# Nonlinear equations and optimization*

Jonathan Goodman

April 8, 1999

## 1 Introduction

This section discusses numerical solution of systems of nonlinear equations and the problem of finding the maximum or minimum of a smooth function of one or more variables (optimization). These problems are closely related. Most optimization algorithms make strong use of the system of equations gotten by setting the partial derivatives of the objective function (the function being minimized or maximized) to zero.

The algorithms discussed here are *iterative*; they produce a sequence of approximate solutions of increasing accuracy. The total algorithm consists of a sequence of *iterations*. At the beginning of each iteration, there is the *current iterate*, $\bar{x}$. One iteration of the algorithm produces a (hopefully) better approximation, $x'$. The solution being sought is $x_*$. The iterative process starts with an *initial guess*, $x_0$. We take this as $\bar{x}$, apply an iteration of the algorithm and produce a better guess, $x_1 = x'$, and so on. We never expect to have the exact answer, $x_*$. Rather, (if the algorithm works) the iterates will *converge* to $x_*$: $x_n \to x_*$ as $n \to \infty$. The rate at which $\|x_n - x_*\| \to 0$ as $n \to \infty$ is the *convergence rate*. A good algorithm will have a rapid convergence rate.

Generally, we stop the iteration process when some *convergence criterion* is satisfied. This criterion could be that the iterates have stopped changing[1]: $\|x_{k+1} - x_k\| \,/\, \|x_k\| \leq \epsilon$. The $\epsilon$ is often called the *tolerance*. We could also stop when the *residuals* reach a certain tolerance level.

Iterative algorithms never produce the exact answer. However, good algorithms and well conditioned problems often allow

$$\frac{\|x_k - x_*\|}{\|x_*\|} \sim \epsilon_{\text{mach}} \ ,$$

which is the best we could hope for on a computer, even from an "exact" solution formula. The fact that we have an iterative algorithm rather than a direct algorithm does not necessarily mean that the eventual computed solution will be less accurate.

Generally, a computer code for solving systems of equations or smooth optimization has a relatively simple "core algorithm" together with *safeguards* that keep the code from giving the wrong answer with poor initial guesses. Producing such software requires several kinds of mathematical analysis and some software engineering skills.

## 2 Solving a single nonlinear equation

The simplest problem is that of solving a single equation in a single variable: $f(x) = 0$. Algorithms for this problem illustrate some features of algorithms for more complex problems. They also differ in significant ways. For example, the bisection algorithm produces a sequence or pairs of points $(a_n, b_n)$ rather than just

*These are course notes for Scientific Computing, given Spring 1999 at the Courant Institute of Mathematical Sciences at New York University. 

[1] Note that we measure relative change rather than absolute change, $\|x_{k+1} - x_k\| \leq \epsilon$.

a sequence of approximate solutions. Algorithms for a single equation may be components of more complex algorithms for solving systems of equations.

## 2.1 Bisection

The bisection algorithm, or bisection search, is the simplest and most robust way to find the root of a single function of one variable. It does not require $f(x)$ to be smooth or even differentiable, but merely a continuous function of $x$. It is based on a simple fact about continuous functions called the *intermediate value theorem*: if $f(a) < 0 < f(b)$ and $f$ is continuous in the closed interval between $a$ and $b$ then there is an $x_*$ between $a$ and $b$ so that $f(x_*) = 0$. In other words, it is impossible to jump from negative to positive without actually crossing zero. The wording of the theorem was chosen to be correct whether or not $a < b$.

The bisection algorithm consists of repeatedly bisecting the interval in which a root is known to lie. Suppose we have an interval[2] $[\bar{a}, \bar{b}]$ with $f(\bar{a}) < 0$ and $f(\bar{b}) > 0$. The intermediate value theorem (or "common sense") tells us that there is a root of $f$ in $[\bar{a}, \bar{b}]$. The uncertainty in the location of the root is the length of the interval: $|b - a|$. To cut that uncertainty in half, we bisect the interval. The midpoint is $\bar{c} = (\bar{a} + \bar{b})/2$. We evaluate $f(\bar{c})$ to determine its sign. If $f(\bar{c}) > 0$ then we know there is a root of $f$ in the sub interval $[\bar{a}, \bar{c}]$. In this case, we take $a' = \bar{a}$ and $'b = \bar{c}$. In the other case, $f(\bar{c} < 0$, we may take $a' = \bar{c}$ and $b' = \bar{b}$. In either case, $f$ changes sign over the half size interval $[a', b']$.

