March 23, 1997

GEOMETRY OPTIMIZATION*

Tamar Schlick

The Howard Hughes Medical Institute and New York University, New York, NY

KEY WORDS: automatic differentiation, conjugate gradient, descent algorithm, large-scale optimization, line search, Newton's method, nonlinear optimization, quasi-Newton, steepest descent, truncated-Newton, trust-region approach

Author Information:

Tamar Schlick
Department of Chemistry and the Courant Institute
of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012
Phone: (212) 998-3116 / (212) 998-3132
Fax: (212) 995-4152
E-mail: schlick@nyu.edu

ABBREVIATIONS USED:

- CG — Conjugate Gradient
- CPU — Central Processing Units
- QN — Quasi Newton
- SD — Steepest Descent
- TN — Truncated Newton

*A contributed article to the **Encyclopedia of Computational Chemistry**, P. von Ragué Schleyer, Editor-in-Chief, and N. L. Allinger, T. Clark, J. Gasteiger, P. A. Kollman, and H. F. Schaefer III, Editors, John Wiley & Sons, 1997.

Contents

List of Figures

List of Tables

Synopsis

Techniques for local geometry optimization relevant to biological macromolecules governed by empirical potential energy surfaces are reviewed, with a focus on methods for large-scale nonlinear optimization problems. Practical recommendations for optimization applications in chemistry are given, with simple examples used to illustrate the performance of the Newton-Raphson-Simpson-Fourier method. Comparisons among three CHARMM minimizers on two molecular models illustrate behavior for more complex problems.

1 Introduction

Geometry optimization is a fundamental component of molecular modeling. The determination of a low-energy conformation for a given **force field** can be the final objective of the computation. Alternatively, the minimum for the system on the specified potential energy surface, in a local or global sense, can serve as a starting or reference point for subsequent calculations. These may be simulations — such as **molecular dynamics** and **free-energy perturbations** — or analyses such as of normal modes (see **conformational analysis** entry).

Already, finding a local minimum (i.e., one near the input point) is a challenging task for a large biological system governed by a nonlinear potential energy function. This is because the optimization scheme must find a minimum from any point along the potential surface, even one associated with a very high-energy, and should not get trapped at local maxima or saddle points. Finite-precision arithmetic and other errors that accumulate over many operations also degrade practical performance in comparison to theoretical expectations. Nonetheless, this local optimization problem is solved in a mathematical sense: convergence to a nearby minimum can be achieved on modern computers; in the mathematical literature, this is referred to as *global convergence* to a local minimum. Indeed, many algorithms have been implemented by numerical analysts and application specialists for this purpose and are also available in widely used molecular mechanics and dynamics packages. Still, their performance and solution quality vary considerably and depend greatly on the user-specified algorithmic convergence parameters and the starting point.

The global optimization problem, in contrast, remains unsolved in general. This is because the exponentially-growing number of minima with system size cannot be exhaustively surveyed. Certainly, effective strategies have been developed in specific application contexts (e.g., for polypeptides) and work well for moderately-sized systems. These global algorithms differ from the local schemes in that they allow the energy to increase throughout the search (in addition to a decrease), making possible escape from local potential wells and entry into others (see the **Monte Carlo** and **Simulated Annealing** entries). These global optimizers can be stochastic or deterministic, or a combination of these, and they sometimes rely on local optimization components.

Both local and global optimization problems lie at the heart of numerous scientific and engineering problems — from the biological and chemical disciplines to architectural and industrial design to economics. Optimization is part of our every day life — responsible for our weather forecasts, flight planning, telephone routing, or the functioning of enzymes in our bodies. The mathematical techniques developed to address these optimization-formulated problems are just as robust and varied as the problems themselves. Their algorithmic complexity has led to many available computer programs that require minimal output from the user (e.g., the starting point and a routine for function evaluation). However, a careful user of these canned software modules — even within standard molecular mechanics and dynamics packages — should understand the fundamental structure of the optimization algorithms and associated performance issues to make their usage both efficient and correct,

in terms of the chemical interpretations. This entry introduces key optimization concepts for this purpose, highlighting the fundamentals of local optimizers for *large-scale nonlinear unconstrained problems*, an important optimization subfield relevant to molecular applications. The most promising approaches among them are discussed, and practical issues, such as parameter variations and termination criteria, are mentioned. Of course, the latter are best learned by experimentation in the context of real problems. The reader is referred to classic texts [1, 2, 3, 4] and some reviews [5, 6, 7] for further details and for other categories of the rich and exciting field of optimization. See also the closely-related entries of **conformational search** and **structure elucidation**.

2 Basic Definitions of Optimization Problems

The methods for solving an optimization task depend on the problem classification. Since the *maximum* of a function f is the *minimum* of the function $-f$, it suffices to deal with minimization. The optimization problem is classified according to the type of independent variables involved (real, integer, mixed), the number of variables (one, few, many), the functional characteristics (linear, least squares, nonlinear, nondifferentiable, separable, etc.), and the problem statment (unconstrained, subject to equality constraints, subject to simple bounds, linearly constrained, nonlinearly constrained, etc.). For each category, suitable algorithms exist that exploit the problem's structure and formulation.

2.1 Problem Formulation

For a vector $\mathbf{x}$ of n components $\{x_i\}$, we write the minimization problem as:

$$\min_{\mathbf{x}} \{f(\mathbf{x})\} , \quad \mathbf{x} \in \mathcal{D}, \tag{1}$$

where f is the objective function and $\mathcal{D}$ is a feasible problem domain for the independent variables. The problem can be subject to m constraints, which can be written generally as a combination of equality and inequality constraints:

$$\begin{aligned} c_i(\mathbf{x}) &= 0 \quad \text{for } i = 1, \ldots, m', \\ c_i(\mathbf{x}) &\leq 0 \quad \text{for } i = m'+1, \ldots, m. \end{aligned} \tag{2}$$

Note that special bound constraints in the form $c_i(\mathbf{x}) = x_i$, where x_i is the ith component of the vector $\mathbf{x}$, or two-sided constraints such as $l_i \leq c_i(\mathbf{x}) \leq u_i$, can be reduced to the above form.

2.2 Independent Variables

In most computational chemistry problems, $\mathbf{x}$ is a real vector in Euclidean space, i.e., $\mathbf{x} \in \Re^n$, and f defines a transformation to a real number, i.e., $f(\mathbf{x}) : \Re^n \rightarrow \Re$. When the components

of $\mathbf{x}$ are integers, the optimization problem is classified as *integer-programming*. When $\mathbf{x}$ is a mixture of real and integer variables, the problem is of *mixed-integer programming* type. Common examples of integer-programming are network optimization and the traveling salesman problem, also classified as combinatorial optimization.

2.3 Function Characteristics

The nature of the function f is the next step in problem classification. Many application areas such as finance and management-planning tackle *linear* or *quadratic* objective functions. These can be written in vector form, respectively, as $f(\mathbf{x}) = \mathbf{b}^{\mathrm{T}}\mathbf{x} + f_0$ and $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + f_0$ where $\mathbf{b}$ is a column vector, f_0 is a scalar, and $\mathbf{A}$ is a constant *symmetric* matrix (i.e., one whose entries satisfy $A_{i,j} = A_{j,i}$) of dimension $n \times n$. The superscripts $^{\mathrm{T}}$ above refer to a vector transpose; thus $\mathbf{x}^{\mathrm{T}}\mathbf{y}$ is an inner product. *Linear programming* problems refer to linear objective functions subject to linear constraints (i.e., a system of linear equations), and *quadratic programming* problems have quadratic objective functions and linear constraints.

Nonlinear functions can be classified further if they are of *least-square* type (i.e., $f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i(\mathbf{x})^2$) or *separable* when f is a sum of subfunctions ($f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$), each dependent only on a subset of the independent variables (i.e., there are many x_j for each f_i for which $f_i(\mathbf{x} + x_j) = f_i(\mathbf{x})$).

The objective function may be nondifferentiable or possess continuous derivatives up to a given order. Since most optimization algorithms exploit derivative information to locate optima, nondifferentiable functions pose special difficulties, and very different algorithmic approaches must be used.

Geometry optimization problems for molecules in the context of standard all-atom force fields in computational chemistry are typically of the multivariate, continuous, and nonlinear type. They can be formulated as constrained (as in adiabatic relaxation) or unconstrained. Discontinuities in the derivatives may be a problem in certain formulations involving truncation, such as of the nonbonded terms (see Section 7). The large number of independent variables for biomolecules, in particular, warrants their classification as *large-scale* and rules out the use of many algorithms that are effective for a small number of variables. However, as will be discussed, effective techniques are available today that achieve rapid convergence even for large systems. In practice, these optimization algorithms must be modest in storage requirements for macromolecular applications and economical in computations, which are dominated by the function and derivative evaluations.

3 Optimization Fundamentals

3.1 Local and Global Minima

The *local* unconstrained optimization problem in the Euclidean space $\Re^n$ can be stated as in equation (1) for $\mathbf{x} \in \mathcal{D} \subset \Re^n$ where $\mathcal{D}$ is a region in the neighborhood of the starting point, $\mathbf{x}_0$. The *global* optimization problem requires $\mathcal{D}$ be the entire feasible space.

