Lecture Notes for

AMSC/CMSC 660, Scientific Computing I

By

Jian-Guo Liu and Simon P. Schurr

jliu@math.umd.edu, spschurr@math.umd.edu

Applied Mathematics and Scientific Computation Program

University of Maryland at College Park

November 2001

Chapter Two - Optimization

1 Introduction

In this chapter we concentrate on the optimization (i.e. minimization or maximization) of nonlinear functions of n variables. This is currently an extremely active area of research, not only due to the enormous number of applications of optimization, but also because many non-optimization problems can be formulated and solved as optimization problems: As an example, consider the problem of finding a solution $x \in \mathbf{R}^n$ - assuming one exists - to the following system of equations:

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ \vdots \qquad\qquad & \vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0, \end{aligned} \tag{1}$$

where $f_1, f_2, \cdots, f_n$ are nonlinear functions. This system can be written more compactly as $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{f} \colon \mathbf{R}^n \to \mathbf{R}^n$. If we define $g(x) \equiv f_1^2 + f_2^2 + \cdots + f_n^2$, and can find an x^* such that g achieves its minimum value - which is 0 - at $x = x^*$, then $f_1 = f_2 = \cdots = f_n = 0$ at $x = x^*$. Hence x^* solves (1). Unfortunately this particular approach is not practical, because g 'inherits' the behaviors of each of the f_i. Therefore if some of the f_i are not well behaved, then g will certainly not be well behaved. In addition, even if we have an algorithm which produces a minimizer x^* of g, it is possible that x^* doesn't satisfy (1). This is because x^* may be a *local* rather than a *global* minimizer of g. A local minimizer gives a better (i.e. lower) value for g than any other point nearby, but a global minimizer gives the minimum value $g = 0$ over all $x \in \mathbf{R}^n$.

Ideally we would like to find global minimizers of g, but this is usually not realistic in practice, and we are forced to settle for methods that find local minimizers. Unless stated otherwise, when we speak of minimizers, we will always be referring to local minimizers. It should be noted that any maximization problem can be converted into a minimization problem, and vice versa, so without loss of generality, we need only consider minimization problems.

Alternatively, if we can find a special function $h(x_1, x_2, \cdots, x_n)$ such that $\frac{\partial h}{\partial x_i} = f_i$ for each $i = 1, 2, \cdots, n$, then the roots of (1) correspond to stationary points of h. If these stationary points are maximizers or minimizers of h, we have transformed the root-finding problem (1) into an optimization problem.

Generally nonlinear problems in optimization cannot be solved via direct methods that yield an guaranteed optimal solution after a finite number of iterations. Instead we need to employ iterative methods that generate an infinite sequence of approximations to a solution of the problem. Of course we would like this sequence to converge to a limit point, and also to have the limit point be a solution of our original problem. We now introduce some important properties of sequences that will help us to characterize the speed with which they converge to a limit - assuming such a limit exists.

Consider the following two sequences of numbers: $\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \frac{1}{2^4}, \cdots$, and $\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^4}, \frac{1}{2^8}, \cdots$. It is clear that both sequences are converging to zero, but the second sequence appears to be converging faster. We can make this idea of 'faster convergence' more precise by introducing the idea of the *order of convergence* of a sequence:

Definition: A sequence of vectors $x_0, x_1, x_2, \cdots \in \mathbf{R}^n$ *converges locally to* ξ *with order* r if

$$\|x_{k+1} - \xi\| \leq C\|x_k - \xi\|^r$$

for all sufficiently large k, and some positive constant C independent of k.

In the above definition, any norm can be used. While the value of C (called the *rate*) is norm-dependent, the order of convergence is not. Relatively large values of r and small values of C correspond to faster convergence.

• If $r = 1$, we have what is known as *linear* convergence, provided we also have $C < 1$. (What happens if $r = 1$ but $C \geq 1$?)
• If $r > 1$, the sequence is said to possess *superlinear* convergence.
• In the special case of $r = 2$, we have *quadratic* convergence.

In the two examples of sequences given above, the limit of the iterates is $\xi = 0$. The first sequence converges linearly to 0, while the second sequence converges quadratically.

In the following sections we present some numerical methods that give rise to sequences that converge either linearly, superlinearly with $r \in (1, 2)$, or quadratically. Although quadratic convergence is an extremely desirable property, we shall see that it only occurs under very restrictive conditions. Superlinear convergence with $r \in (1, 2)$ is also quite attractive, but linear convergence can be problematic - especially if the constant C in the above definition (which must be between 0 and 1), is close to 1. In this case, the convergence is usually too slow to be of much practical value. However, linearly convergent methods do have their uses, and some will be presented in the next section. The main advantage of these methods is that under some conditions, they can be guaranteed to converge to a minimizer, even if the starting points of our method are not close to this minimizer. This is not generally the case for methods that are superlinearly convergent. Thus the issue of a *safeguarded* method arises. While we would like to exploit the fast local convergence of superlinear methods, such convergence is not realized until our iterates are sufficiently close to a minimizer. So we safeguard these superlinear methods with linear methods that hopefully guarantee some progress far from a minimizer. If we get to a stage where successive iterates are 'close enough' to each other, then we suspect that a minimizer is nearby, and would switch to a superlinear method that provides fast local convergence.

2 One-Dimensional Search Methods

The problem of minimizing a function f of one variable can be tackled in many different ways, depending on what type of function f is. In particular, it is important to know if we can obtain information about f and its derivatives at a reasonable cost. If f' is impossible or too expensive to compute, it is common to resort to so-called 'Search Methods' for estimating a minimum value of f. Roughly speaking, these methods consist of evaluating f at a number of points, and based on the behavior of these function values, we compute a new point at which to evaluate f. Hopefully the new point provides a better approximation to a minimizer of f. The question now arises as to how we choose starting points for this method. In particular, how can we find an initial 'interval of uncertainty', i.e. an interval of x values between which we can guarantee the existence of a local minimizer? The objective is to find three points x_1, x_2, and x_3, satisfying

$$x_1 < x_2 < x_3 \qquad \text{and} \qquad f(x_1) > f(x_2) < f(x_3). \tag{2}$$

In the absence of any other information, we could firstly evaluate f at $x = 0$ and $x = 1$. If $f(0) > f(1)$, we conjecture that an improvement in the value of f could be made in the positive direction from $x = 1$, so we evaluate f at $x = 2, 4, 8, 16, \cdots$ until we can find x_1, x_2, and x_3 satisfying (2). If $f(0) < f(1)$, it might be wiser to search the interval [0,1], so we evaluate f at $x = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16} \cdots$. If $f(0) = f(1)$ we are impartial to searching in either direction. If no progress is made using either search direction, we could try searching in the negative direction from $x = 0$.

