Molecular Modeling and Simulation: An Interdisciplinary Guide

Tamar Schlick

June 4, 2003

Book URLs

For Text:
monod.biomath.nyu.edu/index/book.html

For Course:
monod.biomath.nyu.edu/index/course/IndexMM.html

Acronyms, Abbreviations, and Units

A

A	adenine (purine nitrogenous base)
Å	angstrom (10^{-10} m)
AdMLP	adenovirus major late promoter (protein)
AIDS	acquired immune deficiency syndrome
Ala (A)	alanine
Arg (R)	arginine
Asp (D)	asparagine
Asn (N)	aspartic acid
AS	Altona/Sundaralingam (sugar description)
ATP	adenosine triphosphate (energy source)
AZT	zidovudine (AIDS drug)

B

bp	base pair
bps	base pairs
BAC	bacterial artificial chromosome
BOES	Born-Oppenheimer energy surfaces
BPTI	bovine pancreatic trypsin inhibitor
BSE	bovine spongiform encephalopathy (‘mad cow disease’)

C

cm	centimeter (10^{-2} m)
C	cytosine (pyrimidine nitrogenous base)

CAP	catabolite gene activator protein
CASP	Critical Assessment of Techniques for Protein Structure Prediction
CG	Conjugate gradient method (for minimization)
CJD	Creutzfeld-Jakob disease (brain disorder, human version of BSE)
CN	Crigler-Najjar (debilitating disease, gene therapy applications)
CP	Cremer/Pople (sugar description)
CPU	central processing units
Cys (C)	cysteine

D

DFT	density functional theory (quantum mechanics approach)
DH	Debye-Hückel
DNA	deoxyribonucleic acid (also A-, B-, C-, D-, P-, S-, T-, and Z-DNA)
DOE	Department of Energy

E

erg	energy unit (10^{-7} J)
EM	electron microscopy

F

fs	femtosecond (10^{-15} s)
FFT	Fast Fourier Transforms

G

G	guanine (purine nitrogenous base)
Gln (Q)	glutamine
Glu (E)	glutamic acid
Gly (G)	glycine
GSS	Gerstmann-Straussler-Scheinker disease (brain disorder similar to CJD)

H

HDV	hepatitis delta helper virus
His (H)	histidine
HIV	human immunodeficiency virus
HMC	hybrid Monte Carlo
HTH	helix/turn/helix (motif)
Hz	hertz (inverse second)

I

Ile (I)	isoleucine
IHF	integration host factor (protein)

K

kbp	kilobase pairs
kcal/mol	kilocalories per mole (energy unit)
kDa	kilodaltons (mass unit used for proteins)
KR	Kirkwood-Riseman

L

Leu (L)	leucine
Lys (K)	lysine
LCG	linear congruential generator

M

m	meter
mgr	minor groove
ms	millisecond (10^{-3} s)
μs	microsecond (10^{-6} s)
mm	millimeter (10^{-3} m)
MAD	multiple isomorphous replacement (crystallography technique)
MC	Monte Carlo
MD	molecular dynamics
Met (M)	methionine
Mgr	major groove
MIR	multiwavelength anomalous diffraction (crystallography technique)
MLCG	multiplicative linear congruential generator
MTS	multiple-timestep methods (for MD)

N

nm	nanometer (10^{-9} m)
ns	nanosecond (10^{-9} s)
NCBI	National Center for Biotechnology Information
NASA	National Aeronautics and Space Administration
NDB	nucleic acid database (ndbserver.rutgers.edu/)
NIH	National Institutes of Health
NMR	nuclear magnetic resonance
NSF	National Science Foundation

O

OTC	ornithine transcarbamylase (chronic ailment, gene therapy applications)

P

pn	picoNewton (force unit)
ps	picosecond (10^{-12} s)
PB	Poisson-Boltzmann

PBE	Poisson-Boltzmann equation
PC	principal component
PCA	principal component analysis
PCR	polymerase chain reaction
PDB	protein databank (www.rcsb.org/pdb)
Phe (F)	phenylalanine
PIR	Protein Information Resource (pir.georgetown.edu)
PME	particle-mesh Ewald
PNA	peptide nucleic acid (DNA mimic)
Pro (P)	proline
PrP^{C}	prion protein cellular (harmless)
PrP^{Sc}	harmful isoform of PrP^{C}, causes scrapie in sheep
Pur	purine (base)
Pyr	pyrimidine (base)

Q

QM	quantum mechanics
QN	quasi Newton method (for minimization)
QSAR	quantitative structure/activity relationships

R

RCSB	Research Collaboratory for Structural Bioinformatics (www.rcsb.org)
RMS (rms)	root-mean-square
RMSD	root-mean-square deviations
RNA	ribonucleic acid (also cRNA, gRNA, mRNA, rRNA, snRNA, tRNA)
RT	reverse transcriptase (AIDS protein)

S

s	second
Ser (S)	serine
SAR	structure/activity relationships
SCF	self-consistent field (quantum mechanical approach)
SCOP	structural classification of proteins (scop.mrc-lmb.cam.ac.uk/scop/)
SD	steepest descent method (for minimization)
SGI	Silicon Graphics Inc.
SNPs	single-nucleotide polymorphisms (“snips”)
SRY	sex determining region Y (protein)
STS	single-timestep methods (for MD)
SVD	singular value decomposition

T

T	thymine (pyrimidine nitrogenous base)
Thr (T)	threonine

Trp (W)	tryptophan
Tyr (Y)	tyrosine
TBP	TATA-box DNA binding protein (transcription regulator)
TE	transcription efficiency
TMD	targeted molecular dynamics
TN	truncated Newton method (for minimization)
2D	two-dimensional
3D	three-dimensional

U

U	uracil (pyrimidine nitrogenous base)
URL	uniform resource locator
UV	ultraviolet spectroscopy

V

Val (V)	valine

W

WC	Watson/Crick base pairing

10

Multivariate Minimization in Computational Chemistry

Chapter 10 Notation

SYMBOL	DEFINITION
Matrices	
$\mathbf{A}$	symmetric matrix, components $\{A_{ij}\}$
$\widehat{\mathbf{B}}_k$	approximation to Hessian inverse at $\mathbf{x}_k$ (QN methods)
$\mathbf{D}_k$	scaling matrix at step k of minimization method (trust region approach)
$\mathbf{H}$	Hessian matrix, components $H_{ij}(\mathbf{x}) \equiv \partial^2 f(\mathbf{x})/\partial x_i \partial x_j$
$\mathbf{I}$	identity matrix
$\mathbf{M}$	preconditioning matrix, related to $\mathbf{H}$ (TN methods)
$\mathbf{M}_k$	preconditioning matrix at step k of TN method
$\mathbf{U}_k$	QN low-rank update matrix at step k
Vectors	
$\mathbf{b}, \mathbf{y}$	constant vectors
$\mathbf{e}_j$	unit vectors
$\mathbf{g}$	gradient vector of f, components $g_i(\mathbf{x}) \equiv \partial f(\mathbf{x})/\partial x_i$
$\mathbf{g}_k$	gradient vector at $\mathbf{x}_k$ (short hand for $\mathbf{g}(\mathbf{x}_k)$)
$\mathbf{p}_k$	search vector at step k of minimization method
$\mathbf{p}_k^j$	inner-loop CG iterate j for outer-loop search vector $\mathbf{p}_k$ (TN method)
$\mathbf{r}$	residual vector defined in TN methods ($\mathbf{Mz} = \mathbf{r}$)
$\mathbf{s}_k$	displacement vector at step k, $\mathbf{x}_{k+1} - \mathbf{x}_k$ (QN method)
$\mathbf{x}$	vector of n components $\{x_i\}$
$\mathbf{x}_0$	starting point vector for minimization
$\mathbf{x}_k$	minimization iterate at step k of method
$\mathbf{x}^*$	local minimum point of objective function
$\mathbf{y}_k$	gradient difference vector at QN step k, $\mathbf{g}_{k+1} - \mathbf{g}_k$
$\mathbf{z}$	solution vector defined in TN methods ($\mathbf{Mz} = \mathbf{r}$)

Chapter 10 Notation Table (continued)

SYMBOL	DEFINITION
Scalars & Functions	
a, b	numbers
$c_i(\mathbf{x})$	constraint function i
c_r	small positive number (in TN methods)
f_0	constant (function value)
$f(\mathbf{x})$	objective function, dependent on vector $\mathbf{x}$
h	small number (fi nite difference interval)
n	problem dimension
p	convergence order
$q(\mathbf{x})$	quadratic function
$q_k(\mathbf{s})$	quadratic model of objective function
r_k	residual norm at step k of TN methods
α	line search parameter for suffi cient decrease condition
β	line search parameter for suffi cient decrease of curvature (also convergence ratio)
β_k	scheme-dependent scale parameter of search vector $\mathbf{p}_k$ (CG and QN methods)
ϵ_f, ϵ_g	small positive numbers
ϵ_m	small positive number, machine precision
η_k	forcing sequence in TN methods
λ	line search steplength
λ_t	trial line search steplength
ξ	variable in the neighborhood of x for a univariate function $f(x)$
$\phi(\lambda)$	polynomial of steplength λ
Δ_k	size bound in QN methods at step k

'Pon my word Watson, you are coming along wonderfully. We have really done very well indeed. It is true that you have missed everything of importance, but you have hit upon the method.

Arthur Conan Doyle (1859–1930), in *A Case of Identity (1891)*.

10.1 Ubiquitous Optimization: From Enzymes to Weather to Economics

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. It can also serve as a starting point for subsequent calculations, such as molecular dynamics simulations or normal-mode analyses.

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 everyday life — responsible for our weather forecasts, flight planning, telephone routing, microprocessor design, and the functioning of enzymes in our bodies.

10.1.1 Algorithmic Sophistication Demands Basic Understanding

The mathematical techniques developed to address these optimization problems are just as robust and varied as the target problems themselves. The algorithmic complexity of such techniques has led to many available computer programs that require minimal input from the user (e.g., the starting point and a routine for function evaluation).

However, the prudent 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 application both efficient and correct, in terms of the physical interpretations.

This chapter introduces key optimization concepts for this purpose. We also highlight the fundamentals of local optimizers for *large-scale nonlinear unconstrained problems*, an important optimization subfield relevant to biological macromolecules. We describe the most promising approaches among them, and discuss practical issues, such as parameter variations and termination criteria. Of course, the latter are best learned by experimentation in the context of real problems. To illustrate behavior for complex problems, some comparisons among three competitive minimizers are also included, for molecular models minimized in the molecular mechanics and dynamics program CHARMM.

10.1.2 Chapter Overview

Specifically, Section 10.2 introduces optimization fundamentals such as problem formulation and terminology. Section 10.3 describes the basic algorithmic framework of iterative minimization protocols (based on line search and trust region methods); it also discusses convergence criteria and line search procedures and introduces the key concept of descent directions.

In Section 10.4, we present the Newton method, including a historical perspective, and one-dimensional implementations for nonlinear equations as well as optimization. This presentation familiarizes readers with the Newton method framework — the basis for formulating many other optimization methods — and with performance and convergence issues relevant to minimization of multivariate functions.

In Section 10.5, we mention effective methods for large-scale nonlinear optimization, namely quasi-Newton (QN), nonlinear conjugate gradient (CG), and truncated Newton (TN) schemes. Section 10.6 outlines available software and presents comparative performance in CHARMM for two molecular models (a small model system and a protein). Finally, in Sections 10.7 and 10.8, we summa-

rize recommendations to optimization practitioners and offer a future perspective to field developments.

For details, as well as for other categories of the rich and exciting field of optimization, I refer readers to classic texts [29, 16, 22, 55, 66], some reviews [63, 77, 82, 78, 24], and a perspective [86].

10.2 Optimization Fundamentals

The methods for solving an optimization task depend on the problem classification. Since the value of the independent variable that maximizes a function f also minimizes 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 *statement* (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.

10.2.1 Problem Formulation

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

$$\min_{\mathbf{x}} \{f(\mathbf{x})\}, \quad \mathbf{x} \in \mathcal{D}, \tag{10.1}$$

where f is the objective function and $\mathcal{D}$ is a given region (which can be the entire Euclidean space $\Re^n$). The problem can be subject to m constraints, which can be written more generally as a combination of equality and inequality constraints:

$$\begin{aligned} c_i(\mathbf{x}) &= 0 \quad \text{for } i = 1, \ldots, m', \\ c_i(\mathbf{x}) &\leq 0 \quad \text{for } i = m'+1, \ldots, m. \end{aligned} \tag{10.2}$$

This general formulation can be obtained for problems with bound constraints in the form

$$c_i(\mathbf{x}) = x_i \,,$$

where x_i is the ith component of the vector $\mathbf{x}$, or for problems with two-sided constraints such as

$$l_i \leq c_i(\mathbf{x}) \leq u_i \,.$$

In this chapter, we only cover unconstrained optimization formulations. For a comprehensive review of interior methods for continuous nonlinear optimization problems subject to constraints, see [24].

10.2.2 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',[1] also classified as combinatorial optimization. See [86], for example, and references cited therein.

10.2.3 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.

Linear and Quadratic Functions

Linear objectives can be written in vector form as

$$f(\mathbf{x}) = \mathbf{b}^T\mathbf{x} + f_0 \,, \tag{10.3}$$

where $\mathbf{b}$ is a column vector of dimension n, and f_0 is a scalar. Quadratic objective functions can be expressed as

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + f_0 \,, \tag{10.4}$$

where $\mathbf{A}$ is a constant *symmetric* matrix of dimension $n \times n$. (By definition, the n^2 entries of a symmetric matrix $\mathbf{A}$ satisfy $A_{i,j} = A_{j,i}$). The superscripts ^{T} above refer to a vector transpose; thus $\mathbf{x}^T\mathbf{y}$ is an inner product.

Linear programming problems refer to linear objective functions subject to linear constraints (i.e., a system of linear equations and inequalities), and *quadratic programming* problems have quadratic objective functions and linear constraints.

[1] The notorious 'traveling salesman' problem seeks to find the optimal travel route that covers a given number of cities, each one only once, and returning to the home town. Visually, imagine drawing such a route on a map, where each city k for $k = 0, \ldots, n$ is designated by coordinates $\{x_k, y_k\}$. The connected route started at $\{x_0, y_0\}$ covers each city and returns to the original point. Though simple to envision, there are clearly many such routes, and the number of combinations that connect all these cities grows steeply with n. This problem in fact belongs to a class of very difficult problems (known as *NP-complete*) for which no polynomial-complexity algorithm is known (i.e., the computational time for an *exact* solution of this problem increases exponentially with n).

Least-Squares Functions

Nonlinear functions can be classified further. *Least-squares* functions have the form

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i(\mathbf{x})^2 \,. \tag{10.5}$$

Separable Functions

Separable functions can be expressed as a sum of subfunctions, namely

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}) \,, \tag{10.6}$$

where each subfunction f_i depends only on a subset of the independent variables. That is, for each subfunction f_i there are many unit vectors $\mathbf{e}_j$ (with 1 in component j and 0 elsewhere) for which $f_i(\mathbf{x} + \mathbf{e}_j) = f_i(\mathbf{x})$. All molecular mechanics potential functions arising from the local, bonded interactions can be written this way.

Nonsmooth Functions

Because most optimization algorithms exploit derivative information to locate optima, nonsmooth functions pose special difficulties, and very different algorithmic approaches must be used. See [7] and [22, Chapter 14] for a general introduction to nonsmooth optimization, and the two-volume set [35, 36] for the special case of nonsmooth convex problems. Optimization of nonsmooth functions requires new mathematical machinery (e.g., *subdifferentials*) that extends ordinary differentiation and leads to counterparts of most results in differential calculus (Taylor expansions, mean value theorem, etc.).

Potential Energy Functions

Geometry optimization problems for molecular potential functions in the context of standard all-atom force fields in computational chemistry are typically of the multivariate, continuous, and nonlinear type [78]. They can be formulated as constrained (as in adiabatic relaxation, an example of which was shown in Chapter 5) or unconstrained. Discontinuities in the derivatives may be a problem in certain formulations involving truncation, such as of the nonbonded terms (see Section 10.6).

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 we will discuss, effective techniques are available today that achieve rapid convergence even for large systems. In practice, for macromolecular applications these optimization algorithms must be modest in storage requirements and economical in computations, which are dominated by the function and derivative evaluations.

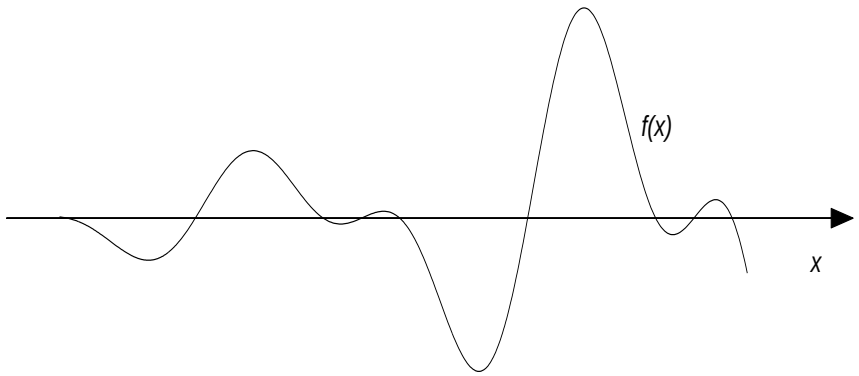

Figure 10.1. A one-dimensional function with several minima. This function was constructed from the actual univariate function at one line search step of the truncated Newton algorithm (see later in chapter) applied to minimization of a small protein's potential energy function.

10.2.4 Local and Global Minima

Definitions

The *local* unconstrained optimization problem in the Euclidean space $\Re^n$ can be stated as in eq. (10.1) for $\mathbf{x} \in \mathcal{D} \subset \Re^n$ where $\mathcal{D}$ denotes a neighborhood of the starting point, $\mathbf{x}_0$. The *global* optimization problem is much more difficult because it requires finding the global minimum among all the local minima, and the number of minima can be exponentially large.

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

$$f(\mathbf{x}^*) < f(\mathbf{y}) \qquad \text{for all } \mathbf{y} \in \mathcal{D}, \ \ \mathbf{y} \neq \mathbf{x}^*. \tag{10.7}$$

The point $\mathbf{x}^*$ is a *weak local minimum* if $f(\mathbf{x}^*) \leq f(\mathbf{y})$.

A *global minimum* $\mathbf{x}^*$ satisfies the stringent requirement that

$$f(\mathbf{x}^*) < f(\mathbf{y}) \qquad \text{for all } \mathbf{y} \neq \mathbf{x}^*. \tag{10.8}$$

See Figure 10.1 for an illustration of a one-dimensional function with several minima. The function corresponds to the actual univariate function minimized in the line search substep of the TN method (see later in chapter for details).

Convergence

Finding a *local minimum* 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 various errors that

accumulate over many operations also degrade practical performance in comparison to theoretical expectations (which can be described as *convergence order*; see Box 10.1). Nonetheless, the local optimization problem is solved in a mathematical sense: convergence to a local minimum can be achieved on modern computers. In the mathematical literature, this is referred to as *global convergence* to a local minimum. Still, though many algorithms are available in widely-used molecular mechanics and dynamics packages, performance and solution quality vary considerably and depend greatly on the user-specified algorithmic convergence parameters and the starting point.

The global optimization problem, by 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. See [23], for example, for a review, the website at www.mat.univie.ac.at/~neum/glopt.html for general information, and homework 13 for the deterministic global optimization approach based on the *diffusion equation* [73].

Global minimization algorithms differ from the local schemes in that they do not necessarily require the energy to decrease systematically, making possible escape from local potential wells and entry into others. Global optimization methods can be stochastic or deterministic, or a combination thereof; they often rely on local optimization components.

Box 10.1: Convergence Definitions

A sequence $\{x_k\}$ *converging* to x^* has *order p* if p is the largest number such that a finite limit β (the "convergence ratio", not to be confused with the line search parameter β) exists, where:

$$0 \leq \lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \beta < \infty. \tag{10.9}$$

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

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.

10.2.5 Derivatives of Multivariate Functions

Gradient

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

$$g_i(\mathbf{x}) \;=\; \partial f(\mathbf{x})/\partial x_i. \tag{10.10}$$

Hessian and Curvature

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

$$H_{i,j}(\mathbf{x}) \;=\; \partial^2 f(\mathbf{x})/\partial x_i \partial x_j\,. \tag{10.11}$$

At a *stationary* point, the gradient is zero. At a minimum point $\mathbf{x}^*$, in addition to stationarity, the curvature is positive. For higher dimensions, convexity is expressed as *positive-definiteness* of the Hessian. A multivariate function is positive-definite at a point $\mathbf{x}^*$ if

$$\mathbf{y}^T\,\mathbf{H}(\mathbf{x}^*)\,\mathbf{y} \;>\; 0 \quad \text{for all nonzero } \mathbf{y}. \tag{10.12}$$

In particular, positive definiteness guarantees that all the eigenvalues are positive at $\mathbf{x}^*$. A *positive semi-definite* matrix has nonnegative eigenvalues; a *negative semi-definite* matrix has nonpositive eigenvalues; and a *negative-definite* matrix has only negative eigenvalues. Otherwise, the matrix is *indefinite*. The utilization of curvature information is important for formulating effective multivariate optimization algorithms.

Figure 10.2 illustrates this notion of curvature for quadratic functions of two variables:

$$q(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}\,.$$

Namely, it displays the *contours* of these functions — curves on which the function is constant — in four cases. These cases are defined by different properties of the matrix $\mathbf{A}$: (a) indefinite, (b) positive definite, (c) negative definite, and (d) singular (i.e., not invertible). Figure 10.3 displays corresponding three-dimensional views of the functions, with circles and a line indicating stationary points. We use similar contour plots later (Figure 10.10) to illustrate paths of different minimization algorithms.

10.2.6 The Hessian of Potential Energy Functions

Sparsity

A matrix is termed *sparse* if it has a large percentage of zero entries; otherwise it is *dense*. (There is no specific threshold percentage of zero elements below which a matrix is considered 'sparse'). A sparse matrix can be *structured*, as in a banded matrix of bandwidth p where there are zeros for $|i-j| > p$. Alternatively, a sparse matrix can be *unstructured*, as shown in Figures 10.4 and 10.5.

In these figures, the matrix indices are the independent variables (three times the number of atoms) of the potential energy function for molecular systems. A point in the matrix position $\{i, j\}$ indicates a nonzero Hessian element for the

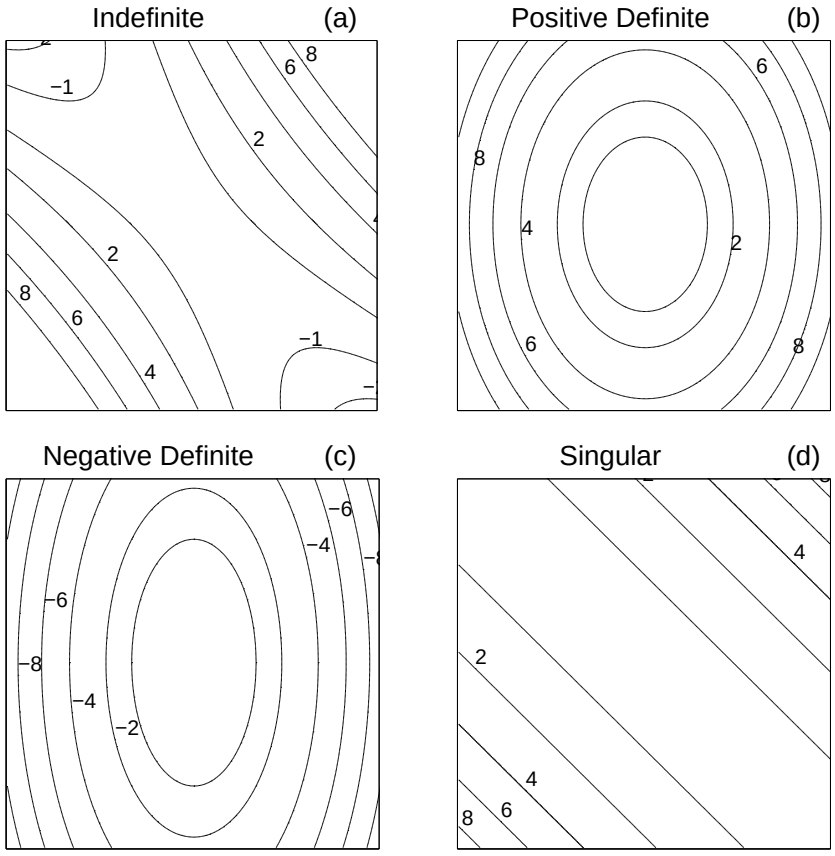

Figure 10.2. Two-dimensional contour curves for the quadratic function $q(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}$ of two variables, where $\mathbf{A}$ is: (a) indefinite, with entries by row 1,2,2,2; (b) positive definite, entries 4,0,0,2; (c) negative definite, entries $-1,0,0,-4$; and (d) singular, entries 1,1,1,1. See also Figure 10.3.

second-derivative term of the potential energy objective function. Examples are shown for various molecular systems. The left-column matrices correspond to the Hessian pattern resulting when 8 Å cutoffs are used for the nonbonded terms. The right-column patterns correspond to only the local, bonded second-derivative terms (bond-length, bond angle, and dihedral-angle). The insets zoom on two submatrices and illustrate how the sparsity pattern repeats in triplets (for the x, y, and z components), and how nearly banded the local Hessian structure is due to the finite range of the bonded interactions.

