CHAPTER 9

Derivative-Free Optimization

Many practical applications require the optimization of functions whose derivatives are not available. Problems of this kind can be solved, in principle, by approximating the gradient (and possibly the Hessian) using finite differences (see Chapter 8), and using these approximate gradients within the algorithms described in earlier chapters. Even though this finite-difference approach is effective in some applications, it cannot be regarded a general-purpose technique for derivative-free optimization because the number of function evaluations required can be excessive and the approach can be unreliable in the presence of noise. (For the purposes of this chapter we define *noise* to be inaccuracy in the function evaluation.) Because of these shortcomings, various algorithms have been developed that do not attempt to approximate the gradient. Rather, they use the function values at a set of sample points to determine a new iterate by some other means.

Derivative-free optimization (DFO) algorithms differ in the way they use the sampled function values to determine the new iterate. One class of methods constructs a linear or quadratic model of the objective function and defines the next iterate by seeking to minimize this model inside a trust region. We pay particular attention to these model-based approaches because they are related to the unconstrained minimization methods described in earlier chapters. Other widely used DFO methods include the simplex-reflection method of Nelder and Mead, pattern-search methods, conjugate-direction methods, and simulated annealing. In this chapter we briefly discuss these methods, with the exception of simulated annealing, which is a nondeterministic approach and has little in common with the other techniques discussed in this book.

Derivative-free optimization methods are not as well developed as gradient-based methods; current algorithms are effective only for small problems. Although most DFO methods have been adapted to handle simple types of constraints, such as bounds, the efficient treatment of general constraints is still the subject of investigation. Consequently, we limit our discussion to the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x). \tag{9.1}$$

Problems in which derivatives are not available arise often in practice. The evaluation of $f(x)$ can, for example, be the result of an experimental measurement or a stochastic simulation, with the underlying analytic form of f unknown. Even if the objective function f is known in analytic form, coding its derivatives may be time consuming or impractical. Automatic differentiation tools (Chapter 8) may not be applicable if $f(x)$ is provided only in the form of binary computer code. Even when the source code is available, these tools cannot be applied if the code is written in a combination of languages.

Methods for derivative-free optimization are often used (with mixed success) to minimize problems with nondifferentiable functions or to try to locate the global minimizer of a function. Since we do not treat nonsmooth optimization or global optimization in this book, we will restrict our attention to smooth problems in which f has a continuous derivative. We do, however, discuss the effects of noise in Sections 9.1 and 9.6.

9.1 FINITE DIFFERENCES AND NOISE

As mentioned above, an obvious DFO approach is to estimate the gradient by using finite differences and then employ a gradient-based method. This approach is sometimes successful and should always be considered, but the finite-difference estimates can be inaccurate when the objective function contains noise. We quantify the effect of noise in this section.

Noise can arise in function evaluations for various reasons. If $f(x)$ depends on a stochastic simulation, there will be a random error in the evaluated function because of the finite number of trials in the simulation. When a differential equation solver or some other complex numerical procedure is needed to calculate f, small but nonzero error tolerances that are used during the calculations will produce noise in the value of f.

In many applications, then, the objective function f has the form

$$f(x) = h(x) + \phi(x), \tag{9.2}$$

where h is a smooth function and ϕ represents the noise. Note that we have written ϕ to be a function of x but in practice it need not be. For instance, if the evaluation of f depends on a simulation, the value of ϕ will generally differ at each evaluation, even at the same x. The form (9.2) is, however, useful for illustrating some of the difficulties caused by noise in gradient estimates and for developing algorithms for derivative-free optimization.

Given a difference interval ϵ, recall that the centered finite-difference approximation (8.7) to the gradient of f at x is defined as follows:

$$\nabla_\epsilon f(x) = \left[\frac{f(x+\epsilon e_i) - f(x-\epsilon e_i)}{2\epsilon}\right]_{i=1,2,\dots,n}, \tag{9.3}$$

where e_i is the ith unit vector (the vector whose only nonzero element is a 1 in the ith position). We wish to relate $\nabla_\epsilon f(x)$ to the gradient of the underlying smooth function $h(x)$, as a function of ϵ and the noise level. For this purpose we define the noise level η to be the largest value of ϕ in a box of edge length 2ϵ centered at x, that is,

$$\eta(x;\epsilon) = \sup_{\|z-x\|_\infty \le \epsilon} |\phi(z)|. \tag{9.4}$$

By applying to the central difference formula (9.3) the argument that led to (8.5), we can establish the following result.

Lemma 9.1.

Suppose that $\nabla^2 h$ is Lipschitz continuous in a neighborhood of the box $\{z \mid \|z-x\|_\infty \le \epsilon\}$ with Lipschitz constant L_h. Then we have

$$\|\nabla_\epsilon f(x) - \nabla h(x)\|_\infty \le L_h \epsilon^2 + \frac{\eta(x;\epsilon)}{\epsilon}. \tag{9.5}$$

Thus the error in the approximation (9.3) comes from both the intrinsic finite difference approximation error (the $O(\epsilon^2)$ term) and the noise (the $\eta(x;\epsilon)/\epsilon$ term). If the noise dominates the difference interval ϵ, we cannot expect any accuracy at all in $\nabla_\epsilon f(x)$, so it will only be pure luck if $-\nabla_\epsilon f(x)$ turns out to be a direction of descent for f.

Instead of computing a tight cluster of function values around the current iterate, as required by a finite-difference approximation to the gradient, it may be preferable to separate these points more widely and use them to construct a model of the objective function. This approach, which we consider in the next section and in Section 9.6, may be more robust to the presence of noise.

9.2 MODEL-BASED METHODS

Some of the most effective algorithms for unconstrained optimization described in the previous chapters compute steps by minimizing a quadratic model of the objective function f. The model is formed by using function and derivative information at the current iterate. When derivatives are not available, we may define the model m_k as the quadratic function that interpolates f at a set of appropriately chosen sample points. Since such a model is usually nonconvex, the model-based methods discussed in this chapter use a trust-region approach to compute the step.

Suppose that at the current iterate x_k we have a set of sample points $Y = \{y^1, y^2, \dots, y^q\}$, with $y^i \in \mathbb{R}^n$, $i = 1, 2, \dots, q$. We assume that x_k is an element of this set and that no point in Y has a lower function value than x_k. We wish to construct a quadratic model of the form

$$m_k(x_k + p) = c + g^T p + \tfrac{1}{2} p^T G p. \tag{9.6}$$

We cannot define $g = \nabla f(x_k)$ and $G = \nabla^2 f(x_k)$ because these derivatives are not available. Instead, we determine the scalar c, the vector $g \in R^n$, and the symmetric matrix $G \in R^{n \times n}$ by imposing the interpolation conditions

$$m_k(y^l) = f(y^l), \qquad l = 1, 2, \dots, q. \tag{9.7}$$

Since there are $\frac{1}{2}(n+1)(n+2)$ coefficients in the model (9.6) (that is, the components of c, g, and G, taking into account the symmetry of G), the interpolation conditions (9.7) determine m_k uniquely only if

$$q = \tfrac{1}{2}(n+1)(n+2). \tag{9.8}$$

In this case, (9.7) can be written as a square linear system of equations in the coefficients of the model. If we choose the interpolation points $y^1, y^2, \dots, y^q$ so that this linear system is nonsingular, the model m_k will be uniquely determined.

Once m_k has been formed, we compute a step p by approximately solving the trust-region subproblem

$$\min_p \; m_k(x_k + p), \qquad \text{subject to} \quad \|p\|_2 \le \Delta, \tag{9.9}$$

for some trust-region radius $\Delta > 0$. We can use one of the techniques described in Chapter 4 to solve this subproblem. If $x_k + p$ gives a sufficient reduction in the objective function, the new iterate is defined as $x_{k+1} = x_k + p$, the trust region radius Δ is updated, and a new iteration commences. Otherwise the step is rejected, and the interpolation set Y may be improved or the trust region shrunk.

