by repeated iterations converges to an optimal solution for the original problem, so the algorithm stops as soon as the successive trial solutions are close enough together to have essentially reached this optimal solution.

### Summary of the Frank-Wolfe Algorithm

*Initialization:* Find a feasible initial trial solution $\mathbf{x}^{(0)}$, for example, by applying linear programming procedures to find an initial BF solution. Set $k = 1$.

*Iteration k:*

**1.** For $j = 1, 2, \ldots, n$, evaluate

$$\frac{\partial f(\mathbf{x})}{\partial x_j} \qquad \text{at } \mathbf{x} = \mathbf{x}^{(k-1)}$$

and set $c_j$ equal to this value.

[20] M. Frank and P. Wolfe, "An Algorithm for Quadratic Programming," *Naval Research Logistics Quarterly,* **3:** 95–110, 1956. Although originally designed for quadratic programming, this algorithm is easily adapted to the case of a general concave objective function considered here.

**2.** Find an optimal solution $\mathbf{x}_{\text{LP}}^{(k)}$ for the following linear programming problem.

$$\text{Maximize} \qquad g(\mathbf{x}) = \sum_{j=1}^{n} c_j x_j,$$

subject to

$$\mathbf{Ax} \leq \mathbf{b} \qquad \text{and} \qquad \mathbf{x} \geq \mathbf{0}.$$

**3.** For the variable $t$ $(0 \leq t \leq 1)$, set

$$h(t) = f(\mathbf{x}) \qquad \text{for } \mathbf{x} = \mathbf{x}^{(k-1)} + t(\mathbf{x}_{\text{LP}}^{(k)} - \mathbf{x}^{(k-1)}),$$

so that $h(t)$ gives the value of $f(\mathbf{x})$ on the line segment between $\mathbf{x}^{(k-1)}$ (where $t = 0$) and $\mathbf{x}_{\text{LP}}^{(k)}$ (where $t = 1$). Use some procedure for one-variable unconstrained optimization (see Sec. 12.4) to maximize $h(t)$ over $0 \leq t \leq 1$, and set $\mathbf{x}^{(k)}$ equal to the corresponding $\mathbf{x}$. Go to the stopping rule.

*Stopping rule:* If $\mathbf{x}^{(k-1)}$ and $\mathbf{x}^{(k)}$ are sufficiently close, stop and use $\mathbf{x}^{(k)}$ (or some extrapolation of $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k-1)}, \mathbf{x}^{(k)}$) as your estimate of an optimal solution. Otherwise, reset $k = k + 1$ and perform another iteration.

Now let us illustrate this procedure.

**Example.** Consider the following linearly constrained convex programming problem:

$$\text{Maximize} \qquad f(\mathbf{x}) = 5x_1 - x_1^2 + 8x_2 - 2x_2^2,$$

subject to

$$3x_1 + 2x_2 \leq 6$$

and

$$x_1 \geq 0, \qquad x_2 \geq 0.$$

Note that

$$\frac{\partial f}{\partial x_1} = 5 - 2x_1, \qquad \frac{\partial f}{\partial x_2} = 8 - 4x_2,$$

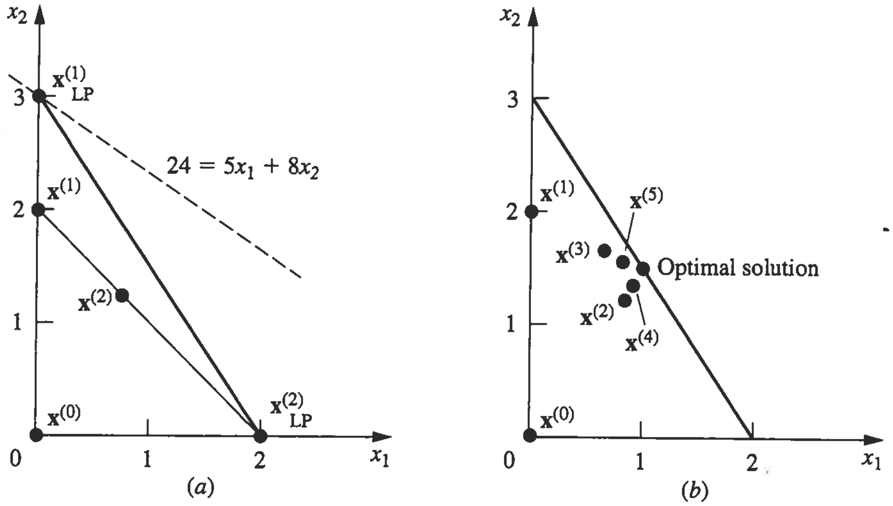
so that the *unconstrained* maximum $\mathbf{x} = (\frac{5}{2}, 2)$ violates the functional constraint. Thus, more work is needed to find the *constrained* maximum.

*Iteration 1:* Because $\mathbf{x} = (0, 0)$ is clearly feasible (and corresponds to the initial BF solution for the linear programming constraints), let us choose it as the initial trial solution $\mathbf{x}^{(0)}$ for the Frank-Wolfe algorithm. Plugging $x_1 = 0$ and $x_2 = 0$ into the expressions for the partial derivatives gives $c_1 = 5$ and $c_2 = 8$, so that $g(\mathbf{x}) = 5x_1 + 8x_2$ is the initial linear approximation of the objective function. Graphically, solving this linear programming problem (see Fig. 12.17*a*) yields $\mathbf{x}_{\text{LP}}^{(1)} = (0, 3)$. For step 3 of the first iteration, the points on the line segment between (0, 0) and (0, 3) shown in Fig. 12.17*a* are expressed by

$$\begin{aligned}(x_1, x_2) &= (0, 0) + t[(0, 3) - (0, 0)] \qquad \text{for } 0 \leq t \leq 1 \\ &= (0, 3t)\end{aligned}$$

as shown in the sixth column of Table 12.6. This expression then gives

$$\begin{aligned}h(t) &= f(0, 3t) = 8(3t) - 2(3t)^2 \\ &= 24t - 18t^2,\end{aligned}$$



■ **FIGURE 12.17**
Illustration of the Frank-Wolfe algorithm.