We also see that although the matrices corresponding to 8 Å cutoffs are sparse for the larger systems, the atom ordering used determines the resulting pattern. For example, the X pattern for the DNA system results from the consecutive ordering

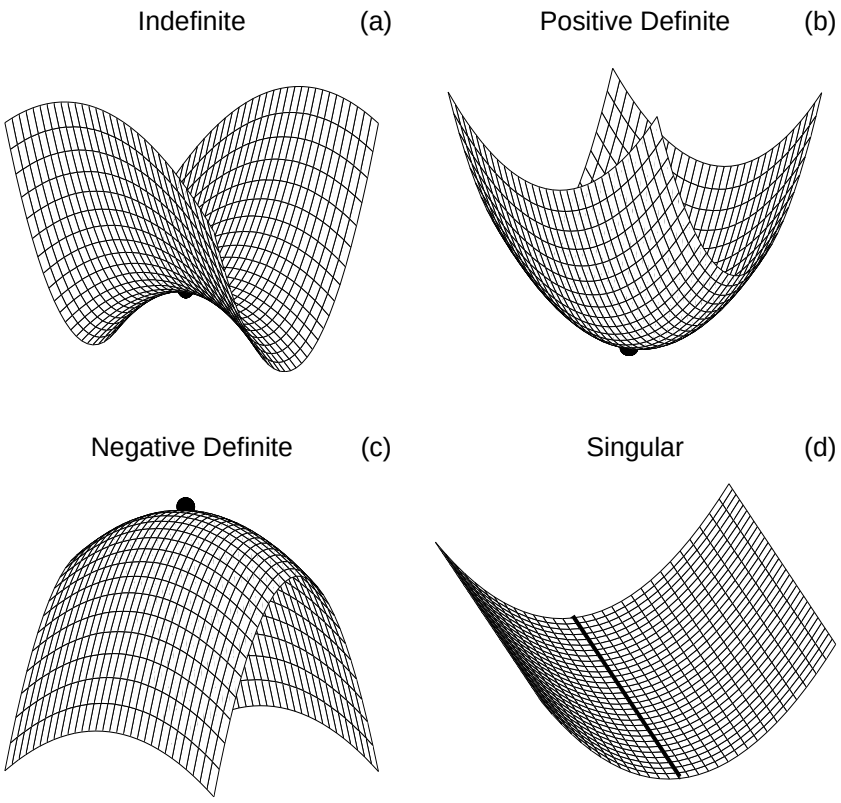

Figure 10.3. Three-dimensional curves for the quadratic functions as described for Figure 10.2. Critical points are shown by thick circles (a–c) and a line (d).

of atoms down one strand and up the complementary strand; the water atoms are numbered following the DNA atoms.

Memory Intensity

Because the formulation of a dense Hessian (n^2 entries) 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.

Exploitation of Derivatives

In most molecular mechanics packages, the second derivatives are programmed, though sparsity (when relevant) is not often exploited in the storage techniques

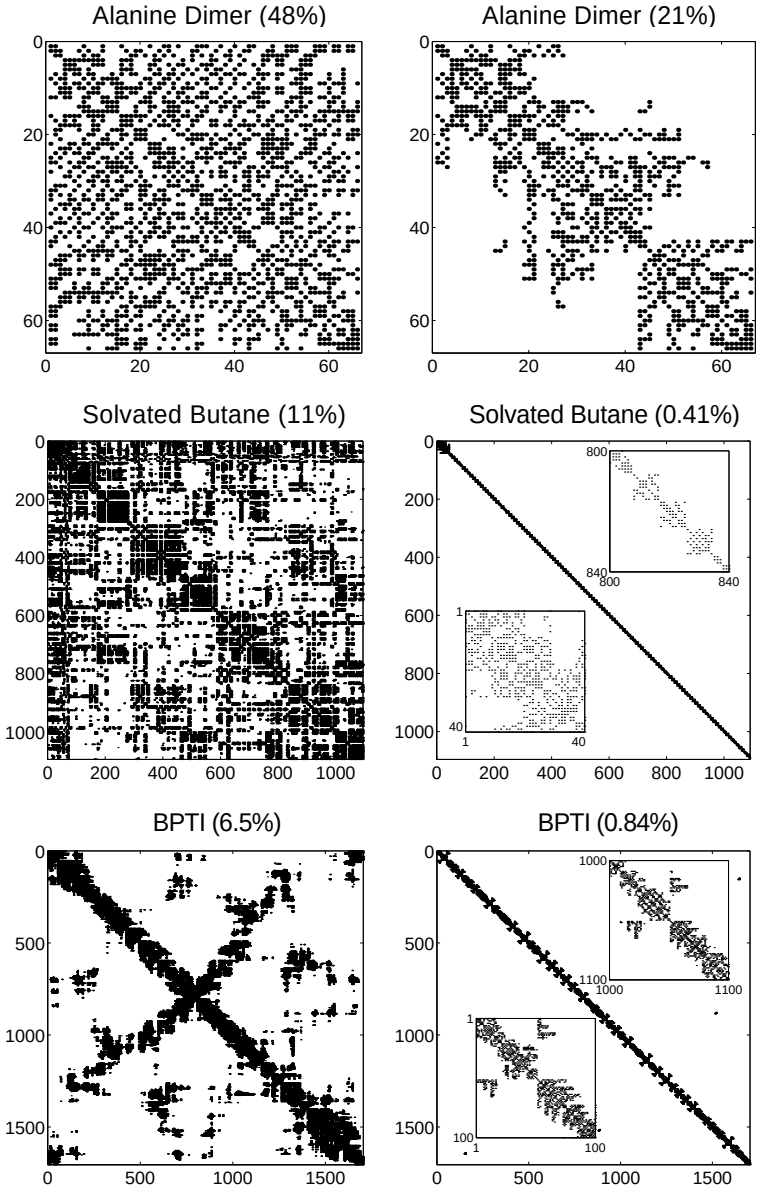

Figure 10.4. Hessian patterns from the potential energy functions of various molecular systems corresponding to 8-Å cutoffs (matrices at left column) or to local terms (right column; bond-length, bond-angle, and dihedral-angle components). The percentage sparsity is shown for each case, and insets show enlargements of some Hessian submatrices. The matrix axes label Cartesian coordinates, i.e., the x, y, z coordinates of each atom in turn; the atom ordering comes from the molecular mechanics package (CHARMM used here).

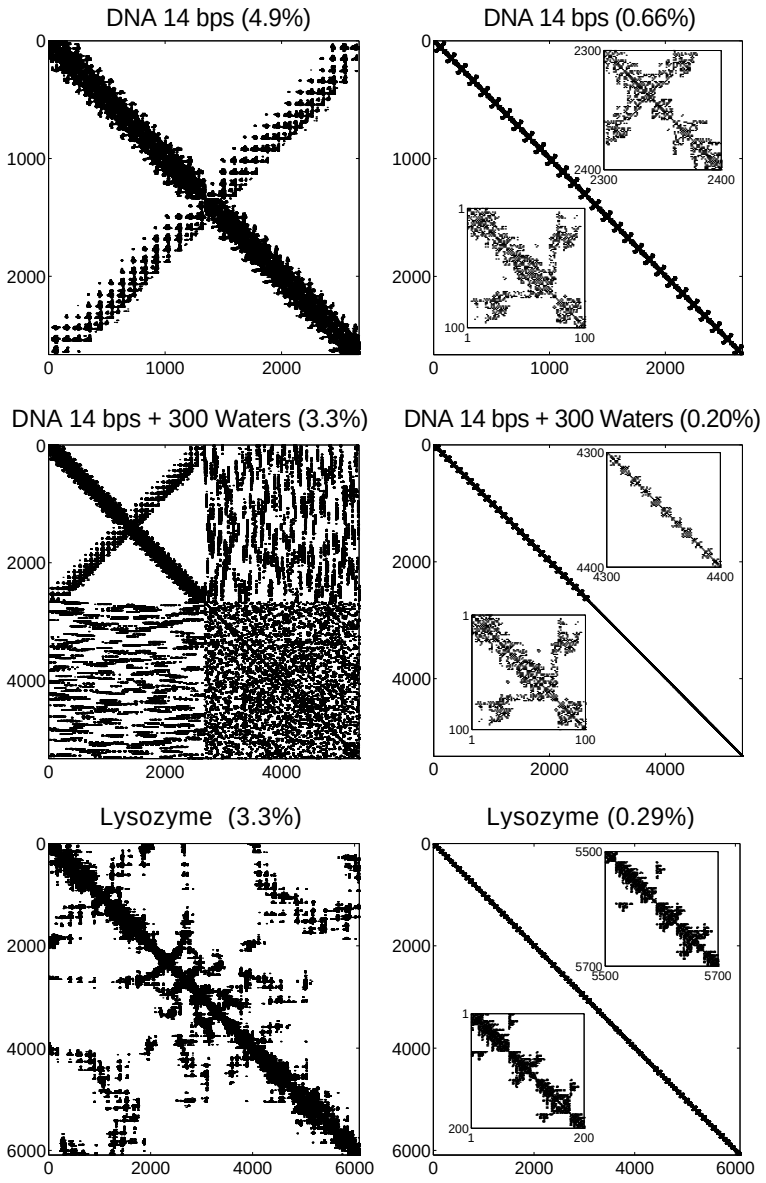

Figure 10.5. Sparse Hessian patterns, continued (see caption to Figure 10.4).

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.

10.3 Basic Algorithmic Components

10.3.1 Greedy Descent

The basic structure of an iterative local optimization algorithm is one of "greedy descent". Namely, a sequence $\{\mathbf{x}_k\}$ is generated from a starting point $\mathbf{x}_0$ in such a way that each iterate attempts to further reduce the value of the objective function.[2]

Two Frameworks

Two algorithmic frameworks are available for such algorithms: 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.

In describing iterative minimization techniques, it is convenient to use short hand notation for quantities used at each step k of the minimization algorithm. Namely, associated with each iterate $\mathbf{x}_k$, we denote the gradient and Hessian at $\mathbf{x}_k$, namely $\mathbf{g}(\mathbf{x}_k)$ and $\mathbf{H}(\mathbf{x}_k)$, as $\mathbf{g}_k$ and $\mathbf{H}_k$. The initial guess for the iterative minimization process ($\mathbf{x}_0$) can be derived from experimental data, where available, or from results of conformational search techniques.

Algorithmic Parameters

The final stopping criteria must be chosen with care to ensure a sufficiently accurate solution and, at the same time, avoid wasting computational effort when further progress is not realized. For example, the norm of the gradient alone (i.e., $\|\mathbf{g}_k\|$) may not be a satisfactory stopping criterion in unconstrained optimization, as it often exhibits oscillations in the course of the optimization [65]; see also Figure 10.11.

The line search framework requires careful implementation of convergence criteria of its own at each step, for a one-dimensional optimization procedure. This segment is a tricky part of minimization methods and requires well tested software with safeguards against many undesirable situations that can occur in practice, like very small steplengths and failure to bracket the univariate minimum (see [16, 66, 89], for example).

We now describe in turn the line search and trust-region frameworks for minimization (Subsections 10.3.2 and 10.3.3); this is followed by a discussion of convergence criteria for the minimization process (Subsection 10.3.4).

[2]This does not imply that the reduction in the gradient norm *is monotonic*; see Figure 10.11 for example.

10.3.2 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 (see subsection 10.3.4).
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, $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{p}_k$, and corresponding gradient $\mathbf{g}_{k+1}$, satisfy:

$$f(\mathbf{x}_{k+1}) \;\leq\; f(\mathbf{x}_k) + \alpha\,\lambda\,\mathbf{g}_k^T\mathbf{p}_k \qquad [\text{"sufficient decrease"}] \tag{10.13}$$

and

$$|\mathbf{g}_{k+1}^T\mathbf{p}_k| \;\leq\; \beta\,|\mathbf{g}_k^T\mathbf{p}_k| \qquad [\text{"sufficient directional derivative reduction"}] \tag{10.14}$$

where $0 < \alpha < \beta < 1$
(e.g., $\alpha = 10^{-4}$, $\beta = 0.9$ in Newton methods).

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.

Step 2: Descent Direction

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

$$\mathbf{g}_k^T\mathbf{p}_k \;<\; 0\,. \tag{10.15}$$

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:

$$\begin{aligned} f(\mathbf{x}+\lambda\mathbf{p}) - f(\mathbf{x}) \quad &= \quad \lambda\,\mathbf{g}(\mathbf{x})^T\mathbf{p} \;+\; \frac{\lambda^2}{2}\,\mathbf{p}^T\mathbf{H}(\mathbf{x})\,\mathbf{p} \\ &\approx \quad \lambda\,\mathbf{g}(\mathbf{x})^T\mathbf{p} \;<\; 0\,, \end{aligned} \tag{10.16}$$

for sufficiently small positive λ.

Steepest Descent

The descent condition is used to define the algorithmic sequence that generates $\mathbf{p}_k$. The simplest way to specify a descent direction is to set

$$\mathbf{p}_k = -\mathbf{g}_k \tag{10.17}$$

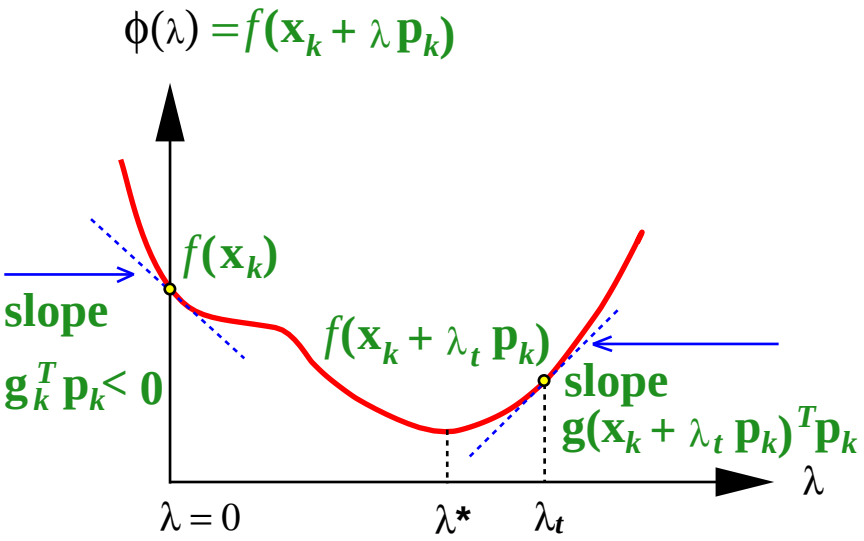

Figure 10.6. One-dimensional line search minimization at step k of the multivariate method, using polynomial approximation to estimate the optimal steplength λ^* in the region $[0, \lambda_t]$.

at each step. This "steepest descent" direction defines the steepest descent (SD) method. SD methods generally lead to improvements quickly but then exhibit slow progress toward a solution. Though it has become customary to recommend the use of SD for initial minimization iterations when the starting function and gradient-norm values are very large, this approach is not necessary when a more robust minimization method is available.

Step 3: The One-Dimensional Optimization Subproblem (Line Search)

The line search procedure, a univariate minimization problem, is typically performed via approximate minimization of a quadratic or cubic polynomial interpolant of the one-dimensional function of λ (given $\mathbf{x}_k$ and $\mathbf{p}_k$):

$$\phi(\lambda) \equiv f(\mathbf{x}_k + \lambda \mathbf{p}_k) \, .$$

See Figure 10.6 for an illustration of the first step of this univariate minimization process, where the minimum (λ^* in figure) is sought in the initial interval $[0, \lambda_t]$ where $f(\mathbf{x}_k) = \phi(0)$ and $f(\mathbf{x}_k + \lambda_t \mathbf{p}_k) = \phi(\lambda_t)$. For details, consult standard texts (e.g., [66]). This iteration process is generally continued until the λ value that minimizes the polynomial interpolant of $\phi(\lambda)$ satisfies the line search criteria (eqs. (10.13) and (10.14)). The resulting steplength λ_k defines the next iterate for minimization of the multivariate function by $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \, \mathbf{p}_k$.

The line search criteria in Step 3 of Algorithm [A1] have been formulated to ensure sufficient decrease of f relative to the size of step (λ) taken. The first con-

dition (eq. (10.13)) 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, eq. (10.14), imposes a lower bound on λ. The control parameters α and β determine the balance between the computational work performed in the line search and the reduction in function achieved. (See [77] for illustrations and further discussion). The work in the line search (number of polynomial interpolations) should be balanced with the overall progress realized in the minimization algorithm.

10.3.3 Trust-Region-Based Descent Algorithm

Algorithm [A2]: Basic Descent By 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 (see subsection 10.3.4).
2. Calculate a step $\mathbf{s}_k$ by solving the subproblem

$$\min_{\mathbf{s}} \{q_k(\mathbf{s})\}\,, \tag{10.18}$$

where q_k is the quadratic model of the objective function:

$$q_k(\mathbf{s}) \;=\; f(\mathbf{x}_k) \;+\; \mathbf{g}_k^T\mathbf{s} \;+\; \frac{1}{2}\,\mathbf{s}^T\,\mathbf{H}_k\,\mathbf{s}\,, \tag{10.19}$$

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

$$\|\,\mathbf{D}_k\mathbf{s}\,\| \;<\Delta_k\,, \tag{10.20}$$

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.

Basic Idea

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:

$$\rho_k \;=\; \frac{f(\mathbf{x}_k) \;-\; f(\mathbf{x}_k + \mathbf{s}_k)}{f(\mathbf{x}_k) \;-\; q_k(\mathbf{s}_k)}\,. \tag{10.21}$$

A value near unity implied that the bound Δ_k imposed on $\mathbf{s}$ can be increased; in contrast, a negative or a small positive 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

$$\mathbf{H}_k \, \mathbf{s} \; = \; -\mathbf{g}_k \tag{10.22}$$

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 [14].

10.3.4 Convergence Criteria

The criteria used to define convergence of the minimization algorithm (in step 1 of Algorithms [A1] or [A2]) 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. A well-structured algorithm should halt the iteration process when progress is poor.

The gradient norm value, together with measures of progress in the function values and the independent variables (e.g., eqs. (10.25) and (10.26)) are used to assess performance. Upper limits for the total number of allowable function and/or gradient evaluations are important safeguards against wasteful computing cycles.

Specifically, reasonable tests for the size of the gradient norm are:

$$\| \mathbf{g}_k \| \; \leq \epsilon_g \, (1 + \mid f(\mathbf{x}_k) \mid) \tag{10.23}$$

or

$$\| \mathbf{g}_k \| \; \leq \epsilon_g \; \max (1, \, \| \, \mathbf{x}_k \, \|) \, , \tag{10.24}$$

where the gradient norm may be set to the Euclidean norm divided by $\sqrt{n}$ (this introduces a dependency on the number of variables). The parameter ϵ_g is a small positive number such as 10^{-6} that might depend on the *machine precision*, ϵ_m; ϵ_m is roughly the largest number for which $1 + \epsilon_m = 1$ in computer representation. The student is encouraged to code a routine for determining ϵ_m.[3]

For example, our truncated-Newton package TNPACK [81, 79, 80] checks the following four conditions at each iteration:

$$f(\mathbf{x}_{k-1}) - f(\mathbf{x}_k) \quad < \quad \epsilon_f \, (1 + |f(\mathbf{x}_k)|) \, , \tag{10.25}$$

$$\| \, \mathbf{x}_{k-1} - \mathbf{x}_k \, \| \quad < \quad (\epsilon_f)^{1/2} \, (1 + \| \, \mathbf{x}_k \, \|) \, , \tag{10.26}$$

$$\| \, \mathbf{g}_k \, \| \quad < \quad (\epsilon_f)^{1/3} \, (1 + |f(\mathbf{x}_k)|) \, , \tag{10.27}$$

$$\| \, \mathbf{g}_k \, \| \quad < \quad \epsilon_g \, (1 + |f(\mathbf{x}_k)|) \, . \tag{10.28}$$

[3] Typically, ϵ_m is 10^{-15} and 10^{-7}, respectively, for double and single-precision IEEE arithmetic [68].

Here, all norms are the Euclidean norm divided by $\sqrt{n}$, and ϵ_f and ϵ_g are small numbers (like 10^{-10} and 10^{-8}, respectively). If the first three conditions above are satisfied, or the fourth condition alone is satisfied, convergence is considered to have been satisfied and the minimization process is halted; otherwise, the loop continues. Note that the first and second conditions test for convergence the sequences of the function values and iterates of the independent variables, respectively, while the third and fourth conditions test the size of the gradient norm.

Box 10.2: Historical Perspective of 'Newton's' Method

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. Viète's work itself had precursors in the 11th-century works of Arabic algebraists.

In 1687, three years after Newton described a root finder for a polynomial, Newton 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.

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

Simpson in 1740 was first to formulate the Newton-Raphson method on the basis of calculus. He applied this iterative scheme to solve 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 the nonlinear solver can be applied to optimization by setting the gradient to zero.

Finally, Fourier in 1831 published the modern version of the method as we know it today in his celebrated book *Analyse des Équations Determinées*. The method for solving $f(x) = 0$ was simply written as: $x_{k+1} = x_k - f(x_k)/f'(x_k)$. Unfortunately, Fourier omitted credits to either Raphson or Simpson, possibly explaining the method's name.

Thus, strictly speaking, it is appropriate to title the method as the Newton-Raphson-Simpson-Fourier method.

10.4 The Newton-Raphson-Simpson-Fourier 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.

Though the method is credited to Newton or Newton and Raphson, key contributions were made also by Fourier and Simpson; see Box 10.2 for a historical perspective. For brevity, we refer to the "Newton-Raphson-Simpson-Fourier method" as Newton's method.

A Fundamental Optimization Tool

Many effective methods for nonlinear, multivariate minimization can be related to Newton's method. Hence, a good understanding of the Newton solver, including performance and convergence behavior, is invaluable for applying 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. The section that follows continues by describing the effective variants known as quasi-Newton, nonlinear conjugate gradient, and truncated-Newton methods.

10.4.1 The One-Dimensional Version of Newton's Method

Iterative Recipe

The modern version of Newton's method (see Box 10.2) for solving $f(x) = 0$ is:

$$x_{k+1} = x_k - f(x_k)/f'(x_k)\,. \tag{10.29}$$

This iterative scheme can be derived easily by using a Taylor expansion to approximate a twice-differentiable function f locally by a quadratic function about x_k:

$$f(x_{k+1}) = f(x_k) + (x_{k+1} - x_k)\,f'(x_k) + \frac{1}{2}(x_{k+1} - x_k)^2\,f''(\xi) \tag{10.30}$$

where $x_k \le \xi \le x_{k+1}$. Omitting the second-derivative term, the solution of $f(x_{k+1}) = 0$ yields the iteration process of eq. (10.29). The related *discrete-Newton* and *quasi-Newton* methods [66] correspond to approximating $f'(x)$ by *finite-differences*, as

$$f'(x_k) \approx [f(x_k + h) - f(x_k)]\,/\,h\,, \tag{10.31}$$

or by the method of *secants*

$$f'(x_k) \approx [f(x_k) - f(x_{k-1})]\,/\,(x_k - x_{k-1})\,, \tag{10.32}$$

where h is a suitably-chosen small number.

