CHAPTER 1

Optimization Methods in Computational Chemistry

Tamar Schlick

Courant Institute of Mathematical Sciences and Chemistry Department, New York University, 251 Mercer Street, New York, New York 10012

INTRODUCTION

In his 1970 *Numerical Methods,* the witty numerical analyst Forman Acton wrote:

> It is with a sense of reluctance that your author introduces this topic, for minimum-seeking methods are often used when a modicum of thought would disclose more appropriate techniques. They are the first refuge of the computational scoundrel, and one feels at times that the world would be a better place if they were quietly abandoned. But even if these techniques are frequently misused, it is equally true that there are problems for which no alternative solution method is known—and so we shall discuss them. A better title for this chapter might be "How to Find Minima—If You Must!"[1]

Although there may be some truth in this view, there is little doubt today that multivariate minimization algorithms are fundamental research tools for scientists in numerous disciplines. A common problem arises when a complex physical system is described by a collection of particles, or combinations of states, in

Reviews in Computational Chemistry, Volume III
Kenny B. Lipkowitz and Donald B. Boyd, Editors
VCH Publishers, Inc. New York, © 1992

a multidimensional phase space. An energy or cost function is associated with each different configuration, and the challenge is to find sets of points that minimize (or maximize) the objective function. Such applications arise frequently in molecular modeling, rational drug design, mathematical biology models, neural networks, combinatorial problems, financial investment planning, architectural design, electronics, meteorology, and computational geometry. I wonder: Would Acton still call us all scoundrels?

The purpose of this chapter is to provide computational chemists a brief background into unconstrained nonlinear optimization methods that seek local minima. Emphasis is placed on methods that are most powerful today for large-scale problems (hundreds to thousands of independent variables) and suitable for potential energy minimization. Only a general practical taste of this very rich and theoretically interesting field is attempted here. For comprehensive treatments of optimization methods, interested readers are referred to a large selection of excellent books on a wide spectrum of levels, from introductory[1,2] to comprehensive[3–7] to specialized-topic volumes.[8–12] The numerical linear algebra and multivariate calculus background can be found in most of the introductory optimization books as well as in standard books on numerical methods and matrix computations.[13,14] Many general concepts mentioned throughout this chapter without explicit citations are discussed in these volumes; however, an attempt is made to make this review as self-contained as possible.

Another introductory note may provide further incentive for novice optimizers to read on. Despite extensive developments in optimization methods in the last decade, large-scale nonlinear optimization still remains an art that requires considerable computing experience, algorithm familiarity, and intuition. In general, "black box" minimization implementations, even those using state-of-the-art algorithms, are only partially successful.

"I have been burnt by too many black boxes," declares Acton in his 1990 preface.[1] There are at least two reasons for our shared sentiment. First, many commercial software vendors, such as NAG and Harwell, have developed general-purpose programs that are, for the most part, easy to use but not always up-to-date with the latest minimization developments. Thus, successful new minimization approaches are often out of reach for nonspecialist mathematicians, let alone scientists in related application fields. Second, application tailoring of algorithms is often very difficult with ready-made software. Such modifications may be crucial in many applications.

For example, a common thread in applications in meteorology, chemistry, or mathematical biology is a natural separability of the objective functions into components of differing complexity (e.g., local and nonlocal interactions). This composition may not only change the relative attractiveness or suitability among different optimization methods, but also lead to very powerful methods for the application at hand when this information is incorporated appropriately.

Such problem tailoring requires some familiarity with the algorithmic modules. It also demands knowledge of the theoretical and practical strengths and weaknesses of the different minimization methods. With rapidly growing improvements in high-performance super and massively parallel machines,[15,16] application-tailored software may be even more important in combination with parallel architectures whose design is motivated by specific applications.

MATHEMATICAL PRELIMINARIES

Notation

We generally denote *scalars* by lowercase Greek letters (e.g., β), column *vectors* by boldface lowercase Roman letters (e.g., $\mathbf{x}$), and *matrices* by capital italic Roman letters (e.g., H). A superscript ^{T} denotes a vector or matrix *transpose*. Thus $\mathbf{x}^T$ is a row vector, $\mathbf{x}^T\mathbf{y}$ is an inner product, and A^T is the transpose of the matrix A. Unless stated otherwise, all vectors belong to $\mathbf{R}^n$, the n-dimensional vector space. Components of a vector are typically written as italic letters with subscripts (e.g., $x_1, x_2, \ldots, x_n$). The standard basis vectors in $\mathbf{R}^n$ are the n vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}$, where $\mathbf{e}_j$ has the entry 1 in the jth component and 0 in all others. Often, the associated vector norm is the standard Euclidean norm, $\|\cdot\|_2$, defined as

$$\|\mathbf{x}\| = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}. \qquad [1]$$

We say that a term is of order n and write $O(n)$ to mean that it is proportional to n.

Problem Statement

We are interested in solving the optimization problem without constraints:

$$\text{minimize } f(\mathbf{x}), \qquad \mathbf{x} \in \mathbf{R}^n \qquad [2]$$

where f is a real-valued function of n variables. We will assume that f is sufficiently smooth, that is, possesses continuous derivatives at least up to second order.

The gradient of f at $\mathbf{x}$ is defined to be the first derivative vector $\nabla f(\mathbf{x})$, or $\mathbf{g}(\mathbf{x})$, whose n components are given by $\mathbf{g}_i(\mathbf{x}) = \partial E(\mathbf{x})/\partial x_i$. The Hessian at $\mathbf{x}$, $H(\mathbf{x})$, is defined to be the $n \times n$ matrix of second partial derivatives with components $H_{ij}(\mathbf{x}) = [\partial^2 f(\mathbf{x})/\partial x_i\, \partial x_j]$. As $\partial^2 f(\mathbf{x})/\partial x_i\, \partial x_j = \partial^2 f(\mathbf{x})/\partial x_j\, \partial x_i$, the Hessian matrix is symmetric: $H_{ij}(\mathbf{x}) = H_{ji}(\mathbf{x})$.

For example, for the following function of two variables,

$$f(\mathbf{x}) = e^{x_1}(4x_1^2 + 4x_1x_2 + 2x_2^2), \qquad [3]$$

the gradient vector is

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 2e^{x_1}(2x_1^2 + 2x_1x_2 + x_2^2 + 4x_1 + 2x_2) \\ 4e^{x_1}(x_1 + x_2) \end{bmatrix} \qquad [4]$$

and the Hessian is the matrix

$$H(\mathbf{x}) = \begin{bmatrix} 2e^{x_1}(2x_1^2 + 2x_1x_2 + x_2^2 + 8x_1 + 4x_2 + 4) & 4e^{x_1}(1 + x_1 + x_2) \\ 4e^{x_1}(1 + x_1 + x_2) & 4e^{x_1} \end{bmatrix}. \qquad [5]$$

Just as in the one-dimensional case the derivative defines the slope of the tangent line to the curve $f(x)$, the gradient vector at $\mathbf{x}$ represents the normal to the tangent hyperplane at the point $\mathbf{x}$. The term *hyperplane* is an extension to $n > 3$ of a plane in three dimensions. All vectors $\mathbf{x},\mathbf{y}$ satisfying $\mathbf{x}^T\mathbf{y} = \gamma$ for some constant γ lie in a hyperplane in higher dimensions. Thus, all vectors $\mathbf{y}$ satisfying $\mathbf{g}(\mathbf{x})^T\mathbf{y} = \gamma$ where $\gamma = \mathbf{g}(\mathbf{x})^T\mathbf{x}$ lie in the tangent hyperplane at $\mathbf{x}$.

A point $\mathbf{x}^* \in \mathbf{R}^n$ is said to be a *strict local minimum* of f if there exists a number $\eta > 0$ such that $f(x^*) < f(\mathbf{x})$ for all $\mathbf{x} \neq \mathbf{x}^*$ within a given distance η from $\mathbf{x}^*$ (i.e., $\|\mathbf{x} - \mathbf{x}^*\| < \eta$). For a *weak minimum*, only $f(\mathbf{x}^*) \leq f(\mathbf{x})$ holds. The point $\mathbf{x}^*$ is a *global minimum* of f if $f(\mathbf{x}^*) < f(\mathbf{x})$ for all $\mathbf{x} \neq \mathbf{x}^*$, $\mathbf{x} \in \mathbf{R}^n$. Figure 1 illustrates these possibilities for a one-dimensional function.

Matrix Characteristics

There are several general characteristics of a matrix that are particularly useful for analysis of minimization algorithms. *Density* of a matrix is a measurement given by the ratio of the nonzero to zero matrix components. A matrix is said to be dense when this ratio is large and sparse when it is small. A sparse matrix may be structured (e.g., block diagonal, band) or unstructured (Figure 2).

A symmetric matrix A is said to be positive-definite if the quadratic form $\mathbf{u}^TA\mathbf{u} > 0$ for all nonzero vectors $\mathbf{u}$. Similarly, the symmetric matrix A is positive-semidefinite if $\mathbf{u}^TA\mathbf{u} \geq 0$ for all nonzero vectors $\mathbf{u}$. Positive-definite matrices have strictly positive eigenvalues. We classify A as negative-definite if $\mathbf{u}^TA\mathbf{u} < 0$ for all nonzero vectors $\mathbf{u}$. A is indefinite if $\mathbf{u}^TA\mathbf{u}$ is positive for some $\mathbf{u}$ and negative for others.

For example, the function $f(\mathbf{x})$ defined in Eq. [3] can be written as the product of e^{x_1} and a generalized quadratic function: $f(x_1,x_2) = e^{x_1}[\mathbf{x}^TA\mathbf{x}]$. The matrix $A = \left[\begin{smallmatrix} 4 & 2 \\ 2 & 2 \end{smallmatrix}\right]$ is positive-definite.

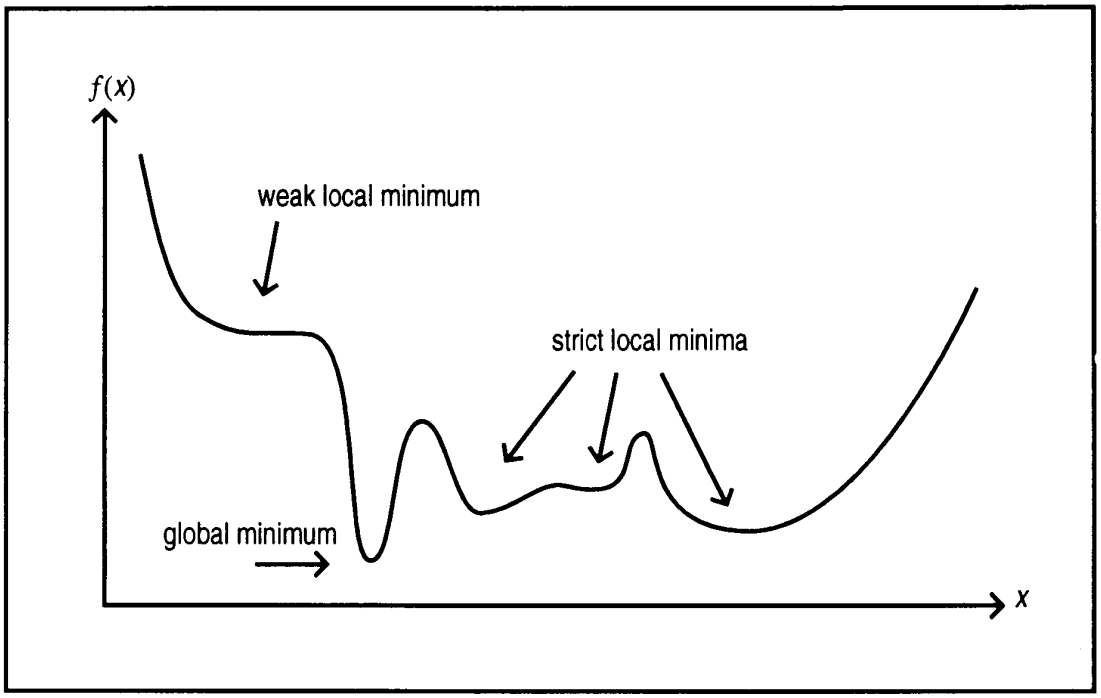

Figure 1 Types of minima.

The Hessian matrix is a generalization in $\mathbf{R}^n$ of the concept of curvature of a function. The positive-definiteness of the Hessian is a generalized notion of positive curvature. Thus, the properties of H are very important in formulating minimum-seeking algorithms.

Higher-order derivatives are rarely used in minimization methods; however, some recent approaches, termed *tensor*, have attempted to approximate higher-order information cheaply.[17]

Conditions at Minima

What are the sufficient conditions that must hold at a solution point of problem [2]? These conditions are simply extensions to $\mathbf{R}^n$ of the well-known first and second derivative conditions for univariate functions. Let us assume that $f(\mathbf{x})$ is a smooth function with continuous first, second, and third derivatives defined for all $\mathbf{x}$. We further suppose that the following conditions hold for the gradient and Hessian of f at $\mathbf{x}^* \in \mathbf{R}^n$:

(i) $\mathbf{g}(\mathbf{x}^*) = 0$,
(ii) $H(\mathbf{x}^*)$ is positive-definite.

Then it can be easily shown that $\mathbf{x}^*$ is a strict local minimum of f.

The Taylor expansion of $f(\mathbf{x})$ about $\mathbf{x}^*$ along a perturbation vector $\mathbf{p} = (\Delta x_1, \Delta x_2, \ldots, \Delta x_n)$ produces

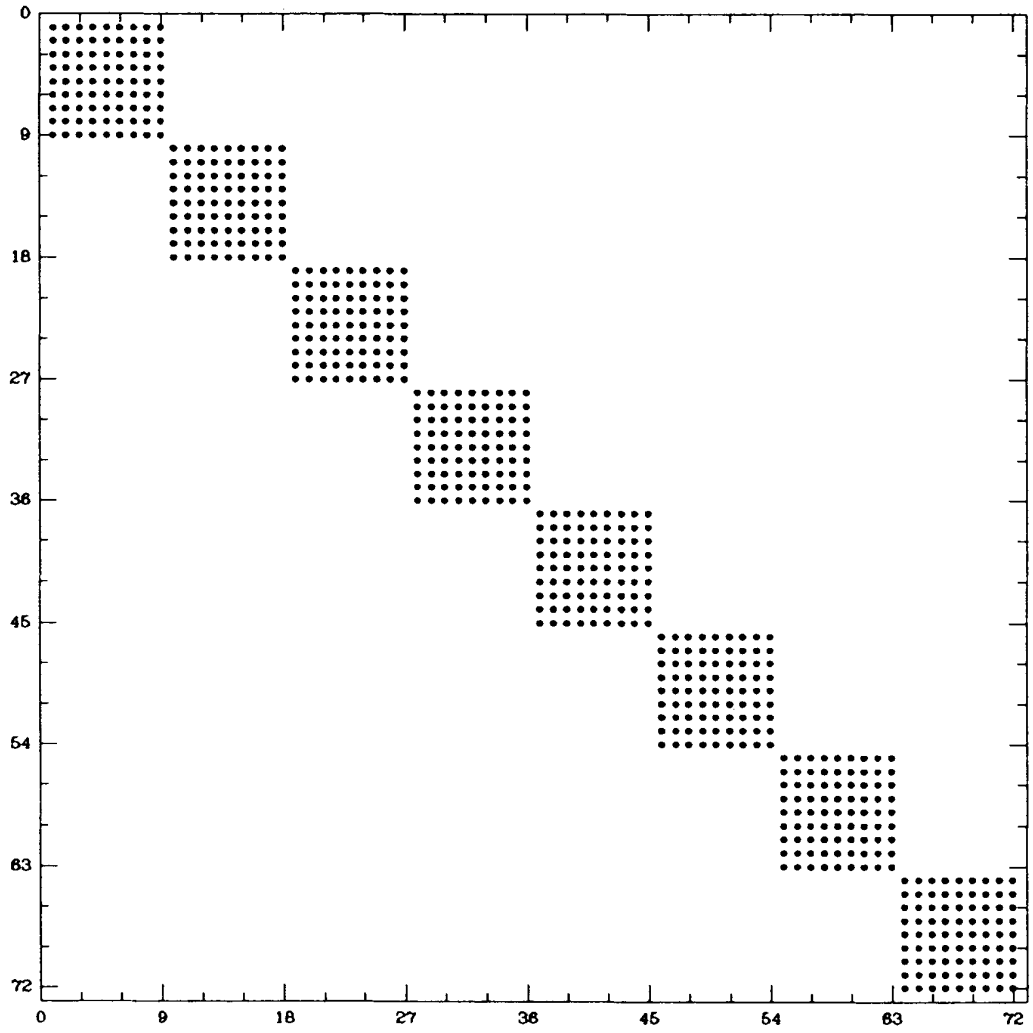

Figure 2 Sample matrix patterns for (**a**) block diagonal and (**b–e**) sparse unstructured. Pattern (**b**) corresponds to the Hessian approximation (preconditioner) for a potential energy model from the local energy terms (bond length, bond angle, and dihedral angle terms), and (**c**) is a reordered matrix pattern that reduces fill-in during the factorization. Pattern (**d**) comes from a molecular dynamics simulation of supercoiled DNA[36] and describes pairs of points along a ribbonlike model of the duplex that come in close contact during the dynamics trajectory; pattern (**e**) is the associated reordered structure that reduces fill-in.

$$f(\mathbf{x}^* + \mathbf{p}) = f(\mathbf{x}^*) + \sum_{i=1}^{n} (\Delta x_i) \frac{\partial f(\mathbf{x}^*)}{\partial x_i} + \frac{1}{2} \sum_{i=1}^{n} (\Delta x_i^2) \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_i^2} + \sum_{i=1}^{n} \sum_{j>i} (\Delta x_i)(\Delta x_j) \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_i \partial x_j} + \text{higher order terms.} \quad [6]$$

More compactly, this expansion can be written as

$$f(\mathbf{x}^* + \mathbf{p}) = f(\mathbf{x}^*) + \mathbf{g}(\mathbf{x}^*)^{\mathrm{T}}\mathbf{p} + \tfrac{1}{2}\mathbf{p}^{\mathrm{T}}H(\mathbf{x}^*)\mathbf{p} + O(\|\mathbf{p}\|^3). \quad [7]$$

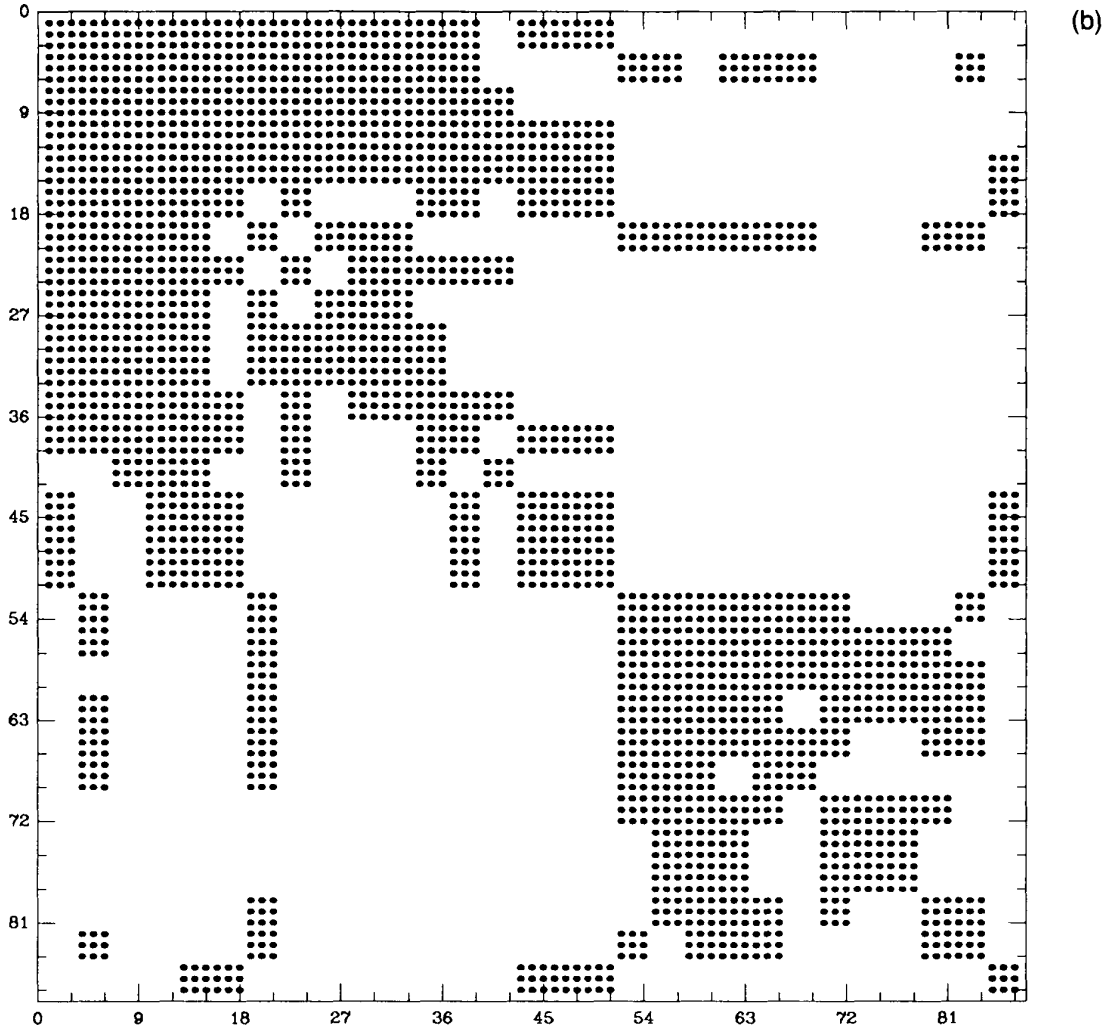

Figure 2 *(continued)*

As condition (i) holds, the gradient term vanishes, and condition (ii) implies that there exists some $\alpha > 0$ for which $\mathbf{p}^T H(\mathbf{x}^*)\mathbf{p} > \alpha\|\mathbf{p}\|^2$. It follows that we can write

$$f(\mathbf{x}^* + \mathbf{p}) - f(\mathbf{x}^*) \geq \frac{\alpha}{2}\|\mathbf{p}\|^2 + O(\|\mathbf{p}\|^3). \qquad [8]$$

For arbitrary small perturbation vectors $\|\mathbf{p}\|$, the first term on the right dominates the second; consequently, the right-hand side is positive. By definition, $\mathbf{x}^*$ is a strict local minimum of f. □

A point $\mathbf{x}^*$ is called a stationary point of f if $\mathbf{g}(\mathbf{x}^*) = 0$ but $H(\mathbf{x}^*)$ is not necessarily positive-definite. Thus, local and global minima are stationary points, but there are more general stationary points, such as saddle points, which are neither local nor global minima. Special techniques are needed for detection of saddle points, which are often related to structural transitions in molecular applications.

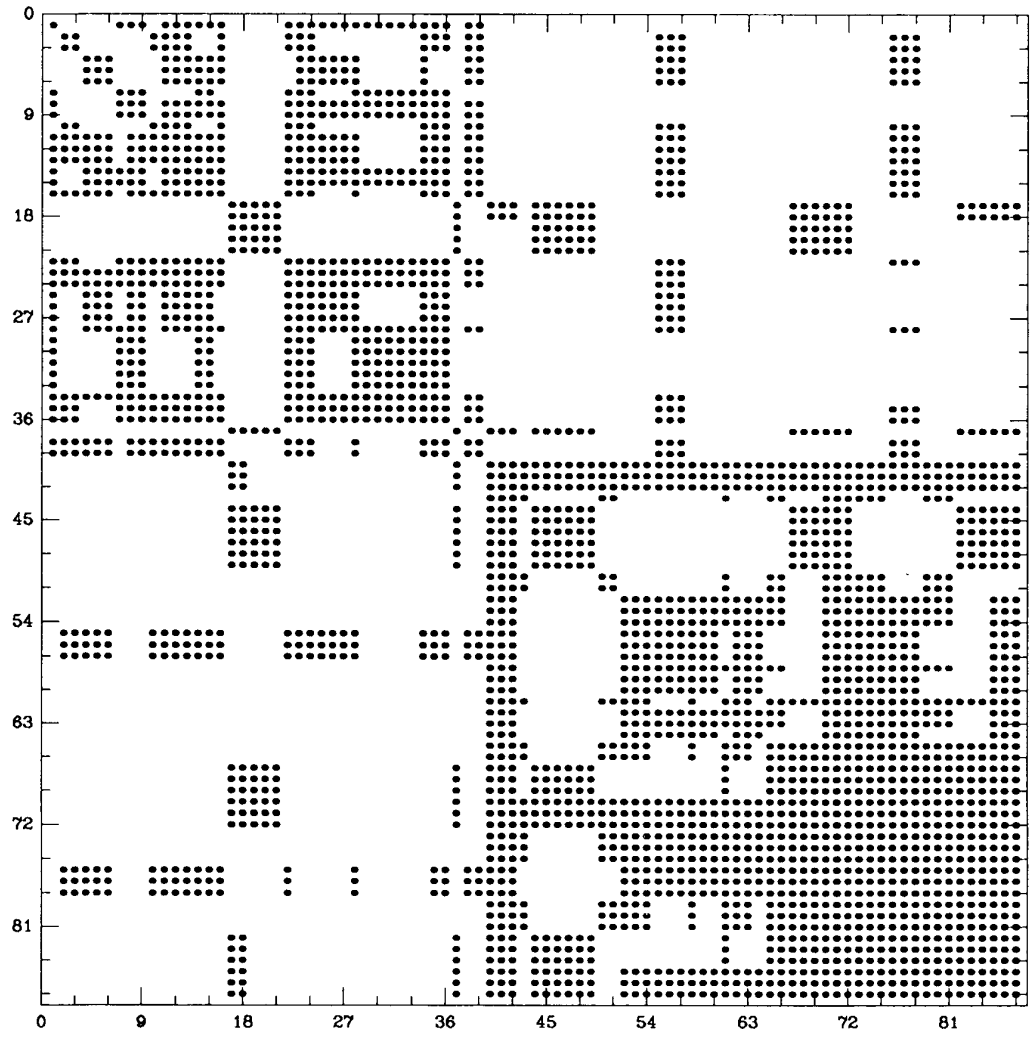

Figure 2 *(continued)*

Figure 3 illustrates the different types of stationary points for one-dimensional functions. The minimum and maximum at the origin are shown, respectively, for a convex and concave quadratic function. Note that the traces in the x,z and y,z planes are parabolas, opening upward (above origin) in the case of the minimum and opening downward (below origin) for the maximum. Elliptical cross sections can be noted in the x,y planes. The saddle point at the origin exhibits parabolic traces in the x,z and y,z planes, opening upward and downward, respectively. The trace in the x,y plane is a pair of intersecting lines.

Analysis of Functions

The positive-definiteness of H is a useful concept in analysis of general functions. Smooth functions can be approximated by quadratic models within a sufficiently small neighborhood of a given point. The local behavior of f can then be analyzed in terms of the properties of H.