In what follows, it will be assumed that we can eventually find an interval (x_1, x_3) in which there exists a local minimizer. We plan to reduce the length of this interval of uncertainty by intelligently choosing points in (x_1, x_3) at which to evaluate f. Some possible stopping criteria for such methods are: (1) the length of the current interval of uncertainty is less than some given tolerance ϵ, and, (2) the total number of evaluations of f has reached some given limit N. The second criterion is used when function evaluations are very expensive.

2.1 The Fibonacci Search Method

If function evaluations are very expensive, we will want to reduce the length of the interval of uncertainty as much as possible after a fixed number of evaluations, N.

Assume that we have found values of x_1, x_2, and x_3 satisfying (2), so that the continuity of f ensures that a minimizer of f lies between x_1 and x_3. In an attempt to reduce the interval of uncertainty, we now choose a point $x_4 \in (x_1, x_3)$ at which to evaluate f. We will assume that $x_4 > x_2$. Qualitatively, after computing $f(x_4)$, we have the two possible cases shown in Figure 1. For the purposes of illustrating this algorithm, we have assumed that f

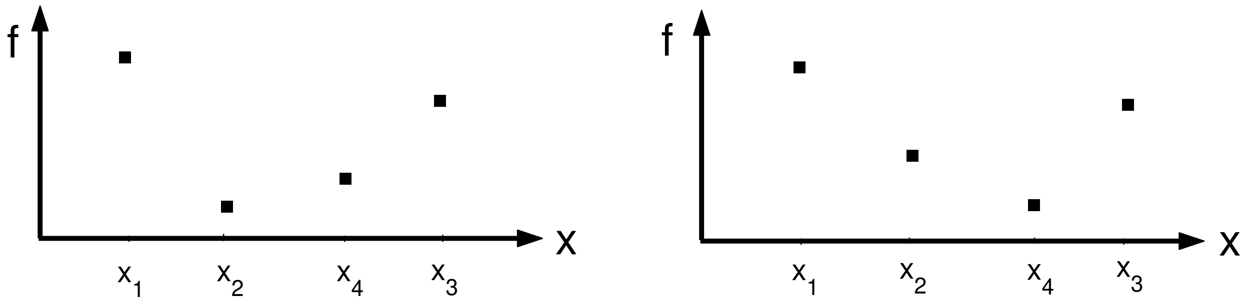

Figure 1: Two possible cases for a Fibonacci search

is unimodal in the interval of uncertainty, i.e. it has a unique extremum - which we will take to be a minimum. The relative sizes of $f(x_1)$ and $f(x_3)$ are not important here.

In the first case, $f(x_2) < f(x_4)$, so our new interval of uncertainty will be (x_1, x_4) with length $x_4 - x_1$. In the second case the new interval of uncertainty is (x_2, x_3) with length $x_3 - x_2$. Of course we do not know a priori which case will occur, so it makes sense to choose x_4 in such a way that the two intervals have equal length, and therefore no bias will be shown towards any particular interval. Hence

$$x_4 = x_1 + x_3 - x_2. \tag{3}$$

(The same result holds if we change our original assumption of $x_4 > x_2$ to $x_4 < x_2$.) Denote the number of function evaluations that we have made at any stage by n. This does not include the function evaluations $f(x_1), f(x_2)$, and $f(x_3)$, since these are assumed to be known at the start.

We then have the following algorithm that searches for a minimizer of f:

1. Initialize with $n = 0$ and x_1, x_2, and x_3 satisfying (2).
2. While $n < N$:
   - Compute x_4 according to (3).
   - Depending on the relative sizes of $f(x_2)$ and $f(x_4)$, update the values of x_1, x_2, and x_3 so that (2) holds, and the interval of uncertainly has been reduced in length.
   - Increase n by 1 (since we've evaluated f at x_4)

In this algorithm, the initial value of x_2 is fixed by whatever method we have used to find an initial interval of uncertainty. Is it possible to instead *choose* x_2 in such a way that the length of the interval of uncertainty decreases faster than it would in the above algorithm? The answer is 'yes'. Let the initial length of the interval (x_1, x_3) be α. It can be shown that if the initial $x_2 \in (x_1, x_3)$ is chosen so that it is a distance $\frac{F_{N-1}}{F_N}\alpha$ from x_1 or x_3, then the length of the final interval of uncertainty after N iterations (i.e. N function evaluations) is a minimum. F_{N-1} and F_N are *Fibonacci numbers* that satisfy the following recursive relation:

$$\begin{aligned} F_0 &= F_1 = 1, \\ F_k &= F_{k-1} + F_{k-2}, \qquad k = 2, 3, \cdots . \end{aligned} \tag{4}$$

The Fibonacci Search Method is simply the above algorithm, except that we initialize with the aforementioned improvement for calculating x_2:

$$x_2 = x_1 + \frac{F_{N-1}}{F_N}\alpha \ , \text{ or } \ x_2 = x_3 - \frac{F_{N-1}}{F_N}\alpha \ . \tag{5}$$

After k iterations of this method, the length of the interval of uncertainty is reduced to $\frac{F_{N-k}}{F_N}\alpha$.

2.2 The Golden Section Search Method

If we are given a tolerance ϵ such that the length of the interval of uncertainty to be made less than ϵ, but we aren't given a maximum number of function evaluations N, it is not possible to compute the initial interior point x_2 using (5). However it can be shown that (4) has an explicit solution given by

$$F_k = \frac{1}{\sqrt{5}}\left(\tau_+^{k+1} - \tau_-^{k+1}\right), \qquad k = 0, 1, \cdots, \qquad \tau_\pm = \frac{1 \pm \sqrt{5}}{2}.$$

$\tau_+ \approx 1.618$ is known as the 'Golden Ratio'. For (moderately) large values of k, we have that

$$|\tau_-| < 1 < |\tau_+| \ \Rightarrow \ |\tau_-|^k \ll 1 \ll |\tau_+|^k \ \Rightarrow \ F_k \approx \frac{1}{\sqrt{5}}\tau_+^{k+1} \ \Rightarrow \ \frac{F_{k-1}}{F_k} \approx \frac{1}{\tau_+}.$$

The Golden Section Search Method is obtained from the Fibonacci Search Method by replacing the ratio $\frac{F_{N-1}}{F_N}$ in (5) by $\frac{1}{\tau_+}$. As $N \to \infty$ this approximation becomes more accurate. It is not difficult to show that for the Golden Section Search Method, the length of the interval of uncertainty decreases at a constant rate of $1/\tau_+ \approx 0.618$ at each iteration. At each stage, the length of the interval of uncertainty is a measure of the maximum error in our approximation to the true solution. So in some sense, we can say that this method is linearly convergent, with rate $1/\tau_+$.

It should be noted that there are other methods for approximating minimizers of one-dimensional continuous functions without using derivatives. An example of an alternative is *quadratic interpolation*. Given values x_1, x_2, and x_3 that satisfy (2), one can find the quadratic polynomial that interpolates the points $(x_1, f(x_1)), (x_2, f(x_2))$ and $(x_3, f(x_3))$. We then define x_4 to be the minimizer of this polynomial. Either one of x_1 or x_3 is then discarded, so we are left with three points that still satisfy (2). Alternatively we could choose to always discard the point with the worst (i.e. largest) function value. However the resulting three points may not necessary satisfy (2).

