Chapter 1

Introduction to Nonlinear Programming

1.1 Preliminaries

This book concentrates on numerical methods for solving the optimal control problem. The fundamental principle of all effective numerical optimization methods is to solve a difficult problem by solving a sequence of simpler subproblems. In particular, the solution of an optimal control problem will require the solution of one or more finite-dimensional subproblems. As a prelude to our discussions on optimal control, this chapter will focus on the nonlinear programming (NLP) problem. The NLP problem requires finding a finite number of variables such that an *objective function* or *performance index* is optimized without violating a set of *constraints*. The NLP problem is often referred to as *parameter optimization*. Important special cases of the NLP problem include *linear programming* (LP), *quadratic programming* (QP), and *least squares* problems.

Before proceeding further, it is worthwhile to establish the notational conventions used throughout the book. This is especially important since the subject matter covers a number of different disciplines, each with its own notational conventions. Our goal is to present a unified treatment of all these fields. As a rule, scalar quantities will be denoted by lowercase letters (e.g., α). Vectors will be denoted by boldface lowercase letters and will usually be considered column vectors, as in

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \tag{1.1}$$

where the individual components of the vector are x_k for $k = 1, \dots, n$. To save space, it will often be convenient to define the *transpose*, as in

$$\mathbf{x}^{\mathsf{T}} = (x_1, x_2, \dots, x_n). \tag{1.2}$$

A *sequence* of vectors will often be denoted as $\mathbf{x}_k, \mathbf{x}_{k+1}, \dots$. Matrices will be denoted by boldface capital letters, as in

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}. \tag{1.3}$$

1.2 Newton's Method in One Variable

The fundamental approach to most iterative schemes was suggested over 300 years ago by Newton. In fact, Newton's method is the basis for all of the algorithms we will describe. We begin with the simplest form of Newton's method and then in subsequent sections generalize the discussion until we have presented one of the most widely used NLP algorithms, namely the *sequential quadratic programming* (SQP) method.

Suppose it is required to find the value of the variable x such that the *constraint* function

$$c(x) = 0. \tag{1.4}$$

Let us denote the solution by x^* and let us assume x is a guess for the solution. The basic idea of Newton's method is to approximate the nonlinear function $c(x)$ by the first two terms in a Taylor series expansion about the current point x. This yields a linear approximation for the constraint function at the new point $\bar{x}$, which is given by

$$c(\bar{x}) = c(x) + c'(x)(\bar{x} - x), \tag{1.5}$$

where $c'(x) = dc/dx$ is the slope of the constraint at x. Using this linear approximation, it is reasonable to compute $\bar{x}$, a new estimate for the root, by solving (1.5) such that $c(\bar{x}) = 0$, i.e.,

$$\bar{x} = x - [c'(x)]^{-1} c(x). \tag{1.6}$$

Typically, we denote $p \equiv \bar{x} - x$ and rewrite (1.6) as

$$\bar{x} = x + p, \tag{1.7}$$

where

$$p = -[c'(x)]^{-1} c(x). \tag{1.8}$$

Of course, in general, $c(x)$ is not a linear function of x, and consequently we cannot expect that $c(\bar{x}) = 0$. However, we might hope that $\bar{x}$ is a better estimate for the root x^* than the original guess x; in other words we might expect that

$$|\bar{x} - x^*| \le |x - x^*| \tag{1.9}$$

and

$$|c(\bar{x})| \le |c(x)|. \tag{1.10}$$

If the new point is an improvement, then it makes sense to repeat the process, thereby defining a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \dots$ with point $(k+1)$ in the sequence given by

$$x^{(k+1)} = x^{(k)} - [c'(x^{(k)})]^{-1} c(x^{(k)}). \tag{1.11}$$

For notational convenience, it usually suffices to present a single step of the algorithm, as in (1.6), instead of explicitly labeling the information at step k using the superscript notation $x^{(k)}$. Nevertheless, it should be understood that the algorithm defines a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$. The sequence is said to *converge* to x^* if

$$\lim_{k\to\infty} |x^{(k)} - x^*| = 0. \tag{1.12}$$

In practice, of course, we are not interested in letting $k \to \infty$. Instead we are satisfied with terminating the sequence when the computed solution is "close" to the answer. Furthermore, the *rate of convergence* is of paramount importance when measuring the computational efficiency of an algorithm. For Newton's method, the rate of convergence is said to be *quadratic* or, more precisely, *q-quadratic* (cf. [71]). The impact of quadratic convergence can be dramatic. Loosely speaking, it implies that each successive estimate of the solution will *double* the number of significant digits!

Example 1.1 NEWTON'S METHOD—ROOT FINDING. To demonstrate, let us suppose we want to solve the constraint

$$c(x) = a_1 + a_2 x + a_3 x^2 = 0, \tag{1.13}$$

where the coefficients a_1, a_2, a_3 are chosen such that $c(0.1) = -0.05$, $c(0.25) = 0$, and $c(0.9) = 0.9$. Table 1.1 presents the Newton iteration sequence beginning from the initial guess $x = 0.85$ and proceeding to the solution at $x^* = 0.25$. Figure 1.1 illustrates the first three iterations. Notice in Table 1.1 that the error between the computed solution and the true value, which is tabulated in the third column, exhibits the expected doubling in significant figures from the fourth iteration to convergence.

So what is wrong with Newton's method? Clearly, quadratic convergence is a very desirable property for an algorithm to possess. Unfortunately, if the initial guess is not sufficiently close to the solution, i.e., within the *region of convergence*, Newton's method may diverge. As a simple example, Dennis and Schnabel [71] suggest applying Newton's method to solve $c(x) = \arctan(x) = 0$. This will diverge when the initial guess $|x^{(0)}| > a$, converge when $|x^{(0)}| < a$, and cycle indefinitely if $|x^{(0)}| = a$, where $a = 1.3917452002707$. In essence, Newton's method behaves well near the solution (*locally*) but lacks something permitting it to converge *globally*. So-called globalization techniques, aimed at correcting this deficiency, will be discussed in subsequent sections. A second difficulty occurs when

Table 1.1. *Newton's method for root finding.*

Iter.	$c(x)$	x	$\lvert x - x^*\rvert$
1	0.79134615384615	0.85000000000000	0.60000000000000
2	0.18530192382759	0.47448669201521	0.22448669201521
3	$3.5942428588261\times10^{-2}$	0.30910437279376	$5.9104372793756\times10^{-2}$
4	$3.6096528286200\times10^{-3}$	0.25669389900972	$6.6938990097217\times10^{-3}$
5	$5.7007630268141\times10^{-5}$	0.25010744198003	$1.0744198002549\times10^{-4}$
6	$1.5161639596584\times10^{-8}$	0.25000002858267	$2.8582665845267\times10^{-8}$
7	$1.0547118733939\times10^{-15}$	0.25000000000000	$1.8873791418628\times10^{-15}$

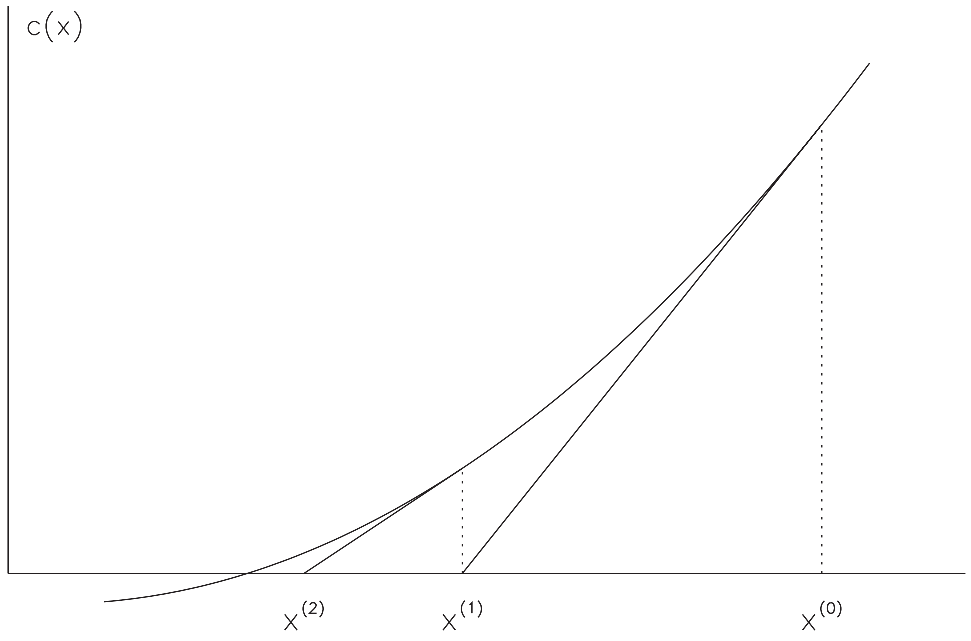

Figure 1.1. *Newton's method for root finding.*

the slope $c'(x) = 0$. Clearly, the correction defined by (1.6) is not well defined in this case. In fact, Newton's method loses its quadratic convergence property if the slope is zero at the solution, i.e., $c'(x^*) = 0$. Finally, Newton's method requires that the slope $c'(x)$ can be computed at every iteration. This may be difficult and/or costly, especially when the function $c(x)$ is complicated.

1.3 Secant Method in One Variable

Motivated by a desire to eliminate the explicit calculation of the slope, one can consider approximating it at x^k by the secant

$$c'(x^k) \approx B = \frac{c(x^k) - c(x^{k-1})}{x^k - x^{k-1}} \equiv \frac{\Delta c}{\Delta x}. \tag{1.14}$$

Notice that this approximation is constructed using two previous iterations but requires values only for the constraint function $c(x)$. This expression can be rewritten to give the so-called secant condition

$$B\Delta x = \Delta c, \tag{1.15}$$

where B is the (scalar) secant approximation to the slope. Using this approximation, it then follows that the Newton iteration (1.6) is replaced by the secant iteration

$$\bar{x} = x - B^{-1}c(x) = x + p, \tag{1.16}$$

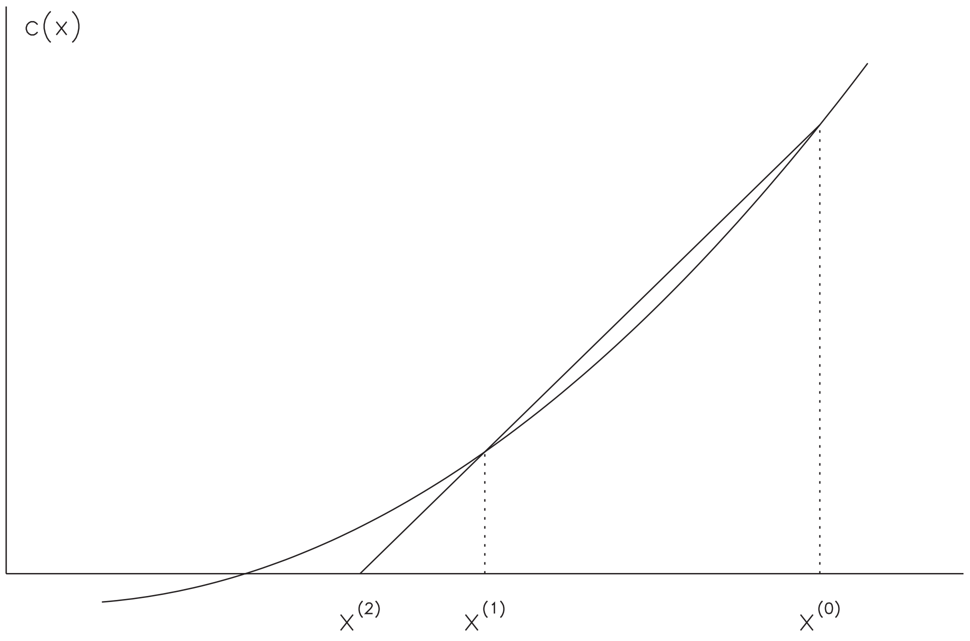

Figure 1.2. *Secant method for root finding.*

which is often written as

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{c(x^k) - c(x^{k-1})} c(x^k). \tag{1.17}$$

Figure 1.2 illustrates a secant iteration applied to Example 1.1 described in the previous section.

Clearly, the virtue of the secant method is that it does not require calculation of the slope $c'(x^k)$. While this may be advantageous when derivatives are difficult to compute, there is a downside! The secant method is *superlinearly* convergent, which, in general, is not as fast as the quadratically convergent Newton algorithm. Thus, we can expect convergence will require more iterations, even though the cost per iteration is less. A distinguishing feature of the secant method is that the slope is approximated using information from previous iterates in lieu of a direct evaluation. This is the simplest example of a so-called quasi-Newton method.

1.4 Newton's Method for Minimization in One Variable

Now let us suppose we want to compute the value x^* such that the nonlinear *objective function* $F(x^*)$ is a minimum. The basic notion of Newton's method for root finding is to approximate the nonlinear constraint function $c(x)$ by a simpler model (i.e., linear) and then compute the root for the linear model. If we are to extend this philosophy to optimization, we must construct an approximate model of the objective function. Just as in the development of (1.5), let us approximate $F(x)$ by the first three terms in a Taylor series expansion about the current point x:

$$F(\bar{x}) = F(x) + F'(x)(\bar{x}-x) + \frac{1}{2}(\bar{x}-x)F''(x)(\bar{x}-x). \tag{1.18}$$

Notice that we cannot use a linear model for the objective because a linear function does not have a finite minimum point. In contrast, a quadratic approximation to $F(x)$ is the simplest approximation that does have a minimum. Now for $\bar{x}$ to be a minimum of the quadratic (1.18), we must have

$$\frac{dF}{d\bar{x}} \equiv F'(\bar{x}) = 0 = F'(x) + F''(x)(\bar{x}-x). \tag{1.19}$$

Solving for the new point yields

$$\bar{x} = x - [F''(x)]^{-1}F'(x). \tag{1.20}$$

The derivation has been motivated by minimizing $F(x)$. Is this equivalent to solving the slope condition $F'(x) = 0$? It would appear that the iterative *optimization* sequence defined by (1.20) is the same as the iterative *root-finding* sequence defined by (1.6), provided we replace $c(x)$ by $F'(x)$. Clearly, a quadratic model for the objective function (1.18) produces a linear model for the slope $F'(x)$. However, the condition $F'(x) = 0$ defines only a *stationary point*, which can be a minimum, a maximum, or a point of inflection. Apparently what is missing is information about the *curvature* of the function, which would determine whether it is concave up, concave down, or neither.

Figure 1.3 illustrates a typical situation. In the illustration, there are two points with zero slopes; however, there is only one minimum point. The minimum point is dis-

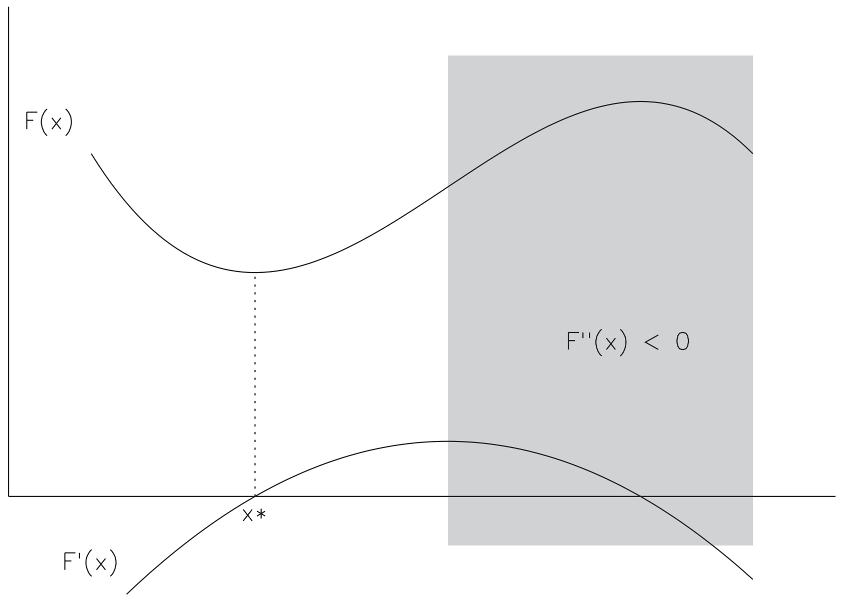

Figure 1.3. *Minimization in one variable.*

tinguished from the maximum by the algebraic sign of the second derivative $F''(x)$. Formally, we have

Necessary Conditions:

$$F'(x^*) = 0, \tag{1.21}$$

$$F''(x^*) \geq 0; \tag{1.22}$$

Sufficient Conditions:

$$F'(x^*) = 0, \tag{1.23}$$

$$F''(x^*) > 0. \tag{1.24}$$

Note that the sufficient conditions require that $F''(x^*) > 0$, defining a *strong local minimizer* in contrast to a *weak local minimizer*, which may have $F''(x^*) = 0$. It is also important to observe that these conditions define a *local* rather than a *global* minimizer.