■ **TABLE 12.6 Application of the Frank-Wolfe algorithm to the example**

| $k$ | $\mathbf{x}^{(k-1)}$ | $c_1$ | $c_2$ | $\mathbf{x}_{\text{LP}}^{(k)}$ | **x for $h(t)$** | $h(t)$ | $t^*$ | $\mathbf{x}^{(k)}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | (0, 0) | 5 | 8 | (0, 3) | (0, 3$t$) | $24t - 18t^2$ | $\frac{2}{3}$ | (0, 2) |
| 2 | (0, 2) | 5 | 0 | (2, 0) | (2$t$, 2 − 2$t$) | $8 + 10t - 12t^2$ | $\frac{5}{12}$ | $\left(\frac{5}{6}, \frac{7}{6}\right)$ |

so that the value $t = t^*$ that maximizes $h(t)$ over $0 \le t \le 1$ may be obtained in this case by setting

$$\frac{dh(t)}{dt} = 24 - 36t = 0,$$

so that $t^* = \frac{2}{3}$. This result yields the next trial solution

$$\mathbf{x}^{(1)} = (0, 0) + \frac{2}{3}[(0, 3) - (0, 0)]$$
$$= (0, 2),$$

which completes the first iteration.

*Iteration 2:* To sketch the calculations that lead to the results in the second row of Table 12.6, note that $\mathbf{x}^{(1)} = (0, 2)$ gives

$$c_1 = 5 - 2(0) = 5,$$
$$c_2 = 8 - 4(2) = 0.$$

For the objective function $g(\mathbf{x}) = 5x_1$, graphically solving the problem over the feasible region in Fig. 12.17$a$ gives $\mathbf{x}_{\text{LP}}^{(2)} = (2, 0)$. Therefore, the expression for the line segment between $\mathbf{x}^{(1)}$ and $\mathbf{x}_{\text{LP}}^{(2)}$ (see Fig. 12.17$a$) is

$$\mathbf{x} = (0, 2) + t[(2, 0) - (0, 2)]$$
$$= (2t, 2 - 2t),$$

so that

$$\begin{aligned} h(t) &= f(2t, 2 - 2t) \\ &= 5(2t) - (2t)^2 + 8(2 - 2t) - 2(2 - 2t)^2 \\ &= 8 + 10t - 12t^2. \end{aligned}$$

Setting

$$\frac{dh(t)}{dt} = 10 - 24t = 0$$

yields $t^* = \frac{5}{12}$. Hence,

$$\begin{aligned} \mathbf{x}^{(2)} &= (0, 2) + \frac{5}{12}[(2, 0) - (0, 2)] \\ &= \left(\frac{5}{6}, \frac{7}{6}\right), \end{aligned}$$

which completes the second iteration.

Figure 12.17*b* shows the trial solutions that are obtained from iterations 3, 4, and 5 as well. You can see how these trial solutions keep alternating between two trajectories that appear to intersect at approximately the point $\mathbf{x} = (1, \frac{3}{2})$. This point is, in fact, the optimal solution, as can be verified by applying the KKT conditions from Sec. 12.6.

This example illustrates a common feature of the Frank-Wolfe algorithm, namely, that the trial solutions alternate between two (or more) trajectories. When they alternate in this way, we can extrapolate the trajectories to their approximate point of intersection to estimate an optimal solution. This estimate tends to be better than using the last trial solution generated. The reason is that the trial solutions tend to converge rather slowly toward an optimal solution, so the last trial solution may still be quite far from optimal.

If you would like to see **another example** of the application of the Frank-Wolfe algorithm, one is included in the Worked Examples section of the book's website. Your OR Tutor provides **an additional example** as well. IOR Tutorial also includes an interactive procedure for this algorithm.

### Some Other Algorithms

We should emphasize that the Frank-Wolfe algorithm is just one example of sequential-approximation algorithms. Many of these algorithms use *quadratic* instead of *linear* approximations at each iteration because quadratic approximations provide a considerably closer fit to the original problem and thus enable the sequence of solutions to converge considerably more rapidly toward an optimal solution than was the case in Fig. 12.17*b*. For this reason, even though sequential linear approximation methods such as the Frank-Wolfe algorithm are relatively straightforward to use, *sequential quadratic approximation methods* now are generally preferred in actual applications. Popular among these are the *quasi-Newton* (or *variable metric*) methods. As already mentioned in Sec. 12.5, these methods use a fast approximation of *Newton's method* and then further adapt this method to take the constraints of the problem into account. To speed up the algorithm, quasi-Newton methods compute a quadratic approximation to the curvature of a nonlinear function without explicitly calculating second (partial) derivatives. (For linearly constrained optimization problems, this nonlinear function is just the objective function; whereas with nonlinear constraints, it is the Lagrangian function described in Appendix 3.) Some quasi-Newton algorithms do not even explicitly form and solve an approximating quadratic programming problem at each iteration, but instead incorporate some of the basic ingredients of *gradient algorithms*. (See Selected Reference 2 for further details about sequential-approximation algorithms.)

We turn now from sequential-approximation algorithms to *sequential unconstrained algorithms*. As mentioned at the beginning of the section, algorithms of the latter type solve the original constrained optimization problem by instead solving a sequence of *unconstrained* optimization problems.

