Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. James C. Spall
Copyright © 2003 John Wiley & Sons, Inc.,
ISBN: 0-471-33052-3

CHAPTER 1

STOCHASTIC SEARCH AND OPTIMIZATION: MOTIVATION AND SUPPORTING RESULTS

Preparation is required before starting any journey. This chapter lays the groundwork for our study of stochastic algorithms. Section 1.1 introduces the two basic—and closely related—problems that are the focus of this book. Section 1.2 summarizes some of the fundamental issues one faces in tackling a problem in stochastic search and optimization. This includes some of the immutable limits to what can be expected in any practical problem. Although classical deterministic optimization is not a prerequisite to this book, a casual understanding of some deterministic methods will be useful here, especially where the methods are analogous to certain stochastic approaches. Hence, Sections 1.3 and 1.4 provide a brief review of some important results related to deterministic search and optimization. Section 1.5 offers some concluding remarks.

Although the majority of the results in this book do not require prerequisites beyond multivariate calculus, basic matrix theory, and introductory probability and statistics, it is not possible to fully analyze and understand some of the stochastic algorithms here without some more advanced mathematics. In that sense, Appendices A, B, and C may be considered companions to this introductory chapter. These appendices review some of the most important prerequisites and introduce a limited amount of material that goes slightly beyond the prerequisites. Appendix A is a summary of some results from multivariate analysis and matrix theory. Appendix B summarizes some of the standard one- and two-sample statistical tests that will be used here. Appendix C summarizes some essential concepts from probability theory.

And with that, let the journey begin!

1.1 INTRODUCTION

1.1.1 General Background

- Managers at a firm are faced with making short- and long-term investment decisions in order to increase profit.
- An aerospace engineer runs computer simulations and conducts wind tunnel tests to refine the design of a missile or aircraft.

- Researchers at a pharmaceuticals firm design laboratory experiments to extract the maximum information about the efficacy of a new drug.
- Traffic engineers in a municipality set the timing strategy for the signals in a traffic network to minimize the delay incurred by vehicles in the network.

The list above is a tiny sample of problems for which stochastic search and optimization have been used. All such problems involve tradeoffs. Choosing among the tradeoffs in the best way is the essence of multivariate search and optimization problems.

Mathematical techniques of search and optimization are aimed at providing a formal means for making the best decisions in problems of the type above. Given the difficulties in many real-world problems and the inherent uncertainty in information that may be available for carrying out the task, *stochastic* search and optimization methods—the theme of this book—have been playing a rapidly growing role.

Much of this book focuses on two general—and closely related—mathematical problems. Let Θ be the domain of allowable values for a vector $\boldsymbol{\theta}$.[1] The two main problems of interest are:

Problem 1	Find the value(s) of a vector $\boldsymbol{\theta} \in \Theta$ that minimize a scalar-valued *loss function* $L(\boldsymbol{\theta})$
	— or —
Problem 2	Find the value(s) of $\boldsymbol{\theta} \in \Theta$ that solve the equation $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$ for some vector-valued function $\boldsymbol{g}(\boldsymbol{\theta})$.

The vector $\boldsymbol{\theta}$ represents a collection of "adjustables" that one is aiming to pick in the best way. The loss function $L(\boldsymbol{\theta})$ is a scalar measure that summarizes the performance of the system for a given value of the adjustables. The domain Θ reflects allowable values (constraints) on the elements of $\boldsymbol{\theta}$. Other common names for the loss function are *performance measure*, *objective function*, *fitness function*, or *criterion*. While Problem 1 above refers to *minimizing* a loss function, a maximization problem (e.g., maximizing profit) can be trivially converted to a minimization problem by changing the sign of the criterion. This book focuses on the problem of minimization. The root-finding function $\boldsymbol{g}(\boldsymbol{\theta})$ (which is generally a vector) often arises via calculating the gradient (derivative) of the loss function (i.e., $\boldsymbol{g}(\boldsymbol{\theta}) = \partial L(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$). More generally, $\boldsymbol{g}(\boldsymbol{\theta})$ may represent a collection of functions that are derived from physical principles related to the system under study.

[1]One digression before proceeding to mention a notational convention for this and the remaining chapters: As discussed in the "Frequently Used Notation" section near the back of the book, a vector will be treated as a column vector and a superscript T on a matrix or vector denotes the transpose operation. So, $\boldsymbol{\theta}^T$ is a row vector for a vector $\boldsymbol{\theta}$. All matrices and vectors are in boldface.

Versions of the two problems above arise in countless areas of practice. A huge amount of effort in society is devoted to "doing the most with the least," which is the essence of the optimization objective. More specifically, one faces problems 1 and/or 2 in virtually all areas of engineering, the physical sciences, business, and medicine. In many problems of practical interest, mathematical search *algorithms*—step-by-step procedures usually implemented on a computer—are used to produce a solution. The focus on algorithms, of course, is central to this book.

In many applications, the two problems above are effectively equivalent and one may choose a formulation oriented directly at optimization (Problem 1) or at root-finding (Problem 2). The choice of formulation typically depends on issues such as the basic applied problem structure and data format, the available and relevant search algorithms, the proclivities of the analyst and traditions of his/her field, and the available software. For example, if one starts with a problem of minimizing $L = L(\boldsymbol{\theta})$ (Problem 1) for a differentiable loss function, standard methods of calculus can sometimes convert this to a problem of finding a solution (root) of the vector equation $\boldsymbol{g}(\boldsymbol{\theta}) = \partial L/\partial\boldsymbol{\theta} = \boldsymbol{0}$ (Problem 2). There may be advantages to such a conversion if the appropriate mathematical assumptions are satisfied and if the information required to compute the gradient is available.

Conversely, Problem 2 can be converted directly into an optimization problem by noting that a $\boldsymbol{\theta}$ such that $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$ is equivalent to a $\boldsymbol{\theta}$ such that $\|\boldsymbol{g}(\boldsymbol{\theta})\|$ is minimized for any vector norm $\|\cdot\|$ (the most common norm is the Euclidean norm $\|\boldsymbol{g}\| = \sqrt{\boldsymbol{g}^T\boldsymbol{g}}$). This conversion from Problem 2 to Problem 1 is used to advantage in a wastewater treatment example in Section 2.3 and in a second-order stochastic approximation method discussed in Section 4.5. We clearly see that there is a close connection between optimization and root-finding, even in root-finding problems that are not motivated by optimization problems per se. Hence, to avoid the cumbersome requirement of separately discussing both optimization and root-finding in the text, most (but not all!) of the discussion in this book will be on search algorithms for optimization problems, with the expectation that the appropriate interpretation as a root-finding problem will be transparent. There will, however, sometimes be cases where a point needs to be highlighted relevant to a unique structure for root-finding. Chapter 4, in particular, discusses such cases.

The two remaining subsections in this section define some basic quantities and demonstrate their role in stochastic search and optimization.

1.1.2 Formal Problem Statement; General Types of Problems and Solutions; Global versus Local Search

As mentioned above, much of this book focuses on search and optimization problems associated with minimizing a loss function $L = L(\boldsymbol{\theta})$. This optimization problem can be formally represented as finding the set:

$$\Theta^* \equiv \arg\min_{\boldsymbol{\theta}\in\Theta} L(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* \in \Theta : L(\boldsymbol{\theta}^*) \le L(\boldsymbol{\theta}) \text{ for all } \boldsymbol{\theta} \in \Theta\}, \qquad (1.1)$$

where $\boldsymbol{\theta}$ is a p-dimensional vector of parameters that are being adjusted and $\Theta \subseteq \mathbb{R}^p$ is the domain for $\boldsymbol{\theta}$ representing constraints on allowable values for $\boldsymbol{\theta}$. ($\mathbb{R}^p$ is Euclidean space of dimension p; see Appendix A.) The "$\arg\min_{\boldsymbol{\theta}\in\Theta}$" statement in (1.1) may equivalently be read as: Θ^* is the set of values $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ ($\boldsymbol{\theta}$ the "argument" in "arg min") that minimize $L(\boldsymbol{\theta})$ subject to $\boldsymbol{\theta}^*$ satisfying the constraints represented in the set Θ. The elements $\boldsymbol{\theta}^* \in \Theta^* \subseteq \Theta$ are equivalent solutions in the sense that they yield identical values of the loss function.

The solution set Θ^* in (1.1) may be a unique point, a countable (finite or infinite) collection of points, or a set containing an uncountable number of points. The three examples below show simple cases illustrating these types of solution sets Θ^*.

Example 1.1—Θ^* contains unique solution. Suppose that $L(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{\theta}$ and $\Theta = \mathbb{R}^p$ (i.e., $\boldsymbol{\theta}$ is unconstrained). The value $\boldsymbol{\theta} = \mathbf{0}$ uniquely minimizes L. Hence, Θ^* is the single point $\boldsymbol{\theta}^* = \mathbf{0}$. ❑

Example 1.2—Θ^* has countable (finite or infinite) number of points. Let θ be a scalar and $L(\theta) = \sin(\theta)$. If $\Theta = [0, 4\pi]$, then $\sin(\theta) = -1$ (its minimum) at the points $\Theta^* = \{3\pi/2,\ 7\pi/2\}$, a countable set with a finite number (two) of elements. On the other hand, if $\Theta = \mathbb{R}^1$, then $\Theta^* = \{\ldots, -5\pi/2, -\pi/2, 3\pi/2, 7\pi/2, \ldots\}$, a countable set with an infinite number of elements. ❑

Example 1.3—Θ^* has uncountable number of points. Suppose that $L(\boldsymbol{\theta}) = (\boldsymbol{\theta}^T\boldsymbol{\theta} - 1)^2$ and $\Theta = \mathbb{R}^p$. This loss function is minimized when $\boldsymbol{\theta}^T\boldsymbol{\theta} = 1$, which is the set of points lying on the surface of a p-dimensional sphere having radius 1. When $p \ge 2$, Θ^* is an uncountable (but bounded) set. ❑

Unlike the simple motivational examples above, this book focuses on problems where L is sufficiently complex so that it is not possible to obtain a closed-form analytical solution to (1.1). This is far and away the most common setting for large-scale optimization problems encountered in practice. Hence to solve for at least one $\boldsymbol{\theta}^*$ in (1.1), an iterative algorithm is used—a step-by-step procedure for moving from an initial guess (or set of guesses) about likely values of $\boldsymbol{\theta}^*$ to a final value that is expected to be closer to the true $\boldsymbol{\theta}^*$ than the initial guess(es).

Because any one value in Θ^* is—with respect to L—as good as any other, it is common in practice to seek only one $\boldsymbol{\theta}^*$ in the algorithmic search process. That is, there is often no reason to use scarce resources to determine the full set Θ^*. Further, there may be "side considerations" that cannot be quantifiably represented in a loss function that may make it desirable to seek only one solution. For example, one solution may be more consistent with historical precedent or physical intuition and may therefore be preferable. Finally, in many problems, Θ^* may, in fact, contain only one $\boldsymbol{\theta}^*$ (i.e., L may have a *unique* minimum).

