Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. James C. Spall

ISBN: 0-471-33052-3

CHAPTER 2

DIRECT METHODS FOR STOCHASTIC SEARCH

This chapter examines several direct search methods for optimization, "direct" in the sense that the algorithms use minimal information about the loss function. These direct methods have the virtues of being simple to implement and having broad applicability (e.g., allowing for either continuous- or discrete-valued θ). Further, it is sometimes possible to make rigorous statements about the convergence properties of the algorithms, which is not always possible in more complex procedures.

Prior to presenting the algorithms, we discuss in Section 2.1 some motivating issues in direct random search methods. The algorithms discussed in Sections 2.2 to 2.4 represent two distinct types of the most popular direct search methods. Sections 2.2 and 2.3 consider the first type—direct search methods involving random selections of θ within the search domain Θ. These are among the most popular and versatile search methods available. Section 2.2 considers noise-free loss measurements and Section 2.3 considers noisy loss measurements. Section 2.4 considers the second type of direct search method—a nonlinear algorithm inspired by the geometric notion of a simplex (and not to be confused with the well-known simplex method of linear programming). Although this technique is frequently presented as a deterministic search method, its origins are in stochastic search involving noisy loss measurements.

In support of this and other chapters, Appendix D discusses some issues in computer-based random number generation; such *pseudorandom* numbers play a critical role in the random search algorithms of this chapter as well as in some of the algorithms to be seen later.

2.1 INTRODUCTION

This chapter focuses on relatively simple *direct search* methods for solving the optimization problem of minimizing a loss function $L = L(\theta)$ subject to the parameter vector θ lying in some set Θ. The information required to implement these methods is essentially only input–output data where θ is the input and $L(\theta)$ (noise-free) or $y(\theta)$ (noisy) is the output. Further, the underlying assumptions

about L are relatively minimal. In particular, there are no requirements that the gradient of L be computable (or even that the gradient *exist*) or that L be unimodal (i.e., that L have only one local optimum, corresponding to the global optimum). This chapter focuses on two distinct types of popular direct search methods. The first is random search and the second is based on the geometric notion of a simplex.

Note that when the random search or nonlinear simplex methods are applied to problems with differentiable L, these techniques make no use of the gradient that exists. Avoiding the use of the gradient can often significantly ease implementation. Additional—and often more powerful—gradient-free methods will be discussed later in this book (principally, in Chapters 6–10, covering simultaneous perturbation stochastic approximation, simulated annealing, and evolutionary computation methods).

With the exception of *some* of the discussion on the nonlinear simplex method, the algorithms of this chapter have at least one of the two characteristics of stochastic search and optimization mentioned in Chapter 1—noisy input information (Property A in Section 1.1) or injected algorithm randomness (Property B). For purposes of algorithm *evaluation* by Monte Carlo, pseudorandom numbers generated via a computer algorithm are used to produce both the noise and the injected randomness. For purposes of *actual application* to a physical system producing its own randomness, pseudorandom numbers are used only for the injected randomness. All pseudorandom numbers in this (and other chapters) are generated via the default generators **`rand`** (uniform) and **`randn`** (normal) in MATLAB (see Appendix D for a brief discussion of these and other generators).

Unless noted otherwise, we restrict the random sampling of the algorithms here to be in the constraint domain Θ (i.e., there is no sampling over a larger set Θ', where $\Theta \subset \Theta'$, as mentioned in Subsection 1.2.1 as being of potential interest in some applications; see also Exercise 2.11). This appears to be the most common form of implementation.

Recall from Subsection 1.2.1 that there are never likely to be fully acceptable automated stopping criteria for stochastic search algorithms. For that reason, this book will generally emphasize algorithm comparisons and stopping based on "budgets" of function evaluations. Nonetheless, sometimes one may wish to augment function budgets or analyst "intuition" via some automated approach.

Two common stopping criteria are given below. Let η represent a small positive number, N be the number of iterations for which the algorithm should be "stable," and $\hat{\boldsymbol{\theta}}_k$ be the estimate for $\boldsymbol{\theta}$ at iteration k. An algorithm may be stopped at iteration $n \geq N$ when, for all $1 \leq j \leq N$, at least one of the two criteria below hold:

$$\left|L(\hat{\boldsymbol{\theta}}_n) - L(\hat{\boldsymbol{\theta}}_{n-j})\right| \leq \eta \quad \text{or} \quad \left\|\hat{\boldsymbol{\theta}}_n - \hat{\boldsymbol{\theta}}_{n-j}\right\| \leq \eta.$$

These criteria express the fact that the algorithm has "settled down" in some sense. The first criterion above is useful when noise-free loss measurements are available, while the second criterion is useful in noise-free *and* noisy cases. Although the criteria have some intuitive appeal, they provide no guarantee of the terminal iterate $\hat{\boldsymbol{\theta}}_n$ being close to an optimum $\boldsymbol{\theta}^*$. Some additional stopping criteria that are closely related to the above are given in Section 2.4.

The next three sections present some simple direct search algorithms, focusing on random search and the nonlinear simplex method (a simplex is the region contained within a collection of $p + 1$ points in $\mathbb{R}^p$, as defined more formally in Section 2.4). Although the simplex algorithm of Section 2.4 is not as simple as the random search methods of Sections 2.2 and 2.3, it is popular and often powerful. As an illustration of its popularity, the simplex method is the main MATLAB gradient-free optimization algorithm (`fminsearch`).

2.2 RANDOM SEARCH WITH NOISE-FREE LOSS MEASUREMENTS

2.2.1 Some Attributes of Direct Random Search

Random search methods for optimization are based on exploring the domain Θ in a random manner to find a point that minimizes $L = L(\boldsymbol{\theta})$. These are the simplest methods of stochastic optimization and can be quite effective in some problems. Their relative simplicity is an appealing feature to both practitioners and theoreticians. These direct random search methods have all or some of the following advantages relative to most other search methods. Some of these attributes were mentioned in the prescient paper of Karnopp (1963):

(i) **Ease of programming.** The methods described below are relatively easy to code in software and can thereby significantly reduce the *human* cost of an optimization process (not to be ignored in practice, but often ignored in published results comparing the efficiency of one algorithm against another!).

(ii) **Use of only L measurements.** The reliance on L measurements alone can significantly reduce the incentive to pick a loss function largely for analytical convenience—perhaps at a sacrifice to the true optimization goals—so that gradients or other ancillary information may be computed. Karnopp (1963) aptly calls this a reduction in "artful contrivance."

(iii) **Reasonable computational efficiency.** Although not generally the most computationally efficient algorithms in practical problems, the algorithms can often provide *reasonable* solutions fairly quickly, especially if the problem dimension p ($= \dim(\boldsymbol{\theta})$) is not too large. This is especially true in those direct search algorithms that make use of some local information in their search (e.g., random search algorithms B and C below). For example, as demonstrated in Anderssen and Bloomfield (1975), random

search may be more efficient than corresponding deterministic algorithms based on searches over multidimensional grids when p is large. The solution from a random search method can usually be augmented with some more powerful—but perhaps more complex—search algorithm if a more accurate solution is required. In fact, the random search algorithms may provide a means of finding "good" initial conditions for some of the more sophisticated algorithms to be presented later.

(iv) **Generality.** The algorithms can apply to virtually any function. The user simply needs to specify the nature of the sampling randomness to allow an adequate search in Θ. Thus, if $\boldsymbol{\theta}$ is continuous valued, the sampling distribution should be continuous (e.g., Gaussian or continuously uniform on Θ); likewise, a discrete-valued $\boldsymbol{\theta}$ calls for a discrete sampling distribution with nonzero probability of hitting the candidate points and a hybrid $\boldsymbol{\theta}$ calls for the appropriate mix of continuous and discrete sampling distributions.

(v) **Theoretical foundation.** Unlike some algorithms, supporting theory is often available to provide guarantees of performance and guidance on the expected accuracy of the solution. In fact, the theory may even be exact in finite samples, which is virtually unheard of in the analysis of other stochastic algorithms.

2.2.2 Three Algorithms for Random Search

Beginning with the most basic algorithm, this subsection describes three direct random search techniques. Because direct random search is a large subject unto itself, only a small selection of algorithms is being presented here in order to keep this subsection of manageable length (see, e.g., Solis and Wets, 1981; Zhigljavsky, 1991; and some references discussed below for additional algorithms). The three algorithms here are intended to convey the essential flavor of most available direct random search algorithms. The methods of this subsection assume perfect (noise-free) values of L.

In the noise-free case, it can be shown that many random search methods converge to an optimum $\boldsymbol{\theta}^*$ in one of the probabilistic senses discussed in Appendix C—almost surely (a.s.), in probability (pr.), or in mean square (m.s.)—as the number of L evaluations gets large (see below). Realistically, however, these convergence results may have limited utility in practice since the algorithms may take a prohibitively large number of function evaluations to reach a value close to $\boldsymbol{\theta}^*$, especially if p is large. This is illustrated below. Nevertheless, the formal convergence provides a guarantee not always available in other approaches.

The simplest random search method is one where we repeatedly sample over Θ such that the current sampling for $\boldsymbol{\theta}$ does not take into account the previous samples. This "blind search" approach does not adapt the current sampling strategy to information that has been garnered in the search up to the

present time. The approach can be implemented in batch (nonrecursive) form simply by laying down a number of points in Θ and taking the value of $\boldsymbol{\theta}$ yielding the lowest L value as our estimate of the optimum. The approach can also be implemented in recursive form as we illustrate below.

The simplest setting for conducting the random sampling of new (candidate) values of $\boldsymbol{\theta}$ is when Θ is a hypercube (a p-fold Cartesian product of intervals on the real line) and we are using uniformly generated values of $\boldsymbol{\theta}$. The uniform distribution is continuous or discrete for the elements of $\boldsymbol{\theta}$ depending on the definitions for these elements. *In fact, this particular (blind search) form of the algorithm is unique among all general stochastic search and optimization algorithms in this book: It is the only one without any adjustable algorithm coefficients that need to be "tuned" to the problem at hand.*

For a domain Θ that is not a hypercube or for other sampling distributions, one may use transformations, rejection methods, or Markov chain Monte Carlo to generate the sample $\boldsymbol{\theta}$ values. For example, if Θ is an irregular shape, one can generate a sample on a hypercube superset containing Θ and then reject the sample point if it lies outside of Θ. Appendix D and Chapter 16 provide a discussion of techniques for random sampling in nontrivial cases (see also Exercise 2.4).

A recursive implementation of the simple random search idea is as follows. This algorithm, called algorithm A here, applies when $\boldsymbol{\theta}$ has continuous, discrete, or hybrid elements.

Algorithm A: Simple Random ("Blind") Search

Step 0 (**Initialization**) Choose an initial value of $\boldsymbol{\theta}$, say $\hat{\boldsymbol{\theta}}_0 \in \Theta$, either randomly or deterministically. (If random, usually a uniform distribution on Θ is used.) Calculate $L(\hat{\boldsymbol{\theta}}_0)$. Set $k = 0$.

Step 1 Generate a new independent value $\boldsymbol{\theta}_{\text{new}}(k+1) \in \Theta$, according to the chosen probability distribution. If $L(\boldsymbol{\theta}_{\text{new}}(k+1)) < L(\hat{\boldsymbol{\theta}}_k)$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_{\text{new}}(k+1)$. Else, take $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k$.

Step 2 Stop if the maximum number of L evaluations has been reached or the user is otherwise satisfied with the current estimate for $\boldsymbol{\theta}$ via appropriate stopping criteria; else, return to step 1 with the new k set to the former $k + 1$.