A particularly prominent sequential unconstrained algorithm that has been widely used since its development in the 1960s is the *sequential unconstrained minimization technique* (or *SUMT* for short).[21] There actually are two main versions of SUMT, one of which is an *exterior-point* algorithm that deals with *infeasible* solutions while using a *penalty function* to force convergence to the feasible region. We shall describe the other version, which is an *interior-point* algorithm that deals directly with *feasible* solutions while using a *barrier function* to force staying inside the feasible region. Although SUMT was originally presented as a minimization technique, we shall convert it to a maximization technique in order to be consistent with the rest of the chapter. Therefore, we continue to assume that the problem is in the form given at the beginning of the chapter and that all the functions are differentiable.

## Sequential Unconstrained Minimization Technique (SUMT)

As the name implies, SUMT replaces the original problem by a *sequence* of *unconstrained* optimization problems whose solutions *converge* to a solution (local maximum) of the original problem. This approach is very attractive because unconstrained optimization problems are much easier to solve (see Sec. 12.5) than those with constraints. Each of the unconstrained problems in this sequence involves choosing a (successively smaller) strictly positive value of a scalar $r$ and then solving for $\mathbf{x}$ so as to

$$\text{Maximize} \qquad P(\mathbf{x}; r) = f(\mathbf{x}) - rB(\mathbf{x}).$$

Here $B(\mathbf{x})$ is a **barrier function** that has the following properties (for $\mathbf{x}$ that are feasible for the original problem):

1. $B(\mathbf{x})$ is *small* when $\mathbf{x}$ is *far* from the boundary of the feasible region.
2. $B(\mathbf{x})$ is *large* when $\mathbf{x}$ is *close* to the boundary of the feasible region.
3. $B(\mathbf{x}) \rightarrow \infty$ as the distance from the (nearest) boundary of the feasible region $\rightarrow 0$.

Thus, by starting the search procedure with a *feasible* initial trial solution and then attempting to increase $P(\mathbf{x}; r)$, $B(\mathbf{x})$ provides a *barrier* that prevents the search from ever crossing (or even reaching) the boundary of the feasible region for the original problem.

The most common choice of $B(\mathbf{x})$ is

$$B(\mathbf{x}) = \sum_{i=1}^{m} \frac{1}{b_i - g_i(\mathbf{x})} + \sum_{j=1}^{n} \frac{1}{x_j}.$$

For feasible values of $\mathbf{x}$, note that the denominator of each term is proportional to the distance of $\mathbf{x}$ from the constraint boundary for the corresponding functional or nonnegativity constraint. Consequently, *each* term is a *boundary repulsion term* that has all the preceding three properties with respect to this particular constraint boundary. Another attractive feature of this $B(\mathbf{x})$ is that when all the assumptions of *convex programming* are satisfied, $P(\mathbf{x}; r)$ is a *concave* function.

Because $B(\mathbf{x})$ keeps the search away from the boundary of the feasible region, you probably are asking the very legitimate question: What happens if the desired solution lies there? This concern is the reason that SUMT involves solving a *sequence* of these unconstrained optimization problems for successively smaller values of $r$ approaching zero (where the final trial solution from each one becomes the initial trial solution for the next). For example, each new $r$ might be obtained from the preceding one by multiplying by a

[21] See Selected Reference 1.

constant $\theta$ $(0 < \theta < 1)$, where a typical value is $\theta = 0.01$. As $r$ approaches 0, $P(\mathbf{x}; r)$ approaches $f(\mathbf{x})$, so the corresponding local maximum of $P(\mathbf{x}; r)$ converges to a local maximum of the original problem. Therefore, it is necessary to solve only enough unconstrained optimization problems to permit extrapolating their solutions to this limiting solution.

How many are enough to permit this extrapolation? When the original problem satisfies the assumptions of convex programming, useful information is available to guide us in this decision. In particular, if $\bar{x}$ is a global maximizer of $P(\mathbf{x}; r)$, then

$$f(\bar{\mathbf{x}}) \leq f(\mathbf{x}^*) \leq f(\bar{\mathbf{x}}) + rB(\bar{\mathbf{x}}),$$

where $\mathbf{x}^*$ is the (unknown) *optimal* solution for the original problem. Thus, $rB(\bar{\mathbf{x}})$ is the *maximum error* (in the value of the objective function) that can result by using $\bar{\mathbf{x}}$ to approximate $\mathbf{x}^*$, and extrapolating beyond $\bar{\mathbf{x}}$ to increase $f(\mathbf{x})$ further decreases this error. If an *error tolerance* is established in advance, then you can stop as soon as $rB(\bar{\mathbf{x}})$ is less than this quantity.

### Summary of SUMT

*Initialization:* Identify a *feasible* initial trial solution $\mathbf{x}^{(0)}$ that is not on the boundary of the feasible region. Set $k = 1$ and choose appropriate strictly positive values for the initial $r$ and for $\theta < 1$ (say, $r = 1$ and $\theta = 0.01$).[22]

*Iteration k:* Starting from $\mathbf{x}^{(k-1)}$, apply a multivariable unconstrained optimization procedure (e.g., the gradient search procedure) such as described in Sec. 12.5 to find a local maximum $\mathbf{x}^{(k)}$ of

$$P(\mathbf{x}; r) = f(\mathbf{x}) - r\left[\sum_{i=1}^{m} \frac{1}{b_i - g_i(\mathbf{x})} + \sum_{j=1}^{n} \frac{1}{x_j}\right].$$

*Stopping rule:* If the change from $\mathbf{x}^{(k-1)}$ to $\mathbf{x}^{(k)}$ is negligible, stop and use $\mathbf{x}^{(k)}$ (or an extrapolation of $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k-1)}, \mathbf{x}^{(k)}$) as your estimate of a *local maximum* of the original problem. Otherwise, reset $k = k + 1$ and $r = \theta r$ and perform another iteration.

