8

NONLINEAR PROGRAMMING WITH CONSTRAINTS

CHAPTER 1 PRESENTS some examples of the constraints that occur in optimization problems. Constraints are classified as being inequality constraints or equality constraints, and as linear or nonlinear. Chapter 7 described the simplex method for solving problems with linear objective functions subject to linear constraints. This chapter treats more difficult problems involving minimization (or maximization) of a nonlinear objective function subject to linear or nonlinear constraints:

$$\begin{aligned} \text{Minimize:} \quad & f(\mathbf{x}) \qquad \mathbf{x} = [x_1\ x_2 \cdots x_n]^T \\ \text{Subject to:} \quad & h_i(\mathbf{x}) = b_i \quad i = 1, 2, \ldots, m \\ & g_j(\mathbf{x}) \le c_j \quad j = 1, \ldots, r \end{aligned} \tag{8.1}$$

The inequality constraints in Problem (8.1) can be transformed into equality constraints as explained in Section 8.4, so we focus first on problems involving only equality constraints.

8.1 DIRECT SUBSTITUTION

One method of handling just one or two linear or nonlinear equality constraints is to solve explicitly for one variable and eliminate that variable from the problem formulation. This is done by direct substitution in the objective function and constraint equations in the problem. In many problems elimination of a single equality constraint is often superior to an approach in which the constraint is retained and some constrained optimization procedure is executed. For example, suppose you want to minimize the following objective function that is subject to a single equality constraint

$$\text{Minimize:} \quad f(\mathbf{x}) = 4x_1^2 + 5x_2^2 \tag{8.2a}$$

$$\text{Subject to:} \quad 2x_1 + 3x_2 = 6 \tag{8.2b}$$

Either x_1 or x_2 can be eliminated without difficulty. Solving for x_1,

$$x_1 = \frac{6 - 3x_2}{2} \tag{8.3}$$

we can substitute for x_1 in Equation (8.2a). The new equivalent objective function in terms of a single variable x_2 is

$$f(x_2) = 14x_2^2 - 36x_2 + 36 \tag{8.4}$$

The constraint in the original problem has now been eliminated, and $f(x_2)$ is an unconstrained function with 1 degree of freedom (one independent variable). Using constraints to eliminate variables is the main idea of the generalized reduced gradient method, as discussed in Section 8.7.

We can now minimize the objective function (8.4), by setting the first derivative of f equal to zero, and solving for the optimal value of x_2:

$$\frac{df(x_2)}{dx_2} = 28x_2 - 36 = 0 \quad x_2^* = 1.286$$

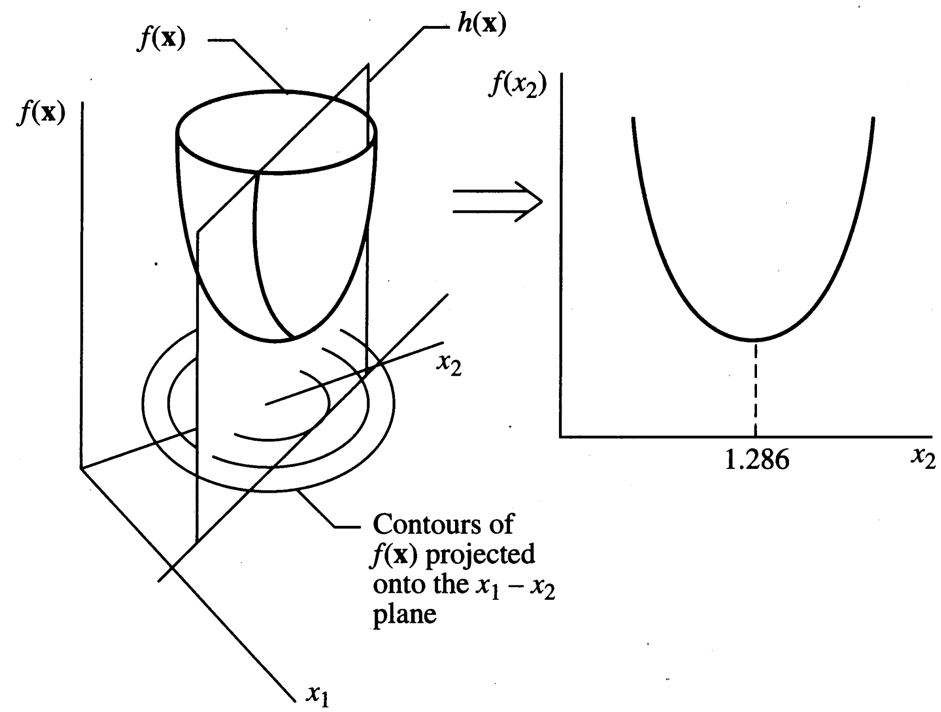

FIGURE 8.1
Graphical representation of a function of two variables reduced to a function of one variable by direct substitution. The unconstrained minimum is at (0,0), the center of the contours.

Once x_2^* is obtained, then, x_1^* can be directly obtained via the constraint (8.2b):

$$x_1^* = \frac{6 - 3x_2^*}{2} = 1.071$$

The geometric interpretation for the preceding problem requires visualizing the objective function as the surface of a paraboloid in three-dimensional space, as shown in Figure 8.1. The projection of the intersection of the paraboloid and the plane representing the constraint onto the $f(x_2) = x_2$ plane is a parabola. We then find the minimum of the resulting parabola. The elimination procedure described earlier is tantamount to projecting the intersection locus onto the x_2 axis. The intersection locus could also be projected onto the x_1 axis (by elimination of x_2). Would you obtain the same result for $\mathbf{x}^*$ as before?

In problems in which there are n variables and m equality constraints, we could attempt to eliminate m variables by direct substitution. If all equality constraints can be removed, and there are no inequality constraints, the objective function can then be differentiated with respect to each of the remaining $(n - m)$ variables and the derivatives set equal to zero. Alternatively, a computer code for unconstrained optimization can be employed to obtain $\mathbf{x}^*$. If the objective function is convex (as in the preceding example) and the constraints form a convex region, then any stationary point is a global minimum. Unfortunately, very few problems in practice assume this simple form or even permit the elimination of all equality constraints.

Consequently, in this chapter we will discuss five major approaches for solving nonlinear programming problems with constraints:

1. Analytic solution by solving the first-order necessary conditions for optimality (Section 8.2)
2. Penalty and barrier methods (Section 8.4)
3. Successive linear programming (Section 8.5)
4. Successive quadratic programming (Section 8.6)
5. Generalized reduced gradient (Section 8.7)

The first of these methods is usually only suitable for small problems with a few variables, but it can generate much useful information and insight when it is applicable. The others are numerical approaches, which must be implemented on a computer.

8.2 FIRST-ORDER NECESSARY CONDITIONS FOR A LOCAL EXTREMUM

As an introduction to this subject, consider the following example.

EXAMPLE 8.1 GRAPHIC INTERPRETATION OF A CONSTRAINED OPTIMIZATION PROBLEM

$$\text{Minimize:}\quad f(x_1, x_2) = x_1 + x_2$$

$$\text{Subject to:}\quad h(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0$$

Solution. This problem is illustrated graphically in Figure E8.1a. Its feasible region is a circle of radius one. Contours of the linear objective $x_1 + x_2$ are lines parallel to the one in the figure. The contour of lowest value that contacts the circle touches it at the point $\mathbf{x}^* = (-0.707, -0.707)$, which is the global minimum. You can solve this problem analytically as an unconstrained problem by substituting for x_1 or x_2 by using the constraint.

Certain relations involving the gradients of f and h hold at $\mathbf{x}^*$ if $\mathbf{x}^*$ is a local minimum. These gradients are

$$\nabla f(\mathbf{x}^*) = [1, 1]$$

$$\nabla h(\mathbf{x}^*) = [2x_1, 2x_2]|_{x^*} = [-1.414, -1.414]$$

and are shown in Figure E8.1b. The gradient of the objective function $\nabla f(\mathbf{x}^*)$ is orthogonal to the tangent plane of the constraint at $\mathbf{x}^*$. In general $\nabla h(\mathbf{x}^*)$ is *always* orthogonal to this tangent plane, hence $\nabla f(\mathbf{x}^*)$ and $\nabla h(\mathbf{x}^*)$ are collinear, that is, they lie on the same line but point in opposite directions. This means the two vectors must be multiples of each other;

$$\nabla f(\mathbf{x}^*) = \lambda^* \nabla h(\mathbf{x}^*) \qquad (a)$$

where $\lambda^* = -1/1.414$ is called the *Lagrange multiplier* for the constraint $h = 0$.

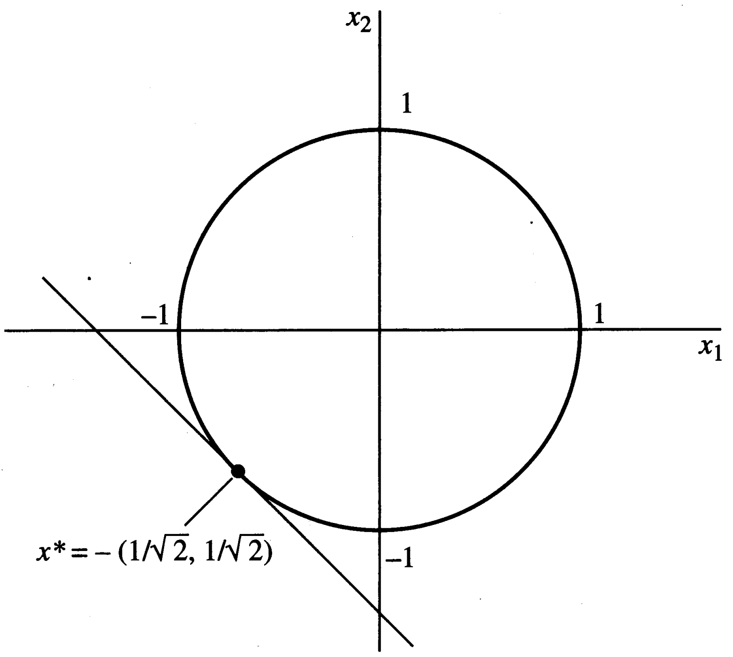

FIGURE E8.1a
Circular feasible region with objective function contours and the constraint.

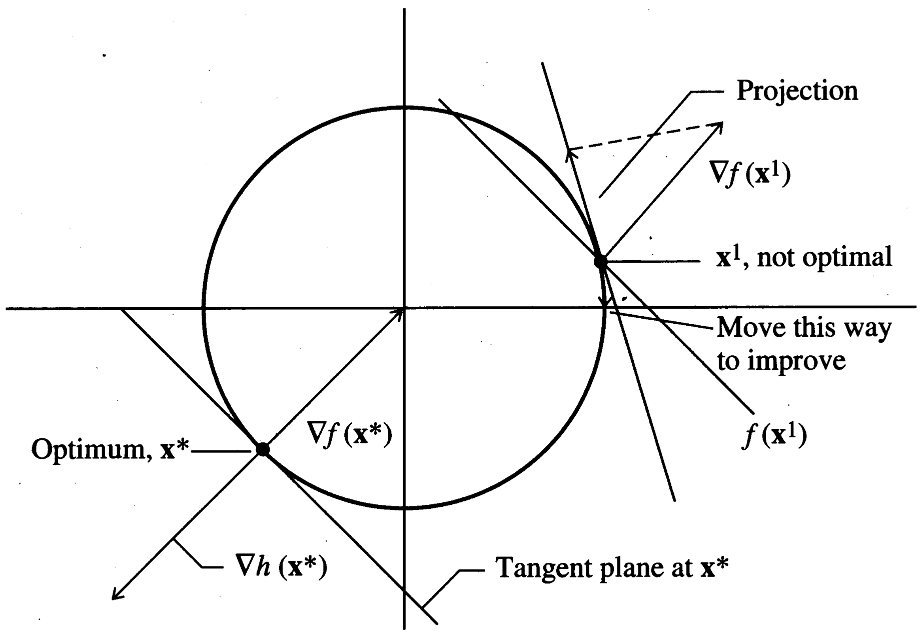

FIGURE E8.1b
Gradients at the optimal point and at a nonoptimal point.

The relationship in Equation (*a*) *must* hold at *any* local optimum of *any* equality-constrained NLP involving smooth functions. To see why, consider the nonoptimal point $\mathbf{x}^1$ in Figure E8.1b. $\nabla f(\mathbf{x}^1)$ is *not* orthogonal to the tangent plane of the constraint at $\mathbf{x}^1$, so it has a nonzero projection on the plane. The negative of this projected gradient is also nonzero, indicating that moving downward along the circle reduces (improves) the objective function. At a local optimum, no small or incremental movement along the constraint (the circle in this problem) away from the optimum can improve the value of the objective function, so the projected gradient must be zero. This can only happen when $\nabla f(\mathbf{x}^*)$ is orthogonal to the tangent plane.

The relation (*a*) in Example 8.1 can be rewritten as

$$\nabla f(\mathbf{x}^*) + \lambda^* \nabla h(\mathbf{x}^*) = 0 \tag{8.5}$$

where $\lambda^* = 0.707$. We now introduce a new function $L(\mathbf{x}, \lambda)$ called the Lagrangian function:

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda h(\mathbf{x}) \tag{8.6}$$

Then Equation (8.5) becomes

$$\nabla_x L(\mathbf{x},\lambda)\big|_{(\mathbf{x}^*,\lambda^*)} = 0 \tag{8.7}$$

so the gradient of the Lagrangian function with respect to **x**, evaluated at $(\mathbf{x}^*, \lambda^*)$, is zero. Equation (8.7), plus the feasibility condition

$$h(\mathbf{x}^*) = 0 \tag{8.8}$$

constitute the first-order necessary conditions for optimality. The scalar λ is called a *Lagrange multiplier*.

Using the necessary conditions to find the optimum

The first-order necessary conditions (8.7) and (8.8) can be used to find an optimal solution. Assume $\mathbf{x}^*$ and λ^* are unknown. The Lagrangian function for the problem in Example 8.1 is

$$L(\mathbf{x},\lambda) = x_1 + x_2 + \lambda(x_1^2 + x_2^2 - 1)$$

Setting the first partial derivatives of L with respect to **x** to zero, we get

$$\frac{\partial L}{\partial x_1} = 1 + 2\lambda x_1 = 0 \tag{8.9}$$

$$\frac{\partial L}{\partial x_2} = 1 + 2\lambda x_2 = 0 \tag{8.10}$$

The feasibility condition (8.8) is

$$x_1^2 + x_2^2 - 1 = 0 \tag{8.11}$$

The first-order necessary conditions for this problem, Equations (8.9)–(8.11), consist of three equations in three unknowns (x_1, x_2, λ). Solving (8.9)–(8.10) for x_1 and x_2 gives

$$x_1 = x_2 = -\frac{1}{2\lambda} \tag{8.12}$$

which shows that x_1 and x_2 are equal at the extremum. Substituting Equation (8.12) into Equation (8.11);

$$\frac{1}{4\lambda^2} + \frac{1}{4\lambda^2} = 1$$

or

$$2\lambda^2 = 1 \tag{8.13}$$

so

$$\lambda = \pm 0.707$$

and

$$x_1 = x_2 = \mp 0.707 \tag{8.14}$$

The minus sign corresponds to the minimum of f, and the plus sign to the maximum.

EXAMPLE 8.2 USE OF LAGRANGE MULTIPLIERS

Consider the problem introduced earlier in Equation (8.2):

$$\text{Minimize:} \quad f(\mathbf{x}) = 4x_1^2 + 5x_2^2 \tag{a}$$

$$\text{Subject to:} \quad h(\mathbf{x}) = 0 = 2x_1 + 3x_2 - 6 \tag{b}$$

Solution. Let

$$L(\mathbf{x},\lambda) = 4x_1^2 + 5x_2^2 + \lambda(2x_1 + 3x_2 - 6) \tag{c}$$

Apply the necessary conditions (8.11) and (8.12)

$$\frac{\partial L(\mathbf{x},\lambda)}{\partial x_1} = 8x_1 + 2\lambda = 0 \tag{d}$$

$$\frac{\partial L(\mathbf{x},\lambda)}{\partial x_2} = 10x_2 + 3\lambda = 0 \tag{e}$$

$$\frac{\partial L(\mathbf{x},\lambda)}{\partial \lambda} = 2x_1 + 3x_2 - 6 = 0 \tag{f}$$

By substitution, $x_1 = -\lambda/4$ and $x_2 = -3\lambda/10$, and therefore Equation (f) becomes

$$2\left(\frac{-\lambda}{4}\right) + 3\left(\frac{-3\lambda}{10}\right) - 6 = 0$$

$$\lambda^* = -4.286$$

$$x_1^* = 1.071$$

$$x_2^* = 1.286$$

8.2.1 Problems Containing Only Equality Constraints

A general equality constrained NLP with m constraints and n variables can be written as

$$\begin{aligned}&\text{Maximize:} \quad f(\mathbf{x}) \\ &\text{Subject to:} \quad h_j(\mathbf{x}) = b_j, \quad j = 1, \dots, m\end{aligned} \tag{8.15}$$

where $\mathbf{x} = (x_1, \dots, x_n)$ is the vector of decision variables, and each b_j is a constant. We assume that the objective f and constraint functions h_j have continuous first partial derivatives. Corresponding to each constraint $h_j = b_j$, define a Lagrange multiplier λ_j and let $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)$ be the vector of these multipliers. The Lagrangian function for the problem is

$$L(\mathbf{x},\boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j[h_j(\mathbf{x}) - b_j] \tag{8.16}$$

and the first-order necessary conditions are

$$\frac{\partial L}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_{j=1}^{m} \lambda_j \frac{\partial h_j}{\partial x_i} = 0, \quad i = 1, \dots, n \tag{8.17}$$

$$h_j(\mathbf{x}) = b_j, \quad j = 1, \dots, m \tag{8.18}$$

Note that there are $n + m$ equations in the $n + m$ unknowns $\mathbf{x}$ and $\boldsymbol{\lambda}$. In Section 8.6 we describe an important class of NLP algorithms called successive quadratic programming (SQP), which solve (8.17)–(8.18) by a variant of Newton's method.

Problem (8.15) must satisfy certain conditions, called constraint qualifications, in order for Equations (8.17)–(8.18) to be applicable. One constraint qualification (see Luenberger, 1984) is that the gradients of the equality constraints, evaluated at $\mathbf{x}^*$, should be linearly independent. Now we can state formally the first order necessary conditions.

First-order necessary conditions for an extremum

Let $\mathbf{x}^*$ be a local minimum or maximum for the problem (8.15), and assume that the constraint gradients $\nabla h_j(\mathbf{x}^*)$, $j = 1, \dots, m$, are linearly independent. Then there exists a vector of Lagrange multipliers $\boldsymbol{\lambda}^* = (\lambda_1^*, \dots, \lambda_m^*)$ such that $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ satisfies the first-order necessary conditions (8.17)–(8.18).

Examples illustrating what can go wrong if the constraint gradients are dependent at $\mathbf{x}^*$ can be found in Luenberger (1984). It is important to remember that *all* local maxima and minima of an NLP satisfy the first-order necessary conditions if the constraint gradients at each such optimum are independent. Also, because these conditions are necessary but not, in general, sufficient, a solution of Equations (8.17)–(8.18) need *not* be a minimum or a maximum at all. It can be a saddle or inflection point. This is exactly what happens in the unconstrained case, where there are no constraint functions $h_j = 0$. Then conditions (8.17)–(8.18) become

$$\nabla f(\mathbf{x}) = \mathbf{0}$$

the familiar condition that the gradient must be zero (see Section 4.5). To tell if a point satisfying the first-order necessary conditions is a minimum, maximum, or neither, second-order sufficiency conditions are needed. These are discussed later in this section.

Sensitivity interpretation of Lagrange multipliers

Sensitivity analysis in NLP indicates how an optimal solution changes as the problem data change. These data include *any* parameters that appear in the objective or constraint functions, or on the right-hand sides of constraints. The Lagrange multipliers $\boldsymbol{\lambda}^*$ provide useful information on right-hand side changes, just as they do for linear programs (which are a special class of NLPs). To illustrate their application in NLP, consider again Example 8.1, with the constraint right-hand side (the square of the radius of the circle) treated as a parameter b;

$$\begin{aligned} \text{Minimize:} \quad & x_1 + x_2 \\ \text{Subject to:} \quad & x_1^2 + x_2^2 = b \end{aligned}$$

The optimal solution of this problem is a function of b, denoted by $(x_1(b), x_2(b))$, as is the optimal multiplier value, $\lambda(b)$. Using the first-order necessary conditions (8.9)–(8.11), rewritten here as

$$\begin{aligned} 1 + 2\lambda x_1 &= 0 \\ 1 + 2\lambda x_2 &= 0 \\ x_1^2 + x_2^2 &= b \end{aligned}$$

The solution of these equations is (check it!);

$$\begin{aligned} x_1^*(b) = x_2^*(b) &= -\left(\frac{b}{2}\right)^{1/2} \\ \lambda^*(b) &= (2b)^{-1/2} \end{aligned}$$

These formulas agree with the previous results for $b = 1$. The minimal objective value, sometimes called the *optimal value function,* is

$$V(b) = x_1(b) + x_2(b) = -(2b)^{1/2}$$

The derivative of the optimal value function is

$$\frac{dV}{db} = -(2b)^{-1/2} = -\lambda^*(b)$$

so the negative of the optimal Lagrange multiplier value is dV/db. Hence, if we solve this problem for a specific b (for example $b = 1$) then the optimal objective value for b close to 1 has the first-order Taylor series approximation

$$\begin{aligned} V(b) &\approx V(1) - \lambda(1)(b - 1) \\ &= -\sqrt{2} - \frac{1}{\sqrt{2}}(b - 1) \end{aligned}$$

To see how useful these Lagrange multipliers are, consider the general problem (8.15), with right-hand sides b_i;

$$\text{Minimize:}\quad f(\mathbf{x})$$

$$\text{Subject to:}\quad h_i(\mathbf{x}) = b_i, \quad i = 1, \dots, m \tag{8.19}$$

Let $\mathbf{b} = (b_1, \dots, b_m)$ be the right-hand side (rhs) vector, and $V(\mathbf{b})$ the optimal objective value. If $\bar{\mathbf{b}}$ is a specific right-hand side vector, and $(\mathbf{x}(\bar{\mathbf{b}}), \boldsymbol{\lambda}(\bar{\mathbf{b}}))$ is a local optimum for $\mathbf{b} = \bar{\mathbf{b}}$, then

$$-\lambda_j(\bar{\mathbf{b}}) = \left.\frac{\partial V}{\partial b_j}\right|_{\bar{\mathbf{b}}} \tag{8.20}$$

The constraints with the largest absolute λ_j values are the ones whose right-hand sides affect the optimal value function V the most, at least for b close to $\bar{\mathbf{b}}$. However, one must account for the units for each b_j in interpreting these values. For example, if some b_j is measured in kilograms and both sides of the constraint $h_j(\mathbf{x}) = b_j$ are multiplied by 2.2, then the new constraint has units of pounds, and its new Lagrange multiplier is 1/2.2 times the old one.

8.2.2 Problems Containing Only Inequality Constraints

The first-order necessary conditions for problems with inequality constraints are called the *Kuhn–Tucker conditions* (also called Karush–Kuhn–Tucker conditions). The idea of a cone aids the understanding of the Kuhn–Tucker conditions (KTC). A cone is a set of points R such that, if $\mathbf{x}$ is in R, $\boldsymbol{\lambda}^T\mathbf{x}$ is also in R for $\boldsymbol{\lambda} \geq 0$. A *convex cone* is a cone that is a convex set. An example of a convex cone in two dimensions is shown in Figure 8.2. In two and three dimensions, the definition of a convex cone coincides with the usual meaning of the word.

It can be shown from the preceding definitions that the set of all nonnegative linear combinations of a finite set of vectors is a convex cone, that is, that the set

$$R = \{\mathbf{x} | \mathbf{x} = \lambda_1\mathbf{x}_1 + \lambda_2\mathbf{x}_2 + \cdots + \lambda_m\mathbf{x}_m, \lambda_i \geq 0, \quad i = 1, \dots, m\}$$

is a convex cone. The vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ are called the generators of the cone. For example, the cone of Figure 8.2 is generated by the vectors [2, 1] and [2, 4]. Thus any vector that can be expressed as a nonnegative linear combination of these vectors lies in the cone. In Figure 8.2 the vector [4, 5] in the cone is given by $[4, 5] = 1 \times [2, 1] + 1 \times [2, 4]$.

Kuhn–Tucker conditions: Geometrical interpretation

The Kuhn–Tucker conditions are predicated on this fact: At any local constrained optimum, no (small) allowable change in the problem variables can improve the value of the objective function. To illustrate this statement, consider the nonlinear programming problem:

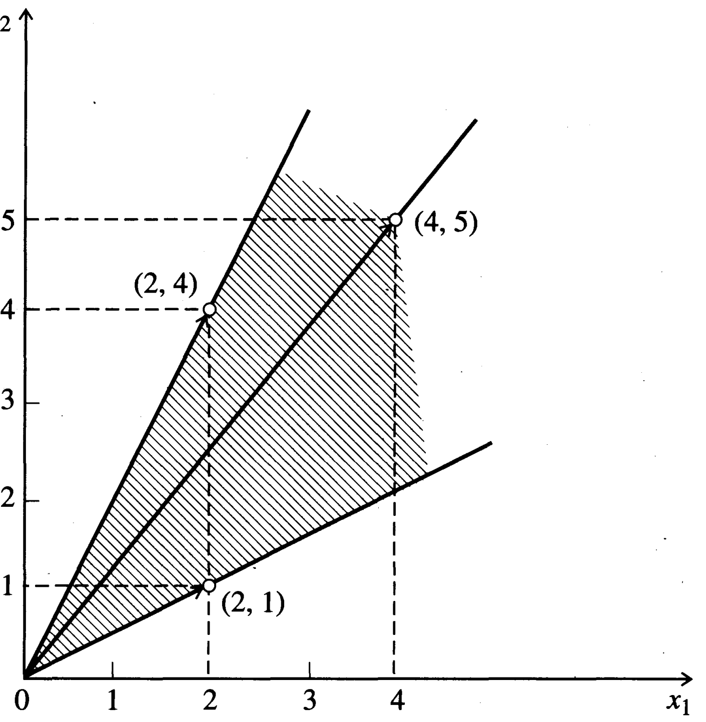

FIGURE 8.2
The shaded region forms a convex cone.

$$\begin{aligned}
\text{Minimize:}\quad & f(x,y) = (x-2)^2 + (y-1)^2 \\
\text{Subject to:}\quad & g_1(x,y) = -y + x^2 \le 0 \\
& g_2(x,y) = x + y \le 2 \\
& g_3(x,y) = y \ge 0
\end{aligned}$$

The problem is shown geometrically in Figure 8.3. It is evident that the optimum is at the intersection of the first two constraints at (1, 1). Because these inequality constraints hold as equalities at (1, 1), they are called *binding*, or *active*, constraints at this point. The third constraint holds as a strict inequality at (1, 1), and it is an *inactive*, or *nonbinding*, constraint at this point. Define a *feasible direction* of search as a vector such that a differential move along that vector violates no constraints. At (1, 1), the set of all feasible directions lies between the line $x + y - 2 = 0$ and the tangent line to $y = x^2$ at (1, 1), that is, the line $y = 2x - 1$. In other words, the set of feasible directions is the cone generated by these lines that are shaded in the figure. The vector $-\nabla f$ points in the direction of the maximum rate of decrease of f, and a small move along any direction making an angle (defined as positive) of less than 90° with $-\nabla f$ will decrease f. Thus, at the optimum, no feasible direction can have an angle of less than 90° between it and $-\nabla f$.