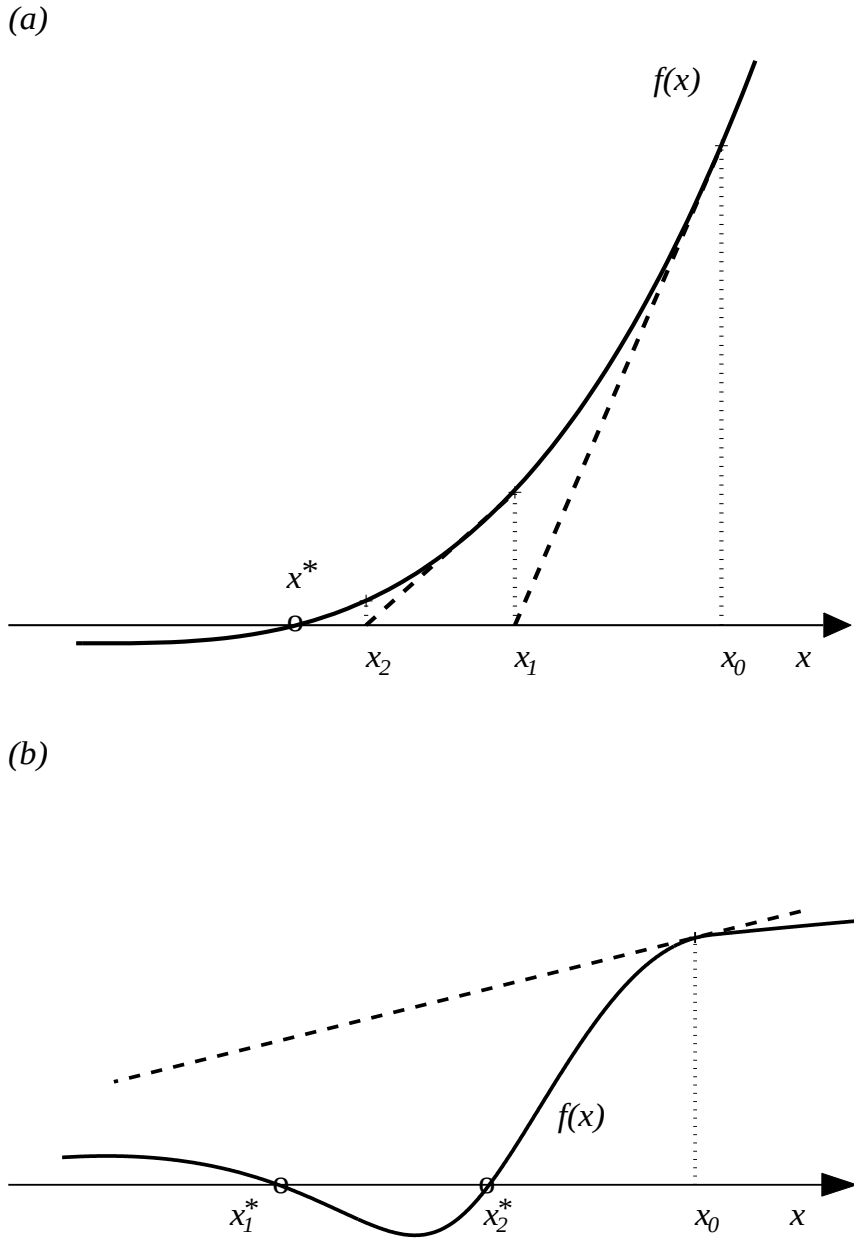

Figure 10.7. Newton's method in one dimension: (a) geometric interpretation and behavior in the ideal case (rapid convergence), and (b) divergent behavior near point where $f'(x) = 0$.

Geometric Interpretation

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 10.7).

Performance

The method works well in the ideal case (Figure 10.7a), when x_0 is near the solution (x^*) and $|f'(\xi)| \geq M > 0$ nearby.

However, difficulties arise when x_0 is far from the solution, or when $f'(x)$ is close to zero (Figure 10.7b). Further difficulties emerge when $f'(x)$ is zero at the solution.

Note that the Newton iteration process is undefined when $f'(x) = 0$ and can exhibit poor numerical behavior when $|f'(x)|$ is very small, as shown in Figure 10.7b. In general, both performance and attainable accuracy of the solver worsen if any of the above complications arise.

A simple example for solving for the square root of a number by Newton's method in Box 10.3 (with associated data in Figure 10.8) illustrates the rapid convergence for the ideal case when the root of $f(x)$ is simple and reasonably separated from the other root. In the non-ideal case (e.g., x_0 far from the solution or $f'(x)$ close to zero), convergence is slow at the beginning but then improves rapidly until the region of *quadratic convergence* is approached. (See Box 10.1 for a definition of quadratic convergence).

The quadratic convergence of Newton's method for a simple root and for x_0 sufficiently close to the solution x^* can easily be shown on the basis of the Taylor expansion. Tensor methods based on fourth-order approximations to the objective function can achieve more rapid convergence [83], but they are not generally applicable. The attainable accuracy for Newton's method depends on the function characteristics, and on whether the root is simple or not.

Box 10.3: Newton's Method: Simple Examples

We can apply Newton's method to solve for the square root of a number a by defining

$$f(x) = x^2 - a = 0;$$

the resulting iterative scheme for computing $x = \sqrt{a}$ is:

$$x_{k+1} = \frac{1}{2}\left[x_k + \frac{a}{x_k}\right], \tag{10.33}$$

defined for $x_k \neq 0$. A computer result from a double-precision program is shown in Figure 10.8 for $a = 0.01$ with four starting points: 5, -100, 1000, and 10^{-6}.

The rapid, *quadratic* convergence (see Box 10.1) can be noted in all cases at the last 3–4 steps. In these steps, the number of correct digits for the solution is approximately doubled from one step to the next! Note the larger number of iterations for convergence when x_0 is near zero ($x_0 = 10^{-6}$ shown). Since the derivative of the objective function is zero at

Figure 10.8. Computer output from the application of Newton's method to solve the simple quadratic $x^2 = a$, $a = 0.01$, from four different starting points.

	Newton iterate, x	Error: $\lvert x - x^* \rvert / x^*$ (for nonzero x^*)
	$x_0 = 5$	
0	5.000000000000000	49.00000000000000
1	2.501000000000000	24.01000000000000
2	1.252499200319872	11.52499200319872
3	0.6302416186767726	5.302416186767727
4	0.3230542746187304	2.230542746187304
5	0.1770044127792565	0.7700441277925646
6	0.1167500897135064	0.1675008971350636
7	0.1012015644103529	1.201564410352878 9E–02
8	0.1000071330766507	7.133076507338161E–05
9	0.1000000002543858	2.543857624548512E–09
10	0.1000000000000000	0.0000000000000000
	$x_0 = -100$	
0	−100.0000000000000	999.0000000000000
1	−50.00005000000000	499.0005000000000
2	−25.00012499990000	249.0012499990000
3	−12.50026249895001	124.0026249895001
4	−6.250531241075214	61.50531241075214
5	−3.126065552544528	30.26065552544528
6	−1.564632230895322	14.64632230895322
7	−0.7855117545897353	6.855117545897353
8	−0.3991211544161648	2.991211544161648
9	−0.2120881016068179	1.120881016068179
10	−0.1296191592706879	0.2961915927068787
11	−0.1033841239244204	3.384123924420362 5E–02
12	−0.1000553871053945	5.538710539446101 1E–04
13	−0.1000000153301663	1.533016627530692 2E–07
14	−0.1000000000000012	1.165734175856414 4E–14
15	−0.1000000000000000	0.0000000000000000
	$x_0 = 1000$	
0	1000.000000000000	9999.000000000000
1	500.0000050000000	4999.000050000000
2	250.0000124999999	2499.000124999999
3	125.0000262499989	1249.000262499989
4	62.50005312499107	624.0005312499106
5	31.25010656242754	311.5010656242753
6	15.62521328066817	155.2521328066817
7	7.812926635966156	77.12926635966156
8	3.907103283034968	38.07103283034968
9	1.954831361974763	18.54831361974762
10	0.9799734463768975	8.799734463768974
11	0.4950889022510404	3.950889022510404
12	0.2576436474057566	1.576436474057566
13	0.1482284733538422	0.4822847335384220
14	0.1078459475072978	7.845947507297779 1E–02
15	0.1002854019724900	2.854019724899958 8E–03
16	0.1000004061123768	4.061123767690189 0E–06
17	0.1000000000008246	8.246181515403350 2E–12
18	0.1000000000000000	0.0000000000000000
	$x_0 = 10^{-6}$	
0	9.999999999999995E–07	0.9999900000000000
1	5000.000000500000	49999.00000499999
2	2500.000001250000	24999.00001250000
3	1250.000002625000	12499.00002625000
4	625.0000053125000	6249.000053124999
5	312.5000106562499	3124.000106562499
6	156.2500213281244	1561.500213281244
7	78.12504266405783	780.2504266405783
8	39.06258533199397	389.6258533199396
9	19.53142066571737	194.3142066571736
10	9.765966330621753	96.65966330621752
11	4.883495147415947	47.83495147415947
12	2.442771430566446	23.42771430566446
13	1.223432570726772	11.23432570726771
14	0.6158031472956539	5.158031472956539
15	0.3160210514743984	2.160210514743984
16	0.1738322565259314	0.7383225652593136
17	0.1156794895626801	0.1567948956268007
18	0.1010626187661944	1.062618766194428 6E–02
19	0.1000055864307498	5.586430749826565 3E–05
20	0.1000000001560323	1.560323259486295 9E–09
21	0.1000000000000000	0.0000000000000000

$x = 0$, the tangent takes the iterates very far, as illustrated in Fig. 10.7b, but then works back systematically toward the solution.

Figure 10.9 further illustrates the notions of of accuracy, convergence, and problem conditioning for solving

$$f(x) = x^3 - bx = 0, \quad b > 0,$$

by Newton's method. The three roots are $-\sqrt{b}$, 0, and $+\sqrt{b}$. The corresponding iterative scheme for solving this cubic polynomial of x becomes:

$$x_{k+1} = 2x_k^3/(3x_k^2 - b). \tag{10.34}$$

Near the point where the Newton iteration is undefi ned ($x_k = \sqrt{b/3}$), the iterative process converges very slowly. When b is small, as in our example ($b = 0.0001$), the three roots are relatively close. The Newton iterates in Figure 10.9 started from -50, $\sqrt{b/3} + 10^{-10}$, -1, 10^{-10}, 0.009, and 0.011 show that the solution obtained depends on the starting point.

Figure 10.9. Computer output from the application of Newton's method to solve $x^3 - bx = 0, b = 10^{-4}$, from various starting points.

Newton iterate, x	Error: $\lvert x - x^* \rvert / x^*$ (for nonzero x^*)
$x_0 = -50$	
0 −50.00000000000000	4999.000000000000
1 −33.33333377777778	3332.333377777778
2 −22.22222318518520	2221.222318518520
3 −14.81481645679016	1480.481645679016
4 −9.876545804526833	986.6545804526833
5 −6.584366119684733	657.4366119684732
6 −4.389580788123710	437.9580788123710
7 −2.926392254584279	291.6392254584279
8 −1.950935763478845	194.0935763478845
9 −1.300635232964303	129.0635232964302
10 −0.8671072413145484	85.71072413145484
11 −0.5780971233434421	56.80971233434420
12 −0.3854365263554411	37.54365263554411
13 −0.2570153518629270	24.70153518629269
14 −0.1714300741869435	16.14300741869435
15 −0.1144164918146208	10.44164918146208
16 −7.647237920628794E−02	6.647237920628794
17 −5.127384346875718E−02	4.127384346875972
18 −3.462153071898276E−02	2.462153071898276
19 −2.374124195754244E−02	1.374124195754924
20 −1.682234658330321 7E−02	0.6822346583303216
21 −1.271226575066434 5E−02	0.2712265750664345
22 −1.067721687917391 9E−02	6.772168791739190 1E−02
23 −1.005941870836113 9E−02	5.941870836113916 1E−03
24 −1.000052234645395 9E−02	5.223464539585998 0E−05
25 −1.000000004092188 4E−02	4.092188393700624 3E−09
26 −1.000000000000000	0.0000000000000000
$x_0 = -1$	
0 −1.000000000000000	99.00000000000000
1 −0.666688889629654 3	65.66888896296543
2 −0.444492594475262 6	43.44925944752625
3 −0.296378399336731 7	28.63783993367316
4 −0.197660607244950 7	18.76606072449507
5 −0.131886260320257 0	12.18862603202570
6 −8.809299242124053 7E−02	7.809299242124054
7 −5.898200850405330 2E−02	4.898200850405330
8 −3.970174665456611 3E−02	2.970174665456611
9 −2.703965279816382 4E−02	1.703965279816382
10 −1.888753150343657 3E−02	0.8887531503436572
11 −1.388951007914815 4E−02	0.3889510079148153
12 −1.119378671795170 9E−02	0.1193786717951709
13 −1.016729236823979 4E−02	1.672923682397940 0E−02
14 −1.000404035769627 1E−02	4.040357696270466 3E−04
15 −1.000000244636725 5E−02	2.446367254604897 6E−07
16 −1.000000000000089 9E−02	8.985867605559860 7E−14
17 −1.000000000000000 2E−02	1.734723475976807 1E−16
$x_0 = 10^{-10}$	
0 1.000000000000000 0E−10	
1 −2.000000000000000 6E−26	
2 1.600000000000012E−73	
3 −8.192000000000017 8E−215	
4 0.0000000000000000	
$x_0 = .009$	
0 8.999999999999999 3E−03	0.1000000000000001
1 1.019580419580419 7E−02	1.958041958041965 8E−02
2 1.000549738015171E−02	5.499738015170632 7E−04
3 1.000000453125297 9E−02	4.531252978737243 5E−07
4 1.000000000003079E−02	3.079134169858832 6E−13
5 9.999999999999985E−03	1.734723475976807 1E−16

Newton iterate, x	Error: $\lvert x - x^* \rvert / x^*$ (for nonzero x^*)
$x_0 = .0058 + 10^{-10}$	
0 5.773502791896257 6E−03	0.4226497208103743
1 111111.1159104916	11111110.59104916
2 74074.07727366130	7407406.727366131
3 49382.71818244116	4938270.818244115
4 32921.81212162789	3292180.212162788
5 21947.87474775260	2194786.474775260
6 14631.91649850275	1463190.649850275
7 9754.610999003351	975460.0999003351
8 6503.073999337846	650306.3999337845
9 4335.382666228647	433537.2666228647
10 2890.255110824224	289024.5110824224
11 1926.836740557172	192682.6740557172
12 1284.557827049647	128454.7827049647
13 856.3718847170644	85636.18847170644
14 570.9145898373255	57090.45898373255
15 380.6097265971409	38059.97265971409
16 253.7398177898131	25372.98177898131
17 169.1598786141209	16914.98786141209
18 112.7732525407821	11276.32525407821
19 75.18216855757360	7517.216855757360
20 50.12144600062744	5011.144600062744
21 33.41429777711917	3340.429777711918
22 22.27619918313077	2226.619918313077
23 14.85080045299750	1484.080045299750
24 9.900535131697179	989.0535131697179
25 6.600358999013134	659.0358999013134
26 4.400242699498232	439.0242699498232
27 2.933500183234001	292.3500183234001
28 1.955674364178697	194.5674364178697
29 1.303794272497512	129.3794272497512
30 0.8692132262697133	85.92132262697133
31 0.5795010512121747	56.95010512121746
32 0.3863723851127763	37.63723851127763
33 0.2576391179580148	24.76391179580147
34 0.1718457086006079	16.18457086006078
35 0.1146932668316612	10.46932668316612
36 7.665642360524273 2E−02	6.665642360524274
37 5.139583003838754E−02	4.139583003838875
38 3.470178655689871 3E−02	2.470178655689871
39 2.379313189699958E−02	1.379313189699986
40 1.685449843973348 5E−02	0.6854498439733484
41 1.273008374162922 3E−02	0.2730083741629223
42 1.068441528897420 7E−02	6.844152889742063 9E−02
43 1.006060094277894 3E−02	6.060094277894265 1E−03
44 1.000054319139779 1E−02	5.431913977903962 7E−05
45 1.000000004425292 4E−02	4.425292424170557 1E−09
46 1.000000000000000 0E−02	0.0000000000000000
$x_0 = .011$	
0 1.099999999999999 9E−02	9.999999999999908E−02
1 1.012167300380228 3E−02	1.216730038022823 7E−02
2 1.000215936055531 7E−02	2.159360553163475E−04
3 1.000000069907347 3E−02	6.990734727208014 5E−08
4 1.000000000000007 3E−02	7.285838599102589 8E−15
5 9.999999999999985E−03	1.734723475976807 1E−16

10.4.2 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:

$$f(x_{k+1}) \approx f(x_k) + (x_{k+1} - x_k) f'(x_k) + \frac{1}{2}(x_{k+1} - x_k)^2 f''(x_k) . \quad (10.35)$$

Since $f(x_k)$ is constant, minimization of the second and third terms on the right-hand-side in eq. (10.35) yields the iteration process:

$$x_{k+1} \;=\; x_k \;-\; f'(x_k)/f''(x_k)\,. \tag{10.36}$$

Thus, we have replaced f and f' of eq. (10.29) by f' and f'', respectively. This Newton scheme for minimizing $f(x)$ is defined as long as the second derivative at x_k is nonzero.

10.4.3 The Multivariate Version of Newton's Method

We generalize Newton's method for minimization in eq. (10.36) to multivariate functions by expanding $f(\mathbf{x})$ locally along a search vector $\mathbf{p}$ (in analogy to eq. (10.35)):

$$f(\mathbf{x}_k + \mathbf{p}_k) \;\approx\; f(\mathbf{x}_k) \;+\; \mathbf{g}(\mathbf{x}_k)^T\mathbf{p}_k \;+\frac{1}{2}\,\mathbf{p}_k^T\mathbf{H}(\mathbf{x}_k)\,\mathbf{p}_k\,. \tag{10.37}$$

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

$$\mathbf{H}_k\,\mathbf{p}_k \;=\; -\mathbf{g}_k\,. \tag{10.38}$$

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

$$\mathbf{x}_{x+1} \;=\; \mathbf{x}_k \;-\; \mathbf{H}_k^{-1}\,\mathbf{g}_k\,. \tag{10.39}$$

Thus, the search vector

$$\mathbf{p}_k = -\mathbf{H}_k^{-1}\,\mathbf{g}_k \tag{10.40}$$

is used at each step of the *classic* Newton method for minimization. This requires repeated solutions of a linear system involving the Hessian. 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 large maximal-to-minimal eigenvalue ratio $\lambda_{\max}/\lambda_{\min}$) 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. Such effective strategies are described in the next section. These include quasi-Newton (QN), nonlinear conjugate gradient (CG), and truncated Newton (TN) methods.

10.5 Effective Large-Scale Minimization Algorithms

The popular methods that fit the descent framework outlined in subsections 10.3.2 and 10.3.3 require gradient information. In addition, the truncated-Newton (TN)

method may require more input to be effective, such as second-derivative information from components of the objective function that can be computed cheaply.

The steepest descent method (recall $\mathbf{p}_k = -\mathbf{g}_k$) can be viewed as a simple version of the $\mathbf{p}_k$ definition in eq. (10.40) in which the Hessian replaced by the identity matrix.

Nonlinear CG methods improve upon (the generally poor) convergence of SD methods by using better search directions than SD that are still cheap to compute.

QN methods, which are closely related to nonlinear CG methods, can also be presented as robust alternatives to the classic Newton method ($\mathbf{p}_k$ by eq. (10.40)) which update curvature information as the algorithm proceeds.

TN methods are another clever and robust alternative to the classic Newton framework that introduce curvature information only when locally warranted, so as to balance computation with realized convergence. Hybrid schemes have also been devised, e.g., limited-memory QN with TN [12].

The methods described in turn in this section — QN, nonlinear CG, and TN methods — render SD obsolete as a general method.

10.5.1 Quasi-Newton (QN)

Basic Idea

QN methods avoid using the actual Hessian and instead build-up curvature information as the algorithm proceeds [66, 28]. Actually, it is often the Hessian inverse ($\widehat{\mathbf{B}}$) that is updated in practice so that a term $\widehat{\mathbf{B}}_k\,\mathbf{g}_k$ replaces $\mathbf{H}_k^{-1}\,\mathbf{g}_k$ in eq. (10.39). Here $\widehat{\mathbf{B}}_k$ is short hand for $\widehat{\mathbf{B}}(\mathbf{x}_k)$.

The Hessian approximation $\mathbf{B}_k$ is derived to satisfy the *quasi-Newton condition* (see below). QN variants define different formulas that satisfy this condition.

Because 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. In practice, $\mathbf{B}_k$ is updated by adding a *low rank* update matrix $\mathbf{U}_k$.

Recent Advances

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., $m = 5$) [62, 54, 26, 66]. The second is the emergence of insightful analyses that explain the relationship between QN and nonlinear CG methods.

QN Condition

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

$$\mathbf{B}_{k+1}\,\mathbf{s}_k \;=\; \mathbf{y}_k\,. \tag{10.41}$$

Here

$$\mathbf{s}_k \;=\; \mathbf{x}_{k+1} \;-\; \mathbf{x}_k \tag{10.42}$$

and

$$\mathbf{y}_k \;=\; \mathbf{g}_{k+1} \;-\; \mathbf{g}_k\,. \tag{10.43}$$

(Note that the 'step vector' $\mathbf{s}_k$ can be equated with the displacement from $\mathbf{x}_k$, namely $\lambda_k\,\mathbf{p}_k$, used in the basic Algorithm [A1]). 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:

$$\mathbf{g}_{k+1} \;-\; \mathbf{g}_k \;=\; \mathbf{H}\,(\mathbf{x}_{k+1} \;-\; \mathbf{x}_k)\,. \tag{10.44}$$

This equation makes clear the origin of the QN condition of eq. (10.41).

Updating Formula

The updating QN formula can be written symbolically as:

$$\mathbf{B}_{k+1} \;=\; \mathbf{B}_k \;+\; \mathbf{U}_k(\mathbf{s}_k,\,\mathbf{y}_k,\,\mathbf{B}_k) \tag{10.45}$$

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: $\mathbf{u}\mathbf{v}^T$. In addition to rank, imposed symmetry and positive-definiteness are used in the formulation of $\mathbf{U}_k$.

BFGS Method

One of the most successful QN formulas in practice is associated with the BFGS method (for its developers Broyden, Fletcher, Goldfarb, 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 $\mathbf{y}_k^T\mathbf{s}_k > 0$. This condition is satisfied automatically for convex functions but may not hold in general without the sufficient reduction of curvature criteria (eq. (10.14)) in the line search. 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

$$\mathbf{B}_{k+1} \;=\; \mathbf{B}_k \;-\; \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^T\mathbf{B}_k^T}{\mathbf{s}_k^T\mathbf{B}_k\mathbf{s}_k} \;+\; \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}\,. \tag{10.46}$$

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

$$\widehat{\mathbf{B}}_{k+1} \;=\; \left(\mathbf{I} \;-\; \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) \widehat{\mathrm{B}}_k \left(\mathbf{I} \;-\; \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\right) \;+\; \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}\,. \tag{10.47}$$

From this $\widehat{\mathbf{B}}$, the BFGS search vector is defined as

$$\mathbf{p}_k = -\widehat{\mathbf{B}}_k\, \mathbf{g}_k\,; \tag{10.48}$$

(compare to the Newton search vector defined in eq. (10.40)).

Practical Implementation

Because we only require 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 by using low-rank QN updates. This requires $\mathcal{O}(n)$ memory to store the successive pairs of update vectors ($\mathbf{s}_k$ and $\mathbf{y}_k$) and the respective inner products $\mathbf{y}_k^T \mathbf{s}_k$.

Limited-memory QN methods reduce storage requirements further by only retaining the $\{\mathbf{s},\, \mathbf{y}\}$ pairs from the previous few iterates (3–7). The identity matrix, $\mathbf{I}$, or a multiple of it, is typically used for the initial Hessian approximation $\mathbf{B}_0$. Updating this scaling at each iteration enhances overall efficiency [54, 26].

The limited-memory BFGS code of Nocedal and co-workers [64] 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 [61]. The method has been extended to constrained optimization [11, 10, 90], used to propose preconditioners for CG methods [59], and combined with TN methods in a QN/TN cyclic fashion [12]. The text of Nocedal and Wright [66] presents a comprehensive description of the limited-memory BFGS method.

10.5.2 Conjugate Gradient (CG)

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 [30]) 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 (eq. (10.38)) for the search vector $\mathbf{p}$ when $\mathbf{H}$ is a constant positive-definite matrix.

CG Search Vector

In each step of the nonlinear CG method, a search vector $\mathbf{p}_k$ 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 $\{\mathbf{p}_k\}$ is given by:

$$\mathbf{p}_{k+1} \;=\; -\mathbf{g}_{k+1} \;+\; \beta_{k+1}\,\mathbf{p}_k\,, \tag{10.49}$$

where $\mathbf{p}_0 = -\mathbf{g}_0$. The scheme-dependent parameter β_k that defines the search vectors is chosen so that if f were a convex quadratic and the line search exact (i.e., $\mathbf{x}_k + \lambda_k \mathbf{p}_k$ minimizes f exactly along $\mathbf{p}_k$), 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 that two sets of vectors ($\{\mathbf{g}\}$ and $\{\mathbf{p}\}$) generated in the CG method satisfy

$$\mathbf{g}_k^T \mathbf{p}_j = 0 \quad \text{for all} \quad j < k\,.$$

This orthogonality condition implies that the search vectors span the entire n-dimensional space after n steps, so that $\mathbf{g}_{n+1} = 0$ in finite arithmetic.

