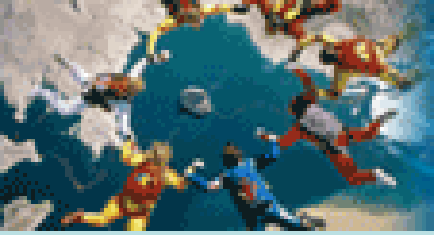


Numerical Methods for Engineers

Steven C. Chapra
Raymond P. Canale

Fifth Edition

~ Roots of Equations ~

Bracketing Methods

Chapter 5

Edited by: Dr. wafaa El-Haweet

Roots of Equations

- Easy

$$ax^2 + bx + c = 0 \quad \Rightarrow \quad x = \frac{-b \mp \sqrt{b^2 - 4ac}}{2a}$$

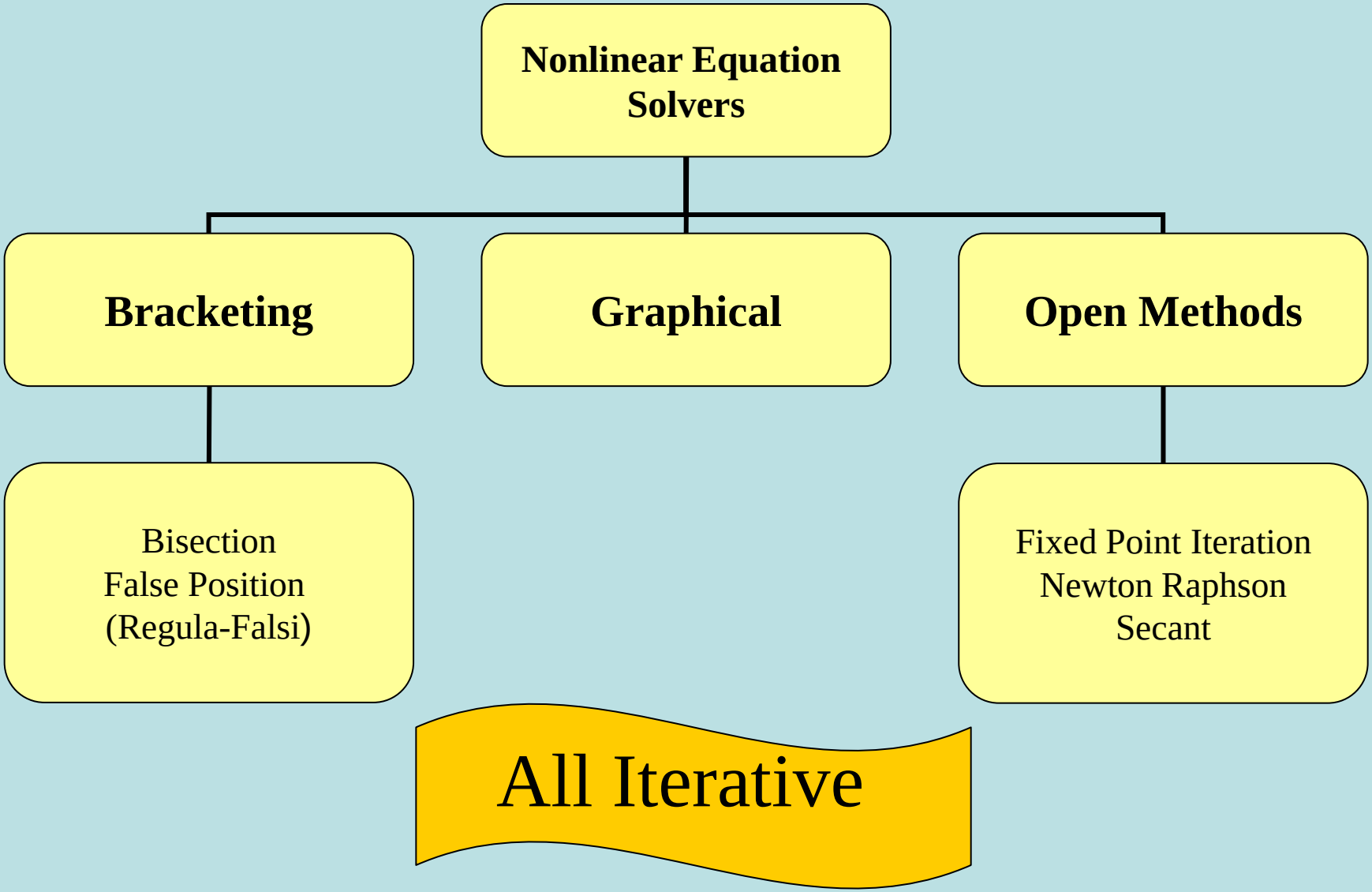
- But, not easy

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0 \quad \Rightarrow \quad x = ?$$

- How about these?

$$\sin x + x = 0 \quad \Rightarrow x = ?$$

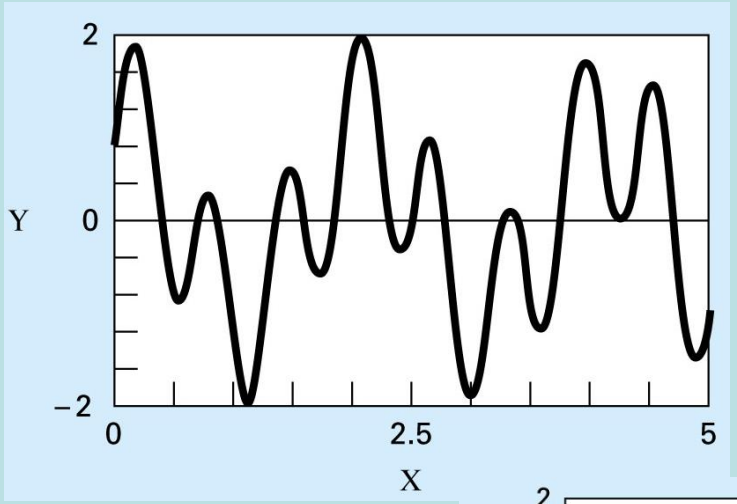
$$\cos(10x) + \sin(3x) = 0 \quad \Rightarrow x = ?$$



Graphical Approach

- Make a plot of the function f(x) and observe where it crosses the x-axis, i.e. f(x) = 0
- Not very practical but can be used to obtain rough estimates for roots
- These estimates can be used as initial guesses for numerical methods that we'll study here.

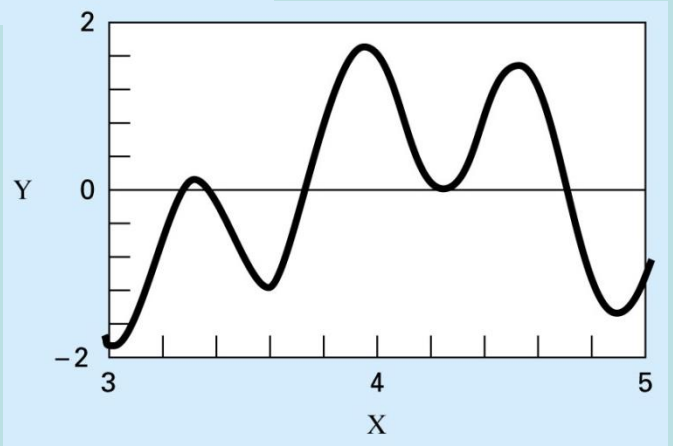
Using MATLAB, plot $f(x)=\sin(10x)+\cos(3x)$



Two distinct roots between

x= 4.2 and 4.3

need to be careful



MANY-MANY roots. What do we do?