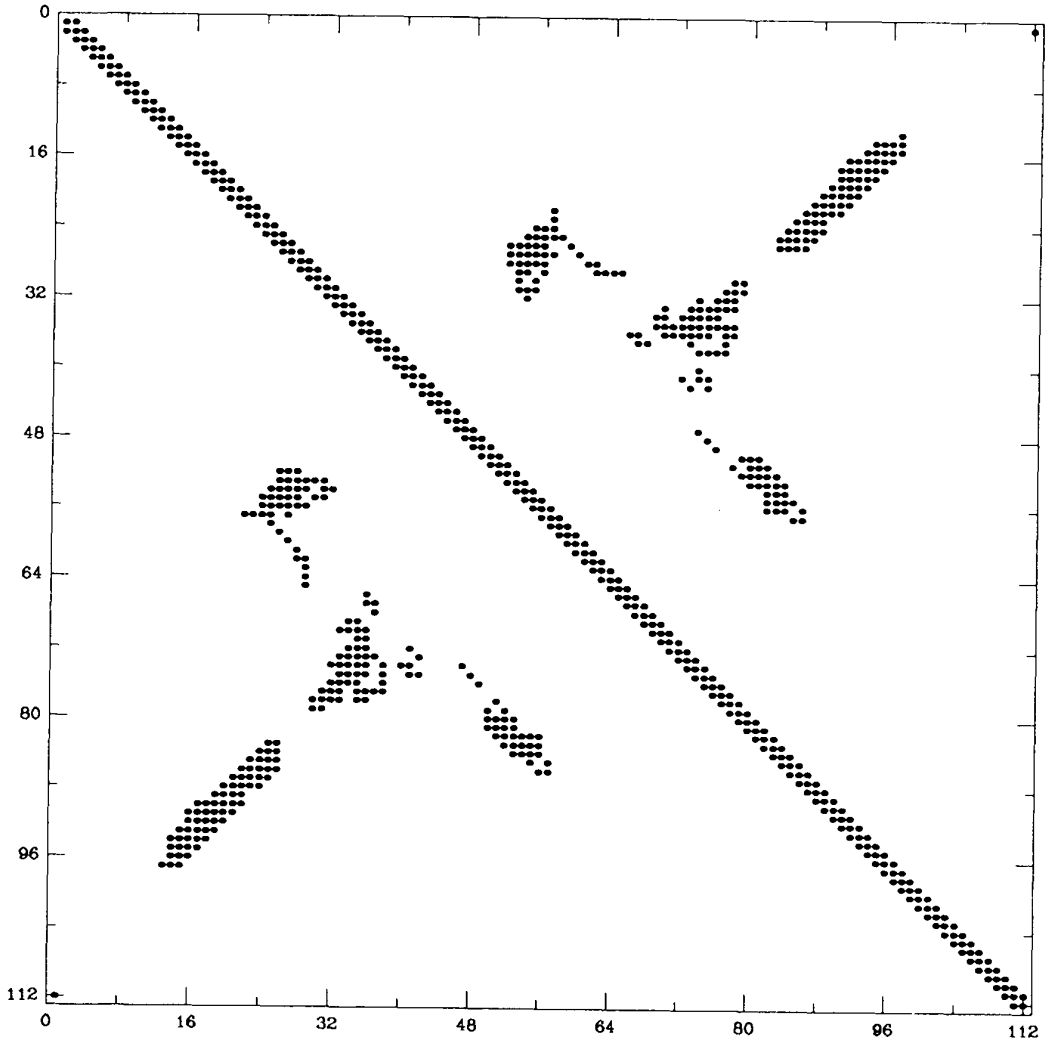

Figure 2 *(continued)*

To see this, consider the Taylor expansion of a quadratic function q about a stationary point $\mathbf{x}^*$:

$$q(\mathbf{x}^* + \mathbf{p}) = q(\mathbf{x}^*) + \tfrac{1}{2}\mathbf{p}^{\mathrm{T}}H(\mathbf{x}^*)\mathbf{p}. \qquad [9]$$

The symmetry of H implies that the n distinct eigenvectors $\{\mathbf{v}_i\}$ associated with the eigenvalues $\{\lambda_i\}$ are orthonormal (i.e., $\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j = 0$ for $i \neq j$, and $\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_i = 1$ for all i). We can then express $\mathbf{p}$ as the linear combination of eigenvectors

$$\mathbf{p} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \qquad [10]$$

for a scalar set $\{\alpha_i\}$. As $\mathbf{v}_i^{\mathrm{T}}H\mathbf{v}_i = \lambda_i$ for each i, the difference in function value caused by a movement along $\mathbf{p}$ is given by

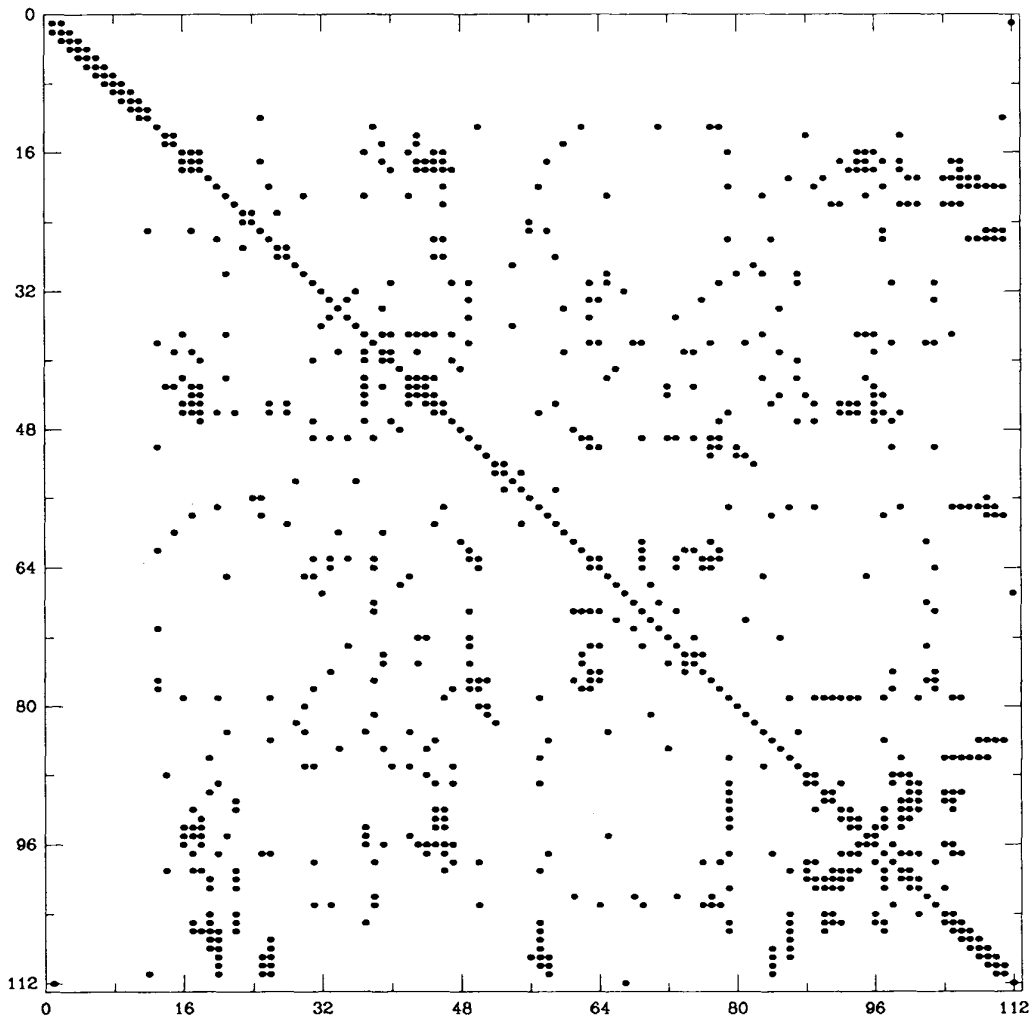

Figure 2 *(continued)*

$$q(\mathbf{x}^* + \mathbf{p}) - q(\mathbf{x}^*) = \frac{1}{2} \sum_{i=1}^{n} \alpha_i^2 \lambda_i. \qquad [11]$$

Clearly, the signs of the eigenvalues influence the change in q along particular directions. For example, when $\mathbf{p} = \alpha \mathbf{v}_i$ for $\alpha > 0$ and some i, q will be strictly increasing if $\lambda_i > 0$ and strictly decreasing if $\lambda_i < 0$ as α increases, respectively. If $\lambda_i = 0$, q is a constant along directions parallel to $\mathbf{v}_i$ (because $H\mathbf{v}_i = 0$) and reduces to a linear function along this direction (the quadratic term in Eq. [9] vanishes). When all the eigenvalues are positive, $\mathbf{x}^*$ is a unique global minimum of q. If H is positive-semidefinite (i.e., has nonnegative eigenvalues), $\mathbf{x}^*$ is a *weak* minimum. If H is indefinite and nonsingular (i.e., no eigenvalues are zero), the critical point $\mathbf{x}^*$ is a saddle point.

For example, the two-dimensional function defined in Eq. [3] has a local minimum at (0,0) and a saddle point at (−2,2). Both points are stationary. At

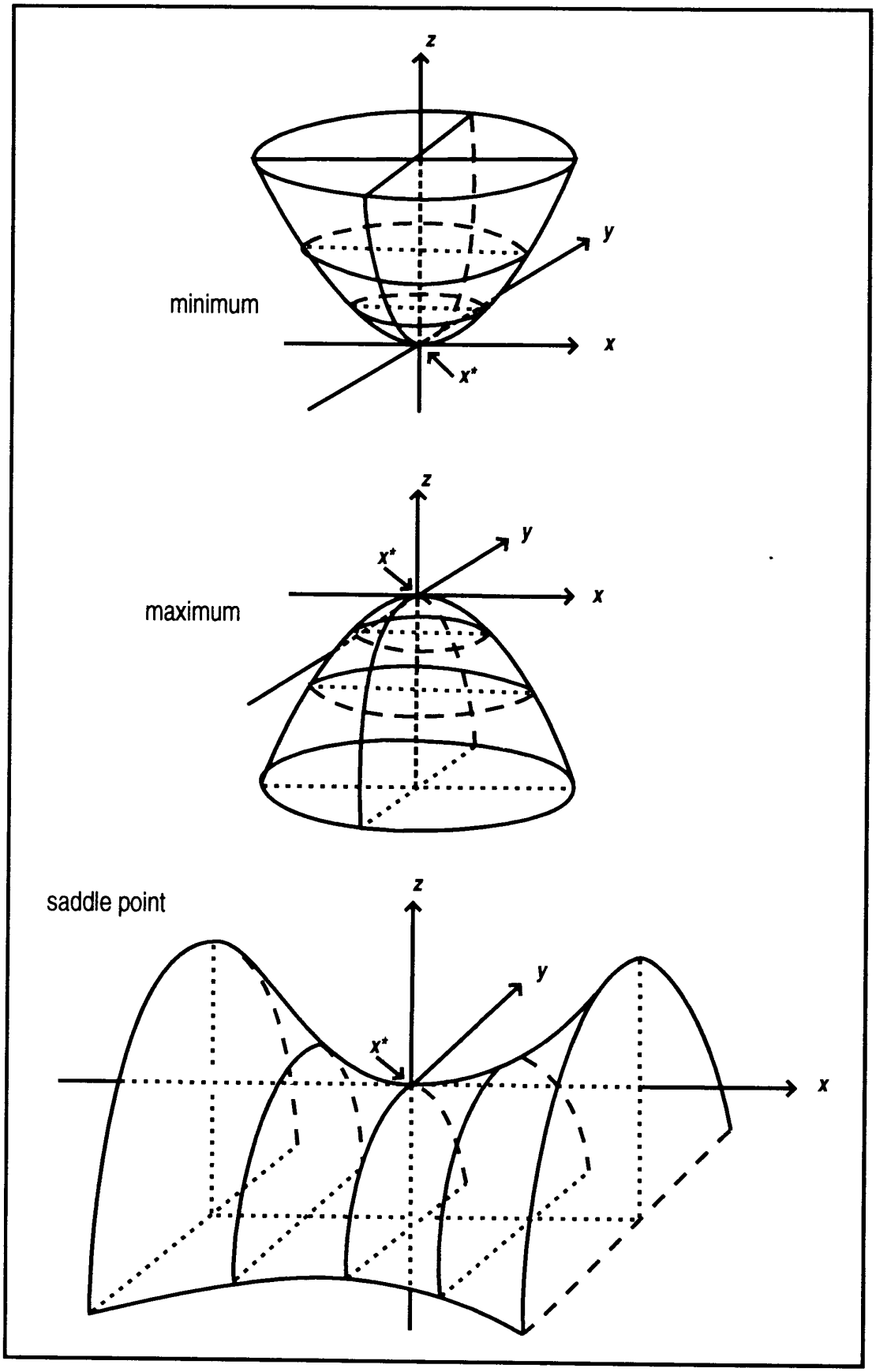

Figure 3 Types of stationary points.

(0,0), $f = 0$ and the Hessian is $\left[\begin{smallmatrix} 8 & 4 \\ 4 & 4 \end{smallmatrix}\right]$, with determinant (product of eigenvalues) of 16 and eigenvalues $6 \pm \sqrt{20}$, or about 10.5 and 1.5. At $(-2,2)$, $f = 8e^{-2} \approx 1.1$, and the Hessian

$$\begin{bmatrix} 0 & 4e^{-2} \\ 4e^{-2} & 4e^{-2} \end{bmatrix}$$

has determinant $-16e^{-4} < 0$ and eigenvalues $2e^{-2}(1 \pm \sqrt{5})$, or about 0.88 and -0.34. The reader can verify that the function $f(\mathbf{x}) = e^{x_1}(x_1 - 2x_2)^2$ has weak local minima for all points of the form $x_1 = 2x_2$, because the Hessian at those points,

$$\begin{bmatrix} 2e^{x_1} & -4e^{x_1} \\ -4e^{x_1} & 8e^{x_1} \end{bmatrix},$$

has eigenvalues 0 and $10e^{x_1} > 0$.

Contour plots of two-dimensional functions help illustrate these concepts. In general, the equation $f(\mathbf{x}) = \gamma$ defines a surface in $\mathbf{R}^{n+1}$. When $n = 2$, the plane curves corresponding to various values of γ generate contour plots (or maps) of a function. Figure 4 shows the contour plots for the two-dimensional functions discussed before. Note, for the first, the two stationary points corresponding to a minimum and saddle point. For the second, note the region of weak local minima. The contour plots are shaded so that darker areas correspond to higher function values.

Figure 5 illustrates more generally various cases that can occur for simple quadratic functions of form $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H\mathbf{x}$, for $n = 2$, where H is a constant matrix. The contour plots display different characteristics when H is (a) positive-definite (elliptical contours with lowest function value at the center) and q is said to be a convex quadratic, (b) positive-semidefinite, (c) indefinite, or (d) negative-definite (elliptical contours with highest function value at the center), and q is a concave quadratic. For this figure, the following matrices are used for those different functions:

(a) $H = \left[\begin{smallmatrix} 4 & 2 \\ 2 & 2 \end{smallmatrix}\right]$, with two positive eigenvalues: $\lambda = 3 \pm \sqrt{5}$.

(b) $H = \left[\begin{smallmatrix} 4 & 2 \\ 2 & 1 \end{smallmatrix}\right]$, with the one zero and one positive eigenvalue: $\lambda = 0, 5$.

(c) $H = \left[\begin{smallmatrix} 2 & \sqrt{6} \\ \sqrt{6} & 1 \end{smallmatrix}\right]$, with one negative and one positive eigenvalue: $\lambda = -1, 4$.

(d) $H = \left[\begin{smallmatrix} -4 & 0 \\ 0 & -1 \end{smallmatrix}\right]$, with two negative eigenvalues: $\lambda = -2, -1$.

As Figure 4 shows, nonlinear functions produce contour maps that are complex composites of these pure quadratic cases.

Contour plots for $n = 2$ also help illustrate paths toward minima specified by various minimization methods. The gradient vector is orthogonal to the contour lines. The familiar notion in one dimension that the negative tangent vector at a point x points toward the minimum of a convex quadratic extends naturally to higher dimensions. Thus, if the contour plots are circular

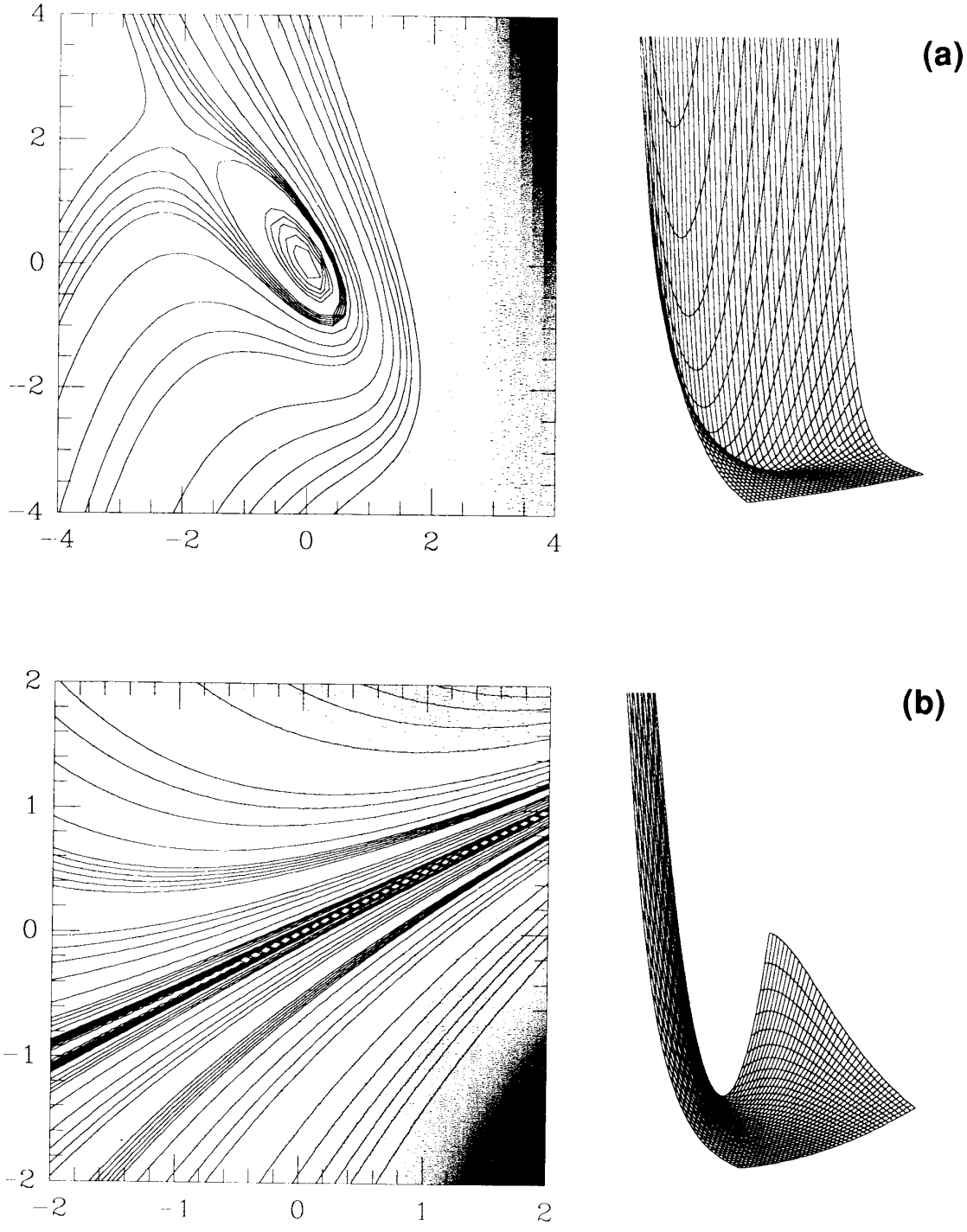

Figure 4 Sample contour plots and partial three-dimensional views of nonlinear two-dimensional functions: (**a**) $f(x_1,x_2) = e^{x_1}(4x_1^2 + 4x_1x_2 + 2x_2^2)$; (**b**) $f(x_1,x_2) = e^{x_1}(x_1 - 2x_2)^2$.

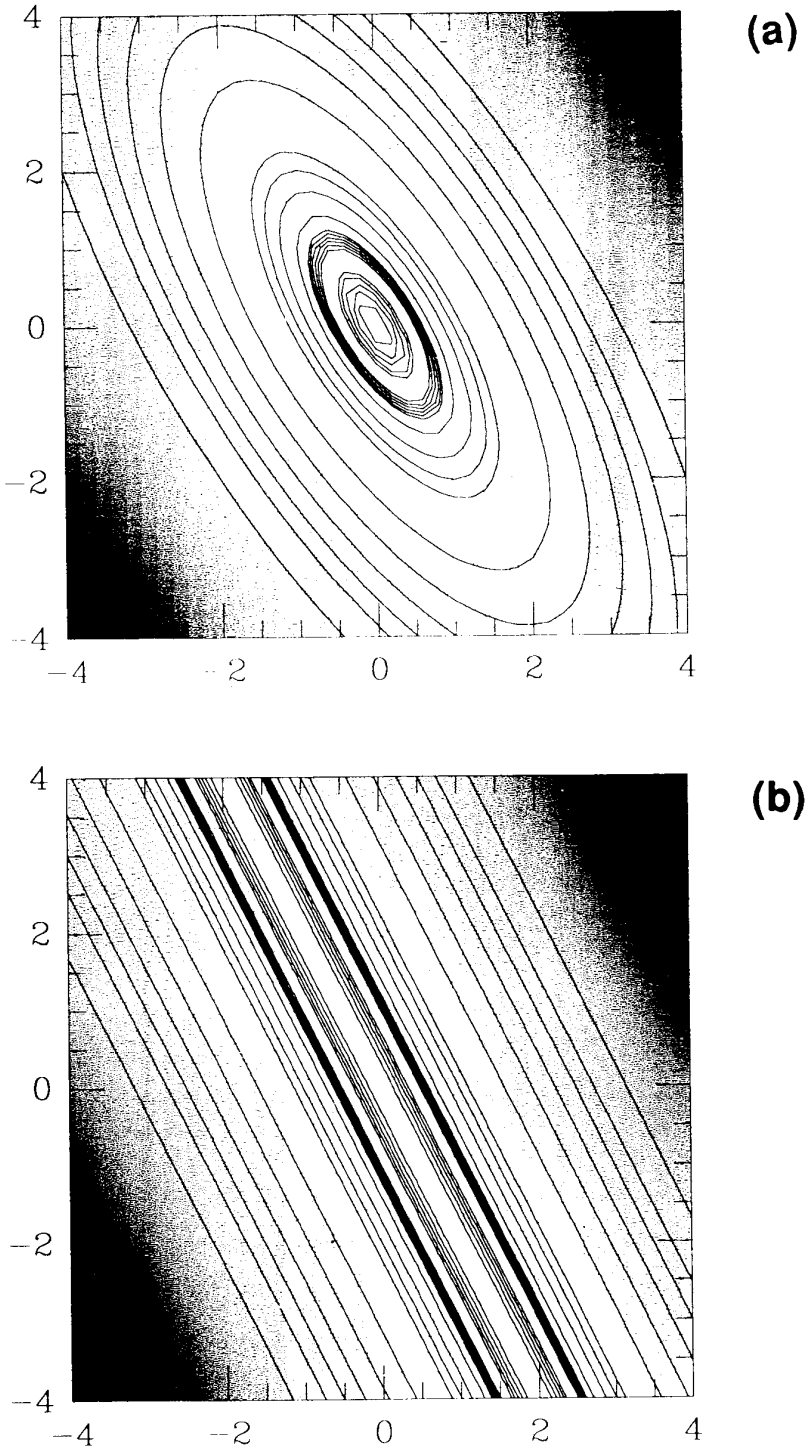

Figure 5 Sample contour plots for quadratic two-dimensional functions $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H\mathbf{x}$, where H is (**a**) positive-definite, (**b**) positive-semidefinite, (**c**) indefinite, or (**d**) negative-definite.

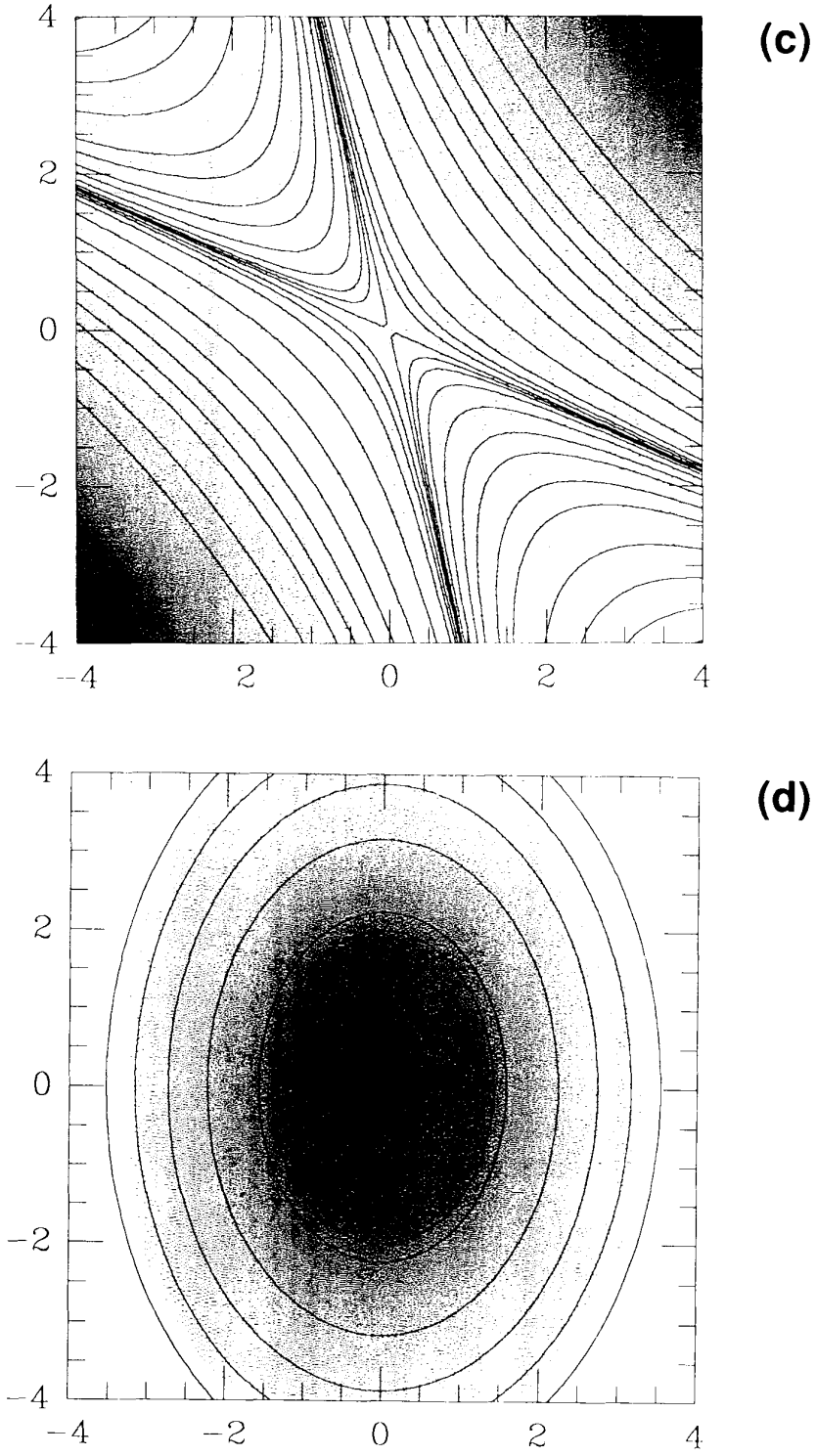

Figure 5 *(continued)*

(e.g., $q(\mathbf{x}) = \mathbf{x}^T H\mathbf{x}$, where H is a positive multiple of the identity matrix), a movement along the gradient will find the minimum in one step. More generally, the shape of the elliptical contours near local minima depends on the eigenvalues $\{\lambda_i\}$ and eigenvectors $\{\mathbf{v}_i\}$ of the Hessian in that neighborhood. The axes of the elliptical contours point in the direction of the orthogonal eigenvectors, and the length of each axis corresponding to the ith eigenvector is proportional to $1/\lambda_i$. Thus for $f(\mathbf{x})$ defined in Eq. [3], $\lambda_1 \approx 7\lambda_2$ near the origin, and the elliptical axes are elongated proportionately (Figure 4). In Figure 5d, for example, the axes of the ellipses are along the basis vectors, the eigenvectors corresponding to the diagonal matrix.