1.5 Newton's Method in Several Variables

The preceding sections have addressed problems involving a single variable. In this section, let us consider generalizing the discussion to functions of many variables. In particular, let us consider how to find the n-vector $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_n)$ such that

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{bmatrix} = \mathbf{0}. \tag{1.25}$$

For the present, let us assume that the number of constraints and variables is the same, i.e., $m = n$. Just as in one variable, a linear approximation to the constraint functions analogous to (1.5) is given by

$$\mathbf{c}(\overline{\mathbf{x}}) = \mathbf{c}(\mathbf{x}) + \mathbf{G}(\overline{\mathbf{x}} - \mathbf{x}), \tag{1.26}$$

where the *Jacobian matrix* $\mathbf{G}$ is defined by

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \cdots & \frac{\partial c_1}{\partial x_n} \\ \frac{\partial c_2}{\partial x_1} & \frac{\partial c_2}{\partial x_2} & \cdots & \frac{\partial c_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial c_m}{\partial x_1} & \frac{\partial c_m}{\partial x_2} & \cdots & \frac{\partial c_m}{\partial x_n} \end{bmatrix}. \tag{1.27}$$

By convention, the m rows of $\mathbf{G}$ correspond to constraints and the n columns to variables. As in one variable, if we require that $\mathbf{c}(\overline{\mathbf{x}}) = \mathbf{0}$ in (1.26), we can solve the linear system

$$\mathbf{G}\mathbf{p} = -\mathbf{c} \tag{1.28}$$

for the *search direction* **p**, which leads to an iteration of the form

$$\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.29}$$

Thus, each Newton iteration requires a linear approximation to the nonlinear constraints **c**, followed by a step from **x** to the solution of the linearized constraints at $\bar{\mathbf{x}}$. Figure 1.4 illustrates a typical situation when $n = m = 2$. It is important to remark that the multidimensional version of Newton's method shares all of the properties of its one-dimensional counterpart. Specifically, the method is quadratically convergent provided it is within a region of convergence, and it may diverge unless appropriate globalization strategies are employed. Furthermore, in order to solve (1.28) it is necessary that the Jacobian **G** be *nonsingular*, which is analogous to requiring that $c'(x) \neq 0$ in the univariate case. And, finally, it is necessary to actually compute **G**, which can be costly.

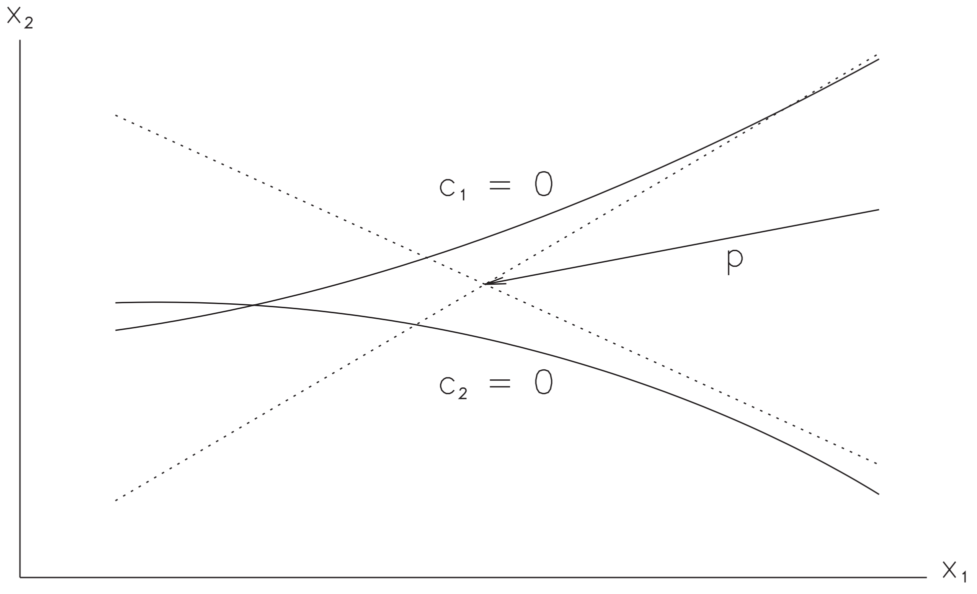

Figure 1.4. *Newton's method in two variables.*

1.6 Unconstrained Optimization

Let us now consider the multidimensional unconstrained minimization problem. Suppose we want to find the n-vector $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_n)$ such that the function $F(\mathbf{x}) = F(x_1, \ldots, x_n)$ is a minimum. Just as in the univariate case (1.18), let us approximate $F(\mathbf{x})$ by the first three terms in a Taylor series expansion about the point **x**:

$$F(\bar{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})(\bar{\mathbf{x}} - \mathbf{x}) + \frac{1}{2}(\bar{\mathbf{x}} - \mathbf{x})^\mathsf{T}\mathbf{H}(\mathbf{x})(\bar{\mathbf{x}} - \mathbf{x}). \tag{1.30}$$

The Taylor series expansion involves the n-dimensional *gradient* vector

$$\mathbf{g}(\mathbf{x}) \equiv \nabla_x F = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{bmatrix} \tag{1.31}$$

and the symmetric $n \times n$ *Hessian matrix*

$$\mathbf{H} \equiv \nabla^2_{xx} F = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_n^2} \end{bmatrix}. \tag{1.32}$$

It is common to define the *search direction* $\mathbf{p} = \overline{\mathbf{x}} - \mathbf{x}$ and then rewrite (1.30) as

$$F(\overline{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}. \tag{1.33}$$

The scalar term $\mathbf{g}^{\mathsf{T}}\mathbf{p}$ is referred to as the *directional derivative* along $\mathbf{p}$ and the scalar term $\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}$ is called the *curvature* or *second directional derivative* in the direction $\mathbf{p}$.

It is instructive to examine the behavior of the series (1.33). First, let us suppose that the expansion is about the minimum point $\mathbf{x}^*$. Now if $\mathbf{x}^*$ is a local minimum, then the objective function must be larger at all neighboring points, that is, $F(\overline{\mathbf{x}}) > F(\mathbf{x}^*)$. In order for this to be true, the slope in all directions must be zero, that is, $(\mathbf{g}^*)^{\mathsf{T}}\mathbf{p} = \mathbf{0}$, which implies we must have

$$\mathbf{g}(\mathbf{x}^*) = \begin{bmatrix} g_1(\mathbf{x}^*) \\ \vdots \\ g_n(\mathbf{x}^*) \end{bmatrix} = \mathbf{0}. \tag{1.34}$$

This is just the multidimensional analogue of the condition (1.21). Furthermore, if the function curves up in all directions, the point $\mathbf{x}^*$ is called a strong local minimum and the third term in the expansion (1.33) must be positive:

$$\mathbf{p}^{\mathsf{T}}\mathbf{H}^*\mathbf{p} > 0. \tag{1.35}$$

A matrix[1] that satisfies this condition is said to be *positive definite*. If there are some directions with zero curvature, i.e., $\mathbf{p}^{\mathsf{T}}\mathbf{H}^*\mathbf{p} \geq 0$, then $\mathbf{H}^*$ is said to be *positive semidefinite*. If there are directions with both positive and negative curvature, the matrix is called *indefinite*. In summary, we have

[1] $\mathbf{H}^* \equiv \mathbf{H}(\mathbf{x}^*)$ (not the conjugate transpose, as in some texts).

Necessary Conditions:

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.36}$$

$$\mathbf{p}^\mathsf{T}\mathbf{H}^*\mathbf{p} \geq 0; \tag{1.37}$$

Sufficient Conditions:

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.38}$$

$$\mathbf{p}^\mathsf{T}\mathbf{H}^*\mathbf{p} > 0. \tag{1.39}$$

The preceding discussion was motivated by an examination of the Taylor series about the minimum point $\mathbf{x}^*$. Let us now consider the same quadratic model about an arbitrary point $\mathbf{x}$. Then it makes sense to choose a new point $\overline{\mathbf{x}}$ such that the gradient at $\overline{\mathbf{x}}$ is zero. The resulting linear approximation to the gradient is just

$$\overline{\mathbf{g}} = \mathbf{0} = \mathbf{g} + \mathbf{H}\mathbf{p}, \tag{1.40}$$

which can be solved to yield the Newton search direction

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}. \tag{1.41}$$

Just as before, the Newton iteration is defined by (1.29). Since this iteration is based on finding a zero of the gradient vector, there is no guarantee that the step will move toward a local minimum rather than a stationary point or maximum. To preclude this, we must insist that the step be downhill, which requires satisfying the so-called descent condition

$$\mathbf{g}^\mathsf{T}\mathbf{p} < 0. \tag{1.42}$$

It is interesting to note that, if we use the Newton direction (1.41), the descent condition becomes

$$\mathbf{g}^\mathsf{T}\mathbf{p} = -\mathbf{g}^\mathsf{T}\mathbf{H}^{-1}\mathbf{g} < 0, \tag{1.43}$$

which can be true only if the Hessian is positive definite, i.e., (1.35) holds.

1.7 Recursive Updates

Regardless of whether Newton's method is used for solving nonlinear equations, as in Section 1.5, or for optimization, as described in Section 1.6, it is necessary to compute derivative information. In particular, one must compute either the Jacobian matrix (1.27) or the Hessian matrix (1.32). For many applications, this can be a costly computational burden. Quasi-Newton methods attempt to construct this information recursively. A brief overview of the most important recursive updates is included, although a more complete discussion can be found in [71], [99], and [82].

The basic idea of a recursive update is to construct a new estimate of the Jacobian or Hessian using information from previous iterates. Most well-known recursive updates are of the form

$$\overline{\mathbf{B}} = \mathbf{B} + \mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x}), \tag{1.44}$$

where the new estimate $\overline{\mathbf{B}}$ is computed from the old estimate $\mathbf{B}$. Typically, this calculation involves a low-rank modification $\mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x})$ that can be computed from the previous step:

$$\Delta\mathbf{c} = \mathbf{c}^k - \mathbf{c}^{k-1}, \tag{1.45}$$

$$\Delta\mathbf{x} = \mathbf{x}^k - \mathbf{x}^{k-1}. \tag{1.46}$$

The usual way to construct the update is to insist that the secant condition

$$\overline{\mathbf{B}}\Delta\mathbf{x} = \Delta\mathbf{c} \tag{1.47}$$

hold and then construct an approximation $\overline{\mathbf{B}}$ that is "close" to the previous estimate $\mathbf{B}$. In Section 1.3, the simplest form of this condition (1.15) led to the secant method. In fact, the generalization of this formula, proposed in 1965 by Broyden [50], is

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{c} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.48}$$

which is referred to as the *secant* or *Broyden update*. The recursive formula constructs a rank-one modification that satisfies the secant condition and minimizes the Frobenius norm between the estimates.

When a quasi-Newton method is used to approximate the Hessian matrix, as required for minimization, one cannot simply replace $\Delta\mathbf{c}$ with $\Delta\mathbf{g}$ in the secant update. In particular, the matrix $\overline{\mathbf{B}}$ constructed using (1.48) is not symmetric. However, there is a rank-one update that does maintain symmetry, known as the *symmetric rank-one* (SR1) update:

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.49}$$

where $\Delta\mathbf{g} \equiv \mathbf{g}^k - \mathbf{g}^{k-1}$. While the SR1 update does preserve symmetry, it does not necessarily maintain a positive definite approximation. In contrast, the update

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \frac{\mathbf{B}\Delta\mathbf{x}(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}}{(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}\Delta\mathbf{x}} \tag{1.50}$$

is a rank-two positive definite secant update provided $(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{g} > 0$ is enforced at each iteration. This update was discovered independently by Broyden [51], Fletcher [81], Goldfarb [103], and Shanno [159] in 1970 and is known as the *BFGS update*.

The effective computational implementation of a quasi-Newton update introduces a number of additional considerations. When solving nonlinear equations, the search direction from (1.28) is $\mathbf{p} = -\mathbf{G}^{-1}\mathbf{c}$, and for optimization problems the search direction given by (1.41) is $\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$. Since the search direction calculation involves the matrix inverse (either $\mathbf{G}^{-1}$ or $\mathbf{H}^{-1}$), one apparent simplification is to apply the recursive update directly to the inverse. In this case, the search direction can be computed simply by computing the matrix-vector product. This approach was proposed by Broyden for nonlinear equations, but has been considerably less successful in practice than the update given by (1.48), and is known as "Broyden's bad update." For unconstrained minimization, let us make the substitutions $\Delta\mathbf{x} \to \Delta\mathbf{g}$, $\Delta\mathbf{g} \to \Delta\mathbf{x}$, and $\mathbf{B} \to \mathbf{B}^{-1}$ in (1.50). By computing the inverse of the resulting expression, one obtains

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g})^{\mathsf{T}} + \Delta\mathbf{g}(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \sigma\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}, \tag{1.51}$$

where

$$\sigma = \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}{(\Delta\mathbf{g}^{\mathsf{T}}\Delta\mathbf{x})^2}.$$

This so-called inverse positive definite secant update is referred to as the *DFP update* for its discoverers Davidon [68] and Fletcher and Powell [85].

Even though many recursive updates can be applied directly to the inverse matrices, most practical implementations do not use this approach. When the matrices **G** and/or **H** are singular, the inverse matrices do not exist. Consequently, it is usually preferable to work directly with **G** and **H**. There are at least three issues that must be addressed by an effective implementation, namely efficiency, numerical conditioning, and storage. The solution of a dense linear system, such as (1.28) or (1.40), requires $\mathcal{O}(n^3)$ operations, compared with the $\mathcal{O}(n^2)$ operations needed to compute the matrix-vector product (1.41). However, this penalty can be avoided by implementing the update in "factored form." For example, the (positive definite) Hessian matrix can be written in terms of its Cholesky factorization as $\mathbf{H} = \mathbf{R}\mathbf{R}^{\mathsf{T}}$. Since the recursive update formulas represent low-rank modifications to **H**, it is possible to derive low-rank modifications to the factors **R**. By updating the matrices in factored form, the cost of computing the search direction can be reduced to $\mathcal{O}(n^2)$ operations, just as when the inverse is recurred directly. Furthermore, when the matrix factorizations are available, it is also possible to deal with rank deficiencies in a more direct fashion. Finally, when storage is an issue, the matrix at iteration k can be represented as the sum of L quasi-Newton updates to the original estimate $\mathbf{B}_0$ in the form

$$\mathbf{B}_k = \mathbf{B}_0 + \sum_{i=1}^{L} \mathbf{u}_i\mathbf{u}_i^{\mathsf{T}} - \sum_{i=1}^{L} \mathbf{v}_i\mathbf{v}_i^{\mathsf{T}}, \qquad (1.52)$$

where the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ denote information from iteration i. If the initial estimate $\mathbf{B}_0$ requires relatively little storage (e.g., is diagonal), then all operations involving the matrix $\mathbf{B}_k$ at iteration k can be performed without explicitly forming the (dense) matrix $\mathbf{B}_k$. This technique, called a *limited memory update*, requires only storing the vectors **u** and **v** over the previous L iterations.

We have motivated the use of a recursive update as a way to construct Jacobian and/or Hessian information. However, we have not discussed how fast an iterative sequence will converge when the recursive update is used instead of the exact information. All of the methods that use a recursive update exhibit *superlinear* convergence provided the matrices are nonsingular. In general, superlinear convergence is not as fast as quadratic convergence. One way to measure the rate of convergence is to compare the behavior of a Newton method and a quasi-Newton method on a quadratic function of n variables. Newton's method will terminate in one step, assuming finite-precision arithmetic errors are negligible. In contrast, a quasi-Newton method will terminate in at most n steps, provided the steplength α is chosen at each iteration to minimize the value of the objective function at the new point $F(\overline{\mathbf{x}}) = F(\mathbf{x} + \alpha\mathbf{p})$. The process of adjusting α is called a *line search* and will be discussed in Section 1.11.