CG Variants

Different formulas for β_k (not to be confused with the line search parameter introduced earlier) have been developed for the nonlinear CG 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 (FR), Polak-Ribi`ere (PR), and Hestenes-Stiefel (HS). They are defined by the parameter β (for eq. (10.49)) as:

$$\beta_{k+1}^{\mathrm{FR}} \;=\; \mathbf{g}_{k+1}^T \mathbf{g}_{k+1} \,/\, \mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k\,, \tag{10.50}$$

$$\beta_{k+1}^{\mathrm{PR}} \;=\; \mathbf{g}_{k+1}^T \mathbf{y}_k \,/\, \mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k\,, \tag{10.51}$$

$$\beta_{k+1}^{\mathrm{HS}} \;=\; \mathbf{g}_{k+1}^T \mathbf{y}_k \,/\, \mathbf{p}_k^{\mathrm{T}} \mathbf{y}_k\,. \tag{10.52}$$

(Recall $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$).

The PR version is often found in software packages. Still, to be effective PR restarts the iteration process, i.e., sets β_k to zero occasionally, for example when β_k becomes negative.

Some important modifications of this version are due to Powell [75], available in the IMSL library, and to Shanno & Phua [85], available in the NAG library. These modifications have slightly more memory requirements but fewer function evaluations. An interesting CG–PR–FR hybrid algorithm might also be an effective alternative [27].

A careful line search is important for nonlinear CG methods.

CG/QN Connection

Key connections between CG and QN-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 QN condition, and thus an appropriate formula for β_k could be obtained from both viewpoints.

The many developments in the 1980s have shown that the limited-memory QN class of algorithms balances the extremely modest storage requirements of nonlinear CG with good convergence properties in practice. The fact that the unit steplength in QN 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 research monograph of [1] 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*.

10.5.3 Truncated-Newton (TN)

Approximate Solution of the Newton Equations

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 [15]? In the context of large-scale nonlinear optimization, an accurate solution of eq. (10.38) is not warranted! Far away from a minimum, any descent direction that can be computed cheaply may still produce 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

$$r_k \equiv \|\mathbf{H}_k\mathbf{p}_k + \mathbf{g}_k\| \leq \eta_k\|\mathbf{g}_k\|\,, \tag{10.53}$$

where η_k is the *forcing sequence*.

This condition on the size of the residual 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 $\|\mathbf{g}_k\| \to 0$ [15]. For example, an effective setting is:

$$\eta_k = \min\{c_r/k\,,\, \|\mathbf{g}_k\|\}\,, \qquad 0 < c_r \leq 1\,. \tag{10.54}$$

This choice forces the residuals to be progressively 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 [61].

Truncated Outer Iteration; Effective Residual

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 eq. (10.38) at each outer step k.

The linear CG method is an excellent candidate since it is simple and very modest in memory. The linear CG algorithm mirrors in structure the general descent method of Algorithm [A1]. That is, it generates search vectors $\{\mathbf{p}_k^1, \mathbf{p}_k^2, \cdots\}$ at each step recursively (as the nonlinear conjugate gradient method of the previous subsection) until the residual (eq. 10.54), or another suitable truncation criterion, is satisfied for the jth iterate $\mathbf{p}_k^j$. However, in place of the line search, an explicit formula for the steplength is used. This expression is derived analytically by minimizing the quadratic model at the current point along $\mathbf{p}_k^j$ and then using the conjugacy condition to simplify the formula.

Preconditioning

To accelerate convergence of this inner iteration process, *preconditioning* is essential in practice. This technique involves modification of eq. (10.38) through application of a closely-related matrix to $\mathbf{H}_k$, $\mathbf{M}_k$ (effectively, multiplication of both sides by the inverse of $\mathbf{M}_k$).

The preconditioner $\mathbf{M}$ is typically chosen as a sparse symmetric 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 [17, 87] 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, with an appropriate (unusual) modified Cholesky factorization [87]. Other possibilities of preconditioners in general contexts have also been developed, such as a matrix derived from the BFGS update (defined in the QN subsection) [59].

Overall Work

Although more complex to implement than QN or nonlinear CG 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 CG require a Hessian/vector product ($\mathbf{Hp}$) at each inner loop iteration, and one solution of a linear system $\mathbf{Mz} = \mathbf{r}$ where $\mathbf{M}$ is the preconditioner. Because $\mathbf{M}$ may be sparse, this linear solution often takes a very small percentage of the total CPU time (e.g., $< 3\%$ [87]). The benefits of

faster convergence generally far outweigh these additional costs associated with the preconditioner.

Hessian/Vector Products

The Hessian/vector products in each linear CG step ($\mathbf{H}_k\,\mathbf{p}_k^j$) are more significant in terms of computer time. For a Hessian formulated with a nonbonded cutoff radius (e.g., 8 Å), many zeros result for the Hessian (see Figures 10.4 and 10.5); when this sparsity is exploited in the multiplication routine, performance is fast compared to a dense matrix/vector product. However, when the Hessian is dense and large in size, the following forward-difference formula of *two gradients* often works faster (we omit subscripts k from $\mathbf{H}$, $\mathbf{p}$, and $\mathbf{x}$ for clarity):

$$\mathbf{H}\,\mathbf{p} \;\approx\; [\mathbf{g}(\mathbf{x}+h\mathbf{p}) \;-\; \mathbf{g}(\mathbf{x})] \,/\, h\,, \tag{10.55}$$

where h is a suitably-chosen small number. The central difference approximation,

$$\mathbf{H}\,\mathbf{p} \;\approx\; [\mathbf{g}(\mathbf{x}+h\mathbf{p}) \;-\; \mathbf{g}(\mathbf{x}-h\mathbf{p})] \,/\, 2h\,, \tag{10.56}$$

may alternatively be used for greater accuracy at the cost of one more gradient evaluation with respect to the one-sided difference formula.

In either case, finding an appropriate value for the finite-difference stepsize h is nontrivial, and the accuracy of the product near the solution (where the gradient components are small) can be problematic.

Performance

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 QN Newton methods. If simplicity is at a 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.

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 can be especially important if normal-mode analysis is performed following minimization.

10.5.4 Simple Example

To illustrate performance of the methods described in this section, we have constructed a nonlinear minimization problem with an objective function dependent on two variables, whose contour lines are shown in Figure 10.10. Though the original problem has more variables, this construct represents a ‘slice’ of the real problem. Illustrations on more realistic, multivariate functions (potential functions of molecular systems) are presented in the next section.

The two-variable problem is derived from our charge optimization procedure [4] that determines electrostatic charge parameters for particles distributed on a virtual surface enclosing a macromolecular system; the electrostatic energy is modeled by a Debye-Hückel potential as an approximation (in the far zone) to a continuum, Poisson-Boltzmann solution to the electrostatic field surrounding the system. The objective function thus reflects the error in electric field (or potential) between the discrete and continuum approximations to the electrostatic potential of the complex macromolecular system.

Specifically, our constructed two-dimensional example seeks to optimize two charge values on the surface of the nucleosome, with the remaining 275 charges fixed.

The contour plots of our function, most readily seen from the darker illustration in Figure 10.10 (bottom right), show the unique minimum lying inside a shallow valley in the function surface.

The steepest descent (SD) path (top left) first overshoots the minimum and then slowly approaches it, reaching the high desired gradient accuracy of order 10^{-12} after nearly 2000 iterations. It also requires two orders of magnitude more CPU time that the other methods.

The CG path (top right) is direct and efficient for this problem, likely because of the quadratic nature of the function. Equally direct and efficient are the Newton minimizers: QN BFGS in Matlab and the TN package TNPACK (bottom illustrations). TNPACK, in particular, achieves a low gradient norm in one step.

10.6 Available Software

Table 10.1 summarizes the available minimizers in several chemistry and mathematics packages. See [60] and the NEOS (Network-Enabled Optimization System) Guide at www.mcs.anl.gov/otc/Guide/ for a compilation of up-to-date mathematical software for optimization and related information, including on www.mcs.anl.gov/otc/Guide/SoftwareGuide/.

10.6.1 Popular Newton and CG

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

10.6.2 CHARMM's ABNR

Of special note is the "adopted-basis Newton-Raphason" method implemented in CHARMM, ABNR. It is a memory-saving adaptation of Newton's method that

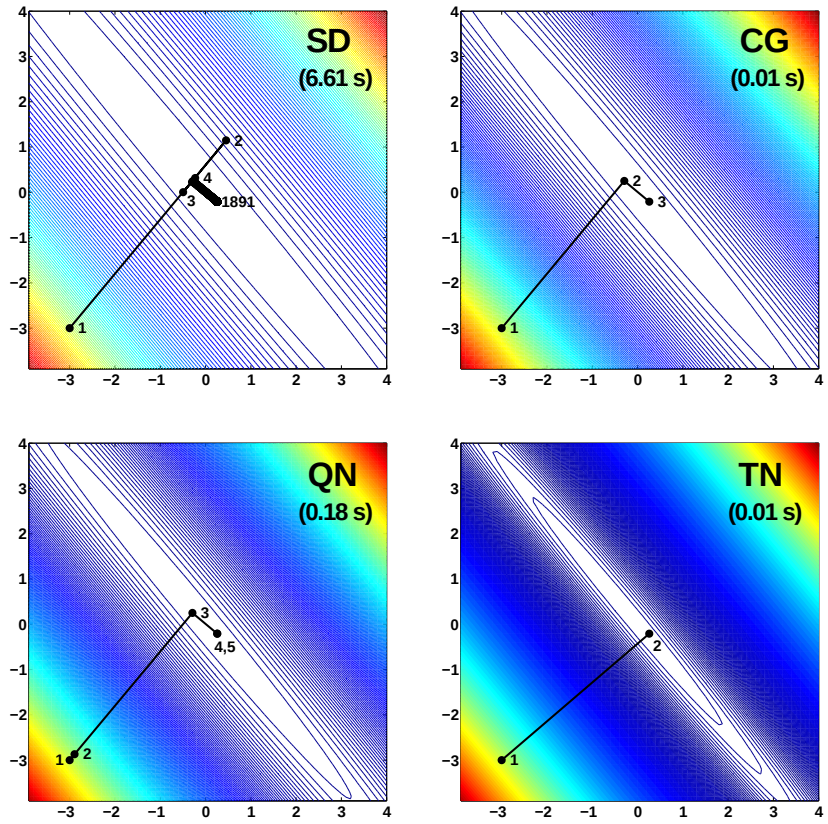

Figure 10.10. Minimization paths (with corresponding CPU times) for a function of two variables shown on top of function contour plots, for steepest descent (SD), nonlinear conjugate gradient (CG), BFGS quasi-Newton (QN), and truncated-Newton (TN) algorithms. See text for functional construction details. All contours are the same, but different levels of resolution are used in each plot to discern both the region near the minimum (darkest contour plot) and the higher-energy regions (lighter plots).

avoids analytic second derivatives. The idea is to use SD 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, SD 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 QN methods in that it uses only recent curvature information and exploits this information to make steady progress toward a solution.

10.6.3 CHARMM's TN

The TN method in CHARMM [17] is detailed elsewhere [81, 79, 80, 87, 88]. It uses a preconditioner constructed from the local chemical interactions (see Figures 10.4 and 10.5, right panels) and determines $\mathbf{p}_k$ from a truncated preconditioned CG loop. When negative curvature is detected, the preconditioned CG loop is halted with a guaranteed direction of descent. Interestingly, numerical analysis and experiments have shown that the method can produce quadratic convergence near a solution regardless of whether the preconditioner is indefinite or not [87]. As implemented, the method is applicable only to moderate system sizes (due to Hessian memory limitations).

10.6.4 Comparative Performance on Molecular Systems

In Table 10.2 we illustrate the minimization performance of three methods in CHARMM — nonlinear CG, ABNR, and TNPACK — for several molecular systems; see [87, 88] for details.

Note that the same minimum is obtained for the small systems (butane and *n*-methyl-alanyl-acetamide) but that different minima typically result for the larger systems.

Considerable differences in CPU times can also be noted. The CG method performs much slower and can fail 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 much faster convergence overall and yields smaller final gradient norms.

Note also that CG requires about two function evaluations per iteration (in the line search), while ABNR employs only one on average. TNPACK uses more than one function evaluation per outer iteration, since the (unscaled) magnitude of the produced search vector often leads to small steplengths at some iterations of the line search. The quadratic convergence of TNPACK is evident from Figure 10.11, where the gradient norm per iteration is shown.

10.7 Practical Recommendations

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 planar geometry [53]; many minimizers will produce the flat ring geometry, but this actually

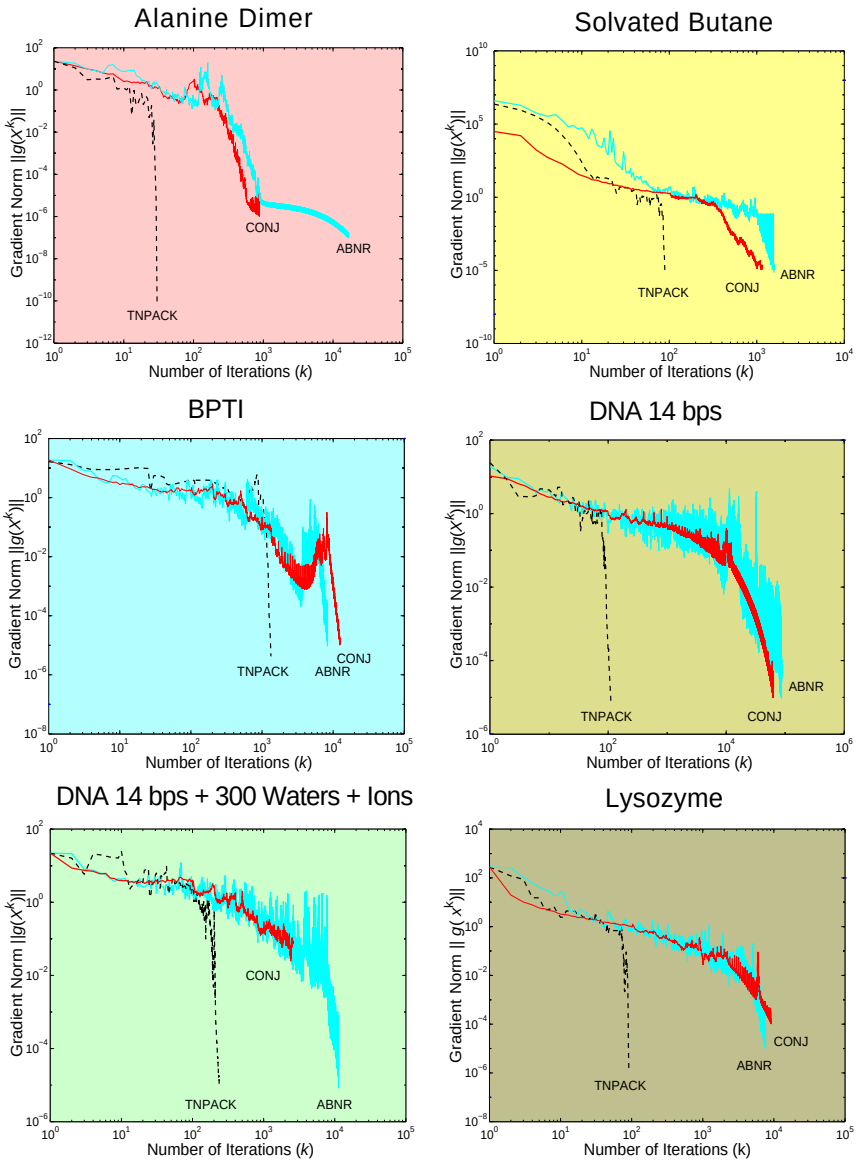

Figure 10.11. Minimization progress (gradient norm) of three CHARMM algorithms (CONJ: nonlinear conjugate gradient, ABNR: adopted-basis Newton-Raphson, and TNPACK: truncated Newton) for various molecular systems. See Figures 10.4 and 10.5 for corresponding sparse matrix patterns; the local Hessians are used as preconditioners for TNPACK. See also Table 10.2 for final function and gradient-norm values and required CPU time.

corresponds to a maximum! Different starting points will produce the correct nonplanar structure. See the homework assignment on minimization (number 10).

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

   Minimization of the DNA in vacuum system in Figure 10.11 and Table 10.2 by three algorithms also reveals very different final energies. This is because the DNA strands have separated! Proper solvation and ions remedy this physical/chemical problem. Interestingly, adding only water keeps the strands nearby but untwists the strands; only added ions and water maintain the proper DNA chemistry.

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 help identify 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 minimization algorithms for explanations.

4. *Check eigenvalues at the solution when possible*. 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 six 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}). 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 or improper physical models!* 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 minimization algorithms can find these min-

ima, 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 computed 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.

Besides artifacts caused by nonbonded cutoffs, improper physical models — such as that for DNA lacking solvent and ions, as discussed above — also produce artificial minima. For this example of DNA in vacuum, parallel strands rather than intertwined polynucleotide strands are produced as a result of minimization.

10.8 Looking Ahead

Only a small subset of topics was covered here in the challenging and ever-evolving field of nonlinear large-scale optimization. Interested readers are referred to the comprehensive treatments in the texts cited at the beginning of this chapter. The increase in computer memory and speed, and the growing availability of parallel computing platforms will undoubtedly influence the development of optimization algorithms in the next decade. Parallel architectures can be exploited in many ways: for performing minimization simulations concurrently from different starting points; for evaluating function and derivatives in tandem; for greater efficiency in the line search or finite-difference approximations; or for 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 routines are available to construct program codes for function derivatives. The construction is based on the chain-rule application to the elementary constituents of a function [32]. It is foreseeable that such codes will introduce greater versatility in Newton methods [64]. The cost of differentiation is not reduced, but the convenience and accuracy may increase.

Function separability is a more general notion than sparsity, since problems associated with sparse Hessians are separable but the reverse is not true. It is also another area where algorithmic growth can be expected [64]. (Recall that separable functions are composites of subfunctions, each of which depends only on a small subset of the independent variables; see eq. (10.6)). Therefore, efficient schemes can be devised in this case to compute the search vector, function curvature, etc., much more cheaply 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 [69, 23, 74] for examples. Scientists from all disciplines will anxiously await all these developments.

Table 10.1. Available optimization algorithms.

Package	Contact	Minimizers
AMBER	www.amber.ucsf.edu/amber/amber.html	SD, nonlinear CG from the IMSL library (due to Powell), and Newton.
CHARMM	yuri.harvard.edu	SD, nonlinear CG (FR, and modified PR version, the latter from the IMSL library),[a] Adopted-Basis Newton (ABNR), Newton, truncated-Newton (TNPACK).
DISCOVER	Biosym Technologies, San Diego, CA	SD, nonlinear CG (PR, FR versions),[a] quasi-Newton, truncated Newton.
DUPLEX	Brian E. Hingerty hingertybe@ornl.gov	Powell's coordinate descent method (no derivatives).[b]
ECEPP/2	QCPE 454 qcpe5.chem.indiana.edu/	Calls SUMSL, a quasi-Newton method based on a trust-region approach (by Gay).
GROMOS	igc.ethz.ch/gromos	SD and nonlinear CG (FR version),[a] both with and without SHAKE constraints.
IMSL Lib.	IMSL, Inc., Sugar Land, TX. www.vni.com/products/imsl/	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).[a]
LANCELOT	Philippe Toint www.cse.clrc.ac.uk/Activity/LANCELOT	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, www.mathworks.com	SD, DFP[c] and BFGS quasi-Newton, simplex algorithm, and others for linear and quadratic programming, least squares, etc..
MMFF94/94S	www.ccl.net/cca/data/MMFF94/	Calls OPTIMOL which uses a BFGS quasi-Newton method, with variable-metric updating scheme, but for Cartesian optimization (there is also a torsion-only optimizer) initiates the initial inverse Hessian approximated from the inverse of a 3×3 block-diagonal Hessian.
MM3	europa.chem.uga.edu	3×3 block-diagonal Newton and full Newton.
MM2	europa.chem.uga.edu	3×3 block-diagonal Newton.
NAG Lib.	NAG, Inc., Downers Grove, IL www.nag.com	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	femto.med.unc.edu/SIGMA/	Nonlinear CG (FR version).[a]
X-PLOR	atb.csb.yale.edu/xplor/	Nonlinear CG (from IMSL library).

[a] FR and PR refer to the Fletcher-Reeves and Polak-Ribi`ere nonlinear CG versions.
[b] See [34, 9] for details on DUPLEX.
[c] DFP is a rank-1 QN method, credited to Davidon, Fletcher, and Powell.

Table 10.2. Performance of three CHARMM minimizers on various molecular systems. See Figures 10.4 and 10.5 for patterns of the preconditioner used in TNPACK and Figure 10.11 for minimization progress. *Note*: For the DNA system in vacuum, though minimization produces a local minimum for each method, the structures are physically incorrect: without proper solvation, the DNA strands intertwine and separate. This also explains the very different values of final energies. For the DNA system with water and ions, the CG method terminates prematurely with an error message; the final gradient is relatively large.

Method[a]	Final f	Final $\|\mathbf{g}\|$	Itns.[b]	f&$\mathbf{g}$ Evals	CPU[c]
N-Methyl-Alanyl-Acetamide ($n = 66$)					
CG	-15.245	9.83×10^{-7}	882	2507	2.34 s
ABNR	-15.245	9.96×10^{-8}	16466	16467	7.47 s
TNPACK	-15.245	7.67×10^{-11}	29 (210)	44	1.32 s
Solvated Butane ($n = 1125$)					
CG	-2374.00	1.27×10^{-5}	1152	3175	49.48 m
ABNR	-2398.22	7.0×10^{-6}	1574	1575	48.52 m
TNPACK	-2381.04	7.7×10^{-6}	90 (1717)	263	59.44 m
BPTI ($n = 1704$)					
CG	-2792.93	9.9×10^{-6}	12469	32661	97.8 m
ABNR	-2792.96	8.9×10^{-6}	8329	8330	25.17 m
TNPACK	-2773.70	4.2×10^{-6}	65 (1335)	240	5.21 m
DNA 14 Bps ($n = 2664$)					
CG	-538.41	9.72×10^{-6}	62669	62670	20.42 h
ABNR	-1633.90	9.23×10^{-6}	86496	86497	6.69 h
TNPACK	-560.68	7.62×10^{-6}	111 (3724)	268	0.54 h
DNA 14 Bps + 300 Waters + Ions ($n = 5364$)					
CG	-11921.00	7.52×10^{-2}	2580	6616	1.62 h
ABNR	-11774.64	8.19×10^{-6}	11306	11307	2.78 h
TNPACK	-11928.50	9.84×10^{-6}	236 (6555)	687	1.75 h
Lysozyme ($n = 6090$)					
CG	-4628.362	9.89×10^{-5}	9231	24064	19.63 h
ABNR	-4631.584	9.97×10^{-6}	7637	7638	6.11 h
TNPACK	-4631.380	1.45×10^{-6}	78 (1848)	218	1.49 h

[a] CG: nonlinear conjugate gradient, ABNR: adopted basis Newton Raphson, TNPACK: truncated Newton based on the TNPACK package.

[b] For TNPACK, the total number of inner (preconditioned CG) iterations is indicated in parentheses, following the number of outer iterations.

[c] s: seconds, m: minutes, h: hours.

11

Monte Carlo Techniques

Chapter 11 Notation

SYMBOL	DEFINITION
Matrices	
M	mass matrix (components m_i)
Vectors	
p (or P)	collective momentum vector
q (or X)	collective position vector
R	collective random force vector
Scalars & Functions	
a	multiplier (of random number generator, a prime)
c	increment (of random number generator)
i, j, k, m, q, r	integers
t	time
u, v	real numbers
v_i	velocity component i
$\{x_i\}, \{\tilde{x}_i\}$	sequences of numbers
B_i	data batch (for MC mean)
E_k	kinetic energy
E_p	potential energy
F	probability distribution function
H (or E)	total energy
M	modulus (of random number generator, usually of order of computer word size); also used for number of batches in a sample
N	number of particles
N	sample size
Rg	radius of gyration
S, T	polynomial functions

Chapter 11 Notation Table (continued)

SYMBOL	DEFINITION
T	temperature
Wr	DNA writhing number
β	Boltzmann factor $1/(k_B\mathrm{T})$
γ	Langevin damping constant
μ	mean (also chemical potential in MC Carlo sampling section)
ρ	probability density function
σ^2	variance (σ is standard deviation)
τ	period (of random number generator)
$\langle A(x)\rangle_{\mathcal{D}}$	mean of property $A(x)$ over domain $\mathcal{D}$
$\lfloor y \rfloor$	largest integer smaller than or equal to y

It is a pollster's maxim that the truth lies not in any one poll but at the center of gravity of several polls.

Michael R. Kagay, *New York Times (Week in Review)*, 19 October 1998.