In practice, the assumption we have made about f being unimodal may not hold. In this case, we are not guaranteed of finding at each iteration a new interval of values satisfying (2). Hence some modifications to the method outlined here may be necessary for a non-unimodal function.

Methods that don't require derivatives also exist for finding minimizers of multidimensional functions. Unfortunately such methods typically require a large number of function evaluations to achieve convergence. Additionally, in many cases it is not possible to provide a satisfactory theoretical analysis of the order of convergence.

3 Newton's Method

In this section, we assume that derivatives of f exist, and are relatively inexpensive to compute. This allows us to explore a wider range of methods for computing solutions to the problem of minimizing f. Probably the most well known algorithm is Newton's Method. For simplicity, we firstly consider the case when $n = 1$. Since f is smooth, we can express it as the following Taylor expansion about some point x_k:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2 + O\big(|x - x_k|^3\big). \tag{6}$$

Suppose that x and x_k are sufficiently close, so that the $O\big(|x - x_k|^3\big)$ term is relatively small. If this term is neglected, then the leading order terms in the right-hand side of (6) provide a quadratic approximation to f. By differentiating this quadratic approximation, we see that a stationary point occurs when

$$f'(x_k) + f''(x_k)(x - x_k) = 0. \tag{7}$$

Moreover this point will be a minimizer if and only if $f''(x_k) > 0$. Writing $g(x) \equiv f'(x)$, and rearranging (7), we have the following formula for x in terms of x_k:

$$x = x_k - \frac{g(x_k)}{g'(x_k)}. \tag{8}$$

This value of x is a minimizer of the quadratic approximation to f, and is hopefully close to ξ, an exact minimizer of f. This leads to an iterative method for approximating ξ: given an initial estimate x_0 of ξ, find the quadratic approximation of the Taylor series about x_0. Define the next iterate x_1 to be the minimizer of this quadratic approximation. We take this to be our updated estimate of ξ. Iterating this process, we obtain the following recursion (known as Newton's Method for a function of one variable):

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}\ , \qquad k = 0, 1, 2, \cdots . \tag{9}$$

There are some problems that can occur in the implementation of (9). For any k, if $g'(x_k)$ vanishes but $g(x_k)$ doesn't, then clearly Newton's Method will be undefined, and we will not be able to obtain the next iterate x_{k+1}. Ideally we would like the $\{x_k\}$ to converge to ξ. However even if the initial estimate x_0 is close to ξ, there is no guarantee that $x_1, x_2, \cdots$ will be close to ξ. Whether or not the iterates converge depends on the behavior of f. In addition, even if the iterates do converge to a limit point, this point may only correspond to a local rather than a global minimizer of f.

However we can show that Newton's Method does converge to a minimizer of f under certain conditions. Suppose that ξ is a local minimizer of f, so that $g(\xi) = 0$. Further, assume that ξ is a simple root of g, viz. $g'(\xi) \neq 0$. Subtracting ξ from both sides of (9), we have that

$$\begin{aligned} x_{k+1} - \xi &= x_k - \xi - \frac{g(x_k) - g(\xi)}{g'(x_k)} \\ &= \frac{(x_k - \xi)g'(x_k) - g(x_k) + g(\xi)}{g'(x_k)}. \end{aligned} \tag{10}$$

Now expand $g(\xi)$ as a Taylor expansion about x_k:

$$g(\xi) = g(x_k) + g'(x_k)(\xi - x_k) + \frac{1}{2}g''(\eta_k)(\xi - x_k)^2,$$

where η_k lies between x_k and ξ. Therefore from (10) we have

$$x_{k+1} - \xi = \frac{g''(\eta_k)}{2g'(x_k)}(x_k - \xi)^2.$$

Now if x_k is in a sufficiently small neighborhood of ξ, then $g'(\xi) \neq 0 \;\Rightarrow\; g'(x_k) \neq 0$, so we can bound $\frac{g''(\eta_k)}{2g'(x_k)}$ by a positive constant. Therefore we have local quadratic convergence of the $\{x_k\}$ to ξ.

3.1 Generalization of Newton's Method to n dimensions

Having derived the above formulas for the one-dimensional case, the extension to n dimensions is not difficult, though it requires some new definitions. Consider a scalar function $f(x)$ where $x \in \mathbf{R}^n$. As before, we wish to find a minimizer of f. Firstly we introduce n-dimensional generalizations of the first and second derivatives of f:

- $g(x) := \nabla f(x) \in \mathbf{R}^n$ is the *gradient* of $f(x)$.
- $H(x) := \nabla^2 f(x) \in \mathbf{R}^{n \times n}$ is the *Hessian* of $f(x)$.

The elements of g and H are the first and second order partial derivatives of f respectively, viz.,

$$g_i = \frac{\partial f}{\partial x_i} \quad \text{and} \quad H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j},$$

where x_i is the i^{th} element of x. In analogue to (6), we can express $f(x)$ as the following Taylor expansion about some point x_k:

$$f(x) = f(x_k) + g(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k) + O\big(|x - x_k|^3\big). \tag{11}$$

If we ignore the higher order terms, we see that the right-hand side of (11) is a quadratic function, and Newton's Method is obtained by minimizing this quadratic function over x.

Consider the following two classes of matrices:
A matrix $M \in \mathbf{R}^{n \times n}$ is *positive semi-definite* (PSD) if $y^T M y \geq 0$ for all $y \in \mathbf{R}^n$.
A matrix $M \in \mathbf{R}^{n \times n}$ is *positive definite* (PD) if $y^T M y > 0$ for all $0 \neq y \in \mathbf{R}^n$.

It can be shown that the quadratic approximation in (11) has a unique minimizer if and only if the Hessian $H(x_k)$ is PD. This minimizer is found by setting the gradient of the quadratic approximation equal to 0:

$$g(x_k) + H(x_k)(x - x_k) = 0. \tag{12}$$

If $H(x_k)$ is PD it is also invertible. Therefore we can solve (12) explicitly for x, which we will denote by x_{k+1}, since it represents an updated approximation to one of the minimizers of f:

$$x_{k+1} = x_k - H(x_k)^{-1} g(x_k). \tag{13}$$

If we think of (13) as being an iterative process for $k = 0, 1, \cdots$, it is just a generalization of (9) to n dimensions. As in the single variable case, Newton's Method in n dimensions can be shown to be locally quadratically convergent. In addition to the deficiencies of Newton's Method that were pointed out in the one-dimensional case, there are two further problems that are inherent in (13). Firstly, since the Hessian contains n^2 elements, we will generally have to compute approximately $n^2/2$ second order partial derivatives of f. (Since f is assumed to be sufficiently smooth, its Hessian H is symmetric, i.e. $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$, so once we've computed the upper triangular elements, we've also computed the lower triangular elements.) Even if n is moderately large (say 20-30), this calculation - which has to be done at every iteration - is quite expensive. Secondly, we are required to invert H at every iteration. Of course we don't compute H^{-1} explicitly in (13); instead, we solve the linear system

$$H(x_k) p_k = -g(x_k) \tag{14}$$

for $p_k \in \mathbf{R}^n$, and then update via $x_{k+1} = x_k + p_k$. Solving this linear system at each iteration is also expensive. If H is not PD, then (14) is not guaranteed to have a unique solution. Even if H is PD, it could be ill-conditioned or close to singular, in which case the solution p_k may not be accurate.

