CHAPTER 9

Unconstrained Optimization

CONTENTS

PREVIOUS chapters have taken a largely *variational* approach to deriving numerical algorithms. That is, we define an *objective function* or *energy* $E(\vec{x})$, possibly with constraints, and pose our algorithms as approaches to a corresponding minimization or maximization problem. A sampling of problems that we solved this way is listed below:

Problem	**§**	**Objective**	**Constraints**
Least-squares	4.1.2	$E(\vec{x}) = \|A\vec{x} - \vec{b}\|_2^2$	None
Project $\vec{b}$ onto $\vec{a}$	5.4.1	$E(c) = \|c\vec{a} - \vec{b}\|_2^2$	None
Eigenvectors of symmetric A	6.1	$E(\vec{x}) = \vec{x}^\top A\vec{x}$	$\|\vec{x}\|_2 = 1$
Pseudoinverse	7.2.1	$E(\vec{x}) = \|\vec{x}\|_2^2$	$A^\top A\vec{x} = A^\top \vec{b}$
Principal component analysis	7.2.5	$E(C) = \|X - CC^\top X\|_{\text{Fro}}$	$C^\top C = I_{d\times d}$
Broyden step	8.2.2	$E(J_k) = \|J_k - J_{k-1}\|_{\text{Fro}}^2$	$J_k \cdot \Delta\vec{x}_k = \Delta f_k$

The formulation of numerical problems in variational language is a powerful and general technique. To make it applicable to a larger class of nonlinear problems, we will design algorithms that can perform minimization or maximization in the absence of a special form for the energy E.

9.1 UNCONSTRAINED OPTIMIZATION: MOTIVATION

In this chapter, we will consider *unconstrained* problems, that is, problems that can be posed as minimizing or maximizing a function $f : \mathbb{R}^n \to \mathbb{R}$ without any constraints on the input $\vec{x}$. It is not difficult to encounter such problems in practice; we explore a few examples below.

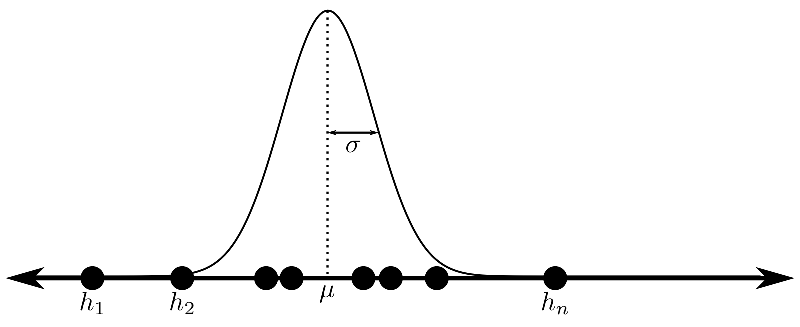

Figure 9.1 Illustration for Example 9.2. Given the heights $h_1, h_2, \ldots, h_n$ of students in a class, we may wish to estimate the mean μ and standard deviation σ of the most likely normal distribution explaining the observed heights.

Example 9.1 (Nonlinear least-squares)**.** Suppose we are given a number of pairs (x_i, y_i) such that $f(x_i) \approx y_i$ and wish to find the best approximating f within a particular class. For instance, if we expect that f is exponential, we should be able to write $f(x) = ce^{ax}$ for some $c, a \in \mathbb{R}$; our job is to find the parameters a and c that best fit the data. One strategy we already developed in Chapter 4 is to minimize the following energy function:

$$E(a,c) = \sum_i (y_i - ce^{ax_i})^2.$$

This form for E is not quadratic in a, so the linear least-squares methods from §4.1.2 do not apply to this minimization problem. Hence, we must employ alternative methods to minimize E.

Example 9.2 (Maximum likelihood estimation)**.** In machine learning, the problem of *parameter estimation* involves examining the results of a randomized experiment and trying to summarize them using a probability distribution of a particular form. For example, we might measure the height of every student in a class to obtain a list of heights h_i for each student i. If we have a lot of students, we can model the distribution of student heights using a *normal distribution*:

$$g(h; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(h-\mu)^2/2\sigma^2},$$

where μ is the mean of the distribution and σ is the standard deviation of the standard "bell curve" shape. This notation is illustrated in Figure 9.1.

Under this normal distribution, the likelihood that we observe height h_i for student i is given by $g(h_i; \mu, \sigma)$, and under the (reasonable) assumption that the height of student i is probabilistically independent of that of student j, the likelihood of observing the entire set of heights observed is proportional to the product

$$P(\{h_1, \ldots, h_n\}; \mu, \sigma) = \prod_i g(h_i; \mu, \sigma).$$

A common method for estimating the parameters μ and σ of g is to maximize P viewed as a function of μ and σ with $\{h_i\}$ fixed; this is called the *maximum-likelihood estimate* of μ and

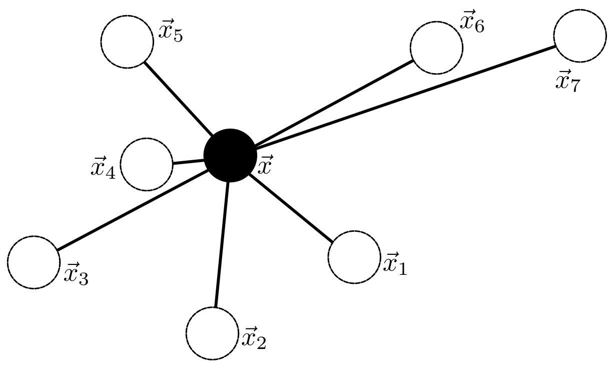

Figure 9.2 The geometric median problem seeks a point $\vec{x}$ minimizing the total (non-squared) distance to a set of data points $\vec{x}_1, \ldots, \vec{x}_k$.

σ. In practice, we usually optimize the *log likelihood* $\ell(\mu, \sigma) \equiv \log P(\{h_1, \ldots, h_n\}; \mu, \sigma)$. This function has the same maxima but enjoys better numerical and mathematical properties.

Example 9.3 (Geometric problems)**.** Many geometric problems encountered in computer graphics and vision do not reduce to least-squares energies. For instance, suppose we have a number of points $\vec{x}_1, \ldots, \vec{x}_k \in \mathbb{R}^n$. If we wish to *cluster* these points, we might wish to summarize them with a single $\vec{x}$ minimizing

$$E(\vec{x}) \equiv \sum_i \|\vec{x} - \vec{x}_i\|_2.$$

The $\vec{x}$ minimizing E is known as the *geometric median* of $\{\vec{x}_1, \ldots, \vec{x}_k\}$, as illustrated in Figure 9.2. Since the norm of the difference $\vec{x} - \vec{x}_i$ in E is not squared, the energy is no longer quadratic in the components of $\vec{x}$.

Example 9.4 (Physical equilibria, adapted from [58])**.** Suppose we attach an object to a set of springs; each spring is anchored at point $\vec{x}_i \in \mathbb{R}^3$ with natural length L_i and constant k_i. In the absence of gravity, if our object is located at position $\vec{p} \in \mathbb{R}^3$, the network of springs has potential energy

$$E(\vec{p}) = \frac{1}{2} \sum_i k_i \left(\|\vec{p} - \vec{x}_i\|_2 - L_i \right)^2.$$

Equilibria of this system are given by local minima of E and represent points $\vec{p}$ at which the spring forces are all balanced. Extensions of this problem are used to visualize graphs $G = (V, E)$, by attaching vertices in V with springs for each pair in E.

9.2 OPTIMALITY

Before discussing how to minimize or maximize a function, we should characterize properties of the maxima and minima we are seeking. With this goal in mind, for a particular $f : \mathbb{R}^n \to \mathbb{R}$ and $\vec{x}^* \in \mathbb{R}^n$, we will derive *optimality conditions* that verify whether $\vec{x}^*$ has the optimal

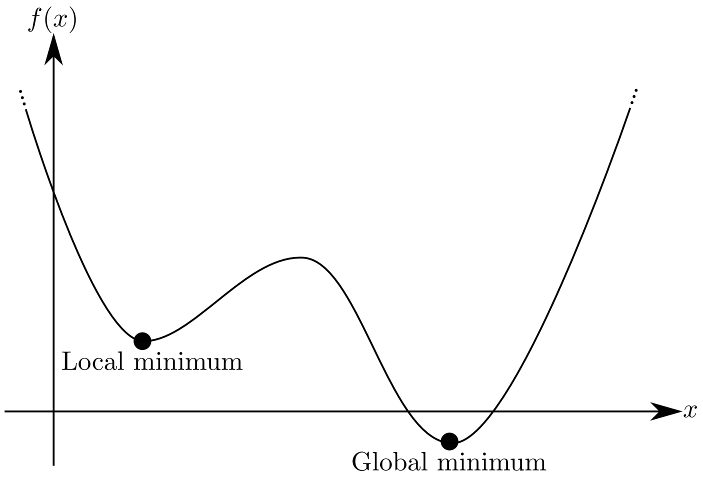

Figure 9.3 A function $f(x)$ with two local minima but only one global minimum.

value $f(\vec{x}^*)$. Maximizing f is the same as minimizing $-f$, so from this section onward the minimization problem is sufficient for our consideration.

