Chapter 6

Nonlinear Equations and Optimization

6.1 Introduction

This chapter discusses two related computational problems. One is *root finding*, or solving systems of nonlinear equations. This means that we seek values of n variables, $(x_1, \ldots, x_n) = x \in R^n$, to satisfy n nonlinear equations $f(x) = (f_1(x), \ldots, f_n(x)) = 0$. We assume that $f(x)$ is a smooth function of x. The other problem is *smooth optimization*, or finding the minimum (or maximum[1]) value of a smooth *objective function*, $V(x)$. These problems are closely related. Optimization algorithms use the gradient of the objective function, solving the system of equations $g(x) = \nabla V(x) = 0$. However, the optimization problem has special structure that makes it easier than the general root finding problem.

The theory here is for *black box* ("closed box" would be more accurate) methods. This means algorithms that do not depend on details of the definitions of the functions $f(x)$ or $V(x)$. Instead, they use procedures that evaluate $f(x)$ or $V(x)$ for a given x value, such as `int fEval( double* f, double* x)` (f and x probably vectors). The code doing the root finding will learn about f only by "user-supplied" procedures, such as `fEval`, that supply values of f or V and their derivatives. The person writing the root finding or optimization code need not "open the box" to see how `fEval` works. This makes it possible for specialists to create general purpose optimization and root finding software that is efficient and robust, without knowing all the problems it may be applied to. It is unlikely that someone could create a general purpose root finder that is as good as the best available on the web.

There is a strong incentive to use derivative information as well as function values. For root finding, we use the $n \times n$ Jacobian matrix, $f'(x)$, with entries $f'(x)_{jk} = \partial_{x_k} f_j(x)$. For optimization, we use the gradient and the $n \times n$ *Hessian* matrix of second partials $H(x)_{jk} = \partial_{x_j} \partial_{x_k} V(x)$. It may seem like too much extra work to go from the n components of f to the n^2 entries of f', but algorithms that use f' often are much faster and more reliable than those that do not.

There are drawbacks to using general-purpose software that treats each specific problem as a black box. Large-scale computing problems usually have specific features that have a big impact on how they should be solved. Reformulating a problem to fit into a generic $f(x) = 0$ or $\min_x V(x)$ form may increase the condition number. Problem-specific solution strategies may be more effective than the generic Newton's method. In particular, the Jacobian or the Hessian may be sparse in a way that general purpose software cannot take advantage of. Some more specialized algorithms are in Exercise 6c (Marquart-Levenberg for nonlinear least squares), and Section **??** (Gauss-Seidel iteration for large systems). iteration

The algorithms discussed here are *iterative* (see Section 2.4). They produce a sequence of approximations, or *iterates*, that should converge to the desired solution, x_*. In the simplest case, each *iteration* starts with a *current iterate*, $\overline{x}$, and produces a *successor iterate*, $x' = \Phi(\overline{x})$. The algorithm starts from an

[1] Optimization refers either to minimization or maximization. But finding the maximum of $V(x)$ is the same as finding the minimum of $-V(x)$.

initial guess[2], x_0, then produces a sequence of iterates $x_{k+1} = \Phi(x_k)$. The algorithm succeeds if the iterates converge to the solution: $x_k \to x_*$ as $k \to \infty$. An iterative method fails if the iterates fail to converge or converge to the wrong answer. For an algorithm that succeeds, the *convergence rate* is the rate at which $\|x_k - x_*\| \to 0$ as $k \to \infty$.

An iterative method is *locally convergent* if it succeeds whenever the initial guess is close enough to the solution. That is, if there is an $R > 0$ so that if $\|x_0 - x_*\| \leq R$ then $x_k \to x_*$ as $k \to \infty$. The algorithms described here, mostly variants of *Newton's method*, all are locally convergent if the problem is non-degenerate (terminology below). An iterative method is *globally convergent* if it finds the answer from any initial guess. Between these extremes are algorithms that are more or less *robust*. The algorithms described here consist of a relatively simple locally convergent method, usually Newton's method, enhanced with *safeguards* that guarantee that some progress is made toward the solution from any $\overline{x}$. We will see that safeguards based on mathematical analysis and reasoning are more effective than heuristics.

All iterative methods need some kind of *convergence criterion* (more properly, *halting criterion*). One natural possibility is to stop when the relative change in x is small enough: $\|x_{k+1} - x_k\| / \|x_k\| \leq \epsilon$. It also makes sense to check that the *residuals*, the components of $f(x)$ or $\nabla V(x)$, are small. Even without roundoff error, an iterative method would be very unlikely to get the exact answer. However, as we saw in Section 2.4, good algorithms and well conditioned problems still allow essentially optimal accuracy: $\|x_k - x_*\| / \|x_*\| \sim \epsilon_{mach}$.

The final section of this chapter is on *iterative methods* that do not use higher derivatives. The discussion applies to linear or nonlinear problems. For optimization, it turns out that the rate of convergence of simple iterative methods is determined by the condition number of H for solving linear systems involving H, see Section 4.3.1 and the condition number formula (4.36). More precisely, the number of iterations needed to reduce the error by a factor of 2 is proportional to $\kappa(H) = \lambda_{\max}(H)/\lambda_{\min}(H)$. This, more than linear algebra roundoff, explains our fixation on condition number. The condition number $\kappa(H) = 10^4$ could arise in a routine partial differential equation problem. This bothers us not so much because it makes us lose 4 out of 16 double precision digits of accuracy, but because it takes tens of thousands of iterations to solve the damn problem.

6.2 Solving a single nonlinear equation

The simplest problem is that of solving a single equation in a single variable: $f(x) = 0$. Single variable problems are easier than multi-variable problems. There are simple criteria that guarantee a solution exists. Some algorithms for one dimensional problems, Newton's method in particular, have analogues for

[2]Here, the subscript denotes the iteration number, not the component. In n dimensions, iterate x_k has components $x_k = (x_{k1}, \ldots, x_{kn})$.

higher dimensional problems. Others, such as bisection, are strictly one dimensional. Algorithms for one dimensional problems are components for algorithms for higher dimensional problems.

