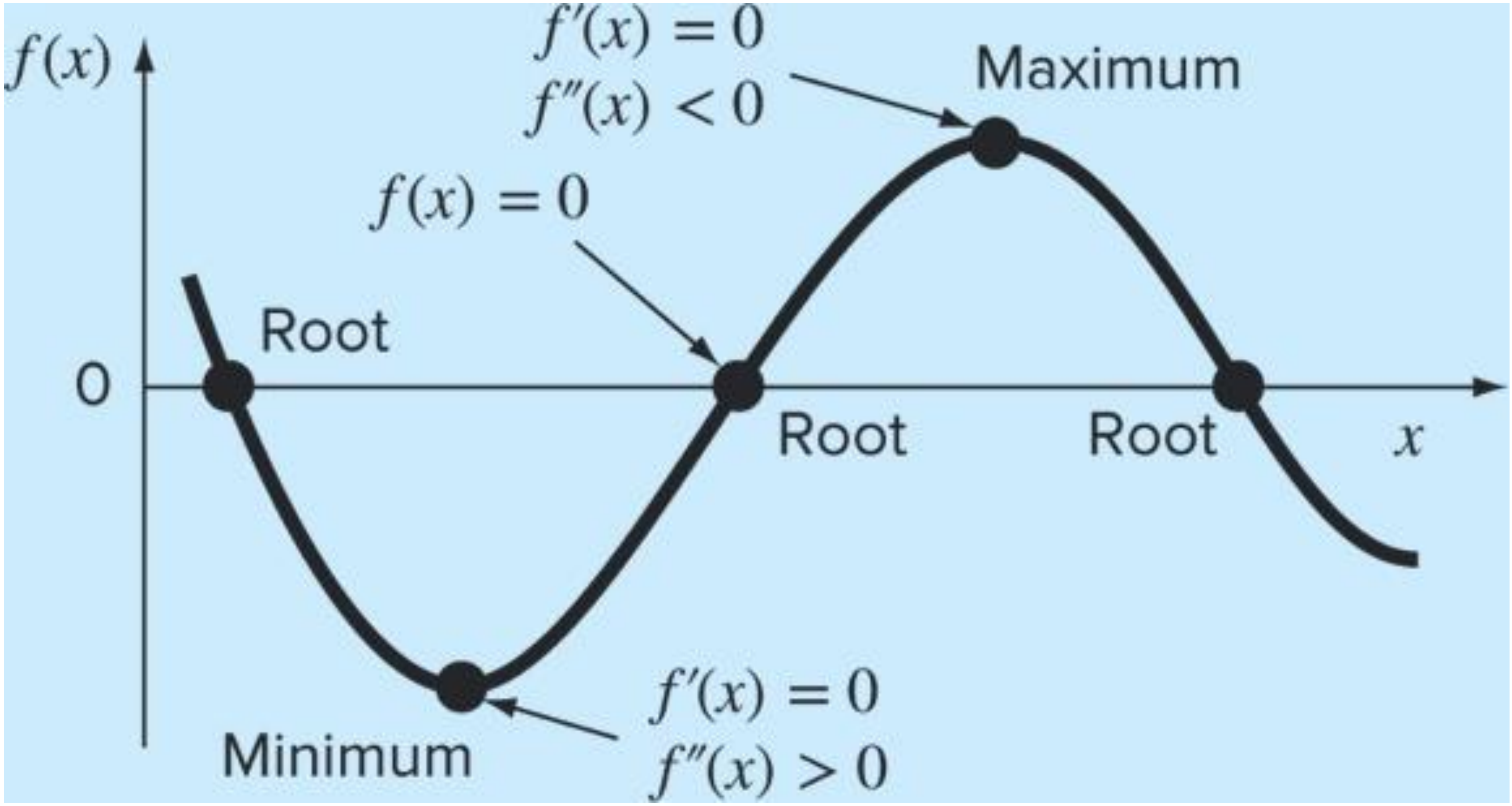
Chapter 5

Roots: Bracketing Methods

Numerical Methods
Fall 2019

Roots

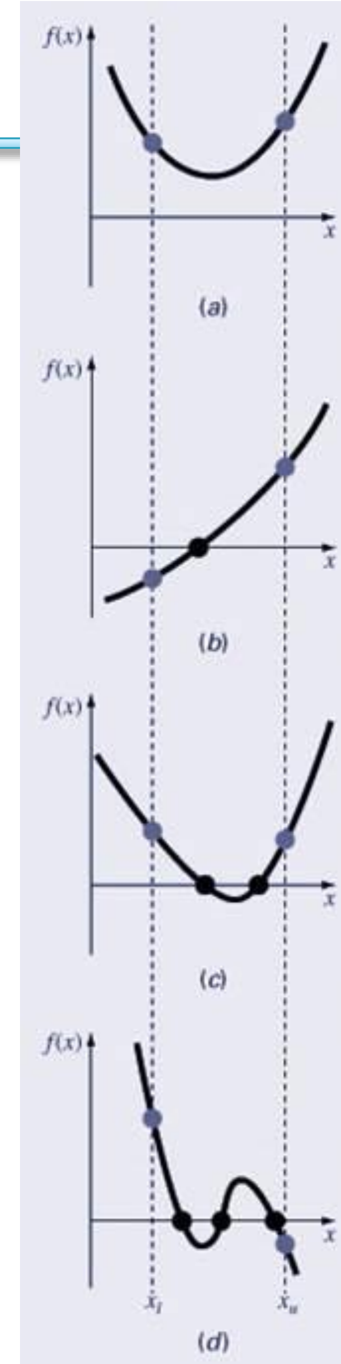
- "Roots" problems occur when some function *f*() can be written in terms of one or more dependent variables *x*, where the solutions to $f(x)=0$ yields the solution to the problem.



Graphical Methods

- A simple method for obtaining the estimate of the root of the equation $f(x)=0$ is to make a plot of the function and observe where it crosses the x-axis.
