# 5

# Roots of Equations: Bracketing Methods

**CHAPTER OBJECTIVES**

The primary objective of this chapter is to acquaint you with bracketing methods for finding the root of a single nonlinear equation. Specific objectives and topics covered are

- Understanding what roots problems are and where they occur in engineering and science.
- Knowing how to determine a root graphically.
- Understanding the incremental search method and its shortcomings.
- Knowing how to solve a roots problem with the bisection method.
- Knowing how to estimate the error of bisection and why it differs from error estimates for other types of root location algorithms.
- Understanding false position and how it differs from bisection.

# How to Solve?

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

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

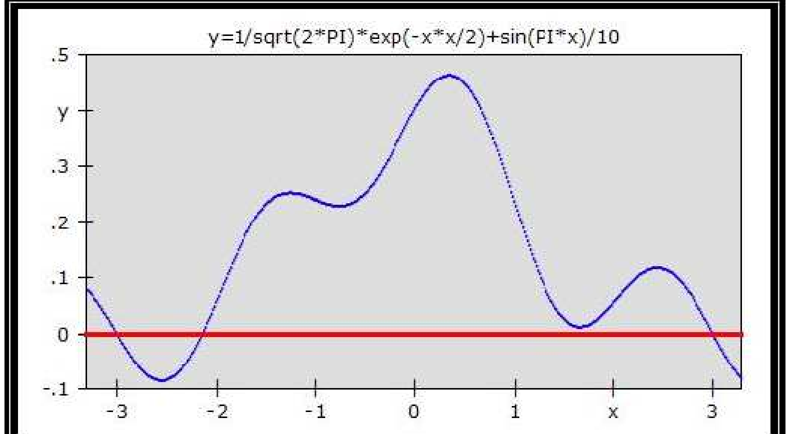
$$\sin x + x = 0 \quad \rightarrow x=?$$


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

$t=4$
$v=36$
$g=9.81$
$c_d=0.25$

$m=?$

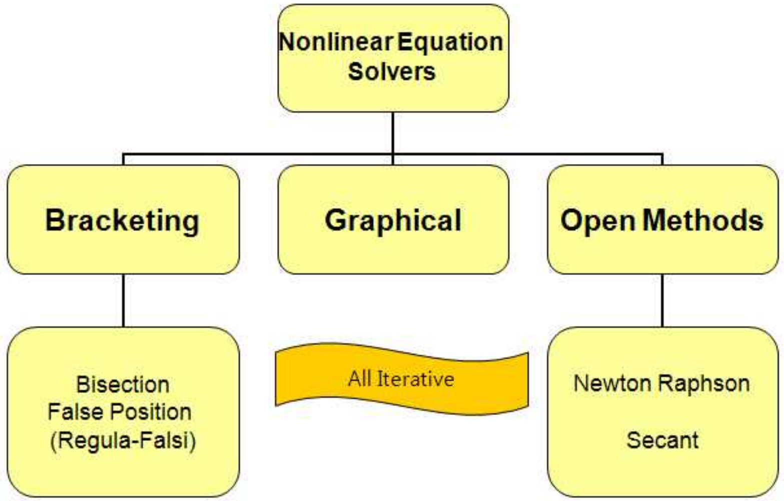
$$f(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2}+\frac{\sin(\pi x)}{10}=0$$




A nonlinear equation, with three solutions of x for f(x)=0 within the given range of -3.3 to +3.3.
The three solutions are x=-2.985, -2.134, and 3.014.

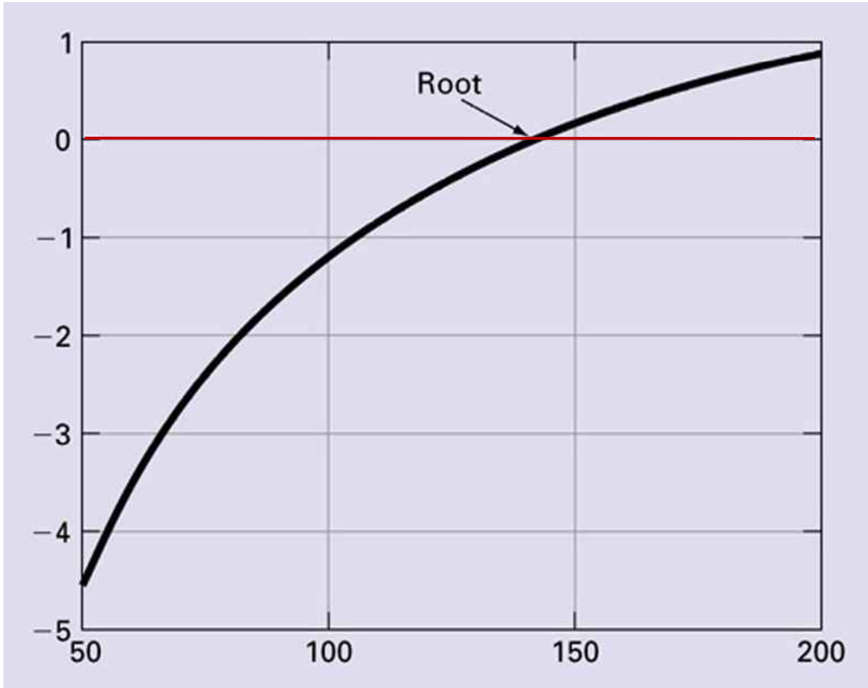
# Techniques for Finding Roots

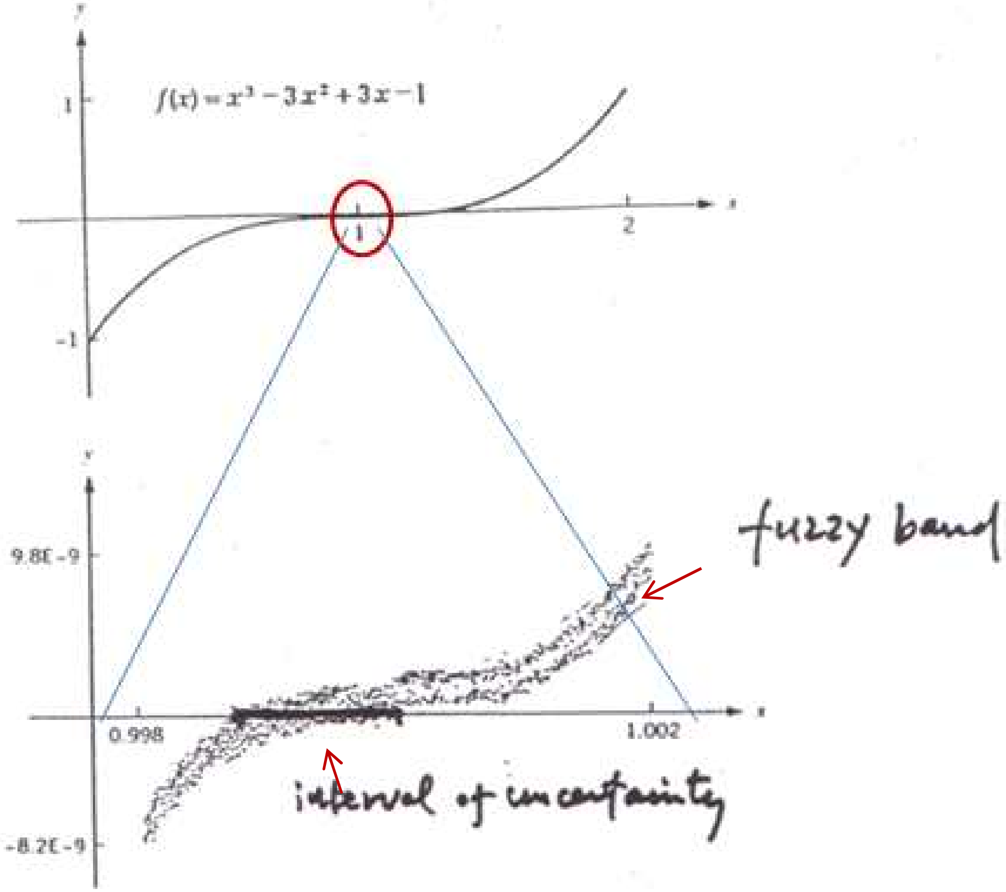
- Graphical Methods
- Bracketing Methods : $[x_l, x_u]$
- Open Methods : $x_0$



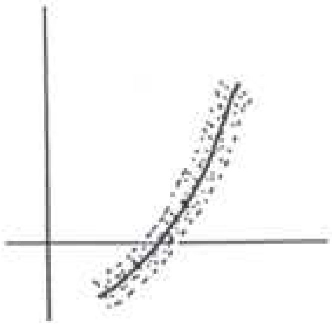
# Graphical Methods

- Good : easy
- Bad : precision (used as initial point)

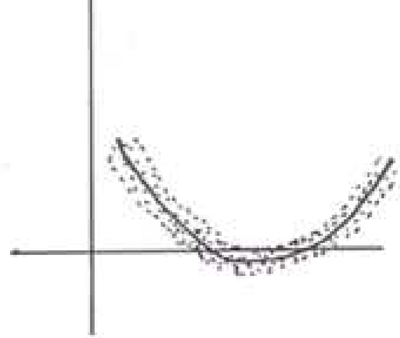



$f(x) = x^3 - 3x^2 + 3x - 1$
1
-1
2
9.8E-9
-8.2E-9
0.998
1.002
fuzzy band
interval of uncertainty

Good



Bad



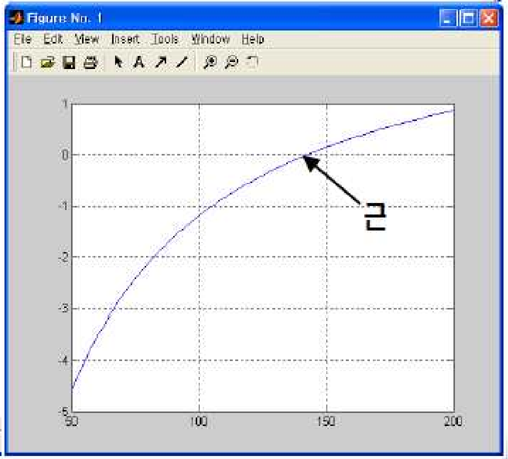
# 예제 5.1

Q. 자유낙하 4초 후의 속도를 36 m/s로 되게 하는 번지 점프하는 사람의 질량을 그래프적인 접근법으로 구하라.

(항력계수는 0.25 kg/m이고, 중력가속도는 9.81 m/s²이다.)

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

```
>> cd = 0.25; g = 9.81; v = 36; t = 4;
>> mp = linspace (50, 200);
>> fp = sqrt(g*mp/cd).*tanh(sqrt(g*cd./mp)*t) - v;
>> plot(mp,fp), grid
```



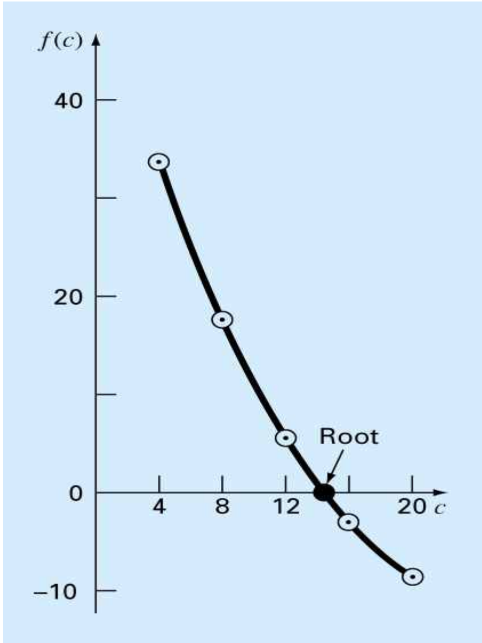

```
>> sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)-v
ans =
   0.0456
>> sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)
ans =
   36.0456
```

y = linspace(a,b) generates a row vector y of 100 points linearly spaced between and including a and b.
y = linspace(a,b,n) generates a row vector y of n points linearly spaced between and including a and b.
For n < 2, linspace returns b.
grid on adds major grid lines to the current axes.

