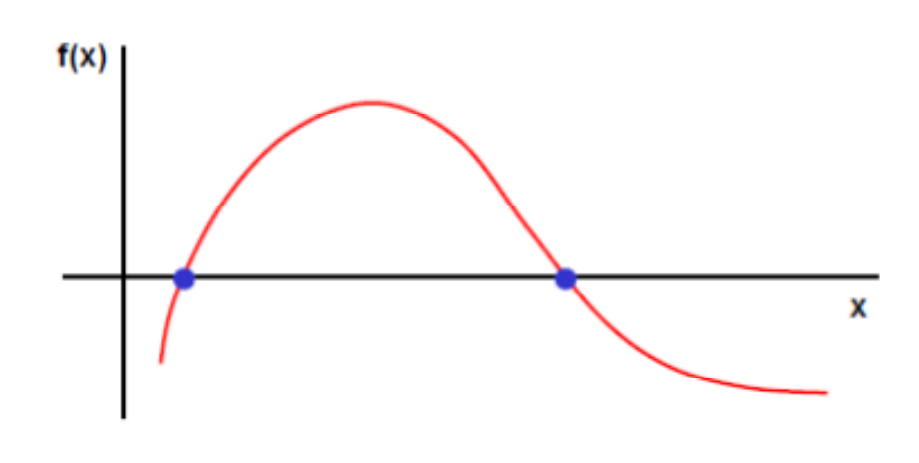
Roots of Equations

Chapter 5



$f(x)$ is given

$f(x_r)=0 \rightarrow x_{r=}?$

Roots of Equations

Chapter 5

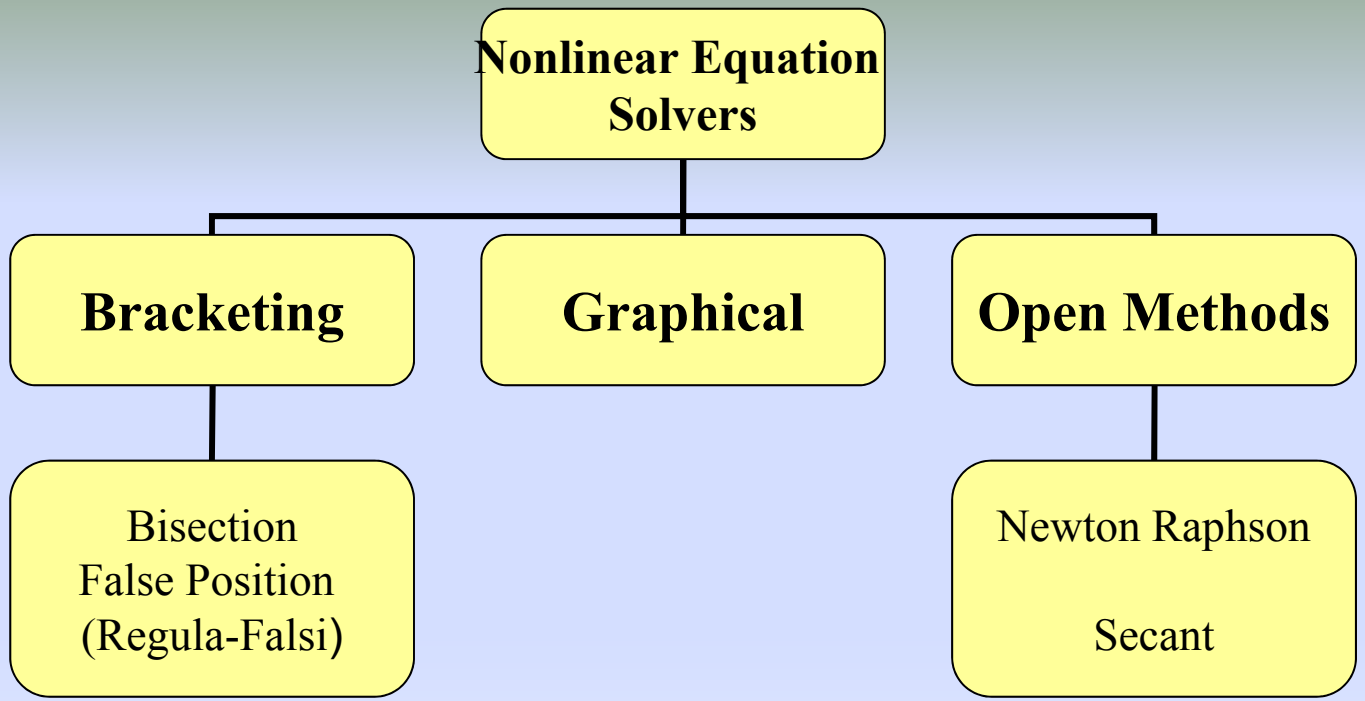
- Roots of a quadric equation:

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

- But how to find the roots of:

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

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



- **Bracketing Methods** (Need two initial estimates that will bracket the root. Always converge.)
  - Bisection Method
  - False-Position Method
- **Open Methods** (Need one or two initial estimates. May diverge.)
  - Simple One-Point Iteration
  - Newton-Raphson Method (Needs the derivative of the function.)
  - Secant method

Graphical Methods

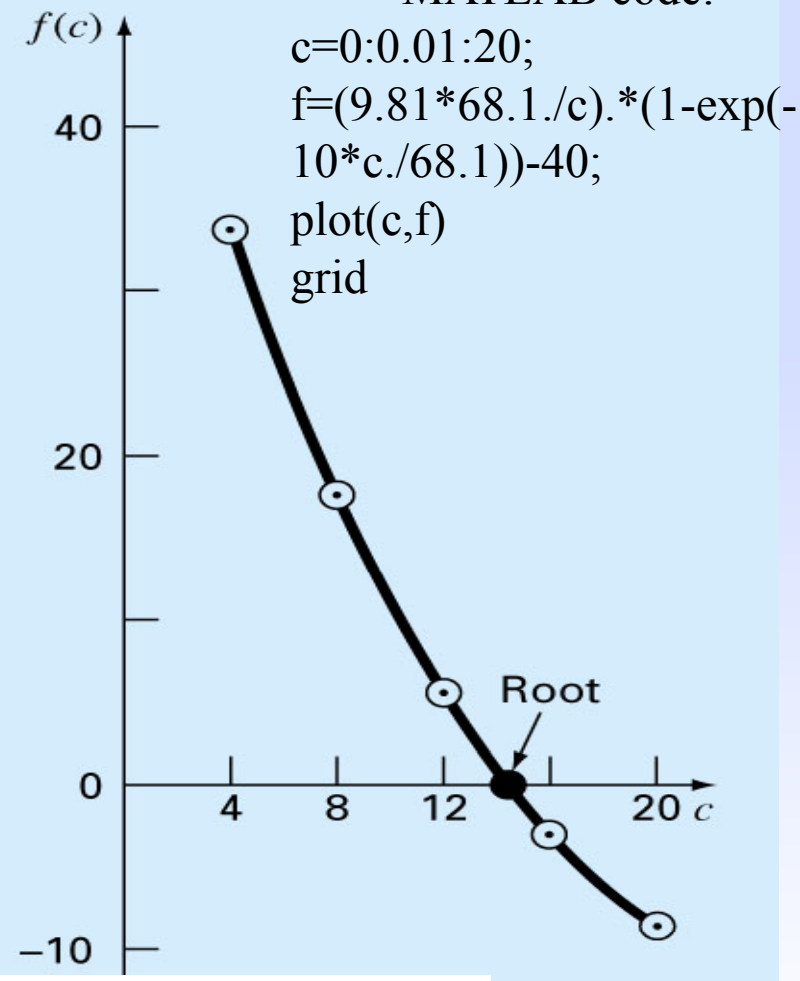
- Two initial guesses for the root are required. These guesses must "bracket" or be on either side of the root.