- Graphing the function can also indicate where roots may be and where some root-finding methods may fail:

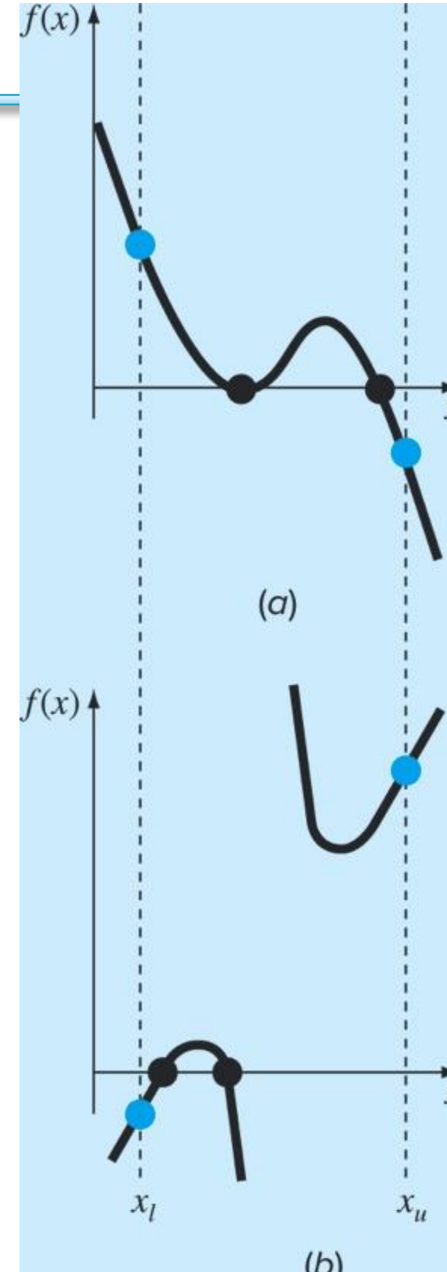
a) Same sign, no roots
b) Different sign, one root
c) Same sign, two roots
d) Different sign, three roots



Graphical Methods

- Some exceptions to the general cases

(a) Multiple roots that occur when the function is tangential to the x axis. End points are of opposite signs, there are an even number of axis interceptions
(b) Discontinuous functions where end points of opposite sign bracket an even number of roots.



