# Chapter 8

# Unconstrained Optimization

In previous chapters, we have chosen to take a largely *variational* approach to deriving standard algorithms for computational linear algebra. That is, we define an *objective function*, possibly with constraints, and pose our algorithms as a minimization or maximization problem. A sampling from our previous discussion is listed below:

| **Problem** | **Objective** | **Constraints** |
|---|---|---|
| Least-squares | $E(\vec{x}) = \|A\vec{x} - \vec{b}\|^2$ | None |
| Project $\vec{b}$ onto $\vec{a}$ | $E(c) = \|c\vec{a} - \vec{b}\|$ | None |
| Eigenvectors of symmetric matrix | $E(\vec{x}) = \vec{x}^\top A\vec{x}$ | $\|\vec{x}\| = 1$ |
| Pseudoinverse | $E(\vec{x}) = \|\vec{x}\|^2$ | $A^\top A\vec{x} = A^\top \vec{b}$ |
| Principal components analysis | $E(C) = \|X - CC^\top X\|_{\text{Fro}}$ | $C^\top C = I_{d\times d}$ |
| Broyden step | $E(J_k) = \|J_k - J_{k-1}\|_{\text{Fro}}^2$ | $J_k \cdot (\vec{x}_k - \vec{x}_{k-1}) = f(\vec{x}_k) - f(\vec{x}_{k-1})$ |

Obviously the formulation of problems in this fashion is a powerful and general approach. For this reason, it is valuable to design algorithms that function in the absence of a special form for the energy $E$, in the same way that we developed strategies for finding roots of $f$ without knowing the form of $f$ *a priori*.

## 8.1 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 requirements on the input. It is not difficult to encounter such problems in practice; we list a few examples below.

**Example 8.1** (Nonlinear least-squares)**.** *Suppose we are given a number of pairs $(x_i, y_i)$ such that $f(x_i) \approx y_i$, and we wish to find the best approximating $f$ within a particular class. For instance, we may expect that $f$ is exponential, in which case we should be able to write $f(x) = ce^{ax}$ for some $c$ and some $a$; our job is to find these parameters. One simple strategy might be to attempt to minimize the following energy:*

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

*This form for $E$ is not quadratic in $a$, so our linear least-squares methods do not apply.*

**Example 8.2** (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, yielding a list of heights $h_i$ for each student $i$. If we have a lot of students, we might 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 $\mu$ is the mean of the distribution and $\sigma$ is the standard deviation.*

*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 probability of observing the entire set of heights observed is given by the product*

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

*A common method for estimating the parameters $\mu$ and $\sigma$ of $g$ is to maximize $P$ viewed as a function of $\mu$ and $\sigma$ with $\{h_i\}$ fixed; this is called the* maximum-likelihood estimate *of $\mu$ and $\sigma$. 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 8.3** (Geometric problems)**.** *Many geometry problems encountered in graphics and vision do not reduce to least-squares energies. For instance, suppose we have a number of points $\vec{x}_1,\ldots,\vec{x}_k \in \mathbb{R}^3$. 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 $\vec{x} \in \mathbb{R}^3$ minimizing $E$ is known as the* geometric median *of $\{\vec{x}_1,\ldots,\vec{x}_k\}$. Notice that the norm of the difference $\vec{x}-\vec{x}_i$ in $E$ is not squared, so the energy is no longer quadratic in the components of $\vec{x}$.*

**Example 8.4** (Physical equilibria, adapted from CITE)**.** *Suppose we attach an object to a set of springs; each spring is anchored at point $\vec{x}_i \in \mathbb{R}^3$ and has 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 minima of $E$ and reflect points $\vec{p}$ at which the spring forces are all balanced. Such systems of equations are used to visualize graphs $G = (V,E)$, by attaching vertices in $V$ with springs for each pair in $E$.*

## 8.2 Optimality

Before discussing how to minimize or maximize a function, we should be clear what it is we are looking for; notice that maximizing $f$ is the same as minimizing $-f$, so the minimization problem is sufficient for our consideration. For a particular $f : \mathbb{R}^n \to \mathbb{R}$ and $\vec{x}^* \in \mathbb{R}^n$, we need to derive *optimality conditions* that verify that $\vec{x}^*$ has the lowest possible value $f(\vec{x}^*)$.