This book considers both continuous and discrete problems, reflecting the available methods in stochastic search and optimization. Figure 1.1 depicts three loss functions with $p = 2$. These loss functions illustrate the generic forms of most practical interest—a continuous function, a hybrid function (discrete in one element of $\boldsymbol{\theta}$; continuous in the other element), and a purely discrete function. In the continuous case, it is often assumed that L is a "smooth" (perhaps several times differentiable) function of $\boldsymbol{\theta}$. Continuous problems arise frequently in applications such as model fitting (parameter estimation), adaptive control, neural network training, signal processing, and experimental design. In the discrete case, $\boldsymbol{\theta}$ can take on only a countable (finite or infinite) number of values (e.g., $\Theta = \{0, 1, 2, \ldots\}$ for a scalar θ). A typical discrete problem is: For a fixed

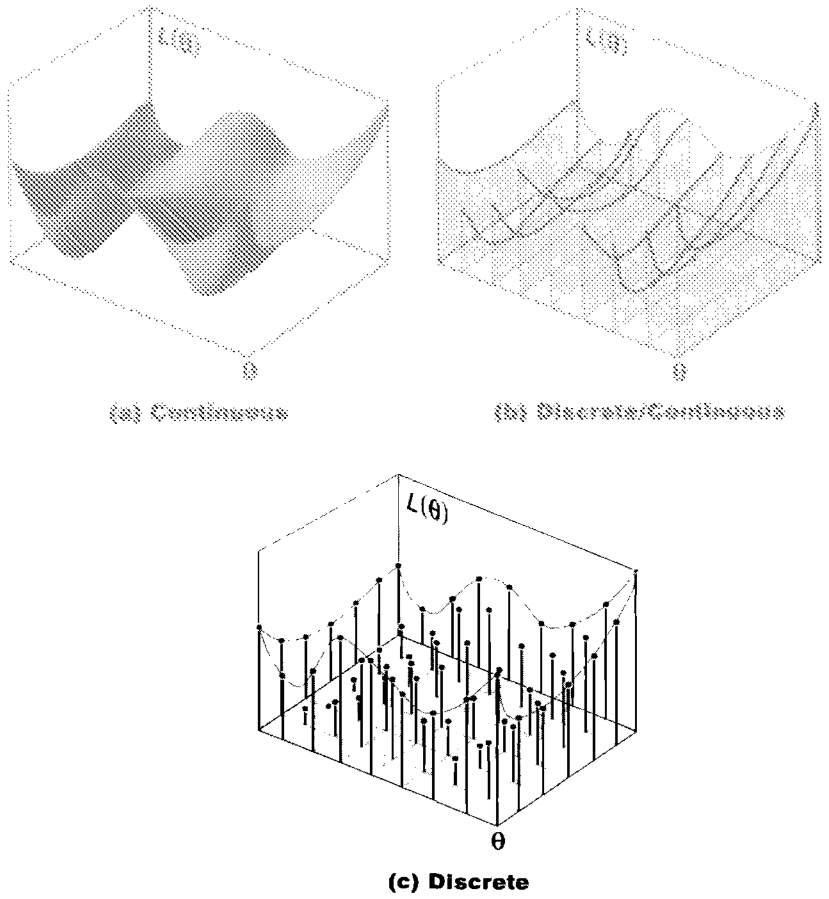

Figure 1.1. Three common types of loss functions for $\boldsymbol{\theta} \in \mathbb{R}^2$. Case (a) is continuous; case (b) is hybrid discrete/continuous (discrete in one element of $\boldsymbol{\theta}$; continuous in the other element); and case (c) is purely discrete in both elements of $\boldsymbol{\theta}$.

number (p) of categories of items, how many of each category are needed to optimize performance? Here, each element of $\boldsymbol{\theta}$ represents one category of items. Discrete optimization—sometimes called *combinatorial optimization*—is a large subject unto itself (resource allocation, network routing, policy planning, etc.).

Together with continuous search and optimization, discrete optimization appears as part of Chapters 2 and 7–10 on random search, stochastic approximation, simulated annealing, and evolutionary computation. Moreover, Chapter 12 and Section 14.5 are devoted exclusively to discrete problems via a discussion of statistical selection methods for comparing a finite number of options.

One of the major distinctions in optimization is between global and local optimization. All other factors being equal, one would always want a globally optimal solution to the optimization problem (i.e., at least one $\boldsymbol{\theta}^*$ in the set of values Θ^*, with each $\boldsymbol{\theta}^* \in \Theta^*$ providing a lower value of L than any $\boldsymbol{\theta} \notin \Theta^*$). In practice, however, a global solution may not be available and one must be satisfied with obtaining a *local* solution. A local solution is better than any in its vicinity (i.e., has a lower value of L), but it will not necessarily correspond to a best (global) solution $\boldsymbol{\theta}^*$ in Θ^*. For example, L may be shaped such that there is a clearly defined minimum point over a broad region of the domain Θ, while there is a very narrow spike at a distant point. If the trough of this spike is lower than any point in the broad region, the local optimal solution $\boldsymbol{\theta}_{\text{local}}$ is better than any nearby $\boldsymbol{\theta}$, but it is not be the best possible $\boldsymbol{\theta}$.

For a case where the problem dimension $p = 1$ (i.e., a scalar θ), Figure 1.2 depicts two problems with distinct local and global minima. Figure 1.2(a) shows a relatively benign setting where, despite the presence of the dip containing the local minimum, many algorithms will be able to find θ^* due to the broad indications of its presence in the nearby values of L. Figure 1.2(b), on the other hand, is an extreme form of the narrow spike idea. Here θ^* is one isolated point. This example illustrates that, *in general*, one cannot be guaranteed of ever

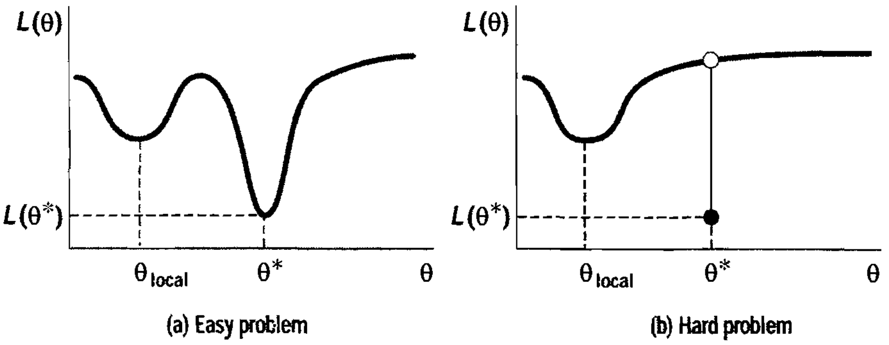

Figure 1.2. Examples of easy and hard problems for global optimization. In (a), a global algorithm will easily avoid θ_{local} and find θ^*; in (b), a global algorithm will only find θ_{local} since it is effectively impossible to find θ^*

obtaining a global solution. Without prior knowledge about the possible existence of such a point, no typical algorithm would be able to find this solution because there is nothing near θ^* to indicate the presence of a minimum (i.e., no "dip" in L).

Because of the inherent limitations of the vast majority of optimization algorithms, it is usually only possible to ensure that an algorithm will approach a local minimum with a finite amount of resources being put into the optimization process. However, since the local minimum may still yield a significantly improved solution (relative to no formal optimization process at all), the local minimum may be a fully acceptable solution for the resources available (human time, money, computer time, etc.) to be spent on the optimization. Much of this book will focus on algorithms that are only guaranteed to yield a local optimum. However, we will also consider some algorithms (random search, stochastic approximation, simulated annealing, genetic algorithms, etc.) that are sometimes able to find global solutions from among multiple local solutions.

1.1.3 Meaning of "Stochastic" in Stochastic Search and Optimization

The focus in this book is *stochastic* search and optimization. The problems and algorithms considered here apply where:

Property A	There is random noise in the measurements of $L(\boldsymbol{\theta})$ or $\boldsymbol{g}(\boldsymbol{\theta})$
	—and/or —
Property B	There is a random choice made in the search direction as the algorithm iterates toward a solution.

The above two properties contrast with classical deterministic search and optimization, where it is assumed that one has perfect information about the loss function (and derivatives, if relevant) and that this information is used to determine the search direction in a deterministic manner at every step of the algorithm. In many practical problems, such information is not available, indicating that deterministic algorithms are inappropriate.

Let $\hat{\boldsymbol{\theta}}_k$ be the generic notation for the estimate for $\boldsymbol{\theta}$ at the kth iteration of whatever algorithm is being considered, $k = 0, 1, 2, \ldots$. Throughout this book, the *specific* mathematical form of $\hat{\boldsymbol{\theta}}_k$ will change as the algorithm being considered changes. With the exception of the deterministic optimization algorithms considered briefly in Section 1.4, $\hat{\boldsymbol{\theta}}_k$ will always be a random vector since it is derived from input under stochastic Properties A and/or B above.

The following notation will be used throughout the book to represent noisy measurements of L and $\boldsymbol{g}$ at a specific $\boldsymbol{\theta}$:

$$y(\boldsymbol{\theta}) \equiv L(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta}), \tag{1.2a}$$

$$\boldsymbol{Y}(\boldsymbol{\theta}) \equiv \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{e}(\boldsymbol{\theta}), \tag{1.2b}$$

where ε and $\boldsymbol{e}$ represent the noise terms. Note that the noise terms show dependence on $\boldsymbol{\theta}$. This dependence is relevant for many applications. It indicates that the common statistical assumption of independent, identically distributed (i.i.d.) noise does not necessarily apply since $\boldsymbol{\theta}$ will be changing as the search process proceeds.

It will sometimes be convenient to use notation slightly different from that in (1.2a, b). Reflecting the fact that the measurements of L and/or $\boldsymbol{g}$ will frequently be at a current estimate $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_k$, and that the noise-generating process may change with k (perhaps as a real-time system evolves in concert with the algorithm evolution in k), let

$$y_k \equiv y_k(\hat{\boldsymbol{\theta}}_k) = L(\hat{\boldsymbol{\theta}}_k) + \varepsilon_k(\hat{\boldsymbol{\theta}}_k), \tag{1.3a}$$

$$\boldsymbol{Y}_k \equiv \boldsymbol{Y}_k(\hat{\boldsymbol{\theta}}_k) = \boldsymbol{g}(\hat{\boldsymbol{\theta}}_k) + \boldsymbol{e}_k(\hat{\boldsymbol{\theta}}_k). \tag{1.3b}$$

Depending on whether it is important to emphasize the dependence on $\boldsymbol{\theta}$ or the dependence on k, the notation $y(\boldsymbol{\theta})$ and y_k will be used interchangeably in the text as a measurement of L; likewise for $\boldsymbol{Y}(\boldsymbol{\theta})$ and $\boldsymbol{Y}_k$ as a measurement of $\boldsymbol{g}$. Slight variations on (1.3a, b) are also seen in applications. For instance, if the noise is not dependent on $\boldsymbol{\theta}$ (e.g., is statistically independent), then, in the case of the loss measurement, $y_k = y_k(\hat{\boldsymbol{\theta}}_k) = L(\hat{\boldsymbol{\theta}}_k) + \varepsilon_k$, where ε_k is an independent noise process.

For stochastic Property A above, the noise is relative to the measurements of L (and/or $\boldsymbol{g}$). This is different from some other uses of the term *noise* in the technical literature. In particular, in many estimation problems, one may have noisy data. This does not, however, necessarily mean that the values of L and/or $\boldsymbol{g}$ used in the optimization for forming the estimate will be noisy. The fundamental distinction comes in how the data are treated in the optimization process. In many estimation problems, a full set of data is collected and an L or $\boldsymbol{g}$ is chosen that is *conditioned on this set of data.* This conditioning removes the randomness from the problem as far as Property A is concerned. For example, in classical (batch) least-squares or maximum likelihood estimation methods, one finds the best parameter estimates *conditioned* on an existing set of noisy data. The estimation criterion (the sum of squares or the likelihood function) therefore becomes deterministic. Such estimation is often associated with so-called "off-line" methods of estimation. This deterministic setting for least-squares estimation is not to be confused with the (stochastic) *recursive* least-squares methods of Chapter 3.

In addition to such recursive methods, there are many problems where the noise associated with Property A *is* relevant. Many examples are sprinkled throughout this book. Noise fundamentally alters the search and optimization process because the algorithm is getting misleading information throughout the search process. The following example illustrates the dangers of a naïve treatment of noise in an optimization context.

Example 1.4—Effect of noisy data on solution to optimization problem. Consider the following loss function with a scalar θ: $L(\theta) = e^{-0.1\theta}\sin(2\theta)$. If the domain for optimization is $\Theta = [0, 7]$, the (unique) minimum occurs at $\theta^* = 3\pi/4 \approx 2.36$, as shown in Figure 1.3. Suppose that the analyst carrying out the optimization is not able to calculate $L(\theta)$, obtaining instead only *noisy* measurements $y(\theta) = L(\theta) + \varepsilon$, where the noises ε are i.i.d. with distribution $N(0, 0.5^2)$ (a normal distribution with mean zero and variance 0.5^2). The analyst uses the $y(\theta)$ measurements in conjunction with an algorithm to attempt to find θ^*.

