Lecture `root1`: Solving Nonlinear Equations: Bisection

September 7, 2022

Summary: Bisection as a method for root finding. Examples illustrating pitfalls of the method. Some aspects taken from C. F. Van Loan's *Introduction to Scientific Computing*.

References: Section 1.1

25-32

Nonlinear Equations

The next several lectures concern solution of the scalar equation

$$f(x) = 0, \tag{1}$$

where $f(x)$ is a nonlinear function. To quote Sauer's textbook (see p. 26): a "function $f(x)$ has a **root** at $x = r$ if $f(r) = 0$." We'll mostly use r to denote a root, but we'll not stick with any one notation. The main difference between a nonlinear equation and a scalar linear equation $ax - b = 0$ is the following. For a nonlinear equation we generally can't address existence and uniqueness of roots, that is we don't know if a root (solution) exists, or how many roots exist. Because there is no corresponding mathematical theory of existence and uniqueness (as we have for linear systems $A\mathbf{x} = \mathbf{b}$, for which $ax = b$ is the scalar case), in numerically solving nonlinear equations (scalar, as is the case here, or otherwise) one encounters the following challenges.

- Specifying an initial guess: we usually require an initial guess to start a "root-finding" algorithm. How should we obtain this guess?
- Distinguishing solutions: if there are two (or more) roots (solutions) and we are only interested in one of them, how do we select the relevant solution?
- Robustness of algorithms: a root-finding algorithm can fail to converge to any root at all (even if the initial guess is very close to a root).

Not all numerical root-finding algorithms suffer from all three of these listed problems. Fixed-point iteration, which we'll study later, is in fact vulnerable to all three; however, this lecture considers an algorithm which is robust under certain assumptions: *bisection*.

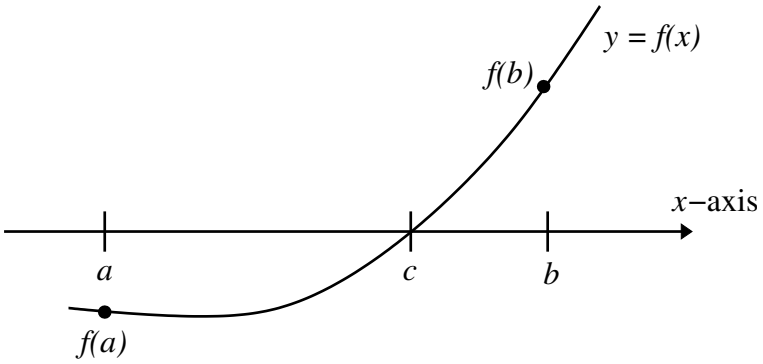

Figure 1: MOTIVATION FOR THE BISECTION METHOD. On $[a, b]$ surrounding c, we have $f(a) < 0$ and $f(b) > 0$; that is, $f(x)$ takes different signs at the endpoints. The case $f(a) > 0$ and $f(b) < 0$ gives rise to a similar figure. Here we have only one root, and it is *simple*, that is $f(x)$ changes sign across c.

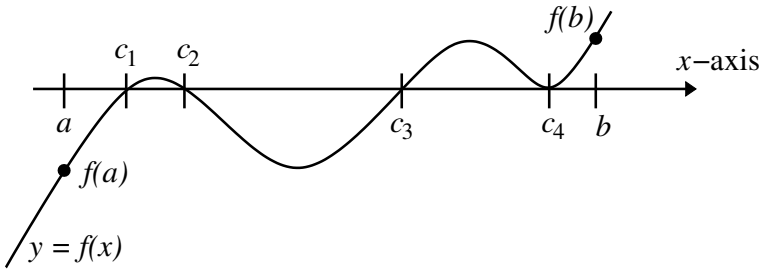

Figure 2: MULTIPLES ROOTS ON (a, b). Given the assumptions of the IVT, there may be more that one root, as depicted here. Here c_1, c_2, and c_3 are simple, while c_4 is not simple.

Bisection

The idea behind bisection is simple. Assume that we are attempting to solve a *scalar* nonlinear equation of the form (1), and that $f(x)$ is continuous. Suppose that a root (solution) r lies on the interval $x \in [a, b]$. Then the length $b-a$ of the interval is the "error" in our knowledge of r: by assumption we know an r lies somewhere on $[a, b]$, but we don't know where. The existence of (at least one) $r \in (a, b)$ can be guaranteed, provided $f(a)f(b) < 0$. [If $f(a)f(b) = 0$ then either a is a root, b is a root, or both a and b are roots; all three cases are ruled out with the assumption $f(a)f(b) < 0$.] This conclusion stems from the Intermediate Value Theorem (IVT):

Theorem. *If $f(x)$ is continuous on $[a, b]$ and $f(a)f(b) < 0$, then there is at least one point $c \in (a, b)$ such that $f(c)$.* (Question: what is the definition of continuity on a closed interval?)

