Bisection Method

Advantages

- Simple and easy to implement
- One function evaluation per iteration
- The size of the interval containing the zero is reduced by 50% after each iteration
- The number of iterations can be determined a priori
- No knowledge of the derivative is needed
- The function does not have to be differentiable

Disadvantage

- Slow to converge
- Good intermediate approximations may be discar

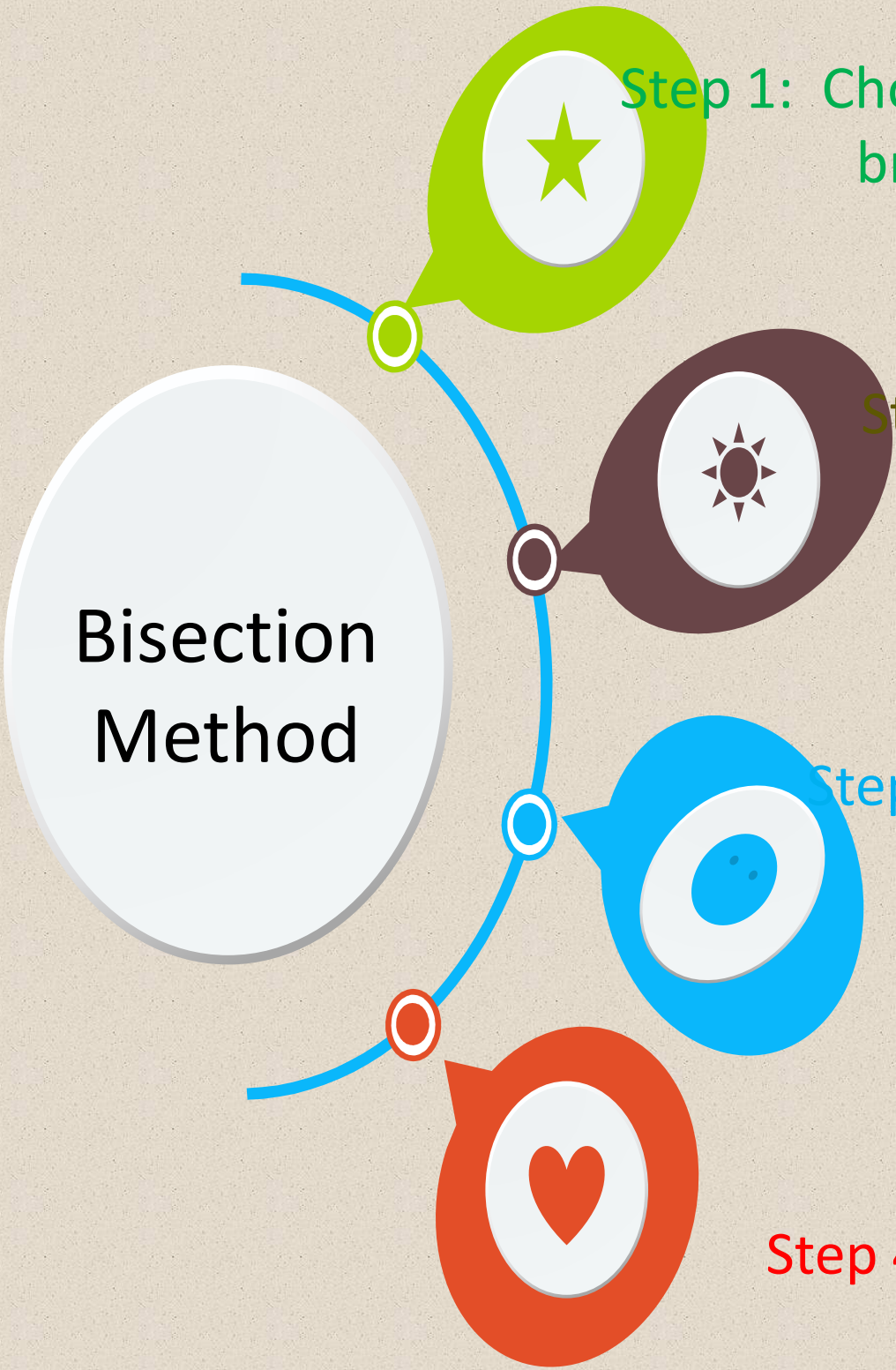
Bisection Method

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 .**



Step 1: Choose xl and xu such that xl and xu bracket the root, i.e. **f(xl)*f(xu) < 0.**

Step 2 : Estimate the root (bisection).

xr = 0.5*(xl + xu)

Step 3: Determine the new bracket.

If **f(xr)*f(xl) < 0** **xu = xr**

else **xl = xr** end

Step 4: Examine if **xr** is the root.

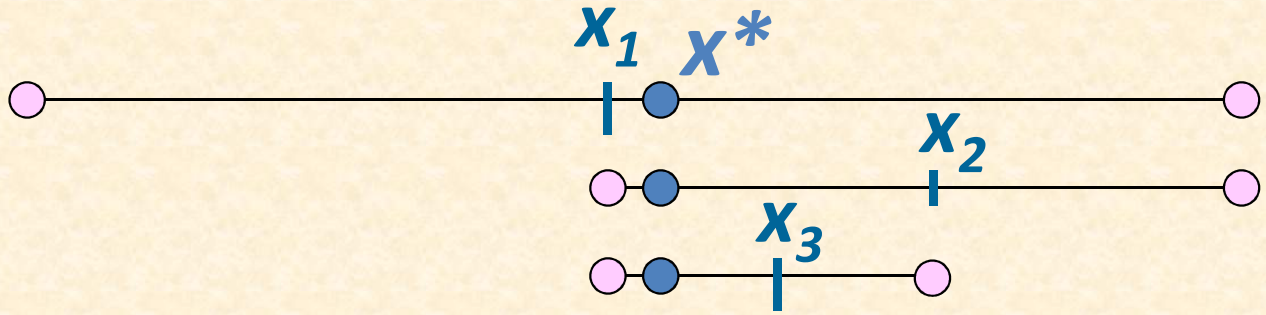
Bisection Method

5. If not Repeat steps 2 and 3 until convergence

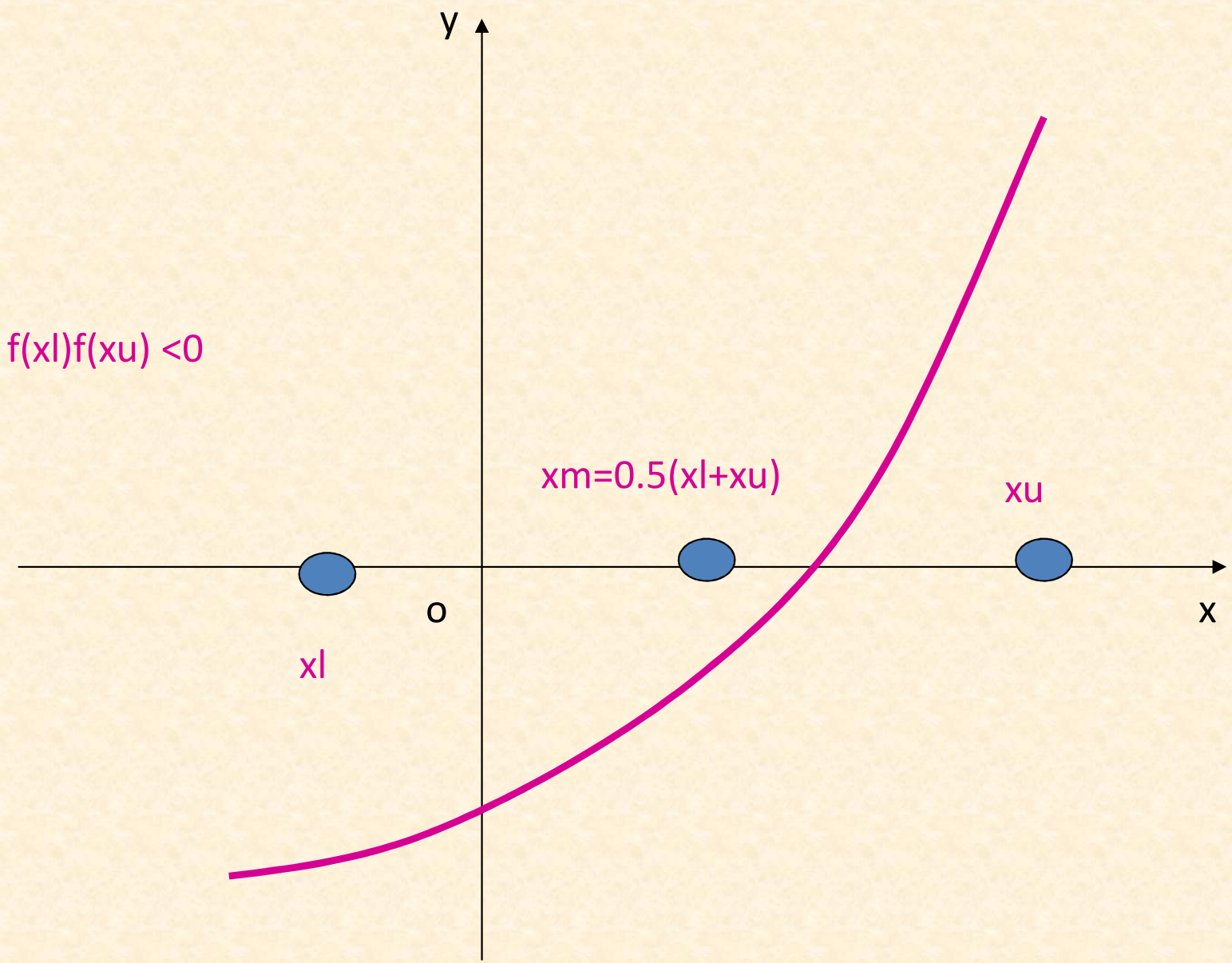
$$(a)\quad f(x_r) \approx 0,\ i.e.,\ \left|f(x_r)\right| \le \varepsilon$$

$$(b)\quad \varepsilon_a = \left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right|(100\%) \le \varepsilon_s \text{ in successive iterations}$$