Finally, we should note that SUMT also can be extended to accommodate *equality* constraints $g_i(\mathbf{x}) = b_i$. One standard way is as follows. For each equality constraint,

$$\frac{-[b_i - g_i(\mathbf{x})]^2}{\sqrt{r}} \qquad \text{replaces} \qquad \frac{-r}{b_i - g_i(\mathbf{x})}$$

in the expression for $P(\mathbf{x}; r)$ given under "Summary of SUMT," and then the same procedure is used. The numerator $-[b_i - g_i(\mathbf{x})]^2$ imposes a large penalty for deviating substantially from satisfying the equality constraint, and then the denominator tremendously increases this penalty as $r$ is decreased to a tiny amount, thereby forcing the sequence of trial solutions to converge toward a point that satisfies the constraint.

SUMT has been widely used because of its simplicity and versatility. However, numerical analysts have found that it is relatively prone to *numerical instability,* so considerable caution is advised. For further information on this issue as well as similar analyses for alternative algorithms, see Selected Reference 3.

**Example.** To illustrate SUMT, consider the following two-variable problem:

$$\text{Maximize} \qquad f(\mathbf{x}) = x_1 x_2,$$

subject to

$$x_1^2 + x_2 \leq 3$$

[22] A reasonable criterion for choosing the initial $r$ is one that makes $rB(\mathbf{x})$ about the same order of magnitude as $f(\mathbf{x})$ for feasible solutions $\mathbf{x}$ that are not particularly close to the boundary.

**TABLE 12.7** Illustration of SUMT

| $k$ | $r$ | $x_1^{(k)}$ | $x_2^{(k)}$ |
|---|---|---|---|
| 0 | | 1 | 1 |
| 1 | 1 | 0.90 | 1.36 |
| 2 | $10^{-2}$ | 0.987 | 1.925 |
| 3 | $10^{-4}$ | 0.998 | 1.993 |
| | | ↓ | ↓ |
| | | 1 | 2 |

and

$$x_1 \geq 0, \qquad x_2 \geq 0.$$

Even though $g_1(\mathbf{x}) = x_1^2 + x_2$ is convex (because each term is convex), this problem is a *nonconvex* programming problem because $f(\mathbf{x}) = x_1x_2$ is *not* concave (see Appendix 2). However, the problem is close enough to being a convex programming problem that SUMT necessarily will still converge to an optimal solution in this case. (We will discuss nonconvex programming further, including the role of SUMT in dealing with such problems, in the next section.)

For the initialization, $(x_1, x_2) = (1, 1)$ is one obvious feasible solution that is not on the boundary of the feasible region, so we can set $\mathbf{x}^{(0)} = (1, 1)$. Reasonable choices for $r$ and $\theta$ are $r = 1$ and $\theta = 0.01$.

For each iteration,

$$P(\mathbf{x}; r) = x_1x_2 - r\left(\frac{1}{3 - x_1^2 - x_2} + \frac{1}{x_1} + \frac{1}{x_2}\right).$$

With $r = 1$, applying the gradient search procedure starting from (1, 1) to maximize this expression eventually leads to $\mathbf{x}^{(1)} = (0.90, 1.36)$. Resetting $r = 0.01$ and restarting the gradient search procedure from (0.90, 1.36) then lead to $\mathbf{x}^{(2)} = (0.983, 1.933)$. One more iteration with $r = 0.01(0.01) = 0.0001$ leads from $\mathbf{x}^{(2)}$ to $\mathbf{x}^{(3)} = (0.998, 1.994)$. This sequence of points, summarized in Table 12.7, quite clearly is converging to (1, 2). Applying the KKT conditions to this solution verifies that it does indeed satisfy the necessary condition for optimality. Graphical analysis demonstrates that $(x_1, x_2) = (1, 2)$ is, in fact, a global maximum (see Prob. 12.9-13*b*).

For this problem, there are no local maxima other than $(x_1, x_2) = (1, 2)$, so reapplying SUMT from various feasible initial trial solutions always leads to this same solution.[23]

The Worked Examples section of the book's website provides **another example** that illustrates the application of SUMT to a convex programming problem in minimization form. You also can go to your OR Tutor to see **an additional example**. An automatic procedure for executing SUMT is included in IOR Tutorial.

## Some Software Options for Convex Programming

As indicated at the end of Sec. 12.7, both Excel and LINGO can solve convex programming problems, but the student version of LINDO and CPLEX cannot except for the special case of quadratic programming (which includes the first example in this

[23] The technical reason is that $f(\mathbf{x})$ is a (strictly) *quasiconcave* function that shares the property of concave functions that a local maximum always is a global maximum. For further information, see M. Avriel, W. E. Diewert, S. Schaible, and I. Zang, *Generalized Concavity*, Plenum, New York, 1985.

section). Details for this example are given in the Excel and LINGO files for this chapter in your OR Courseware. The professional version of MPL supports a large number of solvers, including some that can handle convex programming. One of these, called CONOPT, is included with the student version of MPL that is on the book's website. The convex programming examples that are formulated in this chapter's MPL file have been solved with this solver after setting the model type to Nonlinear (as described at the end of Sec. 12.7).

## 12.10 NONCONVEX PROGRAMMING (WITH SPREADSHEETS)

The assumptions of convex programming (the function $f(\mathbf{x})$ to be maximized is *concave* and all the $g_i(\mathbf{x})$ constraint functions are *convex*) are very convenient ones, because they ensure that any *local maximum* also is a *global maximum.* (If the objective is to *minimize* $f(\mathbf{x})$ instead, then convex programming assumes that $f(\mathbf{x})$ is *convex,* and so on, which ensures that a *local minimum* also is a *global minimum.*) Unfortunately, the nonlinear programming problems that arise in practice frequently fail to satisfy these assumptions. What kind of approach can be used to deal with such *nonconvex programming* problems?

