Function Optimization

Carlo Tomasi

There are three main reasons why most problems in robotics, vision, and arguably every other science or endeavor take on the form of optimization problems. One is that the desired goal may not be achievable, and so we try to get as close as possible to it. The second reason is that there may be more ways to achieve the goal, and so we can choose one by assigning a quality to all the solutions and selecting the best one. The third reason is that we may not know how to solve the system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, so instead we minimize the norm $\|\mathbf{f}(\mathbf{x})\|$, which is a scalar function of the unknown vector $\mathbf{x}$.

The first two situations arise in the context of linear systems. The case in which a linear system admits exactly one exact solution is simple but rare. More often, the system at hand is either incompatible (some say overconstrained) or, at the opposite end, underdetermined. In fact, some problems are both, in a sense: There is no exact solution, but there are several approximate ones, each equally good. In addition, many problems lead to nonlinear equations.

Consider, for instance, the problem of Structure From Motion (SFM) in computer vision. Nonlinear equations describe how points in the world project onto the images taken by cameras at given positions in space. Structure from motion goes the other way around, and attempts to solve these equations: image points are given, and one wants to determine shape and motion, namely, where the points in the world and the cameras are. Because image points come from noisy measurements, they are not exact, and the resulting system is usually incompatible. SFM is then cast as an optimization problem. At the same time, even the exact system (one derived from perfect measurements through exact mathematics) is often close to being underdetermined. For instance, the images may provide insufficient information to recover shape and motion. Then, an additional criterion must be added to define what a "good" solution is. In these cases, the noisy system admits no exact solutions, but has many approximate ones.

Machine learning is also optimization: A loss function defines the discrepancy between actual and desired behavior of the function being learned, and optimization methods reduce the discrepancy by finding the function parameters that minimize the loss 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 $f(\mathbf{x})$ is the same as minimizing its negative $-f(\mathbf{x})$, 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: $\mathbf{x}^*$ is a global minimum of f if *for every* $\mathbf{x}$ we have $f(\mathbf{x}) \geq f(\mathbf{x}^*)$. Global minization 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 $\mathbf{x}_0$, and we descend in the landscape of $f(\mathbf{x})$ until we cannot go down any further. The bottom of the valley is a local minimum.

Local minimization is appropriate if we know how to pick an $\mathbf{x}_0$ that is close to $\mathbf{x}^*$. 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 $\mathbf{x}_0$ when looking for the new minimum, that is, when computing the required control signal.

More formally, we reach the correct minimum $\mathbf{x}^*$ as long as the initial point $\mathbf{x}_0$ is in the *basin of attraction* of $\mathbf{x}^*$, defined as the largest neighborhood of $\mathbf{x}^*$ in which $f(\mathbf{x})$ is convex.

If a good $\mathbf{x}_0$ is not available, one may have to be content with a local minimum $\mathbf{x}^*$. After all, $\mathbf{x}^*$ is always at least as good, and often much better, than $\mathbf{x}_0$. A compromise is to pick several values for $\mathbf{x}_0$ (perhaps at random), compute the corresponding values for $\mathbf{x}^*$, and then pick the one with the lowest value $f(\mathbf{x}^*)$.

1 Choosing a Local Minimization Method

The large number of local minimization methods in the literature can be categorized in terms of how much information about the function f they compute as they move from $\mathbf{x}_0$ through points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ towards the local minimum $\mathbf{x}^*$. Specifically, some methods only look at the values $f(\mathbf{x}_i)$, others compute its gradient, and others yet compute the Hessian of f, that is, the matrix of its second derivatives with respect to $\mathbf{x}$.

Higher derivatives provide approximate information about wider regions around the current point $\mathbf{x}_i$ and allow the algorithm to make larger steps in a good direction. For instance, the *steepest descent* method looks at function value $f(\mathbf{x}_i)$ and gradient $\nabla f(\mathbf{x}_i)$. All the method knows is the direction in which the function decreases fastest *at* $\mathbf{x}_i$. It moves in that direction in the hope that $f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$, but needs some way to figure out how long a step to take to make the decrease $f(\mathbf{x}_i) - f(\mathbf{x}_{i+1})$ as large as possible, or at least reasonably large.

In contrast, *Newton* methods also compute the Hessian

$$Q = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

at $\mathbf{x}_i$. They can then approximate the function with a paraboloid and find its minimum $\mathbf{x}_{i+1}$ analytically, by minimizing a quadratic equation in $\mathbf{x}$. This minimum is not necessarily a minimum of f, because the paraboloid only approximates this function, but then one repeats until convergence.

