MECH 221 Computer Lab 3: Root-Finding

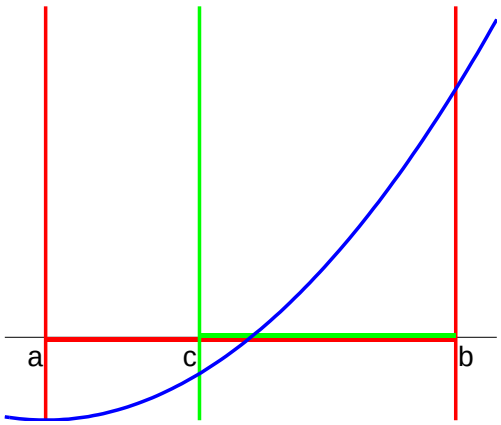
OVERVIEW: *Finding Real Zeros*

Today we will work with a Matlab function named `findzero` that solves equations of the form

$$f(x^*) = 0. \qquad (*)$$

This capability is useful in a huge variety of practical situations. Sometimes a single complex problem requires solving hundreds of different instances of problem $(*)$, so efficiency is an important consideration. In this lab, students will referee a competition between various approaches to select one that finds x^* with the smallest computational effort.

The function f in problem $(*)$ will be specified by the user, who will also provide a real interval $[a, b]$ such that f is positive at one endpoint and negative at the other. The figure below illustrates the scenario. The graph of f appears in blue, and the user-specified interval $[a, b]$ is shown in red.



Our job will be to take the given interval $[a, b]$ and shrink it, by applying steps 1–3 below.

1. Pick some number c in the open interval (a, b). Calculate $f(c)$.
2. If $f(c) = 0$, problem $(*)$ is solved. Tell the user $x^* = c$. Stop and celebrate.
3. If $f(c) \neq 0$, then $f(c)$ must have the same sign as one and only one of $f(a)$ and $f(b)$, because those two are different. So exactly one of the two cases below must arise.

   Case 1: $f(a)f(c) < 0$. Here f has opposite signs at the endpoints of subinterval $[a, c]$.

   Case 2: $f(c)f(b) < 0$. Here f has opposite signs at the endpoints of subinterval $[c, b]$.

   In either case we get a *subinterval* of $[a, b]$ on which f changes sign. Replacing the given $[a, b]$ with this subinterval completes one step of the shrinking operation.

The shrinking process outlined above is clearly repeatable. Applying it many times will produce a subinterval so short that every number it contains will be close enough to the exact theoretical value x^* for practical purposes.

To illustrate, suppose $f(x) = 2\sin(x/6) - 1$. Since $\sin(\pi/6) = \frac{1}{2}$, the number $x^* = \pi$ is an exact solution for equation $(*)$. (There are infinitely many others.) To get a numerical approximation for π from the function `findzero` that we will build, we will note $f(0) = -1 < 0$ and $f(9) = 2\sin(1.5) - 1 > 0$, and enter the commands

```
f = @(x)  2*sin(x/6) - 1;       % inline definition for function f
a = 0;                          % we know f(a)<0
b = 9;                          % we know f(b)>0
subint = findzero(f,[a,b]);     % shrink that bracketing interval!
```

Our function `findzero` returns a 2-element vector `subint` whose entries both agree with `pi` to machine precision.

Of course, $f(x) = 2\sin(x/6) - 1$ is a periodic function, so it has infinitely many roots. Several of these lie in the interval $[0, 88]$. (Note that $f(88) > 0$.) Our function finds one of them: saying

```
findzero(f,[0,88])
```

returns an interval where both endpoints round to `40.840704496667`. This number is extremely close to the exact theoretical solution at 13π.

DETAILS: *Definitions, Procedures, and Choices*

Given a real-valued function $f = f(x)$, we will call a real interval $[a, b]$ a *bracketing interval* if either

(i) $f(a)f(b) < 0$, i.e., f is negative at one end of the interval and positive at the other; or

(ii) $b = a$ and $f(a) = 0$. Note that when $b = a$, the closed interval $[a, b]$ has length 0.

If the given function f happens to be continuous on some bracketing interval $[a, b]$, then the Intermediate Value Theorem guarantees that the interval $[a, b]$ contains at least one number x^* satisfying $f(x^*) = 0$. Alternate terminology: "The interval $[a, b]$ brackets a root of f."