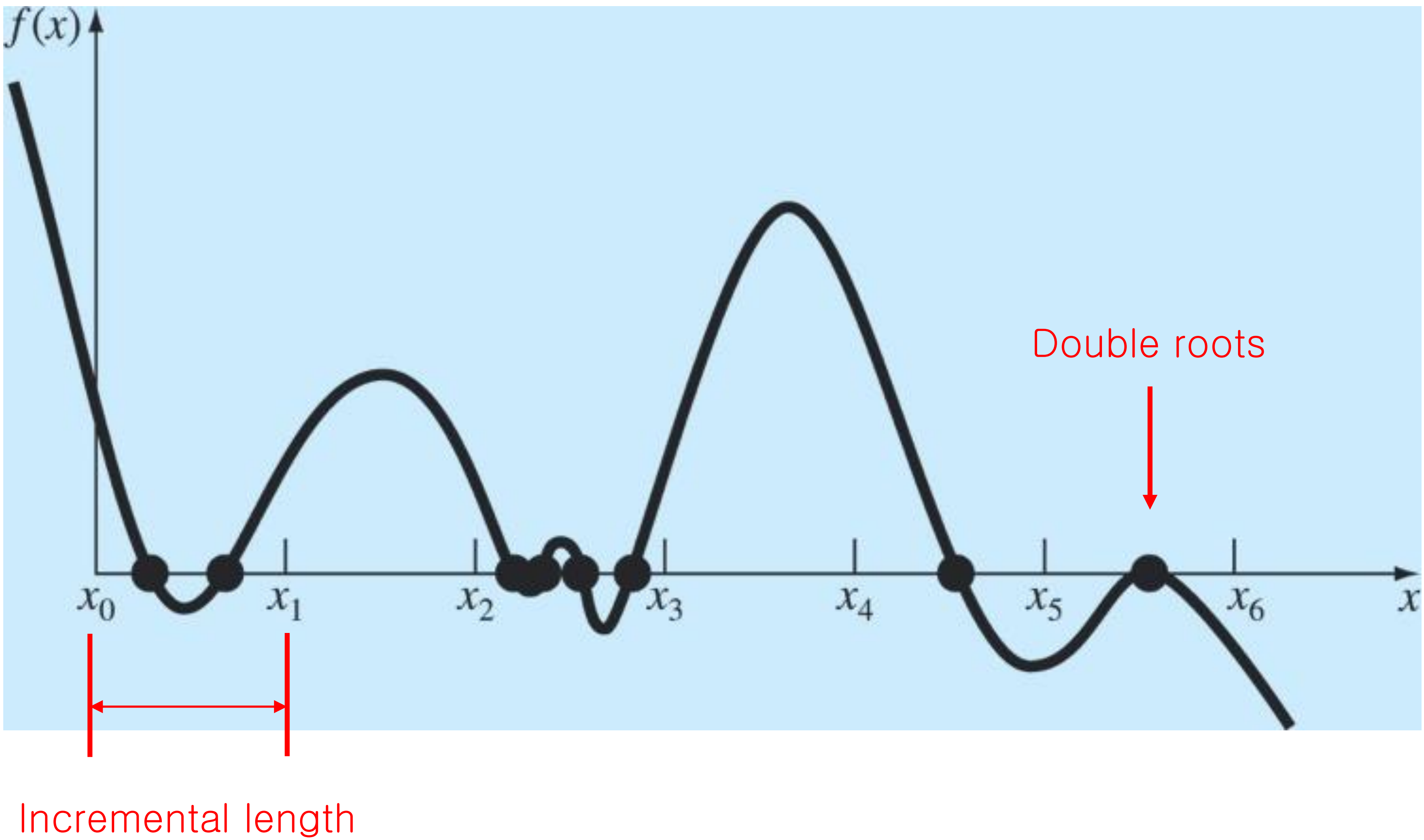
Bracketing Methods

- ***Bracketing methods*** are based on making two initial guesses that "bracket" the root – that is, are on either side of the root.
- Brackets are formed by finding two guesses x_l and x_u where the sign of the function changes; that is, where $f(x_l)\ f(x_u) < 0$
- The ***incremental search*** method tests the value of the function at evenly spaced intervals and finds brackets by identifying function sign changes between neighboring points.

Incremental Search Fails

- If the incremental length are too large, brackets may be missed due to capturing an even number of roots within two points.
- If the length is too small, the search can be very time consuming.