Of course, ideally we would like to find *global* optima of $f$:

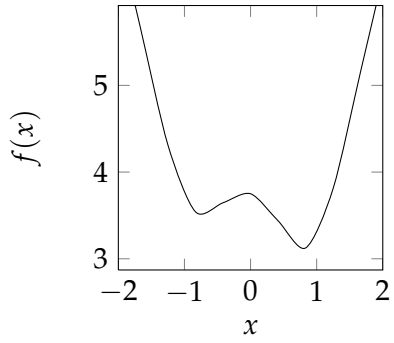


Figure 8.1: A function $f(x)$ with multiple optima.

**Definition 8.1** (Global minimum)**.** *The point $\vec{x}^* \in \mathbb{R}^n$ is a* global minimum *of $f : \mathbb{R}^n \to R$ if $f(\vec{x}^*) \leq f(\vec{x})$ for all $\vec{x} \in \mathbb{R}^n$.*

Finding a global minimum of $f$ without any information about the structure of $f$ effectively requires searching in the dark. For instance, suppose an optimization algorithm identifies the local minimum near $x = -1$ in the function in Figure 8.1. It is nearly impossible to realize that there is a second, lower minimum near $x = 1$ simply by guessing $x$ values—for all we know, there may be third even lower minimum of $f$ at $x = 1000$!

Thus, in many cases we satisfy ourselves by finding a *local* minimum:

**Definition 8.2** (Local minimum)**.** *The point $\vec{x}^* \in \mathbb{R}^n$ is a* local minimum *of $f : \mathbb{R}^n \to R$ if $f(\vec{x}^*) \leq f(\vec{x})$ for all $\vec{x} \in \mathbb{R}^n$ satisfying $\|\vec{x} - \vec{x}^*\| < \varepsilon$ for some $\varepsilon > 0$.*

This definition requires that $\vec{x}^*$ attains the smallest value in some *neighborhood* defined by the radius $\varepsilon$. Notice that local optimization algorithms have a severe limitation that they cannot guarantee that they yield the lowest possible value of $f$, as in Figure 8.1 if the left local minimum is reached; many strategies, heuristic and otherwise, are applied to explore the landscape of possible $\vec{x}$ values to help gain confidence that a local minimum has the best possible value.

### 8.2.1 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. Recall that the gradient vector $\nabla f = (\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$ points in the direction in which $f$ increases the most; the vector $-\nabla f$ points in the direction of greatest decrease. One way to see this is to recall that near a point $\vec{x}_0 \in \mathbb{R}^n$, $f$ looks like the linear function

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

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

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

When $\|\nabla f(\vec{x}_0)\| > 0$, the sign of $\alpha$ determines whether $f$ increases or decreases.

It is not difficult to formalize the above argument to show that if $\vec{x}_0$ is a local minimum, then we must have $\nabla f(\vec{x}_0) = \vec{0}$. Notice this condition is *necessary* but not *sufficient*: maxima and saddle

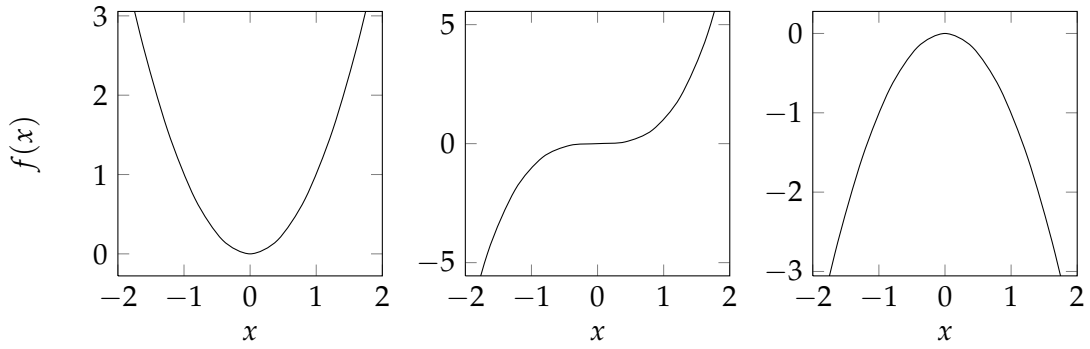


Figure 8.2: Critical points can take many forms; here we show a local minimum, a saddle point, and a local maximum.

