LECTURE NOTES

ECO 613/614

FALL 2007

KAREN A. KOPECKY

Root-Finding Methods

Often we are interested in finding $\mathbf{x}$ such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0},$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ denotes a system of n nonlinear equations and $\mathbf{x}$ is the n-dimensional root. Methods used to solve problems of this form are called *root-finding* or *zero-finding* methods. It is worthwhile to note that the problem of finding a root is equivalent to the problem of finding a fixed-point. To see this consider the fixed-point problem of finding the n-dimensional vector $\mathbf{x}$ such that

$$\mathbf{x} = \mathbf{g}(\mathbf{x})$$

where $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$. Note that we can easily rewrite this fixed-point problem as a root-finding problem by setting $\mathbf{f}(\mathbf{x}) = \mathbf{x} - \mathbf{g}(\mathbf{x})$ and likewise we can recast the root-finding problem into a fixed-point problem by setting $\mathbf{g}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \mathbf{x}$.

Often it will not be possible to solve such nonlinear equation root-finding problems analytically. When this occurs we turn to numerical methods to approximate the solution. The methods employed are usually *iterative*. Generally speaking, algorithms for solving problems numerically can be divided into two main groups: *direct methods* and *iterative methods*. Direct methods are those which can be completed in a predetermined finite number of steps. Iterative methods are methods which converge to the solution over time. These algorithms run until some convergence criterion is met. When choosing which method to use one important consideration is how quickly the algorithm converges to the solution or the method's *convergence rate*.

0.1 Convergence Rates

Distinction is made between two different types of convergence: *quotient-convergence* (q-convergence) and *root-convergence* (r-convergence). Q-convergence, named so because it is defined in terms of the ratio of successive error terms, describes convergence in which the overall error decreases with each iteration. A weaker form of convergence, r-convergence, is used to characterize the convergence rates of algorithms when convergence is not monotonic.

Let us first consider q-convergence. Suppose $\mathbf{x}^*$ is the true solution and $\mathbf{x^{(n)}}$ is the approximate solution at iteration n. Then the error at iteration n is $||\mathbf{x}^* - \mathbf{x^{(n)}}||$. A method is said to q-converge with rate k if

$$\lim_{n\to\infty} \frac{||\mathbf{x}^* - \mathbf{x^{(n+1)}}||}{||\mathbf{x}^* - \mathbf{x^{(n)}}||^k} = C,$$

where C is finite and positive.

- If $k = 1$ and $C < 1$, the convergence rate is *linear*.
- If $1 < k < 2$, the convergence rate is *superlinear*.
- If $k = 2$, the convergence rate is *quadratic*.

In general, we say that the q-order of convergence is k.

Once the approximate solution is close to the true solution, a q-linearly convergent algorithm gains a constant number of digits of accuracy per iteration, whereas a q-superlinearly convergent algorithm gains an increasing number of digits of accuracy with each iteration. So a q-superlinearly convergent algorithm has k times as many digits of accuracy after each iteration and a q-quadratically convergent algorithm has twice as many.

We say that an algorithm is r-convergent of order k if there exists a sequence, $\{v_n\}$, that converges at q-order k to C and the sequence of the algorithm's successive error terms is dominated by $\{v_n\}$, i.e., if $||\mathbf{x}^* - \mathbf{x_n}|| \leq v_n$ for all n.

1 Bisection Method

The bisection method is the simplest and most robust algorithm for finding the root of a one-dimensional continuous function on a closed interval. The basic idea is a follows.

Suppose that $f(\cdot)$ is a continuous function defined over an interval $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs. Then by the intermediate value theorem, there exists at least one $r \in [a, b]$ such that $f(r) = 0$. The method is iterative and each iteration starts by breaking the current interval bracketing the root(s) into two subintervals of equal length. One of the two subintervals must have endpoints of different signs. This subinterval becomes the new interval and the next iteration begins. Thus we can define smaller and smaller intervals such that each interval contains r by looking at subintervals of the current interval and choosing the one in which $f(\cdot)$ changes signs. This process continues until the width of the interval containing a root shrinks below some predetermined error tolerance.