A (strong) *local minimum* $\mathbf{x}^*$ of $f(\mathbf{x})$ satisfies

$$f(\mathbf{x}^*) \ < \ f(\mathbf{y}) \qquad \text{for all } \mathbf{y} \in \mathcal{D}, \ \ \mathbf{y} \neq \mathbf{x}^*. \tag{3}$$

The point $\mathbf{x}^*$ is a *weak local minimum* if equality holds for all $\mathbf{y} \in \mathcal{D}$. A *global minimum* $\mathbf{x}^*$ satisfies the stringent requirement that

$$f(\mathbf{x}^*) \ < \ f(\mathbf{y}) \qquad \text{for all } \mathbf{y} \neq \mathbf{x}^*. \tag{4}$$

See Figure 1 for an illustration of a one-dimensional function with several minima. Figure 1

3.2 Derivatives of Multivariate Functions

When f is a smooth function with continuous first and second derivatives, we define its *gradient vector* of first derivatives by $\mathbf{g}(\mathbf{x})$, where each component of $\mathbf{g}$ is

$$g_i(\mathbf{x}) \ = \ \partial f(\mathbf{x}) / \partial x_i. \tag{5}$$

The $n \times n$ symmetric matrix of second derivatives, $\mathbf{H}(\mathbf{x})$, is called the *Hessian*. Its components are defined as:

$$H_{i,j}(\mathbf{x}) \ = \ \partial^2 f(\mathbf{x}) / \partial x_i \partial x_j \, . \tag{6}$$

At a *stationary* point, the gradient is zero. At a minimum point $\mathbf{x}^*$, in addition to stationarity, the curvature is positive. This curvature generalization to higher dimensions of positive second derivatives (for a univariate function) is known as *positive-definiteness* of the Hessian. This convexity of the multivariate nonlinear function at $\mathbf{x}^*$ can be written as

$$\mathbf{y}^{\mathrm{T}} \, \mathbf{H}(\mathbf{x}^*) \, \mathbf{y} \ > \ 0 \qquad \text{for all nonzero } \mathbf{y}. \tag{7}$$

In particular, positive definiteness guarantees that all the eigenvalues are positive at $\mathbf{x}^*$. A *positive semi-definite* matrix has nonnegative eigenvalues, and a *negative-definite* matrix has only negative eigenvalues; otherwise, the matrix is *indefinite*. Since indefinite Hessians signal the existence of both minima and maxima, utilization of curvature information is important for formulating effective multivariate optimization algorithms. Figure 2 illustrates this notion of curvature for quadratic functions ($\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x}$) of two variables by displaying the contours of these functions (curves on which the function is constant) in four cases for $\mathbf{A}$: (a) indefinite, (b) positive definite, (c) negative definite, and (d) singular (i.e., not invertible). Figure 2

3.3 The Hessian of Potential Energy Functions

A matrix is termed *sparse* if it has a large percentage of zero entries; otherwise it is *dense*. A sparse matrix can be *structured* (as in a banded matrix where there are zeros for $|i-j| > p$) or *unstructured*, as shown in Figure 3a. This figure shows the percentage of nonzero elements (13%) resulting from the nonzero second-derivative terms of the potential energy function for the protein BPTI (bovine pancreatic trypsin inhibitor) when an 8 Å cutoff is used for the nonbonded terms. Figure 3b shows the sparsity pattern for the matrix corresponding only to the bonded second-derivative terms for BPTI (bond-length, bond angle, and dihedral-angle). Here the sparsity is less than 2%. Note from the insets, which zoom on two submatrices, how the sparsity pattern repeats in triplets (for the x, y, and z components) and how nearly banded the matrix structure is. Figure 3

Since formulation of a dense Hessian (n^2 entries, of which $n(n+1)/2$ are unique) is both memory and computation intensive, many Newton techniques for minimization approximate curvature information implicitly and often progressively (i.e., as the algorithm proceeds). Limited-memory versions reduce computational and storage requirements so that they can be applied to very large problems and/or to problems where second derivatives are not available.

In most molecular mechanics packages, the second derivatives are programmed, though sparsity (when relevant) is not often exploited in the storage techniques for large molecular systems. The optimizer should utilize some of this second-derivative information to make the algorithm more efficient. Truncated Newton methods, for example, are designed with this philosophy.

4 Basic Algorithmic Components

The basic structure of an iterative local optimization algorithm is one of "greedy descent". It is based on one of the following two algorithmic frameworks: line-search or trust-region methods. Both are found throughout the literature and in software packages and are essential components of effective descent schemes that guarantee convergence to a local minimum from any starting point. No clear evidence has emerged to render one class superior over another.

The initial guess for the iterative minimization process can be derived from experimental data, where available, or from results of **conformational search** techniques.

4.1 Line-Search-Based Descent Algorithm

Algorithm [A1]: Basic Descent Using Line Search

From a given point $\mathbf{x}_0$, perform for $k = 0, 1, 2, \ldots$ *until convergence*:

1. Test $\mathbf{x}_k$ for convergence ($\| \mathbf{g}(\mathbf{x}_k) \| \leq \epsilon_g [1 + f(\mathbf{x}_k)]$, for example)

2. Calculate a *descent* direction $\mathbf{p}_k$ (method dependent)

3. Determine a steplength λ_k by a one-dimensional line search so that the new position vector, $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{p}_k$, satisfies:

$$f(\mathbf{x}_{k+1}) \;\leq\; f(\mathbf{x}_k) + \alpha\,\lambda\,\mathbf{g}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{p}_k \qquad [\text{"sufficient decrease"}] \tag{8}$$

and

$$|\mathbf{g}(\mathbf{x}_{k+1})^{\mathrm{T}}\mathbf{p}_k| \;\leq\; \beta\,|\mathbf{g}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{p}_k| \qquad [\text{"sufficient directional derivative reduction"}] \tag{9}$$

where $0 < \alpha < \beta < 1$ (typically, $\alpha = 10^{-4}$ and $\beta = 0.9$)

4. Set $\mathbf{x}_{k+1}$ to $\mathbf{x}_k + \lambda_k \mathbf{p}_k$ and k to $k+1$ and go to step 1

4.1.1 Descent Direction

A *descent direction* $\mathbf{p}_k$ is one along which the function can decrease. Formally, we define such a vector as one for which the directional derivative is negative:

$$\mathbf{g}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{p}_k \;<\; 0\;. \tag{10}$$

To see why this property implies that f can be reduced, approximate the nonlinear objective function f at $\mathbf{x}$ by a linear model along the descent direction $\mathbf{p}$, assuming that higher-order terms are smaller than the gradient term. Then we see that the difference in function values is negative:

$$\begin{aligned} f(\mathbf{x}+\lambda\mathbf{p}) - f(\mathbf{x}) \quad &= \quad \lambda\,\mathbf{g}(\mathbf{x})^{\mathrm{T}}\mathbf{p} \;+ \frac{\lambda^2}{2}\,\mathbf{p}^{\mathrm{T}}\mathbf{H}(\mathbf{x})\,\mathbf{p} \\ &\approx \quad \lambda\,\mathbf{g}(\mathbf{x})^{\mathrm{T}}\mathbf{p} \;< 0\,, \end{aligned} \tag{11}$$

for sufficiently small positive λ.

The descent condition is used to define the algorithmic sequence that generates $\mathbf{p}_k$ but is not always tested in practice. In reality, numerical errors can lead to departure from theoretical expectations. Thus, it is often necessary to check explicitly for the descent property of $\mathbf{p}_k$ from equation (10), especially in nonlinear conjugate gradient methods which are very sensitive to roundoff.

4.1.2 The One-Dimensional Optimization Subproblem

The line search procedure in step 3 of Algorithm [A1] is an approximate univariate minimization problem. It is typically performed via quadratic or cubic polynomial interpolation of the one-dimensional function of λ: $\phi(\lambda) \equiv f(\mathbf{x}_k + \lambda\mathbf{p}_k)$. For the polynomial interpolant of $\phi(\lambda)$, $p(\lambda)$, the known values of $\mathbf{x}_k$, $\mathbf{x}_{k+1}$, $\mathbf{g}(\mathbf{x}_k)$, and possibly $\mathbf{g}(\mathbf{x}_k + \lambda\,\mathbf{p}_k)$ are used at

each λ step. The solution to $p'(\lambda) = 0$ then gives an optimal λ value. However, bracketing strategies are also necessary to ensure that the minimum of ϕ is located within the feasible region $[\mathbf{x}_k,\ \mathbf{x}_k + \lambda\,\mathbf{p}_k]$. Typically, the initial trial value for λ is one. Thus, the line search can be considered as a backtracking algorithm for λ in the interval $(0,\ 1]$.

The line search criteria above are formulated to ensure sufficient decrease of f relative to the size of step (λ) taken. The first condition (equation (8)) prescribes an upper limit on acceptable new function values (recall that the second term on the right is negative by the descent property). The second criterion (equation (9)) imposes a lower bound on λ. The controlling parameters α and β determine the balance between the computational work performed in the line search and the reduction in function achieved. Since each line-search iteration requires a new function (and possibly gradient) evaluation to define the new interpolant $p(\lambda)$, most algorithms specify a very small α and a β close to unity; with this combination, the first condition is easily satisfied (assuming that the search vector $\mathbf{p}_k$ produced by the algorithm is well-scaled), and the second condition is very lenient and typically satisfied when the first is. The combination of criteria, however, is important for avoiding difficult situations when divergence of the process or cycling of the iterate can result.

The line search and the procedure that defines $\mathbf{p}_k$ form the central components of the basic descent local optimization algorithm above. The work in the line search (number of polynomial interpolations) should be balanced with the overall progress realized in the minimization algorithm.