In most situations, we ideally would like to find *global* minima of f:

Definition 9.1 (Global minimum)**.** The point $\vec{x}^* \in \mathbb{R}^n$ is a *global minimum* of $f : \mathbb{R}^n \to \mathbb{R}$ if $f(\vec{x}^*) \leq f(\vec{x})$ for all $\vec{x} \in \mathbb{R}^n$.

Finding a global minimum of $f(\vec{x})$ without any bounds on $\vec{x}$ or information about the structure of f effectively requires searching in the dark. For instance, suppose an optimization algorithm identifies the left local minimum in the function in Figure 9.3. It is nearly impossible to realize that there is a second, lower minimum by guessing x values—and for all we know, there may be a third even lower minimum of f miles to the right!

To relax these difficulties, in many cases we are satisfied if we can find a *local* minimum:

Definition 9.2 (Local minimum)**.** The point $\vec{x}^* \in \mathbb{R}^n$ is a *local minimum* of $f : \mathbb{R}^n \to \mathbb{R}$ if there exists some $\varepsilon > 0$ such that $f(\vec{x}^*) \leq f(\vec{x})$ for all $\vec{x} \in \mathbb{R}^n$ satisfying $\|\vec{x} - \vec{x}^*\|_2 < \varepsilon$.

This definition requires that $\vec{x}^*$ attains the smallest value in some *neighborhood* defined by the radius ε. Local optimization algorithms have the severe limitation that they may not find the lowest possible value of f, as in the left local minimum in Figure 9.3. To mitigate these issues, many strategies, heuristic and otherwise, are applied to explore the landscape of possible $\vec{x}$'s to help gain confidence that a local minimum has the best possible value.

9.2.1 Differential Optimality

A familiar story from single- and multi-variable calculus is that finding potential minima and maxima of a function $f : \mathbb{R}^n \to \mathbb{R}$ is more straightforward when f is differentiable. In this case, the gradient vector $\nabla f = ({}^{\partial f}/_{\partial x_1}, \ldots, {}^{\partial f}/_{\partial x_n})$ at $\vec{x}$ points in the direction moving from $\vec{x}$ in which f increases at the fastest rate; the vector $-\nabla f$ points in the direction of greatest decrease. One way to see this is to approximate $f(\vec{x})$ linearly near a point $\vec{x}_0 \in \mathbb{R}^n$:

$$f(\vec{x}) \approx f(\vec{x}_0) + \nabla f(\vec{x}_0) \cdot (\vec{x} - \vec{x}_0).$$

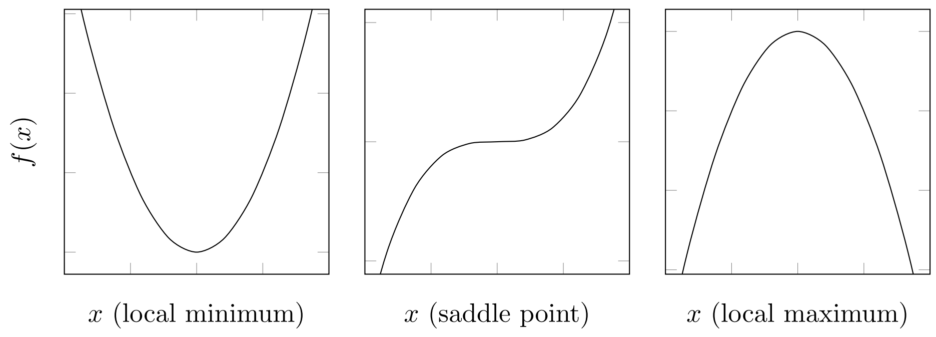

Figure 9.4 Different types of critical points.

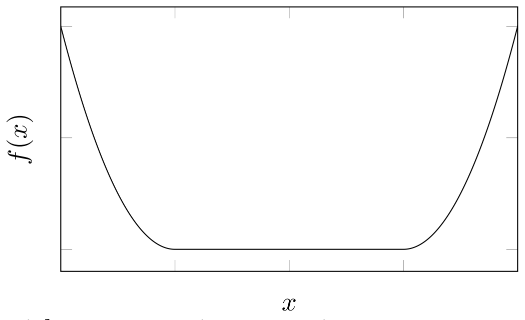

Figure 9.5 A function with many stationary points.

If we take $\vec{x} - \vec{x}_0 = \alpha \nabla f(\vec{x}_0)$, then

$$f(\vec{x}_0 + \alpha \nabla f(\vec{x}_0)) \approx f(\vec{x}_0) + \alpha \|\nabla f(\vec{x}_0)\|_2^2.$$

The value $\|\nabla f(\vec{x}_0)\|_2^2$ is *always* nonnegative, so when $\|\nabla f(\vec{x}_0)\|_2 > 0$, the sign of α determines whether f increases or decreases locally.

By the above argument, if $\vec{x}_0$ is a local minimum, then $\nabla f(\vec{x}_0) = \vec{0}$. This condition is *necessary* but not *sufficient*: Maxima and saddle points also have $\nabla f(\vec{x}_0) = \vec{0}$ as shown in Figure 9.4. Even so, this observation about minima of differentiable functions yields a high-level approach to optimization:

1. Find points $\vec{x}_i$ satisfying $\nabla f(\vec{x}_i) = \vec{0}$.

2. Check which of these points is a local minimum as opposed to a maximum or saddle point.

Given their role in optimization, we give the points $\vec{x}_i$ a special name:

Definition 9.3 (Stationary point)**.** A *stationary point* of $f : \mathbb{R}^n \to \mathbb{R}$ is a point $\vec{x} \in \mathbb{R}^n$ satisfying $\nabla f(\vec{x}) = \vec{0}$.

Our methods for minimization mostly will find stationary points of f and subsequently eliminate those that are not minima.

It is imperative to keep in mind when we can expect minimization algorithms to succeed. In most cases, such as those in Figure 9.4, the stationary points of f are *isolated*, meaning we can write them in a discrete list $\{\vec{x}_0, \vec{x}_1, \ldots\}$. A degenerate case, however, is shown in Figure 9.5; here, an entire interval of values x is composed of stationary points, making it impossible to consider them individually. For the most part, we will ignore such issues as unlikely, poorly conditioned degeneracies.

Suppose we identify a point $\vec{x} \in \mathbb{R}$ as a stationary point of f and wish to check if it is a local minimum. If f is twice-differentiable, we can use its *Hessian* matrix

$$H_f(\vec{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix}.$$

Adding a term to the linearization of f reveals the role of H_f:

$$f(\vec{x}) \approx f(\vec{x}_0) + \nabla f(\vec{x}_0) \cdot (\vec{x} - \vec{x}_0) + \frac{1}{2}(\vec{x} - \vec{x}_0)^\top H_f(\vec{x} - \vec{x}_0).$$

If we substitute a stationary point $\vec{x}^*$, then since $\nabla f(\vec{x}^*) = \vec{0}$,

$$f(\vec{x}) \approx f(\vec{x}^*) + \frac{1}{2}(\vec{x} - \vec{x}^*)^\top H_f(\vec{x} - \vec{x}^*).$$

If H_f is positive definite, then this expression shows $f(\vec{x}) \geq f(\vec{x}^*)$ near $\vec{x}^*$, and thus $\vec{x}^*$ is a local minimum. More generally, a few situations can occur:

- If H_f is *positive definite*, then $\vec{x}^*$ is a local minimum of f.
- If H_f is *negative definite*, then $\vec{x}^*$ is a local maximum of f.
- If H_f is *indefinite*, then $\vec{x}^*$ is a saddle point of f.
- If H_f is *not invertible*, then oddities such as the function in Figure 9.5 can occur; this includes the case where H_f is *semidefinite*.

Checking if a Hessian matrix is positive definite can be accomplished by checking if its Cholesky factorization exists or—more slowly—by verifying that all its eigenvalues are positive. So, when f is sufficiently smooth and the Hessian of f is known, we can check stationary points for optimality using the list above. Many optimization algorithms including the ones we will discuss ignore the non-invertible case and notify the user, since again it is relatively unlikely.

9.2.2 Alternative Conditions for Optimality

If we know more information about $f : \mathbb{R}^n \to \mathbb{R}$, we can provide optimality conditions that are stronger or easier to check than the ones above. These conditions also can help when f is not differentiable but has other geometric properties that make it possible to find a minimum.

