Lecture 7

Root finding I

1 Introduction

For our present purposes, *root finding* is the process of finding a real value of x which solves the equation $f(x)=0$. Since the equation $g(x)=h(x)$ can be rewritten as $f(x)=g(x)-h(x)=0$, this encompasses the solution of any single equation in a single unknown. Ideally we would want to know how many roots exist and what their values are. In some cases, such as for polynomials, there are theoretical results for the number of roots (some of which might be complex) and we have clues about what we are looking for. However, for an arbitrary function $f(x)$ there is not much we can say. Our equation may have no real roots, for example $1+e^{-x}=0$, or, as in the case of $\sin(x)=0$ with roots $x=n\pi, n=0,\pm1,\pm2,\ldots$, there may be an infinite number of roots. We will limit our scope to finding one root – any root. If we fail to find a root it will not mean that the function has no roots, just that our algorithm was unable to find one.

To have any hope of solving the problem we need to make basic assumptions about $f(x)$ that allow us to know if an interval contains a root, if we are "close" to a root, or in what direction (left or right) along the *x* axis a root might lie. At a minimum we will have to assume that our function is *continuous*. Intuitively, a continuous function is one that can be plotted "without lifting pen from paper," while the plot of a discontinuous function has "breaks." Formally a function $f(x)$ is continuous at $x=c$ if for any $\epsilon>0$ there exists a $\delta>0$ such that

$$|x-c|<\delta \rightarrow |f(x)-f(c)|<\epsilon$$

If $f(x)$ is continuous at all points in some interval $a\le x\le b$ then it is continuous over that interval. Continuity allows us to assume that a small change in *x* results in a small change in $f(x)$. It also allows us to know that if $f(a)>0, f(b)<0$ then there is some *x* in the interval (a,b) such that $f(x)=0$, because a continuous curve cannot go from above the *x* axis to below without crossing the *x* axis.

In some cases we will also assume that $f(x)$ is *differentiable*, meaning the limit

$$f'(x)=\lim_{\delta\to 0}\frac{f(x+\delta)-f(x)}{\delta}$$

exists for all *x* of interest. This allows us to approximate the function by the line $f(x+\delta)\approx f(x)+f'(x)\delta$ over at least a small range of *x* values.

2 Graphical solution

The easiest and most intuitive way to solve $f(x)=0$ is to simply plot the function and zoom in on the region where the graph crosses the x axis. For example, say we want to find the first root of $\cos(x)=0$ for $x\ge 0$. We could run the command

```
x = 0:0.01:3;
y = cos(x);
plot(x,y);
```

```
xgrid; //short-hand way to add a grid in Scilab (grid in Matlab)
```

to get the plot shown in Fig. 1. We can see that there is a root in the interval $1.5 \le x \le 1.6$. We then use the magnifying lens tool (Fig. 2) to zoom in on the root and get the plot shown in Fig. 3. From this we can read off the root to three decimal places as