6.2.1 Bisection

The bisection algorithm, or *bisection search*, is the simplest and most robust way to find a zero of a function on one variable. It does not require $f(x)$ to be differentiable, but merely continuous. It is based on a simple topological fact called the *intermediate value theorem*: if $f(x)$ is a continuous real-valued function of x on the interval $a \leq x \leq b$ and $f(a) < 0 < f(b)$, then there is at least one $x_* \in (a, b)$ with $f(x_*) = 0$. A similar theorem applies in the case $b < a$ or $f(a) > 0 > f(b)$.

The bisection search algorithm consists of repeatedly bisecting an interval in which a root is known to lie. Suppose we have an interval[3] $[\overline{a}, \overline{b}]$ with $f(\overline{a}) < 0$ and $f(\overline{b}) > 0$. The intermediate value theorem tells us that there is a root of f in $[\overline{a}, \overline{b}]$. The uncertainty is the location of this root is the length of the interval $\left|\overline{b} - \overline{a}\right|$. To cut that uncertainty in half, we bisect the interval. The midpoint is $\overline{c} = \left(\overline{a} + \overline{b}\right)/2$. We determine the sign of $f(\overline{c})$, probably by evaluating it. If $f(\overline{c}) > 0$ then we know there is a root of f in the sub interval $[\overline{a}, \overline{c}]$. In this case, we take the new interval to be $[a', b']$, with $a' = \overline{a}$ and $b' = \overline{c}$. In the other case, $f(\overline{c}) < 0$, we take $a' = \overline{c}$ and $b' = \overline{b}$. In either case, f changes sign over the half size interval $[a', b']$.

To start the bisection algorithm, we need an initial interval $[a_0, b_0]$ over which f changes sign. Running the bisection procedure then produces intervals $[a_k, b_k]$ whose size decreases at an exponential rate:

$$|b_k - a_k| = 2^{-k} |b_0 - a_0| \ .$$

To get a feeling for the convergence rate, use the approximate formula $2^{10} = 10^3$. This tells us that we get three decimal digits of accuracy for each ten iterations. This may seem good, but Newton's method is much faster, when it works. Moreover, Newton's method generalizes to more than one dimension while there is no useful multidimensional analogue of bisection search. Exponential convergence often is called *linear convergence* because of the linear relationship $|b_{k+1} - a_{k+1}| = \frac{1}{2} |b_k - a_k|$. Newton's method is faster than this.

Although the bisection algorithm is robust, it can fail if the computed approximation to $f(x)$ has the wrong sign. If f is not evaluated exactly, the computed approximation may not be continuous on a fine scale. A bisection code should take this possibility into account, either by refusing to bisect beyond a certain point, or by checking for consistency among the reported signs of f, or by making explicit use of an error estimate for computed f values.

[3]The interval notation $[a, b]$ used here is not intended to imply that $a < b$. For example, the interval $[5, 2]$ consists of all numbers between 5 and 2, endpoints included.

6.2.2 Newton's method for a nonlinear equation

As in the previous section, we want to find a value, x_*, that solves a single nonlinear equation $f(x_*) = 0$. We have procedures that return $f(x)$ and $f'(x)$ for any given x. At each iteration, we have a current iterate, $\overline{x}$ and we want to find an x' that is closer to x_*. Suppose that $\overline{x}$ is close to x_*. The values $f(\overline{x})$ and $f'(\overline{x})$ determine the tangent line to the graph of $f(x)$ at the point $\overline{x}$. The new iterate, x', is the point where this tangent line crosses the x axis. If $f(x)$ is close to zero, then x' should be close to $\overline{x}$ and the tangent line approximation should be close to f at x', which suggests that $f(x')$ should be small.

More analytically, the tangent line approximation (See Section 3.1) is

$$f(x) \approx F^{(1)}(x) = f(\overline{x}) + f'(\overline{x}) \cdot (x - \overline{x}) \ . \tag{6.1}$$

Finding where the line crosses the x axis is the same as setting $F^{(1)}(x) = 0$ and solving for x':

$$x' = \overline{x} - f'(\overline{x})^{-1} f(\overline{x}) \ . \tag{6.2}$$

This is the basic Newton method.

The local convergence rate of Newton's method is governed by the error in the approximation (6.1). The analysis assumes that the root x_* is *non-degenerate*, which means that $f'(x_*) \neq 0$. The convergence for degenerate roots is different, see Exercise 1. For a non-degenerate root, we will have $f'(\overline{x}) \neq 0$ for $\overline{x}$ close enough to x_*. Assuming this, (6.2) implies that $|x' - \overline{x}| = O(|f(\overline{x})|)$. This, together with the Taylor series error bound

$$f(x') - F^{(1)}(x') = O\left(|x' - \overline{x}|^2 \right) \ ,$$

and the *Newton equation* $F^{(1)}(x') = 0$, implies that

$$|f(x')| = O\left(|f(\overline{x})|^2 \right) \ .$$

This means that there is a $C > 0$ so that

$$|f(x')| \leq C \cdot |f(\overline{x})|^2 \ . \tag{6.3}$$

This manifestation of *local quadratic convergence* says that the residual at the next iteration is roughly proportional[4] to the square of the residual at the current iterate.

Quadratic convergence is very fast. In a typical problem, once $x_k - x_*$ is moderately small, the residual will be at roundoff levels in a few more iterations. For example, suppose that[5] $C = 1$ in (6.3) and that $|x_k - x_*| = .1$. Then $|x_{k+1} - x_*| \leq .01$, $|x_{k+2} - x_*| \leq 10^{-4}$, and $|x_{k+4} - x_*| \leq 10^{-16}$. The number

[4] Strictly speaking, (6.3) is just a bound, not an estimate. However, Exercise 2 shows that $f(x')$ really is approximately proportional to $f(\overline{x})^2$.

[5] This does not make sense on dimensional grounds. It would be more accurate and more cumbersome to describe this stuff in terms of relative error.

of correct digits doubles at each iteration. By contrast, a *linearly convergent* iteration with $|x_{k+1} - x_*| \leq .1 \cdot |x_k - x_*|$ gains one digit of accuracy per iteration and takes 15 iterations rather than 4 to go from .1 to 10^{-16} Bisection search needs about 50 iterations to reduce the error by a factor of 10^{15} ($10^{15} = \left(10^3\right)^5 \approx \left(2^{10}\right)^5 = 2^{50}$).