Example 2.1 demonstrates algorithm A on a simple loss function. At the possible expense of belaboring the obvious, we present more details here than in other examples because this is the first stochastic optimization algorithm being covered.

Example 2.1—Illustration of algorithm A on a simple function. With $p = 2$, consider the simple quadratic loss function $L(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{\theta}$ on the domain $\Theta = [1, 3]\times[1, 3]$. The unique minimizing solution to this constrained problem is $\boldsymbol{\theta}^*$

$= [1, 1]^T$. Suppose that an analyst has no knowledge of the functional form of $L(\boldsymbol{\theta})$; rather, the analyst is only able to obtain output L values for given input $\boldsymbol{\theta}$ values. Consider the use of algorithm A with a uniform distribution on Θ for generating the candidate points $\boldsymbol{\theta}_{new}$. Because of the lack of information about L, the analyst lets $\hat{\boldsymbol{\theta}}_0 = [2, 2]^T$, the center point of Θ. Each of the uniformly distributed candidate points in Θ is generated according to the formula $2\boldsymbol{U} + [1, 1]^T$, where $\boldsymbol{U}$ is a two-dimensional random vector in $[0, 1]\times[0, 1]$ generated via the MATLAB pseudorandom generator **rand**.

Table 2.1 shows the details from the first six iterations of one run. Even though only a few iterations are shown, there is ample evidence of the algorithm converging to $\boldsymbol{\theta}^*$. The $\boldsymbol{\theta}$ estimate moves from $[2, 2]^T$ to $[1.34, 1.76]^T$ and the loss value drops from 8.00 to 4.89. Note that in three of the six iterations (at $k = 2$, 4, and 5), the candidate point $\boldsymbol{\theta}_{new}$ is rejected because it does not offer an improvement over the current estimate. The fraction of candidate points rejected will increase as the number of iterations increase because the portion of Θ offering a lower loss value decreases. Equivalently, the *rate* of decay in the loss value (as shown in the last column in Table 2.1) will be reduced. ❑

Theorem 2.1 shows that algorithm A converges a.s. to $\boldsymbol{\theta}^*$ under reasonable conditions. Informally, Theorem 2.1 says that if values of L at or near $L(\boldsymbol{\theta}^*)$ can be reached with nonzero probability based on the sampling probability chosen for generating $\boldsymbol{\theta}_{new}(k)$ (the sampling probability is allowed to vary with k), then $\hat{\boldsymbol{\theta}}_k$ will converge to $\boldsymbol{\theta}^*$ as $k \to \infty$. This theorem uses the concept of the infimum (inf) of a function, which corresponds to the *greatest lower bound* to the function on the specified domain (see Appendix A).

Table 2.1. Details of the first several iterations of algorithm A for the constrained problem with quadratic loss function in Example 2.1. Note decreasing $L(\hat{\boldsymbol{\theta}}_k)$.

k	$\boldsymbol{\theta}_{new}(k)^T$	$L(\boldsymbol{\theta}_{new}(k))$	$\hat{\boldsymbol{\theta}}_k^T$	$L(\hat{\boldsymbol{\theta}}_k)$
0	—	—	[2.00, 2.00]	8.00
1	[2.25, 1.62]	7.69	[2.25, 1.62]	7.69
2	[2.81, 2.58]	14.55	[2.25, 1.62]	7.69
3	[1.93, 1.19]	5.14	[1.93, 1.19]	5.14
4	[2.60, 1.92]	10.45	[1.93, 1.19]	5.14
5	[2.23, 2.58]	11.63	[1.93, 1.19]	5.14
6	[1.34, 1.76]	4.89	[1.34, 1.76]	4.89

Unlike most of the other algorithms in this book, we present the proof of convergence here. Although not trivial, this proof is *relatively* simple as a consequence of the simplicity of the algorithm. Nevertheless, despite the relative simplicity, the proof provides a flavor of some of the arguments that are made in the more difficult proofs not included in this book.

Theorem 2.1. Suppose that $\boldsymbol{\theta}^*$ is the unique minimizer of L on the domain Θ (i.e., Θ^* contains only one point), where $L(\boldsymbol{\theta}^*) > -\infty$ and

$$\inf_{\boldsymbol{\theta} \in \Theta, \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \geq \eta} L(\boldsymbol{\theta}) > L(\boldsymbol{\theta}^*) \tag{2.1}$$[1]

for all $\eta > 0$. Suppose further that for any $\eta > 0$ and for all k, there exists a function $\delta(\eta) > 0$ such that

$$P[\boldsymbol{\theta}_{\text{new}}(k): L(\boldsymbol{\theta}_{\text{new}}(k)) < L(\boldsymbol{\theta}^*) + \eta] \geq \delta(\eta). \tag{2.2}$$

Then, for algorithm A with noise-free loss measurements, $\hat{\boldsymbol{\theta}}_k \to \boldsymbol{\theta}^*$ a.s. as $k \to \infty$.

Proof. (The mathematical level of this proof is slightly beyond the prerequisites of this book, but should be largely accessible to those who understand the material of Appendix C. The proof may be skipped without significantly affecting the understanding of subsequent material.) Because $L(\boldsymbol{\theta}^*) > -\infty$ and $L(\hat{\boldsymbol{\theta}}_k)$ is monotonically nonincreasing due to having noise-free loss measurements, $\lim_{k \to \infty} L(\hat{\boldsymbol{\theta}}_k(\omega))$ exists for each underlying sample point ω, as in Appendix C (e.g., Apostol, 1974, p. 185). Let $S_\eta = \{\boldsymbol{\theta}: L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}^*) + \eta\}$. By condition (2.2), $P(\boldsymbol{\theta}_{\text{new}}(k) \in S_\eta) \geq \delta(\eta)$ for any $\eta > 0$. Hence, by the independence of the sampling distribution, $P(\boldsymbol{\theta}_{\text{new}}(k) \notin S_\eta \ \forall\ k) \leq [1 - \delta(\eta)]^k \to 0$ as $k \to \infty$. So, $\lim_{k \to \infty} L(\hat{\boldsymbol{\theta}}_k) < L(\boldsymbol{\theta}^*) + \eta$ in pr. Because $\eta > 0$ is arbitrarily small, we know that $L(\hat{\boldsymbol{\theta}}_k) \to L(\boldsymbol{\theta}^*)$ in pr. as $k \to \infty$. Then by a standard result in probability theory (e.g., Laha and Rohatgi, 1979, Proposition 1.3.4 and problem 25(b) on p. 63), this implies that $L(\hat{\boldsymbol{\theta}}_{k_j}) \to L(\boldsymbol{\theta}^*)$ a.s. as $j \to \infty$ for some subsequence $\{k_j\}$.

Given the convergence of $L(\hat{\boldsymbol{\theta}}_k)$ to *some* limit for each sample point ω mentioned above, and the fact that the subsequence has to converge to the same point as the full sequence, we know that the a.s. limit of $L(\hat{\boldsymbol{\theta}}_k)$ must also be

[1] The left-hand side of inequality (2.1) should be read as: "The infimum of $L(\boldsymbol{\theta})$ over the set of $\boldsymbol{\theta}$ such that $\boldsymbol{\theta} \in \Theta$ and $\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \geq \eta$." Obviously, η must be such that at least some $\boldsymbol{\theta}$ satisfying $\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \geq \eta$ lie in Θ.

$L(\theta^*)$. The uniqueness of θ^* as given in (2.1) then implies that $\hat{\theta}_k \to \theta^*$ a.s., as we set out to prove. ❑

Figure 2.1 presents three example functions and sampling distributions, two of which (examples (a) and (b)) satisfy the important theorem condition (2.2) and one of which (example (c)) does not. In Figure 2.1(a), there is always some nonzero probability of producing a loss value arbitrarily close to $L(\theta^*)$, while in Figure 2.1(b), there is nonzero probability of landing directly on θ^*; in either case, clearly condition (2.2) is satisfied. On the other hand, for the discontinuous function of Figure 2.1(c), the probability is zero of generating $\theta_{new} = \theta_{new}(k)$ such that $|L(\theta_{new}) - L(\theta^*)| < \eta$ for all η sufficiently small because of the gap between $L(\theta)$ and $L(\theta^*)$ for all $\theta \neq \theta^*$ (recall that $P(\theta_{new} = \theta^*) = 0$ because the sampling is from a continuous distribution). Clearly, condition (2.2) is violated because $P(\theta_{new}: L(\theta_{new}) < L(\theta^*) + \eta) = 0$ for any $0 < \eta < |L(0) - L(\theta^*)|$ since $\theta^* \neq 0$. (Exercise 2.1 shows that Theorem 2.1 applies to the problem of Example 2.1.)

While Theorem 2.1 establishes convergence of the simple random search algorithm, it is also of interest to examine the *rate* of convergence. The rate is intended to tell the analyst how close $\hat{\theta}_k$ is likely to be to θ^* for a given cost of search. As motivated in Subsection 1.2.1, the cost of search here (and through

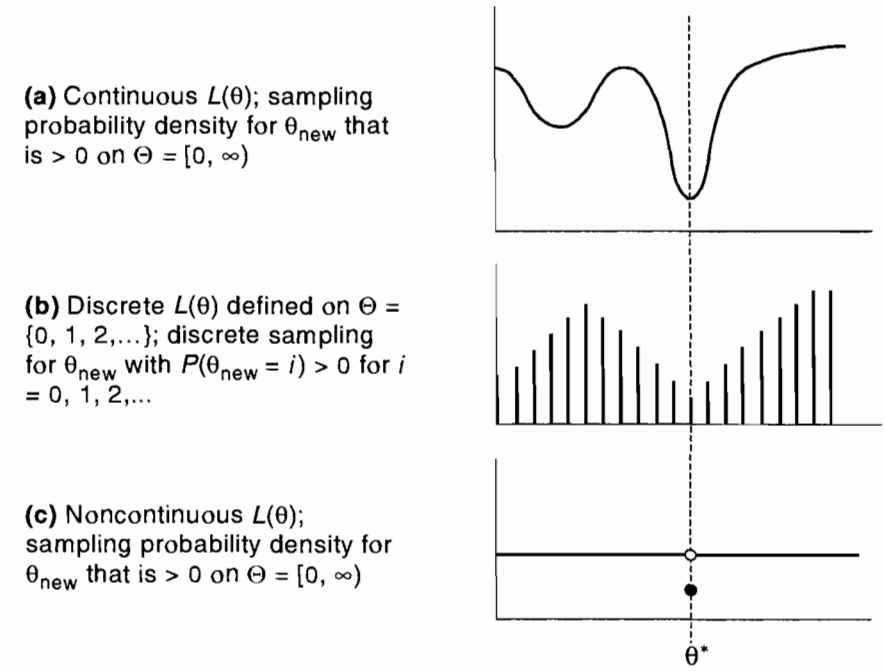

Figure 2.1. Applicability/nonapplicability of Theorem 2.1. Three example loss functions ($L(\theta)$ versus θ for scalar $\theta \geq 0$) and sampling distributions for $\theta_{new} = \theta_{new}(k)$. Condition (2.2) of Theorem 2.1 holds in settings (a) and (b), but does not hold in (c).

much of this book) is expressed in terms of number of loss function evaluations. Knowledge of the rate of convergence is critical in practical applications. Simply knowing that an algorithm converges begs the question of whether the algorithm will yield a practically acceptable solution in any reasonable period. To evaluate the rate, let us specify a *satisfactory region* $S(\boldsymbol{\theta}^*)$ representing some neighborhood of $\boldsymbol{\theta}^*$ providing acceptable accuracy in our solution (e.g., $S(\boldsymbol{\theta}^*)$ might represent a hypercube about $\boldsymbol{\theta}^*$ with the length of each side representing a tolerable error in each coordinate of $\boldsymbol{\theta}$). Suppose that the algorithm is terminated once an iterate lands in $S(\boldsymbol{\theta}^*)$ (perhaps because the L value for any $\boldsymbol{\theta}$ in $S(\boldsymbol{\theta}^*)$ is fully acceptable for the application of interest). So, if $\hat{\boldsymbol{\theta}}_k$ represents the iterate that first lands in $S(\boldsymbol{\theta}^*)$, then $\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k+1} = \ldots = \hat{\boldsymbol{\theta}}_n$, where n is some prior value of maximum number of iterations.

