Numerical Function Optimization

September 8, 2021

1 Introduction

Machine learning is optimization: A risk function defines the discrepancy between actual and desired behavior of the function being learned, and optimization methods find function parameters that reduce the risk over the inputs and outputs in the training set.

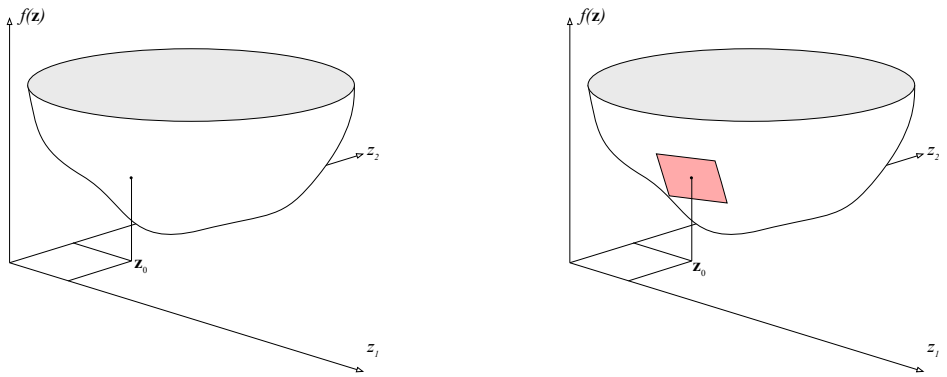
The term "optimization" is meant to subsume both minimization and maximization. However, maximizing the scalar function $f(\mathbf{z})$ is the same as minimizing its negative $-f(\mathbf{z})$, so we consider optimization and minimization to be essentially synonyms. Usually, one is after *global* minima. However, global minima are hard to find, since they involve a universal quantifier: $\mathbf{z}^*$ is a global minimum of f if *for every* $\mathbf{z}$ we have $f(\mathbf{z}) \geq f(\mathbf{z}^*)$, and it is typically infeasible to check all possible $\mathbf{z}$. Global minimization techniques like simulated annealing have been proposed, but their convergence properties depend very strongly on the problem at hand. In this chapter, we consider local minimization: we pick a starting point $\mathbf{z}_0$, and we keep moving to points $\mathbf{z}_1, \mathbf{z}_2, \ldots$ as long as the value of f at those points keeps decreasing. We stop when we cannot go down any further because we have reached a local minimum.

Local minimization may get us close to the *global* minimum $\mathbf{z}^*$ if we know how to pick a $\mathbf{z}_0$ that is close to $\mathbf{z}^*$. This occurs frequently in feedback systems. In these systems, we start at a local (or even a global) minimum. The system then evolves and escapes from the minimum. As soon as this occurs, a control signal is generated to bring the system back to the minimum. Because of this immediate reaction, the old minimum can often be used as a starting point $\mathbf{z}_0$ when looking for the new minimum, that is, when computing the required control signal. More formally, we reach the correct minimum $\mathbf{z}^*$ as long as the initial point $\mathbf{z}_0$ is in the *basin of attraction* of $\mathbf{z}^*$, defined as the largest neighborhood of $\mathbf{z}^*$ in which $f(\mathbf{z})$ is convex.

If a good $\mathbf{z}_0$ is not available, one may have to be content with a local minimum $\hat{\mathbf{z}}$. After all, $\hat{\mathbf{z}}$ is always at least as good, and often much better, than $\mathbf{z}_0$. A compromise that may help when the domain of f has few dimensions is to pick several values for $\mathbf{z}_0$ (perhaps at random), compute the corresponding values for $\hat{\mathbf{z}}$, and then pick the one with the lowest value $f(\hat{\mathbf{z}})$. However, the curse of dimensionality makes this approach futile in many dimensions.

1.1 First Order Optimization Methods

When the function $f(\mathbf{z}) : \mathbb{R}^m \to \mathbb{R}$ is differentiable in the vector $\mathbf{z}$, we can compute its gradient ∇f. Pick some vector $\mathbf{z}_0 \in \mathbb{R}^m$ as a starting point for searching for a value $\hat{\mathbf{z}}$ of $\mathbf{z}$ which, if not optimal, at least yields a value $f(\hat{\mathbf{z}})$ that is smaller than $\mathbf{z}_0$. Figure 1 (a) sketches a function $f(\mathbf{z})$ for $\mathbf{z} \in \mathbb{R}^2$ and a starting point $\mathbf{z}_0$.



(a) (b)

Figure 1: (a) A function $f : \mathbb{R}^2 \to \mathbb{R}$ and an initial point $\mathbf{z}_0$. (b) The shaded quadrilateral is a patch on the plane represented by the first-order Taylor series expansion of f around $\mathbf{z}_0$. Any step in the steepest downhill direction will reduce the value of the Taylor approximation to f. If the length of the step is chosen carefully, the step might reduce the value of f as well, and lead to a new point $\mathbf{z}_1$ (not shown) where f has a lower value than at $\mathbf{z}_0$.

The gradient of f at $\mathbf{z}_0$ can be used to approximate f with its first-order Taylor approximation

$$f(\mathbf{z}) \approx g_1(\mathbf{z}) = f(\mathbf{z}_0) + [\nabla f(\mathbf{z}_0)]^T(\mathbf{z} - \mathbf{z}_0) .$$

This function $g_1(\mathbf{z})$ represents a plane when $m = 2$ (Figure 1 (b)) and a hyperplane more generally, and provides some information about the shape of the graph of f in the vicinity of $\mathbf{z}_0$. Loosely speaking, $g_1(\mathbf{z})$, and in particular the gradient $\nabla\mathbf{z}_0$, tells us which way is uphill, that is, in what direction the value of g_1 (and therefore, in a small neighborhood of $\mathbf{z}_0$, the value of f) increases most rapidly. Then, a small step in the opposite direction, $-\nabla\mathbf{z}_0$, may lead to a new point $\mathbf{z}_1$ such that

$$f(\mathbf{z}_1) < f(\mathbf{z}_0)$$

as long as the size of the step is chosen judiciously. We will see later how to choose good step sizes. This procedure, called a *first-order descent method* can be repeated as long as progress occurs, that is, as long as the value of f decreases, or until the changes become too small to be useful.

If also the Hessian $H(\mathbf{z}_0)$ of f at $\mathbf{z}_0$ is known,

$$H(\mathbf{z}_0) = \left[\begin{array}{ccc} \frac{\partial^2 f}{\partial z_1^2} & \cdots & \frac{\partial^2 f}{\partial z_1 \partial z_m} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial z_m \partial z_1} & \cdots & \frac{\partial^2 f}{\partial z_m^2} \end{array} \right]_{\mathbf{z}=\mathbf{z}_0} ,$$

then so-called *second-order descent methods* can be used, of which *Newton's method* is the archetype. Second-order methods are viable only when the cost of computing the Hessian is bearable. This is typically not the case in deep learning, where the number m of unknowns is too large. While Newton's method is applicable to a few of the smaller optimization problems discussed in this course, its description and analysis are left to an Appendix to keep this section simple, and are optional reading. First-order methods can be used, at the cost of some inefficiency, even for small problems.

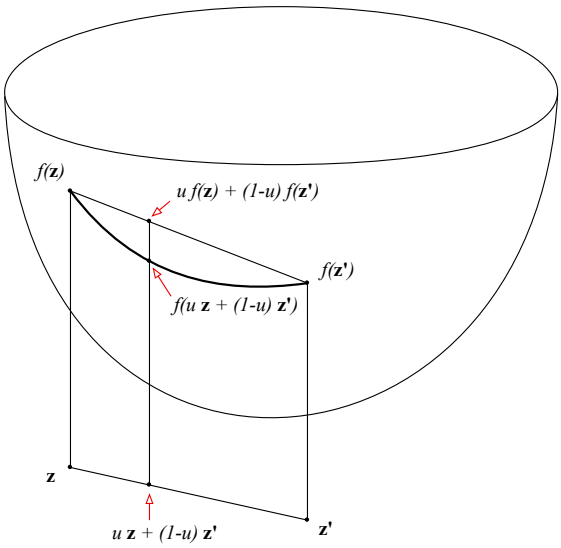


Figure 2: A function $f(\mathbf{z})$ is weakly convex if its graph is never above the line segment through any of its two points.

1.2 Gradient, Hessian, and Convexity

Whether a descent method, regardless of its order, converges at all or, if it does, whether it converges to a global minimum depends on many factors. A particularly favorable situation occurs when f is a convex function of $\mathbf{z}$, in which case strong guarantees can typically be given.

A function $f(\mathbf{z})$ is convex (everywhere) if for every pair of points $\mathbf{z}, \mathbf{z}'$ in the domain of f the graph of f is never above the segment that joins the two graph points $(\mathbf{z}, f(\mathbf{z}))$ and $(\mathbf{z}', f(\mathbf{z}'))$:

$$f(u\mathbf{z} + (1-u)\mathbf{z}') \;\leq\; uf(\mathbf{z}) + (1-u)f(\mathbf{z}') \quad \text{for all } u \in [0,1]$$

(see Figure 2).

To check that a scalar function $f \;:\; \mathbb{R} \to \mathbb{R}$ is (weakly) convex you check that its second derivative, which is the derivative of the derivative, is nonnegative everywhere. For multivariate functions,

$$f \;:\; \mathbb{R}^m \to \mathbb{R}\,,$$

the idea is analogous: You check that its Hessian H, which is the Jacobian of the gradient, is positive semidefinite everywhere. This means that for every $\mathbf{z} \in \mathbb{R}^m$ we have

$$\mathbf{z}^T H \mathbf{z} \;\geq\; 0\,.$$

If f is convex everywhere, any minimum is a global minimum, and therefore if a descent method finds a minimum (that is, a point $\mathbf{z}$ for which $\nabla f(\mathbf{z}) = \mathbf{0}$), that point is also a *global* minimum.

And now for some more detail about first-order descent methods.

2 Local Minimization and Gradient Descent

Suppose that we want to find a local minimum for the scalar function f of the vector variable $\mathbf{z}$, starting from an initial point $\mathbf{z}_0$. Picking an appropriate $\mathbf{z}_0$ is often crucial but also very problem-dependent, so we do not discuss this choice here. We then start from $\mathbf{z}_0$, and we go downhill. At every step of the way, we must make the following decisions:

- In what direction to proceed.
- How long a step to take in that direction.
- When to stop.