BASIC APPROACHES TO LARGE-SCALE OPTIMIZATION

Size and Space Limitations

The task of minimizing potential energy functions arising in molecular mechanics is typical of optimization applications seeking favorable configurational states of a physical system. [18–23] The sheer size of configuration space and complexity of the system introduce two major problems: extensive computational requirements and the multiple-minima problem.

Computational intensity often stems from the long-range interactions among the N particles in the system (e.g., Coulombic forces). Their direct evaluation requires order of N^2 operations. Even if a cutoff radius is introduced, computation of the nonbonded terms dominates computation time. Thus, implementation of fast particle methods[24,25] in molecular mechanics and dynamics calculations may be important for reducing the severity of this problem in the future. For now, if we assume order of N iterations for a minimization algorithm, the computational complexity will be $O(N^3)$, a dimension barely manageable for large biological systems, even on supercomputers. Rapid and reliable performance in practice is then a central focus in selection of suitable minimization algorithms.

The multiple-minimum problem is a severe handicap of many large-scale optimization applications. The state of the art today is such that for reasonable small problems (30 variables or less) suitable algorithms exist for finding all local minima for linear and nonlinear functions. For larger problems, however, many trials are generally required to find local minima, and finding the global minimum cannot be ensured. These features have prompted research in conformational-search techniques independent of, or in combination with, minimization.[26]

Search Techniques

Consider, for example, the two different energy minima of *n*-butane, corresponding to the anti and gauche conformers, as a function of the dihedral

angle about the central C—C bond (Figure 6). This representation is a simplification from the full space of 36 degrees of freedom ($3N - 6$ where $N = 14$ atoms) but suffices for partitioning the sampling space approximately. When we scale up the model to alkane chains of m residues, the number of possible starting points obtained from combinations or rough partitions in monomer structure produces 3^m starting configurations. For polypeptides and polynucleotides, the flexibility of the monomer configurations increases, producing a rough range of 10^m to 25^m reasonable starting points by coarse subdomain partition (e.g., combinations of typical side-chain, main-chain, backbone, or sugar dihedral angles). Exhaustive searches are clearly not feasible.

The buildup technique is a related configurational search strategy, used mostly in protein studies[27,28] and more recently in nucleic acids.[29] Reasonable starting points are constructed by combining minima of conformational building blocks. This rational strategy has performed rather well in practice but provides no guarantee that all biologically important local minima, much less the global minimum, are revealed. As computational time tends to be large for biopolymers, the simplest stochastic global optimization method, simulated annealing,[8,9] may be preferable for very large systems.

Molecular dynamics can also be viewed as a complementary technique for obtaining structural information to potential energy minimization.[30] Although in theory information on all thermally accessible states should be observable, the restriction of the integration time step to a very small value with respect to time scales of collective biomolecular motions[30,31] limits the scope of

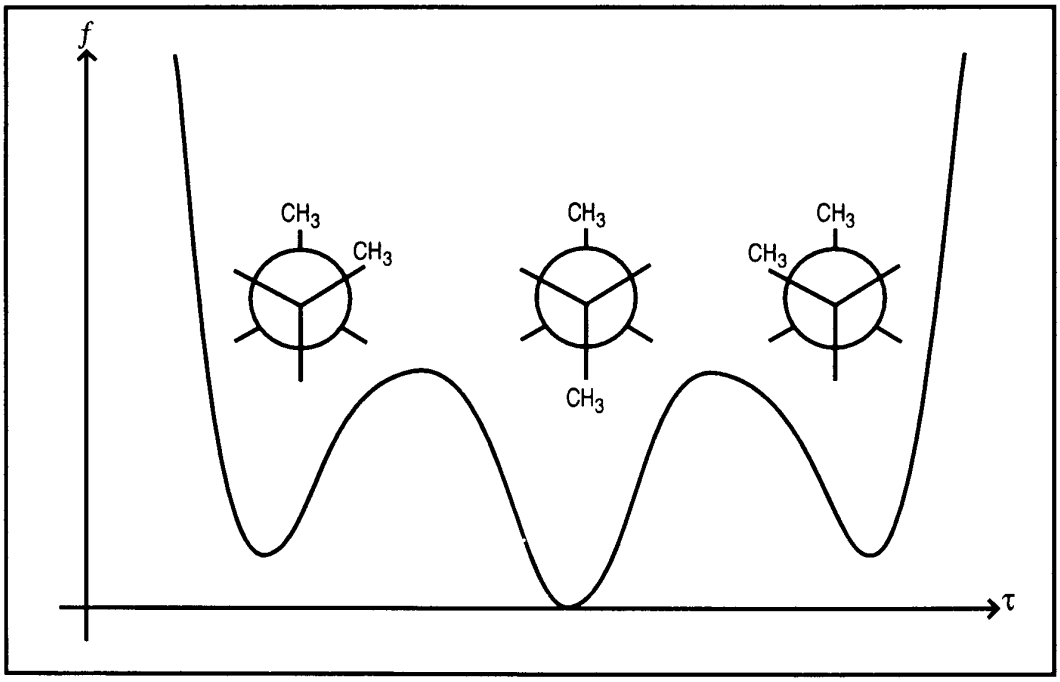

Figure 6 Potential energy of *n*-butane as a function of the dihedral angle about the central C—C bond.

molecular dynamics in practice. Implicit numerical schemes[32–36] or other special techniques[37] may reduce the severity of this problem in the future. Typical molecular dynamics simulations today are effective for refining low-resolution crystal structures and for observing small fluctuations around equilibrium.[30] They are often also effective in combination with minimization as "annealing"-like procedures for complex systems.[38]

Together, these general considerations of sampling and complexity have led to two broad classes of multivariate minimization algorithms: local and global (see Figure 7).

Local and Global Methods

Local methods, the focus of this chapter, are essentially descent methods. They are defined by an iterative procedure: $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots\}$ that attempts to find one local minimum $\mathbf{x}^*$ from a given $\mathbf{x}_0$. In each step, a search vector $\mathbf{p}_k$ is computed by a given strategy, and f is minimized approximately along that direction so that "sufficient decrease" is obtained (see next section). Thus the structure of local methods is basically a "greedy" one: effort is directed toward a nearby local minimum, and steps that may increase the function are not permitted (Figure 8). Their performance is clearly sensitive to the choice of starting point in addition to search direction and algorithmic details.

Global optimization methods attempt to remedy the deficiency of local methods by exploring larger regions of function space. The global minimum of a function can be sought through two classes of approaches: deterministic and stochastic. Deterministic methods usually require the objective function to satisfy certain smoothness properties; they construct a sequence of points converging to lower and lower local minima. Ideally, they attempt to "tunnel" through local barriers, as shown in the schematic path in Figure 7. Computational effort tends to be very large and guarantee of success can only be obtained under specific assumptions. Local minimization methods are often required repeatedly in the framework. Although a variety of interesting algorithmic approaches are currently under investigation,[11,26,39–42] the field of deterministic global optimization is still in its infancy in terms of general large-scale applicability. Nonetheless, by exploiting massively parallel environments for algorithm design,[43,44] we can anticipate much progress in the coming years.

Stochastic global methods, on the other hand, involve systematic manipulation of randomly selected points (Figure 7).[11,42,45,46] Success can be guaranteed only in an asymptotic, stochastic sense, although in practice many applications are very promising. The simulated annealing method is a popular and effective technique for small-to-medium molecular systems.[8,9,47–50] Simulated annealing is very easy to implement and generally requires no derivative computations. It is often a useful technique when the energy formulation is complex, as in macroscopic models of supercoiled DNA.[51]

Local optimization methods have experienced far more extensive development in the last decade. Studies have produced a range of robust and reliable

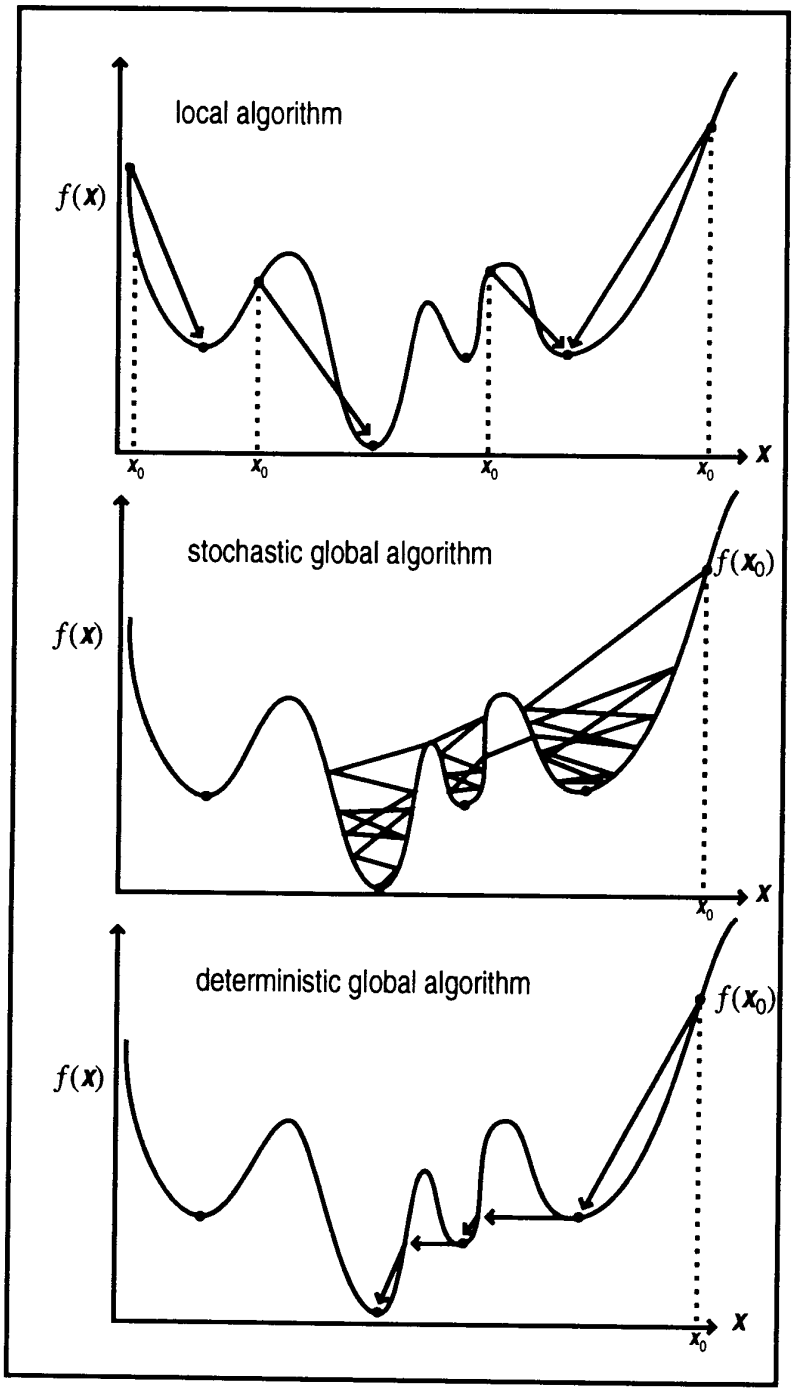

Figure 7 Structure of local and global minimization algorithms.

techniques tailored to problem size, smoothness, complexity, and memory considerations. Newton methods, for example, have produced many variants that effectively lift their original application scope to small or sparse problems only.

In the case of potential energy functions, unconstrained optimization problems can generally be formulated for large, nonlinear, and smooth functions. Obtaining first and second derivatives may be tedious but is definitely

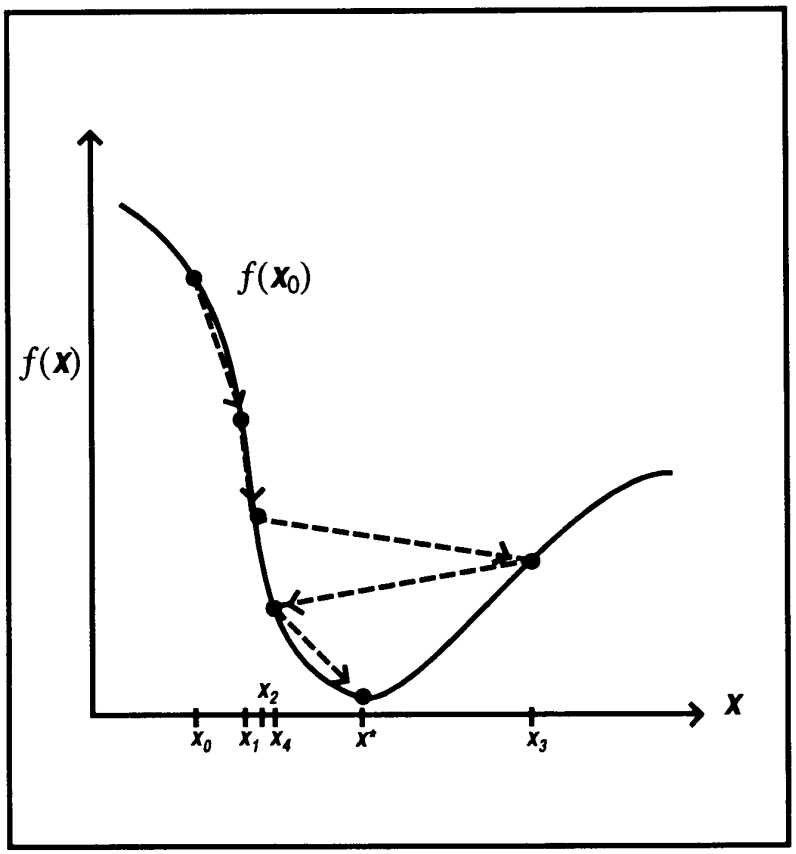

Figure 8 Descent structure of local minimization algorithms.

feasible. This additional effort is in fact profitable overall, because a gain in performance will often be realized. As storage and cost considerations are directly related to the function complexity and derivative calculations, the choice of method for potential energy minimization should be based on the extent of possible derivative manipulations (e.g., first derivatives, first and second derivatives).

After a brief discussion on the structure of descent methods, we highlight general concepts in the three categories of local methods for unconstrained nonlinear functions: nonderivative, first derivative (or gradient), and second derivative methods.

BASIC DESCENT STRUCTURE OF LOCAL METHODS

The fundamental structure of local iterative techniques for solving unconstrained minimization problems is simple. A starting point is chosen, a direction of movement is prescribed according to some algorithm, and a *line search*

or *trust region* approach is performed to determine an appropriate next step. The process is repeated at the new point and the algorithm continues until a local minimum is found. Schematically, a model descent method can be written as follows:

Algorithm [A1]: Basic Descent

* Supply an initial guess $\mathbf{x}_0$.
* For $k = 0,1,2, \ldots$, until convergence
  1. Test $\mathbf{x}_k$ for convergence. [A1]
  2. Calculate a search direction $\mathbf{p}_k$.
  3. Determine an appropriate step length λ_k (or modified step $\mathbf{s}_k$).
  4. Set $\mathbf{x}_{k+1}$ to $\mathbf{x}_k + \lambda_k \mathbf{p}_k$ (or $\mathbf{x}_k + \mathbf{s}_k$).

Descent Directions

It is reasonable to choose a search vector $\mathbf{p}_k$ that will be a descent direction, that is, a direction leading to function reduction. A descent direction $\mathbf{p}$ is defined as one along which the directional derivative is negative:

$$\mathbf{g}(\mathbf{x})^T\mathbf{p} < 0. \qquad [12]$$

When we write the approximation

$$f(\mathbf{x} + \lambda\mathbf{p}) - f(\mathbf{x}) \approx \mathbf{g}(\mathbf{x})^T\mathbf{p}, \qquad [13]$$

we see that the negativity of the right-hand side guarantees that a lower function value can be found along $\mathbf{p}$ for a sufficiently small λ.

Different descent methods are distinguished by their choice of search directions. The various strategies are discussed in the next sections.

Line Search and Trust Region Steps

Both line search and trust region methods are essential components of basic descent schemes for guaranteeing *global convergence*.[3,6,52–55] Of course, only one of the two methods is needed for a given minimization algorithm. This notion of convergence refers to obtaining a strict local minimum $\mathbf{x}^*$ from any given starting point $\mathbf{x}_0$ and not the global minimum of a function.

A line search consists of an approximate one-dimensional minimization of the objective function along the computed direction $\mathbf{p}_k$. This produces an acceptable step λ and a new iterate $\mathbf{x}_k + \lambda\mathbf{p}_k$. Function and gradient evaluations of the objective function are required in each line search iteration. In contrast, the trust region strategy minimizes approximately a local quadratic model of the function using current Hessian information. An optimal step that lies within

the trust region of the quadratic model is calculated, and the new iterate becomes $\mathbf{x}_k + \mathbf{s}_k$.

The trust radius in the trust region approach is estimated on the basis of the local Hessian's characteristics (positive-definite, positive-semidefinite, indefinite). The basic idea is to choose $\mathbf{s}_k$ nearly in the current negative gradient direction ($-\mathbf{g}_k$) when the trust radius is small, and approach the Newton step $-H_k^{-1}\mathbf{g}_k$ as the trust region is increased. (H_k and $\mathbf{g}_k$ denote the Hessian and gradient, respectively, at $\mathbf{x}_k$). Note from condition [12] that these two choices correspond to the extremal cases ($M = I$ and $M = H$) of general descent directions of form $\mathbf{p} = -M^{-1}\mathbf{g}$, where M is a positive-definite approximation to the Hessian.

Although line searches are typically easier to program, trust region methods may be effective when the procedure for determining the search direction $\mathbf{p}_k$ is not necessarily one of descent. This may be the case for methods that use finite-difference approximations to the Hessian in the procedure for specifying $\mathbf{p}_k$ (discussed in later sections). As we shall see later, in BFGS quasi-Newton or truncated Newton methods line searches may be preferable because descent directions are guaranteed.

In general, there has been no clear evidence for superiority of one approach over another. Indeed, both techniques are incorporated into minimization software with approximately equal success. Later we sketch the line search procedure, which we find easier to grasp and simpler to implement in practice. For further perspective and algorithmic details, the reader is referred to the recent review of Dennis and Schnabel and references cited therein.[54]

The line search is essentially an approximate one-dimensional minimization problem. It is usually performed by safeguarded polynomial interpolation.[5,6,54–56] That is, in a typical line step iteration, cubic interpolation is performed in a region of λ that ensures that the minimum of f along $\mathbf{p}$ has been bracketed. The minimum of that polynomial then provides a new candidate for λ. If the search directions are properly scaled, the initial trial point $\lambda_t = 1$ produces a first reasonable trial move from $\mathbf{x}_k$. A simple illustration of such a first line search step is shown in Figure 9. The minimized one-dimensional function at the current point $\mathbf{x}_k$ is defined by $\tilde{f}(\lambda) \equiv f(\mathbf{x}_k + \lambda\mathbf{p}_k)$. The vectors corresponding to different values of λ are set by $\tilde{\mathbf{x}}(\lambda) = \mathbf{x}_k + \lambda\mathbf{p}_k$.

At the first step, a cubic polynomial can be constructed from the two function values $\tilde{f}(0)$, $\tilde{f}(\lambda_t)$ and the two slopes $\tilde{g}(0)$,$\tilde{g}(\lambda_t)$. The slopes are the directional derivatives defined as $\tilde{g}(\lambda) = \mathbf{g}(\mathbf{x}_k + \lambda\mathbf{p}_k)^T\mathbf{p}_k$. Note in the figure a negative slope at $\lambda = 0$, as $\mathbf{p}_k$ is a descent direction.

In general, precautions are needed to ensure that a minimum for $\tilde{f}(\lambda)$ has been bracketed in the trial interval. For example (see Figure 10, for the first step), if (a) the new slope is positive ($\tilde{g}(\lambda_t) > 0$) or (b) the new function value is greater than the old ($\tilde{f}(\lambda_t) > \tilde{f}(0)$), then there is a relative minimum along $\mathbf{p}_k$ between $\tilde{\mathbf{x}}(0)$ and $\tilde{\mathbf{x}}(\lambda_t)$. We can proceed to find it by minimization of a cubic (or quadratic in special cases) interpolant; however, if neither of these conditions holds, the function has decreased and continues to decrease at $\tilde{\mathbf{x}}(\lambda_t)$.

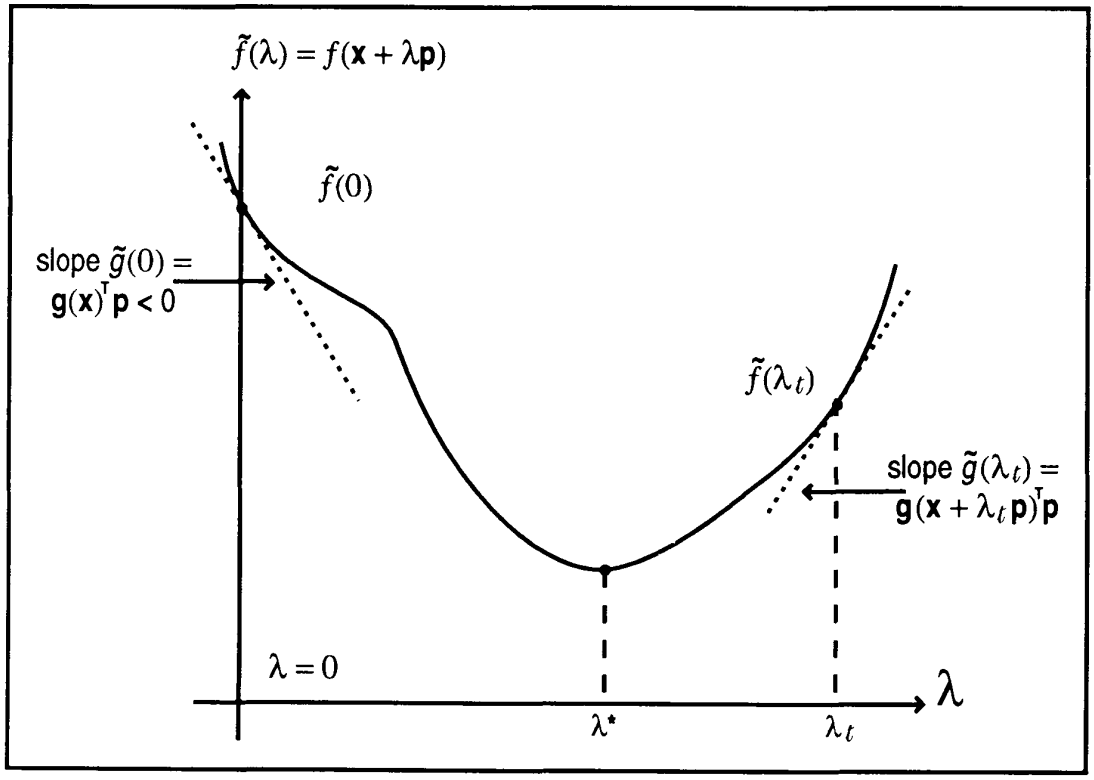

Figure 9 A one-dimensional line search for $\tilde{f}(\lambda) = f(\mathbf{x} + \lambda\mathbf{p})$ by cubic interpolation produces an approximate minimum along $\mathbf{p}$, at $\lambda^*, \tilde{f}(\lambda^*)$.

The step may then be too small and should be increased until a minimum is bracketed. Alternatively, the step may be too large and can be decreased until the new slope is positive (see Figure 10c). More generally, for the bracketed interval $[\lambda_1, \lambda_2]$ and corresponding function and slopes $\tilde{f}_1, \tilde{f}_2, \tilde{g}_1, \tilde{g}_2$, the cubic polynomial passing through $(\lambda_1, \tilde{f}_1)$ and $(\lambda_2, \tilde{f}_2)$ and having the specified slopes is given by

$$p(\lambda) = a(\lambda - \lambda_1)^3 + b(\lambda - \lambda_1)^2 + c(\lambda - \lambda_1) + d, \qquad [14]$$

where

$$a = [-2(\tilde{f}_2 - \tilde{f}_1) + (\tilde{g}_1 + \tilde{g}_2)(\lambda_2 - \lambda_1)]/(\lambda_2 - \lambda_1)^3$$
$$b = [3(\tilde{f}_2 - \tilde{f}_1) - (2\tilde{g}_1 + \tilde{g}_2)(\lambda_2 - \lambda_1)]/(\lambda_2 - \lambda_1)^2$$
$$c = \tilde{g}_1$$
$$d = \tilde{f}_1.$$

A minimum of $p(\lambda)$ can be obtained by setting

$$\lambda = \lambda_1 + \left[-b + \sqrt{b^2 - 3ac} \right] / 3a \qquad [15]$$

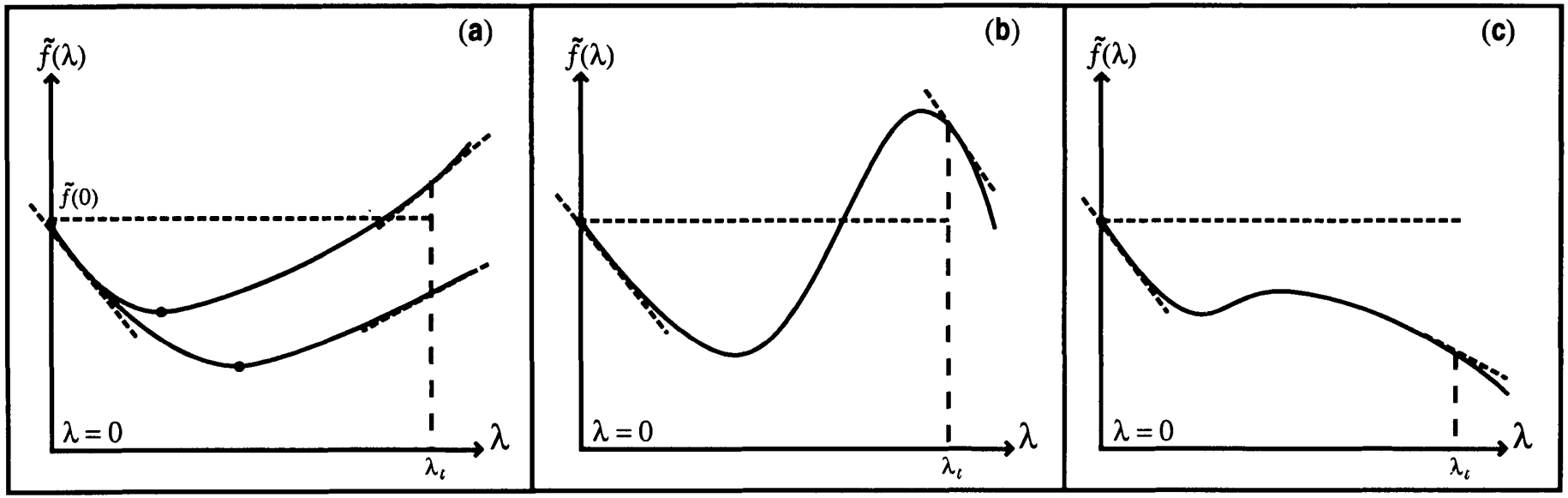

Figure 10 Possible situations in line search algorithms between the current and trial points: (a) The new slope is positive. (b) The new slope is negative but function value is greater. (c) The new slope is negative and function value is lower.

as long as $a \neq 0$ and $b^2 - 3ac \geq 0$. Otherwise, a quadratic interpolant fitted to $\tilde{f}_1, \tilde{g}_1$, and $\tilde{f}_2$ can be constructed with the same coefficients,

$$p(\lambda) = b(\lambda - \lambda_1)^2 + c(\lambda - \lambda_1) + d, \qquad [16]$$

and minimized to produce

$$\lambda = \lambda_1 - c/2b. \qquad [17]$$

The degenerate case of $b = 0$, corresponding to a linear function rather than a quadratic and redundancy among the three values $\{\tilde{f}_1, \tilde{f}_2, \tilde{g}_1\}$, is excluded by construction.

