CHAPTER 1

Introduction to Optimization

Optimization is the process of making something better. An engineer or scientist conjures up a new idea and optimization improves on that idea. Optimization consists in trying variations on an initial concept and using the information gained to improve on the idea. A computer is the perfect tool for optimization as long as the idea or variable influencing the idea can be input in electronic format. Feed the computer some data and out comes the solution. Is this the only solution? Often times not. Is it the best solution? That's a tough question. Optimization is the math tool that we rely on to get these answers.

This chapter begins with an elementary explanation of optimization, then moves on to a historical development of minimum-seeking algorithms. A seemingly simple example reveals many shortfalls of the typical minimum seekers. Since the local optimizers of the past are limited, people have turned to more global methods based upon biological processes. The chapter ends with some background on biological genetics and a brief introduction to the genetic algorithm (GA).

1.1 FINDING THE BEST SOLUTION

The terminology "best" solution implies that there is more than one solution and the solutions are not of equal value. The definition of best is relative to the problem at hand, its method of solution, and the tolerances allowed. Thus the optimal solution depends on the person formulating the problem. Education, opinions, bribes, and amount of sleep are factors influencing the definition of best. Some problems have exact answers or roots, and best has a specific definition. Examples include best home run hitter in baseball and a solution to a linear first-order differential equation. Other problems have various minimum or maximum solutions known as optimal points or extrema, and best may be a relative definition. Examples include best piece of artwork or best musical composition.

Practical Genetic Algorithms, Second Edition, by Randy L. Haupt and Sue Ellen Haupt.
ISBN 0-471-45565 2

1.1.1 What Is Optimization?

Our lives confront us with many opportunities for optimization. What time do we get up in the morning so that we maximize the amount of sleep yet still make it to work on time? What is the best route to work? Which project do we tackle first? When designing something, we shorten the length of this or reduce the weight of that, as we want to minimize the cost or maximize the appeal of a product. Optimization is the process of adjusting the inputs to or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result (Figure 1.1). The input consists of variables; the process or function is known as the cost function, objective function, or fitness function; and the output is the cost or fitness. If the process is an experiment, then the variables are physical inputs to the experiment.

For most of the examples in this book, we define the output from the process or function as the cost. Since cost is something to be minimized, optimization becomes minimization. Sometimes maximizing a function makes more sense. To maximize a function, just slap a minus sign on the front of the output and minimize. As an example, maximizing $1 - x^2$ over $-1 \le x \le 1$ is the same as minimizing $x^2 - 1$ over the same interval. Consequently in this book we address the maximization of some function as a minimization problem.

Life is interesting due to the many decisions and seemingly random events that take place. Quantum theory suggests there are an infinite number of dimensions, and each dimension corresponds to a decision made. Life is also highly nonlinear, so chaos plays an important role too. A small perturbation in the initial condition may result in a very different and unpredictable solution. These theories suggest a high degree of complexity encountered when studying nature or designing products. Science developed simple models to represent certain limited aspects of nature. Most of these simple (and usually linear) models have been optimized. In the future, scientists and engineers must tackle the unsolvable problems of the past, and optimization is a primary tool needed in the intellectual toolbox.

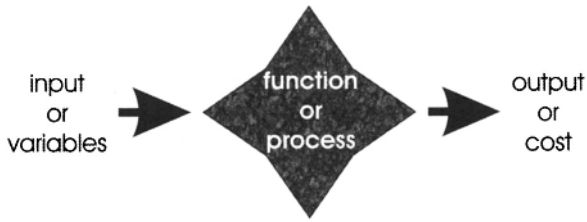

Figure 1.1 Diagram of a function or process that is to be optimized. Optimization varies the input to achieve a desired output.

1.1.2 Root Finding versus Optimization

Approaches to optimization are akin to root or zero finding methods, only harder. Bracketing the root or optimum is a major step in hunting it down. For the one-variable case, finding one positive point and one negative point brackets the zero. On the other hand, bracketing a minimum requires three points, with the middle point having a lower value than either end point. In the mathematical approach, root finding searches for zeros of a function, while optimization finds zeros of the function derivative. Finding the function derivative adds one more step to the optimization process. Many times the derivative does not exist or is very difficult to find. We like the simplicity of root finding problems, so we teach root finding techniques to students of engineering, math, and science courses. Many technical problems are formulated to find roots when they might be more naturally posed as optimization problems; except the toolbox containing the optimization tools is small and inadequate.

Another difficulty with optimization is determining if a given minimum is the best (global) minimum or a suboptimal (local) minimum. Root finding doesn't have this difficulty. One root is as good as another, since all roots force the function to zero.

Finding the minimum of a nonlinear function is especially difficult. Typical approaches to highly nonlinear problems involve either linearizing the problem in a very confined region or restricting the optimization to a small region. In short, we cheat.

1.1.3 Categories of Optimization

Figure 1.2 divides optimization algorithms into six categories. None of these six views or their branches are necessarily mutually exclusive. For instance, a dynamic optimization problem could be either constrained or

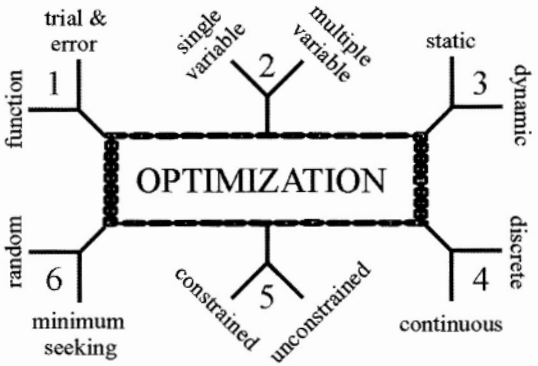

Figure 1.2 Six categories of optimization algorithms.

unconstrained. In addition some of the variables may be discrete and others continuous. Let's begin at the top left of Figure 1.2 and work our way around clockwise.

1. Trial-and-error optimization refers to the process of adjusting variables that affect the output without knowing much about the process that produces the output. A simple example is adjusting the rabbit ears on a TV to get the best picture and audio reception. An antenna engineer can only guess at why certain contortions of the rabbit ears result in a better picture than other contortions. Experimentalists prefer this approach. Many great discoveries, like the discovery and refinement of penicillin as an antibiotic, resulted from the trial-and-error approach to optimization. In contrast, a mathematical formula describes the objective function in function optimization. Various mathematical manipulations of the function lead to the optimal solution. Theoreticians love this theoretical approach.

2. If there is only one variable, the optimization is one-dimensional. A problem having more than one variable requires multidimensional optimization. Optimization becomes increasingly difficult as the number of dimensions increases. Many multidimensional optimization approaches generalize to a series of one-dimensional approaches.