Correspondingly, most minimization algorithms have the following structure, with $\mathbf{z}_0$ given:

$k = 0$
while $\mathbf{z}_k$ is not a minimum
  compute step direction $\mathbf{p}_k$
  compute step size α_k
  $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_k \mathbf{p}_k$
  $k = k + 1$
end.

Different algorithms differ in how each of these instructions is performed, and the two computations regarding the step, direction and size, are sometimes performed together as a single computation that yields the step $\mathbf{s}_k = \alpha_k \mathbf{p}_k$. We will consider *gradient descent*, in which the direction of descent is along the negative of the gradient:

$$\mathbf{p}_k = -\nabla f(\mathbf{z}_k) \; .$$

In the next Section, we describe how to perform gradient descent when f is an average of a very large number of terms, a situations that occurs often in deep learning. We then turn to the questions of step-size selection and conditions on when to stop iterating.

2.1 Stochastic Gradient Descent

In machine learning, the functions f to be minimized are often very expensive to compute. However, they often also have a special structure, in that they are averages of large numbers of terms, because the risk is an average loss over the training set:

$$f(\mathbf{z}) = \frac{1}{N} \sum_{n=1}^{N} \phi_n(\mathbf{z}) \; . \tag{1}$$

In that case, rather than reducing the value of all of $f(\mathbf{z})$ with each step, one can reduce the value of a random sub-average of $f(\mathbf{z})$. Specifically, partition the set of indices $B = \{1, \ldots, N\}$ into J random subsets B_j each of about equal size[1]. The set B is called the *batch*, ad each subset B_j is called a *mini-batch*. Then, the terms in the summation that defines f can be grouped as follows:

$$f(\mathbf{z}) = \frac{1}{N} \sum_{j=1}^{J} \sum_{n \in B_j} \phi_n(\mathbf{z}) \; . \tag{2}$$

[1] Unless N is a multiple of J, one of the subsets will be smaller or larger than all the others.

Instead of taking a step along the exact gradient $\nabla f(\mathbf{z})$, the method of *Stochastic Gradient Descent (SGD) estimates* the gradient $\nabla f_j(\mathbf{z})$ from the j-th mini-batch:

$$f_j(\mathbf{z}) = \frac{1}{|B_j|} \sum_{n \in B_j} \phi_n(\mathbf{z})$$

at every step. In this expression, $|B_j|$ denotes the cardinality of B_j.

To the extent that the mini-batch B_j is a representative sample of the entire batch B, the mini-batch average f_j is an estimate of the entire average f, and therefore so are the gradients:

$$f(\mathbf{z}) \approx f_j(\mathbf{z}) \quad \Rightarrow \quad \nabla f(\mathbf{z}) \approx \nabla f_j(\mathbf{z}) \; .$$

The step is then

$$\mathbf{z}_{k+1} - \mathbf{z}_k = \alpha_k \nabla f_j(\mathbf{z}_k) \tag{3}$$

rather than

$$\mathbf{z}_{k+1} - \mathbf{z}_k = \alpha_k \nabla f(\mathbf{z}_k) \; . \tag{4}$$

Since the mini-batch gradient $\nabla f_j(\mathbf{z})$ approximates the full gradient $\nabla f(\mathbf{z})$, the mini-steps are in the right direction *on average*, even if each step may be somewhat wrong (they are wrong especially when the mini-batches are small).

One can make this statement more quantitative in the following sense [1]: Let $f(\mathbf{z}_k)$ be the value actually obtained after step k. While this value is given and therefore deterministic, the subsequent value depends on the data in the mini-batch j, and is therefore viewed as a random variable. The expectation of the change in function value after the next step is then

$$\begin{aligned} \mathbb{E}[f(\mathbf{z}_{k+1})] - f(\mathbf{z}_k) &= \mathbb{E}[f(\mathbf{z}_k - \alpha_k \nabla f_j(\mathbf{z}_k))] - f(\mathbf{z}_k) \\ &= \mathbb{E}[f(\mathbf{z}_k) - (\nabla f(\mathbf{z}_k))^T \alpha_k \nabla f_j(\mathbf{z}_k))] - f(\mathbf{z}_k) + O(\alpha_k^2) \\ &= -\alpha \|\nabla f(\mathbf{z}_k)\|^2 + O(\alpha_k^2) < 0 \end{aligned}$$

for sufficiently small step sizes α_k. The second equality above is obtained by taking the first-order Taylor series expansion of f around $\mathbf{z}_k$, and the last equality assumes that the approximation of $\nabla f_j(\mathbf{z}_k)$ with $\nabla f(\mathbf{z}_k)$ is correct to first order (which it is on average). This argument shows that *the value of $f(\mathbf{z})$ decreases, on average, at every step as long as the steps are sufficiently small.*

It can also be shown that SGD converges to a local minimum as long as the step lengths α_k decrease fast enough but not too fast with k, in the sense that they meet the following conditions [4]:

$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k = \infty \; . \tag{5}$$

These conditions ensure *asymptotic* convergence (that is, convergence when $k \to \infty$). However, in deep learning, optimization is often stopped before full convergence, as we will see, so these conditions are moot in that context.

Processing all data once is called an *epoch* of optimization, which proceeds through several epochs until convergence.

The *mini-step* (3) is greedy, in that reducing one mini-batch average $f_j(\mathbf{z})$ might even increase some other mini-batch average $f_i(\mathbf{z})$.

The size of the mini-batch affects the variance of the gradient $\nabla f_j(\mathbf{z}_k)$ of the mini-batch optimization target (2) as an estimate of the gradient $\nabla f(\mathbf{z}_k)$ of the full optimization target (1). Estimation variance decreases as the mini-batch size increases, but the storage cost increases with mini-batch size. In deep learning, optimization is often performed on Graphical Processing Units (GPUs), and the size of the mini-batch is constrained by the amount of memory available in the GPU. Thus, one typically selects the largest mini-batch that will fit in GPU memory.

2.2 Step Size

To complete the gradient descent algorithm, stochastic (step (3)) or not (step (4)), we need to specify two more of its aspects: (i) How to determine the step size α_k so at to reach a local minimum of $f(\mathbf{z})$ along the direction of $\mathbf{p}_k$; and (ii) how to check whether a local minimum of f (in any direction) has been reached, so that we can stop. We discuss the first aspect in this Section, and leave the second aspect for the next Section.

Once the direction of descent has been determined, the optimization problem becomes one-dimensional, and amounts to determining how far to move in the direction of the negative gradient. Specifically, let

$$h(\alpha) = f(\mathbf{z}_k + \alpha \mathbf{p}_k) \tag{6}$$

be the scalar function of one variable (that is, $h : \mathbb{R} \to \mathbb{R}$) that is obtained by restricting the function f to the line through the current point $\mathbf{z}_k$ and in the direction of $\mathbf{p}_k = -\nabla f(\mathbf{z}_k)$. We need to find a positive value α_k of α for which $h(\alpha_k) < h(0)$, so that if we let

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_k \mathbf{p}_k$$

we have

$$\mathbf{z}_{k+1} < \mathbf{z}_k \ .$$

We consider the following strategies:

- Move by an arbitrary, predetermined step $\alpha_k = \bar{\alpha} > 0$. This strategy requires no effort for determining α_k, but it may seem *prima facie* a losing proposition: If $\bar{\alpha}$ is too small it may take too many steps to reach a minimum of f. If $\bar{\alpha}$ is too large the sequence of points $\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, \ldots$ may skip over local minima and never converge to one. However, recall that $\bar{\alpha}$ multiplies the negative gradient vector $\mathbf{p}_k = -\nabla f_k$. If the search for a minimum ever gets near one, the gradient is small (because it is zero *at* the minimum), and the step sizes become smaller and smaller as a consequence.

- Move by a decreasing sequence of step sizes, for instance, $\alpha_k = 1/(k+1)$. This strategy satisfies the conditions (5), so this is a favorite strategy in proofs about asymptotic properties.

- Move by a linear combination of $\bar{\alpha}\mathbf{p}_k$ (with a predetermined $\bar{\alpha}$ or a decreasing sequence α_k) and the direction of the previous step $\mathbf{v}_k = \mathbf{z}_k - \mathbf{z}_{k-1}$. This is called the *momentum* method: Just as a moving particle with momentum[2] tends to keep moving with that momentum (Newton's first law), the search for a minimum tends to keep searching in a direction that was promising a step ago and may still be promising now. This method helps accelerate descent when the gradient of f is small, as is the case in plateaus and around shallow minima, and is discussed in somewhat more detail below.

[2]Momentum is mass times velocity.

- Find a local minimum of $h(\alpha)$. If α' is not a local minimum, then one can think of reaching a different value α'' of α so that $h(\alpha'') < h(\alpha')$, so why stop at α'? This strategy is called *line search*, a simple version of which is discussed in more detail below. Of course, even if α_k is a local minimum of h, it need not be a minimum of f, as it may be possible to move downhill from the new point $\mathbf{z}_{k+1} = \mathbf{z}_k - \alpha_k \mathbf{p}_k$ in a direction different from $\mathbf{p}_k$.

Stochastic Gradient Descent requires a decreasing α_k if full convergence is desired, since the conditions (5) must be met. A simple choice in that case is

$$\alpha_k = 1/(k+1)$$

as mentioned above. In deep learning, however, SGD is typically stopped before full convergence, for reasons that will become clear later on. The schedule for α_k is then determined by trial and error on sample problems, with α initially large and decreasing over time. The intent of the schedule is to strike a good balance between sufficiently fast progress (which requires a larger α_k) and the ability of the algorithm to descend into narrow valleys towards a minimum (which requires a smaller α_k). Figure 3 illustrates.

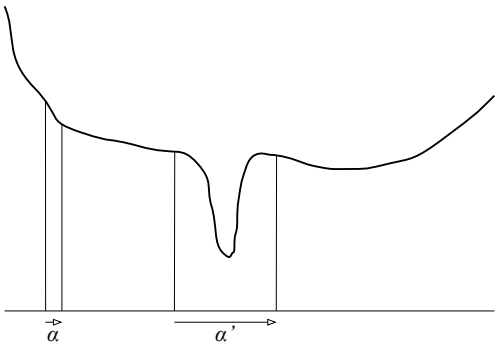


Figure 3: A small step (α) may make overly small progress towards a minimum. A large step (α') may skip over a minimum in a narrow valley.