Consequently (14) cannot be used in practice on (moderately) large problems. This provides us with the necessary motivation to investigate other classes of methods for finding minimizers of functions of n variables. In particular, we seek methods which don't suffer from of the two aforementioned problems associated with Newton's Method in n dimensions.

4 The Secant Method

For a one-dimensional minimization problem, we have derived Newton's Method in (9). This requires the evaluation of g' at each iterate x_k. If g' is not defined everywhere or is too expensive to calculate, then in theory one can use the following approximation:

$$g'(x_k) \approx \frac{g(x_k) - g(x_{k-1})}{x_k - x_{k-1}}. \tag{15}$$

We then have what is known as the Secant Method:

$$x_{k+1} = x_k - \frac{g(x_k)}{g(x_k) - g(x_{k-1})}(x_k - x_{k-1}) \ , \qquad k = 1, 2, \cdots . \tag{16}$$

The method gets its name from the fact that the right-hand side of (15) is the slope of the secant connecting the points $(x_{k-1}, g(x_{k-1}))$ and $(x_k, g(x_k))$. Note that the Secant Method is a 2-point formula in that x_{k-1} and x_k are required to calculate x_{k+1}. However at each iteration, the Secant Method only requires one new evaluation - $g(x_k)$ - since $g(x_{k-1})$ was calculated in the previous iteration. On the other hand, Newton's Method, which is only a

1-point formula, requires the evaluation of $g(x_k)$ as well as $g'(x_k)$. Despite the advantage that the computation of $g'(x_k)$ does not have to be performed at each iteration, the Secant Method suffers from the problem that the approximation in (15) may not always be valid. For example, if x_k and x_{k-1} are both close to a minimizer of f, then $g(x_k) \approx g(x_{k-1})$ so that cancellation errors could occur in (16). It is easily seen that if at any stage $g(x_k) = g(x_{k-1})$, then x_{k+1} is undefined. However in practice this is a rare occurrence. The Secant Method also suffers from the problem that the convergence is only local, i.e. in general, only if the initial estimates of a minimizer are 'sufficiently close' to that minimizer will we then have convergence. In practice it is not possible to quantify precisely the set of starting points that are 'sufficiently close' to the minimizer.

We now compute the order of convergence of the Secant Method for smooth functions of one variable. As noted earlier, we do not require g to be smooth when implementing the Secant Method. However we will assume smoothness here to make the convergence proof easier. It is necessary to assume that the initial iterates are within a sufficiently small neighborhood of the desired minimizer ξ. Therefore we will only be making a statement about the local (as opposed to global) convergence of the Secant Method. Firstly, we need to introduce the concept of a *divided difference*. The divided difference $g[x_0, x_1, \cdots, x_n]$ of real-numbers $x_0, x_1, \cdots, x_n$, is found according to the following recursive relation:

$$g[x_0] = g(x_0),$$

$$\text{for } i = 1, 2, \cdots, n, \qquad g[x_0, x_1, \cdots, x_{i-1}, x_i] = \frac{g[x_1, \cdots, x_i] - g[x_0, \cdots, x_{i-1}]}{x_i - x_0}.$$

It can be shown that if g is smooth enough, divided differences of g can be written in terms of derivatives of g. For example, given any y_0, y_1, y_2, we have that

$$g[y_0, y_1] = g'(\xi_1) \quad \text{for some } \xi_1 \in [\min_{i=0,1} y_i \,, \max_{i=0,1} y_i], \tag{17}$$

$$g[y_0, y_1, y_2] = \frac{1}{2} g''(\xi_2) \quad \text{for some } \xi_2 \in [\min_{i=0,1,2} y_i \,, \max_{i=0,1,2} y_i]. \tag{18}$$

(In general the values of ξ_1 and ξ_2 are unknown.) In addition, divided differences are symmetric with respect to their arguments, i.e. we can permute $x_0, x_1, \cdots, x_n$ in any way, but the divided difference with respect to these numbers will remain unchanged. Subtracting the minimizer ξ from each side of (16), and noting that $g(\xi) = 0$, we have that

$$
\begin{aligned}
x_{k+1}-\xi &= x_k-\xi-\frac{g(x_k)-g(\xi)}{g(x_k)-g(x_{k-1})}(x_k-x_{k-1}) \\
&= x_k-\xi-\frac{g(x_k)-g(\xi)}{g[x_{k-1},x_k]} \\
&= (x_k-\xi)\left(1-\frac{g[x_k,\xi]}{g[x_{k-1},x_k]}\right) \\
&= (x_k-\xi)(x_{k-1}-\xi)\left(\frac{g[x_{k-1},x_k,\xi]}{g[x_{k-1},x_k]}\right) \\
\Longrightarrow |x_{k+1}-\xi| &\le |x_k-\xi|\,|x_{k-1}-\xi|\,\left|\frac{g''(\xi_2^{(k)})}{2g'(\xi_1^{(k)})}\right| \quad \text{in view of (17)-(18).} \qquad (19)
\end{aligned}
$$

The superscripts of ξ_1 and ξ_2 refer to the fact that these values are dependent on the iteration number. If we assume that ξ is a simple root of g, then in a sufficiently small neighborhood of ξ there exists a positive constant M such that for all k,

$$
\left|\frac{g''(\xi_2^{(k)})}{2g'(\xi_1^{(k)})}\right| \le M. \qquad (20)
$$

Now define $e_k := M|x_k-\xi|$. This is a measure of the error in the k^{th} iterate. Since x_0 and x_1 are assumed to be sufficiently close to ξ so that, for example, $e_0, e_1 < 1$, then (19)-(20) give us

$$
e_{k+1} \le e_k e_{k-1}, \qquad k = 1, 2, \cdots. \qquad (21)
$$

Using (21), it isn't difficult to show by induction that the $\{e_k\}$ satisfy

$$
e_k \le \alpha^{r^k}, \quad \text{where} \quad 0<\alpha<1, \quad r=\frac{1+\sqrt{5}}{2}.
$$

(r is the positive root of $r^2 = r+1$, and α depends on e_0, e_1 and r.)

Roughly speaking, $e_k \le \alpha^{r^k}$ means that the order of convergence of the Secant Method is $r \approx 1.618$. Although this order of convergence is not as good as that of Newton's Method, it is still impressive. Later we will generalize the Secant Method to minimize functions of n variables.