3. Dynamic optimization means that the output is a function of time, while static means that the output is independent of time. When living in the suburbs of Boston, there were several ways to drive back and forth to work. What was the best route? From a distance point of view, the problem is static, and the solution can be found using a map or the odometer of a car. In practice, this problem is not simple because of the myriad of variations in the routes. The shortest route isn't necessarily the fastest route. Finding the fastest route is a dynamic problem whose solution depends on the time of day, the weather, accidents, and so on. The static problem is difficult to solve for the best solution, but the added dimension of time increases the challenge of solving the dynamic problem.

4. Optimization can also be distinguished by either discrete or continuous variables. Discrete variables have only a finite number of possible values, whereas continuous variables have an infinite number of possible values. If we are deciding in what order to attack a series of tasks on a list, discrete optimization is employed. Discrete variable optimization is also known as combinatorial optimization, because the optimum solution consists of a certain combination of variables from the finite pool of all possible variables. However, if we are trying to find the minimum value of $f(x)$ on a number line, it is more appropriate to view the problem as continuous.

5. Variables often have limits or constraints. Constrained optimization incorporates variable equalities and inequalities into the cost function. Unconstrained optimization allows the variables to take any value. A constrained variable often converts into an unconstrained variable through a transforma-

tion of variables. Most numerical optimization routines work best with unconstrained variables. Consider the simple constrained example of minimizing $f(x)$ over the interval $-1 \le x \le 1$. The variable converts x into an unconstrained variable u by letting $x = \sin(u)$ and minimizing $f(\sin(u))$ for any value of u. When constrained optimization formulates variables in terms of linear equations and linear constraints, it is called a linear program. When the cost equations or constraints are nonlinear, the problem becomes a nonlinear programming problem.

6. Some algorithms try to minimize the cost by starting from an initial set of variable values. These minimum seekers easily get stuck in local minima but tend to be fast. They are the traditional optimization algorithms and are generally based on calculus methods. Moving from one variable set to another is based on some determinant sequence of steps. On the other hand, random methods use some probabilistic calculations to find variable sets. They tend to be slower but have greater success at finding the global minimum.

1.2 MINIMUM-SEEKING ALGORITHMS

Searching the cost surface (all possible function values) for the minimum cost lies at the heart of all optimization routines. Usually a cost surface has many peaks, valleys, and ridges. An optimization algorithm works much like a hiker trying to find the minimum altitude in Rocky Mountain National Park. Starting at some random location within the park, the goal is to intelligently proceed to find the minimum altitude. There are many ways to hike or glissade to the bottom from a single random point. Once the bottom is found, however, there is no guarantee that an even lower point doesn't lie over the next ridge. Certain constraints, such as cliffs and bears, influence the path of the search as well. Pure downhill approaches usually fail to find the global optimum unless the cost surface is quadratic (bowl-shaped).

There are many good texts that describe optimization methods (e.g., Press et al., 1992; Cuthbert, 1987). A history is given by Boyer and Merzbach (1991). Here we give a very brief review of the development of optimization strategies.

1.2.1 Exhaustive Search

The brute force approach to optimization looks at a sufficiently fine sampling of the cost function to find the global minimum. It is equivalent to spending the time, effort, and resources to thoroughly survey Rocky Mountain National Park. In effect a topographical map can be generated by connecting lines of equal elevation from an interpolation of the sampled points. This exhaustive search requires an extremely large number of cost function evaluations to find the optimum. For example, consider solving the two-dimensional problem

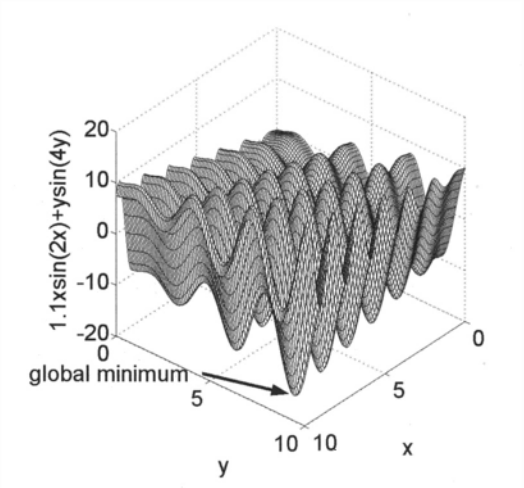

Figure 1.3 Three-dimensional plot of (1.1) in which x and y are sampled at intervals of 0.1.

$$\text{Find the minimum of:} \quad f(x,y) = x\sin(4x) + 1.1y\sin(2y) \tag{1.1}$$

$$\text{Subject to:} \quad 0 \le x \le 10 \quad \text{and} \quad 0 \le y \le 10 \tag{1.2}$$

Figure 1.3 shows a three-dimensional plot of (1.1) in which x and y are sampled at intervals of 0.1, requiring a total of 101^2 function evaluations. This same graph is shown as a contour plot with the global minimum of –18.5547 at (x,y) = (0.9039, 0.8668) marked by a large black dot in Figure 1.4. In this case the global minimum is easy to see. Graphs have aesthetic appeal but are only practical for one- and two-dimensional cost functions. Usually a list of function values is generated over the sampled variables, and then the list is searched for the minimum value. The exhaustive search does the surveying necessary to produce an accurate topographic map. This approach requires checking an extremely large but finite solution space with the number of combinations of different variable values given by

$$V = \prod_{i=1}^{N_{var}} Q_i \tag{1.3}$$

where

V = number of different variable combinations
N_{var} = total number of different variables
Q_i = number of different values that variable i can attain

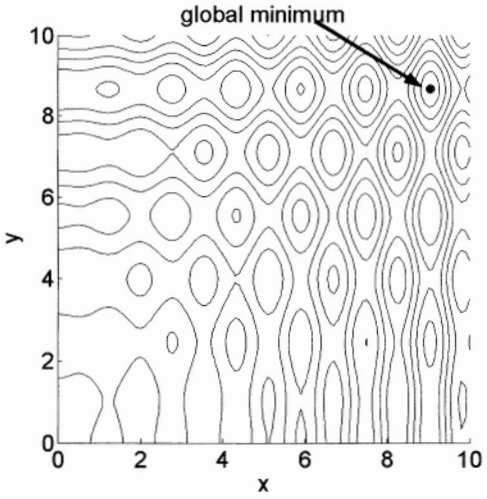

Figure 1.4 Contour plot of (1.1).

With fine enough sampling, exhaustive searches don't get stuck in local minima and work for either continuous or discontinuous variables. However, they take an extremely long time to find the global minimum. Another shortfall of this approach is that the global minimum may be missed due to undersampling. It is easy to undersample when the cost function takes a long time to calculate. Hence exhaustive searches are only practical for a small number of variables in a limited search space.