Without loss of generality, suppose one has a and b such that $f(a)f(b) < 0$ and $f(a) < 0$ and δ is some predetermined convergence tolerance. Then the steps are as follows:

Step 1. Set $n = 1$, $a^{(n)} = a$, and $b^{(n)} = b$.

Step 2. Compute $c^{(n)} = \frac{1}{2}(a^{(n)} + b^{(n)})$

Step 3. If $f(c^{(n)}) > 0$ let $b^{(n+1)} = c^{(n)}$ and $a^{(n+1)} = a^{(n)}$, otherwise let $a^{(n+1)} = c^{(n)}$ and $b^{(n+1)} = b^{(n)}$.

Step 4. If $|b^{(n)} - a^{(n)}| \leq \delta$ then stop and call $c^{(n)}$ a root, otherwise $n = n + 1$ and go step 2.

There are two major benefits of the bisection method. First one is that it is very robust. It is guaranteed to find an approximation to the root within a given degree of accuracy in a known number of iterations. To see this note that, at the initial step, the solution lies in an interval of size $|b - a|$. At the second step, if the solution is not found, it lies in an interval of size $|b - a|/2$. At the nth iteration the solution must be in an interval of size

$$|b^{(n)} - a^{(n)}| = \frac{|b - a|}{2^{n-1}}.$$

Hence if our error tolerance is δ, we can guarantee to be within that tolerance in

$$n = \log_2 \left(\frac{b - a}{\delta} \right) + 1$$

iterations.

The second major benefit is that the method does not rely on the derivatives of the function. Thus it can be used to find roots of non-smooth functions. While a benefit in some cases, the fact that the method doesn't use the derivatives of the function means it is

not exploiting any of information about the curvature of the function to help it locate the root. This is what makes bisection method slow relative to other methods that exploit the functions curvature.

How slow is the bisection method? To compute the rate of convergence note that

$$|c_n - r| \leq \frac{|b_n - a_n|}{2} = \frac{|b-a|}{2^n}$$

Thus the error converges r-linearly to zero since $\{|c_n - r|\}$ is dominated by $\{|b-a|/2^n\}$ and

$$\frac{\frac{|b-a|}{2^{n+1}}}{\frac{|b-a|}{2^n}} = \frac{1}{2}.$$

So the rate at which successive significant digits are found is linear in computational effort.

A weakness of bisection method is that it cannot be generalized to the general n dimensional case. One additional weakness of bisection method, a weakness that many methods suffer from, is that it cannot find the root of a function unless the function value takes opposite signs on either side of that root. Roots of this form, are more difficult to find in general.

What is a reasonable convergence criteria? While your function might analytically pass through zero, it is possible that numerically it is never zero for any floating-point number. Convergence to within 10^{-12} is reasonable when your root lies near one but not reasonable when your root lies near 10^{26}. So we want an error criteria which is relative to the value of the root. This can become problematic though when the root is very close to 0. A common tolerance is to set

$$\delta = \frac{|a| + |b|}{2} u,$$

where u is machine epsilon and limit the number of iterations so that the absolute error is no more than say 10^{-12} or the maximum number of iterations is approximately 40.

While bisection method is slow, it is stable and unlike other methods does not require a good initial guess in order to converge. Often it is combined with other faster methods, that require good initial guesses, to tradeoff stability and speed. Bisection is used initially to develop a good bound on the root. Then that bound is used to generate a good initial guess that is then plugged into another method which only requires a few iterations to quickly increase the accuracy of the solution.

2 Function Iteration

A relatively simple technique for finding a root is *function iteration*. The method consists in recasting the root-finding problem as the fixed-point problem and then iterating on the fixed-point problem. For $\mathrm{g} : \mathbb{R}^n \to \mathbb{R}^n$ and $\mathrm{x} = \mathrm{g}(\mathrm{x})$ The algorithm is simply:

Step 1. Set $n = 1$. Create initial guess $\mathbf{x^{(n)}}$.

Step 2. Compute $\mathbf{x^{(n+1)}} = \mathrm{g}(\mathbf{x^{(n)}})$.