Suppose that the algorithm here is the simple method of collecting one measurement at each increment of 0.1 over the interval defined by Θ (including the endpoints 0 and 7). The estimate of the minimum is the value of θ on this sampling grid [0, 0.1, 0.2,..., 6.9, 7] corresponding to the lowest *measured* loss. Using the MATLAB normal (pseudo) random number generator **randn**, Figure 1.3 shows the set of 71 measurements obtained for the optimization process. These noisy measurements are superimposed on the true loss function. Based on these measurements, the analyst would falsely conclude that the minimum is at $\theta = 5.9$. As shown, this false minimum is far from the actual θ^*. ❑

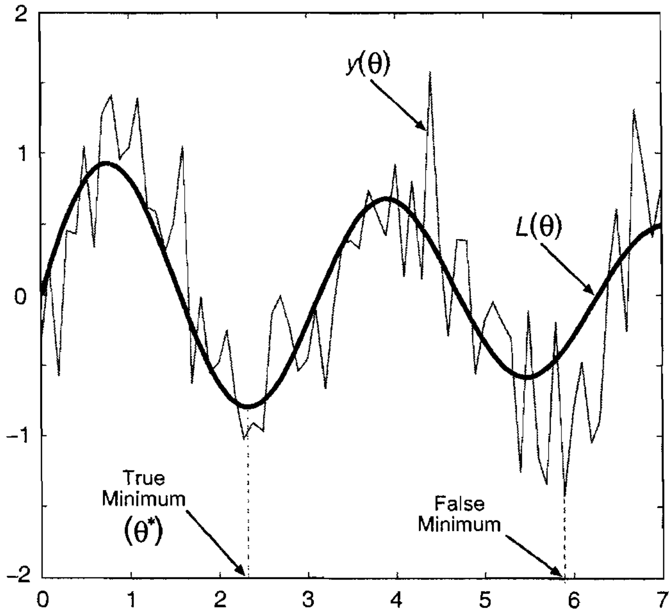

Figure 1.3. Simple loss function $L(\theta)$ with indicated minimum θ^*. Note how noise causes the algorithm to be deceived into sensing that the minimum is at the indicated false minimum.

Noise arises in almost any case where physical system measurements or computer simulations are used to approximate a steady-state criterion. Some specific areas of relevance include:

(i) Real-time estimation and control problems where data are collected "on the fly" as a system is operating. Property A (noisy loss measurements) is relevant because the data arrive in concert with the estimation of $\boldsymbol{\theta}$, with each increment of data being used to *approximate* some long-run (average) criterion such as a mean-squared error (MSE). Each such real-time approximation may be considered a noisy "measurement" of the long-run criterion (Chapters 3–7).

(ii) Problems where estimates of a criterion are formed by computer-based Monte Carlo sampling according to a statistical distribution (e.g., certain quantile estimation problems or Markov chain sampling schemes—see Chapters 4 and 16).

(iii) Problems where large-scale simulations are run as estimates of actual system behavior (Chapters 14 and 15).

(iv) Problems where physical data are processed sequentially, with each sequential data point being used to estimate some overall (average) criterion, such as the MSE. Unlike item (i), the data are not necessarily processed in real time. The sequential processing may be done to reduce the computational burden (versus the batch processing of all data) or to expose the unique impact of each data point.

Examples 1.5 and 1.6 provide conceptual descriptions of two distinct problems involving noise in the loss function measurements. Example 1.5 illustrates setting (i) above, while Example 1.6 depicts setting (iii).

Example 1.5—Noisy loss measurements in a tracking problem. As one specific example of a setting where Property A is relevant, consider the popular MSE criterion mentioned above. An analyst's goal might be to optimize the performance of some system (e.g., the tracking error in a robot control problem) while it is operating. The MSE criterion is

$$L(\boldsymbol{\theta}) = E\left(\left\|\text{actual output} - \text{desired output}\right\|^2\right),$$

where $E(\cdot)$ denotes the expectation (Appendix C) and the actual output depends on the design parameter vector $\boldsymbol{\theta}$ and other factors, including randomness. Only rarely will one be able to compute the MSE (or its derivatives!), as required in deterministic algorithms. In general, computation of the MSE requires complete knowledge of: (i) the probability distribution of all the randomness in the system, (ii) the mechanism by which the randomness manifests itself in the output, and (iii) the ability to carry out the multivariate numerical integration associated with the expected value in the MSE calculation. Rather, one might be able to get a *specific observation* of the squared error $\|\cdot\|^2$, but this differs from the *mean-squared* error. For instance, an engineer may have some spatial and temporal target values for a robot arm to meet in an operation; in lieu of computing the (hopeless) MSE, the engineer measures the deviation from these target values in a set of experiments.

More generally, one can write the observation as a noisy measurement of the mean-squared error,

$$y(\boldsymbol{\theta}) = \|\cdot\|^2 = E\left(\|\cdot\|^2\right) + \varepsilon(\boldsymbol{\theta}),$$

using the notation of (1.2a). The noise $\varepsilon(\boldsymbol{\theta})$ in this case is simply the arithmetic difference between the observed squared error and mean-squared error. Note that the noise in the measurements of L and/or $\boldsymbol{g}$ will often *not* be statistically independent of other randomness in the problem. In particular, the noise often depends on the current estimate for $\boldsymbol{\theta}$, which sometimes complicates the theoretical and practical analysis of an algorithm. ❑

Example 1.6—Optimization via Monte Carlo simulations. Suppose that an analyst is trying to optimize some complex system. As usual, let $\boldsymbol{\theta}$ represent the vector of terms being optimized and $L(\boldsymbol{\theta})$ be the loss function representing some type of "average" performance for the system. Computer simulation models are widely used to analyze and improve systems for which simple analytical analysis is inadequate. If the simulation includes randomness—a.k.a. the simulation is a *Monte Carlo* process—the simulation will, in general, produce results that change randomly with each run. Suppose that the analyst is going to use the computer simulation as a proxy for the true system in the process of optimizing $\boldsymbol{\theta}$. The analyst picks a value for $\boldsymbol{\theta}$, runs the simulation one or more times, evaluates the performance, and then updates $\boldsymbol{\theta}$. The process is then repeated at the new value of $\boldsymbol{\theta}$. The final value of $\boldsymbol{\theta}$ is used in the *real* system.

The evaluation of performance at a given $\boldsymbol{\theta}$, however, is not straightforward because of the randomness in the simulation (which should be acting like the randomness in the real system if the simulation is properly designed). While the goal is to optimize the *average* measure $L(\boldsymbol{\theta})$, the information available at any $\boldsymbol{\theta}$ is only a particular result from one or a small number of simulation runs. The analyst can, in principle, run the simulation many times at a given $\boldsymbol{\theta}$ to effectively average out the randomness. Such averaging, however, is usually impractical, especially if many $\boldsymbol{\theta}$ values must be examined and the simulation is expensive to run. So, for the optimization process, the information available to the analyst is in the general form associated with Property A above:

$$\text{simulation output at } \boldsymbol{\theta} = L(\boldsymbol{\theta}) + \text{noise at } \boldsymbol{\theta}.$$

Hence, an algorithm that formally accommodates noisy loss measurements in attempting to find an optimum $\boldsymbol{\theta}^*$ should be used. Chapters 14 and 15 delve into many issues of optimization that are specific to Monte Carlo simulations. Many of the generic principles in the chapters preceding Chapters 14 and 15 also apply to simulations. ❑

Relative to the second defining property of a stochastic algorithm, Property B above, it is sometimes beneficial to deliberately introduce randomness into the search process as a means of speeding convergence and making the algorithm less sensitive to modeling errors. This injected (Monte Carlo) randomness is usually created via computer-based pseudorandom number generators of the type discussed in Appendix D. Although the introduction of randomness may seem at first thought counterproductive, it is well known to have beneficial effects in some settings. One of the roles of injected randomness in stochastic search is to allow for "spontaneous" movements to unexplored areas of the search space that may contain an unexpectedly good $\boldsymbol{\theta}$ value. The randomness may provide the necessary kick. This is especially relevant in seeking out a global optimum when the search is stalled near a local solution.

Related to this application is the use of Monte Carlo randomness in the important class of algorithms that emulate evolutionary principles of optimization (Chapters 9 and 10); randomness is a central part of both physical and simulated evolution through the introduction of mutations and through the choice of parents. These mutations may sometimes have a beneficial effect by allowing unexpected solutions to be evaluated.

Injected randomness may also be used for the creation of simple random quantities that act like their deterministic counterparts, but which are much easier to obtain and more efficient to compute. An example of this is the simultaneous perturbation approximation of a gradient vector in Chapter 7. This gradient approximation can be used in place of the true gradient in certain optimization schemes, yielding similar general performance.

Yet another area where injected randomness is useful is in numerical integration. Often, Monte Carlo methods are implementable when analytical methods are impractical or impossible. Monte Carlo methods are usually more efficient in high-dimensional problems than deterministic quadrature approaches provided that one is willing to tolerate a small probability of achieving a poor estimate. Chapter 16, on Markov chain Monte Carlo, discusses one very popular general approach for numerical integration via such injected randomness.

1.2 SOME PRINCIPLES OF STOCHASTIC SEARCH AND OPTIMIZATION

We discussed above the problem formulation and some of the basic issues that surround the field of stochastic search and optimization. The two subsections in this section summarize some fundamental underpinnings and limits. The aim here is to provide the reader with some principles for interpreting the results in the remainder of this book. Some of these principles will seem self-evident and some will become apparent as the reader absorbs the results in the book. The first subsection lists some important facts that are relevant to understanding the results of this book and the second subsection is a summary of the immutable limits implied by the "no free lunch" theorems for search and optimization.

1.2.1 Some Key Points

Below is a list of facts that the reader should keep in mind while reading this book or implementing any of the algorithms described here.

Relative efficiency via function evaluations. One of the key issues in comparing algorithms is "efficiency" in solving problems of interest. In essence, this is some representation of the cost of finding an acceptable solution. There are many ways of measuring efficiency, including computer run time, number of algorithm iterations, and number of L and/or g function evaluations. This book will emphasize the latter as generally the most objective measure. Computer times tend to be machine and software dependent and hence do not readily transfer from one setting (or person) to another. Iterations can also be misleading, as one algorithm may have much more demanding per-iteration requirements than another; hence an algorithm producing a solution in a lower number of iterations does not generally do so at lower cost when compared to another algorithm taking more iterations but with lower per-iteration demands. In addition, the algorithm using fewer iterations may have required more "off-line" information in the form of research and analysis to provide information not used or required in the other algorithm (e.g., loss function gradient information in the first algorithm but not in the second).

Comparing algorithms based on number of L and/or g evaluations puts things on a common basis for contrast provided that the algorithms being compared use the same type of information (e.g., the algorithms use only loss evaluations L). The prime motivation is that in practice the L and/or g measurements are usually the dominant cost in the optimization process; the other calculations in the algorithms are often relatively unimportant (and becoming more unimportant as computing power grows). This philosophy is consistent with most complex stochastic optimization problems, where each loss function or gradient measurement may represent a large-scale simulation or a physical experiment. Although we use relatively simple loss functions in the numerical comparisons of this book, these are merely proxies for the more complex functions encountered in practice.

Implications of noisy function measurements. There are fundamental limits of search and optimization with noisy information about the L or g functions (fundamental Property A above). Foremost, perhaps, is that the statistical error of the information fed into the algorithm—and the resulting error of the output of the algorithm—can only be reduced by incurring a significant cost in number of function evaluations. (Ideally, of course, one would like to use perfect information in the algorithm as a way of enhancing the speed of the search.) For the simple case of independent noise, the error decreases at the rate $1/\sqrt{N}$, where N represents the number of L or $\boldsymbol{g}$ measurements fed into the algorithm. This is a classical result in statistics, indicating that a 100-fold increase in function evaluations reduces the error by a factor of 10. More generally, it takes an increase of $2K$ orders of magnitude in the number of measurements to reduce the error by K orders of magnitude. In fact, this $2K$ versus K barrier is frequently optimistic in practical problems, as the noise is often dependent in a way that increases the amount of averaging required to effectively reduce the statistical error.