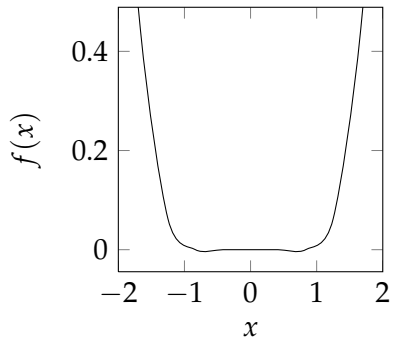


Figure 8.3: A function with many stationary points.

points also have $\nabla f(\vec{x}_0) = \vec{0}$ as illustrated in Figure 8.2. Even so, this observation about minima of differentiable functions yields a common strategy for root-finding:

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 important role in this strategy, we give the points we seek a special name:

**Definition 8.3** (Stationary point)**.** *A* stationary point *of* $f : \mathbb{R}^n \to \mathbb{R}$ *is a point* $\vec{x} \in \mathbb{R}^n$ *satisfying* $\nabla f(\vec{x}) = \vec{0}$.

That is, our strategy for minimization can be to find stationary points of $f$ and then eliminate those that are not minima.

It is important to keep in mind when we can expect our strategies for minimization to succeed. In most cases, such as those shown in Figure 8.2, 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 8.3; here, the entire interval $[-1/2, 1/2]$ is composed of stationary points, making it impossible to consider them one at a time. For the most part, we will ignore such issues as degenerate cases, but will return to them when we consider the conditioning of the minimization problem.

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

matrix:

$$H_f(\vec{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix}$$

We can add another term to our Taylor expansion of $f$ to see 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 by definition we know:

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

If $H_f$ is positive definite, then this expression shows $f(\vec{x}) \geq f(\vec{x}^*)$, and thus $\vec{x}^*$ is a local minimum. More generally, one of 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 8.3 can occur.

Checking if a matrix is positive definite can be accomplished by checking if its Cholesky factorization exists or—more slowly—by checking that all its eigenvalues are positive. Thus, when 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 simply ignore the final case and notify the user, since it is relatively unlikely.

### 8.2.2 Optimality via Function Properties

Occasionally, 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.

One property of $f$ that has strong implications for optimization is *convexity*, illustrated in Figure NUMBER:

**Definition 8.4** (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, the function is* strictly convex.

Convexity implies that if you connect in $\mathbb{R}^n$ two points 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:

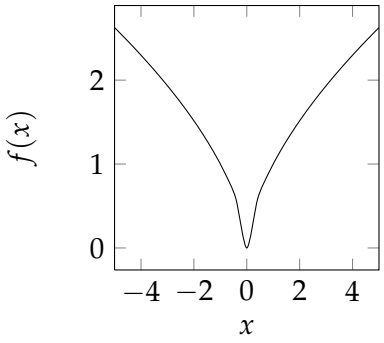


Figure 8.4: A quasiconvex function.

**Proposition 8.1.** *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 $\vec{x}^* \neq \vec{x}$ with $f(\vec{x}^*) < f(\vec{x})$. Then, for $\alpha \in (0, 1)$,

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

But taking $\alpha \to 0$ shows that $\vec{x}$ cannot possibly be a local minimum. □

This proposition and related observations show that it is possible to check if you have reached a *global* minimum of a convex function simply 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 other assumptions about $f$. For example, one weaker version 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 8.4; although it does not have the characteristic "bowl" shape of a convex function, it does have a unique optimum.

## 8.3 One-Dimensional Strategies

As in the last chapter, we will start with one-dimensional optimization of $f : \mathbb{R} \to \mathbb{R}$ and then expand our strategies to more general functions $f : \mathbb{R}^n \to \mathbb{R}$.

### 8.3.1 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 differentiable. Then, as in our derivation of Newton's method for root-finding, we can approximate:

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

The approximation on the right hand side is a parabola whose vertex is located at $x_k - f'(x_k)/f''(x_k)$. Of course, in reality $f$ is not necessarily a parabola, so Newton's method simply iterates the formula

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

This technique is easily-analyzed given the work we already have put into understanding Newton's method for root-finding in the previous chapter. In particular, an alternative way to derive the formula above comes from root-finding on $f'(x)$, since stationary points satisfy $f'(x) = 0$. Thus, 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 to ask is whether the secant method can be applied in an analogous way. Our derivation of Newton's method above finds roots of $f'$, so the secant method could be used to eliminate the evaluation of $f''$ but not $f'$; situations in which we know $f'$ but not $f''$ are relatively rare. A more suitable parallel is to replace the line segments used to approximate $f$ in the secant method with parabolas. This strategy, 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 $f(x_{k-2})$. The derivation of this technique is relatively straightforward, and it converges superlinearly.