4.2 Trust-Region-Based Descent Algorithm

Algorithm [A2]: Basic Descent Using A Trust Region Subsearch

From a given point $\mathbf{x}_0$, perform for $k = 0, 1, 2, \ldots$ *until convergence*:

1. Test $\mathbf{x}_k$ for convergence
2. Calculate a step $\mathbf{s}_k$ by solving the subproblem

$$\min_{\mathbf{s}} \{q_k(\mathbf{s})\}\ , \tag{12}$$

where q_k is the quadratic model of the objective function:

$$q_k(\mathbf{s}) \;=\; f(\mathbf{x}_k) \;+\; \mathbf{g}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{s} \;+\; \frac{1}{2}\mathbf{s}^{\mathrm{T}}\,\mathbf{H}\,(\mathbf{x}_k)\,\mathbf{s}\,, \tag{13}$$

subject to a size bound, Δ_k (a positive value), on $\mathbf{s}$. This bound involves a scaling matrix, $\mathbf{D}_k$, and requires

$$\|\ \mathbf{D}_k\mathbf{s}\ \| < \Delta_k\,, \tag{14}$$

where $\|\cdot\|$ denotes the standard Euclidean norm

3. Set $\mathbf{x}_{k+1}$ to $\mathbf{x}_k + \mathbf{s}_k$ and k to $k+1$ and go to step 1

The idea in trust-region methods — the origin of the quadratic optimization subproblem in step 2 above — is to determine the vector $\mathbf{s}_k$ on the basis of the size of region within which the quadratic functional approximation can be "trusted" (i.e., is reasonable). The quality of the quadratic approximation can be assessed from the following ratio:

$$\rho_k \;=\; \frac{f(\mathbf{x}_k) \;-\; f(\mathbf{x}_k + \mathbf{s}_k)}{f(\mathbf{x}_k) \;-\; q_k(\mathbf{s}_k)} \,. \tag{15}$$

A value near unity implied that the bound Δ_k imposed on $\mathbf{s}$ can be increased; in contrast, a small positive or a negative value for ρ_k implies that the quadratic model is poor, requiring a decrease in Δ_k.

Many Newton methods (see next section for details) based on trust-region approaches determine a candidate $\mathbf{s}_k$ by solving the linear system

$$\mathbf{H}(\mathbf{x}_k)\,\mathbf{s} \;=\; -\mathbf{g}(\mathbf{x}_k) \tag{16}$$

that results from minimizing $q_k(\mathbf{s})$. (A related system may also be formulated). The scaling of this vector $\mathbf{s}$ is determined according to the quality of the quadratic model at the region of approximation. A good source of a trust-region Newton method is the program LANCELOT [8].

4.3 Convergence Criteria

The criteria used to define convergence of the minimization algorithm must be chosen with care. The desire to obtain as accurate a result as possible should be balanced with the amount of computation involved. In other words, it is wasteful to continue a loop when the answer can no longer be improved.

Typically, minimization algorithms test for a sufficiently small gradient norm, i.e.,

$$\parallel \mathbf{g}(\mathbf{x}_k) \parallel \;\leq \epsilon_g \tag{17}$$

where ϵ_g is a small positive number such as 10^{-6}. However, ϵ_g should depend on the *machine precision*, ϵ_m, and the size of the problem. The quantity ϵ_m is roughly the largest number for which $1+\epsilon_m = 1$ in computer representation, rounded up to the nearest order or magnitude*. To introduce a proper dependency on the number of variables, the norm on the left-hand-side of equation (17) may be set to the Euclidean norm divided by $\sqrt{n}$. In addition, the right-hand-side may be modified by the factor $1+ \parallel f(\mathbf{x}_k) \parallel$ or $\max(1, \parallel \mathbf{x}_k \parallel)$.

For some functions and certain regions of the feasible space, the desired accuracy may be difficult to achieve if ϵ_g is smaller than warranted. (See the Newton section for illustrations). In that case, a well-structured algorithm should halt the iteration process when progress is poor. For example, progress can be assessed from the differences between successive iterates

*Typically, ϵ_m is 10^{-15} and 10^{-6}, respectively, for double and single-precision arithmetic.

of the independent vector ($\| \mathbf{x}_{k-1} - \mathbf{x}_k \|$) and/or between the associated function values, $f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})$. The algorithm should stop when these differences become small. The amount of progress realized (in terms of function decrease) could also be related to the size of step taken. In addition, algorithms often introduce limits for the number of allowable function and/or gradient evaluations. All these precautions are especially important for minimization of large-scale functions in macromolecular applications [1].

5 Newton's Method

Newton's[†] method is a classic iterative scheme for solving a nonlinear system $f(\mathbf{x}) = 0$ or for minimizing the multivariate function $f(\mathbf{x})$. These root-finding and minimization problems are closely related since obtaining the minimum of a function $f(x)$ can be formulated as solving for the zeros of $f'(x)$ for which $f''(x) > 0$.

Many effective methods for nonlinear, multivariate minimization can be related to Newton's method. This even includes the nonlinear conjugate gradient method, recently shown to be closely related to quasi-Newton methods. Hence, a good understanding of the Newton solver, including performance and convergence behavior, is invaluable for applying geometry optimization techniques in general. We first discuss the univariate case of Newton's method for obtaining the zeros of a function $f(x)$. In one dimension, instructive diagrams easily illustrate the method's strengths and weaknesses. We then discuss the general multivariate formulations and continue in the next section by describing the effective variants known as quasi-Newton, nonlinear conjugate gradient, and truncated-Newton methods.

5.1 Historical Perspective

A historical note on Newton's method is essential since the method's credit to Sir Isaac Newton is a partial one. Although many references also credit Joseph Raphson, the contributions of mathematicians Thomas Simpson and Jean-Baptiste-Joseph Fourier are also noteworthy. Furthermore, Newton's description of an algebraic procedure for solving for the zeros of a polynomial in 1664 had its roots in the work of the 16th-century French algebraist François Viète, which itself had precursors in the 11th-century works of Arabic algebraists.

In 1687, three years after Newton described a root finder for a polynomial, he described in *Principia Mathematica* an application of his procedure to a nonpolynomial equation. That equation originated from the problem of solving Kepler's equation: determining the position of a planet moving in an elliptical orbit around the sun, given the time elapsed since it was nearest the sun. Newton's procedure was nonetheless purely algebraic and not even iterative, as the solution process at each step was not the same.

In 1690, Raphson turned Newton's method into an iterative one, applying it to the solution of polynomial equations of degree up to ten. His formulation still did not use

[†] See historical note below on the method's name.

calculus; instead he derived explicit polynomial expressions for $f(x)$ and $f'(x)$.

Simpson in 1740 was the first to formulate the Newton-Raphson method on the basis of calculus. He applied the iterative scheme for solving general systems of nonlinear equations. In addition to this important extension of the method to nonlinear systems, Simpson extended the iterative solver to multivariate minimization, noting that by setting the gradient to zero the same method can be applied.

Finally, Fourier in 1831 published the modern version of the method as we know today in his celebrated book *Analyse des Équations Determinées*. The method for solving $f(x) = 0$ was simply written as:

$$x_{k+1} \;=\; x_k \;-\; f(x_k)/f'(x_k)\,. \tag{18}$$

Unfortunately, Fourier omitted credits to either Raphson or Simpson, possibly explaining the method's name.

For brevity, we refer below to the Newton-Raphson-Simpson-Fourier method as Newton's.

5.2 The One-Dimensional Version of Newton's Method

The iterative scheme of equation (18) can be easily derived by using a Taylor expansion to approximate a twice-differentiable function f locally by a quadratic function about x_k:

$$f(x_{k+1}) \;=\; f(x_k) \;+\; (x_{k+1} \;-\; x_k)\, f'(x_k) \;+\; \frac{1}{2}(x_{k+1} \;-\; x_k)^2\, f''(\xi) \tag{19}$$

where $x_k \leq \xi \leq x_{k+1}$. Omitting the second-derivative term, the solution of $f(x_{k+1}) = 0$ yields the iteration process of equation (18). Precursors to this method replaced $f'(x)$ as:

$$f'(x_k) \;\approx\; f(x_k + h) \;-\; f(x_k)\,/\,h \tag{20}$$

or

$$f'(x_k) \;\approx\; f(x_k) \;-\; f(x_{k-1})\,/\,(x_k \;-\; x_{k-1}) \tag{21}$$

where h is a suitably-chosen small number. These approximations — known as *finite-differences* and the method of *secants* — mirror the modern *discrete-Newton* and *quasi-Newton* methods. Note that the use of finite-differences to approximate the Hessian of a multivariate function requires one difference in gradient values for each column of the Hessian. This costly process is not feasible for large functions, unless problem structure can be exploited to reduce the work considerably.

Newton's method in one dimension has a simple geometric interpretation: at each step, approximate $f(x)$ by its tangent at point $\{x_k\,,\, f(x_k)\}$ and take x_{k+1} as the abscissa of the intersection of this line with the x-axis (see Figure 4). The method works well in the ideal case (Figure 4a), when x_0 is near the solution and $|f'(\xi)| \geq M > 0$ nearby. However, difficulties arise when x_0 is far from the solution, x^*, or $f'(x)$ too is close to zero (Figure 4b). This can occur when there are multiple roots so the curvature changes. Further difficulties

Figure 4

emerge when $f'(x)$ is identically zero at the solution (Figures 4c,d), in which case the root is considered "not simple".