5 Search Directions and Line Searches

In the remaining sections of this chapter, we will focus mainly on methods that avoid the pitfalls of Newton's Method. As usual we are attempting to find a sequence of iterates $x_0, x_1, \cdots \in \mathbf{R}^n$ that converge to a minimizer ξ of a function f, using some iterative procedure. At the start of iteration $k+1$, we know x_k and are attempting to find a better approximation x_{k+1} to ξ. Essentially what we are doing is moving from x_k in some *search direction* $p_k \in \mathbf{R}^n$. Once we have decided the direction p_k in which to step, we need to determine the distance $\alpha_k \in \mathbf{R}$ that we move along p_k. We can then write the next iterate as $x_{k+1} = x_k + \alpha_k p_k$. For example, in Newton's Method (13), we have $\alpha_k = 1$ and $p_k = -H(x_k)^{-1}g(x_k)$, where g and H are the gradient and Hessian respectively of f at x_k. Although we have a large amount of flexibility in choosing α_k and p_k, there are some commonsense criteria that need to be met. We want to choose p_k so that when we search in the direction of p_k, it is possible to find some values of f that are superior to that at the current point x_k. If f is smooth enough we have the following Taylor expansion about x_k:

$$f(x_k + \alpha p_k) \approx f(x_k) + \alpha g(x_k)^T p_k \ , \tag{22}$$

provided αp_k is relatively small in length. If $\alpha > 0$ and p_k is chosen so that $g(x_k)^T p_k < 0$, then we have $f(x_k + \alpha p_k) < f(x_k)$, which means that superior (i.e. lower) function values are found by stepping a small distance in the direction of p_k. In this case, p_k is called a *descent direction.* Once we have fixed p_k, we wish to find an $\alpha > 0$ that produces the most descent in f. This process is known as a *line search.* In fact the most descent is obtained by choosing the α that minimizes $f(x_k + \alpha p_k)$. Since x_k and p_k are known at this stage, this is just a one-dimensional minimization problem, and can be handled using the methods discussed in previous sections. However it should be remembered that this problem of finding the best α occurs at *every iteration*, since x_k and p_k vary with k. Consequently, it would take a large amount of time to implement this process even if the number of iterations was only moderately large. In any case, (22) is only valid in a small neighborhood of x_k. So the search from x_k along the direction p_k may not even occur close to a minimizer of f, especially in the initial iterations. Therefore it doesn't make sense to spend time finding the best α too accurately. In fact it is usually not possible to perform an exact line search - and therefore find the best α - unless f is a particularly simple function. So generally we will need to perform *inexact* line searches, where the search will be terminated after 'reasonable progress' has been made in reducing the value of f.

We now discuss how p_k might be chosen. From (22) we see that if we choose p_k so that $g(x_k)^T p_k < 0$ is as small as possible, then locally we can achieve the largest reduction in f. Now the angle θ between the vectors $g(x_k)$ and p_k is given by

$$cos(\theta) = \frac{g(x_k)^T p_k}{\|g(x_k)\|_2 \ \|p_k\|_2}. \tag{23}$$

(Recall that for a vector $y \in \mathbf{R}^n$, $\|y\|_2 = \sqrt{y^T y}$.) For a given gradient $g(x_k)$, if p_k is of fixed length (i.e. norm), then (23) implies that $g(x_k)^T p_k$ is smallest when $\theta = 180°$. That is, $g(x_k)$ and p_k are in opposite directions, or equivalently, p_k is in the direction of $-g(x_k)$. This direction is known as the *steepest descent* direction. The general Steepest Descent algorithm - which searches in the direction of steepest descent - can be expressed as follows:

1. Initialize with x_0 (an initial estimate of a minimizer of f).

2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:

   - $p_k = -g(x_k)$
   - Perform a line search to compute the α_k that minimizes $f(x_k + \alpha p_k)$ over $\alpha > 0$
   - $x_{k+1} = x_k + \alpha_k p_k$

One commonly used stopping criterion is $\|g(x_k)\|/\|g(x_0)\| < \epsilon$ for some known tolerance ϵ. As an example, now consider the case where f is a quadratic function of n variables, i.e. f can be written in the following form:

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c \ , \tag{24}$$

where $A \in \mathbf{R}^{n\times n}$, and $x, b, c \in \mathbf{R}^n$. Suppose that A is symmetric and positive definite (SPD). Then there exists a point where the gradient of f vanishes, and this point is guaranteed to be a unique global minimizer of f. The gradient of f is $g(x) = Ax - b$, so finding the global minimizer of f is equivalent to solving the linear system $Ax = b$. If the search direction p is chosen to be the steepest descent direction, then for a quadratic f, it is possible to perform an exact line search. That is, for fixed x and p, we can compute the exact value of α that minimizes $f(x+\alpha p)$. This is done by solving $\frac{df}{d\alpha} = 0$ for α. Now define the *residual* $r := -g(x) = b - Ax$. We then have that

$$\frac{d}{d\alpha}f(x+\alpha p) = g(x+\alpha p)^T p = \left(A(x+\alpha p) - b\right)^T p = \left(-r + \alpha Ap\right)^T p = -r^T p + \alpha p^T Ap \ ,$$

so the line search parameter is given by

$$\alpha = \frac{r^T p}{p^T Ap}. \tag{25}$$

Care should be taken to distinguish between the formulas for the residual and the search direction. The residual is always defined to be the negative gradient vector, and in the Steepest Descent Method, we *choose* $p = r$ at every iteration. So we have the following algorithm for quadratic functions:

1. Initialize with x_0 (an initial estimate of a minimizer of f).
2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:
   - $p_k = b - Ax_k$
   - $\alpha_k = \frac{p_k^T p_k}{p_k^T Ap_k}$
   - $x_{k+1} = x_k + \alpha_k p_k$

Although the above algorithm is designed with quadratic functions in mind, it can easily be applied to non-quadratic functions as well. For a non-quadratic f, we can just think of the right-hand side of (24) as being a quadratic approximation to f. However A, b, and c will generally not be constant at each iteration - they will depend on x_k.

When applied to quadratic functions with a SPD A, the Steepest Descent Method has the attractive property that it always searches in the best (steepest descent) direction at every iteration, and therefore always gives an improvement in the value of f. Unfortunately the method is also myopic, because this direction may be bad in a global sense. In other words, the steepest descent direction, which is the best direction in a local sense, may not point in the direction of the minimizer to which we are eventually attempting to converge. Therefore convergence to the minimizer can be extremely slow. In fact it can be shown that the convergence is linear with rate $(\kappa - 1)/(\kappa + 1)$, where κ is the condition number of A in the 2-norm. (In terms of eigenvalues of A, κ is the ratio of the maximum eigenvalue to the minimum eigenvalue.)