A possible refinement to the exhaustive search includes first searching a coarse sampling of the fitness function, then progressively narrowing the search to promising regions with a finer toothed comb. This approach is similar to first examining the terrain from a helicopter view, and then surveying the valleys but not the peaks and ridges. It speeds convergence and increases the number of variables that can be searched but also increases the odds of missing the global minimum. Most optimization algorithms employ a variation of this approach and start exploring a relatively large region of the cost surface (take big steps); then they contract the search around the best solutions (take smaller and smaller steps).

1.2.2 Analytical Optimization

Calculus provides the tools and elegance for finding the minimum of many cost functions. The thought process can be simplified to a single variable for a moment, and then an extremum is found by setting the first derivative of a cost function to zero and solving for the variable value. If the second derivative is greater than zero, the extremum is a minimum, and conversely, if the

second derivative is less than zero, the extremum is a maximum. One way to find the extrema of a function of two or more variables is to take the gradient of the function and set it equal to zero, $\nabla f(x, y) = 0$. For example, taking the gradient of equation (1.1) results in

$$\frac{\partial f}{\partial x} = \sin(4x_m) + 4x\cos(4x_m) = 0, \qquad 0 \le x \le 10 \tag{1.4a}$$

and

$$\frac{\partial f}{\partial y} = 1.1\sin(2y_m) + 2.2y_m\cos(2y_m) = 0, \qquad 0 \le y \le 10 \tag{1.4b}$$

Next these equations are solved for their roots, x_m and y_m, which is a family of lines. Extrema occur at the intersection of these lines. Note that these transcendental equations may not always be separable, making it very difficult to find the roots. Finally, the Laplacian of the function is calculated.

$$\frac{\partial^2 f}{\partial x^2} = 8\cos 4x - 16x\sin 4x, \qquad 0 \le x \le 10 \tag{1.5a}$$

and

$$\frac{\partial^2 f}{\partial y^2} = 4.4\cos 2y - 4.4y\sin 2y, \qquad 0 \le y \le 10 \tag{1.5b}$$

The roots are minima when $\nabla^2 f(x_m, y_m) > 0$. Unfortunately, this process doesn't give a clue as to which of the minima is a global minimum. Searching the list of minima for the global minimum makes the second step of finding $\nabla^2 f(x_m, y_m)$ redundant. Instead, $f(x_m, y_m)$ is evaluated at all the extrema; then the list of extrema is searched for the global minimum. This approach is mathematically elegant compared to the exhaustive or random searches. It quickly finds a single minimum but requires a search scheme to find the global minimum. Continuous functions with analytical derivatives are necessary (unless derivatives are taken numerically, which results in even more function evaluations plus a loss of accuracy). If there are too many variables, then it is difficult to find all the extrema. The gradient of the cost function serves as the compass heading pointing to the steepest downhill path. It works well when the minimum is nearby, but cannot deal well with cliffs or boundaries, where the gradient can't be calculated.

In the eighteenth century, Lagrange introduced a technique for incorporating the equality constraints into the cost function. The method, now known as Lagrange multipliers, finds the extrema of a function $f(x, y, \ldots)$ with constraints $g_m(x, y, \ldots) = 0$, by finding the extrema of the new function $F(x, y, \ldots, \kappa_1, \kappa_2, \ldots) = f(x, y, \ldots) + \Sigma_{m=}^{M} \kappa_m g_m(x, y, \ldots)$ (Borowski and Borwein, 1991).

Then, when gradients are taken in terms of the new variables κ_m, the constraints are automatically satisfied.

As an example of this technique, consider equation (1.1) with the constraint $x + y = 0$. The constraints are added to the cost function to produce the new cost function

$$f_\lambda = x\sin(4x) + 1.1y\sin(2y) + \kappa(x + y) \tag{1.6}$$

Taking the gradient of this function of three variables yields

$$\frac{\partial f}{\partial x} = \sin(4x_m) + 4x_m\cos(4x_m) + \kappa = 0$$

$$\frac{\partial f}{\partial y} = 1.1\sin(2y_m) + 2.2y_m\cos(2y_m) + \kappa = 0$$

$$\frac{\partial f}{\partial \kappa} = x_m + y_m = 0 \tag{1.7}$$

Subtracting the second equation from the first and employing $y_m = -x_m$ from the third equation gives

$$4x_m\cos(4x_m) - \sin(4x_m) + 1.1\sin(2x_m) - 2.2x_m\cos(2x_m) = 0 \tag{1.8}$$

where $(x_m, -x_m)$ are the minima of equation (1.6). The solution is once again a family of lines crossing the domain.

The many disadvantages to the calculus approach make it an unlikely candidate to solve most optimization problems encountered in the real world. Even though it is impractical, most numerical approaches are based on it. Typically an algorithm starts at some random point in the search space, calculates a gradient, and then heads downhill to the bottom. These numerical methods head downhill fast; however, they often find the wrong minimum (a local minimum rather than the global minimum) and don't work well with discrete variables. Gravity helps us find the downhill direction when hiking, but we will still most likely end up in a local valley in the complex terrain.

Calculus-based methods were the bag of tricks for optimization theory until von Neumann developed the minimax theorem in game theory (Thompson, 1992). Games require an optimum move strategy to guarantee winning. That same thought process forms the basis for more sophisticated optimization techniques. In addition techniques were needed to find the minimum of cost functions having no analytical gradients. Shortly before and during World War II, Kantorovich, von Neumann, and Leontief solved linear problems in the fields of transportation, game theory, and input-output models (Anderson, 1992). Linear programming concerns the minimization of a linear function of many variables subject to constraints that are linear equations and equalities. In 1947 Dantzig introduced the simplex method, which has been the work-

horse for solving linear programming problems (Williams, 1993). This method has been widely implemented in computer codes since the mid-1960s.

Another category of methods is based on integer programming, an extension of linear programming in which some of the variables can only take integer values (Williams, 1993). Nonlinear techniques were also under investigation during World War II. Karush extended Lagrange multipliers to constraints defined by equalities and inequalities, so a much larger category of problems could be solved. Kuhn and Tucker improved and popularized this technique in 1951 (Pierre, 1992). In the 1950s Newton's method and the method of steepest descent were commonly used.