Step 3. If $||\mathbf{x^{(n+1)}} - \mathbf{x^{(n)}}|| < \delta$ stop, otherwise $n = n + 1$ go to step 2.

In theory, the algorithm will converge to the fixed-point of g if g is continuous and differentiable and if the initial value of x is sufficiently close to the fixed-point of g at which $||\nabla \mathrm{g}(\mathrm{x}^*)|| < 1$. While these conditions are sufficient for convergence they are not, however, necessary and often times the algorithm will converge even those these conditions are not met.

To compute the rate of convergence of the method note that

$$||\mathbf{x^{(n+1)}} - \mathbf{x}|| \leq ||\nabla \mathrm{g}(\mathrm{x}^*)||||\mathbf{x^{(n)}} - \mathbf{x}||$$

Hence the method converges q-linearly with C equal to $||\nabla \mathrm{g}(\mathrm{x}^*)||$.

Given that it is easy to implement, this method may be worth trying before switching to more robust but also more complex methods such as Newton's.

3 Newton's Method

Newton's method is popular and many methods are some variation of it. It is based on the principle of *successive linearization*, a technique in which the harder nonlinear problem is replaced by a succession of linear problems whose solutions converge to the solution of the nonlinear problem.

First to gain intuition, let's consider the one-dimensional case. Given an initial guess of the root $x^{(1)}$, the function is approximated by its first-order Taylor expansion about $x^{(1)}$, which is graphically represented by the line tangent to f at $x^{(1)}$. The updated approximation to the root is the root of the tangent line. This is set at $x^{(2)}$ and the iteration is repeated with $x^{(3)}$ being the root of the tangent line to f at $x^{(2)}$. This continues until the roots have

converged.

In the multivariate case, the method is to choose successive estimates of the root by approximating the function f by a first-order Taylor expansion about the current estimate. So given the current estimate, $\mathbf{x^{(n)}}$, the updated estimate, $\mathbf{x^{(n+1)}}$ is given by solving the linear system of equations

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}\left(\mathbf{x^{(n)}}\right) + \nabla \mathbf{f}\left(\mathbf{x^{(n)}}\right)\left(\mathbf{x} - \mathbf{x^{(n)}}\right) = 0$$

for x. Hence

$$\mathbf{x^{(n+1)}} = \mathbf{x^{(n)}} - \left[\nabla \mathbf{f}\left(\mathbf{x^{(n)}}\right)\right]^{-1} \mathbf{f}\left(\mathbf{x^{(n)}}\right), \tag{1}$$

although, of course, in the n-dimensional case this problem should never be solved by inverting the Jacobian and multiplying it with $\mathbf{f}(\mathbf{x^{(n)}})$.

This suggests the following algorithm:

Step 1. Set $n = 1$. Create initial guess $\mathbf{x^{(n)}}$.

Step 2. Compute $\mathbf{x^{(n+1)}} = \mathbf{x^{(n)}} - \left[\nabla \mathbf{f}\left(\mathbf{x^{(n)}}\right)\right]^{-1} \mathbf{f}\left(\mathbf{x^{(n)}}\right)$ by solving the linear system.

Step 3. If $||\mathbf{x^{(n+1)}} - \mathbf{x^{(n)}}|| < \delta$ stop, otherwise $n = n + 1$ go to step 2.

Newton's algorithm usually displays very fast convergence properties, with high accuracy *close to the solution*. The reason for this is the reliance on the first order expansion of f. Far from a root, terms of order 2 and higher *are* of importance, and equation (1) is of no interest. Let's compute the convergence rate for the univariate case. Suppose the function has an actual root at x^* and that $\{x^{(n)}\}$ is a sequence of iterates that converges to x^* as n goes to infinity. (Note convergence can be proven under certain assumptions but this will not be done here.) Then the error at iteration n is

$$\epsilon^{(n)} = x^{(n)} - x^*.$$

It follows that

$$\begin{aligned}
\epsilon^{(n+1)} &= x^{(n+1)} - x^*, \\
&= x^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(n)}\right)} - x^*, \\
&= \epsilon^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(n)}\right)}.
\end{aligned}$$