Any such c can be taken as our r. See Fig. 1 for a graphical depiction of the IVT. In the figure there is only one $c \in (a, b)$ for which $f(c) = 0$, but there may be more than one, as shown in Fig. 2.

How does this information help us? Suppose that indeed $f(a) < 0$ and $f(b) > 0$, and further

that there is only one root $r \in (a, b)$, as in Fig. 1. Let's consider the midpoint $m = \frac{1}{2}(a + b)$ of the interval. Exactly one of the following three possibilities then holds:

- $f(m) < 0$: then $r \in [m, b]$.
- $f(m) > 0$: then $r \in [a, m]$.
- $f(m) = 0$: then $r = m$.

Suppose $f(m) > 0$. If we define m as the *new* b, then we are back where we started: $f(a) < 0$, $f(b) > 0$, and r lies somewhere in the middle by the IVT. What's the difference? We've chopped our interval in half. The "error" in our knowledge of r is now $m - a = b_{\text{new}} - a = \frac{1}{2}(b_{\text{old}} - a)$, that is half of what it used to be. By repeated application of this process, we can "zero in" on the value of r. Here is the corresponding pseudocode:

1. Start with $f(a)f(b) < 0$; that is, $f(a)$ and $f(b)$ are non-zero and have different signs. Otherwise put, the starting interval $[a, b]$ *brackets* a root of f.
2. Define $m = \frac{1}{2}(a + b)$, and calculate $f(m)$.
3. Do one of the following.
   (a) If $f(m) = 0$, then set $r = m$.
   (b) If $f(a)f(m) < 0$, then set $b \leftarrow m$, and go back to step 2.
   (c) If $f(b)f(m) < 0$, then set $a \leftarrow m$, and go back to step 2.

Here, for example, $b \leftarrow m$ means "overwrite b with m" or "make m the new b". As stated, this algorithm may take forever to find the *exact* value m such that $f(m) = 0$. Thus, we may consider terminating when $|f(m)| < y_{\text{tol}}$ for some small tolerance, say $y_{\text{tol}} = 10^{-8}$. As another termination criterion, we might also stop the algorithm if the length of the interval is sufficiently small, say $|b-a| < x_{\text{tol}}$. Note that, even with $y_{\text{tol}} = x_{\text{tol}}$, the criteria $|f(m)| < y_{\text{tol}}$ and $|b - a| < x_{\text{tol}}$ are not the same! There exist situations for which $|f(x)|$ is small over all of $x \in [a, b]$ containing a root r, yet $|b - a|$ is very large. There also exists situations for which $|b - a|$ is small, but $|f(x)|$ is large for some $x \in [a, b]$. (Can you make a sketch of these situations?) If either of these conditions are met, then we might choose to terminate our algorithm.

Our amended algorithm therefore has the following form.

1. Start with $f(a)f(b) < 0$; that is, $f(a)$ and $f(b)$ are non-zero and have different signs.
2. Define $m = \frac{1}{2}(a + b)$, and calculate $f(m)$.
3. Do one of the following.
   (a) If $|f(m)| < y_{\text{tol}}$ or $\frac{1}{2}|b - a| < x_{\text{tol}}$, terminate the loop and return m as the root.
   (b) If $f(a)f(m) < 0$, then set $b \leftarrow m$, and go back to step 2.
   (c) If $f(b)f(m) < 0$, then set $a \leftarrow m$, and go back to step 2.

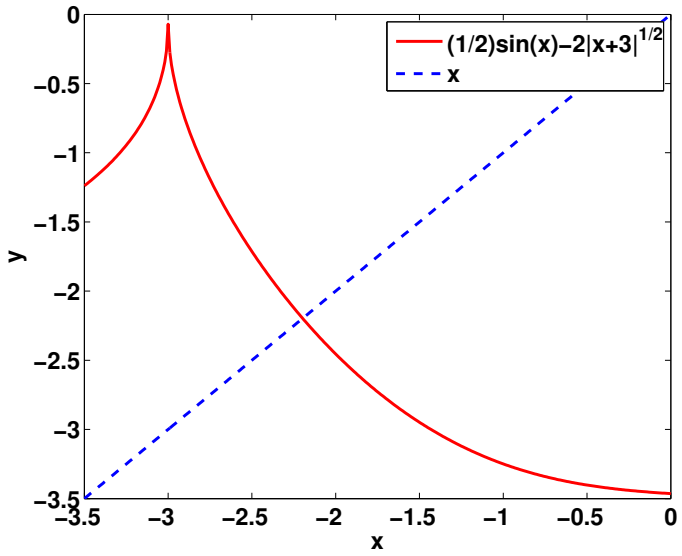

Figure 3: Plot of the function $h(x) = \frac{1}{2}\sin x - 2\sqrt{|x+3|}$ around the root $r \simeq -2.2$ of $f(x) = x - h(x)$.