Note that the Newton iteration process is undefined when $f'(x) = 0$ and can exhibit poor numerical behavior when $|f'(x)|$ is very small. In general, performance and attainable accuracy of the solver worsen if any of the above complications arise.

5.2.1 A Classic Example

A classic application of Newton's method is solving for the square root of a number a. Writing $f(x) = x^2 - a = 0$, we obtain the following iterative scheme for computing $x = \sqrt{a}$:

$$x_{k+1} = \frac{1}{2}\left[x_k + \frac{a}{x_k}\right] . \tag{22}$$

A computer result from a double-precision program is shown below for $a = 9$ and four starting points: 6, -50, 1000, and 10^{-6}.

```
 Newton iterate, x      Error:  |x-x*| / x*

 6.000000000000000       1.000000000000000
 3.750000000000000      0.2500000000000000
 3.075000000000000      2.5000000000000060E-02
 3.000914634146342      3.0487804878050656E-04
 3.000000139383442      4.6461147225803266E-08
 3.000000000000004      1.1842378929335002E-15
 3.000000000000000      0.0000000000000000E+00

-50.00000000000000       15.66666666666667
-25.09000000000000       7.363333333333333
-12.72435432443204       3.241451441477348
-6.715829684400157       1.238609894800052
-4.027973526155810      0.3426578420519366
-3.131173847545831      4.3724615848610281E-02
-3.002747624232596      9.1587474419870440E-04
-3.000001257088485      4.1902949495427794E-07
-3.000000000000263      8.7781633813695706E-14
-3.000000000000000      0.0000000000000000E+00

 1000.000000000000       332.3333333333333
 500.0045000000000       165.6681666666667
 250.0112499190007       82.33708330633358
 125.0236241495426       40.67454138318088
 62.54780527230187       19.84926842410063
 31.34584760656851       9.448615868856171
 15.81648348801446       4.272161162671487
```

```
8.192755049598201        1.730918349866067
4.645643305694222       0.5485477685647405
3.291471138804049       9.7157046268016398E-02
3.012905388073158       4.3017960243858511E-03
3.000027639275030       9.2130916765261386E-06
3.000000000127320       4.2440125488004320E-11
3.000000000000000       0.0000000000000000E+00

9.9999999999999995E-07 -0.9999996666666666
 4500000.000000500        1499999.000000167
 2250000.000001250        749999.0000004167
 1125000.000002625        374999.0000008750
 562500.0000053125        187499.0000017708
 281250.0000106563        93749.00000355208
 140625.0000213281        46874.00000710937
 70312.50004266406        23436.50001422135
 35156.25008533203        11717.75002844401
 17578.12517066602        5858.375056888672
 8789.062841333005        2928.687613777669
 4394.531932666483        1463.843977555494
 2197.266990333083        731.4223301110275
 1098.635543165269        365.2118477217562
 549.3218675724537        182.1072891908179
 274.6691257047825        90.55637523492750
 137.3509462008599        44.78364873361997
 68.70823588892520        21.90274529630840
 34.41961227628149        10.47320409209383
 17.34054555390984        4.780181851303282
 8.929780183224812        1.976593394408271
 4.968821869065736       0.6562739563552453
 3.390058212414881       0.1300194041382937
 3.022439940487617       7.4799801625389977E-03
 3.000083302057113       2.7767352371051619E-05
 3.000000001156507       3.8550229675138326E-10
 3.000000000000000       0.0000000000000000E+00
```

Note the very rapid, *quadratic* convergence in all cases at the last 4–5 steps. In these steps, the number of correct digits for the solution is approximately doubled from one step to the next! This is the ideal case because the root of $f(x)$ is simple and well separated from the other root. When x_0 is not close to the solution, convergence is much slower than quadratic at the beginning but then improves rapidly until the region of quadratic convergence is approached.

Note the longer time for convergence when x_0 is near zero ($x_0 = 10^{-6}$ shown). Since the derivative is zero at $x = 0$, the tangent takes the iterates very far, but then works back systematically toward the solution. If the algorithm of equation (18) is started from $x_0 = 0$,

the sequence of iterates diverges.

We see from this example that the order of convergence associated with a method (see below) measures *asymptotic* behavior; practical considerations such as accuracy and problem conditioning are at least as important.

5.2.2 Convergence Definitions

A sequence $\{x_k\}$ is said to *converge* to x^* *with order* p if p is the largest number such that a finite limit β (the "convergence ratio") exists, where:

$$0 \leq \lim_{k \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \beta < \infty . \tag{23}$$

When $p = 2$, we have *quadratic convergence*. When $p = 1$, we refer to the convergence as *superlinear* if $\beta = 0$ and as *linear* if $\beta \neq 0$.

For example, the reader can verify that the sequences $\{2^{-2^k}\}$, $\{k^{-k}\}$, and $\{2^{-k}\}$ converge, respectively, quadratically, superlinearly, and linearly. Quadratic convergence is faster than superlinear, which in turn is faster than linear.

5.2.3 Performance of Newton's Method

The proof of the quadratic convergence for Newton's method for a simple root and for x_0 sufficiently close to the solution x^* is given in Appendix A. If x_0 is far, bracketing safeguards are essential to guarantee convergence.

Quadratic convergence is often the fastest reference behavior for large-scale functions. Tensor methods based on fourth-order approximations to the objective function can achieve more rapid convergence [9] but they are restricted to problems of less than about 100 variables. A recent proposal has also shown that Newton's method converges faster than quadratic if higher-order derivatives are used [10].

The attainable accuracy for Newton's method depends on the function characteristics, computer type, and on whether the root is simple or not. Essentially, if the root is simple, the best accuracy attainable is of order ϵ_m (machine precision); the constant for the obtainable accuracy depends on the magnitude of $f'(\xi)$ near x^*. For a nonsimple root, the best attainable accuracy is of order $O(\sqrt{\epsilon_m})$, with a constant that depends on $|f''(\xi)|$ near x^*. This constitutes a substantial reduction considering that the typical value of ϵ_m is 10^{-15} for double-precision, and that many other errors are present in large-scale problems.

Appendix B sketches the proof for attainable accuracy in both cases.

5.3 Newton's Method for Minimization

To derive the iteration process of Newton's method for minimization of the one-dimensional $f(x)$, we use a quadratic, rather than linear, approximation:

$$f(x_{k+1}) \approx f(x_k) + (x_{k+1} - x_k) f'(x_k) + \frac{1}{2}(x_{k+1} - x_k)^2 f''(x_k) . \tag{24}$$

Since $f(x_k)$ is constant, minimization of the second and third terms on the right-hand-side in equation (24) yields the iteration process:

$$x_{k+1} = x_k - f'(x_k)/f''(x_k) . \tag{25}$$

Thus, we have replaced f and f' of equation (18) by f' and f'', respectively. This Newton scheme for minimizing $f(x)$ is defined as long as the second derivative at x_k is bound away from zero.

5.4 The Multivariate Version of Newton's Method

We generalize Newton's method for minimization in equation (25) to multivariate functions by expanding $f(\mathbf{x})$ locally along a search vector $\mathbf{p}$ (in analogy to equation (24)):

$$f(\mathbf{x}_k + \mathbf{p}_k) \approx f(\mathbf{x}_k) + \mathbf{g}(\mathbf{x}_k)^{\mathrm{T}} \mathbf{p}_k + \frac{1}{2} \mathbf{p}_k^{\mathrm{T}} \mathbf{H}(\mathbf{x}_k) \mathbf{p}_k . \tag{26}$$

Minimizing the right-hand side leads to solving the linear system of equations, known as the *Newton equations*, for $\mathbf{p}_k$:

$$\mathbf{H}(\mathbf{x}_k) \mathbf{p}_k = -\mathbf{g}(\mathbf{x}_k) . \tag{27}$$

Performing this approximation at each step k to obtain $\mathbf{p}_k$ leads to the iteration process

$$\mathbf{x}_{x+1} = \mathbf{x}_k - \mathbf{H}^{-1}(\mathbf{x}_k) \mathbf{g}(\mathbf{x}_k) . \tag{28}$$

This *classic* Newton method for minimization thus uses the search vector $\mathbf{p}_k = -\mathbf{H}^{-1}(\mathbf{x}_k) \mathbf{g}(\mathbf{x}_k)$ and requires solution of a linear system involving the Hessian at each step. Not only is this an expensive, order n^3 process for general dense matrices; for multivariate functions with many minima and maxima, the Hessian may be ill-conditioned (i.e., have near-zero eigenvalues) or singular (zero eigenvalues) for certain $\mathbf{x}_k$. Thus, in addition to the line-search or trust-region modifications that essentially dampen the Newton step (by scaling $\mathbf{p}_k$ by a positive scalar less than unity), effective strategies must be devised to ensure that $\mathbf{p}_k$ is well defined at each step and that the solution of the linear system at each step be reasonable computationally. Such effective strategies are described in the next section.

6 Effective Large-Scale Minimization Algorithms

The popular methods that fit the descent framework outlined in subsections 4.1 and 4.2 require gradient information. In addition, the truncated-Newton method may require more

input to be effective, such as partial, but explicit, second-derivative information. The methods described in this section render steepest descent (SD) obsolete as a general method. In SD, the descent direction is defined as $\mathbf{p}_k = -\mathbf{g}(\mathbf{x}_k)$ at each step. Only if the energy and gradient at $\mathbf{x}_0$ are very high, SD may be useful for a few iterations, before a better minimizer is applied.