Newton steps are typically much larger than steepest-descent steps, and convergence is much faster. 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 $\mathbf{x}$ lives is small, the extra cost 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 steepest descent and whose convergence rate approaches that of Newton's method in many cases.

Stochastic Gradient Descent (SGD) is a variant of steepest-descent that can be used for functions f that are sums of many differentiable functions:

$$f(\mathbf{x}) = \sum_{k=1}^{m} f_k(\mathbf{x}) \ .$$

In this case, SGD moves in the direction opposite to the gradient of the sum of a random subset (in an extreme version, a singleton) of the f_k terms at each iteration. Clearly, each step is (sometimes dramatically) less expensive than a steepest-descent step. Interestingly, SGD can be shown to converge faster than steepest-descent under suitable conditions, when convergence speed is measured as the total number of operations required (rather than by the number of steps taken).

The next two sections introduce steepest descent and Newton's method. Conjugate gradients is not going to be used in this course, and is discussed for completeness in Appendix D. Stochastic gradient descent is discussed in a separate note in the context of machine learning.

2 Local Minimization and Steepest Descent

Suppose that we want to find a local minimum for the scalar function f of the vector variable $\mathbf{x}$, starting from an initial point $\mathbf{x}_0$. Picking an appropriate $\mathbf{x}_0$ is crucial, but also very problem-dependent. We start from $\mathbf{x}_0$, and we go downhill. At every step of the way, we must make the following decisions:

- Whether to stop.
- In what direction to proceed.
- How long a step to take.

In fact, most minimization algorithms have the following structure:

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

Different algorithms differ in how each of these instructions is performed.

It is intuitively clear that the choice of the step size α_k is important. Too small a step leads to slow convergence, or even to lack of convergence altogether. Too large a step causes overshooting, that is, leaping past the solution. The most disastrous consequence of this is that we may leave the basin of attraction, or that we oscillate back and forth with increasing amplitudes, leading to instability. Even when oscillations decrease, they can slow down convergence considerably.

What is less obvious is that the best direction of descent is not necessarily, and in fact is quite rarely, the direction of steepest descent. Appendix B shows this formally. More intuitively, figure 1 shows the trajectory $\mathbf{x}_k$ superimposed on a set of isocontours of a simple paraboloid (a quadratic function) $f(\mathbf{x})$ for a two-dimensional search space, that is, when $\mathbf{x}$ is a two-dimensional vector.

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

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

where Q is a symmetric, positive definite matrix. *Positive definite* means that for every nonzero $\mathbf{x}$ the quantity $\mathbf{x}^T Q\mathbf{x}$ is positive. In this case, the graph of $f(\mathbf{x}) - c$ is a plane $\mathbf{a}^T\mathbf{x}$ plus a paraboloid. However,

Appendix B shows that adding a linear term $\mathbf{a}^T\mathbf{x}$ (and a constant c) to a paraboloid $\frac{1}{2}\mathbf{x}^T Q\mathbf{x}$ merely shifts the bottom of the paraboloid, both in position ($\mathbf{x}^*$ rather than $\mathbf{0}$) and value ($c - \frac{1}{2}\mathbf{x}^{*T} Q\mathbf{x}^*$ 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:

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

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

$$Q\mathbf{x} = -\mathbf{a} \; . \tag{2}$$

Since Q is positive definite, it is also invertible (why?), and the solution $\mathbf{x}^*$ 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.

Let us therefore assume that we minimize f as given in equation (1), and that at every step we choose the direction of steepest descent. Let us further assume that the length of the step is such that the minimum of f along the steepest-descent direction is reached.

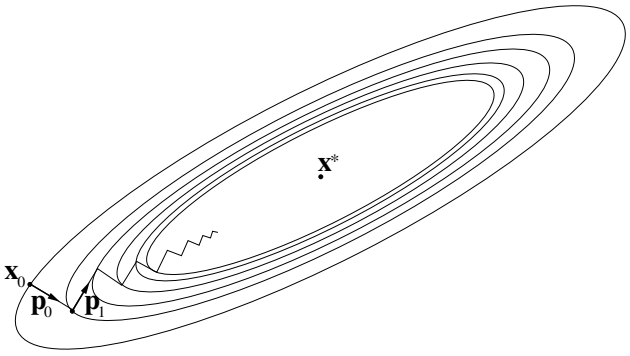

Figure 1: Trajectory of steepest descent.

