Quasi Newton Methods

The quadratic order of convergence of Newton's method is special—none of the other iterative methods we will discuss converge so quickly. However, things can go wrong. Recall that Newton's method corresponds to the fixed point iteration given by

$$\Phi(x) = x - \frac{f(x)}{f'(x)}.$$

Sometimes there is no way to easily compute $f'(x)$. Other times, a bad initial guess for x_0 might lead to a divergent sequence, rather than the solution.

We first discuss a way to modify Newton's method to guarantee convergence. The bisection method for solving $f(x) = 0$ employs the intermediate value theorem to guarantee convergence by bracketing the solution in an ever shrinking interval. Consider the strategy of using Newton's method as long as the iterations stay within the current bracket, and using a bisection step otherwise. The Matlab code is given in Example 9a.

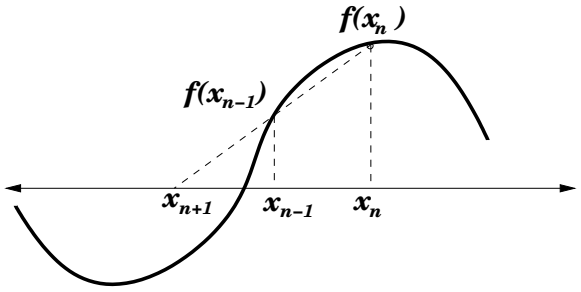
Example 9a

```
function x=hybrid9a(f,phi,a,b,erel)
    fa=feval(f,a);
    fb=feval(f,b);
    if fa*fb>=0 || b<=a
        disp('Not a good initial bracket!');
    end
    c=(a+b)/2;
    for n=1:3000
        fc=feval(f,c);
        if fc==0.0
            x=c;
            return
        end
        if fc*fa>0
            a=c;
        else
            b=c;
        end
        x=c;
        c=feval(phi,x);
        if c<a || c>b
            c=(a+b)/2;
        end
        if abs(c-x)<=erel*c
            break
        end
    end
    x=c;
```

Thus, given a bracket $[a, b]$ such that $f(a)$ and $f(b)$ have opposite signs, let $c = (a + b)/2$. If $\Phi(c) \in [a, b]$ then continue using Newton's method subject to the constraint that the next iteration lies in the bracket $[a, \Phi(c)]$ or $[\Phi(c), b]$ depending on the sign of $f \circ \Phi(c)$; otherwise, take a bisection step. Such a strategy takes bisection steps as necessary until the convergence criterion for Newton's method is met and then finishes with Newton's method. In particular, this hybrid algorithm is guaranteed to converge quadratically.

We now discuss what to do when $f'(x)$ is difficult or impossible to compute. A classical way of dealing with this is the secant method where x_{n+1} is set to the root of the secant line passing through $(x_n, f(x_n))$ and $(x_{n-1}, f(x_{n-1}))$. Graphically we have

Secant Method



The secant method has order of convergence equal to $(1 + \sqrt{5})/2 \approx 1.6180$. Not quadratic as Newton's method, but still quickly converging. Matlab code to perform the secant method is illustrated in Example 9b.

Example 9b

```
1  function x=secant9b(f,x1,x2,erel)
2      f2=feval(f,x1);
3      for n=1:3000
4          f1=f2;
5          f2=feval(f,x2);
6          if abs(f2-f1)<=erel*f2
7              break
8          end
9          x3=x2-f2*(x2-x1)/(f2-f1);
10         x1=x2;
11         x2=x3;
12     end
13     x=x2;
```

Let us return to a discussion of Newton's method. If f' is difficult to compute, then we can consider approximating f' in Newton's method by the constant s. This yields the constant-slope quasi Newton's method given by

$$\Phi(x) = x - \frac{f(x)}{s}.$$

The convergence criterion becomes

$$|\Phi'(x)| = \left|1 - \frac{f'(x)}{s}\right| < 1.$$

Solving this inequality when $x = \alpha$ we find that convergence is ensured when s has the same sign as $f'(\alpha)$ and $|s| > |f'(\alpha)|/2$. Moreover, if by a miracle $s = f'(\alpha)$, then arguments similar to those we used for Newton's method yield quadratic convergence. Obviously the better s approximates $f'(\alpha)$, the faster the iterations will converge.

It is difficult to make a good guess for $f'(\alpha)$ without knowing α in the first place. Therefore, the slope for s is usually updated during a computation as better approximations become available. One way of doing this is by mixing true Newton steps with constant-slope quasi Newton steps. For example, if f' takes 100 times longer to compute than f, one might take a single Newton step followed by 99 constant-slope quasi Newton steps and then repeat. After each Newton step update s to the most recent value of f' computed.