1.2.3 Nelder-Mead Downhill Simplex Method

The development of computers spurred a flurry of activity in the 1960s. In 1965 Nelder and Mead introduced the downhill simplex method (Nelder and Mead, 1965), which doesn't require the calculation of derivatives. A simplex is the most elementary geometrical figure that can be formed in dimension N and has $N + 1$ sides (e.g., a triangle in two-dimensional space). The downhill simplex method starts at $N + 1$ points that form the initial simplex. Only one point of the simplex, P_0, is specified by the user. The other N points are found by

$$P_n = P_0 + c_s e_n \tag{1.9}$$

where e_n are N unit vectors and c_s is a scaling constant. The goal of this method is to move the simplex until it surrounds the minimum, and then to contract the simplex around the minimum until it is within an acceptable error. The steps used to trap the local minimum inside a small simplex are as follows:

1. Creation of the initial triangle. Three vertices start the algorithm: $A = (x_1,y_1)$, $B = (x_2,y_2)$, and $C = (x_3,y_3)$ as shown in Figure 1.5.
2. Reflection. A new point, $D = (x_4,y_4)$, is found as a reflection of the lowest minimum (in this case A) through the midpoint of the line connecting the other two points (B and C). As shown in Figure 1.5, D is found by

$$D = B + C - A \tag{1.10}$$

3. Expansion. If the cost of D is smaller than that at A, then the move was in the right direction and another step is made in that same direction as shown in Figure 1.5. The formula is given by

$$E = \frac{3(B+C)}{2} - 2A \tag{1.11}$$

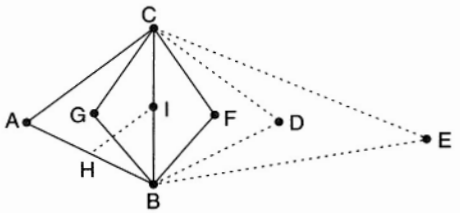

Figure 1.5 Manipulation of the basic simplex, in the case of two dimensions, a triangle in an effort to find the minimum.

4. Contraction. If the new point, D, has the same cost as point A, then two new points are found

$$F = \frac{2A + B + C}{4}$$

$$G = \frac{3(B + C)}{2 - 2A} \tag{1.12}$$

The smallest cost of F and G is kept, thus contracting the simplex as shown in Figure 1.5.

5. Shrinkage. If neither F nor G have smaller costs than A, then the side connecting A and C must move toward B in order to shrink the simplex. The new vertices are given by

$$H = \frac{A + B}{2}$$

$$I = \frac{B + C}{2} \tag{1.13}$$

Each iteration generates a new vertex for the simplex. If this new point is better than at least one of the existing vertices, it replaces the worst vertex. This way the diameter of the simplex gets smaller and the algorithm stops when the diameter reaches a specified tolerance. This algorithm is not known for its speed, but it has a certain robustness that makes it attractive. Figures 1.6 and 1.7 demonstrate the Nelder-Mead algorithm in action on a bowl-shaped surface. Note how the triangle gradually flops down the hill until it surrounds the bottom. The next step would be to shrink itself around the minimum.

Since the Nelder-Mead algorithm gets stuck in local minima, it can be combined with the random search algorithm to find the minimum to (1.1) subject to (1.2). Assuming that there is no prior knowledge of the cost surface, a random first guess is as good as any place to start. How close

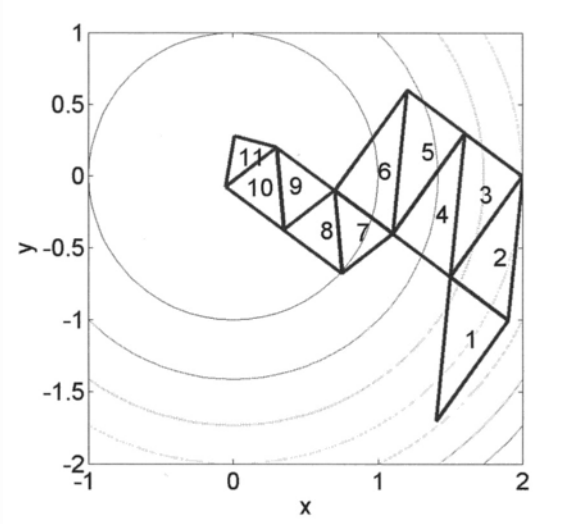

Figure 1.6 Contour plot of the movement of the simplex down hill to surround the minimum.

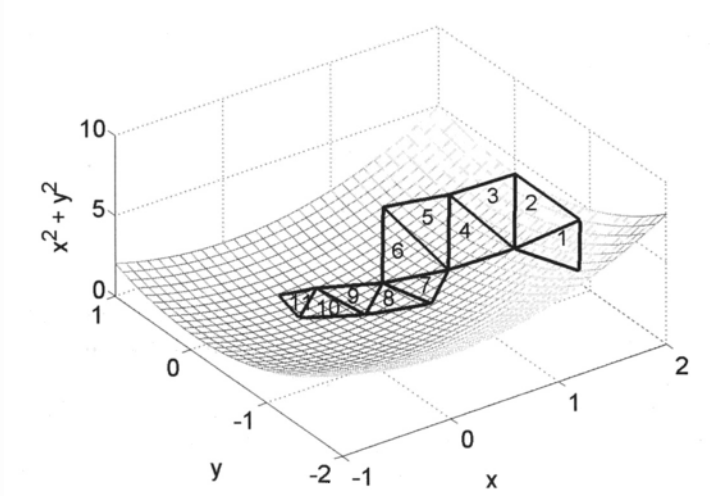

Figure 1.7 Mesh plot of the movement of the simplex down hill to surround the minimum.

TABLE 1.1 Comparison of Nelder-Meade and BFGS Algorithms

		Nelder-Mead			BFGS		
Starting Point		Ending Point			Ending Point		
x	y	x	y	cost	x	y	cost
8.9932	3.7830	9.0390	5.5427	−15.1079	9.0390	2.4567	−11.6835
3.4995	8.9932	5.9011	2.4566	−8.5437	5.9011	2.4566	−8.5437
0.4985	7.3803	1.2283	8.6682	10.7228	0.0000	8.6682	−9.5192
8.4066	5.9238	7.4696	5.5428	−13.5379	9.0390	5.5428	−15.1079
0.8113	6.3148	1.2283	5.5427	−7.2760	0.0000	5.5428	−6.0724
6.8915	1.8475	7.4696	2.4566	−10.1134	7.4696	2.4566	−10.1134
7.3021	9.5406	5.9011	8.6682	−15.4150	7.4696	8.6682	16.9847
5.6989	8.2893	5.9011	8.6682	−15.4150	5.9011	8.6682	16.9847
6.3245	3.2649	5.9011	2.4566	−8.5437	5.9011	2.4566	−8.5437
5.6989	4.6725	5.9011	5.5428	11.9682	5.9011	5.5428	−11.9682
4.0958	0.3226	4.3341	0.0000	−4.3269	4.3341	0.0000	−4.3269
4.2815	8.2111	4.3341	8.6622	−13.8461	4.3341	8.6622	−13.8461
Average				−11.2347			−11.1412

does this guess have to be to the true minimum before the algorithm can find it? Some simple experimentation helps us arrive at this answer. The first two columns in Table 1.1 show twelve random starting values. The ending values and the final costs found by the Nelder-Mead algorithm are found in the next three columns. None of the trials arrived at the global minimum.