Given the independence of the samples at each iteration, the probability of never landing in $S(\boldsymbol{\theta}^*)$ in n iterations for algorithm A is $\prod_{k=1}^{n}[1 - P(\boldsymbol{\theta}_{\text{new}}(k) \in S(\boldsymbol{\theta}^*))]$. Hence, the probability of having a $\boldsymbol{\theta}_{\text{new}}(k)$ land in $S(\boldsymbol{\theta}^*)$ within $k = 1, 2, \ldots, n$ iterations (effectively stopping the algorithm) is

$$P(\hat{\boldsymbol{\theta}}_n \in S(\boldsymbol{\theta}^*)) = 1 - \prod_{k=1}^{n}[1 - P(\boldsymbol{\theta}_{\text{new}}(k) \in S(\boldsymbol{\theta}^*))]. \tag{2.3}$$

One can use (2.3) to answer the question: How many iterations (with one loss function evaluation per iteration) will it take before the $\boldsymbol{\theta}$ estimate is guaranteed to land in $S(\boldsymbol{\theta}^*)$ with probability $1 - \rho$ (ρ a small number)? Setting the left-hand side of (2.3) to $1 - \rho$ and supposing that there is a constant sampling probability $P^* = P(\boldsymbol{\theta}_{\text{new}}(k) \in S(\boldsymbol{\theta}^*))$ for all k, we have

$$n = \frac{\log \rho}{\log(1 - P^*)} \tag{2.4}$$

(see also Exercise 2.3). Here, and elsewhere in the book, $\log(x)$ or $\log x$ represents the *natural* (base $e = 2.71828\ldots$) logarithm of x.

Although (2.4) may appear benign at first glance, this expression grows rapidly as p gets large because of the rate at which P^* approaches 0 (since the relative volume of $S(\boldsymbol{\theta}^*)$ to volume of Θ decays geometrically as p increases). Hence, (2.4) shows the extreme inefficiency of algorithm A in high-dimensional problems, as illustrated in Example 2.2. While (2.3) is in terms of the iterate $\hat{\boldsymbol{\theta}}_n$, a result related to the rate of convergence for $L(\hat{\boldsymbol{\theta}}_n)$ is given in Pflug (1996, p. 24). This result is in terms of extreme-value distributions and also confirms the inefficiency of algorithm A in high-dimensional problems. Despite this inefficiency, the no free lunch theorems of Subsection 1.2.2 indicate that *when averaged across all possible problems*, algorithm A is as efficient as any other method. In practice, however, other algorithms usually have greater *practical* efficiency because of the restrictions to *specific* problems having specific characteristics.

Example 2.2—Varying efficiency of simple random search (algorithm A) as p changes. Let $\Theta = [0, 1]^p$ (the p-dimensional hypercube with minimum and maximum values of 0 and 1 for each component of $\boldsymbol{\theta}$) and suppose that uniform sampling on Θ is used to generate $\boldsymbol{\theta}_{\text{new}}(k)$ for all k. We want to guarantee with probability at least 0.90 that each element of $\boldsymbol{\theta}$ is within 0.04 units of the optimal value. Let the (unknown) optimal $\boldsymbol{\theta}$, $\boldsymbol{\theta}^*$, lie in a p-fold Cartesian product of open intervals, $(0.04, 0.96)^p$, and $\boldsymbol{\theta}^* = [t_1^*, t_2^*, \ldots, t_p^*]^T$. Hence,

$$S(\boldsymbol{\theta}^*) = [t_1^* - 0.04,\ t_1^* + 0.04] \times [t_2^* - 0.04,\ t_2^* + 0.04] \times \ldots \times [t_p^* - 0.04,\ t_p^* + 0.04]$$

and $P^* = 0.08^p$. How many iterations are required to ensure that we land in $S(\boldsymbol{\theta}^*)$ with a probability of 0.90 (i.e., $\rho = 0.10$)? Table 2.2 provides the answer for some values $1 \leq p \leq 10$.

Because each iteration requires one loss evaluation, Table 2.2 illustrates the explosive growth in the number of loss evaluations needed as p increases. This demonstrates the inefficiency of algorithm A in practical high-dimensional problems. ❑

Algorithm A is the simplest random search in that the sampling generating the new $\boldsymbol{\theta}$ value at each iteration is over the entire domain of interest. The sampling does not take account of where the previous estimates of $\boldsymbol{\theta}$ have been. The two algorithms below, although still simple, are slightly more sophisticated in that the random sampling is a function of the position of the current best estimate for $\boldsymbol{\theta}$. In this way, the search is more localized in the neighborhood of that estimate, allowing for a better exploitation of information that has previously been obtained about the shape of the loss function.

Such algorithms are sometimes referred to as *localized algorithms* to emphasize their dependence on the local environment near the current estimate for $\boldsymbol{\theta}$. This terminology is not to be confused with the global versus local algorithms discussed in Chapter 1, where the emphasis is on searching for a global or local solution to the optimization problem. In fact, sometimes a localized algorithm is guaranteed to provide a global solution, as discussed below following the presentation of the first localized algorithm.

Algorithm B is the first of the two localized algorithms we consider. This algorithm was described in Matyas (1965) and Jang et al. (1997, pp. 186–189).

Table 2.2. Number of iterations n to ensure iterate lands in $S(\boldsymbol{\theta}^*)$ with probability 0.90 for varying problem dimension p.

p	1	2	5	10
n	28	359	7.0×10^5	2.1×10^{11}

Algorithm B: Localized Random Search

Step 0 **(Initialization)** Pick an initial guess $\hat{\boldsymbol{\theta}}_0 \in \Theta$, either randomly or with prior information. Set $k = 0$.

Step 1 Generate an independent random vector $\boldsymbol{d}_k \in \mathbb{R}^p$ and add it to the current $\boldsymbol{\theta}$ value, $\hat{\boldsymbol{\theta}}_k$. Check if $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k \in \Theta$. If $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k \notin \Theta$, generate a new $\boldsymbol{d}_k$ and repeat or, alternatively, move $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k$ to the nearest valid point within Θ. Let $\boldsymbol{\theta}_{\text{new}}(k+1)$ equal $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k \in \Theta$ or the aforementioned nearest valid point in Θ.

Step 2 If $L(\boldsymbol{\theta}_{\text{new}}(k+1)) < L(\hat{\boldsymbol{\theta}}_k)$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_{\text{new}}(k+1)$; else, set $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k$.

Step 3 Stop if the maximum number of L evaluations has been reached or the user is otherwise satisfied with the current estimate for $\boldsymbol{\theta}$ via appropriate stopping criteria; else, return to step 1 with the new k set to the former $k + 1$.

Although Matyas (1965) and others have used the (multivariate) normal distribution for generating $\boldsymbol{d}_k$, the user is free to set the distribution of the deviation vector $\boldsymbol{d}_k$. The distribution should have mean zero and each component should have a variation (e.g., standard deviation) consistent with the magnitudes of the corresponding $\boldsymbol{\theta}$ elements. So, for example, if the magnitude of the first component in $\boldsymbol{\theta}$ lies between 0 and 0.05 while the magnitude of the second component is between 0 and 500, the corresponding standard deviations in the components of $\boldsymbol{d}_k$ might also vary by a magnitude of 10,000). This allows the algorithm to assign roughly equal weight to each of the components of $\boldsymbol{\theta}$ as it moves through the search space.

Although not formally allowed in the convergence theorem below, it is often advantageous in practice if the variability in $\boldsymbol{d}_k$ is reduced as k increases. This allows one to focus the search more tightly as evidence is accrued on the location of the solution (as expressed by the location of our current estimate $\hat{\boldsymbol{\theta}}_k$). A simple implementation of this idea would be to reduce the variances by a factor such as k when the normal distribution is used in generating the $\boldsymbol{d}_k$. For the numerical studies below, we use the simple (constant variance) sampling, $\boldsymbol{d}_k \sim N(\boldsymbol{0}, \rho^2 \boldsymbol{I}_p)$ for all k where ρ^2 represents the (common) variance of each of the components in $\boldsymbol{d}_k$.

The convergence theory for the localized algorithms tends to be more restrictive than the theory for algorithm A. Solis and Wets (1981) provide a theorem for global convergence of localized algorithms, but the theorem conditions may not be verifiable in many practical applications. Their theorem would, in principle, cover both algorithm B and the enhanced algorithm C below. Other results related to formal convergence to global optima of various localized random search algorithms appear, for example, in Yakowitz and Fisher (1973) and Zhigljavsky (1991, Chap. 3). An earlier theorem from Matyas (1965) (with

proof corrected in Baba et al., 1977) provides for global convergence of algorithm B if L is a continuous function. The convergence is in the "in probability" (pr.) sense (Appendix C). The theorem allows for more than one global minimum to exist in Θ. Therefore, in general, the result provides no guarantee of $\hat{\boldsymbol{\theta}}_k$ ever settling near any one value $\boldsymbol{\theta}^*$. We present the theorem below.

Theorem 2.2. Let Θ^* represent the set of global minima for L (see Section 1.1). Suppose that L is continuous on a bounded domain Θ and that if $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k \notin \Theta$ at a given iteration, a new $\boldsymbol{d}_k$ is randomly generated (versus the other option of bringing $\hat{\boldsymbol{\theta}}_k + \boldsymbol{d}_k$ back to the nearest point within Θ). For any $\eta > 0$, let $R_\eta = \bigcup_{\boldsymbol{\theta}^* \in \Theta^*} \{\boldsymbol{\theta} : |L(\boldsymbol{\theta}) - L(\boldsymbol{\theta}^*)| < \eta\}$ (i.e., the union over all $\boldsymbol{\theta}^* \in \Theta^*$ of the sets of $\boldsymbol{\theta}$ such that $L(\boldsymbol{\theta})$ is near each $L(\boldsymbol{\theta}^*)$). Then, for algorithm B with the $\boldsymbol{d}_k$ having an i.i.d. $N(\boldsymbol{0}, \boldsymbol{I}_p)$ distribution, $\lim_{k \to \infty} P(\hat{\boldsymbol{\theta}}_k \in R_\eta) = 1$.

Algorithm B above might be considered the most naïve of the localized random search algorithms. More sophisticated approaches are also easy to implement. For instance, if a search in one direction increases L, then it is likely to be beneficial to move in the opposite direction. Further, successive iterations in a direction that tend to consistently reduce L should encourage further iterations in the same direction. Many algorithms exploiting these simple properties exist (e.g., Solis and Wets, 1981; Zhigljavsky, 1991, Chap. 3; Li and Rhinehart, 1998; and the nonlinear simplex algorithm of Section 2.4). An extensive survey emphasizing such algorithms developed prior to 1980 is given in Schwefel (1995, pp. 87–89).