6.1 Quasi-Newton

Quasi-Newton (QN) methods avoid using the actual Hessian and instead build up curvature information as the algorithm proceeds. Actually, it is often the Hessian inverse ($\widehat{\mathbf{B}}$) that is updated in practice so that a term $\widehat{\mathbf{B}}_k\,\mathbf{g}_k$ (where $\widehat{\mathbf{B}}_k$ and $\mathbf{g}_k$ are short hand for $\widehat{\mathbf{B}}(\mathbf{x}_k)$ and $\mathbf{g}(\mathbf{x}_k)$, respectively) replaces $\mathbf{H}^{-1}(\mathbf{x}_k)^{-1}\,\mathbf{g}_k$ in equation (28). The Hessian approximation $\mathbf{B}_k$ is derived to satisfy the *quasi-Newton condition* (see below). Quasi-Newton variants define different formulas that satisfy this condition. Since memory is considered premium for large-scale applications, the matrix $\mathbf{B}_k$ or $\widehat{\mathbf{B}}$ is formulated through several vector operations, avoiding explicit storage of an $n \times n$ matrix; the update $\mathbf{U}_k$, that when added to $\mathbf{B}_k$ defines $\mathbf{B}_{k+1}$, is thus said to be of *low-rank*.

Two important developments have emerged in modern optimization research in connection with QN methodology. The first is the development of limited-memory versions, in which the inverse Hessian approximation at step k only incorporates curvature information generated at the last few m steps (e.g., $m = 5$). The second is the emergence of insightful analyses that explain the relationship between QN and nonlinear conjugate gradient methods.

The QN condition that the new approximation $\mathbf{B}_{k+1}$ must satisfy is:

$$B_{k+1}\,\mathbf{s}_k \;=\; \mathbf{y}_k \tag{29}$$

where

$$\mathbf{s}_k \;=\; \mathbf{x}_{k+1} \;-\; \mathbf{x}_k\,, \tag{30}$$

$$\mathbf{y}_k \;=\; \mathbf{g}_{k+1} \;-\; \mathbf{g}_k\,. \tag{31}$$

If $f(\mathbf{x})$ were a quadratic function, its Hessian $\mathbf{H}$ would be a constant and would satisfy (from the Taylor expansion of the gradient) the following relation:

$$\mathbf{g}_{k+1} \;-\; \mathbf{g}_k \;=\; \mathbf{H}\,(\mathbf{x}_{k+1} \;-\; \mathbf{x}_k)\,. \tag{32}$$

This equation makes clear the origin of the QN condition of equation (29).

The updating QN formula can be written symbolically as:

$$\mathbf{B}_{k+1} \;=\; \mathbf{B}_k \;+\; \mathbf{U}_k(\mathbf{s}_k,\,\mathbf{y}_k,\,\mathbf{B}_k) \tag{33}$$

where $\mathbf{U}_k$ is a matrix of low rank (typically 1 or 2). Note that a rank 1 matrix can be written as the outer product of two vectors: $\mathbf{u}\mathbf{v}^{\mathrm{T}}$. In addition to rank, imposed symmetry and positive-definiteness are used in the formulation of $\mathbf{U}_k$.

One of the most successful QN formulas in practice is associated with the BFGS method (for its developers Broyden, Fletcher, Goldfard, and Shanno). The BFGS update matrix has rank 2 and inherent positive definiteness (i.e., if $\mathbf{B}_k$ is positive definite then $\mathbf{B}_{k+1}$ is positive definite) as long as $\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k < 0$. This condition is satisfied automatically for convex functions but may not hold in general. In practice, the line search must check for the descent property; updates that do not satisfy this condition may be skipped.

The BFGS update formula is given by

$$\mathbf{B}_{k+1} \;=\; \mathbf{B}_k \;-\; \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k\mathbf{s}_k} \;+\; \frac{\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\,. \tag{34}$$

The corresponding formula used in practice to update the inverse of $\mathbf{B}$, $\widehat{\mathbf{B}}$, is:

$$\widehat{\mathbf{B}}_{k+1} \;=\; \left(\mathbf{I} \;-\; \frac{\mathbf{s}_k\mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right)\,\widehat{\mathrm{B}}_k\left(\mathbf{I} \;-\; \frac{\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right) \;+\; \frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\,. \tag{35}$$

From this $\widehat{\mathbf{B}}$, the BFGS search vector is defined as $\mathbf{p}_k = -\widehat{\mathbf{B}}_k\,\mathbf{g}_k$.

Since we only need the product of $\widehat{\mathbf{B}}$ with the gradient (and not $\mathbf{B}$ or $\widehat{\mathbf{B}}$ *per se*), effective matrix/vector products have been developed to minimize storage requirements considerably for this special case of a low-rank matrix update. Typically, O(n) memory is required to store the successive pairs of update vectors ($\mathbf{s}_k$ and $\mathbf{y}_k$) and the respective inner products $\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k$. Limited-memory QN methods reduce storage requirements further by only retaining the {s, y} pairs from the previous few iterates (3–7). Since the older gradient differences used to define $\mathbf{B}_k$ can quickly become irrelevant for multivariate functions, the limited-memory BFGS method restarts the $\mathbf{B}_k$ update with a fresh matrix every few steps. The identity matrix, $\mathbf{I}$, or a multiple of it, is typically used for the initial approximation $\mathbf{B}_0$.

The limited-memory BFGS code of Nocedal and co-workers [11, 12, 13] is one of the most effective methods in this class. The combination of modest memory, requiring only gradient information, and good performance in practice makes it an excellent choice for large-scale multivariate minimization.

6.2 Conjugate Gradient

Nonlinear CG methods form another popular type of optimization scheme for large-scale problems where memory and computational performance are important considerations. These methods were first developed in the 1960s by combining the linear CG method (an iterative technique for solving linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is an $n \times n$ matrix [14]) with line-search techniques. The basic idea is that if f were a convex quadratic function, the resulting nonlinear CG method would reduce to solving the Newton equations (equation (27)) for the constant and positive-definite Hessian $\mathbf{H}$.

In each step of the nonlinear CG method, a search vector $\mathbf{d}_k$ is defined by a recursive formula. A line search is then used as outlined in Algorithm [A1]. The iteration process that

defines the search vectors $\{\mathbf{d}_k\}$ is given by:

$$\mathbf{d}_{k+1} \;=\; -\mathbf{g}_{k+1} \;+\; \beta_{k+1}\,\mathbf{d}_k\,, \tag{36}$$

where $\mathbf{d}_0 = -\mathbf{g}_0$. The parameter β_k, scheme dependent, is chosen so that if f were a convex quadratic and the line search exact (i.e., $\mathbf{x}_k + \lambda_k \mathbf{d}_k$ minimizes f exactly along $\mathbf{d}_k$), then the linear CG process would result. The reduction to the linear CG method in this special case is important because linear CG is known to terminate in at most n steps of exact arithmetic. This finite-termination property relies on the fundamental notion of *mutual conjugacy* ($\mathbf{g}_k^{\mathrm{T}}\mathbf{d}_j = 0$ for all $j < k$) for two sets of vectors ($\{\mathbf{g}\}$ and $\{\mathbf{d}\}$) generated in the CG method; this in turn implies that the search vectors span the entire n-dimensional space after n steps, so a solution should be obtained.

Different formulas for β_k have been developed for the nonlinear case, though they all reduce to the same expressions for convex quadratic functions. These variants exhibit different behavior in practice. Three of the best known algorithms are due to Fletcher-Reeves, Polak-Ribière, and Hestenes-Stiefel. They are given by:

$$\beta_{k+1}^{\mathrm{FR}} \;=\; \mathbf{g}_{k+1}^{\mathrm{T}}\mathbf{g}_{k+1} \,/\, \mathbf{g}_k^{\mathrm{T}}\mathbf{g}_k\,, \tag{37}$$

$$\beta_{k+1}^{\mathrm{PR}} \;=\; \mathbf{g}_{k+1}^{\mathrm{T}}\mathbf{y}_k \,/\, \mathbf{g}_k^{\mathrm{T}}\mathbf{g}_k\,, \tag{38}$$

$$\beta_{k+1}^{\mathrm{HS}} \;=\; \mathbf{g}_{k+1}^{\mathrm{T}}\mathbf{y}_k \,/\, \mathbf{d}_k^{\mathrm{T}}\mathbf{y}_k\,, \tag{39}$$

where $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$. The PR version is often found in software packages, but to be effective it restarts the iteration process (setting β_k to zero) occasionally (e.g., if β_k becomes negative). Some important modifications of this version (e.g., with slightly more memory requirements but fewer function evaluations) are due to Powell [15], available in the IMSL library, and to Shanno & Phua [16], available in the NAG library. A careful line search is important for nonlinear CG methods because the search directions tend to be poorly scaled.

Key connections between CG and quasi-Newton algorithms for minimization began to emerge in the late 1970s. Essentially, it was found that the CG conjugacy property can be closely related to the quasi-Newton condition, and so an appropriate formula for β_k could be obtained from both viewpoints. The many developments in the 1980s have shown that the limited-memory quasi-Newton class of algorithms best balances the extremely modest storage requirements of nonlinear CG with good convergence properties in practice. The fact that the unit steplength in quasi-Newton methods is often acceptable leads to greater efficiency in terms of function evaluations and hence less computational time overall.