11.1 MC Popularity

From Washington D.C. to Wall Street to Los Alamos, statistical techniques termed collectively as Monte Carlo (MC) are powerful problem solvers. Indeed, disciplines as disparate as politics, economics, biology, and high-energy physics rely on MC tools for handling daily tasks.

Many problems that can be formulated as stochastic phenomena and studied by random sampling can be solved through MC simulations. Essentially, a game of chance is played, but with theoretical and practical rules from probability theory, stochastic processes, and statistical physics (Markov chains, Brownian motion, ergodic hypothesis) that lend the 'sport' practical utility.

11.1.1 A Winning Combination

MC methods are used for numerical integration, global optimization, queuing theory, structural mechanics, and solution of large systems of linear, partial differential or integral equations. MC methods are employed widely in statistical physics and chemistry, where the behavior of complex systems of thousands or more atoms in space and time is studied. Their appeal can be explained by a winning combination of simplicity, efficiency, and theoretical grounding.

11.1.2 From Needles to Bombs

Early records of random sampling to solve quantitative problems can be found in the 18th and 19th centuries with needle throwing experiments to calculate geometrical probabilities (George Louis Leclerc, a.k.a. Comte de Buffon, 1777)[1] or to determine π (Simon de Laplace, 1886). In 1901, Lord Kelvin also described an important application to the evaluation of time integrals in the kinetic theory of gases. Yet a novel class of MC methods (using Markov chains) provides the modern roots of MC theory, and is largely credited to the Los Alamos pioneers (Von Neumann, Fermi, Ulam, Metropolis, Teller, and others).

These brilliant scholars studied properties of the newly discovered neutron particles in the middle of the 20th century by formulating mathematical problems in terms of probability and solving analogues by stochastic sampling. Their work led to a surge of publications in the late 1940s and early 1950s on solving problems in statistical mechanics, radiation transport, and other fields by carefully-designed sampling experiments.

Most notable among these works was the famous algorithm of Metropolis *et al.* in 1953 [58]. With the rapid growth of computer speed and the development of many techniques to improve sampling, reduce errors, and enhance efficiency, MC methods have become a powerful utility in many areas of science and engineering.

11.1.3 Chapter Overview

In this chapter, only the most elementary aspects of MC simulations are described, including the generation of uniform and normal random variables, basic probability theory background (see also Box 11.1), and the Metropolis algorithm (due also to A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller).

Such methods can be used in molecular simulations to generate efficiently conformational ensembles that obey *Boltzmann statistics*, that is, the probability of a configuration X with energy $E(X)$ is proportional to $\exp(-E(X)/k_B\,\mathrm{T})$ (k_B is Boltzmann's constant and T is the temperature). From such ensembles, various geometric and energetic means estimated. Low-energy regions can also be identified by decreasing the temperature in the sampling protocol (this is termed "simulated annealing"). Method extensions that are of particular interest to the biomolecular community, such as hybrid MC, are also mentioned.

The codes illustrated in this chapter are provided in Fortran, still the language of choice in some of the popular molecular mechanics and dynamics packages

[1] Buffon used a Monte Carlo integration procedure to solve the following problem: a needle of length L is thrown at a horizontal plane ruled with parallel straight lines separated by $d > L$; what is the probability that the needle will intersect one of these lines? Buffon derived the probability as an integral and attempted an experimental verifi cation by throwing the needle many times and observing the fraction of needle/line intersections. It was Laplace who in the early 1800s generalized Buffon's probability problem and recognized it as a method for calculating π.

like CHARMM and AMBER. Analogous routines written in the C language can be obtained from the course website.

Since MC methods rely strongly on random number generators, the first section of this chapter is devoted to the subject. Students can skip Sections 11.2 and 11.3 if they wish to read material related to other aspects of MC simulations. To see immediately why random number generators are important and how they are used in MC simulations, students may wish to read at the onset Subsection 11.5.4 and see the example there (subroutine `monte`) for calculating π by MC sampling.

Good general introductions to MC simulations can be found in the texts by Kalos and Whitlock [40], Bratley, Fox and Schrage [8], and Frenkel and Smit [25]. There are many web resources for Monte Carlo tutorials, for example, obtained through the Molecular Monte Carlo Home Page of www.cooper.edu/engineering/chemechem/monte.html.

11.1.4 Importance of Error Bars

A point which cannot be overstressed in any introduction to MC methods is the fundamental importance of error bars in any MC estimate. Unlike in politics, perhaps, the reliability of any conclusion (e.g., estimate) in science depends on the associated accuracy. Scientists would no doubt have discarded the results of an election whose "margin of error . . . is far greater than the margin of victory, no matter who wins", an assessment by mathematician John Allen Paulos of the rocky 2000 U.S. Presidential race, between Texas governor George W. Bush (who became President) and former President Clinton's Vice President Albert Gore.

11.2 Random Number Generators

11.2.1 What is Random*?*

The computer sampling performed in MC simulations of stochastic processes relies on generation of "random" numbers. Actually, those numbers are typically *pseudorandom* since a deterministic recursion rule is used to generate a sequence of numbers given an *initial seed* x_0: $x_{i+1} = f(x_i, x_{i-1}, x_{i-2}, \ldots)$, where f is a function.[2] This reproducibility of the sequence is an essential requirement for debugging computer programs. Even sequences obtained via chaos theory (see [31], for example, and references cited therein) are deterministic.

It is essential to use 'good' random number generators in MC applications to avoid artifacts in the results. (The statement by Dilbert's cartoon character, a horned accounting troll, that "you can never be sure" [of randomness] is not an option for scientists! See cartoon posting on the dilbert.com archives for 10/25/01).

[2]We use the term *random* for brevity in most of this chapter, though the terms *pseudorandom* or *quasi-random* are technically correct.

The quality of a generator is determined not only by subjecting the generating algorithm to a large number of established tests (both empirical and theoretical). It is also important to test the *combination* of generator and application. The two examples described in the Artifacts subsection below (following the introduction of generator algorithms) illustrate how the performance of generators is application specific.

Much work has gone into developing random number generators on both serial and parallel computer platforms, as well as associated criteria for testing them. Concurrently, work has focused on the careful implementation of the mathematical expressions to ensure good numerical performance (e.g., avoid overflow or systematic loss in accuracy) and efficiency, on both general and special-purpose hardware.

Novices are well advised to use a routine from a reputable library of programs rather than programming a simple procedure reported in the literature, since many such procedures have not been actually tested comprehensively. Still, caveats are warranted even for some library routines; see below.

The reader is referred to classic texts by Kalos and Whitlock [40], Knuth [41], and Law and Kelton [44] for general introductions into random number generators. Some of these books also review basic probability theory. Good reviews by L'Ecuyer can be found in [45, 48] (see also www.iro.umontreal.ca/~lecuyer) and [57].

11.2.2 Properties of Generators

Let

$$\{ x_1, x_2, \ldots, \ldots \}$$

be a sequence of numbers. In theory, we aim for sequences of numbers that exhibit independence, uniformity, and a long period τ. In addition, it is important that such generators be as portable and efficient as possible.

Uniformity and Subtle Correlations

Most MC algorithms manipulate hypothetical *independent uniformly distributed* random variables (*variates*). That is, the independent variables are assumed to have a *probability density function* ρ (see Box 11.1) that satisfies $\rho_u(x) = 1$ for x in the interval $[0, 1]$ and $\rho_u(x) = 0$ elsewhere.[3] From such uniform variates, we can obtain other probability distributions than the uniform distribution, such as the normal, exponential, Gamma, or Poisson distributions; see subsection below on normal variates and [44] for generating continuous and discrete random variates from many distributions.

[3]We say that x lies in $[a, b]$ if $a \leq x \leq b$ and that x lies in $[a, b)$ if $a \leq x < b$; similarly, x in (a, b) means $a < x < b$.

Roughly speaking, independence of two random variables means that knowledge of one random variate reveals no information about the distribution of the other variate. In the strict sense of probability theory,[4] it is impossible to obtain true independence for random numbers. Generating *uncorrelated* random variates is a weaker goal than independence. (This is because independent random variables are uncorrelated but uncorrelated variables are not independent in general). Though correlations exist even in the best known random number generators, quality random number generators can defer correlations to high-order and high-complexity relations.

Long Period

The period τ associated with a sequence of random numbers is the number of sequential random values before the series repeats itself, that is,

$$x_{i+\tau} = x_i \qquad \text{for all integers } i \geq 0.$$

We require the sequence to have as *long a period as possible* to allow long simulations of independent measures.

The period length is an important consideration for modern large-scale simulations.[5] For a 32-bit computer, if the generator's state uses only 32 bits, the maximum period is usually $2^{30} \sim 10^9$ (assuming 2 bits are lost). This number is not a large number by today's standards. More than one million iterations may be performed in dynamics simulations and far more in MC sampling simulations. Moreover, each iteration may require large random *vectors* (e.g., in Langevin dynamics). Thus, the random number generators that might have been adequate only a decade ago on 32-bit machines quickly exhaust their values for the complex applications at present. Unfortunately, many such generators, which experts deem *unacceptable* [48], are often the default methods for many operating systems and software packages.

State-of-the-art generators use more bits for their state than the computer type and employ combinations of methods to defer correlations to high-order and high-complexity relations. This makes possible formulation of sequences with very long periods. For example, the codes given in [49] produce sequences with period lengths of up to order 2^{200} on 32-bit computers and 2^{400} on 64-bit machines!

Portability

Portability and efficiency are also important criteria of generators.

Portable generators are those that produce the same sequence across standard compilers and machines, within machine accuracy. Portability permits code com-

[4] The random variables x_1 and x_2 are independent if the joint probability density function $\rho(x_1, x_2)$ is equal to the product of the individual probability density functions: $\rho(x_1, x_2) = \rho_1(x_1)\rho_2(x_2)$

[5] Though for complex systems, the state descriptors (e.g., coordinates) are unlikely to be repeated in phase with the cycle of a (short) random number generator, subtle problems may occur in some applications, making the goal of long period generally desirable.

parison and repeatability on different platforms. This requirement is nontrivial because even if the mathematical recipe is identical certain floating-point calculations may involve hardware-wired instructions and branched directives for the sub-operations.

Efficiency

The issue of *speed* of random number generators can be important for some problems that involve a large number of computationally-intensive iterations. (See Table 11.1 for CPU data on different generators). Even if the relative computational cost of random number generators in large-scale applications is small, it is important to use quality compiler optimization utilities to reduce most of the overhead associated with *calling* the random number generator function itself. For this reason, it is also important to use a subroutine that returns a *vector* of random variates if an array of such numbers is desired, rather than calling the function multiple times for each vector component.

Box 11.1: The Probability Density and Distribution Functions

Let X be a random variable that takes on values x. We say that X is a *discrete* random variable if it takes on a countable number of values and *continuous* if it takes on an uncountably-infinite number of values.

The distribution function $F(x)$ (also termed the *cumulative distribution function*) of a random variable X defined as the probability that X takes on a values no larger than x (a real number), that is

$$F(x) = P(X \leq x)\,, \qquad -\infty < x < \infty\,. \tag{11.1}$$

A continuous random variable X has the closely related *probability density function* $\rho(x)$. (For discrete random variables, analogous definitions are formulated using a probability function $p(x)$). This relation is given by:

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} \rho(y)\, dy\,, \qquad -\infty < x < \infty\,. \tag{11.2}$$

Thus, $\rho(x)$ is closely related to the derivative of $F(x)$ (under some additional assumptions of regularity, we have $\rho(x) = F'(x)$).

For example, a uniform random variable on $[0, 1]$ has the probability density function

$$\rho(x) = \begin{cases} 1 & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}\,, \tag{11.3}$$

and the corresponding density function F is defined by:

$$F(x) = \int_0^x \rho(y)\, dy = \int_0^1 1\, dy = x\,. \tag{11.4}$$

The reader can verify that the mean μ (or *expected value*) of this continuous uniform random variable (by definition, $\mu \equiv E(X) = \int_{-\infty}^{\infty} x\, \rho(x)\, dx$) is $\mu = \int_0^1 x\, \rho(x)\, dx = \frac{1}{2}$

and that the variance σ^2 (by definition, $\sigma^2 \equiv E(X-\mu)^2 = E(X^2) - \mu^2$) is $\sigma^2 = \int_0^1 x^2\,\rho(x)\,dx - (\frac{1}{2})^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$.

A Gaussian (or *normal*) random variable with mean μ and variance σ^2 has the density function

$$\rho(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right), \qquad -\infty < x < \infty\,. \tag{11.5}$$

The associated distribution function is often denoted as $\mathcal{N}(\mu, \sigma^2)$. The probability density function for a *standard normal* random variable (with $\mathcal{N}(0,1)$) is

$$\rho(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2), \qquad -\infty < x < \infty\,. \tag{11.6}$$

11.2.3 Linear Congruential Generators (LCG)

The simplest type of random number method is a *linear congruential generator* (LCG), first used in 1948 by D. H. Lehmer.

Basic Recipe

LCGs compute successive iterates by multiplying the previous iterate by a constant, a, adding this product to another constant, c, and then taking the modulus of this result with respect to another large number, M.

Specifically, the LCG recipe relies on three integers. M (the *modulus*) is a large positive number; a (the *multiplier*) is a positive integer less than M and shares no divisors with M; and c (the *increment*) is less than M.

We then generate a sequence of variates from an initial integer seed $\tilde{x}_0$ less than M, namely $\{\tilde{x}_1, \tilde{x}_2, \ldots\}$, according to the recursion relation:

$$\tilde{x}_{i+1} = (a\tilde{x}_i + c) \bmod M, \qquad i = 0, 1, \ldots\,. \tag{11.7}$$

The uniform variates for this LCG are then obtained by division as:

$$x_i = \tilde{x}_i / M\,.$$

If $c = 0$, these real numbers $\{x_i\}$ are in the open unit interval $(0,1)$, and if $c \neq 0$ they are contained in the interval $[0,1)$. When $c = 0$, this LCG is called *multiplicative linear congruential generator* (MLCG).

The recurrence relation of eq. (11.7) has a period no greater than M. If the integers are properly chosen, the period will have the maximal length M. Judicious choice of a, c, and M must be made, as well as thorough tests for randomness of the resulting sequences. See [41, pp. 170–171] for specific recommendations.

Simple Example

As a simple illustration, consider the MLCG sequence with $M = 11$ and $a = 8$ ($c = 0$). From $\tilde{x_0} = 1$, we generate the sequence

$$\begin{aligned} \tilde{x}_{i+1} &= (8\tilde{x}_i) \bmod 11 \implies \\ &\{ \quad 1, 8, 9, 6, 4, 10, 3, 2, 5, 7, \quad 1, 8, 9, \ldots \}. \end{aligned} \tag{11.8}$$

We see that this sequence has the maximal period length of $M - 1 = 10$ and that each integer in the interval $[1, 10]$ is generated exactly once per cycle. The reader can verify that, for this choice of M (with $c = 0$), the values $a = 2, 6, 7, 8$ also have these favorable properties; the other values generate sequences with only two or five elements and hence violate the uniformity criteria strongly. However, as will also be discussed below, even the full-length sequences exhibit unacceptable correlations.

Of course, we are interested in much longer sequence lengths in real applications. Often, M is taken to be the *word size* of the machine and a is a prime number. However, M and a must be chosen with care, and the resulting algorithm carefully programmed, to avoid an integer *overflow* for the product $a\,\tilde{x}_i$; this is explained further below.

IBM's SURAND and Unix's rand and drand48

One old and still widely used MLCG method (possibly because its modulus M is the largest prime that fits in the 32-bit signed integer word used by many computers [3]) is SURAND, though it is considered poor by experts [48] (see discussion under Lattice Structure below and Figure 11.2). Developed by IBM for its system/360 series, SURAND has the values:

SURAND MLCG:

$$a = 7^5 = 16807; \qquad M = 2^{31} - 1 = 2147483647; \qquad c = 0\,.$$

A 'naive' FORTRAN implementation of this generator might be the simple implementation above, that is, include the two statements:

```
seed = mod (a * seed, m)
ranu = seed / m.
```

However, this would not produce the right sequence because of overflow.

To avoid the overflow in the product $a\,\tilde{x}_i$ (or `a * seed` in the code), it is necessary to ensure that all intermediate integers are bounded by $M - 1$. The basic idea, based on [70], is outlined in Box 11.2.

Box 11.2: Avoiding Overflow in Linear Congruential Generator Implementation

To avoid overfbw in the computation of the product $a\,x$ in the implementation of eq. (11.7) (we suppress subscripts for clarity), let us *assume* for the moment that we could factor M as

$$M = a\,q\,, \qquad q = \text{integer}\,. \tag{11.9}$$

Then, we could write the MLCG recursion relation as

$$f(x) = a\,x \bmod M = a\,x \bmod (a\,q) = a\,(x \bmod q)\,.$$

Of course M is a prime, and no such factorization $M = aq$ exists. However, instead of eq. (11.9), we can *approximately factor* M as:

$$M = a\,q + r, \qquad 1 \le r \le a - 1\,, \tag{11.10}$$

where

$$q = M \,\text{div}\, a \equiv \lfloor\, M/a \,\rfloor, \qquad r = M \bmod a\,. \tag{11.11}$$

Here $\lfloor\, y \,\rfloor$ denotes the largest integer smaller than, or equal to, y; in other words, $\lfloor\, M/a \,\rfloor$ is the integer division of M by a. If $r < q$, this approximate factorization is useful since then the magnitude of the intermediate product is not greater than $M - 1$.

For the SURAND MLCG ($a = 16807$ and $M = 2^{31} - 1$), we obtain $q = 127773 > r = 2836$.

This better implementation leads to the following correct implementation of SURAND (see [70, 8] for further details):

```
c*****************************************************************
      double precision function ranu ()
c Good implementation of SURAND. See Park & Miller,
c Comm. ACM 31:1192, 1988.  Subroutine ranset should be called
c (once) before the first function call.
      integer a, m, q, r, seed
      double precision rm
      parameter (a=16807, m=2147483647, q=127773, r=2836, rm=1d0/m)
      common /random/ seed
      save /random/
      data seed /1/
      seed = a * mod(seed, q) - r * (seed/q)
      if (seed .le. 0) seed = seed + m
      ranu = seed * rm
      return
      end
c*****************************************************************
```

However, this LCG is not recommended since there are far better procedures today. There are also faster and simpler ways to implement this recursion [49].

Other known MLCG combinations are the default random number generators available at the time of this writing on our SGI's Unix System Library, rand and drand48, using 32-bit and 48-bit integer arithmetic, respectively. Their parameters are as follows.

rand MLCG:

$$a = 1103515245; \quad M = 2^{31} = 2147483648; \quad c = 12345\,.$$

drand48 MLCG:

$$a = 25214903917; \quad M = 2^{48}; \quad c = 11\,.$$

Note the somewhat confusing online documentation for drand48, which reports a and c in base 8 rather than 10 (273673163155_8 and 13_8, respectively).

We discuss some of the defects of rand and drand48 below (see also Figure 11.2).

Lattice Structure in Linear Congruential Generators

Many statistical tests have been formulated to assess the suitability of random number generators. Linear congruential methods, for example, are known to exhibit correlations in certain hyperspaces; this basic defect is termed *coarse lattice structures*. Essentially, this means that when subsets of such sequences are represented in Euclidean space (two dimensions or higher), a lattice structure emerges; in other words, points lie on a number of hyperplanes rather than cover the space in a random-like manner. This pattern indicates that the sequence is not truly as random and uniform as sought. One way to visualize lattice structure is to plot k-lag pairs of numbers of the sequence, namely $\{x_i,\, x_{i+k}\}$ in the unit-square plane for fixed k. Often, we plot pairs of consecutive numbers in the sequence on the unit square and triplets of consecutive numbers in the sequence on the unit cube.

This defect of LCG methods has been credited to G. Marsaglia in 1968; see also [72]. *Spectral tests* have since been developed to measure such k-dimensional uniformities. Such tests essentially determine the maximum distance between adjacent hyperplanes; the larger this value, the worse the generator.

To illustrate, consider $k = 1$-lag pairs for our simple MLCG above with $M = 11$ and $a = 8$ (see expression in (11.8)). If we plot in two dimensions all consecutive pairs of points, that is:

$$\{1,8\},\{8,9\},\{9,6\},\{6,4\},\ \ldots\ ,\ \{7,1\}\,, \tag{11.12}$$

we see alarmingly that these points lie on four parallel lines with either positive and negative slopes (see Fig. 11.1). The spectral test would determine the maximum distance between these parallel lines.

Figure 11.1 shows the lattice structure generated from the four a values that yield full periods for this generator ($\tilde{x}_{i+1} = (a\,\tilde{x}_i) \bmod 11$). Clearly, defects emerge.

You might think that this toy problem is especially misleading. Unfortunately, even long LCG sequences are known to display such uniform patterns or structures that indicate imperfect uniform sampling.

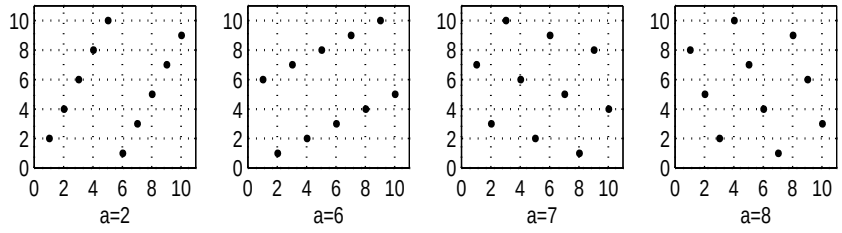

Figure 11.1. Lattice structure in two dimensional space for the multiplicative linear congruential generator (MLCG) generator $y_{i+1} = (a\, y_i) \bmod 11$ for various values of a.

Figure 11.2 shows the structure obtained for SURAND resulting from generating 5 billion random numbers and plotting pairs (in two dimensions) and triplets (in three dimensions) of consecutive (lag $k = 1$) numbers that appear on a subregion of the unit square. Clearly, defects are evident: a regular pattern emerges, indicating limited coverage. These defects would not have been apparent from a similar plot using far fewer numbers in the sequence — say 50,000 — as done in [3, Figure 3.3].

Figure 11.2 also shows patterns obtained from the Unix default generators rand and drand48 discussed above. For both rand and drand48, we also generated 5 billion consecutive numbers in the sequence. The corresponding (lag $k = 1$) plots on subregions of the unit square reveal a lattice pattern for rand but not drand48.

Most texts and review articles on the subject illustrate such patterns (e.g., [3, Figure 3]). For vivid color illustrations of the artifacts introduced by poor random number generators on the lattice structure of a polycrystalline Lennard-Jones spline, see [37], for example.

See also the related Monte Carlo exercise which involves generating 2D and 3D plots to search for structure of a particular (faulty!) random number generator termed RANDU. The LCG RANDU defined in that exercise can already exhibit a high degree of correlation when a relatively small number of sequence points (e.g., 2500) is generated!

11.2.4 Other Generators

To overcome some of these deficiencies of linear congruential generators, other methods have been designed. Two alternative popular classes are *lagged Fibonacci* and *shift-register* generators.

Fibonacci Series

A *Fibonacci series* is one in which each element is the sum of the two preceding values, e.g., $\{1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \ldots\}$. The series is

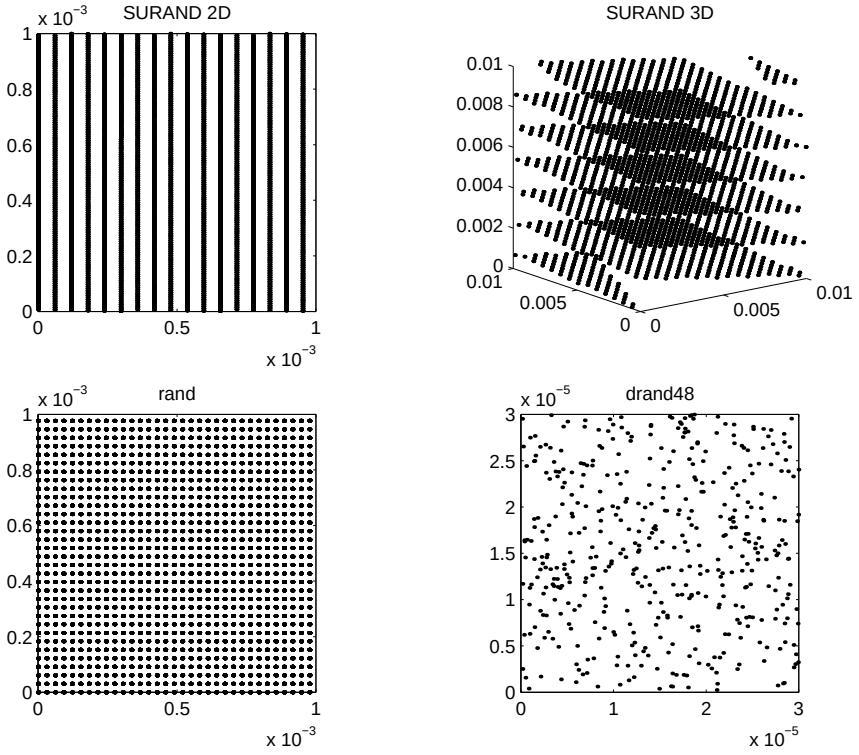

Figure 11.2. Structure corresponding to three linear congruential generators, with basic formula $y_{i+1} = (a\, y_i + c) \bmod M$, and displayed on subregions of the unit square or cube: SURAND ($a = 16807$, $M = 2147483647$, $c = 0$); rand ($a = 1103515245$, $M = 2^{31}$, $c = 12345$); and drand48 ($a = 25214903917$, $M = 2^{48}$, $c = 11$). In all cases, 5 billion numbers in the sequence were generated; fewer numbers correspond to the points displayed on the subregions in view (pairs in 2D and triplets in 3D). The code for SURAND was based on the one given in this text, and the codes for rand and drand48 were these available internally on our SGI's Unix System Library. The programs used for plotting the data are available on the text's website. See also Tables 11.1 and Figure 11.3 for Monte Carlo averages using rand and drand48.

named after the master of unit fractions, made famous for his rabbit breeding question;[6] the answer is 377, the 13th element of the Fibonacci series above.