- If one root of a real and continuous function, $f(x)=0$, is bounded by values $x=x_l$, $x=x_u$ then

  $f(x_l)\times f(x_u) < 0$. (The function changes sign on opposite sides of the root at least one time)

MATLAB code:

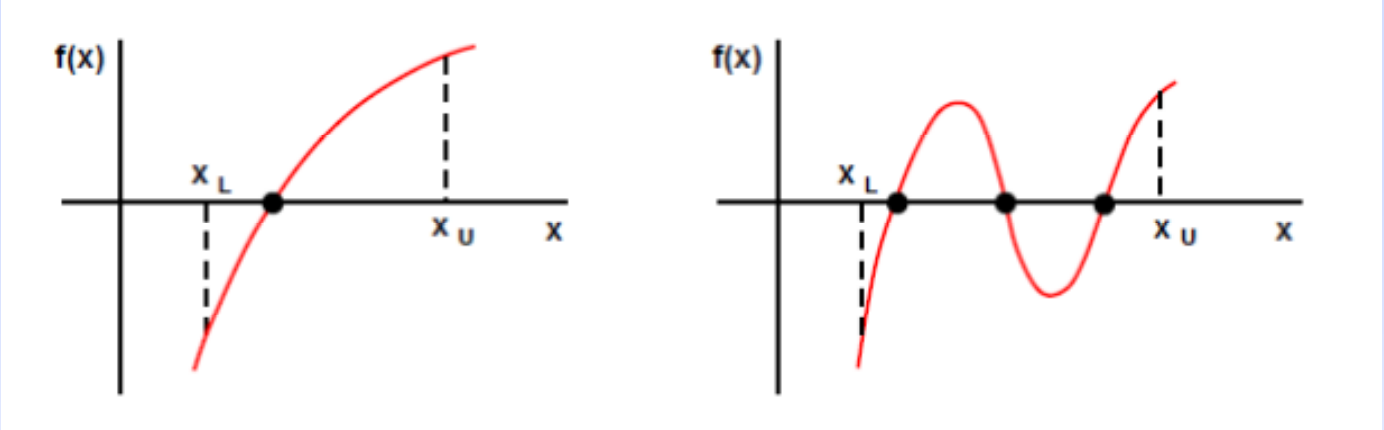
```
c=0:0.01:20;
f=(9.81*68.1./c).*(1-exp(-10*c./68.1))-40;
plot(c,f)
grid
```



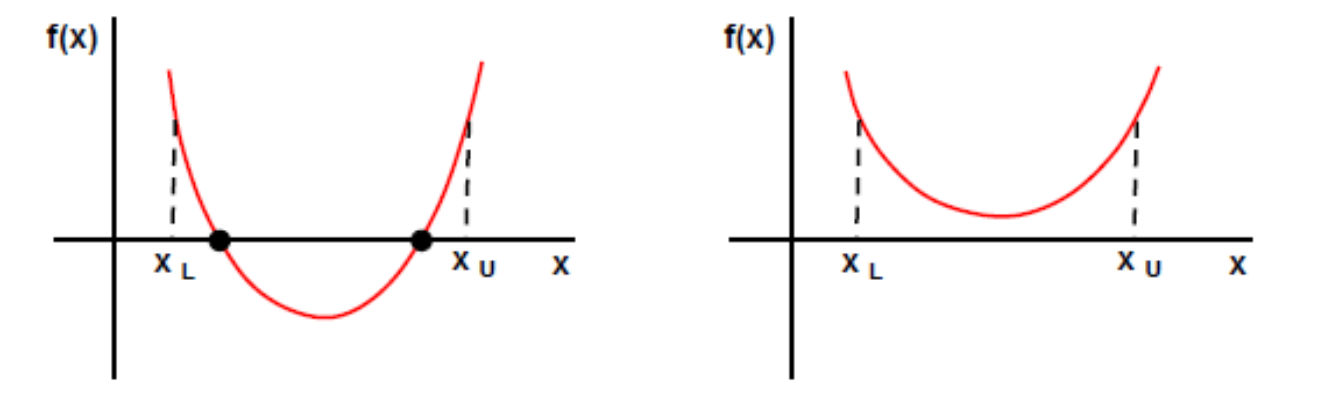
$$f(c) = \frac{9.81(68.1)}{c}(1 - e^{-(c/68.1)10}) - 40$$

FIGURE 5.1
The graphical approach for determining the roots of an equation.

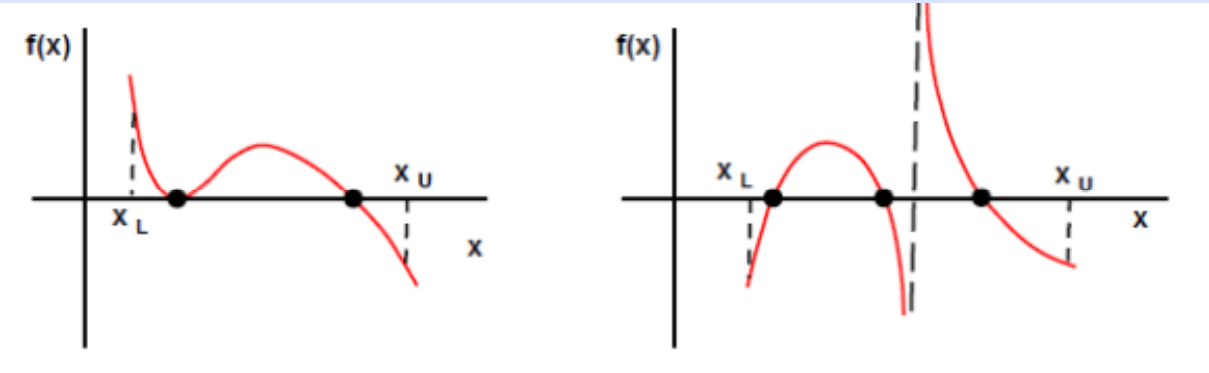
General Idea of Bracketing Methods



Case 1: If $f(x_L)*f(x_u)<0$, then there are odd number of roots



Case 2: If $f(x_L)*f(x_u)>0$, then there are:
i) even number of roots, *ii)* no roots



Violations: *i*) multiple roots *ii*) discontinuities

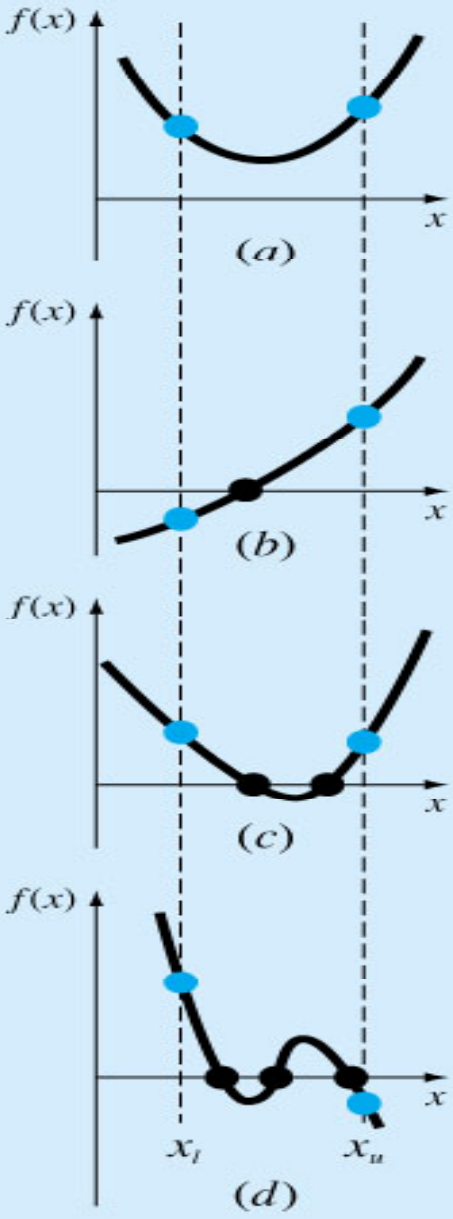


FIGURE 5.2
Illustration of a number of general ways that a root may occur in an interval prescribed by a lower bound x_l and an upper bound x_u. Parts (*a*) and (*c*) indicate that if both $f(x_l)$ and $f(x_u)$ have the same sign, either there will be no roots or there will be an even number of roots within the interval. Parts (*b*) and (*d*) indicate that if the function has different signs at the end points, there will be an odd number of roots in the interval.