Curse of dimensionality. A further limit for multivariate ($p > 1$) searches is that the volume of the search region generally grows geometrically with dimension. This implies that a "naïve" search in a high-dimensional problem will generally be hopeless. The famous control theorist and mathematician Richard Bellman (1920–1984) coined the piquant phrase "curse of dimensionality" to capture this phenomenon. Ho (1997), for example, gives an illustration in a discrete problem where $p = 10$ (which is relatively small by "industrial" standards): if each of the 10 elements of $\boldsymbol{\theta}$ can take on 10 values, there are 10^{10} possible outcomes. If we randomly sample 10,000 values of L uniformly in the domain Θ, the probability of finding one of the best 500 values for $\boldsymbol{\theta}$ (a much easier problem than finding a unique optimum $\boldsymbol{\theta}^*$) is a minuscule 0.0005. With noisy measurements of L, the problem is even worse because it is not known which $\boldsymbol{\theta}$ value corresponds to the lowest loss value from among the sampled $\boldsymbol{\theta}$ points. The lesson here is that, in practice, one will generally have to exploit some problem knowledge to have any hope of carrying out search and optimization in most real-world problems. Exercise 1.6 explores this point further.

Ordinal versus cardinal rankings. An area of stochastic optimization has developed over the last 10–15 years based on the idea of ordinal optimization (e.g., Ho et al., 1992). The basic philosophy here is to seek an answer that is "good enough" rather than to seek the optimal answer. The basis of the approach is ordinal rankings (i.e., is $A > B$?) rather than cardinal rankings (i.e., what is the precise value of $A - B$?). It is customarily much easier to determine relative rankings than it is to determine exact differences. The former usually requires only a minimal amount of averaging, while the latter needs a much more demanding averaging process. Some of the philosophy of ordinal optimization is illustrated in the discussion related to the curse of dimensionality above. In fact, relaxing the goals from finding the optimal value can yield orders of magnitude increases in efficiency in large-dimensional problems with noisy function evaluations.

Constraints. Constraints on the allowable values of $\boldsymbol{\theta}$ are a fact of life. All practical problems involve some restrictions on the relevant variables, although in some applications it may be possible to effectively ignore the constraints. Constraints can be encountered in many different ways, as motivated by the specific application. Note that the constraint set Θ does not necessarily correspond to the set of allowable values for $\boldsymbol{\theta}$ *in the search* since some problems allow for the "trial" values of the search to be outside the set of allowable final estimates. Constraints are usually handled in practice on an ad hoc basis, especially tuned to the problem at hand. There are few general, practical methods that apply broadly in stochastic search and optimization. Michalewicz and Fogel (2000, Chap. 9), for example, discuss some of the practical methods by which constraints are handled in evolutionary computation. Similar methods apply in other stochastic algorithms. Let us summarize the principal types of constraints seen in search problems:

> ***Hard vs. soft constraints.*** In hard constraints, no value of $\boldsymbol{\theta}$ can ever be taken outside of the allowable set. With soft constraints, values of $\boldsymbol{\theta}$ outside the constrained set are allowed during the search process, but it is required that the final estimate for $\boldsymbol{\theta}$ lie inside the constraint set. The former typically arises when the search is performed using an actual physical system, while the latter often arises in simulation-based search (so in the latter, sampling for $\boldsymbol{\theta}$ in the search algorithm might be done over some set Θ' such that $\Theta \subset \Theta'$, which can be of potential interest due to the additional information about the nature of L that this "extended sampling" can sometimes provide).

> ***Explicit vs. implicit constraints.*** For explicit constraints, limits on the values of $\boldsymbol{\theta}$ can be specified directly (e.g., one constraint might be $\|\boldsymbol{\theta}\| \leq c$ for some vector norm $\|\cdot\|$ and constant c; another constraint might be that each component of $\boldsymbol{\theta}$ lies in the interval $[-c, c]$). For implicit constraints, the limits are specified in terms of some complicated function of $\boldsymbol{\theta}$, possibly in terms of allowable values for L. Given the complicated function, there is no method for explicitly representing the constraints on $\boldsymbol{\theta}$. For example, in a control system, if $\boldsymbol{\theta}$ represents some parameters inside the controller software and the real system output must stay within some bounds, then there are implied constraints on what values of $\boldsymbol{\theta}$ are allowed in order to maintain the system inside its allowable region (and if the $\boldsymbol{\theta}$ estimation is done in real time on the physical control system, this may also represent one of the hard constraint cases cited above).

Stopping criteria. In search and optimization, there is traditionally a concern with developing a good *stopping criterion* (i.e., a means of indicating when the algorithm is close enough to the solution that it can be stopped). Unfortunately, the quest for an automatic means of stopping an algorithm with a guaranteed level of accuracy seems doomed to failure in general stochastic search problems. The fundamental reason for this pessimistic message is that in nontrivial problems, there will always be a significant region within Θ that will remain unexplored in any finite number of iterations. Without prior knowledge, there is always the possibility that $\boldsymbol{\theta}^*$ could lie in this unexplored region. This applies even when the functions involved are relatively benign; see Solis and Wets (1981) for mention of this in the context of twice-differentiable convex L (convexity is discussed in Appendix A). Difficulties are compounded when the function measurements include noise. For this reason, most of the comparisons in this book will be done under the constraint that the competing algorithms use the same number of function evaluations. This provides an objective comparison for the same "cost" of search.

Time-varying problems. In many practical settings, the environment changes over time. Hence, the "best" solution to a problem now may not be the best (or even a good) solution to the corresponding problem in the future. Such a formulation is obvious in dynamic problems such as building control systems, where the optimum may change continuously. Although less obvious, the time-varying problem also arises in settings that may appear at first glance to be static. For example, an optimum financial plan for a business or family depends on the external environment, which, of course, changes over time. In some search and optimization problems, the algorithm will be explicitly designed to adapt to a changing environment and "automatically" provide a new estimate at the optimal value (e.g., a control system). In other cases, one needs to restart the process and find a new solution. In either sense, the problem solving may never stop!

Limits of numerical comparisons by Monte Carlo. A common means of comparing algorithms is to run Monte Carlo studies on some test cases (e.g., with particular forms for L). This can be a sound scientific method of gaining insight and can be a useful supplement to theory, much of which is based on asymptotic (infinite sample) analysis. In fact, it is especially popular in certain branches of optimization to create "test suites" of problems, where various algorithms face-off in a numerical showdown.

A danger arises, however, in making *broad* claims about the performance of an algorithm based on the results of numerical studies. Performance can vary tremendously under even small changes in the form of the functions involved or the coefficient settings within the algorithms themselves. One must be careful about drawing conclusions beyond those directly supported by the specific numerical studies performed. This, in fact, is a manifestation of the no free lunch theorems discussed below—outstanding performance on some types of functions is consistent with poor performance on certain other types of functions.

For purposes of drawing objective conclusions about the relative performance of algorithms, it is preferable to use *both* theory and numerical studies. The theory—despite its limitations (e.g., asymptotically based derivations)—provides a basis for drawing broad formal conclusions and pointing to possible limitations in the range of appropriate problems, while the numerical studies give the user a direct feel for algorithm behavior in specific problems in finite samples. Unfortunately, in some popular algorithms, the theory is incomplete or even virtually nonexistent. This has led to some overly broad claims in the literature about algorithm performance on the basis of limited numerical studies. The use of test suites, as mentioned above, offers some enlightenment, but the reader is cautioned to interpret all numerical studies—including those in this book!—with the proverbial grain of salt.

Special note for numerical comparisons with noisy measurements. With the foregoing caveats in mind, Monte Carlo simulations are frequently used to evaluate different algorithms based on noisy function measurements $y = y(\boldsymbol{\theta})$ or $\boldsymbol{Y} = \boldsymbol{Y}(\boldsymbol{\theta})$. In such evaluations, the simulation designer usually knows "truth"; that is, L or $\boldsymbol{g}$ may usually be calculated exactly. In contrast, the algorithms being tested use only the noisy measurements y or $\boldsymbol{Y}$ in their simulated processing to emulate the operations of a real application. In comparing algorithms in such a Monte Carlo fashion, the *final evaluation* should be done based on the exact L or $\boldsymbol{g}$ values, not the noisy values y or $\boldsymbol{Y}$. So, for example, in comparing several optimization algorithms on a specific problem, one may select as "best" the algorithm that produces the lowest value of L at the terminal estimate for $\boldsymbol{\theta}$. It wastes information that is available to the simulation designer to evaluate the algorithms with only the noisy measurements that are used by the algorithms.

Uniqueness vs. nonuniqueness of $\boldsymbol{\theta}^*$. A number of results appear in the literature pertaining to analysis and convergence when there may be multiple global solutions $\boldsymbol{\theta}^*$. In practice, however, this is often not a concern. Many (but certainly not all) real systems have one (unique) globally "best" operating point ($\boldsymbol{\theta}^*$) in the domain Θ, although, of course, there could be many *locally* optimal solutions. So, in order to avoid excessively cumbersome discussion of algorithms and supporting implementation issues and theory, we will often refer to "the" solution $\boldsymbol{\theta}^*$ (versus "a" solution $\boldsymbol{\theta}^*$). In cases where there are multiple $\boldsymbol{\theta}^*$, the results presented here (such as convergence) will typically apply to any one of the multiple solutions (guaranteeing, e.g., convergence into the *set* Θ^*). In practice, an analyst may be quite satisfied to reach a solution at or close to *any* one $\boldsymbol{\theta}^* \in \Theta^*$, so the potential issue of nonuniqueness of $\boldsymbol{\theta}^*$ is sometimes of limited practical concern.

Notational convention. A slight conflict arises when trying to avoid excessively cumbersome notation to indicate a particular element of a vector and to indicate a particular iteration of an estimate for the vector. We will attempt to live with that conflict by establishing the following notational convention. Let $\hat{\boldsymbol{\theta}}_k$ represent the estimate for $\boldsymbol{\theta}$ at the kth iteration of an algorithm and $\hat{\theta}_{ki}$ represent the ith element of the estimate at the kth iteration. On the other hand, the generic representation of the elements of $\boldsymbol{\theta}$ is according to $\boldsymbol{\theta} = [t_1, t_2, \ldots, t_p]^T$. We represent the ith value of the vector $\boldsymbol{\theta}$ by $\boldsymbol{\theta}_i$. This notation is useful in time-varying $\boldsymbol{\theta}$ cases to indicate the value of $\boldsymbol{\theta}$ at time i, and useful when Θ is composed of a discrete number of elements to denote the ith possible value of $\boldsymbol{\theta}$. In cases where a vector quantity is not associated with an iteration process, a simple subscript will denote the indicated element of the vector. For example, x_i denotes the ith element of the vector $\boldsymbol{x}$. This should not create any significant difficulties, as the meaning of the subscript should always be clear from the context.

1.2.2 Limits of Performance: No Free Lunch Theorems

There is a fundamental tradeoff between algorithm efficiency and algorithm robustness (reliability and stability in a broad range of problems). In essence, algorithms that are designed to be very efficient on one type of problem tend to be "brittle" in the sense that they do not reliably transfer to problems of a different type. Hence, there can never be a universally best search algorithm just as there is rarely (never?) a universally best solution to any general problem of society. For example, an airframe design for a plane requiring maneuverability and speed in a battle setting is a very poor design for commercial aviation. One must consider the characteristics of the problem together with the goals of the search and the resources available (computing power, human analysis time, etc.) in choosing an approach.