### The Challenge of Solving Nonconvex Programming Problems

There is no single answer to the above question because there are so many different types of nonconvex programming problems. Some are much more difficult to solve than others. For example, a maximization problem where the objective function is nearly *convex* generally is much more difficult than one where the objective function is nearly concave. (The SUMT example in Sec. 12.9 illustrated a case where the objective function was so close to being concave that the problem could be treated as if it were a convex programming problem.) Similarly, having a feasible region that is *not* a convex set (because some of the $g_i(\mathbf{x})$ functions are not convex) generally is a major complication. Dealing with functions that are not differentiable, or perhaps not even continuous, also tends to be a major complication.

The goal of much ongoing research is to develop efficient **global optimization** procedures for finding a *globally optimal solution* for various types of nonconvex programming problems, and some progress has been made. As one example, LINDO Systems (which produces LINDO, LINGO, and What's Best) now has incorporated a global optimizer into its advanced solver that is shared by some of its software products. In particular, LINGO and What's Best have a multistart option to automatically generate a number of starting points for their nonlinear programming solver in order to quickly find a good solution. If the global option is checked, they next employ the global optimizer. The global optimizer converts a nonconvex programming problem (including even those whose formulation includes logic functions such as IF, AND, OR, and NOT) into several subproblems that are convex programming relaxations of portions of the original problem. The branch-and-bound technique then is used to exhaustively search over the subproblems. Once the procedure runs to completion, the solution found is guaranteed to be a globally optimal solution. (The other possible conclusion is that the problem has no feasible solutions.) The student version of this global optimizer is included in the version of LINGO that is provided on the book's website. However, it is limited to relatively small problems (a maximum of five nonlinear variables out of 500 variables total). The professional version of the global optimizer has successfully solved some much larger problems.

Similarly, MPL now supports a global optimizer called LGO. The student version of LGO is available to you as one of the MPL solvers provided on the book's website. LGO also can be used to solve convex programming problems.

A variety of approaches to global optimization (such as the one incorporated into LINGO described above) are being tried. We will not attempt to survey this advanced topic in any depth. (See Selected Reference 5 for some details.) We instead will begin with a simple case and then introduce a more general approach at the end of the section. We will illustrate our methodology with spreadsheets and Excel software, but other software packages also can be used.

### Using the Excel Solver to Find Local Optima

We now will focus on straightforward approaches to relatively simple types of nonconvex programming problems. In particular, we will consider (maximization) problems where the objective function is nearly concave either over the entire feasible region or within major portions of the feasible region. We also will ignore the added complexity of having nonconvex constraint functions $g_i(\mathbf{x})$ by simply using linear constraints. We will begin by illustrating what can be accomplished by simply applying some algorithm for convex programming to such problems. Although any such algorithm (such as those described in Sec. 12.9) could be selected, we will use the convex programming algorithm that is employed by the Excel Solver for nonlinear programming problems.

For example, consider the following one-variable nonconvex programming problem:

$$\text{Maximize} \qquad Z = 0.5x^5 - 6x^4 + 24.5x^3 - 39x^2 + 20x,$$

subject to

$$x \le 5$$
$$x \ge 0,$$

where $Z$ represents the profit in dollars. Figure 12.18 shows a plot of the profit over the feasible region that demonstrates how highly nonconvex this function is. However, if this graph were not available, it might not be immediately clear that this is *not* a convex programming problem since a little analysis is required to verify that the objective function is not concave over the feasible region. Therefore, suppose that the Excel Solver, which is designed for solving convex programming problems, is applied to this example. Figure 12.19 demonstrates what a difficult time the Excel Solver has in attempting to cope with this

**FIGURE 12.18**
The profit graph for a nonconvex programming example.

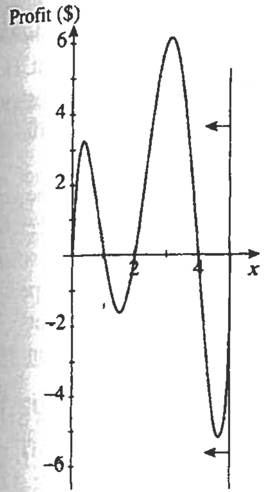
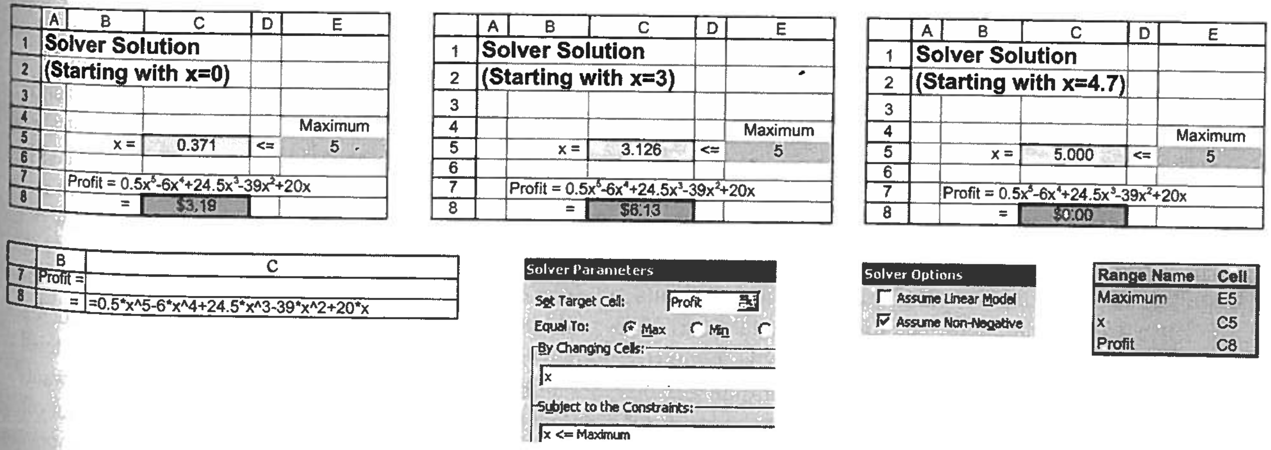


**FIGURE 12.19**
An example of a nonconvex programming problem (depicted in Fig. 12.18) where the Excel Solver obtains three different solutions when it starts with three different initial solutions.