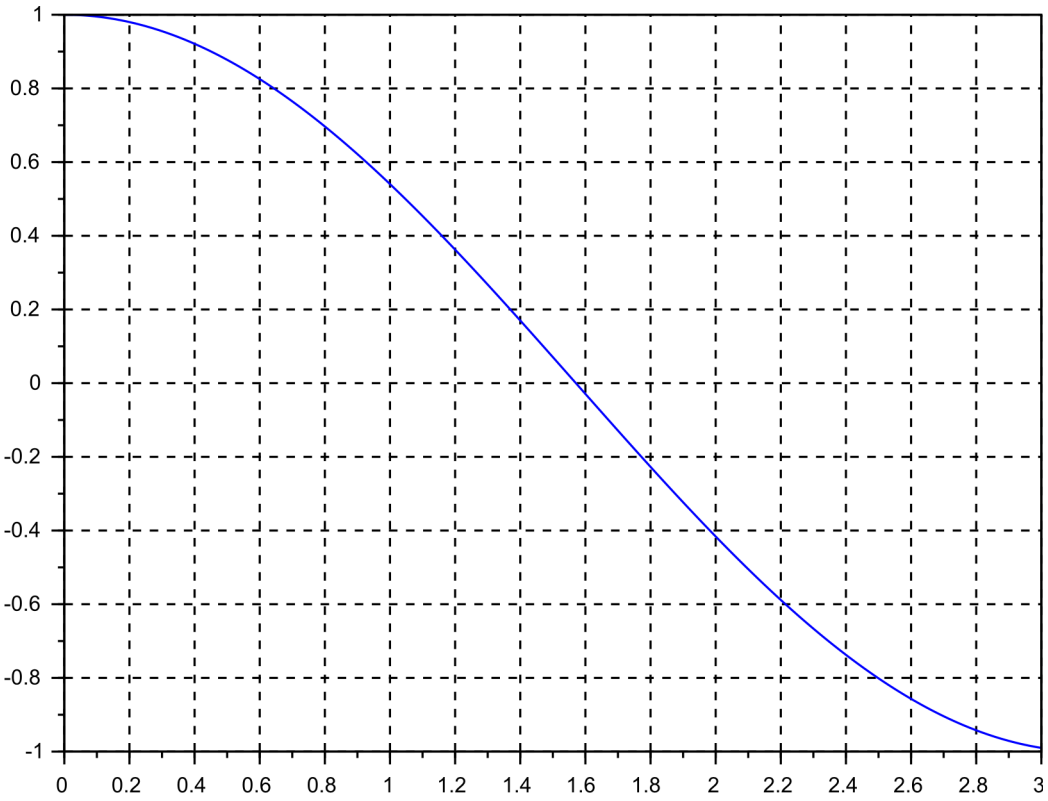

Fig. 1: Plot of cos(x)

$$x = 1.571$$

This approach is easy and intuitive. We can readily search for multiple roots by plotting different ranges of x. However, in many situations we need an automated way to find roots. Scilab (and Matlab) have built in functions to do this, and we will learn how to use those tools. But, in order to understand what those functions are doing, and what limitations they may have, we need to study root-finding algorithms. We start with one of the most basic algorithms called *bisection*.