[6]How many pairs of rabbits can be produced in a year from one rabbit pair? Assume that every month each pair produces a new offspring couple, which from the second month also becomes productive.

A Fibonacci random number generator computes each variate by performing some operation on the previous two iterates. Schematically, we write:

$$\tilde{x}_{i+1} = (\tilde{x}_{i-j} \odot \tilde{x}_{i-k}) \bmod M\,, \qquad j < k, \tag{11.13}$$

where $\odot$ is an arithmetic or logical operation and j and k are *integer lags*.

Multiplicative or additive lagged Fibonacci generators are common, and their periods can be quite long. Popular, easy-to-implement additive generators in this class have the form

$$\tilde{x}_{i+1} = (\tilde{x}_{i-j} + \tilde{x}_{i-k}) \bmod 2^m\,, \qquad j < k, \tag{11.14}$$

where k is sufficiently large (e.g., 1279). The maximal period of such a generator is $(2^k - 1)\, 2^{m-1}$ [41], and the procedure has the advantage of working directly in floating point, without the usual integer-to-floating point conversion.

An example of an additive lagged Fibonacci generator used by the Thinking Machines Library has $j = 5, k = 17, m = 32$ (period around 2^{48}). A multiplicative lagged Fibonacci generator considered better has the form

$$\tilde{x}_{i+1} = (\tilde{x}_{i-j} \times \tilde{x}_{i-k}) \bmod 2^m\,, \qquad j < k, \tag{11.15}$$

though its period, $(2^k - 1)\, 2^{m-3}$, is smaller by a factor of 4. This class of generators is the recommended choice by Knuth [41] and Marsaglia [56], though it was noted [41] that little theory exists to demonstrate their desirable randomness properties. L'Ecuyer later cautioned against the use of these lagged Fibonacci generators, as they display highly unfavorable properties when subjected to certain spectral tests [47].

Shift-Register Generators

Shift-register (or *Tausworthe*) random number generators have a similar form to lagged Fibonacci series generators but employ $M = 2$ in eq. (11.13). This means that only binary bits of a variate are generated and then collected into words by relying on a shift register. The operation $\odot$ is the 'exclusive or'.

An example of a shift-register generator is a k-step method which generates a sequence of k random numbers by splitting this sequence into consecutive blocks and then taking each block as the digit expansion in base p (typically 2). We can thus write this family of generators as:

$$\tilde{x}_{i+k} = \left[\sum_{j=0}^{k-1} a_j\, \tilde{x}_{i+j} \right] \bmod 2\,, \tag{11.16}$$

where the $\{\tilde{x}_i\}$ and $\{a_j\}$ are either 0 or 1. The output values $\{x_i\}$ are constructed from these bits.

Shift register methods with carefully chosen parameters are very fast, have strong uniformity properties, and possess period lengths that are not bounded by the word size of the machine. Though they may not exhibit lattice structure in the same space as LCGs, quality parameters must be selected based on analysis of

lattice structure in a different space (of formal series); see [46, 50] for example. The maximal length of series (11.16) is 2^{k-1}.

Combination Generators

Many other methods exist, including linear matrix generators, nonlinear recurrence generators such as inversive congruential and quadratic, and various combination of these generators. Combining output of good basic generators to create new random sequences can improve the quality of the sequence and increase the length. Schematically, we form $\{\tilde{z}_i\}$ from $\{\tilde{x}_i\}$ and $\{\tilde{y}_i\}$ by defining

$$\tilde{z}_i \;=\; \tilde{x}_i \;\odot\; \tilde{y}_i \,,$$

where $\odot$ is typically a logical (e.g., exclusive-or operator) or addition modulo M. If the associated periods of each sequence, τ_x and τ_y, are relatively prime, the cycle length of $\{\tilde{z}_i\}$ can be as large as the product $\tau_x\,\tau_y$. The properties of the combination sequence will be no worse than those of either sequence and typically better.

L'Ecuyer proposes good combined generators based on the addition of linear congruential sequences of higher order [49]. Three such highly efficient methods are offered programmed in the C language. The two sequences for 32-bit machines have lengths of $2^{191}(\sim 10^{57})$ and $2^{319}(\sim 10^{96})$, and the 64-bit version has the impressive length of $2^{377}(\sim 10^{113})$. All combine two sequences defined by

$$\tilde{x}_{1,i} = (\, a_{1,1}\,\tilde{x}_{1,i-1} + a_{1,2}\,\tilde{x}_{1,i-2} + \cdots a_{1,k}\,\tilde{x}_{1,i-k}\,) \bmod M_1 \,, \qquad (11.17)$$

$$\tilde{x}_{2,i} = (\, a_{2,1}\,\tilde{x}_{2,i-1} + a_{2,2}\,\tilde{x}_{2,i-2} + \cdots a_{2,k}\,\tilde{x}_{2,i-k}\,) \bmod M_2 \,, \qquad (11.18)$$

for $i = 0, 1, \ldots$, where M_1 and M_2 are distinct primes and the two sequences have period lengths $M_1^k - 1$ and $M_2^k - 1$, respectively.

A combined multiplicative linear congruential generator can be formed by adding δ multiples of the variates from the two series, or by forming

$$X_i = (\, \delta_1\,\tilde{x}_{1,i}/M_1 \;+ \delta_2\,\tilde{x}_{2,i}/M_2\,) \,, \qquad (11.19)$$

where δ_1 and δ_2 are integers, each relatively prime to its associated sequence modulus (M_1 and M_2). With properly chosen parameters, the period length of $\{X_i\}$ will be $(M_1^k - 1)(M_2^k - 1)/2$. These formulas can be generalized to more than two sequences.

L'Ecuyer's sequence of length 2^{191} [49] combines two sequences by using $k = 3$ (the number of prior sequence iterates), $M_1 = 2^{32} - 209$, $M_2 = 2^{32} - 22853$, and $a_{1,1} = a_{2,2} = 0$, $a_{1,2} = 1403580$, $a_{1,3} = -810728$, $a_{2,1} = 527612$, $a_{2,3} = -1370589$. The second 32-bit-machine generator uses more terms with $k = 5$, and the long, 64-bit generator has $k = 3$ but two larger moduli, of order 2^{63}. See the programs for these generators (in the C language) in [49].

FORTRAN and C codes for another generator of length $\sim 2^{121}$ with good statistical properties [51] are available on the course website. This generator combines

four MLCGs defined by

$$\tilde{x}_{j,i} = (a_j\, \tilde{x}_{j,i-1}) \bmod M_j\,, \quad j = 1, 2, 3, 4, \tag{11.20}$$

via

$$w_i = \sum_{j=1}^{4} (\delta_j\, \tilde{x}_{j,i}/M_j) \bmod 1, \quad \delta_j = (-1)^{j+1}\,. \tag{11.21}$$

The respective multipliers and moduli are set to $a_1 = 45991$, $a_2 = 207707$, $a_3 = 138556$, $a_4 = 49689$, and $M_1 = 2147483647$, $M_2 = 2147483543$, $M_3 = 2147483423$, $M_4 = 2147483323$, respectively [51].

11.2.5 Artifacts

Two interesting examples that illustrate the importance of quality random number generators and their appropriate testing with the application at hand are summarized in Boxes 11.3 and 11.4.

The first, from a real situation in 1989, reflects a coincidental relationship between the problem (matrix) size and the period of the generator (the matrix dimension divides the period τ), as well as short τ and limited accuracy (single-precision computer arithmetic). These problems could have been avoided by averting this matrix-dimension/generator-period relationship, increasing the generator period, and using double-precision arithmetic.

The second instructive example of "hidden errors" stemming from apparently good random number generators was reported in 1992 [20]. Essentially, the researchers showed that incorrect results can be produced under certain circumstances by random number generators that have a long period and have passed certain tests for randomness. Thus, a careful testing of the combination of random number generator and application is generally warranted. Though thought to be generators of high-quality, the generators used in [20] are known to have unfavorable lattice structure [47]. Of course, these examples also argue for using generators with as-long-a-period as possible.

In addition to possible systematic errors with high-quality random number generators for some algorithms due to subtle correlations, researchers showed that even good generators can yield inconsistent results regardless of the algorithm [76]. Though researchers believed that the two used generators had passed all known statistical tests, the generators produced (with the same algorithm) critical temperature estimates for the continuous clock model that differed by 2%, much higher than intrinsic errors. This model has a second-order phase transition, and the exact answer is not known. Thus, it cannot be determined which result is more reliable.

The best advice, therefore, appears to be not only to use reliable generators with as long periods as possible, but also to experiment with several generators as well as algorithms to the extent possible.

Box 11.3: Accidental Relationship Between Problem Size and Generator Length (1989 Linpack Benchmarks)

In 1989, David Hough, a numerical analyst working at Sun Microsystems, noticed very peculiar behavior in benchmark testing of the package Linpack. (The original posting can be found on the archived NA Digest, Volume 89, Issue 1, 1989). The single-precision factorization of 512×512 randomly-generated matrices produced a perplexing underflow (roughly around 10^{-40}) in the diagonal pivot values. (*Sherlocks: note that* $512 = 2^9$). However, matrices composed of random data from a uniform distribution are known to be remarkably well conditioned! So how can this seemingly well conditioned matrix be nearly singular?

The answer came upon examination of the random number generator and how it was used to set the matrix elements. Specifically, the matrix elements a_{ij} were set to be in the range $[-2, 2]$ according to the following subprogram, which relies on a simple MLCG with $a = 3125$ and $M = 65536$. (*Holmes fans: note that* $M = 2^{16}$).

```
c*********************************************************************
      subroutine matgen(amat,Lda,n,b,norma)
      real amat(Lda,1), b(1), norma, halfm, quartm
      integer a, m
      parameter  (a = 3125, m = 65536)
      parameter  (halfm = 32768.0, quartm = 16384.0)
      iseed = 1325
      norma = 0.0
      do 30 j = 1, n
         do 20 i = 1, n
            iseed = mod (a * iseed, m)
            amat(i,j) = (iseed - halfm) / quartm
            norma  = max (amat(i,j), norma)
   20    continue
   30 continue
      return
      end
c*********************************************************************
```

A quick examination first showed that the period of this MLCG is only $\tau = M/4 = 16384$; the full period of M could have easily been generated by changing one line above, to incorporate the nonzero $c = -1$ increment value. Still, this would have only delayed the underflow problem to a 1024×1024 matrix.

The main problem here lies in the fact that the matrix size chosen, $512 = 2^9$, *divides* the modulus, also a power of 2 ($M = 2^{16}$). Hence, the period $\tau = 2^{14}$ factors $\tau = 512 \times 32$. *This means that the first 32 columns of the "random" matrix are repeated 16 times!* The matrix is subsequently singular, and after each 32 steps of Gaussian elimination the zeros in the lower triangular part of the matrix are reduced by a factor of order (10^{-7}). Clearly, after six rounds of such transformations (each treating 32 columns), the element size in the lower triangle would drop to order $\mathcal{O}(10^{-42})$, explaining the underflow. This problem could have been removed by using matrix sizes that do not divide the period, resorting to double precision arithmetic (underflow threshold of 10^{-300}), and by increasing the period

of the generator substantially.

Note also that besides underflow, the generator could have experienced other problems for matrices smaller than 512×512, since $512 \times 512 = 2^{18}$, and 2^{18} is greater than 2^{14}, the generator's period.

11.2.6 Recommendations

In sum, high-quality, long-sequence random number generators are easy to find in the literature, but they are not necessarily available on default system implementations.

For the best results, an MC simulator is well advised to consult the resident mathematical expert for the most suitable generator and computing platforms with respect to the application at hand.

Certainly, the user should compare application results as obtained for several random number generators. Indeed, L'Ecuyer likens generators to cars [49]: no single model nor size of an automobile can possibly be a universal choice. Good expert advice can be obtained by examining Pierre L'Ecuyer's website (presently at www.iro.umontreal.ca/∼lecuyer). Certainly, given today's computationally-intensive biomolecular simulations, it is advisable to use sequences with long periods. Combined multiplicative linear congruential generators are good choices. See [49] for good recommendations. See also [38, pp. 76–78] for an efficient FORTRAN implementation of a good combination MLCG used for DNA work, though with a small period by today's standards (10^{18}).

Generators particularly suitable for parallel machines are also available [71, 37], characterized by different streams of variates produced by good seeding algorithms and variations in the parameters of the underlying recursion formulas. See the SPRNG scalable library package (Scalable Parallel Random Number Generation) targeted for large-scale parallel Monte Carlo applications: sprng.cs.fsu.edu, for software. The package can be used in C, C++, and Fortran and has been ported to most major computer platforms.

Box 11.4: Accidental Relationship Between Simulation Protocol and Generator Structure (Ising Model)

Ferrenberg *et al.* [20] used different generators in the context of simulating an Ising model, a model characterized by an abrupt, temperature-dependent transition from an ordered to a disordered state. The states, characterized by the spin directionality of the particles, were generated by an algorithm termed Wolff that determines the flips of a cluster on the basis of a random number generator. Surprisingly, the researchers found that the correct answer was approximated far better by the 32-bit multiplicative linear congruential generator SURAND, well recognized to have lattice-structure defects. So why does

the apparently-superior shift register generator produce *systematically incorrect results* — energies that are too low and specifi c heats that are too high?

An explanation to these observations came upon inspection of the Wolff algorithm. Namely, subtle correlations in the random number sequence affect the Wolff algorithm in a specifi c manner! If the high order bits are zero, they will remain zero according to the spin generation algorithm. This in turn leads to a bias in the cluster size generated and hence the type of equilibrium structures generated.

The main message from this work was a note of caution on the effect of subtle correlations within random number generators on the system generation algorithms used for simulating the physical system. This suggests that not only should a generator be tested on its own; it should be tested together with the algorithm used in the MC simulation to reduce the possibility of artifacts.

11.3 Gaussian Random Variates

11.3.1 Manipulation of Uniform Random Variables

Our uniform random variates computed in the last section, U, can be used to generate variates X that correspond to a more general, given probability distribution by a simple transformation. To generate a continuous variate X with distribution function $F(x)$ (see Box 11.1) which is continuous and strictly increasing on $(0, 1)$ (i.e., $0 < F(x) < 1$), we set x to $F^{-1}(u)$ where F^{-1} is the inverse of the function F. The challenge in practice is to establish good algorithms for evaluating $F^{-1}(u)$ to the desired accuracy.

Below we only describe generating variates from a Gaussian (normal) distribution, commonly needed in molecular simulations. For information on generating variates from many other distributions, see [44] and the web page of Luc Devroye (cgm.cs.mcgill.ca/∼luc/), for example.

11.3.2 Normal Variates in Molecular Simulations

A vector of normally-distributed random variates satisfying a given mean (μ) and variance (σ^2) (see Box 11.1, eq. (11.5)) is often required in molecular simulations. One example is the initial velocity vector (of components $\{v_i\}$) in a molecular dynamics simulation corresponding to the target temperature of an n-atom system,

$$\sum_{i=1}^{3n} m_i v_i^2 = 3\, n\, k_B\, \mathrm{T}\,.$$

Another example is the Gaussian random force vector R in Langevin dynamics with zero mean and variance chosen so as to satisfy:

$$\langle R(t)R(t')\rangle = 2\gamma k_B \mathrm{T}\mathbf{M}\delta(t-t') \,, \tag{11.22}$$

where $\mathbf{M}$ is the mass matrix and γ is the damping constant.

In the molecular and Langevin dynamics cases above, we first set each component of the vector vec from a standard normal distribution (zero mean and unit variance) so that the sum is also a normal distribution with the additive means and variances (see Central Limit Theorem below); to obtain the desired variance σ^2 rather than unit variance, we then modify each component according to the vector update relation

$$\mathsf{vec} \leftarrow \sigma\ \mathsf{vec} + \mu \,.$$

Since, by this modification, it is easy to generate normal variates from the normal distribution with mean μ and variance σ^2 ($\mathcal{N}(\mu,\sigma^2)$) from variates sampled from a standard normal distribution ($\mathcal{N}(0,1)$), it suffices to generate standard normal variates.

There are several techniques to set a variate X_u from a Gaussian or normal distribution on the basis of a uniformly distributed variate U (for which procedures were discussed above). Two are described below.

11.3.3 Odeh/Evans Method

One efficient approach was described by Odeh and Evans [67]. For a given u value in the range $0 < u < 1$, the corresponding normal variable x_u is computed to satisfy:

$$u \;=\; \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x_u} \exp(-t^2/2)\; dt \,. \tag{11.23}$$

This is accomplished by approximating x_u as the sum of two terms:

$$x_u \;=\; y + S(y)/T(y) \,, \qquad y = \sqrt{\{\ln(1/u^2)\}} \,,$$

where S and T are polynomials of degree 4 chosen to yield the minimal degree rational approximation to $x_u - y$ with maximum error less than 10^{-7}.

In practice, a vector of random variates is first formed ($U \equiv \{u_1, u_2, \cdots, u_n\}$ in the notation above; see subroutine ranuv below, based on function ranu), from which a standard normal distribution is formulated ($X_U \equiv \{x_{u_1}, x_{u_2}, \cdots, x_{u_n}\}$); each component is then adjusted to yield the desired mean and standard deviation (see subroutine rannv1 below).

```
c*****************************************************************
      subroutine rannv1 (n, vec, mean, var)
c A vector of n pseudorandom numbers is generated, each from a
c standard  normal distribution (mean zero, variance one), based on
c Odeh and Evans, App. Stat. 23:96 (1974).
c For a nonzero mean MU and/or non unity variance,
```

```
c set  vec(i) = mu + sqrt(sigma(i))*vec(i).
c Subroutine ranset should be called before the first subroutine
c call.
      integer n
      double precision vec(n),mean,var(n),
     *  temp,p0,p1,p2,p3,p4,q0,q1,q2,q3,q4
      parameter (p0=-.322232431088d0, p1=-1d0, p2=-.342242088547d0,
     *  p3=-.204231210245d-1, p4=-.453642210148d-4,
     *  q0=.99348462606d-1, q1=.588581570495d0, q2=.531103462366d0,
     *  q3=.10353775285d0, q4=.38560700634d-2)
      if (n .lt. 1) return
      call ranuv(n, vec)
      do 10 i = 1, n
         temp = vec(i)
         if (temp .gt. 0.5d0) vec(i) = 1d0 - vec(i)
         vec(i) = sqrt(log(1d0/vec(i)**2))
         vec(i) = vec(i) +
     *         ((((vec(i) * p4 + p3) * vec(i) + p2) *
     *         vec(i) + p1) * vec(i) + p0) /
     *         ((((vec(i) * q4 + q3) * vec(i) + q2) *
     *         vec(i) + q1) * vec(i) + q0)
         if (temp .lt. 0.5d0) vec(i) = -vec(i)
   10 continue
      do 20 i = 1, n
         vec(i) = sqrt(var(i)) * vec(i) + mean
   20 continue
      return
      end
c*********************************************************************
      subroutine ranuv (n, vec)
c Generate a vector of n pseudorandom uniform variates
      integer n, a, m, q, r, seed
      double precision vec(n), rm
      parameter (a=16807, m=2147483647, q=127773, r=2836, rm=1d0/m)
      common /random/ seed
      save /random/
      if (n .lt. 1) return
      do 10 i = 1, n
         seed = a * mod(seed, q) - r * (seed/q)
         if (seed .le. 0) seed = seed + m
         vec(i) = seed * rm
   10 continue
      return
      end
c*********************************************************************
```

For example, to set the initial velocity vector according to the target temperature using the equipartition theorem (each degree of freedom has $k_B\mathrm{T}/2$ energy at thermal equilibrium), the routines above are used for the velocity vector V of $3n$ components $\{v_i\}$ with $\mu = 0$ and var(i) = $(k_B\mathrm{T})/m_i$. For the Langevin random force vector, the variance for each vector coordinate i is: $(2\gamma k_B m_i \mathrm{T})/\Delta t$, where

the delta function δ in eq. (11.22) is discretized on the basis of the timestep Δt (see also Chapter 12 on molecular dynamics).

11.3.4 Box/Muller/Marsaglia Method

Another popular algorithm to form normal variates x_1 and x_2 is the Box/ Muller/Marsaglia method [41, pp. 117–118]. It involves generating two *uniformly* distributed random variates u_1 and u_2, setting v_1 and v_2 as uniform variates between -1 and $+1$ ($v_1 \leftarrow 2u_1 - 1, v_2 \leftarrow 2u_2 - 1$), checking that $s = v_1^2 + v_2^2$ is less than 1 (if $s \geq 1$, the procedure is repeated), and then setting the two *normal* variates x_1 and x_2 as:

$$x_1 = v_1\sqrt{-2\ln s/s}\,, \qquad x_2 = v_2\sqrt{-2\ln s/s}\,. \tag{11.24}$$

Essentially, we are using the polar-coordinate representation of x_1 and x_2 by v_1 and v_2 ($x_1 = \tilde{r}\cos\tilde{\theta}\,, x_2 = \tilde{r}\sin\tilde{\theta}\,, \tilde{r} = \sqrt{-2\ln s}\,, \tilde{\theta} = \tan^{-1}(v_2/v_1)$) to construct the joint probability distribution of the two normal variates in polar coordinates:

$$\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x_1} e^{-x^2/2}\,dx\right)\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x_2} e^{-y^2/2}\,dy\right)$$

$$= \frac{1}{2\pi}\int_{\substack{\{(r,\theta)\,\mid \\ r\cos\theta \leq x_1 \\ r\sin\theta \leq x_2\}}} e^{-r^2/2}\; r\,dr\,d\theta\,. \tag{11.25}$$

11.4 Means for Monte Carlo Sampling

11.4.1 Expected Values

Armed with a uniform random variate generator, we can now address the important task of estimating a mean property of interest. In molecular simulations, we might seek the average geometric and energetic properties associated with an equilibrium distribution of conformations.