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Solver Solution | | | | |
| 2 | (Starting with x=0) | | | | |
| 3 | | | | | |
| 4 | | | | | Maximum |
| 5 | | x = | 0.371 | <= | 5 |
| 6 | | | | | |
| 7 | | Profit = 0.5x^5-6x^4+24.5x^3-39x^2+20x | | | |
| 8 | | = | $3.19 | | |

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Solver Solution | | | | |
| 2 | (Starting with x=3) | | | | |
| 3 | | | | | |
| 4 | | | | | Maximum |
| 5 | | x = | 3.126 | <= | 5 |
| 6 | | | | | |
| 7 | | Profit = 0.5x^5-6x^4+24.5x^3-39x^2+20x | | | |
| 8 | | = | $6.13 | | |

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Solver Solution | | | | |
| 2 | (Starting with x=4.7) | | | | |
| 3 | | | | | |
| 4 | | | | | Maximum |
| 5 | | x = | 5.000 | <= | 5 |
| 6 | | | | | |
| 7 | | Profit = 0.5x^5-6x^4+24.5x^3-39x^2+20x | | | |
| 8 | | = | $0.00 | | |

| | B | C |
|---|---|---|
| 7 | Profit = | |
| 8 | = | =0.5*x^5-6*x^4+24.5*x^3-39*x^2+20*x |

Solver Parameters
Set Target Cell: Profit
Equal To: ● Max ○ Min ○
By Changing Cells:
x
Subject to the Constraints:
x <= Maximum

Solver Options
□ Assume Linear Model
☑ Assume Non-Negative

| Range Name | Cell |
|---|---|
| Maximum | E5 |
| x | C5 |
| Profit | C8 |

problem. The model is straightforward to formulate in a spreadsheet, with $x$ (C5) as the changing cell and Profit (C8) as the target cell. (Note that the Solver option, Assume Linear Model, is *not* chosen in this case because this is not a linear programming model.) When $x = 0$ is entered as the initial value in the changing cell, the left spreadsheet in Fig. 12.19 shows that the solver then indicates that $x = 0.371$ is the optimal solution with Profit = \$3.19. However, if $x = 3$ is entered as the initial value instead, as in the middle spreadsheet in Fig. 12.19, Solver obtains $x = 3.126$ as the optimal solution with Profit = \$6.13. Trying still another initial value of $x = 4.7$ in the right spreadsheet, Solver now indicates an optimal solution of $x = 5$ with Profit = \$0. What is going on here?

Figure 12.18 helps to explain Solver's difficulties with this problem. Starting at $x = 0$, the profit graph does indeed climb to a peak at $x = 0.371$, as reported in the left spreadsheet of Fig. 12.19. Starting at $x = 3$ instead, the graph climbs to a peak at $x = 3.126$, which is the solution found in the middle spreadsheet. Using the right spreadsheet's starting solution of $x = 4.7$, the graph climbs until it reaches the boundary imposed by the $x \leq 5$ constraint, so $x = 5$ is the peak in that direction. These three peaks are the *local maxima* (or *local optima*) because each one is a maximum of the graph within a local neighborhood of that point. However, only the largest of these local maxima is the *global maximum,* that is, the highest point on the entire graph. Thus, the middle spreadsheet in Fig. 12.19 did succeed in finding the globally optimal solution at $x = 3.126$ with Profit = \$6.13.

The Excel Solver uses the *generalized reduced gradient method,* which adapts the gradient search method described in Sec. 12.5 to solve convex programming problems. Therefore, this algorithm can be thought of as a hill-climbing procedure. It starts at the initial solution entered into the changing cells and then begins climbing that hill until it reaches the peak (or is blocked from climbing further by reaching the boundary imposed by the constraints). The procedure terminates when it reaches this peak (or boundary) and reports this solution. It has no way of detecting whether there is a taller hill somewhere else on the profit graph.

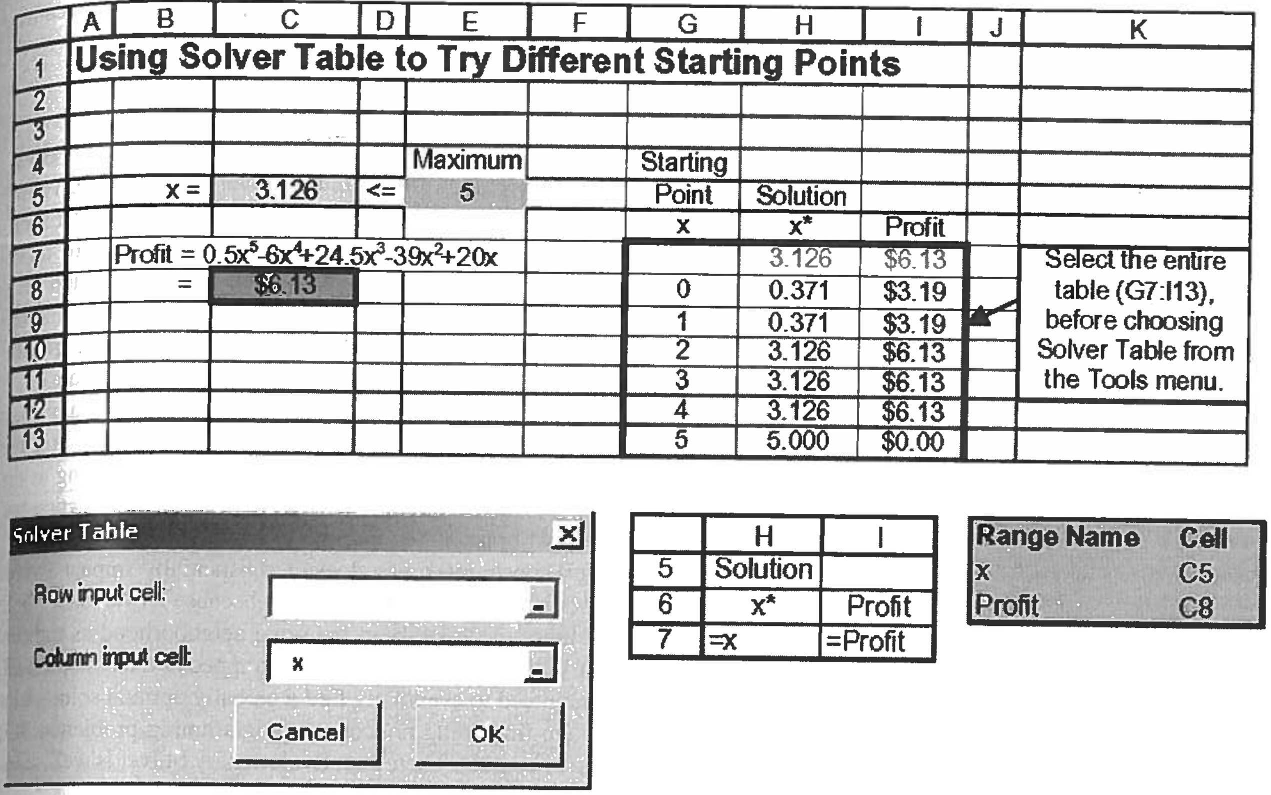
The same thing would happen with any other hill-climbing procedure, such as SUMT (described in Sec. 12.9), that stops when it finds a local maximum. Thus, if SUMT were to be applied to this example with each of the three initial trial solutions used in Fig. 12.19, it would find the same three local maxima found by the Excel Solver.