Figure 5.4a

f(x)=sin 10x+cos 3x

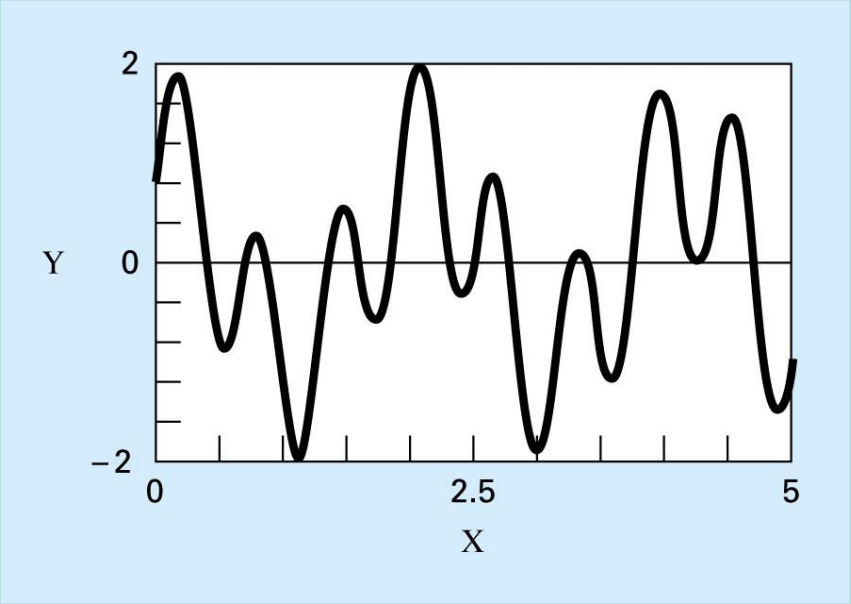


Figure 5.4b

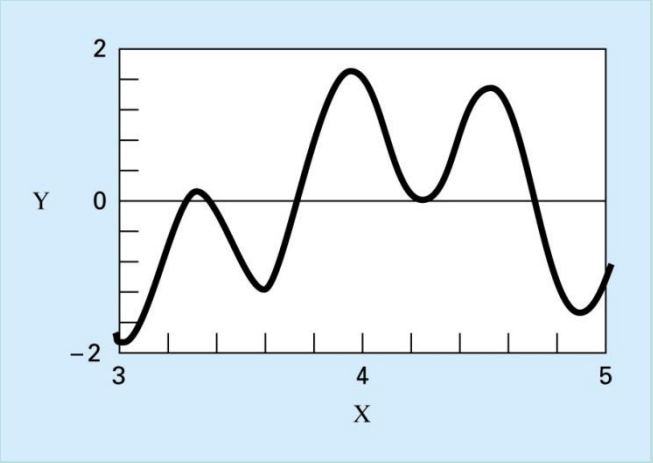
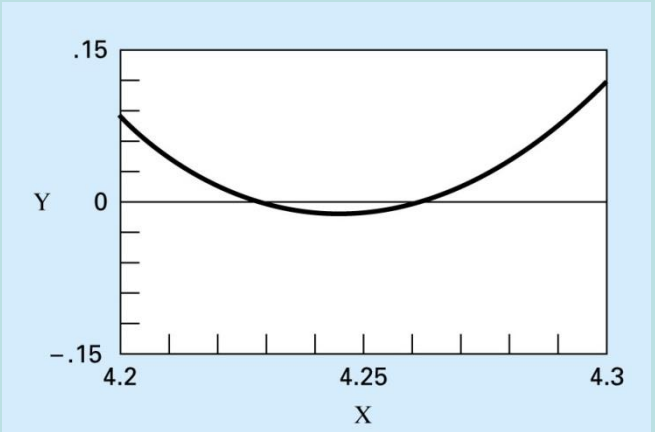
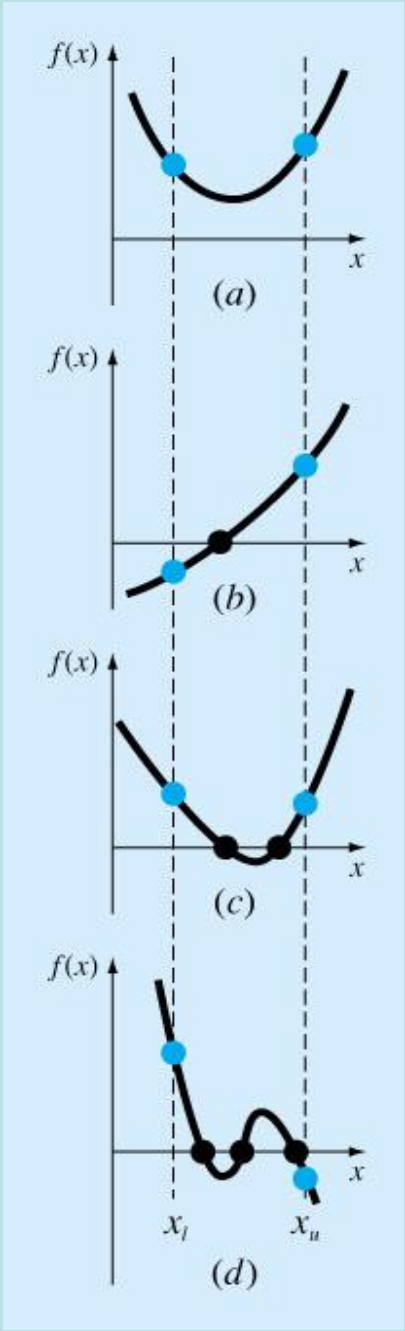


Figure 5.4c

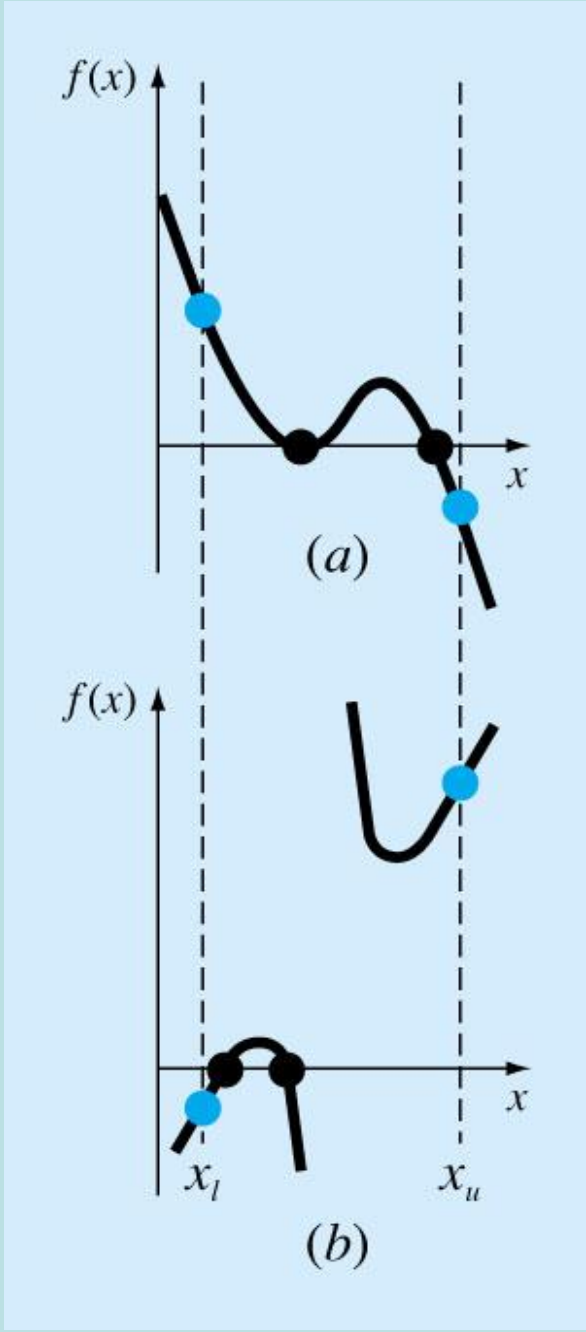


Bracketing:

Odd and even number of roots



exceptions



Bisection Method

Step 1: Choose lower x_l and upper x_u guesses for the root such that the function changes sign over the interval. This can be checked by ensuring that $f(x_l)f(x_u) < 0$.

Step 2: An estimate of the root x_r is determined by

$$x_r = \frac{x_l + x_u}{2}$$

Step 3: Make the following evaluations to determine in which subinterval the root lies:

(a) If $f(x_l)f(x_r) < 0$, the root lies in the lower subinterval. Therefore, set $x_u = x_r$ and return to step 2.

(b) If $f(x_l)f(x_r) > 0$, the root lies in the upper subinterval. Therefore, set $x_l = x_r$ and return to step 2.

(c) If $f(x_l)f(x_r) = 0$, the root equals x_r; terminate the computation.

$$\text{Relative error estimate : } \quad \varepsilon = \frac{\left|x_r^{new} - x_r^{old}\right|}{\left|x_r^{new}\right|} 100\%$$

Termination criteria: $\varepsilon < \varepsilon_{tol}$ OR *Max.Iteration* is reached

MATLAB code

Bisection Method

- **Minimize function evaluations in the code.**

  Why?

- **Because they are costly (takes more time)**

```
% Bisection Method - simple
% function f(x) = exp(-x) - x = 0   sample call:  bisection(-2, 4, 0.001,500)

function  root = bisection(xl, xu, es, imax);

if ((exp(-xl) - xl)*(exp(-xu) - xu))>0       % if guesses do not bracket, exit
    disp('no bracket')
    return
end

for i=1:1:imax

   xr=(xu+xl)/2;                  % compute the midpoint   xr
   ea = abs((xu-xl)/xl);          % approx. relative error

   test= (exp(-xl) - xl) * (exp(-xr) - xr);    % compute    f(xl)*f(xr)

   if (test < 0)    xu=xr;
   else    xl=xr;
   end

   if (test == 0) ea=0;   end
   if (ea < es)  break;   end

end

s=sprintf('\n  Root= %f    #Iterations = %d \n', xr,i); disp(s);
```

How Many Iterations will It Take?

- Length of the first Interval $L_o = x_u - x_l$
- After 1 iteration $L_1 = L_o/2$
- After 2 iterations $L_2 = L_o/4$

….. …..

- After k iterations $L_k = L_o/2^k$
- Then we can write: $\varepsilon_a \le \frac{L_k}{x} \times 100\%$ $\varepsilon_a \le \varepsilon_{es}$

$$|L_k| \le Absolute\ error$$

$$\left|\frac{L_0}{2^k}\right| \le |E_a|$$

$$2^k \ge \left|\frac{L_0}{E_a}\right| \Rightarrow k \ge \log_2\left(\left|\frac{L_0}{E_a}\right|\right)$$

$$\therefore k = int\left(\frac{\log(L_0) - \log(E_a)}{\log(2)}\right)$$

Example 1

- If the absolute magnitude of the error is

$$\frac{\varepsilon_{es} \cdot x}{100\%} = 10^{-4}$$

and L_o=2, how many iterations will you have to do to get the required accuracy in the solution?

$$10^{-4} = \frac{2}{2^k} \quad \Rightarrow 2^k = 2 \times 10^4 \quad \Rightarrow k \cong 14.3 = 15$$

Example 2

A ball has a specific gravity of 0.6 and has a radius of 5.5 cm. What is the distance to which the ball will get submerged when floating in water.