Unfortunately, the quadratic convergence of Newton's method is local. There is no guarantee that $x_k \rightarrow x_*$ as $k \rightarrow \infty$ if the initial guess is not close to x_*. A program for finding x_* must take this possibility into account. See Section 3.7.3 for some ideas on how to do this.

6.3 Newton's method in more than one dimension

Newton's method applies also to solving systems of nonlinear equations. The linear approximation (6.1) applies in dimensions $n > 1$ if $f'(\overline{x})$ is the Jacobian matrix evaluated at $\overline{x}$, and f and $(x - \overline{x})$ are column vectors. We write the *Newton step* as $x' - \overline{x} = z$, so $x' = \overline{x} + z$. Newton's method determines z by replacing the nonlinear equations, $f(\overline{x} + z) = 0$, with the linear approximation,

$$0 = f(\overline{x}) + f'(\overline{x})z \ . \tag{6.4}$$

To carry out one step of Newton's method, we must evaluate the function $f(\overline{x})$, the Jacobian, $f'(\overline{x})$, then solve the linear system of equations (6.4). We may write this as

$$z = -\left(f'(\overline{x})\right)^{-1} f(\overline{x}) \ , \tag{6.5}$$

which is a natural generalization of the one dimensional formula (6.2). In computational practice (see Exercise 1) it usually is more expensive to form $\left(f')\overline{x})\right)^{-1}$ than to solve (6.4).

Newton's method for systems of equations also has quadratic (very fast) local convergence to a non-degenerate solution x_*. As for the one dimensional case, this is because of the error bound in the linear approximation (6.1). For $n > 1$, we write the Taylor approximation error bound in terms of norms:

$$\left\| f(\overline{x} + z) - F^{(1)} \right\| = \left\| f(\overline{x} + z) - \left\{ \ f(\overline{x}) + f'(\overline{x})z \ \right\} \right\| = O\left(\|z\|^2 \right) \ .$$

We see from (6.5) that[6]

$$\|z\| \leq C \, \|f(\overline{x})\| \ .$$

Together, these inequalities imply that if $\overline{x} - x_*$ is small enough then

$$\|f(x')\| = \|f(\overline{x} + z)\| \leq C \, \|f(\overline{x})\|^2 \ ,$$

[6]The definition of a non-degenerate solution is that $f'(x_*)$ is nonsingular. If $\overline{x}$ is close enough to x_*, then $f'(\overline{x})$ will be close enough to $f'(x_*)$ that it also will be nonsingular (See (4.17)). Therefore $\|z\| \leq \left\| \left(f'(\overline{x})\right)^{-1} \right\| \|f(\overline{x})\| \leq C \, \|f(\overline{x})\|$.

which is quadratic convergence, exactly as in the one dimensional case.

In practice, Newton's method can be frustrating for its lack of robustness. The user may need some ingenuity to find an x_0 close enough to x_* to get convergence. In fact, it often is hard to know whether a system of nonlinear equations has a solution at all. There is nothing as useful as the intermediate value theorem from the one dimensional case, and there is no multi-dimensional analogue of the robust but slow bisection method in one dimension.

While Newton's method can suffer from extreme ill conditioning, it has a certain robustness against ill conditioning that comes from its *affine invariance*. Affine invariance states Newton's method is invariant under *affine transformations*. An affine transformation is a mapping $x \to Ax + b$ (it would be linear without the b). An affine transformation[7] of $f(x)$ is $g(y) = Af(By)$, where A and B are invertible $n \times n$ matrices. The affine invariance is that if we start from corresponding initial guesses: $x_0 = By_0$, and create iterates y_k by applying Newton's method to $g(y)$ and x_k by applying Newton's method to $f(x)$, then the iterates also correspond: $x_k = By_k$. This means that Newton's method works exactly as well on the original equations $f(x) = 0$ as on the transformed equations $g(y) = 0$. For example, we can imagine changing from x to variables y in order to give each of the unknowns the same units. If $g(y) = 0$ is the best possible rescaling of the original equations $f(x) = 0$, then applying Newton's method to $f(x) = 0$ gives equivalent iterates.

This argument can be restated informally as saying that Newton's method makes sense on dimensional grounds and therefore is natural. The variables $x_1, \ldots, x_n$ may have different units, as may the functions $f_1, \ldots, f_n$. The n^2 entries $f'(x)$ all may have different units, as may the entries of $(f')^{-1}$. The matrix vector product that determines the components of the Newton step (see (6.4)), $z = -(f')^{-1} f(\overline{x})$, involves adding a number of contributions (entries in the matrix $(f')^{-1}$ multiplying components of f) that might seem likely to have a variety of units. Nevertheless, each of the n terms in the sum implicit in the matrix-vector product (6.5) defining a component z_j has the same units as the corresponding component, x_j. See Section 6.6 for a more detailed discussion of this point.

6.3.1 Quasi-Newton methods

Local quadratic convergence is the incentive for evaluating the Jacobian matrix. Evaluating the Jacobian matrix may not be so expensive if much of the work in evaluating f can be re-used in calculating f'. There are other situations where the Jacobian is nearly impossible to evaluate analytically. One possibility would be to estimate f' using finite differences. Column k of $f'(\overline{x})$ is $\partial_{x_k} f(\overline{x})$. The cheapest and least accurate approximation to this is the first-order one-sided difference formula (e_k is the unit vector in the x_k direction and Δx_k is a step size in that direction. Different components of x may have different units and

[7] Actually, this is a linear transformation. It is traditional to call it affine though the constant terms are missing.

therefore require different step sizes.): $(f(\overline{x} + \Delta x_k e_x) - f(\overline{x}))/\Delta x_k$. Evaluating all of f' in this way would take n extra evaluations of f per iteration, which may be so expensive that it outweighs the fast local convergence.