To reduce the cost of the algorithm, we update the model m_k at every iteration, rather than recomputing it from scratch. In practice, we choose a convenient basis for the space of quadratic polynomials, the most common choices being Lagrange and Newton polynomials. The properties of these bases can be used both to measure appropriateness of the sample set Y and to change this set if necessary. A complete algorithm that treats all these issues effectively is far more complicated than the quasi-Newton methods discussed in Chapter 6. Consequently, we will provide only a broad outline of model-based DFO methods.

As is common in trust-region algorithms, the step-acceptance and trust-region update strategies are based on the ratio between the actual reduction in the function and the reduction predicted by the model, that is,

$$\rho = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}, \tag{9.10}$$

where x_k^+ denotes the trial point. Throughout this section, the integer q is defined by (9.8).

Algorithm 9.1 (Model-Based Derivative-Free Method).

Choose an interpolation set $Y = \{y^1, y^2, \ldots, y^q\}$ such that the linear system defined by (9.7) is nonsingular, and select x_0 as a point in this set such that $f(x_0) \le f(y^i)$ for all $y^i \in Y$. Choose an initial trust region radius Δ_0, a constant $\eta \in (0, 1)$, and set $k \leftarrow 0$.

```
repeat until a convergence test is satisfied:
    Form the quadratic model $m_k(x_k + p)$ that satisfies the interpolation
        conditions (9.7);
    Compute a step $p$ by approximately solving subproblem (9.9);
    Define the trial point as $x_k^+ = x_k + p$;
    Compute the ratio $\rho$ defined by (9.10);
    if $\rho \ge \eta$
        Replace an element of $Y$ by $x_k^+$;
        Choose $\Delta_{k+1} \ge \Delta_k$;
        Set $x_{k+1} \leftarrow x_k^+$;
        Set $k \leftarrow k + 1$ and go to the next iteration;
    else if the set $Y$ need not be improved
        Choose $\Delta_{k+1} < \Delta_k$;
        Set $x_{k+1} \leftarrow x_k$;
        Set $k \leftarrow k + 1$ and go to the next iteration;
    end (if)
```

```
        Invoke a geometry-improving procedure to update $Y$:
                at least one of the points in $Y$ is replaced by some other point,
                with the goal of improving the conditioning of (9.7);
        Set $\Delta_{k+1} \leftarrow \Delta_k$;
        Choose $\hat{x}$ as an element in $Y$ with lowest function value;
        Set $x_k^+ \leftarrow \hat{x}$ and recompute $\rho$ by (9.10);
        **if** $\rho \geq \eta$
                Set $x_{k+1} \leftarrow x_k^+$;
        **else**
                Set $x_{k+1} \leftarrow x_k$;
        **end (if)**
        Set $k \leftarrow k + 1$;
**end (repeat)**
```

The case of $\rho \geq \eta$, in which we obtain sufficient reduction in the merit function, is the simplest. In this case we always accept the trial point x_k^+ as the new iterate, include x_k^+ in Y, and remove an element from Y.

When sufficient reduction is not achieved ($\rho < \eta$), we look at two possible causes: inadequacy of the interpolation set Y and a trust region that is too large. The first cause can arise when the iterates become restricted to a low-dimensional surface of $\mathbb{R}^n$ that does not contain the solution. The algorithm could then be converging to a minimizer in this subset. Behavior such as this can be detected by monitoring the conditioning of the linear system defined by the interpolation conditions (9.7). If the condition number is too high, we change Y to improve it, typically by replacing one element of Y with a new element so as to move the interpolation system (9.7) as far away from singularity as possible. If Y seems adequate, we simply decrease the trust region radius Δ, as is done in the methods of Chapter 4.

A good initial choice for Y is given by the vertices and the midpoints of the edges of a simplex in $\mathbb{R}^n$.

The use of quadratic models limits the size of problems that can be solved in practice. Performing $O(n^2)$ function evaluations just to start the algorithm is onerous, even for moderate values of n (say, $n = 50$). In addition, the cost of the iteration is high. Even by updating the model m_k at every iteration, rather than recomputing it from scratch, the number of operations required to construct m_k and compute a step is $O(n^4)$ [257].

To alleviate these drawbacks, we can replace the quadratic model by a linear model in which the matrix G in (9.6) is set to zero. Since such a model contains only $n + 1$ parameters, we need to retain only $n + 1$ interpolation points in the set Y, and the cost of each iteration is $O(n^3)$. Algorithm 9.1 can be applied with little modification when the model is linear, but it is not rapidly convergent because linear models cannot represent curvature of the problem. Therefore, some model-based algorithms start with $n + 1$ initial points and compute steps using a linear model, but after $q = \frac{1}{2}(n+1)(n+2)$ function values become available, they switch to using quadratic models.

INTERPOLATION AND POLYNOMIAL BASES

We now consider in more detail how to form a model of the objective function using interpolation techniques. We begin by considering a linear model of the form

$$m_k(x_k + p) = f(x_k) + g^T p. \tag{9.11}$$

To determine the vector $g \in \mathbb{R}^n$, we impose the interpolation conditions $m_k(y^l) = f(y^l)$, $l = 1, 2, \ldots, n$, which can be written as

$$(s^l)^T g = f(y^l) - f(x_k), \qquad l = 1, 2, \ldots, n, \tag{9.12}$$

where

$$s^l = y^l - x_k, \qquad l = 1, 2, \ldots, n. \tag{9.13}$$

Conditions (9.12) represent a linear system of equations in which the rows of the coefficient matrix are given by the vectors $(s^l)^T$. It follows that the model (9.11) is determined uniquely by (9.12) if and only if the interpolation points $\{y^1, y^2, \ldots, y^n\}$ are such that the set $\{s^l : l = 1, 2, \ldots, n\}$ is linearly independent. If this condition holds, the simplex formed by the points $x_k, y^1, y^2, \ldots, y^n$ is said to be *nondegenerate.*

Let us now consider how to construct a quadratic model of the form (9.6), with $f = f(x_k)$. We rewrite the model as

$$m_k(x_k + p) = f(x_k) + g^T p + \sum_{i<j} G_{ij} p_i p_j + \tfrac{1}{2} \sum_i G_{ii} p_i^2 \tag{9.14}$$

$$\stackrel{\text{def}}{=} f(x_k) + \hat{g}^T \hat{p}, \tag{9.15}$$

where we have collected the elements of g and G in the $(q-1)$-vector of unknowns

$$\hat{g} \equiv \left(g^T, \{G_{ij}\}_{i<j}, \left\{\tfrac{1}{\sqrt{2}} G_{ii}\right\}\right)^T, \tag{9.16}$$

and where the $(q-1)$-vector $\hat{p}$ is given by

$$\hat{p} \equiv \left(p^T, \{p_i p_j\}_{i<j}, \left\{\tfrac{1}{\sqrt{2}} p_i^2\right\}\right)^T.$$

The model (9.15) has the same form as (9.11), and the determination of the vector of unknown coefficients $\hat{g}$ can be done as in the linear case.

Multivariate quadratic functions can be represented in various ways. The *monomial basis* (9.14) has the advantage that known structure in the Hessian can be imposed easily by setting appropriate elements in G to zero. Other bases are, however, more convenient when one is developing mechanisms for avoiding singularity of the system (9.7).