An example algorithm is given below (from Jang et al., 1997, pp. 187–188), which is a slight simplification of an algorithm in Solis and Wets (1981). The full Solis and Wets algorithm includes an even greater degree of adaptivity to the current environment, but this comes at the expense of more complex implementation.

Algorithm C: Enhanced Localized Random Search

Step 0 **(Initialization)** Pick an initial guess $\hat{\boldsymbol{\theta}}_0 \in \Theta$, either randomly or with prior information. Set $k = 0$. Set bias vector $\boldsymbol{b}_0 = 0$.

Step 1 Generate an independent random vector $\boldsymbol{d}_k$ and add it and the bias term $\boldsymbol{b}_k$ to the current value $\hat{\boldsymbol{\theta}}_k$ (as in algorithm B, a standard distribution for $\boldsymbol{d}_k$ is $N(\boldsymbol{0}, \rho^2 \boldsymbol{I}_p)$). Check if $\hat{\boldsymbol{\theta}}_k + \boldsymbol{b}_k + \boldsymbol{d}_k \in \Theta$. If $\hat{\boldsymbol{\theta}}_k + \boldsymbol{b}_k + \boldsymbol{d}_k \notin \Theta$, generate a new $\boldsymbol{d}_k$ and repeat; alternatively, move $\hat{\boldsymbol{\theta}}_k + \boldsymbol{b}_k + \boldsymbol{d}_k$ to the nearest valid point within Θ. Let $\boldsymbol{\theta}_{\text{new}}(k+1)$ equal $\hat{\boldsymbol{\theta}}_k + \boldsymbol{b}_k + \boldsymbol{d}_k \in \Theta$ or the above-mentioned nearest valid point in Θ.

Step 2 If $L(\boldsymbol{\theta}_{\text{new}}(k+1)) < L(\hat{\boldsymbol{\theta}}_k)$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_{\text{new}}(k+1)$ and $\boldsymbol{b}_{k+1} = 0.2\boldsymbol{b}_k + 0.4\boldsymbol{d}_k$; go to step 5. Otherwise, go to step 3.

Step 3 Analogous to step 1, let $\boldsymbol{\theta}'_{\text{new}}(k+1) = \hat{\boldsymbol{\theta}}_k + \boldsymbol{b}_k - \boldsymbol{d}_k \in \Theta$ or its altered version within Θ. If $L(\boldsymbol{\theta}'_{\text{new}}(k+1)) < L(\hat{\boldsymbol{\theta}}_k)$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}'_{\text{new}}(k+1)$ and $\boldsymbol{b}_{k+1} = \boldsymbol{b}_k - 0.4\boldsymbol{d}_k$; go to step 5.[2] Otherwise, go to step 4.

Step 4 Set $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k$ and $\boldsymbol{b}_{k+1} = 0.5\boldsymbol{b}_k$. Go to step 5.

Step 5 Stop if the maximum number of L evaluations has been reached or the user is otherwise satisfied with the current estimate for $\boldsymbol{\theta}$ via appropriate stopping criteria; else, return to step 1 with the new k set to the former $k+1$.

2.2.3 Example Implementations

Examples 2.3 and 2.4 report on a comparison of the three random search techniques above for a relatively simple $p = 2$ quartic (fourth-order) polynomial loss function. This function was used in Styblinski and Tang (1990, Example 1) to test simulated annealing algorithms. While the examples suggest the superiority of algorithm C, one should keep in mind the cautionary note on the limits of numerical studies in Subsection 1.2.1 together with the limits implied by the no free lunch theorems of Subsection 1.2.2. In particular, the conclusion of these examples may not apply to another problem of interest to the reader. Further, the low dimensionality of this example allows algorithm A to be more competitive than it would be in many realistic problems where $p > 2$. A larger-dimensional ($p = 10$) comparison is given in Example 2.5. Consistent with the discussion of Subsection 1.2.1, the algorithms are compared based on a common number of loss evaluations.

Example 2.3—Comparison of random search techniques A, B, and C (part 1). Let $\boldsymbol{\theta} = [t_1, t_2]^T$ and consider the loss function

$$L(\boldsymbol{\theta}) = \frac{1}{2}\left[(t_1^4 - 16t_1^2 + 5t_1) + (t_2^4 - 16t_2^2 + 5t_2)\right]$$

as given in Styblinski and Tang (1990, Example 1). Suppose that $\Theta = [-8, 8]^2$. Figure 2.2 shows that this function has four local minima, one of which is the unique global minimum at $\boldsymbol{\theta}^* = [-2.9035, -2.9035]^T$ (see Exercise 2.8).

The three algorithms are initialized at $\hat{\boldsymbol{\theta}}_0 = [4.0, 6.4]^T$ as in Styblinski and Tang (1990). Note that $L(\hat{\boldsymbol{\theta}}_0) = 537.2$ and $L(\boldsymbol{\theta}^*) = -78.33$. Each algorithm is run using 500 loss measurements. The random perturbations $\boldsymbol{d}_k$ in algorithms B and C are generated according to a $N(\boldsymbol{0}, \rho^2\boldsymbol{I}_2)$ distribution; ρ is set to 3.0 for both algorithms, found to be approximately optimal after some preliminary numerical testing. Table 2.3 presents the mean values of the loss function at the final $\boldsymbol{\theta}$ estimates over 40 independent runs. Below the mean values are approximate 95

[2] There is no typographical error in step 3; unlike step 2, $\boldsymbol{b}_{k+1}$ in step 3 does not include the multiplier 0.2 on $\boldsymbol{b}_k$.

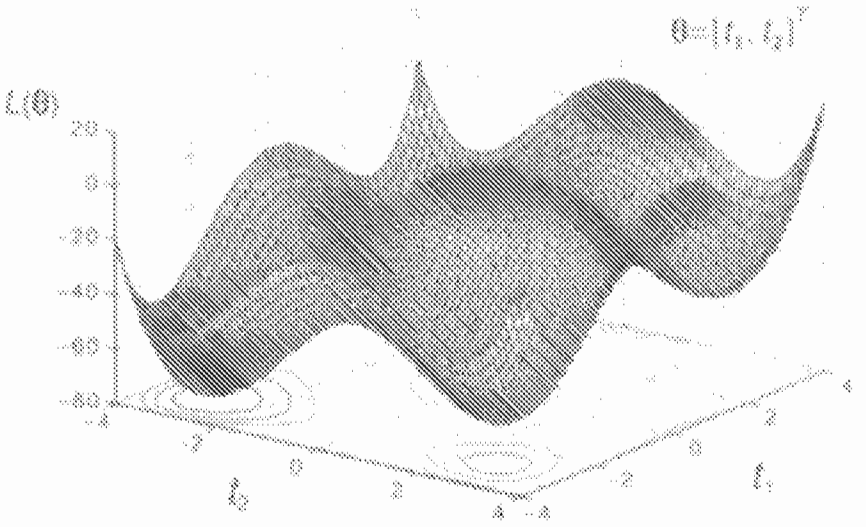

Figure 2.2. Multimodal loss function with unique global minimum at $\boldsymbol{\theta}^*$ near $[-2.9, -2.9]^T$ (Examples 2.3 and 2.4). Plot is on subset of Θ.

percent confidence intervals constructed according to a t-distribution with 40 – 1 = 39 degrees of freedom (see Appendix B). For each algorithm, these intervals are derived from the sample variance s^2 of the terminal loss values over the sample of 40 terminal values. Using basic statistical principles discussed in Appendix B, each confidence interval is constructed according to

$$\left[\text{sample mean} - 2.023\sqrt{s^2/40}\,,\ \ \text{sample mean} + 2.023\sqrt{s^2/40}\right],$$

where 2.023 is the t-value for 0.025 probability (for each of the two tails) with 39 degrees of freedom.

Table 2.3 shows slight evidence that algorithm C is superior in this problem, although the upper value of the confidence interval for algorithm C

Table 2.3. Sample means and approximate 95 percent confidence intervals for terminal values $L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ from algorithms A, B, and C.

Algorithm A	Algorithm B	Algorithm C
2.51 [1.94, 3.08]	0.78 [0.51, 1.04]	0.49 [0.32, 0.67]

overlaps the lower value of the interval for algorithm B (even with shorter 90 percent—versus 95 percent—confidence intervals, there remains some small overlap; see Exercise 2.10). Using the information here (also from Exercise 2.10), a more direct comparison of algorithms B and C is available by constructing the two-sample t-statistic in (B.4) from Appendix B. This yields $t = 1.80$ (for the loss from algorithm B minus the loss from algorithm C). Given that we do not know that the variances of the loss measurements are identical, the nonidentical variance form in (B.3c) of Appendix B is relevant, leading to a degrees of freedom of approximately $68.9 \approx 69$. From the TDIST function in the MS EXCEL spreadsheet, the two-sided P-value is 0.076 (P-values are discussed in Appendix B). This P-value provides *some*—but not overwhelming—evidence of a significant difference between algorithms B and C in this problem. ❑

Example 2.4—Typical loss values and θ estimates for random search algorithms A, B, and C (part 2). Let us continue with the problem of Example 2.3. To evaluate what a "typical" (single) run of algorithm A, B, or C would produce, let us compare the values of $\boldsymbol{\theta}$ after 500 loss evaluations. For algorithms A and B, this corresponds to the value of $\hat{\boldsymbol{\theta}}_{499}$; for algorithm C, the number of iterations is generally less than 499 because some of the iterations may take two loss measurements. Table 2.4 shows typical standardized loss values $L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ together with the terminal $\boldsymbol{\theta}$ estimate that produced those loss values. The loss values are chosen to be the ones closest to the mean loss value in Example 2.3. Each $\boldsymbol{\theta}$ estimate is the value for the final iterate $\hat{\boldsymbol{\theta}}_k$ on the run producing the typical loss value.

The order of the loss values in Table 2.4 is consistent with the relative distances from $\hat{\boldsymbol{\theta}}_k$ to $\boldsymbol{\theta}^*$ (as given by $\|\hat{\boldsymbol{\theta}}_k - \boldsymbol{\theta}^*\|$) of 0.404, 0.224, and 0.173 for algorithms A, B, and C. Such consistent ordering does not always occur in practice. ❑

Table 2.4. Typical terminal $L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ and $\hat{\boldsymbol{\theta}}_k$ values for algorithms A, B, and C (recall that $\boldsymbol{\theta}^* = [-2.9035, -2.9035]^T$).

	Algorithm A	Algorithm B	Algorithm C
$L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ (one of the 40 runs in Example 2.3)	2.60	0.80	0.49
$\hat{\boldsymbol{\theta}}_k$ for above L value	$\begin{bmatrix} -2.547 \\ -3.093 \end{bmatrix}$	$\begin{bmatrix} -2.680 \\ -2.898 \end{bmatrix}$	$\begin{bmatrix} -2.740 \\ -2.959 \end{bmatrix}$

Recall from the discussion of Subsection 1.2.1 that the tunable algorithm coefficients can have a dramatic effect on algorithm performance. For algorithms B and C, the adjustable coefficients pertained to the distribution of the perturbation vector $\boldsymbol{d}_k$. Since we took $\boldsymbol{d}_k$ as normally distributed with covariance matrix of the form $\rho^2 \boldsymbol{I}_2$, this reduced to picking the value of the standard deviation ρ. A value of $\rho = 1.0$ (versus 3.0 in the example) significantly degraded the performance of algorithms B and C (increasing the relevant loss differences from below 1.0 in Tables 2.3 and 2.4 to over 20).

We now present a larger dimensional example related to a well-known test function.