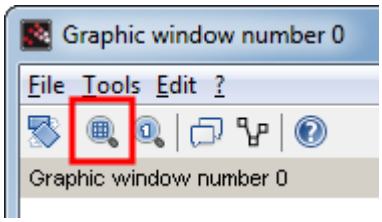

Fig. 2: Zoom tool

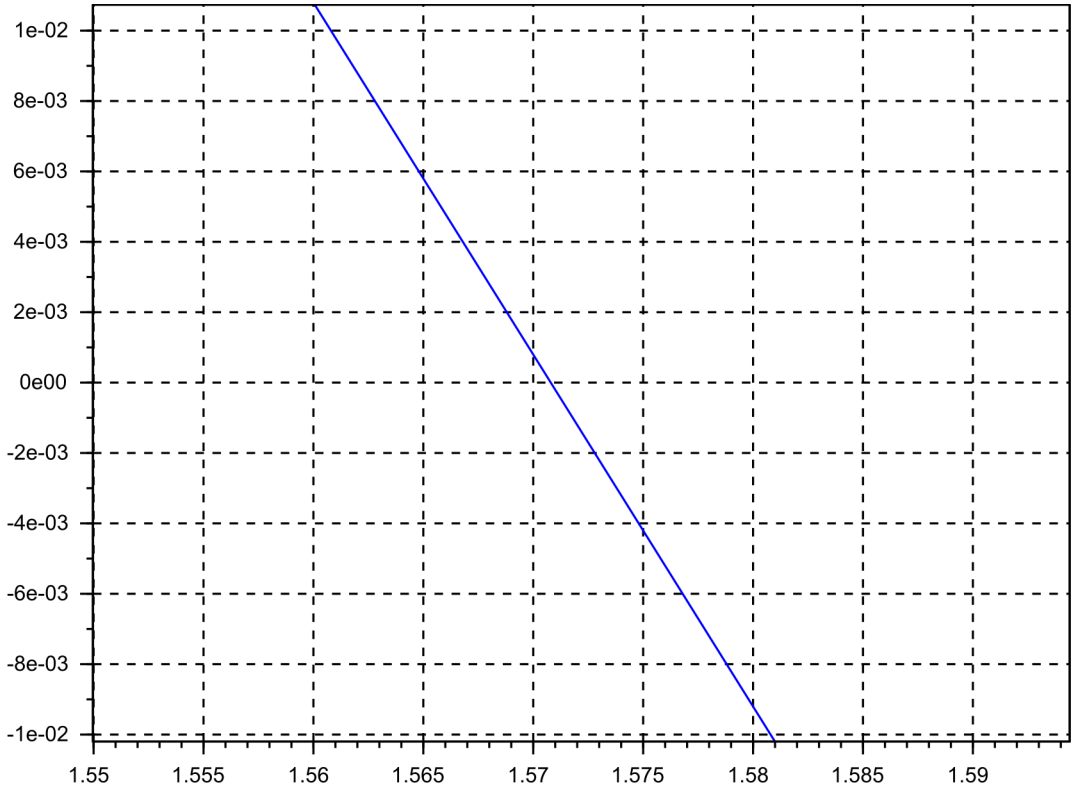

Fig. 3: Zooming in on the root

3 Bisection

If the product of two numbers is negative then one of the numbers must be positive and the other must be negative. Therefore, if $f(x)$ is continuous over an interval $a \le x \le b$, and if $f(a)f(b)<0$ then $f(x)$ is positive at one end of the interval and negative at the other end. Since $f(x)$ is continuous we can conclude that $f(r)=0$ for some r in the interior of the interval, because you cannot draw a continuous curve from a positive value of $f(x)$ to a negative value of $f(x)$, or vice versa, without passing through $f(x)=0$. This is the basis of the *bisection* method, illustrated in Fig. 4.

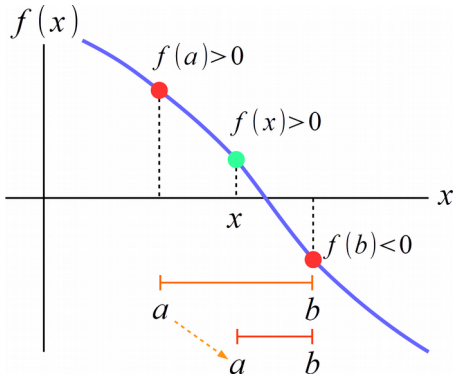

Fig. 4: Bisection

If $f(a)f(b)<0$ then we can estimate the root by the interval's midpoint with an uncertainty of half the length of the interval, that is, $r=(b+a)/2\pm(b-a)/2$. To reduce the uncertainty by half we evaluate the function at the midpoint $x=(b+a)/2$. If $f(x), f(a)$ have the same sign (as in the case illustrated in Fig. 4) we set $a=x$. If $f(x), f(b)$ have the same sign we set $b=x$. In the (very unlikely) event that $f(x)=0$ then $r=x$ is the root. In either of the first two cases we have "bisected" the interval $a\le x\le b$ into an interval half the original size. We then simply repeat this process until the uncertainty $|b-a|/2$ is smaller than desired. This method is simple and guaranteed to work for any continuous function. The algorithm can be represented as follows

Bisection algorithm

repeat until $|b-a|/2$ *is smaller than tol*

set x = midpoint of interval (a,b)

if $f(x)$ *has the same sign as* $f(a)$ *then set* $a=x$

else if $f(x)$ *has the same sign as* $f(b)$ *then set* $b=x$

else $f(x)$ *is zero and we've found the root!*

Function `rootBisection` in the Appendix implements the bisection algorithm.