### 8.3.2 Golden Section Search

We skipped over bisection in our parallel of single-variable root-finding techniques. There are many reasons for this omission. Our motivation for bisection was that it employed only the weakest assumption on $f$ needed to find roots: continuity. The Intermediate Value Theorem does not apply to minima in any intuitive way, however, so it appears such a straightforward approach does not exist.

It is valuable, however, to have at least one minimization strategy 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*:

**Definition 8.5** (Unimodular)**.** *A function $f : [a, b] \to \mathbb{R}$ is* unimodular *if there exists $x^* \in [a, b]$ such that $f$ is decreasing for $x \in [a, x^*]$ and increasing for $x \in [x^*, b]$.*

In other words, a unimodular function decreases for some time, and then begins increasing; no localized minima are allowed. Notice that 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:

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

- If $f(x_1) \geq f(x_0)$, then we know that $f(x) \geq f(x_0)$ for all $x \in [x_1, b]$, and thus we can discard $[x_1, b]$.

This structure suggests a potential strategy for minimization beginning with the interval $[a, b]$ and iteratively removing pieces according to the rules above.

One important detail remains, however. Our convergence guarantee for the bisection algorithm came from the fact that we could remove half of the interval in question in each iteration. We could proceed in a similar fashion, removing a *third* of the interval each time; this requires two evaluations of $f$ during each iteration at new $x_0$ and $x_1$ locations. If evaluating $f$ is expensive, however, we may wish to reuse information from previous iterations to avoid at least one of those two evaluations.

For now $a = 0$ and $b = 1$; the strategies we derive below will work more generally by shifting and scaling. In the absence of more information about $f$, we might as well make a symmetric choice $x_0 = \alpha$ and $x_1 = 1 - \alpha$ for some $\alpha \in (0, 1/2)$. Suppose our iteration removes the rightmost interval $[x_1, b]$. Then, the search interval becomes $[0, 1 - \alpha]$, and we know $f(\alpha)$ from the previous iteration. The next iteration will divide $[0, 1 - \alpha]$ such that $x_0 = \alpha(1 - \alpha)$ and $x_1 = (1 - \alpha)^2$. If we wish to reuse $f(\alpha)$ from the previous iteration, we could set $(1 - \alpha)^2 = \alpha$, yielding:

$$\begin{aligned} \alpha &= \frac{1}{2}(3 - \sqrt{5}) \\ 1 - \alpha &= \frac{1}{2}(\sqrt{5} - 1) \end{aligned}$$

The value of $1 - \alpha \equiv \tau$ above is the *golden ratio*! It allows for the reuse of one of the function evaluations from the previous iterations; a symmetric argument shows that the same choice of $\alpha$ works if we had removed the left interval instead of the right one.

The *golden section search* algorithm makes use of this construction (CITE):

1. Take $\tau \equiv \frac{1}{2}(\sqrt{5} - 1)$., and initialize $a$ and $b$ so that $f$ is unimodular on $[a, b]$.
2. Make an initial subdivision $x_0 = a + (1 - \tau)(b - a)$ and $x_1 = a + \tau(b - a)$.
3. Initialize $f_0 = f(x_0)$ and $f_1 = f(x_1)$.
4. Iterate until $b - a$ is sufficiently small:
   (a) If $f_0 \geq f_1$, then remove the interval $[a, x_0]$ as follows:
      - Move left side: $a \leftarrow x_0$
      - Reuse previous iteration: $x_0 \leftarrow x_1$, $f_0 \leftarrow f_1$
      - Generate new sample: $x_1 \leftarrow a + \tau(b - a)$, $f_1 \leftarrow f(x_1)$

   (b) If $f_1 > f_0$, then remove the interval $[x_1, b]$ as follows:
      - Move right side: $b \leftarrow x_1$
      - Reuse previous iteration: $x_1 \leftarrow x_0$, $f_1 \leftarrow f_0$
      - Generate new sample: $x_0 \leftarrow a + (1 - \tau)(b - a)$, $f_0 \leftarrow f(x_0)$

This algorithm clearly converges unconditionally and linearly. When $f$ is not globally unimodal, it can be difficult to find $[a, b]$ such that $f$ is unimodal on that interval, limiting the applications of this technique somewhat; generally $[a, b]$ is guessed by attempting to bracket a local minimum of $f$.

## 8.4 Multivariable Strategies

We continue in our parallel of our discussion of root-finding by expanding our discussion to multivariable problems. As with root-finding, multivariable problems are considerably more difficult than problems 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 differentiable. Optimization methods more similar to golden section search for non-differentiable functions are of limited applications and are difficult to formulate.