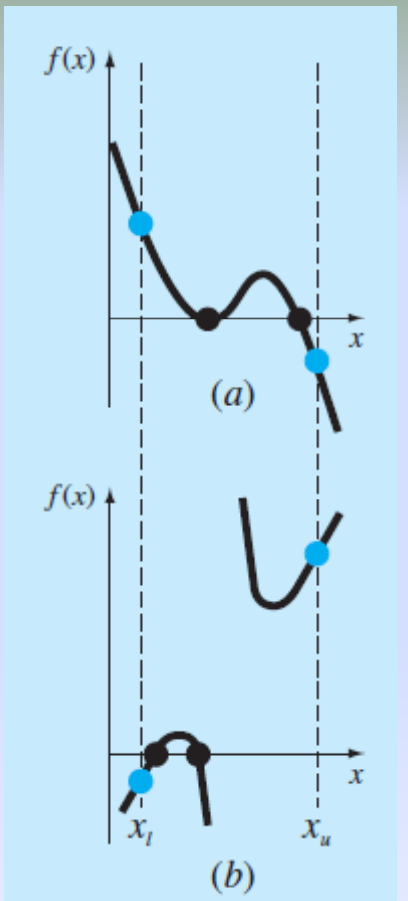
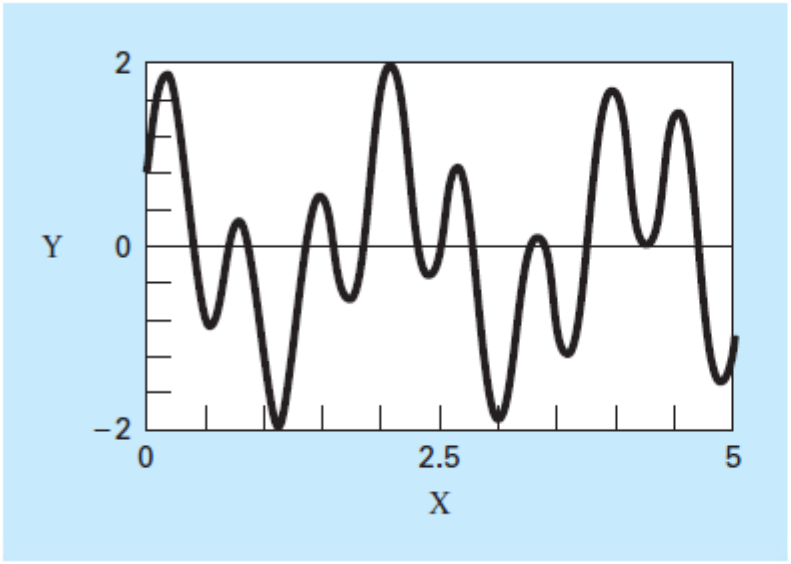


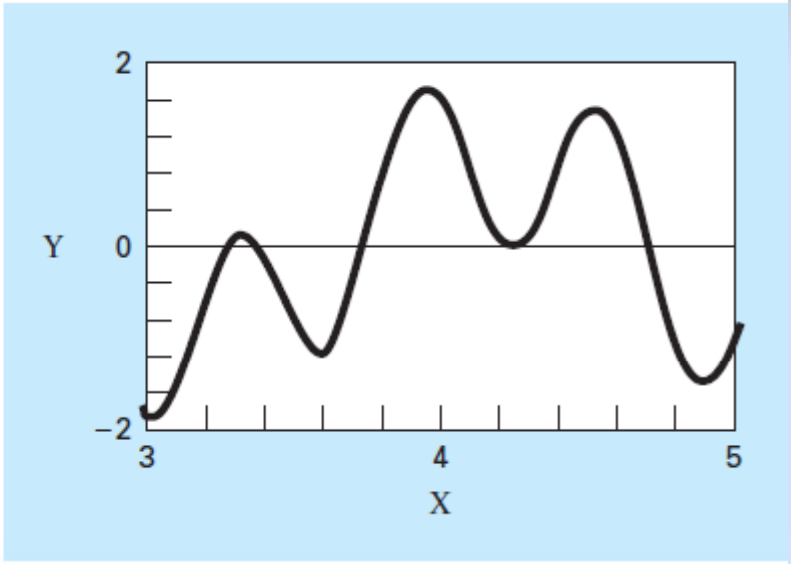
FIGURE 5.3
Illustration of some exceptions to the general cases depicted in Fig. 5.2. (*a*) Multiple root that occurs when the function is tangential to the *x* axis. For this case, although the end points are of opposite signs, there are an even number of axis intersections for the interval. (*b*) Discontinuous function where end points of opposite sign bracket an even number of roots. Special strategies are required for determining the roots for these cases.

Several roots example

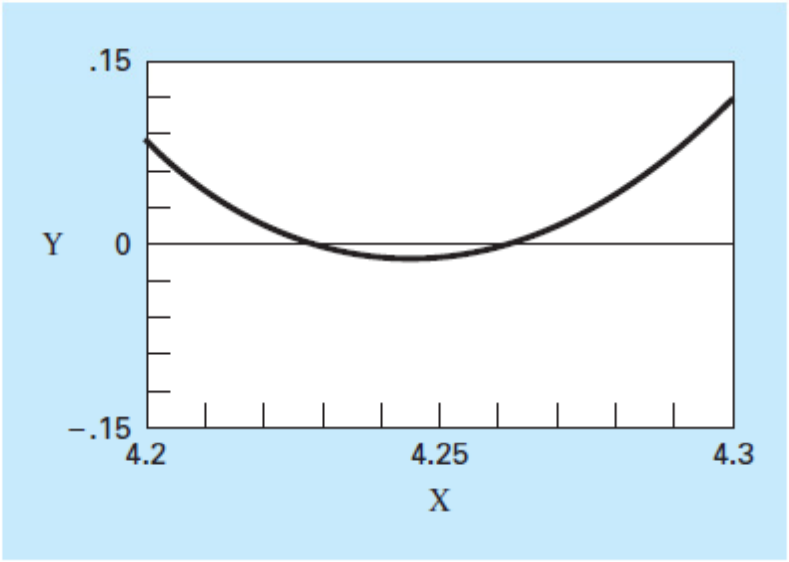
FIGURE 5.4
The progressive enlargement of $f(x) = \sin 10x + \cos 3x$ by the computer. Such interactive graphics permits the analyst to determine that two distinct roots exist between $x = 4.2$ and $x = 4.3$.



(*a*)



(*b*)



The Bisection Method

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 $f(x_l).f(x_u) <0$.

2. Estimate the root by evaluating $f[(x_l+x_u)/2]$.

3. Find the pair
   - 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.
   - 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.

- If $f(x_l).f[(x_l+x_u)/2]=0$, then root is $(x_l+x_u)/2$ and terminate.

4. Compare ε_s with ε_a

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

$$x_r^{\text{new}} - x_r^{\text{old}} = \frac{x_u - x_l}{2} \qquad x_r^{\text{new}} = \frac{x_l + x_u}{2}$$

see Fig 5.8 and 5.9 for details

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

or

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

5. If $\varepsilon_a < \varepsilon_s$, stop.

   Otherwise repeat the process.