- Incremental searches cannot find brackets containing even-multiplicity roots regardless of spacing.

Incremental Search Fails



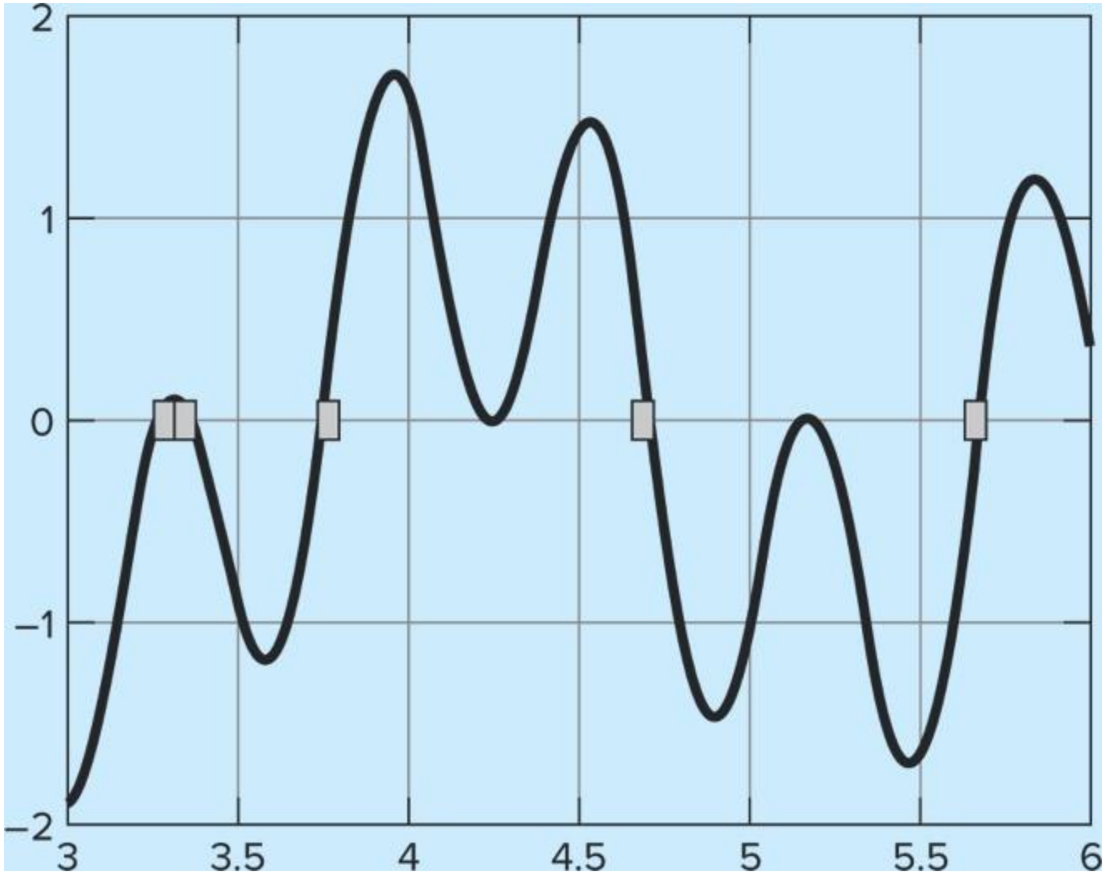
MATLAB code

```
function xb = incsearch(func,xmin,xmax,ns)
if nargin < 3, error('at least 3 arguments required'), end
if nargin < 4, ns = 50; end %if ns blank set to 50
% Incremental search
x = linspace(xmin,xmax,ns);
f = func(x);
nb = 0; xb = []; %xb is null unless sign change detected
for k = 1:length(x)-1
 if sign(f(k)) ~= sign(f(k+1)) %check for sign change
   nb = nb + 1;
   xb(nb,1) = x(k);
   xb(nb,2) = x(k+1);
 end
end
if isempty(xb) %display that no brackets were found
  disp('no brackets found')
  disp('check interval or increase ns')
else
  disp('number of brackets:') %display number of brackets
  disp(nb)
end
```