Box (1965) extended the simplex method and called it the complex method, which stands for constrained simplex method. This approach allows the addition of inequality constraints, uses up to $2N$ vertices, and expands the polyhedron at each normal reflection.

1.2.4 Optimization Based on Line Minimization

The largest category of optimization methods fall under the general title of successive line minimization methods. An algorithm begins at some random point on the cost surface, chooses a direction to move, then moves in that direction until the cost function begins to increase. Next the procedure is repeated in another direction. Devising a sensible direction to move is critical to algorithm convergence and has spawned a variety of approaches.

A very simple approach to line minimization is the coordinate search method (Schwefel, 1995). It starts at an arbitrary point on the cost surface, then does a line minimization along the axis of one of the variables. Next it selects another variable and does another line minimization along that axis. This process continues until a line minimization is done along each of the vari-

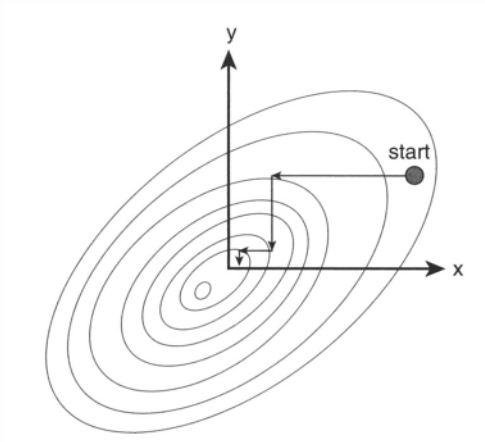

Figure 1.8 Possible path that the coordinate search method might take on a quadratic cost surface.

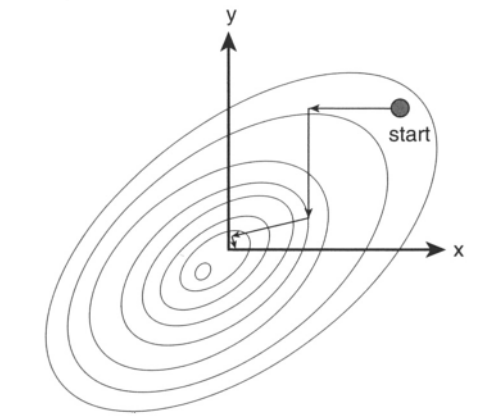

Figure 1.9 Possible path that the Rosenbrock method might take on a quadratic cost surface.

ables. Then the algorithm cycles through the variables until an acceptable solution is found. Figure 1.8 models a possible path the algorithm might take in a quadratic cost surface. In general, this method is slow.

Rosenbrock (1960) developed a method that does not limit search directions to be parallel to the variable axes. The first iteration of the Rosenbrock method uses coordinate search along each variable to find the first improved point (see Figure 1.9). The coordinate axes are then rotated until the first new coordinate axis points from the starting location to the first point. Gram-Schmidt orthogonalization finds the directions of the other new coordinate axes based on the first new coordinate axis. A coordinate search is then per-

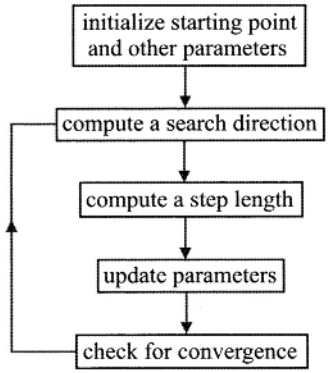

Figure 1.10 Flowchart for a typical line search algorithm.

formed along each new coordinate axis. As before, this process iterates until an acceptable solution is found.

A line search finds the optimum in one direction or dimension. For an n-dimensional objective function, the line search repeats for at least n iterations until the optimum is found. A flowchart of successive line search optimization appears in Figure 1.10. All the algorithms in this category differ in how the search direction at step n is found. For detailed information on the three methods described here, consult Luenberger (1984) and Press et al. (1992).

The steepest descent algorithm originated with Cauchy in 1847 and has been extremely popular. It starts at an arbitrary point on the cost surface and minimizes along the direction of the gradient. The simple formula for the $(n + 1)$th iteration is given by

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \gamma_n \nabla f(x_n, y_n) \tag{1.14}$$

where γ_n is a nonnegative scalar that minimizes the function in the direction of the gradient. By definition, the new gradient formed at each iteration is orthogonal to the previous gradient. If the valley is narrow (ratio of maximum to minimum eigenvalue large), then this algorithm bounces from side to side for many iterations before reaching the bottom. Figure 1.11 shows a possible path of the steepest descent algorithm. Note that the path is orthogonal to the contours and any path is orthogonal to the previous and next path.

The method of steepest descent is not normally used anymore, because more efficient techniques have been developed. Most of these techniques involve some form of Newton's method. Newton's method is based on a mul-

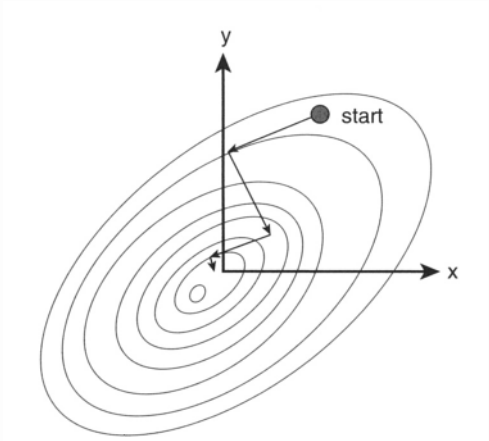

Figure 1.11 Possible path that the steepest descent algorithm might take on a quadratic cost surface.

tidimensional Taylor series expansion of the function about the point x_k given by

$$f(\boldsymbol{x}) = f(x_n) + \nabla f(x_n)(x - x_n)^T + \frac{(x - x_n)}{2!}\mathbf{H}(x - x_n)^T + \ldots \tag{1.15}$$

where

x_n = point about which Taylor series is expanded
x = point near x_n
x^T = transpose of vector (in this case row vector becomes column vector)
$\mathbf{H}$ = Hessian matrix with elements given by $h_{mn} = \partial^2 f/\partial x_m \partial x_n$

Taking the gradient of the first two terms of (1.15) and setting it equal to zero yields

$$\nabla f(x) = \nabla f(x_n) + (x - x_n)\mathbf{H} = 0 \tag{1.16}$$

Starting with a guess x_0, the next point, x_{n+1}, can be found from the previous point, x_n, by

$$x_{n+1} = x_n - \mathbf{H}^{-1}\nabla f(x_n) \tag{1.17}$$

Rarely is the Hessian matrix known. A myriad of algorithms have spawned around this formulation. In general, these techniques are formulated as

$$x_{n+1} = x_n - \alpha_n \mathbf{A}_n \nabla f(x_n) \tag{1.18}$$

where

α_n = step size at iteration n

$\mathbf{A}_n$= approximation to the Hessian matrix at iteration n

Note that when $\mathbf{A}_n = \mathbf{I}$, the identity matrix (1.18) becomes the method of steepest descent, and when $\mathbf{A}_n = \mathbf{H}^{-1}$, (1.18) becomes Newton's method.