Looking at Figure 1, we see that here is one good, but very precarious case in which convergence to the true solution $\mathbf{x}^*$ occurs blindingly fast, in a single step. This happens when the starting point $\mathbf{x}_0$ is at one apex (tip of either axis) of an isocontour ellipse. In that case, the gradient points exactly towards $\mathbf{x}^*$, and one iteration will lead to the minimum $\mathbf{x}^*$.

In all other cases, the line in the direction $\mathbf{p}_k$ of steepest descent, which is orthogonal to the isocontour at $\mathbf{x}_k$, will not pass through $\mathbf{x}^*$. 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 step is orthogonal to the latter isocontour (that is, parallel to the gradient). Thus, at every step the steepest descent trajectory is forced to make a ninety-degree turn. If isocontours were circles ($\sigma_1 = \sigma_n$) centered at $\mathbf{x}^*$, then the first turn would make the new direction point to $\mathbf{x}^*$, 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 $\mathbf{x}^*$, 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.

Thus, the directions of steepest descent are typically poor directions, with the only exceptions of starting at one of the axes of an isocontour ellipsoid or moving among hyperspheres rather than ellipsoids. Nonetheless, steepest 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). The technique of *preconditioning* can improve things by deforming the function $f(\mathbf{x})$ so that its isocontours look closer to hyperspheres. This technique is beyond the scope of this introductory note.

To complete the steepest descent algorithm we need to specify two more of its aspects: (i) How to determine the step size α_k so at to reach the minimum of $f(\mathbf{x})$ along the direction of $\mathbf{p}_k$; and (ii) how to check whether a minimum of f (in any direction) has been reached.

Line Search is a method to find α_k so that a minimum in the direction $\mathbf{p}_k$ is reached. Here is how line search works. Let

$$h(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k) \tag{3}$$

be the scalar function of one variable that is obtained by restricting the function f to the line through the current point $\mathbf{x}_k$ and in the direction of $\mathbf{p}_k$. Line search first determines two points a, c that bracket the desired step size α_k where f achieves a minimum, in the sense that $a \leq \alpha_k \leq c$, and then picks a point between a and c, say, $b = (a + c)/2$. The only difficulty here is to find c. In fact, we can set $a = 0$, corresponding through equation (3) to the starting point $\mathbf{x}_k$. A point c that is on the opposite side of the minimum with respect to a can be found by increasing α through values $\alpha_1 = a, \alpha_2, \ldots$ until $h(\alpha_i)$ is greater than $h(\alpha_{i-1})$. Then, if we can assume that h is convex between α_1 and α_i, we can set $c = \alpha_i$. In fact, 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 the minimum must be somewhere between a and c. Of course, if we cannot assume convexity, we may find the wrong minimum, but there is no general-purpose fix to this problem.