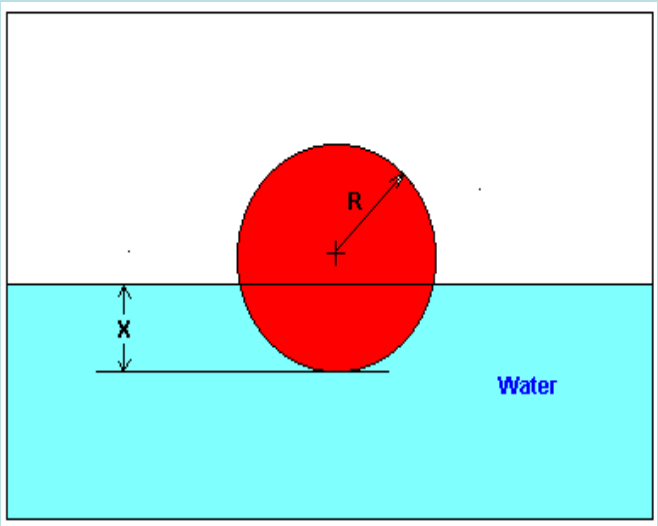


Figure 6 Diagram of the floating ball

Example 1 Cont.

The equation that gives the depth *x* to which the ball is submerged under water is given by

$$x^3 - 0.165x^2 + 3.993\times10^{-4} = 0$$

a) Use the bisection method of finding roots of equations to find the depth *x* to which the ball is submerged under water. Conduct three iterations to estimate the root of the above equation.

b) Find the absolute relative approximate error at the end of each iteration, and the number of significant digits at least correct at the end of each iteration.

Example 1 Cont.

From the physics of the problem, the ball would be submerged between $x = 0$ and $x = 2R$,

where R = radius of the ball,

that is

$$0 \le x \le 2R$$

$$0 \le x \le 2(0.055)$$

$$0 \le x \le 0.11$$

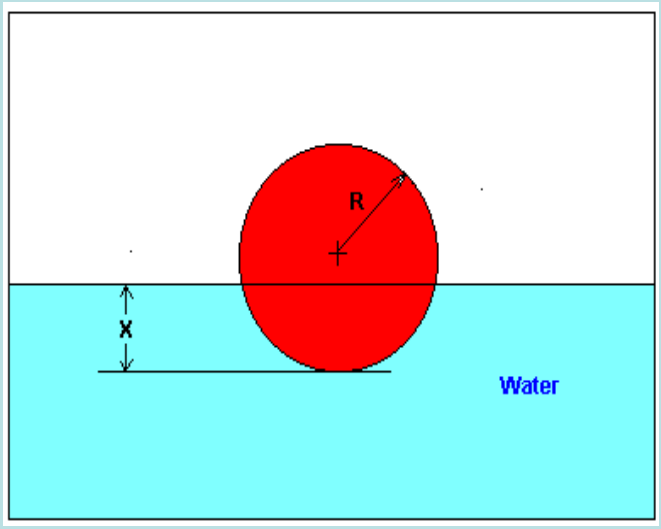


Figure 6 Diagram of the floating ball

Example 1 Cont.

Solution

To aid in the understanding of how this method works to find the root of an equation, the graph of f(x) is shown to the right,

where

$$f(x) = x^3 - 0.165x^2 + 3.993\times10^{-4}$$

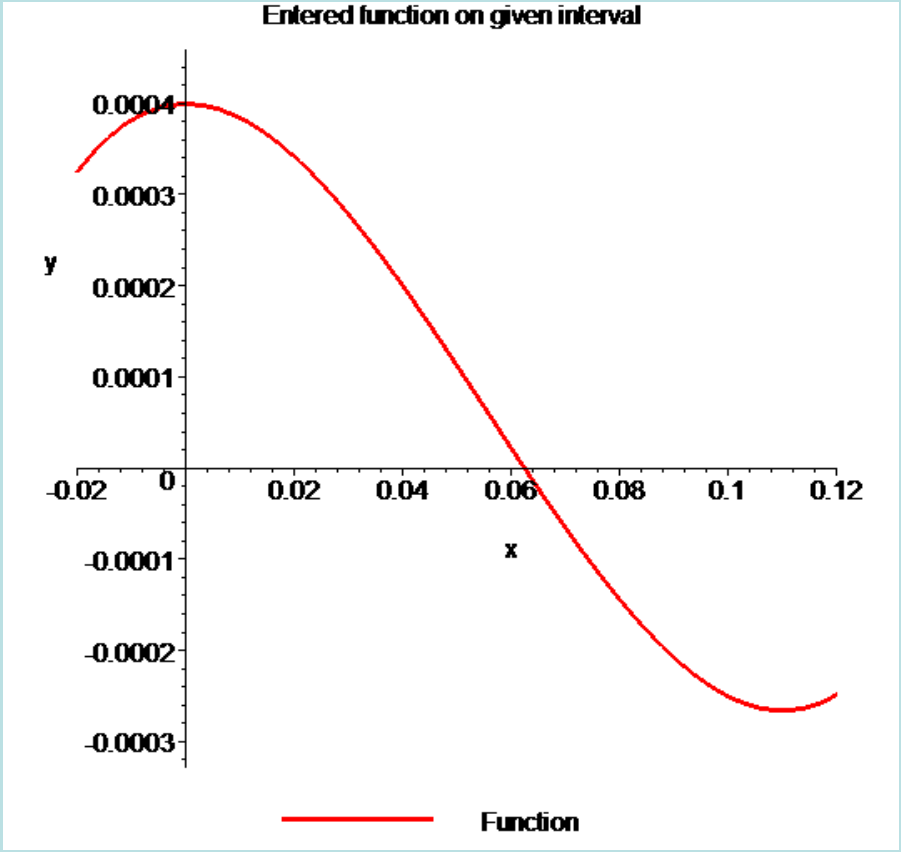


Figure 7 Graph of the function f(x)

Example 1 Cont.

Let us assume

$$x_\ell = 0.00$$

$$x_u = 0.11$$

Check if the function changes sign between x_ℓ and x_u .

$$f(x_l) = f(0) = (0)^3 - 0.165(0)^2 + 3.993\times10^{-4} = 3.993\times10^{-4}$$

$$f(x_u) = f(0.11) = (0.11)^3 - 0.165(0.11)^2 + 3.993\times10^{-4} = -2.662\times10^{-4}$$

Hence

$$f(x_l)f(x_u) = f(0)f(0.11) = \left(3.993\times10^{-4}\right)\left(-2.662\times10^{-4}\right) < 0$$

So there is at least on root between x_ℓ and x_u, that is between 0 and 0.11

Example 1 Cont.

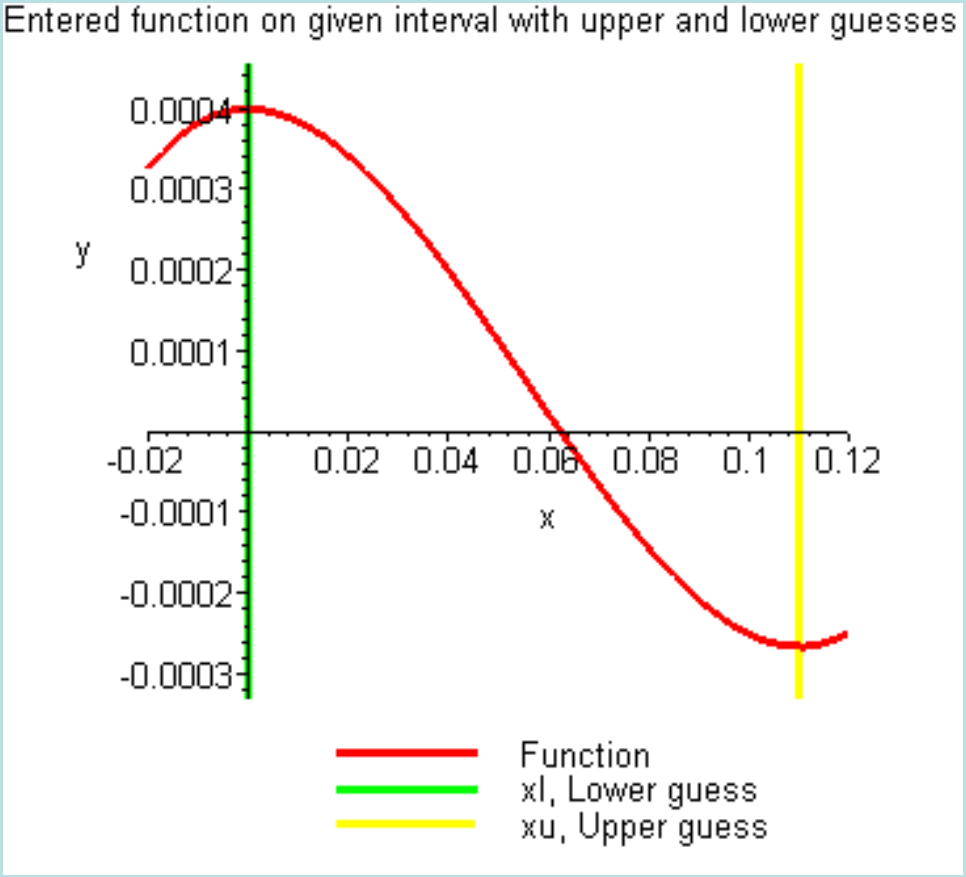


Figure 8 Graph demonstrating sign change between initial limits

Example 1 Cont.