Still, the linear and nonlinear CG methods play important theoretical roles in the numerical analysis literature as well as practical roles in many numerical techniques; see the recent research monograph, [17], for a modern perspective. The linear CG method, in particular, proves ideal for solving the linear subproblem in the truncated Newton method for minimization (discussed next), especially with convergence-accelerating techniques known as *preconditioning*.

6.3 Truncated-Newton

In the early 1980s a very simple but important idea emerged in connection with the Newton equations: why solve this linear system for the search vector $\mathbf{p}_k$ exactly [18]? In the context of large-scale nonlinear optimization, an accurate solution of equation (27) is not warranted! As long as the function and gradient-norm values are high, a descent direction may suffice and should result in greater computational savings, while still producing progress toward a minimum. Only near the solution, where the quadratic model is good, should the system be solved more accurately.

In practice, truncated-Newton (TN) methods allow a nonzero residual, r_k, for the Newton equations. For example, we can require

$$r_k \equiv \| \mathbf{H}_k \mathbf{p}_k + \mathbf{g}_k \| \leq \eta_k \| \mathbf{g}_k \| , \tag{40}$$

where η_k is the *forcing sequence*. This condition on the size of r_k at step k of the minimization scheme, becomes stricter as the gradient norm becomes smaller. Thus, near the solution we solve for $\mathbf{p}_k$ more accurately, whereas far away we permit a cruder approximation. Theoretical work further showed that asymptotic quadratic convergence of the method can be realized for a well chosen η_k as $\| \mathbf{g}_k \| \rightarrow 0$ [18]. For example, an effective setting is:

$$\eta_k = \min \{ c_r / k , \| \mathbf{g}_k \| \} , \qquad 0 < c_r \leq 1 . \tag{41}$$

This choice forces the residuals to be smaller and smaller as the number of iterations (k) increases and as the gradient becomes smaller. Another termination criterion based on the quality of the quadratic approximation has also been suggested [12].

To implement an upper bound on the residual norm in practice, an iterative, rather than direct procedure, that can be "truncated" is required for approximating $\mathbf{p}_k$ from equation (27) at each outer step k. The linear conjugate gradient method is an excellent candidate since it is simple and very modest in memory. The linear conjugate gradient algorithm mirrors in structure the general descent method of Algorithm [A1]. That is, it generates search vectors $\{\mathbf{d}_k\}$ at each step recursively (as the nonlinear conjugate gradient method of the previous subsection) but, in place of the line search, uses an explicit formula for the steplength. This expression is derived analytically by minimizing the quadratic model at the current point along $\mathbf{d}_k$ and then using the conjugacy condition to simplify the formula. However, to accelerate convergence, *preconditioning* is essential in practice. This technique involves modification of equation (27) through application of a closely-related matrix to $\mathbf{H}_k$, $\mathbf{M}_k$ (namely, multiplication of both sides by the inverse of $\mathbf{M}_k$). The preconditioner $\mathbf{M}$ is typically chosen as a sparse matrix that is rapid to assemble and factor. Theoretically, convergence improves if $\mathbf{M}_k^{-1}\mathbf{H}_k$, the coefficient matrix of the new linear system, has clustered eigenvalues or approximates the identity matrix.

The TN code in CHARMM [19] uses a preconditioner from the local chemical interactions (bond length, bond angle, and dihedral-angle terms). This sparse matrix is rapid to compute and was found to be effective in practice, whether the matrix is indefinite or not [20]. Other possibilities of preconditioners in general contexts have also been developed, such as a matrix derived from the BFGS update (defined in the quasi-Newton subsection) [12].

Although more complex to implement than quasi-Newton or nonlinear conjugate gradient methods, TN algorithms can be very efficient overall in terms of total function and gradient evaluations, convergence behavior, and solution accuracy, as long as the many components of the algorithm are carefully formulated (truncation, solution process for the inner loop, preconditioning, etc.).

In terms of the computational work per outer Newton step (k), TN methods based on preconditioned conjugate gradient require: a Hessian/vector product ($\mathbf{Hd}$) at each inner loop iteration, and one solution of a linear system $\mathbf{Mz} = \mathbf{r}$ where $\mathbf{M}$ is the preconditioner. Since $\mathbf{M}$ may be sparse, this linear solution often takes a very small percentage of the total CPU time (e.g., $< 3\%$ [20]). The benefits of faster convergence generally far outweigh these costs.

The Hessian/vector products in each linear conjugate gradient step are more significant. For a Hessian formulated with a nonbonded cutoff radius (e.g., 8 Å), many zeros result for the Hessian (see Figure 3); when this sparsity is exploited in the multiplication routine, performance is fast compared to a dense matrix/vector product. When the Hessian is dense and large in size, the following forward-difference formula of *two gradients* often works faster:

$$\mathbf{H}_k \, \mathbf{d}_k \;\approx\; [\mathbf{g}(\mathbf{x}_k + h\mathbf{d}_k) \;-\; \mathbf{g}(\mathbf{x}_k)] \,/\, h \,, \tag{42}$$

where h is a suitably-chosen small number. A central difference approximation may alternatively be used for greater accuracy, but it requires one more gradient evaluation than the one-sided difference formula above. In either case, finding an appropriate value for h is nontrivial, and the accuracy of the product near the solution (where the gradient components are small) can be problematic.

Thus, TN methods require more care in implementation details and user interface, but their performance is typically at least as good overall as limited-memory quasi-Newton methods. If simplicity is premium, the latter is a better choice. If partial second-derivative information is available, the objective function has many quadratic-like regions, and the user is interested in repeated minimization applications, TN algorithms may be worth the effort (see Table 1). In general, though Newton methods may not always perform best in terms of function calls and CPU time, they are the most reliable of methods for multivariate minimization and have the greatest potential for achieving very small final-gradient norms. This could be especially important if normal-mode analysis is performed following minimization.

Table 1

7 Available Software and Practical Recommendations

Table 2 summarizes the available minimizers in several chemistry and mathematics packages. See Ref. [11] for a recent compilation of mathematical software for optimization.

Table 2

Nonlinear conjugate gradient and various Newton methods are quite popular, but algorithmic details and parameters vary greatly from package to package. In particular, nonlinear conjugate gradient implementations are quite different. Several comprehensive mathematical libraries, such as IMSL, NAG, and MATLAB are sources of quality numerical software.

Table 1: Typical complexity versus performance in some optimization methods (SD: steepest descent, CG: conjuate gradient, QN: quasi-Newton, TN: truncated-Newton)

Complexity/ Convergence	Low	Moderate	High
Slow	SD		
Moderate	CG	QN,TN	Classic Newton
Rapid		QN,TN	Classic Newton, TN

In addition to careful implementation of popular algorithms, some of these companies offer useful service routines, such as for derivative checking and determination of appropriate finite-difference intervals.

Of special note is the adopted-basis Newton method implemented in CHARMM, ABNR. It is a memory-saving adaptation of Newton's method that avoids analytic second derivatives. The idea is to use steepest descent steps for a given number of iterations, m (e.g., 5), after which a set of $m+1$ coordinate and gradient vectors are available. A Hessian is constructed numerically in this $m \times m$ subspace, and all corresponding eigenvalues and eigenvectors are computed. If all eigenvalues are negative, steepest descent steps are used; if some are negative and some are positive, the search direction is modified by a Newton direction constructed from the eigenvectors corresponding to the positive eigenvalues only. In all cases, the n-dimensional search vector $\mathbf{p}_k$ is determined via projection onto the full space. The ABNR algorithm is similar in strategy to limited-memory quasi-Newton methods in that it uses only recent curvature information and exploits this information to make steady progress toward a solution.

The TN method in CHARMM is described in detail elsewhere [19, 20]. It uses a preconditioner constructed from the local chemical interactions (see Figure 3b) and determines $\mathbf{p}_k$ from a truncated preconditioned conjugate gradient loop. When negative curvature is detected, the preconditioned conjugate gradient loop is halted with a guaranteed direction of descent. Interestingly, recent analysis and experiments have shown that the method can produce quadratic convergence near a solution regardless of whether the preconditioner is indefinite or not [20]!

Performance comparisons in CHARMM among the conjugate gradient, ABNR, and TNPACK minimizers are shown in Table 3 for a dipeptide model and a protein. See Ref. [20] for details. Note that the same minimum is obtained for the small system (the dipeptide) but that different minima result for the much larger lysozyme system (though energy differences are not large). Considerable differences in CPU times can be noted, especially for lysozyme, where conjugate gradient is much slower. The conjugate gradient method also fails to produce very small gradient norms. Both Newton methods perform well for these problems, though ABNR is relatively expensive for the small system. TNPACK displays faster convergence and smaller final gradient norms. Note that conjugate gradient requires about 3 function evaluations per iteration (in the line search), while ABNR employs only one.

Table 3

TNPACK uses more than one, since the line search often produces a very small steplength at some iterations because of the indefinite preconditioner. The quadratic convergence of TNPACK is evident from Figure 5, where the gradient norm per iteration is shown. Figure 5

In general, geometry optimization in the context of molecular potential energy functions has many possible caveats. Hence, a novice user especially should take the following precautions to generate as much confidence as possible in a minimization result.

1. *Use many starting points.* There is always the possibility that the method will fail to converge from a certain starting point, or converge to a nearby stationary point that is not a minimum. A case in point is minimization of biphenyl from a flat geometry [21]; many minimizers will produce the flat ring geometry, but this actually corresponds to a maximum! Different starting points will produce the correct nonplanar structure.