Momentum The *momentum method* [3, 6] starts from an initial value $\mathbf{z}_0$ chosen at random and iterates as follows:

$$\begin{aligned} \mathbf{v}_{k+1} &= \mu_k \mathbf{v}_k - \alpha_k \nabla f(\mathbf{z}_k) \\ \mathbf{z}_{k+1} &= \mathbf{z}_k + \mathbf{v}_{k+1} \ . \end{aligned}$$

The vector $\mathbf{v}_{k+1}$ is the *step* or *velocity* that is added to the old value $\mathbf{z}_k$ to compute the new value $\mathbf{z}_{k+1}$. The time-dependent scalar $\mu_k \in [0, 1]$ is the *momentum coefficient*. Standard gradient descent occurs when $\mu_k = 0$. Greater values of μ_k encourage steps in a consistent direction (since the new velocity $\mathbf{v}_{k+1}$ has a greater component in the direction of the old velocity $\mathbf{v}_k$ than if no momentum were present), and these steps accelerate descent when the gradient of $f(\mathbf{z})$ is small, as is the case on plateaus and around shallow minima. The value of μ_k is often varied according to some schedule like the one in Figure 4. The rationale for the increasing values over time is that momentum is more useful in later stages, in which the gradient magnitude is very small as $\mathbf{z}_k$ approaches the minimum. The step size α_k is either fixed or chosen as described above for SGD.

$$\mu_k = \min\left(\mu_{\max}, 1 - \frac{1}{2}\frac{1}{\lfloor \frac{t}{250} \rfloor + 1}\right)$$

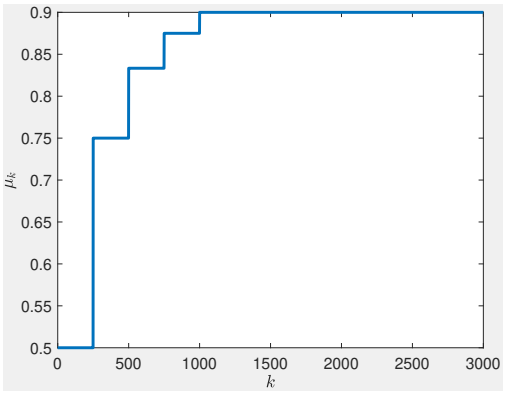


Figure 4: A possible schedule [6] for the momentum coefficient μ_k.

Line Search is a method to find a step α_k that reaches a minimum in the direction $\mathbf{p}_k$. Here is how it works, first at a high level, then in more detail. Line search first determines two points a, c that *bracket* the desired step size α_k where h achieves a minimum, in the sense that $a \leq \alpha_k \leq c$. It then picks a point b such that

$$a < b < c \quad \text{and} \quad h(b) \leq h(a) \quad \text{and} \quad h(b) \leq h(c) \tag{7}$$

(as we will see soon, such a point must exist). A triple (a, b, c) that satisfies these properties is called a *bracketing triple*. To determine α_k, line search computes the midpoint u of the wider of the two intervals $[a, b]$ or $[b, c]$ and then finds a new, narrower bracketing triple that involves u and two points out of a, b, c. This process is repeated, each time shrinking the bracketing triple, until it is so small that it pins α_k down to any desired accuracy.

More specifically, in the first step (bracketing), we can set $a = \alpha_0 = 0$, corresponding through equation (6) to the starting point $\mathbf{z}_k$. A point c that is on the opposite side of a minimum with respect to a can be found by increasing α through values $\alpha_1,\ \alpha_2, \ldots$ until $h(\alpha_i)$ is greater than $h(\alpha_{i-1})$: When doing so, we initially go downhill, because we are moving in the direction of the negative gradient, so once we start going uphill we must have passed a minimum, so we can set $c = \alpha_i$. This is because the derivative of h at a is negative, so the function is initially decreasing, but it is increasing between α_{i-1} and $\alpha_i = c$, so there must be a minimum somewhere between a and c. Of course, unless h happens to be convex between α_1 and α_i, we can only say that we will find *some* local minimum, not the lowest or the closest to α_1. This is an unavoidable fact of life, unless we know something about the underlying function f. The point b can be chosen as α_{i-1}, a point where by construction h has a value lower than both a and c.

So now we have a bracketing triple (a, b, c) that satisfies conditions (7), with a minimum of h somewhere between a and c. Line search proceeds by shrinking this bracketing triple until the difference $c - a$ is smaller than any positive desired accuracy in determining α_k. Each step of shrinking is done as follows:

if $b - a > c - b$
    $u = (a + b)/2$
    if $h(u) > h(b)$

```
        (a, b, c) = (u, b, c)
    otherwise
        (a, b, c) = (a, u, b)
    end
otherwise
    u = (b + c)/2
    if h(u) > h(b)
        (a, b, c) = (a, b, u)
    otherwise
        (a, b, c) = (b, u, c)
    end
end.
```

It is easy to see that in each case the triple (a, b, c) is a bracketing triple (properties (7) hold throughout), and therefore there is a minimum somewhere between a and c.

With the possible exception of the very first split, each split reduces the size of the bracketing interval by at least a quarter. To see this, note that the extent of the reduction is half the size of the longer interval. Therefore, the smallest reduction occurs when the intervals $[a, b]$ and $[b, c]$ are equal in size, because then the longer interval is as short as it gets. In that case, u is the half-point of $[b, c]$. If the new triple is (b, u, c), then its size is half that of the original triple (a, b, c). If the new triple is (a, b, u), then its size is three quarters of the original. The latter is the worst case, and the reduction is by 25 percent, as promised.

A slightly better performance can be achieved by placing point u not in the middle of the longer segment, as done in the code above, but rather in such a way that at each iteration the ratio

$$r(a, b, c) = \frac{\max(b - a, c - b)}{\min(b - a, c - b)}$$

between the length of the longer segment and that of the shorter segment in the bracketing triple is always the same, and equal to

$$w = \frac{1 + \sqrt{5}}{2} \approx 1.618 \,,$$

a number called the *golden ratio*. Appendix A shows that this can be achieved by an appropriate initial placement of b and subsequent placement of u. With this strategy, the ratio between the length of the new bracketing triple and the old is always[3]

$$\frac{w}{1 + w} = w - 1 \approx 0.618$$

rather than somewhere between 1/2 and 3/4. With this strategy, line search is called the *golden ratio line search*. Either way, however, the bracketing triple shrinks exponentially fast.

Variants of Line Search Either way, line search as described above sometimes takes unacceptably many iterations to find a minimum. Some variants of the technique then merely search for a value α_k where $h(\alpha_k)$ is "sufficiently lower" than $h(0)$, rather than searching for a local minimum. Different definitions of "sufficiently lower" yield different variants of line search [2]. These variants are beyond the scope of these notes.

[3]The equality $w/(1 + w) = w - 1$ holds only for this particular value of w, not in general.

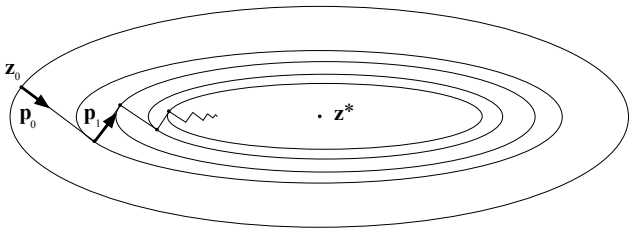


Figure 5: Trajectory of gradient descent on a convex paraboloid.

2.3 Termination Check

One criterion to check whether we are done with minimization is to verify whether the value of $f(\mathbf{z}_k)$ has decreased significantly from $f(\mathbf{z}_{k-1})$: If it has not, very little descent is occurring, and we may want to stop. Another is to check whether $\mathbf{z}_k$ is significantly different from $\mathbf{z}_{k-1}$ (*ditto*). Close to the minimum, the derivatives of f are close to zero, so $|f(\mathbf{z}_k) - f(\mathbf{z}_{k-1})|$ may be very small but $\|\mathbf{z}_k - \mathbf{z}_{k-1}\|$ may still be relatively large. Thus, the check on $\mathbf{z}_k$ is more stringent, and therefore preferable in most cases. This is because one is usually interested in the value of $\hat{\mathbf{z}}$ (the parameters of a predictor), rather than in that of $f(\hat{\mathbf{z}})$ (the residual risk after optimization). In summary, the steepest descent algorithm can be stopped when

$$\|\mathbf{z}_k - \mathbf{z}_{k-1}\| < \epsilon$$

where the positive constant ϵ is provided by the user.

3 Is Gradient Descent with Line Search the Best?

Combining gradient descent with line search seems a winning solution, since the direction of steepest descent would seem to be best, and line search finds at least a local minimum along that direction. However, this combination turns out to be quite rarely optimal, as we now show.

Figure 5 shows the gradient-descent trajectory $\mathbf{z}_k$ superimposed on a set of isocontours of a simple paraboloid (a quadratic function) $f(\mathbf{z})$ for a two-dimensional search space, that is, when $\mathbf{z}$ is a two-dimensional vector.

A convex paraboloid (whether in two or more dimensions) has equation

$$f(\mathbf{z}) = c + \mathbf{a}^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T Q\mathbf{z} \tag{8}$$

where Q is a symmetric, positive definite matrix. *Positive definite* means that for every nonzero $\mathbf{z}$ the quantity $\mathbf{z}^T Q\mathbf{z}$ is positive. In this case, the graph of $f(\mathbf{z})$ is a plane $c + \mathbf{a}^T\mathbf{z}$ plus a paraboloid with its bottom at the origin $\mathbf{0}$. Appendix B shows that adding a linear term $\mathbf{a}^T\mathbf{z}$ (and a constant c) to a paraboloid $\frac{1}{2}\mathbf{z}^T Q\mathbf{z}$ merely shifts the bottom of the paraboloid, both in position ($\mathbf{z}^*$ rather than $\mathbf{0}$) and value ($c - \frac{1}{2}\mathbf{z}^{*T} Q\mathbf{z}^*$ rather than zero). Adding the linear term does not "warp" or "tilt" the shape of the paraboloid in any way.