Iteration 1

The estimate of the root is $x_r = \frac{x_\ell + x_u}{2} = \frac{0 + 0.11}{2} = 0.055$

$$f(x_r) = f(0.055) = (0.055)^3 - 0.165(0.055)^2 + 3.993\times10^{-4} = 6.655\times10^{-5}$$

$$f(x_l) f(x_r) = f(0) f(0.055) = \left(3.993\times10^{-4}\right)\left(6.655\times10^{-5}\right) > 0$$

Hence the root is bracketed between x_r and x_u, that is, between 0.055 and 0.11. So, the lower and upper limits of the new bracket are

$$x_l = 0.055,\ x_u = 0.11$$

At this point, the absolute relative approximate error $|\in_a|$ cannot be calculated as we do not have a previous approximation.

Example 1 Cont.

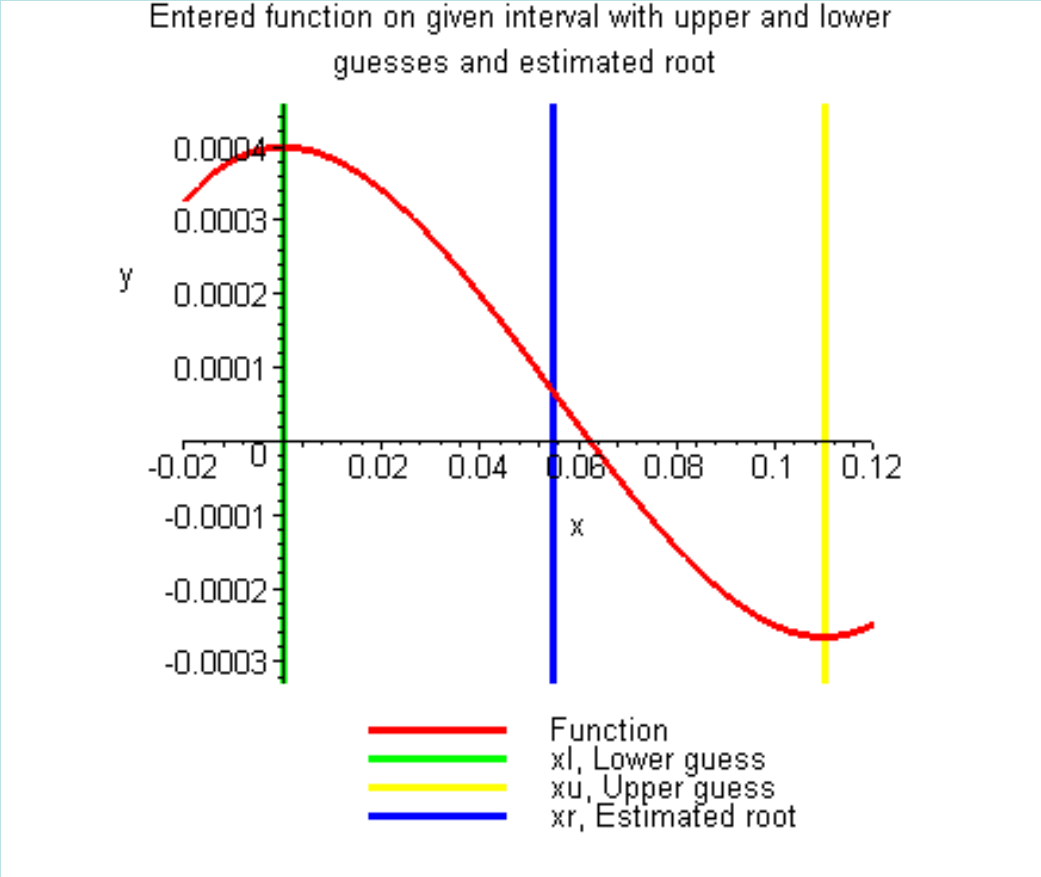


Figure 9 Estimate of the root for Iteration 1

Example 1 Cont.

Iteration 2

The estimate of the root is $x_r = \dfrac{x_\ell + x_u}{2} = \dfrac{0.055 + 0.11}{2} = 0.0825$

$$f(x_r) = f(0.0825) = (0.0825)^3 - 0.165(0.0825)^2 + 3.993\times10^{-4} = -1.622\times10^{-4}$$

$$f(x_l)f(x_r) = f(0.055)f(0.0825) = \left(-1.622\times10^{-4}\right)\left(6.655\times10^{-5}\right) < 0$$

Hence the root is bracketed between x_ℓ and x_r, that is, between 0.055 and 0.0825. So, the lower and upper limits of the new bracket are

$$x_l = 0.055,\ x_u = 0.0825$$

Example 1 Cont.

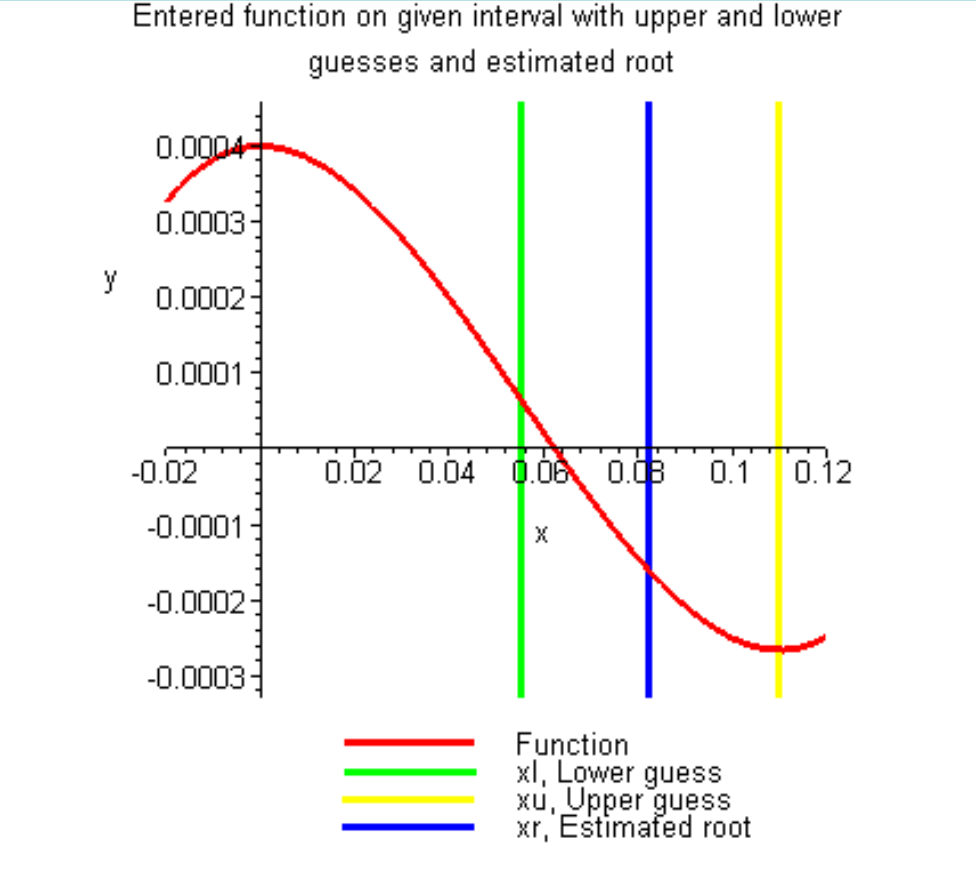


Figure 10 Estimate of the root for Iteration 2

Example 1 Cont.

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 2 is

$$|\epsilon_a| = \left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| \times 100$$

$$= \left|\frac{0.0825 - 0.055}{0.0825}\right| \times 100$$

$$= 33.333\%$$

None of the significant digits are at least correct in the estimate root of $x_r = 0.0825$ because the absolute relative approximate error is greater than 5%.

Example 1 Cont.

Iteration 3

The estimate of the root is $x_r = \dfrac{x_\ell + x_u}{2} = \dfrac{0.055 + 0.0825}{2} = 0.06875$

$$f(x_r) = f(0.06875) = (0.06875)^3 - 0.165(0.06875)^2 + 3.993\times10^{-4} = -5.563\times10^{-5}$$

$$f(x_l)f(x_r) = f(0.055)f(0.06875) = (6.655\times10^{-5})(-5.563\times10^{-5}) < 0$$

Hence the root is bracketed between x_ℓ and x_r, that is, between 0.055 and 0.06875. So, the lower and upper limits of the new bracket are

$$x_l = 0.055,\ x_u = 0.06875$$

Example 1 Cont.

