Lecture 38: Bracketing Algorithms for Root Finding

7. Solving Nonlinear Equations.

Given a function $f : \mathbb{R} \to \mathbb{R}$, we seek a point $x_* \in \mathbb{R}$ such that $f(x_*) = 0$. This x_* is called a *root* of the equation $f(x) = 0$, or simply a *zero* of f. At first, we only require that f be continuous a interval $[a, b]$ of the real line, $f \in C[a, b]$, and that this interval contains the root of interest. The function f could have many different roots; we will only look for one. In practice, f could be quite complicated (e.g., evaluation of a parameter-dependent integral or differential equation) that is expensive to evaluate (e.g., requiring minutes, hours, ...), so we seek algorithms that will produce a solution that is accurate to high precision while keeping evaluations of f to a minimum.

7.1. Bracketing Algorithms.

The first algorithms we study require the user to specify a finite interval $[a_0, b_0]$, called a *bracket*, such that $f(a_0)$ and $f(b_0)$ differ in sign, $f(a_0)f(b_0) < 0$. Since f is continuous, the intermediate value theorem guarantees that f has at least one root x_* in the bracket, $x_* \in (a_0, b_0)$.

7.1.1. Bisection.

The simplest technique for finding that root is the *bisection* algorithm:

For $k = 0, 1, 2, \ldots$

1. Compute $f(c_k)$ for $c_k = \frac{1}{2}(a_k + b_k)$.
2. If $f(c_k) = 0$, exit; otherwise, repeat with $[a_{k+1}, b_{k+1}] := \begin{cases} [a_k, c_k], & \text{if } f(a_k)f(c_k) < 0; \\ [c_k, b_k], & \text{if } f(c_k)f(b_k) < 0. \end{cases}$
3. Stop when the interval $b_{k+1} - a_{k+1}$ is sufficiently small, or if $f(c_k) = 0$.

How does this method converge? Not bad for such a simple method. At the kth stage, there must be a root in the interval $[a_k, b_k]$. Take $c_k = \frac{1}{2}(a_k + b_k)$ as the next estimate to x_*, giving the error $e_k = c_k - x_*$. The *worst* possible error, attained if x_* is at a_k or b_k, is $\frac{1}{2}(b_k - a_k) = 2^{-k-1}(b_0 - a_0)$.

Theorem. The kth bisection point c_k is no further than $(b_0 - a_0)/2^{k+1}$ from a root.

We say this iteration *converges linearly* (the log of the error is bounded by a straight line when plotted against iteration count – an example is given later in this lecture) with rate $\rho = 1/2$. Practically, this means that the error is cut in half at each iteration, *independent of the behavior of f*. Reduction of the initial bracket width by ten orders of magnitude would require roughly $\log_2 10^{10} \approx 33$ iterations.

7.1.2. Regula Falsi.

A simple adjustment to bisection can often yield much quicker convergence. The name of the resulting algorithm, *regula falsi* (literally 'false rule') hints at the technique. As with bisection, begin with an interval $[a_0, b_0] \subset \mathbb{R}$ such that $f(a_0)f(b_0) < 0$. The goal is to be more sophisticated about the choice of the root estimate $c_k \in (a_k, b_k)$. Instead of simply choosing the middle point of the bracket as in bisection, we approximate f with the line $p_k \in \mathcal{P}_1$ that interpolates $(a_k, f(a_k))$ and $(b_k, f(b_k))$, so that $p_k(a_k) = f(a_k)$ and $p(b_k) = f(b_k)$. This unique polynomial is given (in the Newton form) by

$$p_k(x) = f(a_k) + \frac{f(b_k) - f(a_k)}{b_k - a_k}(x - a_k).$$

Now estimate the zero of f in $[a_k, b_k]$ by the zero of the linear model p_k:

$$c_k = \frac{a_k f(b_k) - b_k f(a_k)}{f(b_k) - f(a_k)}.$$

The algorithm then takes the following form:

For $k = 0, 1, 2, \ldots$

1. Compute $f(c_k)$ for $c_k = \dfrac{a_k f(b_k) - b_k f(a_k)}{f(b_k) - f(a_k)}$.
2. If $f(c_k) = 0$, exit; otherwise, repeat with $[a_{k+1}, b_{k+1}] := \begin{cases} [a_k, c_k], & \text{if } f(a_k)f(c_k) < 0; \\ [c_k, b_k], & \text{if } f(c_k)f(b_k) < 0. \end{cases}$
3. Stop when $f(c_k)$ is sufficiently small, or the maximum number of iterations is exceeded.