Of course, if f were this simple, no descent methods would be necessary, because the minimum of f can be found by setting its gradient to zero:

$$\frac{\partial f}{\partial \mathbf{z}} = \mathbf{a} + Q\mathbf{z} = 0$$

so that the minimum $\mathbf{z}^*$ is the solution to the linear system

$$Q\mathbf{z} = -\mathbf{a} \; . \tag{9}$$

Since Q is positive definite, it is also invertible (why?), and the solution $\mathbf{z}^*$ is unique. However, understanding the behavior of minimization algorithms in this simple case is crucial in order to establish the convergence properties of these algorithms for more general functions. This is because all smooth functions can be approximated by paraboloids in a sufficiently small neighborhood of any point (second-order Taylor approximation).

Let us therefore assume that we minimize f as given in equation (8), and that at every step we choose the direction of the negative gradient:

$$\mathbf{p}_k = -\left.\frac{\partial f}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_k} = -\nabla f(\mathbf{z}_k) \; .$$

Let us further assume that the length of the step is determined by line search, so that the (unique) minimum of the restriction of f along the steepest-descent direction is reached (recall that if $f(\mathbf{z})$ is convex so is its restriction $h(\alpha) = f(\mathbf{z}_k - \alpha\mathbf{p}_k)$.

Looking at Figure 5, we see that there is one good, but very precarious case in which convergence to the true solution $\mathbf{z}^*$ occurs blindingly fast, in a single step. This happens when the starting point $\mathbf{z}_0$ is at one apex (tip of either axis) of an isocontour ellipse. In that case, the gradient points exactly towards $\mathbf{z}^*$, and one run of line search will lead to the minimum $\mathbf{z}^*$.

In all other cases, the search line in the direction $\mathbf{p}_k$ of the gradient, which is orthogonal to the isocontour at $\mathbf{z}_k$, will not pass through $\mathbf{z}^*$. The minimum of f along that line is tangent to some other, lower isocontour (or else it would not be a minimum, local or otherwise). The next search direction $\mathbf{p}_{k+1}$ is orthogonal to the latter isocontour (that is, parallel to the gradient at the point of tangency $\mathbf{z}_{k+1}$). Thus, at every step the gradient descent trajectory is forced to make a ninety-degree turn: Step k is tangent to the new isocontour, and step $k+1$ is perpendicular to it. If isocontours were circles ($\sigma_1 = \sigma_n$) centered at $\mathbf{z}^*$, then the first turn would make the new direction point to $\mathbf{z}^*$, and minimization would get there in just one more step. The more elongated the isocontours, the farther away a line orthogonal to an isocontour passes from $\mathbf{z}^*$, and the more steps are required for convergence. This elongation is measured by the so-called *condition number* $\kappa(Q)$ of Q, defined in Appendix B, which contains a more quantitative analysis of the convergence speed of gradient descent.

Thus, the directions of gradient descent are typically poor directions, even when combined with line search, with the only exceptions of starting at one of the axes of an isocontour ellipsoid or moving among hyperspheres rather than ellipsoids. Other than in these rare lucky cases, gradient descent keeps zig-zagging its way in small steps towards the solution, and the number of steps can be very large.

Nonetheless, gradient descent is a popular optimization method because of its low cost per iteration, at the expense of a large number of iterations (the next Section discusses convergence speed). In some situations, the technique of *preconditioning* can improve things by deforming the function $f(\mathbf{z})$ so that its isocontours look closer to hyperspheres. The method of conjugate gradients, described in Appendix D, is based on gradient descent, and achieves performance that is often close to that of Newton's method without computing Hessians. Both techniques are beyond the scope of this introductory note.

3.1 Sub-Gradients

Another reason for the popularity of gradient descent lies in the fact that this method and its variants (SGD, conjugate gradients, and more) can be applied even to *convex* functions that have discontinuities in the derivatives. For example, we will use the *hinge function* (Figure 6 left)

$$\rho(z) = \max\{0, z\}$$

when we study support vector machines. This function is also called the Rectified Linear Unit (ReLU) in the context of deep learning. This function is weakly convex everywhere. Its derivative is zero for $z < 0$ and one for $z > 0$, but is undefined at $z = 0$.

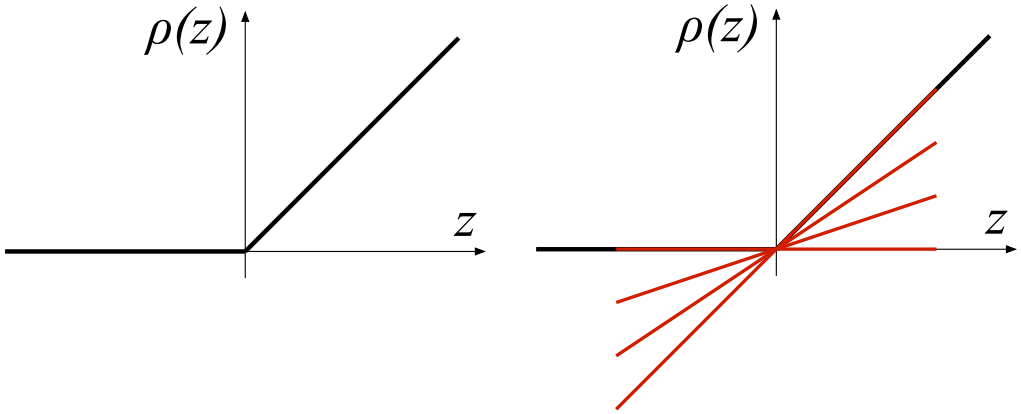


Figure 6: The hinge function (left) is convex and its derivative is discontinuous at the origin. The red lines on the right are all (weakly) below $\rho(z)$.

For cases like this, the derivative at z can be replaced by a *sub-derivative*, which is defined as the slope of any straight line $\ell(z)$ that touches $\rho(z)$ at z (that is, $\ell(z) = \rho(z)$) and that is weakly below ρ everywhere ($\ell(u) \leq \rho(u)$ for all u). Where the derivative is continuous, it is equal to its only sub-derivative there. Where it is not, the sub-derivatives form an interval of real values. For example, any point in $[0, 1]$ is a sub-derivative of the hinge function at the origin, as the right panel in Figure 6 illustrates.

It can then be shown that the asymptotic performance of gradient descent or its variants remains unaltered if the derivative (or, for multivariate functions, the gradient) is replaced by a sub-derivative (sub-gradient) [5]. It does not matter which sub-derivative is used. For instance, for the hinge function it is convenient to use the smallest sub-derivative, that is, zero, when $z = 0$.

References

[1] N. Murata. *A statistical study of online learning*, pages 63–92. Online Learning and Neural Networks (D. Saad, editor). Cambridge University Press, Cambridge, MA, 1998.

[2] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, 1999.

[3] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[4] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[5] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, Cambridge, UK, 2014.

[6] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

Appendices

A The Golden Ratio Line Search

We show that the breakpoint u for line search can be chosen so that at each iteration the ratio

$$r(a,b,c) = \frac{\max(b-a, c-b)}{\min(b-a, c-b)}$$

between the length of the longer segment and that of the shorter segment in the bracketing triple is the same. To find how to do this, let us reason for the case $b-a < c-b$, illustrated in Figure 7. The reasoning for the opposite case is analogous.

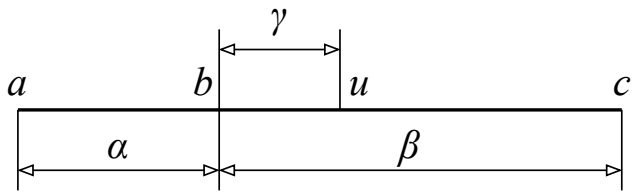


Figure 7: If points b and u are placed so that the ratio $w = \beta/\alpha$ equals the ratio α/γ, then we also have $(\beta-\gamma)/\gamma = w$, and $w \approx 1.618$ remains the same at every iteration of line search.

From the code for line search, we see that u splits the interval $[b,c]$, the longer of the two. Before the split, if we let

$$\alpha = b-a \quad \text{and} \quad \beta = c-b$$

we thus have

$$r(a,b,c) = \frac{\beta}{\alpha}\,.$$

After the split, the triple is either (a,b,u) or (b,u,c). Choose u so that

$$\gamma = u-b < \alpha \quad \text{and} \quad \gamma = u-b < c-u = \beta-\gamma\,.$$

Then

$$r(a,b,u) = \frac{\alpha}{\gamma} \quad \text{and} \quad r(b,u,c) = \frac{\beta-\gamma}{\gamma}\,.$$

Requiring the ratio r to be the same before and after the split in all cases requires

$$r(a,b,c) = r(a,b,u) \quad \text{and} \quad r(a,b,c) = r(b,u,c) \quad \text{that is,} \quad \frac{\beta}{\alpha} = \frac{\alpha}{\gamma} \quad \text{and} \quad \frac{\beta}{\alpha} = \frac{\beta-\gamma}{\gamma}\,.$$

Solving these two equations for γ yields

$$\gamma = \frac{\alpha^2}{\beta} = \frac{\alpha\beta}{\alpha+\beta}$$

and therefore, after simple algebra,

$$w = \frac{1+w}{w} \quad \text{where} \quad w = \frac{\beta}{\alpha}\,.$$

Rearranging terms yields the quadratic equation

$$w^2 - w - 1 = 0$$

which has a solution that is greater than 1 and one that is less than 1. Since $w > 1$, we obtain

$$w = \frac{1+\sqrt{5}}{2} \approx 1.618 \,,$$

a number called the *golden ratio*.

Thus, if b is initially placed between a and c so that

$$r(a,b,c) = \frac{c-b}{b-a} = w$$

and then, when the interval $[b,c]$ is split, the breakpoint u is placed so that

$$r(b,u,c) = \frac{c-u}{b-u} = w \,,$$

we automatically also obtain that

$$r(a,b,u) = \frac{b-a}{u-b} = w \,.$$

The reasoning for the case $b - a > c - b$ is similar and is left as an exercise.

B Gradient Descent on a Paraboloid

This Appendix finds exact formulas for gradient descent when f is a paraboloid. This study will then lead to an analysis of the convergence speed of this optimization method (Appendix B.1).