Quasi Newton methods replace the true $f'(\overline{x})$ in the Newton equations (6.4) by estimates of $f'(\overline{x})$ built up from function values over a sequence of iterations. If we call this approximate Jacobian A_k, the quasi Newton equations are

$$0 = f(x_k) + A_k z_k \ . \tag{6.6}$$

The simplest such method is the *secant method* for one-dimensional root finding. Using the current x_k and $f(x_k)$, and the previous x_{k-1} and $f(x_{k-1})$, We use the slope of the line connecting the current $(x_k, f(x_k))$ to the previous $(x_{k-1}, f(x_{k-1}))$ to estimate the slope of the tangent line at x_k. The result is

$$A_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \ , \quad x_{k+1} = x_k - f(x_k)/A_k \ . \tag{6.7}$$

The local convergence rate of the secant method (6.7) is better than linear ($|x_{k+1} - x_*| \leq C\,|x_k - x_*|$) and worse than quadratic.

Most multidimensional quasi-Newton methods work by updating A_k at each iteration so that $A_{k+1} z_k = f(x_{k+1}) - f(x_k)$. In multidimensions, this does not determine A_{k+1} completely because it represents n equations for the n^2 elements of A_{k+1}. The references give several suggestions for update formulas. The good ones have the property that if you you apply them to linear equations, you find the exact $A = f'$ in n steps. It is not clear that such a property makes quasi Newton methods better than ordinary Newton's method with finite difference approximations to the elements of the Jacobian.

6.4 One variable optimization

Suppose $n = 1$ and we wish to find the minimum of the function of a single variable, $V(x)$. Please bear with the following long list of definitions. We say that x_* is a *local minimum* of V if there is an $R > 0$ so that $V(x_*) \leq V(x)$ whenever $|x - x_*| \leq R$. We say that x_* is a *strict* local minimum if $V(x) > V(x_*)$ whenever $x \neq x_*$ and $|x - x_*| \leq R$. We say that x_* is a *global* minimum if $V(x_*) \leq V(x)$ for all x for which $V(x)$ is defined, and a strict global minimum if $V(x_*) < V(x)$ for all $x \neq x_*$ for which V is defined. Finally, x_* is a *nondegenerate* local minimum if $V''(x_*) > 0$. The Taylor series remainder theorem implies that if $V'(x_*) = 0$ and $V''(x_*) > 0$, then x_* is at least a strict local minimum. The function $V(x)$ is *convex* if [8] $\alpha V(x) + \beta V(y) > V(\alpha x + \beta y)$ whenever $\alpha \geq 0$, $\beta \geq 0$, and $\alpha + \beta = 1$. The function is *strictly convex* if $V''(x) > 0$ for all x. A strictly convex function is convex, but the function $V(x) = x^4$ is not strictly convex, because $V''(0) = 0$. This function has a strict but degenerate global minimum at $x_* = 0$.

[8] The reader should check that this is the same as the geometric condition that the line segment connecting the points $(x, V(x))$ and $(y, V(y))$ lies above the graph of V.

For the curious, there is an analogue of bisection search in one variable optimization called *golden section search*. It applies to any continuous function that is *unimodal*, meaning that V has a single global minimum and no local minima. The *golden mean*[9] is $r = (1+\sqrt{5})/2 \approx 1.62$. At each stage of bisection search we have an interval $[\overline{a}, \overline{b}]$ in which there must be at least one root. At each stage of golden section search we have an interval $[\overline{a}, \overline{c}]$ and a third point $\overline{b} \in [\overline{a}, \overline{c}]$ with

$$|\overline{a} - \overline{c}| = r \left|\overline{a} - \overline{b}\right| \ . \tag{6.8}$$

As with our discussion of bisection search, the notation $[\overline{a}, \overline{c}]$ does not imply that $\overline{a} < \overline{c}$. In bisection, we assume that $f(\overline{a}) \cdot f(\overline{b}) < 0$. Here, we assume that $f(\overline{b}) < f(\overline{a})$ and $f(\overline{b}) < f(\overline{c})$, so that there must be a local minimum within $[\overline{a}, \overline{c}]$. Now (this is the clever part), consider a fourth point in the larger subinterval, $d = (1 - \frac{1}{r})\overline{a} + \frac{1}{r}\overline{b}$. Evaluate $f(\overline{d})$. If $f(\overline{d}) < f(\overline{b})$, take $a' = \overline{a}$, $b' = \overline{d}$, and $c' = \overline{b}$. Otherwise, take $a' = \overline{c}$, $b' = \overline{b}$, and $c' = \overline{d}$, reversing the sense of the interval. In either case, $|a' - c'| = r\,|a' - b'|$, and $f(b') < f(a')$ and $f(b') < f(c')$, so the iteration can continue. Each stage reduces the uncertainty in the minimizer by a factor of $\frac{1}{r}$, since $|a' - c'| = \frac{1}{r}\,|\overline{a} - \overline{c}|$.

6.5 Newton's method for local optimization

Most of the properties listed in Section 6.4 are the same for multi-variable optimization. We denote the gradient as $g(x) = \nabla V(x)$, and the Hessian matrix of second partials as $H(x)$. An x_* with $g(x_*) = 0$ and $H(x_*)$ positive definite (see Section 5.3 and Exercise 2) is called non-degenerate, a natural generalization of the condition V'' for one variable problems. Such a point is at least a local minimum because the Taylor series with error bound is

$$V(x_* + z) - V(x_*) = \frac{1}{2} z^* H(x_*) z + O(\|z\|^3) \ .$$

Exercise 2 shows that the first term on the right is positive and larger than the second if $H(x_*)$ is positive definite and $\|z\|$ is small enough. If $H(x_*)$ is negative definite (obvious definition), the same argument shows that x_* is at least a local maximum. If $H(x_*)$ has some positive and some negative eigenvalues (and $g(x_*) = 0$) then x_* is neither a local minimum nor a local maximum, but is called a *saddle point*. In any case, a local minimum must satisfy $g(x_*) = 0$ if V is differentiable.