One property of f that has strong implications for optimization is *convexity*, illustrated in Figure 9.6(a):

Definition 9.4 (Convex)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* when for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $\alpha \in (0, 1)$ the following relationship holds:

$$f((1 - \alpha)\vec{x} + \alpha\vec{y}) \leq (1 - \alpha)f(\vec{x}) + \alpha f(\vec{y}).$$

When the inequality is strict (replace $\leq$ with $<$), the function is *strictly convex.*

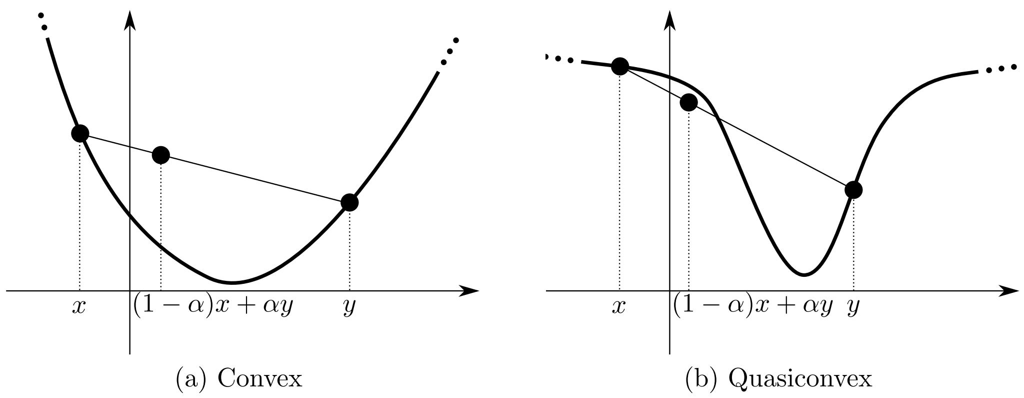

Figure 9.6 (a) Convex functions must be bowl-shaped, while (b) quasiconvex functions can have more complicated features.

Convexity implies that if you connect two points in $\mathbb{R}^n$ with a line, the values of f along the line are less than or equal to those you would obtain by linear interpolation.

Convex functions enjoy many strong properties, the most basic of which is the following:

Proposition 9.1. A local minimum of a convex function $f : \mathbb{R}^n \to \mathbb{R}$ is necessarily a global minimum.

Proof. Take $\vec{x}$ to be such a local minimum and suppose there exists $\vec{x}^*$ with $f(\vec{x}^*) < f(\vec{x})$. Then, for sufficiently small $\alpha \in (0,1)$,

$$\begin{aligned} f(\vec{x}) &\leq f(\vec{x} + \alpha(\vec{x}^* - \vec{x})) \text{ since } \vec{x} \text{ is a local minimum} \\ &\leq (1-\alpha)f(\vec{x}) + \alpha f(\vec{x}^*) \text{ by convexity.} \end{aligned}$$

Moving terms in the inequality $f(\vec{x}) \leq (1-\alpha)f(\vec{x}) + \alpha f(\vec{x}^*)$ shows $f(\vec{x}) \leq f(\vec{x}^*)$. This contradicts our assumption that $f(\vec{x}^*) < f(\vec{x})$, so $\vec{x}$ must minimize f globally. □

This proposition and related observations show that it is possible to check if you have reached a *global* minimum of a convex function by applying first-order optimality. Thus, it is valuable to check by hand if a function being optimized happens to be convex, a situation occurring surprisingly often in scientific computing; one sufficient condition that can be easier to check when f is twice-differentiable is that H_f is positive definite *everywhere.*

Other optimization techniques have guarantees under weaker assumptions about f. For example, one relaxation of convexity is *quasi*-convexity, achieved when

$$f((1-\alpha)\vec{x} + \alpha\vec{y}) \leq \max(f(\vec{x}), f(\vec{y})).$$

An example of a quasiconvex function is shown in Figure 9.6(b). Although it does not have the characteristic "bowl" shape of a convex function, its local minimizers are necessarily global minimizers.

9.3 ONE-DIMENSIONAL STRATEGIES

As in the last chapter, we will start by studying optimization for functions $f : \mathbb{R} \to \mathbb{R}$ of one variable and then expand to more general functions $f : \mathbb{R}^n \to \mathbb{R}$.

9.3.1 Newton's Method

Our principal strategy for minimizing differentiable functions $f : \mathbb{R}^n \to \mathbb{R}$ will be to find stationary points $\vec{x}^*$ satisfying $\nabla f(\vec{x}^*) = 0$. Assuming we can check whether stationary points are maxima, minima, or saddle points as a post-processing step, we will focus on the problem of finding the stationary points $\vec{x}^*$.

To this end, suppose $f : \mathbb{R} \to \mathbb{R}$ is twice-differentiable. Then, following our derivation of Newton's method for root-finding in §8.1.4, we can approximate:

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2.$$

We need to include second-order terms since linear functions have no nontrivial minima or maxima. The approximation on the right-hand side is a parabola whose vertex is located at $x_k - {f'(x_k)}/{f''(x_k)}$.

In reality, f may not be a parabola, so its vertex will not necessarily give a critical point of f directly. So, Newton's method for minimization iteratively minimizes and adjusts the parabolic approximation:

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

This technique is easily analyzed given the work we put into understanding Newton's method in the previous chapter. Specifically, an alternative way to derive the iterative formula above comes from applying Newton's method for root-finding to $f'(x)$, since stationary points x of $f(x)$ satisfy $f'(x) = 0$. Applying results about convergence to a root, in most cases Newton's method for optimization exhibits quadratic convergence, provided the initial guess x_0 is sufficiently close to x^*.

A natural question is whether the secant method can be similarly adapted to minimization. Our derivation of Newton's method above finds roots of f', so the secant method could be used to eliminate f'' but not f' from the optimization formula. One-dimensional situations in which f' is known but not f'' are relatively rare. A more suitable parallel is to replace line segments through the last two iterates, used to approximate f in the secant method for root-finding, with parabolas through the last three iterates. The resulting algorithm, known as *successive parabolic interpolation*, also minimizes a quadratic approximation of f at each iteration, but rather than using $f(x_k)$, $f'(x_k)$, and $f''(x_k)$ to construct the approximation, it uses $f(x_k)$, $f(x_{k-1})$, and $f(x_{k-2})$. This technique can converge superlinearly; in practice, however, it can have drawbacks that make other methods discussed in this chapter more preferable. We explore its design in Exercise 9.3.

9.3.2 Golden Section Search

Since Newton's method for optimization is so closely linked to root-finding, we might ask whether a similar adaptation can be applied to bisection. Unfortunately, this transition is not obvious. A primary reason for using bisection is that it employs the weakest assumption on f needed to find roots: continuity. Continuity is enough to prove the Intermediate Value Theorem, which justifies convergence of bisection. The Intermediate Value Theorem does not apply to extrema of a function in any intuitive way, so it appears that directly using bisection to minimize a function is not so straightforward.

It is valuable, however, to have at least one minimization algorithm available that does not require differentiability of f as an underlying assumption. After all, there are non-differentiable functions that have clear minima, like $f(x) \equiv |x|$ at $x = 0$. To this end, one alternative assumption might be that f is *unimodular*:

```
function GOLDEN-SECTION-SEARCH($f(x), a, b$)
  $\tau \leftarrow \frac{1}{2}(\sqrt{5}-1)$
  $x_0 \leftarrow a + (1-\tau)(b-a)$          ▷ Initial division of interval $a < x_0 < x_1 < b$
  $x_1 \leftarrow a + \tau(b-a)$
  $f_0 \leftarrow f(x_0)$                      ▷ Function values at $x_0$ and $x_1$
  $f_1 \leftarrow f(x_1)$
  for $k \leftarrow 1, 2, 3, \ldots$
    if $|b-a| < \varepsilon$ then              ▷ Golden section search converged
      return $x^* = \frac{1}{2}(a+b)$
    else if $f_0 \geq f_1$ then                ▷ Remove the interval $[a, x_0]$
      $a \leftarrow x_0$                       ▷ Move left side
      $x_0 \leftarrow x_1$                     ▷ Reuse previous iteration
      $f_0 \leftarrow f_1$
      $x_1 \leftarrow a + \tau(b-a)$           ▷ Generate new sample
      $f_1 \leftarrow f(x_1)$
    else if $f_1 > f_0$ then                   ▷ Remove the interval $[x_1, b]$
      $b \leftarrow x_1$                       ▷ Move right side
      $x_1 \leftarrow x_0$                     ▷ Reuse previous iteration
      $f_1 \leftarrow f_0$
      $x_0 \leftarrow a + (1-\tau)(b-a)$       ▷ Generate new sample
      $f_0 \leftarrow f(x_0)$
```

Figure 9.7 The golden section search algorithm finds minima of unimodular functions $f(x)$ on the interval $[a, b]$ even if they are not differentiable.

Definition 9.5 (Unimodular)**.** A function $f : [a, b] \to \mathbb{R}$ is *unimodular* if there exists $x^* \in [a, b]$ such that f is decreasing (or non-increasing) for $x \in [a, x^*]$ and increasing (or non-decreasing) for $x \in [x^*, b]$.

In other words, a unimodular function decreases for some time, and then begins increasing; no localized minima are allowed. Functions like $|x|$ are not differentiable but still are unimodular.