MC Estimate

In its simplest form, we write such a mean, or expected value, as an integral

$$I = \int_{\mathcal{D}} A(x)\,dx = \langle A(x)\rangle_{\mathcal{D}} \tag{11.26}$$

where the average is computed over the uniformly distributed elements $x \in \mathcal{D}$. For example, assume that the function $A(x)$ is defined on [0, 1]. Choose a sequence of N random variates for large N,

$$x_1, x_2, \ldots, x_N\,,$$

and generate corresponding function values $y_i = A(x_i)$:

$$y_1, y_2, \ldots, y_N \, .$$

Then we compute the average, termed the Monte Carlo estimate of I, by:

$$\bar{y}_N = \frac{1}{N} \sum_{i=1}^{N} y_i \, . \tag{11.27}$$

Simple Example: Calculate π by MC

As a simple example, consider calculating π by Monte-Carlo integration of the area of a quarter-circle of radius 1 circumscribed inside the unit square in the plane (with center at the origin of the plane). The integral to be evaluated is:

$$\int_0^1 \int_0^1 \rho(x, y) \, dx \, dy$$

where

$$\rho(x, y) = \begin{cases} 1 & x^2 + y^2 \leq 1 \\ 0 & \text{else} \end{cases} \, .$$

This integral's value is $\pi/4$. A simple Fortran program to perform this integration by Monte Carlo sampling consists of the following:

```
c*****************************************************************
      subroutine monte(nstep)
      implicit none
      integer nstep, i, nin, iseed
      double precision  x, y, tmp, rand

      nin = 0
      iseed = 12345
      call srand(iseed)

      do 30 i = 1, nstep                *
         x = rand()                    /|\
         y = rand()                     |   ^
         tmp = sqrt(x*x + y*y)          |      ^
         if (tmp .lt. 1.d0) then        |
            nin = nin + 1               |         ^
         endif                          |          \
   30 continue                          +-----------*
                                                    /
      print *, nstep, (4.d0 * nin)/nstep
      return
      end
c*****************************************************************
```

Results as a function of the sample size (nstep) are presented in Table 11.1 and Figure 11.3 using the Unix rand and drand48 random number generators and also more sophisticated methods. Since drand48 is the fastest of the generators,

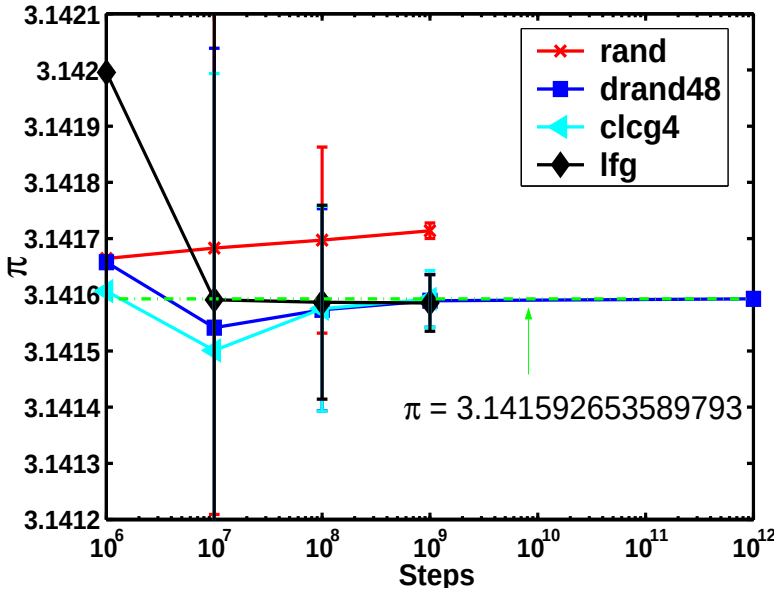

Figure 11.3. Results (means and error bars) of MC estimates for π based on different random number generators, as tabulated in Table 11.1.

we also record the estimated value corresponding to 10^{12} steps (only up to 10^9 steps for the rest).

Note that, unfortunately, this procedure for calculating π is not very accurate. At best, the first six decimal places of $\pi = 3.14159265358979323846\ldots$ are obtained. The accuracy is limited not only by the sample size — statistical error — but also by any possible defects of the random number generator (e.g., lattice structure and limited coverage; see Figure 11.2). Here we see that the accuracy of the means starts to deteriorate after the number of steps exceeds the period length. We also learn from this example that the longer-period generators have greater resolution (another order of magnitude of two).

11.4.2 Error Bars

Law of Large Numbers

According to the *Law of Large Numbers* in probability theory, the average of N sampled random variables converges (in probability) to its expected value. Stated more formally, if the uniform variates are independent and drawn from the same distribution so that the expected value of each y_i is μ, then as $N \to \infty$ the average value $\bar{y}_N$ converges to μ asymptotically:

$$P\{\lim_{N\to\infty}(\bar{y}_N) = \mu\} = 1\,.$$

However, the rate of convergence to the expected value is a different matter and requires stronger assumptions.

Variance

As stressed in this chapter's introduction, it is essential to provide *error bars* when reporting an MC average. The variance of $\bar{y}_N$ is defined as

$$\sigma_{\bar{y}}^2 = \mathrm{var}(\bar{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_N)^2 \,. \tag{11.28}$$

The variance measures the distribution of $\bar{y}$ about its mean μ; the larger N is, the narrower the interval about I where $\bar{y}_N$ can be found.

Variance Relation to Central Limit Theorem

This interval can be determined as a probability of deviation in units of σ on the basis of the *Central Limit Theorem*. This beautiful and powerful result states that as $N \to \infty$, the *limiting distribution* for a *sum of random variates* is the *normal distribution*.

Specifically, if $\{y_1, y_2, \ldots\}$ is a sequence of independent, identically distributed random variates having mean μ and finite nonzero variance σ^2, then the random variable

$$S_N = y_1 + y_2 + \cdots + y_N$$

has the normal density with mean $N\mu$ and variance $N\sigma^2$, $\mathcal{N}(N\mu, N\sigma^2)$. In other words, the normalized random variable

$$\frac{S_N - N\mu}{\sqrt{\mathrm{var}(S_N)}} = \frac{S_N - N\mu}{\sigma\sqrt{N}}$$

has the standard normal distribution:

$$\lim_{N\to\infty} P\left(\frac{S_N - N\mu}{\sigma\sqrt{N}} \le x\right) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \exp\left[-t^2/2\right] dt \,.$$

Thus, in reporting an MC average, we say that we have estimated I within one standard error (i.e., $\sigma/\sqrt{N}$) of $\bar{y}_N$ as:

$$I = \bar{y}_N \pm \sigma_{\bar{y}}/\sqrt{N} \,. \tag{11.29}$$

For example, $N = 10,000$ yields a result that is at best roughly 1% accurate; for correlated data, such as from molecular dynamics simulations, a much larger N is required for that accuracy. Note that this $1/\sqrt{N}$ scaling of errors is a general feature of MC methods and is independent of the space dimension involved; that is why MC is frequently the method of choice for multidimensional integrals.

Since we know that the limiting distribution for $\bar{y}_N$ is the normal distribution $\mathcal{N}$, we say that 68.3% of the time this estimate is within one standard error of

I [40]. The above integral estimates can be generalized to independent random samples from other probability densities, as described in the next section.

Note that the above errors are statistical *and can be controlled. The more serious errors in MC algorithms are the* systematic *errors, such as discussed above in connection with random-number-generator artifacts. Both errors should be monitored to the extent possible.*

11.4.3 Batch Means

When the MC data are highly correlated, the error bars may decrease much more slowly with N as in the above idealized case. The effective number of samples is then N/τ where τ is the decorrelation time (number of steps) for the data. This value can be determined by examining auto and cross-correlation functions for the most slowly-varying properties of the system or by using the method of *batch means*. Extensive mathematical/statistical tests for independence are also available to estimate *confidence intervals* of independent means [44, Chapter 4].

Essentially, we divide the sample size N into M batches $\{B_1, B_2, \ldots, B_M\}$ each of $b = N/m$ elements where M should be significantly greater than τ; we then obtain a mean over each batch sample:

$$\bar{y}_{B_i} = \frac{1}{b} \sum_{y_i \in B_i} y_i \,, \qquad\qquad i = 1, \ldots, M \,, \tag{11.30}$$

and then set the $\bar{y}_N$ estimate as the average over all these means:

$$\bar{y}_M = \frac{1}{M} \sum_{i=1}^{M} \bar{y}_{B_i} \,. \tag{11.31}$$

In reporting the estimator of form (11.29), the relevant sample size (M rather than N) and variance to is determined from the above mean, that is:

$$\sigma^2_{\bar{y}_M} = \mathrm{var}(\bar{y}_M) = \frac{1}{M} \sum_{i=1}^{M} (\bar{y}_{B_i} - \bar{y}_M)^2 \,. \tag{11.32}$$

It can be shown that if the batch size M is sufficiently large, the means of the batches are approximately uncorrelated. In practice, variations on the basic batch means method sketched above, and additional tests, are needed to yield good statistics [44, pages 528–530].

11.5 Monte Carlo Sampling

11.5.1 Density Function

The properties of many molecular systems can be described by a separable Hamiltonian of general form

$$H(q,p) = E_k(p) + E_p(q) \;=\; \frac{1}{2}p^{\mathrm{T}}\mathbf{M}^{-1}p + E_p(q)\,, \tag{11.33}$$

where E_k and E_p are the kinetic and potential energy components, respectively, and q and p are the collective position and momentum vectors of the system. This Hamiltonian function forms the basis for MC simulations applied to estimate various properties of large molecular systems, such as geometric and thermodynamic functions. However, the MC estimates must emulate a probability density function $\rho(q,t)$ or $\rho(q,p,t)$ appropriate for the statistical ensemble (t denotes time). This probability density ρ may or may not be known.

11.5.2 Dynamic and Equilibrium MC: Ergodicity, Detailed Balance

MC simulations can be used to mimic a *dynamic process* (ρ depends on time t), as in Brownian dynamics. They can also generate an ensemble around a *statistical equilibrium*, as in some conformational sampling studies.

Dynamic Process

In the former case, a deterministic rule (such as based on Newton's equations of motion in the diffusive limit) is used to generate each configuration from the previous configuration given initial conditions $\rho(q,p,t_0)$, and that rule determines the resulting $\rho(q,p,t)$ for $t > t_0$.

Below we use the notation X to represent the collective phase-space vector (q,p); when discussing the Metropolis algorithm later, the momentum component drops out. (This variable X should not be confused with the random variable X defined in Box 11.1).

Equilibrium Process

The equilibrium ensemble regime is appropriate when $\rho(X,t) = \rho_0(X)$ for some $t > t_0$, as in the Metropolis algorithm (see below). The *ensemble average* is then considered as an estimate for the *time average*, which may be much more complex to follow. This assumption, though very difficult to prove in practice, is known as the *ergodic hypothesis*.

In this statistical equilibrium case, the rule that generates X_{n+1} from X_n need not have a clear physical interpretation. However, to be useful for sampling, the rule must ensure that any starting distribution $\rho(X,t)$ should tend to the stationary density $\rho_0(X)$ and that the system be *ergodic* (i.e., as $t \to \infty$, the system spends

equal times in equal volumes of phase space); see [33] for a rigorous definition. When the rule also obeys *detailed balance* (i.e., moving from state X to Y is as likely as returning to X from Y), an equilibrium process is approached (though biased techniques may violate detailed balance and still approach the right answer through correcting for violations).

These criteria are crucial for constructing practical sampling algorithms for physical systems; efficient sampling of configurational space is another important aspect of computer simulations, especially for large systems where configuration space cannot be sampled exhaustively.

11.5.3 Statistical Ensembles

Common statistical ensembles used in biomolecular simulations are the *canonical* or constant–NVT (N = number of particles, V = volume, T = temperature), *micro-canonical* or constant–NVE (E = energy), *isothermal-isobaric* or constant–NPT (P = pressure), and *grand canonical* or constant–μVT (μ = chemical potential) [2].

Canonical Ensemble and Boltzmann Factor

The probability density function for the *canonical* ensemble is proportional to the Boltzmann factor:

$$\rho_{\mathrm{NVT}}(X) \propto \exp\left(-\beta E(X)\right), \tag{11.34}$$

where E is the total energy of the system and $\beta = (1/k_B\mathrm{T})$.

(In the Metropolis algorithm, it is sufficient to work with the *potential energy*, since the potential energy is independent of momenta; see below).

Hence for two system states X and X', the corresponding probability ratio is:

$$\frac{\rho_{\mathrm{NVT}}(X)}{\rho_{\mathrm{NVT}}(X')} = \exp\left(-\beta \Delta E\right), \tag{11.35}$$

where

$$\Delta E = E(X) - E(X').$$

See the sketch of Figure 11.4. The normalizing factor in the proportionality relation (eq. (11.34)) is the total partition function for all of phase space. That is:

$$\rho_{\mathrm{NVT}}(X) = \frac{1}{(h^{3\mathrm{N}})\,\mathrm{N}!}\,\frac{\exp\left(-\beta E(X)\right)}{Q_{\mathrm{NVT}}} \tag{11.36}$$

where h is Planck's constant, the factor N! accounts for the indistinguishability of the N particles, and Q_{NVT} is the canonical partition function:

$$Q_{\mathrm{NVT}} = \frac{1}{(h^{3N})\,N!}\int \exp\left(-\beta E(x)\right) dx\,, \tag{11.37}$$

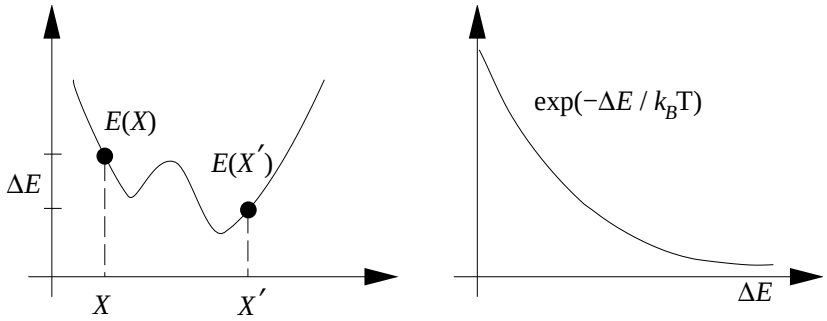

Figure 11.4. The Boltzmann probabilities for two system states X and X' with energies $E(X)$ and $E(X')$ are related at equilibrium by the probability ratio $\exp\left[-\Delta E/k_B\mathrm{T}\right]$.

where x is a point in phase space.

The corresponding means for the system for a function A can then be written as:

$$\langle A(x)\rangle_{\mathrm{NVT}} = \int \rho_{\mathrm{NVT}}(x)\, A(x)\, dx\,. \tag{11.38}$$

The Metropolis algorithm described below is used to generate an appropriate Markov chain (see below) from which the expected value of A is calculated as:

$$\langle A(x)\rangle_{\mathrm{NVT}} = \lim_{M\to\infty} \frac{1}{M}\sum_{i=1}^{M}(A(x_i))\,. \tag{11.39}$$

In other words, the density function is already built into the generation algorithm through an appropriate acceptance probability.

11.5.4 Importance Sampling: Metropolis Algorithm and Markov Chains

To obtain reliable statistical averages, it is essential to use computer time efficiently to concentrate calculations of the configurational-dependent functions in regions that make important contributions. This concept is known as *importance sampling*.

In many applications, it is possible to focus sampling on the *configurational* part of phase space (i.e., E is the potential energy component) since the projection of the corresponding trajectory on the momentum subspace is essentially independent from that projection on the coordinate subspace. *Hence X below is the collective position vector only.*

Markov Chain

Metropolis *et al.* [58] described such an efficient and elegantly simple procedure for the canonical ensemble. In mathematical terms, we generate a *Markov chain*

of molecular states $\{X_1, X_2, X_3, \ldots\}$ constructed to have the limiting distribution $\rho_{\mathrm{NVT}}(X)$. In a Markov chain, each state belongs to a finite set of states contained in the state space $\mathcal{D}_0 \in \mathcal{D}$, and the conditional distribution of each state relative to all the preceding states is equivalent to that conditional distribution relative to the last state:

$$P\{X_{n+1} \in \mathcal{D}_0 \mid X_0, \ldots, X_n\} = P\{X_{n+1} \in \mathcal{D}_0 \mid X_n\}\,;$$

in other words, the outcome X_{n+1} depends only on X_n. The Metropolis algorithm constructs a transition matrix for the Markov chain that is stochastic and ergodic so that the limiting distribution for each state X_i is $\rho_i = \rho_{\mathrm{NVT}}(X_i)$ and thereby generates a phase space trajectory in the canonical ensemble.

This transition matrix is defined by specifying a transitional probability π_{ij} for X_i to X_j so that *microscopic reversibility* is satisfied:

$$\rho_i \pi_{ij} = \rho_j \pi_{ji}\,. \tag{11.40}$$

In other words, the ratio of transitional probabilities depends only on the energy change between states i and j:

$$\frac{\rho_i}{\rho_j} = \frac{\pi_{ji}}{\pi_{ij}} \propto \exp\left(-\beta \Delta E_{ij}\right), \tag{11.41}$$

where $\Delta E_{ij} = E(X_i) - E(X_j)$. See subsection below on MC Moves with examples of biased sampling.

Metropolis Algorithm

Briefly, the Metropolis algorithm generates a trial $\tilde{X}_{i+1}$ from X_i by a system-appropriate random perturbation (satisfying detailed balance) and accepts that state if the corresponding energy is lower. If, however, $E(\tilde{X}_{i+1}) > E(X_i)$, then the new state is accepted with probability $p = \exp\left(-\beta \Delta E\right)$, where $\Delta E = E(\tilde{X}_{i+1}) - E(X_i) > 0$, by comparing p to a uniformly-generated number on (0,1): if $p >$ ran, accept $\tilde{X}_{i+1}$, and if $p \geq$ ran, generate another trial $\tilde{X}_{i+1}$ but recount X_i in the Markov chain (see Fig. 11.4).

The result of this procedure is the acceptance probability at step i of:

$$\begin{aligned} p_{\mathrm{acc,\,MC}} &= \min\left[1, \exp\left(-\beta \Delta E\right)\right] \\ &= \min\left[1, \frac{\rho_{\mathrm{NVT}}(\tilde{X}_{i+1})}{\rho_{\mathrm{NVT}}(X_i)}\right]. \end{aligned} \tag{11.42}$$

In this manner, states with lower energies are always accepted but states with higher energies have a nonzero probability of acceptance too. Consequently, the sequence tends to regions of configuration space with low energies, but the system can always escape to other energy basins.

Simulated Annealing

An extension of the Metropolis algorithm is often employed as a global-minimization technique known as *simulated annealing* where the temperature is

lowered as the simulation evolves in an attempt to locate the global energy basin without getting trapped in local wells.

Simulated annealing can be considered an extension of either MC or molecular/Langevin dynamics. Simulated annealing is often used for refinement of experimental models (NMR or crystallography) with added nonphysical, constraint terms that direct the search to target experimental quantities (e.g., interproton distances or crystallography R factors; see [42], for example, for a review of the application of simulated annealing in such contexts.

Noteworthy is also a Monte Carlo sampling/Minimization (MCM) hybrid method developed by Scheraga and coworkers [52] to search for the global energy minimum of a protein. It generates a large random conformational change followed by local minimization of the potential energy and applies the Metropolis criterion for acceptance or rejection of the new conformation. This procedure is then iterated. Friesner and coworkers have found this MC method to perform well in a variety of applications [19]. Other stochastic global optimization methods have been developed and successfully applied by the Scheraga group [74].

In this connection, see homework 13 for the *deterministic* global optimization approach based on the *diffusion equation* as suggested and implemented for molecular systems by Scheraga and colleagues [73].

Metropolis Algorithm Implementation

The Metropolis algorithm for the canonical ensemble can be implemented with the potential energy E_p rather than the total energy when the target measurement A for MC averaging is velocity independent. This is because the momentum integral can be factored and canceled. From eq. (11.38) combined with eq. (11.34), we expand the state variable x to represent both the momentum (p) and position (q) variables, both over which integration must be performed:

$$\begin{aligned} \langle A(x) \rangle &= \frac{\int \exp\left[-\beta E_k\right] dp \int A(q) \exp\left[-\beta E_p\right] dq}{\int \exp\left[-\beta E_k\right] dp \int \exp\left[-\beta E_p\right] dq} \\ &= \frac{\int A(q) \exp\left[-\beta E_p\right] dq}{\int \exp\left[-\beta E_p\right] dq} . \end{aligned} \tag{11.43}$$

The Metropolis algorithm is summarized below.

Metropolis Algorithm (Canonical Ensemble)

For $i = 0, 1, 2, \ldots$, given X_0:

1. Generate $\tilde{X}_{i+1}$ from X_i by a perturbation technique that satisfies *detailed balance* (i.e., the probability to obtain $\tilde{X}_{i+1}$ from X_i is identical to that going to X_i from $\tilde{X}_{i+1}$).
2. Compute $\Delta E = E(\tilde{X}_{i+1}) - E(X_i)$.
3. **If** $\Delta E \leq 0$ (downhill move), accept X_{i+1} : $X_{i+1} = \tilde{X}_{i+1}$;

Else, set $p = \exp(-\beta \Delta E)$. Then
If $p >$ ran, accept $\tilde{X}_{i+1}$: $X_{i+1} = \tilde{X}_{i+1}$.
Else, reject $\tilde{X}_{i+1}$: $X_{i+1} = X_i$.

4. Continue the i loop.

MC Moves

Specifying appropriate MC moves for step 1 is an art by itself. Ideally, this could be done by perturbing all atoms by independent (symmetric) Gaussian variates with zero mean and variance σ^2, where σ^2 is parameterized to yield a certain acceptance ratio (e.g., 50%). However, in biomolecular simulations, moving all atoms is highly inefficient (that is, leads to a large percentage of rejections) [25], and it has been more common to perturb one or few atoms at each step.

The type of perturbations depends on the system and the energy representation (e.g., rigid or nonrigid molecules, a biomolecule or pure liquid system, Cartesian or internal degrees of freedom). The perturbation can be set as translational, rotational, local, and/or global moves. For example, in the atomistic CHARMM molecular mechanics and dynamics program, protein moves are prescribed from a list of possibilities including rigid-residue translation/rotation, single-atom translation, and single or multiple torsional motions.

For MC simulations of a bead/wormlike chain model of long DNA, we use local translational moves of one bead at a time combined with a rotational move of a chain segment [38]; we must also ensure that no move changes the system's topology (e.g., linking number of a closed chain) for simulating the correct equilibrium ensemble.

Figure 11.5 illustrates such moves for long DNA. Figure 11.6 illustrates corresponding MC (versus Brownian dynamics) distributions of the DNA writhing number (Wr) and the associated mean, as a function of length for two salt environments. Figure 11.7 demonstrates how a faulty move (like moving only a subset of the DNA beads instead of all) can corrupt the probability distributions of Wr and the radius of gyration (Rg). Not only do we note a corruption of the distributions when incorrect MC protocols are used, but a large sensitivity to the initial configuration (sharp distributions around starting configurations).

The rule of thumb usually employed in MC simulations is to aim for a perturbation in Step 1 (e.g., displacement magnitude or the variance σ^2 associated with the random Gaussian variate) that yields about 50% acceptance. Thus, we seek to balance too small a perturbation that moves the system in state space slowly with too large a perturbation that yields high trial-energy configurations, most of which are rejected. *However, the appropriate percentage depends on the application and is best determined by experimentation guided by known outcomes of the statistical means sought.* Much smaller acceptance probabilities may be perfectly adequate for some systems.

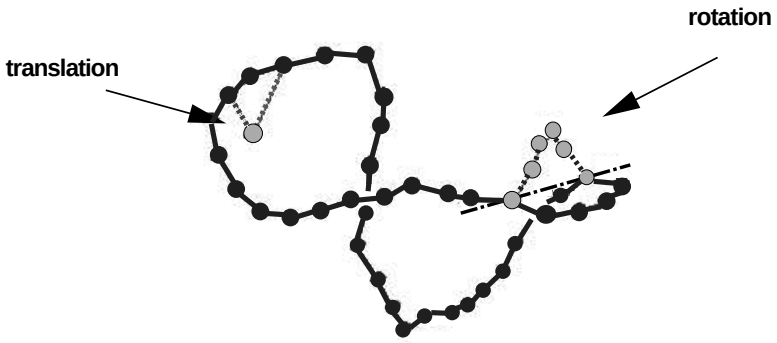

Figure 11.5. Translational and rotational MC moves for a bead model of DNA.

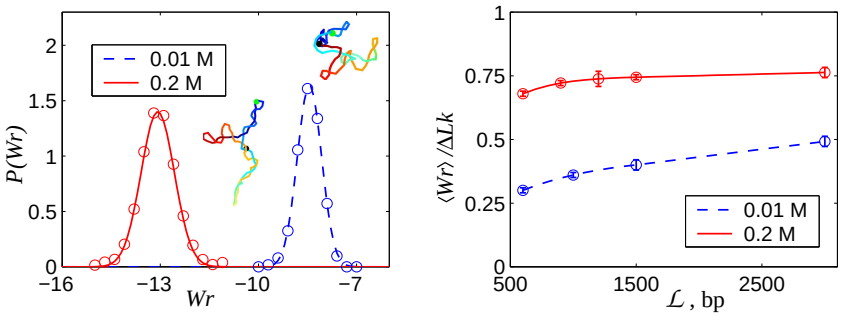

Figure 11.6. MC distributions (curves) versus Brownian dynamics data (circles) of the writhing number (Wr) distribution of supercoiled DNA (left), and mean values of Wr (normalized by the linking number difference) as a function of the DNA length (right), at two salt concentrations. The DNA superhelical density is $\sigma = -0.06$ in both panels, and the left panel involves DNA of length 3000 base pairs. Error bars for BD are shown only if they are larger than the circle symbol.

11.6 Hybrid Monte Carlo

11.6.1 Exploiting Strengths of MC and MD

To enhance the efficiency of MC simulations, a simple idea emerged that attempts to combine the favorable properties of molecular dynamics (MD) simulations — sampling phase space in a directed manner guided by the shape of the energy gradient — with that of MC — sampling phase space more globally. Ideally, following conformation space by MD would generate a correct Boltzmann distribution of states, but the relatively short simulation lengths that are possible (see MD chapters) imply local rather than global sampling.