Two excellent quasi-Newton techniques that construct a sequence of approximations to the Hessian, such that

$$\lim_{n \to \infty} \mathbf{A}_n = \mathbf{H}^{-1} \tag{1.19}$$

The first approach is called the Davidon-Fletcher-Powell (DFP) algorithm (Powell, 1964). Powell developed a method that finds a set of line minimization directions that are linearly independent, mutually conjugate directions (Powell, 1964). The direction assuring the current direction does not "spoil" the minimization of the prior direction is the conjugate direction. The conjugate directions are chosen so that the change in the gradient of the cost function remains perpendicular to the previous direction. If the cost function is quadratic, then the algorithm converges in N_{var} iterations (see Figure 1.12). If the cost function is not quadratic, then repeating the N_{var} iterations several times usually brings the algorithm closer to the minimum. The second algorithm is named the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm,

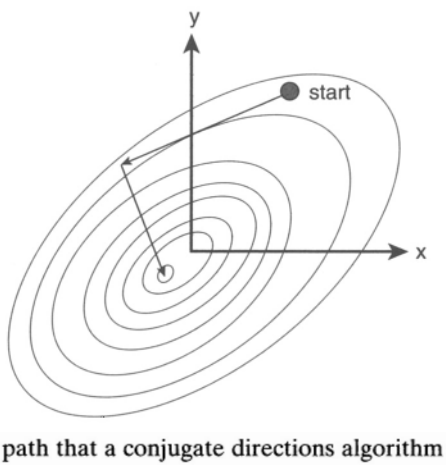

Figure 1.12 Possible path that a conjugate directions algorithm might take on a quadratic cost surface.

discovered by its four namesakes independently in the mid-1960s (Broyden, 1965; Fletcher, 1963; Goldfarb, 1968; Shanno, 1970). Both approaches find a way to approximate this matrix and employ it in determining the appropriate directions of movement. This algorithm is "quasi-Newton" in that it is equivalent to Newton's method for prescribing the next best point to use for the iteration, yet it doesn't use an exact Hessian matrix. The BFGS algorithm is robust and quickly converges, but it requires an extra step to approximate the Hessian compared to the DFP algorithm. These algorithms have the advantages of being fast and working with or without the gradient or Hessian. On the other hand, they have the disadvantages of finding minimum close to the starting point and having an approximation to the Hessian matrix that is close to singular.

Quadratic programming assumes that the cost function is quadratic (variables are squared) and the constraints are linear. This technique is based on Lagrange multipliers and requires derivatives or approximations to derivatives. One powerful method known as recursive quadratic programming solves the quadratic programming problem at each iteration to find the direction of the next step (Luenberger, 1984). The approach of these methods is similar to using very refined surveying tools, which unfortunately still does not guarantee that the hiker will find the lowest point in the park.

1.3 NATURAL OPTIMIZATION METHODS

The methods already discussed take the same basic approach of heading downhill from an arbitrary starting point. They differ in deciding in which direction to move and how far to move. Successive improvements increase the speed of the downhill algorithms but don't add to the algorithm's ability to find a global minimum instead of a local minimum.

All hope is not lost! Some outstanding algorithms have surfaced in recent times. Some of these methods include the genetic algorithm (Holland, 1975), simulated annealing (Kirkpatrick et al., 1983), particle swarm optimization (Parsopoulos and Vrahatis, 2002), ant colony optimization (Dorigo and Maria, 1997), and evolutionary algorithms (Schwefel, 1995). These methods generate new points in the search space by applying operators to current points and statistically moving toward more optimal places in the search space. They rely on an intelligent search of a large but finite solution space using statistical methods. The algorithms do not require taking cost function derivatives and can thus deal with discrete variables and noncontinuous cost functions. They represent processes in nature that are remarkably successful at optimizing natural phenomena. A selection of these algorithms is presented in Chapter 7.

1.4 BIOLOGICAL OPTIMIZATION: NATURAL SELECTION

This section introduces the current scientific understanding of the natural selection process with the purpose of gaining an insight into the construction, application, and terminology of genetic algorithms. Natural selection is discussed in many texts and treatises. Much of the information summarized here is from Curtis (1975) and Grant (1985).

Upon observing the natural world, we can make several generalizations that lead to our view of its origins and workings. First, there is a tremendous diversity of organisms. Second, the degree of complexity in the organisms is striking. Third, many of the features of these organisms have an apparent usefulness. Why is this so? How did they come into being?

Imagine the organisms of today's world as being the results of many iterations in a grand optimization algorithm. The cost function measures survivability, which we wish to maximize. Thus the characteristics of the organisms of the natural world fit into this topological landscape (Grant, 1985). The level of adaptation, the fitness, denotes the elevation of the landscape. The highest points correspond to the most-fit conditions. The environment, as well as how the different species interact, provides the constraints. The process of evolution is the grand algorithm that selects which characteristics produce a species of organism fit for survival. The peaks of the landscape are populated by living organisms. Some peaks are broad and hold a wide range of characteristics encompassing many organisms, while other peaks are very narrow and allow only very specific characteristics. This analogy can be extended to include saddles between peaks as separating different species. If we take a very parochial view and assume that intelligence and ability to alter the environment are the most important aspects of survivability, we can imagine the global maximum peak at this instance in biological time to contain humankind.

To begin to understand the way that this natural landscape was populated involves studying the two components of natural selection: genetics and evolution. Modern biologists subscribe to what is known as the synthetic theory of natural selection—a synthesis of genetics with evolution. There are two main divisions of scale in this synthetic evolutionary theory: macroevolution, which involves the process of division of the organisms into major groups, and microevolution, which deals with the process within specific populations. We will deal with microevolution here and consider macroevolution to be beyond our scope.