So if A is not well conditioned, then $C \approx 1$ and the convergence will be very slow. Additionally, for a non-quadratic function, the matrix A of second order partial derivatives has to be computed at every iteration, since its elements depend on x_k. Therefore the Steepest Descent algorithm is seldom used in practice. However it does provide some insight into how search directions and line searches are incorporated into an algorithm. It also gives us some motivation to develop methods for computing minimizers that converge faster, and require less computation at each iteration.

6 The Conjugate-Gradient Method

As discussed in the previous section, one of the principle problems with the Steepest Descent Method is that the search direction $p_k = -g(x_k)$ depends only on the current point we are at (i.e. x_k). In fact consecutive search directions are orthogonal if f is a quadratic function:

$$p_k^T p_{k+1} \;=\; p_k^T(b - Ax_{k+1}) \;=\; p_k^T\left(b - A(x_k + \alpha_k p_k)\right) \;=\; p_k^T\left(p_k - \alpha_k A p_k\right) \;=\; 0$$

in view of (25) and the fact that $p_k = r_k$ for all k. As a result, the Steepest Descent Method often searches in similar directions to those searched in previous iterations. The Conjugate-Gradient Method (CGM) however uses search directions that are *conjugate with respect to* A. Two directions p_i and p_j are said to be conjugate with respect to A if $p_i^T A p_j = 0$. In the CGM, a set of mutually conjugate directions $p_0, p_1, \cdots$ with $p_i^T A p_j = 0 \;\forall\; i \neq j$ is obtained by setting the initial direction p_0 to be the steepest descent direction $-g(x_0)$, and then taking p_k to be a specific linear combination of the current steepest descent direction $-g(x_k)$ and all previous search directions $p_0, p_1, \cdots, p_{k-1}$. The CGM also differs from the Steepest Descent Method in that way the line searches are carries out. In the latter method, the line search at iteration $k+1$ finds an $\alpha > 0$ that minimizes $f(x_k + \alpha p_k)$. In contrast to this one-dimensional minimization, the CGM at iteration $k+1$ requires an $\alpha \in \mathbf{R}^{k+1}$ that minimizes $f(x_k + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_k p_k)$. If f is quadratic with SPD A, the unique minimizer of this function will occur at

$$\frac{\partial f}{\partial \alpha_i} = 0 \;, \quad i = 0, 1, \cdots, k. \tag{26}$$

But

$$\begin{aligned}
\frac{\partial f}{\partial \alpha_i} &= g(x_k + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_k p_k)^T p_i \qquad (27)\\
&= \left(A(x_k + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_k p_k) - b\right)^T p_i \\
&= \left(-r_k + \alpha_0 A p_0 + \alpha_1 A p_1 + \cdots + \alpha_k A p_k\right)^T p_i \\
&= -r_k^T p_i + \alpha_0 p_0^T A p_i + \alpha_1 p_1^T A p_i + \cdots + \alpha_k p_k^T A p_i \\
&= -r_k^T p_i + \alpha_i p_i^T A p_i \;,
\end{aligned}$$

since $p_0, p_1, \cdots, p_k$ are mutually conjugate. So the line search parameters are given by

$$\alpha_i = \frac{r_k^T p_i}{p_i^T A p_i}\ , \quad i = 0, 1, \cdots, k. \tag{28}$$

Now define x_{k+1} to be the point in x-space at which $f(x)$ reaches its minimum in the line search at iteration $k+1$, i.e. $x_{k+1} = x_k + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_k p_k$, with the $\{\alpha_i\}$ from (28). Then from (26)-(27), we have that $g(x_{k+1})^T p_i = 0$ for $i \leq k$. Hence $r_{k+1}^T p_i = 0$ for $i \leq k$. Equivalently, $r_k^T p_i = 0$ for $i \leq k-1$. It then follows from (28) that $\alpha_i = 0$ for $i \leq k-1$. So we have $x_{k+1} = x_k + \alpha_k p_k$, where $\alpha_k = \frac{r_k^T p_k}{p_k^T A p_k}$.

The CGM for quadratic functions of the form (24) is therefore:

1. Initialize with x_0 (an initial estimate of a minimizer of f), $r_0 = b - Ax_0,\ p_0 = r_0$.

2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:

   - $\alpha_k = \frac{r_k^T p_k}{p_k^T A p_k}$
   - $x_{k+1} = x_k + \alpha_k p_k$
   - $r_{k+1} = r_k - \alpha_k A p_k$
   - $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
   - $p_{k+1} = r_{k+1} + \beta_k p_k$

Some aspects of this algorithm are worth mentioning:
(1) The formula for β_k ensures that the search directions are mutually conjugate.
(2) The formula for r_{k+1} is equivalent to $r_{k+1} = b - Ax_{k+1}$ (which is the definition of the residual), since $x_{k+1} = x_k + \alpha_k p_k$. However the form used in the algorithm is more efficient since the matrix-vector product Ap_k was already computed when we found α_k.
(3) The matrix A need not be stored explicitly; it is sufficient to store the product Ap_k instead. In addition, we need only store three other vectors at each iteration: x_{k+1}, r_{k+1} and p_{k+1}. (We also need to store the inner product $r_k^T r_k$ from the previous iteration.)
(4) If A is SPD, the algorithm will terminate in at most n iterations. That is, within n iterations, we will be guaranteed of obtaining the optimal solution (for a quadratic function). This is an extremely attractive property, which results from the fact that the residuals are mutually orthogonal ($r_i^T r_j = 0$ for $0 \leq i \neq j < n$), and therefore linearly independent, provided they are nonzero. (If $r_i = 0$ for some $i < n$, we are already at the minimizer.) Since no more than n vectors can be linearly independent in $\mathbf{R}^n$, it follows that after n iterations, the method must terminate at the minimizer of f (and also $r_n = 0$.)
(5) It is possible to derive an error estimate for the CGM when applied to a quadratic function with SPD A. It is convenient to measure the error $e_i := x_i - \xi$ (where ξ is the global minimizer of f) using the *A-norm*: $\|y\|_A = \sqrt{y^T A y}$. It can then be shown (see e.g., Stoer and Bulirsch, *Introduction to Numerical Analysis*, 2nd ed., for a full proof), that the error at iteration i can be expressed in terms of the initial error as

$$\|e_i\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|e_0\|_A\ , \tag{29}$$

where κ is as defined in the previous section.

From (29) we see that the convergence is linear with rate $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$. Although this rate is faster than that of the Steepest Descent Method, the convergence is still too slow if κ is large.

(6) As mentioned in the previous section, the minimizer of (24) is the solution of the linear system $Ax = b$ if the Hessian is PD. In fact the CGM was originally designed for solving linear systems.

6.1 The Preconditioned Conjugate-Gradient Method

In the case that A has a large condition number, it becomes necessary to use a *preconditioner* to speed up the otherwise slow convergence of the standard CGM.