This lack of a universal best algorithm is a manifestation of the *no free lunch* (NFL) theorems in Wolpert and Macready (1997), which say, in essence, that an algorithm that is effective on one class of problems is *guaranteed* to be ineffective on another class. One may get a feel for the NFL theorems by considering the famous "needle in a haystack" problem. In the absence of clues about the needle's location, it is clear that no search approach can, on average, beat blind random search. The NFL theorems are discussed in more detail in Chapter 10 on evolutionary computation, reflecting the area of optimization from which they arose. The discussion here is intended to provide a flavor of the implications.

While the NFL theorems are established for discrete optimization with a finite (but arbitrarily large) number of options, their applicability includes most practical continuous problems because virtually all optimization is carried out on 32- or 64-bit digital computers. The theorems apply to the cases of both noise-free and noisy loss measurements. Because of the restriction to a finite number of discrete options, there is a corresponding finite number of possible mappings of the input space (values of $\boldsymbol{\theta}$) to the output space (values of L or y). This number may be huge (Exercise 1.7). In particular, in the noise-free case, if there are N_θ possible values for $\boldsymbol{\theta}$ and N_L possible values for the loss, then by direct enumeration there are $(N_L)^{N_\theta}$ possible mappings of $\boldsymbol{\theta}$ to possible loss values. The NFL theorems indicate that:

> When averaging over all $(N_L)^{N_\theta}$ possible mappings from Θ to the output space (loss values), all algorithms work the same (i.e., none can work better than a blind random search).

The *mappings* in the statement above are sometimes called *problems*. Each mapping $L(\boldsymbol{\theta})$ corresponds to one problem that an analyst may be solving.

The averaging mentioned at the beginning of the NFL statement above is a simple arithmetic mean over the $(N_L)^{N_\theta}$ mappings. The following small-scale example with noise-free loss measurements is intended to provide a flavor of the theorems.

Example 1.7—NFL implications in a small problem. Suppose that $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3\}$ and that there are two possible outcomes for the noise-free loss measurements, $\{L_1, L_2\}$. Hence, $(N_L)^{N_\theta} = 2^3 = 8$. Table 1.1 summarizes the eight possible mappings.

Note that all rows in Table 1.1 have the same number of L_1 and L_2 values. Hence, the mean loss value across all eight possible problems is the same regardless of which $\boldsymbol{\theta}_i$ is chosen. Because the algorithm chooses the $\boldsymbol{\theta}_i$, all algorithms produce the same mean loss value when averaging across all possible problems. As a specific instance of the implication of the NFL theorems, suppose that $L_1 < L_2$. Then, an algorithm that puts priority on picking $\boldsymbol{\theta}_1$ will work well on problems 1, 2, 3, and 7, but poorly on problems 4, 5, 6, and 8. The *average* performance is the same as an algorithm that puts priority on picking $\boldsymbol{\theta}_2$ or $\boldsymbol{\theta}_3$. ❑

Because the NFL theorems seem to paint a discouraging picture that "all algorithms work the same," one might wonder about the value of studying various algorithms, as we do in this book. A book devoted to blind search alone would be a short book indeed! A key qualifier in the NFL theorems is the "averaging over all possible mappings" statement. If a problem has some known structure—and all conceivable practical problems do—*and* the algorithm uses that structure, it is certainly possible that one algorithm will work better than another on the given problem. If the needle is more likely to be located near the surface of the haystack, one can save a lot of time that would otherwise be devoted to burrowing into the haystack.

Table 1.1. The eight possible mappings (problems), representing all possible combinations of inputs ($\boldsymbol{\theta}$) and outputs (loss values). Values shown are subscripts of loss outcome (i.e., 1 represents L_1; 2 represents L_2). For example, under mapping 4, $L(\boldsymbol{\theta}_1) = L_2$ and $L(\boldsymbol{\theta}_2) = L_1$.

Mapping / $\boldsymbol{\theta}$	1	2	3	4	5	6	7	8
$\boldsymbol{\theta}_1$	1	1	1	2	2	2	1	2
$\boldsymbol{\theta}_2$	1	1	2	1	1	2	2	2
$\boldsymbol{\theta}_3$	1	2	2	1	2	1	1	2

Given the above intuition associated with the value of problem structure, an informal mathematical representation of the NFL theorems is

Size of domain of applicability
$\times$ Efficiency on domain of applicability = Universal constant.

That is, an algorithm cannot have *both* wide applicability and uniformly high efficiency.

In the spirit of the NFL theorems, much of the discussion in this book focuses on algorithms that are appropriate in restricted domains. For example, Chapter 3 considers methods that are especially appropriate in linear models; these methods are relatively ineffective in nonlinear models. Cumulatively, the algorithms discussed in this book span a broad range of problems, but any one algorithm will be applicable in a relatively restricted range of problems.

1.3 GRADIENTS, HESSIANS, AND THEIR CONNECTION TO OPTIMIZATION OF "SMOOTH" FUNCTIONS

1.3.1 Definition of Gradient and Hessian in the Context of Loss Functions

As mentioned, many of the algorithms in this book apply to problems where L is continuous, or, more strongly, once or several times differentiable in $\boldsymbol{\theta}$ (some of the same algorithms—e.g., simulated annealing, evolutionary computation, etc.—may also apply to discrete problems). Hence, certain results in multivariate calculus are useful. In particular, some of the methods discussed in this book rely—at least indirectly—on the gradient and (possibly) Hessian matrix of the loss function (possibly embedded in noise).

When considering problems with such "smooth" $L(\boldsymbol{\theta})$, it is of interest to consider some calculus-based aspects of analysis. Recall that $\boldsymbol{\theta} = [t_1, t_2, \ldots, t_p]^T$. The gradient of the loss function is defined as the p-dimensional column vector

$$g(\boldsymbol{\theta}) \equiv \frac{\partial L}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \partial L / \partial t_1 \\ \partial L / \partial t_2 \\ \vdots \\ \partial L / \partial t_p \end{bmatrix}.$$

The Hessian is the matrix of second partial derivatives

$$H(\boldsymbol{\theta}) \equiv \frac{\partial^2 L}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T} = \begin{bmatrix} \partial^2 L/\partial t_1^2 & \partial^2 L/\partial t_1\partial t_2 & \cdots & \partial^2 L/\partial t_1\partial t_p \\ \partial^2 L/\partial t_2\partial t_1 & \partial^2 L/\partial t_2^2 & & \vdots \\ \vdots & & \ddots & \\ \partial^2 L/\partial t_p\partial t_1 & \cdots & \cdots & \partial^2 L/\partial t_p^2 \end{bmatrix},$$

where $\partial^2 L/\partial t_i \partial t_j = \partial^2 L/\partial t_j \partial t_i$ under the assumption of continuity of the terms in the Hessian matrix (so then $\boldsymbol{H}$ is a symmetric matrix). Note that the term "$\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T$" appearing in the denominator of the derivative expression is merely a notational convenience for suggesting a matrix outcome in the differentiation process; it is not to be interpreted as referring to a product of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^T$ or a derivative with respect to such a product.

The optimization setting being emphasized in this book motivates the gradient and Hessian terminology above. In problems of root-finding, the terminology is slightly different. The terms *function* and *Jacobian matrix* typically replace the terms *gradient* and *Hessian matrix*, respectively (so the Jacobian matrix is a collection of first derivatives of the vector-valued function). Specifically, in root-finding, $\boldsymbol{g}(\boldsymbol{\theta})$ is the *function* for which a zero is to be found (a root to $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$) and $\boldsymbol{H}(\boldsymbol{\theta})$ is the *Jacobian matrix*. The author can attest from experience that it is important to recognize this semantic difference when communicating optimization results in fields traditionally based in root-finding!

The Taylor theorems from multivariate analysis are important in the construction and formal analysis of both deterministic and stochastic algorithms. Under appropriate conditions, these theorems allow one to convert a possibly messy nonlinear function into a relatively benign low-order polynomial approximation. Appendix A summarizes some of the important results associated with Taylor theorems.

1.3.2 First- and Second-Order Conditions for Optimization

Algorithms for solving unconstrained local minimization problems (i.e., (1.1) with $\Theta = \mathbb{R}^p$) with smooth loss functions are frequently based upon the first- and second-order derivative conditions. These derivative conditions also apply in some constrained problems and in some of the stochastic search techniques of interest here.

The first-order condition states that at $\boldsymbol{\theta}^*$ it is necessary that

$$\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0} \tag{1.4}$$

when L is a continuously differentiable function. The proof of this result follows from simple Taylor series arguments showing that if (1.4) is not true, then one can move $\boldsymbol{\theta}$ in some direction that reduces the value of $L(\boldsymbol{\theta})$ (use Taylor's theorem A.1 in Appendix A). In particular, if $\boldsymbol{d} \in \mathbb{R}^p$ denotes a vector such that $\boldsymbol{g}(\boldsymbol{\theta})^T\boldsymbol{d} < 0$, then moving $\boldsymbol{\theta}$ to a new value $\boldsymbol{\theta} + \lambda\boldsymbol{d}$ for any $\lambda \in (0, u)$ and some $u > 0$ will guarantee a lower value of L. Directions $\boldsymbol{d}$ such that $\boldsymbol{g}(\boldsymbol{\theta})^T\boldsymbol{d} < 0$ are called *descent directions* for L at $\boldsymbol{\theta}$. However, since one can repeat the same line of reasoning to show that $\boldsymbol{g}(\boldsymbol{\theta}^*) = \boldsymbol{0}$ is a necessary condition for $\boldsymbol{\theta}^*$ to be a local *maximizer* of L, we need some information to distinguish the solution $\boldsymbol{\theta}^*$ as being a local minimum versus a local maximum (or saddlepoint).

The Hessian matrix $\boldsymbol{H} = \boldsymbol{H}(\boldsymbol{\theta})$ plays an important role in making the distinction between local minima and maxima for loss functions that are twice continuously differentiable. In particular, if $\boldsymbol{\theta}'$ is a root of the equation $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$, then $\boldsymbol{H}(\boldsymbol{\theta}')$ being positive definite (see Appendix A) implies that $\boldsymbol{\theta}'$ is a local minimizer of L. Further, it is necessary that $\boldsymbol{H}(\boldsymbol{\theta}')$ be at least positive semidefinite for $\boldsymbol{\theta}'$ to be a minimizing point $\boldsymbol{\theta}^*$. If $\boldsymbol{H}(\boldsymbol{\theta}')$ is positive semidefinite but not positive definite, then one can sometimes evaluate higher-order derivatives to determine if $\boldsymbol{\theta}'$ is a local mimimizer (e.g., Bazaraa et al., 1993, pp. 134–138). Conversely, if $-\boldsymbol{H}(\boldsymbol{\theta}')$ is positive definite (i.e., $\boldsymbol{H}(\boldsymbol{\theta}')$ is negative definite), then $\boldsymbol{\theta}'$ is a local maximizer of L. If $\boldsymbol{H}(\boldsymbol{\theta}')$ is indefinite (i.e., has both positive and negative eigenvalues), then $\boldsymbol{\theta}'$ is a saddlepoint, implying that $\boldsymbol{\theta}'$ is a local minimizer in some components of $\boldsymbol{\theta}$ and a local maximizer in other components. The connection of the Hessian to convexity is discussed in Appendix A.

Exercise 1.10 shows an example where a root of $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$ with a positive semidefinite Hessian is a unique (global) minimum. Two examples of functions where roots of $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$ with positive semidefinite Hessians correspond to saddlepoints are given in Bazaraa et al. (1993, pp. 135–138).

1.4 DETERMINISTIC SEARCH AND OPTIMIZATION: STEEPEST DESCENT AND NEWTON–RAPHSON SEARCH

There are many deterministic optimization algorithms for the problem in (1.1) and it is not the purpose here to survey these algorithms. However, two algorithms in particular are worth discussing since many stochastic search algorithms for "smooth" (differentiable) $L(\boldsymbol{\theta})$ or $\boldsymbol{g}(\boldsymbol{\theta})$ can be motivated by connections to these deterministic algorithms. These are the methods of steepest descent and Newton–Raphson, considered in Subsections 1.4.1 and 1.4.2.