We denote by $\{\phi_i(\cdot)\}_{i=1}^q$ a basis for the linear space of n-dimensional quadratic functions. The function (9.6) can therefore be expressed as

$$m_k(x) = \sum_{i=1}^{q} \alpha_i \phi_i(x),$$

for some coefficients α_i. The interpolation set $Y = \{y^1, y^2, \ldots, y^q\}$ determines the coefficients α_i uniquely if the determinant

$$\delta(Y) \overset{\text{def}}{=} \det \begin{pmatrix} \phi_1(y^1) & \cdots & \phi_1(y^q) \\ \vdots & & \vdots \\ \phi_q(y^1) & \cdots & \phi_q(y^q) \end{pmatrix} \tag{9.17}$$

is nonzero.

As model-based algorithms iterate, the determinant $\delta(Y)$ may approach zero, leading to numerical difficulties or even failure. Several algorithms therefore contain a mechanism for keeping the interpolation points well placed. We now describe one of those mechanisms.

UPDATING THE INTERPOLATION SET

Rather than waiting until the determinant $\delta(Y)$ becomes smaller than a threshold, we may invoke a geometry-improving procedure whenever a trial point does not provide sufficient decrease in f. The goal in this case is to replace one of the interpolation points so that the determinant (9.17) increases in magnitude. To guide us in this exchange, we use the following property of $\delta(Y)$, which we state in terms of Lagrange functions.

For every $y \in Y$, we define the Lagrangian function $L(\cdot, y)$ to be a polynomial of degree at most 2 such that $L(y, y) = 1$ and $L(\hat{y}, y) = 0$ for $\hat{y} \neq y$, $\hat{y} \in Y$. Suppose that the set Y is updated by removing a point y_- and replacing it by some other point y_+, to give the new set Y^+. One can show that (after a suitable normalization and given certain conditions [256])

$$|\delta(Y^+)| \leq |L(y_+, y_-)|\, |\delta(Y)|. \tag{9.18}$$

Algorithm 9.1 can make good use of this inequality to update the interpolation set.

Consider first the case in which trial point x^+ provides sufficient reduction in the objective function ($\rho \geq \eta$). We include x^+ in Y and remove another point y_- from Y.

Motivated by (9.18), we select the outgoing point as follows:

$$y_- = \arg\max_{y\in Y} |L(x^+, y)|.$$

Next, let us consider the case in which the reduction in f is not sufficient ($\rho < \eta$). We first determine whether the set Y should be improved, and for this purpose we use the following rule. We consider Y to be adequate at the current iterate x_k if for all $y^i \in Y$ such that $\|x_k - y^i\| \le \Delta$ we have that $|\delta(Y)|$ cannot be doubled by replacing one of these interpolation points y^i with any point y inside the trust region. If Y is adequate but the reduction in f was not sufficient, we decrease the trust-region radius and begin a new iteration.

If Y is inadequate, the geometry-improving mechanism is invoked. We choose a point $y_- \in Y$ and replace it by some other point y^+ that is chosen solely with the objective of improving the determinant (9.17). For every point $y^i \in Y$, we define its potential replacement y^i_r as

$$y^i_r = \arg\max_{\|y-x_k\|\le\Delta} |L(y, y^i)|.$$

The outgoing point y_- is selected as the point for which $|L(y^i_r, y^i)|$ is maximized over all indices $y^i \in Y$.

Implementing these rules efficiently in practice is not simple, and one must also consider several possible difficulties we have not discussed; see [76]. Strategies for improving the position of the interpolation set are the subject of ongoing investigation and new developments are likely in the coming years.

A METHOD BASED ON MINIMUM-CHANGE UPDATING

We now consider a method that be viewed as an extension of the quasi-Newton approach discussed in Chapter 6. The method uses quadratic models but requires only $O(n^3)$ operations per iteration, substantially fewer than the $O(n^4)$ operations required by the methods described above. To achieve this economy, the method retains only $O(n)$ points for the interpolation conditions (9.7) and absorbs the remaining degrees of freedom in the model (9.6) by requiring that the Hessian of the model change as little as possible from one iteration to the next. This least-change property is one of the key ingredients in quasi-Newton methods, the other ingredient being the requirement that the model interpolate the gradient ∇f at the two most recent points. The method we describe now combines the least-change property with interpolation of function values.

At the kth iteration of the algorithm, a new quadratic model m_{k+1} of the form (9.6) is constructed after taking a step from x_k to x_{k+1}. The coefficients f_{k+1}, g_{k+1}, G_{k+1} of the model m_{k+1} are determined as the solution of the problem

$$\min_{f,g,G} \quad \|G - G_k\|_F^2 \tag{9.19a}$$

$$\begin{aligned} \text{subject to} \quad & G \text{ symmetric} \\ & m(y^l) = f(y^l) \qquad l = 1, 2, \ldots, \hat{q}, \end{aligned} \tag{9.19b}$$

where $\| \cdot \|_F$ denotes the Frobenius norm (see (A.9)), G_k is the Hessian of the previous model m_k, and $\hat{q}$ is an integer comparable to n. One can show that the integer $\hat{q}$ must be chosen larger than $n + 1$ to guarantee that G_{k+1} is not equal to G_k. An appropriate value in practice is $\hat{q} = 2n + 1$; for this choice the number of interpolation points is roughly twice that used for linear models.

Problem (9.19) is an equality-constrained quadratic program whose KKT conditions can be expressed as a system of equations. Once the model m_{k+1} is determined, we compute a new step by solving a trust-region problem of the form (9.9). In this approach, too, it is necessary to ensure that the geometry of the interpolation set Y is adequate. We therefore impose two minimum requirements. First, the set Y should be such that the equations (9.19b) can be satisfied for any right-hand side. Second, the points y^i should not all lie in a hyperplane. If these two conditions hold, problem (9.19) has a unique solution.

A practical algorithm based on the subproblem (9.19) resembles Algorithm 9.1 in that it contains procedures both for generating new iterates and for improving the geometry of the set Y. The implementation described in [260] contains other features to ensure that the interpolation points are well separated and that steps are not too small. A strength of this method is that it requires only $O(n)$ interpolation points to start producing productive steps. In practice the method often approaches a solution with fewer than $\frac{1}{2}(n + 1)(n + 2)$ function evaluations. However, since this approach has been developed only recently, there is insufficient numerical experience to assess its full potential.

9.3 COORDINATE AND PATTERN-SEARCH METHODS

Rather than constructing a model of f explicitly based on function values, coordinate search and pattern-search methods look along certain specified directions from the current iterate for a point with a lower function value. If such a point is found, they step to it and repeat the process, possibly modifying the directions of search for the next iteration. If no satisfactory new point is found, the step length along the current search directions may be adjusted, or new search directions may be generated.

We describe first a simple approach of this type that has been used often in practice. We then consider a generalized approach that is potentially more efficient and has stronger theoretical properties.

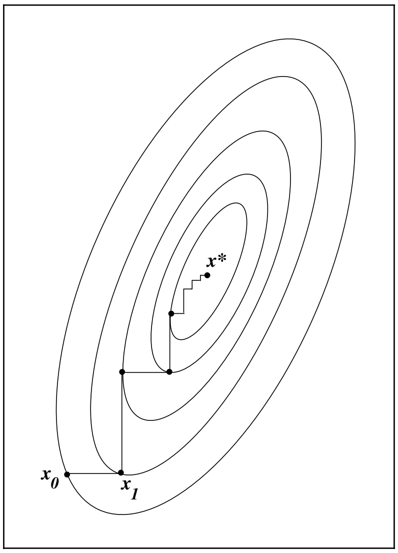

Figure 9.1
Coordinate search method makes slow progress on this function of two variables.