where ε_s is the prespecified stopping criterion

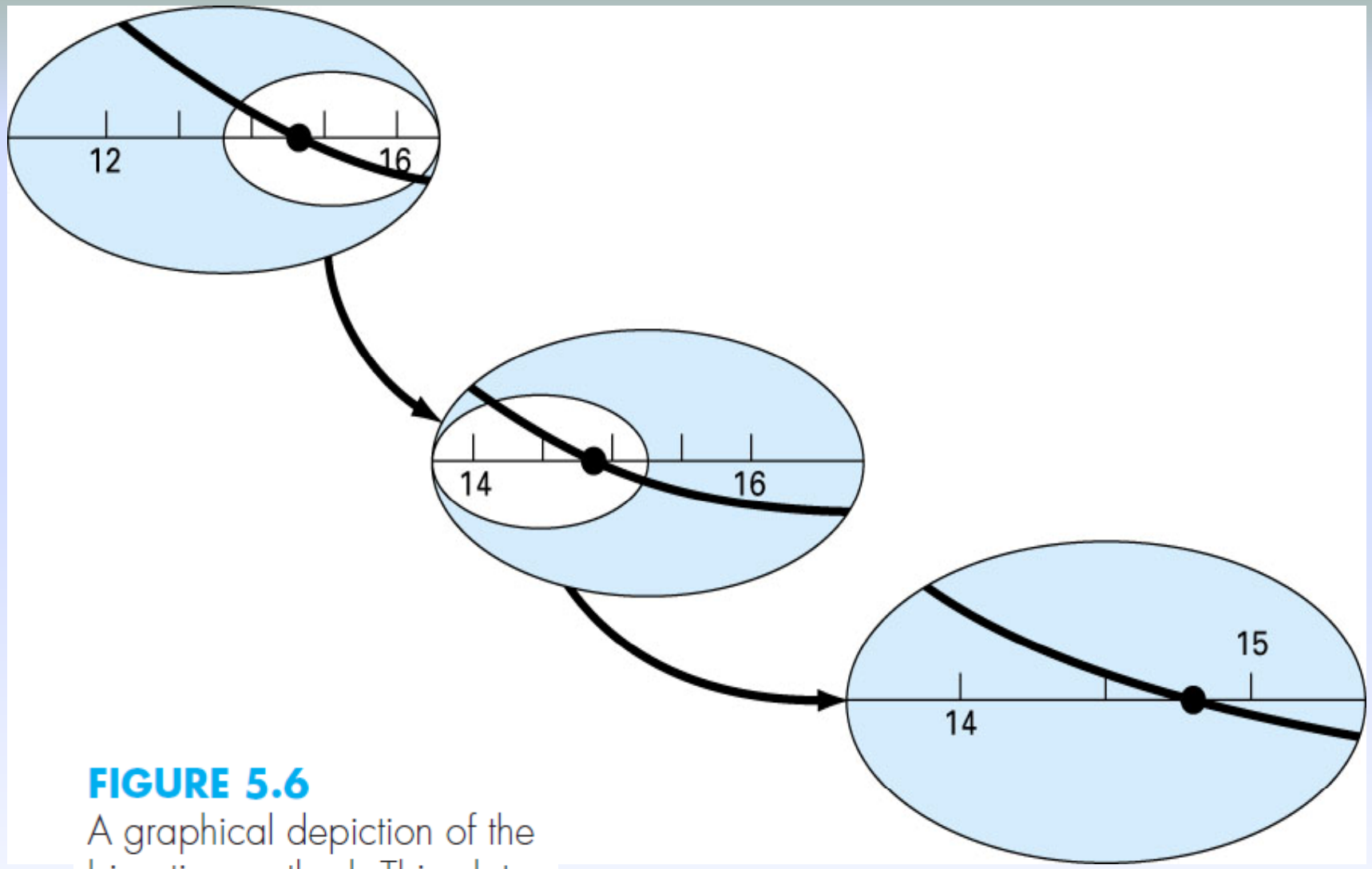


FIGURE 5.6
A graphical depiction of the bisection method. This plot conforms to the first three iterations from Example 5.3.

Evaluation of the Bisection Method

Advantages

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

Disadvantages

- Slow
- Know $x_l=a$ and $x_u=b$ that bound root
- 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 $L_k=L_o/2^k$

$$k = \log_2\left(\frac{L_o}{\varepsilon_s}\right) = \log_2\left(\frac{b-a}{\varepsilon_s}\right) \quad for\ \varepsilon_a \le \varepsilon_s$$

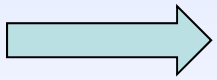
Ex: How many iterations needed in Bisection method for the absolute magnitude of the error to be less than 10^{-4} for a L_o=2.

$$k = \log_2\left(\frac{L_o}{\varepsilon_s}\right) = \log_2\left(\frac{b-a}{\varepsilon_s}\right) \quad for\ \varepsilon_a \le \varepsilon_s$$

```
Command Window
>> log2(2/0.0001)

ans =

   14.2877
```



k=15

Homework:

Pseudocode for the Bisection algorithm is given in Figure 5.11 (pp.134) in our textbook.

Write the MATLAB code for this pseudocode.

The False-Position Method

- If a real root is bounded by x_l and x_u of $f(x)=0$, then we can approximate the solution by doing a linear interpolation between the points $[x_l, f(x_l)]$ and $[x_u, f(x_u)]$ to find the x_r value such that $l(x_r)=0$, $l(x)$ is the linear approximation of $f(x)$.

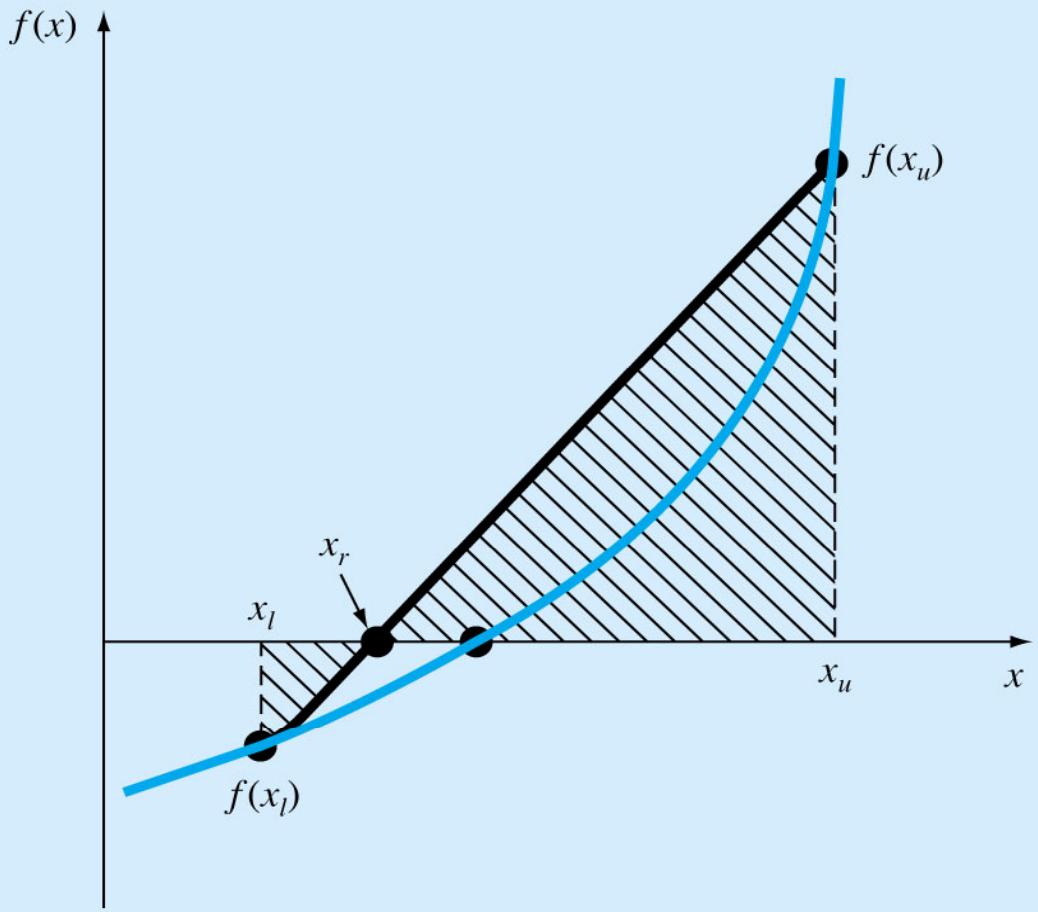


FIGURE 5.12
A graphical depiction of the method of false position. Similar triangles used to derive the formula for the method are shaded.

False Position Method 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 (Refer to Box 5.1 pp136)

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

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.

➔See Sec.5.3.1, Pitfalls of the False-Position Method

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

Finalizing bracketing methods

A plot of the function is always helpful.

to determine the number of all roots, if there are any.

to determine whether the roots are multiple or not.

to determine whether to method converges to the desired root.

to determine the initial guesses.

Incremental search technique can be used to determine the initial guesses.

Start from one end of the region of interest.

Evaluate the function at specified intervals.