Now consider Figure 8.4, in which the gradient vectors ∇g_1 and ∇g_2 are drawn. Note that $-\nabla f$ is contained in the cone generated by ∇g_1 and ∇g_2. What if this were not so? If $-\nabla f$ were slightly above ∇g_2, it would make an angle of less than 90° with a feasible direction just below the line $x + y - 2 = 0$. If $-\nabla f$ were slightly below ∇g_1, it would make an angle of less than 90° with a feasible direction just

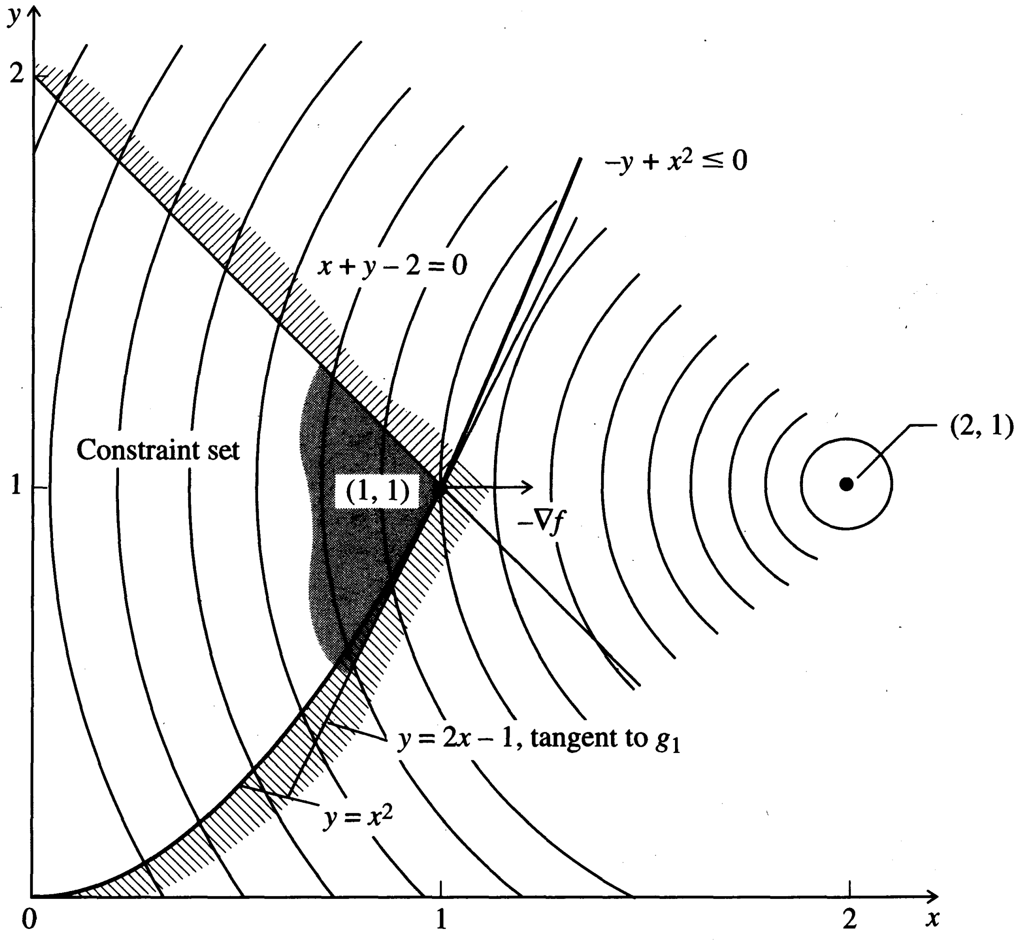

FIGURE 8.3
Geometry of a constrained optimization problem. The feasible region lies within the binding constraints plus the boundaries themselves.

above the line $y = 2x - 1$. Neither case can occur at an optimal point, and both cases are excluded if and only if $-\nabla f$ lies within the cone generated by ∇g_1 and ∇g_2. Of course, this is the same as requiring that ∇f lie within the cone generated by $-\nabla g_1$ and $-\nabla g_2$. This leads to the usual statement of the KTC; that is, if f and all g_i are differentiable, a necessary condition for a point $\mathbf{x}^*$ to be a constrained minimum of the problem

$$\text{Minimize: } f(\mathbf{x})$$
$$\text{Subject to: } g_j(\mathbf{x}) \leq c_j, \quad j = 1, \ldots, r$$

is that, at $\mathbf{x}^*$, ∇f lies within the cone generated by the negative gradients of the binding constraints.

Algebraic statement of the Kuhn–Tucker conditions

The preceding results may be stated in algebraic terms. For ∇f to lie within the cone described earlier, it must be a nonnegative linear combination of the negative gradients of the binding constraints; that is, there must exist Lagrange multipliers u_j^* such that

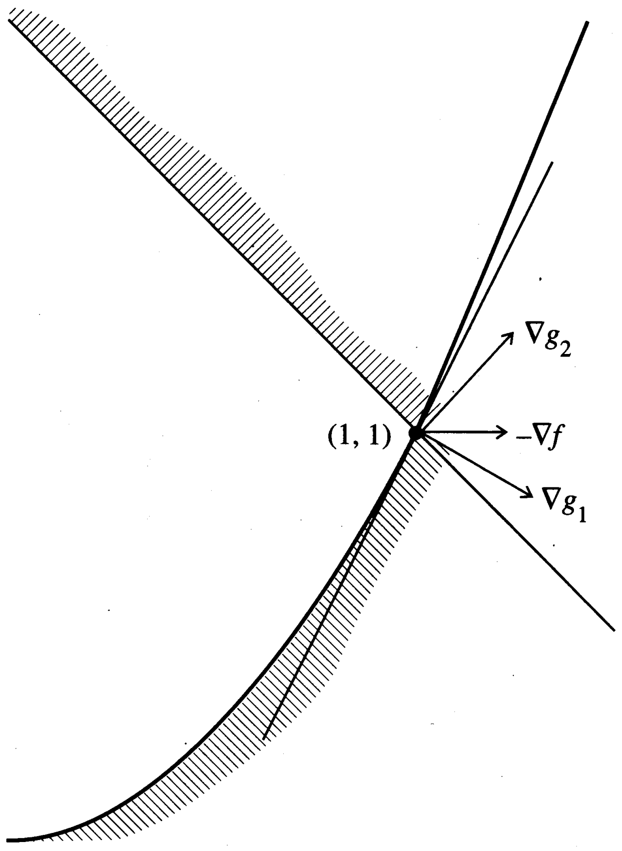

FIGURE 8.4
Gradient of objective contained in convex cone.

$$\nabla f(\mathbf{x}^*) = \sum_{j \in I} u_j^* [-\nabla g_j(\mathbf{x}^*)] \tag{8.21}$$

where

$$u_j^* \geq 0, \quad j \in I \tag{8.22}$$

and I is the set of indices of the binding inequality constraints. The multipliers u_j^* are analogous to λ_i defined for equality constraints.

These results may be restated to include all constraints by defining the multiplier u_j^* to be zero if $g_j(\mathbf{x}^*) < c_j$. In the previous example u_3^*, the multiplier of the inactive constraint g_3, is zero. Then we can say that $u_j^* \geq 0$ if $g_j(\mathbf{x}^*) = c_j$, and $u_j^* = 0$ if $g_j(\mathbf{x}^*) < c_j$, thus the product $u_j^*[g_j(\mathbf{x}) - c_j]$ is zero for all j. This property, that inactive inequality constraints have zero multipliers, is called *complementary slackness*. Conditions (8.21) and (8.22) then become

$$\nabla f(\mathbf{x}^*) + \sum_{j=1}^{r} u_j^* \nabla g_j(\mathbf{x}^*) = 0 \tag{8.23}$$

$$u_j^* \geq 0, \quad u_j^*[g_j(\mathbf{x}^*) - c_j] = 0 \tag{8.24a}$$

$$g_j(\mathbf{x}^*) \leq c_j, \quad j = 1, \ldots, r \tag{8.24b}$$

Relations (8.23) and (8.24) are the form in which the Kuhn–Tucker conditions are usually stated.

Lagrange multipliers

The KTC are closely related to the classical Lagrange multiplier results for equality constrained problems. Form the Lagrangian

$$L(\mathbf{x},\mathbf{u}) = f(\mathbf{x}) + \sum_{j=1}^{r} u_j[g_j(\mathbf{x}) - c_j]$$

where the u_j are viewed as Lagrange multipliers for the inequality constraints $g_j(\mathbf{x}) \le c_j$. Then Equations (8.23) and (8.24) state that $L(\mathbf{x}, \mathbf{u})$ must be stationary in $\mathbf{x}$ at $(\mathbf{x}^*, \mathbf{u}^*)$ with the multipliers $\mathbf{u}^*$ satisfying Equation (8.24). The stationarity of L is the same condition as in the equality-constrained case. The additional conditions in Equation (8.24) arise because the constraints here are inequalities.

8.2.3 Problems Containing both Equality and Inequality Constraints

When both equality and inequality constraints are present, the KTC are stated as follows: Let the problem be

$$\text{Minimize: } f(\mathbf{x}) \tag{8.25}$$

$$\text{Subject to: } h_i(\mathbf{x}) = b_i, \quad i = 1, \ldots, m \tag{8.26a}$$

and

$$g_j(\mathbf{x}) \le c_j, \quad j = 1, \ldots, r \tag{8.26b}$$

Define Lagrange multipliers λ_i associated with the equalities and u_j for the inequalities, and form the Lagrangian function

$$L(\mathbf{x},\boldsymbol{\lambda},\mathbf{u}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i[h_i(\mathbf{x}) - b_i] + \sum_{j=1}^{r} u_j[g_j(\mathbf{x}) - c_j] \tag{8.27}$$

Then, if $\mathbf{x}^*$ is a local minimum of the problems (8.25)–(8.26), there exist vectors of Lagrange multipliers $\boldsymbol{\lambda}^*$ and $\mathbf{u}^*$, such that $\mathbf{x}^*$ is a stationary point of the function $L(\mathbf{x}, \boldsymbol{\lambda}^*, \mathbf{u}^*)$, that is,

$$\nabla_x L(\mathbf{x}^*,\boldsymbol{\lambda}^*,\mathbf{u}^*) = \nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{j=1}^{r} u_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0} \tag{8.28}$$

and complementary slackness hold for the inequalities:

$$u_j^* \ge 0 \qquad u_j^*[g_j(\mathbf{x}^*) - c_j] = 0, \quad j = 1, \ldots, r \tag{8.29}$$

EXAMPLE 8.3 APPLICATION OF THE LAGRANGE MULTIPLIER METHOD WITH NONLINEAR INEQUALITY CONSTRAINTS

Solve the problem

$$\text{Minimize:} \quad f(\mathbf{x}) = x_1 x_2$$

$$\text{Subject to:} \quad g(\mathbf{x}) = x_1^2 + x_2^2 \le 25 \qquad (a)$$

by the Lagrange multiplier method.

Solution. The Lagrange function is

$$L(\mathbf{x},u) = x_1 x_2 + u(x_1^2 + x_2^2 - 25) \qquad (b)$$

The necessary conditions for a stationary point are

$$\frac{\partial L}{\partial x_1} = x_2 + 2ux_1 = 0$$

$$\frac{\partial L}{\partial x_2} = x_1 + 2ux_2 = 0$$

$$\frac{\partial L}{\partial u} = x_1^2 + x_2^2 \le 25 \qquad (c)$$

$$u(25 - x_1^2 - x_2^2) = 0$$

The five simultaneous solutions of Equations (*c*) are listed in Table E8.3. How would you calculate these values?

Columns two and three of Table E8.3 list the components of **x*** that are the stationary solutions of the problem. Note that the solutions with $u > 0$ are minima, those for $u < 0$ are maxima, and $u = 0$ is a saddle point. This is because maximizing f is equivalent to minimizing $-f$, and the KTC for the problem in Equation (*a*) with f replaced by $-f$ are the equations shown in (*c*) with u allowed to be negative. In Fig-

TABLE E8.3
Solutions of Example 8.3 by the Lagrange multiplier method

U	x_1	x_2	Point	C	$f(\mathbf{x})$	Remarks
0	0	0	A	25	0	saddle
0.5	+3.54	−3.54	B	0	−12.5	minimum
	−3.54	+3.54	C	0	−12.5	minimum
−0.5	+3.54	+3.54	D	0	+12.5	maximum
	−3.54	−3.54	E	0	+12.5	maximum

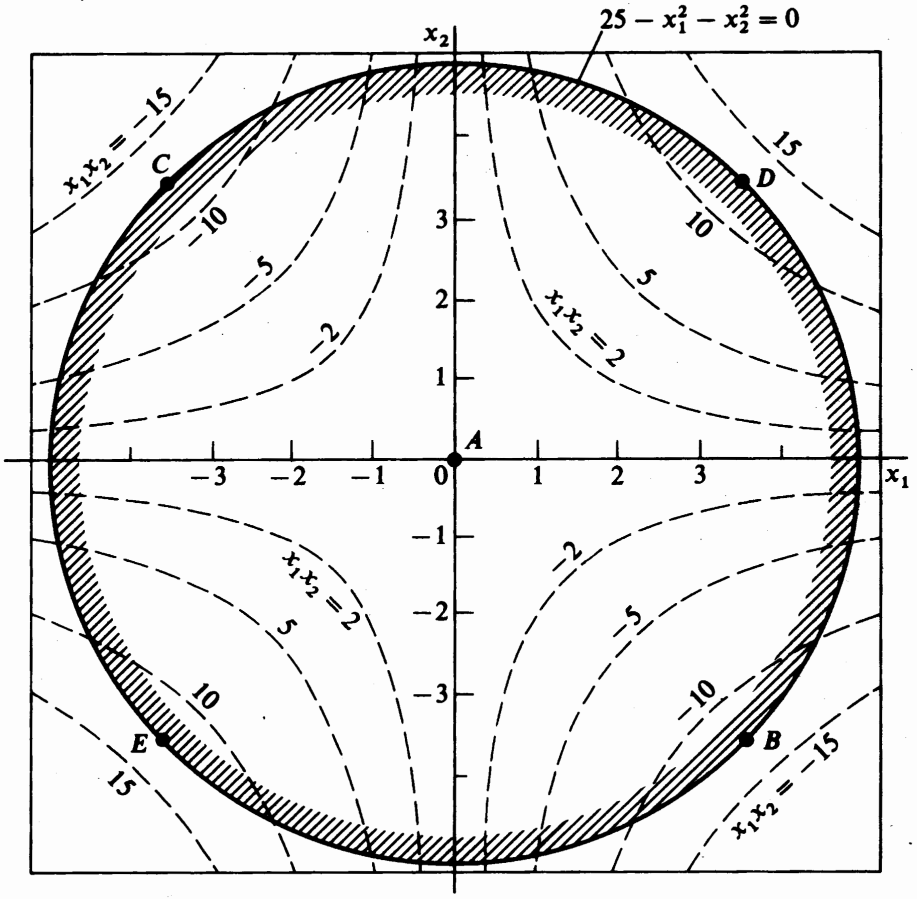

FIGURE E8.3

ure E8.3 the contours of the objective function (hyperbolas) are represented by broken lines, and the feasible region is bounded by the shaded area enclosed by the circle $g(\mathbf{x}) = 25$. Points B and C correspond to the two minima, D and E to the two maxima, and A to the saddle point of $f(\mathbf{x})$.

Lagrange multipliers and sensitivity analysis

At each iteration, NLP algorithms form new estimates not only of the decision variables $\mathbf{x}$ but also of the Lagrange multipliers $\boldsymbol{\lambda}$ and $\mathbf{u}$. If, at these estimates, all constraints are satisfied and the KTC are satisfied to within specified tolerances, the algorithm stops. At a local optimum, the optimal multiplier values provide useful sensitivity information. In the NLP (8.25)–(8.26), let $V^*(\mathbf{b}, \mathbf{c})$ be the optimal value of the objective f at a local minimum, viewed as a function of the right-hand sides of the constraints $\mathbf{b}$ and $\mathbf{c}$. Then, under additional conditions (see Luenberger, 1984, Chapter 10)

$$\lambda_i^* = \frac{-\partial V^*}{\partial b_i}, \quad i = 1, \ldots, m \tag{8.30a}$$

$$u_j^* = \frac{-\partial V^*}{\partial c_j}, \quad j = 1, \dots, r \tag{8.30b}$$

That is, the Lagrange multipliers provide the rate of change of the optimal objective value with respect to changes in the constraint right-hand sides. This information is often of significant value. For example, if the right-hand side of an inequality constraint c_j represents the capacity of a process and this capacity constraint is active at the optimum, then the optimal multiplier value u_j^* equals the rate of decrease of the minimal cost if the capacity is increased. This change is the marginal value of the capacity. In a situation with several active capacity limits, the ones with the largest absolute multipliers should be considered first for possible increases. Examples of the use of Lagrange multipliers for sensitivity analysis in linear programming are given in Chapter 7.

Lagrange multipliers are quite helpful in analyzing parameter sensitivities in problems with multiple constraints. In a typical refinery, a number of different products are manufactured which must usually meet (or exceed) certain specifications in terms of purity as required by the customers. Suppose we carry out a constrained optimization for an objective function that includes several variables that occur in the refinery model, that is, those in the fluid catalytic cracker, in the distillation column, and so on, and arrive at some economic optimum subject to the constraints on product purity. Given the optimum values of the variables plus the Lagrange multipliers corresponding to the product purity, we can then pose the question: How will the profits change if the product specification is either relaxed or made more stringent? To answer this question simply requires examining the Lagrange multiplier for each constraint. As an example, consider the case in which there are three major products (A, B, and C) and the Lagrange multipliers corresponding to each of the three demand inequality constraints are calculated to be:

$$u_A = -0.001$$

$$u_B = -1.0$$

$$u_C = -0.007$$

The values for u_i show (ignoring scaling) that satisfying an additional unit of demand of product B is much more costly than for the other two products.

Convex programming problems

The KTC comprise both the necessary and sufficient conditions for optimality for smooth convex problems. In the problem (8.25)–(8.26), if the objective $f(\mathbf{x})$ and inequality constraint functions g_j are convex, and the equality constraint functions h_j are linear, then the feasible region of the problem is convex, and any local minimum is a global minimum. Further, if $\mathbf{x}^*$ is a feasible solution, if all the problem functions have continuous first derivatives at $\mathbf{x}^*$, and if the gradients of the active constraints at $\mathbf{x}^*$ are independent, then $\mathbf{x}^*$ is optimal if and only if the KTC are satisfied at $\mathbf{x}^*$.

Practical considerations

Many real problems do not satisfy these convexity assumptions. In chemical engineering applications, equality constraints often consist of input–output relations of process units that are often nonlinear. Convexity of the feasible region can only be guaranteed if these constraints are all linear. Also, it is often difficult to tell if an inequality constraint or objective function is convex or not. Hence it is often uncertain if a point satisfying the KTC is a local or global optimum, or even a saddle point. For problems with a few variables we can sometimes find all KTC solutions analytically and pick the one with the best objective function value. Otherwise, most numerical algorithms terminate when the KTC are satisfied to within some tolerance. The user usually specifies two separate tolerances: a feasibility tolerance ε_f and an optimality tolerance ε_o. A point $\bar{x}$ is feasible to within ε_f if

$$|h_i(\bar{\mathbf{x}}) - b_i| \leq \varepsilon_f, \quad \text{for} \quad i = 1, \dots, m$$

and

$$g_j(\bar{\mathbf{x}}) - c_j \leq \varepsilon_f, \quad \text{for} \quad j = 1, \dots, r \tag{8.31a}$$

Furthermore, $\bar{x}$ is optimal to within $(\varepsilon_o, \varepsilon_f)$ if it is feasible to within ε_f and the KTC are satisfied to within ε_o. This means that, in Equations (8.23)–(8.24)

$$\left| \frac{\partial L}{\partial x_i}(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}, \bar{\mathbf{u}}) \right| \leq \varepsilon_o, \quad i = 1, \dots, n$$

and

$$u_j \geq -\varepsilon_o, \quad j = 1, \dots, r \tag{8.31b}$$

Equation (8.31b) corresponds to relaxing the constraint.

Second-order necessary and sufficiency conditions for optimality

The Kuhn–Tucker necessary conditions are satisfied at any local minimum or maximum and at saddle points. If $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*)$ is a Kuhn–Tucker point for the problem (8.25)–(8.26), and the second-order sufficiency conditions are satisfied at that point, optimality is guaranteed. The second order optimality conditions involve the matrix of second partial derivatives with respect to $\mathbf{x}$ (the Hessian matrix of the Lagrangian function), and may be written as follows:

$$\mathbf{y}^T \nabla_x^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*) \mathbf{y} > \mathbf{0} \tag{8.32a}$$

for all nonzero vectors $\mathbf{y}$ such that

$$\mathbf{J}(\mathbf{x}^*)\mathbf{y} = \mathbf{0} \tag{8.32b}$$

where $\mathbf{J}(\mathbf{x}^*)$ is the matrix whose rows are the gradients of the constraints that are active at $\mathbf{x}^*$. Equation (8.32b) defines a set of vectors $\mathbf{y}$ that are orthogonal to the gradients of the active constraints. These vectors constitute the *tangent plane* to the active constraints, which was illustrated in Example 8.1. Hence (8.32a) requires that the Lagrangian Hessian matrix be positive-definite for all vectors **y** on this tangent plane. If the ">" sign in (8.32a) is replaced by "≥", then (8.32a)–(8.32b) plus the KTC are the *second-order necessary conditions* for a local minimum. See Luenberger (1984) or Nash and Sofer (1996) for a more thorough discussion of these second-order conditions.

If no active constraints occur (so **x*** is an unconstrained stationary point), then (8.32a) must hold for all vectors **y**, and the multipliers **λ*** and **u*** are zero, so $\nabla_x^2 L = \nabla_x^2 f$. Hence (8.32a) and (8.32b) reduce to the condition discussed in Section 4.5 that if the Hessian matrix of the objective function, evaluated at **x***, is positive-definite and **x*** is a stationary point, then **x*** is a local unconstrained minimum of f.

EXAMPLE 8.4 USING THE SECOND-ORDER CONDITIONS

As an example, consider the problem:

$$\text{Minimize: } f(\mathbf{x}) = (x_1 - 1)^2 + x_2^2$$

$$\text{Subject to: } x_1 - x_2^2 \le 0$$

Solution. Although the objective function of this problem is convex, the inequality constraint does not define a convex feasible region; as shown in Figure E8.4. The geometric interpretation is to find the points in the feasible region closest to (1, 0). The Lagrangian function for this problem is

$$L(\mathbf{x},u) = (x_1 - 1)^2 + x_2^2 + u(x_1 - x_2^2)$$

and the KTC for a local minimum are

$$\partial L/\partial x_1 = 2(x_1 - 1) + u = 0$$

$$\partial L/\partial x_2 = 2x_2 - 2ux_2 = 0$$

$$x_1 - x_2^2 \le 0, \quad u \ge 0$$

There are three solutions to these conditions: two global minima, at $x_1^* = \frac{1}{2}$, $x_2^* = \pm\sqrt{\frac{1}{2}}, u^* = 1$ with an objective value of 0.75, and a local maximum at $x_1^0 = 0$, $x_2^0 = 0$, $u^0 = 2$ with an objective value of 1.0. These solutions are evident by examining Figure E8.4.

The second order sufficiency conditions show that the first two of these three Kuhn–Tucker points are local minima, and the third is not. The Hessian matrix of the Lagrangian function is

$$\nabla_x^2 L(x,u) = \begin{bmatrix} 2 & 0 \\ 0 & 2(1-u) \end{bmatrix}$$

The Hessian evaluated at $(x_1^0 = 0, x_2^0 = 0, u^0 = 2)$ is

$$\nabla_x^2 L(0,0,2) = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

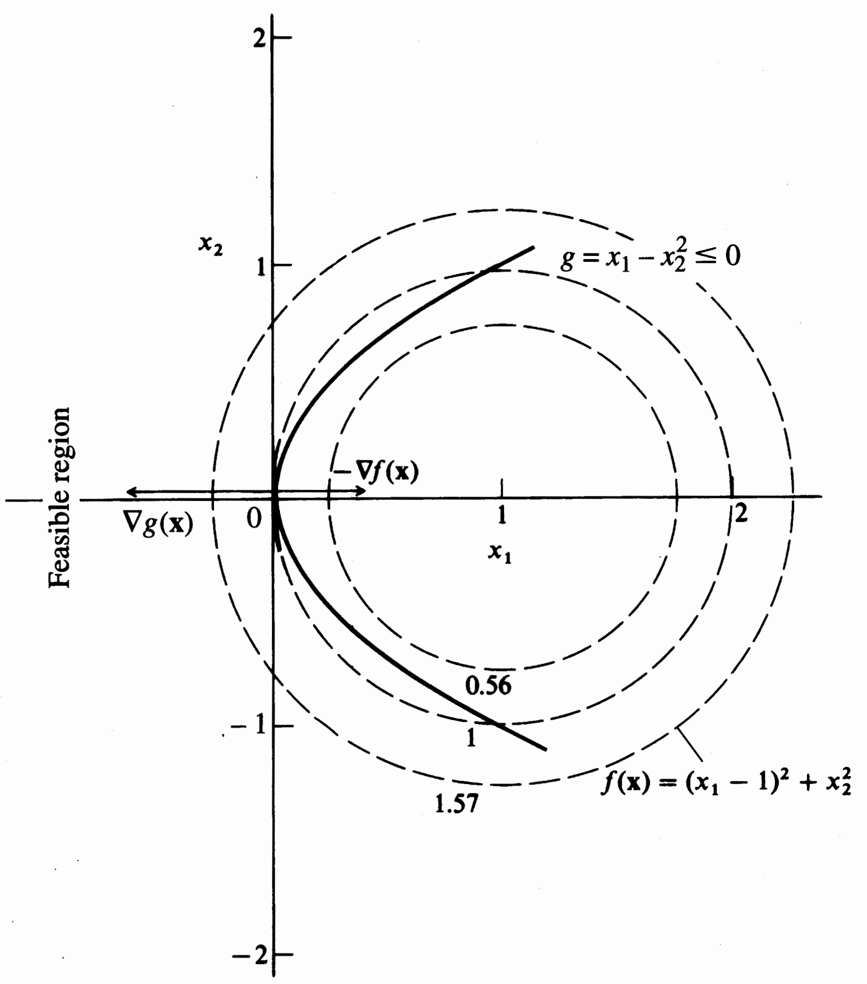

FIGURE E8.4

The second-order necessary conditions require this matrix to be positive-semidefinite on the tangent plane to the active constraints at (0, 0), as defined in expression (8.32b). Here, this tangent plane is the set

$$T = \{\mathbf{y} \mid \nabla g(0,0)^T \mathbf{y} = 0\}$$

The gradient of the constraint function is

$$\nabla^T g(x_1, x_2) = [1 \ \ -2x_2] \qquad \text{so} \qquad \nabla g(0,0) = [1 \ \ 0]$$

Thus the tangent plane at (0, 0) is

$$T = \{\mathbf{y} \mid y_1 = 0\} = \{\mathbf{y} \mid \mathbf{y} = (0, y_2)\}$$

and the quadratic form in (8.32a), evaluated on the tangent plane, is

$$\mathbf{y}^T \nabla_x^2 L(\mathbf{x}^*, \mathbf{u}^*)\mathbf{y} = [0 \ \ y_2] \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} 0 \\ y_2 \end{bmatrix} = -2y_2^2$$

Because $-2y_2^2$ is negative for all nonzero vectors in the set T, the second-order necessary condition is not satisfied, so (0, 0) is not a local minimum.

If we check the minimum at $x_1^* = \frac{1}{2}, x_2^* = \sqrt{\frac{1}{2}}, u^* = 1$, the Lagrangian Hessian evaluated at this point is

$$\nabla_x^2 L(\tfrac{1}{2}, \sqrt{\tfrac{1}{2}}, 1) = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

The constraint gradient at this point is $\begin{bmatrix} 1 & -\sqrt{2} \end{bmatrix}$, so the tangent plane is

$$T = \{\mathbf{y} \mid y_1 - \sqrt{2}y_2 = 0\} = \{\mathbf{y} \mid \mathbf{y} = y_2(-\sqrt{2}, 1)\}$$

On this tangent plane, the quadratic form is

$$y^T \nabla_x^2 L(\mathbf{x}^*, \mathbf{u}^*)\mathbf{y} = y_2^2\begin{bmatrix} -\sqrt{2} & 1 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} -\sqrt{2} \\ 1 \end{bmatrix} = 4y_2^2$$

This is positive for all nonzero vectors in the set T, so the second-order sufficiency conditions are satisfied, and the point is a local minimum.

8.3 QUADRATIC PROGRAMMING

A *quadratic programming* (QP) problem is an optimization problem in which a quadratic objective function of n variables is minimized subject to m linear inequality or equality constraints. A convex QP is the simplest form of a nonlinear programming problem with inequality constraints. A number of practical optimization problems, such as constrained least squares and optimal control of linear systems with quadratic cost functions and linear constraints, are naturally posed as QP problems. In this text we discuss QP as a subproblem to solve general nonlinear programming problems. The algorithms used to solve QPs bear many similarities to algorithms used in solving the linear programming problems discussed in Chapter 7.

In matrix notation, the quadratic programming problem is

$$\begin{aligned} \text{Minimize:} \quad & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \tfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \\ \text{Subject to:} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{8.33}$$

where $\mathbf{c}$ is a vector of constant coefficients, $\mathbf{A}$ is an $(m \times n)$ matrix, and $\mathbf{Q}$ is a symmetric matrix.

The vector $\mathbf{x}$ can contain slack variables, so the equality constraints (8.33) may contain some constraints that were originally inequalities but have been converted to equalities by inserting slacks. Codes for quadratic programming allow arbitrary upper and lower bounds on $\mathbf{x}$; we assume $\mathbf{x} \geq \mathbf{0}$ only for simplicity.

If the equality constraints in (8.33) are independent then, as discussed in Section 8.2, the KTC are the necessary conditions for an optimal solution of the QP. In addition, if $\mathbf{Q}$ is positive-semidefinite in (8.33), the QP objective function is convex. Because the feasible region of a QP is defined by linear constraints, it is always convex, so the QP is then a convex programming problem, and any local solution is a global solution. Also, the KTC are the sufficient conditions for a minimum, and a solution meeting these conditions yields the global optimum. If $\mathbf{Q}$ is not positive-semidefinite, the problem may have an unbounded solution or local minima.