COORDINATE SEARCH METHOD

The coordinate search method (also known as the coordinate descent method or the alternating variables method) cycles through the n coordinate directions $e_1, e_2, \ldots, e_n$, obtaining new iterates by performing a line search along each direction in turn. Specifically, at the first iteration, we fix all components of x except the first one x_1 and find a new value of this component that minimizes (or at least reduces) the objective function. On the next iteration, we repeat the process with the second component x_2, and so on. After n iterations, we return to the first variable and repeat the cycle. Though simple and somewhat intuitive, this method can be quite inefficient in practice, as we illustrate in Figure 9.1 for a quadratic function in two variables. Note that after a few iterations, neither the vertical (x_2) nor the horizontal (x_1) move makes much progress toward the solution at each iteration.

In general, the coordinate search method can iterate infinitely without ever approaching a point where the gradient of the objective function vanishes, even when exact line searches are used. (By contrast, as we showed in Section 3.2, the steepest descent method produces a sequence of iterates $\{x_k\}$ for which $\|\nabla f_k\| \to 0$, under reasonable assumptions.) In fact, a cyclic search along *any* set of linearly independent directions does not guarantee global convergence [243]. Technically speaking, this difficulty arises because the steepest descent search direction $-\nabla f_k$ may become more and more perpendicular to the coordinate search direction. In such circumstances, the Zoutendijk condition (3.14) is satisfied because $\cos\theta_k$ approaches zero rapidly, even when ∇f_k does not approach zero.

When the coordinate search method *does* converge to a solution, it often converges much more slowly than the steepest descent method, and the difference between the two approaches tends to increase with the number of variables. However, coordinate search may still be useful because it does not require calculation of the gradient ∇f_k, and the speed of convergence can be quite acceptable if the variables are loosely coupled in the objective function f.

Many variants of the coordinate search method have been proposed, some of which allow a global convergence property to be proved. One simple variant is a "back-and-forth" approach in which we search along the sequence of directions

$$e_1, e_2, \ldots, e_{n-1}, e_n, e_{n-1}, \ldots, e_2, e_1, e_2, \ldots \qquad \text{(repeats)}.$$

Another approach, suggested by Figure 9.1, is first to perform a sequence of coordinate descent steps and then search along the line joining the first and last points in the cycle. Several algorithms, such as that of Hooke and Jeeves, are based on these ideas; see Fletcher [101] and Gill, Murray, and Wright [130].

The pattern-search approach, described next, generalizes coordinate search in that it allows the use of a richer set of search directions at each iteration.

PATTERN-SEARCH METHODS

We consider pattern-search methods that choose a certain set of search directions at each iterate and evaluate f at a given step length along each of these directions. These candidate points form a "frame," or "stencil," around the current iterate. If a point with a significantly lower function value is found, it is adopted as the new iterate, and the center of the frame is shifted to this new point. Whether shifted or not, the frame may then be altered in some way (the set of search directions may be changed, or the step length may grow or shrink), and the process repeats. For certain methods of this type it is possible to prove global convergence results—typically, that there exists a stationary accumulation point.

The presence of noise or other forms of inexactness in the function values may affect the performance of pattern-search algorithms and certainly impacts the convergence theory. Nonsmoothness may also cause undesirable behavior, as can be shown by simple examples, although satisfactory convergence is often observed on nonsmooth problems.

To define pattern-search methods, we introduce some notation. For the current iterate x_k, we define $\mathcal{D}_k$ to be the set of possible search directions and γ_k to be the line search parameter. The frame consists of the points $x_k + \gamma_k p_k$, for all $p_k \in \mathcal{D}_k$. When one of the points in the frame yields a significant decrease in f, we take the step and may also increase γ_k, so as to expand the frame for the next iteration. If none of the points in the frame has a significantly better function value than f_k, we reduce γ_k (contract the frame), set $x_{k+1} = x_k$, and repeat. In either case, we may change the direction set $\mathcal{D}_k$ prior to the next iteration, subject to certain restrictions.

A more precise description of the algorithm follows.

Algorithm 9.2 (Pattern-Search).

```
Given convergence tolerance $\gamma_{\text{tol}}$, contraction parameter $\theta_{\max}$,
        sufficient decrease function $\rho : [0, \infty) \to \mathbb{R}$ with $\rho(t)$ an increasing
        function of $t$ and $\rho(t)/t \to 0$ as $t \downarrow 0$;
Choose initial point $x_0$, initial step length $\gamma_0 > \gamma_{\text{tol}}$, initial direction set $\mathcal{D}_0$;
for $k = 1, 2, \ldots$
        if $\gamma_k \le \gamma_{\text{tol}}$
                stop;
        if $f(x_k + \gamma_k p_k) < f(x_k) - \rho(\gamma_k)$ for some $p_k \in \mathcal{D}_k$
                Set $x_{k+1} \leftarrow x_k + \gamma_k p_k$ for some such $p_k$;
                Set $\gamma_{k+1} \leftarrow \phi_k \gamma_k$ for some $\phi_k \ge 1$; (* increase step length *)
        else
                Set $x_{k+1} \leftarrow x_k$;
                Set $\gamma_{k+1} \leftarrow \theta_k \gamma_k$, where $0 < \theta_k \le \theta_{\max} < 1$;
                end (if)
end (for)
```

A wise choice of the direction set $\mathcal{D}_k$ is crucial to the practical behavior of this approach and to the theoretical results that can be proved about it. A key condition is that at least one direction in this set should give a direction of descent for f whenever $\nabla f(x_k) \neq 0$ (that is, whenever x_k is not a stationary point). To make this condition specific, we refer to formula (3.12), where we defined the angle between a possible search direction d and the gradient ∇f_k as follows:

$$\cos\theta = \frac{-\nabla f_k^T p}{\|\nabla f_k\| \, \|p\|}. \tag{9.20}$$

Recall from Theorem 3.2 that global convergence of a line-search method to a stationary point of f could be ensured if the search direction d at each iterate x_k satisfied $\cos\theta \ge \delta$, for some constant $\delta > 0$, and if the line search parameter satisfied certain conditions. In the same spirit, we choose $\mathcal{D}_k$ so that at least one direction $p \in \mathcal{D}_k$ will yield $\cos\theta > \delta$, regardless of the value of ∇f_k. This condition is as follows:

$$\kappa(\mathcal{D}_k) \overset{\text{def}}{=} \min_{v \in \mathbb{R}^n} \max_{p \in \mathcal{D}_k} \frac{v^T p}{\|v\| \|p\|} \ge \delta. \tag{9.21}$$

A second condition on $\mathcal{D}_k$ is that the lengths of the vectors in this set are all roughly similar, so that the diameter of the frame formed by this set is captured adequately by the step length parameter γ_k. Thus, we impose the condition

$$\beta_{\min} \le \|p\| \le \beta_{\max}, \quad \text{for all } p \in \mathcal{D}_k, \tag{9.22}$$

for some positive constants $\beta_{\min}$ and $\beta_{\max}$ and all k. If the conditions (9.21) and (9.22) hold, we have for any k that

$$-\nabla f_k^T p \geq \kappa(\mathcal{D}_k)\|\nabla f_k\|\|p\| \geq \delta\beta_{\min}\|\nabla f_k\|, \quad \text{for some } p \in \mathcal{D}_k.$$

Examples of sets $\mathcal{D}_k$ that satisfy the properties (9.21) and (9.22) include the coordinate direction set

$$\{e_1, e_2, \ldots, e_n, -e_1, -e_2, \ldots, -e_n\}, \tag{9.23}$$

and the set of $n + 1$ vectors defined by

$$p_i = \frac{1}{2n}e - e_i, \quad i = 1, 2, \ldots, n; \qquad p_{n+1} = \frac{1}{2n}e, \tag{9.24}$$

where $e = (1, 1, \ldots, 1)^T$. For $n = 3$ these direction sets are sketched in Figure 9.2.