If the sign of the function changes, than there is a root in that interval.

Select your intervals small, otherwise you may miss some of the roots.But if they are too small, incremental search might become too costly.

Incremental search, just by itself, can be used as a root finding technique with very small intervals (not efficient).

Open Methods

Chapter 6

- Open methods are based on formulas that require only a single starting value of x or two starting values that do not necessarily bracket the root. As such, they sometimes *diverge* or move away from the true root but, when the open methods *converge*, they usually do so much more quickly than the bracketing methods.

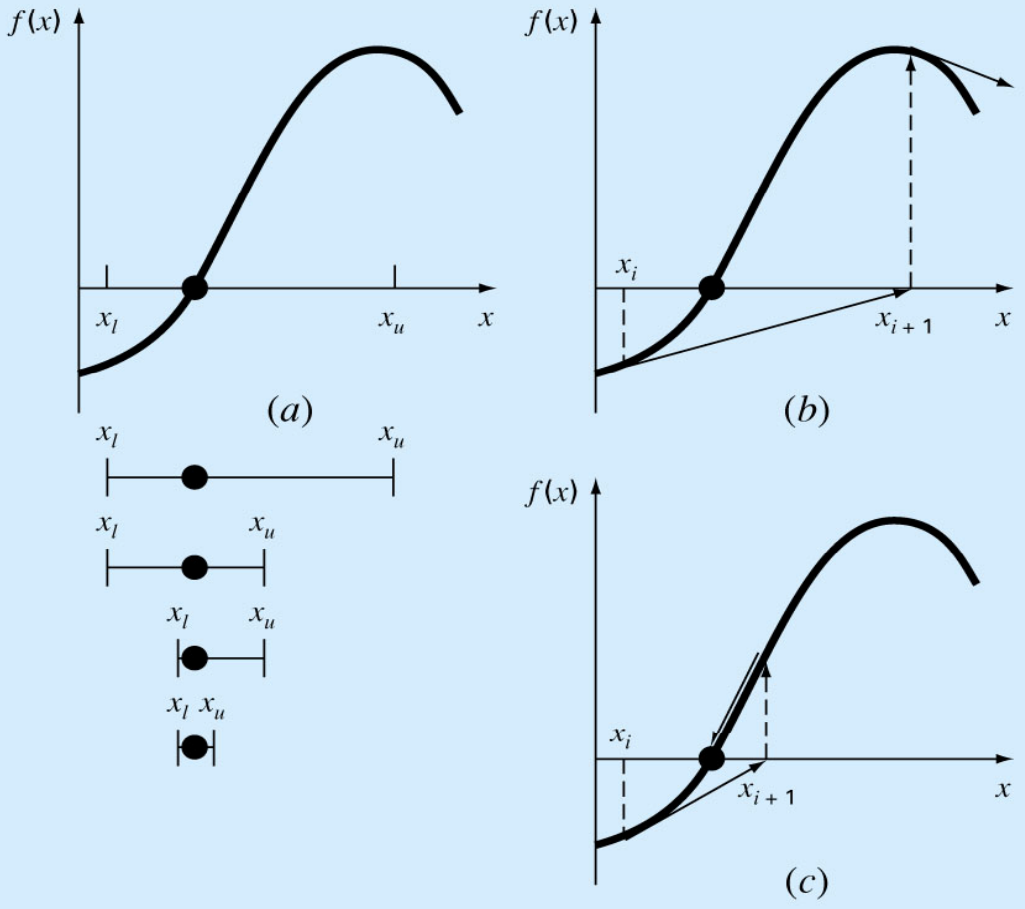


Figure 6.1. Graphical depiction of the fundamental difference between the (a) bracketing and (b) and (c) open methods for root location.

Simple Fixed-point Iteration

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

$$f(x) = 0 \quad \Rightarrow \quad g(x) = x$$

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

•This transformation can be accomplished either by algebraic manipulation or by simply adding x to both sides of the original equation.

Example:

1. $f(x) = x^2 - x - 2 \quad \Rightarrow \quad x = x^2 - 2 \quad \Rightarrow \quad g(x) = x^2 - 2$

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

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

2. $f(x) = \sin x \quad \Rightarrow \quad x = x + \sin x \quad \Rightarrow \quad g(x) = x + \sin x$

Simple Fixed-point Iteration

- Start with an initial guess x_0
- Calculate a new estimate for the root using $x_1 = g(x_0)$
- Iterate like this. General formula is $x_{i+1} = g(x_i)$

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\%$$

- Converges if $|g'(x)|<1$ in the region of interest.

Simple Fixed-Point Iteration Example:

Problem Statement. Use simple fixed-point iteration to locate the root of $f(x) = e^{-x} - x$.

Solution. The function can be separated directly and expressed in the form of Eq. (6.2) as

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

Starting with an initial guess of $x_0 = 0$, this iterative equation can be applied to compute

i	x_i	ε_a (%)	ε_t (%)
0	0		100.0
1	1.000000	100.0	76.3
2	0.367879	171.8	35.1
3	0.692201	46.9	22.1
4	0.500473	38.3	11.8
5	0.606244	17.4	6.89
6	0.545396	11.2	3.83
7	0.579612	5.90	2.20
8	0.560115	3.48	1.24
9	0.571143	1.93	0.705
10	0.564879	1.11	0.399

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

Homework:

Use MATLAB to implement the fixed point iteration method whose pseudocode is given here.

```
FUNCTION Fixpt(x0, es, imax, iter, ea)
  xr = x0
  iter = 0
  DO
    xrold = xr
    xr = g(xrold)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = |(xr − xrold)/xr| · 100
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Fixpt = xr
END Fixpt
```

FIGURE 6.4
Pseudocode for fixed-point iteration. Note that other open methods can be cast in this general format.

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

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

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

Newton-Raphson formula

- A convenient method for functions whose derivatives can be evaluated analytically. It may not be convenient for functions whose derivatives cannot be evaluated analytically.

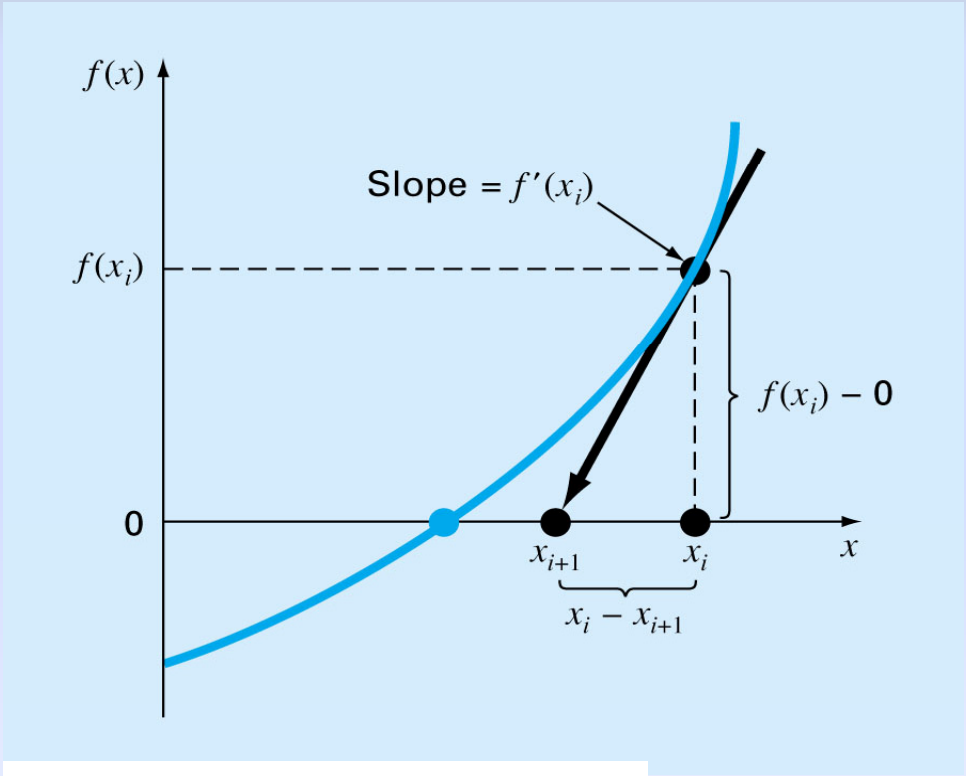


FIGURE 6.5
Graphical depiction of the Newton-Raphson method. A tangent to the function of x_i [that is, $f'(x_i)$] is extrapolated down to the x axis to provide an estimate of the root at x_{i+1}.