MATLAB example

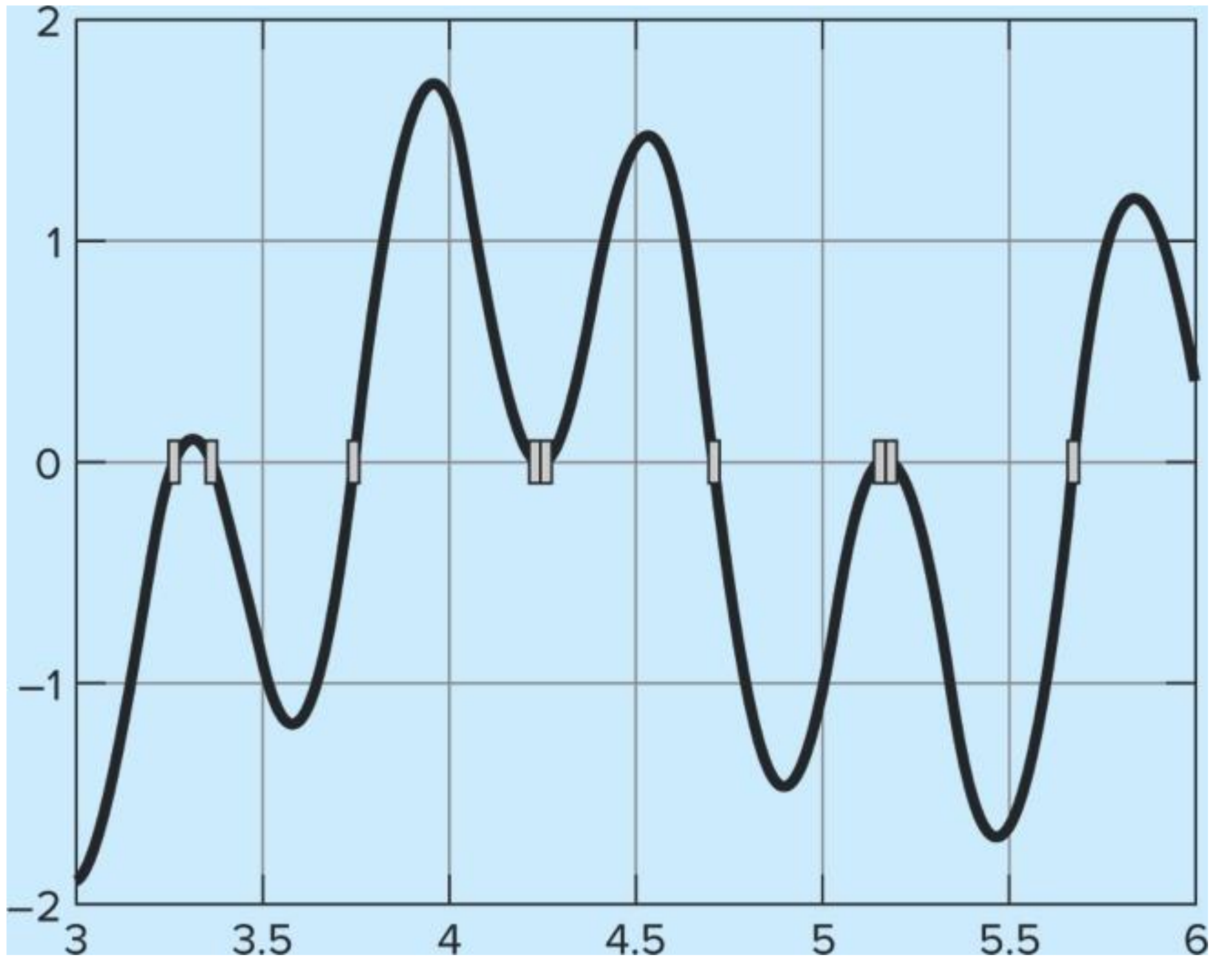
- EX: $f(x) = \sin(10x) + \cos(3x)$

```
>> incsearch(@(x) sin(10*x)+cos(3*x),3,6)
number of brackets:
5
ans =
3.2449 3.3061
3.3061 3.3673
3.7347 3.7959
4.6531 4.7143
5.6327 5.6939
```