Once a new λ and corresponding trial point $\tilde{\mathbf{x}}(\lambda)$ have been determined in a line search iteration, conditions of sufficient progress with respect to the objective function are tested. If these conditions are not satisfied, a new value for λ is sought in another line search step of interpolation, following a backtracking strategy (i.e., reduction of λ_2).

The idea in these tests is to ensure "sufficient decrease" in the objective function for the new point relative to the step taken. These conditions balance the work in the line search procedure with the overall progress realized in the minimization method. The conditions often used in optimization algorithms are derived from the Armijo and Goldstein criterion.[6] They require that the inequalities

$$f(\mathbf{x}_k + \lambda \mathbf{p}_k) \leq f(\mathbf{x}_k) + \alpha \lambda \mathbf{g}(\mathbf{x}_k)^T \mathbf{p} \qquad [18a]$$

and

$$|\mathbf{g}(\mathbf{x}_k + \lambda \mathbf{p}_k)^T \mathbf{p}_k| \leq \beta |\mathbf{g}(\mathbf{x}_k)^T \mathbf{p}_k| \qquad [18b]$$

hold for two constants α, β, where $0 < \alpha < \beta < 1$. The first condition prescribes an upper limit on acceptable new function values. Note that condition [18a] requires that $f(\mathbf{x}_k + \lambda \mathbf{p}_k)$ lie on or below the line $l(\alpha) = f(\mathbf{x}_k) + \alpha \lambda \mathbf{g}(\mathbf{x}_k)^T \mathbf{p}$ (Figure 11). The line $l(1)$ corresponds to the directional derivative at $\mathbf{x}_k$, whereas the line $l(0)$ corresponds to $f(\mathbf{x}_k)$. Thus, the larger α is, the more stringent will be the condition of satisfactory decrease.

The second condition involves changes in the magnitude of the directional derivatives. The smaller β is, the greater the accuracy in determining an optimal λ for a stationary point of f along $\mathbf{p}_k$. Condition [18b] can also be interpreted as a lower bound on λ. In other words, the optimal λ should not be too small (see Figure 11).

Typical values of α and β in line search algorithms are $\alpha = 10^{-4}$ and $\beta = 0.9$. With this combination (small α and large β), usually all choices of λ that satisfy condition [18a] satisfy condition [18b] as well. The combination selected within a framework of an optimization method is important for balanc-

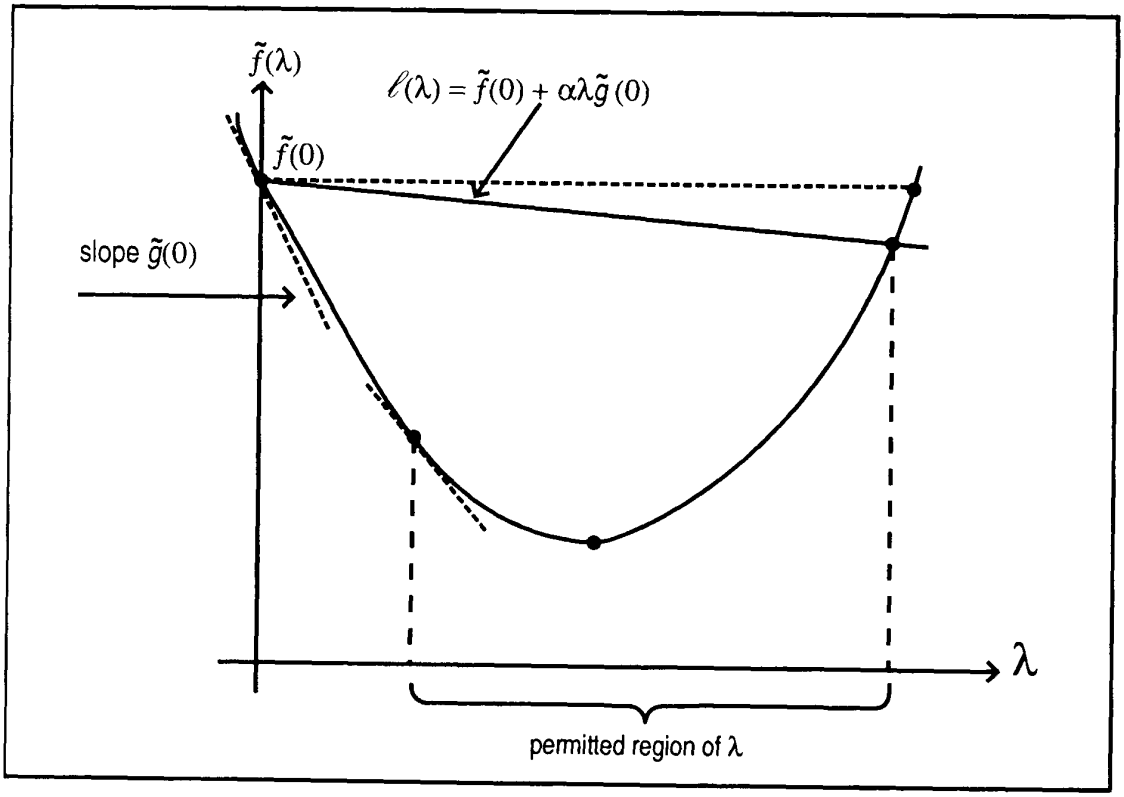

Figure 11 Line search acceptance conditions.

ing the effort expended in each line search phase with the overall progress realized toward a minimum. More accurate line searches, for example, can be formulated with smaller values of β (e.g., 0.5 or 0.2). As we will see, this control of accuracy is important for nonlinear conjugate gradient algorithms. In particular, if the computational effort during the line search is low relative to the work required for determining the search direction (e.g., function and gradient versus Hessian calculations), more accurate line searches may be beneficial; however, if the target problems are complex and the initial guess is not close to a minimum, accurate line searches may be unwarranted.

Convergence Criteria

In the basic descent algorithm [A1], we iterate "until convergence." How exactly do we evaluate the optimality of an approximate minimum $\mathbf{x}^*$? Furthermore, how do we ensure that computations will not continue unnecessarily (a) when no further progress can be realized or (b) beyond attainable accuracy? The accuracy depends on machine precision and cumulative roundoff errors, in addition to algorithmic details.

The simplest test for optimality of each $\mathbf{x}_k$ involves the size of the corresponding gradient. One such test can be formulated as follows:

$$\|\mathbf{g}_k\| \leq \epsilon_g(1 + |f(\mathbf{x}_k)|). \qquad [19]$$

The parameter ϵ_g is a small positive number such as 10^{-8}. A reasonable choice is around the order of the square root of machine precision, ϵ_m, defined as the smallest number x such that the floating point value of $(1 + x)$ is greater than the floating representation of 1. This precision-dependent (i.e., double versus single) and machine-dependent quantity is approximately the value of the unit roundoff, or $2^{-(t+1)}$ for binary computer arithmetic involving t binary digits (or bits) in the fractional part of the number. For example, for double-precision computations on a DEC VAX, $t = 52$ and $\epsilon_m \sim 10^{-16}$. As computational errors will enter from sources other than finite arithmetic, a suitable ϵ_g is then a number greater than or equal to 10^{-8}.

The function term is included in the right-hand side of condition [19] to make the convergence test approximately invariant under function scaling. The value 1 is added because the test may be too stringent in the event that the minimum function value is very small. As problem size increases, the Euclidean norm of the gradient increases also. Therefore for large-scale problems it is more appropriate to check the size of an "average" gradient element by dividing the Euclidean norm by $\sqrt{n}$, for example. Alternatively, the max norm,

$$\|\mathbf{g}\|_\infty = \max_i |g_i|, \qquad [20]$$

may be used to replace $\|\mathbf{g}\|_2$ or $\|\mathbf{g}\|_2/\sqrt{n}$ in the left side of condition [19].

To obtain a measure of progress at each iteration—and possibly halt computations if necessary—a natural test involves the convergence of the sequence of iterates $\{\mathbf{x}_k\}$ as well as the corresponding function value $\{f(\mathbf{x}_k)\}$. A suitable combination can be formulated as

$$f(\mathbf{x}_{k-1}) - f(\mathbf{x}_k) < \epsilon_f(1 + |f(\mathbf{x}_k)|) \qquad [21a]$$

$$\|\mathbf{x}_{k-1} - \mathbf{x}_k\| < (\epsilon_f)^{1/2}(1 + \|\mathbf{x}_k\|), \qquad [21b]$$

where $\epsilon_f > 0$ is a small number that specifies the desired accuracy in the function value. The square root of ϵ_f in condition [21b] is suggested by a Taylor expansion analysis.[5]

In addition to conditions [21a,b], another test involving ϵ_f may be imposed on the gradient norm to evaluate the optimality of the converging iterate:

$$\|\mathbf{g}_k\| < (\epsilon_f)^{1/3}(1 + |f(\mathbf{x}_k)|). \qquad [21c]$$

In both [21b] and [21c], a scaled Euclidean or the max norm may be more appropriate for large-scale functions. Test [21c] uses $(\epsilon_f)^{1/3}$ rather than $(\epsilon_f)^{1/2}$, as suggested theoretically, to reduce the severity of the triplet combination.[5] Test [19] is useful in case an iterate falls very close to a solution or when no further progress can be made for f and $\mathbf{x}$.

Each step of the descent method [A1] may then check conditions [19] and [21a–c]. For $\mathbf{x}_0$, only condition [19] is checked. If either the triplet [21a,b,c] or [19] hold, the iteration process may be halted.

Convergence Characterization

Different descent methods, distinguished by their choice of search directions and implementation details, have varying computational costs and convergence properties.

Convergence properties of most minimization algorithms are analyzed through their application to convex quadratic functions. A general multivariate convex quadratic can be written as

$$q(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^{\mathrm{T}}A\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + c \qquad [22]$$

where A is a positive-definite $n \times n$ matrix, $\mathbf{b}$ a constant vector, $\mathbf{x}$ an arbitrary vector, and c a constant. The constant term may be set to zero, as it does not influence the position of the minimum. The gradient of this function is the vector $A\mathbf{x} + \mathbf{b}$, and the Hessian of q is the matrix A.

Our optimality conditions imply that the minimum $\mathbf{x}^*$ of $q(\mathbf{x})$ can be found by setting the gradient equal to zero:

$$A\mathbf{x}^* = -\mathbf{b}. \qquad [23]$$

As A is positive-definite, this solution is unique, and $\mathbf{x}^*$ is the global minimum of q. Because general functions may be approximated by a quadratic convex function in the neighborhood of their local minima, the convergence properties obtained for convex quadratic functions are usually applied locally to general functions. This does not, however, in any case guarantee good behavior in practice on complex, large-scale functions.

The convergence properties of an algorithm are described by two analytic quantities: *convergence order* and *convergence ratio*. A sequence $\{\mathbf{x}_k\}$ is said to converge to $\mathbf{x}^*$ if the following holds: $\lim_{k\to\infty} \|\mathbf{x}_k - \mathbf{x}^*\| = 0$. The sequence is said to converge to $\mathbf{x}^*$ with order p if p is the largest nonnegative number for which a finite limit β exists, where

$$0 \leq \beta \leq \lim_{k\to\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^p}. \qquad [24]$$

When $p = 1$ and $\beta < 1$, the sequence is said to converge *linearly* (e.g., $\mathbf{x}_k = 2^{-k}$ for $n = 1$); when $p = 1$ and $\beta = 0$, the sequence converges *superlinearly* (e.g., $\mathbf{x}_k = k^{-k}$); and when $p = 2$, the convergence is *quadratic* (e.g., $\mathbf{x}_k = 2^{-2^k}$). Thus, quadratic convergence is more rapid than superlinear, which in turn is faster than linear. The constant β is the associated convergence ratio.

NONDERIVATIVE METHODS

Minimization methods that incorporate only function values generally involve some systematic method to search the conformational space. In coordinate descent methods, the search directions are the standard basis vectors. A sweep through these n search vectors produces a sequential modification of one function variable at a time. Through repeated sweeping of the n-dimensional space, a local minimum might ultimately be found. Unfortunately, this strategy is inefficient and not reliable.[3,4]

A well known variant of the basic coordinate descent scheme is Powell's method.[57] Rather than specifying the search vectors a priori, the standard basis directions are modified as the algorithm progresses. The modification ensures that, when the procedure is applied to a convex quadratic function, n mutually conjugate directions are generated after n sweeps. A set of mutually conjugate directions $\{\mathbf{p}_k\}$ with respect to the (positive-definite) Hessian A of such a convex quadratic is defined by $\mathbf{p}_i^T A\mathbf{p}_j = 0$ for all $i \neq j$. This set possesses the important property that a successive search along each of these directions suffices to find the minimum solution.[3,4] Powell's method thus guarantees that in exact arithmetic (i.e., in absence of roundoff error), the minimum of a convex quadratic function will be found after n sweeps.

Nonderivative minimization methods are generally easy to implement and avoid derivative computations, but their realized convergence properties are rather poor. They may work well in special cases when the function is quite random in character or the variables are essentially uncorrelated. In general, the computational cost, dominated by the number of function evaluations, can be excessively high for functions of many variables and can far outweigh the benefit of avoiding derivative calculations.

If obtaining the analytic derivatives is out of the question, viable alternatives remain. The gradient can be approximated by finite differences of function values, such as

$$g_i(\mathbf{x}) \approx \frac{1}{h_i}[f(\mathbf{x} + h_i\mathbf{e}_i) - f(\mathbf{x})], \qquad [25]$$

for suitably chosen intervals $\{h_i\}$.[5,58] Alternatively, automatic differentiation, essentially a new algebraic construct,[59–61] may be used. In any case, these calculated derivatives may then be used in a gradient or quasi-Newton method (described later). Such alternatives will generally provide significant improvement in computational cost and reliability.

Despite these drawbacks of nonderivative methods, their ease of application has made them the choice for several potential energy applications.[27,62]

GRADIENT METHODS

Methods that use analytic-derivative information clearly possess more information about the smooth objective function. Gradient methods can use the slope of a function, for example, as the direction of movement toward extremum points. Second derivative methods can also incorporate curvature information from the Hessian to find the regions where the function is convex.

Two common gradient methods are steepest descent (SD)[3–5,63] and conjugate gradient (CG).[14,64–67]

Steepest Descent

Steepest descent is one of the oldest and simplest methods. It is actually more important as a theoretical, rather than practical, reference by which to test other methods; however, "steepest descent" steps are often incorporated into other methods (e.g., conjugate gradient, Newton) when roundoff destroys some desirable theoretical properties, progress is slow, or regions of indefinite curvature are encountered.

At each iteration of SD, the search direction is taken as $-\mathbf{g}_k$, the negative gradient of the objective function at $\mathbf{x}_k$. Recall that a descent direction $\mathbf{p}_k$ satisfies $\mathbf{g}_k^T\mathbf{p}_k < 0$. The simplest way to guarantee the negativity of this inner product is to choose $\mathbf{p}_k = -\mathbf{g}_k$. This choice also minimizes the inner product $-\mathbf{g}_k^T\mathbf{p}$ for unit-length vectors and thus gives rise to the name *steepest descent.*

Steepest descent is simple to implement and requires modest storage, $O(n)$; however, progress toward a minimum may be very slow, especially near a solution. The convergence rate of SD when applied to a convex quadratic function, as in Eq. [22], is only linear. The associated convergence ratio is no greater than $[(\kappa - 1)/(\kappa + 1)]^2$ where κ, the condition number, is the ratio of largest to smallest eigenvalues of A:

$$\kappa = \lambda_{max}(A)/\lambda_{min}(A). \qquad [26]$$

As the convergence ratio measures the reduction of the error at every step ($\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \beta\|\mathbf{x}_k - \mathbf{x}^*\|$ for a linear rate), the relevant SD value can be arbitrarily close to 1 when κ is large (Figure 12). In other words, because the n lengths of the elliptical axes belonging to the contours of the function are proportional to the eigenvalue reciprocals, the convergence rate of SD is slowed as the contours of the objective function become more eccentric. Thus, the SD search vectors may in some cases exhibit very inefficient paths toward a solution (see final section for a numerical example).

Conjugate Gradient

The CG method was originally designed to minimize convex quadratic functions. Through several variations, it has also been extended to the general case.[66–72]

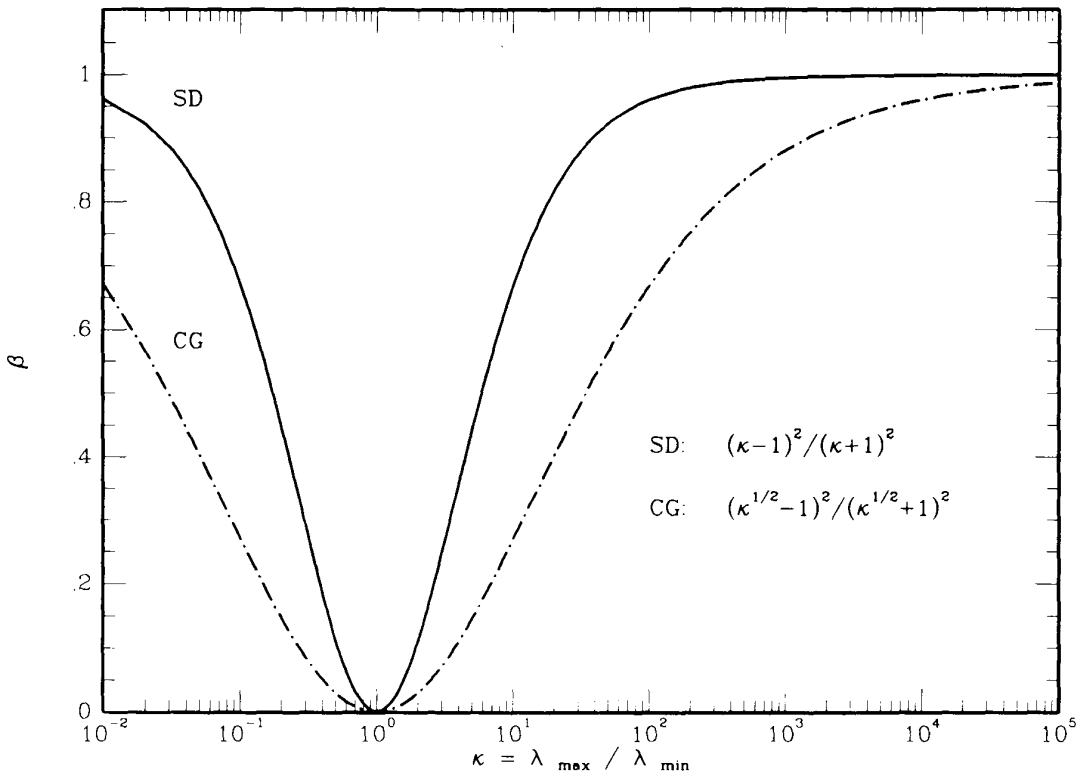

Figure 12 Steepest descent and conjugate gradient quantities that affect the convergence rate for quadratic functions (see text for the distinct context of these functions).

The first iteration in a CG method is the same as in SD, with a step along the current negative gradient vector. Successive directions are constructed differently so that they form a set of mutually conjugate vectors with respect to the (positive-definite) Hessian A of a general convex quadratic function.

Whereas the rate of convergence for SD depends on the ratio of the extremal eigenvalues of A, the convergence properties of CG depend on the entire matrix spectrum. Faster convergence is expected when the eigenvalues are clustered. In exact arithmetic, convergence is obtained in at most n steps. In particular, if A has m distinct eigenvalues, convergence to a solution requires m iterations.

One way to see the convergence dependence of the entire spectrum of A is to consider the following bound on convergence where the size of $\mathbf{x}_k - \mathbf{x}^*$ is measured with respect to the A-norm, defined as

$$\|\mathbf{x}\|_A = (\mathbf{x}^T A \mathbf{x})^{1/2}. \qquad [27]$$

The bound is given by[3,14]

$$\|\mathbf{x}_k - \mathbf{x}^*\|_A^2 \leq 4\|\mathbf{x}_k - \mathbf{x}_0\|_A^2 \left[\frac{\sqrt{k} - 1}{\sqrt{k} + 1} \right]^{2k}. \qquad [28]$$

Clearly rapid convergence is expected when $\kappa \approx 1$, as for SD (see Figure 12). Further estimates of convergence bounds are more difficult but can be derived when certain properties about the eigenvalue distribution are known (e.g., m large eigenvalues and $n - m$ small eigenvalues clustered in a region $[a,b]$]).[3,14] The general notion of convergence for CG methods has been a large area of research[3,14,70–77] for both the convex quadratic and the nonlinear extension cases.

When one refers to the CG method, one often means the *linear* conjugate gradient, that is, the implementation for the convex quadratic form. In this case, minimizing $\frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x}$ is equivalent to solving the linear system $A\mathbf{x} = -\mathbf{b}$. Consequently, the conjugate directions $\mathbf{p}_k$, as well as the lengths λ_k, can be computed in closed form.

In describing the steps of a CG method to solve $A\mathbf{x} = -\mathbf{b}$, the residual vector $A\mathbf{x} + \mathbf{b}$ is useful. We define $\mathbf{r} = -(A\mathbf{x} + \mathbf{b})$ and use the vectors $\{\mathbf{d}_k\}$ below to denote the CG search vectors (for reasons that will become clear in the Newton Methods section). The solution $\mathbf{x}^*$ can then be obtained by the following procedure, once a starting point $\mathbf{x}_0$ is specified.[78,79]

Algorithm [A2]: CG Method to Solve $A\mathbf{x} = -\mathbf{b}$

1. Set $\mathbf{r}_0 = -(A\mathbf{x}_0 + \mathbf{b})$, $\mathbf{d}_0 = \mathbf{r}_0$.
2. For $k = 0,1,2, \ldots,$ until $\mathbf{r}$ is sufficiently small, compute

$$\begin{aligned} \lambda_k &= \mathbf{r}_k^T\mathbf{r}_k / \mathbf{d}_k^T A\mathbf{d}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \lambda_k\mathbf{d}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \lambda_k A\mathbf{d}_k \\ \beta_k &= \mathbf{r}_{k+1}^T\mathbf{r}_{k+1} / \mathbf{r}_k^T\mathbf{r}_k \\ \mathbf{d}_{k+1} &= \mathbf{r}_{k+1} + \beta_k\mathbf{d}_k. \end{aligned} \qquad [A2]$$

Note here that only a few vectors are stored; the product $A\mathbf{d}$ but not knowledge (or storage) of A per se is required; and the cost involves only several scalar and vector operations. The value of the step length λ_k can be derived in the closed form above by minimizing $q(\mathbf{x}_k + \lambda\mathbf{d}_k)$ as a function of λ (producing $\lambda_k = \mathbf{r}_k^T\mathbf{d}_k / \mathbf{d}_k^T A\mathbf{d}_k$) and then using the conjugacy relation $\mathbf{r}_k^T\mathbf{d}_j = 0$ for all $j < k$.[3]

Preconditioning

Performance of the CG method is generally very sensitive to roundoff in the computations that may destroy the mutual conjugacy property. The method was actually neglected for many years until it was realized that a preconditioning technique can be used to accelerate convergence significantly.[14,80,81]

Preconditioning involves modification of the target linear system $A\mathbf{x} = -\mathbf{b}$ through application of a positive-definite preconditioner M that is closely related to A. The modified system can be written as

$$M^{-1}A\mathbf{x} = -M^{-1}\mathbf{b}. \qquad [29]$$

The new coefficient matrix is symmetric as $M^{-1}A$ can be written as $M^{-1/2}AM^{-1/2}$. Preconditioning aims to produce a more clustered eigenvalue structure for $M^{-1}A$ and/or lower condition number than for A to improve the relevant convergence ratio; however, preconditioning also adds to the computational effort by requiring that a linear system involving M (namely, $M\mathbf{z} = \mathbf{r}$) be solved at every step. Thus, it is essential for efficiency of the method that M be factored very rapidly in relation to the original A. This can be achieved, for example, if M is a sparse component of the dense A. Whereas the solution of an $n \times n$ dense linear system requires order of n^3 operations, the work for sparse systems can be as low as order n.[13,14]

The Hessians of potential energy functions, for example, separate naturally into local terms (among atom pairs involved in bonds, bond angles, and dihedral angles) and nonlocal terms (among nonbonded atom pairs). The number of local terms increases linearly with n, whereas the nonlocal terms increase as n^2. Thus, a preconditioner from the local terms is a good choice that has performed well in practice.[23,82]

The recurrence relations for the preconditioned conjugate gradient (PCG) method can be derived from Algorithm [A2] after substituting $\mathbf{x}_k = M^{-1/2}\tilde{\mathbf{x}}_k$ and $\mathbf{r}_k + M^{1/2}\tilde{\mathbf{r}}_k$. New search vectors $\tilde{\mathbf{d}}_k = M^{-1/2}\mathbf{d}_k$ can be used to derive the iteration process, and then the tilde modifiers dropped. The PCG method becomes the following iterative process.

Algorithm [A3]: PCG Method to Solve $A\mathbf{x} = -\mathbf{b}$

1. Set $\mathbf{r}_0 = -(A\mathbf{x}_0 + \mathbf{b})$, $\mathbf{d}_0 = M^{-1}\mathbf{r}_0$.
2. For $k = 0,1,2, \ldots$, until $\mathbf{r}$ is sufficiently small, compute

$$\begin{aligned} \lambda_k &= \mathbf{r}_k^{\mathrm{T}}(M^{-1}\mathbf{r}_k)/\mathbf{d}_k^{\mathrm{T}}A\mathbf{d}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \lambda_k\mathbf{d}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \lambda_k A\mathbf{d}_k \\ \beta_k &= \mathbf{r}_{k+1}^{\mathrm{T}}(M^{-1}\mathbf{r}_{k+1})/\mathbf{r}_k^{\mathrm{T}}(M^{-1}\mathbf{r}_k) \\ \mathbf{d}_{k+1} &= (M^{-1}\mathbf{r}_{k+1}) + \beta_k\mathbf{d}_k. \end{aligned} \qquad [A3]$$

Note above that the system $M\mathbf{z}_k = \mathbf{r}_k$ must be solved repeatedly for $\mathbf{z}_k$ and that the matrix/vector products $A\mathbf{d}_k$ are required as before.

Nonlinear Conjugate Gradient

Extensions of the linear CG method to nonquadratic problems have been developed and extensively researched.[66–78] In the several existing variants, the basic idea is to avoid matrix operations altogether and simply express the search directions recursively as

$$\mathbf{d}_k = -\mathbf{g}_k + \beta_k \mathbf{d}_{k-1}, \qquad [30]$$

for $k = 1,2, \ldots,$ with $\mathbf{d}_0 = -\mathbf{g}_0$. The new iterates for the minimum point can then be set to

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k, \qquad [31]$$

where λ_k is the step length. In comparing this iterative procedure with the linear CG of Algorithm [A2], we note that $\mathbf{r}_k = -\mathbf{g}_k$ for a quadratic function. The aforementioned parameter β_k is chosen so that if f is a convex quadratic and λ_k is the exact one-dimensional minimizer of f along $\mathbf{d}_k$, the nonlinear CG reduces to the linear CG method and terminates in at most n steps (in exact arithmetic).