Endpoint Labelling. Using the symbols x_0 and x_1 for the endpoints of a bracketing interval provides a simple way to keep track of which one has been updated most recently. We can use x_0 for the stale old endpoint, and x_1 for the fresh recent one. But since our method could change either endpoint of a given interval, this interpretation requires us to deal with all of the legitimate possible relations $x_0 < x_1$, $x_0 > x_1$, or even $x_0 = x_1$. The bracketing interval associated with endpoints x_0 and x_1 will be

$$[a, b] = \Big[\min\{x_0, x_1\}, \max\{x_0, x_1\} \Big].$$

Stopping Tests. How do we know when all points of a bracketing interval with endpoints x_0, x_1 are acceptable? The idea of relative improvement provides a reasonable answer. Suppose the user provides a bracketing interval $[a, b]$ and we can produce a bracketing subinterval whose length is not more that $10^{-12}(b-a)$. Our work improves the user's estimated location by 12 orders of magnitude. That's probably good enough. So one stopping test could be to invite the user to specify a small constant τ_x (with a sensible default like 10^{-12}) and declare success when

$$\frac{|x_1 - x_0|}{|b - a|} < \tau_x.$$

Alternatively, we could consider relative improvements in the function value. Suppose we know some constant $f_{\text{typ}} > 0$ that reflects typical values for $|f(x)|$ in the original interval $[a, b]$. We might be satisfied when we achieve

$$\frac{\max\{|f(x_0)|, |f(x_1)|\}}{f_{\text{typ}}} < \tau_f$$

for some small user-specified tolerance τ_f. Notice that the ratios we consider above are pure numbers (any units of x or $f(x)$ cancel), so these stopping tests apply no matter what units or scales apply to x and f. Computational experts call user-specified values like τ_x and τ_f *relative tolerances*. Variable names resembling `reltol` are more descriptive than `tau`.

Pseudocode. Suppose a function f is given, along with two points x_0 and x_1 where $f(x_0)f(x_1) < 0$. Let $f_0 = f(x_0)$, $f_1 = f(x_1)$. Calculate, or ask the user to provide, relative tolerances $\tau_x > 0$ and $\tau_f > 0$. The following iterative process will replace x_0 and x_1 with endpoints of a much smaller bracketing interval.

1. Here we know $x_0 \neq x_1$ and $f_0 f_1 < 0$. Pick some point c strictly between x_0 and x_1. Define $f_c = f(c)$. Consider cases:
   (a) If $f_c = 0$, the degenerate interval $[c, c]$ brackets a root. Replace both x_0 and x_1 with c, and replace both f_0 and f_1 with 0.
   (b) If $f_c f_1 < 0$, the endpoints x_1 and c bracket a root. Replace x_0 and f_0 with x_1 and f_1, respectively. Then replace x_1 and f_1 with c and f_c.
   (c) If $f_c f_0 < 0$, the endpoints x_0 and c bracket a root. Replace x_1 and f_1 with c and f_c, respectively. (Leave x_0 and f_0 unchanged.)
2. Assess (and possibly report) progress so far. Stop the search if any one of these conditions holds:
   (i) The interval with endpoints x_0 and x_1 is short enough, i.e., $\dfrac{|x_1 - x_0|}{|b - a|} < \tau_x$, or
   (ii) The larger of $|f_0|$ or $|f_1|$ is close enough to 0, i.e., $\dfrac{\max\{|f(x_0)|, |f(x_1)|\}}{f_{\text{typ}}} < \tau_f$, or
   (iii) The computer resources available for solving $(*)$ are all used up.
3. Loop back to Step 1.

Stale Data. Notice that Step 1 will replace the point identified by the variable x_1 with the newly-computed x_c, no matter which case occurs. Also, f_1 will be updated to hold the value $f_c = f(x_c)$. In case 1(c), however, the point stored in the variable x_0 does not change. So as the method proceeds, variable x_1 always holds recent information, but the value in x_0 could be several steps old.

Efficiency. Using variables f_0, f_1, and f_c guarantees that f only gets evaluated once in each loop through steps 1–4. In some situations the function f is slow or expensive to calculate, so it would be inefficient to evaluate f at the same point more than once.

Choosing an Intermediate Point. Step 1 in the pseudocode is pretty vague: "Pick some point c strictly between x_0 and x_1." How? Six options present themselves. We will try them all.

(i) Choose at random. That is, pick some number r in the open interval $0 < r < 1$ at random, and let

$$c = x_r \stackrel{\text{def}}{=} x_0 + r(x_1 - x_0). \qquad (B)$$

The expression defining x_r equals x_0 when $r = 0$ and x_1 when $r = 1$, and lies somewhere in between these points whenever $0 < r < 1$.

(ii) Choose the midpoint. That is, get c from equation (B) with $r = \frac{1}{2}$. The resulting process is called The Bisection Method for root-finding.

