Chapter 3

Function Optimization

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 \( f(x) = 0 \), so instead we minimize the norm \( \|f(x)\| \), which is a scalar function of the unknown vector \( x \).

We have encountered the first two situations when talking about 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. While these problems admit no exact solution, they often admit a multitude of approximate solutions. 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 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. On the other hand, the exact system (the one with perfect coefficients) is often close to being underdetermined. For instance, the images may be insufficient to recover a certain shape under a certain 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.

The term “optimization” is meant to subsume both minimization and maximization. However, maximizing the scalar function \( f(x) \) is the same as minimizing \( -f(x) \), 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: \( x^* \) is a global minimum of \( f \) if for every other \( x \) we have \( f(x) \geq f(x^*) \). 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 \( x_0 \), and we descend in the landscape of \( f(x) \) 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 \( x_0 \) that is close to \( x^* \). 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 \( x_0 \) when looking for the new minimum, that is, when computing the required control signal. More formally, we reach the correct minimum \( x^* \) as long as the initial point \( x_0 \) is in the basin of attraction of \( x^* \), defined as the largest neighborhood of \( x^* \) in which \( f(x) \) is convex.

Good references for the discussion in this chapter are *Matrix Computations*, *Practical Optimization*, and...
CHAPTER 3. FUNCTION OPTIMIZATION

Numerical Recipes in C, all of which are listed with full citations in section 1.4.

3.1 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:

\[
\begin{align*}
  k &= 0 \\
  \text{while } &\mathbf{x}_k \text{ is not a minimum} \Rightarrow \\
  &\text{compute step direction } \mathbf{p}_k \text{ with } \|\mathbf{p}_k\| = 1 \\
  &\text{compute step size } \alpha_k \\
  &\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
  &k = k + 1
\end{align*}
\]

end.

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

It is intuitively clear that the choice of the step size \( \alpha_k \) 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, as we now show. Consider a simple but important case,

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

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.

Of course, if \( f \) were this simple, no descent methods would be necessary. In fact 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

\[
Q \mathbf{x} = -\mathbf{a}.
\]

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. In fact, 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 (4.1), and that at every step we choose the direction of steepest descent. In order to simplify the mathematics, we observe that if we let

\[
\mathbf{e}(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T Q(\mathbf{x} - \mathbf{x}^*)
\]
then we have
\[ \tilde{e}(x) = f(x) - c + \frac{1}{2} x^T Q x^* = f(x) - f(x^*) \]  
so that \( \tilde{e} \) and \( f \) differ only by a constant. In fact,
\[ \tilde{e}(x) = \frac{1}{2} (x^T Q x + x^* T Q x^* - 2x^T Q x^*) = \frac{1}{2} x^T Q x + a^T x + \frac{1}{2} x^* T Q x^* = f(x) - c + \frac{1}{2} x^* T Q x^* \]
and from equation (4.2) we obtain
\[ f(x^*) = c + a^T x^* + \frac{1}{2} x^* T Q x^* = c - \frac{1}{2} x^* T Q x^* + \frac{1}{2} x^* T Q x^* = c - \frac{1}{2} x^* T Q x^* . \]
The result (4.3),
\[ \tilde{e}(x) = f(x) - f(x^*), \]
is rather interesting in itself. It says that adding a linear term \( a^T x \) (and a constant \( c \)) to a paraboloid \( \frac{1}{2} x^T Q x \) merely shifts the bottom of the paraboloid, both in position (\( x^* \) rather than \( 0 \)) and value (\( c - \frac{1}{2} x^* T Q x^* \) 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
\[ y = x - x^* , \]
that is, we can shift the origin of the domain to \( x^* \), and study the function
\[ e(y) = \frac{1}{2} y^T Q y \]
instead of \( f \) or \( \tilde{e} \), without loss of generality. We will transform everything back to \( f \) and \( x \) once we are done. Of course, by construction, the new minimum is at
\[ y^* = 0 \]
where \( e \) reaches a value of zero:
\[ e(y^*) = e(0) = 0. \]
However, we let our steepest descent algorithm find this minimum by starting from the initial point
\[ y_0 = x_0 - x^* . \]
At every iteration \( k \), the algorithm chooses the direction of steepest descent, which is in the direction
\[ p_k = - \frac{g_k}{\| g_k \|} \]
opposite to the gradient of \( e \) evaluated at \( y_k \):
\[ g_k = g(y_k) = \frac{\partial e}{\partial y} \bigg|_{y = y_k} = Q y_k . \]
We select for the algorithm the most favorable step size, that is, the one that takes us from \( y_k \) to the lowest point in the direction of \( p_k \). This can be found by differentiating the function
\[ e(y_k + \alpha p_k) = \frac{1}{2} (y_k + \alpha p_k)^T Q (y_k + \alpha p_k) \]
with respect to \( \alpha \), and setting the derivative to zero to obtain the optimal step \( \alpha_k \). We have
\[ \frac{\partial e(y_k + \alpha p_k)}{\partial \alpha} = (y_k + \alpha p_k)^T Q p_k \]
and setting this to zero yields
\[
\alpha_k = -\frac{(Qy_k)^T p_k}{p_k^T Qp_k} = -\frac{g_k^T p_k}{p_k^T Qp_k} = \|g_k\| \frac{p_k^T p_k}{p_k^T Qp_k} = \|g_k\| \frac{g_k^T g_k}{g_k^T Qg_k}.
\] (3.4)