The coordinate descent method described above is similar to the special case of Algorithm 9.2 obtained by setting $\mathcal{D}_k = \{e_i, -e_i\}$ for some $i = 1, 2, \ldots, n$ at each iteration. Note that for this choice of $\mathcal{D}_k$, we have $\kappa(\mathcal{D}_k) = 0$ for all k. Hence, as noted above, $\cos\theta$ can be arbitrarily close to zero at each iteration.

Often, the directions that satisfy the properties (9.21) and (9.22) form only a subset of the direction set $\mathcal{D}_k$, which may contain other directions as well. These additional directions could be chosen heuristically, according to some knowledge of the function f and its scaling, or according to experience on previous iterations. They could also be chosen as linear combinations of the core set of directions (the ones that ensure $\delta > 0$).

Note that Algorithm 9.2 does not require us to choose the point $x_k + \gamma_k p_k$, $p_k \in \mathcal{D}_k$, with the smallest objective value. Indeed, we may save on function evaluations by not evaluating f at all points in the frame, but rather performing the evaluations one at a time and accepting the first candidate point that satisfies the sufficient decrease condition.

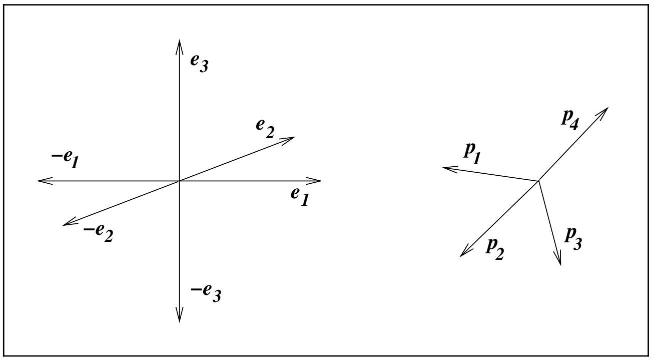

Figure 9.2 Generating search sets in $\mathbb{R}^3$: coordinate direction set (left) and simplex set (right).

Another important detail in the implementation of Algorithm 9.2 is the choice of sufficient decrease function $\rho(t)$. If $\rho(\cdot)$ is chosen to be identically zero, then any candidate point that produces a decrease in f is acceptable as a new iterate. As we have seen in Chapter 3, such a weak condition does not lead to strong global convergence results in general. A more appropriate choice might be $\rho(t) = Mt^{3/2}$, where M is some positive constant.

9.4 A CONJUGATE-DIRECTION METHOD

We have seen in Chapter 5 that the minimizer of a strictly convex quadratic function

$$f(x) = \tfrac{1}{2}x^T Ax - b^T x \tag{9.25}$$

can be located by performing one-dimensional minimizations along a set of n conjugate directions. These directions were defined in Chapter 5 as a linear combination of gradients. In this section, we show how to construct conjugate directions using only function values, and we therefore devise an algorithm for minimizing (9.25) that requires only function value calculations. Naturally, we also consider an extension of this approach to the case of a nonlinear objective f.

We use the *parallel subspace property*, which we describe first for the case $n = 2$. Consider two parallel lines $l_1(\alpha) = x_1 + \alpha p$ and $l_2(\alpha) = x_2 + \alpha p$, where x_1, x_2, and p are given vectors in $\mathbb{R}^2$ and α is the scalar parameter that defines the lines. We show below that if x_1^* and x_2^* denote the minimizers of $f(x)$ along l_1 and l_2, respectively, then $x_1^* - x_2^*$ is conjugate to p. Hence, if we perform a one-dimensional minimization along the line joining x_1^* and x_2^*, we will reach the minimizer of f, because we have successively minimized along the two conjugate directions p and $x_2^* - x_1^*$. This process is illustrated in Figure 9.3.

This observation suggests the following algorithm for minimizing a two-dimensional quadratic function f. We choose a set of linearly independent directions, say the coordinate directions e_1 and e_2. From any initial point x_0, we first minimize f along e_2 to obtain the point x_1. We then perform successive minimizations along e_1 and e_2, starting from x_1, to obtain the point z. It follows from the parallel subspace property that $z - x_1$ is conjugate to e_2 because x_1 and z are minimizers along two lines parallel to e_2. Thus, if we perform a one-dimensional search from x_1 along the direction $z - x_1$, we will locate the minimizer of f.

We now state the parallel subspace minimization property in its most general form. Suppose that x_1, x_2 are two distinct points in $\mathbb{R}^n$ and that $\{p_1, p_2, \dots, p_l\}$ is a set of linearly independent directions in $\mathbb{R}^n$. Let us define the two parallel linear varieties

$$S_1 = \left\{ x_1 + \sum_{i=1}^{l} \alpha_i p_i \,|\, \alpha_i \in \mathbb{R},\ i = 1, 2, \dots, l \right\},$$

$$S_2 = \left\{ x_2 + \sum_{i=1}^{l} \alpha_i p_i \,|\, \alpha_i \in \mathbb{R},\ i = 1, 2, \dots, l \right\}.$$

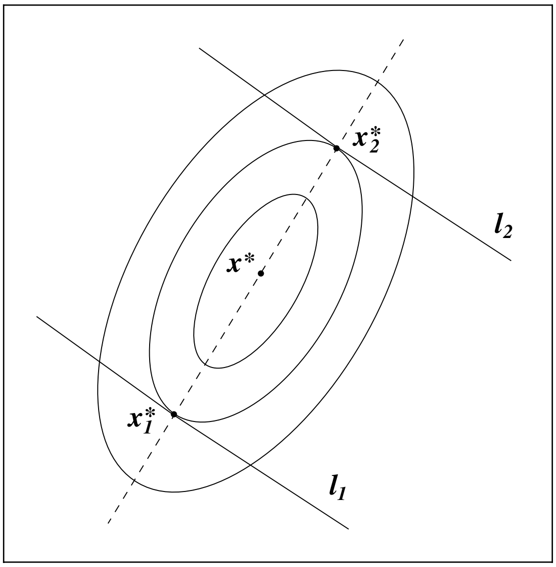

Figure 9.3
Geometric construction of conjugate directions. (The minimizer of f is denoted by x^*.)

If we denote the minimizers of f on S_1 and S_2 by x_1^* and x_2^*, respectively, then $x_2^* - x_1^*$ is conjugate to $p_1, p_2, \ldots, p_l$. It is easy to verify this claim. By the minimization property, we have that

$$\left.\frac{\partial f(x_1^* + \alpha_i p_i)}{\partial \alpha_i}\right|_{\alpha_i=0} = \nabla f(x_1^*)^T p_i = 0, \qquad i = 1, 2, \ldots, l,$$

and similarly for x_2. Therefore we have from (9.25) that

$$\begin{aligned} 0 &= (\nabla f(x_1^*) - \nabla f(x_2^*))^T p_i \\ &= (Ax_1^* - b - Ax_2^* + b)^T p_i \\ &= (x_1^* - x_2^*)^T A p_i, \qquad i = 1, 2, \ldots, l. \end{aligned} \tag{9.26}$$

We now consider the case $n = 3$ and show how the parallel subspace property can be used to generate a set of three conjugate directions. We choose a set of linearly independent directions, say e_1, e_2, e_3. From any starting point x_0 we first minimize f along the last direction e_3 to obtain a point x_1. We then perform three successive one-dimensional minimizations, starting from x_1, along the directions e_1, e_2, e_3 and denote the resulting point by z. Next, we minimize f along the direction $p_1 = z - x_1$ to obtain x_2. As noted earlier, $p_1 = z - x_1$ is conjugate to e_3. We note also that x_2 is the minimizer of f on the set $S_1 = \{y + \alpha_1 e_3 + \alpha_2 p_1 \mid \alpha_1 \in \mathbb{R},\ \alpha_2 \in \mathbb{R}\}$, where y is the intermediate point obtained after minimizing along e_1 and e_2.