1.8 Equality-Constrained Optimization

The preceding sections describe how Newton's method can be applied either to optimize an objective function $F(\mathbf{x})$ *or* to satisfy a set of constraints $\mathbf{c}(\mathbf{x}) = \mathbf{0}$. Suppose now that we want to do both, that is, choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.53}$$

subject to the $m \le n$ constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{0}. \tag{1.54}$$

The classical approach is to define the *Lagrangian*

$$L(\mathbf{x},\boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}), \tag{1.55}$$

where $\boldsymbol{\lambda}$ is an m-vector of *Lagrange multipliers*.[2]

In a manner analogous to the unconstrained case, optimality requires that derivatives with respect to both $\mathbf{x}$ and $\boldsymbol{\lambda}$ be zero. More precisely, necessary conditions for the point $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ to be an optimum are

$$\nabla_x L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \tag{1.56}$$

$$\nabla_\lambda L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}. \tag{1.57}$$

The gradient of L with respect to $\mathbf{x}$ is

$$\nabla_x L = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} = \nabla F - \sum_{i=1}^{m} \lambda_i \nabla c_i \tag{1.58}$$

and the gradient of L with respect to $\boldsymbol{\lambda}$ is

$$\nabla_\lambda L = -\mathbf{c}(\mathbf{x}). \tag{1.59}$$

Just as in the unconstrained case, these conditions do not distinguish between a point that is a minimum, a maximum, or simply a stationary point. As before, we require conditions on the curvature of the objective. Let us define the *Hessian of the Lagrangian* to be

$$\mathbf{H}_L = \nabla_{xx}^2 L = \nabla_{xx}^2 F - \sum_{i=1}^{m} \lambda_i \nabla_{xx}^2 c_i. \tag{1.60}$$

Then a sufficient condition is that

$$\mathbf{v}^{\mathsf{T}}\mathbf{H}_L\mathbf{v} > \mathbf{0} \tag{1.61}$$

for any vector $\mathbf{v}$ in the constraint tangent space. If one compares (1.35) with (1.61), an important difference emerges. For the unconstrained case, we require that the curvature be positive in *all* directions $\mathbf{p}$. However, (1.61) applies only to directions $\mathbf{v}$ in the constraint tangent space.

[2]Some authors define $L = F + \boldsymbol{\omega}^{\mathsf{T}}\mathbf{c}$, where $\boldsymbol{\omega} = -\boldsymbol{\lambda}$, in which case corresponding changes must be made when interpreting the arithmetic sign of the multipliers.

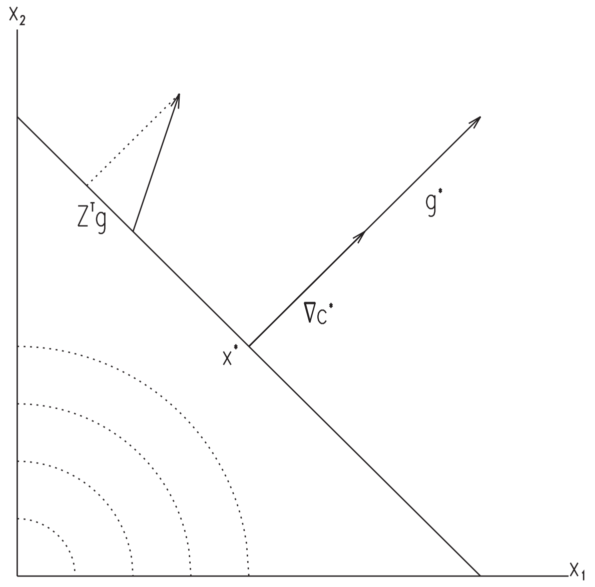

Figure 1.5. *Equality-constrained example.*

Example 1.2 Equality Constrained Minimization. To fully appreciate the meaning of these conditions, consider the simple example problem with two variables and one constraint illustrated in Figure 1.5. Let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = x_1 + x_2 - 2 = 0.$$

The solution is at $\mathbf{x}^* = (1,1)$. Now for this example, the Jacobian is just $\mathbf{G} = \nabla c^{\mathsf{T}} = (1,1)$, which is a vector orthogonal to the constraint. Consequently, if we choose $\mathbf{v}^{\mathsf{T}} = (-a,a)$ for some constant $a \neq 0$, the vector $\mathbf{v}$ is tangent to the constraint, which can be readily verified since $\mathbf{G}\mathbf{v} = (1)(-a)+(1)(a) = 0$. At $\mathbf{x}^*$, the gradient is a linear combination of the constraint gradients; i.e., (1.58) becomes

$$\mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} 2 = \mathbf{0}.$$

Furthermore, from (1.61), the curvature

$$\mathbf{v}^{\mathsf{T}}\mathbf{H}_L\mathbf{v} = [-a,a]\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} -a \\ a \end{bmatrix} = 4a^2$$

is clearly positive.

Notice that, at the optimal solution, the gradient vector is orthogonal to the constraint surface, or equivalently that the projection of the gradient vector onto the constraint surface is zero. A second point is also illustrated in Figure 1.5, demonstrating that the projection of the gradient is nonzero at the suboptimal point. Apparently, then, there is a matrix $\mathbf{Z}$ that projects the gradient onto the constraint surface. This implies that an equivalent form of the necessary condition (1.56) is to require that the *projected gradient* be zero, i.e.,

$$\mathbf{Z}^{\mathsf{T}}\mathbf{g} = \mathbf{0}. \tag{1.62}$$

In a similar fashion, one can define an equivalent form of the sufficient condition (1.61) in terms of the *projected Hessian* matrix

$$\mathbf{Z}^{\mathsf{T}}\mathbf{H}_L\mathbf{Z}. \tag{1.63}$$

In other words, we require the projected Hessian matrix (1.63) to be positive definite. Notice that the projection matrix $\mathbf{Z}$ has n rows and $n_d = n - m$ columns. We refer to the quantity n_d as the *number of degrees of freedom*. Observe also that the projected gradient (1.62) is a vector of length n_d and the projected Hessian (1.63) is $n_d \times n_d$. In general, the choice of $\mathbf{Z}$ is not unique, although for the example problem one can choose any scalar multiple of $\mathbf{Z}^{\mathsf{T}} = (-1, 1)$.

1.8.1 Newton's Method

Let us now apply Newton's method to find the values of $(\mathbf{x}, \boldsymbol{\lambda})$ such that the necessary conditions (1.56) and (1.57) are satisfied. Proceeding formally to construct a Taylor series expansion analogous to (1.26), the expansion about $(\mathbf{x}, \boldsymbol{\lambda})$ for the functions (1.58) and (1.59) is just

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{H}_L(\overline{\mathbf{x}} - \mathbf{x}) - \mathbf{G}^{\mathsf{T}}(\overline{\boldsymbol{\lambda}} - \boldsymbol{\lambda}), \tag{1.64}$$

$$\mathbf{0} = -\mathbf{c} - \mathbf{G}(\overline{\mathbf{x}} - \mathbf{x}). \tag{1.65}$$

After simplification, these equations lead to the linear system analogous to that given by (1.28), which is called the Kuhn–Tucker (KT) or Karush–Kuhn–Tucker (KKT) system:

$$\begin{bmatrix} \mathbf{H}_L & \mathbf{G}^{\mathsf{T}} \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{c} \end{bmatrix}, \tag{1.66}$$

where $\mathbf{p}$ is the search direction for a step $\overline{\mathbf{x}} = \mathbf{x} + \mathbf{p}$ and $\overline{\boldsymbol{\lambda}}$ is the vector of Lagrange multipliers at the new point. Notice that the system is written in terms of the change in the variables, i.e., $\mathbf{p} = \overline{\mathbf{x}} - \mathbf{x}$, but does not involve the change in the multipliers, i.e., $\overline{\boldsymbol{\lambda}} - \boldsymbol{\lambda}$. Instead, it is preferable to explicitly eliminate the term $\mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda}$, which appears in the Taylor series approximation (1.64). Thus, in this instance, Newton's method is based on a linear approximation to the constraints *and* a linear approximation to the gradients. As in the unconstrained optimization case, a linear approximation to the gradient is equivalent to making a quadratic model for the quantity being optimized. It is important to note that the quadratic approximation is made to the Lagrangian (1.55) and not just the objective function F. Because of the underlying quadratic-linear model, Newton's method will converge in one step for a quadratic objective with linear equality constraints.

Although the derivation of the KKT system (1.66) was motivated by a Taylor series expansion, an alternative motivation is to choose $\mathbf{p}$ to minimize the quadratic objective

$$\mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.67}$$

subject to the linear constraints

$$\mathbf{Gp} = -\mathbf{c}. \tag{1.68}$$

It is straightforward to demonstrate that the optimality conditions for this quadratic-linear optimization subproblem are given by the KKT system (1.66).

1.9 Inequality-Constrained Optimization

Section 1.8 introduced the equality-constrained optimization problem. In this section, we consider a problem with inequality constraints. Suppose that we want to choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.69}$$

subject to the m inequality constraints

$$\mathbf{c}(\mathbf{x}) \geq \mathbf{0}. \tag{1.70}$$

In contrast to equality-constrained problems, which require $m \leq n$, the number of inequality constraints can exceed the number of variables. A point that satisfies the constraints is said to be *feasible* and the collection of all feasible points is called the *feasible region*. Conversely, points in the infeasible region violate one or more of the constraints.

Inequality-constrained problems are characterized by a fundamental concept. Specifically, at the solution $\mathbf{x}^*$,

1. some of the constraints will be satisfied as equalities, that is,

$$c_i(\mathbf{x}^*) = 0 \qquad \text{for} \qquad i \in \mathcal{A}, \tag{1.71}$$

where $\mathcal{A}$ is called the *active set*, and

2. some constraints will be strictly satisfied, that is,

$$c_i(\mathbf{x}^*) > 0 \qquad \text{for} \qquad i \in \mathcal{A}', \tag{1.72}$$

where $\mathcal{A}'$ is called the *inactive set*.

Thus, the total set of constraints is partitioned into two subsets, namely the active and inactive constraints. Clearly, the active constraints can be treated using the methods described in Section 1.8. Obviously, an inequality-constrained optimization algorithm needs some mechanism to identify the active constraints. This mechanism is referred to as an *active set strategy*. Fortunately, there is an additional necessary condition for inequality-constrained problems that is essential to active set strategies. Specifically, at the solution, the algebraic sign of the Lagrange multipliers must be correct; that is, we must have

$$\lambda_i^* \geq 0 \qquad \text{for} \qquad i \in \mathcal{A}. \tag{1.73}$$

To illustrate the impact of inequality constraints, let us consider two examples that are modifications of Example 1.2 presented in Section 1.8.

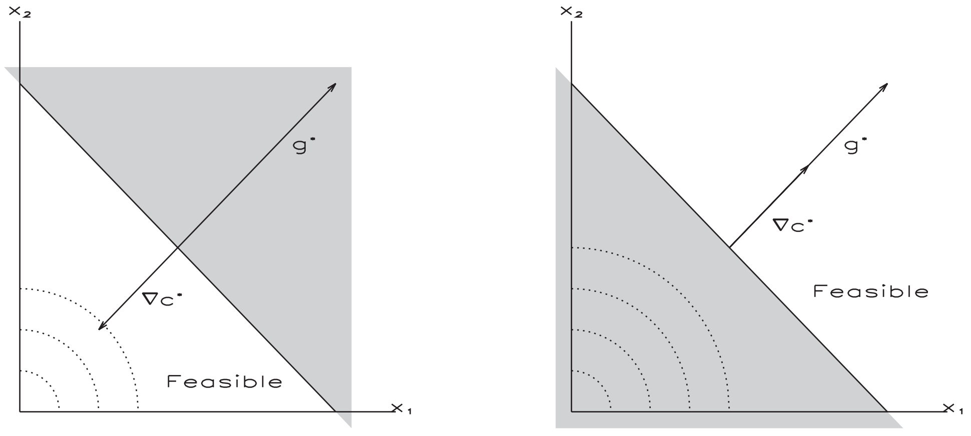

Figure 1.6. *Inequality-constrained examples.*

Example 1.3 Inequality Minimization—Inactive Constraint. The left side of Figure 1.6 illustrates the following problem: Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = 2 - x_1 - x_2 \geq 0.$$

The gradient and Jacobian are given by

$$\mathbf{g} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} -1 & -1 \end{bmatrix}.$$

Now at the point $\mathbf{x}^\mathsf{T} = (1,1)$, the optimality conditions are

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} [-2]$$

and, since $\lambda = -2 < 0$, the constraint should be deleted from the active set. The solution is $\mathbf{x}^\mathsf{T} = (0,0)$.

Example 1.4 Inequality Minimization—Active Constraint. On the other hand, if the sense of the inequality is reversed, we must minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the inequality

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

This problem is illustrated on the right side of Figure 1.6. At the point $\mathbf{x}^\mathsf{T} = (1,1)$, the Lagrange multiplier $\lambda = 2 > 0$. Consequently, the constraint is *active* and cannot be deleted. The solution is $\mathbf{x}^\mathsf{T} = (1,1)$.

1.10 Quadratic Programming

An important special form of optimization problem is referred to as the quadratic programming (QP) problem. A QP problem is characterized by

- a quadratic objective
$$F(\mathbf{x}) = \mathbf{g}^\mathsf{T}\mathbf{x} + \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x} \tag{1.74}$$
- and linear constraints
$$\mathbf{A}\mathbf{x} = \mathbf{a}, \tag{1.75}$$
$$\mathbf{B}\mathbf{x} \geq \mathbf{b}, \tag{1.76}$$

where $\mathbf{H}$ is the positive definite Hessian matrix.

Let us now outline an *active set method* for solving this QP problem using the concepts introduced in Sections 1.8 and 1.9. Assume that an estimate of the active set $\mathcal{A}^0$ is given in addition to a feasible point $\mathbf{x}^0$. A QP algorithm proceeds as follows:

1. Compute the minimum of the quadratic objective subject to the constraints in the active set estimate $\mathcal{A}$ treated as *equalities*, i.e., solve the KKT system (1.66).
2. Take the largest possible step in the direction $\mathbf{p}$ that does not violate any *inactive* inequalities, that is,
$$\overline{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}, \tag{1.77}$$
where the steplength $0 \leq \alpha \leq 1$ is chosen to maintain feasibility with respect to the inactive inequality constraints.
3. If the step is restricted, i.e., $\alpha < 1$, then
   - *add* the limiting inequality to the active set $\mathcal{A}$ and return to step 1;
   - otherwise, take the full step ($\alpha = 1$) and check the sign of the Lagrange multipliers:
     - if all of the inequalities have positive multipliers, terminate the algorithm;
     - otherwise, *delete* the inequality with the most negative λ from the active set and return to step 1.

Notice that step 1 requires the solution of the KKT system (1.66) corresponding to the set of active constraints defined by $\mathcal{A}$. In particular, (1.66) becomes

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}^\mathsf{T} & \widetilde{\mathbf{B}}^\mathsf{T} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \widetilde{\mathbf{B}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \overline{\boldsymbol{\eta}} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{a} \\ \widetilde{\mathbf{b}} \end{bmatrix}, \tag{1.78}$$

where the vector $\widetilde{\mathbf{b}}$ denotes the subset of the vector $\mathbf{b}$ corresponding to the *active* inequality constraints and $\widetilde{\mathbf{B}}$ is the corresponding Jacobian matrix for the constraints in $\mathcal{A}$. The Lagrange multipliers corresponding to the equality constraints are denoted by $\boldsymbol{\eta}$. For the purposes of this discussion, it suffices to say that any reasonable method for solving (1.78) can be used. However, in practice it is extremely important that this solution be computed in an efficient and numerically stable way. This is especially true because every time the active set changes in step 3, it is necessary to solve a different KKT system to compute the new step. This is because each time the active set changes, the matrix $\widetilde{\mathbf{B}}$ is altered by adding or deleting a row. Most efficient QP implementations compute the new step by modifying the previously computed solution and the associated matrix factorizations. In general, the computational expense of the QP problem is dominated by the linear algebra and one should *not* use a "brute force" solution to the KKT system! While space precludes a more detailed discussion of this subject, the interested reader should consult the book by Gill, Murray, and Wright [100]. An approach that is particularly efficient for large, sparse applications will be described in Section 2.3.

It should also be apparent that the number of QP iterations is determined by the initial active set $\mathcal{A}^0$. If the initial active set is correct, i.e., $\mathcal{A}^0 = \mathcal{A}^*$, then the QP algorithm will compute the solution in one iteration. Conversely, many QP iterations may be required if the initial active set differs substantially from the final one. Additional complications can arise when the Hessian matrix $\mathbf{H}$ is indefinite because the possibility of an unbounded solution exists. Furthermore, when a multiplier $\lambda_k \approx 0$, considerable care must be exercised when deleting a constraint in step 3.

As stated, the QP algorithm requires a feasible initial point $\mathbf{x}^0$, which may require a special "phase-1" procedure in the software. It is also worth noting that when $\mathbf{H} = \mathbf{0}$, the objective is linear and the optimization problem is referred to as a *linear programming* or *LP* problem. In this case, the active set algorithm described is equivalent to the *simplex method* proposed by Dantzig [67] in 1947. However, it is important to note that a *unique* solution to a linear program will have n active constraints, whereas there may be many degrees of freedom at the solution of a quadratic program.