1.4.1 Steepest Descent Method

The method of steepest descent is one of the oldest formal optimization techniques. Nevertheless, it remains one of the more popular deterministic approaches. For example, it corresponds to the widely used backpropagation algorithm for neural networks when one is working with a fixed set of input–output data—see Chapter 5. Steepest descent is based on the simple principle that from a given value $\boldsymbol{\theta}$ the best direction to go is the one that produces the largest local change in the loss function (the steepest descent). The gradient vector at the given $\boldsymbol{\theta}$ defines this direction. Hence the algorithm is

$$\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - a_k \boldsymbol{g}(\hat{\boldsymbol{\theta}}_k),\ k = 0, 1, 2, \ldots, \tag{1.5}$$

where k is the iteration count, $\hat{\boldsymbol{\theta}}_0$ is the initial "guess" at $\boldsymbol{\theta}^*$, and $a_k > 0$ is the *step size*, which may be specified a priori (often as a constant $a_k = a$) or picked on an iteration-to-iteration basis as a solution to $\min_{a \geq 0}\{L(\hat{\boldsymbol{\theta}}_k - a\boldsymbol{g}(\hat{\boldsymbol{\theta}}_k))\}$ (this secondary optimization problem is called a *line search*). So, (1.5) states that the new estimate of the best value of $\boldsymbol{\theta}$ is equal to the previous value minus a term proportional to the gradient at the current value.

Dennis and Schnabel (1989, p. 38) discuss implementations of steepest descent when the secondary (line search) optimization problem of solving for a_k at each iteration is eliminated and a simplified (possibly predetermined) form of a_k employed. In fact, in the stochastic approximation and neural network methods discussed in Chapters 4 and 5, the stochastic analogues of steepest descent typically have a_k as a predetermined decaying sequence. The scale factors a_k play a critical role in the algorithm, often determining whether the algorithm converges or diverges. Aside from being called step sizes, these factors are sometimes referred to as *gains* or *learning coefficients*, depending on the field of application. Conditions guaranteeing that the steepest descent iterate converges to $\boldsymbol{\theta}^*$ as $k \to \infty$ are presented in many places (e.g., Bazaraa et al., 1993, pp. 300–308).

Figure 1.4 illustrates why the simple steepest descent recursion in (1.5) can lead to reductions in the L value. Given a multivariate functional relationship of $L(\boldsymbol{\theta})$ versus $\boldsymbol{\theta}$, consider the plane showing L as a function of the ith component of $\boldsymbol{\theta}$ with the other components of $\boldsymbol{\theta}$ fixed. For a relatively simple convex ("bowl-shaped") function, Figure 1.4 shows that when the ith component of the current estimate is on either side of the current minimum for that component, the updated estimate will move towards the minimum in a direction *opposite* the sign of the corresponding element of the gradient vector. In particular, when the current value of $\hat{\theta}_{ki}$ is left of the minimum, (1.5) implies that the new estimate will move to the right according to the search direction given by the negative of the negative sign of the gradient component (= positive direction). Conversely, when the current value is to the right of the optimum, the new estimate will move left since the negative of the positive sign of the ith gradient component leads to a negative direction.

The role of a_k is to regulate how large a step the algorithm takes. Clearly, steps that are too large or too small may prevent the algorithm from ever converging to $\boldsymbol{\theta}^*$, even if the steps are in the correct directions. Steps that are neither too large nor too small will move the algorithm toward $\boldsymbol{\theta}^*$ in a controlled

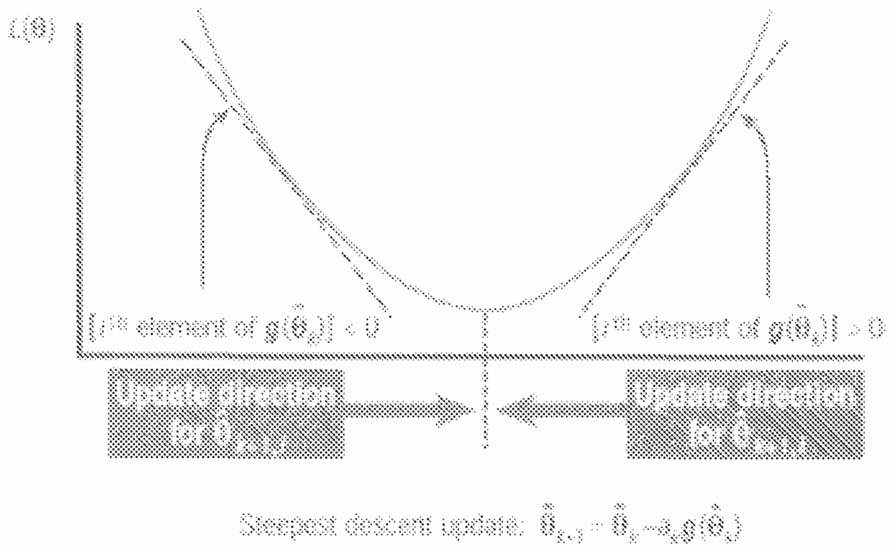
$L(\theta)$
[i^{th} element of $g(\hat{\theta}_k)$] < 0
[i^{th} element of $g(\hat{\theta}_k)$] > 0
Update direction for $\hat{\theta}_{k+1}$
Update direction for $\hat{\theta}_{k+1}$
Steepest descent update: $\hat{\theta}_{k+1} = \hat{\theta}_k - a_k g(\hat{\theta}_k)$

Figure 1.4. Motivation for steepest descent: Update directions for *i*th component of $\boldsymbol{\theta}$ in iterating from $k \to k + 1$.

manner. This involves interplay with the other components of the $\boldsymbol{\theta}$ vector since, in general, the *i*th element of $\boldsymbol{\theta}^*$ is not the same as the *current* minimum in the plane of L versus the *i*th element of $\boldsymbol{\theta}$. These two minima will correspond when all other components of $\boldsymbol{\theta}$ are at their $\boldsymbol{\theta}^*$ values.

Despite steepest descent's virtues of formal convergence and widespread use, it is a relatively inefficient algorithm in contrast to some other approaches that use gradient information. Further, it is sensitive to transformations and scaling. That is, a linear transformation of the underlying $\boldsymbol{\theta}$ vector can dramatically alter the algorithm performance even though the two vectors contain equivalent information about the underlying physical process if the transformation is invertible. Related to this sensitivity to transformations, the algorithm will have difficulty in optimizing some components if the magnitudes of the components differ dramatically (see the discussion in Subsection 1.4.2 and Exercise 1.11 for contrasting behavior with the Newton–Raphson method).

The steepest descent algorithm, however, provides a useful starting point for deriving more powerful algorithms. For example, other algorithms may have a generic steepest descent-type form with some differing details or may use steepest descent as part of the per-iteration calculations, as in trust region and conjugate gradient methods (Dennis and Schnabel, 1989, Sects. 4.2 and 5.2). The steepest descent algorithm also has the virtues of relative simplicity, of guaranteeing at least a local descent in the loss function, and (as we will see with the Robbins-Monro stochastic approximation algorithm of Chapters 4 and 5) of having a ready extension to stochastic problems where the gradient is known only to within some random noise. Example 1.8 illustrates the steepest descent algorithm on a simple $p = 2$ problem, comparing the results of a deterministic search and a search with noisy measurements of the gradient $\boldsymbol{g}(\boldsymbol{\theta})$. Calculations for this example (as for others in the book) are carried out in MATLAB.

Example 1.8—Steepest descent with and without noise. Let $\boldsymbol{\theta} = [t_1, t_2]^T$ and consider the simple loss function and associated (unconstrained) domain:

$$L(\boldsymbol{\theta}) = t_1^4 + t_1^2 + t_1 t_2 + t_2^2 \text{ with } \Theta = \mathbb{R}^2.$$

By solving for $\boldsymbol{\theta}$ in the equation $\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{0}$ and checking for the positive definiteness of the Hessian $\boldsymbol{H}(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$, it is easily seen that there is a unique solution $\boldsymbol{\theta}^* = [0, 0]^T$. For our purpose, however, let us assume that this solution is not known and that a steepest descent algorithm is to be used. In standard deterministic (noise-free) steepest descent, the gradient

$$\boldsymbol{g}(\boldsymbol{\theta}) = \begin{bmatrix} 4t_1^3 + 2t_1 + t_2 \\ t_1 + 2t_2 \end{bmatrix}$$

is assumed directly available.

In addition to the above-mentioned noise-free implementation, we also consider a case where only noisy measurements $\boldsymbol{Y}(\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{e}$ are available. Here, the noise $\boldsymbol{e}$ is i.i.d. $N(\boldsymbol{0}, \boldsymbol{I}_2)$, where $\boldsymbol{I}_2$ denotes the 2×2 identity matrix (so the variance of each element of $\boldsymbol{e}$ is 1). With the exception of this noisy measurement $\boldsymbol{Y}(\boldsymbol{\theta})$ being substituted for the true gradient, we use the basic steepest descent recursion in (1.5) with a constant gain $a_k = a$. Hence, the modified steepest descent algorithm (actually, a simple example of a Robbins–Monro root-finding stochastic approximation algorithm, discussed in Chapters 4 and 5) is $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - a\boldsymbol{Y}(\hat{\boldsymbol{\theta}}_k)$.

From an initial condition $\hat{\boldsymbol{\theta}}_0 = [1, 1]^T$, the noisy and noise-free versions of steepest descent are run for 50 iterations. Based on some limited numerical experimentation with sample runs to $k = 50$ in the noisy case, it is found that $a = 0.05$ tends to produce a mean final loss value (i.e., a mean of $L(\hat{\boldsymbol{\theta}}_{50})$ across several independent replications) that is approximately the lowest feasible under the noise. We use the same a in the noise-free case. Figures 1.5 and 1.6 show the relative performance for the search with and without noise. Figure 1.5 contrasts the search path for the two elements of $\boldsymbol{\theta}$. The two curves in Figure 1.6 show the mean loss over 50 independent replications (all starting at $\hat{\boldsymbol{\theta}}_0$) and the deterministic result. A single noisy run, of course, is much more erratic than the mean of the noisy runs. As motivated in Subsection 1.2.1, the plot in Figure 1.6 showing the mean of 50 runs is based on the exact (noise-free) L values even though the algorithm uses only noisy values $\boldsymbol{Y}$ in its iterations.

The noisy run in Figure 1.5 is slightly better than the typical run in the sense that the terminal loss is slightly below the mean terminal loss value over

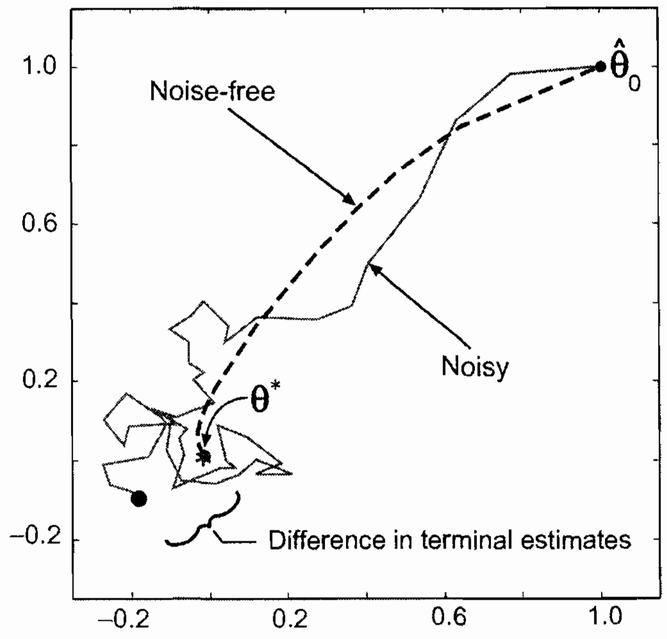

Figure 1.5. Comparison of steepest descent runs with noisy and noise-free gradient input. The indicated search paths are in the plane of t_2 (vertical axis) versus t_1 (horizontal axis). The indicated difference in terminal estimates shows the difference between the final noisy and noise-free estimates for $\boldsymbol{\theta}$.