As stated above, the minimizer in the CGM also solves the linear system $Ax = b$. Now let M be a preconditioner. This is a matrix that is SPD and easily invertible. Moreover we require M to be in some sense a reasonable approximation to A, so that $M^{-1}A$ is 'close' to the identity matrix. Therefore the transformed linear system $M^{-1}Ax = M^{-1}b$ will have a coefficient matrix $M^{-1}A$ with condition number close to 1.

The Preconditioned Conjugate-Gradient Method (PCGM) is as follows:

1. Initialize with x_0 (an initial estimate of a minimizer of f), $r_0 = b - Ax_0$, $q_0 = M^{-1}r_0$, $p_0 = q_0$.

2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:

   - $\alpha_k = \frac{r_k^T q_k}{p_k^T A p_k}$
   - $x_{k+1} = x_k + \alpha_k p_k$
   - $r_{k+1} = r_k - \alpha_k A p_k$
   - $q_{k+1} = M^{-1} r_{k+1}$
   - $\beta_k = \frac{r_{k+1}^T q_{k+1}}{r_k^T q_k}$
   - $p_{k+1} = q_{k+1} + \beta_k p_k$

The main difference between the PCGM and standard CGM is that we have to determine q_{k+1} at each iteration by solving the linear system $Mq_{k+1} = r_{k+1}$. The PCGM has been shown to perform well on a wide range of problems for which the standard CGM hasn't.

Two possibilities for the preconditioner are:

(1) M is a diagonal matrix whose elements are equal to those on the main diagonal of A. Clearly M is extremely easy to invert. However this may not always be a good preconditioner.

(2) If A is sparse (i.e. contains many zero elements), we can compute M using an *incomplete* Cholesky Factorization: we attempt to find the standard Cholesky Factorization $A = LL^T$, where L is a lower triangular matrix. However when computing L, we insist that at any stage, if $A_{ij} = 0$ then $L_{ij} = 0$ also. This is done to preserve the sparsity structure of A.

The convergence result for the PCGM is similar to that in 29 for the standard CGM. However κ is now the condition number of $M^{-1}A$. So if M approximates A well, then κ will be small, and the convergence will be relatively fast.

6.2 The Nonlinear Conjugate-Gradient Method

The CGMs described above are more accurately referred to as Linear CGMs, since they can also be used to solve linear systems of equations. However it is possible to generalize these methods to a method that can minimize non-quadratic functions. This method is known as the Nonlinear CGM. It requires us to compute gradients - though not necessarily Hessians - of f at each iteration. As in the Linear CGM, we need to perform a line search at each iteration. However it is not obvious what the best method is for minimizing $f(x_k + \alpha p_k)$ if f is not quadratic. Perhaps we could use the one-dimensional Newton or Secant Methods. Whether or not we require the Hessian of f to be computed at each iteration will depend on the method we use. In general though, it will not be possible to perform an exact line search, so we will not be able to maintain mutually conjugate directions. In any case, the concept of conjugate directions takes on a different meaning in the Nonlinear CGM, since in general, the Hessian is no longer constant. In other words, in the Linear CGM, p_i and p_j are conjugate with respect to A if $p_i^T A p_j = 0$. However if f is not quadratic, then at each iteration the directions are conjugate with respect to a different A. Another important difference between the Linear and Nonlinear CGMs is that in the latter, there is no obvious choice for β_k. Several formulas have been suggested by researchers, and we give two of these in the following algorithm:

1. Initialize with x_0 (an initial estimate of a minimizer of f), $r_0 = -g(x_0), \ p_0 = r_0$.
2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:
   - Perform a line search to compute the α_k that minimizes $f(x_k + \alpha p_k)$ over $\alpha > 0$
   - $x_{k+1} = x_k + \alpha_k p_k$
   - $r_{k+1} = -g(x_{k+1})$
   - $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ (Fletcher-Reeves) or $\frac{r_{k+1}^T (r_{k+1} - r_k)}{r_k^T r_k}$ (Polak-Ribiére)
   - $p_{k+1} = r_{k+1} + \beta_k p_k$

It should not come as a surprise that this algorithm does not possess the finite termination property of the Linear CGM, and much less can be said about the conditions under which it converges to a minimizer of f. In fact, after every n iterations, it is common to *restart* the algorithm by setting $\beta = 0$, so that the search direction will return to the steepest descent direction ($p = r = -g(x)$). It may take far more than n iterations to achieve convergence. It is possible to use preconditioners in the Nonlinear CGM as well. Some details are given in [1].

7 Quasi-Newton Methods

Although the Linear Conjugate-Gradient Method does alleviate some of the problems associated with Newton's Method, it still requires the computation of second order derivatives at each iteration. To avoid this computationally expensive process, we can choose to approximate the second order derivatives in the Hessian by first order derivatives of the function f

that we are attempting to minimize. In fact (15) is an example of such an approximation, where f is a function of one variable, and $g(x) \equiv f'(x)$. In the case that f is a function of n variables, (15) can no longer be used since it is not possible to divide by the vector $x_k - x_{k-1} \in \mathbf{R}^n$. However we can rearrange (15), and replace k by $k+1$ for convenience to obtain:

$$H(x_{k+1})(x_{k+1} - x_k) \approx g(x_{k+1}) - g(x_k), \tag{30}$$

where H is the Hessian of f. Quasi-Newton Methods approximate the Hessian at x_{k+1} by a matrix B_{k+1}, which is updated at each iteration to reflect the fact that the Hessian will usually be changing. To force B_{k+1} to be 'close' to $H(x_{k+1})$ in some sense, we impose the *Secant condition* (also called the *Quasi-Newton relation*), that basically says that B_{k+1} acts in a similar way towards multiples of $x_{k+1} - x_k$ that $H(x_{k+1})$ does:

$$B_{k+1}(x_{k+1} - x_k) = g(x_{k+1}) - g(x_k). \tag{31}$$

If we define $s_k := x_{k+1} - x_k$ and $y_k := g(x_{k+1}) - g(x_k)$, we have the more compact form of the Secant condition:

$$B_{k+1}s_k = y_k. \tag{32}$$

The Secant condition is just a system of n equations, and therefore is not sufficient to uniquely determine the n^2 unknown entries of B_{k+1}. It is necessary to impose further conditions on B_{k+1}.