Example 1.5 QUADRATIC PROGRAM. To illustrate how a QP algorithm works, let us consider minimizing

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraints

$$c_1(\mathbf{x}) = x_1 + x_2 - 2 \geq 0,$$
$$c_2(\mathbf{x}) = 4 - x_1 - \frac{2}{3}x_2 \geq 0.$$

This problem is illustrated in Figure 1.7. Table 1.2 summarizes the QP steps assuming the iteration begins at $\mathbf{x}^\mathsf{T} = (4,0)$ with $\mathcal{A}^0 = \{c_2\}$. The first step requires solving the KKT system (1.78) evaluated at the point $\mathbf{x}^\mathsf{T} = (4,0)$, that is,

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -\frac{2}{3} \\ -1 & -\frac{2}{3} & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 0 \end{bmatrix}.$$

After this step, the new point is at $\bar{x}_1 = x_1 + p_1 = 2.76923$, and $\bar{x}_2 = x_2 + p_2 = 1.84615$. Since the Lagrange multiplier is negative at the new point, constraint c_2 can be deleted from the active set.

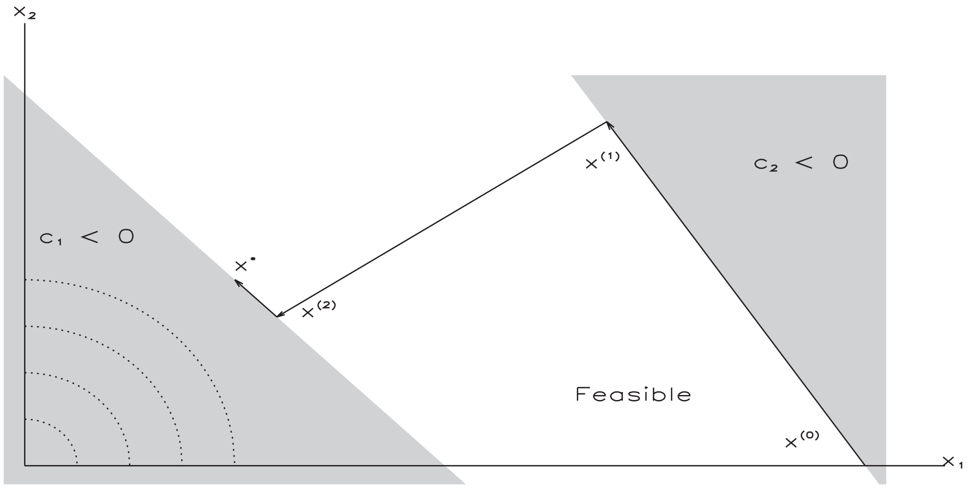

Figure 1.7. *QP example.*

Table 1.2. *QP iterations.*

Iteration	x_1	x_2	Active Set $\mathcal{A}$
0	4	0	$\mathcal{A}^0 = \{c_2\}$
1	2.76923	1.84615	Delete c_2 ($\lambda_2 = -5.53846$).
2	1.2	0.8	Add c_1 ($\alpha = 0.566667$).
3	1	1	$\mathcal{A}^* = \{c_1\}$

The second QP step begins at $\mathbf{x}^\mathsf{T} = (2.76923, 1.84615)$ with no constraints active, i.e., $\mathcal{A}^1 = \{\emptyset\}$. The new step is computed from the KKT system

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \end{bmatrix} = \begin{bmatrix} 5.53846 \\ 3.69230 \end{bmatrix}.$$

However, in this case, it is not possible to take a full step because the first constraint would be violated. Instead, one finds $\bar{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{p}$ with $\alpha = 0.566667$, to give

$$\bar{\mathbf{x}} = \begin{bmatrix} 1.2 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix} - 0.566667 \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix}.$$

In general, the length of the step is given by the simple linear prediction

$$\alpha = \min \left[\frac{-c_i(\mathbf{x})}{\mathbf{p}^\mathsf{T} \nabla c_i} \right] > 0 \qquad \text{for} \qquad i \in \mathcal{A}'. \tag{1.79}$$

Essentially, one must compute the largest step that will not violate any of the (currently) inactive inequality constraints.

Since the second QP iteration terminates by encountering the first constraint c_1, it must be added to the active set so that $\mathcal{A}^2 = \{c_1\}$. Once again the new step is computed from the KKT system, which in this case is just

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_1 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 1.6 \\ 0 \end{bmatrix}.$$

After computing the new point, the solution is given by $\mathbf{x}^* = (1,1)$, and the final active set is $\mathcal{A}^* = \{c_1\}$, with optimal Lagrange multipliers $\boldsymbol{\lambda}^* = (2,0)$.

1.11 Globalization Strategies

1.11.1 Merit Functions

To this point, the development has stressed the application of Newton's method. However, even for the simplest one-dimensional applications described in Section 1.2, Newton's method has deficiencies. Methods for correcting the difficulties with Newton's method are referred to as *globalization strategies*. There are two primary roles performed by a globalization strategy, namely,

1. detecting a problem with the (unmodified) Newton's method and
2. correcting the deficiency.

It should be emphasized that the goal of all globalization strategies is to do *nothing!* Clearly, near a solution it is desirable to retain the quadratic convergence of a Newton step and, consequently, it is imperative that modifications introduced by the globalization process not impede the ultimate behavior.

Referring to (1.28) and (1.29), all Newton iterations are based on linearizations of the form

$$\mathbf{G}\mathbf{p} = -\mathbf{c} \tag{1.80}$$

for the search direction $\mathbf{p}$, which leads to an iteration of the form

$$\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.81}$$

If Newton's method is working properly, the sequence of iterates $\{\mathbf{x}^{(k)}\}$ should converge to the solution $\mathbf{x}^*$. In practice, of course, the solution $\mathbf{x}^*$ is unknown, and we must detect whether the sequence is converging properly using information that is available. The most common way to measure progress is to assign some *merit function*, M, to each iterate $\mathbf{x}^{(k)}$ and then insist that

$$M(\mathbf{x}^{(k+1)}) < M(\mathbf{x}^{(k)}). \tag{1.82}$$

This is one approach for detecting when the Newton sequence is working. When Newton's method is used for unconstrained optimization, an obvious merit function is just $M(\mathbf{x}) = F(\mathbf{x})$. When Newton's method is used to find the root of many functions, as in (1.80), assigning a single value to quantify "progress" is no longer quite so obvious. The most commonly used merit function for nonlinear equations is

$$M(\mathbf{x}) = \frac{1}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.83}$$

If the only purpose for constructing a merit function is to quantify progress in the iterative sequence, then any other norm is also suitable; e.g., we could use

$$M(\mathbf{x}) = \|\mathbf{c}\|_1 = \sum_{i=1}^{m} |c_i|,$$

$$M(\mathbf{x}) = \|\mathbf{c}\|_2 = \left(\sum_{i=1}^{m} c_i^2\right)^{\frac{1}{2}},$$

or

$$M(\mathbf{x}) = \|\mathbf{c}\|_\infty = \max_{i=1}^{m} |c_i|.$$

Choosing a merit function for constrained optimization is still more problematic, for it is necessary to somehow balance the (often) conflicting goals of reducing the objective function while satisfying the constraints. To this point, we have motivated the discussion of a merit function as an artifice to "fix" Newton's method when it is not converging. An alternative approach is to construct some other function P, whose *unconstrained* minimum either is the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. Thus, we might consider solving a sequence of unconstrained problems whose solutions approach the desired constrained optimum. This point of view, which is fundamental to so-called penalty function methods, will be discussed in Section 1.14. One possible candidate is the quadratic penalty function

$$P(\mathbf{x}, \rho) = F(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}), \tag{1.84}$$

where ρ is called the *penalty weight* or *penalty parameter*. When this penalty function is minimized for successively larger values of the penalty weight ρ, it can be shown that the unconstrained minimizers approach the constrained solution. Unfortunately, from a computational viewpoint, this is unattractive since, as the parameter $\rho \to \infty$, the successive unconstrained optimization problems become harder and harder to solve. An alternative, which avoids this ill-conditioning, is the so-called *augmented Lagrangian function* formed by combining the Lagrangian (1.55) with a constraint penalty of the form (1.83) to yield

$$P(\mathbf{x}, \boldsymbol{\lambda}, \rho) = L(\mathbf{x}, \boldsymbol{\lambda}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.85}$$

It can be shown that the unconstrained minimum of this function is equal to the constrained minimum $\mathbf{x}^*$ for a *finite* value of the parameter ρ. Unfortunately, in practice, choosing a "good" value for the penalty weight is nontrivial. If ρ is too large, the unconstrained optimization problem is ill-conditioned and, consequently, very difficult to solve. If ρ is too small, the unconstrained minimum of the augmented Lagrangian may be unbounded and/or the problem may be ill-conditioned. On the other hand, some of these drawbacks are not as critical when the augmented Lagrangian is used only to measure progress as part of a globalization strategy. Thus, we are led to the notion of computing the Newton step $\mathbf{p}$ in the usual way and then measuring progress by insisting that

$$M(\mathbf{x}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \rho) < M(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \rho), \tag{1.86}$$

where the merit function $M(\mathbf{x}, \boldsymbol{\lambda}, \rho) = P(\mathbf{x}, \boldsymbol{\lambda}, \rho)$. In summary, a merit function can be used to quantitatively decide whether or not Newton's method is working.

1.11.2 Line-Search Methods

Assuming for the moment that a merit function is used as an indicator of progress, how does one alter Newton's method when necessary? One approach is to alter the *magnitude* of the step using a *line-search* method. A second approach is to change both the magnitude and the *direction* using a *trust-region* method. The basic notion of a line-search method is to replace the Newton step (1.81) with

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{p}. \tag{1.87}$$

Using this expression, it then follows that the merit function can be written as a function of the single variable α:

$$M(\overline{\mathbf{x}}) = M(\mathbf{x} + \alpha \mathbf{p}) = M(\alpha). \tag{1.88}$$

Furthermore, it is reasonable to choose the steplength α such that $M(\alpha)$ is approximately minimized. Most modern line-search implementations begin with the full Newton step, i.e., $\alpha^{(0)} = 1$. Then estimates are computed until a steplength $\alpha^{(k)}$ is found that satisfies a *sufficient decrease* condition given by the *Goldstein–Armijo* condition

$$0 < -\kappa_1 \alpha^{(k)} M'(0) \le M(0) - M(\alpha^{(k)}) \le -\kappa_2 \alpha^{(k)} M'(0). \tag{1.89}$$

$M'(0) = (\nabla M(0))^{\mathsf{T}} \mathbf{p}$ is the direction derivative at $\alpha = 0$, with the constants κ_1 and κ_2 satisfying $0 < \kappa_1 \le \kappa_2 < 1$. Typical values for the constants are $\kappa_1 = 10^{-4}$ and $\kappa_2 = 0.9$, and do *not* require an "accurate" minimization of M. Nevertheless, most line-search implementations are quite sophisticated and use quadratic and/or cubic polynomial interpolation in conjunction with some type of safeguarding procedure. Furthermore, special-purpose line-search procedures are often employed for different merit functions; e.g., see Murray and Wright [136]. In fact, we have already described two applications that may be viewed as special-purpose line-search algorithms. First, if a quasi-Newton update is used to construct the Hessian, an "exact" line search will produce termination after n steps on a quadratic function. Second, when solving a quadratic program, the steplength given by (1.79) is actually the minimum of the quadratic function restricted such that inactive inequalities are not violated.

Example 1.6 NEWTON METHOD WITH LINE SEARCH. To illustrate how Newton's method works when a line-search strategy is incorporated, let us consider the simple nonlinear system

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} 1 - x_1 \\ 10(x_2 - x_1^2) \end{bmatrix} = \mathbf{0}.$$

Table 1.3. *Newton's method with line search.*

Iter.	x_1	x_2	α	$M(\mathbf{x})$
0	-1.2000	1.0000	$--$	12.100
1	-1.0743	0.72336	0.57157×10^{-1}	11.425
2	-0.94553	0.47352	0.62059×10^{-1}	10.734
3	-0.81340	0.25221	0.67917×10^{-1}	10.025
4	-0.67732	0.61558×10^{-1}	0.75040×10^{-1}	9.2952
5	-0.53662	-0.95719×10^{-1}	0.83883×10^{-1}	8.5411
6	-0.39041	-0.21613	0.95147×10^{-1}	7.7580
7	-0.23751	-0.29499	0.10997	6.9398
8	-0.76248×10^{-1}	-0.32580	0.13031	6.0775
9	0.66751×10^{-1}	-0.30355	0.13287	5.1787
10	0.22101	-0.23204	0.16529	4.2483
11	0.36706	-0.11482	0.18748	3.3142
12	0.55206	0.93929×10^{-1}	0.29229	2.3230
13	1.0000	0.79935	1.0000	2.0131
14	1.0000	1.0000	1.0000	0.12658×10^{-18}

Table 1.4. *Newton's method without line search.*

Iter.	x_1	x_2	α	$M(\mathbf{x})$
0	-1.2000	1.0000	$--$	12.100
1	1.0000	-3.8400	1.0000	1171.3
2	1.0000	1.0000	1.0000	0.85927×10^{-14}

If the merit function $M(\mathbf{x})$ (1.83) is used to monitor progress and the steplength α is adjusted using a line search with polynomial interpolation, one obtains the iteration history summarized in Table 1.3. Notice that every iteration produces a reduction in the merit function as it must. However, in order to achieve this monotonic behavior, it is necessary to modify the steplength on nearly all iterations. Quadratic convergence is observed only during the final few iterations.

It is interesting to compare the behavior of Newton's method, which has a line-search globalization procedure, to that of an unmodified Newton algorithm. Table 1.4 summarizes the iterations when Newton's method is applied *without* a line search. Notice that all steps have $\alpha = 1$. Furthermore, observe that the first step produced a large increase in the constraint error as measured by the merit function M. Nevertheless, in this case, the unmodified Newton method was significantly more efficient than the approach with a line search. For this example, the line search is not only unnecessary but also inefficient. This suggests two questions. First, do we need a globalization strategy? Clearly, the answer is yes. Second, is a line search with merit function the most efficient strategy? Perhaps not. The numerical results suggest that there is considerable room for improvement provided a mechanism can be devised that is sufficiently robust.

1.11.3 Trust-Region Methods

A line-search globalization strategy computes the search direction $\mathbf{p}$ and then adjusts the steplength parameter α in order to define the iteration (1.87). A second alternative is to adjust both the magnitude and direction of the vector $\mathbf{p}$. Treating the current point $\mathbf{x}$ as fixed, suppose that we want to choose the variables $\mathbf{p}$ to minimize

$$F(\mathbf{x}+\mathbf{p}) \approx F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.90}$$

subject to the constraint

$$\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p} \le \delta^2. \tag{1.91}$$

We assume that we can *trust* the prediction $F(\mathbf{x}+\mathbf{p})$ as long as the points lie within the region defined by the *trust radius* δ. Proceeding formally to define the Lagrangian for this problem, one obtains

$$L_T(\mathbf{p},\tau) = F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} - \tau\left[\delta^2 - \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p}\right], \tag{1.92}$$

where τ is the Lagrange multiplier associated with the trust-region inequality constraint. Now the corresponding necessary condition is just

$$\nabla_p L_T(\mathbf{p},\tau) = \mathbf{g} + \mathbf{H}\mathbf{p} + \tau\mathbf{p} = \mathbf{g} + [\mathbf{H} + \tau\mathbf{I}]\,\mathbf{p} = \mathbf{0}. \tag{1.93}$$

There are two possible solutions to this problem. If the trust-radius constraint is inactive, then $\tau = 0$ and the search direction $\mathbf{p}$ is just the unmodified Newton direction. On the other hand, if the trust-radius constraint is active, $\|\mathbf{p}\| = \delta$ and the multiplier $\tau > 0$. In fact, the trust radius δ and the multiplier τ are implicitly (and inversely) related to each other—when δ is large, τ is small, and vice versa. Traditional trust-region methods maintain a current estimate of the trust radius δ and adjust the parameter τ such that the constraint $\|\mathbf{p}\| = \delta$ is approximately satisfied. An alternative is to view the parameter τ as a way to construct a modified Hessian matrix $\mathbf{H} + \tau\mathbf{I}$. This interpretation was suggested by Levenberg [131] for nonlinear least squares (NLS) problems, and we shall often refer to τ as the *Levenberg parameter*. Notice that as τ becomes large, the search direction approaches the gradient direction. Table 1.5 summarizes the limiting behavior of these quantities.