To write the KTC, start with the Lagrangian function

$$L = \mathbf{x}^T\mathbf{c} + \tfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) - \mathbf{u}^T\mathbf{x}$$

and equate the gradient of L (with respect to $\mathbf{x}^T$) to zero (note that $\boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) = (\mathbf{A}\mathbf{x} - \mathbf{b})^T\boldsymbol{\lambda} = (\mathbf{x}^T\mathbf{A}^T - \mathbf{b}^T)\,\boldsymbol{\lambda}$ and $\mathbf{u}^T\mathbf{x} = \mathbf{x}^T\mathbf{u}$)

$$\nabla_x L = \mathbf{c} + \mathbf{Q}\mathbf{x} + \mathbf{A}^T\boldsymbol{\lambda} - \mathbf{u} = \mathbf{0}$$

Then the KTC reduce to the following set of equations:

$$\mathbf{c} + \mathbf{Q}\mathbf{x} + \mathbf{A}^T\boldsymbol{\lambda} - \mathbf{u} = \mathbf{0} \tag{8.34}$$

$$\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \tag{8.35}$$

$$\mathbf{x} \geq \mathbf{0} \qquad \mathbf{u} \geq \mathbf{0} \tag{8.36}$$

$$\mathbf{u}^T\mathbf{x} = 0 \tag{8.37}$$

where the u_i and λ_j are the Lagrange multipliers. If $\mathbf{Q}$ is positive semidefinite, any set of variables $(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*)$ that satisfies (8.34) to (8.37) is an optimal solution to (8.33).

Some QP solvers use these KTC directly by finding a solution satisfying the equations. They are linear except for (8.37), which is called a complementary slackness condition. These conditions were discussed for general inequality constraints in Section 8.2. Applied to the nonnegativity conditions in (8.33), complementary slackness implies that at least one of each pair of variables (u_i, x_i) must be zero. Hence a feasible solution to the KTC can be found by starting with an infeasible complementary solution to the linear constraints (8.34)–(8.36) and using LP pivot operations to minimize the sum of infeasibilities while maintaining complementarity. Because (8.34) and (8.35) have n and m constraints, respectively, the effect is roughly equivalent to solving an LP with $(n + m)$ rows. Because LP "machinery" is used, most commercial LP systems, including those discussed in Chapter 7, contain QP solvers. In addition, a QP can also be solved by any efficient general purpose NLP solver.

8.4 PENALTY, BARRIER, AND AUGMENTED LAGRANGIAN METHODS

The essential idea of a penalty method of nonlinear programming is to transform a constrained problem into a sequence of unconstrained problems.

$$\left.\begin{array}{ll}\text{Minimize:} & f(\mathbf{x}) \\ \text{Subject to:} & \mathbf{g}(\mathbf{x}) \leq 0 \\ & \mathbf{h}(\mathbf{x}) = 0\end{array}\right\} \Rightarrow \text{Minimize: } P(f, \mathbf{g}, \mathbf{h}, r) \tag{8.38}$$

where $P(f, \mathbf{g}, \mathbf{h}, r)$ is a *penalty function*, and r is a positive penalty parameter. After the penalty function is formulated, it is minimized for a series of values of increasing r-values, which force the sequence of minima to approach the optimum of the constrained problem.

As an example, consider the problem

$$\text{Minimize:} \quad f(\mathbf{x}) = (x_1 - 1)^2 + (x_2 - 2)^2$$

$$\text{Subject to:} \quad h(\mathbf{x}) = x_1 + x_2 - 4 = 0$$

We formulate a new unconstrained objective function

$$P(\mathbf{x},r) = (x_1 - 1)^2 + (x_2 - 2)^2 + r(x_1 + x_2 - 4)^2$$

where r is a positive scalar called the penalty parameter, and $r(x_1 + x_2 - 4)^2$ is called the penalty term. Consider a series of minimization problems where we minimize $P(\mathbf{x},r)$ for an increasing sequence of r values tending to infinity. As r increases, the penalty term becomes large for any values of $\mathbf{x}$ that violate the equality constraints in (8.38). As the penalty term grows, the values of $\mathbf{x}_i$ change to those that cause the equality constraint to be satisfied. In the limit the product of r and h^2 approaches zero so that the value of f approaches the value of P. This is shown in Figure 8.5. The constrained optimum is $\mathbf{x}^* = (1.5, 2.5)$ and the unconstrained minimum of the objective is at $(1, 2)$. The point $(1, 2)$ is also the minimum of $P(\mathbf{x}, 0)$. The minimizing points for $r = 1, 10, 100, 1000$ are at the center of the elliptical contours in the figure. Table 8.1 shows r, $x_1(r)$, and $x_2(r)$. It is clear that $\mathbf{x}(r) \to \mathbf{x}^*$ as $r \to \infty$, which can be shown to be true in general (see Luenberger, 1984).

Note how the contours of $P(\mathbf{x}, r)$ bunch up around the constraint line $x_1 + x_2 = 4$ as r becomes large. This happens because, for large r, $P(\mathbf{x}, r)$ increases rapidly as violations of $x_1 + x_2 = 4$ increase, that is, as you move away from this line. This bunching and elongation of the contours of $P(\mathbf{x}, r)$ shows itself in the condition number of $\nabla^2 P(\mathbf{x}, r)$, the Hessian matrix of P. As shown in Appendix A, the condition number of a positive-definite matrix is the ratio of the largest to smallest eigenvalue. Because for large values of r, the eigenvalue ratio is large, $\nabla^2 P$ is said to be *ill-conditioned.* In fact, the condition number of $\nabla^2 P$ approaches ∞ as $r \to \infty$ (see Luenberger, 1984), so P becomes harder and harder to minimize accurately.

TABLE 8.1
Effect of penalty weighting coefficient r on minimum of f

r	x_1	x_2	f
0	1.0000	2.0000	0.0000
0.1	1.0833	2.0833	0.0833
1	1.3333	2.3333	0.3333
10	1.4762	2.4762	0.4762
100	1.4975	2.4975	0.4975
1000	1.4998	2.4998	0.4998
$\mathbf{x}^*$	1.5000	2.5000	0.5000

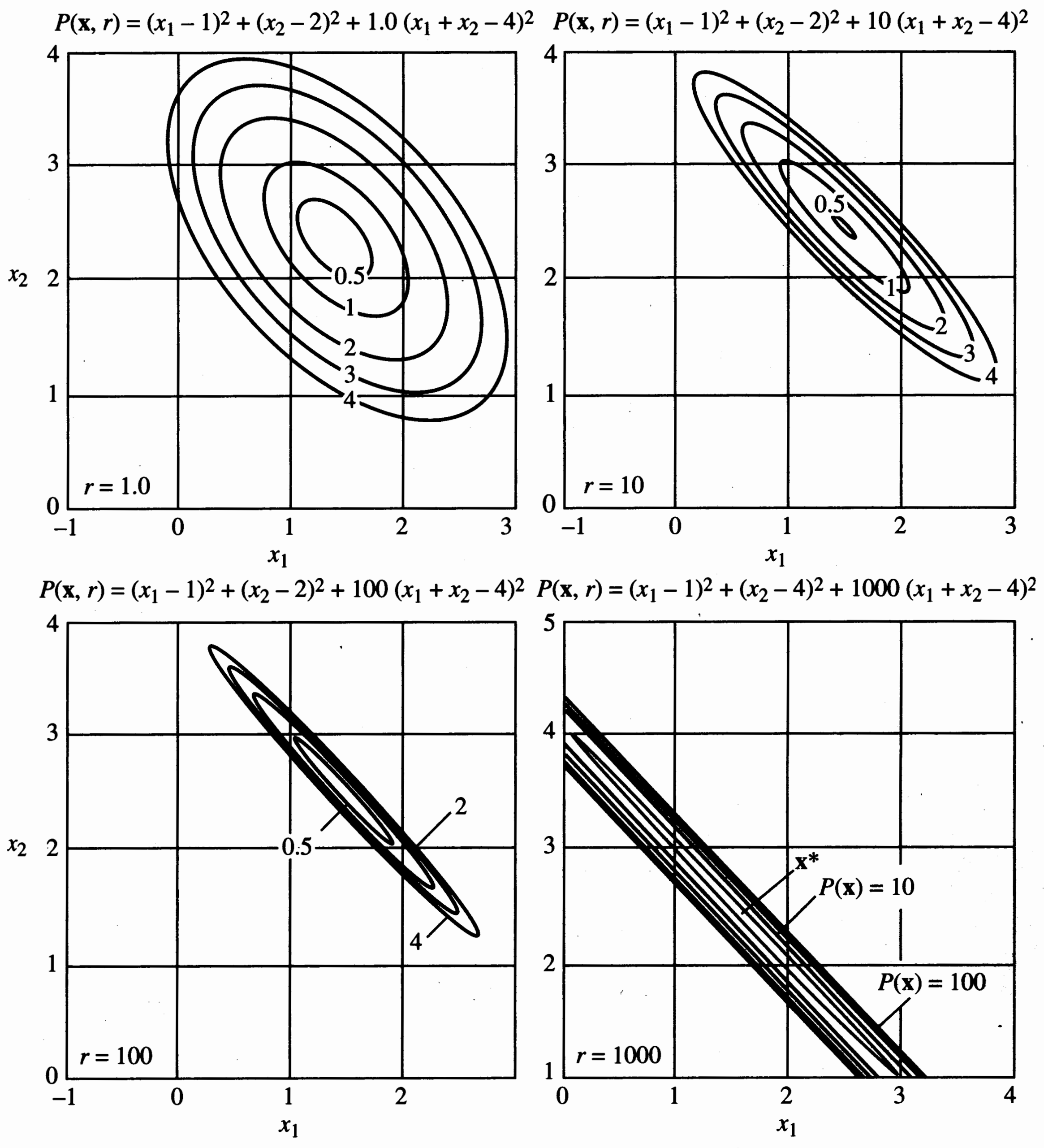

FIGURE 8.5
Transformation of a constrained problem to an unconstrained equivalent problem. The contours of the unconstrained penalty function are shown for different values of r.

The condition number of the Hessian matrix of the objective function is an important measure of difficulty in unconstrained optimization. By definition, the smallest a condition number can be is 1.0. A condition number of 10^5 is moderately large, 10^9 is large, and 10^{14} is extremely large. Recall that, if Newton's method is used to minimize a function f, the Newton search direction $\mathbf{s}$ is found by solving the linear equations

$$(\nabla^2 f)\mathbf{s} = -\nabla f$$

These equations become harder and harder to solve numerically as $\nabla^2 f$ becomes more ill-conditioned. When its condition number exceeds 10^{14}, there will be few if any correct digits in the computed solution using double precision arithmetic (see Luenberger, 1984).

Because of the occurrence of ill-conditioning, "pure" penalty methods have been replaced by more efficient algorithms. In SLP and SQP, a "merit function" is used within the line search phase of these algorithms.

The general form of the quadratic penalty function for a problem of the form (8.25)–(8.26) with both equality and inequality constraints is

$$P_2(\mathbf{x},r) = f(\mathbf{x}) + r\left(\sum_{j=1}^{m} h_j^2(\mathbf{x}) + \sum_{j=1}^{r} [\max\{0, g_j(\mathbf{x})\}]^2 \right) \tag{8.39}$$

The maximum-squared term ensures that a positive penalty is incurred only when the $g_j \leq 0$ constraint is violated.

An exact penalty function

Consider the exact L_1 penalty function; The term "L_1" means that the L_1 (absolute value) norm is used to measure infeasibilities.

$$P_1(\mathbf{x},\mathbf{w1},\mathbf{w2}) = f(\mathbf{x}) + \left[\sum_{j=1}^{m} w1_j |h_j(\mathbf{x})| + \sum_{j=1}^{r} w2_j \max\{0, g_j(\mathbf{x})\} \right] \tag{8.40}$$

where the $w1_j$ and $w2_j$ are positive weights. The second term in h_j produces the same effect as the squared terms in Equation (8.39). When a constraint is violated, there is a positive contribution to the penalty term equal to the amount of the violation rather than the squared amount. In fact, this "sum of violations" or *sum of infeasibilities* is the objective used in phase one of the simplex method to find a feasible solution to a linear program (see Chapter 7).

Let $\mathbf{x}^*$ be a local minimum of the problem (8.25)–(8.26), and let $(\boldsymbol{\lambda}^*, \mathbf{u}^*)$ be a vector of optimal multipliers corresponding to $\mathbf{x}^*$, that is, $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*)$ satisfy the KTC (8.27)–(8.29). If

$$w1_j \geq |\lambda_j^*|, \quad j = 1, \ldots, m \tag{8.41}$$

$$w2_j \geq |u_j^*|, \quad j = 1, \ldots, r \tag{8.42}$$

then $\mathbf{x}^*$ is a local minimum of $P_1(\mathbf{x}, \mathbf{w1}, \mathbf{w2})$. For a proof, see Luenberger (1984). If each penalty weight is larger than the absolute value of the corresponding optimal multiplier, the constrained problem can be solved by a *single* unconstrained minimization of P_1. The penalty weights do not have to approach $+\infty$, and no infinite ill-conditioning occurs. This is why P_1 is called "exact." There are other exact penalty functions; for example, the "augmented Lagrangian" will be discussed subsequently.

Intuitively, P_1 is exact and the squared penalty function P_2 is not because squaring a small infeasibility makes it much smaller, that is, $(10^{-4})^2 = 10^{-8}$. Hence the penalty parameter r in P_2 must increase faster as the infeasibilities get small, and it can never be large enough to make all infeasibilities vanish.

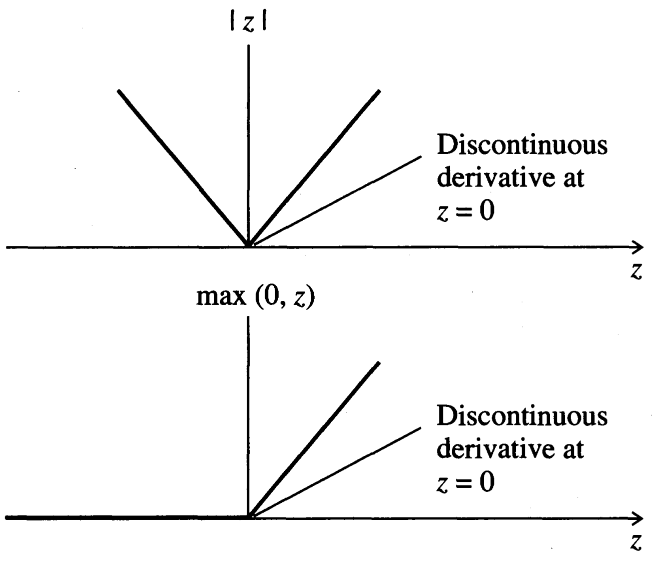
|z|
Discontinuous derivative at z = 0
z
max (0, z)
Discontinuous derivative at z = 0
z

FIGURE 8.6
Discontinuous derivatives in the P_1 penalty function.

Despite the "exactness" feature of P_1, no general-purpose, widely available NLP solver is based solely on the L_1 exact penalty function P_1. This is because P_1 also has a negative characteristic; it is nonsmooth. The term $|h_j(\mathbf{x})|$ has a discontinuous derivative at any point $\mathbf{x}$ where $h_j(\mathbf{x}) = 0$, that is, at any point satisfying the jth equality constraint; in addition, $\max\{0, g_j(\mathbf{x})\}$ has a discontinuous derivative at any $\mathbf{x}$ where $g_j(\mathbf{x}) = 0$, that is, whenever the jth inequality constraint is active, as illustrated in Figure 8.6. These discontinuities occur at any feasible or partially feasible point, so none of the efficient unconstrained minimizers for smooth problems considered in Chapter 6 can be applied, because they eventually encounter points where P_1 is nonsmooth.

An equivalent smooth constrained problem

The problem of minimizing P_1 subject to no constraints is equivalent to the following smooth constrained problem.

$$\text{Minimize:}\quad f(\mathbf{x}) + \sum_{j=1}^{m} w1_j(p1_j + n1_j) + \sum_{j=1}^{r} w2_j(p2_j) \tag{8.43}$$

$$\text{Subject to:}\quad h_j(\mathbf{x}) = p1_j - n1_j,\quad j = 1, \ldots, m \tag{8.44}$$

$$g_j(\mathbf{x}) = p2_j - n2_j,\ j = 1, \ldots, r \tag{8.45}$$

$$\text{all } p1_j, p2_j, n1_j, n2_j \geq 0 \tag{8.46}$$

The p's are "positive deviation" variables and the n's "negative deviation" variables. $p1_j$ and $p2_j$ equal h_j and g_j, respectively, when h_j and g_j are positive, and $n1_j$

and $n2_j$ equal h_j and g_j, respectively, when h_j and g_j are negative, providing that at most one variable in each pair $(p1_j, n1_j)$ and $(p2_j, u2_j)$ is positive, that is,

$$p1_j n1_j = 0, \; p2_j n2_j = 0 \tag{8.47}$$

But Equation (8.47) must hold at any optimal solution of (8.43)–(8.46), as long as all weights $w1_j$ and $w2_j$ are positive. To see why, consider the example $h_1 = -3$, $p1_1 = 2$, $n1_1 = 5$. The objective (8.43) contains a term $w1_1\,(p1_1 + n1_1) = 7w1_1$. The new solution $p1_1 = 0$, $n1_1 = 3$ has an objective contribution of $5w1_1$, so the old solution cannot be optimal.

When (8.44)–(8.47) hold,

$$p1_j + n1_j = |h_j(\mathbf{x})|$$

and

$$p2_j = \max(0, g_j(\mathbf{x}))$$

so the objective (8.43) equals the L_1 exact penalty function (8.40).

The problem (8.43)–(8.46) is called an "elastic" formulation of the original "inelastic" problem (8.11), because the deviation variables allow the constraints to "stretch" (i.e., be violated) at costs per unit of violation $w1_j$ and $w2_j$. This idea of allowing constraints to be violated, but at a price, is an important modeling concept that is widely used. Constraints expressing physical laws or "hard" limits cannot be treated this way—this is equivalent to using infinite weights. However many other constraints are really "soft," for example some customer demands and capacity limits. For further discussions of elastic programming, see Brown (1997). Curve-fitting problems using absolute value (L_1) or minimax (L_∞) norms can also be formulated as smooth constrained problems using deviation variables, as can problems involving multiple objectives, using "goal programming" (Rustem, 1998).

Augmented Lagrangians

The "augmented Lagrangian" is a smooth exact penalty function. For simplicity, we describe it for problems having only equality constraints, but it is easily extended to problems that include inequalities. The augmented Lagrangian function is

$$AL(\mathbf{x}, \boldsymbol{\lambda}, r) = f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j h_j(\mathbf{x}) + r \sum_{j=1}^{m} h_j^2(\mathbf{x}) \tag{8.48}$$

where r is a positive penalty parameter, and the λ_j are Lagrange multipliers. AL is simply the Lagrangian L plus a squared penalty term. Let $\mathbf{x}^*$ be a local minimum of the equality constrained problem

$$\text{Minimize:} \quad f(\mathbf{x})$$

$$\text{Subject to:} \quad h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, m$$

and let $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ satisfy the KTC for this problem. The gradient of AL is

$$\nabla_x AL(\mathbf{x}, \boldsymbol{\lambda}, r) = \nabla f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j \nabla h_j(\mathbf{x}) + 2r \sum_{j=1}^{m} h_j(\mathbf{x}) \nabla h_j(\mathbf{x}) \tag{8.49}$$

Since $\mathbf{x}^*$ is feasible, $h_j(\mathbf{x}^*) = 0$, so if $\boldsymbol{\lambda}$ is set to $\boldsymbol{\lambda}^*$ in the augmented Lagrangian,

$$\nabla_x AL(\mathbf{x}^*, \boldsymbol{\lambda}^*, r) = \nabla f(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j^* \nabla h_j(\mathbf{x}^*) = 0 \qquad (8.50)$$

Hence $\mathbf{x}^*$ is a stationary point of $AL(\mathbf{x}, \boldsymbol{\lambda}^*, r)$ for any r. Not all stationary points are minima, but if $\nabla_x^2 AL(\mathbf{x}^*, \boldsymbol{\lambda}^*, r)$ is positive-definite, then $\mathbf{x}^*$ satisfies the second-order sufficiency conditions, and so it is a local minimum. Luenberger (1984) shows that this is true if r is large enough, that is, there is a threshold $\bar{r} > 0$ such that, if $r > \bar{r}$, then $\nabla_x^2 AL(\mathbf{x}^*, \boldsymbol{\lambda}^*, r)$ is positive-definite. Hence for $r > \bar{r}$, $AL(\mathbf{x}, \boldsymbol{\lambda}^*, r)$ is an exact penalty function.

Again, there is a "catch." In general, $\bar{r}$ and $\boldsymbol{\lambda}^*$ are unknown. Algorithms have been developed that perform a sequence of minimizations of AL, generating successively better estimates of $\boldsymbol{\lambda}^*$ and increasing r if necessary [see Luenberger (1984)]. However, NLP solvers based on these algorithms have now been replaced with better ones based on the SLP, SQP, or GRG algorithms described in this chapter. The function AL does, however, serve as a line search objective in some SQP implementations; see Nocedal and Wright (1999).

Barrier methods

Like penalty methods, *barrier methods* convert a constrained optimization problem into a series of unconstrained ones. The optimal solutions to these unconstrained subproblems are in the interior of the feasible region, and they converge to the constrained solution as a positive barrier parameter approaches zero. This approach contrasts with the behavior of penalty methods, whose unconstrained subproblem solutions converge from outside the feasible region.

To illustrate, consider the example used at the start of Section 8.4 to illustrate penalty methods, but with the equality constraint changed to an inequality:

$$\text{Minimize:} \quad f(\mathbf{x}) = (x_1 - 1)^2 + (x_2 - 2)^2$$

$$\text{Subject to:} \quad g(\mathbf{x}) = x_1 + x_2 - 4 \geq 0$$

The equality constrained problem was graphed in Figure 8.5. The feasible region is now the set of points on and above the line $x_1 + x_2 - 4 = 0$, and the constrained solution is still at the point (1.5, 2.5) where $f = 0.5$.

The logarithmic barrier function for this problem is

$$\begin{aligned} B(\mathbf{x}, r) &= f(\mathbf{x}) - r\ln(g(\mathbf{x})) \\ &= (x_1 - 1)^2 + (x_2 - 2)^2 - r\ln(x_1 + x_2 - 4) \end{aligned}$$

where r is a positive scalar called the barrier parameter. This function is defined only in the interior of the feasible region, where $g(\mathbf{x})$ is positive. Consider minimizing B starting from an interior point. As $\mathbf{x}$ approaches the constraint boundary, $g(\mathbf{x})$ approaches zero, and the barrier term $-r\ln(g(\mathbf{x}))$ approaches infinity, so it creates an infinitely high barrier along this boundary. The penalty forces B to have an

TABLE 8.2
Convergence of barrier function $B(\mathbf{x}, r)$

Barrier parameter, r	$x_1(r)$	$x_2(r)$	Objective	Value of the constraint	Barrier term	$B(\mathbf{x}, r)$
10	2.851	3.851	6.851	2.702	9.938	−3.088
5	2.396	3.396	3.896	1.791	2.915	0.981
1	1.809	2.809	1.309	0.618	−0.481	1.790
0.1	1.546	2.546	0.596	0.092	−0.239	0.835
0.01	1.505	2.505	0.510	0.010	−0.046	0.556
$\mathbf{x}^*$	1.500	2.500	0.500	0.000	0.000	0.500

unconstrained minimum in the interior of the feasible region, and its location depends on the barrier parameter r. If $\mathbf{x}(r)$ is an unconstrained interior minimum of $B(\mathbf{x}, r)$, then as r approaches zero, the barrier term has a decreasing weight, so $\mathbf{x}(r)$ can approach the boundary of the feasible region if the constrained solution is on the boundary. As r approaches zero, $\mathbf{x}(r)$ approaches an optimal solution of the original problem, as shown in Nash and Sofer (1996) and Nocedal and Wright (1999).

To illustrate this behavior, Table 8.2 shows the optimal unconstrained solutions and their associated objective, constraint, and barrier function values for the preceding problem, for a sequence of decreasing r values.

For larger r values, $\mathbf{x}(r)$ is forced further from the constraint boundary. In contrast, as r approaches zero, $x_1(r)$ and $x_2(r)$ converge to their optimal values of 1.5 and 2.5, respectively, and the constraint value approaches zero. The term $-\ln(g(\mathbf{x}))$ approaches infinity, but the weighted barrier term $-r\ln(g(\mathbf{x}))$ approaches zero, and the value of B approaches the optimal objective value.

For a general problem with only inequality constraints:

$$\text{Minimize:} \quad f(\mathbf{x})$$

$$\text{Subject to:} \quad g_i(\mathbf{x}) \geq 0, \quad i = 1, \ldots, m$$

the logarithmic barrier function formulation is

$$\text{Minimize:} \quad B(\mathbf{x}, r) = f(\mathbf{x}) - r \sum_{i=1}^{m} \ln(g_i(\mathbf{x}))$$

As with penalty functions, the condition number of the Hessian matrix $\nabla_x^2 B(\mathbf{x}(r), r)$ approaches infinity as r approaches zero, so B is very difficult to minimize accurately for small r. From a geometric viewpoint, this is because the barrier term approaches infinity rapidly as you move toward the boundary of the feasible region, so the contours of B "bunch up" near this boundary. Hence the barrier approach is not widely used today as a direct method of solving nonlinear programs. When a logarithmic barrier term is used to incorporate only the bounds on the variables, however, this leads to a barrier or *interior-point* method. This approach is very successful in solving large linear programs and is very promising for NLP problems as well. See Nash and Sofer (1996) or Nocedal and Wright (1999) for further details.

Barrier methods are not directly applicable to problems with equality constraints, but equality constraints can be incorporated using a penalty term and inequalities can use a barrier term, leading to a "mixed" penalty–barrier method.

8.5 SUCCESSIVE LINEAR PROGRAMMING

Successive linear programming (SLP) methods solve a sequence of linear programming approximations to a nonlinear programming problem. Recall that if $g_i(\mathbf{x})$ is a nonlinear function and $\mathbf{x}^0$ is the initial value for $\mathbf{x}$, then the first two terms in the Taylor series expansion of $g_i(\mathbf{x})$ around $\mathbf{x}^0$ are

$$g_i(\mathbf{x}) = g_i(\mathbf{x}^0 + \Delta\mathbf{x}) \cong g_i(\mathbf{x}^0) + \nabla g_i(\mathbf{x}^0)^T(\Delta\mathbf{x})$$

The error in this linear approximation approaches zero proportionally to $(\Delta\mathbf{x})^2$ as $\Delta\mathbf{x}$ approaches zero. Given initial values for the variables, all nonlinear functions in the problem are linearized and replaced by their linear Taylor series approximations at this initial point. The variables in the resulting LP are the $\Delta\mathbf{x}_i$'s, representing changes from the base values. In addition, upper and lower bounds (called *step bounds*) are imposed on these change variables because the linear approximation is reasonably accurate only in some neighborhood of the initial point.

The resulting LP is solved; if the new point is an improvement, it becomes the current point and the process is repeated. If the new point does not represent an improvement in the objective, we may be close enough to the optimum to stop or the step bounds may need to be reduced. Successive points generated by this procedure need not be feasible even if the initial point is. The extent of infeasibility generally is reduced as the iterations proceed, however.

We illustrate the basic concepts with a simple example. Consider the following problem:

$$\begin{aligned} \text{Maximize:} \quad & 2x + y \\ \text{Subject to:} \quad & x^2 + y^2 \le 25 \\ & x^2 - y^2 \le 7 \end{aligned}$$

and

$$\begin{aligned} x &\ge 0 \\ y &\ge 0 \end{aligned}$$

with an initial starting point of $(x_c, y_c) = (2, 2)$. Figure 8.7 shows the two nonlinear constraints and one objective function contour with an objective value of 10. Because the value of the objective function increases with increasing x and y, the figure shows that the optimal solution is at the point where the two nonlinear inequalities $x^2 + y^2 \le 25$ and $x^2 - y^2 \le 7$ are active, that is, at the solution of $x^2 + y^2 = 25$ and $x^2 - y^2 = 7$, which is $\mathbf{x}^* = (4, 3)$.

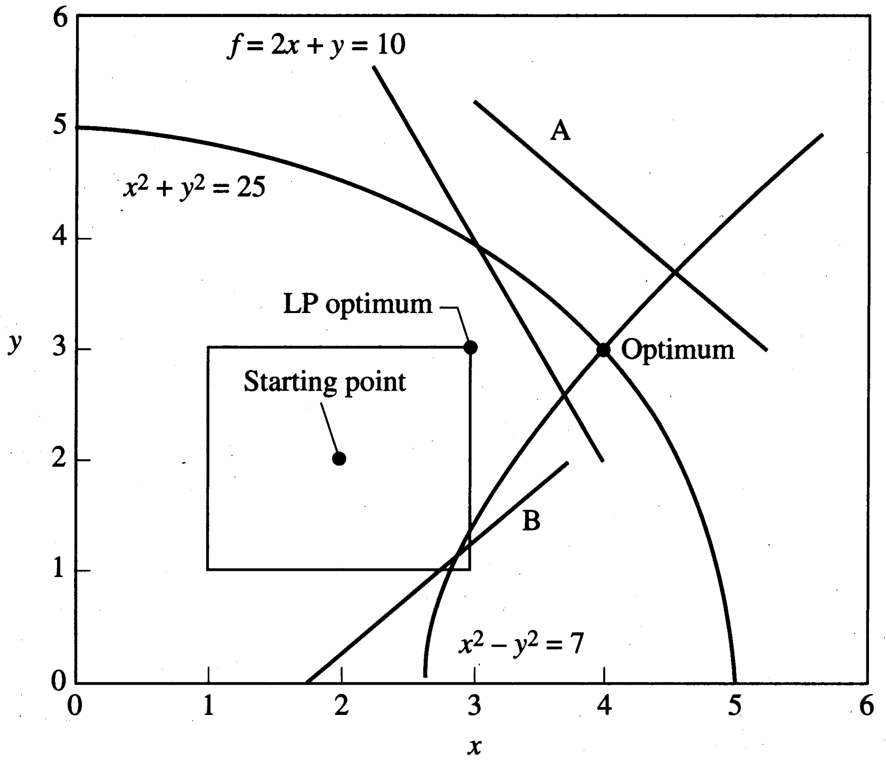

FIGURE 8.7
SLP example with linear objective, nonlinear constraints. Line A is the linearization of $x^2 + y^2 \leq 25$ and line B is the linearization of $x^2 - y^2 \leq 7$.