(iii) Use linear interpolation. That is, imagine drawing the straight line from endpoint (x_0, f_0) to (x_1, f_1). That has slope $m = (f_1 - f_0)/(x_1 - x_0)$, so its equation is

$$y = f_0 + m(x - x_0).$$

Use the x-intercept of this line for c, i.e., solve for c in

$$0 = f_0 + m(c - x_0) \iff c = x_0 - \frac{f_0}{m} = x_0 - \frac{f_0(x_1 - x_0)}{f_1 - f_0} = \frac{f_1 x_0 - f_0 x_1}{f_1 - f_0}. \qquad (RF)$$

This is appealing because the choice of c is influenced by the values fo f that we already taken the trouble to find. (Note that the decisions in the pseudocode depend only the *signs* of the numbers f_0 and f_1, whereas the actual *values* influence c in (RF).) The resulting method is called Regula Falsi, or the method of False Position, because it replaces the true graph of f with a "false" linear approximation. See Wikipedia for some nice pictures.

(iv) Alternate. Use Regula Falsi on the first step and bisection (with $r = \frac{1}{2}$ on the second. Switch at each step, so that every odd step uses (RF) and every even step uses (B).

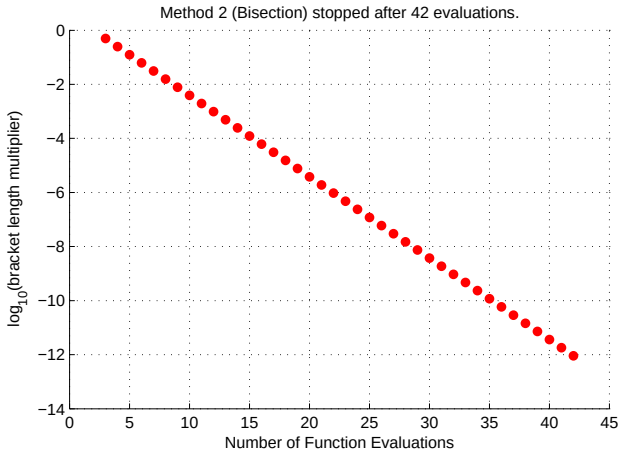
(v) Alternate, but with bias. The pseudocode is laid out so that the variable named x_1 always holds the most recently updated endpoint of the bracketing interval. We expect that formula (RF) should produce a point c quite close to the true root x^*, and both cases 1(b) and 1(c) will update the bracketing interval by putting c into variable x_1. So it seems reasonable to favour the endpoint x_1 a little in the bisection step. Let's try $r = 0.9$ instead of $r = 0.5$ when applying formula (B) in the even-numbered steps.

(vi) Use "The Illinois Algorithm". Details follow. In Step 1, calculate c using equation (RF). Then reconsider the 3-case split. In Case 1(a), we have an exact solution. No change is needed. In Case 1(b), we currently drop the older endpoint x_0, and use the two most recently generated x-values for the ends of the bracketing interval. This makes sense; let's make no change. In Case 1(c), however, the new information about $f(c)$ only moves the point x_1. The older endpoint labelled x_0 gets "stuck", and this could spoil our progress if it keeps happening over and over. Discourage this outcome by completing the update of x_1 and f_1 as planned, but then replacing f_0 with $f_0/2$ before looping back for the next step. This revised value of f_0 will no longer equal $f(x_0)$, and therefore the value of c emitted by the next evaluation of equation (RF) will probably be a worse approximation to x^* than before. But this can actually improve efficiency by triggering Case 1(b) and refreshing the data in the next step. Here is the modified version of Case 1(c), with the changed part shown in italics:

(c) If $f_c f_0 < 0$, the endpoints x_0 and c bracket a root. Replace x_1 with c, and replace f_1 with f_c. *Replace f_0 with $f_0/2$; leave x_0 unchanged.*

Convergence Plots. It's nice to assess a method's efficiency by comparing pictures. Here is an example: taking $f(x) = 2\sin(x/6) - 1$ with $a = 3$ and $b = 3.2$, we can apply the pseudocode above with the random choice of c. Every time we reach Step 4, we add the point $(N, \log_{10}(r))$ to a graph, where N is the total number of points at which $f(x)$ has been evaluated, and $r = |x_1 - x_0|/|b - a|$ is the total interval shrinkage achieved so far. The goal of the method is to descend to the level where $r < 10^{-12}$, so $\log_{10}(r) < -12$, in as few steps as possible; the shape of the plot is also of

interest.