MATLAB example

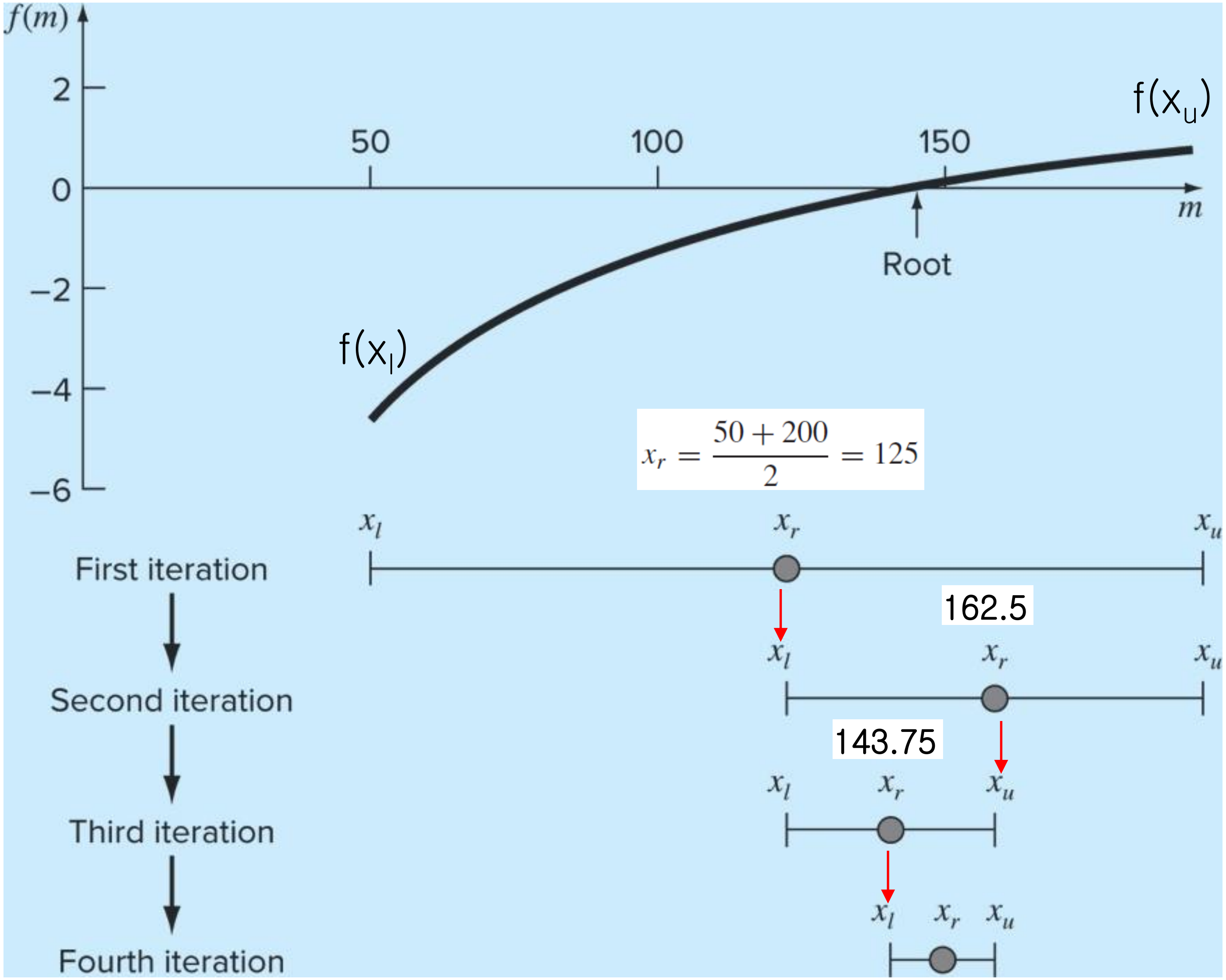
```
>> incsearch(@(x) sin(10*x)+cos(3*x),3,6,100)
number of brackets:
9
ans =
3.2424 3.2727
3.3636 3.3939
3.7273 3.7576
4.2121 4.2424
4.2424 4.2727
4.6970 4.7273
5.1515 5.1818
5.1818 5.2121
5.6667 5.6970
```