Suppose we have two values x_0 and x_1 such that $a < x_0 < x_1 < b$. We can make two observations that will help us formulate an optimization technique for a unimodular function $f(x)$:

- If $f(x_0) \geq f(x_1)$, then $f(x) \geq f(x_1)$ for all $x \in [a, x_0]$. Thus, the interval $[a, x_0]$ can be discarded in a search for the minimum of f.
- If $f(x_1) \geq f(x_0)$, then $f(x) \geq f(x_0)$ for all $x \in [x_1, b]$, and we can discard the interval $[x_1, b]$.

This structure suggests a bisection-like minimization algorithm beginning with the interval $[a, b]$ and iteratively removing pieces according to the rules above. In such an algorithm, we could remove a *third* of the interval each iteration. This requires two evaluations of f, at $x_0 = {}^{2a}/_{3} + {}^{b}/_{3}$ and $x_1 = {}^{a}/_{3} + {}^{2b}/_{3}$. If evaluating f is expensive, however, we may attempt to reduce the number of evaluations per iteration to one.

To design such a method reducing the computational load, we will focus on the case when $a = 0$ and $b = 1$; the strategies we derive below eventually will work more generally by shifting and scaling. In the absence of more information about f, we will make a symmetric

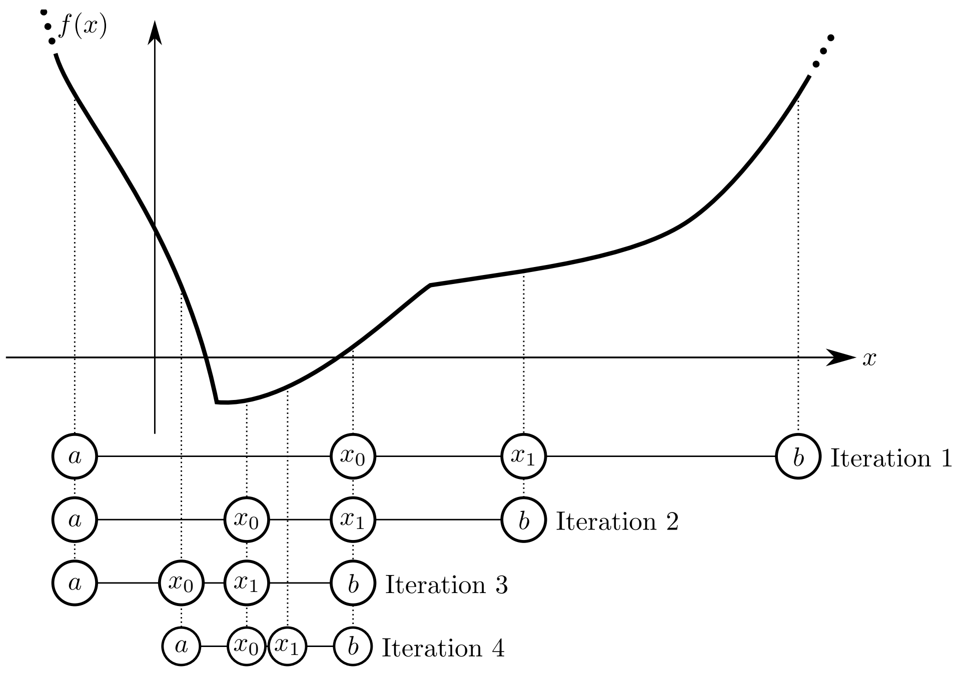

Figure 9.8 Iterations of golden section search on unimodular $f(x)$ shrink the interval $[a, b]$ by eliminating the left segment $[a, x_0]$ or the right segment $[x_1, b]$; each iteration reuses either $f(x_0)$ or $f(x_1)$ via the construction in §9.3.2. In this illustration, each horizontal line represents an iteration of golden section search, with the values a, x_0, x_1, and b labeled in the circles.

choice $x_0 = \alpha$ and $x_1 = 1 - \alpha$ for some $\alpha \in (0, 1/2)$; taking $\alpha = 1/3$ recovers the evenly divided technique suggested above.

Now, suppose that during minimization we can eliminate the rightmost interval $[x_1, b]$ by the rules listed above. In the next iteration, the search interval shrinks to $[0, 1-\alpha]$, with $x_0 = \alpha(1-\alpha)$ and $x_1 = (1-\alpha)^2$. To reuse $f(\alpha)$, we could set $(1-\alpha)^2 = \alpha$, yielding:

$$\alpha = \frac{1}{2}(3 - \sqrt{5})$$
$$1 - \alpha = \frac{1}{2}(\sqrt{5} - 1).$$

The value $1 - \alpha \equiv \tau$ above is the *golden ratio*! A symmetric argument shows that the same choice of α works if we had removed the left interval instead of the right one. In short, “trisection” algorithms minimizing unimodular functions $f(x)$ dividing intervals into segments with length determined using this ratio can reuse a function evaluation from one iteration to the next.

The *golden section search* algorithm, documented in Figure 9.7 and illustrated in Figure 9.8, makes use of this construction to minimize a unimodular function $f(x)$ on the interval $[a, b]$ via subdivision with one evaluation of $f(x)$ per iteration. It converges unconditionally and linearly, since a fraction α of the interval $[a, b]$ bracketing the minimum is removed in each step.

```
function GRADIENT-DESCENT($f(\vec{x}), \vec{x}_0$)
  $\vec{x} \leftarrow \vec{x}_0$
  for $k \leftarrow 1, 2, 3, \ldots$
    DEFINE-FUNCTION($g(t) \equiv f(\vec{x} - t\nabla f(\vec{x}))$)
    $t^* \leftarrow$ LINE-SEARCH($g(t), t \geq 0$)
    $\vec{x} \leftarrow \vec{x} - t^*\nabla f(\vec{x})$          ▷ Update estimate of minimum
    if $\|\nabla f(\vec{x})\|_2 < \varepsilon$ then
      return $x^* = \vec{x}$
```

Figure 9.9 The gradient descent algorithm iteratively minimizes $f : \mathbb{R}^n \to \mathbb{R}$ by solving one-dimensional minimizations through the gradient direction. LINE-SEARCH can be one of the methods from §9.3 for minimization in one dimension. In faster, more advanced techniques, this method can find suboptimal $t^* > 0$ that still decreases $g(t)$ sufficiently to make sure the optimization does not get stuck.

When f is not globally unimodular, golden section search does not apply unless we can find some $[a, b]$ such that f is unimodular on that interval. In some cases, $[a, b]$ can be guessed by attempting to bracket a local minimum of f. For example, [101] suggests stepping farther and farther away from some starting point $x_0 \in \mathbb{R}$, moving downhill from $f(x_0)$ until f increases again, indicating the presence of a local minimum.

9.4 MULTIVARIABLE STRATEGIES

We continue to parallel our discussion of root-finding by expanding from single-variable to multivariable problems. As with root-finding, multivariable optimization problems are considerably more difficult than optimization in a single variable, but they appear so many times in practice that they are worth careful consideration.

Here, we will consider only the case that $f : \mathbb{R}^n \to \mathbb{R}$ is twice-differentiable. Optimization methods similar to golden section search for non-differentiable functions are less common and are difficult to formulate. See, e.g., [74, 17] for consideration of non-differentiable optimization, subgradients, and related concepts.

9.4.1 Gradient Descent

From our previous discussion, $\nabla f(\vec{x})$ points in the direction of "steepest ascent" of f at $\vec{x}$ and $-\nabla f(\vec{x})$ points in the direction of "steepest descent." If nothing else, these properties suggest that when $\nabla f(\vec{x}) \neq \vec{0}$, for small $\alpha > 0$, $f(\vec{x} - \alpha\nabla f(\vec{x})) \leq f(\vec{x})$.

Suppose our current estimate of the minimizer of f is $\vec{x}_k$. A reasonable iterative minimization strategy should seek the next iterate $\vec{x}_{k+1}$ so that $f(\vec{x}_{k+1}) < f(\vec{x}_k)$. Since we do not expect to find a global minimum in one shot, we can make restrictions to simplify the search for $\vec{x}_{k+1}$. A typical simplification is to use a one-variable algorithm from §9.3 on f restricted to a line through $\vec{x}_k$; once we solve the one-dimensional problem for $\vec{x}_{k+1}$, we choose a new line through $\vec{x}_{k+1}$ and repeat.

Consider the function $g_k(t) \equiv f(\vec{x}_k - t\nabla f(\vec{x}_k))$, which restricts f to the line through $\vec{x}_k$ parallel to $-\nabla f(\vec{x}_k)$. We have shown that when $\nabla f(\vec{x}_k) \neq \vec{0}$, $g(t) < g(0)$ for small $t > 0$. Hence, this is a reasonable direction for a restricted search for the new iterate. The resulting *gradient descent* algorithm shown in Figure 9.9 and illustrated in Figure 9.10 iteratively solves one-dimensional problems to improve $\vec{x}_k$.