Let

$$\tilde{e}(\mathbf{z}) = \frac{1}{2}(\mathbf{z} - \mathbf{z}^*)^T Q(\mathbf{z} - \mathbf{z}^*) \,.$$

Then we have

$$\tilde{e}(\mathbf{z}) = f(\mathbf{z}) - c + \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* = f(\mathbf{z}) - f(\mathbf{z}^*) \tag{10}$$

so that $\tilde{e}$ and f differ only by a constant. To show this, we note that the definition of $\mathbf{z}^*$ means that

$$Q\mathbf{z}^* = -\mathbf{a}$$

and so

$$-\mathbf{z}^T Q \mathbf{z}^* = \mathbf{z}^T \mathbf{a} = \mathbf{a}^T \mathbf{z}$$

and therefore

$$\tilde{e}(\mathbf{z}) = \frac{1}{2}(\mathbf{z}^T Q \mathbf{z} + \mathbf{z}^{*T} Q \mathbf{z}^* - 2\mathbf{z}^T Q \mathbf{z}^*) = \frac{1}{2}\mathbf{z}^T Q \mathbf{z} + \mathbf{a}^T \mathbf{z} + \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* = f(\mathbf{z}) - c + \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* \,.$$

We can also write

$$f(\mathbf{z}^*) = c + \mathbf{a}^T \mathbf{z}^* + \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* = c - \mathbf{z}^{*T} Q \mathbf{z}^* + \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* = c - \frac{1}{2}\mathbf{z}^{*T} Q \mathbf{z}^* \,.$$

The result (10),

$$\tilde{e}(\mathbf{z}) = f(\mathbf{z}) - f(\mathbf{z}^*) \ ,$$

is rather interesting in itself. It says that adding a linear term $\mathbf{a}^T\mathbf{z}$ (and a constant c) to a paraboloid $\frac{1}{2}\mathbf{z}^T Q\mathbf{z}$ merely shifts the bottom of the paraboloid, both in position ($\mathbf{z}^*$ rather than $\mathbf{0}$) and value ($c - \frac{1}{2}\mathbf{z}^{*T} Q\mathbf{z}^*$ rather than zero). Adding the linear term does not "warp" or "tilt" the shape of the paraboloid in any way.

Since $\tilde{e}$ is simpler, we consider that we are minimizing $\tilde{e}$ rather than f. In addition, we can let

$$\mathbf{y} = \mathbf{z} - \mathbf{z}^* \ ,$$

that is, we can shift the origin of the domain to $\mathbf{z}^*$, and study the function

$$e(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T Q\mathbf{y}$$

instead of f or $\tilde{e}$, without loss of generality. We will transform everything back to f and $\mathbf{z}$ once we are done. Of course, by construction, the new minimum is at

$$\mathbf{y}^* = \mathbf{0}$$

where e reaches a value of zero:

$$e(\mathbf{y}^*) = e(\mathbf{0}) = 0 \ .$$

However, we let our gradient descent algorithm find this minimum by starting from the initial point

$$\mathbf{y}_0 = \mathbf{z}_0 - \mathbf{z}^* \ .$$

At every iteration k, the algorithm chooses the direction of steepest local descent, which is in the direction

$$\mathbf{p}_k = -\frac{\mathbf{g}_k}{\|\mathbf{g}_k\|}$$

opposite to the gradient of e evaluated at $\mathbf{y}_k$:

$$\mathbf{g}_k = \mathbf{g}(\mathbf{y}_k) = \left.\frac{\partial e}{\partial \mathbf{y}}\right|_{\mathbf{y}=\mathbf{y}_k} = Q\mathbf{y}_k \ .$$

We select for the algorithm the most favorable step size, that is, the one that takes us from $\mathbf{y}_k$ to the lowest point in the direction of $\mathbf{p}_k$. This can be found by differentiating the function

$$e(\mathbf{y}_k + \alpha\mathbf{p}_k) = \frac{1}{2}(\mathbf{y}_k + \alpha\mathbf{p}_k)^T Q(\mathbf{y}_k + \alpha\mathbf{p}_k)$$

with respect to α, and setting the derivative to zero to obtain the optimal step α_k. We have

$$\frac{\partial e(\mathbf{y}_k + \alpha\mathbf{p}_k)}{\partial \alpha} = (\mathbf{y}_k + \alpha\mathbf{p}_k)^T Q\mathbf{p}_k$$

and setting this to zero yields

$$\alpha_k = -\frac{(Q\mathbf{y}_k)^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = -\frac{\mathbf{g}_k^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = \|\mathbf{g}_k\|\frac{\mathbf{p}_k^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = \|\mathbf{g}_k\|\frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T Q\mathbf{g}_k} \ . \tag{11}$$

Thus, the basic step of our gradient descent can be written as follows:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \|\mathbf{g}_k\| \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{p}_k$$

that is,

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{g}_k \ . \tag{12}$$

B.1 The Convergence Speed of Gradient Descent

How much closer does one step of gradient descent bring us to the solution $\mathbf{z}^*$? In other words, how much closer is $\mathbf{z}_{k+1}$ to the true minimum $\mathbf{z}^*$, relative to the distance between $\mathbf{z}_k$ and $\mathbf{z}^*$ at the previous step? The answer is, often not much, as illustrated earlier in Figure 5. To characterize the speed of convergence of various minimization algorithms quantitatively, we introduce the notion of the *order of convergence*. This is defined as the largest value of q for which the

$$\lim_{k \to \infty} \frac{\|\mathbf{z}_{k+1} - \mathbf{z}^*\|}{\|\mathbf{z}_k - \mathbf{z}^*\|^q}$$

is finite and nonzero. If β is this limit, then close to the solution (that is, for large values of k) we have

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\| \approx \beta \|\mathbf{z}_k - \mathbf{z}^*\|^q$$

for a minimization method of order q. In other words, the distance of $\mathbf{z}_k$ from $\mathbf{z}^*$ is reduced by the q-th power at every step, so the higher the order of convergence, the better.

Theorem B.2 below implies that gradient descent has generically (that is, with few exceptions in special cases) a linear order of convergence. In other words, the residuals $|f(\mathbf{z}_k) - f(\mathbf{z}^*)|$ in the *values* of the function being minimized converge linearly. Since the gradient of f approaches zero when $\mathbf{z}_k$ tends to $\mathbf{z}^*$, the *arguments* $\mathbf{z}_k$ to f can converge to $\mathbf{z}^*$ even more slowly.

While Theorem B.2 holds for quadratic functions, any smooth function can be approximated by a quadratic function in small neighborhoods, so the result is general.

The arguments and proofs below are adapted from D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1973, and are based on the following preliminary result.

Lemma B.1 (Kantorovich inequality)**.** *Let Q be a positive definite, symmetric, $n \times n$ matrix. For any vector $\mathbf{y}$ there holds*

$$\frac{(\mathbf{y}^T \mathbf{y})^2}{\mathbf{y}^T Q^{-1} \mathbf{y} \, \mathbf{y}^T Q \mathbf{y}} \geq \frac{4\sigma_1 \sigma_n}{(\sigma_1 + \sigma_n)^2}$$

where σ_1 and σ_n are, respectively, the largest and smallest singular values of Q.

Proof. Let

$$Q = U \Sigma U^T$$

be the singular value decomposition of the symmetric (hence $V = U$) matrix Q. Because Q is positive definite, all its singular values are strictly positive, since the smallest of them satisfies

$$\sigma_n = \min_{\|\mathbf{y}\|=1} \mathbf{y}^T Q \mathbf{y} > 0$$

by the definition of positive definiteness. If we let

$$\mathbf{u} = U^T \mathbf{y}$$

we have

$$\frac{(\mathbf{y}^T\mathbf{y})^2}{\mathbf{y}^T Q^{-1}\mathbf{y}\,\mathbf{y}^T Q\mathbf{y}} = \frac{(\mathbf{y}^T U^T U\mathbf{y})^2}{\mathbf{y}^T U\Sigma^{-1}U^T\mathbf{y}\,\mathbf{y}^T U\Sigma U^T\mathbf{y}} = \frac{(\mathbf{u}^T\mathbf{u})^2}{\mathbf{u}^T\Sigma^{-1}\mathbf{u}\,\mathbf{u}^T\Sigma\mathbf{u}} = \frac{1/\sum_{i=1}^n \theta_i\sigma_i}{\sum_{i=1}^n \theta_i/\sigma_i} = \frac{\phi(\sigma)}{\psi(\sigma)} \tag{13}$$

where the coefficients

$$\theta_i = \frac{z_i^2}{\|\mathbf{u}\|^2}$$

add up to one. If we let

$$\sigma = \sum_{i=1}^n \theta_i\sigma_i \;, \tag{14}$$

then the numerator $\phi(\sigma)$ in (13) is $1/\sigma$. Of course, there are many ways to choose the coefficients θ_i to obtain a particular value of σ. However, each of the singular values σ_j can be obtained by letting $\theta_j = 1$ and all other θ_i to zero. Thus, the values $1/\sigma_j$ for $j = 1, \ldots, n$ are all on the curve $1/\sigma$. The denominator $\psi(\sigma)$ in (13) is a convex combination of points on this curve. Since $1/\sigma$ is a convex function of σ, the values of the denominator $\psi(\sigma)$ of (13) must be in the shaded area in figure 8. This area is delimited from above by the straight line that connects point $(\sigma_1, 1/\sigma_1)$ with point $(\sigma_n, 1/\sigma_n)$, that is, by the line with ordinate

$$\lambda(\sigma) = (\sigma_1 + \sigma_n - \sigma)/(\sigma_1\sigma_n) \;.$$

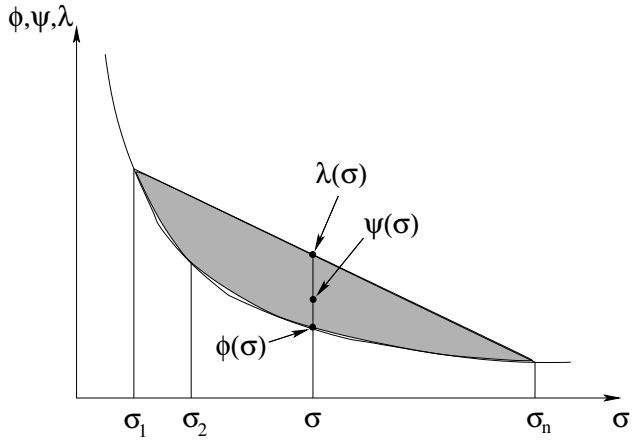


Figure 8: Kantorovich inequality.