3.1 Bracketing a root

To start the bisection algorithm we need two values $x=a,b$ such that $f(a)f(b)<0$. We say that *a* and *b bracket a root*, meaning a root is guaranteed to lie in the interval (a,b). How do we bracket a root in the first place? In an interactive session a plot of $f(x)$ allows you to quickly spot values which bracket a root. In Fig. 1 we immediately see that $a=1.4, b=1.6$ bracket a root.

In some cases a bracketing interval might be obvious from the form of the function. As an example, a problem that comes up in fiber optics is the solution of an equation of the form $f(x)=\sqrt{1-x^2}-\tan(ax)=0$ for $x\ge 0$. Since $f(0)=1>0$ and $f(x\to\pi/(2a))\to-\infty$ we know that the interval $(0,\pi/(2a))$ must bracket a root.

In general there is no sure-fire method for bracketing a root, even if a root exists. Fig. 5 is an example of a function with roots that would be difficult to find unless we happened to start searching near $x=0$.

One approach to getting started is a *grid search*. Let $x_{min}<x<x_{max}$ be an interval in which we believe one or more roots $f(x)=0$ might exist. Let Δx be the smallest scale at which we think the function is likely to change significantly (in Fig. 5 this might be $\Delta x\approx 0.1$. Our search grid is then $(x_{min}, x_{min}+\Delta x, x_{min}+2\Delta x, \ldots, x_{max})$. We search the grid by calculating $f(x)$ at each grid point. Any time we observe a sign change we have found a new root.

Grid searching is a non-interactive version of the "plot and see" method and it has the same pros and cons. On the "pro" side, it will work even with "hidden roots" (as in Fig. 5), and it (ideally) allows us to identify *all* roots in a prescribed interval. On the "con" side, it requires many function evaluations. If the function is "costly" to evaluate, or if the root finding operation is in a

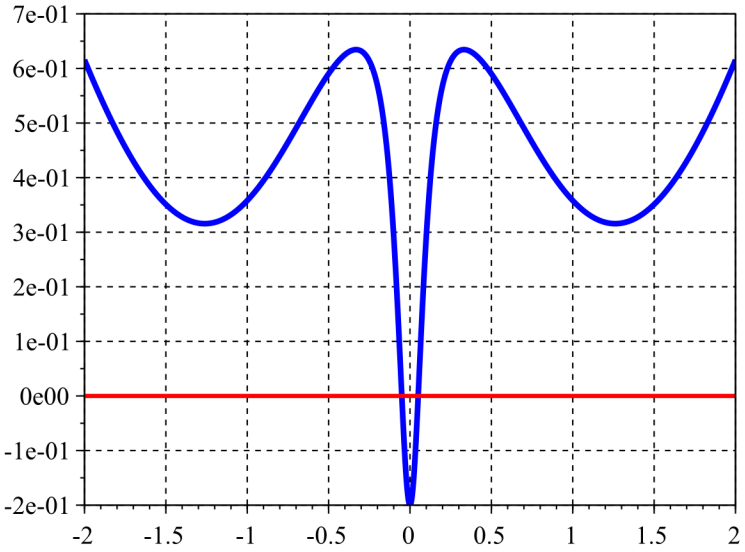

Fig. 5: A difficult root-finding problem

loop that will be repeated many times, then a grid search is not likely to be practical.

Another way to approach root bracketing is to start at an arbitrary value $x=a$ with some step size *h*. We then move along the *x* axis such that $|f(x)|$ is decreasing (that is, we are moving towards $y=0$) until we find a bracket interval (a,b). If $|f(x)|$ starts to increase before we find a bracket then we give up. Increasing the step size at each iteration protects us from getting into a situation where we are "inching along" a function that has a far-away zero. Function `rootBracket` in the Appendix implements this idea.