Now consider the following approximations

$$\begin{aligned} f\left(x^{(n)}\right) &\approx f\left(x^*\right) + f'\left(x^*\right)\epsilon^{(n)} + \frac{1}{2}f''\left(x^*\right)\left(\epsilon^{(n)}\right)^2, \\ f'\left(x^{(n)}\right) &\approx f'\left(x^*\right). \end{aligned}$$

Combining the above equations yields

$$\left[f'\left(x^{(n)}\right)\right]^{-1} f\left(x^{(n)}\right) \simeq \epsilon^{(n)} + \frac{1}{2}\left[f'\left(x^*\right)\right]^{-1} f''(x^*)\left(\epsilon^{(n)}\right)^2.$$

Hence we have that

$$\epsilon^{(n+1)} \approx -\frac{1}{2}\left[f'\left(x^*\right)\right]^{-1} f''(x^*)\left(\epsilon^{(n)}\right)^2.$$

So

$$\frac{|\epsilon^{(n+1)}|}{|\epsilon^{(n)}|^2} \approx \left|\frac{1}{2}\left[f'\left(x^*\right)\right]^{-1} f''(x^*)\right|,$$

i.e, the rate of convergence of Newton's method is q-quadratic. As the estimate gets close to the root the number of significant digits in the solution double with each iteration.

The major benefit of Newton over bisection is the major improvement in the convergence rate. When Newton works it works incredibly fast. But this gain in efficiency does not come without cost. In order to use Newton's at all the function must be differentiable and, in addition, the derivative must be supplied. It is often the case that the function is differentiable but computing the derivative, i.e. the Jacobian matrix of partial derivatives, is a nightmare. The probability of human error becomes very high. It is possible to replace the analytical Jacobian with a finite difference approximation but this comes at a cost of a slower overall rate of convergence. In fact, for the one-dimensional case, secant method, which as we will see only requires one function evaluation at each iteration, always dominates Newton's method. Computing the numerical derivative will require at least one additional function evaluation per an iteration or at least two in total.

In addition, global convergence is likely to fail in many cases. If, for instance, the algorithm hits a local minima or extrema, where $\nabla \mathrm{f}\left(\mathbf{x}^{(\mathbf{n})}\right) = 0$, then it cannot compute the Newton step, and convergence fails. Numerically there can be a problem of ill-conditioning when $\nabla \mathrm{f}\left(\mathbf{x}^{(\mathbf{n})}\right)$ is close to 0. Sometimes these ill-conditioning problems can be solved by rescaling the problem but other times there is no solution but to switch to some other method that does not rely on the Jacobian. Another problem with Newton's method is that

there is no guarantee of convergence. The method may not always converge and/or it may not converge to the desired root when more than one root exists. The success of the method depends crucially on the initial guess.

4 Quasi-Newton Methods

One problem with Newton's method is that it requires to evaluate both the function and its derivative at each step. When the function f is difficult to evaluate, this can be costly. Therefore methods that do not require the evaluation of a derivative are sometimes a better choice. Quasi-Newton methods are methods that are similar to Newton in that they consist of iterations over successive linear-equation solving problems but different in that they don't require an analytical Jacobian. The benefits of not having to deal with the Jacobian do not come for free, they are paid in efficiency–quasi-Newton methods usually converge at a slower rate than Newton's method.

4.1 Secant Method

The secant method is the most popular quasi-Newton method for the univariate case. It is identical to Newton except that it approximates the derivative of f at each iteration n using the root estimates and their function values at the two previous iterations. The derivative of f at point $x^{(n)}$ is approximated by

$$f'\left(x^{(n)}\right) \simeq \frac{f\left(x^{(n)}\right) - f\left(x^{(n-1)}\right)}{x^{(n)} - x^{(n-1)}}.$$

Then the new Newton's step, now the secant step, becomes

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)\left(x^{(n)} - x^{(n-1)}\right)}{f\left(x^{(n)}\right) - f\left(x^{(n-1)}\right)}.$$

The method requires two initial guesses.