For the same vector of coefficients θ_i, the values of $\phi(\sigma)$, $\psi(\sigma)$, and $\lambda(\sigma)$ are on the vertical line corresponding to the value of σ given by (14). Thus an appropriate bound is

$$\frac{\phi(\sigma)}{\psi(\sigma)} \geq \min_{\sigma_1\leq\sigma\leq\sigma_n} \frac{\phi(\sigma)}{\lambda(\sigma)} = \min_{\sigma_1\leq\sigma\leq\sigma_n} \frac{1/\sigma}{(\sigma_1 + \sigma_n - \sigma)/(\sigma_1\sigma_n)} \;.$$

The minimum is achieved at $\sigma = (\sigma_1 + \sigma_n)/2$, yielding the desired result. Δ

Thanks to this lemma, we can now prove the following Theorem on the convergence of the method of gradient descent.

Theorem B.2. *Let*

$$f(\mathbf{z}) = c + \mathbf{a}^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T Q\mathbf{z}$$

be a quadratic function of $\mathbf{z}$*, with* Q *symmetric and positive definite. For any* $\mathbf{z}_0$*, the method of gradient descent has trajectory*

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T Q\mathbf{g}_k}\mathbf{g}_k \tag{15}$$

where

$$\mathbf{g}_k = \mathbf{g}(\mathbf{z}_k) = \left.\frac{\partial f}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_k} = \mathbf{a} + Q\mathbf{z}_k \ .$$

This trajectory converges to the unique minimum point

$$\mathbf{z}^* = -Q^{-1}\mathbf{a}$$

of f*. Furthermore, at every step* k *there holds*

$$f(\mathbf{z}_{k+1}) - f(\mathbf{z}^*) \leq \left(\frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n}\right)^2 (f(\mathbf{z}_k) - f(\mathbf{z}^*))$$

where σ_1 *and* σ_n *are, respectively, the largest and smallest singular value of* Q*.*

The ratio $\kappa(Q) = \sigma_1/\sigma_n$ is called the *condition number* of Q. The larger the condition number, the closer the fraction $(\sigma_1 - \sigma_n)/(\sigma_1 + \sigma_n)$ is to unity, and the slower convergence. When $\kappa(Q) = 1$, we have

$$\frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n} = 0 \ .$$

and convergence is immediate. The more elongated the isocontours, that is, the greater the condition number $\kappa(Q)$, the farther away a line orthogonal to an isocontour passes from $\mathbf{z}^*$, and the more steps are required for convergence.

For general (that is, non-quadratic) f, the analysis above applies once $\mathbf{z}_k$ gets close enough to the minimum, so that f is well approximated by a paraboloid. In this case, Q is the matrix of second derivatives of f with respect to $\mathbf{z}$, and is called the *Hessian* of f. In summary, gradient descent is good for functions that have a well conditioned Hessian near the minimum, but can become arbitrarily slow for poorly conditioned Hessians.

Proof. From the definition of e and from equation (12) we obtain

$$\begin{aligned}
\frac{e(\mathbf{y}_k) - e(\mathbf{y}_{k+1})}{e(\mathbf{y}_k)} &= \frac{\mathbf{y}_k^T Q \mathbf{y}_k - \mathbf{y}_{k+1}^T Q \mathbf{y}_{k+1}}{\mathbf{y}_k^T Q \mathbf{y}_k} \\
&= \frac{\mathbf{y}_k^T Q \mathbf{y}_k - \left(\mathbf{y}_k - \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{g}_k\right)^T Q \left(\mathbf{y}_k - \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{g}_k\right)}{\mathbf{y}_k^T Q \mathbf{y}_k} \\
&= \frac{2 \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{g}_k^T Q \mathbf{y}_k - \left(\frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k}\right)^2 \mathbf{g}_k^T Q \mathbf{g}_k}{\mathbf{y}_k^T Q \mathbf{y}_k} \\
&= \frac{2 \mathbf{g}_k^T \mathbf{g}_k \mathbf{g}_k^T Q \mathbf{y}_k - (\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{y}_k^T Q \mathbf{y}_k \, \mathbf{g}_k^T Q \mathbf{g}_k} \, .
\end{aligned}$$

Since Q is invertible we have

$$\mathbf{g}_k = Q \mathbf{y}_k \quad \Rightarrow \quad \mathbf{y}_k = Q^{-1} \mathbf{g}_k$$

and

$$\mathbf{y}_k^T Q \mathbf{y}_k = \mathbf{g}_k^T Q^{-1} \mathbf{g}_k$$

so that

$$\frac{e(\mathbf{y}_k) - e(\mathbf{y}_{k+1})}{e(\mathbf{y}_k)} = \frac{(\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{g}_k^T Q^{-1} \mathbf{g}_k \, \mathbf{g}_k^T Q \mathbf{g}_k} \, .$$

This can be rewritten as follows by rearranging terms:

$$e(\mathbf{y}_{k+1}) = \left(1 - \frac{(\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{g}_k^T Q^{-1} \mathbf{g}_k \, \mathbf{g}_k^T Q \mathbf{g}_k}\right) e(\mathbf{y}_k) \tag{16}$$

We can now use the lemma on the Kantorovich inequality proven earlier to bound the expression in parentheses and therefore the rate of convergence of gradient descent. From the definitions

$$\mathbf{y} = \mathbf{z} - \mathbf{z}^* \quad \text{and} \quad e(\mathbf{y}) = \frac{1}{2} \mathbf{y}^T Q \mathbf{y} \tag{17}$$

we immediately obtain the expression for gradient descent in terms of f and $\mathbf{z}$. By equations (10) and (16) and the Kantorovich inequality we obtain

$$\begin{aligned}
f(\mathbf{z}_{k+1}) - f(\mathbf{z}^*) &= e(\mathbf{y}_{k+1}) = \left(1 - \frac{(\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{g}_k^T Q^{-1} \mathbf{g}_k \, \mathbf{g}_k^T Q \mathbf{g}_k}\right) e(\mathbf{y}_k) \leq \left(1 - \frac{4 \sigma_1 \sigma_n}{(\sigma_1 + \sigma_n)^2}\right) e(\mathbf{y}_k) \quad (18) \\
&= \left(\frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n}\right)^2 (f(\mathbf{z}_k) - f(\mathbf{z}^*)) \, . \quad (19)
\end{aligned}$$

Since the ratio in the last term is smaller than one, it follows immediately that $f(\mathbf{z}_k) - f(\mathbf{z}^*) \to 0$ and hence, since the minimum of f is unique, that $\mathbf{z}_k \to \mathbf{z}^*$. Δ

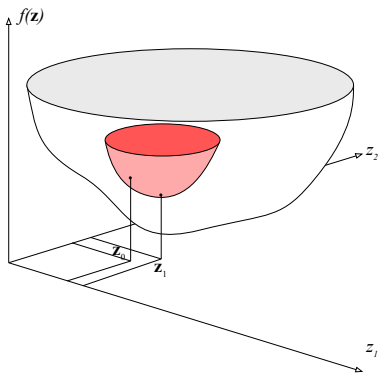


Figure 9: The shaded bowl is a piece of the paraboloid represented by the second-order Taylor series expansion of f around $\mathbf{z}_0$. The minimum $\mathbf{z}_1$ of the paraboloid can be found by solving a linear system in $\mathbf{z}$, and $f(\mathbf{z}_1)$ is often significantly smaller than $f(\mathbf{z}_0)$.

C Newton's Method

When also the Hessian $H(\mathbf{z}_0)$ of f at $\mathbf{z}_0$ is known, in addition to the gradient, the information about the shape of the graph of f in a neighborhood of f is richer, and a second-order Taylor approximation is possible:

$$f \;\approx\; g_2(\mathbf{z}) \;=\; f(\mathbf{z}_0) + [\nabla \mathbf{z}_0]^T(\mathbf{z}-\mathbf{z}_0) + \frac{1}{2}(\mathbf{z}-\mathbf{z}_0)^T H(\mathbf{z}_0)(\mathbf{z}-\mathbf{z}_0)\;.$$

As illustrated in Figure 9, this function looks like a bowl *if the Hessian is positive-semidefinite* (if not, this idea cannot be used). The approximation g_2 is a quadratic function of $\mathbf{z}$, so we can find its minimum by writing the gradient of g_2, setting its components to zero, and solving the resulting linear system in $\mathbf{z}$. The resulting point $\mathbf{z}_1$ is the lowest point of the bowl (see the Figure). Of course, $\mathbf{z}_1$ may *not* be the minimum of f, because g_2 merely approximates f. However, if the approximation is reasonable, the step from $\mathbf{z}_0$ to $\mathbf{z}_1$ is likely to reduce the value of f:

$$f(\mathbf{z}_1) < f(\mathbf{z}_0)\;.$$

While the approximation g_2 to f may not be very good initially, it becomes better and better, as $\mathbf{z}^{(i)}$ tends to a minimum: If the steps get smaller and smaller, the Taylor approximation for f gets better and better, because its quality increases for smaller and smaller neighborhoods. In other words, every smooth function looks like a quadratic function in a small enough neighborhood.

This method is called a *second-order descent method*, and can be shown to have a much faster convergence rate than a first-order method if the function f is smooth.

Looking at even higher-order approximations of f is not promising, because Taylor approximations beyond quadratic are not necessarily convex, and minimizing the approximation is not necessarily easier than minimizing f itself.

Equation (9) tells us how to jump in one step from the starting point $\mathbf{z}_0$ to the minimum $\mathbf{z}_1$ of the approximating paraboloid $g_2(\mathbf{z})$. Of course, when $f(\mathbf{z})$ is not exactly a paraboloid, the new value $\mathbf{z}_1$ will be different from the minimum $\hat{\mathbf{z}}$ of f. Consequently, iterations are needed, but convergence can be expected to be faster.