Three of the best known settings for β_k are titled the Fletcher–Reeves (FR), Polak–Ribière (PR), and Hestenes–Stiefel (HS) formulas.[66–71,77,78] They are given by the formulas

$$\beta_k^{\mathrm{FR}} = \mathbf{g}_k^{\mathrm{T}}\mathbf{g}_k / \mathbf{g}_{k-1}^{\mathrm{T}}\mathbf{g}_{k-1} \qquad [32a]$$

$$\beta_k^{\mathrm{PR}} = \mathbf{g}_k^{\mathrm{T}}(\mathbf{g}_k - \mathbf{g}_{k-1}) / \mathbf{g}_{k-1}^{\mathrm{T}}\mathbf{g}_{k-1} \qquad [32b]$$

$$\beta_k^{\mathrm{HS}} = \mathbf{g}_k^{\mathrm{T}}(\mathbf{g}_k - \mathbf{g}_{k-1}) / \mathbf{d}_{k-1}^{\mathrm{T}}(\mathbf{g}_k - \mathbf{g}_{k-1}). \qquad [32c]$$

The last two formulas are generally preferred in practice, though the first has better theoretical global convergence properties. In fact, very recent research has focused on combining these practical and theoretical properties for construction of more efficient schemes.[77,78] The simple modification of

$$\beta_k = \max\{\beta_k^{\mathrm{PR}}, 0\}, \qquad [33]$$

for example, can be used to prove global convergence of this nonlinear CG method, even with inexact line searches.[77] A more general condition on β_k, including relaxation of its nonnegativity, has also been derived.[78]

The quality of line search in these nonlinear CG algorithms is crucial. (Typically, line searches are used rather than the trust region methods.) Adjustments must be made not only to preserve the mutual conjugacy of the search directions—a property critical for finite termination of the method—but also to ensure that each generated direction is one of descent. A technique known as

restarting is typically used to preserve a linear convergence rate by resetting $\mathbf{d}_k$ to the steepest descent direction, $-\mathbf{g}_k$, after a given number of linear searches (e.g., n).[68] A restart vector may also be formulated as a linear combination of the steepest descent direction and some other vector. This generally results in a method that requires $2n$ to $5n$ iterations to achieve reasonable accuracy,[70,83] though in some cases progress may be very slow. As for the convex quadratic version, the number of iterations is significantly reduced if the Hessian has a clustered eigenvalue structure. Thus, preconditioning may also be used as in the linear case. Often, however, if additional derivative information and memory are available, Newton methods are preferred.

In sum, the greatest virtues of CG methods are their modest storage and computational requirements (both order n) and their better convergence than the SD method. These properties have made them popular linear solvers and minimization choices in many applications[18–20,84–88] and perhaps the only candidates for very large problems. The linear CG is often applied to systems arising from discretizations of partial differential equations,[81,89,90] where the matrices are frequently positive-definite, sparse, and structured.

NEWTON METHODS

Overview

Newton methods are the prototype of second derivative algorithms. (Raphson's name is often omitted when referring to this general class of methods, though both Newton and Raphson deserve credit for the quadratically-convergent method for finding a root of an equation, and its variants.) Several classes of methods exist, including *discrete Newton, quasi-Newton* (QN) (also termed *variable metric* in the older literature), and *truncated Newton* (TN). Historically, the $O(n^2)$ memory requirements and $O(n^3)$ computation associated with solving a linear system directly have restricted Newton methods only (1) to small problems, (2) to problems with special sparsity patterns, or (3) near a solution, after a gradient method has been applied. Fortunately, advances in computing technology and software are making the Newton approach feasible for a wide range of problems. Indeed, effective strategies have been tailored to available storage and computation, exhibiting good performance in theory and practice. This trend undoubtedly will intensify. Very good detailed treatments of Newton methods can be found in the literature,[3–6,52,54] and only general concepts are outlined here.

Two specific classes are emerging as the most powerful techniques for large-scale applications: limited-memory quasi-Newton (LMQN) and truncated Newton methods. LMQN methods attempt to combine the modest storage and computational requirements of CG methods with the superlinear convergence properties of standard (i.e., full memory) QN methods. Similarly, TN

algorithms attempt to retain the rapid quadratic convergence rate of classic Newton methods while making computational requirements feasible for large-scale functions. With advances in automatic differentiation, the appeal of these methods will undoubtedly increase even further, as the reduction in the cost of evaluating Hessian/vector products and the preconditioner makes them very efficient.[61]

All Newton methods are based on approximating the objective function locally by a quadratic model and then minimizing that function approximately. The quadratic model of the objective function f at $\mathbf{x}_k$ along $\mathbf{p}$ is given by the expansion

$$f(\mathbf{x}_k + \mathbf{p}) \approx f(\mathbf{x}_k) + \mathbf{g}_k{}^{\mathrm{T}}\mathbf{p} + \tfrac{1}{2}\mathbf{p}^{\mathrm{T}}H_k\mathbf{p}. \qquad [34]$$

The minimum of the right-hand side is achieved when $\mathbf{p}_k$ is the minimum of the quadratic function:

$$q_k(\mathbf{p}) = \mathbf{g}_k{}^{\mathrm{T}}\mathbf{p} + \tfrac{1}{2}\mathbf{p}^{\mathrm{T}}H_k\mathbf{p}. \qquad [35]$$

Alternatively, such a Newton direction $\mathbf{p}_k$ satisfies the linear system of n simultaneous equations, known as the Newton equation:

$$H_k\mathbf{p} = -\mathbf{g}_k. \qquad [36]$$

In the "classic" Newton method, the Newton direction is used to update each previous iterate by the formula $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$, until convergence. The reader may recognize the one-dimensional version of Newton's method for solving a nonlinear equation $f(x) = 0$: $x_{k+1} = x_k - f(x_k)/f'(x_k)$. The analogous iteration process for minimizing $f(x)$ is $x_{k+1} = x_k - f'(x_k)/f''(x_k)$. Note that the one-dimensional search vector, $-f'(x_k)/f''(x_k)$, is replaced by the Newton direction $-H_k{}^{-1}\mathbf{g}_k$ in the multivariate case. This direction is defined for nonsingular H_k. When $\mathbf{x}_0$ is sufficiently close to a solution $\mathbf{x}^*$, quadratic convergence can be proven for Newton's method.[3-6] That is, a constant β exists such that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \beta\|\mathbf{x}_k - \mathbf{x}^*\|^2. \qquad [37]$$

In practice, this means that the number of digits of accuracy in the solution is approximately doubled at every step. This can be seen from the program output for a simple one-dimensional application of Newton's method to finding the root of a (equivalently, solving $f(x) = x^2 - a = 0$ or minimizing $f(x) = x^3/3 - ax$) (Figure 13).

Unfortunately, there is a disparity between this theoretical convergence result and the practical behavior of the method in general. Thus, modifications of the classic Newton iteration are essential for guaranteeing global convergence, with quadratic convergence rate near the solution.

```
Newton's Iteration for Solving  f(x) = x*x - a = 0

      x(k+1) = ( x(k)*x(k) + a ) / (2. * x(k) )

Single Precision                          Double Precision

   x           |x-s|/s                     X                        |X-S|/S
6.000000    1.0000000E+00    6.000000000000000    1.0000000000000000E+00
3.750000    2.5000000E-01    3.750000000000000    2.5000000000000000E-01
3.075000    2.5000015E-02    3.075000000000000    2.5000000000000004E-02
3.000915    3.0485788E-04    3.000914634146342    3.0487804878050658E-04
3.000000    7.9472862E-08    3.000000139383442    4.6461147336825566E-08
3.000000    0.0000000E+00    3.000000000000003    1.0732155904709847E-15
3.000000    0.0000000E+00    3.000000000000000    1.8503717077085942E-17
3.000000    0.0000000E+00    3.000000000000000    0.0000000000000000E+00

(x = iterate, s = solution)          (X = iterate, S = solution)
```

Figure 13 One-dimensional application of Newton's method for computing the square root of a number (computer output shown). Note in the double-precision version the roundoff in the last steps.

First, when H_k is not positive-definite, the search direction may not exist or may not be a descent direction. Strategies to produce a related positive-definite matrix $\overline{H}_k$, or alternative search directions, become necessary. Second, far away from $\mathbf{x}^*$, the quadratic approximation of expression [34] may be poor, and the Newton direction must be adjusted. A line search, for example, can dampen (scale) the Newton direction when it exists, ensuring sufficient decrease and guaranteeing uniform progress toward a solution. These adjustments lead to the following modified Newton framework (using a line search).

Algorithm [A4]: Modified Newton

*For $k = 0,1,2, \ldots$, until convergence, given $\mathbf{x}_0$,

1. Test $\mathbf{x}_k$ for convergence.
2. Compute a descent direction $\mathbf{p}_k$ so that

$$\|H_k\mathbf{p}_k + \mathbf{g}_k\| \leq \eta_k\|\mathbf{g}_k\|, \qquad \text{[A4]}$$

   where η_k controls the accuracy of the solution and some symmetric matrix $\overline{H}_k$ may approximate H_k.
3. Compute a step length λ so that for $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda\mathbf{p}_k$,

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \alpha\lambda\mathbf{g}_k{}^{\mathrm{T}}\mathbf{p},$$
$$|\mathbf{g}_{k+1}{}^{\mathrm{T}}\mathbf{p}_k| \leq \beta|\mathbf{g}_k{}^{\mathrm{T}}\mathbf{p}_k|,$$

   with $0 < \alpha < \beta < 1$.
4. Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda\mathbf{p}_k$.

Newton variants are constructed by combining various strategies for the individual components above. These involve procedures for formulating H_k or $\overline{H}_k$, dealing with structures of indefinite Hessians, and solving for the modified Newton search direction. For example, when H_k is approximated by finite differences, the discrete Newton subclass emerges.[5,91–94] When H_k, or its inverse, is approximated by some modification of the previously constructed matrix (see later), QN methods are formed.[95–110] When η_k is nonzero, TN methods result,[111–123] because the solution of the Newton system is truncated before completion.

Discrete Newton

Standard discrete Newton methods require n gradient evaluations of $O(n^2)$ operations to compute and symmetrize every Hessian H_k. Each column i of H_k can be approximated by the vector

$$\tilde{\mathbf{h}}_i = \frac{1}{h_i}[\mathbf{g}(\mathbf{x}_k + h_i\mathbf{e}_i) - \mathbf{g}_k], \qquad [38]$$

where h_i is a suitably chosen number.[5,58] This value must balance the roundoff error, proportional to $(1/h_i)$, by formulation, with the truncation error, proportional to h_i. A simple estimate for a well-scaled problem to balance the two errors is $O(\sqrt{\epsilon_m})$. Consequently, a suitable choice for h_i may be $\sqrt{\epsilon_m}$ times the ith component of $\mathbf{x}_k$. For simplicity, all difference parameters $\{h_i\}$ may be taken as a fixed value h. As the difference formulation [38] will not necessarily produce a symmetric matrix $\tilde{H}_k$, whose columns are the vectors $\{\tilde{\mathbf{h}}_k\}$, $i = 1, \ldots, n$, a symmetrizing procedure can be used to construct the matrix

$$\overline{H}_k = \tfrac{1}{2}(\tilde{H}_k + \tilde{H}_k^T). \qquad [39]$$

With exact arithmetic, discrete Newton methods converge quadratically if each h_i goes to zero as $\|\mathbf{g}\|$ does;[6] however, the roundoff error limits the smallest feasible size of difference interval practice and, hence, the accuracy (a combination of roundoff and truncation errors) that can be obtained. As the gradient becomes very small, considerable loss of accuracy may also result from cancellation errors in the numerator. This type of error refers to a large relative error $(|s - s^*|/|s^*|)$ from the subtraction of two quantities (s, s^*) of similar magnitude. Consequently, discrete Newton methods are inappropriate for large-scale problems unless the Hessian has a known sparsity structure. Through special choices of difference perturbations, a known sparsity structure can be exploited in the formulation of $\overline{H}_k$ to reduce computational effort and increase the accuracy significantly.[5,91–93]

Quasi-Newton

Quasi-Newton methods form an interesting class of algorithms that are theoretically closely related to nonlinear CG methods.[6,95,96] They are found to perform very well in practice.[6,100–102,109,110] QN research has been developing

very rapidly since its beginning in the mid-1960s, and recent analyses and discoveries[44,104–107] increase their appeal even further.

Quasi-Newton methods can be viewed as extensions of nonlinear CG methods, in which additional curvature information is used to accelerate convergence. Thus, the required analytic Hessian information, memory, and computational requirements are kept as low as possible, and the main strength of Newton methods—employing curvature information to detect and move away from saddle points efficiently—is retained.

The basic idea in these methods is building up curvature information progressively. At each step of the algorithm, the current approximation to the Hessian (or inverse Hessian, as we shall see) is updated by using new gradient information. The updated matrix itself is not necessarily stored explicitly, as the updating procedure may be defined compactly in terms of a small set of stored vectors. This economizes memory requirements considerably and increases the appeal to large-scale applications.

Consider the Taylor expansion for the one-dimensional function $f(x)$ at step k of the method:

$$f(x_{k+1}) = f(x_k) + f'(x_k)(x_{k+1} - x_k) + \cdots . \qquad [40]$$

From this expansion, we can approximate the derivative as

$$f'(x_k) \approx \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k} . \qquad [41]$$

This is known as the *secant approximation*.[6] In the extension of this idea to higher dimensions and to the second derivatives, we consider the expansion of the gradient

$$\mathbf{g}_{k+1} = \mathbf{g}_k + H_k(\mathbf{x}_{k+1} - \mathbf{x}_k) + \cdots . \qquad [42]$$

Letting

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \qquad [43]$$

represent the "step" taken from $\mathbf{x}_k$ and

$$\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k \qquad [44]$$

represent the difference in the corresponding gradients, we can obtain the following relation if H_k were a constant (equivalently, $f(x)$ a quadratic) equal to H:

$$H\mathbf{s}_k = \mathbf{y}_k. \qquad [45]$$

Thus, in this case we see that each gradient difference $\mathbf{y}_k$ provides information about one *column* of H. It is then reasonable to attempt to construct a family of successive approximation matrices $\{B_k\}$ so that, if H were a constant, the procedure would be consistent with Eq. [45]. Specifically, we require that the new update B_{k+1}, satisfy the *quasi-Newton condition*

$$B_{k+1}\mathbf{s}_k = \mathbf{y}_k. \qquad [46]$$

This condition forms the basis of QN methods. Because it does not uniquely specify how B_{k+1} should be formulated, additional conditions must be imposed, and different formulations generate different QN algorithms.

It is reasonable to assume that B_{k+1} differs from B_k by a low-rank "updating" matrix (i.e., a matrix of rank much less than n). Such an update U_k will depend on $\mathbf{s}_k, \mathbf{y}_k$, and possibly B_k:

$$B_{k+1} = B_k + U_k(\mathbf{s}_k, \mathbf{y}_k, B_k). \qquad [47]$$

For example, a matrix of rank 1 can be formulated as the outer product of two vectors, such as $\mathbf{u}\mathbf{v}^T$. (The rank of this matrix is 1 because all rows are scalar multiples of one other). Applying condition [46] to this update form: $B_{k+1} = B_k + \mathbf{u}\mathbf{v}^T$, we obtain the condition that $\mathbf{u}$ is a vector in the direction of $(\mathbf{y}_k - B_k\mathbf{s}_k)$. If $\mathbf{y}_k = B_k\mathbf{s}_k, B_k$ already satisfies the QN condition [46]. Otherwise, we can write the general rank 1 update formula as:

$$B_{k+1} = B_k + \frac{1}{\mathbf{v}^T\mathbf{s}_k}(\mathbf{y}_k - B_k\mathbf{s}_k)\mathbf{v}^T, \qquad [48]$$

for $\mathbf{v}^T\mathbf{s}_k \neq 0$. *Broyden*'s QN method, for example, uses $\mathbf{v} = \mathbf{s}_k$. Broyden's update does not even guarantee symmetry but is useful for solving nonlinear equations and for deriving a more effective, rank 2 update. To restrict the general rank 1 update form of [48] further, we can impose the condition of symmetry. Symmetry will be "inherited" from B_k to B_{k+1} if $\mathbf{u}\mathbf{v}^T = \alpha\mathbf{u}\mathbf{u}^T$ for some scalar α. Letting that α be $1/\mathbf{v}^T\mathbf{s}_k$, we obtain the general symmetric rank 1 update (SR1) as follows:

$$B_{k+1} = B_k + \frac{1}{(\mathbf{y}_k - B_k\mathbf{s}_k)^T\mathbf{s}_k}(\mathbf{y}_k - B_k\mathbf{s}_k)(\mathbf{y}_k - B_k\mathbf{s}_k)^T. \qquad [49]$$

SR1 will be positive-definite only if $(\mathbf{y}_k - B_k\mathbf{s}_k)^T\mathbf{s}_k > 0$. Thus, rank 2 updates (e.g., $\mathbf{u}_1\mathbf{v}_1^T + \mathbf{u}_2\mathbf{v}_2^T$) were thought until very recently to be more suitable for optimization.

In general, the specific choice of U_k depends on a desired rank (often 2) and imposed constraints of symmetry and positive-definiteness. Different choices for U_k define different QN algorithms, with varying theoretical and practical properties.

Once a new approximation matrix is defined, the search direction in the basic Algorithm [A4] is formulated accordingly. The vector $\mathbf{p}_k$ is computed from

$$B_k\mathbf{p}_k = -\mathbf{g}_k \qquad [50a]$$

or by

$$\mathbf{p}_k = -\tilde{B}_k\mathbf{g}_k, \qquad [50b]$$

depending on whether the QN sequence of matrices attempts to approximate the Hessian ($\{B_k\}$) or inverse Hessian ($\{\tilde{B}_k\}$); however, it is important to keep in mind that the sequence of matrices itself may not converge to the Hessian or inverse Hessian at a solution. Convergence of the iterates to a solution, on the other hand, can be derived, with a superlinear convergence rate.[61,95]

In general, the solution of system [50a] requires $O(n^3)$ operations, and the multiplication in [50b] demands $O(n^2)$ work; however, the updating context of these problems can be used to formulate recursive lower-cost solutions in terms of previously computed quantities. Such formulas may update the factors of the previously decomposed matrix, or use an iteration process based on the Sherman–Morrison–Woodbury formula[6] for $(B + \mathbf{uv}^T)^{-1}$ in combination with a recursive process suggested by Matthies and Strang.[124] The second procedure [50b] is generally preferred in large-scale applications, as it is computationally more economical.

In practice, different QN schemes combine various procedures for the following components: (1) formulation and calculation of the update, (2) calculation of $\mathbf{p}_k$, choice of initial approximation B_0, and (4) subsequent scalings of B_k. These implementation details have been crucial to the recent success of LMQN methods for large-scale problems.

One of the most successful and widely used updating formulas is known as BFGS for its four developers: Broyden, Fletcher, Goldfarb, and Shanno.[6,95] It is a rank 2 update with inherent positive-definiteness (i.e., B_k positive-definite $\Rightarrow$ B_{k+1} positive-definite) that was derived by "symmetrizing" the Broyden rank 1 update.[5,6,95] A sequence of matrices $\{\tilde{B}_k\}$ is generated from a positive-definite $\tilde{B}_0$ (which may be taken as the identity) by the BFGS formula

$$\tilde{B}_{k+1} = \tilde{B}_k + U_k, \qquad [51a]$$

where

$$U(\mathbf{s},\mathbf{y},\tilde{B}) = \frac{\mathbf{s}\mathbf{s}^T}{\mathbf{y}^T\mathbf{s}}\left[\frac{\mathbf{y}^T\tilde{B}\mathbf{y}}{\mathbf{y}^T\mathbf{s}} + 1\right] - \frac{1}{\mathbf{y}^T\mathbf{s}}[\mathbf{s}\mathbf{y}^T\tilde{B} + \tilde{B}\mathbf{y}\mathbf{s}^T]. \qquad [51b]$$

As an alternative to this *sum* form, a *product* version has also been used[100]:

$$\tilde{B}_{k+1} = V_k{}^{\mathrm{T}}\tilde{B}_k V_k + W_k, \qquad [52a]$$

with

$$V(\mathbf{s},\mathbf{y}) = I - \frac{1}{\mathbf{y}^{\mathrm{T}}\mathbf{s}}[\mathbf{s}\mathbf{y}^{\mathrm{T}}], \qquad [52b]$$

$$W(\mathbf{s},\mathbf{y}) = \frac{1}{\mathbf{y}^{\mathrm{T}}\mathbf{s}}[\mathbf{s}\mathbf{s}^{\mathrm{T}}]. \qquad [52c]$$

In limited memory variants, the matrices $\{\tilde{B}_k\}$ are not formed explicitly.[100–103] Rather, pairs of vectors $\{\mathbf{s}_k,\mathbf{y}_k\}$ that define $\{\tilde{B}_k\}$ implicitly are stored separately, and efficient formulas for computing the products $\tilde{B}_k\mathbf{g}_k$ are used.[100] In the LM-BFGS method of Liu and Nocedal,[100,101] for example, an integer u is prescribed to specify the number of corrections (updates) that can be stored. This integer typically ranges between 3 and 7. After computing u vector pairs according to the aforementioned formulas (i.e., exhausting $2nu$ storage locations), a cyclical procedure is used to retain latest and discard oldest information. Specifically, for steps $k > u$, a positive-definite matrix $\tilde{B}_k{}^0$ is set by a suitable scaling of the identity matrix, such as

$$\tilde{B}_k{}^0 = \frac{\mathbf{y}_k{}^{\mathrm{T}}\mathbf{s}_k}{\mathbf{y}_k{}^{\mathrm{T}}\mathbf{y}_k} I. \qquad [53]$$

Subsequently, u BFGS corrections are applied to $\tilde{B}_k{}^0$ using the previous u $\{\mathbf{s}_k,\mathbf{y}_k\}$ pairs. The resulting matrix is used to define the search direction [50b] compactly, using a recursion formula that requires only $O(nu)$ operations.[100] A line search follows.

Surprisingly, experience to date has suggested that LM-BFGS does not increase significantly in performance for a wide range of problems when $u > 7$. It also performs well in comparison to full-memory QN methods and generally much better than nonlinear CG methods.[100–102,110] Scaling is critical, and research is continuing on optimal choices.[101,102] Preconditioning is an option as well, as in CG and TN (discussed later) methods. Preconditioning can be used to replace the scaling strategy of [53] (used to compute the initial search vector via [50b] with $\tilde{B}_k{}^0 = [M_k]^{-1}$, where M is a positive-definite preconditioner. Preconditioning is expected to accelerate performance of LM-BFGS methods significantly, but this area has hardly been tested in practice. Overall, the combination of rapid superlinear convergence and ease of application makes LM-BFGS methods excellent choices for large-scale nonlinear minimization.

Truncated Newton

Truncated Newton methods were introduced in the early 1980s[111–114] and have been gaining popularity ever since.[82,109,110,115–123] Their basis is the following simple observation. An exact solution of the Newton equation at every step is unnecessary and computationally wasteful in the framework of a basic descent method. That is, an exact Newton search direction is unwarranted when the objective function is not well approximated by a convex quadratic and/or the initial point is distant from a solution. Any descent direction will suffice in that case. As a solution to the minimization problem is approached, the quadratic approximation may become more accurate, and more effort in solution of the Newton equation may be warranted.

Thus, the approximate Newton search direction in TN methods is obtained by allowing a nonzero residual norm $r_k = \|\mathbf{r}_k\| = \|H_k\mathbf{p}_k + \mathbf{g}_k\|$ at each step. The size of this residual is monitored systematically according to the progress made. This formulation leads to a doubly nested iteration structure: for every outer Newton iteration k (associated with $\mathbf{x}_k$) there corresponds an inner loop for $\mathbf{p}_k$ $\{\mathbf{p}_k{}^0, \mathbf{p}_k{}^1, \ldots\}$.

As the linear PCG method is one of the most powerful and computationally economical iterative methods for solving large positive-definite linear systems, it is the most suitable choice here for the inner loop.

The performance and large-scale feasibility of a TN algorithm depend on the precise formulation of a truncation criterion and implementation of the inner PCG loop. The PCG process can be terminated when either one of the following conditions is satisfied: (1) the residual r_k is sufficiently small, (2) the quadratic model $q_k(\mathbf{p}_k{}^i)$ as defined in Eq. [35] is sufficiently reduced, or (3) a direction of negative curvature $\mathbf{d}$ is encountered (i.e., $\mathbf{d}^T H_k \mathbf{d}_k < 0$). A negative curvature direction may occur because H_k may not be positive-definite. Recall that the positive-definiteness of the coefficient matrix was crucial for the derivation of the scalar λ_k in closed form (see Algorithm [A2]).

An effective *residual test* (RT)[113] checks at each inner iteration whether the *relative residual* $\tilde{r}_k$ is sufficiently small:

$$\tilde{r}_k = \frac{\|\mathbf{r}_k\|}{\|\mathbf{g}_k\|} \leq \eta_k. \qquad [54]$$

The forcing sequence η_k can be set, for example, to

$$\eta_k = \min\{c_r/k, \|\mathbf{g}_k\|\}, \qquad [55]$$

where $0 < c_r \leq 1$. This procedure produces smaller residuals as the number of outer iterations increases and/or as a stationary point of f is more closely approached ($\|\mathbf{g}_k\| \to 0$). Asymptotic quadratic convergence for the method can be proven since

$$\tilde{r}_k \leq \eta_k < 1 \qquad [56]$$

and $\lim_{k\to\infty} \sup(\tilde{r}_k/\|\mathbf{g}_k\|) < \infty$.[113] Weaker, superlinear convergence can be obtained if some sequence $\{\eta_k\}$ only satisfies Eq. [56] and converges to zero.[113,119] With a good choice of preconditioner, RT is generally satisfactory in practice.

An alternative quadratic truncation test (QT) has also been suggested with associated asymptotic superlinear convergence.[116] This criterion monitors, instead of the relative residual, the sufficient decrease of the quadratic model, $q_k(\mathbf{p})$. Specifically, it checks whether $q_k(\mathbf{p})$ has decreased sufficiently from one inner iteration to the next, in relation to the progress realized per inner iteration:

$$\frac{q_k(\mathbf{p}_k{}^i) - q_k(\mathbf{p}_k{}^{i-1})}{q_k(\mathbf{p}_k{}^i)} \leq \frac{c_q}{i}. \qquad [57]$$

The parameter c_q is analogous to c_r in Eq. [55] and is required to be in the same range: $0 < c_q \leq 1$. As $q(\mathbf{p}) = \frac{1}{2}\mathbf{p}^{\mathrm{T}}(\mathbf{r} + \mathbf{g})$, implementation of this condition does not require additional Hessian/vector products, $H\mathbf{p}$.

These two truncation tests provide a different outer-versus-inner iteration trade-off. Our experience suggests that RT typically leads to fewer Newton iterations and more inner iterations than QT, for approximately the same range of c_r and c_q. For good starting points, RT may be more computationally efficient (in terms of CPU time and solution accuracy); for poor starting points, QT may work better. Precise values for c_r and c_q are problem dependent and require some experimentation for best performance. Our investigations suggest values in the range 10^{-4} to 1 for c_r and 0.1 to 0.5 for c_q.