Thus, the basic step of our steepest descent can be written as follows:
\[y_{k+1} = y_k + \|g_k\| \frac{g_k^T g_k}{g_k^T Qg_k} p_k\]
that is,
\[y_{k+1} = y_k - \frac{g_k^T g_k}{g_k^T Qg_k} g_k.\] (3.5)

How much closer did this step bring us to the solution \(y^* = 0\)? In other words, how much smaller is \(e(y_{k+1})\), relative to the value \(e(y_k)\) at the previous step? The answer is, often not much, as we shall now prove. The arguments and proofs below are adapted from D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1973.

From the definition of \(e\) and from equation (4.5) we obtain
\[
\frac{e(y_k) - e(y_{k+1})}{e(y_k)} = \frac{y_k^T Qy_k - y_{k+1}^T Qy_{k+1}}{y_k^T Qy_k} = \frac{y_k^T Qy_k - (y_k - \frac{g_k^T g_k}{g_k^T Qg_k} g_k)^T Q (y_k - \frac{g_k^T g_k}{g_k^T Qg_k} g_k)}{y_k^T Qy_k} = \frac{2 \frac{g_k^T g_k}{g_k^T Qg_k} g_k^T Qy_k - \left(\frac{g_k^T g_k}{g_k^T Qg_k}\right)^2 g_k^T Qg_k}{y_k^T Qy_k} = \frac{2g_k^T g_k y_k - g_k^T y_k g_k}{y_k^T Qy_k g_k^T Qg_k}.
\]

Since \(Q\) is invertible we have
\[g_k = Qy_k \Rightarrow y_k = Q^{-1} g_k\]
and
\[y_k^T Qy_k = g_k^T Q^{-1} g_k\]
so that
\[
\frac{e(y_k) - e(y_{k+1})}{e(y_k)} = \frac{(g_k^T g_k)^2}{g_k^T Q^{-1} g_k g_k^T Qg_k}.
\]

This can be rewritten as follows by rearranging terms:
\[e(y_{k+1}) = \left(1 - \frac{(g_k^T g_k)^2}{g_k^T Q^{-1} g_k g_k^T Qg_k}\right) e(y_k)\] (3.6)
so if we can bound the expression in parentheses we have a bound on the rate of convergence of steepest descent. To this end, we introduce the following result.