More specifically, let

$$f(\mathbf{z}_k + \Delta\mathbf{z}) \approx f(\mathbf{z}_k) + \mathbf{g}_k^T \Delta\mathbf{z} + \frac{1}{2}\Delta\mathbf{z}^T H_k \Delta\mathbf{z} \tag{20}$$

be the first terms of the Taylor series expansion of f about the current point $\mathbf{z}_k$, where

$$\mathbf{g}_k = \mathbf{g}(\mathbf{z}_k) = \left.\frac{\partial f}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_k}$$

and

$$H_k = H(\mathbf{z}_k) = \left.\frac{\partial^2 f}{\partial \mathbf{z} \partial \mathbf{z}^T}\right|_{\mathbf{z}=\mathbf{z}_k} = \left[\begin{array}{ccc} \frac{\partial^2 f}{\partial z_1^2} & \cdots & \frac{\partial^2 f}{\partial z_1 \partial z_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial z_n \partial z_1} & \cdots & \frac{\partial^2 f}{\partial z_n^2} \end{array}\right]_{\mathbf{z}=\mathbf{z}_k}$$

are the gradient and Hessian of f evaluated at the current point $\mathbf{z}_k$. Notice that even when f is a paraboloid, the gradient $\mathbf{g}_k$ is different from $\mathbf{a}$ as used in equation (8). This is because $\mathbf{a}$ and Q are the coefficients of the Taylor expansion of f around point $\mathbf{z} = 0$, while $\mathbf{g}_k$ and H_k are the coefficients of the Taylor expansion of f around the *current* point $\mathbf{z}_k$. In other words, gradient and Hessian are constantly reevaluated in Newton's method.

To the extent that approximation (20) is valid, we can set the derivatives of $f(\mathbf{z}_k + \Delta\mathbf{z})$ with respect to $\Delta\mathbf{z}$ to zero, and obtain, analogously to equation (9), the linear system

$$H_k \Delta\mathbf{z} = -\mathbf{g}_k \ , \tag{21}$$

whose solution $\Delta\mathbf{z}_k = \alpha_k \mathbf{p}_k$ yields at the same time the step direction $\mathbf{p}_k = \Delta\mathbf{z}_k / \|\Delta\mathbf{z}_k\|$ *and* the step size $\alpha_k = \|\Delta\mathbf{z}_k\|$. The direction is of course undefined once the algorithm has reached a minimum, that is, when $\alpha_k = 0$.

A minimization algorithm in which the step direction $\mathbf{p}_k$ and size α_k are defined in this manner is called *Newton's method*. The corresponding $\mathbf{p}_k$ is termed the *Newton direction*, and the step defined by equation (21) is the *Newton step*.

The greater speed of Newton's method over gradient descent is borne out by analysis: While gradient descent has a linear order of convergence, Newton's method has a quadratic order of convergence (see Section C.1).

For a quadratic function, as in equation (8), steepest descent takes many steps to converge, while Newton's method reaches the minimum in one step. However, this single iteration in Newton's method is more expensive, because it requires both the gradient $\mathbf{g}_k$ and the Hessian H_k to be evaluated, for a total of $m + \binom{m+1}{2}$ derivatives. In addition, system (21) must be solved at each iteration. For very large problems, in which the dimension m of $\mathbf{z}$ is thousands or more, storing and manipulating a Hessian can be prohibitive. In contrast, gradient descent requires only the gradient $\mathbf{g}_k$ for selecting the step direction $\mathbf{p}_k$, and a line search in the direction $\mathbf{p}_k$ to find the step size.

C.1 The Convergence Speed of Newton's Method

Newton's method has at least a quadratic order of convergence. To see this, let

$$\mathbf{y}(\mathbf{z}) = \mathbf{z} - Q(\mathbf{z})^{-1}\mathbf{g}(\mathbf{z})$$

be the place reached by a Newton step starting at $\mathbf{z}$ (see equation (21)), and suppose that at the minimum $\mathbf{z}^*$ the Hessian $Q(\mathbf{z}^*)$ is nonsingular. Then

$$\mathbf{y}(\mathbf{z}^*) = \mathbf{z}^*$$

because $\mathbf{g}(\mathbf{z}^*) = \mathbf{0}$, and

$$\mathbf{z}_{k+1} - \mathbf{z}^* = \mathbf{y}(\mathbf{z}_k) - \mathbf{z}^* = \mathbf{y}(\mathbf{z}_k) - \mathbf{y}(\mathbf{z}^*) \; .$$

From the mean-value theorem, we have

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\| = \|\mathbf{y}(\mathbf{z}_k) - \mathbf{y}(\mathbf{z}^*)\| \leq \left\| \left[\frac{\partial \mathbf{y}}{\partial \mathbf{z}^T} \right]_{\mathbf{z}=\mathbf{z}^*} (\mathbf{z}_k - \mathbf{z}^*) \right\| + \frac{1}{2} \left| \frac{\partial^2 \mathbf{y}}{\partial \mathbf{z} \partial \mathbf{z}^T} \right|_{\mathbf{z}=\hat{\mathbf{z}}} \|\mathbf{z}_k - \mathbf{z}^*\|^2$$

where $\hat{\mathbf{z}}$ is some point on the line between $\mathbf{z}^*$ and $\mathbf{z}_k$. Since $\mathbf{y}(\mathbf{z}^*) = \mathbf{z}^*$, the first derivatives of $\mathbf{y}$ at $\mathbf{z}^*$ are zero, so that the first term in the right-hand side above vanishes, and

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\| \leq c \, \|\mathbf{z}_k - \mathbf{z}^*\|^2$$

where c depends on third-order derivatives of f near $\mathbf{z}^*$. Thus, the convergence rate of Newton's method is of order at least two.

Choosing between gradient descent and Newton's method The analysis above shows that Newton steps are typically much larger than gradient-descent steps, and convergence is much faster when measured by the number of steps taken. However, each Newton step is more expensive, because of the need to compute the Hessian at every point. For problems where the dimension n of the space in which $\mathbf{z}$ lives is small, the extra cost per step may be worth the resulting smaller number of steps. However, the cost to compute Q increases quadratically with n, so at some point Newton methods become overly expensive (in both time and storage) or even practically infeasible.

In those cases, one can compute approximate Hessians, rather than exact ones, in Newton's method, leading to so-called *quasi-Newton* algorithms. Alternatively, one can modify the descent direction away from the gradient in a way that effectively makes the steepest-descent algorithm aware of the second-order derivatives of f. This idea leads to the *conjugate gradients* method, whose computational complexity is similar to that of gradient descent and whose convergence rate approaches that of Newton's method in many cases. This method is discussed for completeness next.

D Conjugate Gradients

The method of conjugate gradients, discussed in this appendix, is motivated by the desire to accelerate convergence with respect to the gradient descent method, but without paying the storage cost of Newton's method.

Newton's method converges faster (quadratically) than gradient descent (linear convergence rate) because it uses more information about the function f being minimized. Gradient descent locally approximates the function with planes, because it only uses gradient information. All it can do is to go downhill. Newton's method approximates f with paraboloids, and then jumps at every iteration to the lowest point of the current approximation. The bottom line is that fast convergence requires work that is equivalent to evaluating the Hessian of f.

Prima facie, the method of conjugate gradients discussed in this section seems to violate this principle: it achieves fast, superlinear convergence, similarly to Newton's method, but it only requires gradient information. This paradox, however, is only apparent. Conjugate gradients works by taking n steps for each of the steps in Newton's method. It effectively solves the linear system (9) of Newton's method, but it does so by a sequence of n one-dimensional minimizations, each requiring one gradient computation and one line search.

Overall, the work done by conjugate gradients is equivalent to that done by Newton's method. However, system (9) is never constructed explicitly, and the matrix Q is never stored. This is very important in cases where $\mathbf{z}$ has thousands or even millions of components. These high-dimensional problems arise typically from the discretization of partial differential equations. Say for instance that we want to compute the motion of points in an image as a consequence of camera motion. Partial differential equations relate image intensities over space and time to the motion of the underlying image features. At every pixel in the image, this motion, called the *motion field*, is represented by a vector whose magnitude and direction describe the velocity of the image feature at that pixel. Thus, if an image has, say, a quarter of a million pixels, there are $n = 500,000$ unknown motion field values. Storing and inverting a $500,000 \times 500,000$ Hessian is out of the question. In cases like these, conjugate gradients saves the day.

The conjugate gradients method described in these notes is the so-called Polak-Ribière variation. It will be introduced in three steps. First, it will be developed for the simple case of minimizing a quadratic function with positive-definite and known Hessian. This quadratic function $f(\mathbf{z})$ was introduced in equation (8). We know that in this case minimizing $f(\mathbf{z})$ is equivalent to solving the linear system (9). Rather than an iterative method, conjugate gradients is a direct method for the quadratic case. This means that the number of iterations is fixed. Specifically, the method converges to the solution in n steps, where n is the number of components of $\mathbf{z}$. Because of the equivalence with a linear system, conjugate gradients for the quadratic case can also be seen as an alternative method for solving a linear system, although the version presented here will only work if the matrix of the system is symmetric and positive definite.

Second, the assumption that the Hessian Q in expression (8) is known will be removed. As discussed above, this is the main reason for using conjugate gradients.

Third, the conjugate gradients method will be extended to general functions $f(\mathbf{z})$. In this case, the method is no longer direct, but iterative, and the cost of finding the minimum depends on the desired accuracy. This occurs because the Hessian of f is no longer a constant, as it was in the quadratic case. As a consequence, a certain property that holds in the quadratic case is now valid only approximately. In spite of this, the convergence rate of conjugate gradients is superlinear, somewhere between Newton's method and gradient descent. Finding tight bounds for the convergence rate of conjugate gradients is hard, and we will omit this proof. We rely instead on the intuition that conjugate gradients solves system (9), and that the quadratic approximation becomes more and more valid as the algorithm converges to the minimum. If the function f starts to behave like a quadratic function early, that is, if f is nearly quadratic in a large neighborhood of the minimum, convergence is fast, as it requires close to the n steps that are necessary in the quadratic case, and each of the steps is simple. This combination of fast convergence, modest storage requirements, and low computational cost per iteration explains the popularity of conjugate gradients methods for the optimization of functions of a large number of variables.