# Bracketing Methods
## (Or, two point methods for finding roots)

- Find roots between a lower bound $[x_l, x_u]$

- Initial guess needed (incremental search).
- Always work if roots exist between the two limits.
- Converge slower than open methods



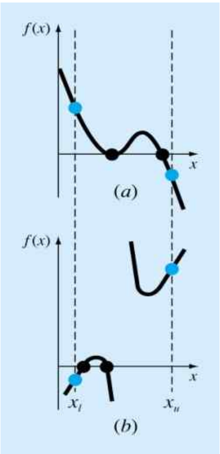
# Conditions

a) $f(x_l)*f(x_u) > 0$, *no root.*

b) $f(x_l)*f(x_u) < 0$, *one root.*

c) $f(x_l)*f(x_u) > 0$, *even number* of roots.

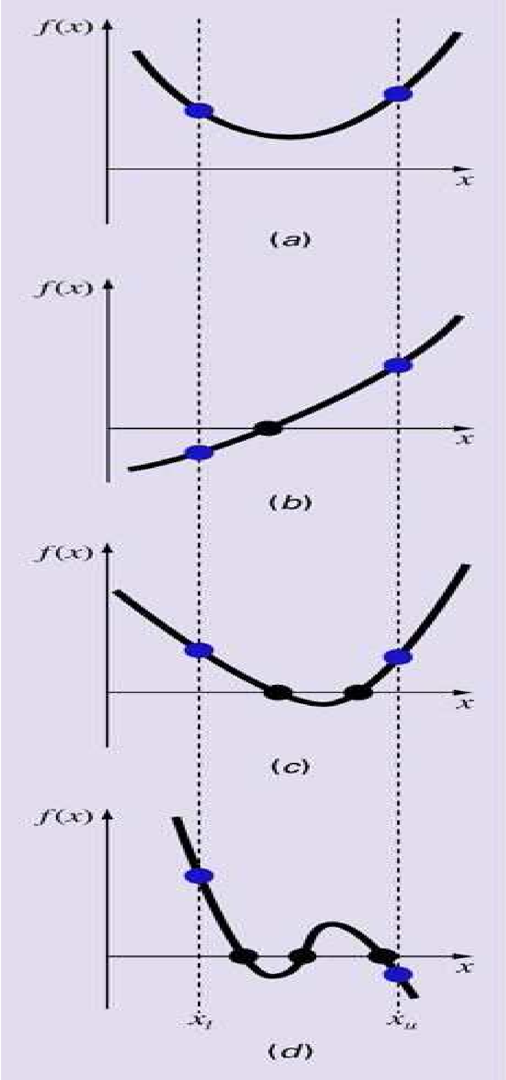
d) $f(x_l)*f(x_u) < 0$, *odd number* of roots.




Two roots (might work for a while!!)

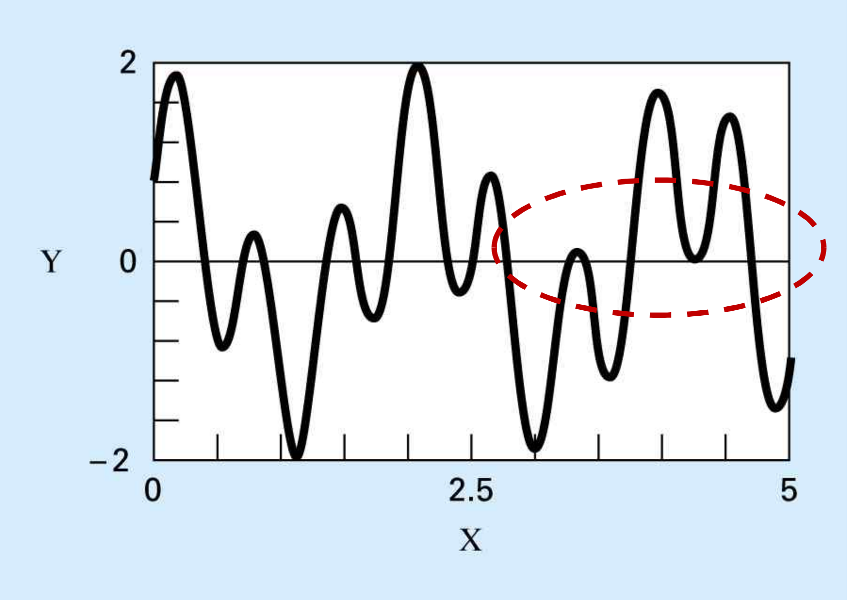
Discontinuous function

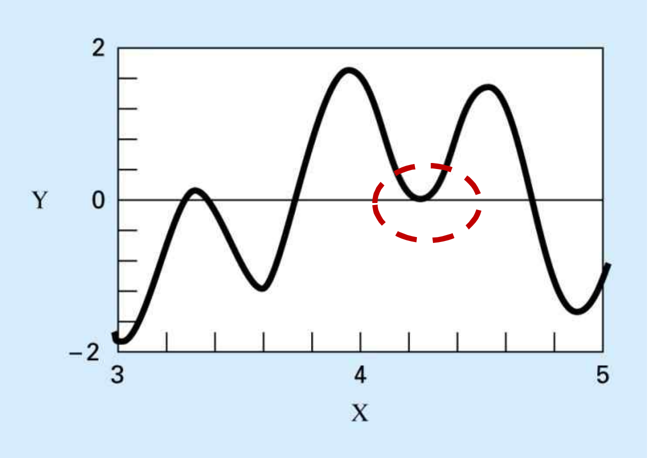
(need special method)

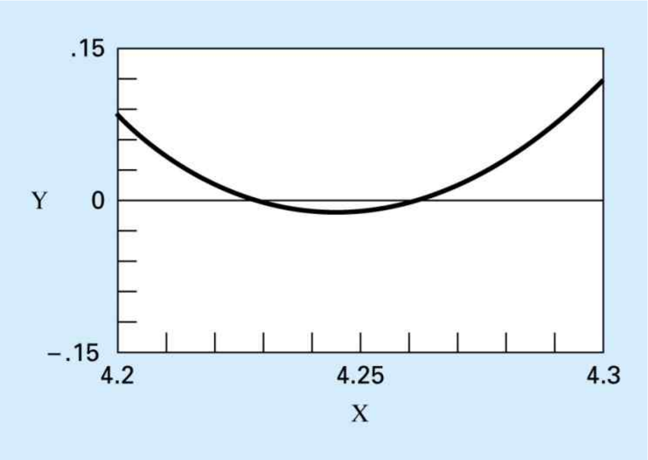


• MANY-MANY roots. What do we do?

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



# Incremental Search

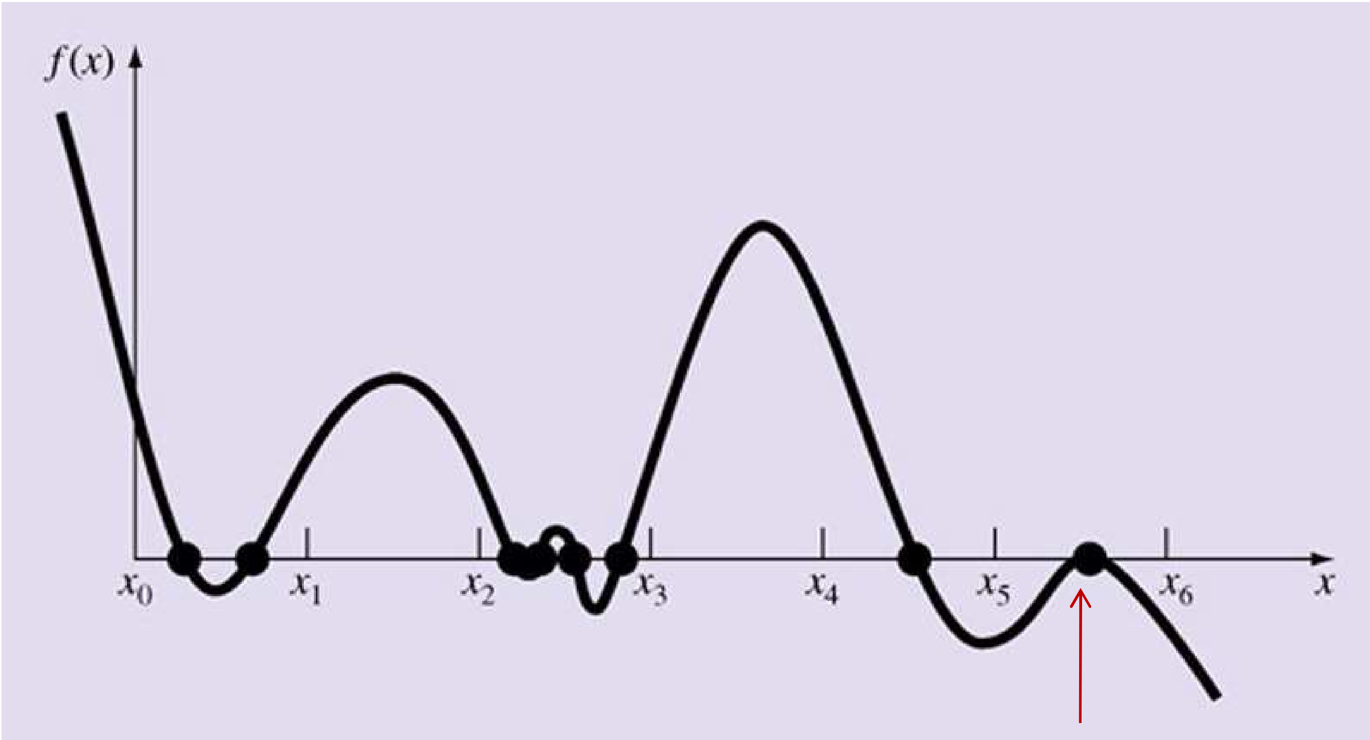
- Goal : find $x_l$ *and* $x_u$ *such that* $f(x_l)*f(x_u)<0$

- Range of search must be decided.
- Large step size may miss some roots.
- Small step size increases computation time.

# Example

- Cases where roots could be missed because the incremental length of the search procedure is too large.

- Note that the last root on the right is multiple and would be missed regardless of the increment length.