3.2 Convergence

The uncertainty in the root, the maximum error in our bisection estimate, is $\epsilon=|b-a|/2$. This decreases by a factor of ½ with each bisection. Therefore the relationship between the error at step *k* and step $k+1$ is

$$\epsilon_{k+1}=\frac{1}{2}\epsilon_k \tag{1}$$

More generally we might have an error $\epsilon=|x-r|$ that decreases as

$$\epsilon_{k+1}\approx\alpha\epsilon_k^q \tag{2}$$

The exponent *q* is called the *order of convergence*. If $q=1$, as it is for bisection, we say the convergence is *linear*, and we call α the *rate of convergence*. For $q>1$ the convergence is said to be *superlinear*, and specifically, if $q=2$ the convergence is *quadratic*. For superlinear convergence α is called the asymptotic *error constant*.

From (1) and (2) we see that bisection converges linearly with a rate of convergence of ½. If the initial error is ϵ_0 then after *k* iterations we have

$$\epsilon_k = \epsilon_0 \left(\frac{1}{2} \right)^k$$

In order to add one decimal digit of accuracy we need to decrease the error by a factor of 1/10. To increase accuracy by *n* digits we require

$$\left(\frac{1}{2} \right)^k = \frac{1}{10^n}$$

Solving for *k* we find

$$k = \frac{n}{\log 2} = 3.32\, n$$

It takes $k \approx 10$ iterations to add $n = 3$ digits of accuracy. All linearly converging root finding algorithms have the characteristic that each additional digit of accuracy requires a given number of algorithm iterations.

Now consider a quadratically convergent method with

$$\epsilon_{k+1} = \epsilon_k^2$$

Suppose $\epsilon_0 = 0.1$, so we start with one decimal place of accuracy. Then

$$\epsilon_1 = 0.01 \;,\; \epsilon_2 = 0.0001 \;,\; \epsilon_3 = 0.00000001 \;,\; \epsilon_4 = 0.0000000000000001$$

The first iteration adds one decimal place of accuracy. The second adds two decimal places. The third adds four and the fourth adds eight. This "runaway" increase in accuracy is what motivates us to explore other root-finding algorithms.

4 Fixed-point iteration

The goal of root finding is to arrive at $x = r$ where $f(r) = 0$. Now consider the following three equations

$$\begin{aligned} 0 &= f(x) \\ 0 &= a\, f(x) \\ x &= x + a\, f(x) \end{aligned}$$

Multiplying the first equation through by *a* produces the second equation. Adding *x* to both sides of the second equation produces the third. All three will have the same roots, provided $a \neq 0$. Now let's define a new function

$$g(x) = x + a\, f(x)$$

Then our root-finding problem can be written

$$x = g(x)$$

the solution of which is

$$r = g(r)$$

What's attractive about this is that it has the form "*x* equals something," which is almost an explicit formula for *x*. The problem is that the "something" itself depends on *x*. So let's imagine starting with a guess for the root $x = x_0$. We might expect that $x_1 = g(x_0)$ would be an improved

guess. We could continue iterating as many times as desired

$$x_1 = g(x_0)\ ,\ x_2 = g(x_1)\ ,\ \ldots\ ,\ x_{k+1} = g(x_k)$$

Our hope is that this converges with $x_k \to r$ as $k \to \infty$. Writing

$$x_k = r + \epsilon_k$$

ϵ_k is the error in the k^{th} root estimate. Suppose that the error is small enough that the 1st order Taylor series

$$g(r+\epsilon) \approx g(r) + g'(r)\epsilon = r + g'(r)\epsilon$$

is accurate. Then

$$r + \epsilon_{k+1} = r + g'(r)\epsilon_k$$

and

$$\epsilon_{k+1} = g'(r)\epsilon_k$$

It follows that

$$\epsilon_k = \left[g'(r)\right]^k \epsilon_0 \tag{3}$$

Comparing (3) to (2) we see that provided

$$\alpha = \left|g'(r)\right| < 1 \tag{4}$$

fixed-point iteration converges linearly with rate of convergence α. The value *r* is said to be a "fixed point" of the iteration since the iteration stays "fixed" at $r = g(r)$. Condition (4) requires

$$\left|1 + a\, f'(r)\right| < 1$$

Provided $f'(r) \neq 0$ we can always find a value *a* satisfying this condition.