This algorithm is relatively simple to implement and requires only one function evaluation per iteration: $f(m)$. Apparently, the algorithm is also more or less a foolproof: we'll always zero in on a root if we can find a starting interval $[a, b]$ that brackets a root of f. Can you explain why there is a $\frac{1}{2}$ factor in the termination criterion (a)? This algorithm isn't quite the one Sauer gives on p. 27. He essentially uses the termination criterion "(a) If $f(m) = 0$ or $\frac{1}{2}|b-a| \leq \epsilon_{\text{tol}}$, terminate the loop and return m as the root." That is, he chooses $y_{\text{tol}} = 0$. Van Loan suggests replacing ϵ_{tol} by $\epsilon_{\text{tol}} + \varepsilon_{\text{mach}} \cdot \max(|a|, |b|)$ to avoid possible loss of convergence due to finite machine precision effects. Here $\varepsilon_{\text{mach}} = 2^{-52} \simeq 2.22 \times 10^{-16}$ is machine precision (Sauer definition, the same as `eps` in MATLAB).

Example 1
Consider the equation

$$x + 2\sqrt{|x+3|} = \frac{1}{2}\sin x. \tag{2}$$

We write a MATLAB function `bisection.m` that implements the algorithm presented above. `bisection.m` takes in three inputs: a function handle (using the `@` notation), a scalar value for a, and a scalar value for b. A default tolerance of $10^{-8} + \varepsilon_{\text{mach}} \cdot \max(|a|, |b|)$ is assumed, and the algorithm is exited whenever `0.5(b-a)` drops below the tolerance. It is assumed that the interval $[a, b]$ brackets a root r. We also define a function `lecroot1_fun1.m` which evaluates the function

$$f(x) = x + 2\sqrt{|x+3|} - \frac{1}{2}\sin x.$$

The solution to $f(x) = 0$ is near $x = -2.2$. To test how robust bisection is, we'll adopt a coarse starting interval of $[-5, 5]$.

```
>> format long; format compact
>> bisection( @lecroot1_fun1 ,-5 ,5)
Converged in 29 iterations with tol 1.0e-08
ans =
  -2.197138844057918
```

Indeed, we see that `bisection` is able to find the sought root. In fact, bisection is **robust** if the equation has only one simple root:

```
bisection( @lecroot1_fun1 ,-50,50)
Converged in 33 iterations with tol 1.0e-08
ans =
  -2.197138842893764

>> bisection(@lecroot1_fun1,-291.02,632.333)
Converged in 36 iterations with tol 1.0e-08
ans =
  -2.197138842720921
```

We see in the example above that bisection works extremely well for an interval on which an equation has only one simple root. Indeed, these results all agree with each other to 8 digits. However, the situation can be more complicated.

Example 2
The function

$$f(x) = x^2$$

obviously has a root at $x = 0$. However, we cannot use bisection to locate this root, since there does not exist any pair of numbers a, b such that $f(a)f(b) < 0$; whence we cannot start the bisection algorithm. The lesson here is that we may not always be able to bracket a root.

Example 3
This example considers a function with more than one root. In fact, the function will have a variable number N of roots. Let

$$f_N(x) = \cos\left(Nx + 0.1\right),$$

and let us look over the interval $x \in [0, \pi]$ for roots. Now we can calculate the roots of this function explicitly:

$$r_i = \frac{(2i-1)\pi}{2N} - \frac{1}{10N}, \qquad i = 1, 2, \ldots, N \tag{3}$$

Thus, $f_N(x)$ has N roots over the interval $[0, \pi]$. This is unlike the examples we've seen before: most algorithms can only determine one root of a function at a time. If there are many roots, we run the risk of "confusing" the algorithm. Which root does bisection zero in on? First we note that for this particular function, N must be odd for us to use the bisection algorithm with $[0, \pi]$ as the starting interval. Indeed,

$$f_N(0) = \cos 0.1, \qquad f_N(\pi) = \cos(N\pi + 0.1) = \cos(N\pi)\cos 0.1 = (-1)^N \cos 0.1,$$

whence $f_N(0)f_N(\pi) > 0$ for N even. So let's assume that N is odd. If asked to numerically find the third root (counting to the right from $x = 0$) for odd $N \geq 3$, what should we do?