# Example

- *sin(10x)+cos(3x)*

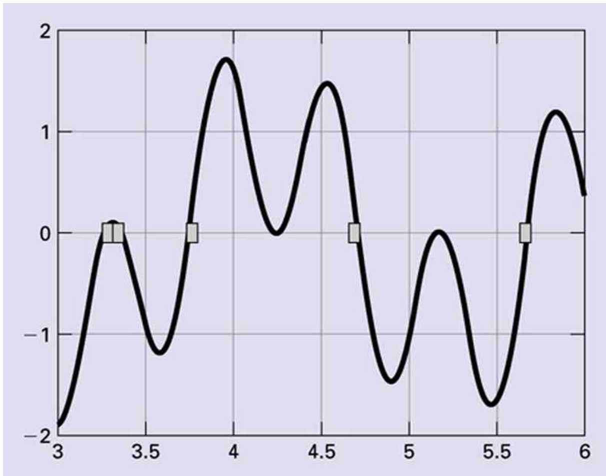
- 50 subintervals from [3,6]
  - by MATLAB incremental search
  - *5 subintervals to have opposite sign*

3.2449, 3.3061
3.3061, 3.3673
3.7347, 3.7959
4.6531, 4.7143
5.6327, 5.6939

```
>> incsearch(inline('sin(10*x)+cos(3*x)'), 3, 6)
nb =
     0      (초기값)
number of brackets:
     5      (Search 결과 값)
ans =
    3.2449    3.3061
    3.3061    3.3673
    3.7347    3.7959
    4.6531    4.7143
    5.6327    5.6939
```

function xb = incsearch(func, xmin, xmax, ns)

ns = number of subintervals (default = 50)



# MATLAB: inline

- inline(expr) constructs an inline function object from expression contained in the string expr

```
f = inline('3*sin(2*x.^2)')

f =
     Inline function:
     f(x) = 3*sin(2*x.^2)

argnames(f)

ans =
    'x'

formula(f)
ans =

3*sin(2*x.^2)
```

```
g = inline('sin(alpha*x)','x','alpha')

g =

     Inline function:
     g(x,alpha) = sin(alpha*x)
```

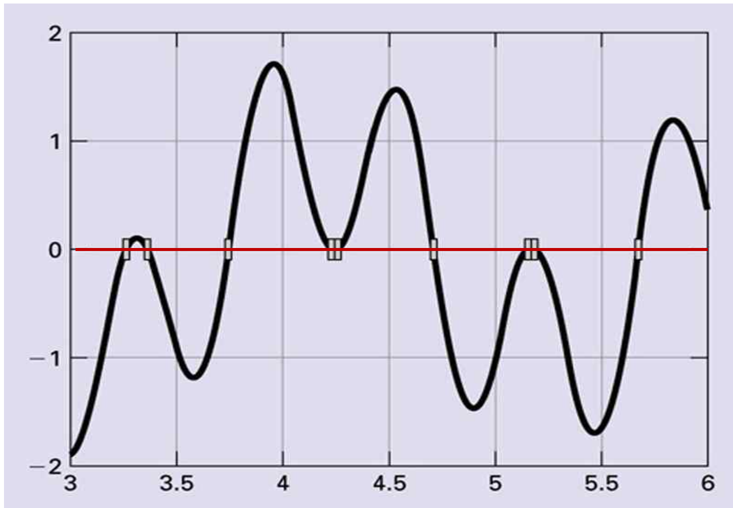
# Example (cont.)

- *sin(10x)+cos(3x)*

- 100 subintervals;
  - *9 subintervals to have opposite sign*

  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

```
>> incsearch(inline('sin(10*x)+cos(3*x)'),3,6,100)
nb =
    0
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
```



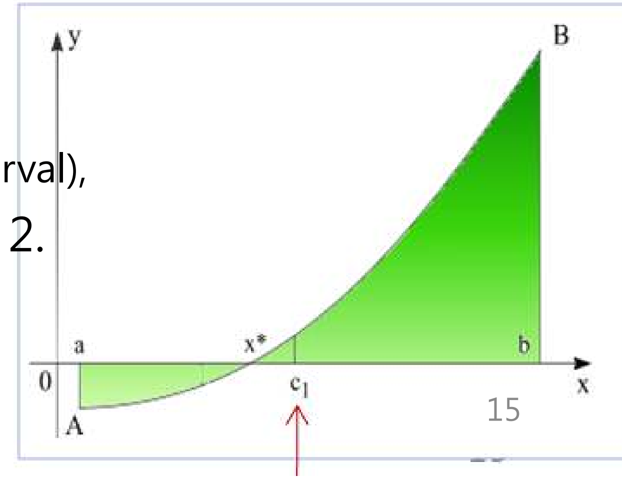
# The Bisection Method

- Iterative : every iteration reduces the bound by half.
- For the arbitrary equation of one variable, $f(x)=0$

1. Pick $x_l$ and $x_u$ such that they bound the root of interest, check if $f(x_l)\cdot f(x_u) < 0$
2. Estimate the root by evaluating $f[(x_l + x_u)/2]$
3. Find the pair

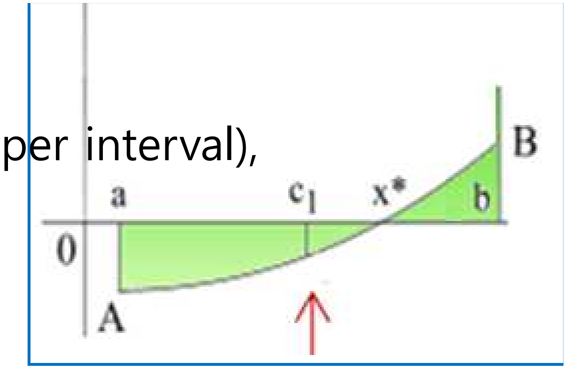
   (3-1) If $f(x_l)*f[(x_l+ x_u)/2]<0$, (root lies in the lower interval),

   then $x_u = (x_l + x_u)/2$ and go to step 2.



(3-2) If $f(x_l)*f[(x_l+x_u)/2]>0$, (root lies in the upper interval),

then $x_l= [(x_l+x_u)/2]$, go to step 2.




(3-3) If $f(x_l)*f[(x_l+x_u)/2]=0$,

then root is $[(x_l+x_u)/2]$ and terminate.

4. Compare $e_s$ with $e_a$ ($e_s$ : error bound (stopping condition))

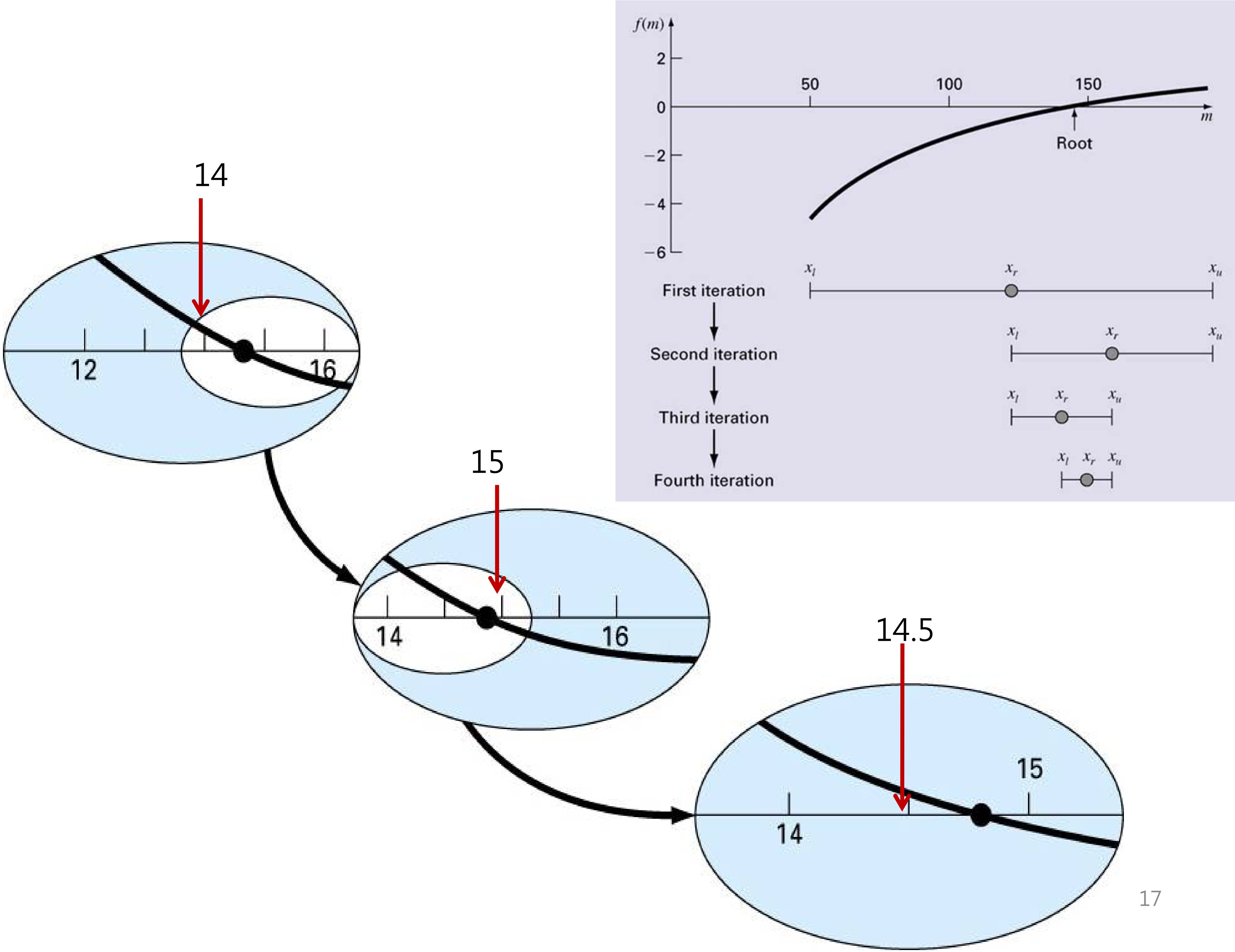
근의 참값을 모르므로 $|\varepsilon_t|$ 를 이용할 수 없다.

근사 상대오차,

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

$$\frac{\left| x_l - \frac{x_l + x_u}{2} \right|}{\left| \frac{x_l + x_u}{2} \right|} \ or \ \frac{\left| x_u - \frac{x_l + x_u}{2} \right|}{\left| \frac{x_l + x_u}{2} \right|}$$

5. If $e_a < e_s$, stop; otherwise repeat the process.


f(m)
2
0
−2
−4
−6
50
100
150
m
Root
First iteration
Second iteration
Third iteration
Fourth iteration
14
12
16
15
14
16
14.5
14
15

# Evaluation of Method

Pros

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

$$n = 1 + \log_2 \frac{\Delta x^0}{E_{a,d}}$$

$\Delta x^0$: initial bound
$E_{a,d}$: desired $E_a$

Cons

- Slow
- Must know *a* and *b* that bound root
- May multiple roots
- No account is taken of $f(x_l)$ and $f(x_u)$,
  - if $f(x_l)$ is closer to zero, it is likely that root is closer to $x_l$ .

# How Many Iterations will It Take?

- Length of the first Interval : $L_o = b - a$
  - After 1 iteration : $L_1 = L_o/2$
  - After 2 iterations : $L_2 = L_o/4$
  - After k iterations : $L_k = L_o/2^k$

- If $L_o = 2$ and the absolute magnitude of the error ($L_k$) is $10^{-4}$

  how many iterations will you have to do to get the required accuracy in the solution?