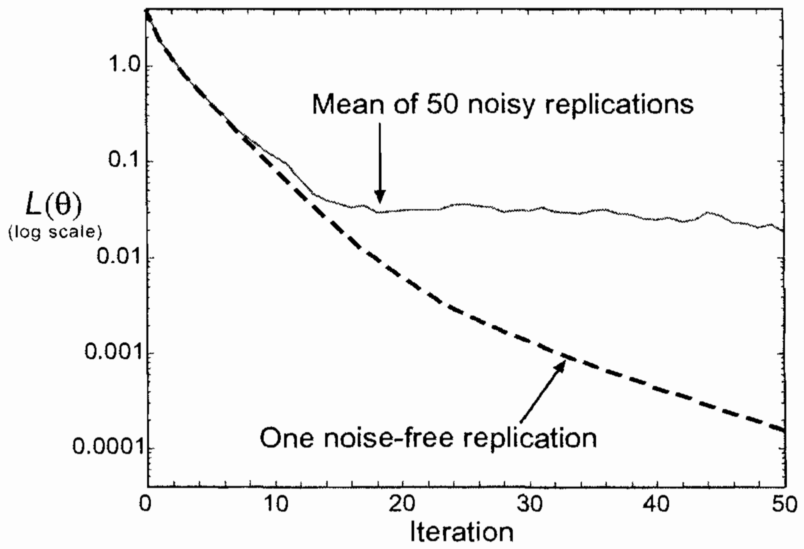

Figure 1.6. Comparison of steepest descent runs with noisy and noise-free gradient input: relative value of L as the iteration proceeds.

50 runs that is shown in Figure 1.6. Nevertheless, the terminal error in the noisy $\hat{\boldsymbol{\theta}}_{50}$ (relative to $\boldsymbol{\theta}^*$) represents about 16 percent of the total distance between $\hat{\boldsymbol{\theta}}_0$ and $\boldsymbol{\theta}^*$; in contrast, the terminal error for the noise-free case is approximately 1.0 percent. Figure 1.5 shows the relatively large difference between the terminal noisy and noise-free estimates of $\boldsymbol{\theta}$. Because the gain a is optimized for the noisy case, the deterministic steepest descent algorithm is capable of performing even better with a modification to the gain.

Figure 1.5 illustrates the common phenomenon—especially in stochastic problems—of the terminal iterate not being the best of the iterates, either in terms of distance between $\hat{\boldsymbol{\theta}}_k$ and the unknown $\boldsymbol{\theta}^*$ or in terms of the value of the loss function. This is not surprising in light of the "misleading" information being provided to the algorithm as a consequence of the noise. Figure 1.6 shows that the mean loss value in the noisy case decreases, although much more slowly than in the noise-free case. ❑

1.4.2 Newton–Raphson Method and Deterministic Convergence Rates

The Newton–Raphson algorithm (sometimes simply called *Newton's method*) builds on the basic recursion in (1.5) by introducing a scaling via the inverse Hessian matrix. This has the advantage of potentially speeding the convergence significantly, but has the possible disadvantage of making the algorithm more unstable. In particular, if it is assumed that L is at least twice continuously differentiable and that $\boldsymbol{H}$ is invertible at all $\boldsymbol{\theta}$ encountered in the search, the algorithm has the form

$$\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - \boldsymbol{H}(\hat{\boldsymbol{\theta}}_k)^{-1}\boldsymbol{g}(\hat{\boldsymbol{\theta}}_k),\ k = 0, 1, 2, \ldots \tag{1.6}$$

It is simple to informally derive this algorithm by expanding $\boldsymbol{g}(\boldsymbol{\theta})$ to first order around an old $\boldsymbol{\theta}$ value to get its approximate value at a new $\boldsymbol{\theta}$:

$$\boldsymbol{g}(\boldsymbol{\theta}_{\text{new}}) \approx \boldsymbol{g}(\boldsymbol{\theta}_{\text{old}}) + \boldsymbol{H}(\boldsymbol{\theta}_{\text{old}})(\boldsymbol{\theta}_{\text{new}} - \boldsymbol{\theta}_{\text{old}}) \tag{1.7}$$

Setting $\boldsymbol{g}(\boldsymbol{\theta}_{\text{new}}) = \mathbf{0}$ and taking "$\approx$" as an exact equality yields the Newton–Raphson recursion, where the subscripts k and k+1 represent old and new, respectively, and provided that $\boldsymbol{H}(\boldsymbol{\theta}_{\text{old}}) = \boldsymbol{H}(\hat{\boldsymbol{\theta}}_k)$ is invertible at each k. Note that if L is a quadratic function, the expansion in (1.7) is exact and the Newton–Raphson algorithm will converge in one step from any starting point—no algorithm can converge faster! This is obviously not a common situation in practice since few practical loss functions will be quadratic. However, near the solution $\boldsymbol{\theta}^*$, L will be nearly quadratic for any twice continuously differentiable function (by Taylor's theorem in Appendix A) and thus one can expect the Newton–Raphson algorithm to be fast once it is near $\boldsymbol{\theta}^*$.

In nonlinear deterministic optimization, "there is a strong suspicion that if any iterative method for any problem in any field is exceptionally effective, then it is a Newton [–Raphson] method in some appropriate context" (Dennis and Schnabel, 1989). For general nonlinear (nonquadratic) loss functions, $\boldsymbol{H}$ may not be positive definite at $\boldsymbol{\theta}$ away from $\boldsymbol{\theta}^*$, which can cause stalling or divergence. For this reason, many modifications of the basic Newton–Raphson method have been introduced with the goal of preserving most of the very fast local convergence while stabilizing the algorithm when it is not close to $\boldsymbol{\theta}^*$ (e.g., Dennis and Schnabel, 1989; Jang et al., 1997, Sect. 6.4). One simple way of helping control possible poor behavior of Newton–Raphson is to introduce a coefficient a_k in front of the $\boldsymbol{H}^{-1}\boldsymbol{g}$ term in (1.6), analogous to the step size in steepest descent. If $a_k < 1$ for all or most k, this may help stabilize the algorithm at the possible expense of slowing its convergence.

In contrast to steepest descent, Newton–Raphson has the desirable property of being transform invariant and unaffected by large scaling differences in the $\boldsymbol{\theta}$ elements (see Exercise 1.13). In particular, if $\boldsymbol{\theta}$ is transformed to $\boldsymbol{\theta}' = \boldsymbol{A}\boldsymbol{\theta}$ for some invertible matrix $\boldsymbol{A}$, then optimization based on $\boldsymbol{\theta}'$ yields the same value of $\boldsymbol{\theta}$ after transforming back from $\boldsymbol{\theta}'$ to $\boldsymbol{\theta}$. For steepest descent, this is *not* true since the underlying Hessian in the $\boldsymbol{\theta}'$-based search becomes $(\boldsymbol{A}^{-1})^T\boldsymbol{H}(\boldsymbol{\theta})\boldsymbol{A}^{-1}$, affecting the convergence properties via a fundamental change in the loss function curvature. Related to this scaling issue is the issue of when a steepest descent and Newton–Raphson algorithm are equivalent. For a quadratic loss function, if $\boldsymbol{H} = \boldsymbol{H}(\boldsymbol{\theta}) = c\boldsymbol{I}_p$ for all $\boldsymbol{\theta}$ and some $c > 0$, c not a function of $\boldsymbol{\theta}$, then the algorithms (1.5) and (1.6) are identical when $a_k = 1/c$ in (1.5) ($\boldsymbol{I}_p$ denotes the $p \times p$ identity matrix).

Figure 1.7 illustrates the relative behavior of the steepest descent (first-order) and Newton–Raphson (second-order) algorithms on three different loss functions with $p = 2$. The level curves in the three cases indicate points in $\boldsymbol{\theta}$-space having the same level of L value. A steepest descent algorithm always moves in a direction perpendicular to the level curve at the current point. Cases (a) and (b) are quadratic surfaces. In (a), the circular level curves correspond to $\boldsymbol{H} = c\boldsymbol{I}_p$ as above. Thus, the two algorithms move in the same direction from the indicated starting point. In (b), the Hessian is nondiagonal, implying a difference in the search directions of the algorithms. While steepest descent moves perpendicular to the level curve, Newton–Raphson in (b) moves directly toward $\boldsymbol{\theta}^*$ because the function is quadratic (the one-step solution mentioned above).

Case (c) in Figure 1.7 illustrates the danger of moving in the second-order direction with a nonquadratic loss function. Here, Newton–Raphson moves in a direction *worse* than steepest descent. In fact, in some such cases, Newton–Raphson can diverge from the solution wildly. This figure is an illustration of the principle mentioned in Subsection 1.2.2: algorithms well suited to particular settings (e.g., quadratic loss functions) tend to be "brittle," performing poorly in other settings. (Newton–Raphson may, however, sometimes perform very well with nonquadratic loss functions; one must consider the algorithm on a case-by-case basis.)

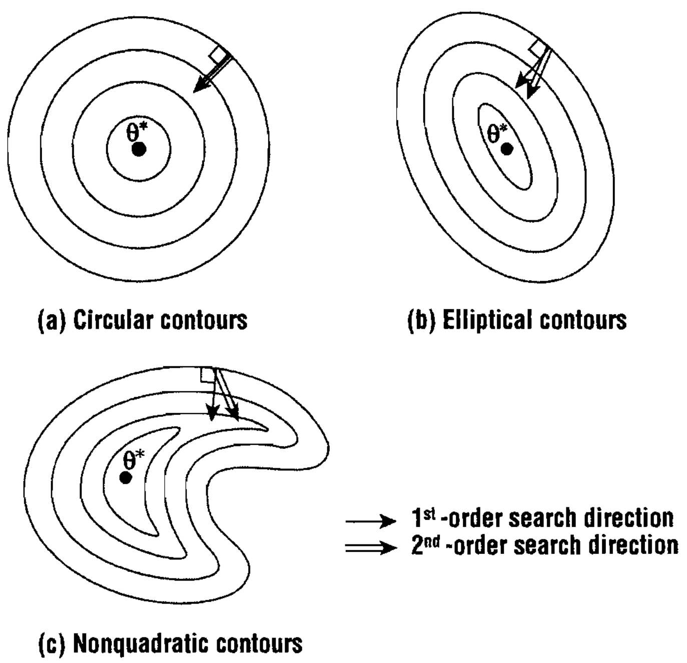
θ^*
θ^*
θ^*
(a) Circular contours
(b) Elliptical contours
(c) Nonquadratic contours
1st-order search direction
2nd-order search direction

Figure 1.7. Contrast of first-order (steepest descent) and second-order (Newton–Raphson) search directions on $p = 2$ problems. In (a), both search directions are the same. In the elliptical (quadratic) surface (b), second-order direction points directly at θ^* while first-order direction is not toward θ^*. Example (c) shows that the second-order direction can be *worse* than the first-order direction in nonquadratic regions of search space (as when starting far from θ^*).

Under the conditions of $\boldsymbol{H}$ being positive definite in a neighborhood of θ^* and assuming that $\hat{\theta}_k$ stays in this neighborhood, then the deterministic rate of convergence for Newton–Raphson is quadratic in the sense that

$$\left\|\hat{\theta}_{k+1} - \theta^*\right\| = O\left(\left\|\hat{\theta}_k - \theta^*\right\|^2\right),$$

where $O(x)$ is the standard big-O order notation implying $O(x)/x \le c$ for some constant $c > 0$ as $x \to 0$. (In the *stochastic* algorithms of this book, different types of rate of convergence measures will be used to accommodate the additional inherent variability.) The above quadratic rate result is proved in many places (e.g., Dennis and Schnabel, 1989; Kelly, 1999a, pp. 13–17). For $\hat{\theta}_k$ sufficiently close to θ^*, this result implies very fast convergence: $\hat{\theta}_{k+1}$ has an error that is proportional to the *square* of the (assumed small) error in the previous iterate $\hat{\theta}_k$. This rate of convergence contrasts with the slower linear rate for the method of steepest descent:

$$\left\|\hat{\boldsymbol{\theta}}_{k+1}-\boldsymbol{\theta}^*\right\| = O(\left\|\hat{\boldsymbol{\theta}}_k-\boldsymbol{\theta}^*\right\|),$$

where the implied constant (c) in the order bound is less than unity (Kelly, 1999a, pp. 45–46). We emphasize that the above rates for both Newton–Raphson and steepest descent depend on the iterates being "sufficiently close" to $\boldsymbol{\theta}^*$.