$$(c)\quad \text{the maximum number of iterations has been reached}$$

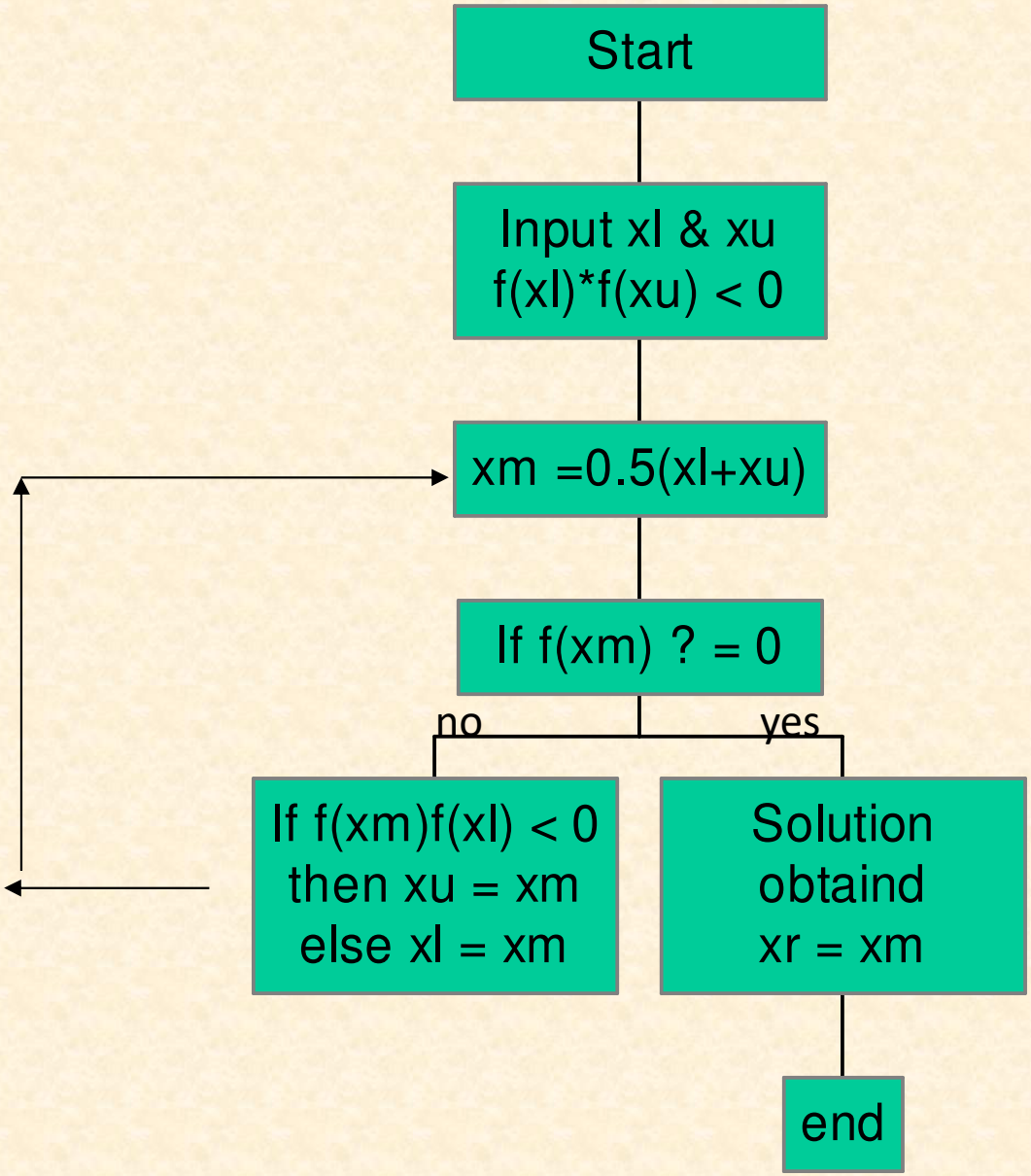


Non-monotonic convergence: x_1 is closer to the root than x_2 or x_3



Bisection Method

Bisection Flowchart



Mass of a Bungee Jumper

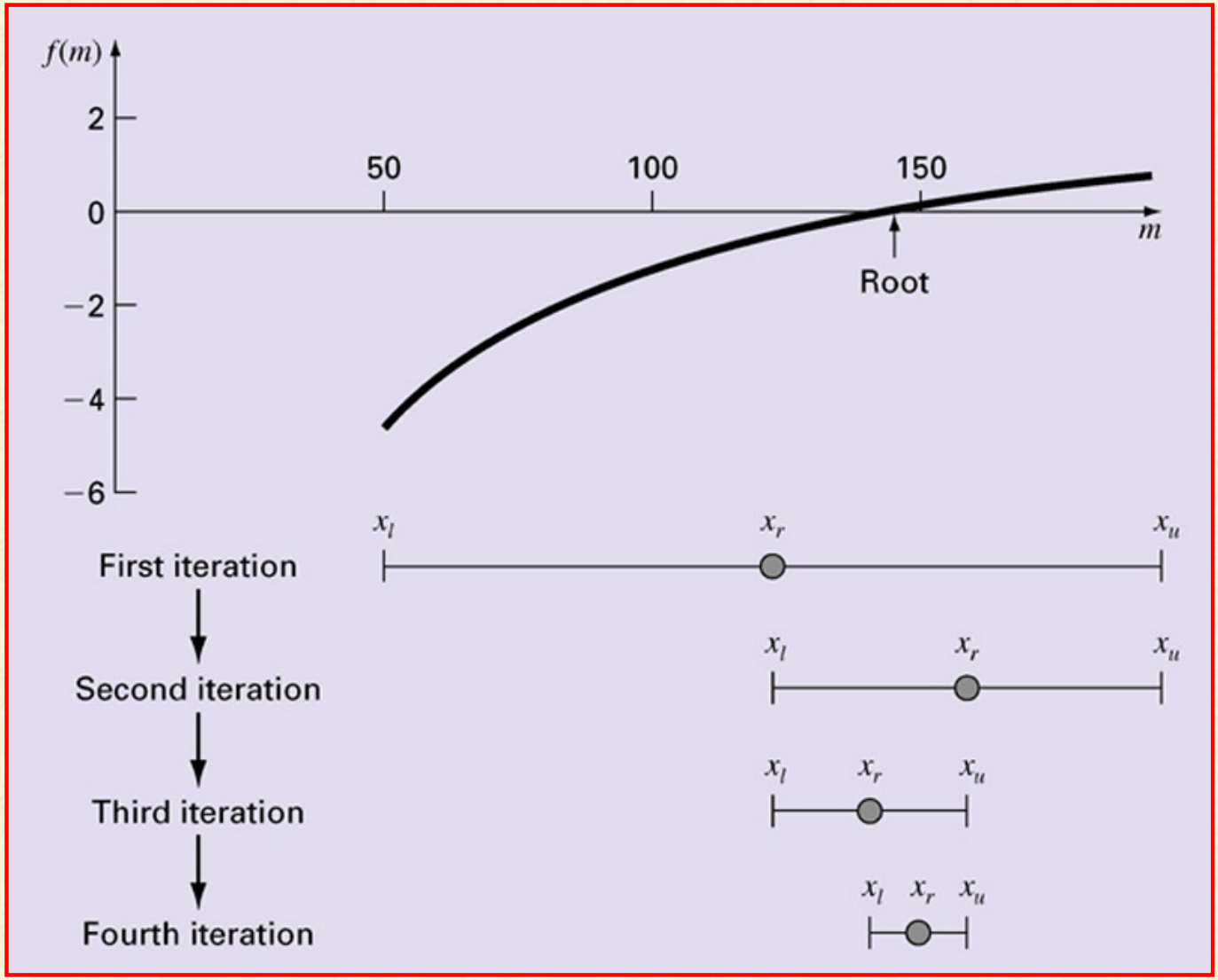
- **Determine the mass m of a bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall.**

$$v(t) = \sqrt{\frac{mg}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

- **Rearrange the equation – solve for *m***

$$f(m) = \sqrt{\frac{mg}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - v(t) = 0$$

Graphical Depiction of Bisection Method



$$f(m) = \sqrt{\frac{mg}{c_d}}\tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - v(t) = 0$$

(50 kg < m < 200 kg)

Example 4

Bisection Method

$$Example: f(x) = x^2 - 2x - 3 = 0$$

$$initial\ estimeates\ [x_l, x_u] = [2.0,\ 3.2]$$

iter	x_l	x_u	x_r	$f(x_r)$	Δx
1	2.0	3.2	2.6	−1.44	1.2
2	2.6	3.2	2.9	−0.39	0.6
3	2.9	3.2	3.05	0.2025	0.3
4	2.9	3.05	2.975	−0.0994	0.15
5	2.975	3.05	3.0125	0.0502	0.075
6	2.975	3.0125	2.99375	−0.02496	0.0375

$f(2) = -3,\ f(3.2) = 0.84$

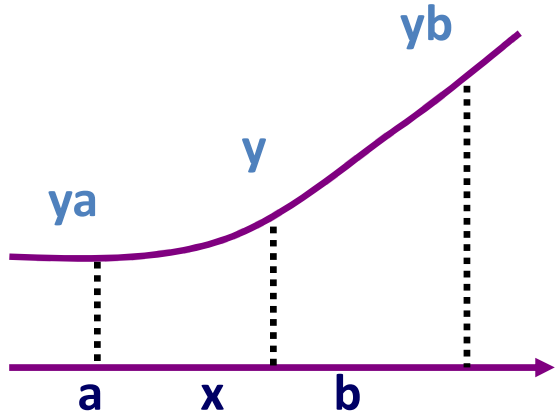
```
function root = bisection(func,xl,xu,es,maxit)
% bisection(func,xl,xu,es,maxit);
%   use bisection method to find the root of a function
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

if func(xl)*func(xu) > 0 % if guesses do not bracket a sign change,
    error('no bracket')  % display an error message and terminate
    return
end
% if necessary, assign default values of maxit and es
if nargin < 5, maxit = 50; end  % if maxit blank, set to 50
if nargin < 4, es = 0.001; end  % if es blank, set to 0.001

%bisection
iter = 0;
xr = xl;
while (1)
    xrold = xr;
    xr = (xl + xu)/2;
    iter = iter + 1;
    if xr ~= 0, ea =abs((xr-xrold)/xr) * 100; end
    test = func(xl) * func(xr);
    if test < 0
        xu = xr;
    elseif test > 0
        xl = xr;
    else
        ea = 0;
    end
    if ea <= es | iter >= maxit, break, end
end
root = xr;
```

Use “inline” command to specify the function “func”

```
function [x,y] = bisect2(func)

% Find root near x1 using the bisection 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) = (a(i) + b(i))/2;  y(i) = feval(func, x(i));
     if((x(i) - a(i)) < es)
        disp('Bisection method has converged'); break;
     end
     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;
     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)
```

An interactive M-file

Use “feval” to evaluate the function “func”



break: terminate a “for” or “while” loop

Examples: Bisection

1. Find root of Manning's equation

$$f(h) = Q - \frac{1}{n}\frac{(bh)^{5/3}}{(b+2h)^{2/3}} S^{1/2} = 0$$

2. Two other functions

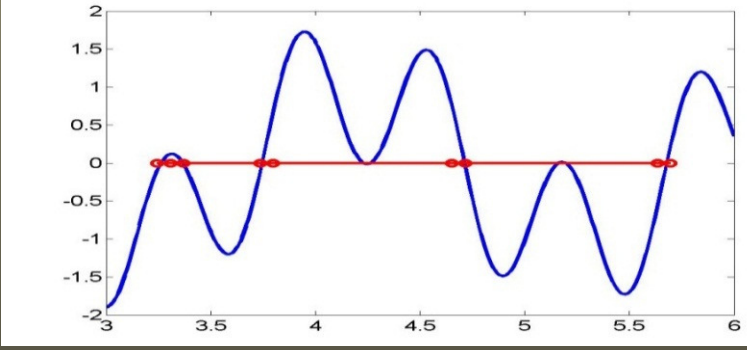
$$f(x) = e^{-x} - x^2 = 0$$

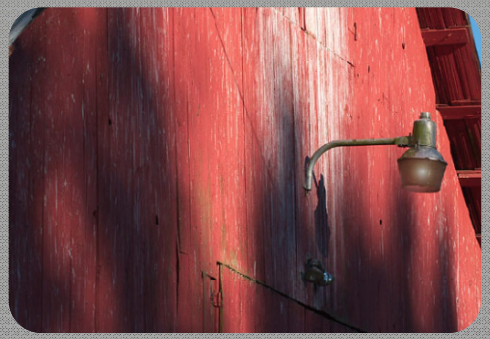
$$f(x) = x^3 - 3x + 1 = 0$$

Bisection Method for Manning Equation

```
»bisect2('manning')
enter lower bound xl = 0
enter upper bound xu = 10
allowable tolerance es = 0.00001
maximum number of iterations maxit = 50
Bisection method has converged
     step        xl         xu         xr       f(xr)
    1.0000         0   10.0000    5.0000   264.0114
    2.0000    5.0000   10.0000    7.5000  -337.3800
    3.0000    5.0000    7.5000    6.2500   -25.2627
    4.0000    5.0000    6.2500    5.6250   122.5629
    5.0000    5.6250    6.2500    5.9375    49.4013
    6.0000    5.9375    6.2500    6.0938    12.2517
    7.0000    6.0938    6.2500    6.1719    -6.4605
    8.0000    6.0938    6.1719    6.1328     2.9069
    9.0000    6.1328    6.1719    6.1523    -1.7740
   10.0000    6.1328    6.1523    6.1426     0.5672
   11.0000    6.1426    6.1523    6.1475    -0.6032
   12.0000    6.1426    6.1475    6.1450    -0.0180
   13.0000    6.1426    6.1450    6.1438     0.2746
   14.0000    6.1438    6.1450    6.1444     0.1283
   15.0000    6.1444    6.1450    6.1447     0.0552
   16.0000    6.1447    6.1450    6.1449     0.0186
   17.0000    6.1449    6.1450    6.1449     0.0003
   18.0000    6.1449    6.1450    6.1450    -0.0088
   19.0000    6.1449    6.1450    6.1450    -0.0042
   20.0000    6.1449    6.1450    6.1450    -0.0020
```

Facts





Any n^{th} order polynomial has exactly n zeros (counting real and complex zeros with their multiplicities)

Any polynomial with an odd order has at least one real zero.

If a function has a zero at *x=r* with multiplicity *m* then the function and its first *(m-1)* derivatives are zero at *x=r and the m^{th} derivative at r is not zero*.

Example 6

One making a bookshelf to carry books that range from 8 ½ ” to 11” in height and would take 29”of space along length. The material is wood having Young’s Modulus 3.667 Msi, thickness 3/8 ” and width 12”. Find the maximum vertical deflection of the bookshelf. The vertical deflection of the shelf is given by

$$v(x) = 0.42493 \times 10^{-4} x^3 - 0.13533 \times 10^{-8} x^5 - 0.66722 \times 10^{-6} x^4 - 0.018507\, x$$

where x is the position along the length of the beam. Hence to find the maximum deflection we need to find where $f(x) = \frac{dv}{dx} = 0$ and conduct the second derivative test.

Example 6

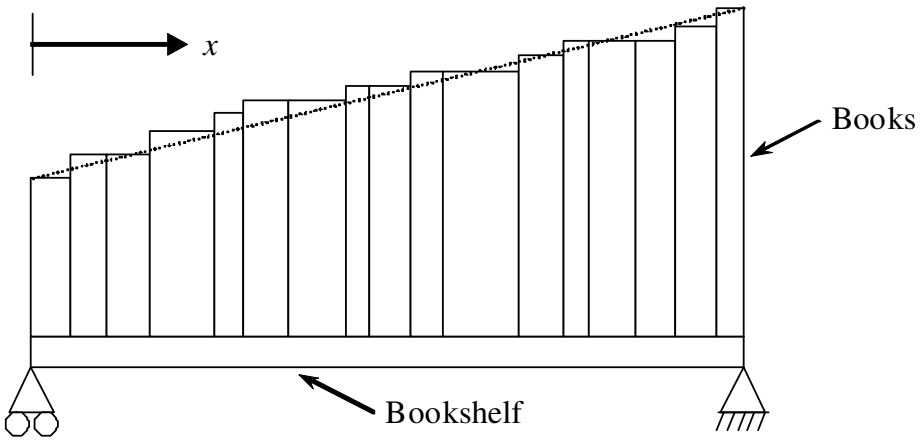


Figure 5 A loaded bookshelf.