2. *Compare results from different algorithms.* Since many packages offer more than one minimizer, experimenting with more than one algorithm is an excellent way to check a computational result. Often, one method may fail to achieve the desired resolution or may converge very slowly. Another reference calculation under the same potential energy surface may help assess the results.

3. *Compare results from different force fields whenever possible.* Putting aside the quality of the minimizer, the local minimum produced by any package is only as good as the force field itself. Since force fields for macromolecules today are far from converging to one another — in fact there are very large differences both in parameters and in functional forms — a better understanding of the energetic properties of various conformations can be obtained by comparing the relative energies of the different configurations as obtained by different force fields. Differences are expected, but the results should shed more insight into the lowest-energy configuration. If significant differences are observed, the researcher could further investigate both the associated force fields (e.g., a larger partial charge, an additional torsional term) and the minimizers for explanations.

4. *Check eigenvalues at the solution.* If the significance of the computed minima is unclear, the corresponding eigenvalues may help diagnose a problem. Near a true minimum, the eigenvalues should all be positive (except for the 6 zero components corresponding to translation and rotation invariance). In finite-precision arithmetic, "zero" will correspond to numbers that are small in absolute value (e.g., 10^{-6}), with "small" depending on both the machine and program precision. Values larger than this tolerance might indicate deviations from a true minimum, perhaps even a maximum or saddle point. In this case, the corresponding structure should be perturbed substantially and another trial of minimization attempted.

5. *Be aware of artificial minima caused by nonbonded cutoffs!* When cutoffs are used for the nonbonded interactions, especially in naive implementations involving sudden truncation or potential-switching methods, the energy and/or gradient can exhibit numerical artifacts: deep energy minima and correspondingly-large gradient value near

the cutoff region. Good minimizers can find these minima, which are correct as far as the numerical formulation is involved, but unfortunately not relevant physically. One way to recognize these artifacts is to note their large energy difference with respect to other minima obtained for the same structure (as obtained from different starting points or minima). These artificial minima should disappear when all the nonbonded interactions are considered, or improved spherical-cutoff treatments (such as force shifting and switching methods) are implemented instead.

8 Looking Ahead

Only a small subset of topics was covered here in the challenging and ever-evolving field of nonlinear large-scale optimization. Key factors that will undoubtedly influence the development of optimization algorithms in the next decade are the increase in computer memory and speed, and the growing availability of parallel platforms. Parallel architectures can be exploited, for example, for performing minimization simulations concurrently from different starting points, evaluating function and derivatives in tandem for greater efficiency in the line search or finite-difference approximations, and performing matrix decompositions in parallel for structured, separable systems.

The increase in computing speed is also making automatic differentiation a powerful resource for nonlinear optimization. In this technique, automatic programs are available that code function derivatives on the basis of a chain-rule application to the elementary constituents of a function [22]. It is foreseeable that such codes will replace finite-difference methods and make Newton methods more powerful [23]. The cost of differentiation is not reduced, but the convenience and accuracy may increase.

Function separability is a more general notion than sparsity, since all sparse systems are separable but the reverse is not true. It is also another area where algorithmic growth can be expected [23]. Separable functions are composites of subfunctions, each of which depends only on a small subset of the independent variables. Therefore, efficient schemes can be devised in this case to compute the search vector, function curvature, etc., much cheaper by exploiting the invariant subspaces of the objective function.

Such advances in local optimization will certainly lead to further progress in solving the global optimization problem as well; see Ref. [24] for recent examples. Scientists from all disciplines will anxiously await all these developments.

9 Related Articles In This Volume

- conformational analysis
- conformational search

- distance geometry
- force fields
- free-energy perturbation calculations
- molecular dynamics
- molecular mechanics
- Monte Carlo simulation methods
- simulated annealing
- structure elucidation

10 Acknowledgments

I thank Dexuan Xie and Fan Xu for preparing the figures. T. Schlick is an investigator of the Howard Hughes Medical Institute.

11 Bibliography

References

[1] Gill P.E.; Murray W.; Wright M.H. *Practical optimization*; Academic Press: London, England, 1983.

[2] Dennis J.E. Jr.; Schnabel R.B. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*; Prentice-Hall, Inc.: Englewood Cliffs, New Jersey, 1983. Reprinted with corrections by SIAM, 1996.

[3] Fletcher R. *Pratcial Methods of Optimization*; John Wiley & Sons: Tiptree, Essex, Great Britain, 1987.

[4] Luenberger D.G. *Linear and Nolinear Programming*; Addison Wesley: Reading, Massachussetts, 1984.

[5] Nocedal J. *Acta Numerica* **1996**, *1*, 199–242.

[6] Schlick T. In *Reviews in Computational Chemistry*; Lipkowitz K.B.; Boyd D.B., Eds.; VCH Publishers: New York, New York, 1992, 1–71.

[7] Schlick T; Parks G. DOE computational sciences education project, 1994. Chapter on Mathematical Optimization. URL: http://csep1.phy.ornl.gov/csep/CSEP.html.

[8] Conn A.R.; Gould N.I.M.; Toint Ph.L. LANCELOT*: A FORTRAN Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*; Springer-Verlag: New York, NY, 1992.

[9] Schnabel R.B.; Chow T. *SIAM J. Opt.* **1991**, *1*, 293–315.

[10] Ford W.F.; Pennline J.A. *SIAM Rev.* **1996**, *36*, 658–659.

[11] Moré J.J.; Wright, S.J. *Optimization Software Guide*, volume 14 of *Frontiers in Applied Mathematics*; SIAM: Philadelphia, PA, 1993. URL: http://www.mcs.anl.gov /home/otc/Guide/SoftwareGuide.

[12] Nash S.G.; Nocedal J. *SIAM J. Opt.* **1991**, *1*, 358–372.

[13] Byrd R.H.; Nocedal J.; Zhu C. In *Nonlinear Optimization and Applications*; Di Pillo G.; Giannessi F.; Eds.; Plenum, 1996.

[14] Golub G.H.; van Loan C.F. *Matrix Computations*; John Hopkins University Press: Baltimore, MD, second edition, 1986.

[15] Powell M.J.D. *Math. Prog.* **1977**, *12*, 241–254.

[16] Shanno D.F. Phua K.H.; *ACM Trans. Math. Softw.* **1980**, *6*, 618–622.

[17] Adams L.; Nazareth J.L.; Eds.; *Linear and Nonlinear Conjugate Gradient-Related Methods*; SIAM: Philadelphia, PA, 1996.

[18] Dembo R.S.; Steihaug T. *Math. Prog.* **1983**, *26*, 190–212.

[19] Derreumaux P; Zhang G.; Brooks B.; Schlick T. *J. Comp. Chem.* **1994**, *15*, 532–552.

[20] Xie D.; Schlick, T. The truncated Newton method search direction: Indefinite preconditioner and chemistry applications. Submitted, 1996.

[21] Lipkowitz K.B. *J. Chem. Educ.* **1995**, *72*, 1070–1075.

[22] Griewank A.; Corliss G.F.; Eds.; *Automatic Differentiation of Algorithms: Theory, Implementation, and Applications*; SIAM: Philadelphia, PA, 1991.

[23] Nocedal J. In *Proceedings of the State of the Art in Numerical Analysis*; Duff I.; Watson A.; Eds.; Oxford University Press, 1996. In Press.

[24] Pardalos P.M.; Shalloway D.; Xue G.; Eds.; *Global Minimization of Nonconvex Energy Functions: Molecular Conformation and Protein Folding*, volume 23 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*; American Mathematical Society: Providence, Rhode Island, 1996.

Appendix A: The Quadratic Convergence of Newton's Method

Let $\epsilon_k = x_k - x^*$, where x_k is the Newton iterate, $x_{k-1} - f(x_k)/f'(x_k)$, and x^* is the solution of $f(x) = 0$. Assume that $f(x)$ has continuous first and second derivatives; and assume that x^* is a simple root, i.e., $f'(x^*) \neq 0$ and $f'(x) \neq 0$ near x^*. We expand $f(x^*)$ in a Taylor series about x_k to obtain:

$$0 \;=\; f(x^*) \;=\; f(x_k) \;+\; (x^* \;-\; x_k)\, f'(x_k) \;+\; \frac{1}{2}(x^* \;-\; x_k)^2\, f''(\xi) \tag{A.1}$$

where $x_k \leq \xi \leq x_{k+1}$. Dividing by $f'(x_k)$ and rearranging, we have:

$$\epsilon_{k+1} \;=\; \frac{\epsilon_k^2}{2} \left(\frac{f''(\xi)}{f'(x_k)} \right) . \tag{A.2}$$

Therefore, we have quadratic convergence from:

$$\lim_{k \to \infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|^2} \;=\; \frac{1}{2} \frac{f''(x^*)}{f'(x^*)} \;\equiv\; \beta \tag{A.3}$$

since $\xi \to x^*$ as $\{x_k\} \to x^*$. Note that the convergence ratio is nonzero if $f''(x^*) \neq 0$. To complete the proof, we must show convergence for the sequence $\{\epsilon_k\}$, i.e., that $|\epsilon_k| \to 0$. For a simple root, we have an upper bound m on $\frac{1}{2}\frac{|f''(\xi)|}{|f'(x)|}$ for x and ξ in the neighborhood of a simple root. Thus if $m\,|x_0 - x^*| < 1$, from equation (A.2) we have $\epsilon_{k+1} \leq m\,|\epsilon_k|^2$, or

$$|m\,\epsilon_{k+1}| \;\leq\; (m\,\epsilon_k)^2 \;\leq\; (m\,\epsilon_0)^{2^k} . \tag{A.4}$$

Since $|m\,\epsilon_0| < 1$,

$$|\epsilon_{k+1}| \;\leq\; \frac{1}{m}\,(m\,\epsilon_0)^{2^k} \;\to\; 0 \tag{A.5}$$

as $k \to \infty$ or $x_k \to x^*$.