We can use Newton's method from Section 6.3 to seek a local minimum by solving the equations $g(x) = 0$, but we must pay attention to the difference between row and column vectors. We have been considering x, the Newton step, z, etc. to be column vectors while $\nabla V(x) = g(x)$ is a row vector. For

[9] This number comes up in many ways. From fibonacci numbers it is $r = \lim_{k\to\infty} f_{k+1}/f_k$. If $(\alpha + \beta)/\alpha = \alpha/\beta$ and $\alpha > \beta$, then $\alpha/\beta = r$. This has the geometric intertretation that if we remove an $\alpha \times \alpha$ square from one end of an $\alpha \times (\alpha + \beta)$ rectangle, then the remaining smaller $\beta \times \alpha$ rectangle has the same aspect ratio as the original $\alpha \times (\alpha + \beta)$ rectangle. Either of these leads to the equation $r^2 = r + 1$.

this reason, we consider applying Newton's method to the column vector of equations $g^*(x) = 0$. The Jacobian matrix of the column vector function $g^*(x)$ is the Hessian H (check this). Therefore, the locally convergent Newton method is

$$x' = \overline{x} + z \ ,$$

where the step z is given by the Newton equations

$$H(\overline{x})z = -g^*(\overline{x}) \ . \tag{6.9}$$

Because it is a special case of Newton's method, it has local quadratic convergence to x_* if x_* is a local non-degenerate local minimum.

Another point of view for the local Newton method is that each iteration minimizes a *quadratic model* of the function $V(\overline{x} + z)$. The three term Taylor series approximation to V about $\overline{x}$ is

$$V(\overline{x} + z) \approx V^{(2)}(\overline{x}, z) = V(\overline{x}) + \nabla V(\overline{x})z + \frac{1}{2} z^* H(\overline{x}) z \ . \tag{6.10}$$

If we minimize $V^{(2)}(\overline{x}, z)$ over z, the result is $z = -H(\overline{x})^{-1} \nabla V(\overline{x})^*$, which is the same as (6.9). As for Newton's method for nonlinear equations, the intuition is that $V^{(2)}(\overline{x}, z)$ will be close to $V(\overline{x} + z)$ for small z. This should make the minimum of $V^{(2)}(\overline{x}, z)$ close to the minimizer of V, which is x_*.

Unfortunately, this simple local method cannot distinguish between between a local minimum, a local maximum, or even a saddle point. If x_* has $\nabla V(x_*) = 0$ (so x_* is a *stationary point*) and $H(x_*)$ is nonsingular, then the iterates $x_{k+1} = x_k - H(x_k)^{-1} g^*(x_k)$ will happily converge to to x_* if $\|x_0 - x_*\|$ is small enough. This could be a local maximum or a saddle point. Moreover, if $\|x_0 - x_*\|$ is not small, we have no idea whether the iterates will converge to anything at all.

The main difference between the unsafeguarded Newton method optimization problem and general systems of nonlinear equations is that the Hessian is symmetric and (close enough to a non-degenerate local minimum) positive definite. The Jacobian f' need not be symmetric. The Choleski decomposition requires about storage for the roughly $\frac{1}{2}n^2$ distinct elements of H and about $\frac{1}{6}n^3$ floating points to compute L. This is about half the storage and work required for a general non-symmetric linear system using the LU factorization.

6.6 Safeguards and global optimization

The real difference between minimization and general systems of equations comes from the possibility of evaluating $V(x)$ and forcing it to decrease from iteration to iteration. It is remarkable that two simple *safeguards* turn the unreliable Newton's method into a much more robust (though not perfect) method that converges to a local minimum from almost any initial guess. These are (*i*) finding a *descent direction* by modifying $H(\overline{x})$ if necessary, and (*ii*) using a one dimensional *line search* to prevent wild steps. Both of the safeguards have the purpose of guaranteeing descent, that $V(x') < V(\overline{x})$.

In principle, this would allow the x_k to converge to a saddle point, but this is extremely unlikely in practice because saddle points are unstable for this process.

The safeguarded methods use the formulation of the *search directions*, p and the *step size*, $t > 0$. One iteration will take the form $x' = \overline{x} + z$, where the step is $z = tp$. We define the search direction to be a *descent direction* if

$$\frac{d}{dt} V(\overline{x} + tp)\Big|_{t=0} = \quad g(\overline{x}) \cdot p < 0 \; . \tag{6.11}$$

This guarantees that if $t > 0$ is small enough, then $V(\overline{x} + tp) < V(\overline{x})$. Then we find a step size, t, that actually achieves this property. If we prevent t from becoming too small, it will be impossible for the iterates to converge except to a stationary point.

We find the search direction by solving a modified Newton equation

$$\widetilde{H} p = -g^*(\overline{x}) \; . \tag{6.12}$$

Putting this into (6.11) gives

$$\frac{d}{dt} V(\overline{x} + tp)\Big|_{t=0} = -g(\overline{x}) \widetilde{H} g(\overline{x})^* \; .$$

This is negative if $\widetilde{H}$ is positive definite (the right hand side is a 1×1 matrix (a number) because g is a row vector). One algorithm for finding a descent direction would be to apply the Choleski decomposition algorithm (see Section 5.3). If the algorithm finds L with $LL^* = H(\overline{x})$, use this L to solve the Newton equation (6.12) with $\widetilde{H} = H(\overline{x}) = LL^*$. If the Choleski algorithm fails to find L, then $H(\overline{x})$ is not positive definite. A possible substitute (but poor in practice, see below) is $\widetilde{H} = I$, which turns Newton's method into *gradient descent*.

A better choice for $\widetilde{H}$ comes from the *modified Choleski* algorithm. This simply replaces the equation

$$l_{kk} = \left(H_{kk} - l_{k1}^2 + \cdots + l_{k,k-1}^2 \right)^{1/2}$$

with the modified equation using the absolute value

$$l_{kk} = \left| H_{kk} - l_{k1}^2 + \cdots + l_{k,k-1}^2 \right|^{1/2} \; . \tag{6.13}$$

Here, H_{kk} is the (k,k) entry of $H(\overline{x})$. This modified Choleski algorithm produces L with $LL^* = H(\overline{x})$ if and only if $H(\overline{x})$ is positive definite. In any case, we take $\widetilde{H} = LL^*$, which is positive definite. Using these non-Choleski factors, the Newton equations become:

$$LL^* p = -g(\overline{x})^* \; . \tag{6.14}$$

It is not entirely clear why the more complicated modified Choleski algorithm is more effective than simply taking $\widetilde{H} = I$ when $H(\overline{x})$ is not positive definite.