## A More Systematic Approach to Finding Local Optima

A common approach to "easy" nonconvex programming problems is to apply some algorithmic hill-climbing procedure that will stop when it finds a *local maximum* and then to restart it a number of times from a variety of initial trial solutions (either chosen randomly or as a systematic cross-section) in order to find as many distinct local maxima as possible. The best of these local maxima is then chosen for implementation. Normally, the hill-climbing procedure is one that has been designed to find a global maximum when all the assumptions of convex programming hold, but it also can operate to find a local maximum when they do not.

When employing the Excel Solver, a systematic way of applying this approach is to use the Solver Table add-in that is provided in your OR Courseware. To demonstrate, we will continue to use the spreadsheet model shown in Fig. 12.19. Figure 12.20 displays how the Solver Table is used to try six different starting points (0, 1, 2, 3, 4, and 5) as the initial trial solutions for this model by executing the following steps. In the first row of the table, enter formulas that refer to the changing cell, $x$ (C5), and the target cell, Profit (C8). Then, select the entire table (G7:I13) and choose Solver Table from the Add-Ins tab (for Excel 2007) or the Tools menu (for earlier versions of Excel). The column input cell entered in the Solver Table dialogue box is the changing cell $x$ (C5), since this is where we want the different starting points in the first column of the table to be entered. (No row input cell is entered in this dialogue box since only a column is being used to list the starting points.)

■ FIGURE
An applicati
Figs. 12.18



| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Using Solver Table to Try Different Starting Points** | | | | | | | | | | |
| 2 | | | | | | | | | | | |
| 3 | | | | | | | | | | | |
| 4 | | | | | Maximum | | Starting | | | | |
| 5 | | x = | 3.126 | <= | 5 | | Point | Solution | | | |
| 6 | | | | | | | x | x* | Profit | | |
| 7 | | Profit = $0.5x^5$-$6x^4$+$24.5x^3$-$39x^2$+$20x$ | | | | | | 3.126 | $6.13 | | |
| 8 | | = | $6.13 | | | | 0 | 0.371 | $3.19 | | |
| 9 | | | | | | | 1 | 0.371 | $3.19 | | |
| 10 | | | | | | | 2 | 3.126 | $6.13 | | |
| 11 | | | | | | | 3 | 3.126 | $6.13 | | |
| 12 | | | | | | | 4 | 3.126 | $6.13 | | |
| 13 | | | | | | | 5 | 5.000 | $0.00 | | |

Select the entire table (G7:I13), before choosing Solver Table from the Tools menu.

Solver Table

Row input cell:

Column input cell: x

| | H | I |
|---|---|---|
| 5 | Solution | |
| 6 | x* | Profit |
| 7 | =x | =Profit |

| Range Name | Cell |
|---|---|
| x | C5 |
| Profit | C8 |

■ **FIGURE 12.20**
An application of the Solver Table (an Excel add-in provided in your OR Courseware) to the example considered in Figs. 12.18 and 12.19.

Clicking OK then causes the Solver Table to re-solve the problem for all these starting points in the first column and fill in the corresponding results (the local maximum for $x$ and Profit referred to in the first row) in the other columns of the table.

The example has only one variable and so only one changing cell. However, the Solver Table also can be used to try multiple starting points for problems with two variables (changing cells). This is done by using the first row and first column of the table to specify different starting points for the two changing cells. Enter an equation referring to the target cell in the upper left-hand corner of the table. Select the entire table and choose Solver Table from the Add-Ins tab or Tools menu, with the two changing cells selected as the column input cell and row input cell. The Solver Table then re-solves the problem for each combination of starting points of the two changing cells and fills in the body of the table with the objective function value of the solution that is found (a local optimum) for each of these combinations. (See Sec. 6.8 for more details about setting up a two-dimensional Solver Table.)

For problems with more than two variables (changing cells), this same approach still can be used to try multiple starting points for any two of the changing cells at a time. However, this becomes a very cumbersome way of trying a broad range of starting points for all the changing cells when there are more than three or four of these cells.