Line search now proceeds by shrinking the bracketing triple (a, b, c) until $c-a$ is smaller than the desired accuracy in determining α_k. Shrinking works as follows:

```
if b − a > c − b
    u = (a + b)/2
    if f(u) > f(b)
        (a, b, c) = (u, b, c)
    otherwise
        (a, b, c) = (a, u, b)
    end
otherwise
    u = (b + c)/2
    if f(u) > f(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 bracketing triple (a, b, c) preserves the property that $f(b) \leq f(a)$ and $f(b) \leq f(c)$, and therefore the minimum is somewhere between a and c.

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[1]

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

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.

Termination Check One criterion to check whether we are done with minimization is to verify whether the value of $f(\mathbf{x}_k)$ has significantly decreased from $f(\mathbf{x}_{k-1})$. Another is to check whether $\mathbf{x}_k$ is significantly different from $\mathbf{x}_{k-1}$. Close to the minimum, the derivatives of f are close to zero, so $|f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})|$ may be very small but $\|\mathbf{x}_k - \mathbf{x}_{k-1}\|$ may still be relatively large. Thus, the check on $\mathbf{x}_k$ is more stringent, and therefore preferable in most cases. In fact, usually one is interested in the value of $\mathbf{x}^*$, rather than in that of $f(\mathbf{x}^*)$. In summary, the steepest descent algorithm can be stopped when

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

where the positive constant ϵ is provided by the user.

2.1 The Convergence Speed of Steepest Descent

How much closer does one step of steepest descent bring us to the solution $\mathbf{x}^*$? In other words, how much smaller is $f(\mathbf{x}_{k+1})$, relative to the value $f(\mathbf{x}_k)$ at the previous step? The answer is, often not much, in a sense made more precise by the following result, proven in Appendix C. While the result holds for quadratic functions, any smooth function can be approximated by a quadratic function in small neighborhoods.

Theorem 2.1. *Let*

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

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

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T Q \mathbf{g}_k} \mathbf{g}_k \tag{4}$$

where

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

This trajectory converges to the unique minimum point

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

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

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

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 $\mathbf{x}^*$, and the more steps are required for convergence.

For general (that is, non-quadratic) f, the analysis above applies once $\mathbf{x}_k$ 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 $\mathbf{x}$, and is called the *Hessian* of f. In summary, steepest descent is good for functions that have a well conditioned Hessian near the minimum, but can become arbitrarily slow for poorly conditioned Hessians.

To characterize the speed of convergence of different minimization algorithms, we introduce the notion of the *order of convergence*. This is defined as the largest value of q for which the

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

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

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

for a minimization method of order q. In other words, the distance of $\mathbf{x}_k$ from $\mathbf{x}^*$ is reduced by the q-th power at every step, so the higher the order of convergence, the better. Theorem 2.1 implies that steepest descent has at best a linear order of convergence. In fact, the residuals $|f(\mathbf{x}_k) - f(\mathbf{x}^*)|$ in the *values* of the function being minimized converge linearly. Since the gradient of f approaches zero when $\mathbf{x}_k$ tends to $\mathbf{x}^*$, the *arguments* $\mathbf{x}_k$ to f can converge to $\mathbf{x}^*$ even more slowly.

3 Newton's Method

If a function can be well approximated by a paraboloid in the region in which minimization is performed, the analysis in the previous section suggests a straight-forward fix to the slow convergence of steepest descent. In fact, equation (2) tells us how to jump in one step from the starting point $\mathbf{x}_0$ to the minimum $\mathbf{x}^*$. Of course, when $f(\mathbf{x})$ is not exactly a paraboloid, the new value $\mathbf{x}_1$ will be different from $\mathbf{x}^*$. Consequently, iterations are needed, but convergence can be expected to be faster. This is the idea of Newton's method, which we now summarize. Let

$$f(\mathbf{x}_k + \Delta\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T Q_k \Delta\mathbf{x} \tag{5}$$

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

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

and

$$Q_k = Q(\mathbf{x}_k) = \left.\frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}^T}\right|_{\mathbf{x}=\mathbf{x}_k} = \left[\begin{array}{ccc} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{array}\right]_{\mathbf{x}=\mathbf{x}_k}$$

are the gradient and Hessian of f evaluated at the current point $\mathbf{x}_k$. Notice that even when f is a paraboloid, the gradient $\mathbf{g}_k$ is different from $\mathbf{a}$ as used in equation (1). This is because $\mathbf{a}$ and Q are the coefficients of the Taylor expansion of f around point $\mathbf{x} = 0$, while $\mathbf{g}_k$ and Q_k are the coefficients of the Taylor expansion of f around the *current* point $\mathbf{x}_k$. In other words, gradient and Hessian are constantly reevaluated in Newton's method.