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

$$n = 1 + \log_2 \frac{\Delta x^0}{E_{a,d}}$$

$\Delta x^0$: initial bound

$E_{a,d}$: desired $E_a$

$L_k/x_k = \varepsilon_a \rightarrow L_k = (\varepsilon_a)(x_k) = 10^{-4}$

```
function root = bisection(func,xl,xu,es,maxit)
% bisection(func,xl,xu,es,maxit):
%   uses 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
  error('no bracket')   %change, display an error message
  return                %and terminate
end
% if necessary, assign default values
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;
```

bisection('func', xl, xu, es, maxit)

```
>> function y = func(x)
y = sin(x);                    % save as func.m
```

Command Window

File Edit View Web Window Help

```
>> bisection('func1',50,200,0.001,50)
    1   125
    2   162.5000
    3   143.7500
    4   134.3750
    5   139.0625
    6   141.4063
    7   142.5781
     ...
   16   142.7361
   17   142.7372
ans =
  142.7372
```

Ready

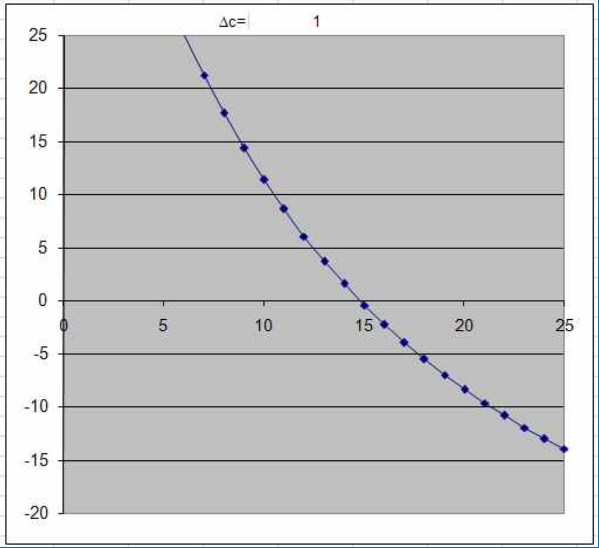
bisection(inline('sin(10*x)+cos(3*x)'), 3.2, 3.3, 0.0001, 50)

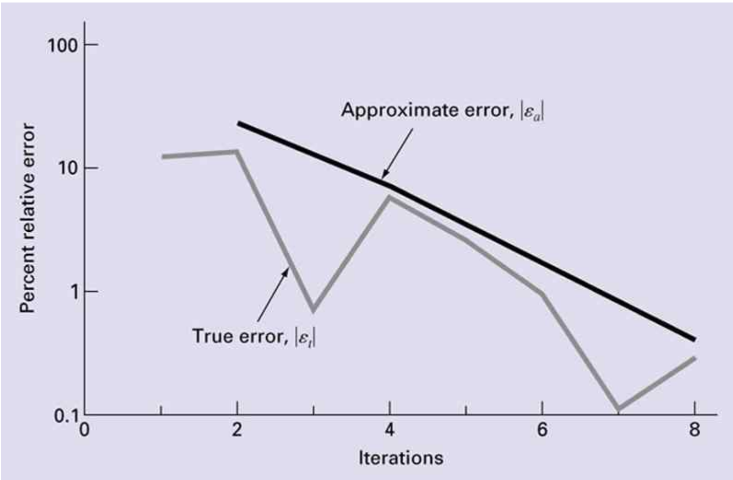
| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| | | | | | | m= | 68.1 |
| | | | | | | g= | 9.8 |
| | | | | | | t= | 10 |
| | | | | | | v= | 40 |

$$v = \frac{gm}{c}\left[1 - e^{-(c/m)t}\right]$$

What is the value of c when
v=40 m/sec and t=10 sec?

| c | f( c) |
|---|---|
| 1 | 51.14433 |
| 2 | 44.92053 |
| 3 | 39.26338 |
| 4 | 34.11484 |
| 5 | 29.42323 |
| 6 | 25.14245 |
| 7 | 21.23141 |
| 8 | 17.65343 |
| 9 | 14.37575 |
| 10 | 11.3691 |
| 11 | 8.607309 |
| 12 | 6.066936 |
| 13 | 3.727002 |
| 14 | 1.568699 |
| 15 | -0.42484 |
| 16 | -2.26876 |
| 17 | -3.97667 |
| 18 | -5.56079 |
| 19 | -7.03214 |
| 20 | -8.40063 |
| 21 | -9.67519 |
| 22 | -10.8639 |
| 23 | -11.974 |
| 24 | -13.0121 |
| 25 | -13.9842 |




| | xlow | xhigh | xr | flow | fhigh | f(xr) | ea(%) | et(%) |
|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 16 | 14 | 6.06694 | -2.2688 | 1.5687 | | 5.27871 |
| 2 | 14 | 16 | 15 | 1.5687 | -2.2688 | -0.4248 | 6.66667 | 1.4871 |
| 3 | 14 | 15 | 14.5 | 1.5687 | -0.4248 | 0.55232 | 3.44828 | 1.8958 |
| 4 | 14.5 | 15 | 14.75 | 0.55232 | -0.4248 | 0.05895 | 1.69492 | 0.20435 |
| 5 | 14.75 | 15 | 14.875 | 0.05895 | -0.4248 | -0.1841 | 0.84034 | 0.64137 |
| 6 | 14.75 | 14.875 | 14.8125 | 0.05895 | -0.1841 | -0.0629 | 0.42194 | 0.21851 |
| 7 | 14.75 | 14.8125 | 14.7813 | 0.05895 | -0.0629 | -0.002 | 0.21142 | 0.00708 |
| 8 | 14.75 | 14.7813 | 14.7656 | 0.05895 | -0.002 | 0.02844 | 0.10582 | 0.09864 |
| 9 | 14.7656 | 14.7813 | 14.7734 | 0.02844 | -0.002 | 0.01319 | 0.05288 | 0.04578 |
| 10 | 14.7734 | 14.7813 | 14.7773 | 0.01319 | -0.002 | 0.00558 | 0.02643 | 0.01935 |
| 11 | 14.7773 | 14.7813 | 14.7793 | 0.00558 | -0.002 | 0.00177 | 0.01322 | 0.00614 |
| 12 | 14.7793 | 14.7813 | 14.7803 | 0.00177 | -0.002 | -0.0001 | 0.00661 | 0.00047 |
| 13 | 14.7793 | 14.7803 | 14.7798 | 0.00177 | -0.0001 | 0.00082 | 0.0033 | 0.00283 |
| 14 | 14.7798 | 14.7803 | 14.78 | 0.00082 | -0.0001 | 0.00034 | 0.00165 | 0.00118 |
| 15 | 14.78 | 14.7803 | 14.7802 | 0.00034 | -0.0001 | 0.0001 | 0.00083 | 0.00035 |
| 16 | 14.7802 | 14.7803 | 14.7802 | 0.0001 | -0.0001 | -2E-05 | 0.00041 | 5.8E-05 |
| 17 | 14.7802 | 14.7802 | 14.7802 | 0.0001 | -2E-05 | 4.3E-05 | 0.00021 | 0.00015 |
| 18 | 14.7802 | 14.7802 | 14.7802 | 4.3E-05 | -2E-05 | 1.3E-05 | 0.0001 | 4.5E-05 |
| 19 | 14.7802 | 14.7802 | 14.7802 | 1.3E-05 | -2E-05 | -2E-06 | 5.2E-05 | 6.4E-06 |
| 20 | 14.7802 | 14.7802 | 14.7802 | 1.3E-05 | -2E-06 | 5.6E-06 | 2.6E-05 | 1.9E-05 |
| 21 | 14.7802 | 14.7802 | 14.7802 | 5.6E-06 | -2E-06 | 1.9E-06 | 1.3E-05 | 6.5E-06 |
| 22 | 14.7802 | 14.7802 | 14.7802 | 1.9E-06 | -2E-06 | 2.4E-08 | 6.5E-06 | 8.5E-08 |
| 23 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -2E-06 | -9E-07 | 3.2E-06 | 3.1E-06 |
| 24 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -9E-07 | -4E-07 | 1.6E-06 | 1.5E-06 |
| 25 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -4E-07 | -2E-07 | 8.1E-07 | 7.2E-07 |
| 26 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -2E-07 | -9E-08 | 4E-07 | 3.2E-07 |
| 27 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -9E-08 | -3E-08 | 2E-07 | 1.2E-07 |
| 28 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -3E-08 | -5E-09 | 1E-07 | 1.6E-08 |
| 29 | 14.7802 | 14.7802 | 14.7802 | 2.4E-08 | -5E-09 | 9.6E-09 | 5E-08 | 3.5E-08 |
| 30 | 14.7802 | 14.7802 | 14.7802 | 9.6E-09 | -5E-09 | 2.4E-09 | 2.5E-08 | 9.5E-09 |
| 31 | 14.7802 | 14.7802 | 14.7802 | 2.4E-09 | -5E-09 | -1E-09 | 1.3E-08 | 3.2E-09 |
| 32 | 14.7802 | 14.7802 | 14.7802 | 2.4E-09 | -1E-09 | 5.4E-10 | 6.3E-09 | 3.2E-09 |
| 33 | 14.7802 | 14.7802 | 14.7802 | 5.4E-10 | -1E-09 | -4E-10 | 3.2E-09 | 0 |

SOLUTION

$\boxed{14.7802}$

# False Position Method

- Similar to bisection but determine the next bound by line equation.
  - Use more information

- Algorithm: repeat the following

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

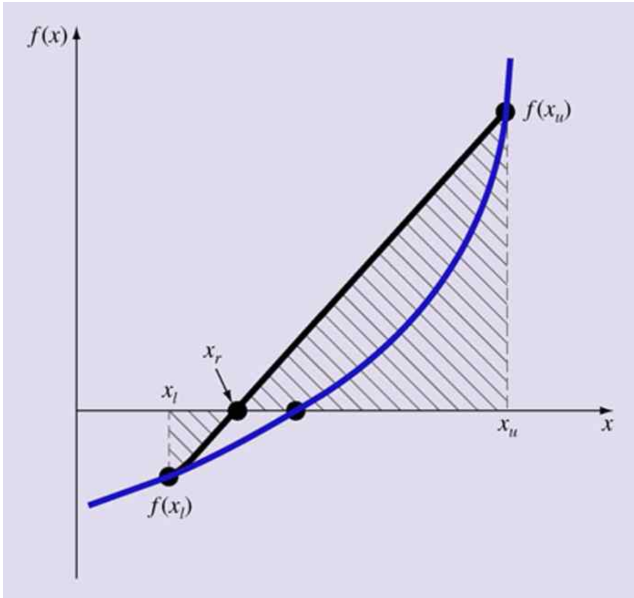
if $f(x_r)f(x_u)<0$

$\quad x_u = x_r$

else

$\quad x_l = x_r$

end




$$y - f(b_k) = \frac{f(b_k) - f(a_k)}{b_k - a_k}(x - b_k).$$

We now choose $c_k$ to be the root of this line, so $c$ is chosen such that

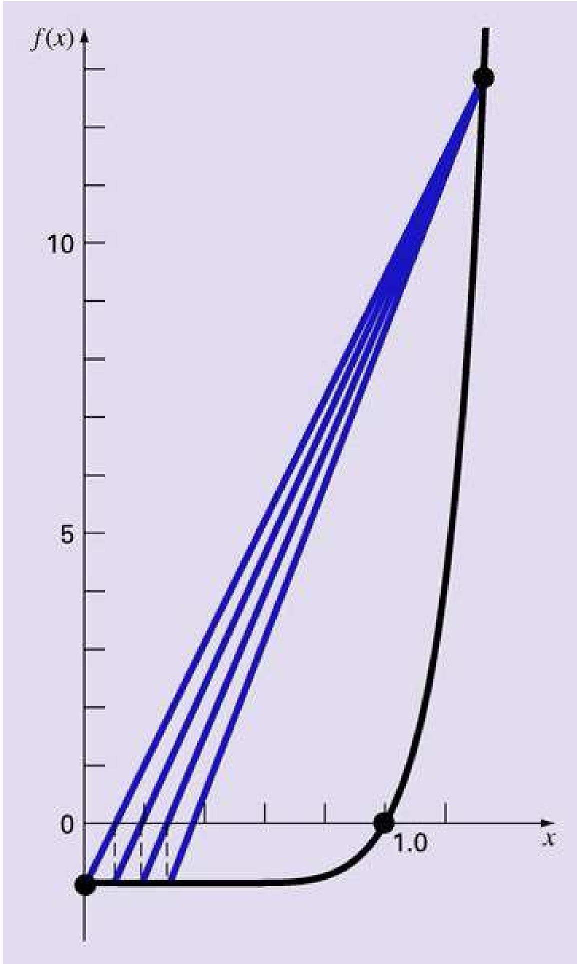
$$f(b_k) + \frac{f(b_k) - f(a_k)}{b_k - a_k}(c_k - b_k) = 0.$$

Solving this equation gives the above equation for $c_k$.

$$c_k = \frac{f(b_k)a_k - f(a_k)b_k}{f(b_k) - f(a_k)}$$

# False Position (cont.)

- Usually converge faster than bisection
  - but not always.
  - Example: $f(x)=x^{10}-1$



Bisection과 False position 을 사용해서 x = 0과 1.3 사이에서 $f(x) = x^{10} - 1$ 의 근을 구하라.