Unfortunately, there generally is no guarantee of finding a globally optimal solution, no matter how many different starting points are tried. Also, if the profit graphs are not smooth (e.g., if they have discontinuities or kinks), then Solver may not even be able to find local optima. Fortunately, Excel's *Premium Solver* provides another search procedure, called *Evolutionary Solver,* to attempt to solve these somewhat more difficult nonconvex programming problems.

### Evolutionary Solver

Frontline Systems, the developer of the standard Solver included with Excel, has developed Premium versions of Solver. One version of Premium Solver (Premium Solver for Education) is available in your OR courseware (but not included with standard Excel). Every version of Premium Solver, including this one, adds a search procedure called **Evolutionary Solver** in the set of tools available to search for an optimal solution for a model. The philosophy of Evolutionary Solver is based on genetics, evolution, and the survival of the fittest. Hence, this type of algorithm is sometimes called a **genetic algorithm.** We will devote Sec. 13.4 to describing how genetic algorithms operate.

Evolutionary Solver has three crucial advantages over the standard Solver (or any other convex programming algorithm) for solving nonconvex programming problems. First, the complexity of the objective function does not impact Evolutionary Solver. As long as the function can be evaluated for a given trial solution, it does not matter if the function has kinks or discontinuities or many local optima. Second, the complexity of the given constraints (including even nonconvex constraints) also doesn't substantially impact Evolutionary Solver (although the *number* of constraints does). Third, because it evaluates whole populations of trial solutions that aren't necessarily in the same neighborhood as the current best trial solution, Evolutionary Solver keeps from getting trapped at a local optimum. In fact, Evolutionary Solver is guaranteed to eventually find a globally optimal solution for any nonlinear programming problem (including nonconvex programming problems), if it is run forever (which is impractical of course). Therefore, Evolutionary Solver is well suited for dealing with many relatively small nonconvex programming problems.

On the other hand, it must be pointed out that Evolutionary Solver is not a panacea. First, it can take *much* longer than the standard Solver to find a final solution. Second, Evolutionary Solver does not perform well on models that have many constraints. Third, Evolutionary Solver is a random process, so running it again on the same model usually will yield a different final solution. Finally, the best solution found typically is not quite optimal (although it may be very close). Evolutionary Solver does not continuously move toward better solutions. Rather it is more like an intelligent search engine, trying out different random solutions. Thus, while it is quite likely to end up with a solution that is very close to optimal, it almost never returns the exact globally optimal solution on most types of nonlinear programming problems. Consequently, if often can be beneficial to run the standard Solver (GRG Nonlinear option) after the Evolutionary Solver, starting with the final solution obtained by the Evolutionary Solver, to see if this solution can be improved by searching around its neighborhood.

## 12.11 CONCLUSIONS

Practical optimization problems frequently involve *nonlinear* behavior that must be taken into account. It is sometimes possible to *reformulate* these nonlinearities to fit into a linear programming format, as can be done for *separable programming* problems. However, it is frequently necessary to use a *nonlinear programming* formulation.

In contrast to the case of the simplex method for linear programming, there is no efficient all-purpose algorithm that can be used to solve all nonlinear programming problems. In fact, some of these problems cannot be solved in a very satisfactory manner by any method. However, considerable progress has been made for some important classes of problems, including *quadratic programming, convex programming,* and certain special types of *nonconvex programming.* A variety of algorithms that frequently perform well are available for these cases. Some of these algorithms incorporate highly efficient procedures for *unconstrained optimization* for a portion of each iteration, and some use a succession of linear or quadratic approximations to the original problem.

There has been a strong emphasis in recent years on developing high-quality, reliable *software packages* for general use in applying the best of these algorithms. For example, several powerful software packages have been developed in the Systems Optimization Laboratory at Stanford University. These packages are widely used elsewhere for solving many of the types of problems discussed in this chapter (as well as linear programming problems). The steady improvements being made in both algorithmic techniques and software now are bringing some rather large problems into the range of computational feasibility.

Research in nonlinear programming remains very active.

## SELECTED REFERENCES

1. Fiacco, A. V., and G. P. McCormick: *Nonlinear Programming: Sequential Unconstrained Minimization Techniques,* Classics in Applied Mathematics 4, Society for Industrial and Applied Mathematics, Philadelphia, 1990. (Reprint of a classic book published in 1968.)
2. Fletcher, R.: *Practical Methods of Optimization*, 2nd ed., Wiley, New York, 2000.
3. Gill, P. E., W. Murray, and M. H. Wright: *Practical Optimization,* Academic Press, London, 1981.
4. Hillier, F. S., and M. S. Hillier: *Introduction to Management Science: A Modeling and Case Studies Approach with Spreadsheets,* 3rd ed., McGraw-Hill/Irwin, Burr Ridge, IL, 2008, chap. 8.
5. Leyffer, S., and J. More (eds.): Special Issue on Deterministic Global Optimization and Applications, *Mathematical Programming*, Series B, **103**(2), June 2005.
6. Luenberger, D., and Y. Ye: *Linear and Nonlinear Programming*, 3rd ed., Springer, New York, 2008.
7. Miller, R. E.: *Optimization: Foundations and Applications,* Wiley, New York, 1999.
8. Rardin, D.: *Optimization in Operations Research*, Prentice-Hall, Upper Saddle River, NJ, 1998.

## LEARNING AIDS FOR THIS CHAPTER ON OUR WEBSITE (www.mhhe.com/hillier)

### Worked Examples:

Examples for Chapter 12

### Demonstration Examples in OR Tutor:

Gradient Search Procedure
Frank-Wolfe Algorithm
Sequential Unconstrained Minimization Technique—SUMT

### Interactive Procedures in IOR Tutorial:

Interactive One-Dimensional Search Procedure
Interactive Gradient Search Procedure
Interactive Modified Simplex Method
Interactive Frank-Wolfe Algorithm

### Automatic Procedures in IOR Tutorial:

Automatic Gradient Search Procedure
Sequential Unconstrained Minimization Technique—SUMT