Next consider any optimization problem with n variables. Let $\bar{\mathbf{x}}$ be any feasible point, and let $n_{\text{act}}(\bar{\mathbf{x}})$ be the number of active constraints at $\bar{\mathbf{x}}$. Recall that a constraint is active at $\bar{\mathbf{x}}$ if it holds as an equality constraint there. Hence all equality constraints are active at any feasible point, but an inequality constraint may be active or inactive. Remember to include simple upper or lower bounds on the variables when counting active constraints. We define the number of degrees of freedom at $\bar{\mathbf{x}}$ as

$$\text{dof}(\bar{\mathbf{x}}) = n - n_{\text{act}}(\bar{\mathbf{x}})$$

Definition: A feasible point $\bar{\mathbf{x}}$ is called a *vertex* if $\text{dof}(\bar{\mathbf{x}}) \leq 0$, and the Jacobian of the active constraints at $\bar{\mathbf{x}}$ has rank n where n is the number of variables. It is a *nondegenerate* vertex if $\text{dof}(\bar{\mathbf{x}}) = 0$, and a *degenerate* vertex if $\text{dof}(\bar{\mathbf{x}}) < 0$, in which case $|\text{dof}(\bar{\mathbf{x}})|$ is called the *degree of degeneracy* at $\bar{\mathbf{x}}$.

The requirement that there be at least n independent linearized constraints at $\bar{\mathbf{x}}$ is included to rule out situations where, for example, some of the active constraints are just multiples of one another. In the example $\text{dof}(\bar{\mathbf{x}}) = 0$.

Returning to the example, the optimal point $\mathbf{x}^* = (4, 3)$ is a nondegenerate vertex because

$$n = 2, \quad n_{\text{act}}(\bar{\mathbf{x}}) = 2$$

and

$$\text{dof}(\bar{\mathbf{x}}) = 2 - 2 = 0$$

Clearly a vertex is a point where n or more independent constraints intersect in n-dimensional space to produce a point. Recall the discussion of LPs in Chapter 7; if an LP has an optimal solution, an optimal vertex (or extreme point) solution exists. Of course, this rule is not true for nonlinear problems. Optimal solutions $\mathbf{x}^*$ of unconstrained NLPs have $\text{dof}(\bar{\mathbf{x}}) = n$, since $n_{\text{act}}(\bar{\mathbf{x}}) = 0$ (i.e., there are no constraints). Hence $\text{dof}(\bar{\mathbf{x}})$ measures how tightly constrained the point $\bar{\mathbf{x}}$ is, ranging from no active constraints $(\text{dof}(\bar{\mathbf{x}}) = n)$ to completely determined by active constraints $(\text{dof}(\bar{\mathbf{x}}) \le 0)$. Degenerate vertices have "extra" constraints passing through them, that is, more than n pass through the same point. In the example, one can pass any number of redundant lines or curves through (4, 3) in Figure 8.7 without affecting the feasibility of the optimal point.

If $\text{dof}(\bar{\mathbf{x}}) = n - n_{\text{act}}(\bar{\mathbf{x}}) = d > 0$, then there are more problem variables than active constraints at $\bar{\mathbf{x}}$, so the $(n - d)$ active constraints can be solved for $n - d$ dependent or basic variables, each of which depends on the remaining d independent or nonbasic variables. Generalized reduced gradient (GRG) algorithms use the active constraints at a point to solve for an equal number of dependent or basic variables in terms of the remaining independent ones, as does the simplex method for LPs.

Continuing with the example, we linearize each function about $(x_c, y_c) = (2, 2)$ and impose step bounds of 1 on both Δx and Δy, leading to the following LP:

$$\begin{aligned}
\text{Maximize:}\quad & 2x_c + y_c + 2\Delta x + \Delta y = 2\Delta x + \Delta y + 6\\
\text{Subject to:}\quad & x_c^2 + y_c^2 + 2x_c\Delta x + 2y_c\Delta y = 4\Delta x + 4\Delta y + 8 \le 25\\
& x_c^2 - y_c^2 + 2x_c\Delta x - 2y_c\Delta y = 4\Delta x - 4\Delta y \le 7\\
& 2 + \Delta x \ge 0, \qquad 2 + \Delta y \ge 0\\
& -1 \le \Delta x \le 1, \qquad -1 \le \Delta y \le 1
\end{aligned}$$

The first two bounds require that the new point $(2 + \Delta x, 2 + \Delta y)$ satisfy the original bounds. The second two bounds, called *step bounds*, are imposed to ensure that the errors between the nonlinear problem functions and their linearizations are not too large.

Rearranging terms in the linearized LP yields the following SLP subproblem:

$$\begin{aligned}
\text{Maximize:}\quad & 2\Delta x + \Delta y\\
\text{Subject to:}\quad & \Delta x + \Delta y \le 4.25\\
& \Delta x - \Delta y \le 1.75
\end{aligned}$$

and

$$-1 \le \Delta x \le 1$$
$$-1 \le \Delta y \le 1$$

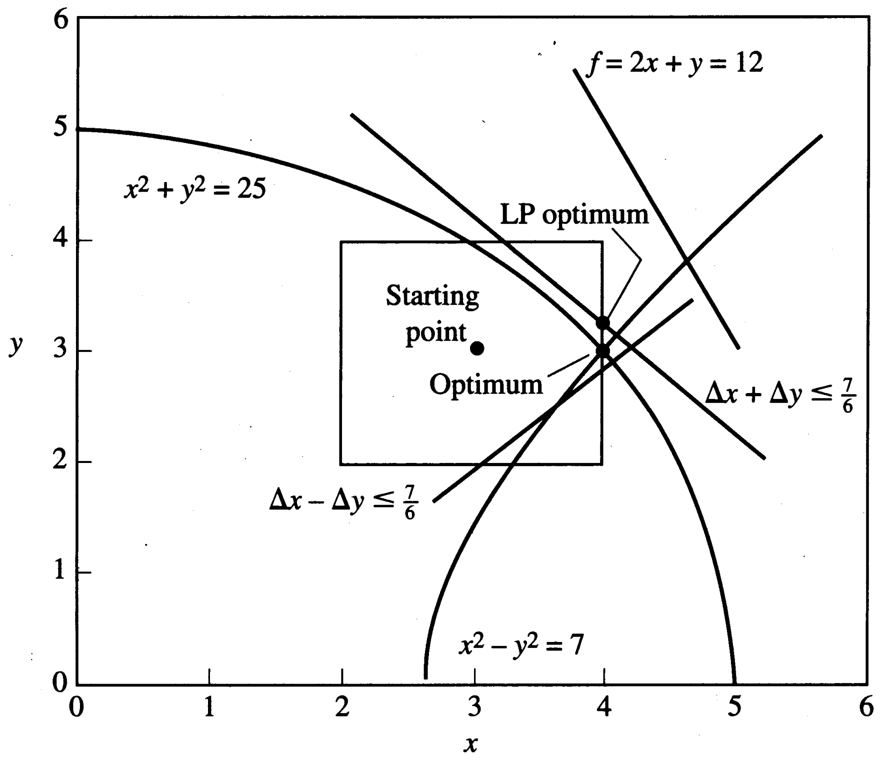

FIGURE 8.8
SLP example with linear objective, nonlinear constraints.

Figure 8.7 also shows these LP constraints. Its optimal solution is at $(\Delta x, \Delta y) = (1, 1)$, which gives $(x_n, y_n) = (3, 3)$. This point is determined entirely by the step bounds. This is an improved point, as can be seen by evaluating the original functions, so we set $x_c = x_n$ and repeat these steps to get the next LP.

$$\begin{aligned} \text{Maximize:} \quad & 2\Delta x + \Delta y \\ \text{Subject to:} \quad & \Delta x + \Delta y \leq \tfrac{7}{6} \\ & \Delta x - \Delta y \leq \tfrac{7}{6} \end{aligned}$$

and

$$-1 \leq \Delta x \leq 1$$
$$-1 \leq \Delta y \leq 1$$

The feasible region can be seen in Figure 8.8 and the optimal solution is at $(\Delta x, \Delta y) = (1, \frac{1}{6})$ or $(x_n, y_n) = (4, 3.167)$. This point is at the intersection of the constraints $\Delta x + \Delta y \leq \frac{7}{6}$ and $\Delta x = 1$, so one step bound is still active at the LP optimum.

The SLP subproblem at (4, 3.167) is shown graphically in Figure 8.9. The LP solution is now at the point (4, 3.005), which is very close to the optimal point $\mathbf{x}^*$. This point $(\mathbf{x}_n)$ is determined by linearization of the two active constraints, as are all further iterates. Now consider Newton's method for equation-solving applied to the two active constraints, $x^2 + y^2 = 25$ and $x^2 - y^2 = 7$. Newton's method involves

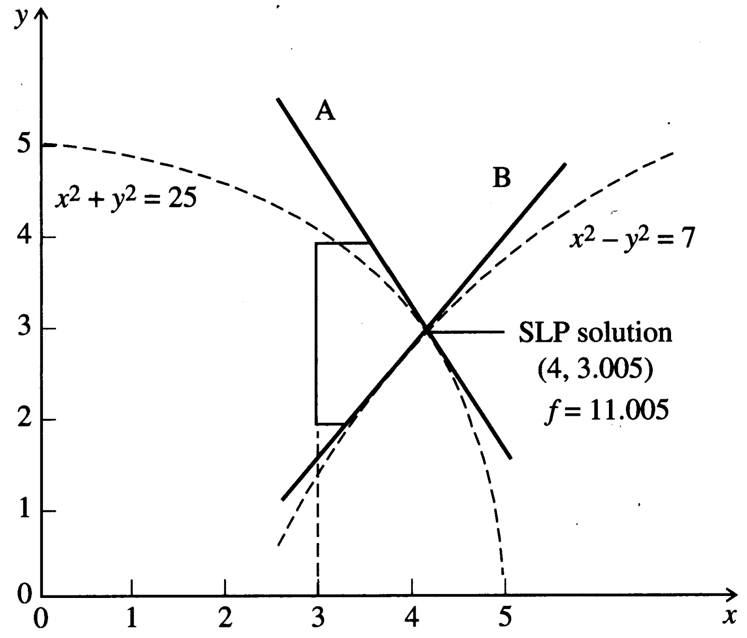

FIGURE 8.9
The optimal point after solving the third SLP subproblem. A is the linearization of $x^2 + y^2 = 25$ and B is the linearization of $x^2 - y^2 = 7$.

linearizing these two equations and solving for $(\Delta x, \Delta y)$, exactly as SLP is now doing. Hence, when SLP converges to a vertex optimum, it eventually becomes Newton's method applied to the active constraints. As discussed in Chapter 5, the method has quadratic convergence, that is, the new error is bounded by a constant times the previous error squared. This is the most rapid convergence we could hope to obtain, so SLP is very efficient when the optimum is at a constraint vertex.

SLP convergence is much slower, however, when the point it is converging toward is not a vertex. To illustrate, we replace the objective of the example with $x + 2y$. This rotates the objective contour counterclockwise, so when it is shifted upward, the optimum is at $\mathbf{x}^* = (2.2, 4.4)$, where only one constraint, $x^2 + y^2 \leq 25$, is active. Because the number of degrees of freedom at $\mathbf{x}^*$ is $2 - 1 = 1$, this point is not a vertex. Figure 8.10 shows the feasible region of the SLP subproblem starting at (2, 5), using step bounds of 1.0 for both Δx and Δy.

The point (2, 5) is slightly infeasible, and the SLP subproblem is

$$\begin{aligned} \text{Maximize:} \quad & f = \Delta x + 2\Delta y \\ & 4\Delta x + 10\Delta y \leq -4 \\ \text{Subject to:} \quad & -1 \leq \Delta x \leq 1 \\ & -1 \leq \Delta y \leq 1 \end{aligned}$$

We ignore the constraint $x^2 - y^2 \leq 7$ because its linearization is redundant in this subproblem. The LP optimum is at $\Delta x = 1, \Delta y = -0.8$, so the new point is (3, 4.2),

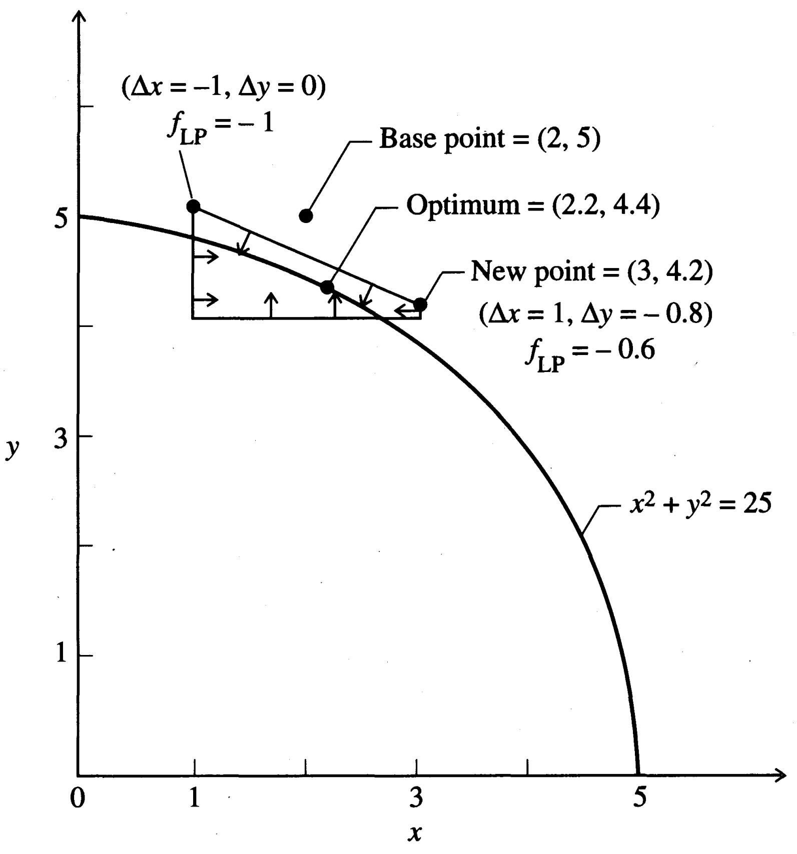

FIGURE 8.10
SLP subproblem at (2, 5) for the revised example ($f = x + 2y$).

which is on the "other side" of the optimum. If we continue this process without reducing the step bounds, the iterates will oscillate about the optimum and never converge to it because the new point will always be at the intersection of the linearized constraint and a step bound.

The penalty SLP algorithm (PSLP), described in Zhang et al. (1985) and discussed in the next section, contains logic for reducing the step bounds so that convergence to the optimal solution is guaranteed. The sequence of points generated by PSLP for this problem, starting at (2, 5), with initial step bounds of 0.9, is shown in Table 8.3. The algorithm converges, but much more slowly than before. The rate of convergence is linear, as occurs in the steepest descent method for unconstrained optimization. The step bounds must be reduced to force convergence, as is shown in the "max step bound" column. The significance of the "ratio" column is explained in the next section.

8.5.1 Penalty Successive Linear Programming

The PSLP algorithm is a steepest descent procedure applied to the exact L_1 penalty function (see Section 8.4). It uses a trust region strategy (see Section 6.3.2) to guar-

TABLE 8.3
Convergence of PSLP on the modified Griffith–Stewart problem

Iteration	Objective	Sum of infeasibilities	Ratio	Max step bound
0	12.000	4.000		0.900
1	11.2900	0.538	0.870	0.900
2	11.2169	0.238	0.560	0.900
3	11.2247	0.251	−0.060	0.450
4	11.1950	0.065	0.720	0.450
5	11.1821	0.064	0.020	0.225
6	11.1810	0.015	0.760	0.450
7	11.1903	0.064	−3.080	0.225
8	11.1839	0.016	−0.010	0.113
9	11.1807	3.94E-03*	0.750	0.113
10	11.1812	3.97E-03	−0.010	0.056
11	11.1805	9.86E-04	0.750	0.056
12	11.1805	9.89E-03	0.000	0.028
13	11.1804	2.46E-04	0.750	0.028
14	11.1804	2.46E-04	0.000	0.014
15	11.1803	6.16E-05	0.750	0.014
16	11.1804	2.47E-04	−3.010	0.007
17	11.1804	6.17E-05	0.000	0.003
18	11.1803	0.000	0.750	0.003
OPT	11.1803	0.000		

*E-03 represents 10^{-3}.

antee convergence. To explain PSLP, we begin with an NLP in the following general form:

$$\text{Minimize:} \quad f(\mathbf{x})$$
$$\text{Subject to:} \quad \mathbf{g}(\mathbf{x}) = \mathbf{b} \tag{8.51}$$

and

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{8.52}$$

Any inequalities have been converted to equalities using slack variables, which are included in **x**. The exact L_1 penalty function for this problem is

$$P(\mathbf{x},w) = f(\mathbf{x}) + w\sum_{i=1}^{m} |g_i(\mathbf{x}) - b_i| \tag{8.53}$$

If the penalty weight w is larger than the maximum of the absolute multiplier values for the problem, then minimizing $P(x, w)$ subject to $\mathbf{l} \le \mathbf{x} \le \mathbf{u}$ is equivalent to minimizing f in the original problem. Often, such a threshold is known in advance, say from the solution of a closely related problem. If w is too small, PSLP will usually converge to an infeasible local minimum of P, and w can then be increased. Infeasibility in the original NLP is detected if several increases of w fail to yield a feasible point. In the following, we drop the dependence of P on w, calling it simply $P(\mathbf{x})$.

Let $\mathbf{x}^k$ be the value of $\mathbf{x}$ at the start of PSLP iteration k. A piecewise linear function that closely approximates $P(\mathbf{x})$ for $\mathbf{x}$ near $\mathbf{x}^k$ is

$$P1(\Delta\mathbf{x},\mathbf{x}^k) = f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)\Delta x + w\sum_{i=1}^{m} |g_i(\mathbf{x}^k) + \nabla g_i^T(\mathbf{x}^k)\Delta\mathbf{x} - b_i| \quad (8.54)$$

As Δx approaches 0, $P1(\Delta\mathbf{x}, \mathbf{x}^k)$ approaches $P(\mathbf{x}^k)$, so $P1$ approximates P arbitrarily well if $\Delta\mathbf{x}$ is small enough. We ensure that $\Delta\mathbf{x}$ is small enough by imposing the step bounds

$$-\mathbf{s}^k \le \Delta\mathbf{x} \le \mathbf{s}^k \quad (8.55)$$

where $\mathbf{s}^k$ is a vector of positive step bounds at iteration k, which are varied dynamically during the execution of PSLP. We also want the new point $\mathbf{x}^k + \Delta\mathbf{x}$ to satisfy the original bounds, so we impose the constraints

$$\mathbf{l} \le \mathbf{x}^k + \Delta\mathbf{x} \le \mathbf{u} \quad (8.56)$$

The trust region problem is to choose $\Delta\mathbf{x}$ to minimize $P1$ in (8.54) subject to the trust region bounds (8.55) and (8.56). As discussed in Section (8.4), this piecewise linear problem can be transformed into an LP by introducing deviation variables p_i and n_i. The absolute value terms become $(p_i + n_i)$ and their arguments are set equal to $p_i - n_i$. The equivalent LP is

Problem LP($\mathbf{x}^k$, $\mathbf{s}^k$)

$$\text{Minimize:} \quad f + \nabla f^T\Delta\mathbf{x} + w\sum_i (p_i + n_i) \quad (8.57)$$

$$\text{Subject to:} \quad g_i + \nabla g_i^T\Delta\mathbf{x} - b_i = p_i - n_i, \quad i = 1, \dots, m \quad (8.58)$$

$$-\mathbf{s}^k \le \Delta\mathbf{x} \le \mathbf{s}^k, \quad \mathbf{l} \le \mathbf{x}^k + \Delta\mathbf{x} \le \mathbf{u}, \quad p \ge 0, \quad n \ge 0, \quad i = 1, \dots n$$

where all functions and gradients are evaluated at $\mathbf{x}^k$.

Let $\Delta\mathbf{x}^k$ solve LP ($\mathbf{x}^k$, $\mathbf{s}^k$). The new point $\mathbf{x}^k + \Delta\mathbf{x}^k$ is "better" than $\mathbf{x}^k$ if

$$P(\mathbf{x}^k + \Delta\mathbf{x}^k) < P(\mathbf{x}^k)$$

The actual reduction in P is

$$ared_k = P(\mathbf{x}^k) - P(\mathbf{x}^k + \Delta\mathbf{x}^k)$$

Of course, $ared_k$ can be negative because P need not be reduced if the step bounds $\mathbf{s}^k$ are too large. To decide whether $\mathbf{s}^k$ should be increased, decreased, or left the same, we compare $ared_k$ with the reduction predicted by the piecewise linear "model" or approximation to P, $P1$. This predicted reduction is

$$pred_k = P1(0,\mathbf{x}^k) - P1(\Delta\mathbf{x}^k,\mathbf{x}^k) \quad (8.59)$$

Remember that $\Delta\mathbf{x}^k$ solves LP $(\mathbf{x}^k, \mathbf{s}^k)$, $\Delta\mathbf{x} = 0$ is feasible in this LP, and $P1$ is its objective. Because the minimal objective value is never larger than the value at any feasible solution

$$pred_k \geq 0 \tag{8.60}$$

If $pred_k = 0$, then no changes $\Delta\mathbf{x}$ within the rectangular trust region (8.58) can reduce $P1$ below the value $P1(0, \mathbf{x}^k)$. Then $\mathbf{x}^k$ is called a stationary point of the nonsmooth function P, that is, the condition $pred_k = 0$ is analogous to the condition $\nabla f(\mathbf{x}^k) = \mathbf{0}$ for smooth functions. If $pred_k = 0$, the PSLP algorithm stops. Otherwise $pred_k > 0$, so we can compute the ratio of actual to predicted reduction.

$$ratio_k = \frac{ared_k}{pred_k} \tag{8.61}$$

Changes in the step bounds are based on $ratio_k$. Its ideal value is 1.0 because then the model function $P1$ agrees perfectly with the true function P. If the ratio is close to 1.0, we increase the step bounds; if it is far from 1.0, we decrease them; and if it is in between, no changes are made. To make this precise, we set two thresholds u and l; a ratio above u (typical value is 0.75) is "close" to 1.0, and a ratio below l (typical value is 0.25) is "far" from l. Then, the steps in PSLP iteration k are;

1. Solve the LP subproblem LP $(\mathbf{x}^k, \mathbf{s}^k)$, obtaining an optimal solution $\Delta\mathbf{x}^k$, and Lagrange multiplier estimates $\boldsymbol{\lambda}^k$. These are the LP multipliers for the equalities in (8.58).
2. Check the stopping criteria, including
   a. $pred_k$ is nearly zero.
   b. The KTC are nearly satisfied.
   c. $\mathbf{x}^k$ is nearly feasible and the fractional objective change is small enough.
3. Compute $ared_k$, $pred_k$, and $ratio_k$.
4. If $ratio_k < 0$, $\mathbf{s}^k \leftarrow \mathbf{s}^k/2$, go to step 1 (reject the new point).
5. $\mathbf{x}^k \leftarrow \mathbf{x}^k + \Delta\mathbf{x}^k$ (accept the new point).
6. If $ratio_k < l$, $\mathbf{s}^k \leftarrow \mathbf{s}^k/2$.
   If $ratio_k > u$, $\mathbf{s}^k \leftarrow 2\mathbf{s}^k$.
7. Go to step 1 with $k \leftarrow k + 1$.

Step 4 rejects the new point and decreases the step bounds if $ratio_k < 0$. This step can only be repeated a finite number of times because, as the step bounds approach zero, the ratio approaches 1.0. Step 6 decreases the size of the trust region if the ratio is too small, and increases it if the ratio is close to 1.0. Zhang et al. (1986) proved that a similar SLP algorithm converges to a stationary point of P from any initial point.

Table 8.3 shows output generated by this PSLP algorithm when it is applied to the test problem of Section 8.5 using the objective $x + 2y$. This version of the problem has a nonvertex optimum with one degree of freedom. We mentioned the slow linear convergence of PSLP in this problem previously. Consider the "ratio" and "max step bound" columns of Table 8.2. Note that very small positive or negative ratios occur at every other iteration, with each such occurrence forcing a reduction of all step bounds (they are divided by 2.0). After each reduction (once two reductions are needed), a positive ratio occurs and the new point is accepted. When the ratio is negative, the new point is rejected.

8.6 SUCCESSIVE QUADRATIC PROGRAMMING

Successive quadratic programming (SQP) methods solve a sequence of quadratic programming approximations to a nonlinear programming problem. Quadratic programs (QPs) have a quadratic objective function and linear constraints, and there exist efficient procedures for solving them; see Section 8.3. As in SLP, the linear constraints are linearizations of the actual constraints about the selected point. The objective is a quadratic approximation to the Lagrangian function, and the algorithm is simply Newton's method applied to the KTC of the problem.

Problem formulation with equality constraints

To derive SQP, we again consider a general NLP of the form (8.51)–(8.52), but temporarily ignore the bounds to simplify the explanation;

$$\text{Minimize:} \quad f(\mathbf{x}) \tag{8.62}$$

$$\text{Subject to:} \quad \mathbf{g}(\mathbf{x}) = \mathbf{b}$$

The Lagrangian function for this problem is

$$L(\mathbf{x},\boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{g}(\mathbf{x}) - \mathbf{b}) \tag{8.63}$$

and the KTC are

$$\nabla_{\mathbf{x}} L = \nabla f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}) = \mathbf{0} \tag{8.64}$$

and

$$\mathbf{g}(\mathbf{x}) = \mathbf{b} \tag{8.65}$$

As discussed in Section (8.2), Equations (8.64) and (8.65) is a set of $(n + m)$ nonlinear equations in the n unknowns $\mathbf{x}$ and m unknown multipliers $\boldsymbol{\lambda}$. Assume we have some initial guess at a solution $(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}})$. To solve Equations (8.64)–(8.65) by Newton's method, we replace each equation by its first-order Taylor series approximation about $(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}})$. The linearization of (8.64) with respect to $\mathbf{x}$ and $\boldsymbol{\lambda}$ (the arguments are suppressed)

$$\nabla_{\mathbf{x}} L + \nabla_x^2 L \Delta\mathbf{x} + \nabla\mathbf{g}^T \Delta\boldsymbol{\lambda} = \mathbf{0} \tag{8.66}$$

and that for (8.65) is

$$\mathbf{g} + \nabla\mathbf{g}\,\Delta\mathbf{x} = \mathbf{0} \tag{8.67}$$

In Equations (8.66)–(8.67) all functions and derivatives are evaluated at $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$, $\nabla \mathbf{g}$ is the Jacobian matrix of **g**, and $\nabla_x^2 L$ is the Hessian matrix of the Lagrangian.

$$\nabla_x^2 L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \nabla^2 f(\bar{\mathbf{x}}) + \sum_{i=1}^{m} \bar{\lambda}_i \nabla^2 g_i(\bar{\mathbf{x}}) \tag{8.68}$$

Note that second derivatives of all problem functions are now involved.

For problems with only equality constraints, we could simply solve the linear equations (8.66)–(8.67) for $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda})$ and iterate. To accommodate both equalities and inequalities, an alternative viewpoint is useful. Consider the quadratic programming problem

$$\text{Minimize:} \quad \nabla L^T \Delta\mathbf{x} + \tfrac{1}{2} \Delta\mathbf{x}^T \nabla_x^2 L \, \Delta\mathbf{x} \tag{8.69}$$

$$\text{Subject to:} \quad \mathbf{g} + \nabla \mathbf{g} \, \Delta\mathbf{x} = 0 \tag{8.70}$$

If we call the Lagrange multipliers for (8.70) $\Delta\boldsymbol{\lambda}$, the Lagrangian for the QP is

$$L_1(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}) = \nabla L^T \Delta\mathbf{x} + \tfrac{1}{2} \Delta\mathbf{x}^T \nabla_x^2 L \, \Delta\mathbf{x} + \Delta\boldsymbol{\lambda}^T (\mathbf{g} + \nabla \mathbf{g} \, \Delta\mathbf{x}) \tag{8.71}$$

Setting the derivatives of L_1 with respect to $\Delta\mathbf{x}$ and $\Delta\boldsymbol{\lambda}$ equal to zero yields the Newton equations (8.66)–(8.67) so they are the KTC of the QP (8.69)–(8.70). Hence in the equality-constrained case, we can compute the Newton step $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda})$ either by solving the linear equations (8.66)–(8.67) or by solving the QP (8.69)–(8.70).