Note that Step 3 differs from the bisection method. In the former case, we are forcing the bracket width $b_k - a_k$ to zero as we find our root. In the present case, there is nothing in the algorithm to drive that width to zero: We will still always converge (in exact arithmetic) even though the bracket length does not typically decrease to zero. Analysis of *regula falsi* is more complicated than the trivial bisection analysis; we give a convergence proof only for a special case.

Theorem. Suppose $f \in C^2[a_0, b_0]$ for $a_0 < b_0$ with $f(a_0) < 0 < f(b_0)$ and $f''(x) \geq 0$ for all $x \in [a_0, b_0]$. Then *regula falsi* converges.

Proof. (See Stoer & Bulirsch, *Introduction to Numerical Analysis*, 2nd ed., §5.9.)

The condition that $f''(x) \geq 0$ for $x \in [a_0, b_0]$ means that f is convex on this interval, and hence $p_0(x) \geq f(x)$ for all $x \in [a_0, b_0]$. (If $p_0(x) < f(x)$ for some $x \in (a_0, b_0)$, then f has a local maximum at $\widehat{x} \in (a_0, b_0)$, implying that $f''(\widehat{x}) < 0$.) Since $p_0(c_0) = 0$, it follows that $f(c_0) \leq 0$, and so the new bracket will be $[a_1, b_1] = [c_0, b_0]$. If $f(c_0) = 0$, we have converged; otherwise, since $f''(x) \geq 0$ on $[a_1, b_1] \subset [a_0, b_0]$ and $f(a_1) = f(c_0) < 0 < f(b_0) = f(b_1)$, we can repeat this argument over again to show that $[a_2, b_2] = [c_1, b_1]$, and in general, $[a_{k+1}, b_{k+1}] = [c_k, b_k]$. Since $c_k > a_k = c_{k-1}$, we see that the points c_k are monotonically increasing, while we always have $b_k = b_{k-1} = \cdots = b_1 = b_0$. Since $c_k \leq b_k = \cdots = b_0$, the sequence $\{c_k\} = \{a_{k-1}\}$ is bounded. A fundamental result in real analysis tells us that bounded, monotone sequences must converge.† Thus, $\lim_{k\to\infty} a_k = \alpha$ with $f(\alpha) \leq 0$, and we have

$$\alpha = \frac{\alpha f(b_0) - b_0 f(\alpha)}{f(b_0) - f(\alpha)}.$$

This can be rearranged to get $(\alpha - b_0)f(\alpha) = 0$. Since $f(b_k) = f(b_0) > 0$, we must have $\alpha \neq b_0$, so it must be that $f(\alpha) = 0$. Thus, *regula falsi* converges in this setting. ■

Conditioning. When $|f'(x_0)| \gg 0$, the desired root is easy to pick out. In cases where $f'(x_0) \approx 0$, the root will be *ill-conditioned*, and it will often be difficult to locate. This is the case, for example, when x_0 is a multiple root of f. (You may find it strange that the more copies of a root you have, the more difficult it can be to compute it!)

Deflation. What is one to do if multiple distinct roots are required? One approach is to choose a new initial bracket that omits all known roots. Another technique, though numerically fragile, is to work with $\widehat{f}(x) := f(x)/(x - x_0)$, where x_0 is the previously computed root.

† If this result is unfamiliar, a few minutes of reflection should convince you that it is reasonable. (Imagine a ladder with infinitely many rungs stretching from floor to ceiling in a room with finite height: eventually the rungs must get closer and closer.) For a proof, see Rudin, *Principles of Mathematical Analysis*, Theorem 3.14.