In the PCG process of the inner loop, Hessian/vector multiplications ($H\mathbf{d}$) and linear solutions of the system $M\mathbf{z} = \mathbf{r}$ for the preconditioner M are required repeatedly (see the linear PCG Algorithm [A3]). The products $H\mathbf{d}$ can generally be computed satisfactorily by the following finite-difference design of gradients, at the expense of only one additional gradient evaluation per inner iteration:

$$H_k\mathbf{d} \approx \frac{\mathbf{g}(\mathbf{x}_k + h\mathbf{d}) - \mathbf{g}(\mathbf{x}_k)}{h}. \qquad [58]$$

For simplicity, h may be the same value for all components of $\mathbf{x}$. As discussed in reference to discrete Newton methods, the interval h must be a suitably chosen small number, such as

$$h = \frac{2\sqrt{\epsilon_m}(1 + \|\mathbf{x}_k\|)}{\|\mathbf{d}\|}, \qquad [59]$$

to balance roundoff with truncation error.[5,58] This gradient-difference approximation to $H\mathbf{d}$ avoids explicit full Hessian calculations and is advantageous in

practice. In molecular mechanics and dynamics, for example, the expensive second derivatives of the nonbonded interactions need not be calculated explicitly when this option is used.[23,80,122]

The preconditioner is problem dependent and should be chosen in large-scale applications as a sparse approximation to H that can be factored rapidly. A Cholesky factorization of a positive-definite matrix M produces

$$M = LDL^T, \qquad [60]$$

where M is a lower triangular matrix with unit diagonals, and D is a diagonal matrix with positive entries.[5,14] Once the factors are available for each new M of the outer iteration, the repeated solutions for $\mathbf{z}$ at each inner iteration can be obtained by forward and backward substitutions:

$$\begin{gathered} LDL^T\mathbf{z} = \mathbf{r} \Rightarrow \\ L\mathbf{y} = \mathbf{r}, \qquad L^T\mathbf{z} = D^{-1}\mathbf{y}\,. \end{gathered} \qquad [61]$$

For example, a "local" preconditioner from the bond lengths, bond angles, and dihedral angle terms in computational chemistry calculations is a very successful choice of preconditioner whose computation grows slowly with n (approximately linearly).[122] Such a direct solution of $M\mathbf{z} = \mathbf{r}$ is feasible for large problems only when the sparsity is exploited in the factorization.[89,90,125–133] In sparse factorization techniques, such as the Yale Sparse Matrix Package (YSMP), only the nonzero values are compactly stored. Typically, this involves only several one-dimensional arrays that contain both the elements and the associated index pointers. Arithmetic operations are then performed on the nonzeros only.[121,125–128] Furthermore, a reordering of the rows and columns of M before the factorization can also be applied to minimize "fill-in" (i.e., the introduction of nonzeros in the factors of M in positions that were zeros in M).[89,125–129,132–134] This can be seen, for example, by comparing the structures of the two matrices in Figures 2b (original ordering) and 2c (after reordering by the minimum degree algorithm).[129,132] If the sparsity structure of M can be specified a priori (e.g., by a natural connectivity structure of the physical model[122,123]), the permutation matrix for reordering can be determined only once, at the onset of minimization. For example, the minimum-degree reordering[129,132] requires $O(n^2)$ operations. The subsequent factorization and triangular solution involve $O(l)$ to $O(l^2)$ operations, where l is the number of nonzeros in the Cholesky factor L of M. For band or block-band matrices, for example, the sparsity pattern is preserved in the Cholesky factors. Thus, if the number of nonzeros in each row of M is a constant c, independent of n, $l = cn$ and the additional work, associated with preconditioning at each inner iteration, can be as low as $O(n)$. Indeed, efficient block reorderings and block truncated Newton methods have been investigated.[118,135,136]

When the preconditioner is constructed from a natural separability of the problem into terms of differing complexity, it may not necessarily be positive-definite, as required for straightforward implementation of PCG. A very useful technique for optimization has been a replacement of the standard Cholesky factorization of positive-definite systems by a modified Cholesky (MC) algorithm.[5,137–139] The MC process detects indefiniteness during the factorization itself and produces a decomposition for

$$\overline{M} = M + \overline{D} = LDL^{\mathrm{T}}, \qquad [62]$$

where D is a positive diagonal matrix, at a similar cost to the standard factorization. When M is positive-definite, $\overline{D}$ is identically zero. Otherwise, the choice of modifications attempts to balance prescription of large values, which guarantee positive-definiteness but perturb M excessively, with small, just sufficient additions that may lead to large modifications later on. Such larger modifications arise from the algebraic formulas for the elements of the factors that involve division by the diagonals.[5,14]

Two effective MC strategies have been used in the context of optimization: the Gill, Murray, and Wright[5] and Schnabel and Eskow[137,138] procedures. These modified factorizations can also be more generally applied to modified Newton methods for solving $H\mathbf{p} = -\mathbf{g}$ when the Hessian is available. This technique produces an effective descent search direction $\mathbf{p} = -[H^{-1} + \overline{D}]\mathbf{g}$. In either case (indefinite preconditioner or indefinite Hessian), MC factorizations are appropriate. Although the MC solution may bear little resemblance to the solution of the original system (when it exists), positive-definite preconditioners or descent directions will be produced appropriately.

Overall, a combination of effective truncation criterion, preconditioner, factorization of the preconditioner, and calculation of Hessian/vector products can produce a very powerful TN algorithm. Perhaps more than others, TN methods require problem tailoring for best performance.

The inner loop of a TN algorithm at Newton step k (step 2 of algorithm [A4]) is sketched here. For clarity, we omit the subscript k from $\mathbf{p}$, $\mathbf{g}$, H, M, and q. Thus, the sequence of vectors $\{\mathbf{p}^i\}$ denotes the PCG iterates for $\mathbf{p}_k$. A small positive number δ for the negative curvature test, such as $\sqrt{\epsilon_m}$, is chosen, along with appropriate values of c_r or c_q (for truncation tests RT and QT).

Algorithm [A5]: Truncated Newton (Inner Loop of Outer Step *k*)

0. *Initialization*
   Set $\mathbf{p}^0 = 0$, $q^0 \equiv q_k(\mathbf{p}^0) = 0$, $\mathbf{r}_0 = -\mathbf{g}$, and $\mathbf{d}_0 = M^{-1}\mathbf{r}_0$.
   For $i = 0,1,2, \ldots$, proceed as follows:
1. *Negative curvature test*
   If $\mathbf{d}_i^{\mathrm{T}} H \mathbf{d}_i < \delta \mathbf{d}_i^{\mathrm{T}} \mathbf{d}_i$,
   exit inner loop with search direction

$$\mathbf{p} = \begin{cases} \mathbf{d}_0 & \text{if } i = 0 \\ \mathbf{p}^i & \text{otherwise.} \end{cases}$$

2. *Truncation test*

$$\begin{aligned} &\alpha_i = \mathbf{r}_i^{\mathrm{T}}(M^{-1}\mathbf{r}_i)/\mathbf{d}_i^{\mathrm{T}}H\mathbf{d}_i \\ &\mathbf{p}^{i+1} = \mathbf{p}^i + \alpha_i\mathbf{d}_i \\ &\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i H\mathbf{d}_i \\ &q^{i+1} = \tfrac{1}{2}(\mathbf{r}_{i+1} + \mathbf{g})^{\mathrm{T}}\,\mathbf{p}^{i+1} \qquad \text{(for QT)} \\ &\text{If } \|\mathbf{r}_{i+1}\| \le \min\{c_r/k,\ \|\mathbf{g}\|\} \cdot \|\mathbf{g}\| \qquad \text{(for RT)} \\ \text{or} \quad &\text{if } (1 - q^i/q^{i+1}) \le c_q/i \qquad \text{(for QT)} \end{aligned} \qquad \text{[A5]}$$

exit inner loop with search direction $\mathbf{p} = \mathbf{p}^{i+1}$.

3. *Continuation of PCG*

$$\begin{aligned} \beta_k &= \mathbf{r}_{i+1}^{\mathrm{T}}(M^{-1}\mathbf{r}_{i+1})/\mathbf{r}_i^{\mathrm{T}}(M^{-1}\mathbf{r}_i) \\ \mathbf{d}_{i+1} &= (M^{-1}\mathbf{r}_{i+1}) + \beta_k\mathbf{d}_i \end{aligned}$$

Note that in case of negative curvature, $\mathbf{p}$ is set in step 1 to $-M^{-1}\mathbf{g}$ if this occurs at the first PCG iteration, or to the current iterate for $\mathbf{p}_k$ thereafter. These choices are guaranteed directions of descent.[114] Alternate descent directions can also be used, such as $-\mathbf{g}$ or $\mathbf{d}_i$, but the "default" choices above have been found to be satisfactory in practice.

PERSPECTIVE AND COMPUTATIONAL EXAMPLES

Comparisons

Table 1 compares computational effort, storage requirements, and performance characteristics among five methods: "classic" Newton (for reference), nonlinear CG, full-memory QN, limited-memory QN, and TN methods.

We assume that the function value and gradient are evaluated together in αn operations (additions and multiplications), where n is the problem size and α is a problem-dependent number. The Hessian can then be computed in $(\alpha/2)n(n + 1)$ operations. When a sparse preconditioner M is used, we denote its number of nonzeros by m and the number of nonzeros in its Cholesky factor, L, by l. (We assume here that M either is positive-definite or is factored by a modified Cholesky factorization.) Thus M can be computed in about $(\alpha/2)nm$ operations. As discussed in the previous section, it is advantageous to reorder the variables a priori to minimize the fill-in for M. Alternatively, a precon-

Table 1 Comparison Among Five Minimization Methods[a]

	Classic Newton	Nonlinear Conjugate Gradient	Quasi-Newton (Full Memory)	Quasi-Newton (Limited Memory)	Truncated Newton
Tasks per iteration					
Form **g**	αn	αn	αn	αn	αn
Form H	$(\alpha/2)n(n+1)$				
Form M					$m(\alpha n)$
Factor M					$O(l) - (l^2)$
Calculate **p**	$\frac{1}{6}n^3$	$7n$	$O(n^2)$	$23n$ $[+ O(l) - O(l^2)]$	IT$(\alpha n + 7n + O(l))$ $< \alpha n$
Perform line search		$\sim (2\text{–}4)\alpha n$	αn	αn	
Storage	$O(n^2)$	$3n-7n$	$O(n^2)$	$14n$ $[+ O(\max\{n,m\})]$	$O(\max\{n, m\})$
Advantages	Locally quadratically convergent	Easy to implement Modest storage	Locally superlinearly convergent	Adapts well to available storage	Exploits problem structure to accelerate convergence (by

		and computation cost		preconditioning) Can display local quadratic convergence
Disadvantages	Requires $O(n^2)$ storage and $O(n^3)$ work	Convergence often slow Requires fairly accurate line search	Requires $O(n^2)$ storage	Requires construction and factorization of preconditioner Performance may be slow for highly nonlinear functions when directions of negative curvature are detected repeatedly

[a] *Key:* **g**, gradient; H, Hessian; M, preconditioner; **p**, search vector; n, problem size; α, positive constant (problem dependent); m, number of nonzeros of M; l, number of nonzeros of the Cholesky factor of M; IT, number of inner preconditioned conjugate gradient iterations per outer Newton step. The LMQN method models an LM-BFGS algorithm with five updates and a compact formula for the matrix/vector multiplications; terms in brackets are relevant if preconditioning is used to scale the initial QN direction by the inverse of a preconditioner. The cost per inner TN iteration assumes that Hessian/vector products are computed by a finite-difference design and the preconditioner's linear system is solved efficiently from its Cholesky factors by forward and backward substitutions. Each line search iteration requires a new set of function and gradient for a new cubic interpolant (αn operations by assumption), but the actual number of iterations varies widely with problem size and specified accuracy of line search.

ditioner with a factorization-conserving pattern may be used. By the use of efficient factorization schemes (e.g., the YSMP described earlier), M can be factored in $O(l)$ to $O(l^2)$ operations, and $M\mathbf{z} = \mathbf{r}$ can be solved in $O(l)$ operations.[126–128]

Calculation of the search vector varies from method to method. For the classic Newton method, we assume that the Newton equation [36] is solved via Cholesky factorization. For the nonlinear CG method, we consider a simple recursive relation such as described in Eq. [30]. The computational complexity for obtaining $\mathbf{p}$ in CG ($7n$) is a typical estimate reflecting intermediate quantities that are computed to determine whether restarts and/or modifications of the updating scalars are necessary.[61,83] In the full-memory QN method, we assume that an inverse QN approximation $\tilde{B}$ is computed and the search vector is obtained by an efficient multiplication scheme.[100,101] For the limited-memory version, we model the BFGS algorithm of Liu and Nocedal[100,101] with the number of updates fixed at 5. If preconditioning were used in LM-BFGS, the additional quantities indicated in brackets (for calculating $\mathbf{p}$ and storing M) are relevant.

Truncated Newton methods can be competitive only with preconditioning. Thus, the operation count for obtaining $\mathbf{p}$ in TN reflects IT inner PCG iterations per Newton step. Each such inner iteration involves the following operations: an $H\mathbf{d}$ multiplication (αn, for an additional gradient evaluation in this finite-difference approximation [58]); calculation of the PCG vectors and scalars ($7n$, Algorithm [A4]); and numerical solution of $M\mathbf{z} = \mathbf{r}$ by forward and backward substitution ($O(l)$, see [61]).

The line search requirements are more difficult to estimate as they are strongly problem and size dependent. Recall that each line search iteration requires an additional set of function and gradient for a new cubic interpolant. Although CG methods clearly perform better with fairly accurate line searches (a rough estimate of two to four gradients is given), TN (with preconditioning) and QN methods often produce search vectors that already satisfy the decrease criteria [18a,b] with $\lambda = 1$. Thus, the complexity αn for TN's and QN's line search (one line search iteration per Newton iteration) is generally an overestimate. Although the overall computational effort per Newton step is shown in Table 1 for the five methods, a fair comparison among them considers the total number of inner iterations in the case of TN methods.

Table 1 demonstrates that a wide range of storage and computation can be accommodated. The CG, LMQN, and TN algorithms are clearly the minimization candidates for large-scale problems. An efficient CG method with restarts, CONMIN, has been developed by Shano and Phua[83] and implemented in the NAG minimization library. The same code also contains a full-memory BFGS QN method. The LM-BFGS method[100,101] will soon be available in the Harwell software library (routine VA05AD). TN packages have been developed by Nash[109,115–117] and by Schlick and Fogelson (TNPACK).[82,121,122] TNPACK is distributed through *ACM Transactions on Mathematical Software* and netlib. Nash's TN package uses an automatic diagonal preconditioner derived from

the BFGS update matrix; TNPACK factors a user-supplied sparse preconditioner by a modified Cholesky factorization.[139]

Performance comparisons on standard optimization test problems[140,141] (some of which involve unusually difficult characteristics of scaling, structure, etc., that are rarely encountered in practice) and a few real-life problems suggest that the LM-BFGS and TN methods are preferable overall for reliability and efficiency.[101,102,109,110] For highly nonlinear problems, the LM-BFGS method generally requires fewer function evaluations (the time-dominating operation) than CONMIN and performs better than the tested TN algorithm of Nash. The TN method was found to perform better than LM-BFGS for functions that are nearly quadratic. A reasonable explanation for this is that the approximate Newton step is a good search direction for such problem structure. In contrast, both Nash's algorithm and TNPACK were found to perform better on very large-scale problems in meteorology (15,000 variables), where the objective function is quasi-quadratic.[110]

Problem-tailored preconditioning improves performance of both LM-BFGS and TN methods substantially,[80,122] but this remains to be tested systematically for large-scale problems.

Several computational examples are shown in Figures 14 and 15 and Tables 2 to 6. Three minimization codes are examined: CONMIN (nonlinear CG and full-memory BFGS), LM-BFGS, and TNPACK (TN method). Preconditioning options, the number of stored updates (for LM-BFGS), starting points, and problem dimensions are varied for analysis.

Numerical Example I: Rosenbrock Minimization

Rosenbrock's function is often used as a minimization test problem, because its minimum lies at the base of a "banana-shaped valley" and can be difficult to locate. This function is defined for even integers n as the sum

$$f(\mathbf{x}) = \sum_{j=1,3,5,\ldots,n-1} [(1 - x_j)^2 + 100(x_{j+1} - x_j^2)^2]. \qquad [63]$$

The contour plot of Rosenbrock's function for $n = 2$ is shown in Figure 14. The minimum point is (1,1), where $f(\mathbf{x}) = 0$. The gradient components of this function are given by

$$\left.\begin{aligned} g_{j+1} &= 200(x_{j+1} + x_j^2) \\ g_j &= -2[x_j g_{j+1} + (1 - x_j)] \end{aligned}\right\}, \quad j = 1,3,5,\ldots,n-1, \qquad [64]$$

and the Hessian is the 2×2 block diagonal matrix with entries

$$\left.\begin{aligned} H_{j+1,j+1} &= 200 \\ H_{j+1,j} &= -400x_j \\ H_{j,j} &= -2(x_j H_{j+1,j} + g_{j+1} - 1) \end{aligned}\right\}, \quad j = 1,3,5,\ldots,n-1. \qquad [65]$$

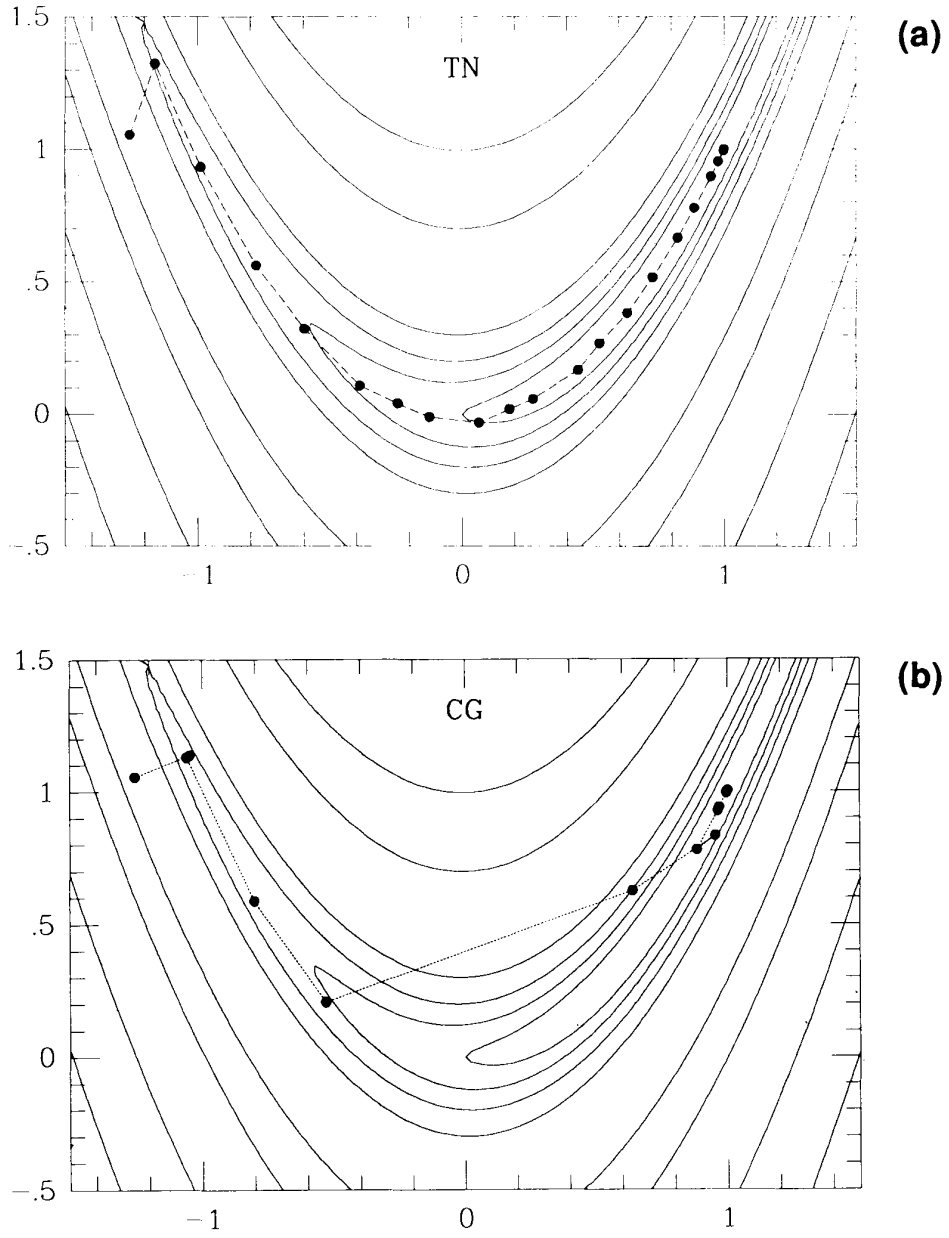

Figure 14 Minimization paths by different methods for the two-dimensional Rosenbrock function $f(x_1,x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$:(**a**) truncated Newton, (**b**) conjugate gradient, (**c**) BFGS quasi-Newton, (**d**) limited-memory BFGS, (**e**) steepest descent, first 150 iterations. See program output in Figure 15.

(Note that these formulas are given in a form most efficient for programming.) For $n = 2$, the two eigenvalues of the Hessian at the minimum are $\lambda_1 = 1001.6$ and $\lambda_2 = 0.4$ (thus $\kappa = 2.5 \times 10^3$), so the contours are quite elongated near the minimum.

Table 2 shows results for minimization in the two-dimensional case. We use this example to display the various minimization paths on the function's

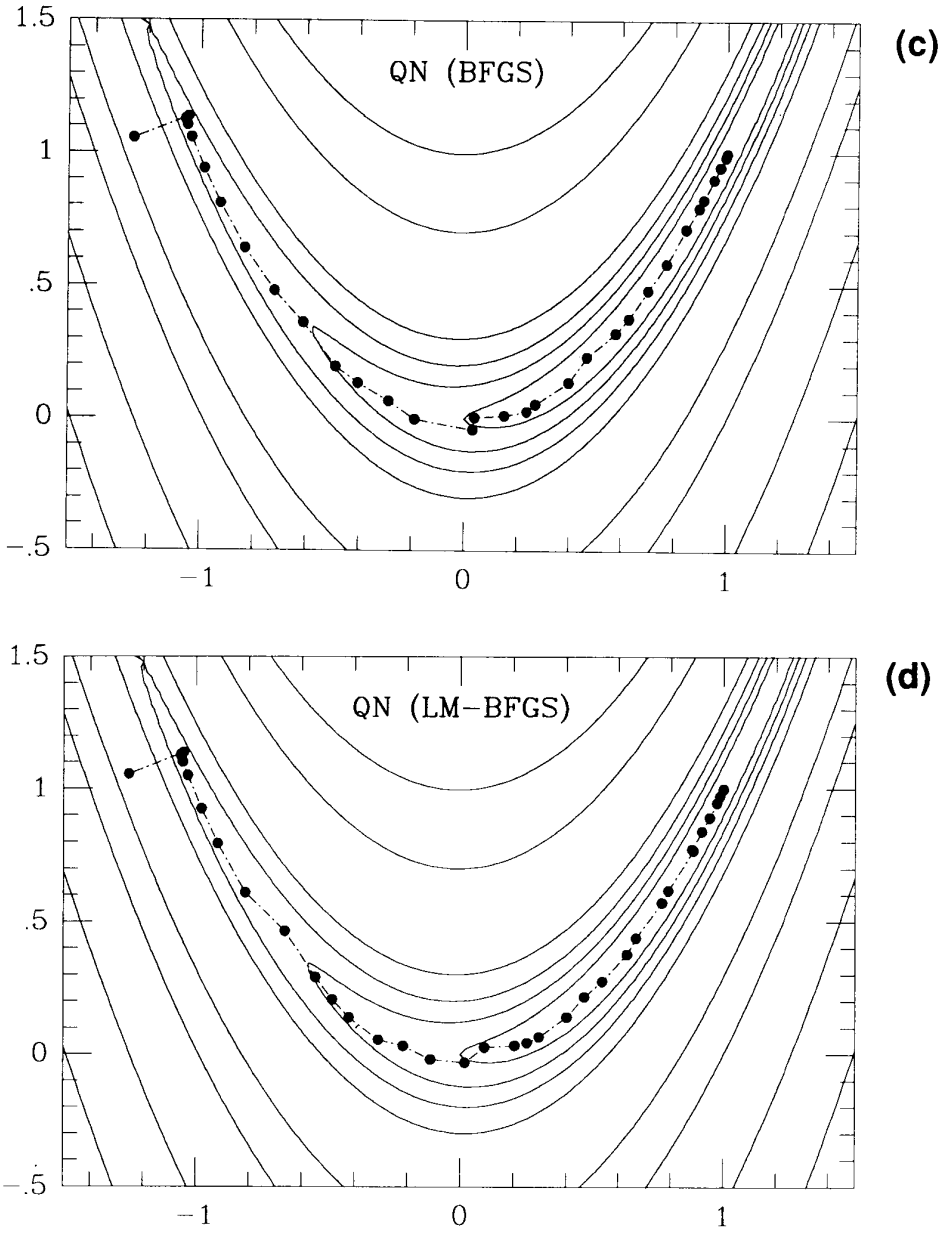

Figure 14 *(continued)*

contour plots (Figure 14). TN uses a diagonal preconditioner, and, unless otherwise stated, LM-BFGS uses five stored updates. Progress is reported by the number of iterations (for TN, both outer and inner), the number of function and gradient evaluations (NFG), CPU time, and final gradient norm and function values. TN uses the convergence criteria described in [19] and [21a–c] with $\epsilon_f = \epsilon_g = 10^{-8}$, whereas the other methods use condition [19] with $\epsilon_g = 10^{-8}$.

We note from Table 2 that performance of all methods is very similar in terms of iteration and function cost, accuracy, and time. All are equally satisfac-

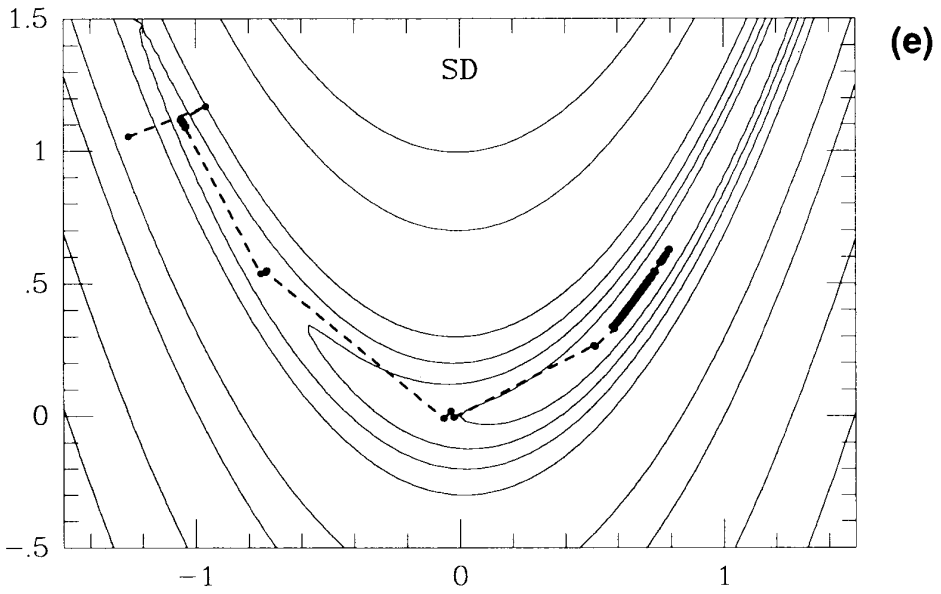

Figure 14 *(continued)*

tory. In TN, the residual norm is essentially zero after two PCG iterations per Newton step (as expected because $n = 2$), so the behavior of the method is essentially that of a nontruncated Newton method (criterion RT defined in [54] and [55] was used with $c_r = 0.5$). For the same reason (low dimensionality), the full and limited-memory QN methods are almost identical in performance (the number of updates in LM-BFGS exceeds n). The paths in the Newton methods are more systematic, following down the valley; the CG path is more arbitrary. The large step between the fourth and fifth iterate of CG results from a distant minimum detected in the line search. Nonetheless, CG performs well overall.

Partial output produced from these programs is shown in Figure 15. Note the quadratic convergence rate in TN and the slightly slower convergence in the other methods. CG requires more than two function evaluations per step, and its step lengths vary significantly in magnitude from iteration to iteration. In contrast, the Newton methods require around one function evaluation per Newton step, and the step lengths are often one.

For comparison, a steepest descent procedure was also examined. For this purpose, the TN code was modified so that only one inner iteration was performed, without preconditioning, and the exit search direction was the negative gradient. We note from Figure 14e and the output in Figure 15 that the SD progress is initially very rapid but then becomes excessively slow. Only 150 steps of minimization are shown in the contour figure, but the output indicates how slow progress is through several hundred iterations (the process was then stopped). This is typical of the SD method and explains why SD steps may often work well only initially, in regions far away from stationary points.