The equation that gives the position *x* where the deflection is maximum is given by

$$f(x) = -0.67665\times10^{-8}x^4 - 0.26689\times10^{-5}x^3 + 0.12748\times10^{-3}x^2 - 0.018507 = 0$$

Use the bisection method of finding roots of equations to find the position *x* where the deflection is maximum. Conduct three iterations to estimate the root of the above equation. 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 6

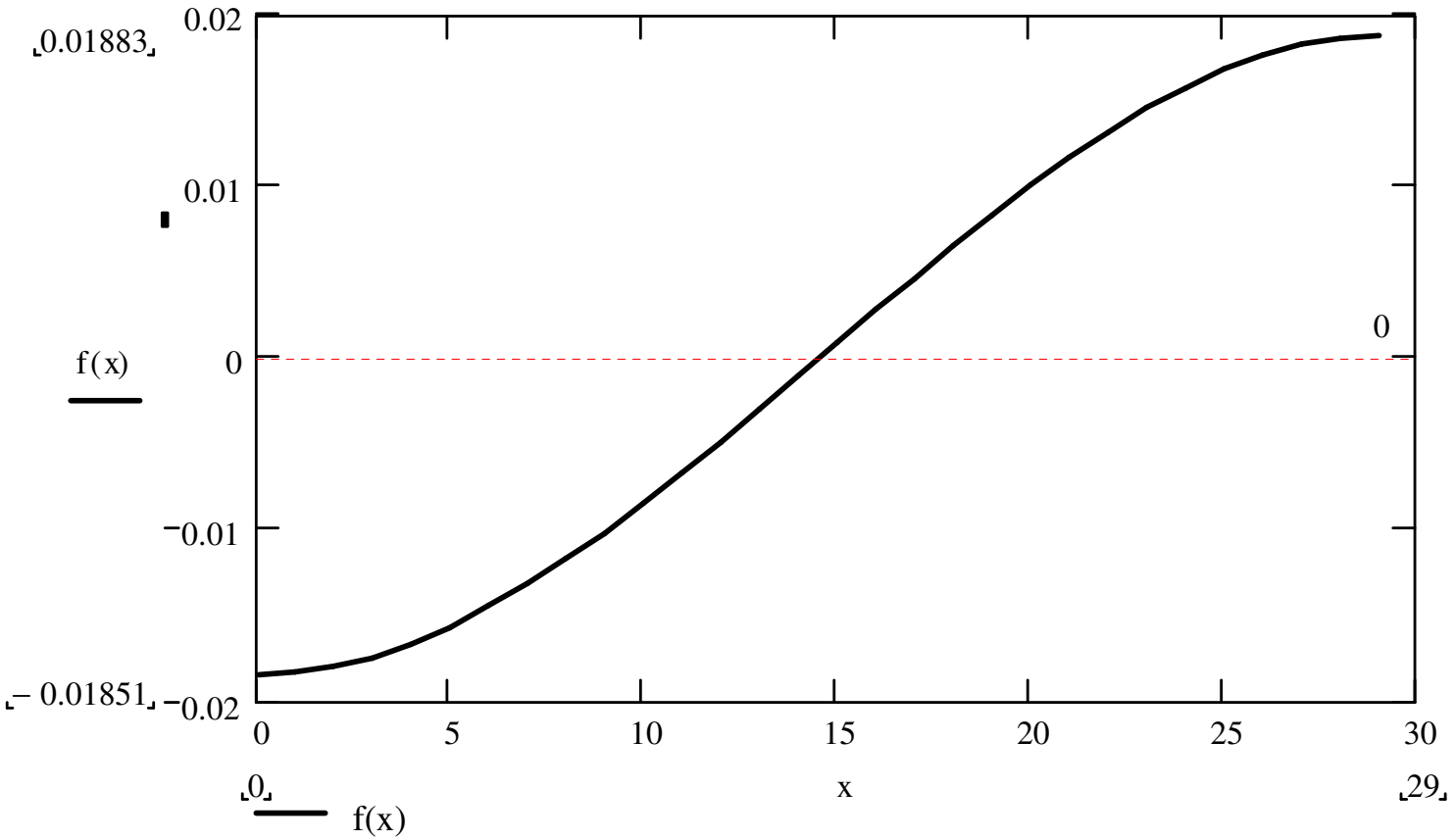


Figure 6 Graph of the function f(x).

$$f(x) = -0.67665\times10^{-8}x^4 - 0.26689\times10^{-5}x^3 + 0.12748\times10^{-3}x^2 - 0.018507 = 0$$

Example 6

Solution

From the physics of the problem, the maximum deflection would be between $x = 0$ and $x = L$, where

$$L = \text{length of the bookshelf}$$

that is

$$0 \le x \le L$$

$$0 \le x \le 29$$

Let us assume

$$x_l = 0,\ x_u = 29$$

Example 6

Check if the function changes sign between x_l and x_u .

$$f(x_l) = f(0)$$

$$= -0.67665\times10^{-8}(0)^4 - 0.26689\times10^{-5}(0)^3 + 0.12748\times10^{-3}(0)^2 - 0.018507$$

$$= -0.018507$$

$$f(x_u) = f(29)$$

$$= -0.67665\times10^{-8}(29)^4 - 0.26689\times10^{-5}(29)^3 + 0.12748\times10^{-3}(29)^2 - 0.018507$$

$$= 0.018826$$

Hence

$$f(x_l)f(x_u) = f(0)f(29) = (-0.018507)(-0.018826) < 0$$

So there is at least one root between x_l and x_u that is between 0 and 29.

Example 6

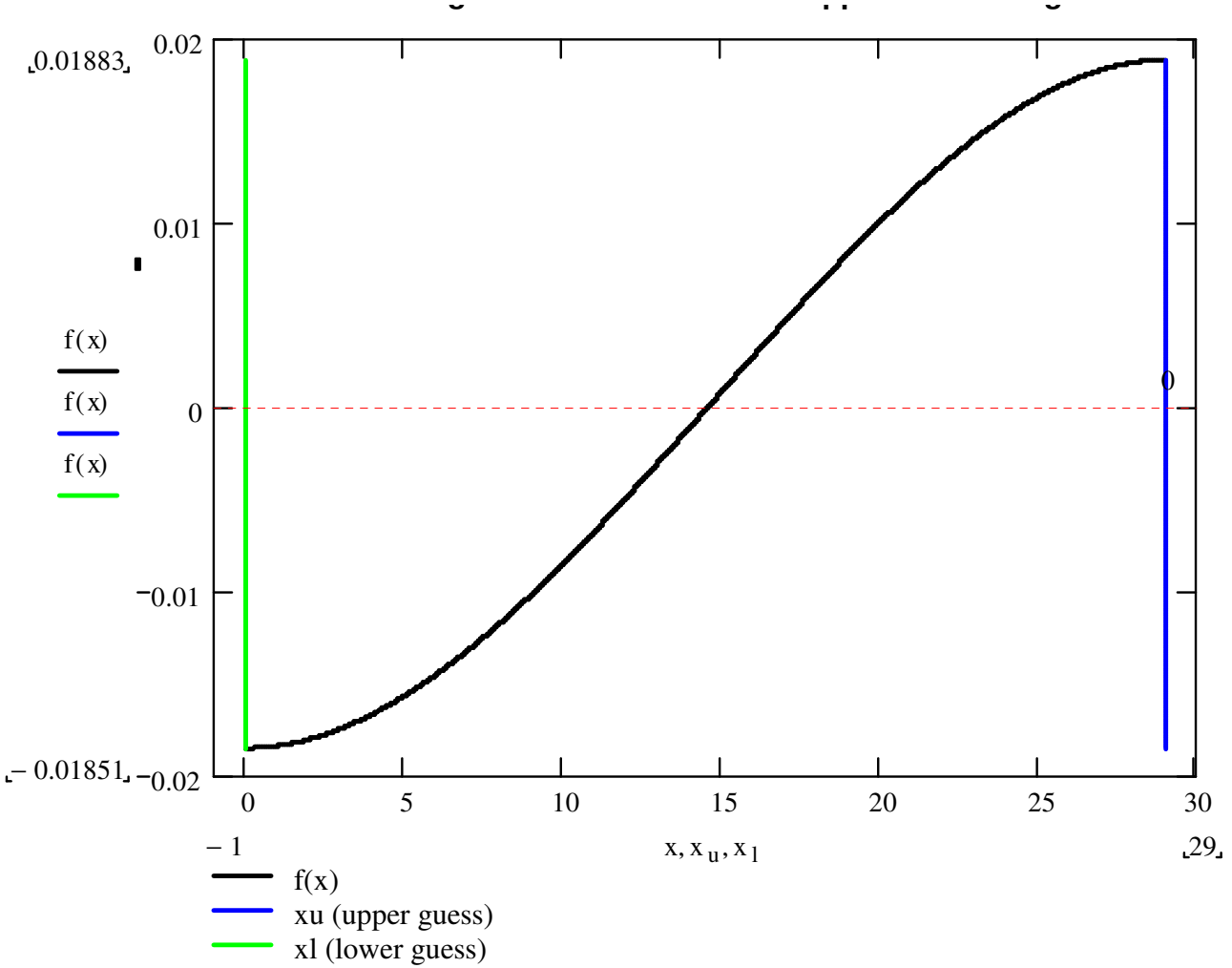


Figure 7 Checking the validity of the bracket.

Example 6

Iteration 1

The estimate of the root is $x_m = \frac{x_l + x_u}{2} = \frac{0+29}{2} = 14.5$

$$f(x_m) = f(14.9)$$

$$= -0.67665\times10^{-8}(14.5)^4 - 0.26689\times10^{-5}(14.5)^3 + 0.12748\times10^{-3}(14.5)^2 - 0.018507$$

$$= -1.4007\times10^{-4}$$

$$f(x_m)f(x_u) = f(14.5)f(29) = (-1.4007\times10^{-4})(0.018826) < 0$$

The root is bracketed between x_m and x_u.

The lower and upper limits of the new bracket are

$$x_l = 14.5,\ x_u = 29$$

The absolute relative approximate error $|\in_a|$ cannot be calculated as we do not have a previous approximation.

Example 6

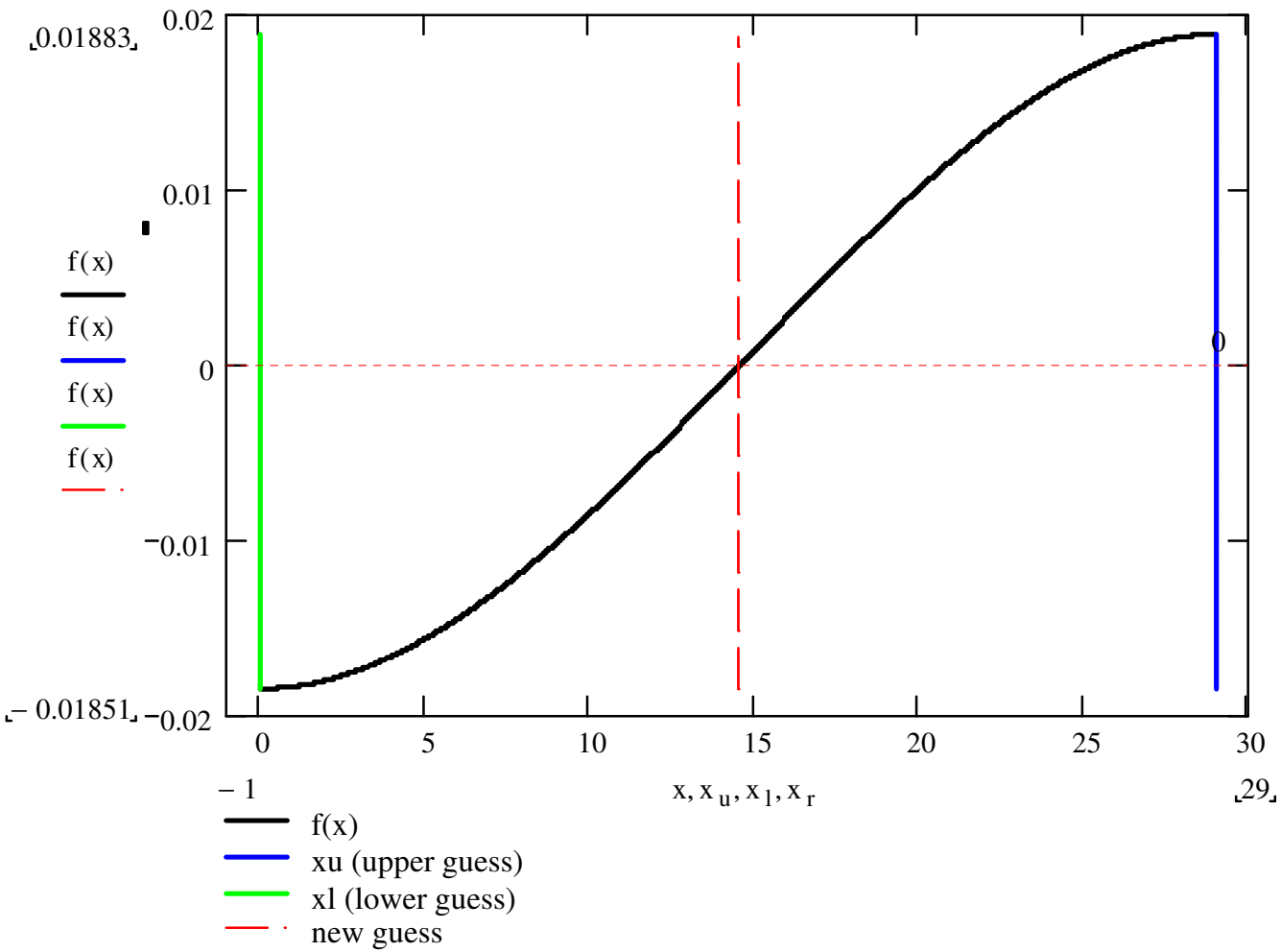


Figure 8 Graph of the estimate of the root after Iteration 1.

Example 6

Iteration 2

The estimate of the root is $x_m = \dfrac{x_l + x_u}{2} = \dfrac{14.5+29}{2} = 21.75$

$$f(x_m) = f(21.75)$$

$$= -0.67665\times10^{-8}(21.75)^4 - 0.26689\times10^{-5}(21.75)^3 + 0.12748\times10^{-3}(21.75)^2 - 0.018507$$

$$= 0.012824$$

$$f(x_m)f(x_u) = f(14.5)f(21.75) = \left(-1.4007\times10^{-4}\right)(0.012824) < 0$$

The root is bracketed between x_l and x_m .