Inclusion of both equality and inequality constraints

When the original problem has a mixture of equalities and inequalities, it can be transformed into a problem with equalities and simple bounds by adding slacks, so the problem has an objective function f, equalities (8.62), and bounds

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{8.72}$$

Repeating the previous development for this problem, Newton's method applied to the KTC yields a mixed system of equations and inequalities for the Newton step $(\Delta\mathbf{x}, \Delta\boldsymbol{\lambda})$. This system is the KTC for the QP in (8.69)–(8.70) with the additional bound constraints

$$\mathbf{l} \le \bar{\mathbf{x}} + \Delta\mathbf{x} \le \mathbf{u} \tag{8.73}$$

Hence the QP subproblem now has both equality and inequality constraints and must be solved by some iterative QP algorithm.

The approximate Hessian

Solving a QP with a positive-definite Hessian is fairly easy. Several good algorithms all converge in a finite number of iterations; see Section 8.3. However, the Hessian of the QP presented in (8.69), (8.70), and (8.73) is $\nabla_x^2 L\ (\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$, and this matrix need not be positive-definite, even if $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is an optimal point. In addition, to compute $\nabla_x^2 L$, one must compute second derivatives of all problem functions.

Both difficulties are eliminated by replacing $\nabla_x^2 L$ by a positive-definite quasi-Newton (QN) approximation **B**, which is updated using only values of L and $\nabla_x L$ (See Section 6.4 for a discussion of QN updates.) Most SQP algorithms use Powell's modification (see Nash and Sofer, 1996) of the BFGS update. Hence the QP subproblem becomes

QP($\bar{\mathbf{x}}$,B)

$$\text{Minimize:} \quad \nabla L^T \Delta\mathbf{x} + \tfrac{1}{2}\Delta\mathbf{x}^T \mathbf{B}\,\Delta\mathbf{x} \tag{8.74}$$

$$\text{Subject to:} \quad \nabla\mathbf{g}\,\Delta\mathbf{x} = -\mathbf{g}, \quad \mathbf{l} \le \bar{\mathbf{x}} + \Delta\mathbf{x} \le \mathbf{u} \tag{8.75}$$

The SQP line search

To arrive at a reliable algorithm, one more difficulty must be overcome. Newton and quasi-Newton methods may not converge if a step size of 1.0 is used at each step. Both trust region and line search versions of SQP have been developed that converge reliably [see Nocedal and Wright (1999) and Nash and Sofer (1996)]. A widely used line search strategy is to use the L_1 exact penalty function $P(\mathbf{x}, w)$ in (8.53) as the function to be minimized during the line search. This function also plays a central role in the PSLP algorithm discussed in Section 8.5. In a line search SQP algorithm, $P(\mathbf{x}, w)$ is used only to determine the step size along the direction determined by the QP solution, $\Delta\mathbf{x}$. Let $\mathbf{x}$ be the current iterate, and let $\Delta\mathbf{x}$ solve the QP subproblem, QP(**x**, **B**). The L_1 exact penalty function for the NLP problem is

$$P(\mathbf{x},w) = f(\mathbf{x}) + \sum_{i=1}^{m} w_i\,|g_i(\mathbf{x}) - b_i| \tag{8.76}$$

where a separate penalty weight w_i is used for each constraint. The SQP line search chooses a positive step size α to find an approximate minimum of

$$r(\alpha) = P(\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x}, w) \tag{8.77}$$

A typical line search algorithm, which uses the derivative of $r(\alpha)$ evaluated at $\alpha = 0$, denoted by $r'(0)$, is

1. $\alpha \leftarrow 1$
2. If

$$r(\alpha) < r(0) - 0.1\alpha r'(0) \tag{8.78}$$

   stop and return the current α value.
3. Let α_1 be the unique minimum of the convex quadratic function that passes through $r(0)$, $r'(0)$, and $r(\alpha)$. Take the new estimate of α as

$$\alpha \leftarrow \max(0.1\alpha, \alpha_1) \tag{8.79}$$

4. Go to step 2.

This backtracking line search tries $\alpha = 1.0$ first and accepts it if the "sufficient decrease" criterion (8.78) is met. This criterion is also used in unconstrained minimization, as discussed in Section 6.3.2. If $\alpha = 1.0$ fails the test (8.78), a safeguarded quadratic fit (8.79) chooses the next α. The trust region in (8.79) ensures that the new α is not too small.

SQP algorithm

Based on this line search and the QP subproblem QP $(\mathbf{x}, \mathbf{B})$ in (8.74)–(8.75), a typical SQP algorithm follows:

1. Initialize: $\mathbf{B}^0 \leftarrow \mathbf{I}$ (or some other positive-definite matrix), $\mathbf{x}^0 \leftarrow \mathbf{x}$ (user-provided initial point), $k \leftarrow 0$.
2. Solve the QP subproblem QP $(\mathbf{x}^k, \mathbf{B}^k)$, yielding a solution $\Delta\, \mathbf{x}^k$ and Lagrange multiplier estimates $\boldsymbol{\lambda}^k$.
3. Check the termination criteria (KTC, fractional objective change), and stop if any are satisfied to within the specified tolerances.
4. Update the penalty weights w in the penalty function $p(\mathbf{x}, w)$. See Nash and Sofer (1996) for details. Let the new weights be w^k.
5. Apply the line search algorithm just described to the function

$$r_k(\alpha) = P(\mathbf{x}^k + \alpha \Delta \mathbf{x}^k, w^k)$$

   yielding a positive step size α^k
6. $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \Delta \mathbf{x}^k, \boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k$
7. Evaluate all problem functions and their gradients at the new point. Update the matrix $\mathbf{B}^k$ (Nash and Sofer, 1996) using

$$L(\mathbf{x}^k), \quad L(\mathbf{x}^{k+1}), \quad \nabla_x L(\mathbf{x}^k, \boldsymbol{\lambda}^k), \quad \nabla_x L(\mathbf{x}^{k+1}, \boldsymbol{\lambda}^k)$$

8. Replace k by $k + 1$, and go to step 2

Convergence of SQP

Because of the quasi-Newton updating of $\mathbf{B}^k$, this SQP algorithm estimates second-order information, that is, $\mathbf{B}^k$ is a positive-definite approximation of $\nabla_x^2 L$. Hence a correctly implemented SQP algorithm can have a superlinear convergence rate, just as the BFGS algorithm for unconstrained minimization is superlinearly convergent. If the optimum is not at a vertex, SQP usually requires fewer iterations than SLP, but each iteration requires solution of a QP, which is often much slower than solving an LP (as SLP does). Hence each iteration takes longer than the corresponding SLP iteration. In addition, the approximate Hessian matrix $\mathbf{B}^k$ is *dense*, even when the matrix it approximates, $\nabla_x^2 L$, is sparse, so the algorithm gets slower and requires more storage (order of n^2) as the number of variables n increases. For problems with $n > 1000$, say, the SQP algorithm posed here is not practical. However, similar methods using sparse approximations to $\nabla_x^2 L$ do exist, and these can solve much larger problems.

SQP code performance

Table 8.4 shows the convergence of an SQP algorithm very similar to the one described here, applied to the Griffith–Stewart test problem of Section 8.5, using the

TABLE 8.4
Convergence of SQP on modified Griffith–Stewart problem

Iteration	Objective	Sum of infeasibilities
0	12.0000	4.000
1	11.2069	0.172
2	11.1810	0.015
3	11.1831	0.012
4	11.1803	2.1E-06
OPT	11.1803	0.000

objective $x + 2y$. This is the same problem as solved by PSLP in Table 8.3, using the same initial point (2, 5). Comparing the two tables shows that SQP converges much more rapidly on this problem than PSLP. This is because of the second-order information (second derivatives) estimated in the matrices $\mathbf{B}^k$. The price one pays for this rapid convergence is the need to store and manipulate the dense matrices $\mathbf{B}^k$, and to solve a more difficult subproblem (a QP instead of an LP). For problems with several thousand constraints and variables, these disadvantages usually mean that SLP is preferred. In fact, SLP is widely used in the oil and chemical industries to solve large production planning models. See Baker and Lasdon (1985) for details.

8.7 THE GENERALIZED REDUCED GRADIENT METHOD

The *generalized reduced gradient* (GRG) algorithm was first developed in the late 1960s by Jean Abadie (Abadie and Carpentier, 1969) and has since been refined by several other researchers. In this section we discuss the fundamental concepts of GRG and describe the version of GRG that is implemented in GRG2, the most widely available nonlinear optimizer [Lasdon et al., 1978; Lasdon and Waren, 1978; Smith and Lasdon, 1992].

GRG algorithms use a basic descent algorithm described below for unconstrained problems. We state the steps here:

General descent algorithm

1. Compute the gradient of $f(\mathbf{x})$ at the current point $\mathbf{x}_c$, giving $\nabla f(\mathbf{x}_c)$.
2. If the current point $\mathbf{x}_c$ is close enough to being optimal, stop.
3. Compute a search direction $\mathbf{d}_c$ using the gradient $\nabla f(\mathbf{x}_c)$ and perhaps other information such as the previous search direction.
4. Determine how far to move along the current search direction $\mathbf{d}_c$, starting from the current point $\mathbf{x}_c$. This distance α_c is most often an approximation of the value of α that minimizes the objective function $f(\mathbf{x}_c + \alpha\mathbf{d}_c)$ and is used to determine the next point $\mathbf{x}_n = (\mathbf{x}_c + \alpha_c\mathbf{d}_c)$.
5. Replace the current point $\mathbf{x}_c$ by the next point $\mathbf{x}_n$, and return to step 1.

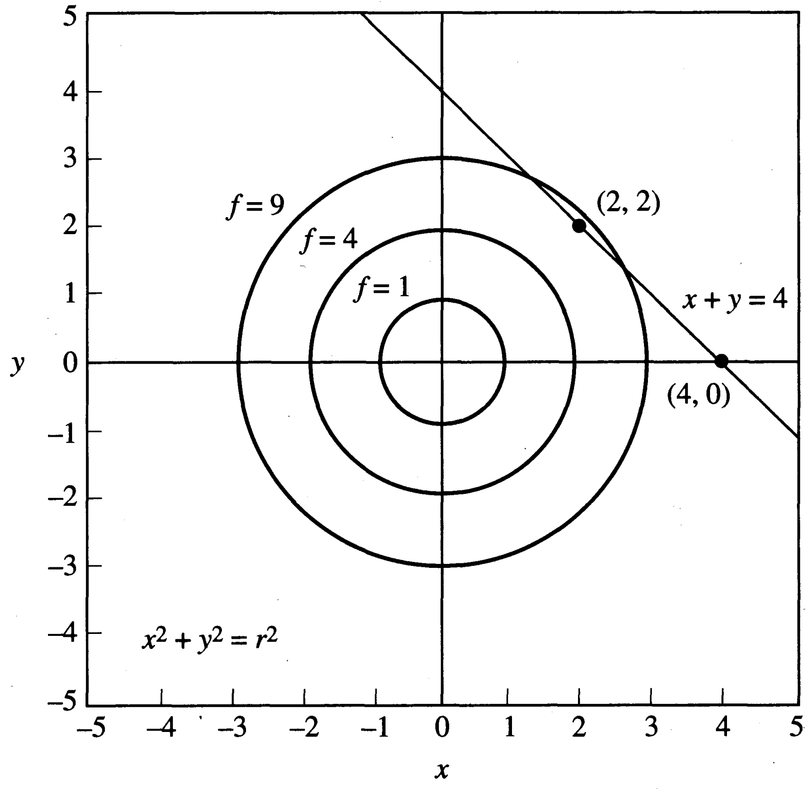

FIGURE 8.11
Circular objective contours and the linear equality constraint for the GRG example.

Equality constraints

To explain how GRG algorithms handle equality constraints, consider the following problem:

$$\text{Minimize:} \quad x^2 + y^2$$

$$\text{Subject to:} \quad x + y = 4$$

The geometry of this problem is shown in Figure 8.11. The linear equality constraint is a straight line, and the contours of constant objective function values are circles centered at the origin. From a geometric point of view, the problem is to find the point on the line that is closest to the origin at $x = 0, y = 0$. The solution to the problem is at $x = 2, y = 2$, where the objective function value is 8.

GRG takes a direct and natural approach to solve this problem. It uses the equality constraint to solve for one of the variables in terms of the other. For example, if we solve for x, the constraint becomes

$$x = 4 - y \tag{8.80}$$

Whenever a value is specified for y, the appropriate value for x, which keeps the equality constraint satisfied, can easily be calculated. We call y the independent, or nonbasic, variable and x the dependent, or basic, variable. Because x is now determined by y, this problem can be reduced to one involving only y by substituting $(4 - y)$ for x in the objective function to give:

$$F(y) = (4 - y)^2 + y^2$$

The function $F(y)$ is called the reduced objective function, and the reduced problem is to minimize $F(y)$ subject to no constraints. Once the optimal value of y is found, the optimal value of x is computed from Equation (8.80).

Because the reduced problem is unconstrained and quite simple, it can be solved either analytically or by the iterative descent algorithm described earlier. First, let us solve the problem analytically. We set the gradient of $F(y)$, called the *reduced gradient*, to zero giving:

$$\begin{aligned}\nabla F(y) &= \frac{dF(y)}{dy} = -2(4 - y) + 2y \\ &= -8 + 4y = 0\end{aligned}$$

Solving this equation we get $y = 2$. Substituting this value in (8.80) gives $x = 2$ and $(x, y) = (2, 2)$ is, of course, the same solution as the geometric one.

Now apply the steps of the descent algorithm to minimize $F(y)$ in the reduced problem, starting from an initial $y_c = 0$, for which the corresponding $x_c = 4$. Computing the reduced gradient gives $\nabla F(y_c) = \nabla F(0) = -8$, which is not close enough to zero to be judged optimal so we proceed with step 3. The initial search direction is the negative reduced gradient direction, so $d = 8$ and we proceed to the line search of step 4. New points are given by

$$\begin{aligned}y &= y_c + \alpha d \\ &= 0 + 8\alpha\end{aligned} \tag{8.81}$$

where α is the step size. We start at (4, 0) with $\alpha = 0$; as α increases, y also increases. This increase is determined by Equation (8.81) and keeps (x, y) on the equality constraint shown in Figure 8.11.

Next α is selected to minimize $g(\alpha)$, the reduced objective function evaluated along the current search direction, which is given by

$$\begin{aligned}g(\alpha) &= F(y_c + \alpha d) \\ &= F(0 + 8\alpha) \\ &= (4 - 8\alpha)^2 + (8\alpha)^2\end{aligned}$$

Again, in this simple case, we can proceed analytically to determine α by setting the derivative of $g(\alpha)$ to zero to get

$$\begin{aligned}\frac{dg(\alpha)}{d\alpha} &= -16(4 - 8\alpha) + 128\alpha \\ &= -64 + 256\alpha = 0\end{aligned}$$

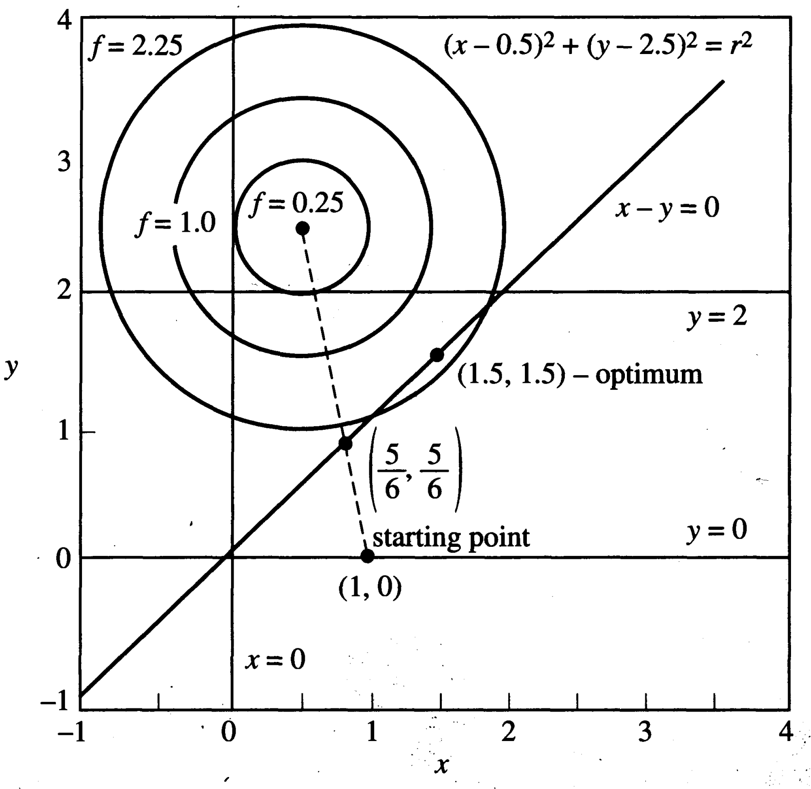

FIGURE 8.12
Circular objective function contours and linear inequality constraint.

Solving for α gives $\alpha = \frac{1}{4}$. Substituting this value into Equation (8.81) gives $y_n = 2$ and then (8.80) gives $x_n = 2$, which is the optimal solution.

Inequality constraints

Now examine how GRG proceeds when some of the constraints are inequalities and there are bounds on some or all of the variables. Consider the following problem:

$$\begin{aligned} \text{Minimize:}\quad & (x - 0.5)^2 + (y - 2.5)^2 \\ \text{Subject to:}\quad & x - y \geq 0 \\ & 0 \leq x \\ & 0 \leq y \leq 2 \end{aligned}$$

The feasible region and some contours of the objective function are shown in Figure 8.12. The goal is to find the feasible point that is closest to the point (0.5, 2.5), which is (1.5, 1.5).

GRG converts inequality constraints to equalities by introducing slack variables. If s is the slack in this case, the inequality $x - y \geq 0$ becomes $x - y - s = 0$. We must also add the bound for the slack, $s \geq 0$, giving the new problem:

$$\begin{aligned}
\text{Minimize:} \quad & (x - 0.5)^2 + (y - 2.5)^2 \\
\text{Subject to:} \quad & x - y - s = 0 \\
& 0 \leq x \\
& 0 \leq y \leq 2 \\
& 0 \leq s
\end{aligned}$$

Let the starting point be (1, 0), at which the objective value is 6.5 and the inequality is satisfied strictly, that is, its slack is positive ($s = 1$). At this point the bounds are also all satisfied, although y is at its lower bound. Because all of the constraints (except for bounds) are inactive at the starting point, there are no equalities that must be solved for values of dependent variables. Hence we proceed to minimize the objective subject only to the bounds on the nonbasic variables x and y. There are no basic variables. The reduced problem is simply the original problem ignoring the inequality constraint. In solving this reduced problem, we do keep track of the inequality. If it becomes active or violated, then the reduced problem changes.

To solve this first reduced problem, follow the steps of the descent algorithm outlined at the start of this section with some straightforward modifications that account for the bounds on x and y. When a nonbasic variable is at a bound, we must decide whether it should be allowed to leave the bound or be forced to remain at that bound for the next iteration. Those nonbasic variables that will not be kept at their bounds are called *superbasic variables* [this term was coined by Murtaugh and Saunders (1982)]. In step 1 the reduced gradient of $f(x,y)$ is

$$\begin{aligned}
\nabla F(x,y) &= \left[\frac{\partial F(x,y)}{\partial x} \; \frac{\partial F(x,y)}{\partial y} \right]^T \\
&= [2(x - 0.5) \; 2(y - 2.5)]^T \\
\nabla F(1,0) &= [1 \; -5]^T
\end{aligned}$$

In this example x is a superbasic variable. To decide whether y should also be a superbasic variable and be allowed to leave its bound, examine the value of its reduced gradient component. Because this value (-5) is negative, then moving y from its bound into the feasible region, that is, increasing the value of y, decreases the objective value. We therefore consider letting y leave its bound. In GRG, a nonbasic variable at a bound is allowed to leave that bound only if (1) doing so improves the value of the objective and (2) the predicted improvement is large compared with the improvement obtained by varying only the current superbasic variables. In this example, because the magnitude of the y component of the reduced gradient is five times the magnitude of the x component, the incentive to release y from its bound is large. Thus y is added to the list of superbasic variables.

In step 3 of the descent algorithm (because the gradient is clearly not small enough to stop), the first search direction is chosen as the negative gradient direction:

$$\mathbf{d}_c = -[1 \; -5] = [-1 \; 5]$$

In Figure 8.12, this direction (the dashed line) points to the center of the circular objective function contours at (0.5, 2.5). In step 4, the line search moves along $\mathbf{d}_c$

until either the objective stops decreasing or some constraint or variable bound is reached. In this example the condition that is first encountered is that the constraint $x - y \geq 0$ reaches its bound, and we then select the intersection of the search direction and the constraint $x - y = 0$ as the next point. This is the point $(\frac{5}{6}, \frac{5}{6})$ where $F = \frac{26}{9} = 2.889$.

Because we now have reached an active constraint, use it to solve for one variable in terms of the other, as in the earlier equality constrained example. Let x be the basic, or dependent, variable, and **y** and **s** the nonbasic (independent) ones. Solving the constraint for x in terms of y and the slack s yields

$$x = y + s$$

The reduced objective is obtained by substituting this relation for x in the objective function:

$$F(y,s) = (y + s - 0.5)^2 + (y - 2.5)^2$$

The reduced gradient is

$$\begin{aligned}\nabla F(y,s) &= 2[(y + s - 0.5) + (y - 2.5) \qquad (y + s - 0.5)]^T \\ &= [4y + 2s - 6 \qquad 2y + 2s - 1]^T\end{aligned}$$

which evaluated at $(\frac{5}{6}, 0)$ is

$$\nabla F(\tfrac{5}{6},0) = [-\tfrac{8}{3} \quad \tfrac{2}{3}]^T$$

The variable y becomes superbasic. Because s is at its lower bound of zero, consider whether s should be allowed to leave its bound, that is, be a superbasic variable. Because its reduced gradient term is $\frac{2}{3}$, increasing s (which is the only feasible change for s) increases the objective value. Because we are minimizing F, fix s at zero; this corresponds to staying on the line $x = y$. The search direction $d = \frac{8}{3}$ and new values for y are generated from

$$y = \tfrac{5}{6} + \tfrac{8}{3}\alpha$$

where α is the step size from the current point. The function to be minimized by the line search is

$$\begin{aligned}g(\alpha) &= F(y,s) = F(\tfrac{5}{6} + \tfrac{8}{3}\alpha, 0) \\ &= [\tfrac{1}{3} + \tfrac{8}{3}\alpha]^2 + [\tfrac{-5}{3} + \tfrac{8}{3}\alpha]^2\end{aligned}$$

The optimal step size of $\alpha = \frac{1}{4}$ is determined by setting $dg(\alpha)/d\alpha = 0$, which gives the next point as $y_n = 1.5$. Because s has been fixed at zero, we are on the line $x = y$ and at step 5 we have $(x_c, y_c) = (1.5, 1.5)$, which is the optimal value for this problem. To confirm this, return to step 1 of our descent algorithm, and calculate the reduced gradient of $F(y,s)$ at $(1.5, 0)$ to get

$$\nabla F(y,s) = \nabla F(1.5,0) = [0 \quad 1]$$

First, the first element in the reduced gradient with respect to the superbasic variable y is zero. Second, because the reduced gradient (the derivative with respect to s) is 1, increasing s (the only feasible change to s) causes an increase in the objective value. These are the two necessary conditions for optimality for this reduced problem and the algorithm terminates at (1.5, 1.5) with an objective value of 2.0.

Nonlinear constraints

To illustrate how GRG handles nonlinear constraints, replace the linear constraint of the previous example by

$$(x-2)^2 + y^2 \leq 4$$

The new problem is

$$\begin{aligned} \text{Minimize:} \quad & (x-0.5)^2 + (y-2.5)^2 \\ \text{Subject to:} \quad & (x-2)^2 + y^2 \leq 4 \\ & 0 \leq x \\ & 0 \leq y \leq 2 \end{aligned}$$

The feasible region is shown in Figure 8.13. It is bounded by a semicircle of radius 2 centered at (2, 0) and by the x axis. The point in this region closest to (0.5, 2.5) is optimal, which is (0.971, 1.715).

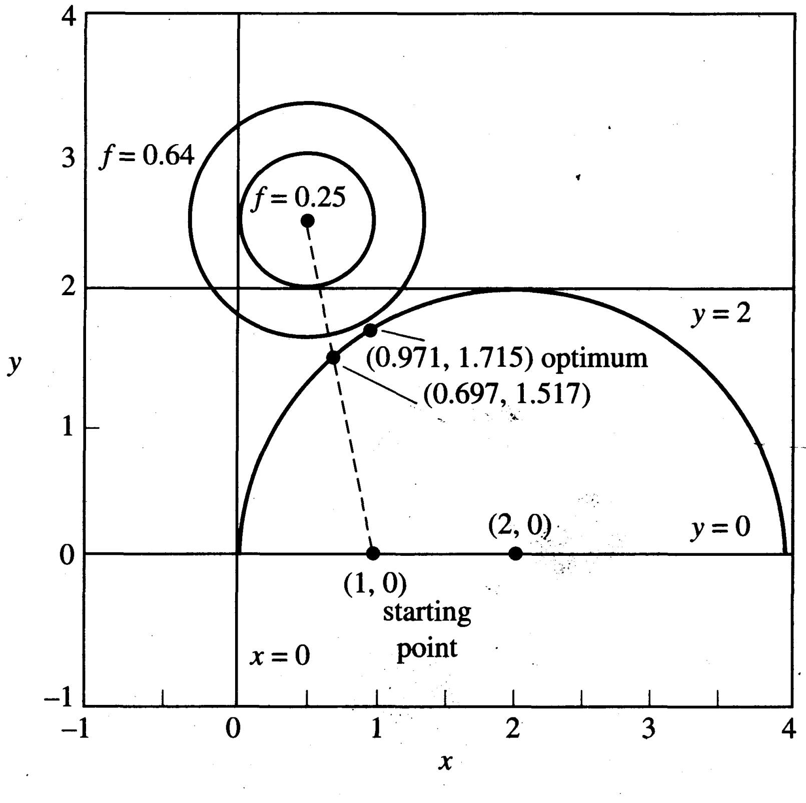

FIGURE 8.13
Circular objective function contours with a nonlinear inequality constraint.

We again start from the point (1, 0). At this point the nonlinear constraint is inactive, y is released from its lower bound to become superbasic (along with x) and progress continues along the negative gradient direction until the constraint is encountered. The intersection of the constraint and the negative gradient direction from the starting point is at (0.697, 1.715). Now the nonlinear constraint is active. Adding a slack variable s gives

$$(x - 2)^2 + y^2 + s = 4 \tag{8.82}$$

To form the reduced problem, this equation must be solved for one variable in terms of the other two. The logic in GRG for selecting which variables are to be basic is complex and is not discussed here [see Lasdon et al. (1978); and Lasdon and Waren (1978) for more information]. In this example GRG selects x as basic.

Solving (8.82) for x yields

$$x = 2 + \sqrt{4 - y^2 - s}$$

The reduced objective is obtained by substituting this expression into the objective function. The slack s will be fixed at its current zero value for the next iteration because moving into the interior of the circle from (0.697, 1.517) increases the objective. Thus, as in the linearly constrained example, y is again the only superbasic variable at this stage.

Because analytic solution of the active constraints for the basic variables is rarely possible, especially when some of the constraints are nonlinear, a numerical procedure must be used. GRG uses a variation of Newton's method which, in this example, works as follows. With $s = 0$, the equation to be solved for x is

$$(x - 2)^2 + y^2 - 4 = 0 \tag{8.83}$$

GRG determines a new value for y as before, by choosing a search direction d and then a step size α. Because this is the first iteration for the current reduced problem, the direction d is the negative reduced gradient. The line search subroutine in GRG chooses an initial value for α. At (0.697, 1.517), $d = 1.508$ and the initial value for α is 0.050. Thus the first new value for y, say y_1, is

$$y_1 = y_c + \alpha d = 1.517 + 0.050(1.508) = 1.592$$

Substituting this value into Equation (8.83) gives

$$g(x) = (x - 2)^2 - 1.466 = 0 \tag{8.84}$$

Given an initial guess x_0 for x, Newton's method is used to solve Equation (8.84) for x by replacing the left-hand side of (8.84) by its first-order Taylor series approximation at x_0:

$$g(x_0) + \left(\frac{\partial g(x_0)}{\partial x}\right)(x - x_0) = 0$$

Solving this equation for x and calling this result x_1 yields

$$x_1 = x_0 - \left(\frac{\partial g(x_0)}{\partial x}\right)^{-1} g(x_0) \tag{8.85}$$

If $g(x_1)$ is close enough to zero, x_1 is accepted as the solution and this procedure stops. "Close enough" is determined by a feasibility tolerance E_f (which can be set by the user, and has a default value of 0.0001) using the criterion:

$$\text{abs}[g(x_1)] \leq E_f \tag{8.86}$$

If this criterion is not satisfied, x_1 replaces x_0, and a new iteration of Newton's method begins. For this example, the sequence of x and y values generated by GRG is

Iteration	x	$g(x)$
Initial point	0.7849	−0.134E-01
1	0.7900	−0.940E-03
2	0.7904	−0.675E-04

In the "pure" Newton's method, $\partial g(x)/\partial x$ is reevaluated at each new value of x. In GRG, $\partial g(x)/\partial x$ is evaluated only once for each line search, at the point from which the line search begins. In this example, $\partial g(x)/\partial x$ evaluated at $x = 0.697$ is 2.606, so the GRG formula corresponding to (8.85) is

$$x_1 = x_0 - 0.383g(x_0)$$

This variation on Newton's method usually requires more iterations than the pure version, but it takes much less work per iteration, especially when there are two or more basic variables. In the multivariable case the matrix $\nabla \mathbf{g}(\mathbf{x})$ (called the basis matrix, as in linear programming) replaces $\partial \mathbf{g}/\partial \mathbf{x}$ in the Newton equation (8.85), and $\mathbf{g}(\mathbf{x}_0)$ is the vector of active constraint values at $\mathbf{x}_0$.