Table 2 Rosenbrock Minimization, $n = 2^a$

Method	Iterations (Outer/Inner)	NFG	CPU (s)	$\|\mathbf{g}^*\|_2$	f^*
TN	22/43	27	10	5.2×10^{-11}	1.7×10^{-23}
CG	14	31	9	1.1×10^{-12}	7.0×10^{-27}
BFGS	38	47	9	3.6×10^{-10}	1.3×10^{-22}
LM-BFGS	40	49	9	4.2×10^{-10}	2.3×10^{-22}

[a]The starting point $(-1.25,1.05)$ has $f = 3.2 \times 10^1$ and $\|\mathbf{g}\|_2 = 2.8 \times 10^3$. See Figures 14 and 15.

Table 3 shows results for Rosenbrock minimization with $n = 1000$. An asymmetrically perturbed starting point was used (see footnote in table). Note that TN is very economical in terms of iteration cost and function evaluations, and the CG method is equally cheap and effective. It is likely that CG performs well because of the clustered eigenvalue structure. As the Hessian has a 2×2 block diagonal structure, multiple eigenvalues result. The full-memory BFGS method is inappropriate here because of the high dimensionality (CPU time is greater by three orders of magnitude than for other methods). LM-BFGS requires more iterations and function evaluations and consequently more time than TN and CG, but differences are still within the same order of magnitude. These results demonstrate how effective Newton variants for large dimensions can be and how, even in terms of time, they are similar or better than the CG method. In reliability they typically exceed the best gradient methods.

Numerical Example II: Deoxycytidine

The molecular model of deoxycytidine is examined in Tables 2 and 3 to analyze the issues of preconditioning and number of LM-BFGS updates. Energy model details are described elsewhere,[22,23] and here it suffices to note that several well-separated local minima correspond to different feasible combinations of the sugar's pseudorotation parameter and the glycosyl (sugar-to-base orientation) dihedral angle.

A natural preconditioner consisting of the local potential energy terms has been found to be very effective here. Its nonzero structure is shown in Figure 2b: an overall block-diagonal clustered pattern is apparent from the separate bond coupling within the sugar and base units. The symmetry in the x, y, z components of the Cartesian variables also produces smaller 3×3 block repeating patterns. In the original labeling scheme, the x, y, z components of each atom are numbered in turn. After a YSMP reordering in TN, the pattern shown in Figure 2c results, producing a smaller factorization fill-in, as the first nonzeros in some rows are pushed further to the right. Our experience has

Performance for the Rosenbrock Function (n = 2)

(a) Truncated Newton

ITN	NF	F	GNORM/sqrt(N)	STEPLEN	x(1)	x(2)
0	1	31.9712644016	0.200976E+03	0.0	-1.25403023	1.05403023
1	2	4.7190600539	0.110278E+02	0.0	-1.16031887	1.32351830
2	4	4.1312457547	0.158624E+02	0.5	-0.98763945	0.93294226
3	5	3.3543889611	0.136434E+02	1.0	-0.77836768	0.56206156
4	6	2.6579313439	0.915315E+01	1.0	-0.59613799	0.32217285
5	7	2.1148470328	0.913832E+01	1.0	-0.38726128	0.10634181
6	8	1.5904843317	0.427944E+01	1.0	-0.24462545	0.03949664
7	10	1.3252913900	0.429829E+01	1.0	-0.12393941	-0.00954918
8	11	1.0008130364	0.503915E+01	1.0	0.06394650	-0.03121196
9	12	0.6904597544	0.196500E+01	1.0	0.18007895	0.01894168
10	14	0.5571407008	0.231509E+01	0.4	0.27158530	0.05746357
11	15	0.3962408358	0.503692E+01	1.0	0.44261475	0.16665674
12	16	0.2309761792	0.985671E+00	1.0	0.52398231	0.26793678
13	18	0.1579123666	0.285678E+01	0.5	0.62920614	0.38160901
14	19	0.0839849202	0.199371E+01	1.0	0.72530978	0.51683837
15	20	0.0404234905	0.232090E+01	1.0	0.82178771	0.66602705
16	21	0.0149451840	0.974807E+00	1.0	0.88406490	0.77769229
17	22	0.0043820612	0.123077E+01	1.0	0.94935508	0.89701226
18	23	0.0005990766	0.203154E+00	1.0	0.97669286	0.95318159
19	24	0.0000260883	0.125880E+00	1.0	0.99696927	0.99353659
20	25	0.0000000592	0.219316E-02	1.0	0.99976973	0.99953167
21	26	0.0000000000	0.162610E-04	1.0	0.99999964	0.99999923
22	27	0.0000000000	0.363783E-10		1.00000000	1.00000000

(b) CONMIN Conjugate Gradient

ITN	NF	F	GNORM	STEPLEN	x(1)	x(2)
0	1	31.9712644016	0.28D+03	0.0	-1.25403023	1.05403023
1	3	4.3447824695	0.15D+02	0.0	-1.04681947	1.13524071
2	5	4.2431104032	0.18D+01	1.3	-1.05851069	1.12795763
3	8	3.5167814324	0.23D+02	448.6	-0.80080080	0.58894666
4	10	2.9090812308	0.24D+02	1.7	-0.53169809	0.20767072
5	12	4.8533295171	0.71D+02	0.5	0.63952750	0.62632901
6	14	0.5540987318	0.32D+02	0.0	0.95243688	0.83285031
7	16	0.0136801009	0.28D+00	0.5	0.88358200	0.77959043
8	19	0.0027257658	0.16D+01	194.5	0.96370498	0.92497436
9	21	0.0010528129	0.20D+00	1.4	0.96771111	0.93678481
10	23	0.0001239198	0.47D+00	44.4	0.99651984	0.99199439
11	25	0.0000001044	0.54D-02	0.3	1.00030472	1.00059877
12	27	0.0000000000	0.60D-04	5.8	1.00000048	1.00000109
13	29	0.0000000000	0.10D-06	0.2	0.99999999	0.99999999
14	31	0.0000000000	0.11D-11		1.00000000	1.00000000

(c) CONMIN Quasi Newton (BFGS)

ITN	NF	F	GNORM	STEPLEN	x(1)	x(2)
0	1	31.9712644016	0.28D+03	0.0	-1.25403023	1.05403023
1	3	4.3447824695	0.15D+02	0.0	-1.04681947	1.13524071
2	4	4.2475141998	0.34D+01	1.0	-1.05608572	1.12946827
3	5	4.2407855297	0.18D+01	1.0	-1.05797103	1.12674634
4	6	4.2375454227	0.20D+01	1.0	-1.05771783	1.12454866
5	7	4.2107433097	0.55D+01	1.0	-1.05176394	1.10303239
6	8	4.1604732801	0.10D+02	1.0	-1.03496426	1.05722488
7	9	4.0489517601	0.18D+02	1.0	-0.98560409	0.93880742
8	10	3.9108605034	0.23D+02	1.0	-0.92370225	0.80737499
9	11	3.6375512593	0.24D+02	1.0	-0.83242480	0.64003770
10	12	3.1130242950	0.17D+02	1.0	-0.71996811	0.47901784
11	13	2.6057378503	0.66D+01	1.0	-0.60967227	0.35957879
12	15	2.4248857977	0.15D+02	0.3	-0.48915091	0.19373672

Figure 15 Minimization output by the different methods corresponding to Figure 14.

13	16	2.0625458857	0.99D+01	1.0	-0.40294777	0.13166130
14	17	1.6800702763	0.54D+01	1.0	-0.28627104	0.06595828
15	19	1.5573528088	0.93D+01	0.4	-0.18795897	-0.00289528
16	20	1.1262179775	0.89D+01	1.0	0.03313867	-0.04265079
17	22	0.9210925703	0.20D+01	0.5	0.04038753	0.00316896
18	23	0.7341555025	0.27D+01	1.0	0.15290463	0.01050157
19	25	0.6702828768	0.62D+01	0.3	0.23896553	0.02692019
20	26	0.5695542610	0.41D+01	1.0	0.27253092	0.05418754
21	27	0.4177832527	0.55D+01	1.0	0.39964271	0.13576552
22	28	0.2958148057	0.40D+01	1.0	0.46848904	0.23101928
23	30	0.1939731541	0.34D+01	0.5	0.57898880	0.32229639
24	31	0.1703468728	0.51D+01	1.0	0.62756400	0.37604941
25	32	0.0961174146	0.22D+01	1.0	0.70008751	0.48226764
26	33	0.0642768826	0.36D+01	1.0	0.77097074	0.58352275
27	34	0.0252997774	0.14D+01	1.0	0.84353739	0.71441754
28	36	0.0148034988	0.22D+01	0.3	0.89361486	0.79264354
29	37	0.0097699591	0.16D+01	1.0	0.91082832	0.82534400
30	38	0.0028700428	0.76D+00	1.0	0.95022938	0.90095364
31	39	0.0007585852	0.32D+00	1.0	0.97376724	0.94738342
32	40	0.0001490214	0.44D+00	1.0	0.99336586	0.98575098
33	42	0.0000243098	0.44D-01	0.3	0.99513193	0.99036576
34	43	0.0000002273	0.12D-01	1.0	0.99961778	0.99920720
35	44	0.0000000009	0.12D-02	1.0	0.99998676	0.99997077
36	45	0.0000000000	0.20D-04	1.0	0.99999953	0.99999910
37	46	0.0000000000	0.36D-06	1.0	1.00000002	1.00000003
38	47	0.0000000000	0.36D-09		1.00000000	1.00000000

(d) Limited Memory Quasi Newton (LM-BFGS)

ITN	NF	F	GNORM	STEPLEN	x(1)	x(2)
0	1	0.319713D+02	0.28D+01	0.0	-1.25403023	1.05403023
1	3	0.434478D+01	0.15D+02	0.0	-1.04681947	1.13524071
2	4	0.424751D+01	0.34D+01	1.0	-1.05608572	1.12946827
3	5	0.424023D+01	0.18D+01	1.0	-1.05783541	1.12646330
4	6	0.423599D+01	0.21D+01	1.0	-1.05745111	1.12357217
5	7	0.420697D+01	0.58D+01	1.0	-1.05072830	1.10017373
6	8	0.415140D+01	0.11D+02	1.0	-1.03174877	1.04921001
7	9	0.403468D+01	0.19D+02	1.0	-0.97885197	0.92367947
8	10	0.389041D+01	0.23D+02	1.0	-0.91692131	0.79428758
9	11	0.356400D+01	0.23D+02	1.0	-0.81307500	0.60848319
10	12	0.282372D+01	0.56D+01	1.0	-0.66334001	0.46389861
11	14	0.241029D+01	0.59D+01	0.3	-0.54862332	0.29000961
12	16	0.230043D+01	0.11D+02	0.2	-0.48566319	0.20533463
13	17	0.217535D+01	0.12D+02	1.0	-0.42201285	0.13895081
14	18	0.188409D+01	0.11D+02	1.0	-0.31093596	0.05599548
15	19	0.149586D+01	0.47D+01	1.0	-0.21445210	0.03151132
16	21	0.134450D+01	0.75D+01	0.5	-0.11220442	-0.02019741
17	22	0.106786D+01	0.66D+01	1.0	0.01739154	-0.03168826
18	23	0.858817D+00	0.43D+01	1.0	0.09066241	0.02608642
19	25	0.638090D+00	0.20D+01	0.4	0.20664523	0.03338670
20	27	0.602068D+00	0.41D+01	0.2	0.25172298	0.04283403
21	28	0.546289D+00	0.48D+01	1.0	0.29814112	0.06571853
22	29	0.403118D+00	0.49D+01	1.0	0.40273171	0.14065476
23	30	0.283682D+00	0.81D+00	1.0	0.46852891	0.21602615
24	32	0.233596D+00	0.35D+01	0.3	0.53753329	0.27489902
25	33	0.185462D+00	0.66D+01	1.0	0.63165581	0.37667664
26	34	0.114369D+00	0.11D+01	1.0	0.66504445	0.43762155
27	35	0.757133D-01	0.49D+01	1.0	0.76525532	0.57126012
28	36	0.472804D-01	0.16D+01	1.0	0.78893336	0.61718967
29	37	0.318827D-01	0.53D+01	1.0	0.88470457	0.76906776
30	38	0.146830D-01	0.37D+00	1.0	0.87887674	0.77277349
31	39	0.698355D-02	0.45D+00	1.0	0.91761875	0.84062103
32	41	0.380960D-02	0.12D+01	0.4	0.94577378	0.89153996
33	42	0.135593D-02	0.11D+01	1.0	0.97389620	0.94587665
34	43	0.243668D-03	0.25D+00	1.0	0.98522645	0.97117525
35	44	0.151047D-04	0.16D+00	1.0	0.99864344	0.99692451

Figure 15 *(continued)*

36	45	0.121905D-05	0.29D-01	1.0	0.99912536	0.99818410
37	46	0.443379D-09	0.90D-03	1.0	1.00000699	1.00001200
38	47	0.674012D-12	0.29D-04	1.0	0.99999946	0.99999899
39	48	0.724553D-17	0.33D-07	1.0	1.00000000	0.99999999
40	49	0.233483D-21	0.42D-09	1.0	1.00000000	1.00000000

(e) Steepest Descent

ITN	F	GNORM/sqrt(N)	x(1)	x(2)
0	31.9712644016	0.200976E+03	-1.25403023	1.05403023
1	9.7707222610	0.721645E+02	-0.96186693	1.16853549
2	4.2274537586	0.217954E+01	-1.05592075	1.11750564
3	4.2213111549	0.192507E+01	-1.05082879	1.11665583
4	4.2155533092	0.142088E+01	-1.05233890	1.11329794
5	4.2079229687	0.325712E+01	-1.04414056	1.10737954
6	4.1963209609	0.134753E+01	-1.04749029	1.10364253
7	4.1854935028	0.437931E+01	-1.03516795	1.09244969
8	4.1658385218	0.125080E+01	-1.03952827	1.08846948
9	3.1678216258	0.995583E+01	-0.75355730	0.53737593
10	3.0208933705	0.278715E+01	-0.72899166	0.54917178
11	3.0046493622	0.143926E+01	-0.73218938	0.54255831
12	1.1403403446	0.242356E+01	-0.06093734	-0.00843255
13	1.0995164007	0.277594E+01	-0.03410434	0.01852531
14	1.0468611342	0.165908E+01	-0.02177217	-0.00485774
15	0.2470165941	0.201467E+01	0.50828643	0.26558999
16	0.2365334708	0.480024E+00	0.51418382	0.26211319
17	0.1990534051	0.328990E+01	0.58755398	0.32820743
18	0.1769626768	0.788015E+00	0.57947114	0.33687380
19	0.1755935169	0.739630E+00	0.58391475	0.33598997
20	0.1740143784	0.451228E+00	0.58296243	0.33887544
21	0.1725102330	0.794984E+00	0.58797298	0.34047394
22	0.1707997980	0.435533E+00	0.58685531	0.34334435
23	0.1692217280	0.823421E+00	0.59214305	0.34527202
24	0.1674507388	0.425773E+00	0.59093640	0.34812090
25	0.1658646065	0.819472E+00	0.59622131	0.35016254
26	0.1641328576	0.419263E+00	0.59501209	0.35295471
27	0.1625880839	0.795206E+00	0.60010779	0.35495799
28	0.1609558783	0.414558E+00	0.59894953	0.35767097
29	0.1594766286	0.762886E+00	0.60377971	0.35956385
30	0.1579616513	0.411215E+00	0.60269350	0.36219448
31	0.1565595758	0.727546E+00	0.60722979	0.36394143
32	0.1551620911	0.409214E+00	0.60622380	0.36649539
33	0.1538434159	0.690754E+00	0.61045468	0.36807468
34	0.1525580902	0.408733E+00	0.60953350	0.37056153
35	0.1513258950	0.653010E+00	0.61345478	0.37195791
36	0.1501451969	0.410099E+00	0.61262257	0.37439032
37	0.1489998542	0.614568E+00	0.61623530	0.37559322
38	0.1479146236	0.413814E+00	0.61549694	0.37798783
39	0.1468534597	0.575755E+00	0.61880816	0.37899131
40	0.1458524468	0.420602E+00	0.61816998	0.38137071
.				
50	0.1369909914	0.543482E+00	0.62987873	0.39685848
.				
100	0.0835178107	0.362718E+00	0.71101517	0.50530633
.				
200	0.0316149860	0.135948E+00	0.82231263	0.67554856
.				
300	0.0066762375	0.614025E-01	0.91833070	0.84307902
.				
400	0.0041614876	0.443911E-01	0.93556265	0.87497226
.				
500	0.0009081323	0.360715E-01	0.96994575	0.94057398
.				
600	0.0006846991	0.210984E-01	0.97384144	0.94830167
.				
700	0.0004944224	0.145696E-01	0.97778643	0.95596732

Figure 15 *(continued)*

```
 .
 800   0.0000669359   0.201061E-01   0.99187333   0.98371820
 .
 900   0.0000399588   0.119952E-01   0.99370840   0.98739518
 .
1000   0.0000282049   0.833224E-02   0.99470865   0.98939985
 .
1100   0.0000216950   0.623094E-02   0.99535619   0.99069789
 .
1200   0.0000175591   0.486285E-02   0.99582017   0.99162812
 .
 .
 .
```

Figure 15 *(continued)*

shown that this local preconditioner is often positive-definite, with possible exceptions at the first few minimization iterations.[139]

Tables 4 and 5 summarize performance results for two different starting points. The first (**x1**) is closer to a minimum than the second (**x2**) and has lower function value and gradient norm by about four orders of magnitude (see table footnotes for details). From both starting points, we first note how well preconditioning works in TN. The residual truncation criterion of [54] and [55] was used here with $c_r = 0.5$. With preconditioning, the number of inner (PCG) iterations is reduced by two to three orders of magnitude. Even the number of Newton iterations is reduced, and the time is accelerated by a factor of 2 to 3. Not only is precision of the resulting gradient norm not sacrificed; it improves. This is a typical observation with good preconditioning.

The CG method requires a large number of iterations and function evaluations (approximately twice the number of iterations). Although its cost per iteration is low, computational time is significantly greater than TN as well as the optimal version of LM-BFGS. BFGS is feasible for this problem size ($n = 87$) and performs relatively better for **x1** than **x2**, presumably because the procedure for updating from a better approximation to the minimum provides better curvature information to direct progress.

The unpreconditioned LM-BFGS runs are not very efficient overall, although their cost per iteration is far better than that of BFGS; however, precon-

Table 3 Rosenbrock Minimization, $n = 1000$[a]

Method	Iterations (Outer/Inner)	NFG	CPU (s)	$\|\mathbf{g}^*\|_2$	f^*
TN	23/127	30	24	1.5×10^{-7}	1.6×10^{-17}
CG	52	114	21	3.8×10^{-8}	1.1×10^{-18}
BFGS	188	208	2880	2.2×10^{-7}	4.8×10^{-14}
LM-BFGS	249	283	33	3.0×10^{-7}	1.1×10^{-13}

[a] The starting point corresponds to a random perturbation of the point $(-1.2, 1.0, -1.2, 1.0, \ldots)$ and has $f = 8.1 \times 10^3$ and $\|\mathbf{g}\|_2 = 4.0 \times 10^3$.

Table 4 Deoxycytidine Minimization from **x1**, $n = 87$[a]

Method	Iterations (Outer/Inner)	NFG	CPU (in 38 s units)
TN, no P	23/1536	27	2.8
TN, P	16/96	20	1.0
CG	1520	3002	11.8
BFGS	354	357	2.9
LM-BFGS, no P, $u = 5$	1551	1598	5.0
LM-BFGS, P			
$u = 5$	46	57	1.7
$u = 4$	46	58	1.7
$u = 3$	51	66	1.8
$u = 2$	66	81	3.1
$u = 1$	66	88	3.1
$u = 0$	711	1423	27.2

[a]The starting point corresponds to a pseudorotation puckering angle of 0° and a glycosyl dihedral angle $\chi = 180°$. The energy at this point has a value of $f = 1.4$ and $\|\mathbf{g}\|_2 = 7.6 \times 10^1$. At the minimum, $f_* = -5.7$ and $\|\mathbf{g}_*\|_2$ in all methods ranges from $O(10^{-8})$ to $O(10^{-5})$. The symbols "P" and "u" in the Method column refer to *preconditioning* (by the matrix of local Hessian interactions) and to the number of stored updates in LM-BFGS, respectively.

Table 5 Deoxycytidine Minimization from **x2**, $n = 87$[a]

Method	Iterations (Outer/Inner)	NFG	CPU (in 40 s units)
TN, no P	22/771	28	2.5
TN, P	20/81	24	1.0
CG	1029	2043	5.7
BFGS	1017	1021	4.9
LM-BFGS, no P, $u = 5$	831	862	2.5
LM-BFGS, P			
$u = 5$	71	187	2.3
$u = 4$	76	244	3.5
$u = 3$	74	200	3.4
$u = 2$	71	148	3.0
$u = 1$	66	137	3.0
$u = 0$	91	149	3.3

[a]The starting point corresponds to a pseudorotation puckering angle of −90° and a glycosyl dihedral angle $\chi = 90°$. The energy at this point has a value of $f = 5.1 \times 10^4$ and $\|\mathbf{g}\|_2 = 6.9 \times 10^5$. At the minimum, $f_* = -5.0$ and $\|\mathbf{g}\|_2$ in all methods ranges from $O(10^{-8})$ to $O(10^{-5})$. See footnote *a* to Table 4.

ditioning introduces significant improvements: the number of iterations and function evaluations drops sharply, and time is reduced as well. This illustrates the significant savings and improvement in reliability for Newton methods despite the more complex and expensive computations involved. TN and LM-BFGS with preconditioning are clearly the best methods overall for this problem.

The experiments with varying numbers of stored updates in the preconditioned LM-BFGS runs are meant to help assess the relative importance of updating and preconditioning. Overall performance deteriorates as the number of updates decreases, but not systematically. When $u = 0$, no updating is performed and the inverse of the preconditioner is used to scale the initial search direction. For **x1**, this version clearly shows a large performance deterioration, so the additional work in preconditioning does not pay in terms of time. A likely explanation is that, because **x1** is a good starting point, the QN updating is important for good performance. Interestingly, even a "little" updating here (e.g., $u = 1$) produces a very significant improvement. For **x2** on the other hand, preconditioning rather than updating appears to lead to iteration-count improvements.

In all these deoxycytidine runs, computation time is approximately proportional to the number of function and gradient evaluations. This is typical in molecular computations, as the evaluation and differentiation of the potential energy are expensive and the time-determining factors. The trends observed here extend to biomolecular models and suggest that, with local preconditioning, TN and LM-BFGS are the methods of choice.

Numerical Example III: Water Clusters

The water cluster examples in Table 6 are meant to illustrate a somewhat more difficult minimization problem. Optimal cluster-network geometries are more difficult to locate not only because of the difficulty in constructing good starting points; the optimal configurations are dominated by long-range nonbonded forces that are computationally not feasible to consider in preconditioners.

For these water runs, the potential energy consists of intermolecular Lennard-Jones and Coulombic terms, with the addition of intramolecular bond lengths and bond angle terms to permit a flexible model.[34] Only these intramolecular terms are used as a preconditioner. This produces a 9×9 block-diagonal (see Figure 2a) indefinite matrix, with approximately $n/3$ negative eigenvalues.[139] The local preconditioner of deoxycytidine, in contrast, is usually positive-definite with the possible exception of one isolated large negative eigenvalue.[139] Starting points for the water clusters of varying sizes are constructed so as to occupy a regular box in three dimensions (e.g., a $3 \times 3 \times 3$ cube for 27 molecules).[34] These arrangements are often poor approximations

Table 6 Water Cluster Minimization[a]

n	Method	Iterations (Outer/Inner)	NFG	CPU (in 24-s units)
18	TN, P	61/252	101	1.0
18	CG	448	899	1.0
18	BFGS	183	212	1.0
18	LM-BFGS, no P	603	655	1.4
18	LM-BFGS, P	204	222	1.0
72	TN, P	128/1417	223	3.3
72	CG	856	1715	2.5
72	BFGS	416	419	2.1
72	LM-BFGS, no P	1355	1408	2.5
72	LM-BFGS, P	904	934	4.9
243	TN, P	98/1723	184	37
243	CG	5742	11512	110
243	BFGS	1548	1669	76
243	LM-BFGS, no P	8066	8325	120
243	LM-BFGS, P	7017	7209	140

[a]Starting points were selected pseudorandomly so that the molecules occupy a regular three-dimensional box.[34] The starting energies and gradient norms for the dimer ($n = 18$), 8 molecules ($n = 72$), and 27 molecules ($n = 243$) are -5.5, 2.9×10; 4.8×10^2, 7.4×10^2; and 5.5×10^2, 7.5×10^2, respectively. The three corresponding minima are -6.95 for $n = 18$, between -73 and -76 for $n = 72$, and between -296 and -302 for $n = 243$. (The minimum obtained varies slightly from method to method in the higher dimensions.) The final gradient norms range from $O(10^{-6})$ to $O(10^{-4})$. A less strict convergence parameter, $\epsilon_g = 10^{-4}$, was used in all methods, and ϵ_f was also set to 10^{-4} for TN (see formulas [19], [21a–c]). The preconditioner here comes from the intramolecular bond length and bond angle terms and is a 9×9 block diagonal matrix. Five updates are used in LM-BFGS. TN used the quadratic truncation test with $c_q = 0.2$ (see condition [57]) and a limit of 25 PCG iterations per Newton step.

to the optimal hydrogen-bonded network geometries and become poorer as n increases, because the optimal shapes become more spherical.

These problem difficulties are reflected by an increased number of iterations, function evaluations, and time in all the minimization runs. For $n = 18$, performance of all methods is very similar in terms of time as most time is dominated by program printing and analysis. CG requires more iterations but is very fast. LM-BFGS is computationally efficient in terms of time per iteration, and preconditioning accelerates convergence. Dramatic acceleration by preconditioning is not noted here as in deoxycytidine, because the preconditioners are not as good approximations to the Hessian. Although these preconditioners generally produce better performance than the unpreconditioned version, the use of indefinite preconditioners is not typical and unclear as an optimal process.

For higher dimensions, we note that TN performs very well despite the relatively poor preconditioner and starting point. A limit on the maximum

PCG iterations per Newton iteration (25) reduces the total number of PCG iterations considerably without any sacrifice in performance. The quadratic truncation test of [57] was used here with $c_q = 0.2$. The modified Cholesky factorization for the preconditioner apparently produces good search vectors. CG requires more iterations but is relatively economical in terms of time. BFGS works well, but the limited-memory variants are not as efficient. Furthermore, preconditioning does not improve performance significantly and requires more time. These features are likely to reflect the ill conditioning of this problem. For these types of problems, performance of Newton methods may be very sensitive to the starting point and program parameters. Consequently, they may exhibit behavior that does not necessarily scale up systematically with size. CG methods reveal similar characteristics, but they may require less user information (i.e., are easier to implement). Typically, for such difficult problems, it is recommended that local minimization in combination with search strategies or stochastic approaches be used. Formulation of a better, positive-definite approximation to the Hessian may also improve performance, and the use of an automatic technique, among many, as offered in a recent FORTRAN package,[130] may be helpful. It should be emphasized that preconditioning is a very difficult topic as it is highly problem dependent. Thus, very little systematic research on tailoring has been done. Perhaps the emergence of powerful Newton variants will spur further developments in this area.

New Technologies

Future developments in the field of optimization will undoubtedly be influenced by recent interest and rapid developments in new technologies—powerful vector and parallel machines. Indeed, their exploitation for algorithm design and solution of "grand challenge" applications[15,16] is expected to bring new advances in the field of computational chemistry, in particular.

Supercomputers can provide speedup over traditional architectures by optimizing both scalar and vector computations. This can be accomplished by pipelining data as well as offering special hardware instructions for calculating intrinsic functions (e.g., $\exp(x)$, $\sqrt{x}$), arithmetic, and array operations. In addition, parallel computers can execute several operations concurrently. Multiple instructions can be specified for multiple data streams in MIMD designs, whereas the same instructions can be applied to multiple data streams in SIMD prototypes. Communication among processors is crucial for efficient algorithm design so that the full parallel apparatus is exploited. These issues will only increase in significance as massively parallel networks enter into regular use.