It can be shown that the rate of convergence of secant method is superlinear with $k \approx 1.62$. Hence it is faster than bisection but slower than Newton's. However, the convergence rate, which measures the number of iterations required to reach a certain level of accuracy, is not all that matters when evaluating the computational efficiency of the algorithm. The

number of floating-point operations, or flops, per an iteration should also be taken into account. If each iteration requires many flops, even though an algorithm has a faster rate of convergence it may take longer to reach a desired level of accuracy. For this reason it is often the case that secant method is faster than Newton's method. Secant Method has the advantage of only requiring one function evaluation for each iteration. This can more than compensate for the slower convergence rate when the function and its derivative are costly to evaluate. On the down side, like Newton's Method, secant method is not very robust, especially when the initial guesses are far from root.

4.2 Broyden's Method

For the multivariate case, the most popular quasi-Newton method is Broyden's method, a generalization of the univariate secant method to the multivariate case. The method generates a sequence of vectors $\mathbf{x^{(n)}}$ and matrices $\mathbf{A^{(n)}}$ that approximate the root of $\mathbf{f}$ and the Jacobian $\nabla \mathbf{f}$ evaluated at the root, respectively. It needs an initial guess of the root, $\mathbf{x^{(0)}}$ and the Jacobian at the root, $\mathbf{A^{(0)}}$. Often a numerical approximation to the Jacobian at $\mathbf{x^{(0)}}$ is used for $\mathbf{A^{(0)}}$. Alternatively, an identity matrix, appropriately scaled, can be used but will usually cause the algorithm to take longer to converge.

Given $\mathbf{x^{(n)}}$ and $\mathbf{A^{(n)}}$, the updated root approximation is found by solving the linear problem obtained by replacing $\mathbf{f}$ with its first-order Taylor expansion about $\mathbf{x^{(n)}}$,

$$\mathbf{f(x)} \approx \mathbf{f}\left(\mathbf{x^{(n)}}\right) + \mathbf{A^{(n)}}\left(\mathbf{x} - \mathbf{x^{(n)}}\right) = \mathbf{0},$$

and finding the root. This yields a rule for updating the estimate of the root of

$$\mathbf{x^{(n+1)}} = \mathbf{x^{(n)}} - \left[\mathbf{A^{(n)}}\right]^{-1} \mathbf{f}\left(\mathbf{x^{(n)}}\right).$$

The next step is to update the Jacobian approximate, $\mathbf{A^{(n)}}$. This is done by making the smallest possible change to $\mathbf{A^{(n)}}$ as measured in the Frobenius matrix norm,

$$||B||_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}|B_{ij}|^2\right)^{1/2},$$

but enforcing that the *secant condition*,

$$\mathbf{f}\left(\mathbf{x^{(n+1)}}\right) - \mathbf{f}\left(\mathbf{x^{(n)}}\right) = \mathbf{A}\left(\mathbf{x^{(n+1)}} - \mathbf{x^{(n)}}\right), \tag{2}$$

holds, i.e., by setting

$$\mathbf{A}^{(\mathbf{n+1})} = \arg\min_{\mathbf{A}} ||\mathbf{A} - \mathbf{A}^{(\mathbf{n})}||_F$$

subject to (2). This gives the rule for updating the approximation to the Jacobian

$$\mathbf{A}^{(\mathbf{n+1})} = \mathbf{A}^{(\mathbf{n})} + \left[\mathbf{f}\left(\mathbf{x}^{(\mathbf{n+1})}\right) - \mathbf{f}\left(\mathbf{x}^{(\mathbf{n})}\right) - \mathbf{A}^{(\mathbf{n})}\mathbf{d}^{(\mathbf{n})}\right] \frac{\left(\mathbf{d}^{(\mathbf{n})}\right)'}{\left(\mathbf{d}^{(\mathbf{n})}\right)' \mathbf{d}^{(\mathbf{n})}},$$

where $\mathbf{d}^{(\mathbf{n})} = \mathbf{x}^{(\mathbf{n+1})} - \mathbf{x}^{(\mathbf{n})}$.