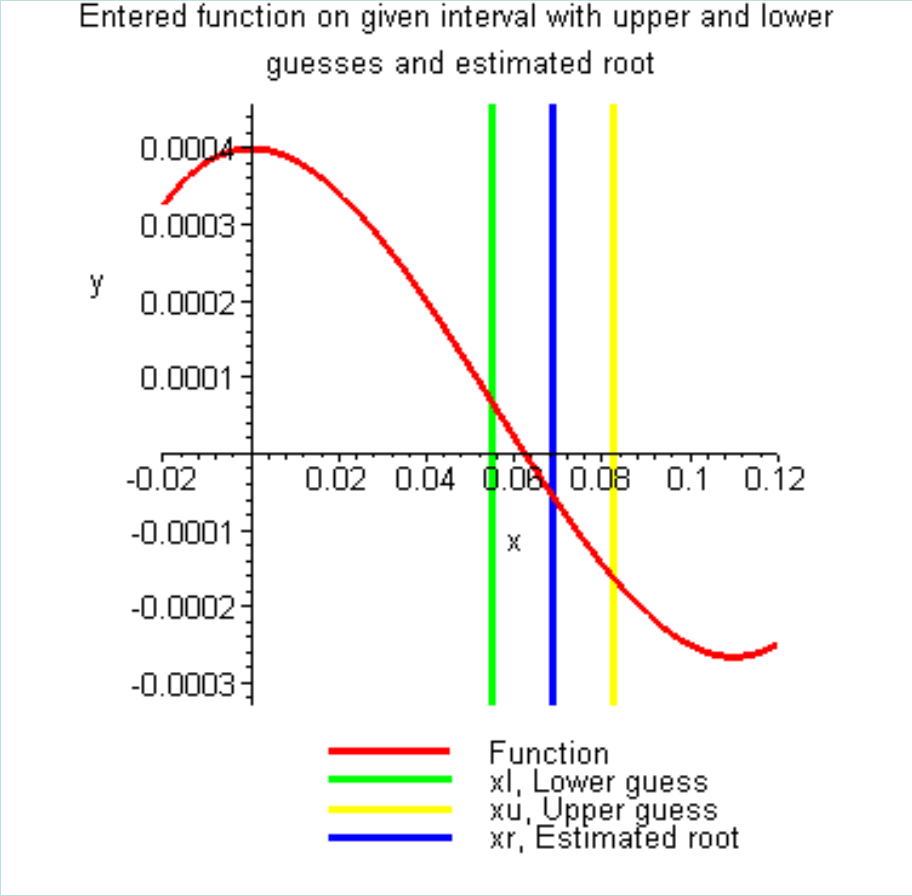


Figure 11 Estimate of the root for Iteration 3

Example 1 Cont.

The absolute relative approximate error $|\epsilon_a|$ at the end of Iteration 3 is

$$|\epsilon_a| = \left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| \times 100$$

$$= \left|\frac{0.06875 - 0.0825}{0.06875}\right| \times 100$$

$$= 20\%$$

Still none of the significant digits are at least correct in the estimated root of the equation as the absolute relative approximate error is greater than 5%.

Seven more iterations were conducted and these iterations are shown in Table 1.

Table 1 Cont.

Table 1 Root of f(x)=0 as function of number of iterations for bisection method.

Iteration	x_ℓ	x_u	X_r	$\lvert\epsilon_a\rvert\%$	$f(x_r)$
1	0.00000	0.11	0.055	----------	6.655×10^{-5}
2	0.055	0.11	0.0825	33.33	-1.622×10^{-4}
3	0.055	0.0825	0.06875	20.00	-5.563×10^{-5}
4	0.055	0.06875	0.06188	11.11	4.484×10^{-6}
5	0.06188	0.06875	0.06531	5.263	-2.593×10^{-5}
6	0.06188	0.06531	0.06359	2.702	-1.0804×10^{-5}
7	0.06188	0.06359	0.06273	1.370	-3.176×10^{-6}
8	0.06188	0.06273	0.0623	0.6897	6.497×10^{-7}
9	0.0623	0.06273	0.06252	0.3436	-1.265×10^{-6}
10	0.0623	0.06252	0.06241	0.1721	-3.0768×10^{-7}

Table 1 Cont.

Hence the number of significant digits at least correct is given by the largest value or m for which

$$\left|\in_a\right| \le 0.5\times 10^{2-m}$$

$$0.1721 \le 0.5\times 10^{2-m}$$

$$0.3442 \le 10^{2-m}$$

$$\log(0.3442) \le 2-m$$

$$m \le 2-\log(0.3442) = 2.463$$

So

$$m = 2$$

The number of significant digits at least correct in the estimated root of 0.06241 at the end of the 10th iteration is 2.

Bisection Method

Pros

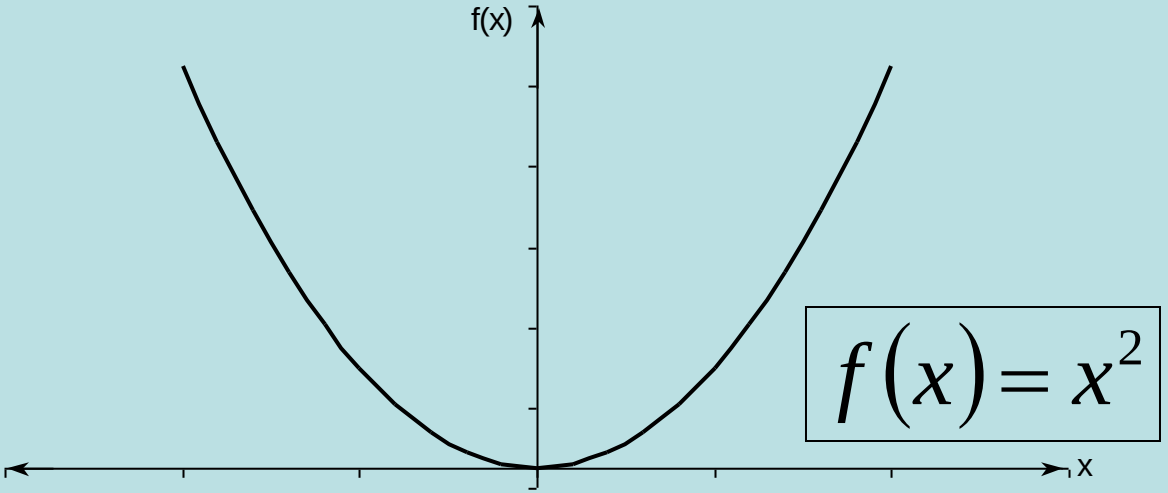
- Easy
- Always finds a root
- Number of iterations required to attain an absolute error can be computed a priori.

Cons

- Slow
- Need to find initial guesses for x_l and x_u
- No account is taken of the fact that if $f(x_l)$ is closer to zero, it is likely that root is closer to x_l .

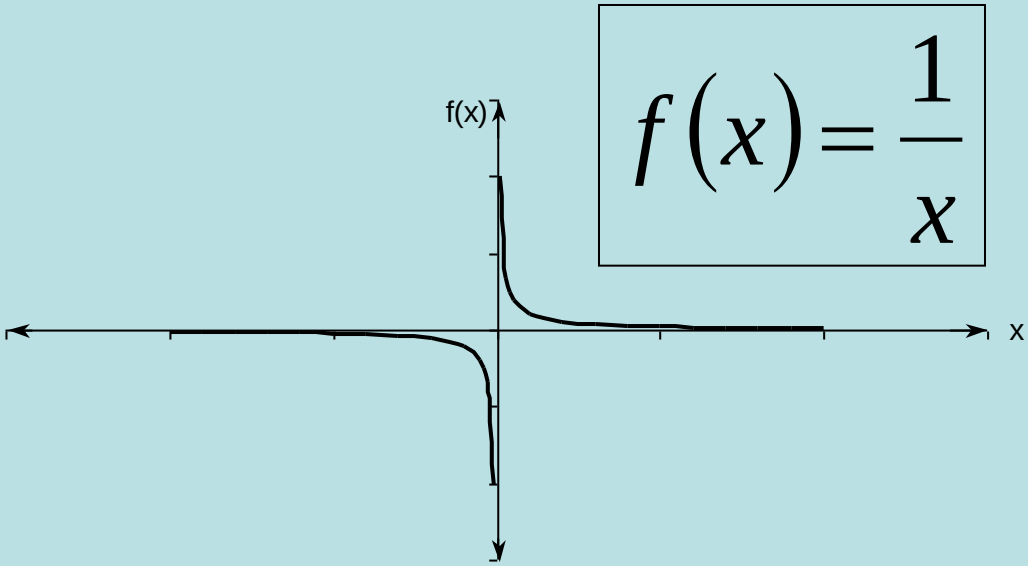
Drawbacks (continued)

- If a function f(x) is such that it just touches the x-axis it will be unable to find the lower and upper guesses.