In general, one of the first steps in optimizing codes for these architectures is implementation of standard basic linear algebra subroutines (BLAS). These routines—continuously being improved, expanded, and adapted optimally to more machines—perform operations such as dot products ($\mathbf{x}^T\mathbf{y}$) and

vector manipulations ($ax + y$), as well as matrix/vector and matrix/matrix operations. Thus, operations, such as in Eq. [38] or [50b], can be executed very efficiently. In particular, if n is very large, segmentation among the processors may also be involved. A new library of FORTRAN 77 subroutines, LAPACK, focuses on design and implementation of standard numerical linear algebra tasks (e.g., systems of linear equations, eigenvalue and singular value problems) to achieve high efficiency and accuracy on vector processors, high-performance workstations, and shared-memory multiprocessors.[142] At this writing, up-to-date information may be obtained by sending the message "send index from lapack" to the electronic-mail address netlib@ornl.gov or contacting J. Dongarra and S. Ostrouchov at lapack@cs.utk.edu.

Specific strategies for optimization algorithms have been quite recent and are not yet unified.[44,54] For parallel computers, natural improvements may involve the following ideas: (1) performing multiple minimization procedures concurrently from different starting points; (2) evaluating function and derivatives concurrently at different points[136] (e.g., for a finite-difference approximation of gradient or Hessian or for an improved line search); (3) performing matrix operations or decompositions in parallel[118,133] for special structured systems (e.g., Cholesky factorizations of block-band preconditioners).

With increased computer storage and speed, the feasible methods for solution of very large (e.g., $O(10^5)$ or more variables) nonlinear optimization problems arising in important applications (macromolecular structure, meteorology, economics) will undoubtedly expand considerably and make possible new orders of resolution.

ACKNOWLEDGMENTS

I thank the editors of this volume for inviting my contribution. I am grateful to I. M. Navon, J. Nocedal, M. L. Overton, and R. B. Schnabel for providing many preprints and useful suggestions. I thank Connie Engle for her outstanding typing and sense of humor. The work is supported by the National Science Foundation (CHE-9002146 and Presidential Young Investigator Award ASC-9157582), the Searle Scholar Program, the Whitaker Foundation, and the Faculty of Arts and Science at New York University. Computer facilities were provided by the Academic Computing Facility at New York University.

REFERENCES

1. F. S. Acton, *Numerical Methods That Usually Work*, Chap. 17, Mathematical Association of America, Washington, D.C., 1990 (updated from the 1970 edition).
2. P. E. Gill, W. Murray, and M. H. Wright, *Numerical Linear Algebra on Optimization*, Vol. 1, Addison-Wesley, Redwood City, Calif., 1991.
3. D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd ed., Addison-Wesley, Reading, Mass., 1984.

4. R. Fletcher, *Practical Methods of Optimization,* 2nd ed., John Wiley & Sons, Tiptree, Essex, United Kingdom, 1987.
5. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,* Academic Press, New York, 1983.
6. J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Englewood Cliffs, N.J., 1983.
7. P. G. Ciarlet, *Introduction to Numerical Linear Algebra and Optimization,* Cambridge University Press, Cambridge, United Kingdom, 1989.
8. P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications,* D. Reidel, Dordrecht, 1987.
9. E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines,* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Tiptree, Essex, 1990.
10. G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, Eds., *Handbook in Operations Research Management Science,* Vol. 1, Elsevier Science/North-Holland, Amsterdam, 1989.
11. C. A. Floudas and P. M. Pardalos, Eds., *Recent Advances in Global Optimization,* Princeton Series in Computer Science, Princeton University Press, Princeton, N.J., 1991.
12. P. T. Boggs, R. H. Byrd, and R. B. Schnabel, Eds., *Numerical Optimization 1984,* SIAM, Philadelphia, 1985.
13. G. Dahlquist and Å Björk, *Numerical Methods,* Prentice-Hall, Englewood Cliffs, N.J., 1974.
14. G. H. Golub and C. F. Van Loan, *Matrix Computations,* Johns Hopkins University Press, Baltimore, Md., 1984.
15. *Grand Challenges: High Performance Computing and Communications,* Report by the Committee on Physical, Mathematical, and Engineering Sciences, Office of Science and Technology Policy, Washington, D.C., 1991.
16. *Mathematical Foundations of High-Performance Computing and Communications,* Board of Mathematical Sciences, National Research Council, National Academy Press, Washington, D.C., 1991.
17. R. B. Schnabel and T.-T. Chow, *SIAM J. Opt.,* **1,** 293 (1991). Tensor Methods for Unconstrained Optimization Using Second Derivatives.
18. U. Burkert and N. L. Allinger, *Molecular Mechanics,* ACS Monograph 177, American Chemical Society, Washington, D.C., 1987.
19. S. J. Weiner, P. A. Kollman, D. T. Nguyen, and D. A. Case, *J. Comput. Chem.,* **7,** 230 (1986). An All Atom Force Field for Simulations of Proteins and Nucleic Acids.
20. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comput. Chem.,* **4,** 187 (1983). CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations.
21. S. Lifson and A. Warshel, *J. Chem. Phys.,* **49,** 5116 (1968). Consistent Force Field for Calculations of Conformations, Vibrational Spectra, and Enthalpies of Cycloalkane and *n*-Alkane Molecules.
22. T. Schlick, dissertation, Courant Institute, Department of Mathematics, New York University (1987). Modeling and Minimization Techniques for Predicting Three-Dimensional Structures of Large Biological Molecules.
23. T. Schlick, B. E. Hingerty, C. S. Peskin, M. L. Overton, and S. Broyde, in *Theoretical Biochemistry and Molecular Biophysics,* Vol. I, pp. 39–58, D. L. Beveridge and R. Lavery, Eds., Adenine Press, Guilderland, New York, 1991. Search Strategies, Minimization Algorithms, and Molecular Dynamics Simulations for Exploring Conformational Spaces of Nucleic Acids.
24. L. Greengard and V. Rokhlin, *J. Comput. Phys.,* **73,** 325 (1987). A Fast Algorithm for Particle Simulations.
25. J. Carrier, L. Greengard, and V. Rokhlin, *SIAM J. Sci. Statist. Comput.,* **9,** 669 (1987). A Fast Adaptive Multipole Algorithm for Particle Simulations.

26. A. R. Leach, in *Reviews in Computational Chemistry,* Vol. 2, K. B. Lipkowitz and D. B. Boyd, Eds., VCH Publishers, New York, 1991. A Survey of Methods for Searching the Conformational Space of Small and Medium-Sized Molecules.
27. M. Pincus, R. Klausner, and H. A. Scheraga, *Proc. Natl. Acad. Sci. USA,* **79,** 5107 (1982). Calculation of the Three-Dimensional Structure of the Membrane Bound Portion of Milittin from Its Amino Acids.
28. H. A. Scheraga, *Biopolymers,* **22,** 1 (1983). Recent Progress in Theoretical Treatment of Protein Folding.
29. B. E. Hingerty, S. Figueroa, T. L. Hayden, and S. Broyde, *Biopolymers,* **28,** 1195 (1989). Prediction of DNA Structure from Sequence: A Buildup Technique.
30. J. A. McCammon and S. C. Harvey, *Dynamics of Proteins and Nucleic Acids,* Cambridge University Press, Cambridge, 1987.
31. T. Schlick, *Comput. Chem.,* **15,** 251 (1991). New Approaches to Potential Energy Minimization and Molecular Dynamics Algorithms.
32. C. S. Peskin and T. Schlick, *Comm. Pure Appl. Math.,* **42,** 1001 (1989). Molecular Dynamics by the Backward-Euler Method.
33. T. Schlick and C. S. Peskin, *Comm. Pure Appl. Math,* **42,** 1141 (1989). Can Classical Equations Simulate Quantum-Mechanical Behavior? A Molecular Dynamics Investigation of a Diatomic Molecule with a Morse Potential.
34. T. Schlick, S. Figueroa, and M. Mezei, *J. Chem. Phys.,* **94,** 2118 (1991). A Molecular Dynamics Simulation of a Water Droplet by the Implicit-Euler/Langevin Scheme.
35. A. Nyberg and T. Schlick, *J. Chem. Phys.,* **95,** 4986 (1991). A Computational Investigation of Dynamic Properties with the Implicit-Euler Scheme for Molecular Dynamics Simulation.
36. T. Schlick and W. K. Olson, *J. Mol. Biol.,* **223,** 1089 (1992). Computer Simulations of Supercoiled DNA Energies and Dynamics.
37. M. E. Tuckerman, B. J. Berne, and A. Rossi, *J. Chem. Phys.,* **94,** 1465 (1991). Molecular Dynamics Algorithm for Multiple Time Scales: Systems with Disparate Masses.
38. R. K. Z. Tan and S. C. Harvey, in *Theoretical Biochemistry and Molecular Biophysics,* Vol. I, pp. 125–137, D. L. Beveridge and R. Lavery, Eds., Adenine Press, Guilderland, New York, 1991. Succinct Macromolecular Models: Applications to Supercoiled DNA.
39. L. Piela, J. Kostrowicki, and H. A. Scheraga, *J. Phys. Chem.,* **93,** 3339 (1989). The Multiple-Minima Problem in Conformational Analysis of Molecules. Deformation of the Potential Energy Hypersurface by the Diffusion Equation Method.
40. E. O. Purisima and H. A. Scheraga, *Proc. Natl. Acad. Sci. USA,* **83,** 2782 (1986). An Approach to the Multiple-Minima Problem by Relaxing Dimensionality.
41. A. V. Levy and S. Gomez, in *Numerical Optimization 1984,* P. T. Boggs, R. H. Byrd, and R. B. Schnabel, Eds., pp. 213–244, SIAM, Philadelphia, 1985. The Tunneling Method Applied to Global Optimization.
42. A. H. G. Rinnooy Kan and G. T. Timmer, in *Handbooks in Operations Research and Management Science,* Vol. 1, G. L. Nemhauser, A.H.G. Rinnooy Kan, and M. J. Todd, Eds., Elsevier Science/North-Holland, Amsterdam, 1989. Global Optimization.
43. R. H. Byrd, E. Eskow, R. B. Schnabel, and S. L. Smith, *Parallel Global Optimization: Numerical Methods, Dynamic Scheduling Methods, and Application to Molecular Configuration,* Computer Science Report CU-CS-553-91, University of Colorado, Boulder, 1991.
44. R. B. Schnabel, in *Mathematical Programming,* M. Iri and K. Tanabe, Eds., pp. 227–261, Kluwer Academic, Dordrecht, 1989. Sequential and Parallel Methods for Unconstrained Optimization.
45. R. H. Byrd, C. L. Dert, A. H. G. Rinnooy Kan, and R. B. Schnabel, *Math, Prog.,* **46,** 1(1990). Concurrent Stochastic Methods for Global Optimization.
46. S. Smith, E. Eskow, and R. B. Schnabel, *Adaptive Asynchronous Stochastic Global Optimization Algorithms for Sequential and Parallel Computation,* Computer Science Report CV-CS-449-89. University of Colorado, Boulder, 1989.

47. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, *Science,* **220,** 671 (1983). Optimization by Simulated Annealing.

48. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.,* **21,** 1087 (1953). Equation of State Calculations by Fast Computing Machines.

49. A. Dekkers and E. Aarts, *Math Prog.,* **50,** 367 (1991). Global Optimization and Simulated Annealing.

50. I. O. Bohachevsky, M. E. Johnson, and M. L. Stein, *Technometrics,* **28,** 209 (1986). Generalized Simulated Annealing for Function Optimization.

51. M.-H. Hao and W. K. Olson, *Macromolecules,* **22,** 3292 (1989). Searching the Global Equilibrium Configurations of Supercoiled DNA by Simulated Annealing.

52. J. Moré and D. C. Sorenson, in *Studies in Numerical Analysis,* G. H. Golub, Ed., pp. 29–82, Mathematical Association of America, Washington, D.C., 1984. Newton's Method.

53. J. Moré, in *Mathematical Programming, The State of the Art,* A. Bachem, M. Grotschel, and G. Korte, Eds., pp. 256–287, Springer-Verlag, New York, 1983. Recent Developments in Algorithms and Software for Trust Region Methods.

54. J. E. Dennis, Jr. and R. B. Schnabel, in *Handbook in Operations Research and Mathematical Sciences,* Vol. 1, G. L. Nemhauser, A.H.G. Rinnooy Kan, and M. J. Todd, et al., pp. 1–72, Eds., Elsevier Science/North-Holland, Amsterdam, 1989. A View of Unconstrained Optimization.

55. J. Moré and D. J. Thuente, *Mathematics and Computer Science, Division Preprint MCS-P153-0590,* Argonne National Laboratory, Argonne, Ill., 1990. On Line Search Algorithms with Guaranteed Sufficient Decrease.

56. W. C. Davidon, *Variable Metric Method for Minimization,* Report ANL-5990 (rev.), Argonne National Laboratory, Argonne, Ill., 1959.

57. M. J. D. Powell, *Comput. J.,* **7,** 155 (1964). An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives.

58. P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *SIAM J. Sci. Statist. Comput.,* **4,** 310 (1983). Computing Forward-Difference Intervals for Numerical Optimization.

59. L. B. Rall, *Automatic Differentiation—Techniques and Applications,* Lecture Notes in Computer Science 120, Springer-Verlag, Berlin/New York, 1981.

60. A. Griewank, in *Mathematical Programming 1988,* pp. 83–107, Kluwer Academic, Tokyo, 1988. On Automatic Differentiation.

61. L. C. W. Dixon, *SIAM J. Opt.,* **1,** 475 (1991). On the Impact of Automatic Differentiation on the Relative Performance of Parallel Truncated Newton and Variable Metric Algorithms.

62. B. E. Hingerty and S. Broyde, *Biopolymers,* **24,** 2279 (1985). Carcinogen-Base Stacking and Base-Base Stacking in dCpdG Modified by (+) and (−) Anti BPDE.

63. A. Cauchy, *Comp. Rend. Acad. Sci. Paris,* 536 (1847). Méthode Générale pour las Résolution des Systèmes d'Equations Simultanées.

64. M. R. Hestenes and E. Stiefel, *J. Res. Natl. Bur. Stand.,* **49,** 409 (1952). Methods of Conjugate Gradients for Solving Linear Systems.

65. M. R. Hestenes, *Conjugate Direction Methods in Optimization,* Springer-Verlag, New York, 1980.

66. R. Fletcher and C. M. Reeves, *Comput. J.,* **7,** 149 (1964). Function Minimization by Conjugate Gradients.

67. M. J. D. Powell, in *Lecture Notes in Mathematics,* Vol. 1066, pp. 122–141, Springer, Berlin/New York, 1984. Nonconvex Minimization Calculations and the Conjugate Gradient Method.

68. M. J. D. Powell, *Math, Prog.,* **12,** 241 (1977). Restart Procedures of the Conjugate Gradient Method.

69. M. J. D. Powell, *Math. Programming,* **11,** 42 (1976). Some Convergence Properties of the Conjugate Gradient Method.

70. D. F. Shanno, *Math Oper. Res.*, **3**, 244 (1978). Conjugate Gradient Methods with Inexact Searches.

71. E. Polak and G. Ribière, *Rev. Fr. Inform. Rech. Oper.*, **16**, 35 (1969). Note sur la Convergence de Méthodes de Directions Conjuguées.

72. J. Stoer, *Theory Numer. Math.*, **28**, 343 (1977). On the Relation between Quadratic Termination and Convergence Properties of Minimization Algorithms, Part I.

73. H. P. Crowder and P. Wolfe, *IBM J. Res. Develop.*, **16**, 431 (1972). Linear Convergence of the Conjugate Gradient Method.

74. A. Cohen, *SIAM J. Numer. Anal.*, **9**, 248 (1972). Rate of Convergence of Several Conjugate Gradient Algorithms.

75. P. Baptist and J. Stoer, *Numer. Math.*, **28**, 367 (1977). On the Relation between Quadratic Termination and Convergence Properties of Minimization Algorithms.

76. M. Al-Baali, *Inst. Math. Appl. J. Numer. Anal.*, **5**, 121 (1985). Descent Property and Global Convergence of the Fletcher-Reeves Method with Inexact Linear Search.

77. J. C. Gilbert and J. Nocedal, *SIAM J. Opt.*, **2**, 21 (1992). Global Convergence Properties of Conjugate Gradient Methods for Optimization.

78. Y. F. Hu and C. Storey, *J. Opt. Theor. Appl.*, **71**, 399 (1991). Global Convergence Result for Conjugate Gradient Methods.

79. O. Axelsson and G. Lindskog, *Numer. Math.*, **48**, 449 (1989). On the Rate of Convergence of the Preconditioned Conjugate Gradient Method.

80. J. D. Evans, *J. Inst. Math. Appl.*, **4**, 295 (1967). The Use of Pre-conditioning in Iterative Methods for Solving Linear Equations with Symmetric Positive Definite Matrices.

81. P. Concus, G. H. Golub, and D. P. O'Leary, in *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, Eds., Academic Press, New York, 1976, pp. 309–332. A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations.

82. T. Schlick and M. Overton, *J. Comput. Chem.*, **8**, 1025 (1987). A Powerful Truncated Newton Method for Potential Energy Minimization.

83. D. F. Shanno and K. H. Phua, *ACM Trans. Math. Software*, **6**, 618 (1980). Remark on Algorithm 500: Minimization of Unconstrained Multivariate Functions.

84. R. A. Scott and H. A. Scheraga, *J. Chem. Phys.*, **42**, 2209 (1965). Method for Calculating Internal Rotation Barriers.

85. R. A. Scott and H. A. Scheraga, *J. Chem. Phys.*, **44**, 3054 (1966). Conformational Analysis of Macromolecules. II. The Rotational Isomeric States of the Normal Hydrocarbons.

86. M. Levitt, *J. Mol. Biol.*, **168**, 595 (1983). Molecular Dynamics of Native Protein: I. Computer Simulation of Trajectories.

87. B. Lesyng and W. Saenger, *Carbohydrate Res.*, **133**, 187 (1984). Influence of the Orientation of Hydroxyl Groups on the Puckering Modes of Furanoid Rings.

88. C.-S. Tung, S. C. Harvey and J. A. McCammon, *Biopolymers*, **23**, 2173 (1984). Large-Amplitude Bending Motions in Phenylalanine Transfer RNA.

89. A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

90. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

91. Ph. L. Toint, in *Sparse Matrices and Their Uses*, I. S. Duff, Ed., Academic Press, New York, 1981. Towards an Efficient Sparsity Exploiting Newton Method for Minimization.

92. A. R. Curtis, M. J. D. Powell, and J. K. Reid, *J. Inst. Math.*, **13**, 117 (1974). On the Estimation of Sparse Jacobian Matrices.

93. M. J. D. Powell and Ph. L. Toint, *SIAM J. Numer. Anal.*, **16**, 1060 (1979). On the Estimation of Sparse Hessian Matrices.

94. D. P. O'Leary, *Math. Prog.*, **23**, 20 (1982). A Discrete Newton Algorithm for Minimizing a Function of Many Variables.

95. J. E. Dennis, Jr. and J. J. Moré, *SIAM Rev.*, **19**, 46 (1977). Quasi-Newton Methods, Motivation and Theory.

96. P. E. Gill and W. Murray, *J. Inst. Math. Appl.*, **9**, 91 (1972). Quasi-Newton Methods for Unconstrained Optimization.

97. A. Griewank and Ph. L. Toint, in *Nonlinear Optimization 1981*, M. J. D. Powell, Ed., pp. 301–312, Academic Press, New York, 1982. On the Unconstrained Optimization of Partially Separable Functions.

98. A. Griewank and Ph. L. Toint, *Numer. Math.*, **39**, 119 (1982). Partitioned Variable Metric Updates for Large Structured Optimization Problems.

99. D. F. Shanno and P. C. Kettler, *Math. Comput.*, **24**, 657 (1970). Optimal Conditioning of Quasi-Newton Methods.

100. J. Nocedal, *Math. Comput.*, **35**, 773 (1980). Updating Quasi-Newton Matrices with Limited Storage.

101. D. C. Liu and J. Nocedal, *Math. Prog.*, **45**, 503 (1989). On the Limited Memory BFGS Method for Large Scale Optimization.

102. J. C. Gilbert and C. Lemaréchal, *Math. Prog.*, **45**, 407 (1989). Some Numerical Experiments with Variable-Storage Quasi-Newton Algorithms.

103. A. Buckley and A. LeNir, *Math. Prog.*, **27**, 155 (1983). QN-like Variable Storage Conjugate Gradients.

104. M. J. D. Powell, *Math. Prog.*, **34**, 34 (1986). How Bad are the BFGS and DFP Methods When the Objective Function Is Quadratic?

105. R. H. Byrd, J. Nocedal, and Y. Yuan, *SIAM J. Numer. Anal.*, **24**, 1171 (1987). Global Convergence of a Class of Quasi-Newton Methods on Convex Problems.

106. A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *SIAM J. Numer. Anal.*, **25**, 433 (1988). Global Convergence of a Class of Trust Region Algorithms for Optimization with Simple Bounds.

107. A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *Math. Comput.*, **50**, 399 (1988). Testing a Class of Methods for Solving Minimization Problems with Simple Bounds on the Variables.

108. A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *Math. Prog.*, **2**, 177 (1991). Convergence of Quasi-Newton Matrices Generated by the Symmetric Rank One Update.

109. S. G. Nash and J. Nocedal, *SIAM J. Opt.*, **1** 358–372 (1991). A Numerical Study of the Limited Memory BFGS Method and the Truncated-Newton Method for Large-Scale Optimization.

110. X. Zou, I. M. Navon, F. X. Le Dimet, A. Nouailler, and T. Schlick, *SIAM J. Opt.*, in press. A Comparison of Efficient Large-Scale Minimization Algorithms for Optimal Control Applications in Meteorology.

111. S. C. Eisenstat and H. F. Walker, preprint, 1992. Globally Convergent Inexact Newton Methods.

112. R. S. Dembo, in *Nonlinear Optimization*, M. J. D. Powell, Ed., pp. 361–373. Academic Press, New York, 1982. Large Scale Nonlinear Optimization.

113. R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *SIAM J. Numer. Anal.*, **19**, 400 (1982). Inexact Newton Methods.

114. R. S. Dembo and T. Steihaug, *Math. Prog.*, **26**, 190 (1983). Truncated-Newton Algorithms for Large-Scale Unconstrained Optimization.

115. S. G. Nash, in *Numerical Optimization 1984*, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, Eds., pp. 119–136, SIAM, Philadelphia, 1985. Solving Nonlinear Programming Problems Using Truncated-Newton Techniques.

116. S. G. Nash and A. Sofer, *Oper. Res. Lett.*, **9**, 219 (1990). Assessing a Search Direction Within a Truncated Newton Method.

117. S. G. Nash, *SIAM J. Sci. Statist. Comput.*, **6**, 599 (1985). Preconditioning of Truncated-Newton Methods.

118. S. A. Zenios and M. C. Pinar, *SIAM J. Sci. Statist. Comput.*, **13** (Sept. 1992). Parallel Block-Partitioning of Truncated Newton for Nonlinear Network Optimization.

119. P. Deuflhard, in *Proceedings of the Copper Mountain Conference on Iterative Methods, Copper Mountain, Colorado, April 1–5, 1990.* Global Inexact Newton Methods for Very Large Scale Nonlinear Problems.

120. J. W. Ponder and F. M. Richards, *J. Comput. Chem.*, **8**, 1016 (1987). An Efficient Newton-like Method for Molecular Mechanics Energy Minimization of Large Molecules.

121. T. Schlick and A. Fogelson, *ACM Trans. Math. Software*, **18**, 46 (1992). TNPACK—A Truncated Newton Minimization Package for Large-Scale Problems. I. Algorithm and Usage.

122. T. Schlick and A. Fogelson, *ACM Trans. Math. Software*, **18**, 71 (1992). TNPACK—A Truncated Newton Minimization Package for Large-Scale Problems. II. Implementation Examples.

123. L. Fauci and A. Fogelson, *Comm. Pure Appl. Math.*, in press. Truncated Newton Methods and the Modeling of Immersed Elastic Structures.

124. H. Matthies and G. Strang, *Int. J. Numer. Methods Engin.*, **14**, 1613 (1979). The Solution of Nonlinear Finite Element Equations.

125. S. C. Eisenstat, M. H. Shultz, and A. H. Sherman, in *Advances in Computer Methods for Partial Differential Equations*, R. Vichnevetsky, Ed., pp. 33–39, AICA, New Brunswick, N.J., 1975. Efficient Implementation of Sparse Symmetric Gaussian Elimination.

126. S. C. Eisenstat, M. H. Shultz, and A. H. Sherman, in *Advances in Computer Methods for Partial Differential Equations*, R. Vichnevetsky, Ed., pp. 40–45, AICA, New Brunswick, N.J., 1975. Application of Sparse Matrix Methods to Partial Differential Equations.

127. S. C. Eisenstat, M. H. Schultz, and A. H. Sherman, *SIAM J. Sci. Statist. Comput.*, **2**, 225 (1981). Algorithms and Data Structures for Sparse Symmetric Gaussian Elimination.

128. S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *Int. J. Numer. Methods Engin.*, **18**, 1145 (1982). Yale Sparse Matrix Package. I. The Symmetric Codes.

129. D. J. Rose, in *Graph Theory and Computing*, R. C. Read, Ed., Academic Press, New York, 1972. A Graph-Theoretic Study of the Numerical Solution of Sparse Positive-Definite Systems of Linear Equations.

130. T. C. Oppe, W. D. Joubert, and D. R. Kincaid, *NSPCG User's Guide. Version 1.0: A Package for Solving Large Sparse Linear Systems by Various Iterative Methods.* Technical Report CNA-216, Center for Numerical Analysis, University of Texas at Austin, 1988.

131. Y. Saad, *SPARSEKIT: A Basic Toolkit for Sparse Matrix Computations*, NASA Ames Research Center, Moffett Field, Calif., May 21, 1990.

132. J. W. H. Liu, *ACM Trans. Math. Software*, **11**, 141 (1985). Modification of the Minimum-Degree Algorithm by Multiple Elimination.

133. A. George, M. T. Heath, J. Liu, and E. Ng, *SIAM J. Statist. Comput.*, **9**, 327 (1988). Sparse Cholesky Factorization on a Local-Memory Multiprocessor.

134. J. M. Ortega, *SIAM J. Opt.*, **1**, 565 (1991). Orderings for Conjugate Gradient Preconditioners.

135. J. O'Neil and D. B. Szyld, *SIAM J. Sci. Statist. Comput.*, **11**, 811 (1990). A Block Ordering Method for Sparse Matrices.

136. S. G. Nash and A. Sofer, *SIAM J. Opt.*, **1**, 530 (1991). A General-Purpose Parallel Algorithm for Unconstrained Optimization.

137. R. B. Schnabel and E. Eskow, *SIAM J. Sci. Statist. Comput.*, **11**, 1136 (1990). A New Modified Cholesky Factorization.

138. E. Eskow and R. B. Schnabel, *Software for a New Modified Cholesky Factorization.* Computer Science Department Technical Report CU-CS-443-89, University of Colorado, Boulder, 1989.

139. T. Schlick, *SIAM J. Sci. Statist. Comput.,* in press. Modified Cholesky Factorizations for Sparse Preconditioners.

140. J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *ACM Trans. Math. Software,* 7, 17 (1981). Testing Unconstrained Optimization Software.

141. J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *ACM Trans. Math. Software,* 7, 136 (1981). Algorithm 566: FORTRAN Subroutines for Testing Unconstrained Optimization Software.

142. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide,* Society for Industrial and Applied Mathematics, Philadelphia, 1992.