Another quasi Newton method is attractive for solving systems of equations. Consider the simultaneous solution (x, y) to the system

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0. \end{cases}$$

Those who recall the notational tricks from linear algebra will realize that we can rewrite this system as the single vector equation $F(v) = 0$ by setting

$$v = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{and} \qquad F(v) = \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix}.$$

Then

$$F' = \begin{bmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \\ \dfrac{\partial g}{\partial x} & \dfrac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}$$

and Newton's method becomes

$$\Phi(v) = v - [F'(v)]^{-1} F(v).$$

Newton's method applied to a system of equations provides a concrete example where F' is difficult to compute. Given a system of n linear equations with n unknowns, Newton's method entails the computation of n^2 partial derivatives and then the inversion of an $n \times n$ matrix. Until we have studied numerical linear algebra, we'd best find a simpler method.

A simpler method may be obtained by treating f as a function of x only with y fixed and similarly g as a function of y only with x fixed. Thus, we define Newton iterations for each equation separately focusing on only one variable at a time. In particular, given an initial guess (x_0, y_0) we iterate as

$$x_{n+1} = x_n - \frac{f(x_n, y_n)}{f_x(x_n, y_n)} \qquad \text{and} \qquad y_{n+1} = y_n - \frac{g(x_{n+1}, y_n)}{g_y(x_{n+1}, y_n)}.$$

The above scheme can be generalized to systems with arbitrarily many equations. Note that by working with each equation separately we avoid inverting F'. Hence, this method is computationally much simpler. As an aside, if $F(v) = Av - b$ where A is an $n \times n$ matrix, then this method reduces to the Gauss-Seidel method for solving $Av = b$.

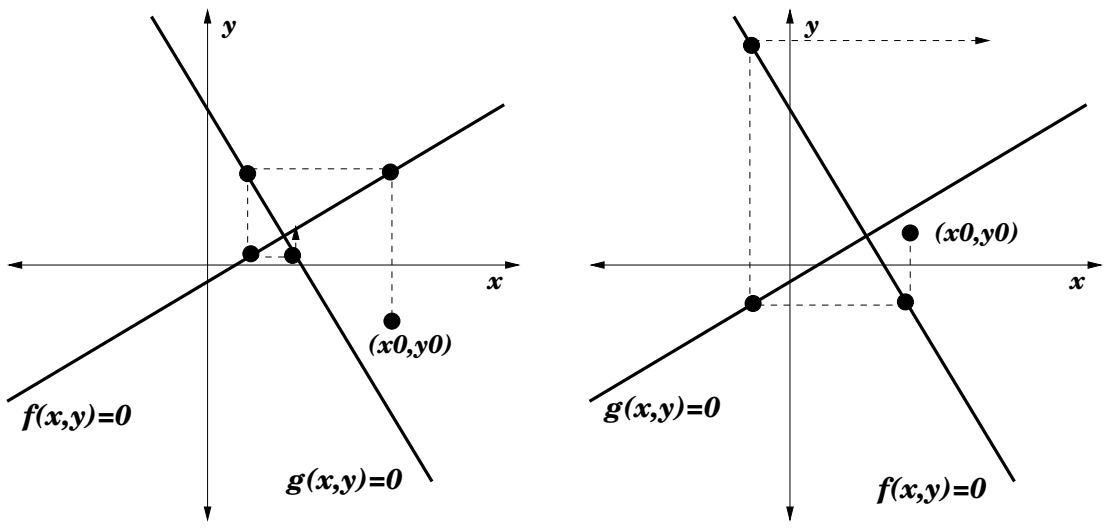
Since this simple iterative scheme for solving systems is no longer a true Newton's method, we can no longer expect quadratic convergence. We no longer use f_y or g_x in our computations, so in some sense, half the terms in $[F'(v)]^{-1}$ have been neglected. As a result, the step sizes tend to be smaller than they should be. To compensate for this we introduce a relaxation parameter ω with $0 < \omega < 2$ to obtain

$$x_{n+1} = x_n - \omega \frac{f(x_n, y_n)}{f_x(x_n, y_n)} \qquad \text{and} \qquad y_{n+1} = y_n - \omega \frac{g(x_{n+1}, y_n)}{g_y(x_{n+1}, y_n)}.$$

To speed convergence we are primarily concerned with making the steps bigger. Often values of ω between 1.3 and 1.8 can cut computational time by a factor of 10 or more. Again, it is straightforward to generalize this strategy to systems with arbitrarily many equations. As in Greenspan and Cassuli Chapter 1.5 we shall call this the generalized Newton's method for solving systems of equations. When the generalized Newton's method is used for solving the linear equation $Av = b$ it is called successive over relaxation.

The generalized Newton's method is no longer guaranteed to converge—even in a very small neighborhood of the solution. Care must be made in choosing which function to call f and which to call g. To illustrate how important the ordering of f and g is, consider the case where both f and g are linear functions. Then a true Newton's method would find the solution in one step, whereas our generalized Newton's method will converge only when the sum of the angles between f and the x axis and between g and the y axis total less than 180 degrees. This is illustrated as

Choosing f and g in the General Newton's Method.



With the correct labeling of f and g the generalized Newton's method converges. With the other choice it diverges. When we are solving n simultaneous equations some orderings are better than others. A change of variables may also help speed convergence. This is called preconditioning. Alternatively, one could make additional $\omega/2$ half-steps involving the derivatives f_y and g_x just before each of the given steps.