## Bisection

| 반복 | $x_l$ | $x_u$ | $x_r$ | $\varepsilon_a(\%)$ | $\varepsilon_t(\%)$ |
|---|---|---|---|---|---|
| 1 | 0 | 1.3 | 0.65 | 100.0 | 35.0 |
| 2 | 0.65 | 1.3 | 0.975 | 33.3 | 2.5 |
| 3 | 0.975 | 1.3 | 1.1375 | 14.3 | 13.8 |
| 4 | 0.975 | 1.1375 | 1.05625 | 7.7 | 5.6 |
| 5 | 0.975 | 1.05625 | 1.015625 | 4.0 | 1.6 |

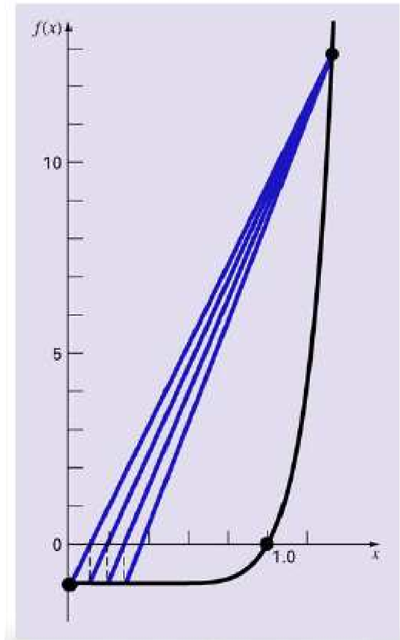
```
>> bisectnew(inline('x^10-1'), 0, 1.5, 0.0001)

ans =

    1.0000
```

## False position

| 반복 | $x_l$ | $x_u$ | $x_r$ | $\varepsilon_a(\%)$ | $\varepsilon_t(\%)$ |
|---|---|---|---|---|---|
| 1 | 0 | 1.3 | 0.09430 | | 90.6 |
| 2 | 0.09430 | 1.3 | 0.18176 | 48.1 | 81.8 |
| 3 | 0.18176 | 1.3 | 0.26287 | 30.9 | 73.7 |
| 4 | 0.26287 | 1.3 | 0.33811 | 22.3 | 66.2 |
| 5 | 0.33811 | 1.3 | 0.40788 | 17.1 | 59.2 |



# 6

# Roots of Equations: Open Methods

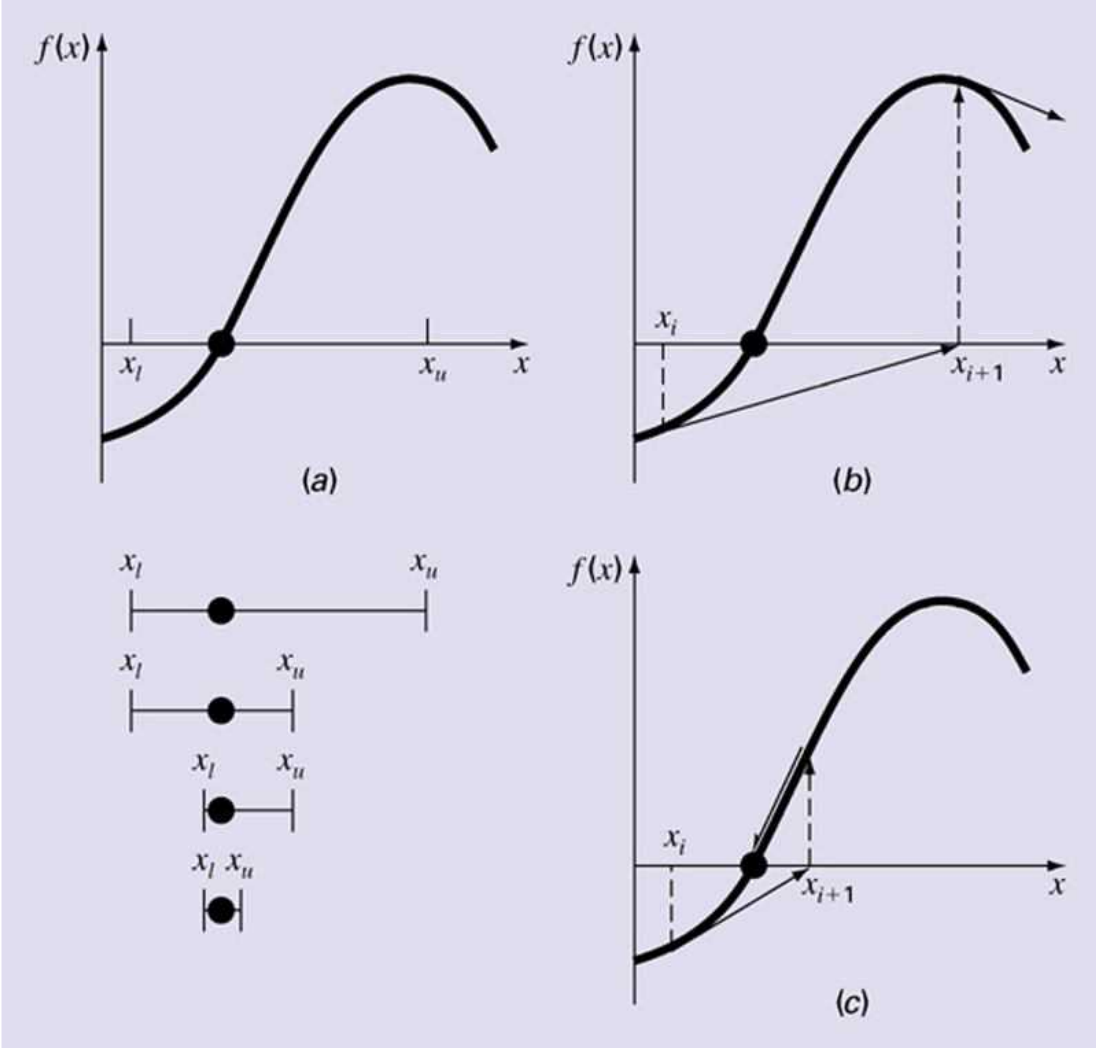
**CHAPTER OBJECTIVES**

The primary objective of this chapter is to acquaint you with open methods for finding the root of a single nonlinear equation. Specific objectives and topics covered are

- Recognizing the difference between bracketing and open methods for root location.
- Understanding the fixed-point iteration method and how you can evaluate its convergence characteristics.
- Knowing how to solve a roots problem with the Newton-Raphson method and appreciating the concept of quadratic convergence.
- Knowing how to implement both the secant and the modified secant methods.
- Knowing how to use MATLAB's `fzero` function to estimate roots.
- Learning how to manipulate and determine the roots of polynomials with MATLAB.

# Open Methods

- One initial guess.
- Convergence not guaranteed.
- Usually faster than bracketing methods.



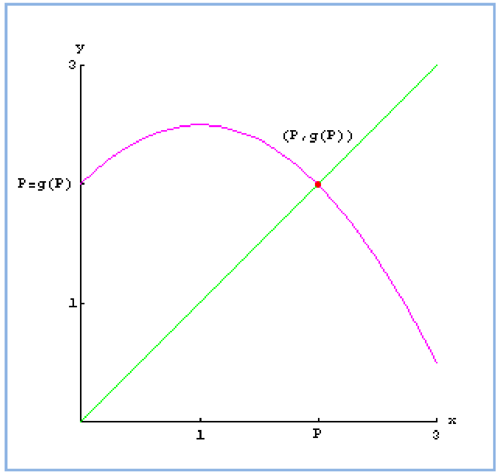
# Open Methods

- Simple Fixed-Point Iteration
- Newton-Raphson Method
- Secant Methods
- Modified Secant Method

# Simple Fixed-point Iteration

- Rearrange the function so that x is on the left side of the equation:

$$f(x)=0 \quad \rightarrow \quad g(x)=x$$
$$x_k = g(x_{k-1}) \qquad x_o \text{ given, } k=1,2,\ldots$$




- Bracketing methods are "convergent".
- Fixed-point methods may sometime "diverge", depending on the stating point (initial guess) and how the function behaves.