MATLAB code. A bracketing algorithm for zero-finding available in the MATLAB routine `fzero.m`. This is more sophisticated than the two algorithms described here, but the basic principle is the same. Below are simple MATLAB codes that implement bisection and *regula falsi.*

```
 function xstar = bisect(f,a,b)
% Compute a root of the function f using bisection.
% f:    a function name, e.g., bisect('sin',3,4), or bisect('myfun',0,1)
% a, b: a starting bracket:  f(a)*f(b) < 0.
 fa = feval(f,a);
 fb = feval(f,b);         % evaluate f at the bracket endpoints
 delta = (b-a);           % width of initial bracket
 k = 0; fc = inf;         % initialize loop control variables
 while (delta/(2^k)>1e-18) & abs(fc)>1e-18
    c = (a+b)/2; fc = feval(f,c);   % evaluate function at bracket midpoint
    if fa*fc < 0, b=c; fb = fc;     % update new bracket
    else a=c; fa=fc; end
    k = k+1;
    fprintf(' %3d  %20.14f  %10.7e\n', k, c, fc);
 end
 xstar = c;
```

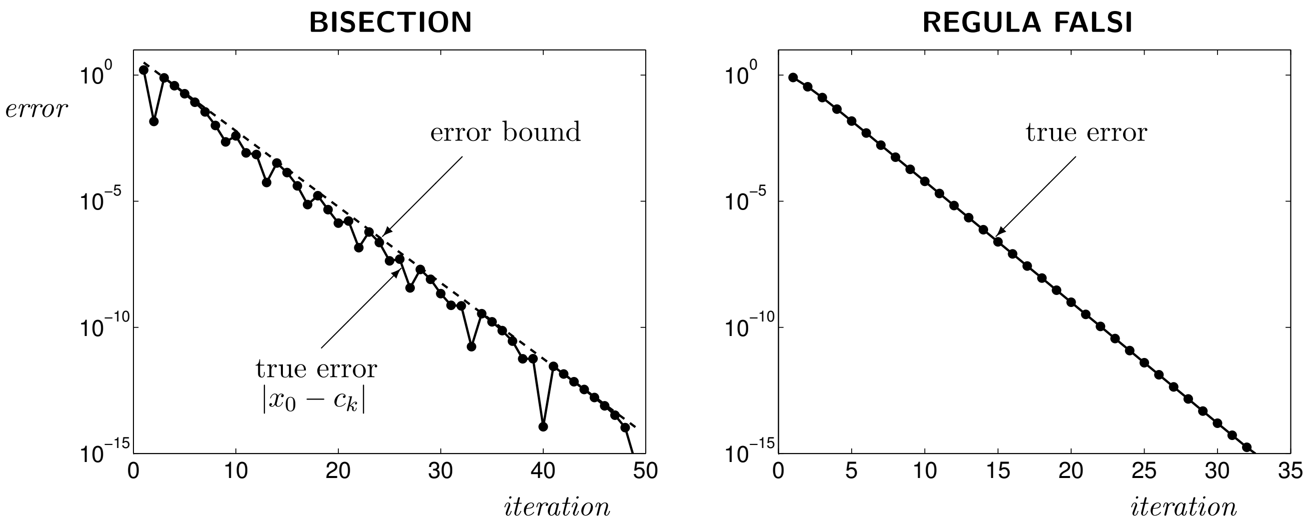
```
 function xstar = regulafalsi(f,a,b)
% Compute a root of the function f using regula falsi
% f:    a function name, e.g., regulafalsi('sin',3,4), or regulafalsi('myfun',0,1)
% a, b: a starting bracket:  f(a)*f(b) < 0.
 fa = feval(f,a);
 fb = feval(f,b);         % evaluate f at the bracket endpoints
 delta = (b-a);           % width of initial bracket
 k = 0; fc = inf;         % initialize loop control variables
 maxit = 1000;
 while (abs(fc)>1e-15) & (k < maxit)
    c = (a*fb - b*fa)/(fb-fa);    % generate new root estimate
    fc = feval(f,c);              % evaluate function at new root estimate
    if fa*fc < 0, b=c; fb = fc;   % update new bracket
    else a=c; fa=fc; end
    k = k+1;
    fprintf(' %3d  %20.14f  %10.7e\n', k, c, fc);
 end
 xstar = c;
```