Many algorithms aim to capture most of the enhanced speed of Newton–Raphson, but aim to do so with greater stability and without the need to explicitly calculate the Hessian matrix (which is difficult to obtain in many practical problems). These algorithms are designed to avoid the potential divergence illustrated in Figure 1.7(c). Included in this class of algorithms are the popular conjugate gradient or quasi-Newton methods (e.g., Dennis and Schnabel, 1989; Bazaraa et al., 1993, pp. 312–345; Rustagi, 1994, pp. 72–75; Kelly, 1999a, Chap. 4). Most of these algorithms have a deterministic convergence rate between linear and quadratic.

All of the above rates are critically dependent on the deterministic structure of the problem. If one has only gradient and Hessian information in the presence of random noise, these rates and/or implied constants no longer apply. As part of the stochastic approximation discussion in Chapters 4–7, we see analogous algorithms, where modifications are made to a basic first-order algorithm to capture the speedup due to second-order search directions while preserving stability.

1.5 CONCLUDING REMARKS

This chapter sets the stage for the remainder of the book. We discussed the meaning of *stochastic* in the context of search and optimization, noting that it pertains to random noise in measurements of the loss function or root-finding function and/or to randomness that is injected in a Monte Carlo fashion. Although much of this book focuses on standard algorithms for stochastic search and optimization, we also consider some closely related aspects of stochastic modeling. These include model selection and experimental design.

Real-world problem solving is often difficult. We discussed some of the reasons why this is so. These include: (i) the presence of noise in the function evaluations; (ii) the difficulties in distinguishing a globally optimal solution from locally optimal solutions; (iii) the "curse of dimensionality," which causes the size of the search space to grow exponentially with problem dimension; (iv) the difficulties associated with constraints and the need for problem-specific methods; and (v) the lack of stationarity in the solution as a result of the conditions of the problem changing over time. A good treatment of both technical and philosophical challenges in real-world problem solving is Michalewicz and Fogel (2000).

The "no free lunch" theorems are a fundamental barrier to exaggerated claims of the power and efficiency of any specific algorithm. These theorems indicate that if an algorithm is especially efficient on one set of problems, it is *guaranteed* to be inefficient on another set of problems. The way to cope with the negative implication of these theorems in practice is to restrict an algorithm to a particular class of problems *and* to have the algorithm exploit the structure in this class. Most of the algorithms in this book are based on this principle of exploiting available structure, although in many cases the available structure is an incomplete representation of the process.

One related general area not emphasized in this book is *stochastic programming* (e.g., Birge and Louveaux, 1997). Although not universally defined as such, stochastic programming tends to emphasize methods that are direct applications of deterministic linear and nonlinear programming techniques. Often, the random aspects of the problem are removed by replacing the relevant random quantities with corresponding (fixed) mean values. Although such an approach is powerful in the right context, it requires information that is often not available in practical applications. For example, in the framework of this chapter, many methods in stochastic programming require direct knowledge of the loss function $L(\boldsymbol{\theta})$, rather than using only the noisy measurements $y(\boldsymbol{\theta})$. One of the appealing features of stochastic programming is the ability to deal with nontrivial constraints in the same way as deterministic methods. This benefit accrues because of the transformation of the stochastic problem to an effectively deterministic problem.

Many of the methods in this book are designed to handle the challenges outlined above, having applicability to broad areas such as estimation, Monte Carlo simulation, and control. It will also be clear that there are many exciting areas—in both algorithm development and applications—that have yet to be fully explored. In applying methods from this book, it is important to keep in mind the fundamental goals of a scientific or other investigation: "Better a rough answer to the right question than an exact answer to the wrong one" (aphorism possibly due to Lord Kelvin). In a wide range of problems, the extensive set of tools here will include the right tools for the right question.

EXERCISES[2]

1.1 Suppose that the noise ε has a symmetric triangular probability density function over the interval $[-1, 1]$ (i.e., with x the dummy variable for the density function, the density is $1-|x|$ for $x \in [-1, 1]$ and 0 for $x \notin [-1, 1]$). What is var(ε) (the variance of ε)?

1.2 For a scalar θ, identify the maximum and minimum of the function $\theta/2+\sqrt{1-\theta^2}$ on the domain $[-1, 1]$.

[2] Appendices A–C also contain exercises relevant to this chapter. Answers to selected exercises for this and other chapters/appendices are at the back of the book and at the book's Web site.

1.3 Based on investments represented by $\boldsymbol{\theta}$, suppose that each component of some deterministic vector function $s(\boldsymbol{\theta})$ represents the amount of output in one segment of a firm's business and that each component of the random vector $\boldsymbol{\pi}$ represents the profit resulting from 1 unit of output in the corresponding segment (so $\dim(s(\boldsymbol{\theta})) = \dim(\boldsymbol{\pi})$ with all segments of the business represented in these vectors). Suppose that the total profit of the firm can be measured and that $\mathrm{cov}(\boldsymbol{\pi}) = \boldsymbol{\Sigma}$. Formulate the stochastic optimization problem of finding the investments to maximize mean profit subject to having the variance of the profit be no greater than some bound $C > 0$. In particular, write down the *general forms* for $L(\boldsymbol{\theta})$, $y(\boldsymbol{\theta})$, and Θ, and show that ε depends on $\boldsymbol{\theta}$. (In practice, a simulation or real data on the firm could be used to produce the $y(\boldsymbol{\theta})$ values needed in the optimization.)

1.4 Suppose that Θ contains only two options, $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$, and that $L(\boldsymbol{\theta}_1) = 0$ and $L(\boldsymbol{\theta}_2) = 1$. Suppose that the analyst's budget allows only one (noisy) measurement $y(\boldsymbol{\theta}_i)$ at each $\boldsymbol{\theta}_i$ and that the noise terms (ε) are independently $N(0, 1)$ distributed for both measurements. What is the probability of incorrectly selecting $\boldsymbol{\theta}_2$ as the optimal $\boldsymbol{\theta}$ when the choice is based on the $\boldsymbol{\theta}_i$ having the lowest $y(\boldsymbol{\theta}_i)$?[3]

1.5 Suppose that $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$ and that only noisy measurements of L are available, where the noise process ε is i.i.d. over all measurements (i.e., does not depend on $\boldsymbol{\theta}$). Suppose that the sample means of 20 measurements at each of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are 5.4 and 6.6, respectively, with corresponding sample variances 2.2 and 3.2 (using the standard definition in Appendix B). Based on the appropriate two-sample test (Appendix B), discuss whether such data provide sufficient evidence to claim that there is a significant difference in $L(\boldsymbol{\theta}_1)$ and $L(\boldsymbol{\theta}_2)$ and discuss at least one caveat associated with the conclusion in light of what is *not* stated in the assumptions.[4]

1.6 Suppose that each of the p components in $\boldsymbol{\theta}$ can take on one of N values and suppose that we sample uniformly and independently (with replacement) in Θ in search of an optimum.

(a) What is the general expression for the probability of finding at least one of the V best points in Θ when we generate K (noise-free) values of $L(\boldsymbol{\theta})$?

(b) With $p = 10$ and $N = 15$, what is the probability of finding at least one of the 10,000 best points in Θ when we generate 100,000 values of $L(\boldsymbol{\theta})$?

1.7 Consider a two-dimensional $\boldsymbol{\theta}$ where $\Theta = [1.0, 2.1) \times [1.0, 2.1)$ (i.e., a Cartesian product of intervals with endpoints 2.1 not in Θ) and $L(\boldsymbol{\theta}) \in \{0, 1, 2, 3, 4\}$. Suppose the computer accuracy is such that each component of $\boldsymbol{\theta}$ is evaluated at increments of 2.2×10^{-16} in Θ. (Using IEEE standards, this is the approximate accuracy of floating-point numbers on a 32-bit

[3] A normal distribution table is needed for this and several other exercises in the book. Such tables are available in almost any introductory statistics textbook or in various forms in MS EXCEL (e.g., the NORMDIST function).

[4] Analogous to Exercise 1.4, this exercise requires a t-distribution table, available in almost any introductory statistics textbook or in MS EXCEL (e.g., the TDIST function).

computer operating in double precision near the value 1.0.) What is the number of mappings in the average that forms the basis for the NFL theorems? Express the answer in the general form 10^x by solving for x.

1.8 Let $\boldsymbol{\theta} = [t_1, t_2, t_3]^T$ and

$$L(\boldsymbol{\theta}) = t_1^2 + 2t_2^2 + 3t_3^2 + 3t_1t_2 + 4t_1t_3 - 3t_2t_3.$$

Verify that $\boldsymbol{\theta} = \mathbf{0}$ is a solution to $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0}$. Determine whether this solution is a minimum, maximum, or saddlepoint.

1.9 Using Taylor's theorem A.2 in Appendix A, prove that L is a convex function on Θ if $\boldsymbol{H}$ is positive semidefinite at all points in Θ.

1.10 Let $\boldsymbol{\theta} = [t_1, t_2]^T$. Apply the steepest descent method to the function $L(\boldsymbol{\theta}) = (t_1 - 2)^4 + (t_1 - 2t_2)^2$ beginning with the initial guess $\hat{\boldsymbol{\theta}}_0 = [0, 3]^T$ and using $a_k = 0.062$, 0.24, 0.11, 0.31, 0.12, and 0.36 for $k = 0$, 1, 2, 3, 4, and 5, respectively (yielding $\hat{\boldsymbol{\theta}}_1$ to $\hat{\boldsymbol{\theta}}_6$). Report values for L and $\boldsymbol{\theta}$ after each of the six iterations. (For comparison purposes, note that $L(\hat{\boldsymbol{\theta}}_0) = 52.0$, $\boldsymbol{\theta}^* = [2, 1]^T$ and $L(\boldsymbol{\theta}^*) = 0$.)

1.11 Using the same function and initial guess as Exercise 1.10, apply the Newton–Raphson method in eqn. (1.6) and report the stated values at the indicated number of iterations.

1.12 For the function in Exercise 1.10, do the following:

(a) Run the standard deterministic steepest descent algorithm for 1000 iterations with $a_k = 0.0275$ for all k.

(b) Suppose that we only observe $\boldsymbol{Y}(\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{e}$, where the noise $\boldsymbol{e}$ is independently $N(\mathbf{0}, 0.1^2\boldsymbol{I}_2)$ distributed. Use the steepest descent algorithm with $\boldsymbol{Y}(\boldsymbol{\theta})$ replacing $\boldsymbol{g}(\boldsymbol{\theta})$ and $a_k = 0.0275$ (as in part (a)). Determine the sample mean of the final loss values from 100 realizations of 1000 iterations per realization, i.e., form a sample mean of the 100 values of $L(\hat{\boldsymbol{\theta}}_{1000})$. Compare the mean final loss function value here with the deterministic value from part (a). (This is an example of a root-finding stochastic approximation algorithm, as discussed in Chapters 4 and 5.)

1.13 Fill in the missing details in the arguments in Subsection 1.4.2 regarding the invariance of the Newton–Raphson method to transformations and different scaling.

1.14 Suppose that $\left\|\hat{\boldsymbol{\theta}}_k - \boldsymbol{\theta}^*\right\| = f(k)$ in the steepest descent or Newton–Raphson algorithm.

(a) Give an example $f(k)$ that satisfies the big-O result of Subsection 1.4.2 for the steepest descent algorithm.

(b) Give an analogous example $f(k)$ for the Newton–Raphson algorithm.