Table 1.5. *Trust-region behavior.*

$\tau \to \infty$	$\tau \to 0$
$\delta \to 0$	$\delta \to \infty$
$\mathbf{p} \propto -\mathbf{g}$	$\mathbf{p} \to -\mathbf{H}^{-1}\mathbf{g}$
$\|\mathbf{p}\| \to 0$	$\|\mathbf{p}\| \to \|\mathbf{H}^{-1}\mathbf{g}\|$

Although the trust-region approach has been introduced as a globalization technique for an unconstrained optimization problem, the basic idea is far more general. The trust-region concepts can be extended to constrained optimization problems as well as, obviously, to least squares and root-solving applications. Other definitions of the trust region itself using alternative norms have been suggested. Furthermore, constrained applications have been formulated with more than one trust region to reflect the conflicting aims of constraint satisfaction and objective function reduction. Finally, it should be emphasized that trust-region concepts are often used in conjunction with other globalization ideas. For example, a common way to modify the trust radius from one iteration to the next is to compare the values of the predicted and actual reduction in a suitable merit function.

1.11.4 Filters

When solving a constrained optimization problem, it is necessary to introduce some quantitative method for deciding that a Newton iterate is working. One approach for achieving this is to introduce a merit function that combines the conflicting goals of constraint satisfaction and objective function reduction. However, merit functions must explicitly assign some relative weight to each conflicting goal, and choosing the correct penalty weight is often difficult. This dilemma was illustrated by the results summarized in Tables 1.3 and 1.4. Recently, Fletcher and Leyffer [84] have introduced an approach that will accept a Newton step if *either* the objective function *or* the constraint violation is decreased. The filter approach recognizes that there are two conflicting aims in nonlinear programming. The first is to minimize the objective function, that is, choose $\mathbf{x}$ to minimize

$$F(\mathbf{x}). \tag{1.94}$$

The second is to choose $\mathbf{x}$ to minimize the constraint violation

$$v[\mathbf{c}(\mathbf{x})], \tag{1.95}$$

where the *violation* can be measured using any suitable norm. Fletcher and Leyffer define $v\,[\mathbf{c}(\mathbf{x})] \equiv \|\widetilde{\mathbf{c}}\|_1$, where $\tilde{c}_k = \min(0, c_k)$, for inequalities of the form $c_k \geq 0$. The basic idea is to compare information from the current iteration to information from previous iterates and then "filter" out the bad iterates.

To formalize this, denote the values of the objective and constraint violation at the point $\mathbf{x}^{(k)}$ by

$$\{F^{(k)}, v^{(k)}\} \equiv \{F(\mathbf{x}^{(k)}), v[\mathbf{c}(\mathbf{x}^{(k)})]\}. \tag{1.96}$$

When comparing the information at two different points $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(j)}$, a pair $\{F^{(k)}, v^{(k)}\}$ is said to *dominate* another pair $\{F^{(j)}, v^{(j)}\}$ if and only if both $F^{(k)} \leq F^{(j)}$ and $v^{(k)} \leq v^{(j)}$. Using this definition, we can then define a *filter* as a list of pairs

$$\mathcal{F} = \begin{bmatrix} F^{(1)}, v^{(1)} \\ F^{(2)}, v^{(2)} \\ \vdots \\ F^{(K)}, v^{(K)} \end{bmatrix} \tag{1.97}$$

such that no pair dominates any other. A new point $\{F^{(\ell)}, v^{(\ell)}\}$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter.

Example 1.7 FILTER GLOBALIZATION. To illustrate how this procedure works, let us revisit Example 1.6, posed as the following optimization problem: Minimize

$$F(\mathbf{x}) = (1 - x_1)^2$$

subject to

$$\mathbf{c}(\mathbf{x}) = 10(x_2 - x_1^2) = 0.$$

Table 1.6 summarizes the behavior of a filter algorithm using the same sequence of iterates as in Table 1.4. The problem is illustrated in Figure 1.8. The iteration begins at $\mathbf{x}^\mathsf{T} = (-1.2, 1)$, and the first filter entry is $(F^{(1)}, v^{(1)}) = (4.84, 4.4)$. The dark shaded region in Figure 1.8 defines points that would be rejected by the filter because both the objective function and constraint violation would be worse than the first filter entry values. The second iteration at the point $\mathbf{x}^\mathsf{T} = (1, -3.84)$ is *accepted* by the filter because the objective function is better even though the constraint violation is worse. This point is added to the filter, thereby defining a new “unacceptable” region, which is the union of the regions defined by each point in the filter (i.e., the dark and light shaded regions). The final iterate does not violate the regions defined by either of the filter entries and is accepted as the solution.

Observe that the filter provides only a mechanism for accepting or rejecting an iterate. It does not suggest how to correct the step if a point is rejected by the filter. Fletcher

Table 1.6. *Filter iteration summary.*

Iter.	x_1	x_2	$F(\mathbf{x})$	$v[\mathbf{c}(\mathbf{x})]$
0	−1.2000	1.0000	4.84	4.4
1	1.0000	−3.8400	0	48.4
2	1.0000	1.0000	0	0

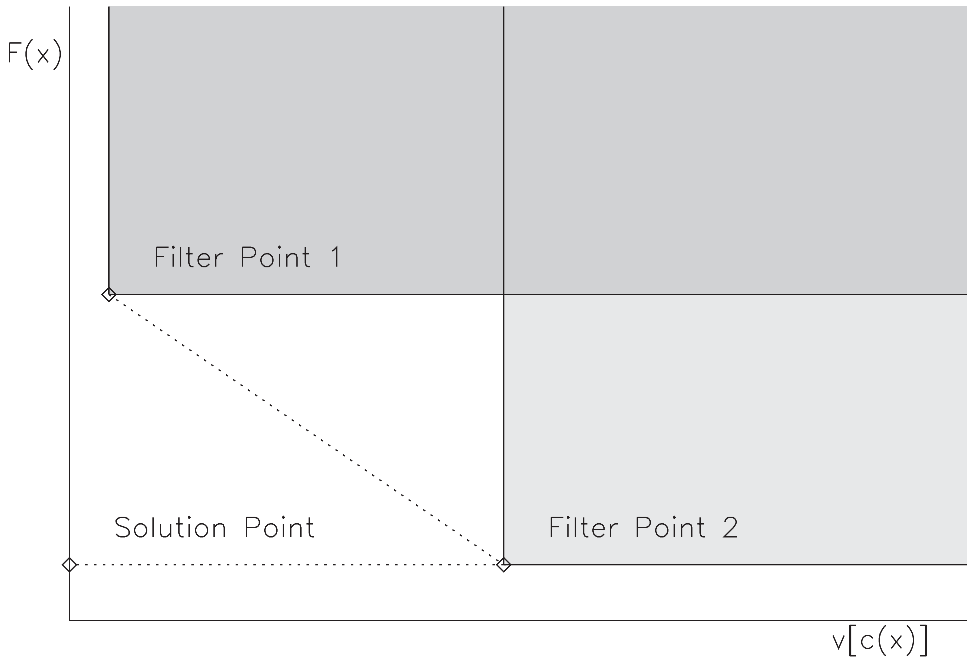

Figure 1.8. *NLP filter.*

and Leyffer use the filter in conjunction with a trust-region approach. On the other hand, a line-search technique that simply reduces the steplength is also an acceptable method for correcting the iterate. Practical implementation of the filter mechanism also must preclude a sequence of points that becomes unbounded in either $F(\mathbf{x})$ or $v\,[\mathbf{c}(\mathbf{x})]$. A generous overestimate of the upper bound on $F(\mathbf{x})$ can be included as an additional "northwest corner" entry in the filter. Similarly, an absolute upper limit on the constraint violation can be included as a "southeast corner" entry in the filter. Computation of these extreme values is readily accommodated by including the value of the largest Lagrange multiplier for each iterate in the list of saved information. The amount of information saved in the filter list is usually rather small because, when a new entry $\{F^{(\ell)}, v^{(\ell)}\}$ is added, all points dominated by the new entry are deleted from the filter.

1.12 Nonlinear Programming

The general nonlinear programming (NLP) problem can be stated as follows: Find the n-vector $\mathbf{x}^{\mathsf{T}} = (x_1, \ldots, x_n)$ to *minimize* the scalar objective function

$$F(\mathbf{x}) \tag{1.98}$$

subject to the m constraints

$$\mathbf{c}_L \le \mathbf{c}(\mathbf{x}) \le \mathbf{c}_U \tag{1.99}$$

and the simple bounds

$$\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U. \tag{1.100}$$

Equality constraints can be imposed by setting $\mathbf{c}_L = \mathbf{c}_U$.

The KKT necessary conditions for $\mathbf{x}^*$ to be an optimal point require that

- $\mathbf{x}^*$ be feasible, i.e., (1.99) and (1.100) are satisfied;
- the Lagrange multipliers $\boldsymbol{\lambda}$ corresponding to (1.99) and $\boldsymbol{\nu}$ corresponding to (1.100) satisfy
$$\mathbf{g} = \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} + \boldsymbol{\nu}; \tag{1.101}$$
- the Lagrange multipliers for the inequalities be
$$\begin{cases} \text{nonpositive} & \text{for active upper bounds,} \\ \text{zero} & \text{for strictly satisfied constraints,} \\ \text{nonnegative} & \text{for active lower bounds;} \end{cases}$$
- the Jacobian $\widetilde{\mathbf{G}}$ corresponding to the active constraints have full row rank (the *constraint qualification test*).

The constraint qualification test implies that the gradients of the active constraints are linearly independent, and as such is often referred to as *linear independence constraint qualification* and abbreviated as LIQC. A weaker condition than LIQC is referred to as *Mangasarian–Fromovitz constraint qualification* or MFQC. In the simplest setting with only inequalities $\mathbf{c}(\mathbf{x}) \ge \mathbf{0}$, it ensures the existence of a direction along which, to first order, all active constraints strictly increase, i.e. there exists a vector $\mathbf{p}$ such that $\widetilde{\mathbf{G}}\mathbf{p} > \mathbf{0}$.

1.13 An SQP Algorithm

Let us now outline the basic steps of a sequential quadratic programming or SQP method for solving the general NLP problem. SQP methods are among the most widely used algorithms for solving general NLPs, and there are many variations of the basic approach. The method described is implemented in the $\mathbb{SOCS}$ [38] software and is similar to the algorithm employed by the NPSOL [96] software. For additional information on SQP methods, see, for example, Fletcher [82] and Gill, Murray, and Wright [99].

The basic approach described in Section 1.8 was to introduce a quadratic approximation to the Lagrangian and a linear approximation to the constraints. This development led to (1.67) and (1.68). An identical approach is followed here. Assume that the iteration begins at the point $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$. The fundamental idea is to solve the following QP subproblem:

Compute $\mathbf{p}$ to minimize a quadratic approximation to the Lagrangian

$$\mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.102}$$

subject to the linear approximation to the constraints

$$\mathbf{b}_L \le \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \le \mathbf{b}_U \tag{1.103}$$

with bound vectors defined by

$$\mathbf{b}_L = \begin{bmatrix} \mathbf{c}_L - \mathbf{c} \\ \mathbf{x}_L - \mathbf{x} \end{bmatrix}, \qquad \mathbf{b}_U = \begin{bmatrix} \mathbf{c}_U - \mathbf{c} \\ \mathbf{x}_U - \mathbf{x} \end{bmatrix}. \tag{1.104}$$

The solution of this QP subproblem defines more than just the search direction $\mathbf{p}$. In particular, the quadratic program determines an estimate of the active set of constraints and an estimate of the corresponding Lagrange multipliers $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$. Thus, it is possible to define search directions for the multipliers

$$\Delta\boldsymbol{\lambda} \equiv \widehat{\boldsymbol{\lambda}} - \boldsymbol{\lambda}, \tag{1.105}$$

$$\Delta\boldsymbol{\nu} \equiv \widehat{\boldsymbol{\nu}} - \boldsymbol{\nu}. \tag{1.106}$$

Finally, the QP solution can be used to construct information needed to compute a merit function. A linear prediction for the value of the constraints is given by

$$\bar{\mathbf{s}} \equiv \mathbf{G}\mathbf{p} + \mathbf{c}. \tag{1.107}$$

Assuming we begin the iteration with an estimate for the *slack variables* $\mathbf{s}$, then it follows that

$$\Delta\mathbf{s} \equiv \bar{\mathbf{s}} - \mathbf{s} = \mathbf{G}\mathbf{p} + (\mathbf{c} - \mathbf{s}). \tag{1.108}$$

The term $(\mathbf{c} - \mathbf{s})$ has intentionally been isolated in this expression because it measures the "deviation from linearity" in the constraints $\mathbf{c}$. Notice that if the constraints $\mathbf{c}(\mathbf{x})$ are linear functions of the variables $\mathbf{x}$, then the term $(\mathbf{c} - \mathbf{s}) = \mathbf{0}$. Now the augmented search direction formed by collecting these expressions is given by

$$\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\boldsymbol{\lambda}} \\ \bar{\boldsymbol{\nu}} \\ \bar{\mathbf{s}} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \mathbf{s} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{p} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\nu} \\ \Delta\mathbf{s} \end{bmatrix}. \tag{1.109}$$

As before, the scalar α defines the steplength. Observe that when a full Newton step is taken ($\alpha = 1$), the new NLP Lagrange multipliers are just the QP multipliers, i.e., $\overline{\boldsymbol{\lambda}} = \widehat{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}} = \widehat{\boldsymbol{\nu}}$.

Like all Newton methods, it is necessary to incorporate some type of globalization strategy. To this end, Gill et al. [95] suggest a modified form of the augmented Lagrangian merit function (1.85) given by

$$M(\mathbf{x},\boldsymbol{\lambda},\mathbf{s}) = F - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{c}-\mathbf{s}) + \frac{1}{2}(\mathbf{c}-\mathbf{s})^{\mathsf{T}}\boldsymbol{\Theta}(\mathbf{c}-\mathbf{s}), \tag{1.110}$$

where $\boldsymbol{\Theta}$ is a diagonal matrix of penalty weights. They also prove convergence for an SQP algorithm using this merit function, provided the SQP subproblem has a solution. This requires bounds on the derivatives and condition number of the Hessian matrix. Thus, the steplength α must be chosen to satisfy a sufficient decrease condition of the form (1.89). Most robust implementations incorporate a safeguarded line-search algorithm using quadratic and/or cubic polynomial interpolation. In addition, it is necessary that the penalty weights $\boldsymbol{\Theta}$ be chosen such that

$$M'(0) \leq -\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}. \tag{1.111}$$

As a practical matter, this single condition does not uniquely define all of the penalty weights in the matrix $\boldsymbol{\Theta}$, and so a minimum norm estimate is used. Estimates for the slack variables $\mathbf{s}$ are also needed to construct the slack direction (1.108), and these quantities can be computed by minimizing M as a function of $\mathbf{s}$ before taking the step. We postpone details of these calculations to the next chapter.

Example 1.8 SQP METHOD. To illustrate the behavior of an SQP method, let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 + \log(x_1 x_2) \tag{1.112}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 x_2 \geq 1 \tag{1.113}$$

with bounds

$$0 \leq x_1 \leq 10, \tag{1.114}$$

$$0 \leq x_2 \leq 10. \tag{1.115}$$

Assume that $\mathbf{x}^{(0)} = (0.5, 2)^{\mathsf{T}}$ is an initial guess for the solution.

Figure 1.9 illustrates the iteration history when the SQP algorithm implemented in $\mathbb{SOCS}$ is applied to this problem. Notice that it is not possible to evaluate the objective function at the first predicted point $(1,0)$. This is considered a "function error" by the $\mathbb{SOCS}$ software, and the response in the line search is to reduce the steplength. In this case, $\alpha = 9.04543273 \times 10^{-2}$ produces the point $(0.545227, 1.81909)$. Subsequent steps proceed to the solution at $(1,1)$.

1.14 Interior-Point Methods

Section 1.11 described using a penalty function as part of globalization strategy for Newton's method. The fundamental idea is to construct a function whose *unconstrained* minimum either is the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. Let

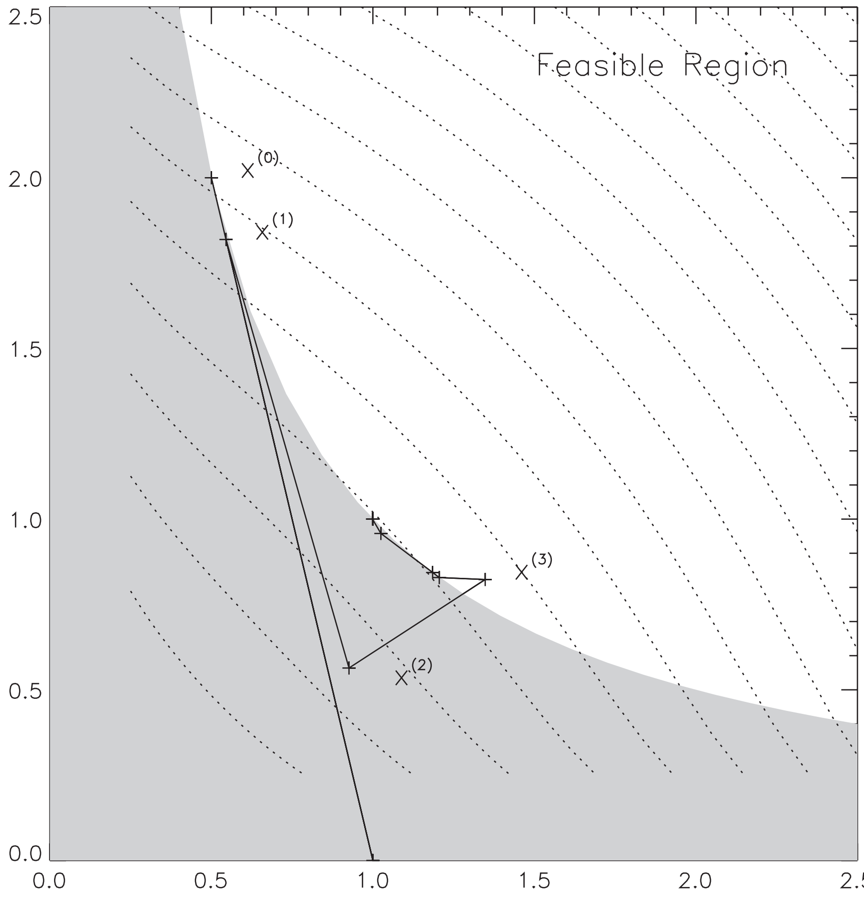

Figure 1.9. *NLP example.*

us focus on a particular penalty function method referred to interchangeably as a *barrier method* or *interior-point method*.