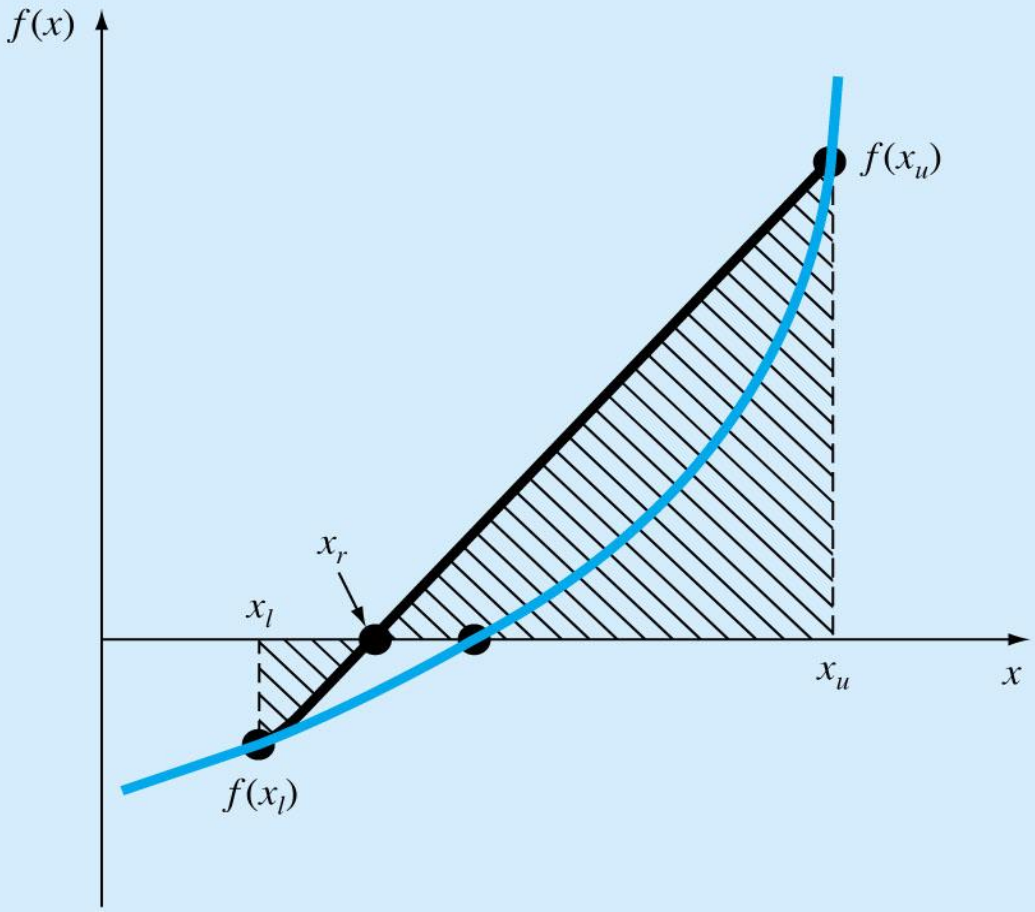
Drawbacks (continued)

- Function changes sign but root does not exist



The False-Position Method (Regula-Falsi)

- We can approximate the solution by doing a *linear interpolation* between $f(x_u)$ and $f(x_l)$
- Find x_r such that $l(x_r)=0$, where $l(x)$ is the linear approximation of $f(x)$ between x_l and x_u
- Derive x_r using similar triangles



$$x_r = \frac{x_l f_u - x_u f_l}{f_u - f_l}$$

Procedure

1. Find a pair of values of x, x_l and x_u such that $f_l=f(x_l) <0$ and $f_u=f(x_u) >0$.
2. Estimate the value of the root from the following formula

$$x_r = \frac{x_l f_u - x_u f_l}{f_u - f_l}$$

and evaluate $f(x_r)$.

3. Use the new point to replace one of the original points, keeping the two points on opposite sides of the x axis.

If $f(x_r)<0$ then $x_l=x_r$ == > $f_l=f(x_r)$

If $f(x_r)>0$ then $x_u=x_r$ == > $f_u=f(x_r)$

If $f(x_r)=0$ then you have found the root and need go no further!

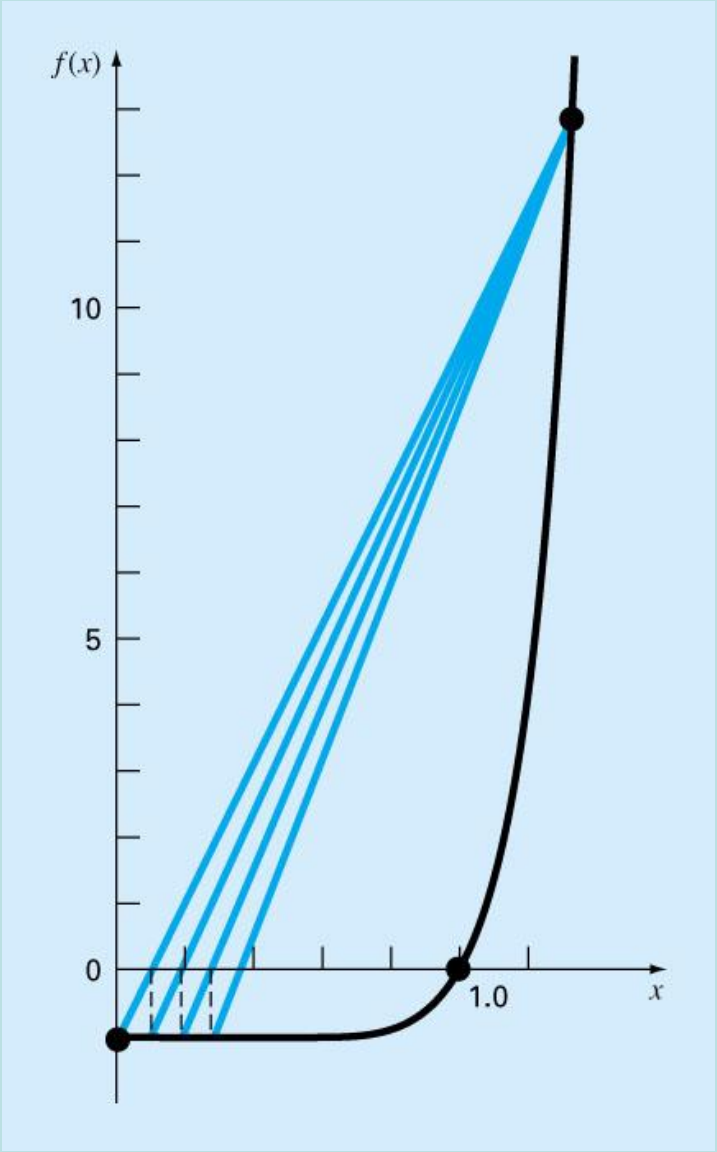
4. See if the new x_l and x_u are close enough for convergence to be declared. If they are not go back to step 2.

- Why this method?
  - Faster
  - Always converges for a single root.

➔Pitfalls of the False-Position Method

Note: Always check by substituting estimated root in the original equation to determine whether $f(x_r) \approx 0$.

The False-Position Method



Works well, but not always!

🡸🡸 Here is a pitfall ☹

Modified False-Position

One way to mitigate the "one-sided" nature of the **false position** (i.e. the pitfall case) is to have the algorithm detect when one of the bounds is stuck.

If this occurs, then the original formula $\mathbf{x_r = (x_l + x_u)/2}$ can be used

False-position (Regula-Falsi)

Linear Interpolation Method

```
function [x,y] = false_position(func)

% Find root near x1 using the false position method.
% Input:  func      string containing name of function
%         xl,xu     initial guesses
%         es        allowable tolerance in computed root
%         maxit     maximum number of iterations
% Output: x         row vector of approximations to root

xl = input('enter lower bound xl = ');
xu = input('enter upper bound xu = ');
es = input('allowable tolerance es = ');
maxit = input('maximum number of iterations maxit = ');

a(1) = xl; b(1) = xu;
ya(1) = feval(func, a(1)); yb(1)=feval(func, b(1));
if ya(1) * yb(1) > 0.0
     error('Function has same sign at end points')
end
for i = 1:maxit
     x(i) = b(i) - yb(i)*(b(i)-a(i))/(yb(i)-ya(i));
     y(i) = feval(func, x(i));
     if y(i) == 0.0
        disp('exact zero found'); break;
     elseif y(i) * ya(i) < 0
        a(i+1) = a(i); ya(i+1) = ya(i);
        b(i+1) = x(i); yb(i+1) = y(i);
     else
        a(i+1) = x(i); ya(i+1) = y(i);
        b(i+1) = b(i); yb(i+1) = yb(i);
     end;
      if((i > 1) & (abs(x(i) - x(i-1)) < es))
          disp('False position method has converged'); break
      end
     iter = i;
end
if(iter >= maxit)
        disp('zero not found to desired tolerance');
end
n=length(x); k=1:n; out = [k'  a(1:n)'  b(1:n)'  x'  y'];
disp('      step      xl          xu          xr          f(xr)')
disp(out)
```

Linear interpolation

False-Position (Linear Interpolation) Method Manning Equation

```
>> false_position('manning')
enter lower bound xl = 0
enter upper bound xu = 10
allowable tolerance es = 0.00001
maximum number of iterations maxit = 50
False position method has converged
```

