

Bisection Algorithm: A non-linear solver

- In solving dynamic models we will often encounter the task of solving non-linear equations. There are many methods to solve these types of problems, which usually take the form of zeros, and fixed points. If $f : R^n \rightarrow R^n$, then a zero of f is any x such that $f(x) = 0$, and any fixed point of f is any x such that $f(x) = x$. The existence of solutions to these problems are analyzed by the fixed point theorem theory.
- In one dimensional problems, or in problems such that the main task can be simplified to be one-dimensional, one of the simplest and most robust methods is bisection.
- In order to find the zero we can look in a very short interval $[a, b]$ in which a function f has a change of sign, since the corresponding continuous function must be zero somewhere within such an interval. An interval in which the sign of f differs at its endpoints is called a bracket.
- Suppose f is continuous and $f(a) < 0 < f(b)$ for some $a, b, a < b$. Under these conditions, the Intermediate Value Theorem assures us that there is some zero of f in (a, b) . Bisection uses this result iteratively to compute a zero.
- Consider then $c = 1/2(a + b)$, the midpoint of $[a, b]$. If $f(c) = 0$, we found a solution. If $f(c) < 0$ then the IVT says there is a zero of f in (c, b) . The bisection method continues then focusing on this last interval. If $f(c) > 0$, then there is zero in (a, c) , and the bisection continues with the latter interval. In either case the algorithm continues with a smaller interval. There could be multiple zeros, but we are after finding one of them, the properties of the function we are trying to find a zero of should tell us about uniqueness.
- While this method is simple it displays the important components of any nonlinear equation solver: Make an initial guess, compute iterates, and check if the last iterate is acceptable as a solution. The algorithm can be summarized as follows (Judd 1998):

Find a zero of $f(x)$, $f : R^1 \rightarrow R^1$.

Initialization: Initialize and bracket a zero: Find $x^L < x^R$ such that $f(x^L)f(x^R) < 0$, and choose stopping rule parameters $\epsilon > 0$, and $\delta > 0$.

Step 1: Compute midpoint. $x^M = (x^L + x^R)/2$.

Step 2: Refine the bounds. If $f(x^M)f(x^L) < 0$, $x^R = x^M$ and do not change x^L . Otherwise $x^L = x^M$ and leave x^R unchanged.

Step 3: Check Stopping rule: If $x^R - x^L \leq \epsilon(1 + |x^L| + |x^R|)$ or if $|f(x^M)| \leq \delta$, then stop and report solution at x^M . Otherwise go to Step 1.

Stopping Rules:

- Iterative schemes rarely find the true solution, and a lot of times continuing iterating has little value because the error in computing $f(x)$ will eventually dominate the differences between successive iterates. Therefore, a critical component of any non-linear solver is the stopping rule. The rule computes an estimate of the distance to a solution, and stops when that estimate is sufficiently small.

- In the summary of the algorithm we stopped whenever the bracketing interval is so small that we do not care about any further precision. This is controlled by ϵ , which does not have to be small. The choice of ϵ has to be reasonable considering the order of magnitude of the amounts of the problem and the machine precision. This is important because bisection makes no use of the magnitudes of the function values, only their signs. Notice that given a starting interval $[a, b]$ the length of the interval after k iterations is $\frac{(b-a)}{2^k}$, so that achieving an error tolerance of ϵ requires

$$\left\lceil \log_2 \left(\frac{b-a}{\epsilon} \right) \right\rceil \quad (1)$$

iterations, regardless of the particular function f involved.

We also stop when round-off error associated with computing f make it impossible to pin down the zero any further. We stop when $f(x^M)$ is less than the expected error in calculating f . This is controlled by δ .

Convergence:

- If any iterative method is to be useful, it must converge to the solution and do so at a reasonable speed. Convergence properties for bisection are quite obvious. Bisection will always converge to a solution once we have found an initial pair of points, x^L and x^R , that bracket a zero. Bisection is a relatively slow method, but in tests using Matlab, for example, is quite efficient. Notice that its rates of convergence is linear.

Finite Differences to Compute Derivatives

Derivatives are a measure of the sensitivity of the function to infinitesimal changes in the values of the variables. The approach here is to make small but finite perturbations in the values of x and examine the resulting differences in the function values. By taking ratios of the function difference to variable difference, we obtain approximations to the derivatives.

The most common way to compute numerical derivatives are finite difference methods. Specifically, if $f : R \rightarrow R$, then one way to approximate $f'(x)$ is the one-sided finite difference formula

$$f'(x) = \frac{f(x+h) - f(x)}{h} \quad (2)$$

where $h = \max(\epsilon x, \epsilon)$ is the step size and ϵ is chosen appropriately, usually around 10^{-6} . We want h to be small relative to x , but also stay away from 0 so the equation above is well-behaved.

More generally, if $f : R^n \rightarrow R$, then the one-sided formula for $\delta f / \delta x_i$ is

$$\delta f / \delta x_i = \frac{f(x_1, x_2, \dots, x_i + h_i, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{h_i} \quad (3)$$

where $h_i = \max(\epsilon x_i, \epsilon)$. The cost is basically computing one evaluation of the function.

Sometimes the one-sided difference will not be accurate enough for our purposes. Since in general we do not want to compute analytic derivatives, we seek better finite-difference formulas. A better approximation is the two-sided formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \quad (4)$$

which differs from $f'(x)$ by $(h^2/6)f'''(\xi)$, for some $\xi \in [x-h, x+h]$. The round-off error of the approximation is ϵ/h , and the resulting upper bound error is much lower. Effectively we reduce the error from order $\epsilon^{1/2}$ to order $\epsilon^{2/3}$.

Using a similar rationale we can compute the second derivative

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}, \quad (5)$$

which is second order accurate.

All these formulas come from the Taylor series expansions of a smooth function $f : R \rightarrow R$, for which we want to approximate its first and second derivative:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \dots \quad (6)$$

and

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{6}h^3 + \dots \quad (7)$$

It is easy to see that we obtain the formulas above by solving for $f'(x)$ in one or the other, and the two-sided difference by subtracting the second from the first. We obtain the second derivative by adding the two series. Notice that the remainder of the series is a very small function of h .