Example 2.5—Random search algorithms applied to Rosenbrock function. Consider the well-known Rosenbrock test function in the optimization literature. This test function was first presented in Rosenbrock (1960) for the $p = 2$ setting, with higher-dimensional extensions given in, among other places, Moré et al. (1981) (note the typographical error in the function presentation on line 21(b) of p. 26 of the Moré et al. reference: an $i - 1$ should be $2i - 1$). This test function has an interesting shape in that the solution lies in a curved valley when considered in two-dimensional space. For general p, the function has the fourth-order polynomial form

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{p/2} \left[100(t_{2i} - t_{2i-1}^2)^2 + (1 - t_{2i-1})^2 \right], \tag{2.5}$$

where $\boldsymbol{\theta} = [t_1, t_2, \ldots, t_p]^T$ and p is divisible by 2.

Let us consider a problem where $p = 10$. Note that $L(\boldsymbol{\theta}^*) = 0$ at $\boldsymbol{\theta}^* = [1, 1, \ldots, 1]^T$. Let the constraint set be the Cartesian product of intervals $[-4, 4]$: $\Theta = [-4, 4]^{10}$. Table 2.5 presents the results of the study based on 40 independent runs for each algorithm. Each run of algorithms A and B uses 1000 loss evaluations; algorithm C is terminated at the 1000th loss evaluation or the first possible loss after the 1000th (it is not possible to specify a priori the exact number of measurements needed by algorithm C). Each run was started at the

Table 2.5. Sample means and approximate 95 percent confidence intervals for terminal Rosenbrock loss values $L(\hat{\boldsymbol{\theta}}_k)$ from algorithms A, B, and C (loss values relative to $L(\hat{\boldsymbol{\theta}}_0) = 121$; $L(\boldsymbol{\theta}^*) = 0$).

Algorithm A	Algorithm B	Algorithm C
121 [121, 121]	20.07 [19.78, 20.36]	19.80 [19.23, 20.37]

initial condition $\hat{\boldsymbol{\theta}}_0 = [-1.2, 1, -1.2, 1, \ldots, 1]^T$ (so $L(\hat{\boldsymbol{\theta}}_0) = 121$; this is a common initial condition in numerical studies with this function). The confidence intervals are computed in the manner of Example 2.3. In algorithm A, standard uniform sampling over Θ is used. For algorithms B and C, the perturbations are in the standard form $\boldsymbol{d}_k \sim N(\mathbf{0}, \rho^2 \boldsymbol{I}_{10})$, with $\rho = 0.05$.

Algorithm A does not move away from the initial condition in any of the 40 runs due to the large size of the sampling region (relative to regions of Θ that produce improvements in the loss function). That is, not one of the 40,000 total loss evaluations from uniform random sampling in Θ produced a loss value lower than the initial value (recall the "curse of dimensionality" in Subsection 1.2.1). Algorithms B and C perform better that algorithm A, with both B and C showing significant improvement relative to the initial condition. It is clear, however, that more loss evaluations are required to bring the solution close to $L(\boldsymbol{\theta}^*) = 0$. ❑

2.3 RANDOM SEARCH WITH NOISY LOSS MEASUREMENTS

The description and analysis of the random search algorithms above assume perfect (noise-free) measurements of the loss function. This is usually considered a critical part of such algorithms (Pflug, 1996, p. 25). In contrast to the noise-free case, random search methods with noisy loss evaluations of the form $y(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta})$ generally do not formally converge. However, there are means by which the random search techniques can be modified to accommodate noisy measurements, at least on a heuristic basis. Some of the limited formal convergence theory for random search as applied to the noisy measurement case includes Yakowitz and Fisher (1973, Sect. 4) and Zhigljavsky (1991, Chap. 3).

The most obvious way to attempt to cope with noise is to collect several (possibly many) function evaluations $y(\boldsymbol{\theta})$ at each value of $\boldsymbol{\theta}$ generated in the search process and then average these values. If the number of function evaluations at each $\boldsymbol{\theta}$ is sufficiently large (relative to the noise magnitude), then the averaged value can be effectively treated as one perfect measurement of the loss. As mentioned in Subsection 1.2.1, however, this can be costly since the error decreases only at the rate of $1/\sqrt{N}$ when averaging N function evaluations with independent noise.

Another approach is to alter the algorithm's key decision criterion to build in some robustness to the noise. In particular, the algorithm resists changing the current best estimate for $\boldsymbol{\theta}$ unless there is "significant" probabilistic evidence that the new value will lead to an improved value of the loss. As an example, the acceptance threshold in step 2 of algorithm B can be altered to:

Step 2–modified If $y(\boldsymbol{\theta}_{\text{new}}(k+1)) < y(\hat{\boldsymbol{\theta}}_k) - \tau_k$, set $\hat{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_{\text{new}}(k+1)$; else set $\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k$, where $\tau_k \geq 0$. (2.6)

The same idea applies to step 1 of algorithm A or steps 2 and 3 of algorithm C.

It may be convenient to view the τ_k in terms of the number of standard deviations in the measurement noise that the new measurement must improve on the old measurement before $\boldsymbol{\theta}$ will be changed. So, if one had estimated the standard deviation of the measurement noise to be 0.1 for all k, then $\tau_k = 0.2$ indicates that a new $\boldsymbol{\theta}$ is accepted only if it shows a "two-sigma" improvement in the *measured* loss value (using the standard vernacular where *sigma* represents a standard deviation). The sigma interpretation is often convenient if the noise is at least approximately normally distributed. A simple table lookup reveals the approximate probability of making a right or wrong decision (in terms of whether the new $\boldsymbol{\theta}$ value really is better or worse than the current value).

Obviously, both of the averaging and altered threshold approaches for coping with noise have significant shortcomings. Averaging may greatly increase the number of loss evaluations required to obtain sufficient accuracy. The altered threshold may suffer the same shortcoming by rejecting too many changes in $\boldsymbol{\theta}$ due to the conservative criterion. Nevertheless, the presence of noise in the loss evaluations—even a small amount of noise—makes the optimization problem so much harder than deterministic problems that there is little choice but to accept these penalties if one wants to use a simple random search. We will see in later chapters that other general approaches, such as stochastic approximation, tend to be more adept at coping with noise at the price of a more restrictive problem setting.

Let us consider an example of algorithm B applied in a problem with noise. This example uses both averaging of the loss measurements and the modification of the acceptance threshold as in (2.6). The example is motivated by a problem in wastewater treatment, as described in Spall and Cristion (1997) (based on an earlier model of Dochain and Bastin, 1984). The aim is to find an open-loop controller. More sophisticated closed-loop controllers that take account of the time-varying current state of a system are discussed in Chapters 3, 6, and 7. As part of this example, we illustrate the process of converting a root-finding problem to a minimization problem. The general conversion approach outlined below would apply in other problems as well.

The three related examples below consider this open-loop control problem. Example 2.6 goes through the general process of converting the root-finding problem to an optimization problem. Example 2.7 presents numerical results. This numerical example compares a naïve implementation (ignoring the noise) of algorithm B with a "proper" implementation based on averaging of the loss function measurements. Example 2.8 continues with the wastewater example as an illustration of the general degrading effects of having noise in the optimization process.

Example 2.6—Conversion of root-finding to optimization in the context of two-dimensional problem. Without getting into most of the specifics of the

wastewater treatment problem, let us outline how root-finding can be converted to optimization for which algorithms such as the random search methods A, B, and C readily apply. The original root-finding problem is one of determining a $\boldsymbol{\theta}$ such that $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0} \in \mathbb{R}^2$ based on noisy measurements $\boldsymbol{Y}(\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{e}$, with $\boldsymbol{e}$ being a normally distributed vector. Suppose that the noise $\boldsymbol{e}$ has mean zero and covariance matrix $\text{diag}[\sigma_1^2, \sigma_2^2]$ (so the components of $\boldsymbol{e}$ are independent because of the normality).

Let $\boldsymbol{W}$ be a matrix that assigns relative weights to the two elements of $\boldsymbol{g}(\boldsymbol{\theta})$. In particular, assume that

$$\boldsymbol{W} = \begin{bmatrix} w & 0 \\ 0 & 1-w \end{bmatrix}$$

with $0 < w < 1$. Clearly, finding a $\boldsymbol{\theta}$ such that $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0}$ is equivalent to finding a $\boldsymbol{\theta}$ such that $\boldsymbol{g}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{g}(\boldsymbol{\theta}) = 0$. To convert this relationship into one directly usable with measurements $\boldsymbol{Y}(\boldsymbol{\theta})$, the root-finding problem is converted to an optimization problem $\min_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta})$ according to the loss function

$$L(\boldsymbol{\theta}) = E\left[\boldsymbol{Y}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{Y}(\boldsymbol{\theta})\right] = \boldsymbol{g}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{g}(\boldsymbol{\theta}) + [\sigma_1, \sigma_2] \boldsymbol{W} \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix}. \tag{2.7}$$

(Differentiating (2.7) with respect to $\boldsymbol{\theta}$ illustrates that $\boldsymbol{g}(\boldsymbol{\theta})$ is a general root-finding quantity not necessarily corresponding to $\partial L / \partial \boldsymbol{\theta}$.)

The additive constant (the term after the "+" sign) in (2.7) does not affect the solution for $\boldsymbol{\theta}$. If $\boldsymbol{\theta}$ is such that $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0}$, then $\boldsymbol{\theta}$ minimizes $L(\boldsymbol{\theta})$. Including the additive constant in the loss function, as in (2.7), is convenient because the readily available noisy observation

$$\begin{aligned} y(\boldsymbol{\theta}) &= \boldsymbol{Y}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{Y}(\boldsymbol{\theta}) \\ &= L(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta}) \end{aligned} \tag{2.8}$$

is an unbiased measurement of $L(\boldsymbol{\theta})$ at any $\boldsymbol{\theta}$ (i.e., $E[y(\boldsymbol{\theta})] = L(\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$). Combining (2.7) and (2.8) indicates that the noise $\varepsilon(\boldsymbol{\theta})$ in the loss function measurement is

$$\varepsilon(\boldsymbol{\theta}) = 2\boldsymbol{g}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{e}(\boldsymbol{\theta}) + \boldsymbol{e}(\boldsymbol{\theta})^T \boldsymbol{W} \boldsymbol{e}(\boldsymbol{\theta}) - [\sigma_1, \sigma_2] \boldsymbol{W} \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix}. \tag{2.9}$$

From (2.9), note that the noise in the loss measurements collected during a search process depends on the current estimate for $\boldsymbol{\theta}$. ❑

Example 2.7—Numerical results on optimization with noisy loss measurements in wastewater treatment model. With conventional notation that u represents a control variable, let $\boldsymbol{\theta} = [u_1, u_2]^T$ be the two control inputs for the physical model for the wastewater treatment process. Let the two variances satisfy $\sigma_1^2 = \sigma_2^2 = 1$ and the constraint region $\Theta = [0, 5] \times [0, 5]$; the measurement noise $\boldsymbol{e}$ is normally distributed (so the measurement noise $\varepsilon(\boldsymbol{\theta})$ in (2.9) is *not* normal). From Spall and Cristion (1997), suppose that "nature" provides measurements of the form

$$\boldsymbol{Y}(\boldsymbol{\theta}) = \begin{bmatrix} 1-u_1 \\ 1-0.5u_1u_2-u_2 \end{bmatrix} + \boldsymbol{e}, \tag{2.10}$$

where the top expression on the right-hand side is associated with the methane gas byproduct state and the bottom expression is associated with the water cleanliness state. It is assumed that the analyst does not know the analytical structure on the right-hand side of (2.10), but does know that the two control inputs u_1 and u_2 are somehow used to regulate $\boldsymbol{Y}$. Note that the control u_1 affects both states. Following the discussion in Example 2.6, the noisy measurement $\boldsymbol{Y}$ above can be converted into the loss function measurement in an optimization setting. Suppose that the analyst puts 10 percent weighting ($w = 0.10$) on the methane and 90 percent weighting on the water cleanliness.