Note that the initial guess for x in row 1 of the preceding table is 0.7849, not its base value of 0.697. GRG derives this initial estimate by using the vector that is tangent to the nonlinear constraint at (0.697, 1.517), as shown in Figure 8.14. Given $y_1 = 1.592$, the x value on this tangent vector is 0.7849. The tangent vector value is used because it usually provides a good initial guess and results in fewer Newton iterations.

Of course, Newton's method does not always converge. GRG assumes Newton's method has failed if more than ITLIM iterations occur before the Newton termination criterion (8.86) is met or if the norm of the error in the active constraints ever increases from its previous value (an occurrence indicating that Newton's method is diverging). ITLIM has a default value of 10. If Newton's method fails but an improved point has been found, the line search is terminated and a new GRG iteration begins. Otherwise the step size in the line search is reduced and GRG tries again. The output from GRG that shows the progress of the line search at iteration 4 is

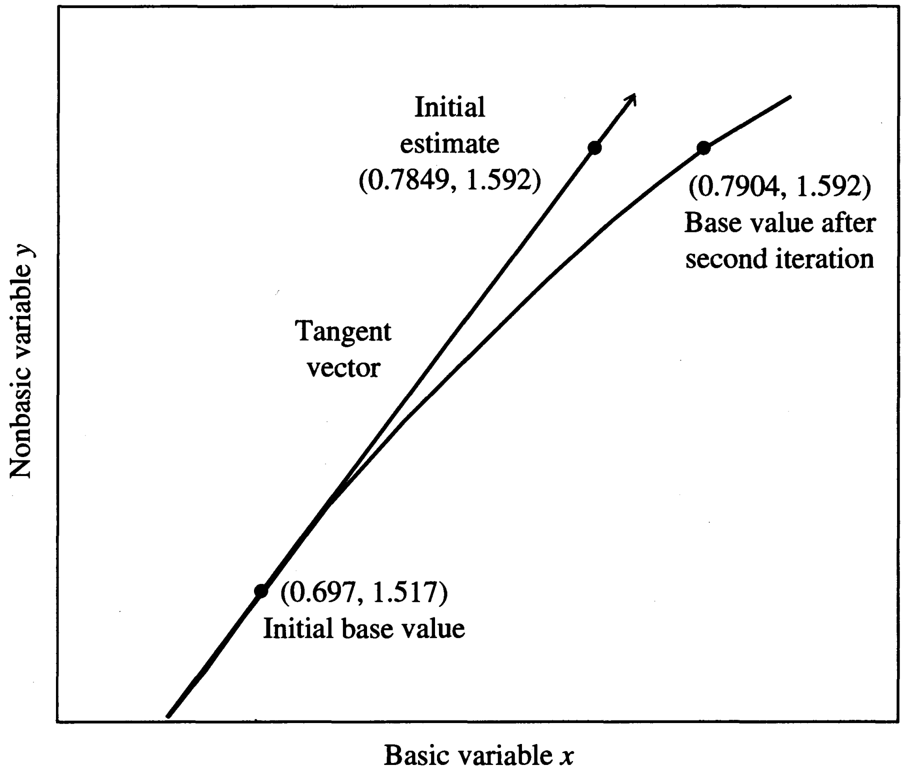
Initial estimate (0.7849, 1.592)
(0.7904, 1.592) Base value after second iteration
Tangent vector
(0.697, 1.517) Initial base value
Nonbasic variable y
Basic variable x

FIGURE 8.14
Initial estimate for Newton's method to return to the nonlinear constraint.

```
STEP = 5.028E-02 OBJ = 9.073E-01 NEWTON ITERS = 2
STEP = 1.005E-01 OBJ = 8.491E-01 NEWTON ITERS = 4
STEP = 2.011E-01 OBJ = 9.128E-01 NEWTON ITERS = 8
QUADRATIC INTERPOLATION
STEP = 1.242E-01 OBJ = 8.386E-01 NEWTON ITERS = 4
```

Note that as the line search process continues and the total step from the initial point gets larger, the number of Newton iterations generally increases. This increase occurs because the linear approximation to the active constraints, at the initial point (0.697, 1.517), becomes less and less accurate as we move further from that point.

Infeasible starting point

If the initial values of the variables do not satisfy all of the constraints, GRG starts with a phase I objective function (as is also done in linear programming) and attempts to find a feasible solution. To illustrate this approach consider a problem that has no objective function and has the following three constraints:

$$x^2 + y^2 \leq 4$$

$$x + y \geq 1$$

$$x - y = 0$$

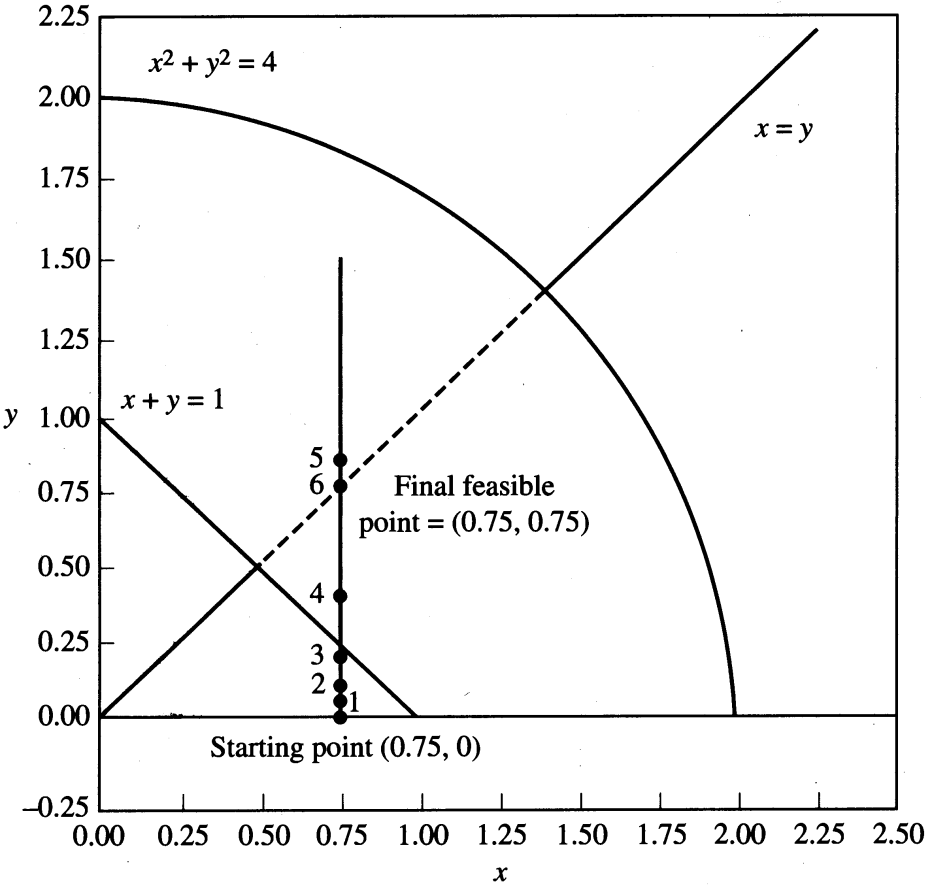

FIGURE 8.15
Finding a feasible point in GRG; the feasible region is the dashed line.

We use a starting point of (0.75, 0). The feasible region is shown in Figure 8.15 as the dashed line segment. At the initial point constraint 1 is strictly satisfied, but constraints 2 and 3 are violated. GRG constructs the phase I objective function as the sum of the absolute values of all constraint violations. For this case the sum of the infeasibilities (*sinf*) is

$$\begin{aligned} sinf(x,y) &= (x - y) + [1 - (x + y)] \\ &= 0.75 + 0.25 \\ &= 1.0 \end{aligned}$$

The first term is the violation of constraint 2 and the second term is the violation of constraint 3. Note that both terms are arranged so that the violations are positive.

The optimization problem solved by GRG is

$$\begin{aligned} \text{Minimize:} \quad & sinf(x,y) \\ \text{Subject to:} \quad & x^2 + y^2 \le 4 \end{aligned}$$

At the initial point, the preceding nonlinear constraint is inactive, the reduced objective is just *sinf*(*x*, *y*), and the reduced gradient is

$$\nabla sinf(x, y) = [0 \quad -2]$$

The initial search direction is, as usual, the negative reduced gradient direction so $\mathbf{d} = [0 \quad 2]$ and we move from (0.75, 0) straight up toward the line $x + y = 1$. The output from GRG is shown in the following box.

Iteration number	Objective function	Number binding	Number superbasics	Number infeasible	Norm of reduced gradient	Hessian condition

```
   0       1.000E+00      1          2            2       2.000E+00   1.000E+00
       STEP = 2.500000E-02                   OBJ = 9.00000E-01
       STEP = 5.000000E-02                   OBJ = 8.00000E-01
       STEP = 1.000000E-01                   OBJ = 6.00000E-01
       STEP = 2.000000E-01                   OBJ = 3.50000E-01
       STEP = 4.000000E-01                   OBJ = 0.00000E+00
CONSTRAINT #3 VIOLATED BOUND
ALL VIOLATED CONSTRAINTS SATISFIED. NOW BEGIN TO OPTIMIZE TRUE OBJECTIVE
```

Iteration number	Objective function	Number binding	Number superbasics	Number infeasible	Norm of reduced gradient	Hessian condition

```
   0       1.000E+00      1          2            0       2.000E+00   1.000E+00

KUHN-TUCKER CONDITIONS SATISFIED
```

As can be seen in the output shown in the box, at the starting point (iteration 0) there are two infeasible constraints, two superbasics, and $sinf = 1$. Using the usual formula, (*x*, *y*) for the first line search is calculated as follows:

$$\begin{aligned}(x,y) &= (x_c,y_c) + \alpha(d_1,d_2)\\ &= (x_c + \alpha d_1, y_c + \alpha d_2)\\ &= (0.75 + 0\alpha, 0 + 2\alpha)\\ &= (0.75, 2\alpha)\end{aligned}$$

It is clear that the *x* values remain fixed at 0.75 and the *y* values are twice the step size at each step. In Figure 8.15 these steps are labeled 1 through 6. At step 5, GRG detects the change in sign of constraint number 3 and backs up until the constraint is binding. Because at this stage (*x*,*y*) is feasible, GRG prints the message

```
ALL VIOLATED CONSTRAINTS SATISFIED
```

If the problem had an objective function, GRG would begin minimizing the "true" objective, starting from this feasible point. Because we did not specify an objective for this problem, the algorithm stops. Minimizing *sinf* to find a feasible point, if needed, is phase I of the GRG algorithm; optimization of the true objective is phase II. If GRG cannot find a feasible solution, then phase I will terminate with a positive value of *sinf* and report that no feasible solution was found.

8.8 RELATIVE ADVANTAGES AND DISADVANTAGES OF NLP METHODS

Table 8.5 summarizes the relative merits of SLP, SQP, and GRG algorithms, focusing on their application to problems with many nonlinear equality constraints. One feature appears as both an advantage and a disadvantage—whether or not the algorithm can violate the nonlinear constraints of the problem by relatively large amounts during the solution process.

SLP and SQP usually generate points with large violations. This can cause difficulties, especially in models with log or fractional power expressions, because negative arguments for these functions may be generated. Such problems have been documented in reference to complex chemical process examples (Sarma and

TABLE 8.5
Relative merits of SLP, SQP, and GRG algorithms

Algorithm	Relative advantages	Relative disadvantages
SLP	Widely used in practice Rapid convergence when optimum is at a vertex Can handle very large problems Does not attempt to satisfy equalities at each iteration Can benefit from improvements to LP solvers	May converge slowly on problems with nonvertex optima Will usually violate nonlinear constraints until convergence to optimum, often by large amounts
SQP	Usually requires the fewest function and gradient evaluations of all three algorithms (by far) Does not attempt to satisfy equalities at each iteration	Will usually violate nonlinear constraints until convergence, often by large amounts
GRG	Probably most robust of all three methods Versatile—especially good for unconstrained or linearly constrained problems but also works well for nonlinear constraints Once it reaches a feasible solution it remains feasible and then can be stopped at any stage with an improved solution	Needs to satisfy equalities at each step of the algorithm

Reklaitis, 1982) in which SLP and some exterior penalty-type algorithms failed, but the GRG code succeeded and was quite efficient. On the other hand, algorithms that do not attempt to satisfy the equalities at each step can be faster than those that do (Berna et al., 1980). The fact that SLP and SQP satisfy any linear constraints at each iteration should ease the difficulties cited in Table 8.5 but does not eliminate them.

In some situations the optimization process must be terminated before the algorithm has reached optimality and the current point must be used or discarded. These cases usually arise in on-line process control in which time limits force timely decisions. In such cases, maintaining feasibility during the optimization process may be a requirement for the optimizer because an intermediate infeasible point makes a solution unusable.

Clearly, all three algorithms have advantages that dictate their use in certain situations. For large problems, SLP software is used most widely, because it is relatively easy to implement given a good LP code. Large-scale versions of GRG and SQP are increasingly employed, however.

8.9 AVAILABLE NLP SOFTWARE

In this section we survey implementations of the algorithms described in Sections 8.5 through 8.7. Although an increasingly large proportion of NLP users employ systems with higher level user interfaces to optimizers, such as spreadsheets and algebraic modeling systems, all such systems have at their core adaptations of one or more "stand-alone" optimization packages. By stand-alone we mean software designed specifically to accept the specification of a nonlinear program, attempt to solve it, and return the results of that attempt to the user or to an invoking application. The NLP capabilities and characteristics of those higher level systems therefore naturally derive from those of their incorporated optimizers. As a result, we begin our discussion with an overview of significant stand-alone NLP optimizers. We also illustrate, for a simple NLP problem, the inputs and outputs of some of the optimizers described later on. A comprehensive list of vendors and sources for the products discussed in this section (as well as for a large number of linear, unconstrained, and discrete optimization products) is found in Moré and Wright (1993) and Wright (2000). Advertisements for many of the systems described here can be found in the monthly magazine *OR/MS Today,* published by INFORMS (Institute for Operations Research and the Management Sciences). This magazine is an excellent source of information on analytical software of all kinds. The June 1998 issue contains an excellent NLP software survey (Nash, 1998).

8.9.1 Optimizers for Stand-Alone Operation or Embedded Applications

Most existing NLP optimizers are FORTRAN-based, although C versions are becoming more prevalent. Most are capable of operation as true stand-alone systems (the user must usually code or modify main programs and routines that return function values) or as subsystems that are embedded in larger systems and solve problems generated by or posed through those systems. Some vendors supply source code, and others supply only object code for the customers' target platform. Details are available from the vendors as noted later on or in Moré and Wright (1993) and Wright (2000). All NLP optimizers require that the user supply the following:

- A specification of the NLP problem to be solved—at a minimum, the number of functions, the number of variables, which function is the optimization objective, bounds on the functions and variables (if different from some default scheme), and initial values of some or all variables (the system may supply default values, but using these is recommended only as a last resort).
- One or more subprograms that supply to the optimizer, on demand, the values of the functions for a specified set of variable values. Some systems also allow the user the option of supplying derivative values.

GRG-based optimizers

GRG2. This code is presently the most widely distributed for the generalized reduced gradient and its operation is explained in Section 8.7. In addition to its use as a stand-alone system, it is the optimizer employed by the "Solver" optimization options within the spreadsheet programs Microsoft Excel, Novell's Quattro Pro, Lotus 1-2-3, and the GINO interactive solver.

In stand-alone operation, GRG2 requires the user to code a calling program in FORTRAN or C that allocates working storage and passes through its argument list the problem specifications and any nondefault values for user-modified options (an option using text files for problem specifications also exists). In addition, the user must code a subroutine that accepts as input a vector of variable values and returns a vector of function values calculated from the inputs. All constraints are assumed to be of the form

$$l_i \leq g_i(\mathbf{x}) \leq u_i$$

where l_i and u_i are (constant) lower and upper bounds.

GRG2 represents the problem Jacobian (i.e., the matrix of first partial derivatives) as a dense matrix. As a result, the effective limit on the size of problems that can be solved by GRG2 is a few hundred active constraints (excluding variable bounds). Beyond this size, the overhead associated with inversion and other linear algebra operations begins to severely degrade performance. References for descriptions of the GRG2 implementation are in Liebman et al. (1985) and Lasdon et al. (1978).

LSGRG2. This extension of GRG2 employs sparse matrix representations and manipulations and extends the practical size limit to at least 1000 variables and constraints. The interfaces to LSGRG2 are very close to those described earlier for GRG2. LSGRG2 has been interfaced to the GAMS algebraic-modeling system. Performance tests and comparisons on several large models from the GAMS library are described by Smith and Lasdon (1992).

CONOPT. This is another widely used implementation of the GRG algorithm. Like LSGRG2, it is designed to solve large, sparse problems. CONOPT is available as a stand-alone system, callable subsystem, or as one of the optimizers callable by the GAMS systems. Description of the implementation and performance of CONOPT is given by Drud (1994).

SQP-based optimizers

Implementations of the SQP algorithm described in Section 8.6 are

SQP. This is a sister code to GRG2 and available from the same source. The interfaces to SQP are very similar to those of GRG2. SQP is useful for small problems as well as large sparse ones, employing sparse matrix structures throughout. The implementation and performance of SQP are documented in Fan, et al. (1988).

NPSOL. This is a dense matrix SQP code developed at Stanford University. It is available from the same source as MINOS (see following description of MINOS). Additional details are available in Moré and Wright (1993).

NLPQL. This is another SQP implementation, callable as a subroutine and notable for its use of reverse communication. The called subsystem returns codes to the calling program, indicating what information is required on reentry. (Moré and Wright, 1993).

MINOS. This employs a modified augmented Lagrangian algorithm described in Murtagh and Saunders (1982). MINOS uses sparse matrix representations throughout and is capable of solving nonlinear problems exceeding 1000 variables and rows. MINOS is also capable of exploiting, to the greatest extent possible, the presence of purely linear variables and functions. Because the user must communicate this structure to the optimizer, the greatest utility of this feature results from coupling MINOS to higher level modeling systems that can determine problem structure. As a stand-alone system, problem specifications and user options are supplied to MINOS via an external text file, and problem Jacobian information is supplied through another file. As with the other optimizers described here, the user must supply FORTRAN routines that compute function values and, optionally, derivatives. MINOS is the default optimizer option under the GAMS system for both linear and nonlinear problems. Details for stand-alone use of MINOS and additional references are given in Murtagh and Saunders (1982).

Mathematical software libraries

Many of the major callable libraries of mathematical software include at least one general NLP component (i.e., capable of solving problems with nonlinear constraints). IMSL provides individual callable routines for most variations of linear and nonlinear constraints and objectives. The NAG FORTRAN Library (also available as a toolbox of MATLAB) contains an SQP method for constrained problems and a variety of routines for unconstrained or specialized optimization problems. In addition, most such libraries, even those without specific constrained NLP solvers, contain routines that perform such tasks as equation solving, unconstrained optimization, and various linear algebra operations. These routines can be used as subalgorithm components to build customized NLP solvers. References for the IMSL and NAG libraries and their vendors may be found in Moré and Wright (1993).

8.9.2 Spreadsheet Optimizers

In the 1980s, a major move away from FORTRAN and C optimization began as optimizers, first LP solvers, and then NLP solvers were interfaced to spreadsheet systems for desktop computers. The spreadsheet has become, de facto, the universal user interface for entering and manipulating numeric data. Spreadsheet vendors are increasingly incorporating analytic tools accessible from the spreadsheet interface and able, through that interface, to access external databases. Examples include statistical packages, optimizers, and equation solvers.

The Excel Solver. Microsoft Excel, beginning with version 3.0 in 1991, incorporates an NLP solver that operates on the values and formulas of a spreadsheet model. Versions 4.0 and later include an LP solver and mixed-integer programming (MIP) capability for both linear and nonlinear problems. The user specifies a set of cell addresses to be independently adjusted (the decision variables), a set of formula cells whose values are to be constrained (the constraints), and a formula cell designated as the optimization objective. The solver uses the spreadsheet interpreter to evaluate the constraint and objective functions, and approximates derivatives, using finite differences. The NLP solution engine for the Excel Solver is GRG2 (see Section 8.7).

For examples that use the Excel Solver, see Chapters 7, 9, and 10. For a description of the design and use of the Excel Solver, see Fylstra, et al. (1998). An enhanced version of the Excel Solver, which can handle larger problems, is faster, and includes enhanced solvers is available from Frontline Systems—see www.frontsys.com. This website contains a wealth of information on spreadsheet optimization.

The Quattro Pro Solver. The same team that packaged and developed the Excel Solver also interfaced the same NLP engine (GRG2) to the Quattro Pro spreadsheet. Solver operation and problem specification mechanisms are similar to those for Excel.

LOTUS 123. The LOTUS 123 WINDOWS-based products incorporate linear and nonlinear solvers that operate in a fashion similar to those described earlier and use the same solver engines.

8.9.3 Algebraic Modeling Systems

An algebraic modeling system normally accepts the specification of a model in text as a system of algebraic equations. The system parses the equations and generates a representation of the expressions that can be numerically evaluated by its interpreter. In addition, some analysis is done to determine the structure of the model and to generate expressions for evaluating the Jacobian matrix. The processed model is then available for presentation to an equation solver or optimizer. The following paragraphs describe four algebraic modeling systems with NLP capabilities.

GAMS—General algebraic modeling system

The general algebraic modeling system (GAMS) allows specification and solution of large-scale optimization problems. The modeling language is algebraic with a FORTRAN-like style. The default NLP solver for GAMS is MINOS with ZOOM-XMP available for mixed-integer programming. Optional interfaces are available for most currently available LP, NLP, and MILP solvers. GAMS is available on a wide variety of platforms ranging from PCs to workstations and mainframes. Examples of GAMS models and solution output are given in Chapter 9. General references, system details, and user procedures are given in Brooke and coworkers (1992). See www.gams.com for more information.

AMPL

The main features of a mathematical programming language (AMPL) include an interactive environment for setting up and solving mathematical programs; the ability to select among several solvers; and a powerful set construct that allows for indexed, named, and nested sets. This set construct allows large-scale optimization problems to be stated tersely and in a form close to their natural algebraic expression. AMPL is described in Fourer et al. (1993). A WINDOWS version, AMPL PLUS, is available, with a graphical user interface (GUI) that greatly enhances productivity.

MPL and AIMMS

Both MPL and the advanced interactive multidimensional modeling software (AIMMS) are algebraic modeling languages operating under Microsoft Windows, with convenient GUIs; powerful modeling languages; and excellent connections to external files, spreadsheets, databases, and a wide variety of linear and nonlinear solvers. See www.maximal-usa.com for MPL, and www.aimms.com for AIMMS.

8.10 USING NLP SOFTWARE

This section addresses some of the problems with NLP optimization software. The primary determinant of solution reliability with LP solvers is numerical stability and accuracy. If the linear algebra subsystem of an LP solver is strong in these areas, the solver will almost always terminate with one of three conditions—optimal, infeasible, unbounded—or will run up against a time or iteration limit set by the user prior to detecting one of those conditions. In contrast, many additional factors affect NLP solvers and their ability to obtain and recognize a solution.

8.10.1 Evaluation of Derivatives: Issues and Problems

All major NLP algorithms require estimation of first derivatives of the problem functions to obtain a solution and to evaluate the optimality conditions. If the values of the derivatives are computed inaccurately, the algorithm may progress very slowly, choose poor directions for movement, and terminate due to lack of progress or reaching the iteration limits at points far from the actual optimum, or, in extreme cases, actually declare optimality at nonoptimal points.

Finite difference substitutes for derivatives

When the user, whether working on stand-alone software or through a spreadsheet, supplies only the values of the problem functions at a proposed point, the NLP code computes the first partial derivatives by finite differences. Each function is evaluated at a base point and then at a perturbed point. The difference between the function values is then divided by the perturbation distance to obtain an approximation of the first derivative at the base point. If the perturbation is in the positive direction from the base point, we call the resulting approximation a forward difference approximation. For highly nonlinear functions, accuracy in the values of derivatives may be improved by using central differences; here, the base point is perturbed both forward and backward, and the derivative approximation is formed from the difference of the function values at those points. The price for this increased accuracy is that central differences require twice as many function evaluations of forward differences. If the functions are inexpensive to evaluate, the additional effort may be modest, but for large problems with complex functions, the use of central differences may dramatically increase solution times. Most NLP codes possess options that enable the user to specify the use of central differences. Some codes attempt to assess derivative accuracy as the solution progresses and switch to central differences automatically if the switch seems warranted.

A critical factor in the accuracy of finite difference approximations for derivatives is the value of the perturbation step. The default values employed by all NLP codes (generally 1.E-6 to 1.E-7 times the value of the variable) yield good accuracy when the problem functions can be evaluated to full machine precision. When problem functions cannot be evaluated to this accuracy (perhaps due to functions that are the result of iterative computations), the default step is often too small. The resulting derivative approximations then contain significant error. If the function(s) are highly nonlinear in the neighborhood of the base point, the default perturbation step may be too large to accurately approximate the tangent to the function at that point. Special care must be taken in derivative computation if the problem functions are not closed-form functions in compiled code or a modeling language (or, equivalently, a sequence of simple computations in a spreadsheet). If each function evaluation involves convergence of a simulation, solution of simultaneous equations, or convergence of an empirical model, the interaction between the derivative perturbation step and the convergence criteria of the functions strongly affects the derivative accuracy, solution progress, and reliability. In such cases, increasing the perturbation step by two or three orders of magnitude may aid the solution process.

Analytic derivatives

Algebraic modeling systems, such as those described in Section 8.9.3, accept user-provided expressions for the objective and constraint functions and process them to produce additional expressions for the analytic first partial derivatives of these functions with respect to all decision variables. These expressions are exact, so the derivatives are evaluated to full machine precision (about 15 correct decimal digits using double precision arithmetic), and they are used by any derivative-based nonlinear code that is interfaced to the system. Finite-difference approximations to first derivatives have at most seven or eight significant digits. Hence, an NLP code used within an algebraic modeling system can be expected to produce more accurate results in fewer iterations than the same solver using finite-difference derivatives. Chemical process simulators like Aspen also compute analytic derivatives and provide these to their nonlinear optimizers. Spreadsheet solvers currently use finite-difference approximations to derivatives.

Of course, many models in chemical and other engineering disciplines are difficult to express in a modeling language, because these are usually coded in FORTRAN or C (referred to as "general purpose" programming languages), as are many existing "legacy" models, which were developed before modeling systems became widely used. General-purpose languages offer great flexibility, and models coded in these languages generally execute about ten times faster than those in an algebraic modeling system because FORTRAN and C are compiled, whereas statements in algebraic modeling systems are interpreted. This additional speed is especially important in on-line control applications (see Chapter 16).

Derivatives in FORTRAN or C models may be approximated by differencing, or expressions for the derivatives can be derived by hand and coded in subroutines used by a solver. Anyone who has tried to write expressions for first derivatives of many complex functions of many variables knows how error-prone and tedious this process is. These shortcomings motivated the development of computer programs for *automatic differentiation (AD)*. Given FORTRAN or C source code which evaluates the functions, plus the user's specification of which variables in the program are independent, AD software augments the given program with additional statements that compute partial derivatives of all functions with respect to all independent variables. In other words, using AD along with FORTRAN or C produces a program that computes the functions and their first derivatives.

Currently, the most widely used AD codes are ADIFOR (automatic differentiation of FORTRAN) and ADIC (automatic differentiation of C). These are available at no charge from the Mathematics and Computer Science division of Argonne National Laboratories—see www.mcs.anl.gov for information on downloading the software and further information on AD. This software has been successfully applied to several difficult problems in aeronautical and structural design as well as chemical process modeling.

8.10.2 What to Do When an NLP Algorithm Is Not "Working"

Probably the most common mode of failure of NLP algorithms is termination due to "fractional change" (i.e., when the difference in successive objective function values is a small fraction of the value itself over a set of consecutive iterations) at a point where the Kuhn–Tucker optimality conditions are far from satisfied. Sometimes this criterion is not considered, so the algorithm terminates due to an iteration limit. Termination at a significantly nonoptimal point is an indication that the algorithm is unable to make any further progress. Such lack of progress is often associated with poor derivative accuracy, which can lead to search directions that do not improve the objective function. In such cases, the user should analyze the problem functions and perhaps experiment with different derivative steps or different starting points.

Parameter adjustment

Most NLP solvers use a set of default tolerances and parameters that control the algorithm's determination of which values are "nonzero," when constraints are satisfied, when optimality conditions are met, and other tuning factors.