Fixed-point iteration

Set $g(x) = x + a\, f(x)$ *for some choice of a*

From initial root estimate x_0

iterate until $\left|x_{n+1} - x_n\right| < tol$

$x_{n+1} = g(x_n)$

if iteration diverges, select another value of a

Example: Suppose we wish to solve

$$f(x)=x^2-2=0$$

By inspection the roots are $\pm\sqrt{2}$. Let

$$g(x)=x+f(x)=x+x^2-2$$

Since

$$g'(\sqrt{2})=1+2\sqrt{2}>1$$

we expect that fixed point iteration will fail. Starting very close to a root with $x=1.4$. iteration gives the sequence of values

$$1.36\,,1.2096\,,0.6727322\,,-0.8746993\,,-2.1096004$$

which is clearly moving away from the root. On the other hand taking

$$g(x)=x-0.25\,f(x)=x-0.25(x^2-2)$$

for which

$$g'(\sqrt{2})=0.293$$

we get the sequence of values

$$1.412975\,,1.4138504\,,1.4141072\,,1.4141824\,,1.4142044\,,1.4142109$$

which is converging to $\sqrt{2}=1.4142136$.

Interestingly, if $f'(r)\neq 0$ we have

$$\alpha=\left|g'(r)\right|=\left|1+a\,f'(r)\right|=0$$

Solving for *a*

$$a=-\frac{1}{f'(r)} \tag{5}$$

It seems we might achieve superlinear convergence for this choice of *a*. This is one way to motivate *Newton's method* which we cover in the next lecture.

4.1 Root bracketing vs. root polishing

In the vocabulary of numerical analysis, a method like bisection which starts with and always maintains a root bracket $(a\,,b)$ is called a "root bracketing" method. A technique such as fixed-point iteration which starts at some value x_0 and generates a sequence $x_1, x_2, x_3, \ldots, x_k$ is called a "root polishing" method. Root bracketing has the advantages that the existence of a bracket *guarantees* (for a continuous function) the existence of a root within that bracket, and the error in taking the midpoint of the bracketing interval as the root is *guaranteed* to be no more than half the width of the interval. The price we pay for these guarantees is that we have to have a root bracketed from the start. Root polishing has the advantage that we don't need a root bracket to start, just a single value of *x*. On the down side root polishing techniques are not guaranteed to converge to a root, even if we start off near one, and they don't give us a rigorous bound on the error in the value of a root if we do.