The lower and upper limits of the new bracket are

$$x_l = 14.5,\ x_u = 21.75$$

Example 6

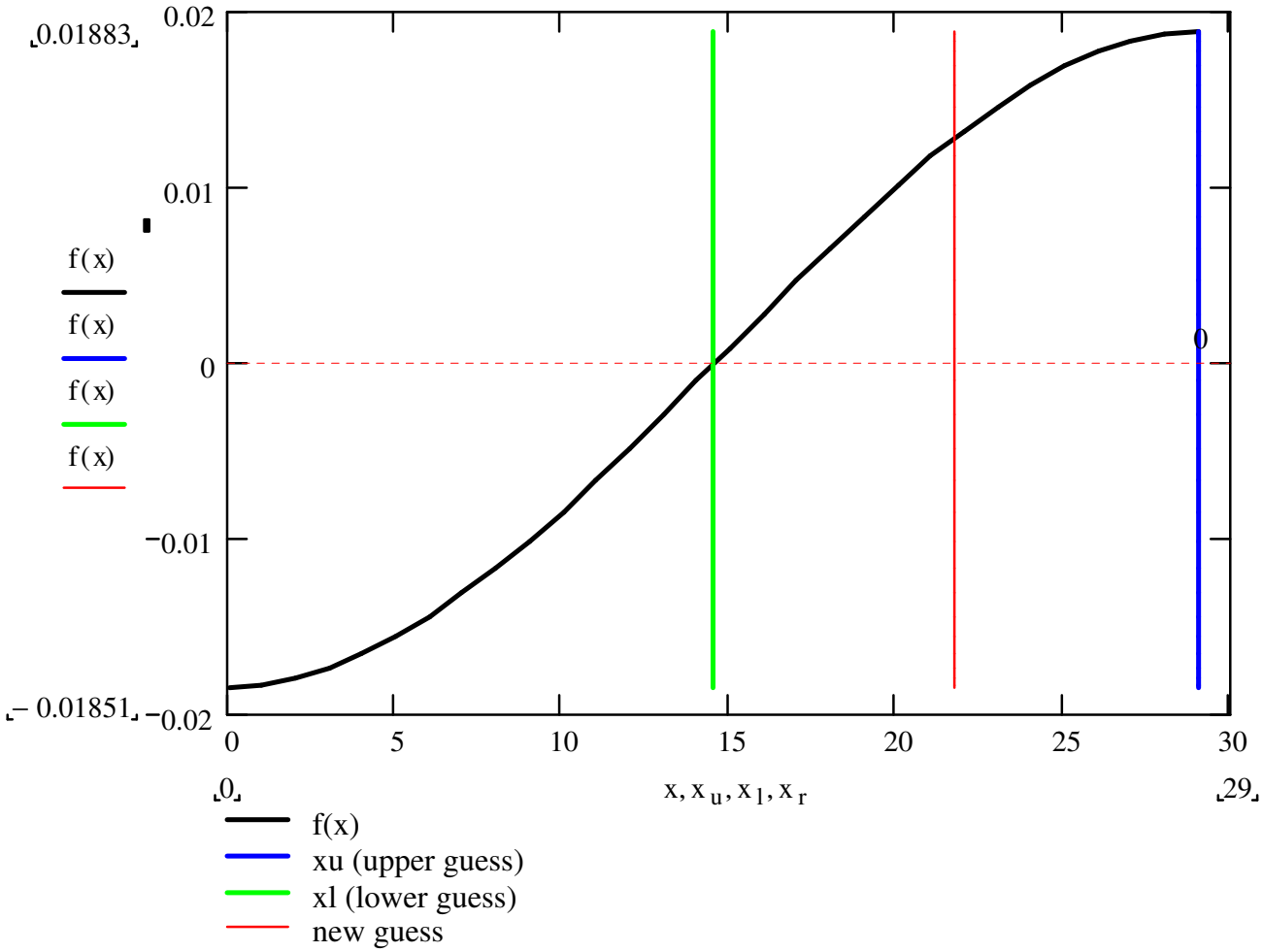


Figure 9 Graph of the estimate of the root after Iteration 2.

Example 6

The absolute relative approximate error at the end of Iteration 2 is

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

$$= \left|\frac{21.75 - 14.5}{21.75}\right| \times 100$$

$$= 33.333\%$$

None of the significant digits are at least correct in the estimated root

$$x_m = 21.75$$

as the absolute relative approximate error is greater than 5%.

Example 6

Iteration 3

The estimate of the root is $x_m = \dfrac{x_l + x_u}{2} = \dfrac{14.5 + 21.75}{2} = 18.125$

$$f(x_m) = f(18.125)$$

$$= -0.67665\times10^{-8}(18.125)^4 - 0.26689\times10^{-5}(18.125)^3 + 0.12748\times10^{-3}(18.125)^2 - 0.018507$$

$$= 6.7502\times10^{-3}$$

$$f(x_l)f(x_m) = f(14.5)f(18.125) = \left(-1.4007\times10^{-4}\right)\left(6.7502\times10^{-3}\right) < 0$$

The root is bracketed between x_l and x_m .

The lower and upper limits of the new bracket are

$$x_l = 14.5,\ x_u = 18.125$$

Example 6

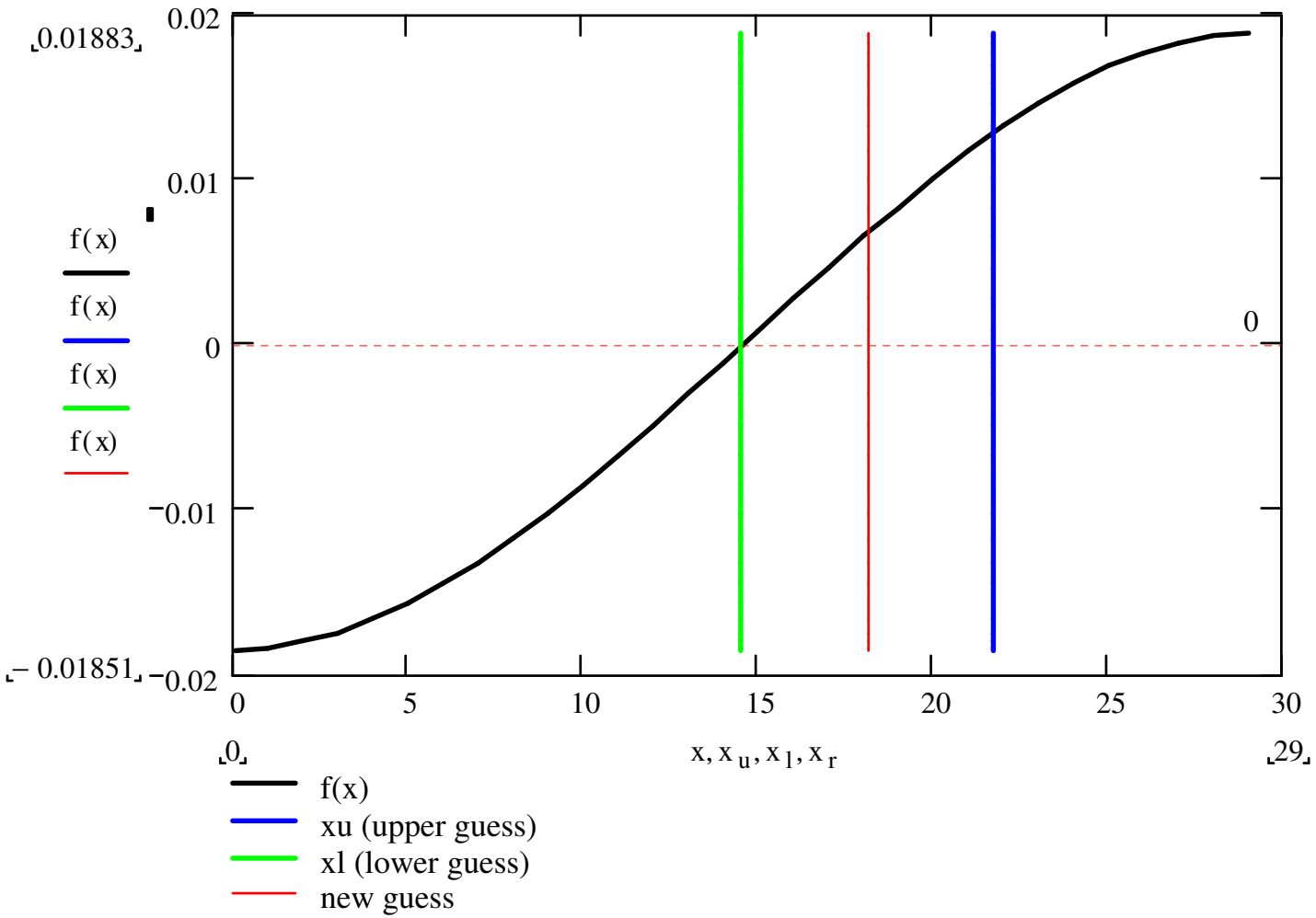


Figure 10 Graph of the estimate of the root after Iteration 3.

Example 6

The absolute relative approximate error at the end of Iteration 3 is

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

$$= \left|\frac{18.125 - 21.75}{18.125}\right| \times 100$$

$$= 20\%$$

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

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

Example 6

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

Iteration	x_λ	x_u	x_m	$\lvert\in_a\rvert\%$	$f(x_m)$
1	0	29	14.5	----------	-1.3992×10^{-4}
2	14.5	29	21.75	33.333	0.012824
3	14.5	21.75	18.125	20	6.7502×10^{-3}
4	14.5	18.125	16.313	11.111	3.3509×10^{-3}
5	14.5	16.313	15.406	5.8824	1.6099×10^{-3}
6	14.5	15.406	14.953	3.0303	7.3521×10^{-4}
7	14.5	14.953	14.727	1.5385	2.9753×10^{-4}
8	14.5	14.727	14.613	0.77519	7.8708×10^{-5}
9	14.5	14.613	14.557	0.38911	-3.0688×10^{-5}
10	14.5566	14.613	14.585	0.19417	2.4009×10^{-5}

Example 6

At the end of the 10th iteration,

$$\left|\in_a\right| = 0.19417\,\%$$

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

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

$$0.19417 \le 0.5\times 10^{2-m}$$

$$0.38835 \le 10^{2-m}$$

$$\log(0.38835) \le 2-m$$

$$m \le 2-\log(0.38835) = 2.4108$$

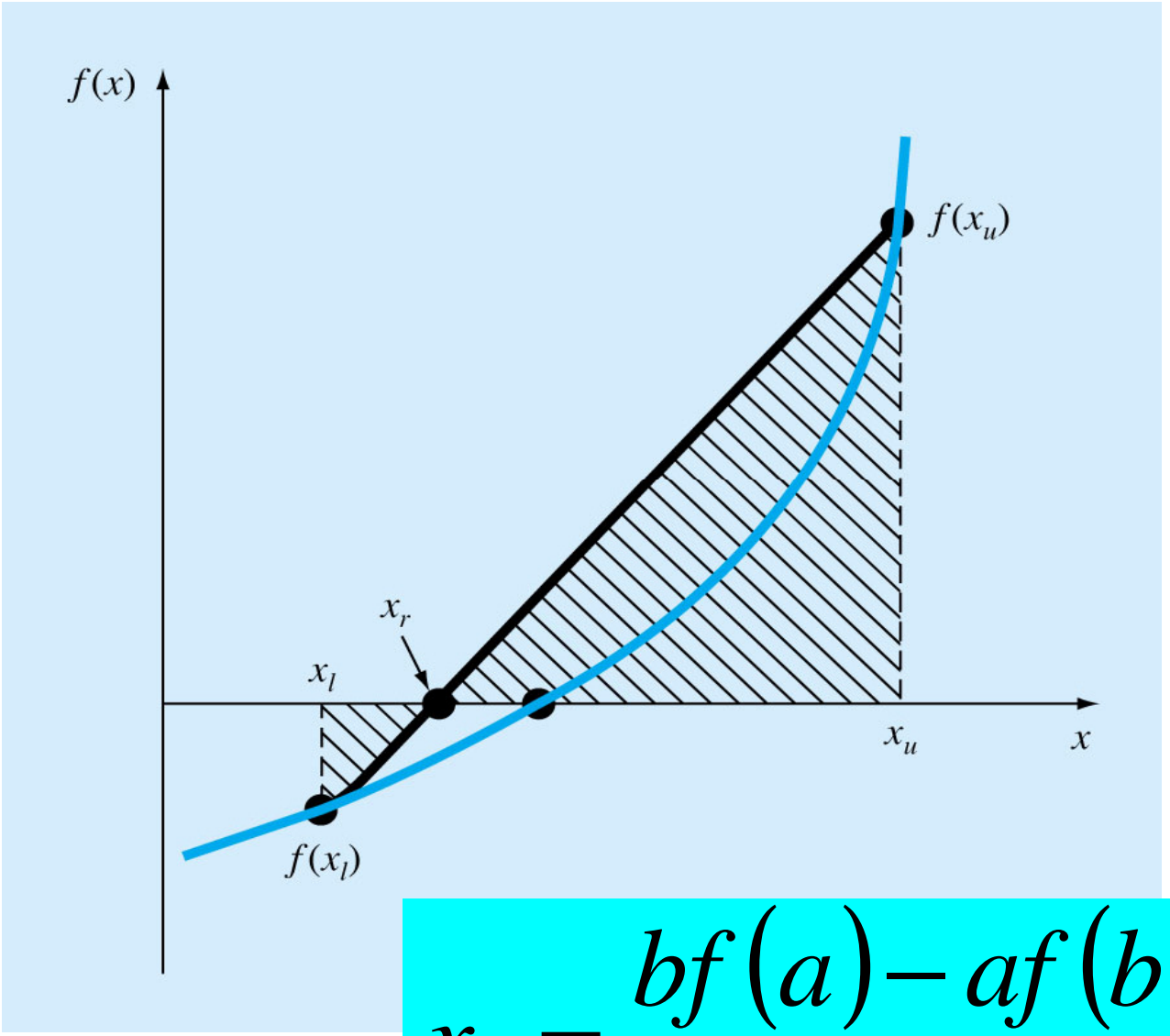
So

$$m = 2$$

The number of significant digits at least correct in the estimated root 14.585 is 2.