**Lemma 3.1.1 (Kantorovich inequality)** Let \(Q\) be a positive definite, symmetric, \(n \times n\) matrix. For any vector \(y\) there holds
\[
\frac{(y^T y)^2}{y^T Q^{-1} y y^T Q y} \geq \frac{4\sigma_1 \sigma_n}{(\sigma_1 + \sigma_n)^2}
\]
where \(\sigma_1\) and \(\sigma_n\) are, respectively, the largest and smallest singular values of \(Q\).
Proof. Let
\[ Q = UΣU^T \]
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
\[ σ_n = \min_{||y||=1} y^T Q y > 0 \]
by the definition of positive definiteness. If we let
\[ z = U^T y \]
we have
\[ \frac{(y^T y)^2}{y^T Q^{-1} y y^T Q y} = \frac{(y^T U^T U y)^2}{y^T UΣ^{-1} U^T y y^T UΣ U^T y} = \frac{(z^T z)^2}{z^T Σ^{-1} z Σ z} = \frac{1}{\sum_{i=1}^n θ_i σ_i} = φ(σ) \]
where the coefficients
\[ θ_i = \frac{z_i^2}{||z||^2} \]
add up to one. If we let
\[ σ = \sum_{i=1}^n θ_i σ_i \]
then the numerator \( φ(σ) \) in (4.7) is \( 1/σ \). 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 \( θ_j = 1 \) and all other \( θ_i \) to zero. Thus, the values \( 1/σ_j \) for \( j = 1, \ldots, n \) are all on the curve \( 1/σ \). The denominator \( ψ(σ) \) in (4.7) is a convex combination of points on this curve. Since \( 1/σ \) is a convex function of \( σ \), the values of the denominator \( ψ(σ) \) of (4.7) must be in the shaded area in figure 4.1. This area is delimited from above by the straight line that connects point \((σ_1, 1/σ_1)\) with point \((σ_n, 1/σ_n)\), that is, by the line with ordinate
\[ λ(σ) = (σ_1 + σ_n - σ)/(σ_1 σ_n) \]

For the same vector of coefficients \( θ_i \), the values of \( φ(σ) \), \( ψ(σ) \), and \( λ(σ) \) are on the vertical line corresponding to the value of \( σ \) given by (4.8). Thus an appropriate bound is
\[ \frac{φ(σ)}{ψ(σ)} ≥ \min_{σ_1 ≤ σ ≤ σ_n} \frac{φ(σ)}{λ(σ)} = \min_{σ_1 ≤ σ ≤ σ_n} \frac{1/σ}{(σ_1 + σ_n - σ)/(σ_1 σ_n)} \]
The minimum is achieved at \( σ = (σ_1 + σ_n)/2 \), yielding the desired result. \( \Delta \)

Thanks to this lemma, we can state the main result on the convergence of the method of steepest descent.

**Theorem 3.1.2** Let
\[ f(x) = c + a^T x + \frac{1}{2} x^T Q x \]
be a quadratic function of \( x \), with \( Q \) symmetric and positive definite. For any \( x_0 \), the method of steepest descent
\[ x_{k+1} = x_k - \frac{g_k^T g_k}{g_k^T Q g_k} g_k \]
where
\[ g_k = g(x_k) = \frac{∂f}{∂x} \bigg|_{x=x_k} = a + Q x_k \]
converges to the unique minimum point
\[ x^* = -Q^{-1}a \]
of \( f \). Furthermore, at every step \( k \) there holds
\[ f(x_{k+1}) - f(x^*) \leq \left( \frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n} \right)^2 (f(x_k) - f(x^*)) \]
where \( \sigma_1 \) and \( \sigma_n \) are, respectively, the largest and smallest singular value of \( Q \).

**Proof.** From the definitions
\[ y = x - x^* \quad \text{and} \quad e(y) = \frac{1}{2} y^T Q y \]
we immediately obtain the expression for steepest descent in terms of \( f \) and \( x \). By equations (4.3) and (4.6) and the Kantorovich inequality we obtain
\[
\begin{align*}
f(x_{k+1}) - f(x^*) &= e(y_{k+1}) = \left( 1 - \frac{(g_k^T g_k)^2}{g_k^T Q^{-1} g_k} \right) e(y_k) \leq \left( 1 - \frac{4\sigma_1 \sigma_n}{(\sigma_1 + \sigma_n)^2} \right) e(y_k) \\
&= \left( \frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n} \right)^2 (f(x_k) - f(x^*)).
\end{align*}
\]
Since the ratio in the last term is smaller than one, it follows immediately that \( f(x_k) - f(x^*) \to 0 \) and hence, since the minimum of \( f \) is unique, that \( x_k \to x^* \).

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. It is easily seen why this happens in
3.1. LOCAL MINIMIZATION AND STEEPEST DESCENT

Figure 3.2: Trajectory of steepest descent.

the case in which $x$ is a two-dimensional vector, as in figure 4.2, which shows the trajectory $x_k$ superimposed on a set of isocontours of $f(x)$.

There is one good, but very precarious case, namely, when the starting point $x_0$ is at one apex (tip of either axis) of an isocontour ellipse. In that case, one iteration will lead to the minimum $x^*$. In all other cases, the line in the direction $p_k$ of steepest descent, which is orthogonal to the isocontour at $x_k$, will not pass through $x^*$. The minimum of $f$ along that line is tangent to some other, lower isocontour. 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 $x^*$, then the first turn would make the new direction point to $x^*$, and minimization would get there in just one more step. This case, in which $\kappa(Q) = 1$, is consistent with our analysis, because then

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

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 $x^*$, and the more steps are required for convergence.