Let's see what happens when we run the bisection algorithm for $N = 5$:

```
>> bisection(@lecroot1_fun2,0,pi)
Converged in 28 iterations with tol 1.0e-08
ans =
   0.294159266399887
```

Which root is this? We can easily graph $f_5(x)$ over the interval $[0, \pi]$, as we have done in Fig. 4. We see that the third root looks to be around 1.5 [we could also substitute $i = 3$ into Eq. (3)]. However, we've found the first root at 0.3 [as seen graphically, or by using $i = 1$ in Eq. (3)]. Is there any way to tell the bisection algorithm to find the third root? The only control we have over the algorithm is the choice of initial interval on which the search is started. Graphically, we see that the third root looks to be between $x = 1.25$ and $x = 2$. Let's tell the algorithm to start on that interval:

```
>> bisection(@lecroot1_fun2,1.25,2)
Converged in 26 iterations with tol 1.0e-08
ans =
   1.550796324387193
```

This is the third root we sought.

What have we learned from the examples above?

- If we *know* that an interval brackets a single simple root of a function, as in **Example 1**, then bisection provides a foolproof method for finding that root.
- Bisection is useless unless we can start with an interval $[a, b]$ for which the function of interest satisfies $f(a)f(b) < 0$.
- If the function of interest has more than one root on the starting interval, there is no guarantee that we'll converge to the root we desire. To rectify this situation, we can try to modify our initial interval so that it brackets only the root we're interested in.

Bisection is a nice algorithm because of the first point above; under these mild assumptions it is robust. Of course, one needs to specify the initial bracketing interval, but this specification is really no more difficult than providing an initial guess, as will be needed for either fixed-point or Newton iteration. The main drawback to bisection, which we shall not highlight here, is that it cannot be easily generalized to nonlinear *systems* of equations. Both fixed-point iteration and Newton's method generalize from the scalar case to the system case.

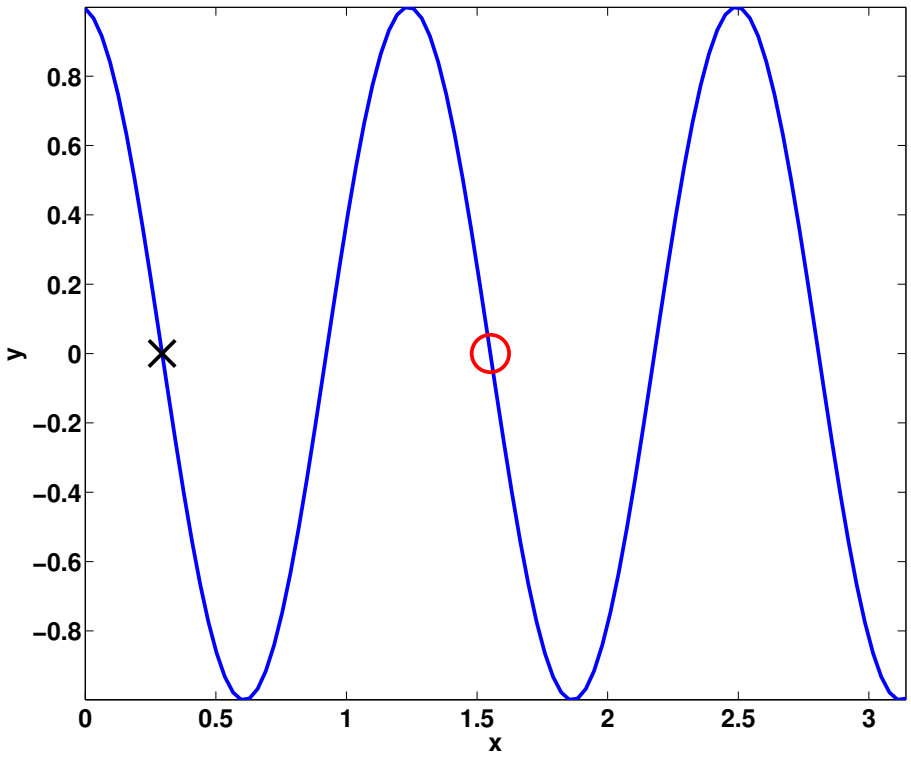

Figure 4: Graph of the function $f_5(x) = \cos(5x + 0.1)$ over $x \in [0, \pi]$. The black X denotes the root found by bisection given the starting interval of $[0, \pi]$. The red circle is the root we want to find.