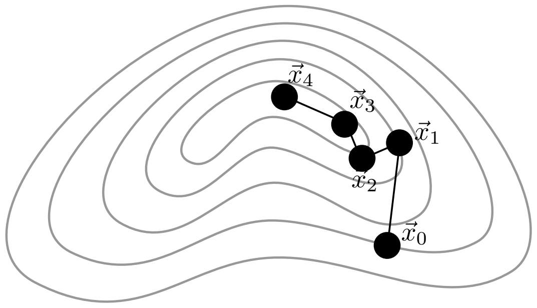

Figure 9.10 Gradient descent on a function $f : \mathbb{R}^2 \to \mathbb{R}$, whose level sets are shown in gray. The gradient $\nabla f(\vec{x})$ points perpendicular to the level sets of f, as in Figure 1.6; gradient descent iteratively minimizes f along the line through this direction.

Each iteration of gradient descent decreases $f(\vec{x}_k)$, so these values converge assuming they are bounded below. The approximations $\vec{x}_k$ only stop changing when $\nabla f(\vec{x}_k) \approx \vec{0}$, showing that gradient descent must at least reach a local minimum; convergence can be slow for some functions f, however.

Rather than solving the one-variable problem exactly in each step, line search can be replaced by a method that finds points along the line that decrease the objective a non-negligible if suboptimal amount. It is more difficult to guarantee convergence in this case, since each step may not reach a local minimum on the line, but the computational savings can be considerable since full one-dimensional minimization is avoided; see [90] for details.

Taking the more limited line search strategy to an extreme, sometimes a fixed $t > 0$ is used for *all* iterations to avoid line search altogether. This choice of t is known in machine learning as the *learning rate* and trades off between taking large minimization steps and potentially skipping over a minimum. Gradient descent with a constant step is unlikely to converge to a minimum in this case, but depending on f it may settle in some neighborhood of the optimal point; see Exercise 9.7 for an error bound of this method in one case.

9.4.2 Newton's Method in Multiple Variables

Paralleling our derivation of the single-variable case in §9.3.1, we can write a Taylor series approximation of $f : \mathbb{R}^n \to \mathbb{R}$ using its Hessian matrix H_f:

$$f(\vec{x}) \approx f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top \cdot (\vec{x} - \vec{x}_k) + \frac{1}{2}(\vec{x} - \vec{x}_k)^\top \cdot H_f(\vec{x}_k) \cdot (\vec{x} - \vec{x}_k).$$

Differentiating with respect to $\vec{x}$ and setting the result equal to zero yields the following iterative scheme:

$$\vec{x}_{k+1} = \vec{x}_k - [H_f(\vec{x}_k)]^{-1} \nabla f(\vec{x}_k).$$

This expression generalizes Newton's method from §9.3.1, and once again it converges quadratically when $\vec{x}_0$ is near a minimum.

Newton's method can be more efficient than gradient descent depending on the objective f since it makes use of both first- and second-order information. Gradient descent has no knowledge of H_f; it proceeds analogously to walking downhill by looking only at your feet. By using H_f, Newton's method has a larger picture of the shape of f nearby.

Each iteration of gradient descent potentially requires many evaluations of f during line search. On the other hand, we must evaluate and invert the Hessian H_f during each iteration of Newton's method. These implementation differences do not affect the number of iterations to convergence but do affect the computational time taken per iteration of the two methods.

When H_f is nearly singular, Newton's method can take very large steps away from the current estimate of the minimum. These large steps are a good idea if the second-order approximation of f is accurate, but as the step becomes large the quality of this approximation can degenerate. One way to take more conservative steps is to "dampen" the change in $\vec{x}$ using a small multiplier $\gamma > 0$:

$$\vec{x}_{k+1} = \vec{x}_k - \gamma[H_f(\vec{x}_k)]^{-1}\nabla f(\vec{x}_k).$$

A more expensive but safer strategy is to do line search from $\vec{x}_k$ along the direction $-[H_f(\vec{x}_k)]^{-1}\nabla f(\vec{x}_k)$.

When H_f is not positive definite, the objective locally might look like a saddle or peak rather than a bowl. In this case, jumping to an approximate stationary point might not make sense. To address this issue, adaptive techniques check if H_f is positive definite before applying a Newton step; if it is not positive definite, the methods revert to gradient descent to find a better approximation of the minimum. Alternatively, they can modify H_f, for example, by projecting onto the closest positive definite matrix (see Exercise 9.8).

9.4.3 Optimization without Hessians: BFGS

Newton's method can be difficult to apply to complicated or high-dimensional functions $f : \mathbb{R}^n \to \mathbb{R}$. The Hessian of f is often more expensive to evaluate than f or ∇f, and each Hessian H_f is used to solve only one linear system of equations, eliminating potential savings from LU or QR factorization. Additionally, H_f has size $n \times n$, requiring $O(n^2)$ space, which might be too large. Since Newton's method deals with *approximations* of f in each iteration anyway, we might attempt to formulate less expensive second-order approximations that still outperform gradient descent.

As in our discussion of root-finding in §8.2.2, techniques for minimization that imitate Newton's method but use approximate derivatives are called *quasi-Newton methods*. They can have similarly strong convergence properties without the need for explicit re-evaluation and even inversion of the Hessian at each iteration. Here, we will follow the development of [90] to motivate one modern technique for quasi-Newton optimization.

Suppose we wish to minimize $f : \mathbb{R}^n \to \mathbb{R}$ iteratively. Near the current estimate $\vec{x}_k$ of the minimizer, we might estimate f with a quadratic function:

$$f(\vec{x}_k + \delta\vec{x}) \approx f(\vec{x}_k) + \nabla f(\vec{x}_k) \cdot \delta\vec{x} + \frac{1}{2}(\delta\vec{x})^\top B_k(\delta\vec{x}).$$

Here, we require that our approximation agrees with f to first order at $\vec{x}_k$, but we will allow the estimate of the Hessian B_k to differ from the actual Hessian of f.

Slightly generalizing Newton's method in §9.4.2, this quadratic approximation is minimized by taking $\delta\vec{x} = -B_k^{-1}\nabla f(\vec{x}_k)$. In case $\|\delta\vec{x}\|_2$ is large and we do not wish to take such a large step, we will allow ourselves to scale this difference by a step size α_k determined, e.g., using a line search procedure, yielding the iteration

$$\vec{x}_{k+1} = \vec{x}_k - \alpha_k B_k^{-1}\nabla f(\vec{x}_k).$$

Our goal is to estimate B_{k+1} by updating B_k, so that we can repeat this process.

```
function BFGS($f(\vec{x}), \vec{x}_0$)
  $H \leftarrow I_{n\times n}$
  $\vec{x} \leftarrow \vec{x}_0$
  for $k \leftarrow 1, 2, 3, \ldots$
    if $\|\nabla f(\vec{x})\| < \varepsilon$ then
      return $x^* = \vec{x}$
    $\vec{p} \leftarrow -H_k \nabla f(\vec{x})$                          ▷ Next search direction
    $\alpha \leftarrow$ COMPUTE-ALPHA($f, \vec{p}, \vec{x}, \vec{y}$)     ▷ Satisfy positive definite condition
    $\vec{s} \leftarrow \alpha\vec{p}$                                    ▷ Displacement of $\vec{x}$
    $\vec{x} \leftarrow \vec{x} + \vec{s}$                                ▷ Update estimate
    $\vec{y} \leftarrow \nabla f(\vec{x}+\vec{s}) - \nabla f(\vec{x})$    ▷ Change in gradient

    $\rho \leftarrow 1/\vec{y}\cdot\vec{s}$          ▷ Apply BFGS update to inverse Hessian approximation
    $H \leftarrow (I_{n\times n} - \rho\vec{s}\vec{y}^\top)H(I_{n\times n} - \rho\vec{y}\vec{s}^\top) + \rho\vec{s}\vec{s}^\top$
```

Figure 9.11 The BFGS algorithm for finding a local minimum of differentiable $f(\vec{x})$ without its Hessian. The function COMPUTE-ALPHA finds large $\alpha > 0$ satisfying $\vec{y}\cdot\vec{s} > 0$, where $\vec{y} = \nabla f(\vec{x}+\vec{s}) - \nabla f(\vec{x})$ and $\vec{s} = \alpha\vec{p}$.

The Hessian of f is nothing more than the derivative of ∇f, so like Broyden's method we can use previous iterates to impose a secant-style condition on B_{k+1}:

$$B_{k+1}(\vec{x}_{k+1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k).$$

For convenience of notation, we will define $\vec{s}_k \equiv \vec{x}_{k+1} - \vec{x}_k$ and $\vec{y}_k \equiv \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$, simplifying this condition to $B_{k+1}\vec{s}_k = \vec{y}_k$.

Given the optimization at hand, we wish for B_k to have two properties:

- B_k should be a symmetric matrix, like the Hessian H_f.
- B_k should be positive (semi-)definite, so that we are seeking minima of f rather than maxima or saddle points.

These conditions eliminate the possibility of using the Broyden estimate we developed in the previous chapter.