For general (that is, non-quadratic) $f$, the analysis above applies once $x_k$ 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 $x$, 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

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^q}$$

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

$$\|x_{k+1} - x^*\| \approx \beta \|x_k - x^*\|^q$$

for a minimization method of order $q$. In other words, the distance of $x_k$ from $x^*$ is reduced by the $q$-th power at every step, so the higher the order of convergence, the better. Theorem 4.1.2 implies that steepest
descent has at best a linear order of convergence. In fact, the residuals \(|f(x_k) - f(x^*)|\) in the values of the function being minimized converge linearly. Since the gradient of \(f\) approaches zero when \(x_k\) tends to \(x^*\), the arguments \(x_k\) to \(f\) can converge to \(x^*\) even more slowly.

To complete the steepest descent algorithm we need to specify how to check whether a minimum has been reached. One criterion is to check whether the value of \(f(x_k)\) has significantly decreased from \(f(x_{k-1})\). Another is to check whether \(x_k\) is significantly different from \(x_{k-1}\). Close to the minimum, the derivatives of \(f\) are close to zero, so \(|f(x_k) - f(x_{k-1})|\) may be very small but \(|x_k - x_{k-1}|\) may still be relatively large. Thus, the check on \(x_k\) is more stringent, and therefore preferable in most cases. In fact, usually one is interested in the value of \(x^*\), rather than in that of \(f(x^*)\). In summary, the steepest descent algorithm can be stopped when

\[
\|x_k - x_{k-1}\| < \epsilon
\]

where the positive constant \(\epsilon\) is provided by the user.

In our analysis of steepest descent, we used the Hessian \(Q\) in order to compute the optimal step size \(\alpha\) (see equation (4.4)). We used \(Q\) because it was available, but its computation during steepest descent would in general be overkill. In fact, only gradient information is necessary to find \(p_k\), and a line search in the direction of \(p_k\) can be used to determine the step size \(\alpha_k\). In contrast, the Hessian of \(f(x)\) requires computing \(\binom{\epsilon}{2}\) second derivatives if \(x\) is an \(n\)-dimensional vector.

Using line search to find \(\alpha_k\) guarantees that a minimum in the direction \(p_k\) is actually reached even when the parabolic approximation is inadequate. Here is how line search works.

Let

\[
h(\alpha) = f(x_k + \alpha p_k)
\]

be the scalar function of one variable that is obtained by restricting the function \(f\) to the line through the current point \(x_k\) and in the direction of \(p_k\). Line search first determines two points \(a, c\) that bracket the desired minimum \(\alpha_k\), in the sense that \(a \leq \alpha_k \leq c\), and then picks a point between \(a\) and \(c\), say, \(b = (\alpha + c)/2\). The only difficulty here is to find \(c\). In fact, we can set \(a = 0\), corresponding through equation (4.13) to the starting point \(x_k\). A point \(c\) that is on the opposite side of the minimum with respect to \(a\) can be found by increasing \(\alpha\) 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 \(\alpha_1\) and \(\alpha_i\), 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 \(\alpha_{i-1}\) 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 \(\alpha_k\). Shrinking works as follows:

```plaintext
if b - a > c - b
    u = (a + b)/2
    if f(u) > f(b)
        (a, b, c) = (u, b, c)
    else
        (a, b, c) = (a, u, b)
    end
else
    u = (b + c)/2
    if f(u) > f(b)
        (a, b, c) = (a, b, u)
    else
        (a, b, c) = (b, u, c)
    end
end
```
3.2. Newton’s Method

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 \(c\). In addition, at every step the interval \((a, c)\) shrinks to \(3/4\) of its previous size, so line search will find the minimum in a number of steps that is logarithmic in the desired accuracy.