A new iteration now commences. We discard e_1 and define the new set of search directions as e_2, e_3, p_1. We perform one-dimensional minimizations along e_2, e_3, p_1, starting from x_2, to obtain the point $\hat{z}$. Note that $\hat{z}$ can be viewed as the minimizer of f on the set $S_2 = \{\hat{y} + \alpha_1 e_3 + \alpha_2 p_1 \,|\, \alpha_1 \in \mathbb{R},\ \alpha_2 \in \mathbb{R}\}$, for some intermediate point $\hat{y}$. Therefore, by applying the parallel subspace minimization property to the sets S_1 and S_2 just defined, we have that $p_2 = \hat{z} - x_2$ is conjugate to both e_3 and p_1. We then minimize f along p_2 to obtain a point x_3, which is the minimizer of f. This procedure thus generates the conjugate directions e_3, p_1, p_2.

We can now state the general algorithm, which consists of an inner and an outer iteration. In the inner iteration, n one-dimensional minimizations are performed along a set of linearly independent directions. Upon completion of the inner iteration, a new conjugate direction is generated, which replaces one of the previously stored search directions.

Algorithm 9.3 (DFO Method of Conjugate Directions).
Choose an initial point x_0 and set $p_i = e_i$, for $i = 1, 2, \dots, n$;
Compute x_1 as the minimizer of f along the line $x_0 + \alpha p_n$;
Set $k \leftarrow 1$.

```
repeat until a convergence test is satisfied
    Set $z_1 \leftarrow x_k$;
    for $j = 1, 2, \dots, n$
        Calculate $\alpha_j$ so that $f(z_j + \alpha_j p_j)$ is minimized;
        Set $z_{j+1} \leftarrow z_j + \alpha_j p_j$;
    end (for)
    Set $p_j \leftarrow p_{j+1}$ for $j = 1, 2, \dots, n-1$ and $p_n \leftarrow z_{n+1} - z_1$;
    Calculate $\alpha_n$ so that $f(z_{n+1} + \alpha_n p_n)$ is minimized;
    Set $x_{k+1} \leftarrow z_{n+1} + \alpha_n p_n$;
    Set $k \leftarrow k + 1$;
end (repeat)
```

The line searches can be performed by quadratic interpolation using three function values along each search direction. Since the restriction of (9.25) to a line is a (strictly convex) quadratic, the interpolating quadratic matches it exactly, and the one-dimensional minimizer can easily be computed. Note that at the end of (the outer) iteration k, the directions $p_{n-k}, p_{n-k+1}, \dots, p_n$ are conjugate by the property mentioned above. Thus the algorithm terminates at the minimizer of (9.25) after $n - 1$ iterations, provided none of the conjugate directions is zero. Unfortunately, this possibility cannot be ruled out, and some safeguards described below must be incorporated to improve robustness. In the (usual) case that Algorithm 9.3 terminates after $n - 1$ iterations, it will perform $O(n^2)$ function evaluations.

Algorithm 9.3 can be extended to minimize nonquadratic objective functions. The only change is in the line search, which must be performed approximately, using interpolation. Because of the possible nonconvexity, this one-dimensional search must be done with care; see Brent [39] for a treatment of this subject. Numerical experience indicates that this extension of Algorithm 9.3 performs adequately for small-dimensional problems but that sometimes the directions $\{p_i\}$ tend to become linearly dependent. Several modifications of the algorithm have been proposed to guard against this possibility. One such modification measures the degree to which the directions $\{p_i\}$ are conjugate. To do so, we define the scaled directions

$$\hat{p}_i = \frac{p_i}{\sqrt{p_i^T A p_i}}, \qquad i = 1, 2, \ldots, n. \tag{9.27}$$

One can show [239] that the quantity

$$|\det(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n)| \tag{9.28}$$

is maximized if and only if the vectors p_i are conjugate with respect to A. This result suggests that we should *not* replace one of the existing search directions in the set $\{p_1, p_2, \ldots, p_n\}$ by the most recently generated conjugate direction if this action causes the quantity (9.28) to decrease.

Procedure 9.4 implements this strategy for the case of the quadratic objective function (9.25). Some algebraic manipulations (which we do not present here) show that we can compute the scaled directions $\hat{p}_i$ without using the Hessian A because the terms $p_i^T A p_i$ are available from the line search along p_i. Further, only comparisons using computed function values are needed to ensure that (9.28) does not increase. The following procedure is invoked immediately after the execution of the inner iteration (or for-loop) of Algorithm 9.3.

Procedure 9.4 (Updating of the Set of Directions).

```
Find the integer $m \in \{1, 2, \ldots, n\}$ such that $\psi_m = f(x_{m-1}) - f(x_m)$
        is maximized;
Let $f_1 = f(z_1)$, $f_2 = f(z_{n+1})$, and $f_3 = f(2z_{n+1} - z_1)$;
if $f_3 \geq f_1$ or $(f_1 - 2f_2 + f_3)(f_1 - f_2 - \psi_m)^2 \geq \frac{1}{2}\psi_m(f_1 - f_3)^2$
        Keep the set $p_1, p_2, \ldots, p_n$ unchanged and set $x_{k+1} \leftarrow z_{n+1}$;
else
        Set $\hat{p} \leftarrow z_{n+1} - z_1$ and calculate $\hat{\alpha}$ so that $f(z_{n+1} + \hat{\alpha}\hat{p})$ is minimized;
        Set $x_{k+1} \leftarrow z_{n+1} + \alpha\hat{p}$;
        Remove $p_m$ from the set of directions and add $\hat{p}$ to this set;
end (if)
```

This procedure can be applied to general objective functions by implementing inexact one-dimensional line searches. The resulting conjugate-gradient method has been found to be useful for solving small dimensional problems.

9.5 NELDER–MEAD METHOD

The Nelder–Mead simplex-reflection method has been a popular DFO method since its introduction in 1965 [223]. It takes its name from the fact that at any stage of the algorithm, we keep track of $n + 1$ points of interest in $\mathbb{R}^n$, whose convex hull forms a simplex. (The method has nothing to do with the simplex method for linear programming discussed in Chapter 13.) Given a simplex S with vertices $\{z_1, z_2, \dots, z_{n+1}\}$, we can define an associated matrix $V(S)$ by taking the n edges along V from one of its vertices (z_1, say), as follows:

$$V(S) = [z_2 - z_1, z_3 - z_1, \dots, z_{n+1} - z_1] .$$

The simplex is said to be *nondegenerate* or *nonsingular* if V is a nonsingular matrix. (For example, a simplex in $\mathbb{R}^3$ is nondegenerate if its four vertices are not coplanar.)

In a single iteration of the Nelder–Mead algorithm, we seek to remove the vertex with the worst function value and replace it with another point with a better value. The new point is obtained by reflecting, expanding, or contracting the simplex along the line joining the worst vertex with the centroid of the remaining vertices. If we cannot find a better point in this manner, we retain only the vertex with the *best* function value, and we shrink the simplex by moving all other vertices toward this value.