Bisection method

- The ***bisection method*** is a variation of the incremental search method in which the interval is always divided in half.
- If a function changes sign over an interval, the function value at the midpoint is evaluated.
- The location of the root is then determined as lying within the subinterval where the sign change occurs.
- The absolute error is reduced by a factor of 2 for each iteration.

Bisection method



Programming Bisection method

```
function [root,fx,ea,iter]=bisect(func,xl,xu,es,maxit,varargin)
if nargin<3,error('at least 3 input arguments required'),end
test = func(xl,varargin{:})*func(xu,varargin{:});
if test>0,error('no sign change'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
iter = 0; xr = xl; ea = 100;
while (1)
  xrold = xr;
  xr = (xl + xu)/2;
  iter = iter + 1;
  if xr ~= 0,ea = abs((xr - xrold)/xr) * 100;end
  test = func(xl,varargin{:})*func(xr,varargin{:});
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit,break,end
end
root = xr; fx = func(xr, varargin{:});
```

Programming Bisection method

- Percent relative error

$$|\varepsilon_a| = \left|\frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}}\right| 100\%$$

$\varepsilon_s = 0.5\%$.

Iteration	x_l	x_u	x_r	$\lvert\varepsilon_a\rvert$ (%)	$\lvert\varepsilon_t\rvert$ (%)
1	50	200	125		12.43
2	125	200	162.5	23.08	13.85
3	125	162.5	143.75	13.04	0.71
4	125	143.75	134.375	6.98	5.86
5	134.375	143.75	139.0625	3.37	2.58
6	139.0625	143.75	141.4063	1.66	0.93
7	141.4063	143.75	142.5781	0.82	0.11
8	142.5781	143.75	143.1641	0.41	0.30