Feasibility and optimality tolerances

Most NLP solvers evaluate the first-order optimality conditions and declare optimality when a feasible solution meets these conditions to within a specified tolerance. Problems that reach what appear to be optimal solutions in a practical sense but require many additional iterations to actually declare optimality may be sped up by increasing the optimality or feasibility tolerances. See Equations (8.31a) and (8.31b) for definitions of these tolerances. Conversely, problems that terminate at points near optimality may often reach improved solutions by decreasing the optimality or feasibility tolerances if derivative accuracy is high enough.

Other "tuning" issues

The feasibility tolerance is a critical parameter for GRG algorithms because it represents the convergence tolerance for the Newton iterations (see Section 8.7 for details of the GRG algorithm). Increasing this tolerance from its default value may speed convergence of slow problems, whereas decreasing it may yield a more accurate solution (at some sacrifice of speed) or "unstick" a sequence of iterations that are going nowhere. MINOS requires specification of a parameter that penalizes constraint violations. Penalty parameter values affect the balance between seeking feasibility and improving of the objective function.

Scaling

The performance of most NLP algorithms (particularly on large problems) is greatly influenced by the relative scale of the variables, function values, and Jacobian elements. In general, NLP problems in which the absolute values of these quantities lie within a few orders of magnitude of each other (say in the range 0–100) tend to solve (if solutions exist) faster and with fewer numerical difficulties. Most codes either scale problems by default or allow the user to specify that the problem be scaled. Users can take advantage of these scaling procedures by building models that are reasonably scaled in the beginning.

Model formulation

Users can enhance the reliability of any NLP solver by considering the following simple model formulation issues:

- Avoid constructs that may result in discontinuities or undefined function arguments. Use exponential functions rather than logs. Avoid denominator terms that may tend toward zero (i.e., $1/x$ or $1/(x-1)$, etc.), multiplying out these denominators where possible.
- Be sensitive to possible "domain violations," that is, the potential for the optimizer to move variables to values for which the functions are not defined (negative log arguments, negative square roots, negative bases for fractional exponents) or for which the functions that make up the model are not valid expressions of the systems being modeled.

Starting points

The performance of NLP solvers is strongly influenced by the point from which the solution process is started. Points such as the origin (0, 0, . . .) should be avoided because there may be a number of zero derivatives at that point (as well as problems with infinite values). In general, any point where a substantial number of zero derivatives are possible is undesirable, as is any point where tiny denominator values are possible. Finally, for models of physical processes, the user should avoid starting points that do not represent realistic operating conditions. Such points may cause the solver to move toward points that are stationary points but unacceptable configurations of the physical system.

Local and global optima

As was discussed in Section 4.3, a global optimum is a feasible solution that has the best objective value. A local optimum has an objective value that is better than that of any "nearby" feasible solution. All NLP algorithms and solvers here are only capable of finding local optima. For convex programs, any local optimum is also global. Unfortunately, many NLPs are not convex or cannot be guaranteed to be convex, hence we must consider any solution returned by an NLP solver to be local. The user should examine the solution for reasonableness, perhaps re-solving the problem from several starting points to investigate what local optima exist and how these solutions differ from one another. He/she can also try a global optimizer; see Chapter 10.

REFERENCES

Abadie, J.; and J. Carpentier. "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints." In *Optimization,* R. Fletcher, ed. Academic Press, New York (1969), pp. 37–47.

Baker, T. E.; and L. Lasdon. "Successive Linear Programming at Exxon." *Manage Sci* **31:** (3), 264–274 (1985).

Berna, T. J.; M. H. Locke; and A. W. Westerberg. "A New Approach to Optimization of Chemical Processes." *AIChE J* **26:** 37–43 (1980).

Brooke, A.; et al. *GAMS: A User's Guide*. Boyd and Fraser, Danvers, MA (1992).

Brown, G. G.; R. F. Dell; and R. K. Wood. "Optimization and Persistence." *Interfaces* **27** (5): 15–37 (1997).

DiBella, C. W.; and W. F. Stevens. "Process Optimization by Nonlinear Programming." *I & E C Process Des Dev* **4:** 16–20 (1965).

Drud, A. "CONOPT—A Large-Scale GRG-Code." *ORSA J Comput* **6** (2): Spring (1994).

Fan, Y.; S. Sarkar; and L. Lasdon. "Experiments with Successive Quadratic Programming Algorithms." *J Optim Theory Appli* **56:** (3), 359–383 (March 1988).

Fourer, R.; D. M. Gay; and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Programming, San Francisco (1993).

Fylstra, D.; L. Lasdon; J. Waton.; and A. Waven. "Design and Use of the Microsoft Excel Solver." *Interfaces* **28:** (5), 29–55 (September–October, 1998).

Klein, M.; and R. R. Kimpel. "Application of Linearly Constrained Nonlinear Optimization to Plant to Location and Sizing." *J Indus Engin* **18:** 90 (1967).

Lasdon, L., et al. "Design and Testing of a Generalized Relaxed Gradient Code for Nonlinear Programming." *ACM Trans Math Soft* **4:** (1) 34–50 (1978).

Lasdon, L. S.; and A. D. Waren. "Generalized Reduced Gradient Software for Linearly and Nonlinearly Constrained Problems." *Design and Implementation of Optimization Software*, H. J. Greenberg, ed., Sijthoff and Noordhoff, Holland (1978), pp. 363–397.

Liebman, J. F., et al. *Modeling and Optimization with GINO*. Boyd and Fraser, Danvers, MA (1986).

Luenberger, D. G. *Linear and Nonlinear Programming*, 2nd ed. Addison-Wesley, Menlo Park, CA (1984).

Luus, R.; and T. Jaakola. "Optimization of Nonlinear Function Subject to Equality Constraints." *Chem Process Des Develop* **12:** 380–383 (1973).

Moré, J. J.; and S. J. Wright. *Frontiers in Applied Mathematics: Optimization Software Guide*. SIAM, Philadelphia, PA (1993).

Murtagh, B. A.; and M. A. Saunders. "A Projected Lagrangian Algorithm and Its Implementation for Sparse Nonlinear Constraints." *Math Prog Study* **16:** 84–117 (1982).

Nash, S. G. "Nonlinear Programming." *OR/MS Today*, 36–45 (June 1998).

Nash, S. G.; and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, New York (1996).

Nocedal, J.; and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research, New York (1999).

Rustem, B. *Algorithms for Nonlinear Programming and Multiple Objective Functions.* Wiley, New York (1998).

Sarma, P. V. L. N.; and G. V. Reklaitis. "Optimization of a Complex Chemical Process Using an Equation Oriented Model." *Math Prog Study* **20:** 113–160 (1982).

Smith, S.; and L. Lasdon. "Solving Large Sparse Nonlinear Programs Using GRG." *ORSA J Comput* **4:** (1) 3–15 (Winter 1992).

Williams, T. J.; and R. E. Otto. "A Generalized Chemical Processing Model for the Investigation of Computer Control." *A I E E Trans* **79** (Communications and Electronics) 458–473 (1960).

Wright, S. J. "Algorithms and Software for Linear and Nonlinear Programming. FOCAPD Proceedings, *AIChE Symp Ser* **96:** 323, 58–69 (2000).

Zhang, J.; N. Kim; and L. Lasdon. "An Improved Successive Linear Programming Algorithm." *Manage Sci* **31:** 1312–1331 (1985).

SUPPLEMENTARY REFERENCES

Bhatia, T. K.; and L. T. Biegler. "Multiperiod Design and Planning with Interior Point Methods." *Comp Chem Engin* **23:** 919 (1999).

de Gouvêa, M. T.; and D. Odloak. "A New Treatment of Inconsistent Quadratic Programs in an SQP-Based Algorithm." *Comp Chem Engin* **22:** 1623 (1998).

Dennis, J. E. Jr.; and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia, PA (1996).

Duvall, P. M.; and J. B. Riggs. "On-line Optimization of the Tennessee Eastman Challenge Problem." *J Proc Control* **10:** 19 (1999).

Rustem, B. *Algorithms for Nonlinear Programming and Multiple Objective Decisions.* Wiley, New York (1998).

Ternet, D. J.; and L. T. Biegler. "Recent Improvements to a Multiplier-free Reduced Hessian Successive Quadratic Programming Algorithm." *Comp Chem Engin* **22:** 963 (1998).

Turton, R.; R. C. Bailie; W. B. Whiting; and J. Shaewitz. *Analysis, Synthesis, and Design of Chemical Processes*. Prentice-Hall, Upper Saddle River, NJ (1998).

Vassiliadis, V. S.; and S. A. Brooks. "Application of the Modified Barrier Method in Large-Scale Quadratic Programming Problems." *Comp Chem Engin* **22:** 1197–1205 (1998).

PROBLEMS

8.1 Solve

$$\text{Minimize:} \quad -x_1$$
$$\text{Subject to:} \quad x_1 + x_2^4 = 0$$

by solving the constraint for x_1 and substituting into the objective function. Do you get $\mathbf{x}^* = [0 \quad 0]^T$?

8.2 Solve

$$\text{Minimize:} \quad -x_1^2$$
$$\text{Subject to:} \quad 10^{-5}x_2^2 + x_1 = 1$$

by solving the constraint for x_1 and substituting into the objective function. Do you get $\mathbf{x}^* = [1 \quad 0]^T$?

8.3 Explain in *no more* than three sentences how the nonlinear inequality constraints in a nonlinear programming problem can be converted into equality constraints. Demonstrate for $g(\mathbf{x}) = x_1x_2 + x_2^2 + e^{x_3} \leq 4$.

8.4 Use the method of Lagrange multipliers to solve the following problem. Find the values of x_1, x_2, and ω that

$$\text{Minimize:} \quad f(\mathbf{x}) = x_1^2 + x_2^2$$
$$\text{Subject to:} \quad h(\mathbf{x}) = 2x_1 + x_2 - 2 = 0$$

8.5 Solve the following problem via the Lagrange multiplier method:
Find the maximum and minimum distances from the origin to the curve

$$5x_1^2 + 6x_1x_2 + 5x_2^2 = 8$$

Hint: The distance $\sqrt{x_1^2 + x_2^2}$ is the objective function.

8.6 Show that Lagrange multipliers do not exist for the following problem:

$$\text{Minimize:} \quad f(\mathbf{x}) = x_1^2 + x_2^2$$
$$\text{Subject to:} \quad (x_1 - 1)^3 - x_2^2 = 0$$

8.7 Examine the reactor in Figure P8.7. The objective function, $f(c, T) = (c - c_r)^2 + T^2$ is subject to the constraint $c = c_0 + e^T$ and also $c_0 < K$, where c_r is the set point for the outlet concentration, a constant, and K is a constant.

Find the minimum value of the objective function using Lagrange multipliers for the case in which $K = c_r - 2$.

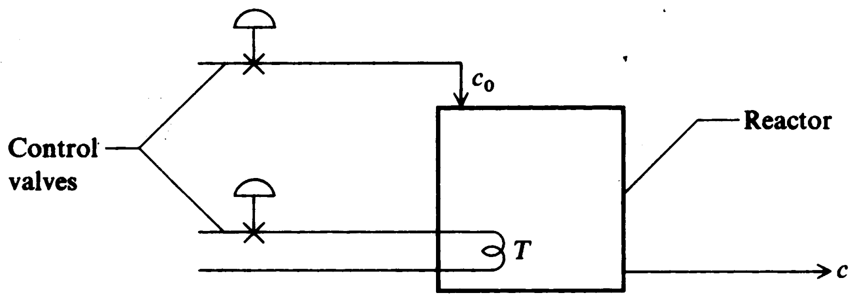

FIGURE P8.7

8.8 Examine the continuous through-circulation dryer problem posed by Luus and Jaakola (1973):

Maximize: P given by

$$P = 0.0064x_1[1 - \exp(-0.184x_1^{0.3}x_2)]$$

Subject to: the power constraint

$$(3000 + x_1)x_1^2x_2 = 1.2 \times 10^{13}$$

and the moisture content distribution constraint

$$\exp(0.184x_1^{0.3}x_2) = 4.1$$

They obtained the solution $\mathbf{x}^* = [31{,}766 \quad 0.342]$ $P = 153.71$. Does this problem satisfy the first-order conditions?

Repeat for the problem of minimizing the capital investment for batch processes. The problem is to choose x_1, x_2, and x_3 to minimize

$$P = 592V^{0.65} + 582V^{0.39} + 1200V^{0.52} + 370\left(\frac{V}{x_1}\right)^{0.22}$$
$$+ 250\left(\frac{V}{x_2}\right)^{0.40} + 210\left(\frac{V}{x_2}\right)^{0.62} + 250\left(\frac{V}{x_3}\right)^{0.40}$$
$$+ 200\left(\frac{V}{x_3}\right)^{0.85}$$

subject to the simple constraint

$$V = 50(10 + x_1 + x_2 + x_3)$$

They obtained the solution $P = 126{,}302.9$ and

$$\mathbf{x}^* = [0.11114 \quad 1.46175 \quad 3.42476]$$

8.9 Maximize: $f = x_1^2 + x_2^2 + 4x_1x_2$

Subject to: $x_1 + x_2 = 8$

(a) Form the Lagrangian L. Set up the necessary conditions for a maximum, and solve for the optimum.
(b) If the constraint is changed to $x_1 + x_2 = 8.01$, compute f and L without resolving as in part a.

8.10 (a) Minimize $f = x_1^2 + x_2^2 + 10x_1 + 20x_2 + 25$

Subject to: $x_1 + x_2 = 0$

using the Lagrange multiplier technique. Calculate the optimum values of x_1, x_2, λ, and f.
(b) Using sensitivity analysis, determine the increase in f^{opt} when the constraint is changed to $x_1 + x_2 = 0.01$.
(c) Let the constraint be added to f by a penalty function:

$$P = f + r(x_1 + x_2)^2$$

Find the optimum of P with respect to x_1 and x_2 (an unconstrained problem), noting that x_1^* and x_2^* are functions of r.
(d) Is there a relationship between r, x_1^*, x_2^*, and λ^*?
(e) Perform the second derivative test on P; is it convex for $P \gg 1$?

8.11 Is the problem

$$\text{Minimize:} \quad f(\mathbf{x}) = x_1^3 + 4x_2^2 - 4x_1$$

$$\text{Subject to:} \quad 2x_2 - x_1 \geq 12$$

a convex programming problem?

8.12 Determine whether the vector $x^T = [0 \quad 0]$ is an optimal solution of the problem

$$\text{Minimize:} \quad f(\mathbf{x}) = (x_1 - 1)^2 + x_2^2$$

$$\text{Subject to:} \quad h(\mathbf{x}) = x_1^2 + x_2^2 + x_1 + x_2 = 0$$

$$g(\mathbf{x}) = -x_1 + x_2^2 \geq 0$$

8.13 Determine whether the point $x = [0 \quad 0 \quad 0]^T$ is a local minimum of the problem:

$$\text{Minimize:} \quad f(\mathbf{x}) = \tfrac{4}{3}(x_1^2 - x_1x_2)^{3/4} + x_3$$

$$\text{Subject to:} \quad x_1^2 + x_2^2 + x_3^2 = 0$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

$$x_3 \geq 0$$

Show all computations.

8.14 Test whether the solution $\mathbf{x}^* = [2 \quad 2]^T$ meets the sufficient conditions for a minimum of the following problem.

$$\text{Minimize:} \quad f(\mathbf{x}) = -x_1^2x_2$$

$$\text{Subject to:} \quad h_1(\mathbf{x}) = x_1x_2 + \left(\frac{x_1^2}{2}\right) = 6$$

$$g_2(\mathbf{x}) = x_1 - x_2 \geq 0$$

8.15 Do (a) the necessary and (b) the sufficient conditions hold at the optimum $\mathbf{x}^* = [0.82 \quad 0.91]^T$ for the following problem?

$$\text{Minimize:} \quad f(\mathbf{x}) = (x_1 - 2)^2 + (x_2 - 1)^2$$

$$\text{Subject to:} \quad g_1(\mathbf{x}) = -\frac{x_1^2}{4} - x_2^2 + 1 \geq 0$$

$$h_2(\mathbf{x}) = x_1 - 2x_2 + 1 = 0$$

Note: $\mathbf{x}^* = \left[\left(-1 + \sqrt{7}\right)/2 \quad \left(1 + \sqrt{7}\right)/4\right]^T$ exactly.

8.16 Does the following solution $\mathbf{x}^* = \begin{bmatrix} \frac{1}{3} & \frac{5}{3} \end{bmatrix}^T$ meet the sufficient conditions for a minimum of the following problem?

$$\begin{aligned} \text{Minimize:} \quad & f(\mathbf{x}) = -\ln(1 + x_1) - \ln(1 + x_2)^2 \\ \text{Subject to:} \quad & g_1(\mathbf{x}) = x_1 + x_2 - 2 \le 0 \\ & g_2(\mathbf{x}) = x_1 \ge 0 \\ & g_3(\mathbf{x}) = x_2 \ge 0 \end{aligned}$$

8.17 Solve the following problems via a quadratic programming code.

*P*8.4 *P*8.9 *P*8.20 *P*8.22a

8.18 Find the stationary point of the function $f(\mathbf{x}) = x_1^2 + x_2^2 + 4x_1x_2$ subject to the constraint $x_1 + x_2 = 8$. Use direct substitution. What kind of stationary point is it?

For the same objective function and constraint, form a new function

$$P = x_1^2 + x_2^2 + 4x_1x_2 + r(x_1 + x_2 - 8)^2$$

where r is a large number. Then optimize P.

(a) Find the stationary point of P with respect to x_1 and x_2, solving for x_1^* and x_2^* in terms of r.
(b) Find x_1^*, x_2^* as $r \to \infty$
(c) Does $P^* \to f^*$ for $r \to \infty$?

8.19 Minimize: $x_2^2 - x_1^2$

Subject to: $x_1^2 + x_2^2 = 4$.

(a) Use Lagrange multipliers
(b) Use a penalty function.

8.20 The problem is to

$$\begin{aligned} \text{Minimize:} \quad & f(\mathbf{x}) = x_1^2 + 6x_1 + x_2^2 + 9 \\ \text{Subject to:} \quad & g_i(\mathbf{x}) = x_i \ge 0, \quad \text{for } i = 1, 2 \end{aligned}$$

From the starting vector $\mathbf{x}^0 = [1 \quad 0.5]^T$.

(a) Formulate a penalty function suitable to use for an unconstrained optimization algorithm.
(b) Is the penalty function convex?

8.21 A statement in a textbook is

> The penalty term of an augmented Lagrangian method is designed to add positive curvature so that the Hessian of the augmented function is positive-definite.

Is this statement correct?

8.22 Formulate the following problems as

(a) Penalty function problems
(b) Augmented Lagrangian problems

(1)
$$\text{Minimize: } f(\mathbf{x}) = 2x_1^2 - 2x_1x_2 + 2x_2^2 - 6x_1 + 6$$
$$\text{Subject to: } h(\mathbf{x}) = x_1 + x_2 - 2 = 0$$

(2)
$$\text{Minimize: } f(\mathbf{x}) = x_1^3 - 3x_1x_2 + 4$$
$$\text{Subject to: } g(\mathbf{x}) = 5x_1 + 2x_2 \geq 18$$
$$h(\mathbf{x}) = -2x_1 + x_2^2 - 5 = 0$$

8.23 Comment on the following proposed penalty functions suggested for use with the problem.

$$\text{Minimize: } f(\mathbf{x})$$
$$\text{Subject to: } g_i(\mathbf{x}) \geq 0, \quad i = 1,2, \ldots, m$$

starting from a feasible point. The P functions are

(a)
$$P(\mathbf{x},\mathbf{x}^k) = \frac{1}{f(\mathbf{x}^k) - f(\mathbf{x})} + \sum_{j=1}^{m} \frac{1}{g_j(\mathbf{x})}$$

(b)
$$P(\mathbf{x},r) = f(\mathbf{x}) - r\sum_{j=1}^{m} \ln g_j(\mathbf{x})$$

(c)
$$P(\mathbf{x},r^k) = f(\mathbf{x}) + r^k\sum_{j=1}^{m} \frac{1}{g_j(\mathbf{x})}$$

What advantages might they have compared with one another? What disadvantages?

8.24 The problem of optimizing production from several plants with different cost structures and distributing the products to several distribution centers is common in the chemical industry. Newer plants often yield lower cost products because we learn from the mistakes made in designing the original plant. Due to plant expansions, rather unusual cost curves can result. The key cost factor is the incremental variable cost, which gives the cost per pound of an additional pound of product. Ordinarily, this variable cost is a function of production level.

Consider three different plants producing a product called DAB. The Frag plant located in Europe has an original design capacity of 100×10^6 lb/year but has been expanded to produce as high as 170×10^6 lb/year. The incremental variable cost for this plant decreases slightly up to 120×10^6 lb/year, but for higher production rates severe reaction conditions cause the yields to deteriorate, causing a gradual increase in the variable cost, as shown by the following equation. No significant byproducts are sold from this plant. Using VC = variable cost is \$/100 lb and x = production level $\times$ 10^{-6} lb/year

$$VC = 4.5 - (x - 100)(0.005), \quad 100 \le x \le 120$$

$$VC = 4.4 + (x - 120)(0.02), \quad 120 \le x \le 170$$

The Swung-Lo plant, located in the Far East, is a relatively new plant with an improved reactor/recycle design. This plant can be operated between 80×10^6 and 120×10^6 lb/year and has a constant variable cost of \$5.00/100 lb.

The Hogshooter plant, located in the United States, has a range of operation from 120×10^6 to 200×10^6 lb/year. The variable cost structure is rather complicated due to the effects of extreme reaction conditions, separation tower limitation, and several byproducts, which are affected by environmental considerations. These considerations cause a discontinuity in the incremental variable cost curve at 140×10^6 lb/year as given by the following equations:

$$VC = 3.9 + (x - 120)(0.005), \quad 120 \le x \le 140$$

$$VC = 4.6 + (x - 140)(0.01), \quad 140 \le x \le 200$$

The three main customers for the DAB are located in the Europe ($C1$), the Far East ($C2$), and the United States ($C3$), respectively. The following matrix shows the transportation costs of (¢/lb) and total demand to the customers ($C1$, $C2$, $C3$) with plant locations denoted as $A1$ (Frag), $A2$ (Swung-Lo), and $A3$ (Hogshooter). The closest pairing geographically is $A1$-$C1$; $A2$-$C2$; and $A3$-$C3$.

	$A1$	$A2$	$A3$	**Total demand**
$C1$	0.2	0.7	0.6	140
$C2$	0.7	0.3	0.8	100
$C3$	0.6	0.8	0.2	170

Use an iterative method based on successive linearization of the objective function to determine the optimum distribution plan for the product, DAB. Use an LP code to minimize total cost at each iteration.

8.25 Maximize: $f(\mathbf{x}) = 0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$

Subject to:

$$1 - x_3^2 - x_4^2 \ge 0$$

$$1 - x_9^2 \ge 0$$

$$1 - x_5^2 - x_6^2 \ge 0$$

$$1 - x_1^2 - (x_2 - x_9)^2 \ge 0$$

$$1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \ge 0$$

$$1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \ge 0$$

$$1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \ge 0$$

$$1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \ge 0$$

$$1 - x_7^2 - (x_8 - x_9)^2 \geq 0$$

$$x_1x_4 - x_2x_3 \geq 0$$

$$x_3x_9 \geq 0$$

$$-x_5x_9 \geq 0$$

$$x_5x_8 - x_6x_7 \geq 0$$

$$x_9 \geq 0$$

Starting point: $x_0^i = 1, \quad i = 1,9$

Solve using an SQP code.

8.26 Solve the following over-constrained problem.

$$\begin{aligned}
\text{Minimize:} \quad & f(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 \\
\text{Subject to:} \quad & g_1(\mathbf{x}) = -2x_1 - x_2 \geq -5 \\
& g_2(\mathbf{x}) = -x_1 - x_3 \geq -2 \\
& g_3(\mathbf{x}) = -x_1 - 2x_2 - x_3 \geq -10 \\
& h_1(\mathbf{x}) = 2x_1 - 2x_2 + x_3 = -2 \\
& h_2(\mathbf{x}) = 10x_1 + 8x_2 - 14x_3 = 26 \\
& h_3(\mathbf{x}) = -4x_1 + 5x_2 - 6x_3 = 6 \\
& x_1 \geq 1 \quad x_2 \geq 2 \quad x_3 \geq 0
\end{aligned}$$

Starting point: $\mathbf{x}^0 = [1 \quad 1 \quad 1]^T$

Use successive quadratic programming.

8.27 Solve the following problems by the generalized reduced-gradient method. Also, count the number of function evaluations, gradient evaluations, constraint evaluations, and evaluations of the gradient of the constraints.

(a) Minimize: $f(\mathbf{x}) = -(x_1^2 + x_2^2 + x_3^2)$

Subject to: $x_1 + 2x_2 + 3x_3 - 1 = 0$

$$x_1^2 + \frac{x_2^2}{2} + \frac{x_3^2}{4} - 4 = 0$$

Use various starting points.
$\mathbf{x}^0 = [n \ n \ n]$, where $n = 2, 4, 6, 8, 10, -2, -4, -6, -8, -10$

(b) Minimize: $f(\mathbf{x}) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2$

$$+ (x_3 - x_4)^4 + (x_4 - x_5)^4$$

Subject to: $x_1 + x_2^2 + x_3^3 - 2 - 3\sqrt{2} = 0$

$x_2 - x_3^2 + x_4 + 2 - 2\sqrt{2} = 0$

$(x_1)(x_5) - 2 = 0$

$\mathbf{x}^0 = [n \ n \ n \ n \ n]^T$, where $n = 2, 4, 6, 8, 10, -2, -4, -6, -8, -10$

8.28 At stage $k = 2$, the generalized reduced-gradient method is to be applied to the following problem at the point $\mathbf{x} = [0 \ 1 \ 1]^T$.

Minimize: $f(\mathbf{x}) = 2x_1^2 + 2x_2^2 + x_3^2 - 2x_1x_2 - 4x_1 - 6x_2$

Subject to: $x_1 + x_2 + x_3 = 2$

$x_1^2 + 5x_2 = 5$

$x_1 \geq 0, \quad i = 1,2,3$

(a) Compute the component step direction (+ or −) and value of each of the three variables after searching in the selected direction.
(b) Reduce $f(\mathbf{x})$ in the search direction.

Explain (only) in detail how you would reach the feasible point to start the next stage ($k = 3$) of optimization.

8.29 Answer true or false:

(a) In the generalized reduced-gradient method of solving NLP problems, the nonlinear constraints and the objective function are repeatedly linearized.
(b) Successive quadratic programming is based on the application of Newton's method to some of the optimality conditions for the Lagrangian function of the problem, that is, the sum of the objective function and the product of the Lagrangian multipliers times the equality constraints.

8.30 Solve the following problems using
i. A generalized reduced-gradient code
ii. A successive quadratic programming code. Compare your results.