# Example

$$f(x) = x^2 - x - 2 \qquad x \geq 0$$

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

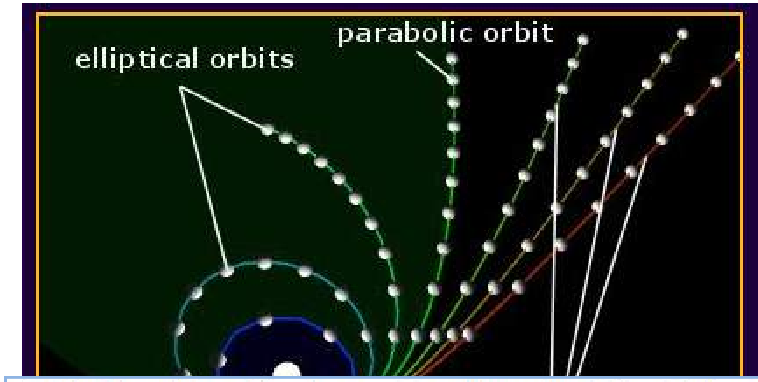
$$or$$

$$g(x) = \sqrt{x+2}$$

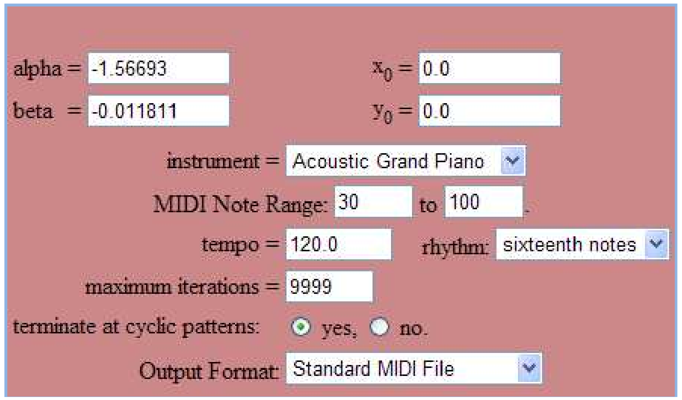
$$or$$

$$g(x) = 1 + \frac{2}{x}$$

$$\vdots$$

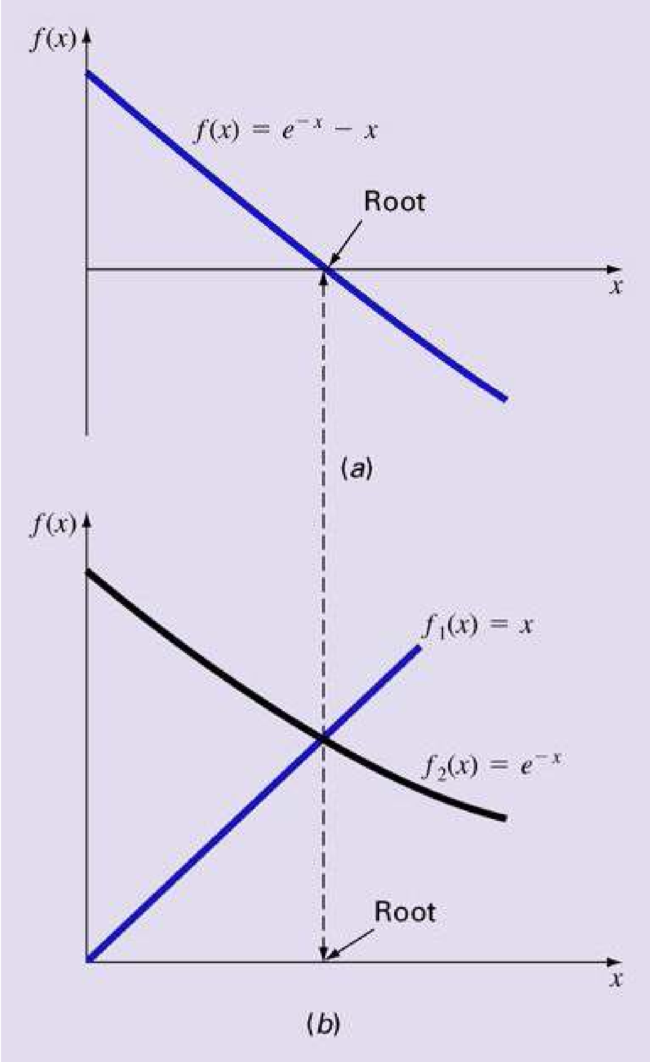


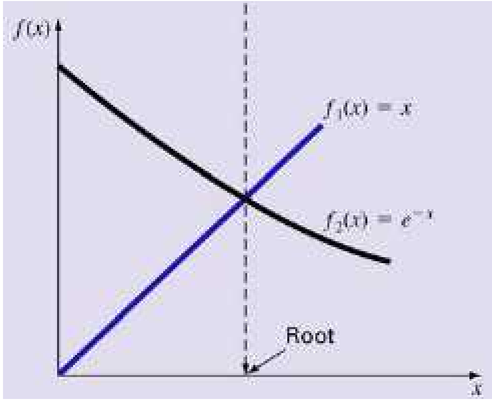


# the root of $f(x) = e^{-x} - x$.

- Two alternative graphical methods for determining the root
  - *Root at the point where it* crosses the *x axis;*
  - *Root at the intersection of* the component functions.
    - *$x=g(x)$* can be expressed as a pair of equations: *$y_1=x$ and $y_2=g(x)$*
    - Plot them separately.






| $i$ | $x_i$ | $\|\varepsilon_a\|$, % | $\|\varepsilon_t\|$, % | $\|\varepsilon_t\|_i / \|\varepsilon_t\|_{i-1}$ |
|---|---|---|---|---|
| 0 | 0.0000 | | 100.000 | |
| 1 | 1.0000 | 100.000 | 76.322 | 0.763 |
| 2 | 0.3679 | 171.828 | 35.135 | 0.460 |
| 3 | 0.6922 | 46.854 | 22.050 | 0.628 |
| 4 | 0.5005 | 38.309 | 11.755 | 0.533 |
| 5 | 0.6062 | 17.447 | 6.894 | 0.586 |
| 6 | 0.5454 | 11.157 | 3.835 | 0.556 |
| 7 | 0.5796 | 5.903 | 2.199 | 0.573 |
| 8 | 0.5601 | 3.481 | 1.239 | 0.564 |
| 9 | 0.5711 | 1.931 | 0.705 | 0.569 |
| 10 | 0.5649 | 1.109 | 0.399 | 0.566 |

Thus, each iteration brings the estimate closer to the true value of the root: 0.56714329.

# Convergence

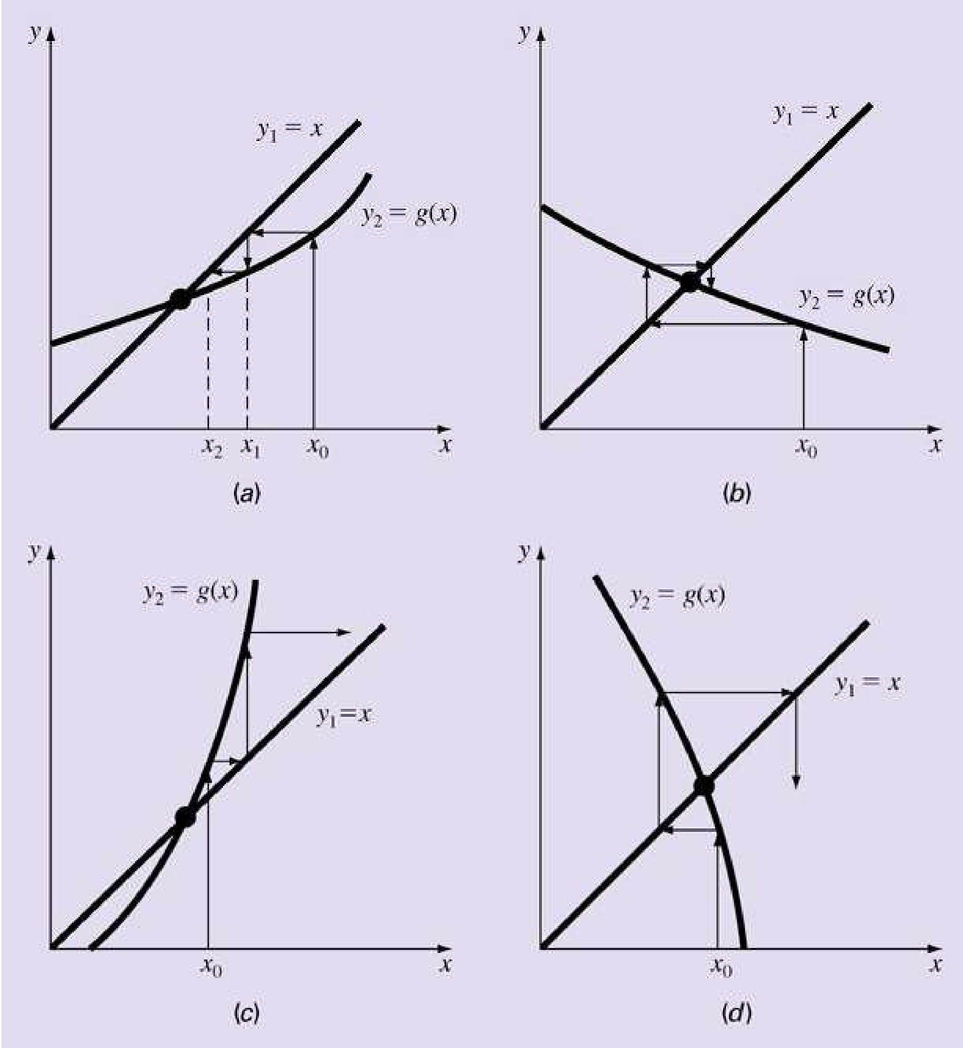
- (a) and (b) : convergence
- (c) and (d) : divergence

- (a) and (c) : monotone
- (b) and (c) : oscillating

- Convergence occurs when

$$\boxed{|g'(x)| < 1}$$

- When the method converges, the error is roughly proportional to or less than the error of the previous step, therefore it is called “linearly convergent.”



# Newton-Raphson Method

- Most widely used method.
- Based on Taylor series expansion:

$$f(x_{i+1}) = f(x_i) + f'(x_i)\Delta x + f''(x_i)\frac{\Delta x^2}{2!} + O\Delta x^3$$

The root is the value of $x_{i+1}$ when $f(x_{i+1}) = 0$

Rearranging,

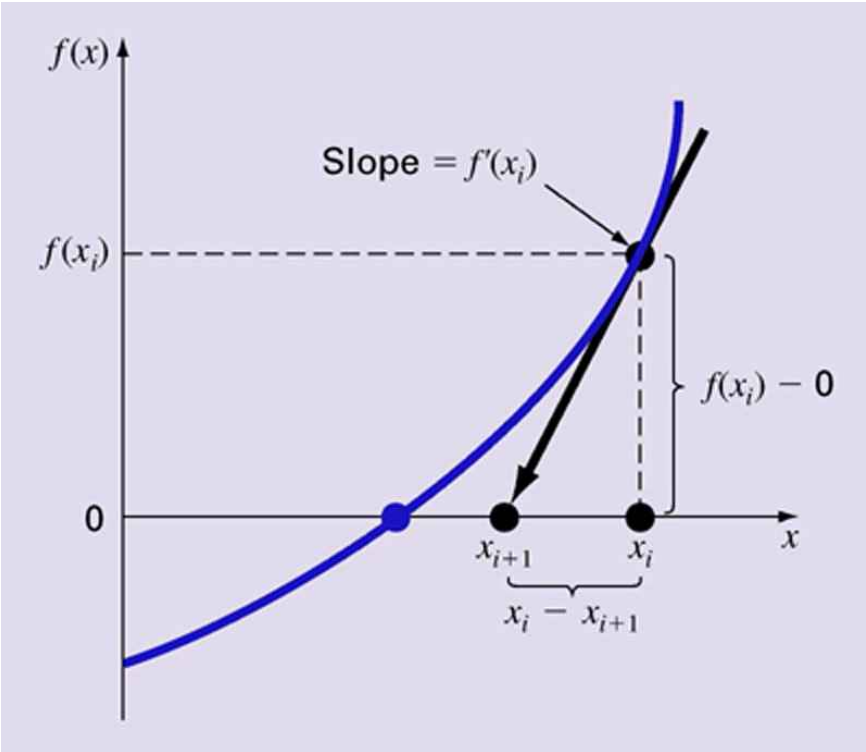
Solve for

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}$$

# Newton-Raphson Method

- Determine the next guess by the intersection of the line tangential to the current guess and the *xaxis.*