The False-Position Method (Regula-Falsi)

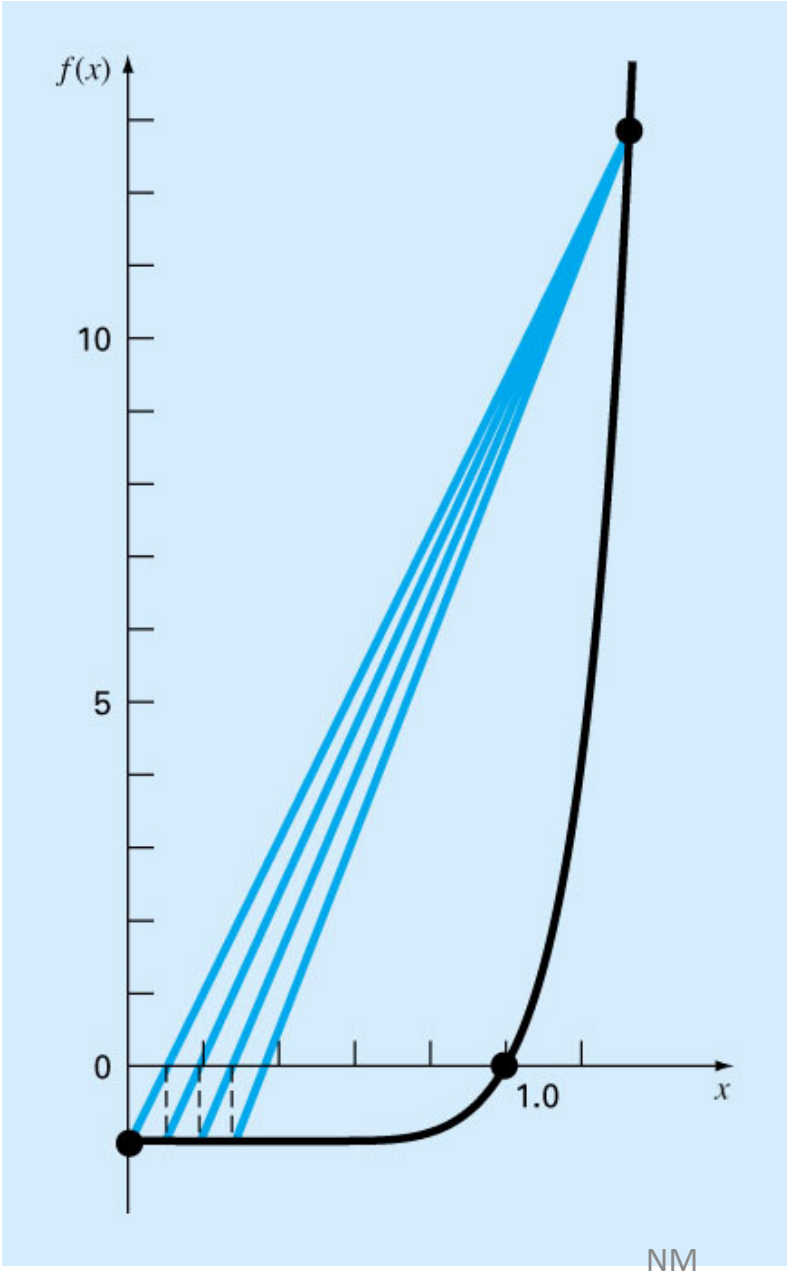
- **One can approximate the solution by doing a *linear interpolation* between $f(x_u)$ and $f(x_l)$**
- **Find x_r such that $l(x_r)=0$, where $l(x)$ is the linear approximation of $f(x)$ between x_l and x_u**
- **Derive x_r using similar triangles**



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

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

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.

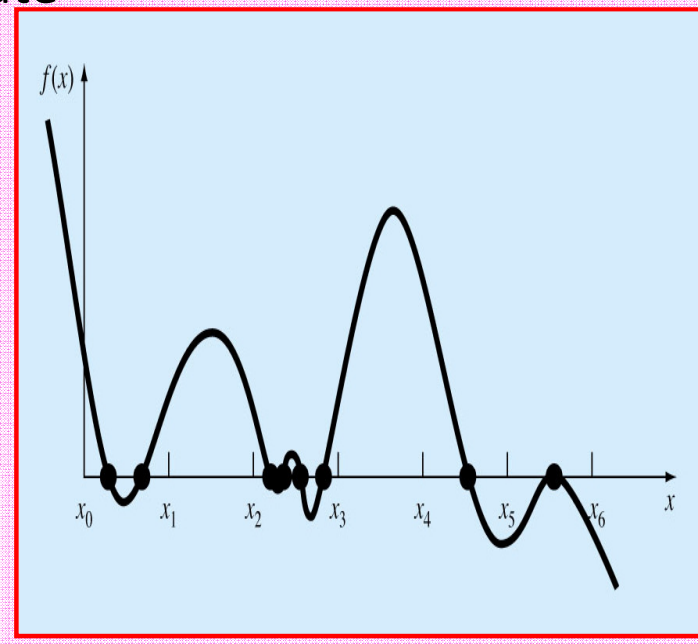
If this occurs, then the original formula $x_r = (x_l + x_u)/2$ can be used

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

Ultimate solution:

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.

False-Position (point) Method

Why bother with another method?

- The bisection method is simple and guaranteed to converge (single root)
- But the convergence is slow and non-monotonic!
- The bisection method is a brute force method and makes no use of information about the function
- Bisection only the sign, not the value $f(x_k)$ itself
- False-position method takes advantage of function curve shape
- False position method may converge more quickly

Algorithm for False-Position Method

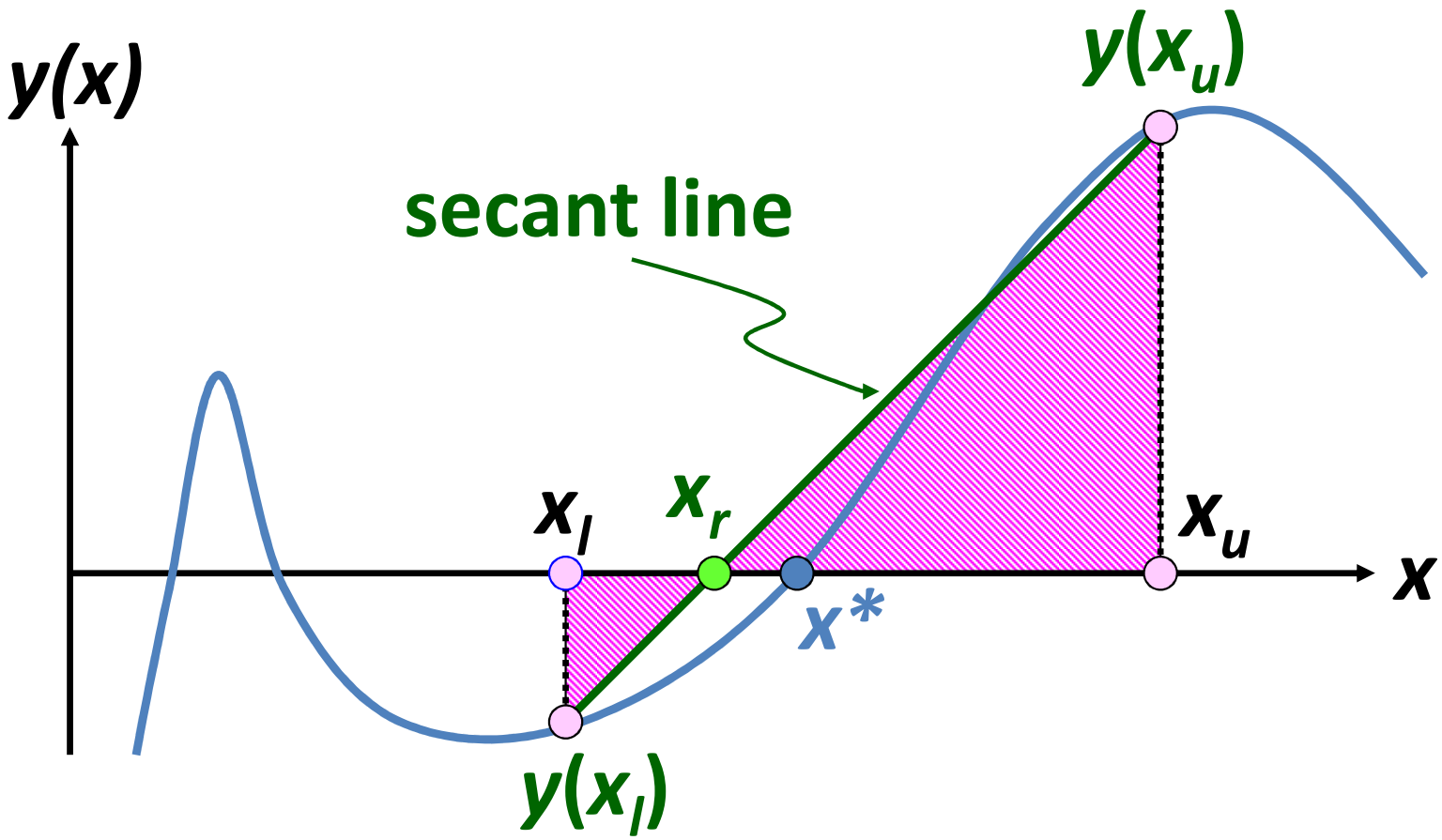
1. **Start with $[x_l, x_u]$ with $f(x_l) \cdot f(x_u) < 0$ (still need to bracket the root)**
2. **Draw a straight line to approximate the root**

$$f(x_r) = 0 \Rightarrow x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

3. **Check signs of $f(x_l) \cdot f(x_r)$ and $f(x_r) \cdot f(x_u)$ so that $[x_l, x_u]$ always bracket the root**

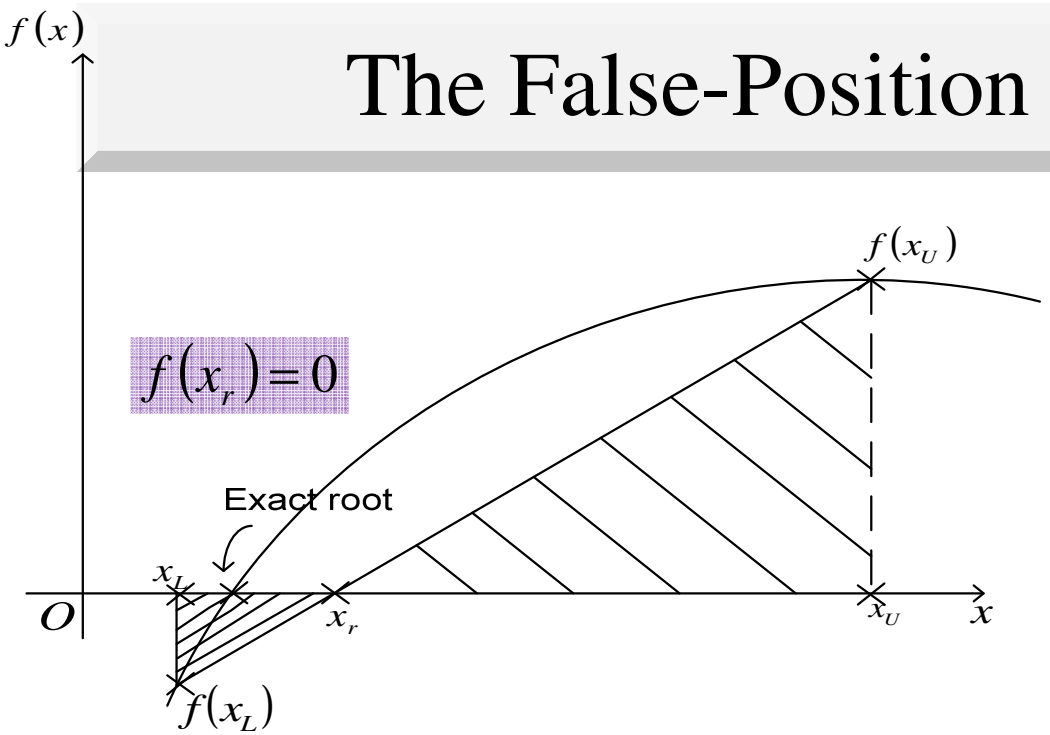
Maybe less efficient than the bisection method for highly nonlinear functions

False-Position Method



Straight line (linear) approximation to exact curve

The False-Position Method (Regula-Falsi)



$$x_r = \frac{x_L + x_U}{2}$$

$$\frac{0 - f(x_L)}{x_r - x_L} = \frac{0 - f(x_U)}{x_r - x_U}$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