One possible explanation has to do with units. Let us suppose that U_k represents the units of x_k, such as seconds, dollars, kilograms, etc. Let us also suppose that $V(x)$ is dimensionless. In this case the units of $H_{jk} = \partial_{x_j}\partial_{x_k}V$ are $[H_{jk}] = 1/U_jU_k$. We can verify by studying the Choleski decomposition equations from Section **??** that the entries of L have units $[l_{jk}] = 1/U_j$, whether we use the actual equations or the modification (6.13). We solve (**??**) in two stages, first $Lq = -\nabla V^*$, then $L^*p = q$. Looking at units, it is clear that all the elements of q are dimensionless and that the elements of p have units $[p_k] = U_k$. Thus, the modified Choleski algorithm produces a search direction that component by component has the same units as x. This allows the update formula $x' = \overline{x} + tp$ to make sense with a dimensionless t. The reader should check that the choice $\widetilde{H} = I$ does not have this property in general, even if we allow t to have units, if the U_k are different.

The second safeguard is a limited *line search*. In general, line search means minimizing the function $\phi(t) = V(x + tp)$ over the single variable t. This could be done using golden section search, but a much more rudimentary binary search process suffices as a safeguard. In this binary search, we evaluate $\phi(0) = V(\overline{x})$ and $\phi(1) = V(\overline{x} + p)$. If $\phi(1) > \phi(0)$, the step size is too large. In that case, we keep reducing t by a factor of 2 (`t = t/2;`) until $\phi(t) < \phi(0)$, or we give up. If p is a search direction, we will have $\phi(t) < \phi(0)$ for small enough t and this bisection process will halt after finitely many reductions of t. If $\phi(1) < \phi(0)$, we enter a greedy process of increasing t by factors of 2 until $\phi(2t) > \phi(t)$. This process will halt after finitely many doublings if the set of x with $V(x) < V(\overline{x})$ is bounded.

A desirable feature is that the safeguarded algorithm gives the ordinary Newton step, and rapid (quadratic) local convergence, if $\overline{x}$ is close enough to a nondegenerate local minimum. The modified Hessian will correspond to the actual Hessian if $H(x_*)$ is positive definite and $\overline{x}$ is close enough to x_*. The step size will be the default $t = 1$ if $\overline{x}$ is close enough to x_* because the quadratic model (6.10) will be accurate. The quadratic model has $V^{(2)}(\overline{x}, 2z) > V^{(2)}(\overline{x}, z)$, because z is the minimizer of $V^{(2)}$.

6.7 Gradient descent and iterative methods

The *gradient descent* optimization algorithm uses the identity matrix as the approximate Hessian, $\widetilde{H} = I$, so the (negative of the) gradient becomes the search direction: $p = -\nabla V(\overline{x})^*$. This seems to make sense geometrically, as the negative gradient is the steepest downhill direction (leading to the name *method of steepest descent*). With a proper line search, gradient descent has the theoretical global robustness properties of the more sophisticated Newton method with the modified Choleski approximate Hessian. But much of the research in sophisticated optimization methods is motivated by the fact that simple gradient descent converges slowly in many applications.

One indication of trouble with gradient descent is that the formula,

$$x' = \overline{x} - t\nabla V(\overline{x})^* \ , \tag{6.15}$$

does not make dimensional sense in general, see Section 6.6. Written in components, (6.15) is $x'_k = \overline{x}_k - t\partial_{x_k} V(\overline{x})$. Applied for $k = 1$, this makes dimensional sense if the units of t satisfy $[t] = [x_1^2]/[V]$. If the units of x_2 are different from those of x_1, the x_2 equation forces units of t inconsistent with those from the x_1 equation.

We can understand the slow convergence of gradient descent by studying how it works on the model problem $V(x) = \frac{1}{2}x^* H x$, with a symmetric and positive definite H. This is the same as assuming that the local minimum is nondegenerate and the local approximation (6.10) is exact[10]. In this case the gradient satisfies $g(x)^* = \nabla V(x)^* = Hx$, so solving the Newton equations (6.9) gives the exact solution in one iteration. We study the gradient method with a fixed step size[11], t, which implies $x_{k+1} = x_k - tHx_k$. We write this as

$$x_{k+1} = Mx_k \ , \tag{6.16}$$

where

$$M = I - tH \ . \tag{6.17}$$

The convergence rate of gradient descent in this case is the rate at which $x_k \to 0$ in the iteration (6.16).

This, in turn, is related to the eigenvalues of M. Since H and M are symmetric, we may choose an orthonormal basis in which both are diagonal: $H = \text{diag}(\lambda_1, \ldots \lambda_n)$, and $M = \text{diag}(\mu_1, \ldots \mu_n)$. The λ_j and positive, so we may assume that $0 < \lambda_{\min} = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n = \lambda_{\max}$ The formula (6.17) implies that

$$\mu_j = 1 - t\lambda_j \ . \tag{6.18}$$

After k iterations of (6.16), we have $x_{kj} = \mu_j^k x_{0j}$, where x_{kj} component j of the iterate x_k. Clearly, the rate at which $x_k \to 0$ depends on the *spectral gap*,

$$\rho = 1 - \max_j |\mu_j| \ ,$$

in the sense that the estimate

$$\|x_k\| \leq (1 - \rho)^k \, \|x_0\|$$

is sharp (take $x_0 = e_1$ or $x_0 = e_n$). The optimal step size, t is the one that maximizes ρ, which leads to (see (6.18)

$$\begin{aligned} 1 - \rho &= \mu_{\max} = 1 - t\lambda_{\min} \\ \rho - 1 &= \mu_{\min} = 1 - t\lambda_{\max} \ . \end{aligned}$$

Solving these gives the optimizing value $t = 2/(\lambda_{\min} + \lambda_{\min})$ and

$$\rho = 2 \cdot \frac{\lambda_{\min}}{\lambda_{\max}} \approx \frac{2}{\kappa(H)} \ . \tag{6.19}$$

[10] We simplified the problem but not lost generality by taking $x_* = 0$ here.

[11] See Exercise 7 for an example showing that line search does not improve the situation very much.

