Nonlinear equations:
The bisection method

MATH2070: Numerical Methods in Scientific Computing I

Location: http://people.sc.fsu.edu/~jburkardt/classes/math2070_2019/nonlinear_bisection/nonlinear_bisection.pdf

---

Why does a function cross the axis? To get to the other sign!

Mathematics provides methods for finding solutions to some algebraic equations like $x^2 - 2x - 15 = 0$. However, for most equations with any kind of complication, there is no way to come to an exact solution for x. The bisection method provides a computational path to solving a nonlinear equation.

The bisection method

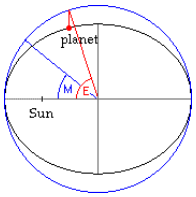
Given a nonlinear equation:

- *rewrite it as* $f(x) = 0$
- *find points* a *and* b *at which* $f(x)$ *takes opposite signs;*
- *shrink this interval until you are close enough to a solution.*

1 Example: Kepler's equation

Kepler's equation comes from an astronomical problem. It relates an important quantity, E, the eccentric anomaly of an orbit, to two easily measured items, the mean anomaly M and the orbital eccentricity e:

$$M = E - e \sin(E)$$



E is the "height" difference between an ideal and elliptic orbit.

Although we may know specific values of M and e, it is not possible to rewrite this equation to determine the exact value of E except for a few special cases. Suppose that we have determined that $M = 5$ and $e = 2$. To determine the value of E, we need to solve the equation:

$$5 = E - 2\sin(E)$$

We rewrite this equation in functional form:

$$f(E) = 5 - E + 2\sin(E) = 0$$

Assuming that E^* is an exact solution of this equation, we have three common measures of error:

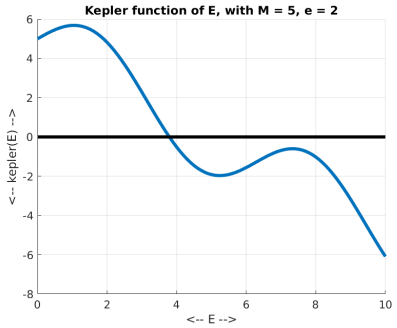
- $|E^* - E|$ is the (absolute) error;
- $\frac{|E^* - E|}{|E^*|}$ is the relative error;
- $|f(E)|$ is the (absolute) residual error;

To begin our investigation, let's write a MATLAB function to evaluate the residual:

```
function value = kepler ( x )
  value = 5.0 - x + 2.0 * sin ( x );
  return
end
```

Listing 1: kepler.m

We use plotting to search for a region in which the function crosses the axis:



By evaluation, we see that kepler(0) = 5 and kepler(10) = −6.0880. As E moves from 0 to 10, $f(E)$ moves from positive to negative. Assuming $f(E)$ is continuous, there must be a solution E^* somewhere in $[0, 10]$. Looking only at the change in sign, our best guess might be that the solution is near $E = 5$.

In fact, kepler(5) = −1.9178, which suggests we can find the root in $[0, 5]$. And if we keep looking at the midpoint of our current interval, we can repeatedly cut our uncertainty in half, and get as close to the root as we want (up to the limits of the machine accuracy).

2 Pseudocode #1: Bisection

Assume we know xn and xp so that $f(xn) < 0$ and $f(xp) > 0$. Our bisection algorithm might look like:

```
Input: function f, negative and positive arguments x- and x+

Loop a few times

  Set x to the average of x- and x+
```

```
6
7     If f(x) is negative, replace x- by x
8     else replace x+ by x
9
10  End loop
11
12  Output: updated values of x- and x+
```

Listing 2: bisection1_pseudocode.txt

3 MATLAB implementation #1:

Our simple bisection algorithm needs to stop at some point, so let's just let it run 10 steps:

```
1  function [ xn, xp ] = bisection1 ( f, xn, xp )
2
3    for it = 1 : 10
4
5      x = ( xn + xp ) / 2.0;
6
7      if ( f ( x ) < 0.0 )
8        xn = x;
9      else
10       xp = x;
11     end
12
13   end
14
15   return
16 end
```

Listing 3: bisection1.m

4 Pseudocode #2: Bisection

In class, we talked about sensible ways to tell an iteration to continue, or to stop with success or failure. This means we need to supply an iteration limit, and tolerances on x and $f(x)$. Our second pseudocode might be:

```
1
2   Input: function f, negative and positive arguments x- and x+, xtol, ftol, itmax
3
4   Begin loop
5
6     Set x to the average of x- and x+
7
8     If f(x) is negative, replace x- by x
9     else replace x+ by x
10
11    new = | x+ - x- |
12
13    if new is less than xtol and |f| is less than ftol, success
14    else if it > itmax, failure
15    else repeat
16
17  End loop
18
19  Output: updated values of x- and x+ and it
```