3.2 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 (4.2) tells us how to jump in one step from the starting point \(x_0\) to the minimum \(x^*\). Of course, when \(f(x)\) is not exactly a paraboloid, the new value \(x_1\) will be different from \(x^*\). 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

\[
f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k) \cdot \Delta x + \frac{1}{2} \Delta x^T Q_k \Delta x
\]

be the first terms of the Taylor series expansion of \(f\) about the current point \(x_k\), where

\[
\nabla f(x_k) = \left. \frac{\partial f}{\partial x} \right|_{x=x_k}
\]

and

\[
Q_k = Q(x_k) = \left. \frac{\partial^2 f}{\partial x \partial x^T} \right|_{x=x_k} = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_m \partial x_m}
\end{bmatrix}
\]

are the gradient and Hessian of \(f\) evaluated at the current point \(x_k\). Notice that even when \(f\) is a paraboloid, the gradient \(g_k\) is different from \(a\) as used in equation (4.1). In fact, \(a\) and \(Q\) are the coefficients of the Taylor expansion of \(f\) around point \(x = 0\), while \(g_k\) and \(Q_k\) are the coefficients of the Taylor expansion of \(f\) around the current point \(x_k\). In other words, gradient and Hessian are constantly reevaluated in Newton’s method.

To the extent that approximation (4.14) is valid, we can set the derivatives of \(f(x_k + \Delta x)\) with respect to \(\Delta x\) to zero, and obtain, analogously to equation (4.2), the linear system

\[
Q_k \Delta x = -\nabla f(x_k)
\]

whose solution \(\Delta x_k = \alpha_k p_k\) yields at the same time the step direction \(p_k = \Delta x_k/\|\Delta x_k\|\) and the step size \(\alpha_k = \|\Delta x_k\|\). 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 \(p_k\) and size \(\alpha_k\) are defined in this manner is called Newton’s method. The corresponding \(p_k\) is termed the Newton direction, and the step defined by equation (4.15) 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. In fact, let

\[
y(x) = x - Q(x)^{-1} \nabla f(x)
\]

be the place reached by a Newton step starting at \(x\) (see equation (4.15)), and suppose that at the minimum \(x^*\) the Hessian \(Q(x^*)\) is nonsingular. Then

\[
y(x^*) = x^*
\]
because \( g(x^*) = 0 \), and 
\[
x_{k+1} - x^* = y(x_k) - x^* = y(x_k) - y(x^*) .
\]
From the mean-value theorem, we have
\[
\|x_{k+1} - x^*\| = \|y(x_k) - y(x^*)\| \leq \left\| \left[ \frac{\partial y}{\partial x^T} \right] _{x=x^*} (x_k - x^*) \right\| + \frac{1}{2} \left\| \frac{\partial^2 y}{\partial x\partial x^T} \right\| _{x=x^*} \|x_k - x^*\|^2
\]
where \( \hat{x} \) is some point on the line between \( x^* \) and \( x_k \). Since \( y(x^*) = x^* \), the first derivatives of \( y \) at \( x^* \) are zero, so that the first term in the right-hand side above vanishes, and
\[
\|x_{k+1} - x^*\| \leq c \|x_k - x^*\|^2
\]
where \( c \) depends on third-order derivatives of \( f \) near \( x^* \). Thus, the convergence rate of Newton’s method is of order at least two.

For a quadratic function, as in equation (4.1), 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 \( g_k \) 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 (4.15) must be solved. For very large problems, in which the dimension \( n \) of \( x \) is thousands or more, storing and manipulating a Hessian can be prohibitive. In contrast, steepest descent requires the gradient \( g_k \) for selecting the step direction \( p_k \), and a line search in the direction \( p_k \) to find the step size. The method of conjugate gradients, discussed in the next section, 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.

### 3.3 Conjugate Gradients

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 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 (4.2) 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 (4.2) is never constructed explicitly, and the matrix \( Q \) is never stored. This is very important in cases where \( x \) 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-Ribi` ere 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 \( f(x) \) was introduced in equation (4.1).
3.3. CONJUGATE GRADIENTS

We know that in this case minimizing $f(x)$ is equivalent to solving the linear system (4.2). 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 $x$. 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 (4.1) 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 $f(x)$. 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 (4.2), 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.

3.3.1 The Quadratic Case

Suppose that we want to minimize the quadratic function

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

(3.16)

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

$$Q x = -a.$$ 

(3.17)

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

$$g_k = Q x_k + a$$

that is, the gradient of $f(x)$, evaluated at $n$ different points $x_1, \ldots, x_n$. 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.