If we take $k = 2/\rho \approx \kappa(H)$ iterations, and H is ill conditioned so that k is large, the error is reduced roughly by a factor of

$$(1-\rho)^k = \left(1 - \frac{2}{k}\right)^k \approx e^{-2} .$$

This justifies what we said in the Introduction, that it takes $k = \kappa(H)$ iterations to reduce the error by a fixed factor.

6.8 Resources and further reading

The book by Ortega and Reinbolt has a more detailed discussion of Newton's method for solving systems of nonlinear equations. The book*Practical Optimization* by Phillip Gill, Walter Murray, and my colleague Margaret Wright, has much more on nonlinear optimization including methods for constrained optimization problems. There is much public domain software for smooth optimization problems, but I don't think much of it is useful.

The set of initial guesses x_0 so that $k_k \to x_*$ as $t \to \infty$ is the *basin of attraction* of x_*. If the method is locally convergent, the basin of attraction contains a ball of radius R about x_*. The boundary of the basin of attraction can be a beatiful fractal set. The picture book *Fractals* by Benoit Mandelbrot, some of the most attractive fractals arise in this way.

6.9 Exercises

1. Study the convergence of Newton's method applied to solving the equation $f(x) = x^2 = 0$. Show that the root $x_* = 0$ is *degenerate* in that $f'(x_*) = 0$. The Newton iterates are x_k satisfying $x_{k+1} = x_k - f(x_k)/f'(x_k)$. Show that the local convergence in this case is *linear*, which means that there is an $\alpha < 1$ with $|x_{k+1} - x_*| \approx \alpha\,|x_k - x_*|$. Note that so called linear convergence still implies that $x_k - x_* \to 0$ exponentially. Nevertheless, contrast this linear local convergence with the quadratic local convergence for a nondegenerate problem.

2. Use the Taylor expansion to second order to derive the approximation

$$f(x') \approx C(\overline{x}) f(\overline{x})^2 = \frac{1}{2}\frac{f''(\overline{x})}{f'(\overline{x})^2} \cdot f(\overline{x})^2 . \tag{6.20}$$

   Derive a similar expression that shows that $(x' - x_*)$ is approproximately proportional to $(\overline{x} - x_*)^2$. Use (6.20) to predict that applying Newton's method to finding solving the equation $\sin(x) = 0$ will have superquadratic convergence. What makes $C(x)$ large, and the convergence slow, is (i) small $f'(x)$ (a nearly degenerate problem), and (ii) large $f''(x)$ (a highly nonlinear problem).

3. The function $f(x) = x/\sqrt{1+x^2}$ has a unique root: $f(x) = 0$ only for $x = 0$. Show that the unsafeguarded Newton method gives $x_{k+1} = x_k^3$. Conclude that the method succeeds if and only if $|x_0| < 1$. Draw graphs to illustrate the first few iterates when $x_0 = .5$ and $x_0 = 1.5$. Note that Newton's method for this problem has local cubic convergence, which is even faster than the more typical local quadratic convergence. The formula (6.20) explains why.

4. Suppose $n = 2$ and x_1 has units of (electric) charge, x_2 has units of mass, $f_1(x_1, x_2)$ has units of length, and $f_2(x_1, x_2)$ has units of time. Find the units of each of the four entries of $(f')^{-1}$. Verify the claims about the units of the step, z, at the end of Section 6.3.

5. Suppose x_* satisfies $f(x_*) = 0$. The *basin of attraction* of x_* is the set of x so that if $x_0 = x$ then $x_k \to x_*$ as $k \to \infty$. If $f'(x_*)$ is non-singular, the basin of attraction of x_* under unsafeguarded Newton's method includes at least a neighborhood of x_*, because Newton's method is locally convergent. Exercise 3 has an example in one dimension where the basin of attraction of $x_* = 0$ is the open interval (endpoints not included) $(-1, 1)$. Now consider the two dimensional problem of finding roots of $f(z) = z^2 - 1$, where $z = x+iy$. Written out in its real components, $f(x, y) = (x^2-y^2-1, 2xy)$. The basin of attraction of the solution $z_* = 1$ ($(x_*, y_*) = (1, 0)$) includes a neighborhood of $z = 1$ but surprisingly many many other points in the complex plane. This *Mandlebrot set* is one of the most beautiful examples of a two dimensional fractal. The purpose of this exercise is to make a pretty picture, not to learn about scientific computing.

   (a) Show that Newton iteration is $z_{k+1} = z_k - \frac{z_k^2-1}{2z_k}$.

   (b) Set $z_k = 1 + w_k$ and show that $w_{k+1} = \frac{3}{2}w_k^2/(1 + w_k)$.

   (c) Use this to show that if $|w_k| < \frac{1}{4}$, then $|w_{k+1}| < \frac{1}{2}|w_k|$. Hint: Show $|1 + w_k| > \frac{3}{4}$. Argue that this implies that the basin of attraction of $z_* = 1$ includes at least a disk of radius $\frac{1}{4}$ about z_*, which is a quantitative form of local convergence.

   (d) Show that if $|z_k - 1| < \frac{1}{4}$ for some k, then z_0 is in the basin of attraction of $z_* = 1$. (This is the point of parts (b) and (c).)

   (e) Use part (d) to make a picture of the Mandlebrot set. Hint: Divide the rectangle $|x| < R_x$, $0 \le y \le R_y$ into a regular grid of small cells of size $\Delta x \times \Delta y$. Start Newton's method from the center of each cell. Color the cell if $|z_k - 1| < \frac{1}{4}$ for some $k \le N$. See how the picture depends on the parameters Δx, Δy, R_x, R_y, and N.

6. A *saddle point*[12] is an x so that $\nabla V(x) = 0$ and the Hessian, $H(x)$, is nonsingular and has at least one negative eigenvalue. We do not want the

[12]The usual definition of saddle point is that H should have at least one positive and one negative eigenvalue and no zero eigenvalues. The simpler criterion here suffices for this application.

iterates to converge to a saddle point, but most Newton type optimization algorithms seem to have that potential. All the safeguarded optimization methods we discussed have $\Phi(x) = x$ if x is a saddle point because they all find the search direction by solving $\widetilde{H}p = -\nabla V(x)$.