First, we need a bit of background on heredity at the cellular level. A *gene* is the basic unit of heredity. An organism's genes are carried on one of a pair of *chromosomes* in the form of deoxyribonucleic acid (DNA). The DNA is in the form of a double helix and carries a symbolic system of base-pair sequences that determine the sequence of enzymes and other proteins in an organism. This sequence does not vary and is known as the *genetic code* of the organism. Each cell of the organism contains the same number of chromo-

somes. For instance, the number of chromosomes per body cell is 6 for mosquitoes, 26 for frogs, 46 for humans, and 94 for goldfish. Genes often occur with two functional forms, each representing a different characteristic. Each of these forms is known as an *allele*. For instance, a human may carry one allele for brown eyes and another for blue eyes. The combination of alleles on the chromosomes determines the traits of the individual. Often one allele is dominant and the other recessive, so that the dominant allele is what is manifested in the organism, although the recessive one may still be passed on to its offspring. If the allele for brown eyes is dominant, the organism will have brown eyes. However, it can still pass the blue allele to its offspring. If the second allele from the other parent is also for blue eyes, the child will be blue-eyed.

The study of genetics began with the experiments of Gregor Mendel. Born in 1822, Mendel attended the University of Vienna, where he studied both biology and mathematics. After failing his exams, he became a monk. It was in the monastery garden where he performed his famous pea plant experiments. Mendel revolutionized experimentation by applying mathematics and statistics to analyzing and predicting his results. By his hypothesizing and careful planning of experiments, he was able to understand the basic concepts of genetic inheritance for the first time, publishing his results in 1865. As with many brilliant discoveries, his findings were not appreciated in his own time.

Mendel's pea plant experiments were instrumental in delineating how traits are passed from one generation to another. One reason that Mendel's experiments were so successful is that pea plants are normally self-pollinating and seldom cross-pollinate without intervention. The self-pollination is easily prevented. Another reason that Mendel's experiments worked was the fact that he spent several years prior to the actual experimentation documenting the inheritable traits and which ones were easily separable and bred pure. This allowed him to crossbreed his plants and observe the characteristics of the offspring and of the next generation. By carefully observing the distribution of traits, he was able to hypothesize his first law—the principle of segregation; that is, that there must be factors that are inherited in pairs, one from each parent. These factors are indeed the genes and their different realizations are the alleles. When both alleles of a gene pair are the same, they are *homozygous*. When they are different, they are *heterozygous*. The brown-blue allele for eye color of a parent was heterozygous while the blue-blue combination of the offspring is homozygous. The trait actually observed is the *phenotype*, but the actual combination of alleles is the *genotype*. Although the parent organism had a brown-blue eye color phenotype, its genotype is for brown eyes (the dominant form). The genotype must be inferred from the phenotype percentages of the succeeding generation as well as the parent itself. Since the offspring had blue eyes, we can infer that each parent had a blue allele to pass along, even though the phenotype of each parent was brown eyes. Therefore, since the offspring was homozygous, carrying two alleles for blue eyes, both parents must be heterozygous, having one brown and one blue allele. Mendel's second law is the principle of independent assortment. This principle states

that the inheritance of the allele for one trait is independent of that for another. The eye color is irrelevant when determining the size of the individual.

To understand how genes combine into phenotypes, it is helpful to understand some basics of cell division. Reproduction in very simple, single-celled organisms occurs by cell division, known as *mitosis*. During the phases of mitosis, the chromosome material is exactly copied and passed onto the offspring. In such simple organisms the daughter cells are identical to the parent. There is little opportunity for evolution of such organisms. Unless a mutation occurs, the species propagates unchanged. Higher organisms have developed a more efficient method of passing on traits to their offspring—sexual reproduction. The process of cell division that occurs then is called *meiosis*. The *gamete*, or reproductive cell, has half the number of chromosomes as the other body cells. Thus the gametes cells are called *haploid*, while the body cells are *diploid*. Only these diploid body cells contain the full genetic code. The diploid number of chromosomes is reduced by half to form the haploid number for the gametes. In preparation for meiosis, the gamete cells are duplicated. Then the gamete cells from the mother join with those from the father (this process is not discussed here). They arrange themselves in *homologous* pairs; that is, each chromosome matches with one of the same length and shape. As they match up, they join at the *kinetochore*, a random point on this matched chromosome pair (or actually tetrad in most cases). As meiosis progresses, the kinetochores divide so that a left portion of the mother chromosome is conjoined with the right portion of the father, and visa versa for the other portions. This process is known as *crossing over*. The resulting cell has the full diploid number of chromosomes. Through this crossing over, the genetic material of the mother and father has been combined in a manner to produce a unique individual offspring. This process allows changes to occur in the species.

Now we turn to discussing the second component of natural selection—evolution—and one of its first proponents, Charles Darwin. Darwin refined his ideas during his voyage as naturalist on the *Beagle*, especially during his visits to the Galapagos Islands. Darwin's theory of evolution was based on four primary premises. First, like begets like; equivalently, an offspring has many of the characteristics of its parents. This premise implies that the population is stable. Second, there are variations in characteristics between individuals that can be passed from one generation to the next. The third premise is that only a small percentage of the offspring produced survive to adulthood. Finally, which of the offspring survive depends on their inherited characteristics. These premises combine to produce the theory of natural selection. In modern evolutionary theory an understanding of genetics adds impetus to the explanation of the stages of natural selection.

A group of interbreeding individuals is called a *population*. Under static conditions the characteristics of the population are defined by the *Hardy-Weinberg Law*. This principle states that the frequency of occurrence of the alleles will stay the same within an inbreeding population if there are no per-

turbations. Thus, although the individuals show great variety, the statistics of the population remain the same. However, we know that few populations are static for very long. When the population is no longer static, the proportion of allele frequencies is no longer constant between generations and evolution occurs. This dynamic process requires an external forcing. The forcing may be grouped into four specific types. (1) *Mutations* may occur; that is, a random change occurs in the characteristics of a gene. This change may be passed along to the offspring. Mutations may be spontaneous or due to external factors such as exposure to environmental factors. (2) *Gene flow* may result from introduction of new organisms into the breeding population. (3) *Genetic drift* may occur solely due to chance. In small populations certain alleles may sometimes be eliminated in the random combinations. (4) *Natural selection* operates to choose the *most fit* individuals for further reproduction. In this process certain alleles may produce an individual that is more prepared to deal with its environment. For instance, fleeter animals may be better at catching prey or running from predators, thus being more likely to survive to breed. Therefore certain characteristics are *selected* into the breeding pool.

Thus we see that these ideas return to natural selection. The important components have been how the genes combine and cross over to produce new individuals with combinations of traits and how the dynamics of a large population interact to select for certain traits. These factors may move this offspring either up toward a peak or down into the valley. If it goes too far into the valley, it may not survive to mate—better adapted ones will. After a long period of time the pool of organisms becomes well adapted to its environment. However, the environment is dynamic. The predators and prey, as well as factors such as the weather and geological upheaval, are also constantly changing. These changes act to revise the optimization equation. That is what makes life (and genetic algorithms) interesting.