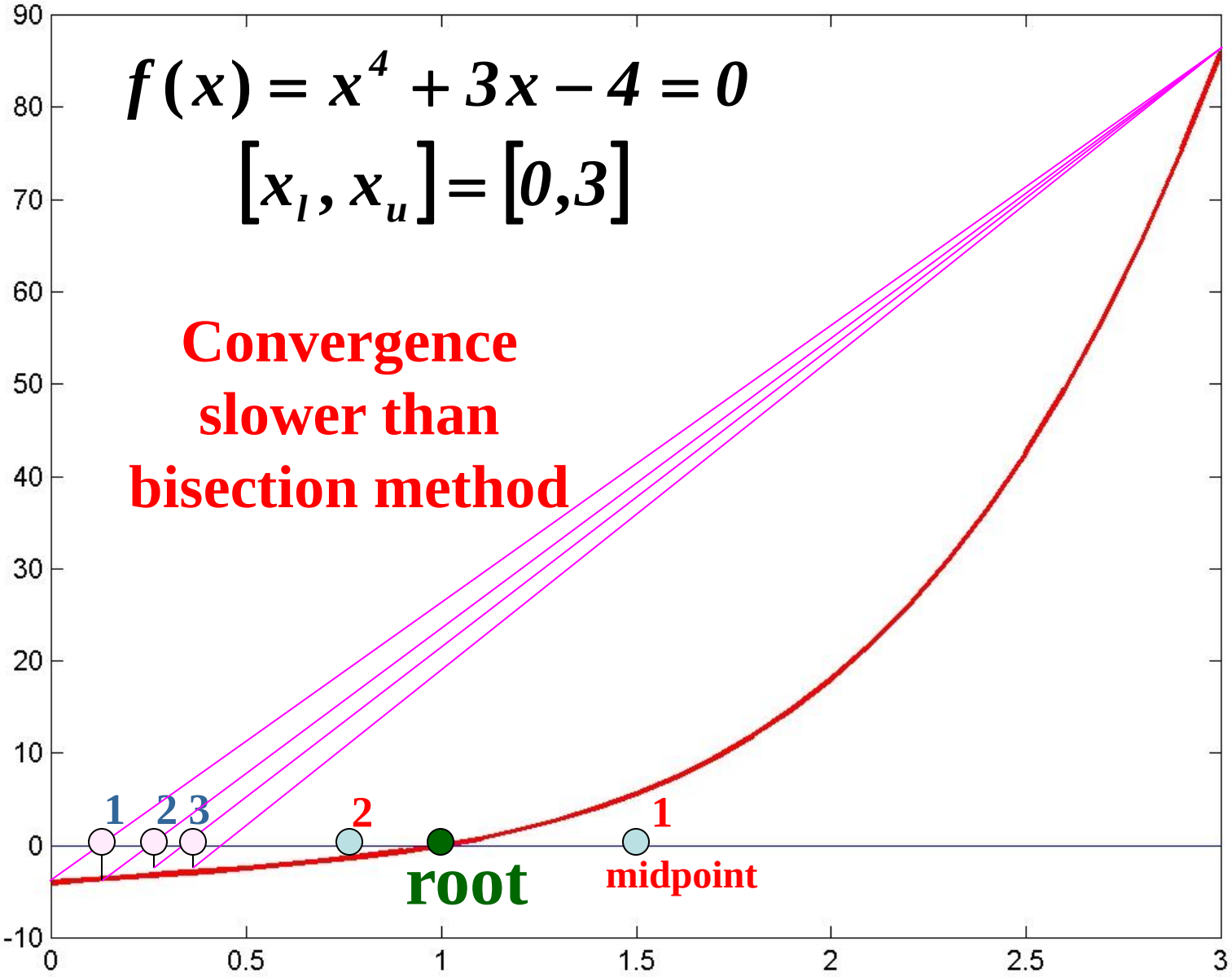
step	xl	xu	xr	f(xr)
1.0000	0	10.0000	4.9661	271.4771
2.0000	4.9661	10.0000	6.0295	27.5652
3.0000	6.0295	10.0000	6.1346	2.4677
4.0000	6.1346	10.0000	6.1440	0.2184
5.0000	6.1440	10.0000	6.1449	0.0193
6.0000	6.1449	10.0000	6.1449	0.0017
7.0000	6.1449	10.0000	6.1449	0.0002

- Much faster convergence than the bisection method
- May be slower than bisection method for some cases

Convergence Rate

- Why don't we always use false position method?
- There are times it may converge very, very slowly.
- Example:

$$f(x) = x^4 + 3x - 4 = 0$$



Bisection Method

```
» xl = 0; xu = 3; es = 0.00001; maxit = 100;
» [xr,fr]=bisect2(inline('x^4+3*x-4'))
Bisection method has converged
```

step	xl	xu	xr	f(x)
1.0000	0	3.0000	1.5000	5.5625
2.0000	0	1.5000	0.7500	-1.4336
3.0000	0.7500	1.5000	1.1250	0.9768
4.0000	0.7500	1.1250	0.9375	-0.4150
5.0000	0.9375	1.1250	1.0312	0.2247
6.0000	0.9375	1.0312	0.9844	-0.1079
7.0000	0.9844	1.0312	1.0078	0.0551
8.0000	0.9844	1.0078	0.9961	-0.0273
9.0000	0.9961	1.0078	1.0020	0.0137
10.0000	0.9961	1.0020	0.9990	-0.0068
11.0000	0.9990	1.0020	1.0005	0.0034
12.0000	0.9990	1.0005	0.9998	-0.0017
13.0000	0.9998	1.0005	1.0001	0.0009
14.0000	0.9998	1.0001	0.9999	-0.0004
15.0000	0.9999	1.0001	1.0000	0.0002
16.0000	0.9999	1.0000	1.0000	-0.0001
17.0000	1.0000	1.0000	1.0000	0.0001
18.0000	1.0000	1.0000	1.0000	0.0000
19.0000	1.0000	1.0000	1.0000	0.0000

$$f(x) = x^4 + 3x - 4 = 0$$

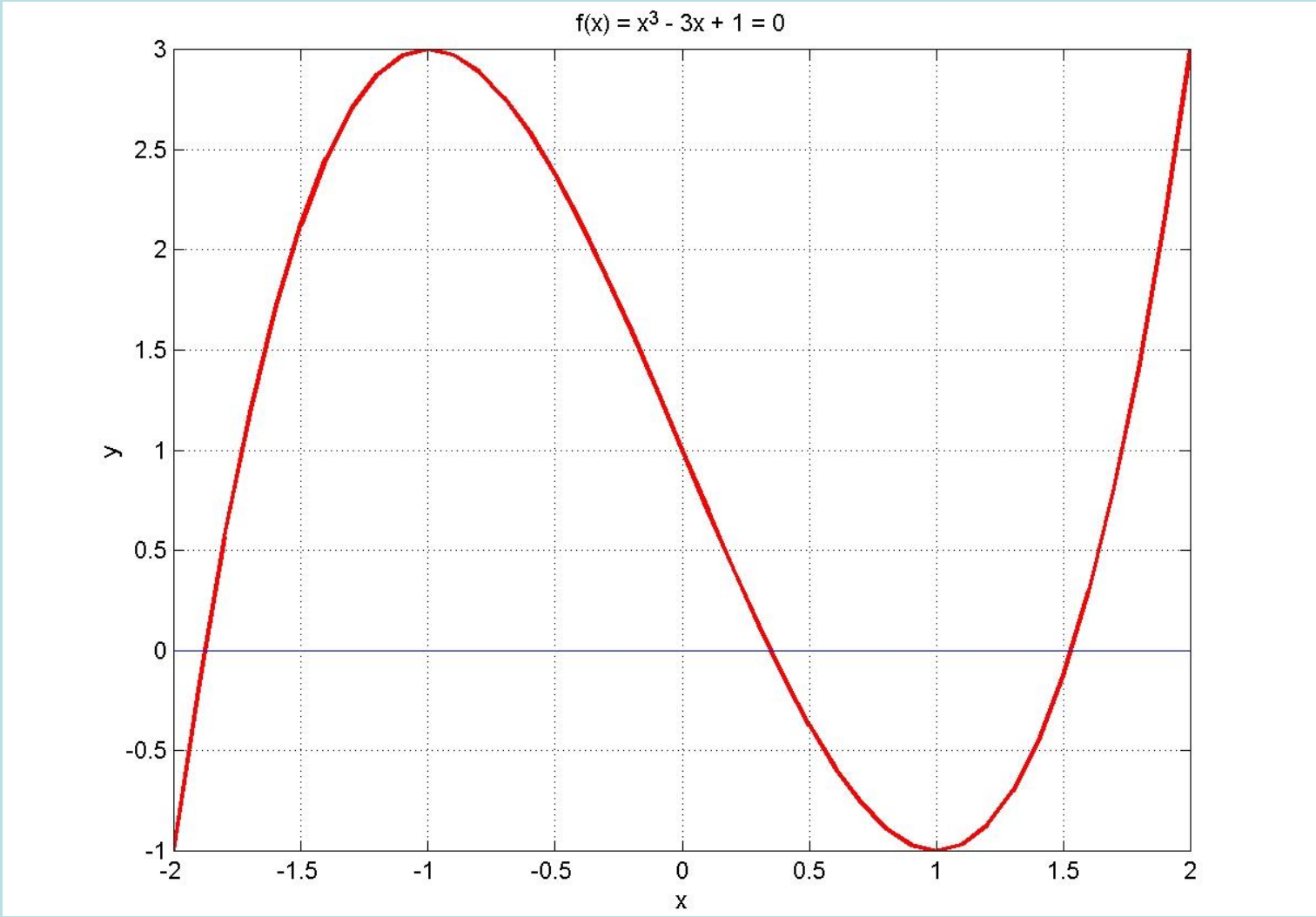
False-Position Method

```
» xl = 0; xu = 3; es = 0.00001; maxit = 100;
» [xr,fr]=false_position(inline('x^4+3*x-4'))
False position method has converged
```

step	xl	xu	xr	f(xr)
1.0000	0	3.0000	0.1333	-3.5997
2.0000	0.1333	3.0000	0.2485	-3.2507
3.0000	0.2485	3.0000	0.3487	-2.9391
4.0000	0.3487	3.0000	0.4363	-2.6548
5.0000	0.4363	3.0000	0.5131	-2.3914
6.0000	0.5131	3.0000	0.5804	-2.1454
7.0000	0.5804	3.0000	0.6393	-1.9152
8.0000	0.6393	3.0000	0.6907	-1.7003
9.0000	0.6907	3.0000	0.7355	-1.5010
10.0000	0.7355	3.0000	0.7743	-1.3176
11.0000	0.7743	3.0000	0.8079	-1.1503
12.0000	0.8079	3.0000	0.8368	-0.9991
13.0000	0.8368	3.0000	0.8617	-0.8637
14.0000	0.8617	3.0000	0.8829	-0.7434
15.0000	0.8829	3.0000	0.9011	-0.6375
16.0000	0.9011	3.0000	0.9165	-0.5448
17.0000	0.9165	3.0000	0.9296	-0.4642
18.0000	0.9296	3.0000	0.9408	-0.3945
19.0000	0.9408	3.0000	0.9502	-0.3345
20.0000	0.9502	3.0000	0.9581	-0.2831
		....		
40.0000	0.9985	3.0000	0.9988	-0.0086
		....		
58.0000	0.9999	3.0000	0.9999	-0.0004