Bisection Error

- The absolute error of the bisection method is solely dependent on the absolute error at the start of the process (the space between the two guesses) and the number of iterations:

$$E_a^0 = x_u^0 - x_l^0 = \Delta x^0$$

$$E_a^n = \frac{\Delta x^0}{2^n}$$

- The required number of iterations to obtain a particular absolute error can be calculated based on the initial guesses:

$$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right)$$

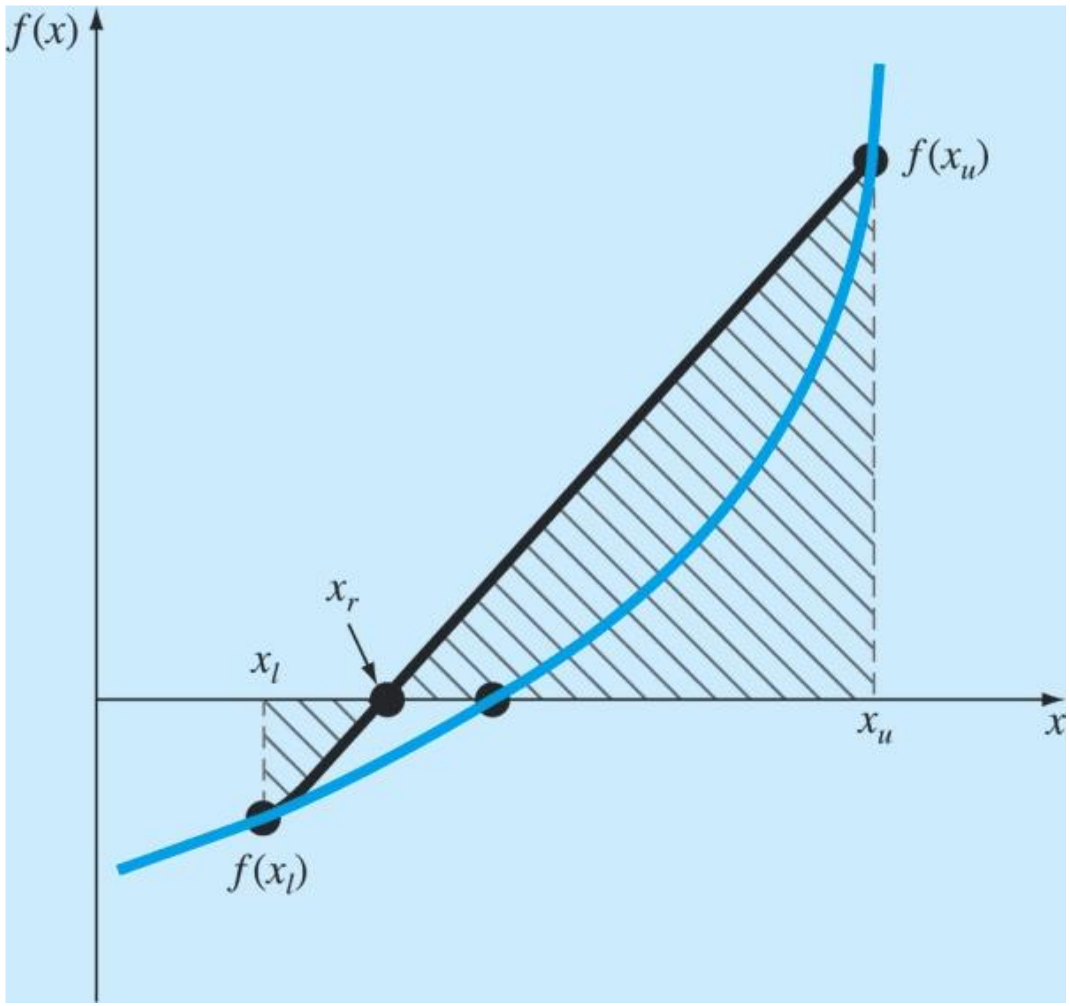
$E_{a,d}$ is the desired error,

False Position method

- The *false position* method is another bracketing method.
- It determines the next guess not by splitting the bracket in half but by connecting the endpoints with a straight line and determining the location of the intercept of the straight line (x_r).
- The value of x_r then replaces whichever of the two initial guesses yields a function value with the same sign as $f(x_r)$.

False Position Illustration

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



Bisection vs. False Position

- Bisection does not take into account the shape of the function; this can be good or bad depending on the function!
- Slow convergence of the false-position method

$$f(x) = x^{10} - 1$$