In order to illustrate the primary features let us first consider a simple example proposed by Powell [146]. For this example in two variables (x_1, x_2), the objective is to minimize x_2 subject to the constraints

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) \geq -1 \tag{1.116}$$

for $k = 1, 2, \ldots, m$. A solution is at $(0, -1)$ and the suggested initial guess is $\mathbf{x}^{(0)} = (.8, .5)$. This is a linear programming problem since both the objective function and the constraints are linear functions of the variables.

To understand how an optimization algorithm would behave on a problem like this it is instructive to consider a physical analogy. One can view the constraints as "fences," and the objective function as a plane surface in two dimensions. The path followed by a

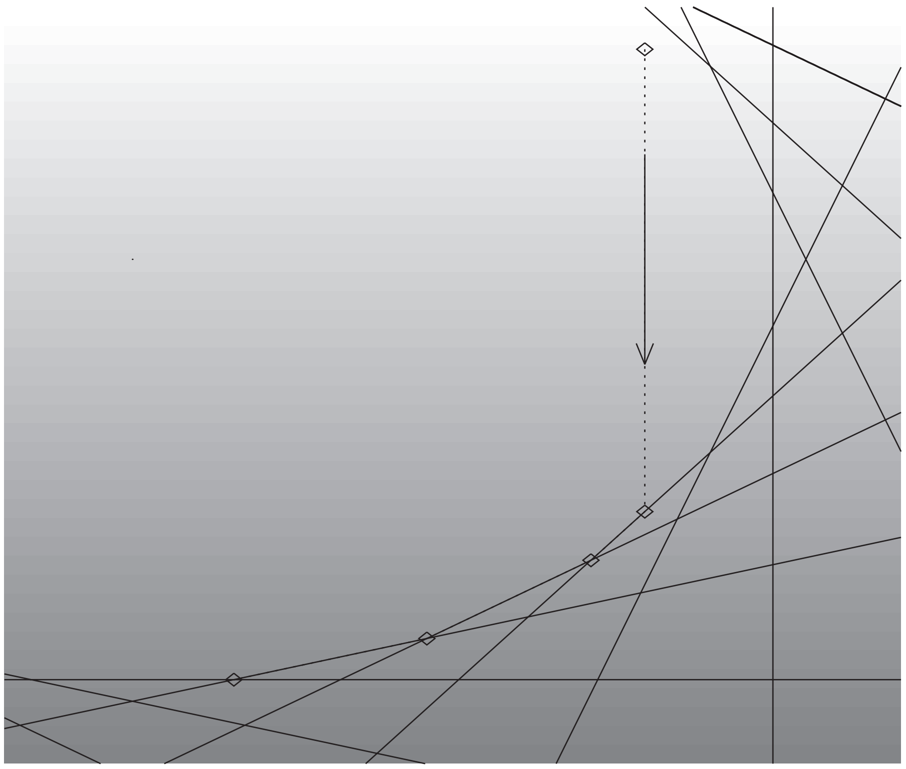

Figure 1.10. *Ball rolling downhill inside fences.*

ball rolling downhill inside the fenced region is illustrated in Figure 1.10. It is clear that the ball will roll downhill until it encounters a fence, and then it will follow one fence until it encounters another. Ultimately the ball will stop at the lowest point within the fenced region and will lie on one of the boundaries. Viewed mathematically, at the solution some of the constraints will be strictly feasible, i.e.,

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) > -1,$$

and the remainder will be satisfied as equalities, i.e.,

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) = -1.$$

Constraints that are strictly feasible are said to be *inactive* (cf. (1.72)) and the others are said to be *active* (cf. (1.72)). An obvious computational hurdle is to correctly identify which constraints are active and which are inactive. In this physical example at each fence "corner" the active set changes. Clearly the number of constraints m will alter the difficulty of this linear programming problem.

The *active set method* discussed in Section 1.10 solves a sequence of equality constrained problems. In contrast, penalty function methods replace the constrained optimization problem by a *sequence* of unconstrained problems, and as such were referred to as *sequential unconstrained minimization techniques* (SUMT) by Fiacco and McCormick [79]. Thus, we can minimize the function

$$\beta(\mathbf{x},\mu) = x_2 - \mu \sum_{k=1}^{m} \ln b_k(\mathbf{x}), \tag{1.117}$$

where

$$b_k(\mathbf{x}) = x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) + 1. \tag{1.118}$$

Notice that the term $\ln b_k(\mathbf{x})$ becomes large if the constraints $b_k(\mathbf{x})$ are near zero, and consequently serves as a "barrier" to the minimization. Thus, when minimizing $\beta(\mathbf{x},\mu)$ iterates will be driven away from the constraint boundary at $b_k(\mathbf{x}) = 0$. The influence of the barrier term is controlled by the parameter μ called the *barrier parameter*, and the function $\beta(\mathbf{x},\mu)$ is called the log-barrier or simply *barrier function*.

How should the barrier parameter be chosen? Since we would like to approximate the original constrained solution we would like to have the barrier parameter μ be "small." Figure 1.11 illustrates the active set method when compared to the barrier method when the value of $\mu = 10^{-3}$. There are two important observations here. First, the path for the barrier method and the active set method are nearly the same. Thus, by starting with a small value for μ we will replicate the undesirable properties of an active set method. Second, while the "downhill" direction is well defined for the barrier objective, there is little information horizontally. In fact, since the contours are nearly parallel to the x_1-axis almost any value for x_1 will yield the same objective. In other words, the unconstrained barrier objective is nearly singular!

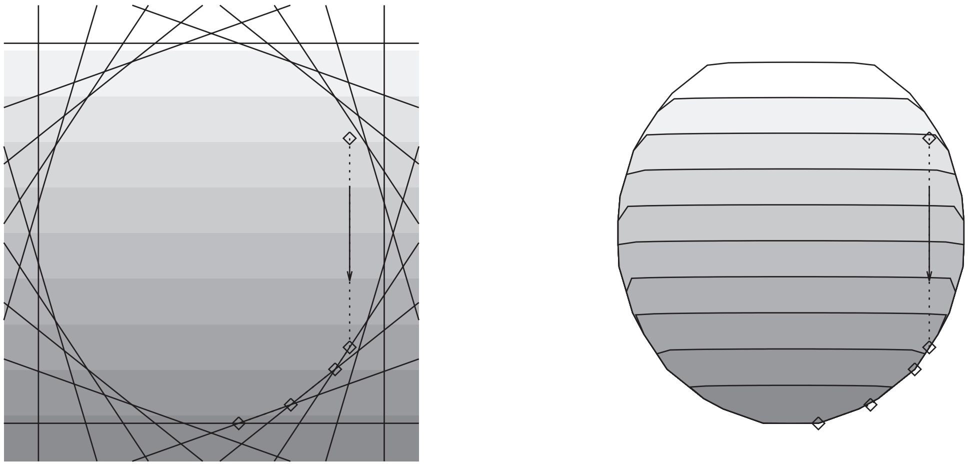

Figure 1.11. *Active set versus barrier method with small μ.*

On the other hand suppose we choose a "large" value for μ and then minimize the barrier function. The solution to this problem serves as a good initial guess for a second

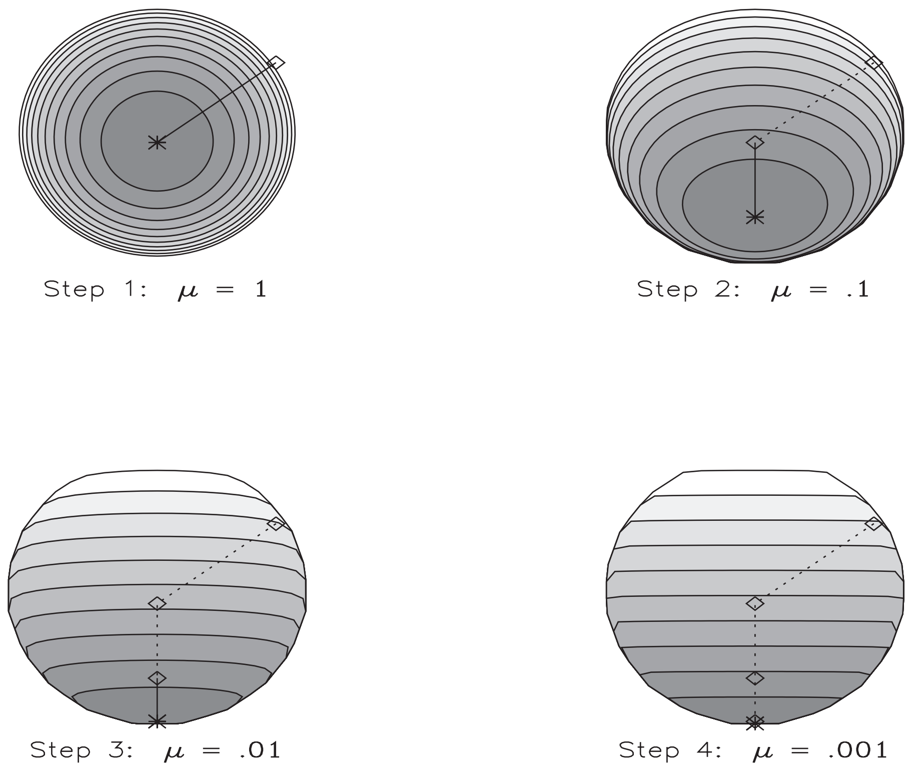

Figure 1.12. *Barrier method steps.*

minimization with a smaller value of μ. Figure 1.12 illustrates the contours of the log-barrier function for this example when $m = 20$. The contours are plotted for four different values of the barrier parameter, namely $\mu = 1, .1, .01, .001$. The figure also illustrates the steps taken as we progress from one barrier minimizer to another. Notice when the barrier parameter is large ($\mu = 1$) the contours of the log-barrier function are almost circular about the minimum point, whereas they become progressively more severe as μ is reduced. What is more important is that the sequence of iterates tends to approach the solution along a path that is "orthogonal" to the constraints instead of "tangential" to them. This observation illustrates the barrier parameter reduction strategy. We first minimize the barrier function with a "large" value of μ. The solution to this problem then serves as an initial guess for another barrier minimization with a smaller value of μ. In fact, by solving a sequence of unconstrained problems, it can be shown that the unconstrained solution points approach the constrained solution as μ goes to zero. Furthermore, notice that the unconstrained minimizers in Figure 1.12 all lie inside the feasible region; hence the method is called an *interior-point algorithm.* For obvious reasons the trajectory followed by the iterates is referred to as the *central path* and is illustrated in Figure 1.13.

It should be noted that the logarithmic barrier function (1.117) is not the only choice for a penalty function. For example, Fiacco and McCormick [79] also suggest another

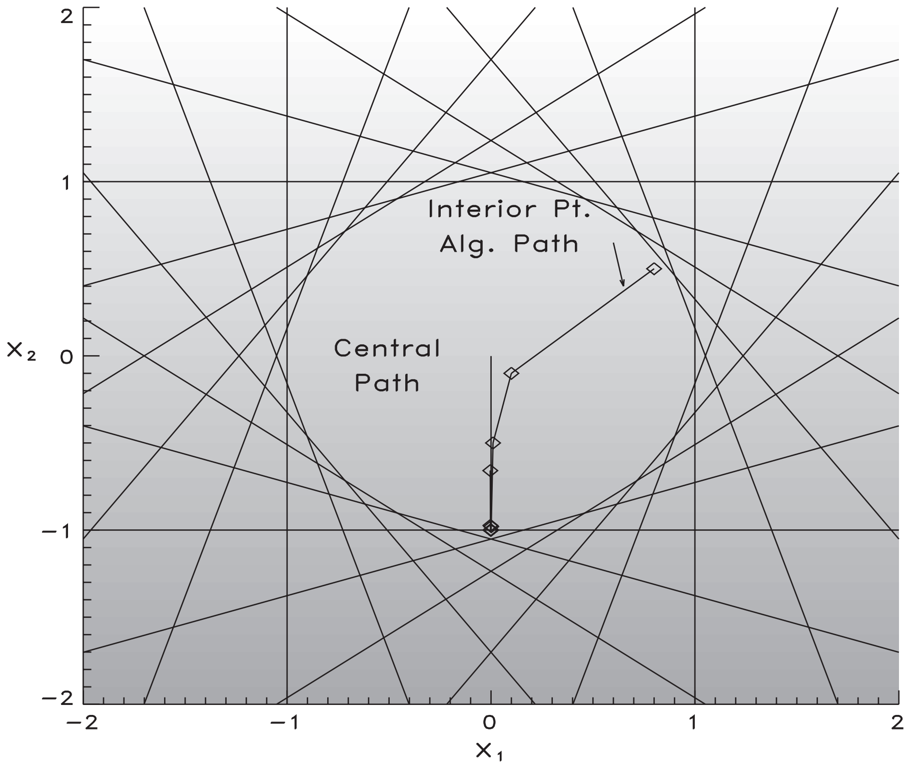

Figure 1.13. *Interior-point algorithm.*

interior-point penalty function

$$P(\mathbf{x},\mu) = F(\mathbf{x}) + \mu^2 \sum_{k=1}^{m} \frac{1}{c_k(\mathbf{x})} \tag{1.119}$$

for treating inequality constraints $c_k(\mathbf{x}) \geq 0$. Furthermore, the quadratic penalty function introduced in (1.84) can be extended to inequality constraints

$$P(\mathbf{x},\mu) = F(\mathbf{x}) + \frac{1}{2\mu} \sum_{k=1}^{m} \{\min[0, c_k(\mathbf{x})]\}^2 . \tag{1.120}$$

When (1.120) is minimized for a sequence of penalty weights $\mu \to 0$, the unconstrained minimizers approach the solution from outside the feasible region, and thus it is referred to as an *exterior point penalty function*. Unfortunately, the Hessian matrix is both discontinuous and ill-conditioned as $\mu \to 0$, and consequently the computational effectiveness is significantly degraded.

Early implementations of interior-point methods applied standard unconstrained minimization techniques to the barrier objective (1.117). Unfortunately, there are a number of subtle issues that can significantly degrade the computational performance of a naive barrier implementation. In particular, some care must be exercised when computing the iterates to avoid ill-conditioning for small values of μ, and an approach that accurately solves the underlying linear equations will be presented in Section 2.11. For efficiency we do not accurately minimize the barrier function before reducing μ and robustness is enhanced by a number of globalization techniques.