Consider the case $n = 3$, in which the variable $x$ in $f(x)$ is a three-dimensional vector. Then the quadratic function $f(x)$ is constant over ellipsoids, called isosurfaces, centered at the minimum $x^*$. How can we start from a point $x_0$ on one of these ellipsoids and reach $x^*$ 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 $x_0$ to $x_1$, and the line between $x_0$ and $x_1$ is tangent to an isosurface at $x_1$. The next step is in the direction of the gradient, so that the new direction $p_1$ is orthogonal to the previous direction $p_0$. This would then take us to $x^*$ right away. Suppose however that we cannot afford to compute this special direction $p_1$ orthogonal
CHAPTER 3. FUNCTION OPTIMIZATION

to \( p_0 \), but that we can only compute some direction \( p_1 \) orthogonal to \( p_0 \) (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 \( x^* \). 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 \( p_0 \), the second step then yields the minimum in the space spanned by \( p_0 \) and \( p_1 \), and so forth. After \( n \) steps we must be done, since \( p_0, \ldots, p_{n-1} \) 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

\[
\frac{1}{2} x^T Q x = \frac{1}{2} y^T y
\]

where

\[
y = \Sigma^{1/2} U^T x.
\]  

(3.18)

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 \( q_i \) and \( q_j \) are orthogonal in the spherical context, that is, if

\[
q_i^T q_j = 0,
\]

what does this translate into in terms of the directions \( p_i \) and \( p_j \) for the ellipsoidal problem? We have

\[
q_{i,j} = \Sigma^{1/2} U^T p_{i,j},
\]

so that orthogonality for \( q_{i,j} \) becomes

\[
p_i^T \Sigma^{1/2} \Sigma^{1/2} U^T p_j = 0
\]
or

\[
p^T Q p_j = 0.
\]  

(3.19)

This condition is called \( Q \)-conjugacy, or \( Q \)-orthogonality: if equation (4.19) holds, then \( p_i \) and \( p_j \) 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 \( p_0, \ldots, p_{n-1} \) that are mutually conjugate, and if we do line minimization along each direction \( p_k \), we reach the minimum in at most \( n \) steps. Of course, we cannot use the transformation (4.18) in the algorithm, because \( \Sigma \) 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 \( x_0 \) to \( x_n \):

\[
g_0 = g(x_0)
p_0 = -g_0
\]

for \( k = 0, \ldots, n-1 \)

\[
\alpha_k = \arg \min_{\alpha \geq 0} f(x_k + \alpha p_k)
\]

\[
x_{k+1} = x_k + \alpha_k p_k
\]

\[
g_{k+1} = g(x_{k+1})
\]

\[
\gamma_k = \frac{g_{k+1}^T Q p_k}{p_{k}^T Q p_k}
\]

\[
p_{k+1} = -g_{k+1} + \gamma_k p_k
\]

end
3.3. CONJUGATE GRADIENTS

where

\[ \mathbf{g}_k = \mathbf{g}(\mathbf{x}_k) = \frac{\partial f}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_k} \]

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,

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

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 Gram-Schmidt (theorem 2.4.2) to produce conjugate rather than orthogonal vectors. Details can be found in Polak’s book mentioned earlier.

3.3.2 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 \( \gamma_k \) contains the Hessian \( \mathbf{Q} \), which is too large. We now show that \( \gamma_k \) can be rewritten in terms of the gradient values \( \mathbf{g}_k \) and \( \mathbf{g}_{k+1} \) only. To this end, we notice that

\[
\begin{align*}
\mathbf{g}_{k+1} &= \mathbf{g}_k + \alpha_k \mathbf{Q} \mathbf{p}_k, \\
\alpha_k \mathbf{Q} \mathbf{p}_k &= \mathbf{g}_{k+1} - \mathbf{g}_k.
\end{align*}
\]

In fact,

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

so that

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

We can therefore write

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

and \( \mathbf{Q} \) has disappeared.

This expression for \( \gamma_k \) 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 \( \gamma_k \) 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-Ribiére formula

\[
\gamma_k = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k}.
\]
3.3.3 Extension to General Functions

We now know how to minimize the quadratic function (4.16) in $n$ steps, without ever constructing the Hessian explicitly. When the function $f(x)$ 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 $x_k$. Strictly speaking, we then lose conjugacy, since $p_k$ and $p_{k+1}$ are associated to different Hessians. However, as the algorithm approaches the minimum $x^*$, the quadratic approximation becomes more and more valid, and a few cycles of $n$ iterations each will achieve convergence.