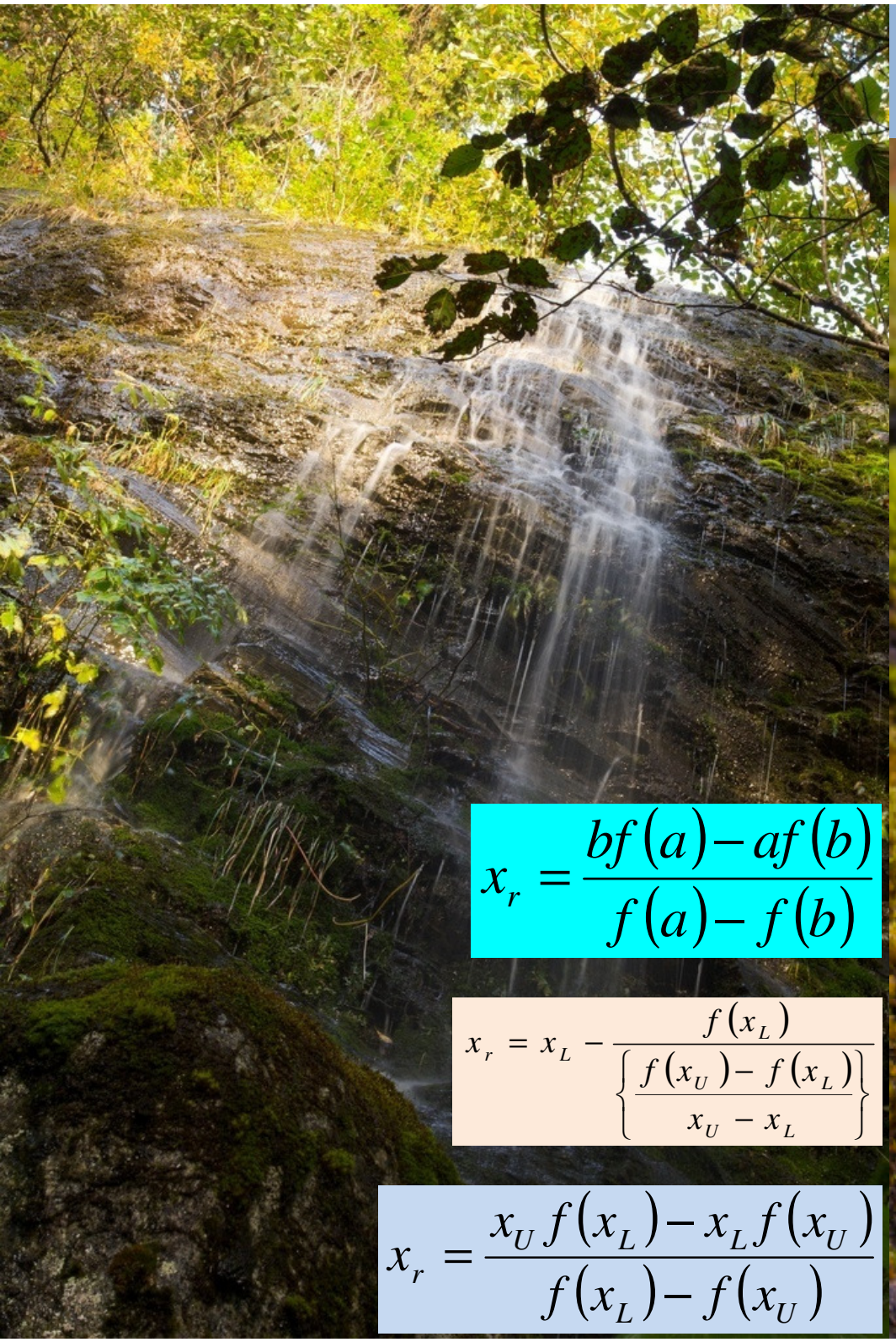
$$(x_r - x_L) f(x_U) = (x_r - x_U) f(x_L)$$

$$x_r = x_L - \frac{f(x_L)}{\left\{\frac{f(x_U) - f(x_L)}{x_U - x_L}\right\}}$$

$$x_U f(x_L) - x_L f(x_U) = x_r \{f(x_L) - f(x_U)\}$$

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

False Point Method



Step 1: Choose xl and xu. f(xl)*f(xu) < 0.

Step 2 : Estimate the root: Find the intersection of the line connecting the two points (xl, f(xl)), and (xu,f(xu)) and the x-axis. $x_r = x_u - f(x_u)(x_u - x_l)/(f(x_u) - f(x_l))$

Step 3: Determine the new bracket. If $f(x_r)*f(x_l) < 0$, then $x_u = x_r$ else $x_l = x_r$

Step 4: whether or not repeat the iteration.

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

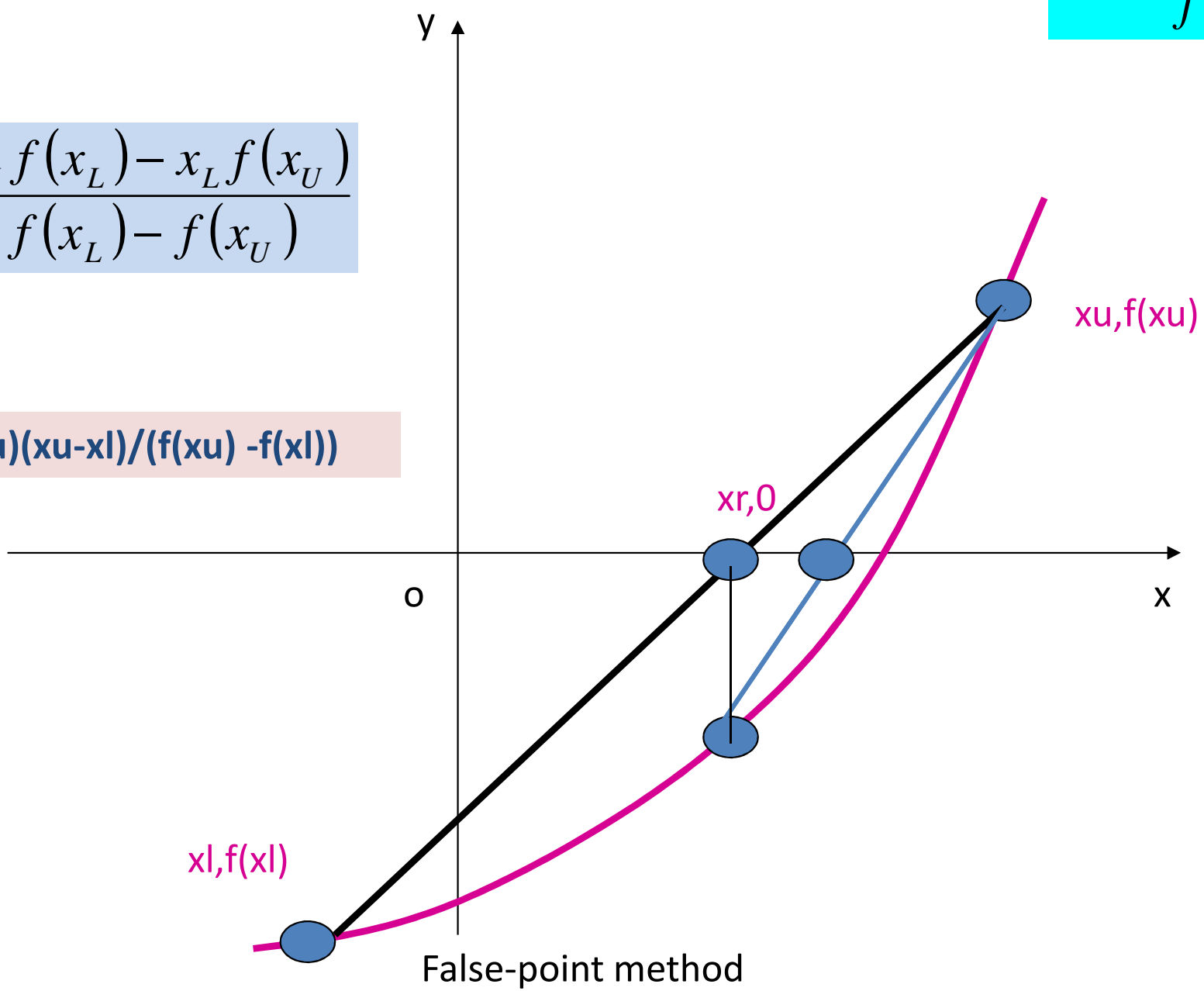
$$x_r = x_L - \frac{f(x_L)}{\left\{\frac{f(x_U) - f(x_L)}{x_U - x_L}\right\}}$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

$$x_r = \frac{b f(a) - a f(b)}{f(a) - f(b)}$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

xr=xu - f(xu)(xu-xl)/(f(xu) -f(xl))



False-point method

False-Position Method

- From geometry, similar triangles have similar ratios of sides

$$slope = \frac{y(x_u) - y(x_l)}{x_u - x_l} = \frac{y(x_u) - y(x_r)}{x_u - x_r}$$

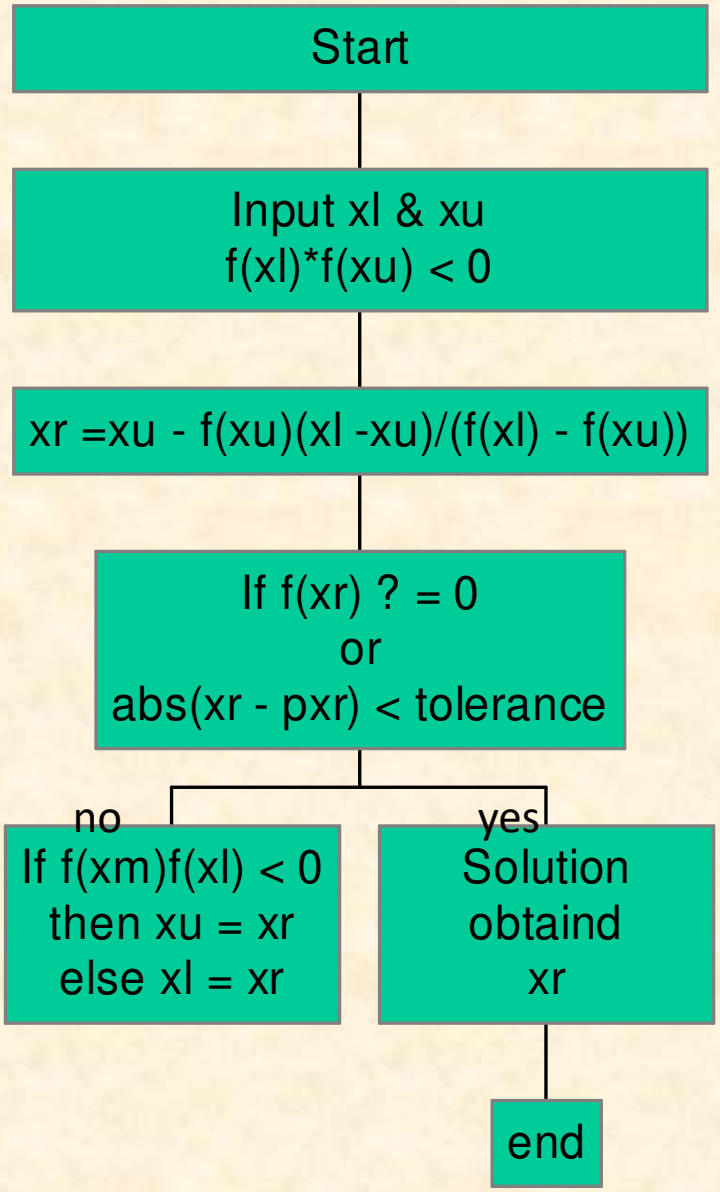
- The new approximate for the root : $y(x_r) = 0$
- This can be rearranged to yield False Position formula

$$x_r = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} f(x_u)$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

False-point Method Flowchart



$$x_r = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} f(x_u)$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

Example 1

False-Position

$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

$$Example: f(x) = x^2 - 2x - 3 = 0$$

$$initial\ estimeates\ [x_l, x_u] = [2.0,\ 3.2]$$

$$x_r = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} f(x_u)$$

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

$$x_r = x_L - \frac{f(x_L)}{\left\{\frac{f(x_U) - f(x_L)}{x_U - x_L}\right\}}$$

iter	x_l	x_u	x_r	$f(x_r)$
1	2.0	3.2	2.9375	−0.2461
2	2.9375	3.2	2.9968	−0.01207
3	2.99698	3.2	2.999856	−0.000576
4	2.999856	3.2	2.99999315	−0.0000274

f(2) = –3, f(3.2) = 0.84

Example 2

Find the root of $f(x)=(x-4)^2(x+2)=0$ using the initial guesses of x_l = -2.5 and x_u = -1 and a pre-specified tolerance of 0.1%.

$$f(x)=(x-4)^2(x+2)=0$$

$$x_r=\frac{bf(a)-af(b)}{f(a)-f(b)}$$

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

$$0.02706 \le 0.5\times 10^{2-m}$$

$$m \le 3.2666$$

Iteration	x_L	x_U	$f(x_L)$	$f(x_U)$	x_r	$\|\in_a\|\%$	$f(x_m)$
1	-2.5	-1	-21.13	25.00	-1.813	N/A	6.319
2	-2.5	-1.813	-21.13	6.319	-1.971	8.024	1.028
3	-2.5	-1.971	-21.13	1.028	-1.996	1.229	0.1542
4	-2.5	-1.996	-21.13	0.1542	-1.999	0.1828	0.02286
5	-2.5	-1.999	-21.13	0.02286	-2.000	0.02706	0.003383