To the extent that approximation (5) is valid, we can set the derivatives of $f(\mathbf{x}_k + \Delta\mathbf{x})$ with respect to $\Delta\mathbf{x}$ to zero, and obtain, analogously to equation (2), the linear system

$$Q_k \Delta\mathbf{x} = -\mathbf{g}_k \ , \tag{6}$$

whose solution $\Delta\mathbf{x}_k = \alpha_k \mathbf{p}_k$ yields at the same time the step direction $\mathbf{p}_k = \Delta\mathbf{x}_k / \|\Delta\mathbf{x}_k\|$ and the step size $\alpha_k = \|\Delta\mathbf{x}_k\|$. 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 (6) is the *Newton step*.

The greater speed of Newton's method over steepest descent is borne out by analysis: while steepest descent has a linear order of convergence, Newton's method is quadratic. To see this, let

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

be the place reached by a Newton step starting at $\mathbf{x}$ (see equation (6)), and suppose that at the minimum $\mathbf{x}^*$ the Hessian $Q(\mathbf{x}^*)$ is nonsingular. Then

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

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

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

From the mean-value theorem, we have

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

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

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

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

For a quadratic function, as in equation (1), 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 Q_k to be evaluated, for a total of $n + \binom{n}{2}$ derivatives. In addition, the Hessian must be inverted, or, at least, system (6) must be solved. For very large problems, in which the dimension n of $\mathbf{x}$ is thousands or more, storing and manipulating a Hessian can be prohibitive. In contrast, steepest descent requires 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.

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 2. The reasoning for the opposite case is analogous.

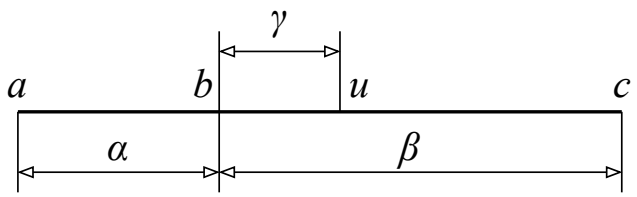

Figure 2: If points b and u are placed so that the ratio $w = \beta/\alpha$ equals the ratio α/γ, then we also have $(\beta - \gamma)/\gamma = w$, and $w \approx 1.618$ remains the same at every iteration of line search.

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 Steepest Descent on a Paraboloid

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

Let

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

Then we have

$$\tilde{e}(\mathbf{x}) = f(\mathbf{x}) - c + \frac{1}{2}\mathbf{x}^{*T} Q \mathbf{x}^* = f(\mathbf{x}) - f(\mathbf{x}^*) \tag{7}$$

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

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

and so

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

and therefore

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

We can also write

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

The result (7),

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

is rather interesting in itself. It says that adding a linear term $\mathbf{a}^T\mathbf{x}$ (and a constant c) to a paraboloid $\frac{1}{2}\mathbf{x}^T Q \mathbf{x}$ merely shifts the bottom of the paraboloid, both in position ($\mathbf{x}^*$ rather than $\mathbf{0}$) and value ($c - \frac{1}{2}\mathbf{x}^{*T} Q \mathbf{x}^*$ 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

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

that is, we can shift the origin of the domain to $\mathbf{x}^*$, 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 $\mathbf{x}$ 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 steepest descent algorithm find this minimum by starting from the initial point

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

At every iteration k, the algorithm chooses the direction of steepest 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

$$\alpha_k = -\frac{(Q\mathbf{y}_k)^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = -\frac{\mathbf{g}_k^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = \|\mathbf{g}_k\|\frac{\mathbf{p}_k^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k} = \|\mathbf{g}_k\|\frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T Q\mathbf{g}_k} \ . \tag{8}$$

Thus, the basic step of our steepest 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,

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T Q\mathbf{g}_k}\mathbf{g}_k \ . \tag{9}$$

C Proof of Theorem 2.1

This Appendix proves the main result on the convergence speed of steepest descent. 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 C.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

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

we have

$$\frac{(\mathbf{y}^T\mathbf{y})^2}{\mathbf{y}^T Q^{-1}\mathbf{y}\,\mathbf{y}^T Q\mathbf{y}} = \frac{(\mathbf{y}^T U^T U\mathbf{y})^2}{\mathbf{y}^T U\Sigma^{-1}U^T\mathbf{y}\,\mathbf{y}^T U\Sigma U^T\mathbf{y}} = \frac{(\mathbf{z}^T\mathbf{z})^2}{\mathbf{z}^T\Sigma^{-1}\mathbf{z}\,\mathbf{z}^T\Sigma\mathbf{z}} = \frac{1/\sum_{i=1}^n \theta_i\sigma_i}{\sum_{i=1}^n \theta_i/\sigma_i} = \frac{\phi(\sigma)}{\psi(\sigma)} \tag{10}$$

where the coefficients

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

add up to one. If we let

$$\sigma = \sum_{i=1}^n \theta_i\sigma_i \;, \tag{11}$$

then the numerator $\phi(\sigma)$ in (10) 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 (10) 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 (10) must be in the shaded area in figure 3. 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

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