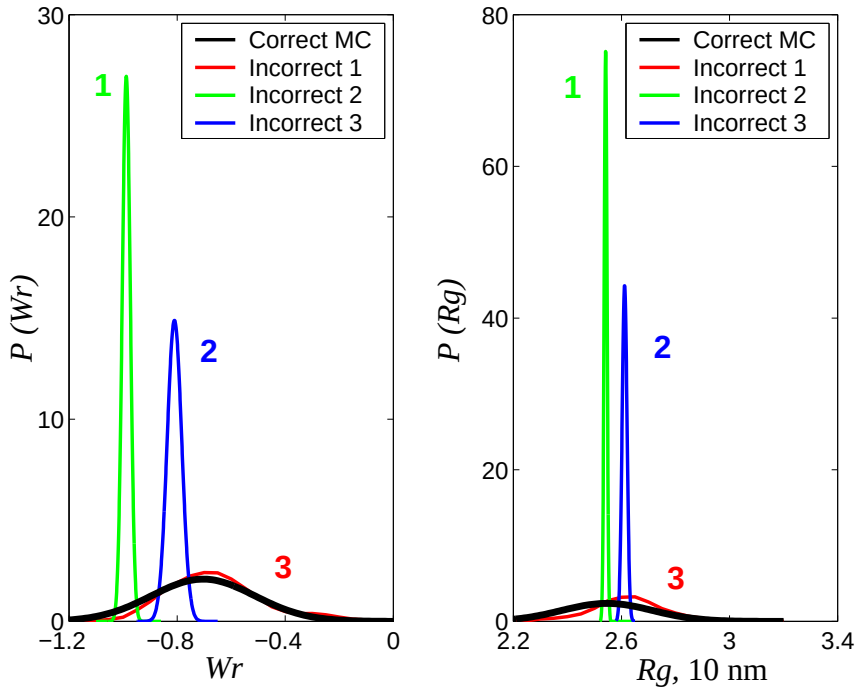

Figure 11.7. DNA writhing number Wr (left) and radius of gyration Rg distributions as generated by one correct versus three incorrect MC procedures. The incorrect MC protocols allow only a subset of the beads to move: 25 out of 30 ("Incorrect 1"), or 5 out of 30 ("Incorrect 2, 3"); the last two schemes are started from different initial points. The DNA modeled has 600 base pairs, with superhelical density of $\sigma = -0.04$, and the MC simulations consist of ten million steps.

Though in theory a good MC protocol would sample configuration space exhaustively, this becomes more difficult and inefficient in practice as the system size increases. When the cost of evaluating the energy function for large biomolecular systems is also a factor, millions of MC steps (with possibly many rejections) can become quite expensive.

11.6.2 Overall Idea

The idea to overcome these MC difficulties with *hybrid Monte Carlo* (HMC) [18] is to combine global updates in position space via MD with reasonable acceptance criteria by MC.

The first step of a hybrid MC method uses a molecular dynamics framework to specify the system's candidate move: $X[i\Delta t] \longrightarrow X[(i+1)\Delta t]$ where Δt is the timestep. The MD algorithm must be *time reversible* and volume preserving so as to ensure detailed balance. The commonly used symplectic Verlet method satisfies

this requirement (see Chapters 12 and 13). Since the recursive MD recipe relies on the velocities in addition to positions, the required velocity vectors $V = \mathbf{M}^{-1}P$ (P here designates momentum, not to be confused with the symbol used earlier for *probability*) are generated from a Gaussian distribution so as to obtain the target kinetic energy at the temperature T assuming energy equipartition. That is, the velocity components are drawn from a Gaussian distribution proportional to the Boltzmann factor applied to the kinetic energy:

$$\rho(P) \propto \exp\left[-\beta\, E_k(P)\right]. \tag{11.44}$$

Following this MD-guided step, the second step of the hybrid MC method applies the standard Metropolis acceptance criterion where the energy in the Boltzmann factor is the *Hamiltonian* (potential *plus* kinetic energy). Namely, the Metropolis criterion is applied to accept the new candidate $\tilde{X}_{i+1}$ with probability

$$\begin{aligned} p_{\text{acc, HMC, }\rho} &= \min\left[1, \exp\left[-\beta\left(H(\tilde{X}_{i+1}, \tilde{P}_{i+1}) - H(X_i, P_i)\right)\right]\right] \\ &= \min\left[1, \frac{\exp\left[-\beta E_p(\tilde{X}_{i+1})\right]\, \exp\left[-\beta E_k(\tilde{P}_{i+1})\right]}{\exp\left[-\beta E_p(X_i)\right]\, \exp\left[-\beta E_k(P_i)\right]}\right] \\ &= \min\left[1, \frac{\rho_{\text{NVT}}(\tilde{X}_{i+1}) \exp\left[-\beta E_k(\tilde{P}_{i+1})\right]}{\rho_{\text{NVT}}(X_i) \exp\left[-\beta E_k(P_i)\right]}\right]. \end{aligned} \tag{11.45}$$

HMC methods have been quite successful for biomolecular sampling, and many other extensions to general ensembles and hybrid methods have been described.

Note that *generalized ensembles* can be emulated by HMC methods by applying weighting factors to the Metropolis-generated configurations. That is, to sample from a general ensemble with $\mu(x)$ rather than $\rho(x)$, we adjust the acceptance criteria of eq. (11.45) to be:

$$p_{\text{acc, HMC, }\mu} = \min\left[1, \frac{\mu(\tilde{X}_{i+1}) \exp\left[-\beta E_k(\tilde{P}_{i+1})\right]}{\mu(X_i) \exp\left[-\beta E_k(P_i)\right]}\right]. \tag{11.46}$$

The weighting must be accomplished to maintain *detailed balance*.

11.6.3 Variants and Other Hybrid Approaches

MC variants include *Smart MC*, *J-walking*, stochastic dynamics, umbrella sampling, various methods to enhance barrier crossing events (e.g., the transition path sampling method of Chandler and coworkers [6]), and potential-modification approaches such as smoothing techniques. See [5], for example, for a perspective. Interesting also are papers describing an adaptive-temperature HMC method for a mixed-canonical ensemble that enhances sampling [21] and an efficient Monte Carlo method based on known molecular minima for medium-sized flexible molecules [84]. The simple MC/MD combination [43] — moving some particles by

MC rules and others by MD — may be more effective for solvated biomolecular systems than either MC or MD alone.

There has been continued discussion on the relative merits of MC and MD for biomolecules (e.g., [13, 39]). The limits of MD for simulating slow molecular events is clearly recognized, as is its high computational cost, though only MD can ultimately yield detailed dynamic information such as folding pathways and rates of conformational changes. At present, a practitioner is well-advised to use a combination of MC, MD, and local and global minimization algorithms, as appropriate, for the problems at hand.

Table 11.1. Results of computing π by MC integration with various random number generators and sample sizes by the procedure described in the text, subroutine monte (on an SGI R12K/300 MHz Octane processor). The value σ is computed from 100 runs for each NSTEP value, except for 10^{12}, for which it is based on one run. rand has $\tau \approx 2^{31}$; 10^9 calls require 771 seconds (13 min.). drand48 has $\tau \approx 2^{48}$; 10^9 calls require 656 seconds (11 min.). clg4 has $\tau \approx 2^{31}$; 10^9 calls require 5205 seconds (87 min.). lfg has $\tau \approx 2^{64}$; 10^9 calls require 2106 seconds (35 min.). See also Figure 11.3.

Nstep	Estimate	Error	σ, s.d.
rand, IRIX 6.5 system library			
10^2	3.1604000000000E+00	$1.8807346410208\mathrm{E}-02$	1.59E−01
10^3	3.1346800000000E+00	$-6.9126535897936\mathrm{E}-03$	4.75E−02
10^4	3.1383680000000E+00	$-3.2246535897933\mathrm{E}-03$	1.79E−02
10^5	3.1416736000000E+00	$8.0946410208504\mathrm{E}-05$	5.22E−03
10^6	3.1416646400000E+00	$7.1986410206115\mathrm{E}-05$	1.61E−03
10^7	3.1416831800000E+00	$9.0526410207570\mathrm{E}-05$	4.74E−04
10^8	3.1416971676000E+00	$1.0451401020672\mathrm{E}-04$	1.65E−04
10^9	3.1417139545200E+00	$1.2130093020657\mathrm{E}-04$	1.37E−05
drand48, IRIX 6.5 system library			
10^2	3.1408000000000E+00	$-7.9265358979264\mathrm{E}-04$	1.72E−01
10^3	3.1456400000000E+00	$4.0473464102098\mathrm{E}-03$	5.62E−02
10^4	3.1431880000000E+00	$1.5953464102063\mathrm{E}-03$	1.53E−02
10^5	3.1418572000000E+00	$2.6454641020690\mathrm{E}-04$	5.92E−03
10^6	3.1416580000000E+00	$6.5346410206946\mathrm{E}-05$	1.61E−03
10^7	3.1415415040000E+00	$-5.1149589793908\mathrm{E}-05$	4.97E−04
10^8	3.1415733104000E+00	$-1.9343189793464\mathrm{E}-05$	1.79E−04
10^9	3.1415892614400E+00	$-3.3921497935019\mathrm{E}-06$	4.70E−05
10^{12}	3.1415928451280E+00	$1.9153820707274\mathrm{E}-07$	1.00E−06
clg4, based on four linear congruential generators [51]			
10^2	3.1600000000000E+00	$1.8407346410206\mathrm{E}-02$	1.51E−01
10^3	3.1413200000000E+00	$-2.7265358979189\mathrm{E}-04$	4.98E−02
10^4	3.1405440000000E+00	$-1.0486535897933\mathrm{E}-03$	1.74E−02
10^5	3.1416476000000E+00	$5.4946410205758\mathrm{E}-05$	4.94E−03
10^6	3.1416064800000E+00	$1.3826410206530\mathrm{E}-05$	1.59E−03
10^7	3.1415008760000E+00	$-9.1777589792841\mathrm{E}-05$	4.92E−04
10^8	3.1415751016000E+00	$-1.7551989793141\mathrm{E}-05$	1.82E−04
10^9	3.1415925054000E+00	$-1.4818979332532\mathrm{E}-07$	5.05E−05
lfg, SPRNG package, modified lagged-Fibonacci generator			
10^2	3.1372000000000E+00	$-4.3926535897922\mathrm{E}-03$	1.70E−01
10^3	3.1414400000000E+00	$-1.5265358979244\mathrm{E}-04$	5.20E−02
10^4	3.1442160000000E+00	$2.6233464102083\mathrm{E}-03$	1.55E−02
10^5	3.1431008000000E+00	$1.5081464102074\mathrm{E}-03$	5.23E−03
10^6	3.1419958000000E+00	$4.0314641020611\mathrm{E}-04$	1.64E−03
10^7	3.1415909920000E+00	$-1.6615897910910\mathrm{E}-06$	5.46E−04
10^8	3.1415866680000E+00	$-5.9855897953653\mathrm{E}-06$	1.72E−04
10^9	3.1415854581200E+00	$-7.1954697937748\mathrm{E}-06$	5.03E−05

References

[1] L. Adams and J. L. Nazareth, editors. *Linear and Nonlinear Conjugate Gradient-Related Methods*. SIAM, Philadelphia, PA, 1996.

[2] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, NY, 1990.

[3] S. L. Anderson. Random number generators on vector supercomputers and other advanced architectures. *SIAM Rev.*, 32:221–251, 1990.

[4] D. Beard and T. Schlick. Modeling salt-mediated electrostatics of macromolecules: The algorithm DiSCO (Discrete Charge Surface Charge Optimization) and its application to the nucleosome. *Biopolymers*, 58:106–115, 2001.

[5] B. J. Berne and J. E. Straub. Novel methods of sampling phase space in the simulation of biological systems. *Curr. Opin. Struct. Biol.*, 7:181–189, 1997.

[6] P. G. Bolhuis, D. Chandler, C. Dellago, and P. L. Geissler. Transition path sampling: Throwing ropes over rough mountain passes, in the dark. *Annu. Rev. Phys. Chem*, 53:291–318, 2002.

[7] J. M. Borwein and A. S. Lewis. *Convex Analysis and Nonlinear Optimization. Theory and Examples*, volume 3 of *Canadian Mathematical Society (CMS) Books in Mathematics*. Springer-Verlag, New York, NY, 2000.

[8] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, NY, 1987.

[9] S. Broyde and B. E. Hingerty. Effective computational strategies for determining structures of carcinogen damaged DNA. *J. Comput. Phys.*, 151:313–332, 1999.

[10] R. H. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Stat. Comput.*, 16:1190–1208, 1995.

[11] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Prog.*, 63:129–156, 1994.

[12] R. H. Byrd, J. Nocedal, and C. Zhu. Towards a discrete Newton method with memory for large-scale optimization. In G. Di Pillo and F. Giannessi, editors, *Nonlinear Optimization and Applications*. Plenum, 1996.

[13] J. B. Clarage, T. Romo, B. K. Andrews, B. M. Pettitt, and G. N. Philipps, Jr. A sampling problem in molecular dynamics simulations of macromolecules. *Proc. Natl. Acad. Sci. USA*, 92:3288–3292, 1995.

[14] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. 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.

[15] R. S. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Math. Prog.*, 26:190–212, 1983.

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

[17] P. Derreumaux, G. Zhang, B. Brooks, and T. Schlick. A truncated-Newton method adapted for CHARMM and biomolecular applications. *J. Comput. Chem.*, 15:532–552, 1994.

[18] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Phys. Lett. B*, 195:216–222, 1987.

[19] V. A. Eyrich, D. M. Standley, and R. A. Friesner. Prediction of protein tertiary structure to low resolution: Performance for a large and structurally diverse test set. *J. Mol. Biol.*, 14:725–742, 1999.

[20] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong. Monte Carlo simulations: Hidden errors from "good" random number generators. *Phys. Rev. Lett.*, 69:3382–3384, 1992.

[21] A. Fischer, F. Cordes, and C. Sch¨utte. Hybrid Monte Carlo with adaptive temperature in mixed-canonical ensemble: Effi cient conformational analysis of RNA. *J. Comput. Chem.*, 19:1689–1697, 1998.

[22] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Tiptree, Essex, Great Britain, second edition, 1987.

[23] C. A. Floudas and P. Pardalos, editors. *Optimization in Computational Chemistry and Molecular Biology: Local and Global Approaches*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[24] A. Forsgren, P. E. Gill, and M. H. Wright. Interior methods for nonlinear optimization. *SIAM Rev.*, 44:525–597, 2002.

[25] D. Frenkel and B. Smit. *Understanding Molecular Simulations. From Algorithms to Applications*. Academic Press, San Diego, CA, 1996.

[26] J. C. Gilbert and C. Lemarechal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Math. Prog. B*, 45:407–435, 1989.

[27] J. C. Gilbert and J. Nocedal. Global convergence properties of conjugate gradient methods for optimization. Technical Report 1268, Institut National de Recherche en Informatique et en Automatique, January 1991.

[28] P. E. Gill and M. W. Leonard. Reduced-Hessian quasi-Newton methods for unconstrained optimization. *SIAM J. Optim.*, 12:209–237, 2001.

[29] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, England, 1983.

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

[31] J. A. González and R. Pino. A random number generator based on unpredictable chaotic functions. *Comput. Phys. Comm.*, 120:109–114, 1999.

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

[33] T. L. Hill. *An Introduction to Statistical Thermodynamics*. Dover, New York, NY, 1986.

[34] B. E. Hingerty, S. Figueroa, T. L. Hayden, and S. Broyde. Prediction of DNA structure from sequence: A buildup technique. *Biopolymers*, 8:1195–1222, 1989.

[35] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization. Algorithms I*, volume 305 of *Grundlehren der mathematischen Wissenschaften. A Series of Comprehensive Studies in Mathematics*. Springer-Verlag, Berlin and Heidelberg, 1993.

[36] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization. Algorithms II*, volume 306 of *Grundlehren der mathematischen Wissenschaften. A Series of Comprehensive Studies in Mathematics*. Springer-Verlag, Berlin and Heidelberg, 1993.

[37] B. L. Holian, O. E. Percus, T. T. Warnock, and P. A. Whitlock. Pseudorandom number generator for massively parallel molecular-dynamics applications. *Phys. Rev. E*, 50:1607–1615, 1994.

[38] H. Jian. *A Combined Wormlike-Chain and Bead Model for Dynamic Simulations of Long DNA*. PhD thesis, New York University, Department of Physics, New York, NY, October 1997.

[39] W. L. Jorgensen and J. Tirado-Rives. Monte Carlo vs. molecular dynamics for conformational sampling. *J. Phys. Chem.*, 100:14508–14513, 1996.

[40] M. H. Kalos and P. A. Whitlock. *Monte Carlo Methods. Volume I: Basics*. John Wiley & Sons, New York, NY, 1986.

[41] D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Methods*. Addison-Wesley, Reading, Massachusetts, second edition, 1981.

[42] A. Korostelev, R. Bertram, and M. S. Chapman. Simulated-annealing real-space refi nement as a tool in model building. *Acta Cryst.*, D58:761–767, 2002.

[43] L. J. LaBerge and J. C. Tully. A rigorous procedure for combining molecular dynamics and Monte Carlo simulation algorithms. *Chem. Phys.*, 260:183–191, 2000.

[44] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Series in Industrial Engineering and Management Science. McGraw-Hill, Boston, MA, third edition, 2000.

[45] P. L'Ecuyer. Uniform random number generation. *Ann. Oper. Res.*, 53:77–120, 1994.

[46] P. L'Ecuyer. Maximally-equidistributed combined Tausworthe generators. *Math. Comput.*, 65:203–213, 1996.

[47] P. L'Ecuyer. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *Informs J. Comput.*, 9:57–60, 1997.

[48] P. L'Ecuyer. Random number generation. In J. Banks, editor, *Handbook on Simulation*, chapter 4, pages 93–137. John Wiley & Sons, New York, NY, 1998.

[49] P. L'Ecuyer. Good parameter sets for combined multiple recursive random number generators. *Oper. Res.*, 47:159–164, 1999.

[50] P. L'Ecuyer. Tables of maximally-equidistributed combined LFSR generators. *Math. Comput.*, 68:261–269, 1999.

[51] P. L'Ecuyer and T. H. Andres. A random number generator based on the combination of four LCGs. *Mathematics and Computers in Simulation*, 44:99–107, 1997.

[52] Z. Li and H. A. Scheraga. Monte Carlo-minimization approach to the multiple-minima problem in protein folding. *Proc. Natl. Acad. Sci. USA*, 84:6611–6615, 1987.

[53] K. B. Lipkowitz. Abuses of molecular mechanics. Pitfalls to avoid. *J. Chem. Educ.*, 72:1070–1075, 1995.

[54] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large-scale optimization. *Math. Prog. B*, 45:503–528, 1989.

[55] D.G. Luenberger. *Linear and Nolinear Programming*. Addison Wesley, Reading, Massachusetts, 1984.

[56] G. Marsaglia and L.-H Tsay. Matrices and the structure of random number sequences. *Lin. Alg. Appl.*, 67:147–156, 1985.

[57] M. Mascagni. Some methods of parallel pseudorandom number generation. In M. T. Heath, A. Ranade, and R S. Schreiber, editors, *Algorithms for Parallel Processing*, volume 105 of *IMA Volumes in Mathematics and Its Applications*, pages 277–288. Springer-Verlag, New York, NY, 1999.

[58] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[59] J. L. Morales and J. Nocedal. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Opt.*, 10:1079–1096, 2000. (also Technical Report 97/07, Optimization Technology Center, Northwestern University, 1997).

[60] J. J. Moré and S. J. Wright. *Optimization Software Guide*, volume 14 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, PA, 1993. See www.mcs.anl.gov/otc/Guide/ and www.mcs.anl.gov/otc/Guide/SoftwareGuide/ for updated information on the software guide.

[61] S. G. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and the truncated-Newton method for large-scale optimization. *SIAM J. Opt.*, 1:358–372, 1991.

[62] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[63] J. Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numerica*, 1:199–242, 1992.

[64] J. Nocedal. Large-scale unconstrained optimization. In A. Watson and I. Duff, editors, *The State of the Art in Numerical Analysis*, pages 311–338. Oxford University Press, 1997.

[65] J. Nocedal, A. Sartenaer, and C. Zhu. On the behavior of the gradient norm in the steepest descent method. Technical report, CERFACS, Toulouse, France (May 2000), 2000.

[66] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Verlag, New York, NY, 1999.

[67] R. E. Odeh and J. O. Evans. The percentage points of the normal distribution. *App. Stat.*, 23:96–97, 1974.

[68] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, Philadelphia, PA, 2001.

[69] P. M. Pardalos, D. Shalloway, and G. Xue, editors. *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.

[70] S. C. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Comm. ACM*, 31:1192–1201, 1988.

[71] O. E. Percus and M. H. Kalos. Random number generators for MIMD parallel processors. *J. Paral. Dist. Comput.*, 6:477–497, 1989.

[72] O. E. Percus and J. K. Percus. Intrinsic relations in the structure of linear congruential generators modulo 2^{β}. *Stat. Prob. Let.*, 15:381–383, 1992.

[73] L. Piela, J. Kostrowicki, and H. A. Scheraga. The multiple-minima problem in conformational analysis of molecules. deformation of the potential energy hypersurface by the diffusion equation method. *J. Phys. Chem.*, 93:3339–3346, 1989.

[74] J. Pillardy, C. Czaplewiski, A. Liwo, J. Lee, D. R. Ripoll, R. Kaźmierkiewicz, S. Oldziej, W. J. Wedemeyer, K. D. Gibson, Y. A. Arnautova, J. Saunders, Y.-J. Ye, and H. A. Scheraga. Recent improvements in prediction of protein structure by global optimization of a potential energy function. *Proc. Natl. Acad. Sci. USA*, 98:2329–2333, 2001.

[75] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Math. Prog.*, 12:241–254, 1977.

[76] F. J. Resende and B. V. Costa. Using random number generators in Monte Carlo simulations. *Phys. Rev. E*, 58:5183–5184, 1998.

[77] T. Schlick. Optimization methods in computational chemistry. In K. B. Lipkowitz and D. B. Boyd, editors, *Reviews in Computational Chemistry*, volume III, pages 1–71. VCH Publishers, New York, NY, 1992.

[78] T. Schlick. Geometry optimization. In P. von Ragué Schleyer (Editor-in Chief), N. L. Allinger, T. Clark, J. Gasteiger, P. A. Kollman, and H. F. Schaefer, III, editors, *Encyclopedia of Computational Chemistry*, volume 2, pages 1136–1157. John Wiley & Sons, West Sussex, England, 1998.

[79] T. Schlick and A. Fogelson. TNPACK —A truncated Newton minimization package for large-scale problems: I. Algorithm and usage. *ACM Trans. Math. Softw.*, 14:46–70, 1992.

[80] T. Schlick and A. Fogelson. TNPACK —A truncated Newton minimization package for large-scale problems: II. Implementation examples. *ACM Trans. Math. Softw.*, 14:71–111, 1992.

[81] T. Schlick and M. L. Overton. A powerful truncated Newton method for potential energy functions. *J. Comput. Chem.*, 8:1025–1039, 1987.

[82] T. Schlick and G. Parks. DOE computational sciences education project, 1994. Chapter on Mathematical Optimization. URL: csep1.phy.ornl.gov/.

[83] R. B. Schnabel and T. Chow. Tensor methods for unconstrained optimization. *SIAM J. Opt.*, 1:293–315, 1991.

[84] H. Senderowitz and W. C. Still. MC(JBW): Simple but smart Monte Carlo algorithm for free energy simulations of multiconformational molecules. *J. Comput. Chem.*, 19:1736–1745, 1998.

[85] D. F. Shanno and K. H. Phua. Remark on Algorithm 500: Minimization of unconstrained multivariate functions. *ACM Trans. Math. Softw.*, 6:618–622, 1980.

[86] M. H. Wright. What, if anything, is new in optimization? In J. M. Ball and J. C. R. Hunt, editors, *ICIAM'99: Proceedings of the 4th International Congress on Industrial and Applied Mathematics*, pages 259–270, Oxford, England, 2000. Oxford University Press. Also available in modified form as Technical Report 00-4-08, Bell Laboratories, Computing Sciences Research Center, Murray Hill, New Jersey 07074; cm.bell-labs.com/cm/cs/doc/00/4-08.ps.gz.

[87] D. Xie and T. Schlick. Efficient implementation of the truncated Newton method for large-scale chemistry applications. *SIAM J. Opt.*, 10(1):132–154, 1999.

[88] D. Xie and T. Schlick. Remark on the updated truncated Newton minimization package, *Algorithm 702*. *ACM Trans. Math. Softw.*, 25(1):108–122, 1999.

[89] D. Xie and T. Schlick. A more lenient stopping rule for line search algorithms. *Opt. Math. Softw.*, 17:683–700, 2002.

[90] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B, FORTRAN subroutines for large scale bound constrained optimization. *ACM Trans. Math. Softw.*, 23:550–560, 1997.

Index