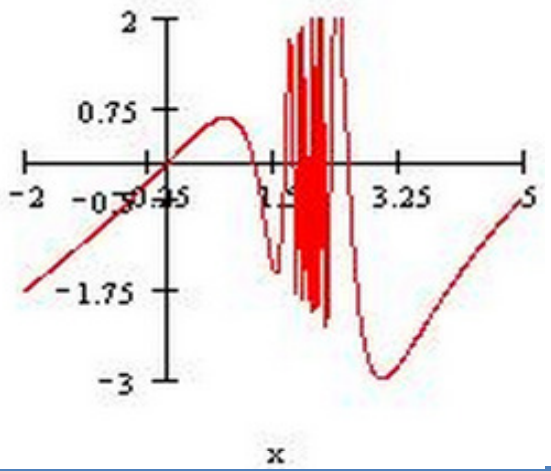
Example 3

False- Position

Find the root of **x * cos[(x)/ (x-2)]=0,** a = 1 and b = 1.5

$$x_r = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} f(x_u)$$

$$c = x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$



the roots of **x * cos[(x)/ (x-2)]=0** is approximately **1.222**

f(a) = 0.54; f(b) = -1.485

Iteration No.	a	b	c	f(a) * f(c)
1	1	1.5	1.133	0.159 (+ve)
2	1.133	1.5	1.194	0.032 (+ve)
3	1.194	1.5	1.214	3.192E-3 (+ve)
4	1.214	1.5	1.22	2.586E-4(+ve)
5	1.22	1.5	1.222	1.646E-5 (+ve)
6	1.222	1.5	1.222	3.811E-9(+ve)

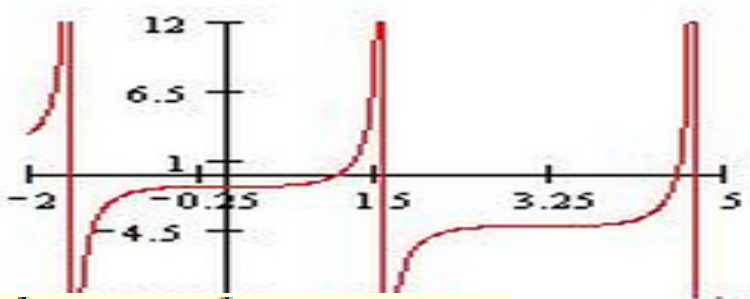
Example 4

Find the root of **tan(x)-x-1 = 0,** a = 0.5 and b = 1.5

f(a) = -0.9537; f(b) = 11.6

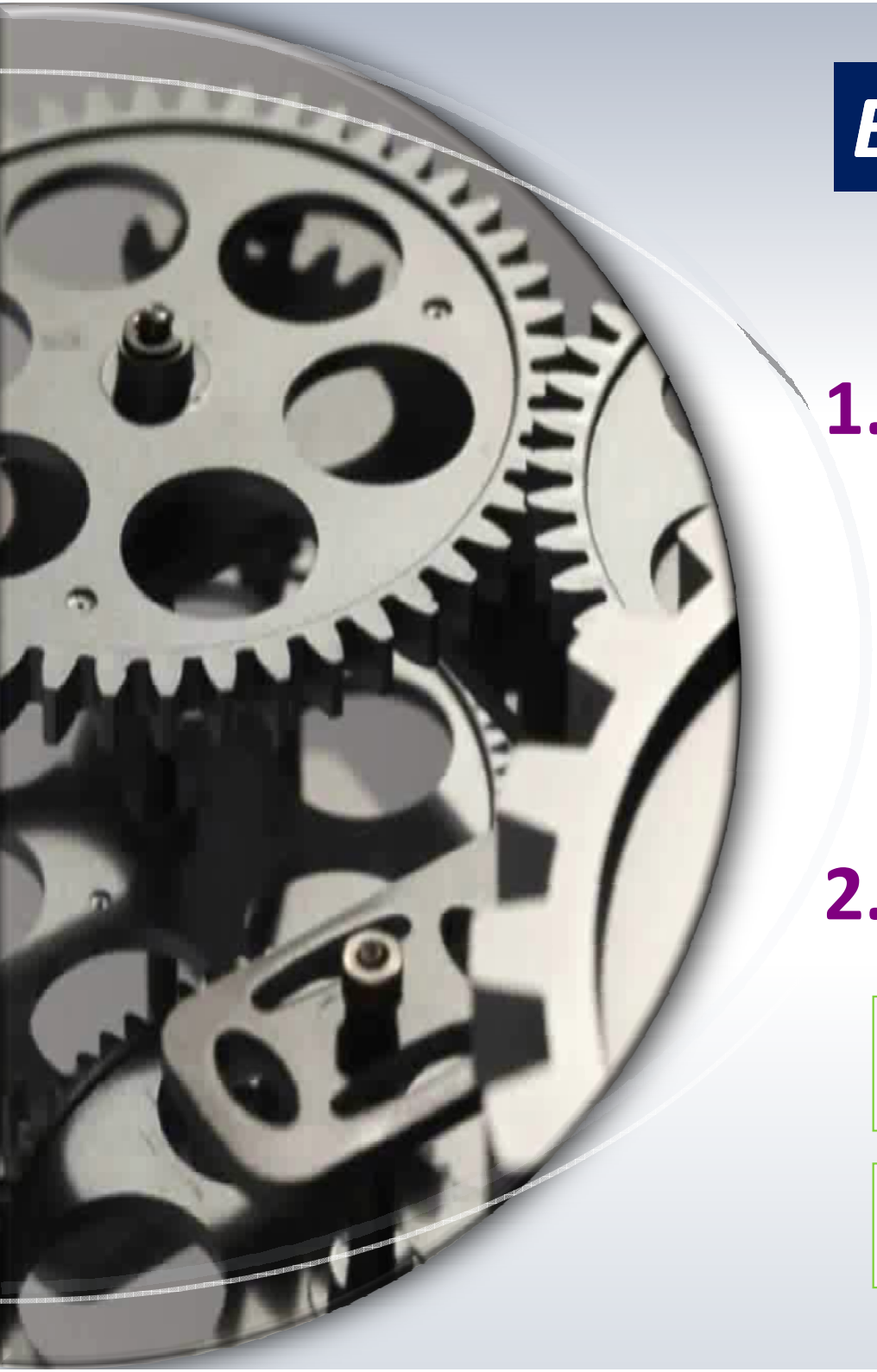
$$x_r = \frac{bf(a) - af(b)}{f(a) - f(b)}$$

$$x_r = x_u - \frac{x_l - x_u}{f(x_l) - f(x_u)} f(x_u)$$



the roots of **tan(x)-x-1 = 0** is approximately **1.129**

Iteration No.	a	b	c	f(a) * f(c)
1	0.5	1.5	0.576	0.8836 (+ve)
2	0.576	1.5	0.644	0.8274 (+ve)
3	0.644	1.5	0.705	0.762 (+ve)
4	0.705	1.5	0.76	0.692 (+ve)
5	0.76	1.5	0.808	0.616 (+ve)
6	0.808	1.5	0.851	0.541 (+ve)
	.	.	.	.
33	1.128	1.5	1.129	1.859E-4 (+ve)
34	1.129	1.5	1.129	2.947E-6 (+ve)



Examples: False-Position

1. Find root of Manning's equation

$$f(h)=Q-\frac{1}{n}\frac{(bh)^{5/3}}{(b+2h)^{2/3}}S^{1/2}=0$$

2. Some other functions

$$f(x)=e^{-2x}-x^2=0$$

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

False-position (Regula-Falsi)