1.5 THE GENETIC ALGORITHM

The genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function). The method was developed by John Holland (1975) over the course of the 1960s and 1970s and finally popularized by one of his students, David Goldberg, who was able to solve a difficult problem involving the control of gas-pipeline transmission for his dissertation (Goldberg, 1989). Holland's original work was summarized in his book. He was the first to try to develop a theoretical basis for GAs through his schema theorem. The work of De Jong (1975) showed the usefulness of the GA for function optimization and made the first concerted effort to find optimized GA parameters. Goldberg has probably contributed the most fuel to the GA fire with his successful appli-

cations and excellent book (1989). Since then, many versions of evolutionary programming have been tried with varying degrees of success.

Some of the advantages of a GA include that it

- Optimizes with continuous or discrete variables,
- Doesn't require derivative information,
- Simultaneously searches from a wide sampling of the cost surface,
- Deals with a large number of variables,
- Is well suited for parallel computers,
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum),
- Provides a list of optimum variables, not just a single solution,
- May encode the variables so that the optimization is done with the encoded variables, and
- Works with numerically generated data, experimental data, or analytical functions.

These advantages are intriguing and produce stunning results when traditional optimization approaches fail miserably.

Of course, the GA is not the best way to solve every problem. For instance, the traditional methods have been tuned to quickly find the solution of a well-behaved convex analytical function of only a few variables. For such cases the calculus-based methods outperform the GA, quickly finding the minimum while the GA is still analyzing the costs of the initial population. For these problems the optimizer should use the experience of the past and employ these quick methods. However, many realistic problems do not fall into this category. In addition, for problems that are not overly difficult, other methods may find the solution faster than the GA. The large population of solutions that gives the GA its power is also its bane when it comes to speed on a serial computer—the cost function of each of those solutions must be evaluated. However, if a parallel computer is available, each processor can evaluate a separate function at the same time. Thus the GA is optimally suited for such parallel computations.

This book shows how to use a GA to optimize problems. Chapter 2 introduces the binary form while using the algorithm to find the highest point in Rocky Mountain National Park. Chapter 3 describes another version of the algorithm that employs continuous variables. We demonstrate this method with a GA solution to equation (1.1) subject to constraints (1.2). The remainder of the book presents refinements to the algorithm by solving more problems, winding its way from easier, less technical problems into more difficult problems that cannot be solved by other methods. Our goal is to give specific ways to deal with certain types of problems that may be typical of the ones faced by scientists and engineers on a day-to-day basis.

BIBLIOGRAPHY

Anderson, D. Z. 1992. Linear programming. In *McGraw-Hill Encyclopedia of Science and Technology* **10**. New York: McGraw-Hill, pp. 86–88.

Borowski, E. J., and J. M. Borwein. 1991. *Mathematics Dictionary*. New York: HarperCollins.

Box, M. J. 1965. A comparison of several current optimization methods and the use of transformations in constrained problems. *Comput. J.* **8**:67–77.

Boyer, C. B., and U. C. Merzbach. 1991. *A History of Mathematics*. New York: Wiley.

Broyden, G. C. 1965. A class of methods for solving nonlinear simultaneous equations. *Math. Comput.* **19**:577–593.

Curtis, H. 1975. *Biology*, 2nd ed. New York: Worth.

Cuthbert, T. R. Jr. 1987. *Optimization Using Personal Computers*. New York: Wiley.

De Jong, K. A. 1975. Analysis of the behavior of a class of genetic adaptive systems. Ph.D. Dissertation. University of Michigan, Ann Arbor.

Dorigo, M., and G. Maria. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1**:53–66.

Fletcher, R. 1963. Generalized inverses for nonlinear equations and optimization. In R. Rabinowitz (ed.), *Numerical Methods for Non-linear Algebraic Equations*. London: Gordon and Breach.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

Goldfarb, D., and B. Lapidus. 1968. Conjugate gradient method for nonlinear programming problems with linear constraints. *I&EC Fundam.* **7**:142–151.

Grant, V. 1985. *The Evolutionary Process*. New York: Columbia University Press.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* **220**:671–680.

Luenberger, D. G. 1984. *Linear and Nonlinear Programming*, Reading, MA: Addison-Wesley.

Nelder, J. A., and R. Mead. 1965. A simplex method for function minimization. *Comput. J.* **7**:308–313.

Parsopoulos, K. E., and M. N. Vrahatis. 2002. Recent approaches to global optimization problems through particle swarm optimization. In *Natural Computing*. Netherlands: Kluwer Academic, pp. 235–306.

Pierre, D. A. 1992. Optimization. In *McGraw-Hill Encyclopedia of Science and Technology* **12**. New York: McGraw-Hill, pp. 476–482.

Powell, M. J. D. 1964. An efficient way for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.* **7**:155–162.

Press, W. H., S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. 1992. *Numerical Recipes*. New York: Cambridge University Press.

Rosenbrock, H. H. 1960. An automatic method for finding the greatest or least value of a function. *Comput. J.* **3**:175–184.

Schwefel, H. 1995. *Evolution and Optimum Seeking.* New York: Wiley.

Shanno, D. F. 1970. An accelerated gradient projection method for linearly constrained nonlinear estimation. *SIAM J. Appl. Math.* **18**:322–334.

Thompson, G. L. 1992. Game theory. In *McGraw-Hill Encyclopedia of Science and Technology* **7**. New York: McGraw-Hill, pp. 555–560.

Williams, H. P. 1993. *Model Solving in Mathematical Programming.* New York: Wiley.

EXERCISES

Use the following local optimizers:

a. Nelder-Mead downhill simplex

b. BFGS

c. DFP

d. Steepest descent

e. Random search

1. Find the minimum of _____ (one of the functions in Appendix I) using _____ (one of the local optimizers).

2. Try _____ different random starting values to find the minimum. What do you observe?

3. Combine 1 and 2, and find the minimum 25 times using random starting points. How often is the minimum found?

4. Compare the following algorithms:_____

5. Since local optimizers often decrease the step size when approaching the minimum, running the algorithm again after it has found a minimum increases the odds of getting a better solution. Repeat 3 in such a way that the solution is used as a starting point by the algorithm on a second run. Does this help? With which functions? Why?

6. Some of the MATLAB optimization routines give you a choice of providing the exact derivatives. Do these algorithms converge better with the exact derivatives or approximate numerical derivatives?

7. Find the minimum of $f = u^2 + 2v^2 + w^2 + x^2$ subject to $u + 3v - w + x = 2$ and $2u - v + w - 2x = 4$ using Lagrange multipliers. Assume no constraints. (Answer: $u = 67/69$, $v = 6/69$, $w = 14/69$, $x = 67/69$ with $\kappa_1 = -26/69$, $\kappa_2 = -54/69$.)

8. Many problems have constraints on the variables. Using a transformation of variables, convert (1.1) and (1.2) into an unconstrained optimization problem, and try one of the local optimizers. Does the transformation used affect the speed of convergence?