$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

```
function root = newtraph(func,dfunc,xr,es,maxit)
% newtraph(func,dfunc,xguess,es,maxit):
%   uses Newton-Raphson method to find root of a function
% input:
%   func = name of function
%   dfunc = name of derivative of function
%   xguess = initial guess
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

% if necessary, assign default values
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

% Newton-Raphson
iter = 0;
while (1)
  xrold = xr;
  xr = xr - func(xr)/dfunc(xr);
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

# Find the root of $f(x) = e^{-x} - x$ (initial guess $x_0=0$ )

$$f'(x) = -e^{-x} - 1$$

$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

| $i$ | $x_i$ | $\lvert\varepsilon_t\rvert$, % |
|---|---|---|
| 0 | 0 | 100 |
| 1 | 0.500000000 | 11.8 |
| 2 | 0.566311003 | 0.147 |
| 3 | 0.567143165 | 0.0000220 |
| 4 | 0.567143290 | $< 10^{-8}$ |

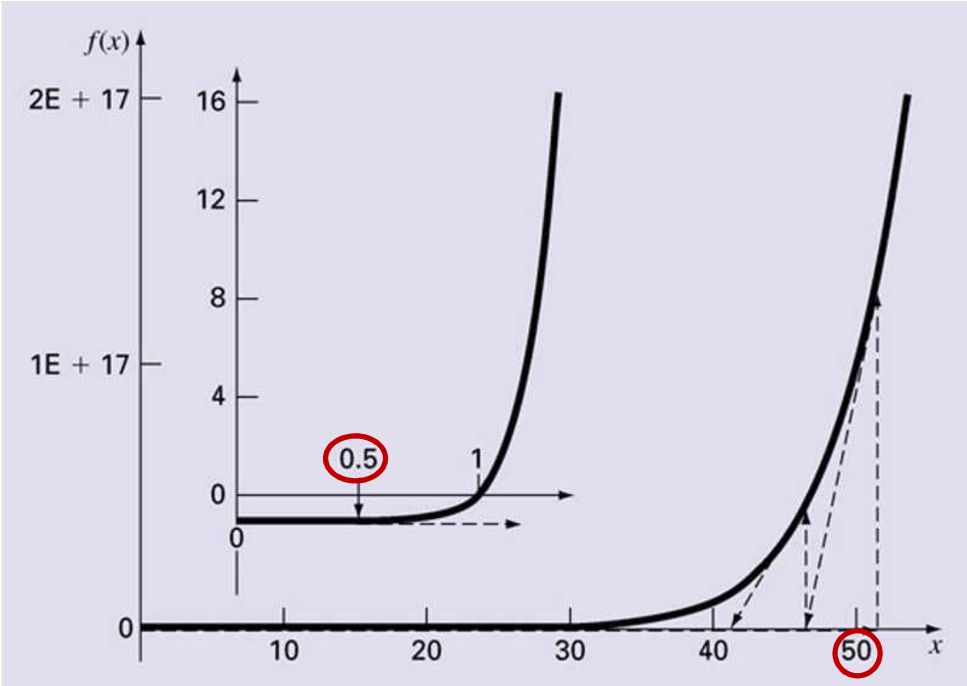
- Quadratic convergence

$$E_{t,i+1} = \frac{g''(\xi)}{2} E_{t,i}^2, \quad g(\xi) = x - \frac{f(x)}{f'(x)}$$

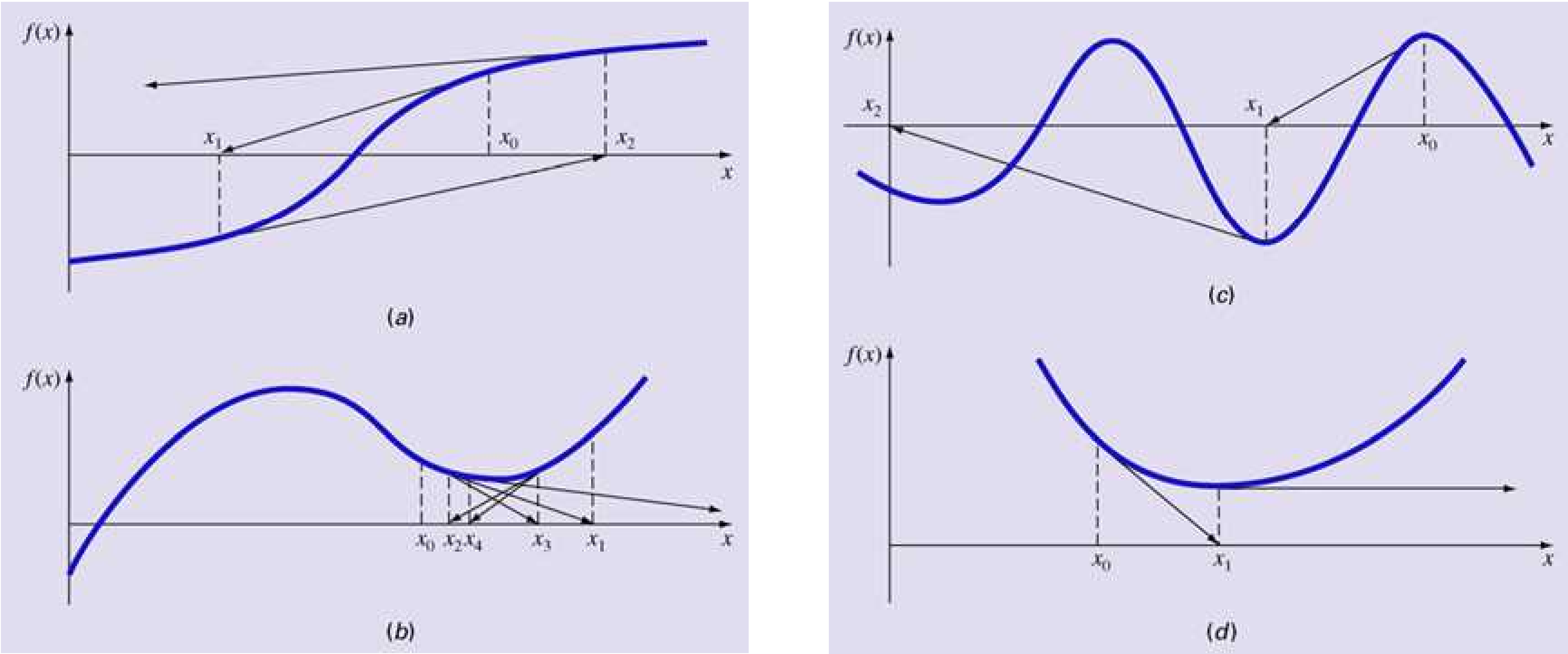
# May Slow Convergence

- Find the root of $f(x) = x^{10} - 1$ (initial guess $x_0=0.5$ )

| $i$ | $x_i$ | $\lvert\varepsilon_a\rvert$, % |
|---|---|---|
| 0 | 0.5 | |
| 1 | 51.65 | 99.032 |
| 2 | 46.485 | 11.111 |
| 3 | 41.8365 | 11.111 |
| 4 | 37.65285 | 11.111 |
| • | | |
| • | | |
| • | | |
| 40 | 1.002316 | 2.130 |
| 41 | 1.000024 | 0.229 |
| 42 | 1 | 0.002 |



# Poor Convergence Examples



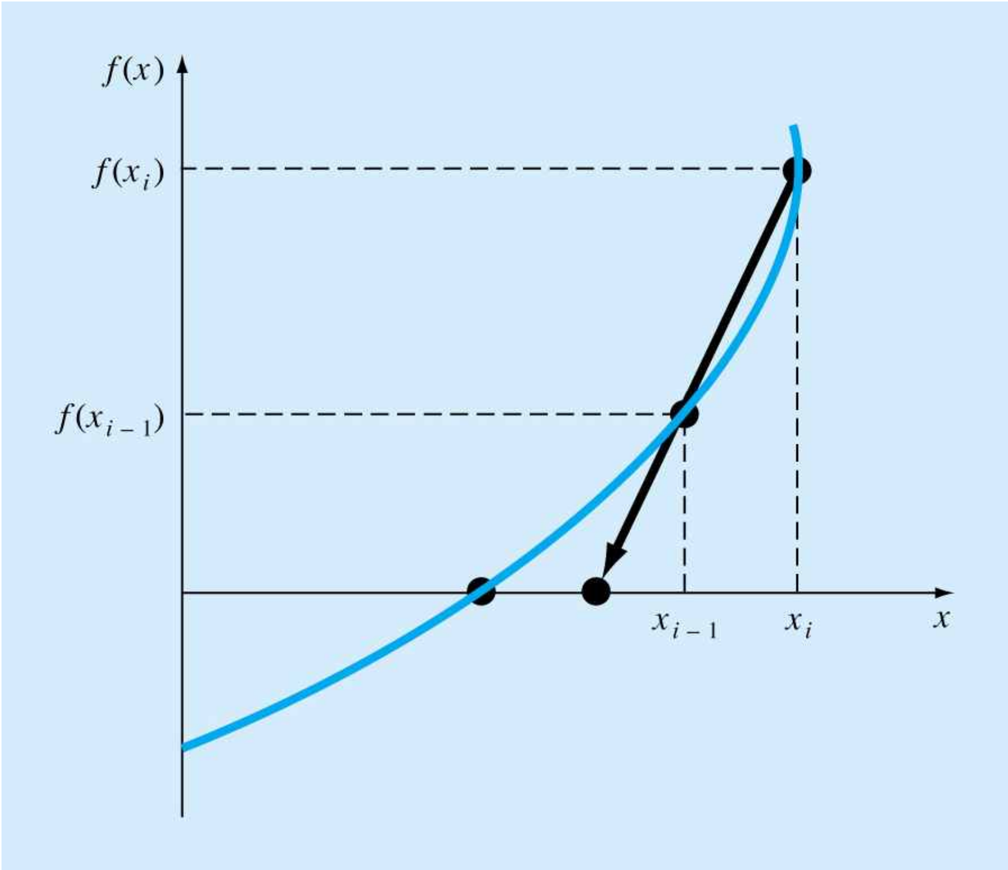
# Secant Methods

- A slight variation of Newton-Raphson method for functions whose derivatives are difficult to evaluate.
  - Derivative can be approximated by a backward finite divided difference.
  - Need two initial points

$$\frac{1}{f'(x_i)} \cong \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \qquad i = 1,2,3,\ldots$$

- Requires two initial estimates of *x*
- However, because f(x) is not required to change signs between estimates, it is not classified as a “bracketing” method.
- The secant method has the same properties as Newton’s method. Convergence is not guaranteed for all $x_o$, $f(x)$.



# Modified Secant Methods

- Same as Secant Methods except replacing the derivative with approximation
  - Need judicious choice of δ

$$f'(x_i) \approx \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

- Repeat

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}$$

# Convergence Criteria

- Simple Fixed-Point Iteration
  - initial value falls in a region that

$$|g'(x)| < 1$$

- Newton Raphson Method
  - initial value falls in a region that

$$\left| \frac{f(x) f''(x)}{[f'(x)]^2} \right| < 1$$

Fixed point

| | c | g( c) | ea(%) | et(%) |
|---|---|---|---|---|
| 1 | 20 | 15.7997 | | 6.89762 |
| 2 | 15.7997 | 15.045 | 4.77677 | 1.79137 |
| 3 | 15.045 | 14.8528 | 1.27718 | 0.49131 |
| 4 | 14.8528 | 14.8004 | 0.35292 | 0.13666 |
| 5 | 14.8004 | 14.7858 | 0.09837 | 0.03816 |
| 6 | 14.7858 | 14.7818 | 0.02748 | 0.01067 |
| 7 | 14.7818 | 14.7806 | 0.00768 | 0.00298 |
| 8 | 14.7806 | 14.7803 | 0.00215 | 0.00083 |
| 9 | 14.7803 | 14.7802 | 0.0006 | 0.00023 |
| 10 | 14.7802 | 14.7802 | 0.00017 | 6.5E-05 |
| 11 | 14.7802 | 14.7802 | 4.7E-05 | 1.8E-05 |
| 12 | 14.7802 | 14.7802 | 1.3E-05 | 5.1E-06 |
| 13 | 14.7802 | 14.7802 | 3.7E-06 | 1.4E-06 |
| 14 | 14.7802 | 14.7802 | 1E-06 | 4E-07 |
| 15 | 14.7802 | 14.7802 | 2.9E-07 | 1.1E-07 |
| 16 | 14.7802 | 14.7802 | 8E-08 | 3E-08 |
| 17 | 14.7802 | 14.7802 | 2.2E-08 | 7.5E-09 |
| 18 | 14.7802 | 14.7802 | 6.3E-09 | 1.2E-09 |
| 19 | 14.7802 | 14.7802 | 1.8E-09 | 5.8E-10 |
| 20 | 14.7802 | 14.7802 | 4.9E-10 | 1.1E-09 |
| 21 | 14.7802 | 14.7802 | 1.4E-10 | 1.2E-09 |
| 22 | 14.7802 | 14.7802 | 3.8E-11 | 1.2E-09 |
| 23 | 14.7802 | 14.7802 | 1.1E-11 | 1.3E-09 |
| 24 | 14.7802 | 14.7802 | 3E-12 | 1.3E-09 |
| 25 | 14.7802 | 14.7802 | 8.3E-13 | 1.3E-09 |
| 26 | 14.7802 | 14.7802 | 2.4E-13 | 1.3E-09 |
| 27 | 14.7802 | 14.7802 | 6E-14 | 1.3E-09 |
| 28 | 14.7802 | 14.7802 | 1.2E-14 | 1.3E-09 |
| 29 | 14.7802 | 14.7802 | 0 | 1.3E-09 |
| 30 | 14.7802 | 14.7802 | 0 | 1.3E-09 |
| 31 | 14.7802 | 14.7802 | 0 | 1.3E-09 |
| 32 | 14.7802 | 14.7802 | 0 | 1.3E-09 |
| 33 | 14.7802 | 14.7802 | 0 | 1.3E-09 |

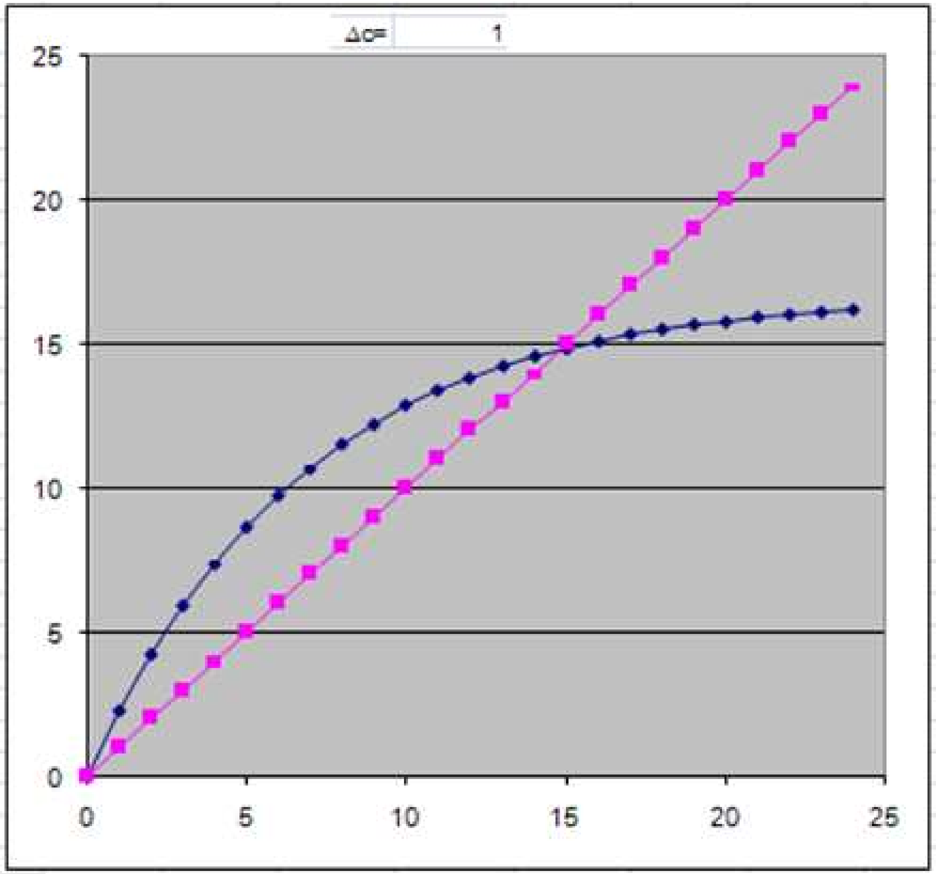
SOLUTION

14.7802

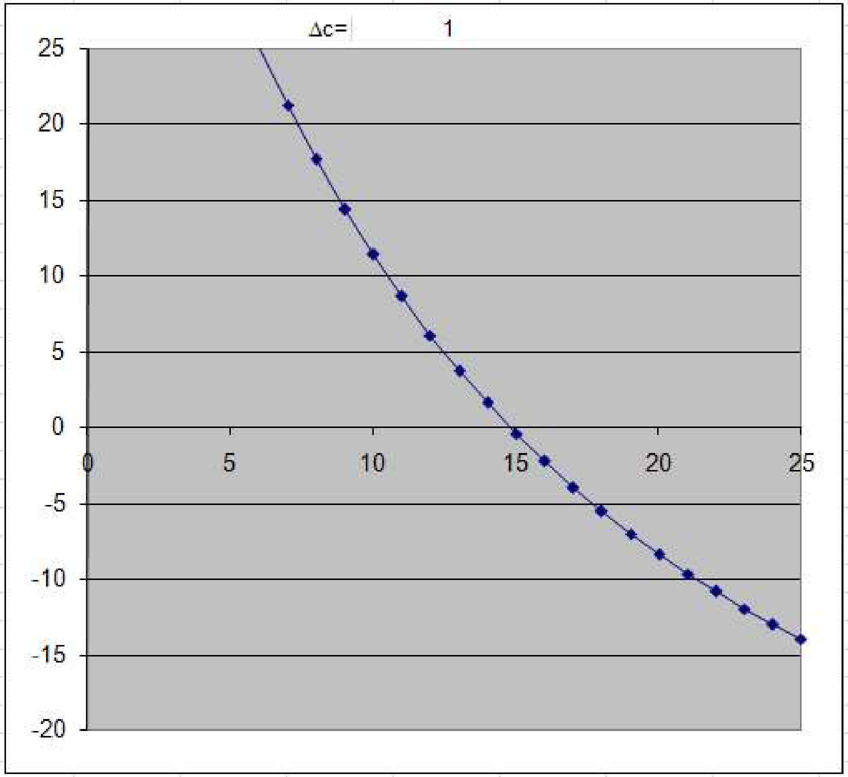