To start the bisection algorithm, we must produce an initial interval $[a_0, b_0]$ that brackets a root of $f$. Running the bisection procedure over and over produces a sequence of intervals with whose size is decreasing exponentially at the rate

$$|b_n - a_n| = 2^{-n} |b_0 - a_0| \quad .$$

To get a feeling for the convergence rate, use the (approximate) formula $2^{10} = 10^3$. In this context, the formula tells us that we get three decimal digits of accuracy for each ten iterations. This may seem pretty 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.

Although the bisection algorithm is robust, there is a way it can fail. On a computer, $f(x)$ is not evaluated exactly, but with some roundoff or other error. This roundoff may change the sign of $f$, as reported from the procedure that calculates it. Another way to say this is that because $f$ is not evaluated exactly, the computed approximation to $f$ may not be continuous on a fine scale. A bisection code should take this possibility into account some how, 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.

## 2.2 Newton's method for a nonlinear equation

Newton's method is a *locally convergent* method. This means that the iterates will converge to a root if the initial guess is close enough. How you find a close enough initial guess is your business. For one dimensional problems, you might use bisection search. For optimization problems there are safeguards that allow the algorithm to make progress even from a poor initial guess.

There is a geometric explanation of the Newton iteration for functions of one variable. Suppose that our current iterate, $\bar{x}$, is close to the root, $x_*$. We are able to evaluate $f(\bar{x})$ and $f'(\bar{x})$ but do now know $x_*$. From the information we have, we construct the line tangent to the graph at $(\bar{x}, f(\bar{x}))$. The new iterate, $x'$, is the point where this tangent line crosses the $x$ axis. If $\bar{x}$ is close to $x_*$, then the tangent line will be close to the graph of $f$ still at $x'$, so $x'$ will be quite close to $x_*$.

To program this method, we need a formula for $x'$. The line tangent to the graph at $(\bar{x}, f(\bar{x}))$ has slope $f'(\bar{x})$. Therefore the formula for the line is

$$y - f(\bar{x}) = f'(\bar{x}) \cdot (x - \bar{x}) \quad . \tag{1}$$

We find where this line crosses the $x$ axis by setting $y = 0$ and solving for $x = x'$

$$x' = \bar{x} - f'(\bar{x})^{-1} f(\bar{x}) \quad . \tag{2}$$

[2] 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.

This is Newton's method.

An analytical, as opposed to geometric, approach to Newton's method will help us understand its convergence rate and also make it clear how to generalize it to problems with more than one variable. Again supposing that the root, $x_*$, is close to our current iterate, $\bar{x}$, the Taylor series approximation to $f$ about $\bar{x}$ should be very useful at $x_*$. Keeping just two terms of this series replaces $f$ with a linear approximation:

$$f(x) \approx f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) \quad . \tag{3}$$

We try to set $f = 0$ be setting the approximation on the right side of (3) to zero. This gives

$$0 = f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) \quad , \tag{4}$$

which is the same as (2).

The local convergence rate of Newton's method is governed by the error in the approximation (3). Therefore, we restate (3) more precisely as

$$f(x) = f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) + O(|x - \bar{x}|^2) \quad . \tag{5}$$

Together with (4), this implies that

$$f(x') = O(|x - \bar{x}|^2) \quad .$$

From (2) this becomes

$$|f(x')| \leq C \, |f(\bar{x})|^2 \quad . \tag{6}$$

In other words, one iteration of Newton's method roughly squares the residual. That means that the residual could go, say, from .3 to .1 to .01 to $10^{-4}$ to $10^{-8}$ on four Newton steps. This *local quadratic convergence* of Newton's method is faster than the *linear convergence*[3] of bisection search. Normally, quadratic reduction of the residuals also implies quadratic reduction of the error. In a neighborhood of $x_*$, $f(x) \sim f'(x_*)\,(x - x_*)$, so (6) also implies

$$|x' - x_*| \leq C \, |\bar{x} - x_*|^2 \quad .$$

Newton's method is not as robust as bisection search. The constant in the quadratic convergence estimate (6) will be large if $f'$ is small or $f''$ is large near $x_*$. How far is too far is hard to know in advance. Any code based on Newton's method must take into account the possibility of non convergence.

### 2.3 Minimization problems in one dimension

Again in the situation where $x$ is a single variable, we want to minimize a function, $V(x)$. That is, we want to find $x_*$ so that $V(x_*) < V(x)$ for any $x \neq x_*$. For differentiable $V$, we can find $x_*$ by solving the nonlinear equation

$$V'(x_*) = 0 \quad . \tag{7}$$

We can find the solution of (7) by applying Newton's method to the function $f(x) = V'(x)$. We have to compute $V'(x)$ and $V''(x)$, but we will get quadratic local convergence.