From (2.7), an exact solution is available by minimizing the quadratic expression

$$L(\boldsymbol{\theta}) = 0.10(1-u_1)^2 + 0.90(1-0.5u_1u_2-u_2)^2 + 1 \tag{2.11}$$

(the additive "1" contribution from the variances σ_1^2 and σ_2^2 is, of course, irrelevant here since the variances do not depend on $\boldsymbol{\theta}$). The above L yields the unique solution $\boldsymbol{\theta}^* = [1, 2/3]^T$ and $L(\boldsymbol{\theta}^*) = 1.0$ by solving $\partial L/\partial \boldsymbol{\theta} = [\partial L/\partial u_1, \partial L/\partial u_2]^T = \mathbf{0}$ and verifying that the root is a minimum of L.

Table 2.6 contrasts the mean values of $L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ at the terminal iteration over 50 independent runs (the loss value $L(\boldsymbol{\theta}^*)$ has been subtracted out

Table 2.6. Sample means and approximate 95 percent confidence intervals for terminal values of $L(\hat{\boldsymbol{\theta}}_k) - L(\boldsymbol{\theta}^*)$ from algorithm B without and with averaging of noisy loss measurements.

Number of loss measurements y	Algorithm B without averaging	Algorithm B with averaging
100	0.80 [0.59, 1.01]	0.59 [0.47, 0.70]
2000	0.78 [0.64, 0.92]	0.38 [0.28, 0.48]

to more clearly show the relative performance). Below the means are approximate 95 percent confidence intervals derived as in Appendix B ("approximate" because the loss measurements based on eqns. (2.7)–(2.10) are not normally distributed). The table shows four settings. Two are for the small-sample case where only 100 noisy loss measurements are used; the other two are when 2000 measurements are used.

Note that the *true* loss function (2.11) (not the noisy measurements in (2.8)) is used to produce the numbers in Table 2.6. Hence, while the algorithm produces a solution using only noisy measurements, the *evaluation* of the algorithm is with the true loss function. Of course, the true loss would not be available in real-world applications where only noisy measurements are available. The same approach of evaluation based on true loss functions is used throughout this book when Monte Carlo simulations are being used for testing algorithms with noisy function measurements (this was discussed in Subsection 1.2.1).

For each of the sample sizes, the table shows results for algorithm B where no averaging is used (i.e., the noisy loss measurements are used directly in the algorithm in place of the L values) and where an average of four noisy measurements is used at each iteration. To preserve a common number of measurements being used for both the no-averaging and averaging cases, the no-averaging case ran more iterations than the averaging case. All results use the simple sampling, $\boldsymbol{d}_k \sim N(\mathbf{0}, \rho^2 \boldsymbol{I}_2)$, for the perturbations with $\rho = 2$. Constraints are imposed by mapping any component of $\boldsymbol{\theta}$ that lies outside the interval [0, 5] to its nearest end point (0 or 5). An initial condition of $\hat{\boldsymbol{\theta}}_0 = [3, 3]^T$ is used for each of the 50 runs (so $L(\hat{\boldsymbol{\theta}}_0) - L(\boldsymbol{\theta}^*) = 38.43$).

Table 2.6 shows strong evidence of the superiority of algorithm B with averaging in the large-sample case and modest evidence in the small-sample case. The altered acceptance criterion mentioned in (2.6) was also tested in this application, but the averaging method produced lower loss values (of course, this is one specific case and results here may not transfer to another setting). ❑

Example 2.8—Illustration of degrading effects of noise on the optimization process using wastewater treatment model. As a final study in this sequence of examples related to the wastewater model, let us compare the results in Table 2.6 with results based on noise-free loss measurements. The purpose here is to illustrate the detrimental effects of noise in the optimization process, although noise-free loss measurements would *not* be available in a real-world implementation for this application. That is, we are simply using the loss function above as a convenient example while dropping the noise shown in eqns. (2.7) – (2.9) (i.e., $\sigma_1^2 = \sigma_2^2 = 0$ and $\boldsymbol{e} = \mathbf{0}$).

With the algorithm coefficient settings remaining as in Example 2.7—which is not optimal for the noise-free case!—the sample mean loss values are 0.041 and 0.0019 for 100 and 2000 loss measurements, respectively. The best corresponding numbers in Table 2.6 are at least 14 and 200 times larger,

respectively, indicating the significantly greater difficulty of reaching an optimum in the presence of noisy function measurements. The greater distance between $\boldsymbol{\theta}^*$ and the terminal $\hat{\boldsymbol{\theta}}_k$ is also apparent. Based on the standard distance measure (Euclidean norm $\|\cdot\|$), the mean of the distances between $\boldsymbol{\theta}^*$ and the terminal iterate $\hat{\boldsymbol{\theta}}_k$ for 2000 measurements is 11 times greater (0.93 versus 0.084) in the noisy case than the noise-free case. These comparisons become even more dramatic if the coefficient ρ in algorithm B is tuned for noise-free measurements. ❑

2.4 NONLINEAR SIMPLEX (NELDER–MEAD) ALGORITHM

2.4.1 Basic Method

Another popular direct search method is the nonlinear simplex algorithm of Nelder and Mead (1965). This algorithm for nonlinear optimization is based on the concept of a simplex, a geometric object that is the *convex hull* of $p + 1$ points in $\mathbb{R}^p$ not lying in the same hyperplane. (The convex hull is the smallest set enclosing the $p + 1$ points such that a line segment connecting any two points in the set lies in the set.) Note that the nonlinear simplex algorithm is not directly related to the popular simplex method for linear programming. With $p = 2$, a simplex is a triangle; with $p = 3$, a simplex is a pyramid.

The nonlinear simplex algorithm is the baseline gradient-free multivariate optimization technique in MATLAB (version 6) (the function **`fminsearch`**) and has arguably been called the most popular of the optimization techniques based on comparing loss function values only (Barton and Ivey, 1996). Based on extensive practical experience, the method often produces significant decreases in the loss function in the first few iterations. Interestingly, however, despite its popularity and practical effectiveness in a range of problems, there is no *general* convergence theory in either the deterministic or stochastic case (Barton and Ivey, 1996). In fact, many demonstrations of nonconvergence for specific functions or classes of functions have been given (e.g., Lagarias et al., 1998; McKinnon, 1998; Kelly, 1999b).

Although the nonlinear simplex algorithm was proposed in the Nelder and Mead (1965) paper as a deterministic method, it has frequently been used in a stochastic setting with noisy loss function measurements. Further, the algorithm is built on the simplex ideas in Spendley et al. (1962), which *do* explicitly account for noise in the function measurements.

The basic idea in the nonlinear simplex algorithm is that at each iteration a new point is generated in or near the current simplex. Usually, this new point replaces one of the current simplex vertices, yielding a new simplex. To determine this new point, a reflection step is introduced where the new point is chosen as the reflection (across the hyperplane spanned by the other vertices) of the vertex that currently has the worst (highest) value of $L(\boldsymbol{\theta})$. The aim is to

move the simplex in a direction away from the high values of the loss function and toward the lowest values of the loss. This process is continued until the size of the simplex is sufficiently small, in which case the solution is taken as a point inside the simplex.

The basic nonlinear simplex steps are given below for unconstrained optimization ($\Theta = \mathbb{R}^p$). Figure 2.3 illustrates these steps for the $p = 2$ case. If the problem involves explicit constraints, then the steps below can be modified so that any vertex outside of Θ is moved to the nearest point in Θ.

The algorithm steps below are identical to the steps in the MATLAB **fminsearch** routine (also, e.g., Kelly, 1999b). These steps differ from the original Nelder–Mead simplex algorithm in two small ways: (i) Certain strict inequalities here ($<$ or $>$) are nonstrict inequalities ($\leq$ or $\geq$) in the original algorithm (or vice versa) and (ii) the condition $L(\boldsymbol{\theta}_{exp}) < L(\boldsymbol{\theta}_{refl})$ in step 2b here is $L(\boldsymbol{\theta}_{exp}) < L(\boldsymbol{\theta}_{min})$ in the original algorithm. These changes from the original algorithm have become standard practice (Lagarias et al., 1998). Modification (i) is useful in tie breaking, while modification (ii) has been adopted based on accumulated experience of generally better performance. To avoid excessive subscript notation, the iteration counter k is suppressed in the algorithm presentation below. Note that one or more function evaluations may be required in each iteration.

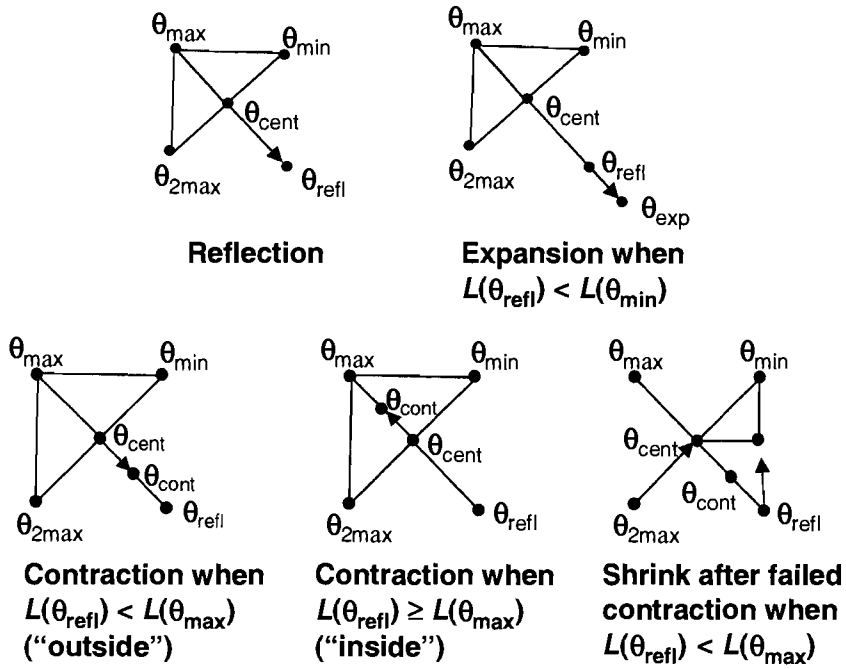

Figure 2.3. Nonlinear simplex algorithm for $p = 2$ with noise-free loss measurements (adapted from Barton and Ivey, 1996).

While the steps below are given for noise-free loss measurements, the same basic algorithm is often used with noisy measurements. Then, $y(\boldsymbol{\theta})$ values replace the indicated $L(\boldsymbol{\theta})$ evaluations. In Subsection 2.4.2, we comment on some fine tuning of these steps to accommodate noisy measurements of the loss function.

Nonlinear Simplex (Nelder–Mead) Algorithm (Unconstrained)

Step 0 **(Initialization)** Generate an initial set of $p + 1$ extreme points in $\mathbb{R}^p$, say $\boldsymbol{\theta}^i$ ($i = 1, 2, \ldots, p + 1$), representing the vertices of the initial simplex. Calculate $L(\boldsymbol{\theta}^i)$ for each extreme point. Set values for the algorithm coefficients α, β, γ, and δ. Default settings are specified by Nelder and Mead (1965) as 1.0, 0.5, 2.0, and 0.5, respectively (also used in MATLAB **`fminsearch`**).