Example: Rate of Convergence

```
» x = -2:0.1:2; y = x.^3-3*x+1; z = x*0;
» H = plot(x,y,'r',x,z,'b'); grid on; set(H,'LineWidth',3.0);
» xlabel('x'); ylabel('y'); title('f(x) = x^3 - 3x + 1 = 0');
```



Comparison of rate of convergence for bisection and false-position method

```
>> bisect2(inline('x^3-3*x+1'))
enter lower bound xl = 0
enter upper bound xu = 1
allowable tolerance es = 1.e-20
maximum number of iterations maxit = 100
exact zero found
```

step	xl	xu	xr	f(xr)
1.0000	0	1.0000	0.5000	-0.3750
2.0000	0	0.5000	0.2500	0.2656
3.0000	0.2500	0.5000	0.3750	-0.0723
4.0000	0.2500	0.3750	0.3125	0.0930
5.0000	0.3125	0.3750	0.3438	0.0094
6.0000	0.3438	0.3750	0.3594	-0.0317
7.0000	0.3438	0.3594	0.3516	-0.0112
8.0000	0.3438	0.3516	0.3477	-0.0009
9.0000	0.3438	0.3477	0.3457	0.0042
10.0000	0.3457	0.3477	0.3467	0.0016
11.0000	0.3467	0.3477	0.3472	0.0003
12.0000	0.3472	0.3477	0.3474	-0.0003
13.0000	0.3472	0.3474	0.3473	0.0000
14.0000	0.3473	0.3474	0.3474	-0.0001
		.. .		
		.. .		
50.0000	0.3473	0.3473	0.3473	-0.0000
51.0000	0.3473	0.3473	0.3473	0.0000
52.0000	0.3473	0.3473	0.3473	-0.0000
53.0000	0.3473	0.3473	0.3473	-0.0000
54.0000	0.3473	0.3473	0.3473	0

Continued on next page

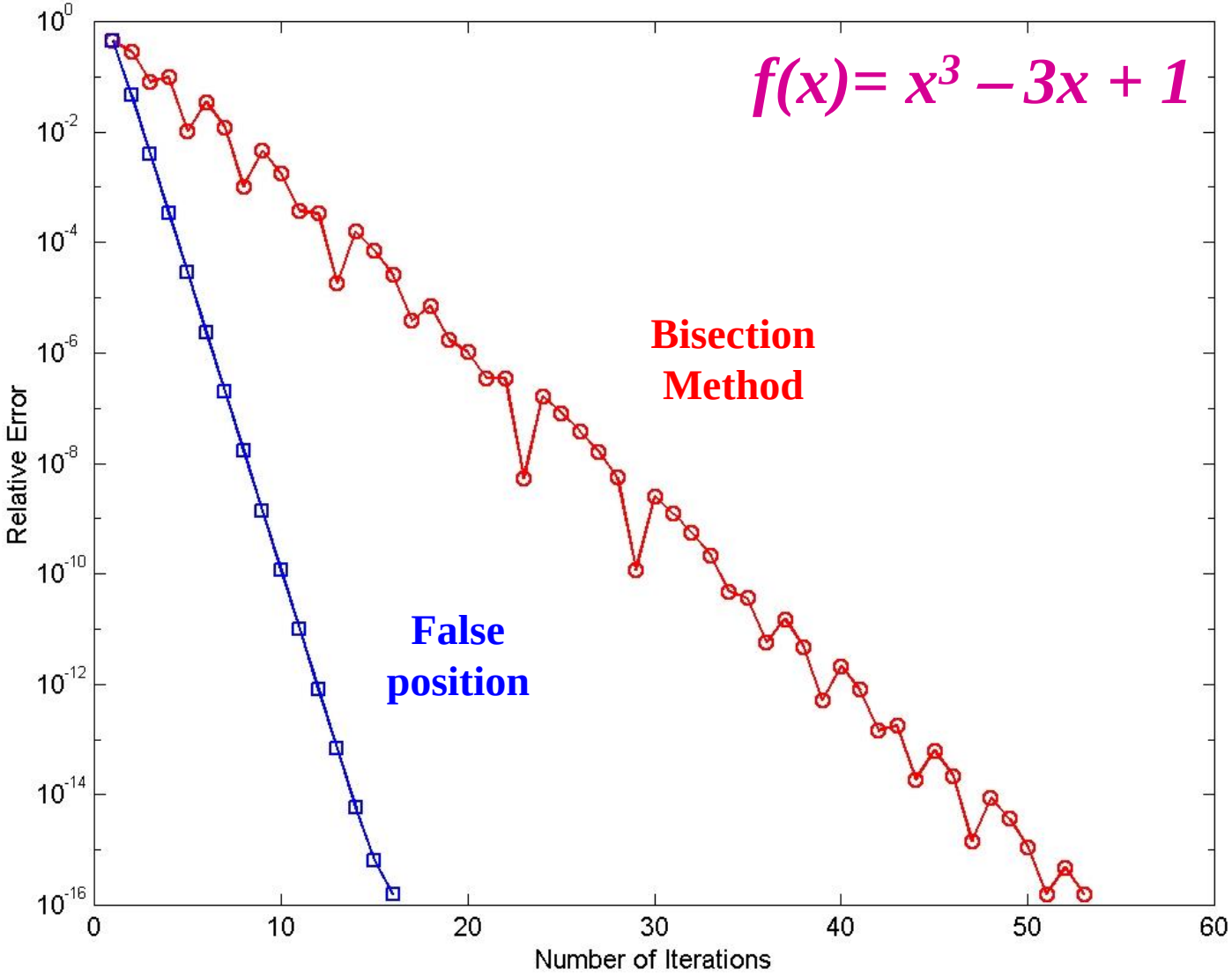
```
>> false_position(inline('x^3-3*x+1'))
enter lower bound xl = 0
enter upper bound xu = 1
allowable tolerance es = 1.e-20
maximum number of iterations maxit = 100
exact zero found
                    step       xl          xu          xr       f(xr)
                  1.0000          0      1.0000      0.5000     -0.3750
                  2.0000          0      0.5000      0.3636     -0.0428
                  3.0000          0      0.3636      0.3487     -0.0037
                  4.0000          0      0.3487      0.3474     -0.0003
                  5.0000          0      0.3474      0.3473     -0.0000
                  6.0000          0      0.3473      0.3473     -0.0000
                  7.0000          0      0.3473      0.3473     -0.0000
                  8.0000          0      0.3473      0.3473     -0.0000
                  9.0000          0      0.3473      0.3473     -0.0000
                 10.0000          0      0.3473      0.3473     -0.0000
                 11.0000          0      0.3473      0.3473     -0.0000
                 12.0000          0      0.3473      0.3473     -0.0000
                 13.0000          0      0.3473      0.3473     -0.0000
                 14.0000          0      0.3473      0.3473     -0.0000
                 15.0000          0      0.3473      0.3473     -0.0000
                 16.0000          0      0.3473      0.3473     -0.0000
                 17.0000          0      0.3473      0.3473           0
```

$$f(x) = x^3 - 3x + 1 = 0$$

Compute relative errors

```
 iter1=length(x1); iter2=length(x2); k1=1:iter1; k2=1:iter2;
>> root1=x1(iter1); root2=x2(iter2);
>> error1=abs((x1-root1)/root1); error2=abs((x2-root2)/root2);
>> H=semilogy(k1,error1,'ro-',k2,error2,'bs-'); set(H,'LineWidth',2.0);
>> xlabel('Number of Iterations'); ylabel('Relative Error');
```

Rate of Convergence



How to find good initial guesses?

- Start at one end of the region of interest (x_a) and evaluate $f(x_a)$, $f(x_a+\Delta x)$, $f(x_a+2\Delta x)$, $f(x_a+3\Delta x)$, ...

- Continue until the *sign* of the result changes.
  If that happens between $f(x_a+k*\Delta x)$ and $f(x_a+(k+1)*\Delta x)$

  then pick $x_l = x_a+k*\Delta x$ and $x_u = x_a+(k+1)*\Delta x$

Problem:

if Δx is too small ➔ search is very time consuming

if Δx is too large ➔ could jump over two closely spaced roots

Suggestions:

- Generate random x values and evaluate f(x) each time until you find two values that satisfy f(x1)*f(x2) < 0
- Know the application and plot the function to see the location of the roots, and pick x_l and x_u accordingly to start the iterations.