Problem Statement. Use the Newton-Raphson method to estimate the root of $f(x) = e^{-x} - x$, employing an initial guess of $x_0 = 0$.

Solution. The first derivative of the function can be evaluated as

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

which can be substituted along with the original function into Eq. (6.6) to give

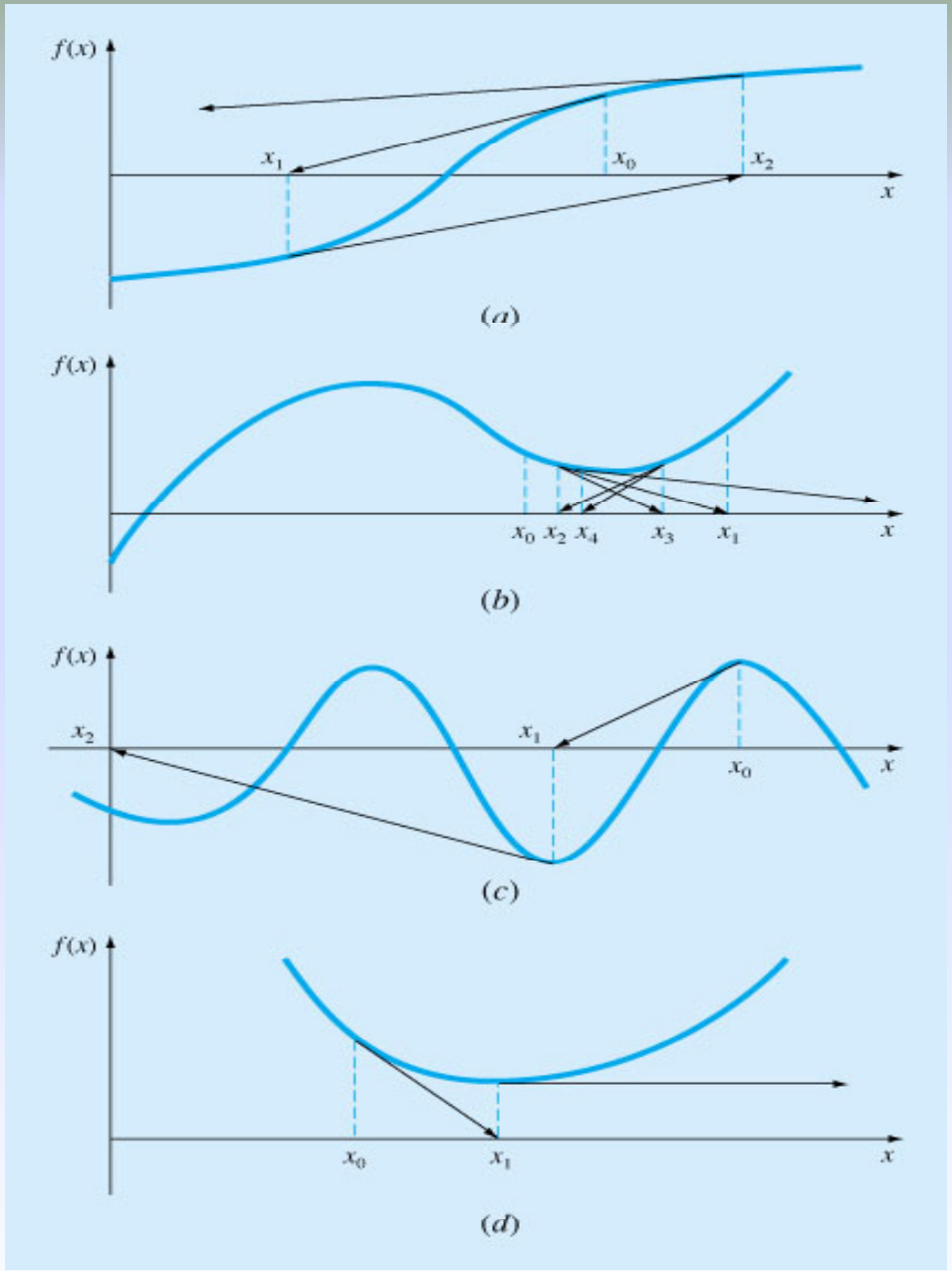
$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

Starting with an initial guess of $x_0 = 0$, this iterative equation can be applied to compute

i	x_i	ε_t (%)
0	0	100
1	0.500000000	11.8
2	0.566311003	0.147
3	0.567143165	0.0000220
4	0.567143290	$< 10^{-8}$

Thus, the approach rapidly converges on the true root. Notice that the true percent relative error at each iteration decreases much faster than it does in simple fixed-point iteration (compare with Example 6.1).

•Although the Newton-Raphson method is often very efficient, there are situations where it performs poorly. *Examples:* (a) an inflection point [that is, $f''(x)=0$] occurs in the vicinity of a root, (b) may tend to oscillate around a local maximum or minimum, (c) initial guess that is close to one root can jump to a location several roots away, (d) a zero slope [$f'(x)=0$] is truly a disaster because it causes division by zero in the Newton-Raphson formula.



Homework: Try the following MATLAB code (an implementation of the Newton-Raphson method) to find the roots of some functions.

```
function [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit,varargin)
% newtraph: Newton-Raphson root location zeroes
%  [root,ea,iter]=newtraph(func,dfunc,xr,es,maxit,p1,p2,...):
%        uses Newton-Raphson method to find the root of func
% input:
%   func = name of function
%   dfunc = name of derivative of function
%   xr = initial guess
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   root = real root
%   ea = approximate relative error (%)
%   iter = number of iterations

if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es),es=0.0001;end
if nargin<5|isempty(maxit),maxit=50;end
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;
```

The Secant Method

- A slight variation of Newton's method for functions whose derivatives are difficult to evaluate. For these cases the derivative can be approximated by a backward finite divided difference.

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

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

- Requires two initial estimates of x , e.g, x_o, x_1. 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 $x_o, f(x)$.

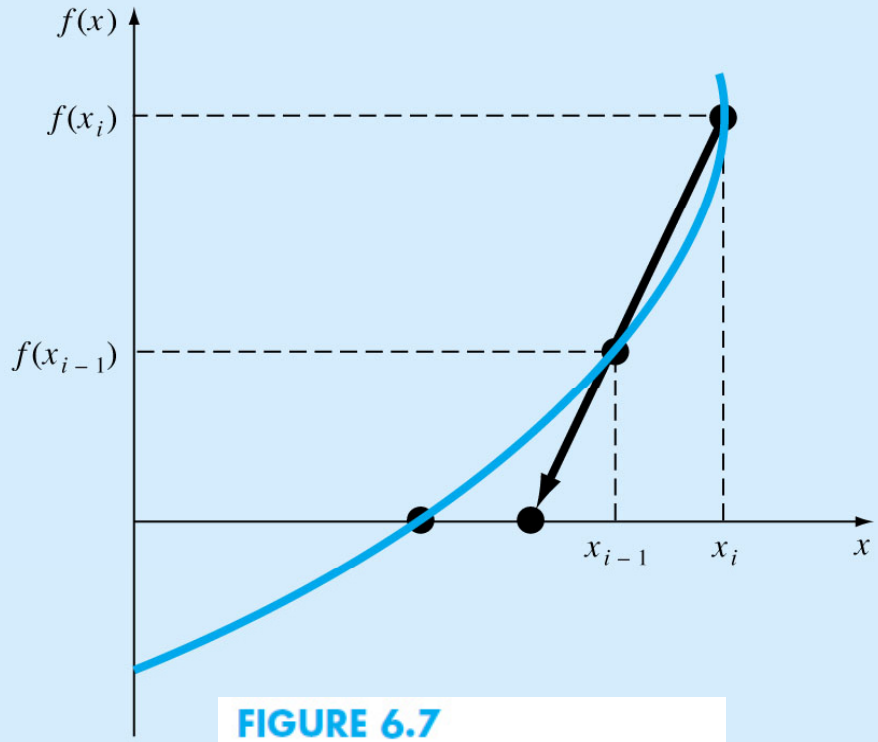


FIGURE 6.7
Graphical depiction of the secant method. This technique is similar to the Newton-Raphson technique (Fig. 6.5) in the sense that an estimate of the root is predicted by extrapolating a tangent of the function to the x axis. However, the secant method uses a difference rather than a derivative to estimate the slope.