### 8.4.1 Gradient Descent

Recall from our previous discussion that $\nabla f(\vec{x})$ points in the direction of "steepest ascent" of $f$ at $\vec{x}$; similarly, the vector $-\nabla f(\vec{x})$ is the direction of "steepest descent." If nothing else, this definition guarantees that when $\nabla f(\vec{x}) \neq \vec{0}$, for small $\alpha > 0$ we must have

$$f(\vec{x} - \alpha \nabla f(\vec{x})) \leq f(\vec{x}).$$

Suppose our current estimate of the location of the minimum of $f$ is $\vec{x}_k$. Then, we might wish to choose $\vec{x}_{k+1}$ so that $f(\vec{x}_{k+1}) < f(\vec{x}_k)$ for an iterative minimization strategy. One way to simplify the search for $\vec{x}_{k+1}$ would be to use one of our one-dimensional algorithms from §8.3 on a simpler problem. In particular, 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 $\nabla f(\vec{x}_k)$. Thanks to our discussion of the gradient, we know that small $t$ will yield a decrease in $f$.

The *gradient descent* algorithm iteratively solves these one-dimensional problems to improve our estimate of $\vec{x}_k$:

1. Choose an initial estimate $\vec{x}_0$
2. Iterate until convergence of $\vec{x}_k$:
   (a) Take $g_k(t) \equiv f(\vec{x}_k - t\nabla f(\vec{x}_k))$
   (b) Use a one-dimensional algorithm to find $t^*$ minimizing $g_k$ over all $t \geq 0$ ("line search")
   (c) Take $\vec{x}_{k+1} \equiv \vec{x}_k - t^*\nabla f(\vec{x}_k)$

Each iteration of gradient descent decreases $f(\vec{x}_k)$, so the objective values converge. The algorithm only terminates when $\nabla f(\vec{x}_k) \approx \vec{0}$, showing that gradient descent must at least reach a local minimum; convergence is slow for most functions $f$, however. The line search process can be replaced by a method that simply decreases the objective a non-negligible if suboptimal amount, although it is more difficult to guarantee convergence in this case.

### 8.4.2 Newton's Method

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

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

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

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

It is easy to double check that this expression is a generalization of that in §8.3.1, 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 optimization objective $f$. Recall that each iteration of gradient descent potentially requires many evaluations of $f$ during the line search procedure. On the other hand, we must evaluate and invert the Hessian $H_f$ during each iteration of Newton's method. Notice that these factors do not affect the number of iterations but do affect runtime: this is a tradeoff that may not be obvious via traditional analysis.

It is intuitive why Newton's method converges quickly when it is near an optimum. In particular, 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.

When $H_f$ is not positive definite, however, 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. Thus, adaptive techniques might check if $H_f$ is positive definite before applying a Newton step; if it is not positive definite, the methods can revert to gradient descent to find a better approximation of the minimum. Alternatively, they can modify $H_f$ by, e.g., projecting onto the closest positive definite matrix.

### 8.4.3 Optimization without Derivatives: BFGS

Newton's method can be difficult to apply to complicated functions $f : \mathbb{R}^n \to \mathbb{R}$. The second derivative of $f$ might be considerably more involved than the form of $f$, and $H_f$ changes with each iteration, making it difficult to reuse work from previous iterations. Additionally, $H_f$ has size $n \times n$, so storing $H_f$ requires $O(n^2)$ space, which can be unacceptable.

As in our discussion of root-finding, techniques for minimization that imitate Newton's method but use approximate derivatives are called *quasi-Newton methods*. Often they can have similarly strong convergence properties without the need for explicit re-evaluation and even factorization of the Hessian at each iteration. In our discussion, we will follow the development of (CITE NOCEDAL AND WRIGHT).

Suppose we wish to minimize $f : \mathbb{R}^n \to \mathbb{R}$ using an iterative scheme. Near the current estimate $\vec{x}_k$ of the root, we might estimate $f$ with a quadratic model:

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

Notice that we have asked that our approximation agrees with $f$ to first order at $\vec{x}_k$; as in Broyden's method for root-finding, however, we will allow our estimate of the Hessian $B_k$ to vary.