1.15 Mathematical Program with Complementarity Conditions

Many practical problems require modeling disjunctive behavior. Suppose, we have a model that is represented by three sets of variables **x**, **y**, and **z**. Furthermore suppose some of the variables (**x** and **y**) have a complementarity relationship such that either one or both must be at its bound. Formally, we can state this complementarity relationship by requiring that

$$\begin{array}{ccc} x_k = 0 & OR & y_k = 0, \\ \mathbf{x} \geq \mathbf{0}, & & \mathbf{y} \geq \mathbf{0} \end{array}$$

for all k variables in the set. This complementarity relationship is usually denoted

$$\mathbf{0} \leq \mathbf{x} \perp \mathbf{y} \geq \mathbf{0}. \tag{1.121}$$

There are several equivalent ways to impose the complementarity relationship:

$$\mathbf{x}^{\mathsf{T}}\mathbf{y} = 0, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \tag{1.122a}$$

$$x_k y_k = 0 \quad \forall k, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \tag{1.122b}$$

$$x_k y_k \leq 0 \quad \forall k, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}. \tag{1.122c}$$

A *mathematical program with complementarity conditions (MPCC)* requires finding the variables $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to minimize

$$F(\mathbf{x}, \mathbf{y}, \mathbf{z}) \tag{1.123a}$$

subject to the constraints

$$\mathbf{b}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{0}, \tag{1.123b}$$

$$\mathbf{c}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \geq \mathbf{0}, \tag{1.123c}$$

$$\mathbf{0} \leq \mathbf{x} \perp \mathbf{y} \geq \mathbf{0}. \tag{1.123d}$$

Now, an important property of the MPCC (1.123a)–(1.123d) is that it can be viewed as a *bilevel optimization problem*, that is, an optimization problem within an optimization problem. Specifically the solution to the MPCC (1.123a)–(1.123d) is equivalent to finding the variables $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to minimize

$$F(\mathbf{x}, \mathbf{y}, \mathbf{z}) \tag{1.124a}$$

subject to the constraints

$$\mathbf{b}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{0}, \tag{1.124b}$$

$$\mathbf{c}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \geq \mathbf{0}, \tag{1.124c}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{1.124d}$$

$$\mathbf{y} = \arg\left\{ \min_{\widehat{\mathbf{y}}} \mathbf{x}^{\mathsf{T}}\widehat{\mathbf{y}} \quad : \quad \widehat{\mathbf{y}} \geq \mathbf{0} \right\}. \tag{1.124e}$$

Observe that (1.124e) requires solving the inner level optimization problem for the variables $\widehat{\mathbf{y}}$ that minimize

$$\mathbf{x}^{\mathsf{T}}\widehat{\mathbf{y}} \tag{1.125a}$$

subject to the constraints

$$\widehat{\mathbf{y}} \geq \mathbf{0} \tag{1.125b}$$

treating $\mathbf{x}$ as fixed. In other words the variables $\widehat{\mathbf{y}}$ are constrained to satisfy the optimality conditions for the inner level optimization problem. When viewed in this manner the KKT optimality conditions for the inner level problem are referred to as equilibrium conditions, and the overall problem is referred to as a *mathematical program with equilibrium constraints (MPEC)*. Observe that the inner level problem (1.125a)–(1.125b) in this development is just a linear programming problem in the variables $\widehat{\mathbf{y}}$. More general forms for the inner level problem are possible leading to different equilibrium conditions. Consequently these problems are often referred to as mathematical programs with *variational inequalities*.

1.15.1 The Signum or Sign Operator

Suppose the problem functions involve the signum or sign function given by

$$sgn[f(x)] = \begin{cases} 1 & \text{if } f(x) > 0, \\ -1 & \text{if } f(x) < 0, \\ 0 & \text{if } f(x) = 0. \end{cases} \tag{1.126}$$

This disjunctive behavior can be modeled by introducing an additional variable y, where

$$y = sgn[f(x)]. \tag{1.127}$$

It then follows that for a fixed value of x one can compute y to minimize the objective

$$-f(x)y \tag{1.128a}$$

subject to the constraints

$$-1 \leq y \leq 1. \tag{1.128b}$$

Since (1.128a)–(1.128b) is just an NLP in the single variable y, the Lagrangian (1.55) is

$$L(y) = -f(x)y - p(y+1) - q(1-y), \tag{1.129a}$$

where p and q are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$-f(x) - p + q = 0, \tag{1.129b}$$

$$p(y+1) = 0, \tag{1.129c}$$

$$q(1-y) = 0, \tag{1.129d}$$

$$p \geq 0, \tag{1.129e}$$

$$q \geq 0. \tag{1.129f}$$

The first optimality condition (1.129b) defines a stationary point of the Lagrangian $\nabla_y L = 0$, and the complementarity conditions are given by (1.129c) and (1.129d). Observe that the complementarity conditions (1.129c) and (1.129d), which correspond to (1.122b), can also be written as

$$0 \le p \perp (y+1) \ge 0, \tag{1.130}$$

$$0 \le q \perp (1-y) \ge 0. \tag{1.131}$$

1.15.2 The Absolute Value Operator

Suppose the problem functions involve the absolute value operation $|f(x)|$. To model this discontinuous behavior one can introduce two variables where the variable

$$z = |f(x)| = f(x)y \tag{1.132}$$

and the second variable $y = sgn[f(x)]$ is chosen as before to minimize the objective

$$-f(x)y \tag{1.133a}$$

subject to the constraints

$$-1 \le y \le 1. \tag{1.133b}$$

After using (1.129b)–(1.129f) and eliminating the variable y one obtains

$$z = p + q, \tag{1.133c}$$

$$f(x) = p - q, \tag{1.133d}$$

$$0 \le p \perp q \ge 0. \tag{1.133e}$$

1.15.3 The Maximum Value Operator

When the problem functions involve the maximum value operation $\max[f(x), a]$ for some constant a, the discontinuous behavior can be modeled by defining the variable

$$z = \max[f(x), a] = f(x) + [a - f(x)]y, \tag{1.134}$$

where the second variable y is chosen to minimize the objective

$$[f(x) - a]y \tag{1.135a}$$

subject to the constraints

$$0 \le y \le 1. \tag{1.135b}$$

As before (1.135a)–(1.135b) is an NLP in the single variable y, with the Lagrangian given by

$$L(y) = [f(x) - a]y - py - q(1-y), \tag{1.136a}$$

where p and q are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$[f(x) - a] - p + q = 0, \tag{1.136b}$$

$$py = 0, \tag{1.136c}$$

$$q(1-y) = 0, \tag{1.136d}$$

$$p \ge 0, \tag{1.136e}$$

$$q \ge 0. \tag{1.136f}$$

Here the complementarity conditions can be written as

$$0 \le p \perp y \ge 0,$$
$$0 \le q \perp (1-y) \ge 0.$$

1.15.4 The Minimum Value Operator

Similarly when problem functions involve the minimum value operation $\min[f(x),a]$ for some constant a, the discontinuous behavior can be modeled by writing the variable

$$z = \min[f(x),a] = f(x) + [f(x)-a]y, \tag{1.137}$$

where the second variable y is chosen to minimize the objective

$$[f(x)-a]y \tag{1.138a}$$

subject to the constraints

$$-1 \le y \le 0. \tag{1.138b}$$

In this case the Lagrangian is given by

$$L(y) = [f(x)-a]y - p(y+1) - q(-y), \tag{1.139a}$$

where p and q are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$[f(x)-a] - p + q = 0, \tag{1.139b}$$
$$p(y+1) = 0, \tag{1.139c}$$
$$q(-y) = 0, \tag{1.139d}$$
$$p \ge 0, \tag{1.139e}$$
$$q \ge 0. \tag{1.139f}$$

Here the complementarity conditions can be written as

$$0 \le p \perp (1+y) \ge 0,$$
$$0 \le q \perp (-y) \ge 0.$$

1.15.5 Solving an MPEC

The examples presented in the previous sections are all directed toward a single goal—the formulation of an MPEC as a standard NLP. The key notion is to explicitly specify the necessary conditions for the lower level optimization problem as constraints for the "outer" NLP. For example when dealing with the signum function, three new variables were introduced, namely (y,p,q), in addition to the constraints (1.129b)–(1.129f). At first, it might appear that with appropriate reformulation an MPEC can be treated as a standard NLP, using any standard algorithm. Unfortunately, the NLP constraints violate the linear independence constraint qualification (LIQC) test, as well as the weaker Mangasarian–Fromovitz constraint qualification (MFQC). A number of different strategies have been investigated to deal with this lack of regularity (cf. [7]). Since the constraint Jacobian is rank deficient and detecting rank deficiency is problematic, the solution schemes share a common theme, which is to solve a "slightly" perturbed problem. Thus one can modify the form of (1.122a) to $\mathbf{x}^\mathsf{T}\mathbf{y} = \epsilon$, or replace (1.122b) with $x_k y_k = \epsilon$, and similarly for (1.122b) enforce $x_k y_k \leq \epsilon$. An effective alternative is to move the complementarity condition from the constraints into the NLP objective using an exact penalty function.

1.16 What Can Go Wrong

The material in this chapter has been presented to give the reader an understanding of how a method *should* work in practice. The discussion is designed to help users efficiently use high-quality optimization software to solve practical problems. However, in practice, things do go wrong! In this section, we describe a number of common difficulties that can be encountered and suggest remedial action to correct the deficiencies. For ease of presentation, it is convenient to discuss the difficulties individually. Of course, real applications may involve more than a single difficulty and the user must be prepared to correct all problems before obtaining satisfactory performance from optimization software.

1.16.1 Infeasible Constraints

One of the most common difficulties encountered occurs when the NLP problem has infeasible constraints. To be more precise, infeasible constraints have *no* solution.

Example 1.9 INFEASIBLE CONSTRAINTS. For example, the following constraints have no feasible region:

$$\begin{aligned} c_1(\mathbf{x}) &= x_1^2 x_2 - 1 \geq 0, \\ c_2(\mathbf{x}) &= x_2 \leq -\tfrac{1}{10}. \end{aligned} \tag{1.140}$$

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- the QP subproblem has no solution, which can occur if the linearized constraints have no solution;
- many NLP iterations produce very little progress;
- the penalty parameters become very large, or the barrier parameter becomes small;
- the condition number of the projected Hessian matrix becomes large; or
- the Lagrange multipliers become large.

Although most robust software will attempt to "gracefully" detect this situation, ultimately the only remedy is to reformulate the problem!

1.16.2 Rank-Deficient Constraints

In contrast to the previous situation, it is possible that the constraints are consistent; that is, they do have a solution. However, at the solution, the Jacobian matrix may be either ill-conditioned or rank deficient.

Example 1.10 RANK-DEFICIENT CONSTRAINTS. For an example originally suggested by Kuhn and Tucker [126], consider minimizing

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 \tag{1.141}$$

subject to the constraints

$$\begin{aligned} c_1(\mathbf{x}) &= x_1 \geq 0, \\ c_2(\mathbf{x}) &= x_2 \geq 0, \\ c_3(\mathbf{x}) &= (1 - x_1)^3 - x_2 \geq 0. \end{aligned} \tag{1.142}$$

The solution to this problem is $\mathbf{x}^* = (1,0)^\mathsf{T}$ and, clearly, $\mathbf{c}(\mathbf{x}^*) = \mathbf{0}$. However, at the solution, the active set is $\mathcal{A}^* = \{c_2, c_3\}$ and the Jacobian matrix corresponding to these constraints is rank deficient, i.e.,

$$\widetilde{\mathbf{G}} = \begin{bmatrix} 0 & 1 \\ -3(1-x_1)^2 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}. \tag{1.143}$$

This example violates the linear independent constraint qualification test.

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- many NLP iterations produce very little progress;
- the penalty parameters become very large or the barrier parameter becomes very small;
- the Lagrange multipliers become large; or
- the rank deficiency in $\widetilde{\mathbf{G}}$ is detected.

Unfortunately, detecting rank deficiency in the Jacobian is not a straightforward numerical task! Consequently, it is quite common to confuse this difficulty with inconsistent constraints. Again, the remedy is to reformulate the problem!

1.16.3 Constraint Redundancy

A third type of difficulty can occur when the problem formulation contains constraints that are redundant. Thus, although the constraints have a solution, they may be unnecessary to the problem formulation. The following two examples illustrate the situation.

Example 1.11 REDUNDANCY (FULL-RANK). Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 \tag{1.144}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 - x_2 = 0. \tag{1.145}$$

In this case, the unconstrained solution is $\mathbf{x}^* = (0,0)^\mathsf{T}$, and the constraint is trivially satisfied. Constraint c_1 can be eliminated from the formulation without changing the answer.

Example 1.12 REDUNDANCY (RANK-DEFICIENT). Minimize

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 + x_3^2 \tag{1.146}$$

subject to the constraints

$$\begin{aligned} c_1(\mathbf{x}) &= x_1 + x_2 - 1 &= 0, \\ c_2(\mathbf{x}) &= 2x_1 + 2x_2 - 2 &= 0. \end{aligned} \tag{1.147}$$

In this case, constraint $c_2 = 2c_1$ and, clearly, one of the constraints is redundant. Note also that the Jacobian matrix $\widetilde{\mathbf{G}}$ is rank deficient.

Symptoms indicating redundancy such as in Example 1.11 include

- Lagrange multipliers near zero and
- difficulty detecting the active set.

On the other hand, constraint redundancy such as in Example 1.12 is likely to exhibit symptoms similar to the rank-deficient cases discussed previously. Obviously, the remedy is to reformulate the problem and eliminate the redundant constraints!

1.16.4 Discontinuities

Perhaps the single biggest obstacle encountered in the practical application of NLP methods is the presence of discontinuous behavior. All of the numerical methods described assume that the objective and constraint functions are continuous and differentiable. When discontinuities are present in the problem functions, the standard quadratic/linear models that form the basis for most NLP methods are no longer appropriate. In fact, the KKT necessary conditions do not apply!

Unfortunately, in practice, there are many common examples of discontinuous behavior. Typical sources include

- branching caused by IF tests in code;
- absolute value, max, and min functions;
- linear interpolation of tabular data; and
- "internal" iteration loops, e.g., adaptive quadrature, root finding.

The most common symptoms of discontinuous functions are

- slow convergence or divergence,
- small steps ($\alpha \approx 0$) in the line search, and
- possible ill-conditioning of the Hessian matrix.

Example 1.13 TREATING ABSOLUTE VALUES.

Bad Formulation	*Good Formulation*
Find (x, y) to minimize $$\lvert x\rvert$$ subject to $$y - x^2 = 1.$$	Find (x_1, x_2, y) to minimize $$x_1 + x_2$$ subject to $$\begin{aligned} y-(x_1-x_2)^2 &= 1, \\ x_1 &\geq 0, \\ x_2 &\geq 0. \end{aligned}$$

The "trick" motivated by the MPEC formulation (1.133c)–(1.133e) is to replace $x \rightarrow (x_1 - x_2)$ and then observe that $|x| \rightarrow (x_1 + x_2)$. The optimal solution for the preferred formulation is $\mathbf{x}^* = (0, 0, 1)$. As expected, the preferred formulation is solved by $\mathbb{SOCS}$ in 47 evaluations. In contrast, the original formulation requires 117 function evaluations and terminates with a small step warning. A more realistic illustration of this difficulty will be presented in Example 6.17. Although complementarity is not explicitly enforced in this example in general it may be necessary to augment the objective with a term $\rho x_1 x_2$ to ensure that $0 \leq x_1 \perp x_2 \geq 0$.

Example 1.14 MINIMAX PROBLEMS. Minimizing maximum values (*minimax problems*).

Bad Formulation	*Good Formulation*
Find (x_1, x_2) to minimize $$\max_k \lvert y_k - d_k\rvert$$ for $k = 1, 2, 3$, where $$\begin{aligned} y_k &= x_1 + x_2 k, \\ d &= (1, 1.5, 1.2). \end{aligned}$$	Find (x_1, x_2, s) to minimize $$s$$ for $k = 1, 2, 3$, where $$\begin{aligned} y_k - d_k - s &\leq 0, \\ y_k - d_k + s &\geq 0. \end{aligned}$$

The trick here is to introduce the slack variable s as an absolute bound on the quantities being minimized, i.e., $|y_k - d_k| \leq s$. The absolute value function is then eliminated since $-s \leq y_k - d_k \leq s$ can be rewritten as two inequality constraints. The optimal solution for the preferred formulation is $\mathbf{x}^* = (1.1, 0.1, 0.2)$. As expected, the preferred formulation is solved by $\mathbb{SOCS}$ in 30 evaluations. In contrast, the original formulation terminates with a small step warning after 271 evaluations at the point $\mathbf{x}^* = (1.09963, 0.100148)$.