Example:

The Secant Method

Problem Statement. Use the secant method to estimate the root of $f(x) = e^{-x} - x$. Start with initial estimates of $x_{-1} = 0$ and $x_0 = 1.0$.

Solution. Recall that the true root is 0.56714329. . . .

First iteration:

$$x_{-1} = 0 \qquad f(x_{-1}) = 1.00000$$

$$x_0 = 1 \qquad f(x_0) = -0.63212$$

$$x_1 = 1 - \frac{-0.63212(0 - 1)}{1 - (-0.63212)} = 0.61270 \qquad \varepsilon_t = 8.0\%$$

Second iteration:

$$x_0 = 1 \qquad f(x_0) = -0.63212$$

$$x_1 = 0.61270 \qquad f(x_1) = -0.07081$$

(Note that both estimates are now on the same side of the root.)

$$x_2 = 0.61270 - \frac{-0.07081(1 - 0.61270)}{-0.63212 - (-0.07081)} = 0.56384 \qquad \varepsilon_t = 0.58\%$$

Third iteration:

$$x_1 = 0.61270 \qquad f(x_1) = -0.07081$$

$$x_2 = 0.56384 \qquad f(x_2) = 0.00518$$

$$x_3 = 0.56384 - \frac{0.00518(0.61270 - 0.56384)}{-0.07081 - (-0.00518)} = 0.56717 \qquad \varepsilon_t = 0.0048\%$$

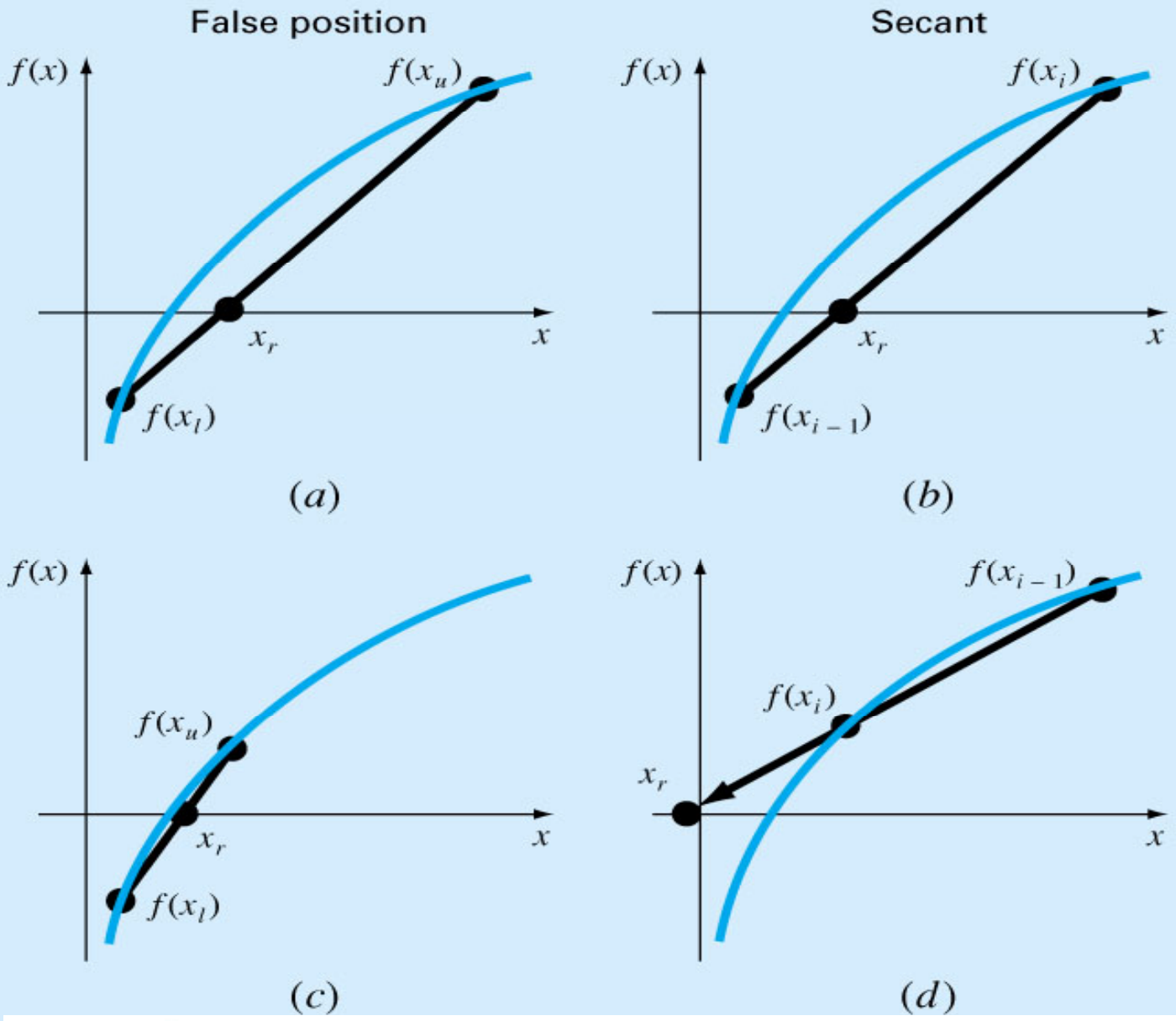


FIGURE 6.8
Comparison of the false-position and the secant methods. The first iterations (*a*) and (*b*) for both techniques are identical. However, for the second iterations (*c*) and (*d*), the points used differ. As a consequence, the secant method can diverge, as indicated in (*d*).

Although the secant method may be divergent, when it converges it usually does so at a quicker rate than the false-position method. This figure demonstrates the superiority of the secant method in this regard.