Appendix B: The Attainable Accuracy of Newton's Method

Case 1 (Simple Root). We assume that $\bar{x}$ is our computer approximation to the root x^* and that the first derivative satisfies $|f'(\xi)| \geq M$ for ξ between $\overline{x}$ and x^*. We also assume that the computed approximation to $f(x)$, namely $\bar{f}(x)$, incurs an error of at most δ, i.e., $|\bar{f}(x) - f(x)| \leq \delta$. Using the Taylor expansion, we write (since $f(x^*) = 0$):

$$f(\bar{x}) \;=\; f(\bar{x}) \;-\; f(x^*) \;=\; (\bar{x} - x^*)\, f'(\xi)\,, \qquad \bar{x} \leq \xi \leq x^*\,. \tag{B.1}$$

From this approximation, we have

$$|\bar{x} - x^*| \;=\; |f(\bar{x})| \,/\, |f'(\xi)| \;\leq\; |f(\bar{x})| \,/\, M \;\leq\; (|\bar{f}(\bar{x})| + \delta) \,/\, M\,, \tag{B.2}$$

or (since $\bar{f}(\bar{x}) = 0$),

$$|\bar{x} - x^*| \;\leq\; \delta/M\,. \tag{B.3}$$

Thus, the attainable accuracy, δ/M, is of order ϵ_m if $\delta \approx \epsilon_m$ and $1/M \approx 1$. The value $1/M$ depends on the curvature at x^*: the larger M is, the greater is the attainable accuracy.

Case 2 (Multiple Roots). We now assume that $\bar{x}$ is our computer approximation to an "ill-conditioned" root x_2^*, i.e., one for which $f'(x) \approx 0$ near x_2^*. Further, we assume a bound of the second derivative: $|f''(\xi)| \geq M'$ for ξ between $\overline{x}$ and x_2^*. We now expand $f(\bar{x})$ about x_2^*, but since $f'(x_2^*) = 0$, we use the second-derivative term:

$$f(\bar{x}) \;=\; f(\bar{x}) \;-\; f(x_2^*) \;=\; (\bar{x} - x_2^*)\, f'(x_2^*) \;+\frac{1}{2}(\bar{x} - x_2^*)^2\, f''(\xi)\,, \qquad \bar{x} \leq \xi \leq x_2^*\,. \tag{B.4}$$

Thus,

$$f(\bar{x}) = \frac{1}{2}(\bar{x} - x_2^*)^2\, f''(\xi)\,, \tag{B.5}$$

and

$$|\bar{x} - x_2^*| \;=\; \sqrt{\frac{2\,|f(\bar{x})|}{|f''(\xi)|}} \;\leq \sqrt{\frac{2\delta}{M'}}\,. \tag{B.6}$$

The attainable accuracy now depends on $2/M'$ and, if $\delta \approx \epsilon_m$, the best attainable accuracy is of order $\sqrt{\epsilon_m}$.

GLOSSARY:

Convergence order, p — the largest number such that a finite limit β (the convergence ratio) exists where $0 \leq \lim_{k\to\infty} \frac{|x_{k+1}-x^*|}{|x_k-x^*|^p} = \beta < \infty$. Linear, superlinear, and quadratic convergence refer to, respectively, $p = 1$ and $\beta \neq 0$, $p = 1$ and $\beta = 0$, and $p = 2$.

Descent direction — a direction along which function reduction can be achieved.

Gradient — the vector of n components corresponding to the first partial derivatives of a multivariate objective function of n variables.

Hessian — symmetric $n \times n$ matrix of second partial derivatives of a multivariate objective function of n variables.

Line search — a one-dimensional minimization process, component of many nonlinear optimization methods, performed via quadratic or cubic interpolation in combination with bracketing strategies.

Newton's method — a classic iterative scheme for solving for the zeros of a nonlinear function or minimizing a function. Credit for the method is also due to Raphson, Simpson, and Fourier.

Positive definite — a property of a symmetric matrix that generalizes to higher dimensions the notion of positive curvature.

Sparse matrix — a matrix with a large percentage of zeros.

Trust region — an optimization framework which determines the search vector at each step according to the size of region in which the objective function is well approximated by a quadratic model.

Figure 1: A one-dimensional function containing several minima

Figure 2: Contour curves for a quadratic function ($\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x}$) of two variables, where $\mathbf{A}$ is: (a) indefinite, with entries by row 1,2,2,2; (b) positive definite, entries 4,0,0,2; (c) negative definite, entries -1,0,0,-4; and (d) singular, entries 1,1,1,1.

Figure 3: Sparse matrix structures resulting from the Hessian of the potential energy function of the protein BPTI: (a) when 8-Å cutoffs are used for the nonbonded terms, 13% nonzeros; and (b) when only bond-length, bond-angle, and dihedral-angle terms are included, with insets showing enlarged submatrices.

Figure 4: Newton's method in one dimension: (a) geometric interpretation and behavior in the ideal case; (b) divergent behavior; and (c),(d) behavior in difficult cases of nonsimple roots.

Figure 5: Minimization progress (gradient norm) of three CHARMM minimizers for: (a) alanine dipeptide; and (b) lysozyme. See Table 3.

Table 2: Available optimization algorithms

Package	Contact	Minimizers
AMBER	http://www.amber.ucsf.edu /amber/amber.html	SD, nonlinear CG from the IMSL library (due to Powell), and Newton
CHARMM	Martin Karplus marci@brel.u-strasbg.fr	SD, nonlinear CG (FR, and modified PR[a] version, the latter from the IMSL library), Adopted-Basis Newton (ABNR), Newton, truncated-Newton (TNPACK)
DISCOVER	Biosym Technologies, San Diego, CA	SD, nonlinear CG (PR, FR versions), quasi-Newton, truncated Newton
DUPLEX	Brian E. Hingerty hingertybe@ornl.gov	Powell's coordinate descent method (no derivatives)
ECEPP/2	QCPE 454 http://qcpe5.chem.indiana .edu/qcpe.html	Calls SUMSL, a quasi-Newton method based on a trust-region approach (by Gay)
GROMOS	http://igc.ethz.ch/gromos	SD and nonlinear CG (FR version), both with and without SHAKE constraints
IMSL Lib.	IMSL, Inc., Sugar Land, TX http://www.vni.com/adt .dir/imslinfo.html	Many routines for constrained and unconstrained minimization (nonsmooth, no derivatives, quadratic and linear programming, least-squares, nonlinear, etc.), including a nonlinear CG method of Powell (modified PR version with restarts)
LANCELOT	Philippe Toint pht@math.fundp.ac.be	Various Newton methods for constrained and unconstrained nonlinear optimization, specializing in large-scale problems and including a trust-region Newton method and an algorithm for nonlinear least squares that exploits partial separability
MATLAB	The Math Works, Inc., Natick, MA info@mathworks.com, http://www.mathworks.com	SD, DFP[b] and BFGS quasi-Newton, simplex algorithm, and others for linear and quadratic programming, least squares, etc.
MMFF94	Tom Halgren halgren@merck.com	Calls OPTIMOL which uses a BFGS quasi-Newton method, with the initial inverse Hessian approximated from the inverse of a 3×3 block-diagonal Hessian
MM4, MM3	http://europa.chem.uga.edu	3×3 block-diagonal Newton and full Newton
MM2	http://europa.chem.uga.edu	3×3 block-diagonal Newton
NAG Lib.	NAG, Inc., Downers Grove, IL	Quasi-Newton, modified Newton and nonlinear CG (CONMIN by Shanno & Phua, modified PR version); also quadratic programming, least squares minimization, and many service routines
SiGMA	http://femto.med.unc.edu /SIGMA/	Nonlinear CG (FR version)
X-PLOR	http://xplor.csb.yale.edu	Nonlinear CG (from IMSL library)

[a] FR and PR refer to the Fletcher-Reeves and Polark-Ribière nonlinear CG versions

[b] DFP is a rank-1 QN method, credited to Davidon, Fletcher, and Powell

Table 3: Performance among three CHARMM minimizers on two molecular systems

Minimizer	Final f	Final $\|\mathbf{g}\|$	Iterations	f & $\mathbf{g}$ Evals	CPU Time
Alanine dipeptide (66 variables)					
CG	-15.24501	9.83×10^{-7}	882	2507	2.34 sec.
ABNR	-15.24501	9.96×10^{-8}	16466	16467	7.47 sec.
TNPACK	-15.24501	7.67×10^{-11}	29 (210 PCG)	44	1.32 sec.
Lysozyme (6090 variables)					
CG	-4628.362	9.89×10^{-5}	9231	24064	19.63 hrs.
ABNR	-4631.584	9.97×10^{-6}	7637	7638	6.11 hrs.
TNPACK	-4631.380	1.45×10^{-6}	78 (1848 PCG)	218	1.49 hrs.