I pause to clarify some terminology associated with minimization (or "optimization"). A differentiable function of $x$, $V(x)$, may or may not have a minimum value. The function $V(x) = x$ clearly has no minimum value. The function $V(x) = 1/(1 + x^2)$ has an "infemum", which is 0 (corresponding to $x = \pm\infty$), but mathematicians do not call this a minimum because it is not "attained": there is no $x_*$ with $V(x_*) = 0$. The function $V(x) = -1/(1 + x^2)$ has a minimum value at $x_* = 0$. A point $x_*$ is a *local minimum* if it gives the smallest value of $V$ for some range of nearby $x$ values, that is if $V(x_*) \leq V(x)$ for all $x$ with $|x - x_*| \leq \epsilon$. It is a *strict* local minimum if $V(x_*) < V(x)$ for $|x_* - x| \leq \epsilon$ and $x \neq x_*$. The function $V(x) = \frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2$ has a strict local minimum at $x = 1$, but the (*global*) minimum is at $x = -2$.

[3] The error at the next step is linearly proportional to the error at the current step. For bisection, 1/2 is the constant of proportionality.

Although we may consider a minimization problem as a nonlinear equation ($V'(x) = 0$), there are two features of minimization problems that distinguish optimization from general nonlinear equation solving. The first is that (7) is not the only criterion for $x_*$ to be a minimizer. We must also have

$$V''(x_*) \geq 0 \ .$$

In nondegenerate cases, this is $V''(x_*) > 0$. One consequence of this is that we should not take the "Newton step" (see below) if $V'' < 0$. This might take us to a local maximum but we are not likely to be close to a local minimum.

The second special feature of minimization is that it is possible to guarantee that $V(x') < V(\bar{x})$ using safeguards discussed below. Because of this, we either go down forever or, like Alice, eventually reach at least a local minimum[4]. This is the reason safeguarded optimization codes can be more robust than general nonlinear equation solving software.

Both safeguards are modifications of Newton's method. the unmodified method, (2) in the present notation is

$$x' = \bar{x} - V''(\bar{x})V'(\bar{x}) \ . \tag{8}$$

This is written

$$x' = \bar{x} + p \ , \tag{9}$$

where

$$p = -V''(\bar{x})V'(\bar{x}) \tag{10}$$

is the *Newton step.*

The first safeguard is to make sure that $p$ is a *descent direction*, that is, that a sufficiently small step in the direction of $p$ reduces $V$ a little. This means that

$$\frac{d}{dt} V(\bar{x} + tp) < 0 \quad \text{when } t = 0. \tag{11}$$

The formula (10) will have this property if $V''(\bar{x}) > 0$ but not otherwise. The simplest fix is to replace (10) with a safeguarded version

$$p = -\left|V''(\bar{x})\right|^{-1} V'(\bar{x}) \ . \tag{12}$$

We probably need to safeguard against the possibility that $V''(\bar{x}) = 0$, possibly replacing it with $\epsilon_{\text{mach}}$ in that case. Note that the safeguarded search direction formula (12) is the same as the unsafeguarded Newton search direction (10) in the neighborhood of a local minimum (where $V'' > 0$). This allows us to use the actual Newton step, and get quadratic convergence, near a local minimum.

The second safeguard protects us from taking a step that is too large or too small. Among the many possible ways to do this, I discuss a form of binary line search. This line search takes a *step* in the search direction of size$t$. That is, (9) is replaced by

$$x' = \bar{x} + tp \ . \tag{13}$$

Define $\phi(t) = V(\bar{x} + tp)$. This search is not supposed to find the actual minimum of $\phi$ accurately. It merely protects against steps that are much too big or small. A binary search can guarantee that we find (or die trying) a $t$ so that there is at least a local minimum, $t_*$, of $\phi$ within a factor of 2 of $t$. That is $t/2 < t_* < 2t$. This will be the case if

$$\phi(t) < \phi(2t) \tag{14}$$

and

$$\phi(t) < \phi(t/2) \tag{15}$$

Actually, we often replace (8) with the weaker condition $\phi(t) < \phi(0)$, which is less work to check and usually leds to the same robustness.

[4] It is possible, but very unlikely, to arrive at a stationary point that is not a local minimum. For example, if $V(x) = x^3$, we might approach $x = 0$ through negative but increasing $x$ values.