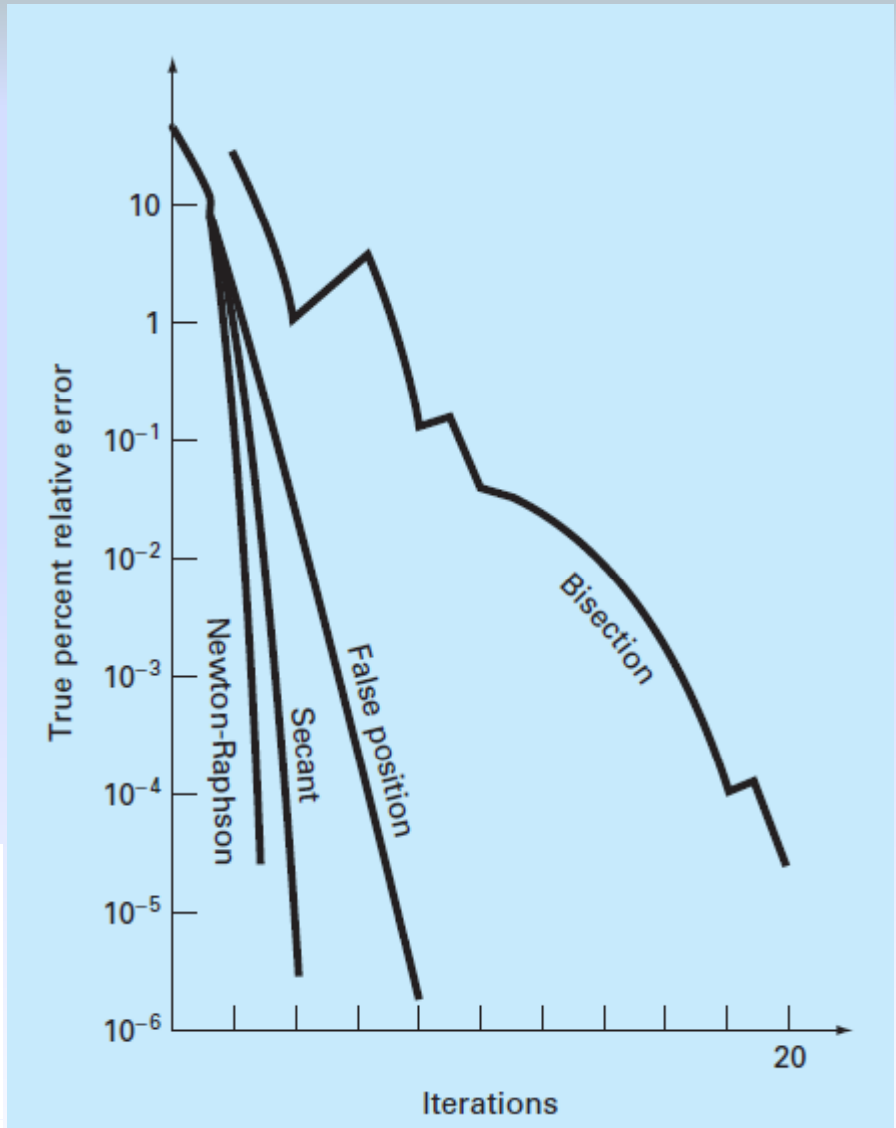


FIGURE 6.9
Comparison of the true percent relative errors ε_t for the methods to determine the roots of $f(x) = e^{-x} - x$.

Brent's Method

- The general idea behind the Brent's root finding method is whenever possible to use one of the quick open methods. In the event that these generate an unacceptable result (i.e., a root estimate that falls outside the bracket), the algorithm reverts to the more conservative bisection method.
- Although bisection may be slower, it generates an estimate guaranteed to fall within the bracket. This process is then repeated until the root is located to within an acceptable tolerance. As might be expected, bisection typically dominates at first but as the root is approached, the technique shifts to the faster open methods.

(See Fig.6.12 on pp 165)

Homework: Try *fzero* command for different functions.

MATLAB FUNCTION: `fzero`

The `fzero` function is designed to find the real root of a single equation. A simple representation of its syntax is

```
fzero(function,x0)
```

where *function* is the name of the function being evaluated, and *x0* is the initial guess. Note that two guesses that bracket the root can be passed as a vector:

```
fzero(function,[x0 x1])
```

where *x0* and *x1* are guesses that bracket a sign change.

Here is a simple MATLAB session that solves for the root of a simple quadratic: $x^2 - 9$. Clearly two roots exist at -3 and 3. To find the negative root:

```
>> x = fzero(@(x) x^2-9,-4)

x =
    -3
```

If we want to find the positive root, use a guess that is near it:

```
>> x = fzero(@(x) x^2-9,4)

x =
     3
```

Multiple Roots

A *multiple* root corresponds to a point where a function is tangent to the x-axis. For example, a double root results from

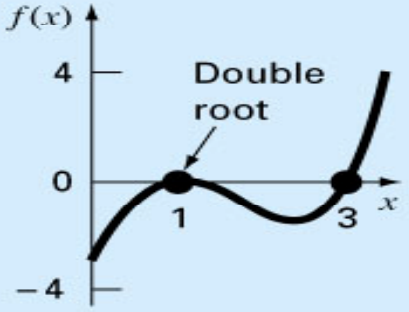
$$f(x) = (x - 3)(x - 1)(x - 1)$$

A *triple* root corresponds to the case where one x value makes three terms in an equation equal to zero, as in:

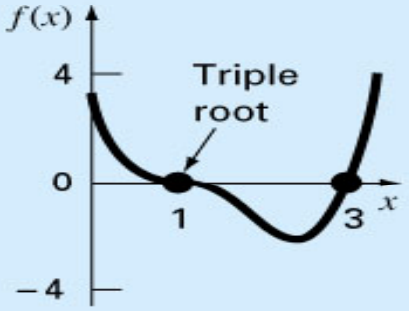
$$f(x) = (x - 3)(x - 1)(x - 1)(x - 1)$$

$$= x^4 - 6x^3 + 12x^2 - 10x + 3.$$

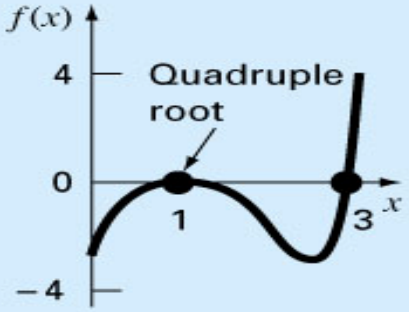
Multiple roots pose some difficulties for many of the numerical methods described here.



(a)



(b)



(c)

FIGURE 6.13
Examples of multiple roots that are tangential to the *x* axis. Notice that the function does not cross the axis on either side of even multiple roots (*a*) and (*c*), whereas it crosses the axis for odd cases (*b*).

Multiple Roots

- None of the methods deal with multiple roots efficiently, however, one way to deal with problems is as follows:

Set $u(x_i) = \dfrac{f(x_i)}{f'(x_i)}$

Then find $x_i + 1 = x_i - \dfrac{u(x_i)}{u'(x_i)}$

This function has roots at all the same locations as the original function

alternative form can be adopted to the Newton-Raphson method

- "Multiple root" corresponds to a point where a function is tangent to the x-axis.
- Difficulties
  - Function does not change sign at the multiple root, therefore, cannot use bracketing methods.
  - Both $f(x)$ and $f'(x)=0$, division by zero with Newton's and Secant methods.
- However, for polynomials whose coefficients are exactly given as integers or rational numbers, there is an efficient method to *factorize* them into factors that have only simple roots and whose coefficients are also exactly given. This method, called *square-free factorization*, is based on the multiple roots of a polynomial being the roots of the greatest common divisor of the polynomial and its derivative.

Polynomials

- Polynomials are a special type of nonlinear algebraic equation of the general form:

$$f_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_{n-1} x^2 + a_n x + a_{n+1}$$

- n: is the order of the polynomial,
- a_i: constant coefficients.
- In many (but not all) cases, the coefficients will be real. For such cases, the roots can be real and/or complex. In general, an n^{th} order polynomial will have n roots.

Using MATLAB to Manipulate Polynomials and Determine Their Roots

Problem Statement. Use the following equation to explore how MATLAB can be employed to manipulate polynomials:

$$f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$$

Note that this polynomial has three real roots: 0.5, −1.0, and 2; and one pair of complex roots: $1 \pm 0.5i$.

MATLAB Function: `roots`

```
Command Window
>> f5=[1 -3.5 2.75 2.125 -3.875 1.25]

f5 =

    1.0000   -3.5000    2.7500    2.1250   -3.8750    1.2500

>> roots(f5)

ans =

   2.0000 + 0.0000i
  -1.0000 + 0.0000i
   1.0000 + 0.5000i
   1.0000 - 0.5000i
   0.5000 + 0.0000i
```

Problem Statement. The roots of a 3rd degree polynomial are given as 1, -1 and 2. Find the polynomial.

MATLAB Function: `poly`

```
Command Window
>> r=[-1 1 2]

r =

    -1     1     2

>> poly(r)

ans =

     1    -2    -1     2

>> r2=[1 2 -1]

r2 =

     1     2    -1

>> poly(r2)

ans =

     1    -2    -1     2
```

Roots are entered

Answer: x^3-2x^2-x+2

```
Command Window
>> roots([1 -2 -1 2])

ans =

   -1.0000
    2.0000
    1.0000
```

Other MATLAB Functions related with polynomials:

>>conv : Convolution and polynomial multiplication

>>polyval : Polynomial evaluation

```
Command Window
>> p1=poly(1)

p1 =

     1    -1

>> p2=poly(-1)

p2 =

     1     1

>> p3=conv(p1,p2)

p3 =

     1     0    -1

>> roots(p3)

ans =

    -1
     1
```

p1=x-1

p2=x+1

p3=p1*p2=(x-1) (x+1)=x^2-1

roots of p3 are -1 and 1

```
>> polyval(p3,2)

ans =

     3
```