Step 1 **(Reflection)** Identify the vertices where the maximum, second highest, and minimum L values occur. Let $\boldsymbol{\theta}_{\max}$, $\boldsymbol{\theta}_{2\max}$, and $\boldsymbol{\theta}_{\min}$ represent these points, respectively. Let $\boldsymbol{\theta}_{\text{cent}}$ represent the centroid (mean) of all $\boldsymbol{\theta}^i$ except for $\boldsymbol{\theta}_{\max}$. Generate a new candidate vertex $\boldsymbol{\theta}_{\text{refl}}$ by reflecting $\boldsymbol{\theta}_{\max}$ through $\boldsymbol{\theta}_{\text{cent}}$ according to $\boldsymbol{\theta}_{\text{refl}} = (1 + \alpha)\boldsymbol{\theta}_{\text{cent}} - \alpha\boldsymbol{\theta}_{\max}$ ($\alpha > 0$).

Step 2a **(Accept reflection)** If $L(\boldsymbol{\theta}_{\min}) \leq L(\boldsymbol{\theta}_{\text{refl}}) < L(\boldsymbol{\theta}_{2\max})$, then $\boldsymbol{\theta}_{\text{refl}}$ replaces $\boldsymbol{\theta}_{\max}$ in the simplex and proceed to step 3; else, go to step 2b.

Step 2b **(Expansion)** If $L(\boldsymbol{\theta}_{\text{refl}}) < L(\boldsymbol{\theta}_{\min})$, then the reflection is expanded according to $\boldsymbol{\theta}_{\text{exp}} = \gamma\boldsymbol{\theta}_{\text{refl}} + (1 - \gamma)\boldsymbol{\theta}_{\text{cent}}$, where the expansion coefficient $\gamma > 1$; else, go to step 2c. If $L(\boldsymbol{\theta}_{\text{exp}}) < L(\boldsymbol{\theta}_{\text{refl}})$, then $\boldsymbol{\theta}_{\text{exp}}$ replaces $\boldsymbol{\theta}_{\max}$ in the simplex; otherwise, the expansion is rejected and $\boldsymbol{\theta}_{\text{refl}}$ replaces $\boldsymbol{\theta}_{\max}$. Go to step 3.

Step 2c **(Contraction)** If $L(\boldsymbol{\theta}_{\text{refl}}) \geq L(\boldsymbol{\theta}_{2\max})$, then the simplex contracts to reflect the poor $\boldsymbol{\theta}_{\text{refl}}$. Consider two cases: (i) $L(\boldsymbol{\theta}_{2\max}) \leq L(\boldsymbol{\theta}_{\text{refl}}) < L(\boldsymbol{\theta}_{\max})$ (sometimes called *outside contraction*) or (ii) $L(\boldsymbol{\theta}_{\text{refl}}) \geq L(\boldsymbol{\theta}_{\max})$ (*inside contraction*). The contraction point is determined by $\boldsymbol{\theta}_{\text{cont}} = \beta\boldsymbol{\theta}_{\max/\text{refl}} + (1 - \beta)\boldsymbol{\theta}_{\text{cent}}$, $0 \leq \beta \leq 1$, where $\boldsymbol{\theta}_{\max/\text{refl}} = \boldsymbol{\theta}_{\text{refl}}$ in case (i) or $\boldsymbol{\theta}_{\max/\text{refl}} = \boldsymbol{\theta}_{\max}$ in case (ii). In case (i), if $L(\boldsymbol{\theta}_{\text{cont}}) \leq L(\boldsymbol{\theta}_{\text{refl}})$, the contraction is accepted. In case (ii), if $L(\boldsymbol{\theta}_{\text{cont}}) < L(\boldsymbol{\theta}_{\max})$, the contraction is accepted. If the contraction is accepted, replace $\boldsymbol{\theta}_{\max}$ with $\boldsymbol{\theta}_{\text{cont}}$ and go to step 3. If the contraction is not accepted, go to step 2d.

Step 2d **(Shrink)** The contraction has failed and the entire simplex shrinks according to a factor of $0 < \delta < 1$, retaining only $\boldsymbol{\theta}_{\min}$. This is done by replacing each vertex $\boldsymbol{\theta}^i$ (except $\boldsymbol{\theta}_{\min}$) by $\delta\boldsymbol{\theta}^i + (1 - \delta)\boldsymbol{\theta}_{\min}$. Go to step 3.

Step 3 **(Termination)** Stop if a convergence criterion is satisfied or if the maximum number of function evaluations has been reached; else, return to step 1.

2.4.2 Adaptation for Noisy Loss Measurements

It is sometimes acceptable to replace the exact L evaluations shown in the algorithm steps above with measurements of the loss function that include noise (so values of $y(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \varepsilon(\boldsymbol{\theta})$ replace the indicated $L(\boldsymbol{\theta})$ values). As discussed in Barton and Ivey (1996), the justification for making this direct substitution of possibly noisy function evaluations is the relative insensitivity of the algorithm to noise by virtue of its using only *ranks* of the loss function values, not the function values themselves (reminiscent of the ordinal versus cardinal discussion of Subsection 1.2.1). Small noise effects have little impact on ranks and therefore have little effect on the path of the algorithm. On the other hand, large noise effects *will* frequently change the relative ranks of the loss values, leading to changes in the algorithm convergence properties. In such a case, the nonlinear simplex algorithm may perform poorly, and one may do better with one of the stochastic approximation algorithms in Chapters 4–7, which are explicitly designed to cope with noise.

Several authors have provided modifications of the nonlinear simplex algorithm to adapt to noisy loss measurements. The basic structure of the algorithm remains the same. The modifications tend to be small—but crucial—refinements. The most obvious modification is to average several loss measurements at a given $\boldsymbol{\theta}$, as discussed in Section 2.3 for the random search methods. If there is an adequate amount of averaging given the level of noise, the algorithm described above can be used verbatim with the averaged loss values representing the loss measurements. Another change involves the choice of the convergence criterion. (This, of course, is irrelevant if one simply uses a "budget" of loss evaluations as discussed in Subsection 1.2.1.) A common criterion for the deterministic nonlinear simplex algorithm is to stop when

$$\left\{ \sum_{i=1}^{p+1} \left[L(\boldsymbol{\theta}^i) - L(\boldsymbol{\theta}_{\text{cent}}) \right]^2 \right\}^{1/2}$$

is sufficiently small.

The above criterion may be inappropriate with noisy loss values. For convergence, the differences in the loss values in the simplex are supposed to be small. The algorithm may never converge in practice because the noises may dominate the underlying differences in the true (noise-less) loss values. Hence, a criterion based on simplex size may be more appropriate. Dennis and Woods (1987) suggest the stopping criterion

$$\frac{\max_i \left\| \boldsymbol{\theta}^i - \boldsymbol{\theta}_{\min} \right\|}{\max\left\{1, \left\| \boldsymbol{\theta}_{\min} \right\|\right\}},$$

where the maximization in the numerator is over all extreme points in the simplex (the denominator is simply a normalization quantity). Because one

always has exact knowledge of the θ values, this criterion is less affected by noise than criteria involving loss measurements.

Noise introduces an unwelcome feedback process when the simplex is small. Recall that the noises tend to dominate the differences in loss values at the extreme points. Tomick et al. (1995) and Barton and Ivey (1996) show that noise can lead to premature reductions of simplex size due to an increase in the probabilities of invoking the contraction or shrink steps of the algorithm (steps 2c or 2d). Certainly, *some* tendency to reduce the size of the simplex is desirable to enhance the likelihood of the algorithm clustering around θ^*, but an excessive shrinkage tendency leads to the algorithm stopping before it has approached θ^*.

Ernst (1968), Deming and Parker (1978), Tomick et al. (1995), Barton and Ivey (1996), and Kelly (1999b) are among the references that present methods for coping with this premature convergence. Probably the easiest means by which this tendency can be arrested with noisy loss measurements is to alter the algorithm coefficients. Barton and Ivey (1996) recommend an increase of δ from the nominal value of 0.5 above to 0.9; this will cause a shrinkage of only 10 percent instead of the standard 50 percent.

The above modifications once again illustrate the important role in stochastic optimization of the user-specified algorithm coefficients. Two other possible modifications suggested by Barton and Ivey (1996) pertain to resampling loss measurements with the aim of providing stronger statistical evidence of the need for a reduction in the simplex size; this resampling involves discarding the previous value at a given θ (vs. simply averaging multiple values at a given θ). Barton and Ivey (1996) provide statistical evidence over a range of 18 test problems that the increase in δ leads to improved behavior (also see Example 2.9). The evidence for their other recommended changes is more mixed. In any specific application, it is likely that tuning the coefficients α, β, γ, and δ would lead to improved behavior in the presence of noise (and perhaps even without noise). Unfortunately, this is sometimes difficult, especially if the loss measurements are expensive.

Example 2.9—Nonlinear simplex algorithm applied to Rosenbrock function with and without noise. Using the **`fminsearch`** function in MATLAB (version 6), this study evaluates the nonlinear simplex method in a problem with the Rosenbrock function in (2.5) with $p = 10$. The evaluation considers noise-free and noisy function measurements. We also compare the performance with random search algorithm B. In the noisy case, $y(\theta) = L(\theta) + \varepsilon$, where ε represents an i.i.d. $N(0, \sigma^2)$ noise term. As in Example 2.5, let $\hat{\theta}_0 = [-1.2, 1, -1.2, 1, \ldots, 1]^T$. Suppose that 4000 measurements of the loss function are used in each replication and that the problem is unconstrained (i.e., $\Theta = \mathbb{R}^{10}$).

Two sets of algorithm coefficient values are used for the simplex method. The first is the default set, as given in step 0 above, and the second follows the recommendation above for coping with noisy loss measurements (i.e., δ is manually changed from 0.5 to 0.9 in **`fminsearch`** while other coefficients

remain at the default levels). The algorithm B results are based on perturbations $\boldsymbol{d}_k \sim N(\mathbf{0}, 0.1^2\boldsymbol{I}_{10})$. Algorithm B is implemented in a "baseline" form as given in the steps in Subsection 2.2.2 (i.e., y values are directly substituted for L values). Improved results are possible in the noisy case by using "thresholding" (modified step 2 in (2.6)); see comments below.

For values of $\sigma = 0$ and $\sigma = 5$, Table 2.7 shows the mean terminal loss values together with the approximate 95 percent confidence intervals computed using a t-distribution as in Example 2.3. The confidence intervals are computed from 40 independent runs, each using the 4000 measurements mentioned above.

For this problem, the simplex method outperforms algorithm B in the noise-free case, while the converse is true in the noisy case. Note, however, that the change in δ produces a significant improvement in the simplex method (nonoverlapping confidence intervals), as anticipated by Barton and Ivey (1996). (Although not shown in Table 2.7, thresholding likewise improves the performance of algorithm B by dropping the sample mean loss value to below 20 when $\sigma = 5$.) While the results of this study do not necessarily transfer to other noisy problems, the relatively poor performance of an "industry standard" search method such as **`fminsearch`** serves as a cautionary demonstration of the effects of noise. ❑

2.5 CONCLUDING REMARKS

The focus in this chapter has been some of the popular search methods for optimization based only on direct measurements of the loss function. We considered two broad types of such algorithms. The first is random search, based on searching through the domain Θ using randomly generated steps, usually generated in a Monte Carlo fashion via a pseudorandom number generator. The second is a geometrically motivated search based on the idea of a simplex. The specific nonlinear simplex algorithm emphasized here is sometimes called the

Table 2.7. Sample means and approximate 95 percent confidence intervals for terminal Rosenbrock loss values from the nonlinear simplex algorithm and algorithm B. Simplex method is implemented with default parameter settings and with δ changed. (For comparison purposes, note that $L(\hat{\boldsymbol{\theta}}_k) = 121$ and $L(\boldsymbol{\theta}^*) = 0$).