In subsequent chapters, a great deal of attention will be given to the correct formulation of optimal control applications such that discontinuous behavior can be avoided. In fact, Section 4.12 is devoted entirely to one such application. Nevertheless, it is worth

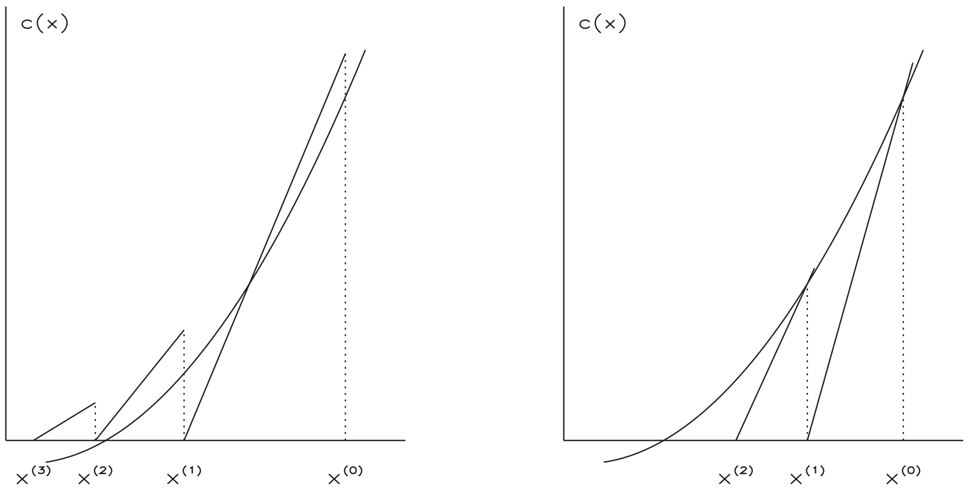

Figure 1.14. *The effects of noise.*

mentioning that specific remedial action for these problems includes

- recoding and/or reformulating to eliminate branching, absolute value, max, and min problems (cf. Examples 1.13 and 1.14);
- interpolation or approximation of tabular data using functions with continuity through second derivatives (cf. Example 6.4);
- avoiding the use of variable-step, variable-order integration when optimizing; and
- reformulation of "internal" iterations (e.g., Kepler's equation) using additional "external" NLP constraints (cf. Example 6.6).

Because discontinuous behavior plays such a major role in the success or failure of an NLP application, it is worthwhile to understand the effects of "noise" on the Newton iteration. Figure 1.14 illustrates a typical Newton iteration in the presence of two types of noise. Let us assume that we are trying to solve a constraint $c(x) = 0$. However, because of errors in the function evaluation, what the Newton method "sees" is the contaminated result $c(x) + \epsilon$. Because the intent is to drive the constraint to zero, ultimately the value will be "swamped" by the noise ϵ and we expect that the iteration will *not* converge! This situation is shown in the left portion of Figure 1.14. On the other hand, suppose that the value of the constraint is correct but the slope is contaminated by noise as illustrated in the right portion of Figure 1.14. Thus, instead of computing the true slope $c'(x)$, the Newton iterate actually uses the value $c'(x) + \epsilon$. In this case, we can expect the iteration to locate a solution, but at a degraded *rate* of convergence. We can carry this analysis further by viewing optimization as equivalent to solving $\nabla_x L = 0$. Then, by analogy, if the gradient is contaminated by noise, $\nabla_x L + \epsilon$, the iteration will not converge, because ultimately the true gradient will be dominated by the noise. On the other hand, if the Hessian information is noisy, $\nabla^2_{xx} L + \epsilon$, we expect the *rate* of convergence to be degraded. It is worth noting that an approximation to the slope (e.g., secant or quasi-Newton) in the absence of noise changes the Newton iterates in a similar fashion.

1.16.5 Scaling

Scaling affects everything! Poor scaling can make a good algorithm bad. Scaling changes the convergence rate, termination tests, and numerical conditioning. The most common way to scale a problem is to introduce variable scaling of the form

$$\tilde{x}_k = u_k x_k + r_k \tag{1.148}$$

for $k = 1, \dots, n$. In this expression, the scaled variables $\tilde{x}_k$ are related to the unscaled variables by the variable scale weights u_k and shifts r_k. In like fashion, the objective and constraints are commonly scaled using

$$\tilde{c}_k = w_k c_k, \tag{1.149}$$

$$\tilde{F} = w_0 F \tag{1.150}$$

for $k = 1, \dots, m$. The intent is to let the optimization algorithm work with the "well-scaled" quantities $\tilde{x}$, $\tilde{c}$, and $\tilde{F}$ in order to improve the performance of the algorithm. Unfortunately, it is not at all clear what it means to be well scaled!

Nevertheless, conventional wisdom suggests that some attempt should be made to construct a well-scaled problem and, consequently, we give the following *hints* (*not rules*) for good scaling:

- normalize the independent variables to have the same "range," e.g.,
$$0 \le \tilde{x}_k \le 1;$$
- normalize the dependent functions to have the same magnitude, i.e.,
$$F \approx c_1 \approx c_2 \approx \cdots \approx c_m \approx 1;$$
- normalize the rows and columns of the Jacobian to be of the same magnitude;
- scale the dependent functions so that the Lagrange multipliers are one:
$$|\lambda_1| \approx |\lambda_2| \approx \cdots \approx |\lambda_m| \approx 1;$$
- scale the problem so that the condition number of the projected Hessian matrix is close to one;
- scale the problem so that the condition number of the KKT matrix (1.66) is close to one.

Constructing an automatic computational procedure that will meet all of these goals is probably not possible. Furthermore, even if a strategy could be devised that produces "good scaling" at one point, say $\mathbf{x}$ for nonlinear functions, this may be "bad scaling" at another point $\bar{\mathbf{x}}$. As a practical matter in the $\mathbb{SOCS}$ software, we attempt to produce a well-scaled Jacobian at the user-supplied initial point, but also allow the user to override this scaling by input. Details of the approach are deferred to Section 4.8. For more helpful discussion on this subject, the reader is referred to [99, Chapter 8].

1.16.6 Nonunique Solution

For most of the difficulties presented an optimization algorithm will either fail to produce a solution and/or struggle to find one. But what if the problem formulation has too many solutions?

Example 1.15 NONUNIQUE SOLUTION. Minimize

$$F(\mathbf{x}) = (x_1 - x_2)^2. \tag{1.151}$$

In this case, there are an infinite number of solutions corresponding to points on the line $x_1 = x_2$. Everywhere on this line the objective function $F = 0$ and the gradient $\mathbf{g} = \mathbf{0}$; however, clearly the sufficient condition (1.61) is violated.

The projected Hessian matrix must be positive definite only in the neighborhood of the solution, and even a well-posed problem may encounter an indefinite or negative definite projected Hessian during intermediate iterations. Consequently most computational algorithms try to "fix" the problem by making the Hessian approximation positive definite. Using a positive definite quasi-Newton update such as the BFGS (1.50) and/or a trust-region strategy (1.93) are two such "fixes." Unfortunately when the algorithm uses this strategy it may converge with no indication of trouble. However, if the iterations are repeated with a different initial guess, it is quite likely that a different solution will be computed. In fact, there may be no obvious symptoms of difficulty reported by the software unless the user happens to notice nonrepeatability in the solution! Detecting the difficulty and then correcting it requires user insight. Example 6.11 illustrates this situation.

1.17 Derivative Approximation by Finite Differences

Derivative information is an important ingredient in all optimization algorithms, and consequently it is useful to review one of the most prevalent methods for computing these quantities. Consider the Taylor series expansion about the point x

$$f(x+\delta) = f(x) + f'(x)\delta + \frac{1}{2} f''(x)\delta^2 + \cdots, \tag{1.152}$$

where the perturbation size $\delta > 0$. Dividing through by δ yields

$$\begin{aligned} f'(x) &= \frac{f(x+\delta) - f(x)}{\delta} - \frac{\delta}{2} f''(x)\ldots \\ &= \frac{f(x+\delta) - f(x)}{\delta} + \mathcal{O}(\delta). \end{aligned} \tag{1.153}$$

The terms denoted by $\mathcal{O}(\delta)$ are said to be of *order* δ. If the $\mathcal{O}(\delta)$ terms are ignored, one obtains the *forward difference approximation*

$$f'(x) \approx \frac{f(x+\delta) - f(x)}{\delta}. \tag{1.154}$$

Because the Taylor series expansion is truncated after the first term, it is common to refer to

$$\eta_T = \mathcal{O}(\delta) \approx \frac{\delta}{2} |f''(x)| \tag{1.155}$$

as the *truncation error*. An estimate for the truncation error is provided by the magnitude of the first ignored term in the expansion, $\frac{\delta}{2}|f''(x)|$. This is considered a *first order* approximation, because δ appears to the first power, and in general an approximation is said to have order k if δ^k appears in the truncation error. On a digital computer the difference approximation (1.154) must be evaluated using finite precision arithmetic, which introduces a second source of error. It is referred to as roundoff or *cancellation error* and can be approximated by

$$\eta'_1 = \frac{2\epsilon_a}{\delta}, \tag{1.156}$$

where ϵ_a is an estimate of the absolute function precision. For the sake of reference a number of common finite difference approximations are summarized in Table 1.7.

Table 1.7. *Finite difference derivative approximations.*

First Order (Forward Difference) First Derivative: ($\eta'_1 = \frac{2\epsilon_a}{\delta}$)

$$f' = \frac{1}{\delta}\left[f(x+\delta) - f(x)\right]. \tag{1.157}$$

Second Order (Central Difference) First Derivative: ($\eta'_2 = \frac{\epsilon_a}{\delta}$)

$$f' = \frac{1}{2\delta}\left[f(x+\delta) - f(x-\delta)\right]. \tag{1.158}$$

Fourth Order First Derivative: ($\eta'_4 = \frac{3\epsilon_a}{2\delta}$)

$$f' = \frac{1}{12\delta}\left[-f(x+2\delta) + 8f(x+\delta) - 8f(x-\delta) + f(x-2\delta)\right]. \tag{1.159}$$

Second Order Second Derivative: ($\eta''_2 = \frac{4\epsilon_a}{\delta^2}$)

$$f'' = \frac{1}{\delta^2}\left[f(x+\delta) - 2f(x) + f(x-\delta)\right]. \tag{1.160}$$

Fourth Order Second Derivative: ($\eta''_4 = \frac{16\epsilon_a}{3\delta^2}$)

$$f'' = \frac{1}{12\delta^2}\left[-f(x+2\delta) + 16f(x+\delta) - 30f(x) + 16f(x-\delta) - f(x-2\delta)\right]. \tag{1.161}$$

Second Order Third Derivative: ($\eta'''_2 = \frac{3\epsilon_a}{\delta^3}$)

$$f''' = \frac{1}{2\delta^3}\left[-f(x-2\delta) + 2f(x-\delta) - 2f(x+\delta) + f(x+2\delta)\right]. \tag{1.162}$$

A reasonable choice for the perturbation size δ can be computed by using the finite difference error estimates. An estimate for the total finite difference error in the case of a forward difference first derivative is

$$\eta = \eta_T + \eta'_1 = \frac{\delta}{2}|f''(x)| + \frac{2\epsilon_a}{\delta}. \tag{1.163}$$

This error is minimized by choosing

$$\delta_1^* = 2\sqrt{\frac{\epsilon_a}{|f''(x)|}}. \tag{1.164}$$

In a similar fashion the error in a central difference first derivative is approximated by

$$\eta = \frac{\delta^2}{6}|f'''(x)| + \frac{\epsilon_a}{\delta}, \tag{1.165}$$

which can be minimized by setting

$$\delta_2^* = \left(\frac{3\epsilon_a}{|f'''(x)|}\right)^{\frac{1}{3}}. \tag{1.166}$$

At first blush, making use of these expressions in a computational algorithm appears problematic. If the goal is to estimate a first derivative using the forward difference formula (1.157), according to (1.164) an estimate for the second derivative $f''(x)$ is also needed! Similarly if the goal is to construct a central difference first derivative using (1.158), one needs a third derivative in order to minimize the error according to (1.166). Furthermore these estimates change as the point x changes. Fortunately for most practical applications a "good" estimate for the perturbation size usually suffices. It is reasonable to compute the optimal perturbation size only at a "typical" value for x rather than at every optimization iterate. In the $\mathbb{SOCS}$ software, the perturbation size is estimated at the initial iterate $x^{(0)}$, and thereafter held fixed for subsequent iterations unless there is some other indication of suspicious behavior.

Suppose we are trying to compute a central difference first derivative using (1.158). How does one compute the third derivative $f'''(x)$ needed in (1.166)? Fortunately a finite difference approximation to $f'''(x)$ is given by (1.162) for some perturbation size, say h. This estimate requires evaluating the function at four perturbed points in addition to the nominal. Now, the third derivative approximation has error in it which is related to the trial perturbation h. So, after evaluating $f'''(x)$ using the trial perturbation, a new estimate for the trial perturbation can be computed. Typically, this process is repeated only once or twice, before an acceptable accuracy is obtained in the finite difference third derivative $f'''(x)$. Finally, the third derivative can be used to construct the optimal central difference perturbation from (1.166). Gill, Murray, and Wright [99, Chapter 8] describe an algorithm for constructing forward difference perturbation size estimates, which can be extended to accommodate central difference estimates. It should be emphasized that one first computes the third derivative using a sequence of trial perturbations, and only then is an estimate for the first derivative perturbation (1.166) constructed. As described, the procedure is applicable for computing derivatives of a single function with respect to a single variable. Extending the approach to many variables, that is, for constructing partial derivatives, is trivial and will be discussed in Section 2.2. In contrast, when differentiating many functions using the same finite difference perturbation, there may well be some loss of accuracy in the approximate derivatives.

1.17.1 Difference Estimates in Differential Equations

In addition to their use within a nonlinear optimization algorithm, finite difference derivative estimates may appear in another context. Suppose the function $f(x)$ represents a physical quantity defined in the region $x_L \le x \le x_U$. The mathematical description for a physical process typically involves derivatives, e.g., $f'(x)$. One way to treat this situation is to

Table 1.8. *Difference estimates in differential equations.*

Second Order (Three Point) First Derivative			
f_1'	$=$	$\frac{1}{2\delta}\left[-3f_1+4f_2-f_3\right]$,	
f_k'	$=$	$\frac{1}{2\delta}\left[f_{k+1}-f_{k-1}\right]$,	$k=2,\ldots,(n-1)$,
f_n'	$=$	$\frac{1}{2\delta}\left[f_{n-2}-4f_{n-1}+3f_n\right]$.	
Fourth Order (Five Point) First Derivative			
f_1'	$=$	$\frac{1}{12\delta}\left[-25f_1+48f_2-36f_3+16f_4-3f_5\right]$,	
f_2'	$=$	$\frac{1}{12\delta}\left[-3f_1-10f_2+18f_3-6f_4+f_5\right]$,	
f_k'	$=$	$\frac{1}{12\delta}\left[-f_{k+2}+8f_{k+1}-8f_{k-1}+f_{k-2}\right]$,	$k=3,\ldots,(n-2)$,
f_{n-1}'	$=$	$\frac{1}{12\delta}\left[-f_{n-4}+6f_{n-3}-18f_{n-2}+10f_{n-1}+3f_n\right]$,	
f_n'	$=$	$\frac{1}{12\delta}\left[+3f_{n-4}-16f_{n-3}+36f_{n-2}-48f_{n-1}+25f_n\right]$.	
Second Order (Three Point) Second Derivative			
f_1''	$=$	$\frac{1}{\delta^2}\left[f_1-2f_2+f_3\right]$,	
f_k''	$=$	$\frac{1}{\delta^2}\left[f_{k-1}-2f_k+f_{k+1}\right]$,	$k=2,\ldots,(n-1)$,
f_n''	$=$	$\frac{1}{\delta^2}\left[f_{n-2}-2f_{n-1}+f_n\right]$.	
Fourth Order (Five Point) Second Derivative			
f_1''	$=$	$\frac{1}{12\delta^2}\left[45f_1-154f_2+214f_3-156f_4+61f_5-10f_6\right]$,	
f_2''	$=$	$\frac{1}{12\delta^2}\left[10f_1-15f_2-4f_3+14f_4-6f_5+f_6\right]$,	
f_k''	$=$	$\frac{1}{12\delta^2}\left[-f_{k-2}+16f_{k-1}-30f_k+16f_{k+1}-f_{k+2}\right]$,	$k=3,\ldots,(n-2)$,
f_{n-1}''	$=$	$\frac{1}{12\delta^2}\left[f_{n-5}-6f_{n-4}+14f_{n-3}-4f_{n-2}-15f_{n-1}+10f_n\right]$,	
f_n''	$=$	$\frac{1}{12\delta^2}\left[-10f_{n-5}+61f_{n-4}-156f_{n-3}+214f_{n-2}-154f_{n-1}+45f_n\right]$.	

introduce a spatial discretization with n "lines," i.e.,

$$x_k=(k-1)\delta+x_L, \qquad \delta=\frac{x_U-x_L}{n-1}, \qquad k=1,\ldots,n. \tag{1.167}$$

If we denote the function values at these points by $f_k=f(x_k)$, then it is tempting to use the difference estimates given in Table 1.7 with one slight modification. Specifically one must construct derivative estimates at the endpoints without utilizing values outside of the region. For example a naive application of the central difference approximation

$$f'(x_1)=\frac{1}{2\delta}\left[f_2-f_0\right] \tag{1.168}$$

requires the value $f_0=f(x_L-\delta)$. This is not desirable since the physical process may not be defined outside the region. To account for this, the difference approximations must be altered to utilize information strictly interior to the region $x_L\le x\le x_U$. It is straightforward to construct a polynomial interpolant for the nearest *interior* points $x_L\le x_k\le x_U$ and then differentiate the interpolating polynomial. For the sake of reference Table 1.8 summarizes the more commonly used formulas.