$$v = \frac{gm}{c}\left[1 - e^{-(c/m)t}\right]$$

| | |
|---|---|
| m= | 68.1 |
| g= | 9.8 |
| t= | 10 |
| v= | 40 |

What is the value of c when v=40 m/sec and t=10 sec?



| c | f( c) |
|---|---|
| 1 | 51.14433 |
| 2 | 44.92053 |
| 3 | 39.26338 |
| 4 | 34.11484 |
| 5 | 29.42323 |
| 6 | 25.14245 |
| 7 | 21.23141 |
| 8 | 17.65343 |
| 9 | 14.37575 |
| 10 | 11.3691 |
| 11 | 8.607309 |
| 12 | 6.066936 |
| 13 | 3.727002 |
| 14 | 1.568699 |
| 15 | -0.42484 |
| 16 | -2.26876 |
| 17 | -3.97667 |
| 18 | -5.56079 |
| 19 | -7.03214 |
| 20 | -8.40063 |
| 21 | -9.67519 |
| 22 | -10.8639 |
| 23 | -11.974 |
| 24 | -13.0121 |
| 25 | -13.9842 |




SOLUTION
14.7802

**Secant**

i

| | c | f( c) | ea(%) | et(%) |
|---|---|---|---|---|
| 1 | 5 | 29.42323 | | 66.17097 |
| 2 | 8 | 17.65343 | 60 | 45.87355 |
| 3 | 12.49968 | 4.873897 | 56.24594 | 15.42962 |
| 4 | 14.21578 | 1.125146 | 13.72916 | 3.818813 |
| 5 | 14.73084 | 0.096416 | 3.623218 | 0.333959 |
| 6 | 14.77912 | 0.002118 | 0.327705 | 0.007348 |
| 7 | 14.7802 | 4.09E-06 | 0.007335 | 1.42E-05 |
| 8 | 14.7802 | 1.74E-10 | 1.42E-05 | 1.87E-09 |

**Newton Raphson**

| | c | f( c) | f ' ( c) | ea(%) | et(%) |
|---|---|---|---|---|---|
| 1 | 10 | 11.3691 | -2.8801 | | 32.34193 |
| 2 | 13.94747 | 1.677831 | -2.08193 | 39.47472 | 5.634103 |
| 3 | 14.75337 | 0.052361 | -1.95365 | 5.778123 | 0.181526 |
| 4 | 14.78018 | 5.49E-05 | -1.94955 | 0.181665 | 0.000191 |
| 5 | 14.7802 | 6.06E-11 | -1.94955 | 0.000191 | 1.47E-09 |
| 6 | 14.7802 | 0 | -1.94955 | 2.1E-10 | 1.26E-09 |
| 7 | 14.7802 | 0 | -1.94955 | 0 | 1.26E-09 |
| 8 | 14.7802 | 0 | -1.94955 | 0 | 1.26E-09 |

# MATLAB's fzero Function

- Provides the best qualities of both bracketing methods and open methods.
  - Using an initial guess:

    *x* = fzero(*function*, *x0*)
    [*x*, *fx*] = fzero(*function*, *x0*)

    - *function* is the name of the function being evaluated
    - *x0* is the initial guess
    - *x* is the location of the root
    - *fx* is the function evaluated at that root

  - Using an initial bracket:

    *x* = fzero(*function*, [*x0* *x1*])
    [*x*, *fx*] = fzero(*function*, [*x0* *x1*])

    - As above, except *x0* and *x1* are guesses that *must* bracket a sign change

- x = fzero(inline('x^2-9'), -4) % inline : Matlab 함수정의
- x = fzero(inline('x^2-9'), [0 4])

# fzero Options

- Options may be passed to fzero as a third input argument
  - options = optimset('par1', val1, 'par2', val2,…)
    - parn is the name of the parameter to be set
    - valn is the value to which to set that parameter
  - The parameters commonly used with fzero are:
    - display : when set to 'iter' displays a detailed record of all the iterations
    - tolx : a positive scalar that sets a termination tolerance on x.
- options = optimset('display', 'iter');
- [x, fx] = fzero(@(x) x^10-1, 0.5, options)
    - Uses fzero to find roots of $f(x)=x^{10}-1$ starting with an initial guess of $x=0.5$.
    - MATLAB reports x=1, fx=0 after 35 function counts

Anonymous functions : a quick means of creating functions without having to store to a file each time.
construct either at the MATLAB command line or in any function or script.
The syntax for creating an anonymous function is

```
fhandle = @(arglist) expr
```

# Polynomials

- MATLAB has a built in program called **roots** to determine all the roots of a polynomial - including imaginary and complex ones.

- x = roots(c)
  - x is a column vector containing the roots
  - c is a row vector containing the polynomial coefficients

- Example:
  - Find the roots of

    $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$

  - x = roots([1 -3.5 2.75 2.125 -3.875 1.25])

# Polynomials (cont)

- MATLAB's poly function can be used to determine polynomial coefficients if roots are given:
  - b = poly([0.5 -1])
    - Finds f(x) where f(x)=0 for x=0.5 and x=-1 (roots)
    - MATLAB reports b = [1.000 0.5000 -0.5000]
    - This corresponds to $f(x)=x^2+0.5x-0.5$

- MATLAB's polyval function can evaluate a polynomial at one or more points:
  - a = [1 -3.5 2.75 2.125 -3.875 1.25];
    - If used as coefficients of a polynomial, this corresponds to $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$
  - polyval(a, 1)
    - This calculates f(1), which MATLAB reports as -0.2500

# Chapter ends...

MATLAB Lab

Chapter Problems Report