We specify a single step of the algorithm after some defining some notation. The $n+1$ vertices of the current simplex are denoted by $\{x_1, x_2, \dots, x_{n+1}\}$, where we choose the ordering so that

$$f(x_1) \le f(x_2) \le \cdots \le f(x_{n+1}).$$

The centroid of the best n points is denoted by

$$\bar{x} = \sum_{i=1}^{n} x_i .$$

Points along the line joining $\bar{x}$ and the "worst" vertex x_{n+1} are denoted by

$$\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x}).$$

Procedure 9.5 (One Step of Nelder–Mead Simplex).

```
Compute the reflection point $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$;
if $f(x_1) \le f_{-1} < f(x_n)$
        (* reflected point is neither best nor worst in the new simplex *)
        replace $x_{n+1}$ by $\bar{x}(-1)$ and go to next iteration;
else if $f_{-1} < f(x_1)$
```

```
        (* reflected point is better than the current best; try to
                go farther along this direction *)
        Compute the expansion point $\bar{x}(-2)$ and evaluate $f_{-2} = f(\bar{x}(-2))$;
        if $f_{-2} < f_{-1}$
                replace $x_{n+1}$ by $x_{-2}$ and go to next iteration;
        else
                replace $x_{n+1}$ by $x_{-1}$ and go to next iteration;
else if $f_{-1} \geq f(x_n)$
        (* reflected point is still worse than $x_n$; contract *)
        if $f(x_n) \leq f_{-1} < f(x_{n+1})$
                (* try to perform "outside" contraction *)
                evaluate $f_{-1/2} = \bar{x}(-1/2)$;
                if $f_{-1/2} \leq f_{-1}$
                        replace $x_{n+1}$ by $x_{-1/2}$ and go to next iteration;
        else
                (* try to perform "inside" contraction *)
                evaluate $f_{1/2} = \bar{x}(1/2)$;
                if $f_{1/2} < f_{n+1}$
                        replace $x_{n+1}$ by $x_{1/2}$ and go to next iteration;
        (* neither outside nor inside contraction was acceptable;
                shrink the simplex toward $x_1$ *)
        replace $x_i \leftarrow (1/2)(x_1 + x_i)$ for $i = 2, 3, \ldots, n+1$;
```

Procedure 9.5 is illustrated on a three-dimensional example in Figure 9.4. The worst current vertex is x_3, and the possible replacement points are $\bar{x}(-1)$, $\bar{x}(-2)$, $\bar{x}(-\frac{1}{2})$, $\bar{x}(\frac{1}{2})$. If none of the replacement points proves to be satisfactory, the simplex is shrunk to the smaller triangle indicated by the dotted line, which retains the best vertex x_1. The scalars t used in defining the candidate points $\bar{x}(t)$ have been assigned the specific (and standard) values -1, -2, $-\frac{1}{2}$, and $\frac{1}{2}$ in our description above. Different choices are also possible, subject to certain restrictions.

Practical performance of the Nelder–Mead algorithm is often reasonable, though stagnation has been observed to occur at nonoptimal points. Restarting can be used when stagnation is detected; see Kelley [178]. Note that unless the final shrinkage step is performed, the average function value

$$\frac{1}{n+1}\sum_{i=1}^{n+1} f(x_i) \tag{9.29}$$

will decrease at each step. When f is convex, even the shrinkage step is guaranteed not to increase the average function value.

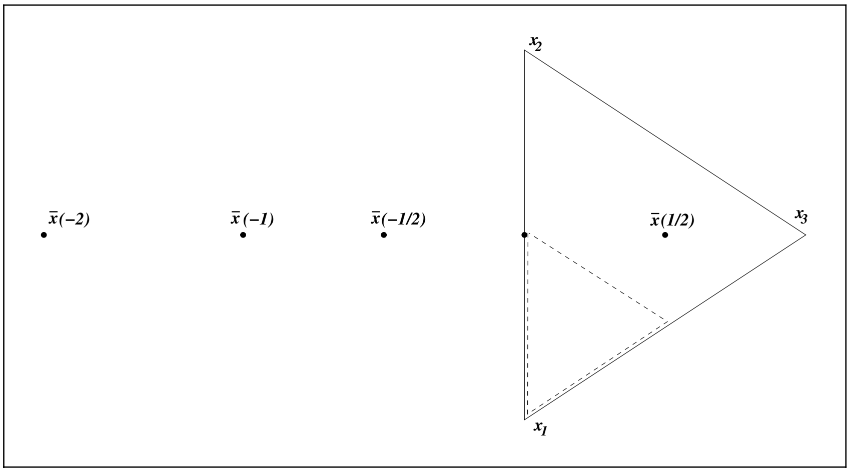

Figure 9.4 One step of the Nelder–Mead simplex method in $\mathbb{R}^3$, showing current simplex (solid triangle with vertices x_1, x_2, x_3), reflection point $\bar{x}(-1)$, expansion point $\bar{x}(-2)$, inside contraction point $\bar{x}(\frac{1}{2})$, outside contraction point $\bar{x}(-\frac{1}{2})$, and shrunken simplex (dotted triangle).

A limited amount of convergence theory has been developed for the Nelder–Mead method in recent years; see, for example, Kelley [179] and Lagarias et al. [186].

9.6 IMPLICIT FILTERING

We now describe an algorithm designed for functions whose evaluations are modeled by (9.2), where h is smooth. This *implicit filtering* approach is, in its simplest form, a variant of the steepest descent algorithm with line search discussed in Chapter 3, in which the gradient ∇f_k is replaced by a finite difference estimate such as (9.3), with a difference parameter ϵ that may not be particularly small.

Implicit filtering works best on functions for which the noise level decreases as the iterates approach a solution. This situation may occur when we have control over the noise level, as is the case when f is obtained by solving a differential equation to a user-specified tolerance, or by running a stochastic simulation for a user-specified number of trials (where an increase in the number of trials usually produces a decrease in the noise). The implicit filtering algorithm decreases ϵ systematically (but, one hopes, not as rapidly as the decay in error) so as to maintain reasonable accuracy in $\nabla_\epsilon f(x)$, given the noise level at the current value of x. For each value of ϵ, it performs an inner loop that is simply an Armijo line search using the search direction $-\nabla_\epsilon f(x)$. If the inner loop is unable to find a satisfactory step length after backtracking at least $a_{\max}$ times, we return to the outer loop, choose a smaller value of ϵ, and repeat. A formal specification follows.

Algorithm 9.6 (Implicit Filtering).

```
Choose a sequence $\{\epsilon_k\} \downarrow 0$, Armijo parameters $c$ and $\rho$ in $(0, 1)$,
        maximum backtracking parameter $a_{\max}$;
Set $k \leftarrow 1$, Choose initial point $x = x_0$;
repeat
        increment_k ← false;
        repeat
                Compute $f(x)$ and $\nabla_{\epsilon_k} f(x)$;
                if $\|\nabla_{\epsilon_k} f(x)\| \le \epsilon_k$
                        increment_k ← true;
                else
                        Find the smallest integer $m$ between 0 and $a_{\max}$ such that
                                $f\left(x - \rho^m \nabla_{\epsilon_k} f(x)\right) \le f(x) - c\rho^m \|\nabla_{\epsilon_k} f(x)\|_2^2$;
                        if no such $m$ exists
                                increment_k ← true;
                        else
                                $x \leftarrow x - \rho^m \nabla_{\epsilon} f(x)$;
        until increment_k;
        $x_k \leftarrow x$; $k \leftarrow k + 1$;
until a termination test is satisfied.
```

Note that the inner loop in Algorithm 9.6 is essentially the backtracking line search algorithm—Algorithm 3.1 of Chapter 3—with a convergence criterion added to detect whether the minimum appears to have been found to within the accuracy implied by the difference parameter ϵ_k. If the gradient estimate $\nabla_{\epsilon_k} f$ is small, or if the line search fails to find a satisfactory new iterate (indicating that the gradient approximation $\nabla_{\epsilon_k} f(x)$ is insufficiently accurate to produce descent in f), we decrease the difference parameter to ϵ_{k+1} and proceed.