7.1 Rank-one Updates

For a Quasi-Newton Method to be practical, we would like each B_k to be evaluated using significantly fewer function evaluations than that required for $H(x_k)$. One way of achieving this is to use an *update formula* for B_k. In other words we would like to obtain $B_{k+1} \approx H(x_{k+1})$ by using B_k from the previous iteration. This can be done easily by means of a *rank-one update*: $B_{k+1} = B_k + C_k$, where C_k is a rank-one matrix. A rank-one matrix can be written as an outer product $v_k w_k^T$ of two vectors $v_k, w_k \in \mathbf{R}^n$. Substituting our proposed rank-one update into the Secant condition, we have that

$$y_k = B_{k+1}s_k = (B_k + v_k w_k^T)s_k \implies v_k(w_k^T s_k) = y_k - B_k s_k.$$

Provided w_k is chosen so that $w_k^T s_k \neq 0$, we then have that

$$v_k = \frac{y_k - B_k s_k}{w_k^T s_k}. \tag{33}$$

An example of a w_k that satisfies $w_k^T s_k \neq 0$ is $w_k = s_k$. (Note that $s_k \neq 0$, otherwise $x_{k+1} = x_k$ and we have converged.) So one possible rank-one update is *Broyden's Method*:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^T}{s_k^T s_k}. \tag{34}$$

This formula can be used to solve systems of nonlinear equations of the form (1), where the system is written as $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, and $\mathbf{g} : \mathbf{R}^n \to \mathbf{R}^n$ is the same function used in the definition of y_k. It is more suitable for solving systems of nonlinear equations then the other methods given in this section since the B_k do *not* possess the *hereditary symmetry* property. In other words, even if B_k is symmetric, B_{k+1} need not be. In general, we cannot assume any special structure on the system of equations $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, so to use a Quasi-Newton Method with a hereditary symmetric B_k would impose special structure that isn't necessarily inherent in the original problem.

However if we wish to solve a minimization problem instead, then because the Hessian is symmetric (if f is smooth enough), it seems reasonable to ask that each B_k also be symmetric. In other words, if B_k is symmetric, we would also like $B_{k+1} = B_k + v_k w_k^T$ to be symmetric. This can only occur if $v_k w_k^T$ is symmetric, which is equivalent to saying that w_k is a multiple of v_k: $w_k = \gamma_k v_k$ for some scalar γ_k. From (33), we have that v_k - hence w_k - is in the direction of $y_k - B_k s_k$. We can then evaluate γ_k, and obtain the unique *symmetric rank-one* update:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \tag{35}$$

(As an exercise, verify that (34) and (35) satisfy the Secant Condition.)

The algorithm which implements the above rank-one updates is as follows:

1. Initialize with x_0 (an initial estimate of a minimizer of f), $g_0 = g(x_0)$, and B_0 (an estimate of the initial Hessian $H(x_0)$).

2. For $k = 0, 1, \cdots$ until some stopping criterion is satisfied:

   - Solve the linear system $B_k p_k = -g_k$ for p_k
   - If we are solving a minimization problem, perform a line search to compute the α_k that minimizes $f(x_k + \alpha p_k)$ over $\alpha > 0$. If we are solving a nonlinear system of equations, take $\alpha_k = 1$.
   - $s_k = \alpha_k p_k$
   - $x_{k+1} = x_k + s_k$
   - $g_{k+1} = g(x_{k+1})$
   - $y_{k+1} = g_{k+1} - g_k$
   - Compute B_{k+1} via (34) or (35)

With regards to this algorithm, the following points should be noted:
(1) In the absence of any other information, we can take B_0 to be the identity matrix.
(2) The linear system $B_k p_k = -g_k$ is analogous to that which is solved in Newton's Method - c.f. (14). Since B_{k+1} and B_k only differ by a rank-one matrix, we can use a special updating formula for the *inverse* of a rank-one matrix to compute B_{k+1}^{-1} in terms of B_k^{-1} at a relatively small cost. Then we can compute p_k directly via $p_k = -B_k^{-1} g_k$. This is more efficient than solving a new system $B_k p_k = -g_k$ at each iteration.
(3) The number of iterations required for convergence depends considerably on the complexity of the function to be minimized or the system of equations to be solved. As in the Nonlinear CGM, it is common to *restart* the algorithm after say every n or $2n$ iterations.

7.2 Rank-two Updates

Although rank-one updates are relatively easy to compute, the B_k in (34) and (35) lack some important properties that force the corresponding Quasi-Newton Methods to converge to a minimizer. In particular, it would be desirable if B_k was PD. If so, then the direction p_k obtained from solving $B_k p_k = -g_k$ in the above algorithm would be guaranteed to be a descent direction: $p_k^T g_k = -p_k^T B_k p_k < 0$ from the definition of positive definiteness. Additionally, at a minimizer, it is necessary that the Hessian be at least PSD - in fact it is usually PD - so it would make sense for each B_k to have the same property. Hence we will insist that our update formulas exhibit *hereditary positive definiteness*. This means that if B_k is PD, then B_{k+1} will also be PD for all k. So provided we initialize with a PD B_0, all of the B_k will be PD. Since rank-one matrices can be written in the outer-product form described earlier, the updates in the previous section were particularly easy to implement. However in order to accommodate the demand for hereditary positive definiteness, we are forced to look beyond the class of rank-one matrices, and into rank-two matrices. Fortunately though, a rank-two matrix can be written as the sum of two rank-one matrices, so we can still work with outer products of vectors as before. Therefore the amount of computation involved in rank-two updates is kept down to a reasonable level. The following family of rank-two matrices with parameter $\phi \in \mathbf{R}$ all exhibit hereditary positive definiteness, hereditary symmetry, and satisfy the Secant condition (see [2]):

$$\begin{gathered} B_{k+1} = B_k - \frac{(B_k s_k)(B_k s_k)^T}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} + \phi(s_k^T B_k s_k) v_k v_k^T \\ \text{where } v_k = \frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}. \end{gathered} \tag{36}$$

The update has been observed to perform relatively well when $\phi = 0$ and $\phi = 1$. When $\phi = 0$ we have the *BFGS formula*, due to Broyden, Fletcher, Goldfarb, and Shanno. When $\phi = 1$ we have the *DFP formula*, due to Davidon, Fletcher, and Powell. If $\phi \neq 0$, the right-hand side of the update formula in (36) involves the sum of three rank-one matrices. Therefore one might be led to believe that we have a rank-three update. However it can be shown that these three rank-one matrices can be combined into two rank-one matrices.

Due to the special structure of the above formula, care should be taken so that the implementation is done in an efficient manner. For example, in (36) it should be noted if $\phi = 0$, then $B_k s_k$ appears three times, and if $\phi \neq 0$, then $y_k^T s_k$ and $s_k^T B_k s_k$ also appear multiple times. Obviously the implementation should be done in such a way that the same quantity is not computed more than once.

Since Quasi-Newton Methods are only approximations to the classical Newton Method, we cannot hope to attain quadratic convergence. However if we place suitable restrictions on f, ϕ, and the line search procedure, then superlinear convergence (of order $r \in (1,2)$) can be obtained. We have only presented a small sample of Quasi-Newton Methods here. Many variations have been proposed in the literature, most of which attempt to reduce the storage requirements that are so crucial when solving problems with large numbers of variables. Some of these are given in [1],[2], and [3].

References

[1] P. E. Gill , W. Murray, and M. H. Wright, *Practical Optimization*, 1981.

[2] S. G. Nash and A. Sofer, *Linear and Nonlinear Programming*, 1996.

[3] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, 1996.

[4] B. D. Bunday and G. R. Garside, *Optimisation Methods in Pascal*, 1987.