Linear Interpolation Method

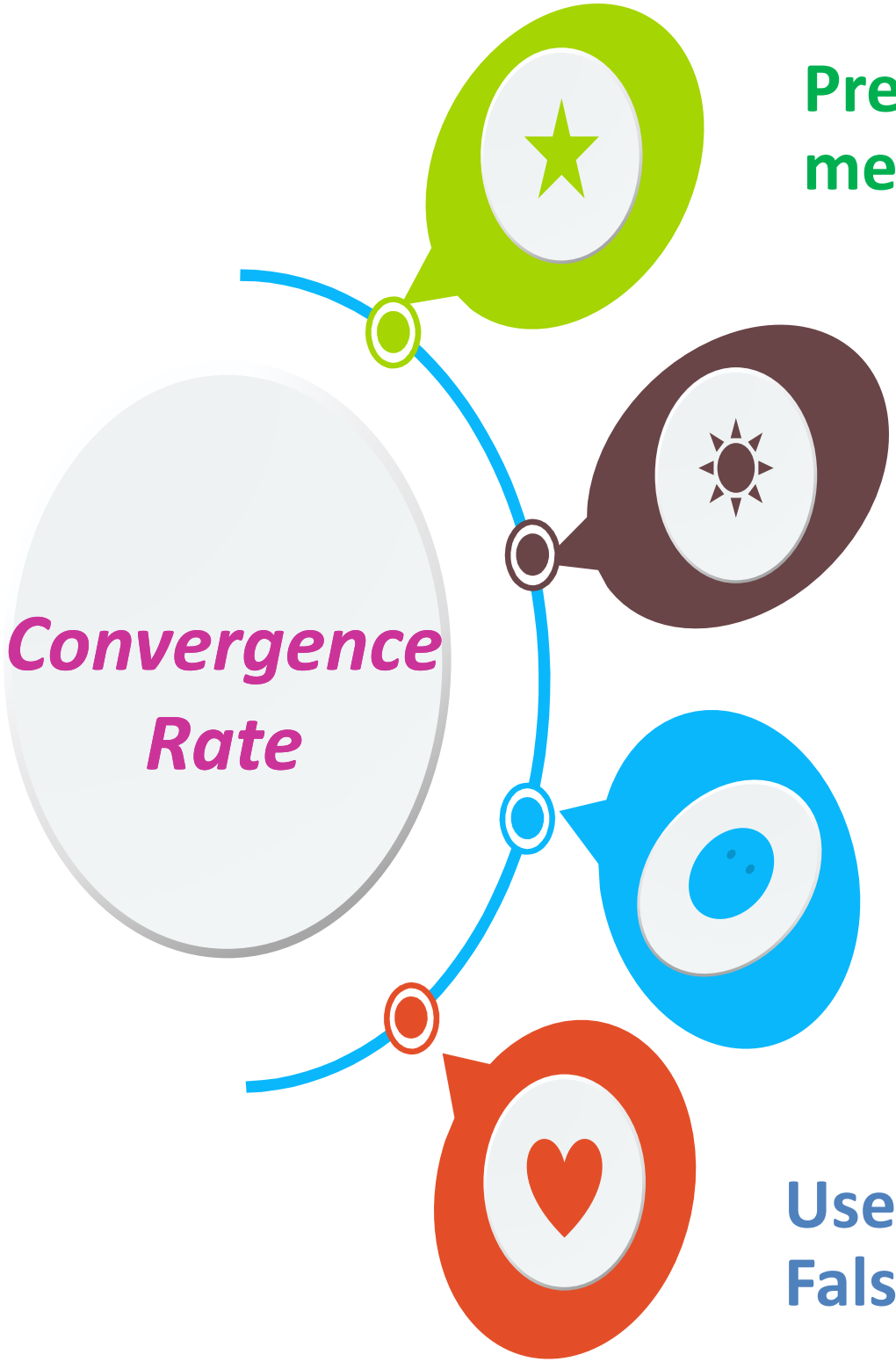
```
function [x,y] = false_position(func)

% 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('  NM      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



Prefer use false position method

There are times it may converge and less iterations.

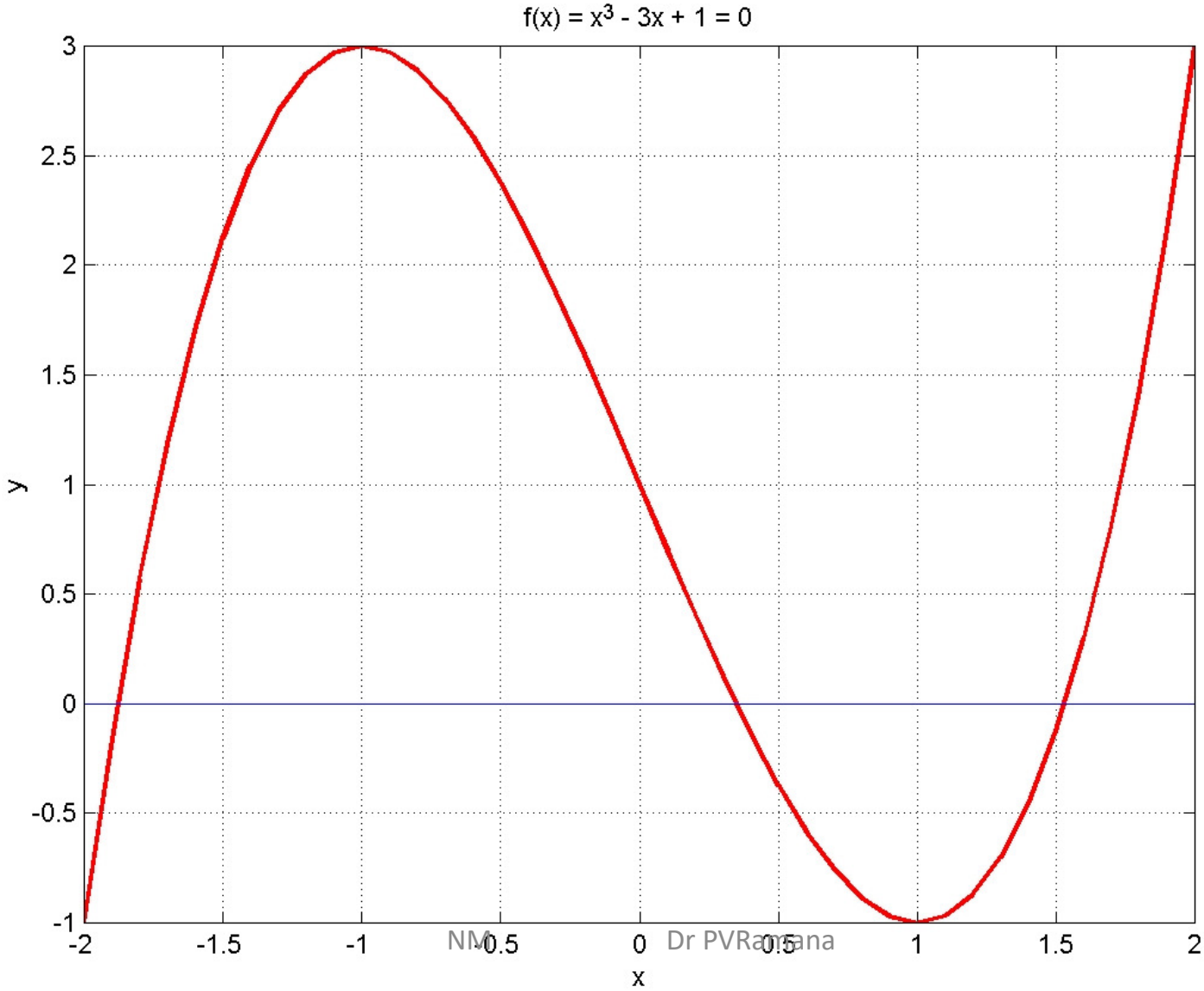
Example:

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

Use both Bisection as well as False position methods

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

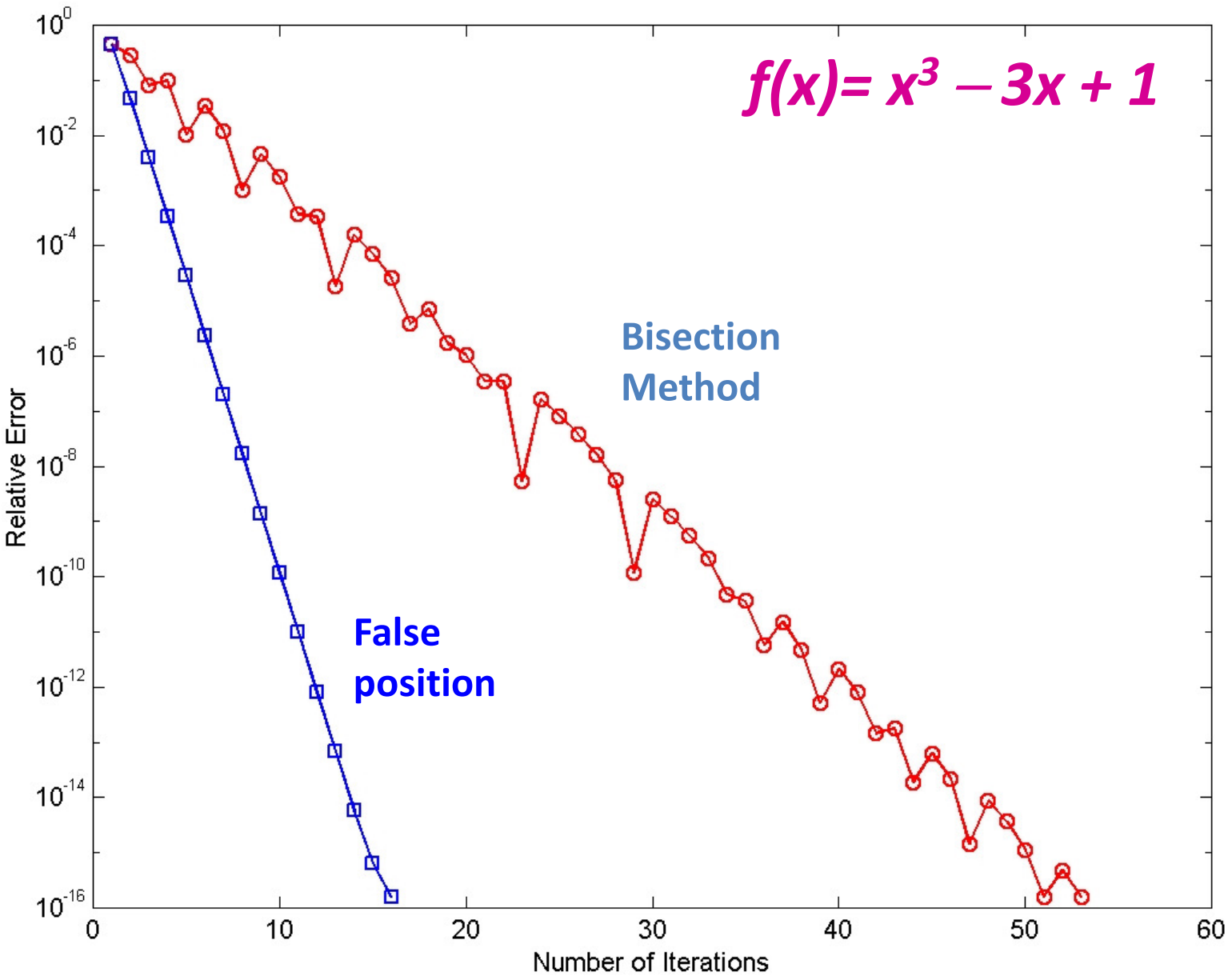
```
>> false_position(inline('x^3-3*x+1'))
enter lower bound xl = 0
enter upper bound xu = 1
allowable tolerance es = 1.e-20
maximum number of iterations maxit = 100
exact zero found
      step       xl         xu         xr      f(xr)
    1.0000         0    1.0000     0.5000   -0.3750
    2.0000         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
```

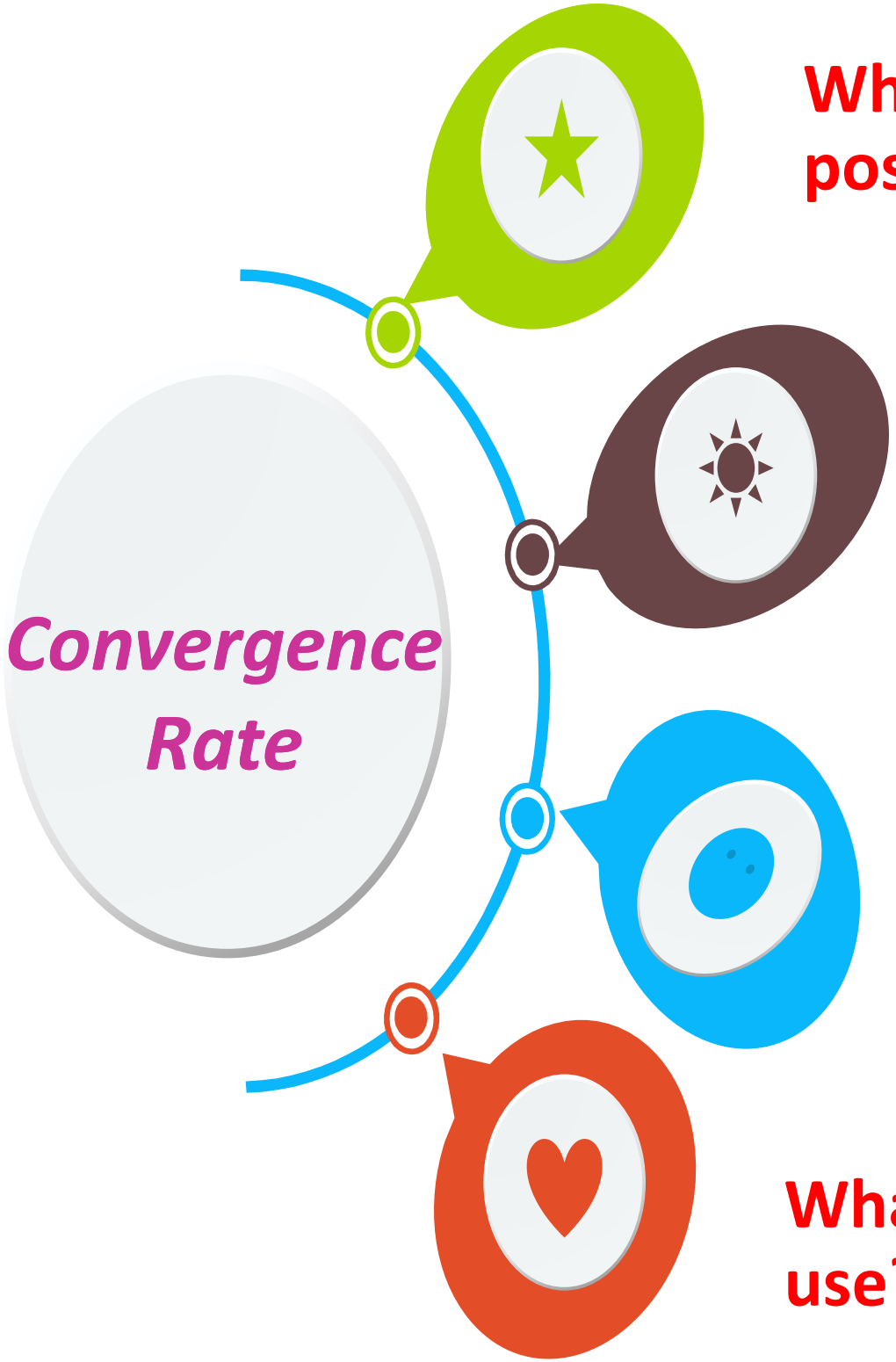
$f(x) = x^3 - 3x + 1 = 0$

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





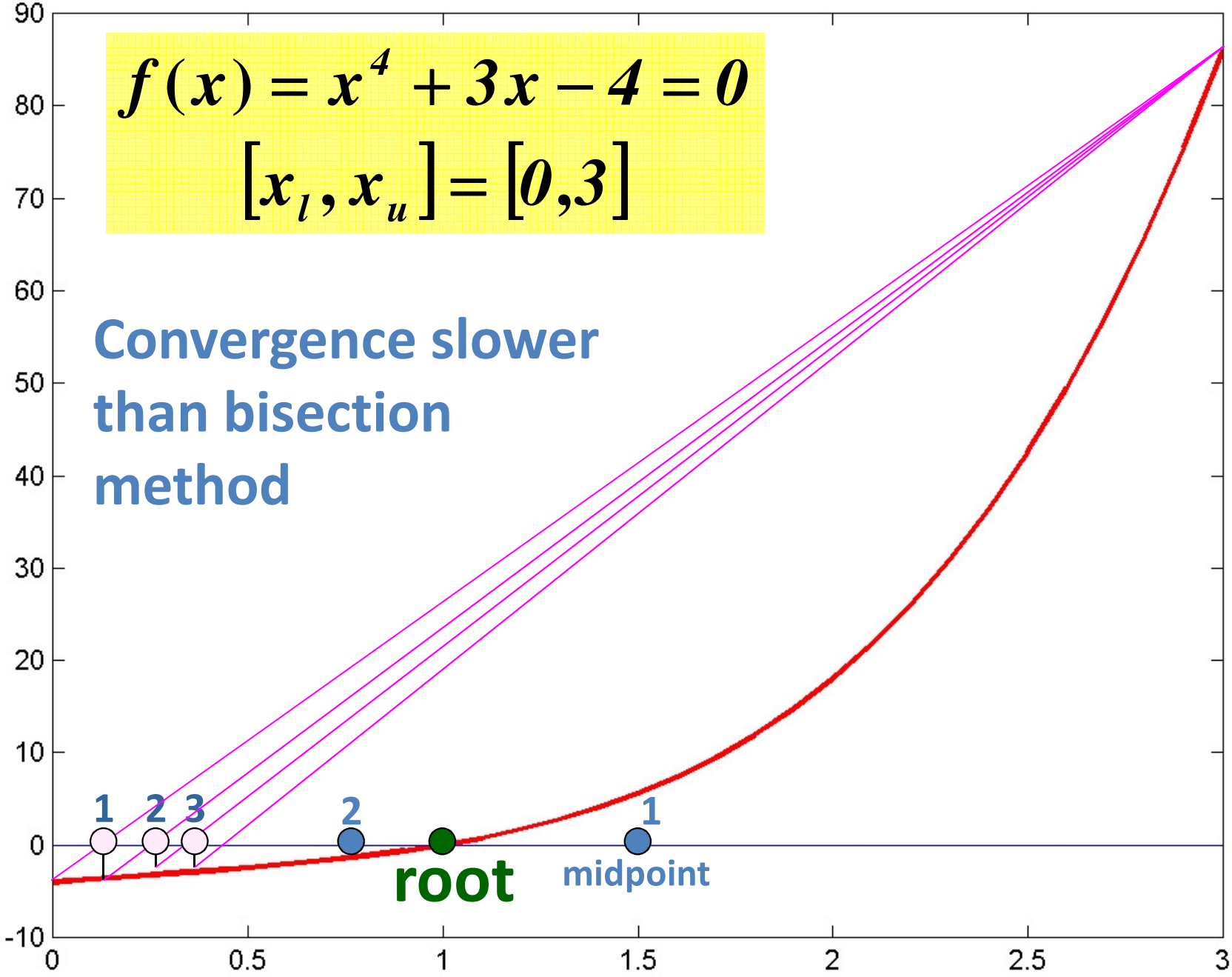
Why don't always use false position method?

There are times it may converge very, very slowly.

Example:

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

What other methods one can use?


$f(x) = x^4 + 3x - 4 = 0$
$[x_l, x_u] = [0,3]$
Convergence slower than bisection method
1 2 3
2
1
root
midpoint
90
80
70
60
50
40
30
20
10
0
-10
0
0.5
1
1.5
2
2.5
3

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

Thank you