The positive definite constraint implicitly puts a condition on the relationship between $\vec{s}_k$ and $\vec{y}_k$. Pre-multiplying the relationship $B_{k+1}\vec{s}_k = \vec{y}_k$ by $\vec{s}_k^\top$ shows $\vec{s}_k^\top B_{k+1}\vec{s}_k = \vec{s}_k^\top\vec{y}_k$. For B_{k+1} to be positive definite, we must then have $\vec{s}_k \cdot \vec{y}_k > 0$. This observation can guide our choice of α_k; it must hold for sufficiently small $\alpha_k > 0$.

Assume that $\vec{s}_k$ and $\vec{y}_k$ satisfy the positive definite compatibility condition. Then, we can write down a Broyden-style optimization problem leading to an updated Hessian approximation B_{k+1}:

$$\begin{aligned} \text{minimize}_{B_{k+1}} \quad & \|B_{k+1} - B_k\| \\ \text{subject to} \quad & B_{k+1}^\top = B_{k+1} \\ & B_{k+1}\vec{s}_k = \vec{y}_k. \end{aligned}$$

For appropriate choice of norms $\|\cdot\|$, this optimization yields the well-known DFP (Davidon-Fletcher-Powell) iterative scheme.

Rather than working out the details of the DFP scheme, we derive a more popular method known as the BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm, in Figure 9.11. The BFGS algorithm is motivated by reconsidering the construction of B_{k+1} in DFP. We use B_k when minimizing the second-order approximation, taking $\delta\vec{x} = -B_k^{-1}\nabla f(\vec{x}_k)$. Based on this formula, the behavior of our iterative minimizer is dictated by the *inverse* matrix B_k^{-1}. Asking that $\|B_{k+1} - B_k\|$ is small can still imply relatively large differences between B_k^{-1} and B_{k+1}^{-1}!

With this observation in mind, BFGS makes a small alteration to the optimization for B_k. Rather than updating B_k in each iteration, we can compute its inverse $H_k \equiv B_k^{-1}$ directly. We choose to use standard notation for BFGS in this section, but a common point of confusion is that H now represents an approximate *inverse* Hessian; this is the **not** the same as the Hessian H_f in §9.4.2 and elsewhere.

Now, the condition $B_{k+1}\vec{s}_k = \vec{y}_k$ gets reversed to $\vec{s}_k = H_{k+1}\vec{y}_k$; the condition that B_k is symmetric is the same as the condition that H_k is symmetric. After these changes, the BFGS algorithm updates H_k by solving an optimization problem

$$\begin{array}{rl} \text{minimize}_{H_{k+1}} & \|H_{k+1} - H_k\| \\ \text{subject to} & H_{k+1}^\top = H_{k+1} \\ & \vec{s}_k = H_{k+1}\vec{y}_k. \end{array}$$

This construction has the convenient side benefit of not requiring matrix inversion to compute $\delta\vec{x} = -H_k\nabla f(\vec{x}_k)$.

To derive a formula for H_{k+1}, we must decide on a matrix norm $\|\cdot\|$. The *Frobenius* norm looks closest to least-squares optimization, making it likely we can generate a closed-form expression for H_{k+1}. This norm, however, has one serious drawback for modeling Hessian matrices and their inverses. The Hessian matrix has entries $(H_f)_{ij} = {}^{\partial^2 f}/_{\partial x_i \partial x_j}$. Often, the quantities x_i for different i can have different *units.* Consider maximizing the profit (in dollars) made by selling a cheeseburger of radius r (in inches) and price p (in dollars), a function $f : (\text{inches}, \text{dollars}) \to \text{dollars}$. Squaring quantities in different units and adding them up does not make sense.

Suppose we find a symmetric positive definite matrix W so that $W\vec{s}_k = \vec{y}_k$; we will check in the exercises that such a matrix exists. This matrix takes the units of $\vec{s}_k = \vec{x}_{k+1} - \vec{x}_k$ to those of $\vec{y}_k = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$. Taking inspiration from the expression $\|A\|_{\text{Fro}}^2 = \text{Tr}(A^\top A)$, we can define a *weighted Frobenius norm* of a matrix A as

$$\|A\|_W^2 \equiv \text{Tr}(A^\top W^\top A W).$$

Unlike the Frobenius norm of H_{k+1}, this expression has consistent units when applied to the optimization for H_{k+1}.

When both W and A are symmetric with columns $\vec{w}_i$ and $\vec{a}_i$, respectively, expanding the expression above shows:

$$\|A\|_W^2 = \sum_{ij} (\vec{w}_i \cdot \vec{a}_j)(\vec{w}_j \cdot \vec{a}_i).$$

This choice of norm combined with the choice of W yields a particularly clean formula for H_{k+1} given H_k, $\vec{s}_k$, and $\vec{y}_k$:

$$H_{k+1} = (I_{n\times n} - \rho_k \vec{s}_k \vec{y}_k^\top) H_k (I_{n\times n} - \rho_k \vec{y}_k \vec{s}_k^\top) + \rho_k \vec{s}_k \vec{s}_k^\top,$$

where $\rho_k \equiv {}^1/_{\vec{y}_k \cdot \vec{s}_k}$. We show in the appendix to this chapter how to derive this formula, which remarkably has no W dependence. The proof requires a number of algebraic steps but conceptually is no more difficult than direct application of Lagrange multipliers for constrained optimization (see Theorem 1.1).

The BFGS algorithm avoids the need to compute and invert a Hessian matrix for f, but it still requires $O(n^2)$ storage for H_k. The L-BFGS ("Limited-Memory BFGS") variant avoids this issue by keeping a limited history of vectors $\vec{y}_k$ and $\vec{s}_k$ and using these to apply H_k by expanding its formula recursively. L-BFGS can have *better* convergence than BFGS despite its compact use of space, since old vectors $\vec{y}_k$ and $\vec{s}_k$ may no longer be relevant and *should* be ignored. Exercise 9.11 derives this technique.

9.5 EXERCISES

9.1 Suppose $A \in \mathbb{R}^{n\times n}$. Show that $f(\vec{x}) = \|A\vec{x} - \vec{b}\|_2^2$ is a convex function. When is $g(\vec{x}) = \vec{x}^\top A\vec{x} + \vec{b}^\top \vec{x} + c$ convex?

9.2 Some observations about convex and quasiconvex functions:

(a) Show that every convex function is quasiconvex, but that some quasiconvex functions are not convex.

(b) Show that any local minimum of a continuous, strictly quasiconvex function $f : \mathbb{R}^n \to \mathbb{R}$ is also a global minimum of f. Here, *strict* quasiconvexity replaces the $\leq$ in the definition of quasiconvex functions with $<$.

(c) Show that the sum of two convex functions is convex, but give a counterexample showing that the sum of two quasiconvex functions may not be quasiconvex.

(d) Suppose $f(x)$ and $g(x)$ are quasiconvex. Show that $h(x) = \max(f(x), g(x))$ is quasiconvex.

9.3 In §9.3.1, we suggested the possibility of using parabolas rather than secants to minimize a function $f : \mathbb{R} \to \mathbb{R}$ without knowing any of its derivatives. Here, we outline the design of such an algorithm:

(a) Suppose we are given three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ with distinct x values. Show that the vertex of the parabola $y = ax^2 + bx + c$ through these points is given by:

$$x = x_2 - \frac{(x_2 - x_1)^2(y_2 - y_3) - (x_2 - x_3)^2(y_2 - y_1)}{2(x_2 - x_1)(y_2 - y_3) - (x_2 - x_3)(y_2 - y_1)}.$$

(b) Use this formula to propose an iterative technique for minimizing a function of one variable without using any of its derivatives.

(c) What happens when the three points in Exercise 9.3a are collinear? Does this suggest a failure mode of successive parabolic interpolation?

(d) Does the formula in Exercise 9.3a distinguish between maxima and minima of parabolas? Does this suggest a second failure mode?

9.4 Show that a strictly convex function $f : [a, b] \to \mathbb{R}$ is unimodular.

9.5 We might ask how well we can expect methods like golden section search can work after introducing finite precision arithmetic. We step through a few analytical steps from [101]:

(a) Suppose we have bracketed a local minimum x^* of differentiable $f(x)$ in a small interval. Justify the following approximation in this interval:

$$f(x) \approx f(x^*) + \frac{1}{2}f''(x^*)(x - x^*)^2.$$

(b) Suppose we wish to refine the interval containing the minimum until the second term in this approximation is negligible. Show that if we wish to upper-bound the absolute value of the ratio of the two terms in Exercise 9.5a by ε, we should enforce

$$|x - x^*| < \sqrt{\frac{2\varepsilon|f(x^*)|}{|f''(x^*)|}}.$$

(c) By taking ε to be machine precision as in §2.1.2, conclude that the size of the interval in which $f(x)$ and $f(x^*)$ are indistinguishable numerically grows like $\sqrt{\varepsilon}$. Based on this observation, can golden section search bracket a minimizer within machine precision?
Hint: For small $\varepsilon > 0$, $\sqrt{\varepsilon} \gg \varepsilon$.