Accuracy. Here we have assumed that we calculate $f(x)$ to perfect accuracy, an unrealistic expectation on a computer. If we attempt to compute x_* to very high accuracy, we will eventually experience errors due to inaccuracies in our function $f(x)$. For example, $f(x)$ may come from approximating the solution to a differential equation, were there is some approximation error we must be concerned about; more generally, the accuracy of f will be limited by the computer's floating point arithmetic. One must also be cautious of subtracting one like quantity from another (as in construction of c_k in both algorithms), which can give rise to *catastrophic cancellation.*

Minimization. A closely related problem is finding a local *minimum* of f. Note that this can be accomplished by computing and analyzing the zeros of f'.[‡]

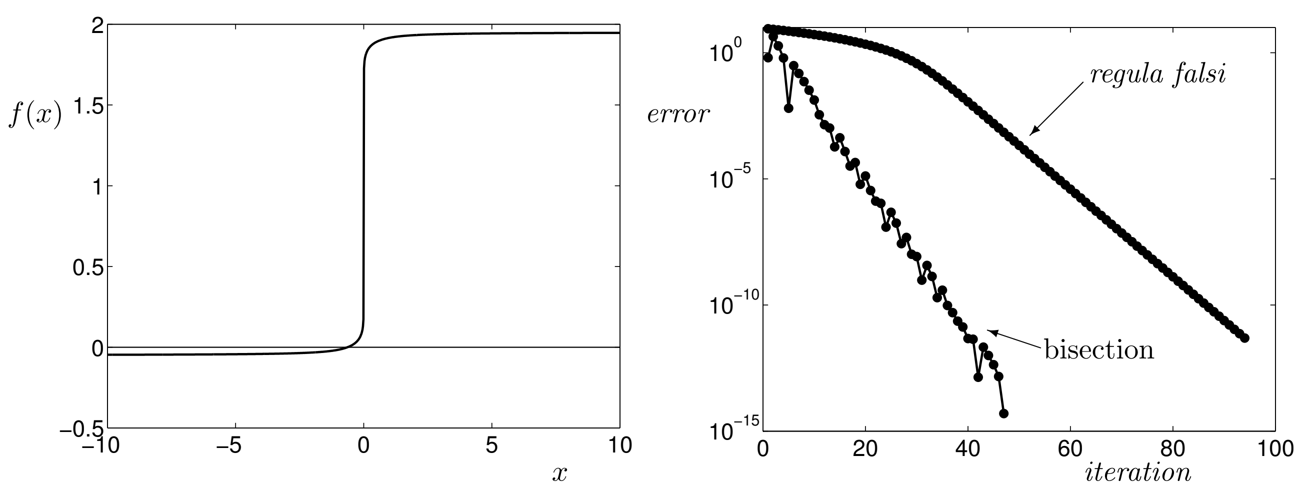
[‡]For details, see J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 2nd ed., Springer-Verlag, 1993, §5.9, or L. W. Johnson and R. D. Riess, *Numerical Analysis*, 2nd ed., Addison-Wesley, 1982, §4.2.

Below we show the convergence behavior of bisection and *regula falsi* when applied to solve the nonlinear equation $M = E - e\sin E$ for the unknown E, a famous problem from celestial mechanics known as *Kepler's equation*; see §7.4 in Lecture 40.



Error in root computed for Kepler's equation with $M = 4\pi/3$, $e = 0.8$ and initial bracket $[0, 2\pi]$.

Is *regula falsi* always superior to bisection? For any function for which we can construct a root bracket, one can always rig that initial bracket so the root is exactly at its midpoint, $\frac{1}{2}(a_0 + b_0)$, giving convergence of bisection in a single iteration. For most such functions, the first regula falsi iterate is different, and not a root of our function. Can one construct less contrived examples? Consider the function shown on the left below;[§] we see on the right that bisection outperforms *regula falsi*. The plot on the right shows the convergence of bisection and *regula falsi* for this example. *Regula falsi* begins much slower, then speeds up, but even this improved rate is slower than the rate of 1/2 guaranteed for bisection.



[§]This function is $f(x) = \mathrm{sign}(\tan^{-1}(x)) * |2\tan^{-1}(x)/\pi|^{1/20} + 19/20$, whose only root is at $x \approx -0.6312881\ldots$.