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 (11). 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 Theorem 2.1 on the convergence of the method of steepest descent. The theorem is restated here for convenience.

Let

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

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

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

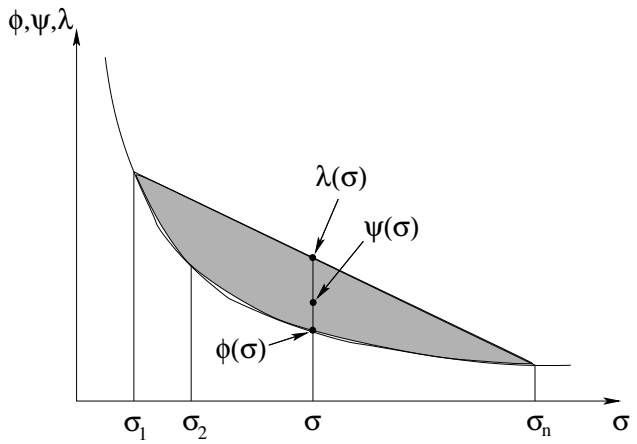

Figure 3: Kantorovich inequality.

where

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

This trajectory converges to the unique minimum point

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

of f. Furthermore, at every step k there holds

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

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

Proof. From the definition of e and from equation (9) 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:

$$e(\mathbf{y}_{k+1}) = \left(1 - \frac{(\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{g}_k^T Q^{-1} \mathbf{g}_k \, \mathbf{g}_k^T Q \mathbf{g}_k}\right) e(\mathbf{y}_k) \tag{13}$$

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 steepest descent. From the definitions

$$\mathbf{y} = \mathbf{x} - \mathbf{x}^* \quad \text{and} \quad e(\mathbf{y}) = \frac{1}{2} \mathbf{y}^T Q \mathbf{y} \tag{14}$$

we immediately obtain the expression for steepest descent in terms of f and $\mathbf{x}$. By equations (7) and (13) and the Kantorovich inequality we obtain

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) = e(\mathbf{y}_{k+1}) = \left(1 - \frac{(\mathbf{g}_k^T \mathbf{g}_k)^2}{\mathbf{g}_k^T Q^{-1} \mathbf{g}_k \, \mathbf{g}_k^T Q \mathbf{g}_k}\right) e(\mathbf{y}_k) \leq \left(1 - \frac{4\sigma_1 \sigma_n}{(\sigma_1 + \sigma_n)^2}\right) e(\mathbf{y}_k) \tag{15}$$

$$= \left(\frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n}\right)^2 (f(\mathbf{x}_k) - f(\mathbf{x}^*)) \; . \tag{16}$$

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

D Conjugate Gradients

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

Newton's method converges faster (quadratically) than steepest descent (linear convergence rate) because it uses more information about the function f being minimized. Steepest 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 (2) 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 (2) is never constructed explicitly, and the matrix Q is never stored. This is very important in

cases where $\mathbf{x}$ 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 $f(\mathbf{x})$ was introduced in equation (1). We know that in this case minimizing $f(\mathbf{x})$ is equivalent to solving the linear system (2). 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 $\mathbf{x}$. 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 (1) 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 $f(\mathbf{x})$. 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 steepest 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 (2), 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

$$f(\mathbf{x}) = c + \mathbf{a}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T Q\mathbf{x} \tag{17}$$

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

$$Q\mathbf{x} = -\mathbf{a}\ . \tag{18}$$

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

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

that is, the gradient of $f(\mathbf{x})$, evaluated at n different points $\mathbf{x}_1, \ldots, \mathbf{x}_n$. 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 $\mathbf{x}$ in $f(\mathbf{x})$ is a three-dimensional vector. Then the quadratic function $f(\mathbf{x})$ is constant over ellipsoids, called *isosurfaces*, centered at the minimum $\mathbf{x}^*$. How can we start from a point $\mathbf{x}_0$ on one of these ellipsoids and reach $\mathbf{x}^*$ by a finite sequence of one-dimensional searches? In connection with steepest 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 $\mathbf{x}_0$ to $\mathbf{x}_1$, and the line between $\mathbf{x}_0$ and $\mathbf{x}_1$ is tangent to an isosurface at $\mathbf{x}_1$. 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 $\mathbf{x}^*$ 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 $\mathbf{x}^*$. In fact, 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

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

where

$$\mathbf{y} = \Sigma^{1/2}U^T\mathbf{x} \; . \tag{19}$$

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

$$\mathbf{p}_i^T Q\mathbf{p}_j = 0 \; . \tag{20}$$

This condition is called Q-*conjugacy*, or Q-*orthogonality*: if equation (20) 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 (19) 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 $\mathbf{x}_0$ to $\mathbf{x}_n$:

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

where

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

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

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,

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

so that

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

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 $\mathbf{x}_{k+1}$, while the gradient $\mathbf{g}_{k+1}$ is orthogonal to the isosurface at $\mathbf{x}_{k+1}$. 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 (17) in n steps, without ever constructing the Hessian explicitly. When the function $f(\mathbf{x})$ is arbitrary, the same algorithm can be used.

However, n iterations will not suffice. In fact, the Hessian, which was constant for the quadratic case, now is a function of $\mathbf{x}_k$. 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 $\mathbf{x}^*$, the quadratic approximation becomes more and more valid, and a few cycles of n iterations each will achieve convergence.