DH 9.6 For a convex function $f : U \to \mathbb{R}^n$, where $U \subseteq \mathbb{R}^n$ is convex and open, define a *subgradient* of f at $\vec{x}_0 \in U$ to be any vector $\vec{s} \in \mathbb{R}^n$ such that

$$f(\vec{x}) - f(\vec{x}_0) \geq \vec{s} \cdot (\vec{x} - \vec{x}_0)$$

for all $\vec{x} \in U$ [112]. The subgradient is a plausible choice for generalizing the notion of a gradient at a point where f is not differentiable. The *subdifferential* $\partial f(\vec{x}_0)$ is the set of all subgradients of f at $\vec{x}_0$.

For the remainder of this question, assume that f is convex and continuous:

(a) What is $\partial f(0)$ for the function $f(x) = |x|$?

(b) Suppose we wish to minimize (convex and continuous) $f : \mathbb{R}^n \to \mathbb{R}$, which may not be differentiable everywhere. Propose an optimality condition involving subdifferentials for a point $\vec{x}^*$ to be a minimizer of f. Show that your condition holds if and only if $\vec{x}^*$ globally minimizes f.

DH 9.7 Continuing the previous problem, the *subgradient method* extends gradient descent to a wider class of functions. Analogously to gradient descent, the subgradient method performs the iteration

$$\vec{x}_{k+1} \equiv \vec{x}_k - \alpha_k \vec{g}_k,$$

where α_k is a step size and g_k is *any* subgradient of f at $\vec{x}_k$. This method might not decrease f in each iteration, so instead we keep track of the best iterate we have seen so far, $\vec{x}_k^{\text{best}}$. We will use $\vec{x}^*$ to denote the minimizer of f on U.

In the following parts, assume that we fix $\alpha > 0$ to be a constant with no dependence on k, that f is Lipschitz continuous with constant $C > 0$, and that $\|\vec{x}_1 - \vec{x}^*\|_2 \leq B$ for some $B > 0$. Under these assumptions, we will show that

$$\lim_{k\to\infty} f(\vec{x}_k^{\text{best}}) \leq f(\vec{x}^*) + \frac{C^2}{2}\alpha,$$

a bound characterizing convergence of the subgradient method.

(a) Derive an upper bound for the error $\|\vec{x}_{k+1} - \vec{x}^*\|_2^2$ of $\vec{x}_{k+1}$ in terms of $\|\vec{x}_k - \vec{x}^*\|_2^2$, $\vec{g}_k$, α, $f(\vec{x}_k)$, and $f(\vec{x}^*)$.

(b) By recursively applying the result from Exercise 9.7a, provide an upper bound for the squared error of $\vec{x}_{k+1}$ in terms of B, α, the subgradients, and evaluations of f.

(c) Incorporate $f(\vec{x}_k^{\text{best}})$ and the bounds given at the beginning of the problem into your result, and take a limit as $k \to \infty$ to obtain the desired conclusion.

(d) Suppose we are willing to run subgradient descent for exactly k steps. Suggest a choice of α for this case; your formula for α can and should involve k.

SC 9.8 This problem will demonstrate how to project a Hessian onto the nearest positive definite matrix. Some optimization techniques use this operation to avoid attempting to minimize in directions where a function is not bowl-shaped.

(a) Suppose $M, U \in \mathbb{R}^{n\times n}$, where M is symmetric and U is orthogonal. Show that $\|UMU^\top\|_{\text{Fro}} = \|M\|_{\text{Fro}}$.

(b) Decompose $M = Q\Lambda Q^\top$, where Λ is a diagonal matrix of eigenvalues and Q is an orthogonal matrix of eigenvectors. Using the result of the previous part, explain how the positive semidefinite matrix $\bar{M}$ closest to M with respect to the Frobenius norm can be constructed by clamping the negative eigenvalues in Λ to zero.

9.9 Our derivation of the BFGS algorithm in §9.4.3 depended on the existence of a symmetric positive definite matrix W satisfying $W\vec{s}_k = \vec{y}_k$. Show that one such matrix is $W \equiv \bar{G}_k$, where $\bar{G}_k$ is the *average Hessian* [90]:

$$\bar{G}_k \equiv \int_0^1 H_f(\vec{x}_k + \tau\vec{s}_k)\,d\tau.$$

Do we ever have to compute W in the course of running BFGS?

9.10 Derive an explicit update formula for obtaining B_{k+1} from B_k in the Davidon-Fletcher-Powell scheme mentioned in §9.4.3. Use the $\|\cdot\|_W$ norm introduced in the derivation of BFGS, but with the reversed assumption $W\vec{y}_k = \vec{s}_k$.

9.11 The matrix H used in the BFGS algorithm generally is dense, requiring $O(n^2)$ storage for $f : \mathbb{R}^n \to \mathbb{R}$. This scaling may be infeasible for large n.

(a) Provide an alternative approach to storing H requiring $O(nk)$ storage in iteration k of BFGS.
Hint: Your algorithm may have to "remember" data from previous iterations.

(b) If we need to run for many iterations, the storage from the previous part can exceed the $O(n^2)$ limit we were attempting to avoid. Propose an approximation to H that uses no more than $O(nk_{\max})$ storage, for a user-specified constant $k_{\max}$.

9.12 The BFGS and DFP algorithms update (inverse) Hessian approximations using matrices of rank two. For simplicity, the *symmetric-rank-1* (SR1) update restricts changes to be rank one instead [90].

(a) Suppose $B_{k+1} = B_k + \sigma\vec{v}\vec{v}^\top$, where $|\sigma| = 1$ and $\vec{y}_k = B_{k+1}\vec{s}_k$. Show that under these conditions we must have

$$B_{k+1} = B_k + \frac{(\vec{y}_k - B_k\vec{s}_k)(\vec{y}_k - B_k\vec{s}_k)^\top}{(\vec{y}_k - B_k\vec{s}_k)^\top\vec{s}_k}.$$

(b) Suppose $H_k \equiv B_k^{-1}$. Show that H_k can be updated as

$$H_{k+1} = H_k + \frac{(\vec{s}_k - H_k\vec{y}_k)(\vec{s}_k - H_k\vec{y}_k)^\top}{(\vec{s}_k - H_k\vec{y}_k)^\top\vec{y}_k}.$$

Hint: Use the result of Exercise 8.7.

9.13 Here we examine some changes to the gradient descent algorithm for unconstrained optimization on a function f.

(a) In machine learning, the *stochastic gradient descent* algorithm can be used to optimize many common objective functions:

(i) Give an example of a practical optimization problem with an objective taking the form $f(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N} g(\vec{x}_i - \vec{x})$ for some function $g : \mathbb{R}^n \to \mathbb{R}$.

(ii) Propose a randomized approximation of ∇f summing no more than k terms (for some $k \ll N$) assuming the $\vec{x}_i$'s are similar to one another. Discuss advantages and drawbacks of using such an approximation.

(b) The "line search" part of gradient descent must be considered carefully:

(i) Suppose an iterative optimization routine gives a sequence of estimates $\vec{x}_1, \vec{x}_2, \ldots$ of the position $\vec{x}^*$ of the minimum of f. Is it enough to assume $f(\vec{x}_k) < f(\vec{x}_{k-1})$ to guarantee that the $\vec{x}_k$'s converge to a local minimum? Why?

(ii) Suppose we run gradient descent. If we suppose $f(\vec{x}) \geq 0$ for all $\vec{x}$ and that we are able to find t^* exactly in each iteration, show that $f(\vec{x}_k)$ converges as $k \to \infty$.

(iii) Explain how the optimization in 9.13(b)ii for t^* can be overkill. In particular, explain how the Wolfe conditions (you will have to look these up!) relax the assumption that we can find t^*.

9.14 Sometimes we are greedy and wish to optimize multiple objectives simultaneously. For example, we might want to fire a rocket to reach an optimal point in time *and* space. It may not be possible to carry out both tasks simultaneously, but some theories attempt to reconcile multiple optimization objectives.

Suppose we are given functions $f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})$. A point $\vec{y}$ is said to *Pareto dominate* another point $\vec{x}$ if $f_i(\vec{y}) \leq f_i(\vec{x})$ for all i and $f_j(\vec{y}) < f_j(\vec{x})$ for some $j \in \{1, \ldots, k\}$. A point $\vec{x}^*$ is *Pareto optimal* if it is not dominated by any point $\vec{y}$. Assume $f_1, \ldots, f_k$ are strictly convex functions on a closed, convex set $S \subset \mathbb{R}^n$ (in particular, assume each f_i is minimized at a unique point $\vec{x}_i^*$).

(a) Show that the set of Pareto optimal points is nonempty in this case.

(b) Suppose $\sum_i \gamma_i = 1$ and $\gamma_i > 0$ for all i. Show that the minimizer $\vec{x}^*$ of $g(\vec{x}) \equiv \sum_i \gamma_i f_i(\vec{x})$ is Pareto optimal.
Note: One strategy for multi-objective optimization is to promote $\vec{\gamma}$ to a variable with constraints $\vec{\gamma} \geq \vec{0}$ and $\sum_i \gamma_i = 1$.