D.1 The Quadratic Case

Suppose that we want to minimize the quadratic function

$$f(\mathbf{z}) = c + \mathbf{a}^T\mathbf{z} + \frac{1}{2}\mathbf{z}^T Q\mathbf{z} \tag{22}$$

where Q is a symmetric, positive definite matrix, and $\mathbf{z}$ has n components. As we saw in our discussion of gradient descent, the minimum $\mathbf{z}^*$ is the solution to the linear system

$$Q\mathbf{z} = -\mathbf{a}\ . \tag{23}$$

We know how to solve such a system. However, all the methods we have seen so far involve explicit manipulation of the matrix Q. We now consider an alternative solution method that does not need Q, but only the quantity

$$\mathbf{g}_k = Q\mathbf{z}_k + \mathbf{a}$$

that is, the gradient of $f(\mathbf{z})$, evaluated at n different points $\mathbf{z}_1, \ldots, \mathbf{z}_n$. We will see that the conjugate gradients method requires n gradient evaluations and n line searches *in lieu* of each $n \times n$ matrix inversion in Newton's method.

Formal proofs can be found in Elijah Polak, *Optimization — Algorithms and consistent approximations*, Springer, NY, 1997. The arguments offered below appeal to intuition.

Consider the case $n = 3$, in which the variable $\mathbf{z}$ in $f(\mathbf{z})$ is a three-dimensional vector. Then the quadratic function $f(\mathbf{z})$ is constant over ellipsoids, called *isosurfaces*, centered at the minimum $\mathbf{z}^*$. How can we start from a point $\mathbf{z}_0$ on one of these ellipsoids and reach $\mathbf{z}^*$ by a finite sequence of one-dimensional searches? In connection with gradient descent, we noticed that for poorly conditioned Hessians orthogonal directions lead to many small steps, that is, to slow convergence.

When the ellipsoids are spheres, on the other hand, this works much better. The first step takes from $\mathbf{z}_0$ to $\mathbf{z}_1$, and the line between $\mathbf{z}_0$ and $\mathbf{z}_1$ is tangent to an isosurface at $\mathbf{z}_1$. The next step is in the direction of the gradient, so that the new direction $\mathbf{p}_1$ is orthogonal to the previous direction $\mathbf{p}_0$. This would then take us to $\mathbf{z}^*$ right away. Suppose however that we cannot afford to compute this special direction $\mathbf{p}_1$ orthogonal to $\mathbf{p}_0$, but that we can only compute *some* direction $\mathbf{p}_1$ orthogonal to $\mathbf{p}_0$ (there is an $n-1$-dimensional space of such directions!). It is easy to see that in that case n steps will take us to $\mathbf{z}^*$. This is because since isosurfaces are spheres, each line minimization is independent of the others: The first step yields the minimum in the space spanned by $\mathbf{p}_0$, the second step then yields the minimum in the space spanned by $\mathbf{p}_0$ and $\mathbf{p}_1$, and so forth. After n steps we must be done, since $\mathbf{p}_0 \ldots, \mathbf{p}_{n-1}$ span the whole space.

In summary, any set of orthogonal directions, with a line search in each direction, will lead to the minimum for spherical isosurfaces. Given an arbitrary set of ellipsoidal isosurfaces, there is a one-to-one mapping with a spherical system: if $Q = U\Sigma U^T$ is the SVD of the symmetric, positive definite matrix Q, then we can write

$$\frac{1}{2}\mathbf{z}^T Q\mathbf{z} = \frac{1}{2}\mathbf{y}^T\mathbf{y}$$

where

$$\mathbf{y} = \Sigma^{1/2}U^T\mathbf{z}\ . \tag{24}$$

Consequently, there must be a condition for the original problem (in terms of Q) that is equivalent to orthogonality for the spherical problem. If two directions $\mathbf{q}_i$ and $\mathbf{q}_j$ are orthogonal in the spherical context, that is, if

$$\mathbf{q}_i^T \mathbf{q}_j = 0 \ ,$$

what does this translate into in terms of the directions $\mathbf{p}_i$ and $\mathbf{p}_j$ for the ellipsoidal problem? We have

$$\mathbf{q}_{i,j} = \Sigma^{1/2} U^T \mathbf{p}_{i,j} \ ,$$

so that orthogonality for $\mathbf{q}_{i,j}$ becomes

$$\mathbf{p}_i^T U \Sigma^{1/2} \Sigma^{1/2} U^T \mathbf{p}_j = 0$$

or

$$\mathbf{p}_i^T Q \mathbf{p}_j = 0 \ . \tag{25}$$

This condition is called *Q-conjugacy*, or *Q-orthogonality*: if equation (25) holds, then $\mathbf{p}_i$ and $\mathbf{p}_j$ are said to be Q-conjugate or Q-orthogonal to each other. We will henceforth simply say "conjugate" for brevity.

In summary, if we can find n directions $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$ that are mutually conjugate, and if we do line minimization along each direction $\mathbf{p}_k$, we reach the minimum in at most n steps. Of course, we cannot use the transformation (24) in the algorithm, because Σ and especially U^T are too large. So now we need to find a method for generating n conjugate directions without using either Q or its SVD. We do this in two steps. First, we find conjugate directions whose definitions do involve Q. Then, in the next subsection, we rewrite these expressions without Q.

Here is the procedure, due to Hestenes and Stiefel (*Methods of conjugate gradients for solving linear systems*, J. Res. Bureau National Standards, section B, Vol 49, pp. 409-436, 1952), which also incorporates the steps from $\mathbf{z}_0$ to $\mathbf{z}_n$:

$\mathbf{g}_0 = \mathbf{g}(\mathbf{z}_0)$
$\mathbf{p}_0 = -\mathbf{g}_0$
for $k = 0 \ldots, n-1$
  $\alpha_k = \arg\min_{\alpha \geq 0} f(\mathbf{z}_k + \alpha \mathbf{p}_k)$
  $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{g}_{k+1} = \mathbf{g}(\mathbf{z}_{k+1})$
  $\gamma_k = \frac{\mathbf{g}_{k+1}^T Q \mathbf{p}_k}{\mathbf{p}_k^T Q \mathbf{p}_k}$
  $\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \gamma_k \mathbf{p}_k$
end

where

$$\mathbf{g}_k = \mathbf{g}(\mathbf{z}_k) = \left. \frac{\partial f}{\partial \mathbf{z}} \right|_{\mathbf{z}=\mathbf{z}_k}$$

is the gradient of f at $\mathbf{z}_k$.

It is simple to see that $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$ are conjugate. In fact,

$$\begin{aligned}\mathbf{p}_k^T Q \mathbf{p}_{k+1} &= \mathbf{p}_k^T Q(-\mathbf{g}_{k+1} + \gamma_k \mathbf{p}_k) \\ &= -\mathbf{p}_k^T Q \mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^T Q \mathbf{p}_k}{\mathbf{p}_k^T Q \mathbf{p}_k} \mathbf{p}_k^T Q \mathbf{p}_k \\ &= -\mathbf{p}_k^T Q \mathbf{g}_{k+1} + \mathbf{g}_{k+1}^T Q \mathbf{p}_k = 0 \ .\end{aligned}$$

It is somewhat more cumbersome to show that $\mathbf{p}_i$ and $\mathbf{p}_{k+1}$ for $i = 0, \ldots, k$ are also conjugate. This can be done by induction. The proof is based on the observation that the vectors $\mathbf{p}_k$ are found by a generalization of the Gram-Schmidt orthogonalization method to produce conjugate rather than orthogonal vectors. Details can be found in Polak's book mentioned earlier.

D.2 Removing the Hessian

The algorithm shown in the previous subsection is a correct conjugate gradients algorithm. However, it is computationally inadequate because the expression for γ_k contains the Hessian Q, which is too large. We now show that γ_k can be rewritten in terms of the gradient values $\mathbf{g}_k$ and $\mathbf{g}_{k+1}$ only. To this end, we notice that

$$\mathbf{g}_{k+1} = \mathbf{g}_k + \alpha_k Q \mathbf{p}_k \ ,$$

or

$$\alpha_k Q \mathbf{p}_k = \mathbf{g}_{k+1} - \mathbf{g}_k \ .$$

In fact,

$$\mathbf{g}(\mathbf{z}) = \mathbf{a} + Q\mathbf{z}$$

so that

$$\mathbf{g}_{k+1} = \mathbf{g}(\mathbf{z}_{k+1}) = \mathbf{g}(\mathbf{z}_k + \alpha_k \mathbf{p}_k) = \mathbf{a} + Q(\mathbf{z}_k + \alpha_k \mathbf{p}_k) = \mathbf{g}_k + \alpha_k Q \mathbf{p}_k \ .$$

We can therefore write

$$\gamma_k = \frac{\mathbf{g}_{k+1}^T Q \mathbf{p}_k}{\mathbf{p}_k^T Q \mathbf{p}_k} = \frac{\mathbf{g}_{k+1}^T \alpha_k Q \mathbf{p}_k}{\mathbf{p}_k^T \alpha_k Q \mathbf{p}_k} = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{p}_k^T (\mathbf{g}_{k+1} - \mathbf{g}_k)} \ ,$$

and Q has disappeared.

This expression for γ_k can be further simplified by noticing that

$$\mathbf{p}_k^T \mathbf{g}_{k+1} = 0$$

because the line along $\mathbf{p}_k$ is tangent to an isosurface at $\mathbf{z}_{k+1}$, while the gradient $\mathbf{g}_{k+1}$ is orthogonal to the isosurface at $\mathbf{z}_{k+1}$. Similarly,

$$\mathbf{p}_{k-1}^T \mathbf{g}_k = 0 \ .$$

Then, the denominator of γ_k becomes

$$\mathbf{p}_k^T (\mathbf{g}_{k+1} - \mathbf{g}_k) = -\mathbf{p}_k^T \mathbf{g}_k = (\mathbf{g}_k - \gamma_{k-1} \mathbf{p}_{k-1})^T \mathbf{g}_k = \mathbf{g}_k^T \mathbf{g}_k \ .$$

In conclusion, we obtain the *Polak-Ribière formula*

$$\gamma_k = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k} \ .$$

D.3 Extension to General Functions

We now know how to minimize the quadratic function (22) in n steps, without ever constructing the Hessian explicitly. When the function $f(\mathbf{z})$ is arbitrary, the same algorithm can be used.

However, n iterations will not suffice. This is because the Hessian, which was constant for the quadratic case, now is a function of $\mathbf{z}_k$. Strictly speaking, we then lose conjugacy, since $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$ are associated to different Hessians. However, as the algorithm approaches the minimum $\hat{\mathbf{z}}$, the quadratic approximation becomes more and more valid, and a few cycles of n iterations each will achieve convergence.