Broyden's method can be accelerated by avoiding solving the linear system at each iteration. To do this the inverse of the Jacobian rather than the Jacobian itself is stored and updated at each iteration. Broyden's methods with inverse update generates a sequence of vectors $\mathbf{x}^{(\mathbf{n})}$ and $\mathbf{B}^{(\mathbf{n})}$ that approximate the root and the inverse Jacobian at the root, respectively. The rule for updating the estimate of the root is

$$\mathbf{x}^{(\mathbf{n+1})} = \mathbf{x}^{(\mathbf{n})} - \mathbf{B}^{(\mathbf{n})}\mathbf{f}\left(\mathbf{x}^{(\mathbf{n})}\right)$$

and the inverse Jacobian update rule is

$$\mathbf{B}^{(\mathbf{n+1})} = \mathbf{B}^{(\mathbf{n})} + \frac{\left(\mathbf{d}^{(\mathbf{n})} - \mathbf{u}^{(\mathbf{n})}\right)\mathbf{d}^{(\mathbf{n})}\mathbf{B}^{(\mathbf{n})}}{\left(\mathbf{d}^{(\mathbf{n})}\right)'\mathbf{u}^{(\mathbf{n})}}$$

where $\mathbf{u}^{(\mathbf{n})} = \mathbf{B}^{(\mathbf{n})}\left[\mathbf{f}\left(\mathbf{x}^{(\mathbf{n+1})}\right) - \mathbf{f}\left(\mathbf{x}^{(\mathbf{n})}\right)\right]$. Most often Broyden's method with inverse update is used instead of Broyden directly since the inverse method way is faster.

The method is similar to Newton's method in that it will converge in cases where f is continuous and differentiable, the initial guess is "sufficiently" close to the root, the Jacobian at the root is invertible and not ill-conditioned, and if $\mathbf{A}^{(\mathbf{0})}$ is "'sufficiently" close to the Jacobian at the root or $\mathbf{B}^{(\mathbf{0})}$ is "sufficiently" close to the inverse of the Jacobian at the root. And just like Newton's method there is no general formula for determining what "sufficiently" close actually means. It is also worthwhile to note that, convergence of $\mathbf{x}^{(\mathbf{n})}$ to a root does not guarantee convergence of either $\mathbf{A}^{(\mathbf{n})}$ to the Jacobian at the root or $\mathbf{B}^{(\mathbf{n})}$ to the inverse Jacobian at the root. It is not necessary that this convergence occurs to find the root and and not typical either.

The rate of convergence of Broyden's method is similar to that of secant method, it converges at a superlinear rate with $k \approx 1.62$. Hence it is slower than Newton's in number of iterations required to obtain a given level of accuracy but this doesn't mean that it is slower

than Newton when the number of flops involved for each iteration are taken into account. Broyden's method with inverse update requires a function evaluation and a matrix-vector multiplication. Newton's method requires a function evaluation, a derivative evaluation or approximation, and the solution of a linear equation. Which of the two is more efficient will depend on the dimension of x and the the difficulty involved in obtaining the derivative or approximation to the derivative at each iteration. If the derivative is costly to evaluate or approximate, Broyden's method is usually more efficient but when derivatives are easy to evaluate and the dimension of x is fairly small, Newton's method may be better.

5 Some Comments

A common error that occurs when using gradient-based root-finding methods, is human error in entering in the function and/or the analytic derivative. One way to check for such errors is to compare the analytic derivative to a numerical approximation. This is not that difficult to do and can save time debugging time in complex problems.

Another common problem that occurs when using Newton and quasi-Newton methods to find roots is that poor initial guesses are given. While there is some art to finding a good initial guess, one technique that can help to reduce the damage incurred by poor initial guesses called "backstepping". If the full Newton step say dx does not offer an improvement over the current iterate x, then one "backsteps" toward the current iterate x by repeatedly cutting dx in half until $\mathbf{x} + \mathbf{dx}$ does offer an improvement. Here improvement is measured by the Euclidean norm $||\mathbf{f}(\mathbf{x})|| = \mathbf{f}(\mathbf{x})'\mathbf{f}(\mathbf{x})/\mathbf{2}$ Since $||\mathbf{f}(\mathbf{x})|| = \mathbf{0}$ at the root one can view an iterate as making an improvement if $||\mathbf{f}(\mathbf{x})|| > ||\mathbf{f}(\mathbf{x} + \mathbf{dx})||$. Backstepping can help Newton and quasi-Newton methods from taking huge steps in the wrong directions, improving the robust of the algorithms. To avoid getting stuck at a local minimum when backstepping you can allow the algorithm to backstep until either $||\mathbf{f}(\mathbf{x})|| > ||\mathbf{f}(\mathbf{x} + \mathbf{dx})||$ or $||\mathbf{f}(\mathbf{x} + \mathbf{dx}/\mathbf{2})|| > ||\mathbf{f}(\mathbf{x} + \mathbf{dx})||$.