(c) Suppose $\vec{x}_i^*$ minimizes $f_i(\vec{x})$ over all possible $\vec{x}$. Write vector $\vec{z} \in \mathbb{R}^k$ with components $z_i = f_i(\vec{x}_i^*)$. Show that the minimizer $\vec{x}^*$ of $h(\vec{x}) \equiv \sum_i (f_i(\vec{x}) - z_i)^2$ is Pareto optimal.
Note: This part and the previous part represent two possible *scalarizations* of the multi-objective optimization problem that can be used to find Pareto optimal points.

9.6 APPENDIX: DERIVATION OF BFGS UPDATE

In this optional appendix, we derive in detail the BFGS update from §9.4.3.* Our optimization for H_{k+1} has the following Lagrange multiplier expression (for ease of notation we take $H_{k+1} \equiv H$ and $H_k = H^*$):

$$\begin{aligned}\Lambda &\equiv \sum_{ij} (\vec{w}_i \cdot (\vec{h}_j - \vec{h}_j^*))(\vec{w}_j \cdot (\vec{h}_i - \vec{h}_i^*)) - \sum_{i<j} \alpha_{ij}(H_{ij} - H_{ji}) - \vec{\lambda}^\top (H\vec{y}_k - \vec{s}_k)\\ &= \sum_{ij} (\vec{w}_i \cdot (\vec{h}_j - \vec{h}_j^*))(\vec{w}_j \cdot (\vec{h}_i - \vec{h}_i^*)) - \sum_{ij} \alpha_{ij} H_{ij} - \vec{\lambda}^\top (H\vec{y}_k - \vec{s}_k) \text{ if we define } \alpha_{ij} = -\alpha_{ji}\end{aligned}$$

Taking derivatives to find critical points shows (for $\vec{y} \equiv \vec{y}_k, \vec{s} \equiv \vec{s}_k$):

$$\begin{aligned}0 = \frac{\partial \Lambda}{\partial H_{ij}} &= \sum_\ell 2w_{i\ell}(\vec{w}_j \cdot (\vec{h}_\ell - \vec{h}_\ell^*)) - \alpha_{ij} - \lambda_i y_j\\ &= 2\sum_\ell w_{i\ell}(W^\top (H - H^*))_{j\ell} - \alpha_{ij} - \lambda_i y_j\\ &= 2\sum_\ell (W^\top (H - H^*))_{j\ell} w_{\ell i} - \alpha_{ij} - \lambda_i y_j \text{ by symmetry of } W\\ &= 2(W^\top (H - H^*)W)_{ji} - \alpha_{ij} - \lambda_i y_j\\ &= 2(W(H - H^*)W)_{ij} - \alpha_{ij} - \lambda_i y_j \text{ by symmetry of } W \text{ and } H.\end{aligned}$$

So, in matrix form we have the following list of facts:

$$\begin{aligned}0 &= 2W(H - H^*)W - A - \vec{\lambda}\vec{y}^\top, \text{ where } A_{ij} = \alpha_{ij}\\ A^\top &= -A\\ W^\top &= W\\ H^\top &= H\\ (H^*)^\top &= H^*\\ H\vec{y} &= \vec{s}\\ W\vec{s} &= \vec{y}.\end{aligned}$$

We can achieve a pair of relationships using transposition combined with symmetry of H and W and asymmetry of A:

$$\begin{aligned}0 &= 2W(H - H^*)W - A - \vec{\lambda}\vec{y}^\top\\ 0 &= 2W(H - H^*)W + A - \vec{y}\vec{\lambda}^\top\\ \implies 0 &= 4W(H - H^*)W - \vec{\lambda}\vec{y}^\top - \vec{y}\vec{\lambda}^\top.\end{aligned}$$

*Special thanks to Tao Du for debugging several parts of this derivation.

Post-multiplying this relationship by $\vec{s}$ shows:

$$\vec{0} = 4(\vec{y} - WH^*\vec{y}) - \vec{\lambda}(\vec{y}\cdot\vec{s}) - \vec{y}(\vec{\lambda}\cdot\vec{s}).$$

Now, take the dot product with $\vec{s}$:

$$0 = 4(\vec{y}\cdot\vec{s}) - 4(\vec{y}^\top H^*\vec{y}) - 2(\vec{y}\cdot\vec{s})(\vec{\lambda}\cdot\vec{s}).$$

This shows:

$$\vec{\lambda}\cdot\vec{s} = 2\rho\vec{y}^\top(\vec{s} - H^*\vec{y}), \text{ for } \rho \equiv {}^1\!/_{\vec{y}\cdot\vec{s}}.$$

Now, we substitute this into our vector equality:

$$\begin{aligned}
\vec{0} &= 4(\vec{y} - WH^*\vec{y}) - \vec{\lambda}(\vec{y}\cdot\vec{s}) - \vec{y}(\vec{\lambda}\cdot\vec{s}) \text{ from before}\\
&= 4(\vec{y} - WH^*\vec{y}) - \vec{\lambda}(\vec{y}\cdot\vec{s}) - \vec{y}[2\rho\vec{y}^\top(\vec{s} - H^*\vec{y})] \text{ from our simplification}\\
\implies \vec{\lambda} &= 4\rho(\vec{y} - WH^*\vec{y}) - 2\rho^2[\vec{y}^\top(\vec{s} - H^*\vec{y})]\vec{y}.
\end{aligned}$$

Post-multiplying by $\vec{y}^\top$ shows:

$$\vec{\lambda}\vec{y}^\top = 4\rho(\vec{y} - WH^*\vec{y})\vec{y}^\top - 2\rho^2[\vec{y}^\top(\vec{s} - H^*\vec{y})]\vec{y}\vec{y}^\top.$$

Taking the transpose,

$$\vec{y}\vec{\lambda}^\top = 4\rho\vec{y}(\vec{y}^\top - \vec{y}^\top H^* W) - 2\rho^2[\vec{y}^\top(\vec{s} - H^*\vec{y})]\vec{y}\vec{y}^\top.$$

Combining these results and dividing by four shows:

$$\frac{1}{4}(\vec{\lambda}\vec{y}^\top + \vec{y}\vec{\lambda}^\top) = \rho(2\vec{y}\vec{y}^\top - WH^*\vec{y}\vec{y}^\top - \vec{y}\vec{y}^\top H^* W) - \rho^2[\vec{y}^\top(\vec{s} - H^*\vec{y})]\vec{y}\vec{y}^\top.$$

Now, we will pre- and post-multiply by W^{-1}. Since $W\vec{s} = \vec{y}$, we can equivalently write $\vec{s} = W^{-1}\vec{y}$. Furthermore, by symmetry of W we then know $\vec{y}^\top W^{-1} = \vec{s}^\top$. Applying these identities to the expression above shows:

$$\begin{aligned}
\frac{1}{4}W^{-1}(\vec{\lambda}\vec{y}^\top + \vec{y}\vec{\lambda}^\top)W^{-1} &= 2\rho\vec{s}\vec{s}^\top - \rho H^*\vec{y}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^* - \rho^2(\vec{y}^\top\vec{s})\vec{s}\vec{s}^\top + \rho^2(\vec{y}^\top H^*\vec{y})\vec{s}\vec{s}^\top\\
&= 2\rho\vec{s}\vec{s}^\top - \rho H^*\vec{y}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^* - \rho\vec{s}\vec{s}^\top + \vec{s}\rho^2(\vec{y}^\top H^*\vec{y})\vec{s}^\top\\
&\qquad \text{by definition of } \rho\\
&= \rho\vec{s}\vec{s}^\top - \rho H^*\vec{y}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^* + \vec{s}\rho^2(\vec{y}^\top H^*\vec{y})\vec{s}^\top.
\end{aligned}$$

Finally, we can conclude our derivation of the BFGS step as follows:

$$\begin{aligned}
0 &= 4W(H - H^*)W - \vec{\lambda}\vec{y}^\top - \vec{y}\vec{\lambda}^\top \text{ from before}\\
\implies H &= \frac{1}{4}W^{-1}(\vec{\lambda}\vec{y}^\top + \vec{y}\vec{\lambda}^\top)W^{-1} + H^*\\
&= \rho\vec{s}\vec{s}^\top - \rho H^*\vec{y}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^* + \vec{s}\rho^2(\vec{y}^\top H^*\vec{y})\vec{s}^\top + H^* \text{ from the last paragraph}\\
&= H^*(I - \rho\vec{y}\vec{s}^\top) + \rho\vec{s}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^* + (\rho\vec{s}\vec{y}^\top)H^*(\rho\vec{y}\vec{s}^\top)\\
&= H^*(I - \rho\vec{y}\vec{s}^\top) + \rho\vec{s}\vec{s}^\top - \rho\vec{s}\vec{y}^\top H^*(I - \rho\vec{y}\vec{s}^\top)\\
&= \rho\vec{s}\vec{s}^\top + (I - \rho\vec{s}\vec{y}^\top)H^*(I - \rho\vec{y}\vec{s}^\top).
\end{aligned}$$

This final expression is exactly the BFGS step introduced in the chapter.