PRELAB ASSIGNMENT

Suppose $f(x) = x^2 - 4x + 2$, $x_0 = 0$, and $x_1 = 2$. Make two copies of a sketch showing the graph $y = f(x)$, $0 \le x \le 2$ and the line segment joining (x_0, f_0) to (x_1, f_1). Use (RF) to calculate c and f_c. Below the sketch, show the numerical values of x_0, f_0, x_1, f_1, c, and f_c. The add different material to the two sketches, as detailed below.

1. On sketch 1, illustrate an ordinary Regula Falsi step. In detail, use the update rules given in the original pseudocode to determine new values for x_0, f_0, x_1, and f_1. Draw the new line segment joining the updated (x_0, f_0) and (x_1, f_1). Calculate the next value of c from (RF) and show it on the sketch.
2. On sketch 2, illustrate a step of the Illinois algorithm. That is, use the Illinois update rules to determine new values for x_0, f_0, x_1, and f_1. Draw the new line segment joining the updated (x_0, f_0) and (x_1, f_1). Calculate the next value of c from (RF) and show it on the sketch.

For both approaches, show the values of x_0, f_0, x_1, f_1, c, and f_c produced in the second step. (So each sketch should come with a 2-row table. The second line segment on the sketch and the second row in the table should be different between the two figures.)

LAB ACTIVITIES

Download two files from Connect:

- ☐ `findzero.m` – a template for the root-finding function, and
- ☐ `runlab3.m` – a script file that activates the function `findzero`.

Study both files to see how they work. Run the script `runlab3` to see what happens.

1. Improve the script `runlab3` by adding lines that will produce a reasonable sketch of the graph $y = f(x)$ on the closed interval $a \le x \le b$, and then draw a prominent red circle around the midpoint of the subinterval returned by the function `findroot`. (Recall the command `plot(x,y,'ro')` for this.)
2. As supplied, the function `findroot` initializes an empty plot that should eventually hold a convergence diagram like the one shown above. Insert the plotting commands required to make this happen. Test.

3. In the template file `findroot.m`, there is space and structure to support all six of the methods detailed above for picking an intermediate point c. However, only method 1 is correctly implemented. (That's the random choice, corresponding to item (i) in the list above.) Complete and test the implementations for methods 2–5. *Suggestion*: Work with one method at a time, and test it before tackling the next one.
4. Fix up method 6, the Illinois algorithm. Unlike methods 2–5, this calls for modifications that go beyond just the given `switch` statement. Take care not to break the other methods when adding this one to the list of working options!
5. Experiment with all six methods on some of the test functions listed below. Identify the two that seem least useful and drop them from the final competition. Run the remaining four methods on each of the four functions for your assigned lab day. Record the number of function evaluations used by each of the 4 methods to find a root for each of the 4 functions.
6. Write up and hand in your findings. In detail, your submission should include
   - ☐ Printed copies of your modified script `runlab3.m` and function `findzero.m`. Remember that your name and UBC ID must appear in the electronic form of these resources, so it gets printed by the computer.
   - ☐ A few lines of text naming the two methods you discarded and briefly explaining why you dropped them.
   - ☐ A table with 4 rows and 4 columns. Each row should deal with a different function; each column should deal with a different method. The values in the table should be the number of function evaluations required to find a root.
   - ☐ Four convergence plots for the same function, one for each of your selected methods. Use one of the four functions in the table described above. Choose one where the convergence styles of the different methods are obviously different.
   - ☐ A brief discussion of which method you consider best, based on your evaluation counts and convergence plots.

TEST FUNCTIONS

A careful analysis of the Illinois algorithm was published by Dowell and Jarratt in *BIT*, volume 11 (1971), pages 168–174. A copy is available on Connect. The test functions suggested below are taken from that paper, where you can find 16-entry tables like the ones described above.

Monday. Let $f_n(x) = 2xe^{-n} + 1 - 2e^{-nx}$. Your 4 functions come from using $n = 1, 5, 15, 20$. Use the starting interval $[a, b] = [0, 1]$.

Tuesday. Let $f_n(x) = (1+(1-n)^2)x-(1-nx)^2$. Your 4 functions come from using $n = 2, 5, 15, 20$. Use the starting interval $[a, b] = [0, 1]$.

Wednesday. Let $f_n(x) = x^2 - (1 - x)^n$. Your 4 functions come from using $n = 2, 5, 15, 20$. Use the starting interval $[a, b] = [0, 1]$.

Friday. Let $f_n(x) = e^{-nx}(x - 1) + x^n$. Your 4 functions come from using $n = 1, 5, 10, 15$. Use the starting interval $[a, b] = [0, 1]$.

Challenge. Let $f_n(x) = (nx - 1)/((n - 1)x)$ for the 4 values $n = 2, 5, 15, 20$. Use the starting interval $[a, b] = [0.01, 1]$.