To find a $t$ suitable $t$, start with initial guess $t = 1$, corresponding to the unsafeguarded Newton step. Evaluate $\phi(t)$. If $\phi(t) > \phi(0)$, replace $t$ by $t/2$ and repeat. If $\phi(t) < \phi(0)$, evaluate $\phi(2t)$. If $\phi(2t) < \phi(t)$, replace $t$ by $2t$ and repeat. If $\phi(t) < \phi(2t)$, either stop or evaluate $\phi(t/2)$, depending on your level of paranoia. Continue these steps until either (14) or (15) are satisfied, or $t$ is too large. The first safeguard, insuring that $p$ is a descent direction, insures that $t$ will not be decreases forever. If we try to minimize an unbounded function such as $\phi(t) = -t$, then $t$ could be increased forever.

An important feature of both safeguards is that they all produce the unsafeguarded Newton step if $\bar{x}$ is close enough to a local minimum. This preserves the very desirable local quadratic convergence of Newton's method. The first safeguard is not at all expensive, in terms of computer work. The second, line search, requires us to evaluate $V$ several times. If optimization is expensive, it is usually because evaluating $V$ is expensive. In those cases, we want to use as few extra evaluation as possible. The minimal safeguard requires us to evaluate $V(\bar{x} + 2p)$ as well as $V(\bar{x} + p)$, which is twice as many evaluations as would be needed by the unsafeguarded method. In view of this, we might, for example, turn off the safeguards if we are pretty sure they are not needed. However, in my experience, safeguards are usually worth the wait.

# 3 Multidimensional problems

Much of the above applies immediately to multidimensional problems. If we want to solve $n$ equations for the $n$ unknowns $x_1, \ldots, x_n$. The linear approximation (3) is still correct. The only difference is that $y$, $f$, and $x$ are vectors and $f'$ is the $n \times n$ jacobian matrix of all first partials with matrix elements:

$$\left(f'(\bar{x})\right)_{jk} = \frac{\partial f_j(\bar{x})}{\partial x_k} \quad .$$

The unsafeguarded Newton step can still be computed using (2). The error estimate (5) still implies local quadratic convergence.

Newton's method is just one of many iterative methods for solving systems of equations or finding minima of function. Although Newton's method may lack robustness, particularly from a poor initial guess, it has another kind of robustness that accounts for part of its wide use. This is the fact that it is dimensionally correct. Each of the terms in the sum

$$p_j = \sum_k f'^{-1}_{jk} f_k$$

has the same units as $x_j$. This allows Newton's method to work well even when the individual equations have different units. A mathematician's way of saying the same thing is that Newton's method is *affine invariant*. That means that if $g(x) = Af(Bx)$ for some invertible matrices, $A$ and $B$, then Newton's method will work precisely as well on $g$ as it does on $f$. We can view this positively or negatively. We can think that, effectively, the optimal scaling has already been applied, or that no scaling will help.

## 3.1 Multidimensional Optimization

Newton's method also for minimization also goes over naturally to multidimensional problems. The Jacobian matrix $f'$ is replaced by the *Hessian* matrix $V''$ , whose entries are given by

$$\left(V''\right)_{jk} = \frac{\partial^2 V(\bar{x})}{\partial x_j \partial x_k} \quad .$$

Optimization problems have the feature that the Newton equations involve a symmetric matrix. Furthermore, near a nondegenerate local minimum, the Hessian is positive definite. Therefore, we should get the search direction, $p$, by solving the Newton equations:

$$V(\bar{x})'' p = - \bigtriangledown V(\bar{x}) \quad ,$$

using the Choleski decomposition of the hessian matrix, $V''$. This is the unsafeguarded method that has local quadratic convergence to a local nondegenerate minimum.

The as with one dimensional minimization, there are two safeguards that guarantee that the algorithm will find a local minimum[5], or die trying (i.e. have iterates going to infinity with decreasing function values). The two safeguards are finding a search direction that is a descent direction, and doing a bisection line search to make sure that $t$ is within a factor of 2 of a minimizing step size.

In one dimension, the Newton step fails to be a descent direction if the hessian is negative. The hessian (which is a $1 \times 1$ matrix in one dimension) will be positive near a nondegenerate local minimum[6], but it may not be positive far away. In more than one dimension, the hessian matrix will be positive definite near a nondegenerate local minimum but may not be positive definite far away. We want the hessian to be positive definite because this guarantees that the search direction is a descent direction. Suppose we compute a search direction, $p$ be solving a linear system of equations

$$Ap = - \bigtriangledown V(\bar{x}) \ . \tag{16}$$

Then $p$ will be a descent direction if $A$ is symmetric and positive devinite. To see this, note that if $A$ is symmetric, then $A^{-1}$ is positive definite if and only if $A$ itself is positive definite. Now calculate the derivative in (11). This is

$$\frac{d}{dt} V(\bar{x} + pt) = \bigtriangledown(V(\bar{x})^t \cdot p = - \bigtriangledown V(\bar{x})^t \cdot A^{-1} \cdot \bigtriangledown V(\bar{x}) \ . \tag{17}$$

The quantity on the right must be negative if $\bigtriangledown V \neq 0$ and $A$ is positive definite.