(a) Let $V(x) = x_1^2 - x_2^2$ and suppose $\overline{x}$ is on the x_1 axis. Show that with the modified Choleski, x' also is on the x_1 axis, so the iterates converge to the saddle point, $x = 0$. Hint: $\widetilde{H}$ has a simple form in this case.

(b) Show that if $\overline{x}_2 \neq 0$, and $t > 0$ is the step size, and we use the bisection search that increases the step size until $\phi(t) = V(\overline{(}x) + tp)$ satisfies $\phi(2t) > \phi(t)$, then one of the following occurs:

i. The bisection search does not terminate, $t \to \infty$, and $\phi(t) \to -\infty$. This would be considered good, since the minimum of V is $-\infty$.

ii. The line search terminates with t satisfying $\phi(t) = V(x') < 0$. In this case, subsequent iterates cannot converge to $x = 0$ because that would force V to converge to zero, while our modified Newton strategy guarantees that V decreases at each iteration.

(c) *Nonlinear least squares* means finding $x \in R^m$ to minimize $V(x) = \|f(x) - b\|_{l^2}^2$, where $f(x) = (f_1(x), \ldots, f_n(x))^*$ is a column vector of n nonlinear functions of the m unknowns, and $b \in R^n$ is a vector we are trying to approximate. If $f(x)$ is linear (there is an $n \times m$ matrix A with $f(x) = Ax$), then minimizing $V(x)$ is a linear least squares problem. The *Marquart Levenberg* iterative algorithm solves a linear least squares problem at each iteration. If the current iterate is $\overline{x}$, let the linearization of f be the $n \times m$ Jacobian matrix A with entries $a_{ij} = \partial_{x_j} f_i(\overline{x})$. Calculate the step, p, by solving

$$\min_p \|Ap - (b - f(\overline{x}))\|_{l^2} \ . \tag{6.21}$$

Then take the next iterate to be $x' = \overline{x} + p$.

i. Show that this algorithm has local quadratic convergence if the residual at the solution has zero residual: $r(x_*) = f(x_*) - b = 0$, but not otherwise (in general).

ii. Show that p is a descent direction.

iii. Describe a safeguarded algorithm that probably will converge to at least a local minimum or diverge.

7. This exercise shows a connection between the slowness of gradient descent and the condition number of H in one very special case. Consider minimizing the model quadratic function in two dimensions $V(x) = \frac{1}{2}\left(\lambda_1 x_1^2 + \lambda_2 x_2^2\right)$ using gradient descent. Suppose the line search is done exactly, choosing t to minimize $\phi(t) = V(\overline{x} + tp)$, where $p = \nabla V(\overline{x})$. In

general it is hard to describe the effect of many minimization steps because the iteration $\overline{x} \rightarrow x'$ is nonlinear. Nevertheless, there is one case we can understand.

(a) Show that for any V, gradient descent with exact line search has p_{k+1} orthogonal to p_k. Hint: otherwise, the step size t_k was not optimal.

(b) In the two dimensional quadratic optimization problem at hand, show that if p_k is in the direction of $(-1,-1)^*$, then p_{k+1} is in the direction of $(-1,1)^*$.

(c) Show that p_k is in the direction of $(-1,-1)^*$ if and only if $(x_1,x_2) = r(\lambda_2,\lambda_1)$, for some r,

(d) Since the optimum is $x_* = (0,0)^*$, the error is $\|(x_1,x_2)\|$. Show that if p_0 is in the direction of $(-1,-1)$, then the error decreases exactly by a factor of $\rho = (\lambda_1 - \lambda_2)/(\lambda_1 + \lambda_2)$ if $\lambda_1 \geq \lambda_2$ (including the case $\lambda_1 = \lambda_2$).

(e) Show that if $\lambda_1 \gg \lambda_2$, then $\rho \approx 1 - 2\lambda_2/\lambda_1 = 1 - 2/\kappa(H)$, where $\kappa(H)$ is the linear systems condition number of H.

(f) Still supposing $\lambda_1 \gg \lambda_2$, show that it takes roughly $n = 1/\kappa(H)$ iterations to reduce the error by a factor of e^2.

8. This exercise walks you through construction of a robust optimizer. It is as much an exercise in constructing scientific software as in optimization techniques. You will apply it to finding the minimum of the two variable function

$$V(x,y) = \frac{\psi(x,y)}{\sqrt{1+\psi(x,y)^2}} \quad , \quad \psi(x,y) = \psi_0 + wx^2 + \big(y - a\sin(x)\big)^2 \, .$$

Hand in output documenting what you for each of the of the parts below.

(a) Write procedures that evaluate $V(x,y)$, $g(x,y)$, and $H(x,y)$ analytically. Write a tester that uses finite differences to verify that g and H are correct.

(b) Implement a local Newton's method without safeguards as in Section 6.5. Use the Choleski decomposition code from Exercise 3. Report failure if H is not positive definite. Include a stoping criterion and a maximum iteration count, neither hard wired. Verify local quadratic convergence starting from initial guess $(x_0,y_0) = (.3,.3)$ with parameters $\psi_0 = .5$, $w = .5$, and $a = .5$. Find an initial condition from which the unsafeguarded method fails.

(c) Modify the Choleski decomposition code Exercise 5.3 to do the modified Choleski decomposition described in Section 6.6. This should require you to change a single line of code.

(d) Write a procedure that implements the limited line search strategy described in Section 6.6. This also should have a maximum iteration

count that is not hard wired. Write the procedure so that it sees only a scalar function $\phi(t)$. Test on:

i. $\phi(t) = (t - .9)^2$ (should succeed with $t = 1$).

ii. $\phi(t) = (t - .01)^2$ (should succeed after several step size reductions).

iii. $\phi(t) = (t - 100)^2$ (should succeed after several step size doublings).

iv. $\phi(t) = t$ (should fail after too many step size reductions).

v. $\phi(t) = -t$ (should fail after too many doublings).

(e) Combine the procedures from parts (c) and (d) to create a robust global optimization code. Try the code on our test problem with $(x_0, y_0) = (10, 10)$ and parameters $\psi_0 = .5$, $w = .02$, and $a = 1$. Make plot that shows contour lines of V and all the iterates.