This quadratic model 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 considerable step, we will allow ourselves to scale this difference by a step size $\alpha_k$, yielding

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

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

The Hessian of $f$ is nothing more than the derivative of $\nabla f$, so we can write a secant-style condition on $B_{k+1}$:

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

We will substitute $\vec{s}_k \equiv \vec{x}_{k+1} - \vec{x}_k$ and $\vec{y}_k \equiv \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$, yielding an equivalent condition $B_{k+1}\vec{s}_k = \vec{y}_k$.

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 rather than maxima or saddle points.

The symmetry condition is enough to 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 $\vec{y}_k$. In particular, premultiplying 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 $\alpha_k$; it is easy to see that it holds for sufficiently small $\alpha_k > 0$.

Assume that $\vec{s}_k$ and $\vec{y}_k$ satisfy our compatibility condition. With this in place, we can write down a Broyden-style optimization leading to a possible approximation $B_{k+1}$:

$$\begin{array}{rl} \text{minimize}_{B_{k+1}} & \|B_{k+1} - B_k\| \\ \text{such that} & B_{k+1}^\top = B_{k+1} \\ & B_{k+1}\vec{s}_k = \vec{y}_k \end{array}$$

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

Rather than work out the details of the DFP scheme, we move on to a more popular method known as the BFGS (Broyden-Fletcher-Goldfarb-Shanno) formula, which appears in many modern systems. Notice that—ignoring our choice of $\alpha_k$ for now—our second-order approximation was minimized by taking $\delta\vec{x} = -B_k^{-1}\nabla f(\vec{x}_k)$. Thus, in the end the behavior of our iterative scheme is dictated by the *inverse* matrix $B_k^{-1}$. Asking that $\|B_{k+1} - B_k\|$ is small can still imply relatively bad differences between the action of $B_k^{-1}$ and that of $B_{k+1}^{-1}$!

With this observation in mind, the BFGS scheme makes a small alteration to the above derivation. Rather than computing $B_k$ at each iteration, we can compute its inverse $H_k \equiv B_k^{-1}$ directly. Now our 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 asking that $H_k$ is symmetric. We solve an optimization

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

This construction has the nice 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 $\|\cdot\|$. As with our previous discussion, the *Frobenius* norm looks closest to least-squares optimization, making it likely we can generate a closed-form expression for $H_{k+1}$ rather than having to solve the minimization above as a subroutine of BFGS optimization.

The Frobenius norm, however, has one serious drawback for Hessian matrices. Recall that the Hessian matrix has entries $(H_f)_{ij} = \partial f_i / \partial x_j$. Often the quantities $x_i$ for different $i$ can have different *units*; e.g. consider maximizing the profit (in dollars) made by selling a cheeseburger of radius $r$ (in inches) and price $p$ (in dollars), leading to $f : (\text{inches}, \text{dollars}) \to \text{dollars}$. Squaring these different quantities 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. Such a 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 our expression $\|A\|_{\text{Fro}}^2 = \text{Tr}(A^\top A)$, we can define a *weighted Frobenius norm* of a matrix $A$ as

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

It is straightforward to check that this expression has consistent units when applied to our optimization for $H_{k+1}$. When both $W$ and $A$ are symmetric with columns $\vec{w}_i$ and $\vec{a}_i$, resp., 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 $\rho_k \equiv 1/\vec{y}\cdot\vec{s}$. We show in the Appendix to this chapter how to derive this formula.

The BFGS algorithm avoids the need to compute and invert a Hessian matrix for $f$, but it still requires $O(n^2)$ storage for $H_k$. A useful variant known as L-BFGS ("Limited-Memory BFGS") avoids this issue by keeping a limited history of vectors $\vec{y}_k$ and $\vec{s}_k$ and applying $H_k$ by expanding its formula recursively. This approach actually can have *better* numerical properties despite its compact use of space; in particular, old vectors $\vec{y}_k$ and $\vec{s}_k$ may no longer be relevant and *should* be ignored.

## 8.5 Problems

List of ideas:

- Derive Gauss-Newton
- Stochastic methods, AdaGrad
- VSCG algorithm
- Wolfe conditions for gradient descent; plug into BFGS
- Sherman-Morrison-Woodbury formula for $B_k$ for BFGS
- Prove BFGS converges; show existence of a matrix $W$
- (Generalized) reduced gradient algorithm
- Condition number for optimization

## Appendix: Derivation of BFGS Update[1]

Our optimization for $H_{k+1}$ has the following Lagrange multiplier expression (for ease of notation we take $H_{k+1} \equiv H$ and $H_k = H^*$):

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

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 $A$:

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

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:

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

[1]Special thanks to Tao Du for debugging several parts of this derivation.

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:

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

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.