Listing 4: bisection2_pseudocode.txt

5 MATLAB implementation #2:

Our revised code might be:

```
function [ xn, xp, it ] = bisection2 ( f, xn, xp, xtol, ftol, itmax )

  it = 0;

  while ( true )

    it = it + 1;

    x = ( xn + xp ) / 2.0;

    if ( f ( x ) < 0.0 )
      xn = x;
    else
      xp = x;
    end

    new = abs ( xn - xp )

    if ( ...
      ( new <= xtol ) && ...
      ( abs ( f ( x ) ) <= ftol ) ...
    )
      return
    end

    if ( itmax <= it )
      return
    end

  end

  return
end
```

Listing 5: bisection2.m

6 Bisection for the Kepler equation

Test bisection code #2 on our Kepler equation:

```
xn = 10.0;
xp = 0.0;
xtol = 0.000001;
ftol = 0.000001;
itmax = 50;

[ xn, xp, it ] = bisection2 ( @(x)kepler(x), xn, xp, xtol, ftol, itmax );
```

Listing 6: kepler_bisection.m

Print statements in the script (not shown) report the following results:

```
  After 24 iterations:
  F(3.79084) = -1.52126e-06 (negative)
  F(3.79083) = 2.43377e-08 (positive)

  it <= itmax?  true
  |xn-xp| <= xtol?  true
```

```
7   min(|f(xn)|,|f(xp)|) <= ftol?  true
```

Listing 7: kepler_bisection.txt

To apply the bisection code to another problem, write a function file to evaluate $f(x)$, and modify the test script by specifying appropriate points xn and xp. You might also want to adjust the tolerances or the iteration limit.

7 Exercise: A trigonometric function

Consider the equationcos$(x) = x$ and suppose we want to find a solution x. Use the bisection method to approximate such a solution. To do this:

1. create a function file *trig.m* that evaluates the function.
2. create a script file like *trig_bisection.m* that calls `bisection2()` to find a zero.
3. run your script;
4. report the number of iterations required;
5. report as your solution x the average of the two endpoints;
6. report the value of $f(x)$;

8 MATLAB implementation #3: ALPHA, the update ratio:

In class, the quantity α or "alpha" was defined as the ratio $\alpha = \frac{\text{this x update}}{\text{previous x update}}$. For the bisection method, we can think of α as the ratio of the new interval to the old one, so that for bisection, α is always $\frac{1}{2}$. Nonetheless, here is how we could go through the motions of computing α:

```
1   function [ xn, xp, it ] = bisection3 ( f, xn, xp, xtol, ftol, itmax )
2
3     it = 0;
4
5     while ( true )
6
7       old = abs ( xp - xn );
8
9       it = it + 1;
10
11      x = ( xn + xp ) / 2.0;
12
13      if ( f ( x ) < 0.0 )
14        xn = x;
15      else
16        xp = x;
17      end
18
19      new = abs ( xp - xn );
20      alpha = new / old;
21
22      if ( ...
23        ( abs ( xn - xp ) <= xtol ) && ...
24        ( abs ( f ( x ) ) <= ftol ) ...
25      )
26        return
27      end
28
29      if ( itmax <= it )
30        return
31      end
```

```
32
33    end
34
35    return
36  end
```

Listing 8: bisection3.m

9 Fixed point iteration

In class, we saw a general method to solve nonlinear equations, called *fixed point iteration*. To find x^* that satisfies $f(x) = 0$, we repeatedly update $x \leftarrow g(x)$, where $g(x)$ is chosen so that $x^* = g(x^*)$.

We found a root of $f(x) = 3 * x - e^x = 0$ by using the fixed point function $g(x) = e^x/3$. Pseudocode for this computation would be:

```
1  itmax <-- ?
2  xtol <-- ?
3  ftol <-- ?
4
5  it <-- 0
6  xold <-- 0
7  x <-- initial value    (1.0 for this case)
8  old <-- 0
9  new <-- 0
10
11 Loop
12
13    it <-- it + 1
14    xold <-- x
15    x <-- g(x)           (x = exp(x)/3; for this case)
16    old <-- new
17    new <-- abs ( x - xold )
18
19    if old is not 0, alpha <-- new / old, print alpha
20    if new <= xtol and |f(x)| <= ftol, break from loop with success
21    if it > itmax, break from loop with failure
22
23 End loop
24
25 Print x, f(x)
26
27 function value = f ( x )
28    value <-- ?;  ( value = 3 * x - exp ( x ); for this case )
29    return
30 end
```

Listing 9: fixed_point_pseudocode.txt

10 Assignment #3: Implement and test a fixed point iteration

Use the pseudocode above as a guide, write a script called *hw3.m* which uses fixed point iteration to find a root of the function $f(x) = x^2 - 5$. Use the fixed point function $g(x) = 1 + x - x^2/5$.

Turn in: your file *hw3.m* by Friday, September 13.