6 Hybrid Methods

While bisection is sometimes paired with Newton's or secant method to provide stability until one has bracketed the root by a small enough interval that a good initial guess is much more likely, another approach it to use *Brent's method*.

6.1 Brent's Method

An excellent method that solves both the problems discussed above is Brent's method. Brent's method is a hybrid of bisection and *inverse quadratic interpolation* that can be used to find roots of univariate functions. Unlike bisection, at its best, Brent's method will result in superlinear convergence to the root. At its worse, it is bisection and converges linearly. Its value over other superlinear convergence methods is that Brent's method guarantees converge so long as a root exists within the initial interval and the function can be evaluated at every point in the interval.

To understand Brent's method we must first discuss inverse quadratic interpolation (IQI). IQI is similar to secant's method. Remember that the strategy for updating the root at a given iteration with secant method is to approximate f by a linear function and use the root of that linear function as the updated guess of the root of f. IQI uses a similar approach. At a given iteration, it approximates f by an inverse quadratic function and uses the root of the inverse quadratic as the updated estimate of the root of f. Why use an inverse quadratic instead of just a quadratic function? The quadratic function may not have real roots and even if it does they may not be easy to compute, and it may not be easy to choose which root to use as the next iterate. In contrast by approximating f with the an inverse quadratic function, say $\tilde{f}$ the updated estimate of the root is just $\tilde{f}(0)$.

To generate the approximate to f we do inverse interpolation, i.e., we approximate the inverse of f by a a quadratic. Given three values a, b and c and there function values $f(a)$, $f(b)$, and $f(c)$ we can compute this approximation by polynomial interpolation with Lagrange basis functions. Then

$$x = \frac{a[y - f(a)][y - f(b)]}{[f(a) - f(b)][f(a) - f(c)]} + \frac{b[y - f(a)][y - f(c)]}{[f(b) - f(a)][f(b) - f(c)]} + \frac{c[y - f(b)][y - f(c)]}{[f(c) - f(b)][f(b) - f(c)]}$$

and setting y to zero gives the updated estimate of the root,

$$x = b + P/Q$$

where $R \equiv f(b)/f(c)$, $S \equiv f(b)/f(a)$, $T = f(a)/f(c)$ and

$$P = S[T(R-T)(c-b) - (1-R)(b-a)]$$

and

$$Q = (T-1)(R-1)(S-1)$$

Note the b should always be the current best estimate of the root. Then P/Q is the difference between the current best estimate and the updated estimate.

The process is repeated with b replaced by the new approximation, a replaced by the old b and c replaced by the old a. Note that only one new function evaluation is needed per an iteration. The converge rate is superlinear with $r \approx 1.839$.

The inverse quadratic method, like Newton's runs the risk of sending the current estimate of the root far away. This occurs when Q gets close to zero. Brent's method maintains brackets on the root and checks where IQI wants to go next. When it wants to go outside the bounds Brent switches to bisection method. Also is IQI is converging to slowly, Brent's switches to bisection.

References

- **Miranda, Mario J. and Paul L. Fackler.** 2002. *Applied Computational Economics and Finance.* Cambridge, MA: MIT Press.
- **Nocedal, Jorge and Stephen J. wright.** 1999. *Numerical Optimization.* Springer-Verlag New York, Inc.
- **Press, William H.; Saul A. Teukolsky; William T. Vetterling; and Brian P. Flannery.** 1992. *Numerical Recipes in C.* New York, N.Y.: Press Syndicate of the University of Cambridge.