The safeguard based on this observation is to compute the step using (16) where the matrix $A$ is $V''(\bar{x})$ if $V''(\bar{x})$ is positive definite. Otherwise, we use a positive definite matrix related to $V''(\bar{x})$ rather than $V''(\bar{x})$ itself. There are several ways to choose $A$. I recommend a simple heuristic called the *modified Choleski* method. This method starts from the assumption that one would solve the equations (16) using the Choleski decomposition of $A$. We start with the hypothesis that $A = V''(\bar{x})$ and begin to Choleski decompose $V''(\bar{x})$. If $V''(\bar{x})$ is positive definite, we will find a lower triangular matrix, $L$ with $LL^t = V''(\bar{x})$. If $V''(\bar{x})$ is not positive definite, we will produce an $L$ that does not satisfy $LL^t = V''(\bar{x})$. We will take $A = LL^t$. This guarantees that $A$ will be positive definite. It also makes the equations (16) easy to solve.

You might think this is a very complicated way to find a positive definite matrix. After all, the identity matrix is also positive definite. If we take $A = I$ the equations (16) are particularly easy to solve. Taking $p = -\bigtriangledown V(\bar{x})$ is called the *gradient* method, or the *steepest descent* method. This usually works poorly in high dimensional problems met in the real world. I believe this is because it does not make sense dimensionally. The components of $\bigtriangledown V$ do not have the units of the corresponging components of $x$. Actually, the modified Choleski method is not that big a change of the program we would write for unsafeguarded Newton's method. In the unsafeguarded computation we would take the Choleski decomposition of $V''(\bar{x})$. In the safeguarded algorithm we would want to determine whether $V''(\bar{x})$ is positive definite. The simplest way to do this is to start computing the Choleski decomposition. If we succeed in finding a Choleski facor, then $V''(\bar{x})$ was positive definite. Otherwise it was not.

Finally, the algorithm. If we are looking for the Choleski factorization of a symmetric matrix, $B$, $LL^t = B$, at some point we have compute the diagonal entry, $l_{kk}$, of $L$ be solving

$$b_{kk} = l_{k1}^2 + \cdots + l_{kk}^2 \ ,$$

which leads to

$$l_{kk} = \left(b_{kk} - l_{k1}^2 + \cdots + l_{k,k-1}^2\right)^{1/2} \ . \tag{18}$$

If the quantities in the square roots are all positive, then we can take the Choleski factorization of $B$. If any one of them is negative, then $B$ is not positive definite. The modified Choleski algorithm replaces (18) with the modified formula

$$l_{kk} = \left|b_{kk} - l_{k1}^2 + \cdots + l_{k,k-1}^2\right|^{1/2} \ . \tag{19}$$

This leads to the Choleski factor of $B$, if there is one. Otherwise, it produces a matrix related to $B$ that is positive definite. Using (19) instead of (18) is one multidimensional generalization of using (12) instead

[5] As in the one dimensional case, there is a very small chance of finding a stationary point that is not a local minimum. Such points are called *saddle points* in multidimensional problems.

[6] This is the definition of the term *nondegenerate* in this context.

of (10) in one dimension. In fact, it reduces to this in the one dimensional case. You might have other suggestions of how to modify the Choleski decomposition when $V''(\bar{x})$ is not positive definite. This one has the desirable property that $L$ has the same units, so that $A$ has the same units as $V''(\bar{x})$.

The other safeguard is a line search exactly like the one discussed in the one dimensional context.

With all this, the overall minimization algorithm is as follows. Start with an initial guess, $x_0$. After some number of iterations, we have a *current iterate*, $x_k$. We want to compute a next iterate, $x_{k+1}$ from this. For this purpose, we call $x_k$ $\bar{x}$. The new iterate $x_{k+1}$ will be the $x'$ discussed now. First, compute $\bigtriangledown V(\bar{x})$ and $V''(\bar{x})$. Compute the possibly modified Choleski decomposition of $V''(\bar{x})$. Use this to solve the equations (16). Do a line search for a good value of $t$ in $V(\bar{x} + tp)$. Then $x' = \bar{x} + tp$ is the new iterate. This process continues until we are close enough to the solution.