5 Appendix – Scilab code

```
0001  ////////////////////////////////////////////////////////////////////
0002  // rootBisection.sci
0003  // 2014-06-04, Scott Hudson, for pedagogic purposes
0004  // Implements bisection method for finding a root f(x) = 0.
0005  // Requires a and b to bracket a root, f(a)*f(b)<0.
0006  // Returns root as r with maximum error tol.
0007  ////////////////////////////////////////////////////////////////////
0008  function r=rootBisection(a, b, f, tol)
0009    fa = f(a);
0010    fb = f(b);
0011    if (fa*fb>=0) //make sure a,b bracket a root
0012      error('rootBisection: fa*fb>=0');
0013    end
0014    while (abs(b-a)/2>tol) //stop when error in root < tol
0015      x = (a+b)/2; //midpoint of interval
0016      fx = f(x);
0017      if (sign(fx)==sign(fa)) //r is in (x,b)
0018        a = x;                //move a to x
0019        fa = fx;
0020      elseif (sign(fx)==sign(fb)) //r is in (a,x)
0021        b = x;                    //move b to x
0022        fb = fx;
0023      else //unlikely case that fx==0, sign(fx)==0, we found the root
0024        a = x; //shrink interval to zero width a=b=x
0025        b = x;
0026      end
0027    end
0028    r = (a+b)/2; //midpoint of last bracket interval is root estimate
0029  endfunction
```

```
0001  ////////////////////////////////////////////////////////////////////
0002  // rootBracket.sci
0003  // 2014-06-04, Scott Hudson, for pedagogic purposes
0004  // Given a function f(x), starting point x=x0 and a stepsize h
0005  // search for a and b such that f(x) changes sign over [a,b] hence
0006  // bracketing a root.
0007  ////////////////////////////////////////////////////////////////////
0008  function [a, b]=rootBracket(f, x0, h)
0009    a = x0;
0010    fa = f(a);
0011    b = a+h;
0012    fb = f(b);
0013    done = (sign(fa)~=sign(fb)); //if the signs differ we're done
0014    if (~done) //if we don't have a bracket
0015      if (abs(fb)>abs(fa)) //see if a->b is moving away from x axis
0016        h = -h; //if so step in the other direction
0017        b = a; //and we will start from a instead of b
0018        fb = fa;
0019      end
0020    end
0021    while (~done)
0022      a = b; //take another step
0023      fa = fb;
0024      h = 2*h; //take bigger steps each time
0025      b = a+h;
0026      fb = f(b);
0027      done = (sign(fa)~=sign(fb));
0028      if ((abs(fb)>abs(fa))&(~done)) //we're now going uphill, give up
0029        error("rootBracket: cannot find a bracket\n");
0030      end
0031    end
0032  endfunction
```

6 Appendix – Matlab code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rootBisection.m
%% 2014-06-04, Scott Hudson, for pedagogic purposes
%% Implements bisection method for finding a root f(x) = 0.
%% Requires a and b to bracket a root, f(a)*f(b)<0.
%% Returns root as r with maximum error tol.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function r = rootBisection(a,b,f,tol)
  fa = f(a);
  fb = f(b);
  if (fa*fb>=0) %%make sure a,b bracket a root
    error('rootBisection: fa*fb>=0');
  end
  while (abs(b-a)/2>tol) %%stop when error in root < tol
    x = (a+b)/2; %%midpoint of interval
    fx = f(x);
    if (sign(fx)==sign(fa)) %%r is in (x,b)
      a = x;                %%move a to x
      fa = fx;
    elseif (sign(fx)==sign(fb)) %%r is in (a,x)
      b = x;                    %%move b to x
      fb = fx;
    else %%unlikely case that fx==0, sign(fx)==0, we found the root
      a = x; %%shrink interval to zero width a=b=x
      b = x;
    end
  end
  r = (a+b)/2; %%midpoint of last bracket interval is root estimate
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rootBracket.m
%% 2014-06-04, Scott Hudson, for pedagogic purposes
%% Given a function f(x), starting point x=x0 and a stepsize h
%% search for a and b such that f(x) changes sign over [a,b] hence
%% bracketing a root.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [a,b] = rootBracket(f,x0,h)
  a = x0;
  fa = f(a);
  b = a+h;
  fb = f(b);
  done = (sign(fa)~=sign(fb)); %%if the signs differ we're done
  if (~done) %%if we don't have a bracket
    if (abs(fb)>abs(fa)) %%see if a->b is moving away from x axis
      h = -h; %%if so step in the other direction
      b = a; %%and we will start from a instead of b
      fb = fa;
    end
  end
  while (~done)
    a = b; %%take another step
    fa = fb;
    h = 2*h; %%take bigger steps each time
    b = a+h;
    fb = f(b);
    done = (sign(fa)~=sign(fb));
    if ((abs(fb)>abs(fa))&(~done)) %%we're now going uphill, give up
      error('rootBracket: cannot find a bracket');
    end
  end
end
```