Noise level	Simplex (Default)	Simplex ($\delta = 0.90$)	Algorithm B
$\sigma = 0$	10.11 (no interval)	10.11 (no interval)	17.97 [16.90, 19.04]
$\sigma = 5$	28.31 [26.80, 29.82]	25.28 [24.41, 26.15]	22.95 [22.03, 23.86]

Nelder–Mead algorithm. The simplex approach is based on moving a shrinking region (a simplex) through Θ. When the algorithm works properly, the simplex collapses onto $\boldsymbol{\theta}^*$. As a testament to the popularity of this algorithm, the baseline gradient-free multivariate optimization technique in MATLAB (version 6) (the function **fminsearch**) is an implementation of the nonlinear simplex algorithm. This method is not to be confused with the simplex method of linear programming.

The methods of this chapter are versatile and broadly applicable. Because they require only loss measurements, they are often easier to implement than other methods that require relatively detailed information about the loss function (e.g., gradient information). In particular, a user might find the localized random search algorithm B to be useful in many problems, especially if only modest precision is required in the solution. This algorithm is relatively easy to implement and has a long record of reasonable practical efficiency.

In their standard forms, the algorithms of this chapter are used with noise-free measurements of the loss function. We also discussed some modifications to accommodate noisy measurements. Nonetheless, we saw, as predicted in Chapter 1, that noise can significantly degrade the results of the search process. The stochastic approximation methods in Chapters 4–7 are explicitly designed to cope with noise.

EXERCISES

2.1 Show that the conditions of Theorem 2.1 apply to $L(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{\theta}$ when using a uniform distribution for $\boldsymbol{\theta}_{\text{new}}$ on the domain $\Theta = [-1, 1] \times [0, 2]$. Given the knowledge of $\boldsymbol{\theta}^*$, present an explicit form for the lower bound $\delta(\eta)$ that appears in (2.2).

2.2 Let $\boldsymbol{\theta} = [a, b, c]^T$ and suppose that

$$L(\boldsymbol{\theta}) = -\left\{ \det \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix} \right\}^2 .$$

(a) Write $L(\boldsymbol{\theta})$ as an algebraic function of a, b, c.

(b) Using algorithm A, search for $\boldsymbol{\theta}^*$ from an initial condition $\hat{\boldsymbol{\theta}}_0 = [0, 0, 0]^T$ subject to $\Theta = [-1, 1]^3$ (i.e., each of a, b, c are between -1 and 1). For one realization of the search process, show how the estimates of $\boldsymbol{\theta}$ and the corresponding values of $L(\boldsymbol{\theta})$ change as the number of loss evaluations range over 100, 1000, 10,000, 100,000, and 1,000,000.

(c) Comment on the nonuniqueness of the solution $\boldsymbol{\theta}^*$ and its observed or likely effect on the search process. (The general form of $L(\boldsymbol{\theta})$ in this

problem arises in the experimental design discussion of Chapter 17; see, in particular, Example 17.6 and some of the accompanying exercises.)

2.3 Formula (2.4) may be numerically unreliable when P^* is very small. A more numerically stable form that is accurate for small P^* is $n \approx -\log \rho / P^*$. Derive this approximation and compare its accuracy with the exact values from (2.4) for the problem of Example 2.2 with the corresponding values of $1 \le p \le 10$ in Table 2.2. Show three significant digits of accuracy for both the exact and approximate values of n (this is greater accuracy than shown in Table 2.2).

2.4 Suppose that $p = 3$ and $\Theta = \{\boldsymbol{\theta}: \boldsymbol{\theta}^T\boldsymbol{\theta} \le 1\}$. Consider algorithm A where uniform sampling on a superset of Θ, $\Theta' = [-1, 1]^3$, is used (with the sample rejected if it lies outside of Θ) to generate each of the candidate $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{new}(k)$ values. What is the expected number of sample values of $\boldsymbol{\theta} \in \Theta'$ needed to produce the $\boldsymbol{\theta}_{new} \in \Theta$ values used in 100 iterations?

2.5 Sketch or describe a loss function and domain Θ where condition (2.1) in Theorem 2.1 does *not* hold, but where Θ^* still has only one (unique) element $\boldsymbol{\theta}^*$. Discuss or numerically demonstrate how algorithm A may not converge to this unique $\boldsymbol{\theta}^*$ for this problem.

2.6 Let $\Theta = [-1, 1]^p$ and suppose that uniform sampling on Θ is used to generate $\boldsymbol{\theta}_{new}(k)$ for all k. We want to guarantee with probability 0.90 that the first $p/2$ elements of $\boldsymbol{\theta}$ are within 0.04 units of the optimal and the next $p/2$ elements are within 0.10 units (p an even number). Let the (unknown) $\boldsymbol{\theta}^*$ lie in $[-0.90, 0.90]^p$. How many loss measurements in algorithm A are needed if $p = 2$? $p = 10$?

2.7 Consider a modified form of algorithm A where the sampling in step 1 is replaced with sampling over a domain that changes with k. In particular, suppose that $\boldsymbol{\theta}_{new}(k+1)$ is generated uniformly on a sampling domain $\Theta_{k+1} \equiv \Theta_k \cap \{\boldsymbol{\theta}: L(\boldsymbol{\theta}) < L(\hat{\boldsymbol{\theta}}_k)\}$, where Θ_0 is the problem domain Θ (so $\Theta_{k+1} \subseteq \Theta_k \subseteq \ldots \subseteq \Theta_0 = \Theta$). Let V_k denote the volume of Θ_k. Show that the expected volume is reduced by half at every iteration. That is, show that $E(V_k/V_{k-1}) = 1/2$. (Hint: First show that $P(V_k/V_{k-1} \le r) = r$ for all $r \in [0, 1]$. Note that this algorithm is an idealized adaptive search algorithm that is not usually implementable due to the unknown shape of the Θ_k domains; the reduction of candidate volume by half at every iteration is very fast.)

2.8 By any numerical or analytical means, identify the minima, maxima, and saddlepoints of the loss function in Example 2.3.

2.9 Consider the function $L(\boldsymbol{\theta}) = \sum_{i=1}^{p/2} t_i^4 + \boldsymbol{\theta}^T\boldsymbol{B}\boldsymbol{\theta}$, where p is an even number, $\boldsymbol{\theta} = [t_1, t_2, \ldots, t_p]^T$, and $\boldsymbol{B}$ is a symmetric matrix with 1's on the diagonals and 0.5's elsewhere. Apply algorithm B with $\boldsymbol{d}_k \sim N(\boldsymbol{0}, \rho^2\boldsymbol{I}_p)$, $\Theta = \mathbb{R}^p$, and $\hat{\boldsymbol{\theta}}_0 = [1, 1, \ldots, 1]^T$. Use 100 noise-free loss measurements per replication as follows:

(a) Using $p = 20$, compute the mean terminal loss values based on 40 replications for $\rho = 0.125, 0.25, 0.5$, and 1.

(b) Using $p = 2$, compute the mean terminal loss values based on 40 replications and the same four values of ρ as in part (a).

(c) Normalize the mean loss values in part (a) and the mean loss values in part (b) by their respective $L(\hat{\boldsymbol{\theta}}_0)$ (i.e., divide the mean loss values by $L(\hat{\boldsymbol{\theta}}_0)$, which is the only normalization required since $L(\boldsymbol{\theta}^*) = 0$). How do the normalized loss values with $p = 20$ compare with the normalized values for $p = 2$? Comment on the effects of increased dimension.

2.10 From the information provided in Example 2.3, determine the approximate sample standard deviations (s) for the terminal loss values from algorithms A, B, and C, and compute 90 percent confidence intervals for the mean values shown in the example. Show that there is still some overlap in these intervals.

2.11 If the problem setting allowed it, determine on a conceptual basis which, if any, of the algorithms A, B, or C would potentially benefit by allowing sampling over a region Θ' that contains Θ in the sense that Θ is a strict subset of Θ' (i.e., $\Theta \subset \Theta'$). Here, *benefit* refers to greater accuracy of the $\boldsymbol{\theta}$ estimate or a lower final value of the loss function for a given number of loss evaluations. If any algorithm shows a potential benefit, give a conceptual or numerical example where the benefit is realized (if numerical, do repeated runs until there is strong statistical evidence of a benefit).

2.12 In contrast to general results with noisy loss measurements, provide a conceptual example (with justification) of a problem with noise where algorithm A is guaranteed to converge. (Hint: Discrete or continuous $\boldsymbol{\theta}$ may be considered.)

2.13 Suppose that only noisy measurements of the loss function $L(\boldsymbol{\theta}) = t_1^4 + t_1^2 + t_1 t_2 + t_2^2$ (introduced in Subsection 1.4.1) are available ($\boldsymbol{\theta} = [t_1, t_2]^T$). The noise is additive independent $N(0, \sigma^2)$. Suppose that algorithm B is going to be used with $\hat{\boldsymbol{\theta}}_0 = [1, 1]^T$, $\boldsymbol{d}_k \sim N(\boldsymbol{0}, 0.125^2 \boldsymbol{I}_2)$ for all k, and $\Theta = \mathbb{R}^2$. Carry out numerical analysis to determine whether the loss averaging or altered threshold (2.6) should be used to cope with the noise. In particular:

(a) For $\sigma = 0.001$, 0.01, 0.1, and 1.0 with 10,000 loss measurements, determine approximately optimal values for the amount of averaging and for the altered threshold. For each of these values compute the sample mean of the terminal loss value in 40 independent runs. Which method (averaging or altered threshold) seems to work better for this problem? (You may base your analysis on only the sample means—it is not necessary to do a formal statistical analysis.)

(b) Compute the sample mean of the terminal loss value in the noise-free case (40 runs). Show that this is significantly lower than the best of the values in part (a) for the smallest noise level.

2.14 Consider the wastewater treatment problem in Example 2.7. For the initial condition, noise levels, and domain Θ in the example, use algorithms A and C to produce tables analogous to Table 2.6 in Example 2.7 (choose ρ appropriately for algorithm C). In particular, calculate the means of 50 terminal loss values for 100 and 2000 measurements in the algorithms, including 95 percent confidence intervals, for algorithms A and C. For each algorithm, use the averaged and unaveraged implementations discussed in Example 2.7 (so the table you produce should have eight numerical entries—

averaged and unaveraged implementations for the two algorithms for the two sample sizes). Comment on how the results of these two algorithms compare with those of algorithm B in Example 2.7.

2.15 Using the $p = 20$ version of the loss function in Exercise 2.9 together with $\hat{\theta}_0 = [1, 1, \ldots, 1]^T$ and 1000 loss measurements:

(a) Implement the nonlinear simplex algorithm using the default coefficient settings and noise-free measurements (you may use **`fminsearch`**).

(b) Repeat part (a) with the exception of having the algorithm use loss measurements with additive, independent $N(0, 0.2^2)$ noise. Generate 40 replications and report the sample mean and approximate 95 percent confidence interval for the terminal loss value.

(c) Repeat part (b) with the exception of setting $\delta = 0.9$. Contrast the results of parts (b) and (c).

2.16 For the nonlinear simplex algorithm, elaborate on why noise causes an increased tendency to reduce the size of the simplex relative to a deterministic case with the same values of algorithm coefficients α, β, γ, and δ.