(a) Minimize: $f(\mathbf{x}) = \sum_{k=1}^{10} \left(\frac{1}{k} x_k^2 + k x_k + k^2 \right)^2$

Subject to: $h_1(\mathbf{x}) = x_1 + x_3 + x_5 + x_7 + x_9 = 0$

$h_2(\mathbf{x}) = x_2 + 2x_4 + 3x_6 + 4x_8 + 5x_{10} = 0$

$h_3(\mathbf{x}) = 2x_2 - 5x_5 + 8x_8 = 0$

$g_1(\mathbf{x}) = -x_1 + 3x_4 - 5x_7 + x_{10} \geq 0$

$$g_2(\mathbf{x}) = -x_1 - 2x_2 - 4x_4 - 8x_8 \geq -100$$

$$g_3(\mathbf{x}) = -x_1 - 3x_3 - 6x_6 + 9x_9 \geq -50$$

$$-10^3 \leq x_i \leq 10^3, \quad i = 1,2, \ldots, 10$$

Starting point (feasible): $x_i^0 = 0, \quad i = 1,2, \ldots, 10$

$$f(\mathbf{x}^0) = 25{,}333.0$$

(b) Minimize: $f(\mathbf{x}) = \sum_{i=1}^{11} x_i + \sum_{i=1}^{10} (x_i + x_{i+1})$

Subject to: $x_i \geq 0, \quad i = 1, \ldots, 11$

$$h_1(\mathbf{x}) = 0.1x_1 + 0.2x_7 + 0.3x_8 + 0.2x_9 + 0.2x_{11} = 1.0$$

$$h_2(\mathbf{x}) = 0.1x_2 + 0.2x_8 + 0.3x_9 + 0.4x_{10} + 1.0x_{11} = 2.0$$

$$h_3(\mathbf{x}) = 0.1x_3 + 0.2x_8 + 0.3x_9 + 0.4x_{10} + 2.0x_{11} = 3.0$$

$$g_4(\mathbf{x}) = x_4 + x_8 + 0.5x_9 + 0.5x_{10} + 1.0x_{11} \geq 1.0$$

$$g_5(\mathbf{x}) = 2.0x_5 + x_6 + 0.5x_7 + 0.5x_8 + 0.25x_9 + 0.25x_{10} + 0.5x_{11} \geq 1.0$$

$$g_6(\mathbf{x}) = x_4 + x_6 + x_8 + x_9 + x_{10} + x_{11} \geq 1.0$$

$$g_7(\mathbf{x}) = 0.1x_1 + 1.2x_7 + 1.2x_8 + 1.4x_9 + 1.1x_{10} + 2.0x_{11} \geq 1.0$$

Starting point (feasible): $x_i = 1.0, \quad i = 1,2, \ldots, 11$

(c) Maximize: $f(\mathbf{x}) = 3x_1 e^{-0.1x_1x_6} + 4x_2 + x_3^2 + 7x_4 + \dfrac{10}{x_5} + x_6$

Subject to: $-x_4 + x_5 - x_6 = 0.1$

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 10$$

$$2x_1 + x_2 + x_3 + 3x_4 \geq 2$$

$$-8x_1 - 3x_2 - 4x_3 + x_4 - x_5 \geq -10$$

$$-2x_1 - 6x_2 - x_3 - 3x_4 - x_6 \geq -13$$

$$-x_1 - 4x_2 - 5x_3 - 2x_4 \geq -18$$

$$-20 \leq x_i \leq 20, \quad i = 1, \ldots, 6$$

Starting point (nonfeasible): $x_i = 1.0, \quad i = 1, \ldots, 6$

(d) Minimize: $f(\mathbf{x}) = x_1^2 + 2x_2^2 + 3x_3^2 + 4x_4^2 + 5x_5^2$

Subject to: $h_1(\mathbf{x}) = 2x_1 + x_2 - 4x_3 + x_4 - x_5 = 0$

$h_2(\mathbf{x}) = 5x_1 - 2x_3 + x_4 - x_5 = 0$

$g_1(\mathbf{x}) = x_1 + 2x_2 + x_3 \geq 6$

$g_2(\mathbf{x}) = 4x_3 + x_4 - 2x_5 \leq 0$

Starting point (nonfeasible): $x_i = 1, \quad i = 1, \ldots, 5$

(e) Minimize: $f(\mathbf{x}) = (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4$

Subject to: $x_1 + 2x_2 + 3x_3 - 6 = 0$

$x_2 + 2x_3 + 3x_4 - 6 = 0$

$x_3 + 2x_4 + 3x_5 - 6 = 0$

Starting point (feasible): $\mathbf{x}^0 = [35 \quad -31 \quad 11 \quad 5 \quad -5]^T$

(f) Minimize: $f(\mathbf{x}) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$

Subject to: $x_1^2 x_4 + \sin(x_4 - x_5) - 2\sqrt{2} = 0$

$x_2 + x_3^4 x_4^2 - 8 - \sqrt{2} = 0$

Starting point: $\mathbf{x}^0 = [2 \quad 2 \quad 2 \quad 2 \quad 2]^T$

(g) Minimize: $f(\mathbf{x}) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4$

Subject to: $x_1 + x_2^2 + x_3^3 - 2 - 3\sqrt{2} = 0$

$x_2 - x_3^2 + x_4 + 2 - 2\sqrt{2} = 0$

$x_1 x_5 - 2 = 0$

Starting points: $x_1 = \pm 10, x_2 = \pm 8, x_3 = \pm 6, x_4 = \pm 4, x_5 = \pm 2$

8.31 Explain in *no more* than three sentences how an initially feasible starting point can be obtained in solving a nonlinear programming problem. Demonstrate on the problem

$$\text{Maximize:} \quad f(\mathbf{x}) = \left[(1 + x_1)^2 + x_2^2\right]^{-1}$$

Subject to: $g_1(\mathbf{x}) = 4 - x_1^2 - x_2^2 \leq 0$

$g_2(\mathbf{x}) = x_1^2 + x_2^2 - 16 \leq 0$

$h_1(\mathbf{x}) = x_1 - x_2 = 3$

8.32

Minimize: $-x_1$

Subject to: $\exp(x_1^4 x_2) - 1 = 0$

$\exp(x_1^4 + x_2^4 - 1) = 1$

Starting point (nonfeasible): $\mathbf{x}^0 = [2.0, 1.0]^T$

Do you get both solutions?

$$\mathbf{x}^* = [\pm 1.0, 0.0]^T \quad \text{and} \quad [0.0, \pm 1.0]^T$$

8.33

Minimize: $-x_1$

Subject to: $x_1^5 + 3x_1^4x_2^2 + 3x_1^2x_2^4 + x_2^6 - 4x_1^4 + 8x_1^2x_2^2 - 4x_2^4 = 0$

Starting point (nonfeasible): $\mathbf{x}^0 = [-2.0, 2.0]^T$

Do you get all three solutions?

$$\mathbf{x}^* = [2.0, 0.0]^T, \quad [0.0, 0.0]^T, \quad [0.0, 2.0]^T$$

8.34 The cost of constructing a distillation column can be written

$$C = C_pAN + C_sHAN + C_f + C_d + C_b + C_L + C_x \tag{a}$$

where C = Total cost, \$
C_p = cost per square foot of plate area, \$/ft^2
A = column cross-sectional area, ft^2
N = number of plates
$N_{\min}$ = minimum number of plates
C_s = cost of shell, \$/ft^3
H = distance between plates, ft
C_f = cost of feed pump, \$
C_d = cost of distillate pump, \$
C_b = cost of bottoms pump, \$
C_L = cost of reflux pump, \$
C_x = other fixed costs, \$

The problem is to minimize the total cost, once produce specifications and the throughput are fixed and the product and feed pumping costs are fixed; that is, C_f, C_d, C_L, and C_b are fixed. After selection of the material of construction, the costs are determined; that is, C_p, C_s, C_x are also fixed.

The process variables can be related through two empirical equations:

$$\frac{L}{D} = \left[\frac{1}{1 - (N_{\min}/N)}\right]^x \left(\frac{L}{D}\right)_{\min} \tag{b}$$

$$A = K(L + D)^\beta \tag{c}$$

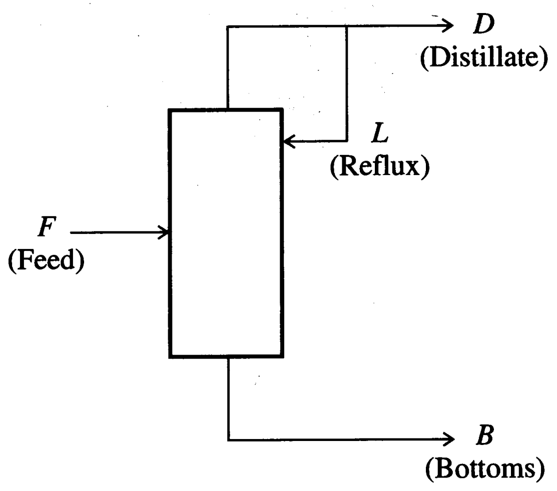

FIGURE P8.34

For simplicity choose $\alpha = \beta = 1$; then

$$\frac{L}{D} = \left[\frac{1}{1 - (N_{\min}/N)}\right]\left(\frac{L}{D}\right)_{\min} \tag{b'}$$

$$A = K(L + D) \tag{c'}$$

For a certain separation and distillation column the following parameters are known to apply:

$C_p = 30$	$C_x = 8000$
$C_s = 10$	$F = 1500$
$H = 2$	$D = 1000$
$C_f = 4000$	$N_{\min} = 5$
$C_d = 3000$	$\left(\frac{L}{D}\right)_{\min} = 1$
$C_b = 2000$	$K = \frac{1}{100}\frac{(\text{h})(\text{ft}^2)}{\text{lb}}$

The pump cost for the reflux stream can be expressed as

$$C_L = 5000 + 0.7L \tag{d}$$

(a) Determine the process decision or independent variables. Which variables are dependent?
(b) Find the minimum total cost and corresponding values of the variables.

8.35 A chemical manufacturing company sells three products and has found that its revenue function is $f = 10x + 4.4y^2 + 2z$, where x, y, and z are the monthly production rates of each chemical. It is found from breakeven charts that it is necessary to impose the following limits on the production rates:

$$x \geq 2$$

$$\tfrac{1}{2}z^2 + y^2 \geq 3$$

In addition, only a limited amount of raw material is available; hence the following restrictions must be imposed on the production schedule:

$$x + 4y + 5z \leq 32$$

$$x + 3y + 2z \leq 29$$

Determine the best production schedule for this company, and find the best value of the revenue function.

8.36 A problem in chemical equilibrium is to minimize

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i \left(w_i + \ln P + \ln \frac{x_i}{\sum_{i=1}^{n} x_i} \right)$$

subject to the material balances

$$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2$$

$$x_4 + 2x_5 + x_6 + x_7 = 1$$

$$x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1$$

Given $P = 750$ and w_i,

i	w_i	i	w_i
1	−10.021	6	−18.918
2	−21.096	7	−28.032
3	−37.986	8	−14.640
4	−9.846	9	−30.594
5	−28.653	10	−26.111

what is $\mathbf{x}^*$ and $f(\mathbf{x}^*)$?

8.37 The objective is to fit a fifth-order polynomial to the curve $y = x^{1/3}$. To avoid fluctuations from the desired curve, divide the curve into ten points.

$$\mathbf{x}_i (i = 1, \ldots, 10) = (0.5, 1, 4.5, 8, 17.5, 27, 45.5, 64, 94.5, 125)$$

and fit the polynomial (find the values of a_i)

$$P(\mathbf{a},\mathbf{x}) = a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$$

by solving the following problem

$$\text{Minimize:} \quad f(\mathbf{x}) \sum_{i=1}^{10} [P(\mathbf{a}, x_i) - x_i^{1/3})^2$$

$$\text{Subject to:} \quad 0 \le P(\mathbf{a},j) \le 5, \quad j = 1,8,27,64$$

$$P(\mathbf{a}, 125) = 5$$

8.38 The Williams–Otto process as posed in this problem involves ten variables and seven constraints leaving 3 degrees of freedom. Three starting points are shown in Table P8.38.1. Find the maximum Q and the values of the ten variables from one of the starting points (S.P.). The minimum is very flat.

Figure P8.38 shows a simplified block diagram of the process. The plant consists of a perfectly stirred reactor, a decanter, and a distillation column in series. There is recycle from the column reboiler to the reactor.

The mathematical descriptions of each plant unit are summarized in Tables P8.38.2 and P8.38.3. The return function for this process as proposed by Williams and Otto (1960) and slightly modified by DiBella and Stevens (1965) to a variable reactor volume problem is

$$\text{Maximize:} \quad Q = \frac{100}{600\,V_p}[8400(0.3F_P + 0.0068\,F_D - 0.02F_A - 0.03F_B - 0.01F_G)$$

$$- 2.22F_R - (0.124)(8400)(0.3F_P + 0.0068F_D) - 60V_p]$$

$$= \text{Return}\,(\%)$$

TABLE P8.38.1
Starting points

Variable	x	x_1	x_2	x_3
F_{RA}	1	18,187	6,000	22,381
F_{RB}	2	60,815	35,000	72,297
F_{RC}	3	3,331	10,000	4,391
F_{RE}	4	60,542	15,000	78,144
F_G	5	3,609	5,000	3,140
F_{RP}	6	10,817	6,000	12,557
α	7	0.761	0.789	0.81
F_A	8	13,546	10,000	12,876
F_B	9	31,523	40,000	29,416
T	10	656	610	648

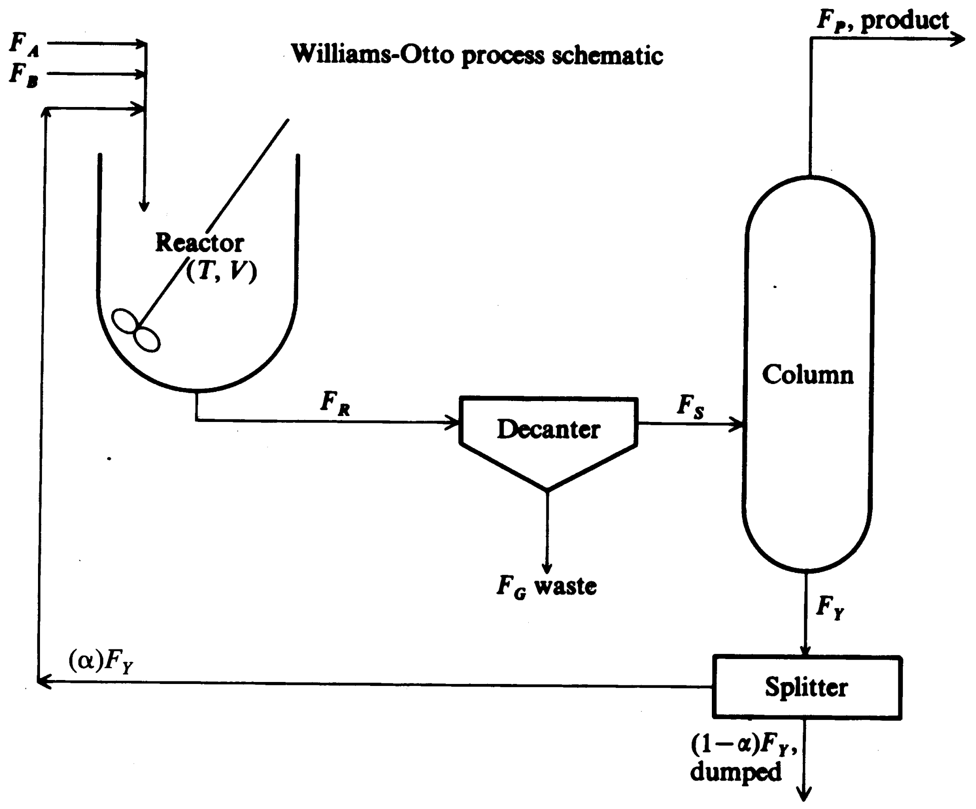

FIGURE P8.38

TABLE P8.38.2
g_i = residual of mass balance on component i, $i = A, B, C, E, G$

Constraints
$g_1 = F_A + F_{TA} - F_{RA} - P_1 = 0$
$g_2 = F_B + F_{TB} - F_{RB} - P_1 - P_2 = 0$
$g_3 = F_{TC} + 2P_1 - F_{RC} - 2P_2 - P_3 = 0$
$g_4 = F_{TE} + 2P_2 - F_{RE} = 0$
$g_5 = F_{TC} + P_2 - F_{RG} - 0.5P_3 = 0$
g_6 = overall mass balance
$= F_A + F_B - F_D - F_P = 0$
g_7 = production requirement
$= F_p - 4763 = 0$
$0 \le \alpha \le 1 \quad 500 \le T \le 1000$

TABLE P8.38.3
Williams–Otto unit mathematical models

Decanter			
$F_{Si} = F_{Ri}$	$i = A, B, C, E, P$		
$F_{SG} = 0$			
$F_{C_b} = F_{RC}$			
Distillation column			
$F_{Yi} = F_{Si}$	$i = A, B, C, E, G$		
$F_P = F_{SP} - 0.1F_{SE}$			
$F_{YP} = F_{SP} - F_P$			
Splitter			
$F_{Ti} = \alpha F_{Yi}$	$i = A, B, C, E, G, P$		
$F_{Di} = (1 - \alpha)F_{Yi}$			
Reactor ($V = 0.0002964\ F_R$)	i	a_i	b_i
$K_i = a_i \exp(-b_i/T)V/F_R^2$			
$P_1 = K_1F_{RA}F_{RB}$	1	5.9755×10^9	12,000
$P_2 = K_2F_{RB}F_{RC}$	2	2.5962×10^{12}	15,000
$P_3 = K_3F_{RC}F_{RG}$	3	9.6283×10^{15}	20,000

TABLE P8.38.4
Williams–Otto process nomenclature

F_A, F_B	Fresh feeds of components A, B (lb/h)
F_R	Total reactor output flow rate
F_{Ri}	Reactor output flow rate of component i
F_{Si}	Decanter output flow rate of component i
F_G	Decanter bottoms flow rate of component G
F_P	Column overhead flow rate of component P
F_{Yi}	Column bottoms flow rate of component i
F_D	Total column bottoms takeoff flow rate
F_{Di}	Column bottoms takeoff flow rate of component i
F_T	Total column bottoms recycle flow rate
F_{Ti}	Column bottoms recycle flow rate of component i
α	Fraction of column bottoms recycled to reactor
V	Reactor volume (ft^3)
ρ	Density of reaction mixture (assumed constant, 50 lb/ft^3)

8.39 Klein and Klimpel (1967) described an NLP involving the optimal selection of plant sites and plant sizes over time. The functions representing fixed and working capital were of the form

$$\text{Fixed capital:} \qquad \text{Cost} = a_0 + a_1 S^{a_2}$$

$$\text{Working capital:} \quad \text{Cost} = b_0 + b_1 P + b_2 S^{a_2}$$

where S = plant size
P = annual production
a's, b's = known constants obtained empirically

Variable annual costs were expressed in the form of

$$\text{Cost} = P(c_1 + c_2 S + c_3 S^{c_4})$$

Transportation costs were assumed to be proportional to the size of the shipments for a given source and destination.

The objective function is the net present value, NPV (sum of the discounted cash flows), using a discount rate of 10 percent. All flows except capital were assumed to be uniformly distributed over the year; working capital was added or subtracted instantaneously at the beginning of each year, and fixed capital was added only in the zero year.

The continuous discounting factors were

1. For instantaneous funds,

$$F_i = e^{-ry} \quad (r = \text{interest rate}, y = \text{years hence})$$

2. For uniformly flowing funds,

$$F_u = \frac{e^r - 1}{r} e^{-ry}$$

The variable y may be positive (after year zero) or negative (before year zero) or zero (for year ending with point zero in time).

As prices and revenue were not considered, maximization of net present value was equivalent to minimization of net cost.

Let P_{ijk} be the amount of product shipped from location $i(i = 1, 2, 3, 4)$ to market $j\ (j = 1, 2, 3)$ in year $k\ (k = 0, 1, 2, 3)$. Let S_i and $\bar{S}_1$ be, respectively, the size of plant in location i, and a variable restricted to 0 or 1, depending on whether S_i is 0. Furthermore, let M_{ojk} be the market demand at center j in year k. Finally, for the sake of convenience, let P_{iok} denote the total production in plant i during year k.

The nonlinear programming problem is: Find S_i and P_{ijk} that will

$$\text{Maximize:} \sum_i \text{NPV} \quad \text{(including shipping)}$$

$$\text{Subject to:} \sum_i P_{ijk} = M_{ojk}$$

$$S_i \geq 0, \qquad P_{ijk} \geq 0$$

Table P8.39.1 indicates how the net present value was determined for location 1; NPV relations for the other locations were similarly formed. Table P8.39.2 lists the

TABLE P8.39.1
Net percent value

1. Contribution of fixed capital (plant 1)

Year	Fixed capital	Discount factor	Discounted cash flow
0(1967)	$0.7\bar{S}_1 + 1.5S_1^6$	1.0517	$-0.7362\bar{S}_1 - 1.5775S_1^6$

2. Contribution of working capital (plant 1)

Year end	Working capital	Discount factor	Discounted cash flow at 10% discount rate
0	$0.4\bar{S}_1 + 0.2P_{101} + 0.05S_1^6$	1.000	$-0.4\bar{S}_1 - 0.2P_{101} - 0.05S_1^6$
1	$0.2(P_{102} - P_{101})$	0.9048	$-0.1810P_{102} + 0.1810P_{101}$
2	$0.2(P_{103} - P_{102})$	0.8187	$-0.1637P_{103} + 0.1637P_{102}$
3	$-0.4\bar{S}_1 - 0.2P_{103} - 0.05S_1^6$	0.7408	$+0.2963\bar{S}_1 + 0.1482P_{103} + 0.0370S_1^6$

3. Contribution of operational cost (plant 1) a. Cost tabulation (excluding shipping)

Year	Amount	Depreciation*	Other costs
1	P_{101}	$0.4667\bar{S}_1 + 1.0S_1^6$	$0.03\bar{S}_1 - 0.01S_1 + 0.05S_1^{0.45} + 0.07S_1^{0.6} + 0.1P_{101}$ $-0.05P_{101}S_1 + 0.4P_{101}S_1^{-0.55}$
2	P_{102}	$0.1167\bar{S}_1 + 0.25S_1^6$	$0.03S_1 - 0.01S_1 + 0.05S_1^{0.45} + 0.07S_1^{0.6} + 0.095P_{102}$ $-0.0048P_{102}S_1 + 0.38P_{102}S_1^{-0.55}$
3	P_{103}	$0.1166\bar{S}_1 + 0.25S_1^6$	$0.03S_1 - 0.01S_1 + 0.05S_1^{0.45} + 0.07S_1^{0.6} + 0.0903P_{103}$ $-0.0045P_{103}S_1 + 0.361P_{103}S_1$

b. Discounted cash flow of costs (plant 1)

Year	Discount factor	Discounted cost flow at 10% discount rate
1	0.9516	$0.1983S_1 + 0.0049\bar{S}_1 - 0.0247S_1^{0.45} + 0.4221S_1^{0.6} - 0.0495P_{101}$ $+0.0025P_{101}S_1 - 0.1979P_{101}S_1^{-0.55}$
2	0.8611	$0.0348S_1 + 0.0045\bar{S}_1 - 0.0224S_1^{0.45} + 0.0720S_1^{0.6} - 0.0425P_{102}$ $+0.0020P_{102}S_1 - 0.1702P_{102}S_1^{-0.55}$
3	0.7791	$0.0315S_1 + 0.0041\bar{S}_1 - 0.0203S_1^{0.45} + 0.0651S_1^{0.6} - 0.0366P_{103}$ $+0.0017P_{103}S_1 - 0.1463P_{103}S_1^{0.55}$

4. Contribution of shipping costs (from plant 1)

Year	Discount factor	Shipping cost	Discounted cash flow at 10% discount rate
1	0.9516	$0.8P_{121} + 0.5P_{121}$	$-0.396P_{121} - 0.247P_{131}$
2	0.8611	$0.7P_{122} + 0.45P_{132}$	$-0.313P_{122} - 0.201P_{132}$
3	0.7791	$0.6P_{123} + 0.4P_{133}$	$-0.243P_{123} - 0.162P_{133}$

*Method of double rate-declining balance and straight-line crossover was used.

TABLE P8.39.2
The objective function

$$
\begin{aligned}
Z_{\max} = {} & -0.5753\bar{S}_1 - 1.0313S_1^{0.6} - 0.0685P_{101} - 0.0597P_{102} - 0.0522P_{103} + 0.0135S_1 \\
& -0.0674S_1^{0.45} + 0.0025P_{101}S_1 + 0.0020P_{102}S_1 + 0.0017P_{103}S_1 \\
& -0.1979P_{101}S_1^{-0.55} - 0.1702P_{102}S_1^{-0.55} - 0.1463P_{103}S_1^{0.55} - 0.396P_{121} \\
& -0.247P_{131} - 0.313P_{122} - 0.202P_{132} - 0.243P_{123} - 0.162P_{133} - 0.3428\bar{S}_2 \\
& -0.8920S_2^{0.6} - 0.0685P_{201} - 0.0597P_{202} - 0.0522P_{203} + 0.0135S_2 - 0.0809S_1^{0.45} \\
& +0.0025P_{201}S_2 + 0.0020P_{202}S_2 + 0.0017P_{203}S_2 - 0.02227P_{201}S_2^{-0.55} \\
& -0.1914P_{202}S_2^{-0.55} - 0.1645P_{203}S_2^{-0.55} - 0.396P_{211} - 0.495P_{231} - 0.313P_{212} \\
& -0.448P_{232} - 0.243P_{213} - 0.405P_{233} - 0.3164\bar{S}_3 - 1.2987S_3^{0.6} - 0.0942P_{301} \\
& -0.0819P_{302} - 0.0712P_{303} - 0.0539S_3^{0.45} + 0.0030P_{301}S_3 + 0.0026P_{302}S_3 \\
& +0.0022P_{303}S_3 - 0.2227P_{301}S_3^{-0.55} - 0.1914P_{302}S_3^{-0.55} - 0.1645P_{303}S_3^{0.55} \\
& -0.247P_{311} - 0.495P_{321} - 0.202P_{312} - 0.448P_{322} - 0.162P_{313} - 0.405P_{323} \\
& -0.2441\bar{S}_4 - 1.3707S_4^{0.6} - 0.0577P_{401} - 0.0504P_{402} - 0.0440P_{403} \\
& +0.0020P_{401}S_4 + 0.0017P_{402}S_4 + 0.0015P_{403}S_4 - 0.1484P_{401}S_4^{-0.55} \\
& -0.1276P_{402}S_4^{-0.55} - 0.1097P_{403}S_4^{-0.55} - 0.495P_{411} - 0.099P_{421} - 0.040P_{431} \\
& -0.448P_{412} - 0.090P_{422} - 0.040P_{432} - 0.405P_{413} - 0.088P_{423} - 0.041P_{433}
\end{aligned}
$$

TABLE P8.39.3
The constants

(1) $S_1 + S_2 + S_3 + S_4 = 10$	(2) $P_{111} + P_{211} + P_{311} + P_{411} = 1$	
(3) $P_{112} + P_{212} + P_{312} + P_{412} = 4$	(4) $P_{113} + P_{213} + P_{313} + P_{413} = 5$	
(5) $P_{121} + P_{221} + P_{321} + P_{421} = 2$	(6) $P_{122} + P_{222} + P_{322} + P_{422} = 3$	
(7) $P_{123} + P_{223} + P_{323} + P_{423} = 2$	(8) $P_{131} + P_{231} + P_{331} + P_{431} = 4$	
(9) $P_{132} + P_{232} + P_{332} + P_{432} = 3$	(10) $P_{133} + P_{233} + P_{323} + P_{433} = 2$	
(11) $P_{101} - S_2 \le 0$	(12) $P_{102} - S_1 \le 0$	(13) $P_{103} - S_1 \le 0$
(14) $P_{201} - S_2 \le 0$	(15) $P_{202} - S_2 \le 0$	(16) $P_{203} - S_2 \le 0$
(17) $P_{301} - S_3 \le 0$	(18) $P_{302} - S_3 \le 0$	(19) $P_{303} - S_2 \le 0$
(20) $P_{401} - S_4 \le 0$	(21) $P_{402} - S_4 \le 0$	(22) $P_{403} - S_4 \le 0$

overall objective function, and Table P8.39.3 lists (1) the 22 constraints, (2) one equation constraining the total plant capacity to be 10 million pounds per year, (3) nine equations requiring satisfaction of the three markets every year, and (4) 12 inequalities calling for plant production not to exceed plant capacity. In addition, the nonnegativity constraints are applicable to all 40 variables. Thus the problem has 10 linear equality constraints and 52 inequality constraints.

8.40 Consider the problem of minimizing the purchase of fuel oil when it is needed to produce an output of 50 MW from a two-boiler turbine–generator combination that can use fuel oil or blast furnace gas (BFG) or any combination of these. The maximum available BFG is specified.

By applying nonlinear curve fitting, we obtained the fuel requirements for the two generators explicitly in terms of MW produced. For generator 1 we have the fuel requirements for fuel oil in tons per hour (x_{11})

$$f_1 = 1.4609 + 0.15186x_{11} + 0.00145x_{11}^2$$

and for BFG in fuel units per hour (x_{12})

$$f_2 = 1.5742 + 0.1631x_{12} + 0.001358x_{12}^2$$

where $(x_{11} + x_{12})$ is the output in MW of generator 1. The range of operation of the generator is

$$18 \le (x_{11} + x_{12}) \le 30$$

Similarly for generator 2 the requirement for fuel oil is

$$g_1 = 0.8008 + 0.2031x_{21} + 0.000916x_{21}^2$$

and for BFG,

$$g_2 = 0.7266 + 0.2256x_{22} + 0.000778x_{22}^2$$

where $(x_{21} + x_{22})$ is the output in MW of generator 2. The range of operation of the second generator is

$$14 \le (x_{21} + x_{22}) \le 25$$

It is assumed that only 10.0 fuel units of BFG are available each hour and that each generator may use any combination of fuel oil or BFG. It is further assumed that when a combination of fuel oil and BFG is used, the effects are additive.

The problem is to produce 50 MW from the two generators in such a way that the amount of fuel oil consumed is minimum. Use successive linear programming.

8.41 For the purposes of planning you are asked to determine the optimal heat exchanger areas for the sequence of three exchangers as shown in Figure P8.41.
Data:

Exchanger	Overall heat transfer coefficient [U Btu/(h)(ft^2)(°F)]	Area required (ft^2)	Duty (Btu/h)
1	$U_1 = 120$	A_1	Q_1
2	$U_2 = 80$	A_2	Q_2
3	$U_3 = 40$	A_3	Q_3

$wCp = 10^5$ Btu/(h)(°F)

Hint: Find the temperatures T_1, T_2, T_3 such that $\sum A_i$ is a minimum.

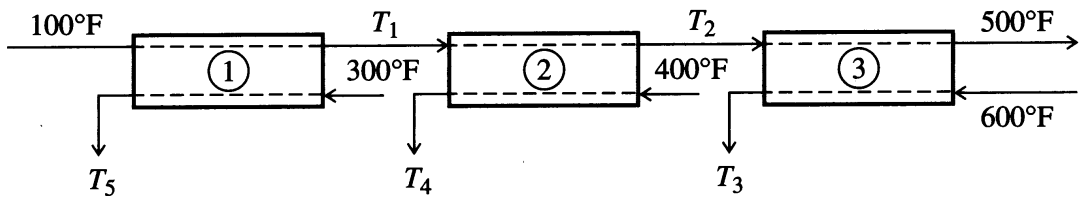

FIGURE P8.41