A basic convergence result for Algorithm 9.6 is the following.

Theorem 9.2.

Suppose that $\nabla^2 h$ is Lipschitz continuous, that Algorithm 9.6 generates an infinite sequence of iterates $\{x_k\}$, and that

$$\lim_{k\to\infty} \epsilon_k^2 + \frac{\eta(x_k; \epsilon_k)}{\epsilon_k} = 0.$$

Suppose, too, that all but a finite number of inner loops in Algorithm 9.6 terminate with $\|\nabla_{\epsilon_k} f(x_k)\| \le \epsilon_k$. Then all limit points of the sequence $\{x_k\}$ are stationary.

Proof. Using $\{\epsilon_k\} \downarrow 0$, we have under our assumptions on inner loop termination that $\nabla_{\epsilon_k} f(x_k) \to 0$. By invoking the error bound (9.5) and noting that the right-hand side of this expression is approaching zero, we conclude that $\nabla h(x_k) \to 0$. Hence all limit points satisfy $\nabla h(x) = 0$, as claimed. □

More sophisticated versions of implicit filtering methods can be derived by using the gradient estimate $\nabla_{\epsilon_k} f$ to construct quasi-Newton approximate Hessians, and thus generating quasi-Newton search directions instead of the negative-approximate-gradient search direction used in Algorithm 9.6.

NOTES AND REFERENCES

A classical reference on derivative-free methods is Brent [39], which focuses primarily on one-dimensional problems and includes discussion of roundoff errors and global minimization. Recent surveys on derivative-free methods include Wright [314], Powell [256], Conn, Scheinberg, and Toint [76], and Kolda, Lewis, and Torczon [183].

The first model-based method for derivative-free optimization was proposed by Winfield [307]. It uses quadratic models, which are determined by the interpolation conditions (9.7), and computes steps by solving a subproblem of the form (9.9). Practical procedures for improving the geometry of the interpolation points were first developed by Powell in the context of model-based methods using linear and quadratic polynomials; see [256] for a review of this work.

Conn, Scheinberg, and Toint [75] propose and analyze model-based methods and study the use of Newton fundamental polynomials. Methods that combine minimum change updating and interpolation are discussed by Powell [258, 260]. Our presentation of model-based methods in Section 9.2 is based on [76, 259, 258].

For a comprehensive discussion of pattern-search methods of the type discussed here, we refer the reader to the review paper of Kolda, Lewis, and Torczon [183], and the references therein.

The method of conjugate directions given in Algorithm 9.3 was proposed by Powell [239]. For a discussion on the rate of convergence of the coordinate descent method and for more references about this method, see Luenberger [195]. For further information on implicit filtering, see Kelley [179] and Choi and Kelley [60] and the references therein.

Software packages that implement model-based methods include COBYLA [258], DFO [75], UOBYQA [257], WEDGE [200], and NEWUOA [260]. The earliest code is COBYLA, which employs linear models. DFO, UOBYQA, and WEDGE use quadratic models, whereas the method based on minimum change updating (9.19) is implemented in NEWUOA. A pattern-search method is implemented in APPS [171], while DIRECT [173] is designed to find a global solution.

EXERCISES

9.1 Prove Lemma 9.1.

9.2

(a) Verify that the number of interpolation conditions to uniquely determine the coefficients in (9.6) are $q = \frac{1}{2}(n+1)(n+2)$.

(b) Verify that the number of vertices and midpoints of the edges of a nondegenerate simplex in R^n add up to $q = \frac{1}{2}(n+1)(n+2)$ and can therefore be used as the initial interpolation set in a DFO algorithm.

(c) How many interpolation conditions would be required to determine the coefficients in (9.6) if the matrix G were identically 0? How many if G were diagonal? How many if G were tridiagonal?

9.3 Describe conditions on the vectors s^l that guarantee that the model (9.14) is uniquely determined.

9.4 Consider the determination of a quadratic function in two variables.

(a) Show that six points on a line do not determine the quadratic.

(b) Show that six points in a circle in the plane do not uniquely determine the quadratic.

9.5 Use induction to show that at the end of the outer iteration k of Algorithm 9.3, the directions $p_{n-k}, p_{n-k+1}, \ldots, p_n$ are conjugate. Use this fact to show that if the step lengths α_i in Algorithm 9.3 are never zero, the iteration terminates at the minimizer of (9.25) after at most n outer iterations.

9.6 Write a program that computes the one-dimensional minimizer of a strictly convex quadratic function f along a direction p using quadratic interpolation. Describe the formulas used in your program.

9.7 Find the quadratic function

$$m(x_1, x_2) = f + g_1 x_1 + g_2 x_2 + \frac{1}{2} G_{11}^2 x_1^2 + G_{12} x_1 x_2 + \frac{1}{2} G_{22}^2 x_2^2$$

that interpolates the following data: $x_0 = y^1 = (0,0)^T$, $y^2 = (1,0)^T$, $y^3 = (2,0)^T$, $y^4 = (1,1)^T$, $y^5 = (0,2)^T$, $y^6 = (0,1)^T$, and $f(y^1) = 1$, $f(y^2) = 2.0084$, $f(y^3) = 7.0091$, $f(y^4) = 1.0168$, $f(y^5) = -0.9909$, and $f(y^6) = -0.9916$.

9.8 Find the value of δ for which the coordinate generating set (9.23) satisfies the property (9.21).

9.9 Show that $\kappa(\mathcal{D}_k) = 0$, where $\kappa(\cdot)$ is defined by (9.21) and $\mathcal{D}_k = \{e_i, -e_i\}$ for any $i = 1, 2, \ldots, n$.

9.10 (Hard) Prove that the generating set (9.24) satisfies the property (9.21) for a certain value $\delta > 0$, and find this value of δ.

9.11 Justify the statement that the average function value at the Nelder–Mead simplex points will decrease over one step if any of the points $\bar{x}(-1), \bar{x}(-2), \bar{x}(-\frac{1}{2}), \bar{x}(\frac{1}{2})$ are adopted as a replacement for x_{n+1}.

9.12 Show that if f is a convex function, the shrinkage step in the Nelder–Mead simplex method will not increase the average value of the function over the simplex vertices defined by (9.29). Show that unless $f(x_1) = f(x_2) = \cdots = f(x_{n+1})$, the average value will in fact decrease.

9.13 Suppose for the f defined in (9.2), we define the approximate gradient $\nabla_\epsilon f(x)$ by the forward-difference formula

$$\nabla_\epsilon f(x) = \left[\frac{f(x + \epsilon e_i) - f(x)}{\epsilon} \right]_{i=1,2,\dots,n},$$

rather than the central-difference formula (9.3). (This formula requires only half as many function evaluations but is less accurate.) For this definition, prove the following variant of Lemma 9.1: Suppose that $\nabla h(x)$ is Lipschitz continuous in a neighborhood of the box $\{z \mid z \ge x, \; \|z - x\|_\infty \le \epsilon\}$ with Lipschitz constant L_h. Then we have

$$\|\nabla_\epsilon f(x) - \nabla h(x)\|_\infty \le L_h \epsilon + \frac{\eta(x;\epsilon)}{\epsilon},$$

where $\eta(x;\epsilon)$ is redefined as follows:

$$\eta(x;\epsilon) = \sup_{z \ge x, \; \|z - x\|_\infty \le \epsilon} |\phi(z)|.$$