NUMERICAL METHODS

Lecture 2

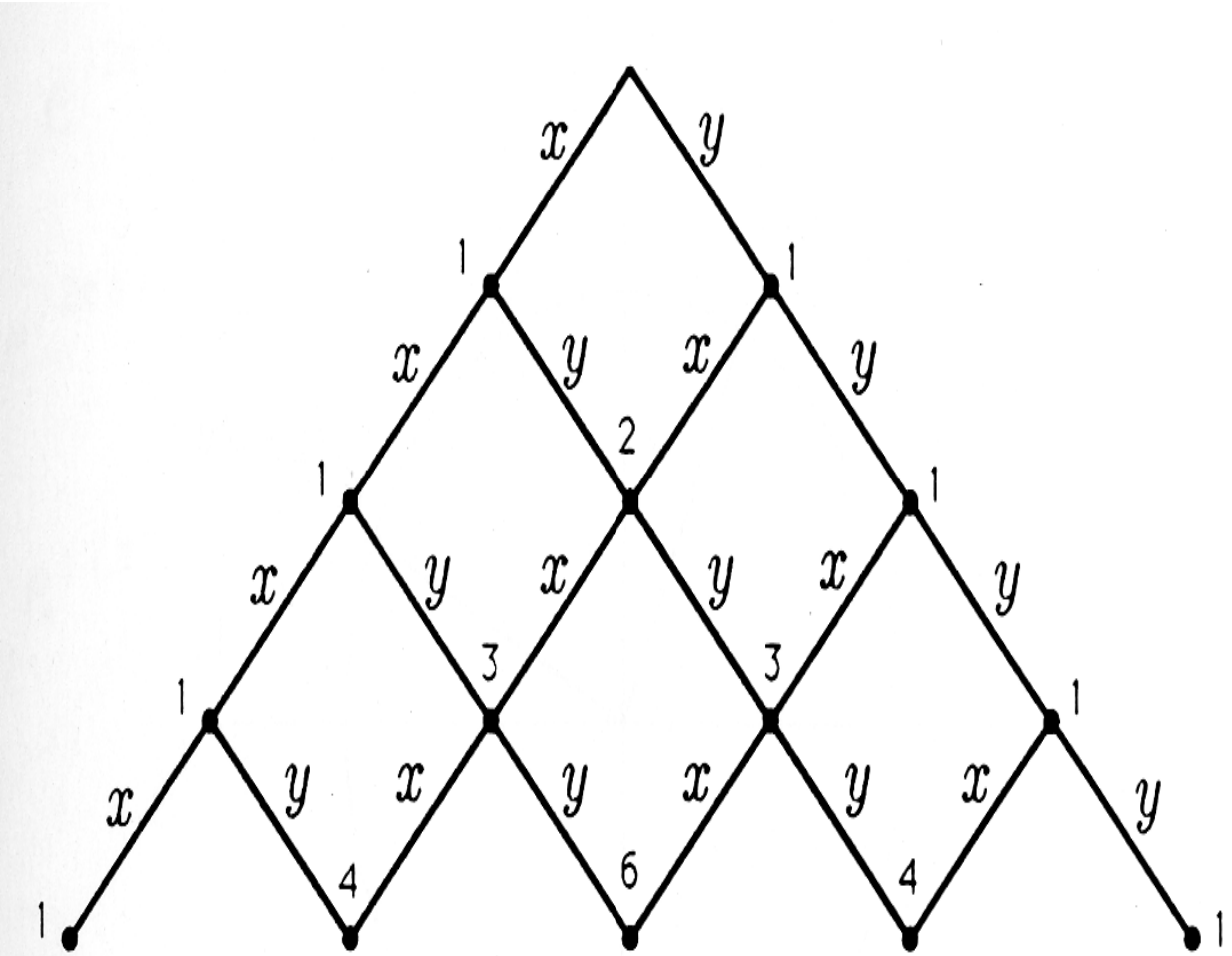
Dr. P V Ramana

Pascal's Triangle

Used for determining a complete set of polynomial terms in 1D:

$$u = \sum_{i=0}^{\infty} a_i \, xi$$

- **Greater the degree of freedom, less stiff will be element.**
- **Interpolation functions are easier to develop with lineal/ areal/ volumes coordinates. The polynomial for 2D**



$$(x+y)^0 = 1$$ — 0th row

$$(x+y)^1 = 1x + 1y$$ — 1st row

$$(x+y)^2 = 1x^2 + 2xy + 1y^2$$ — 2nd row

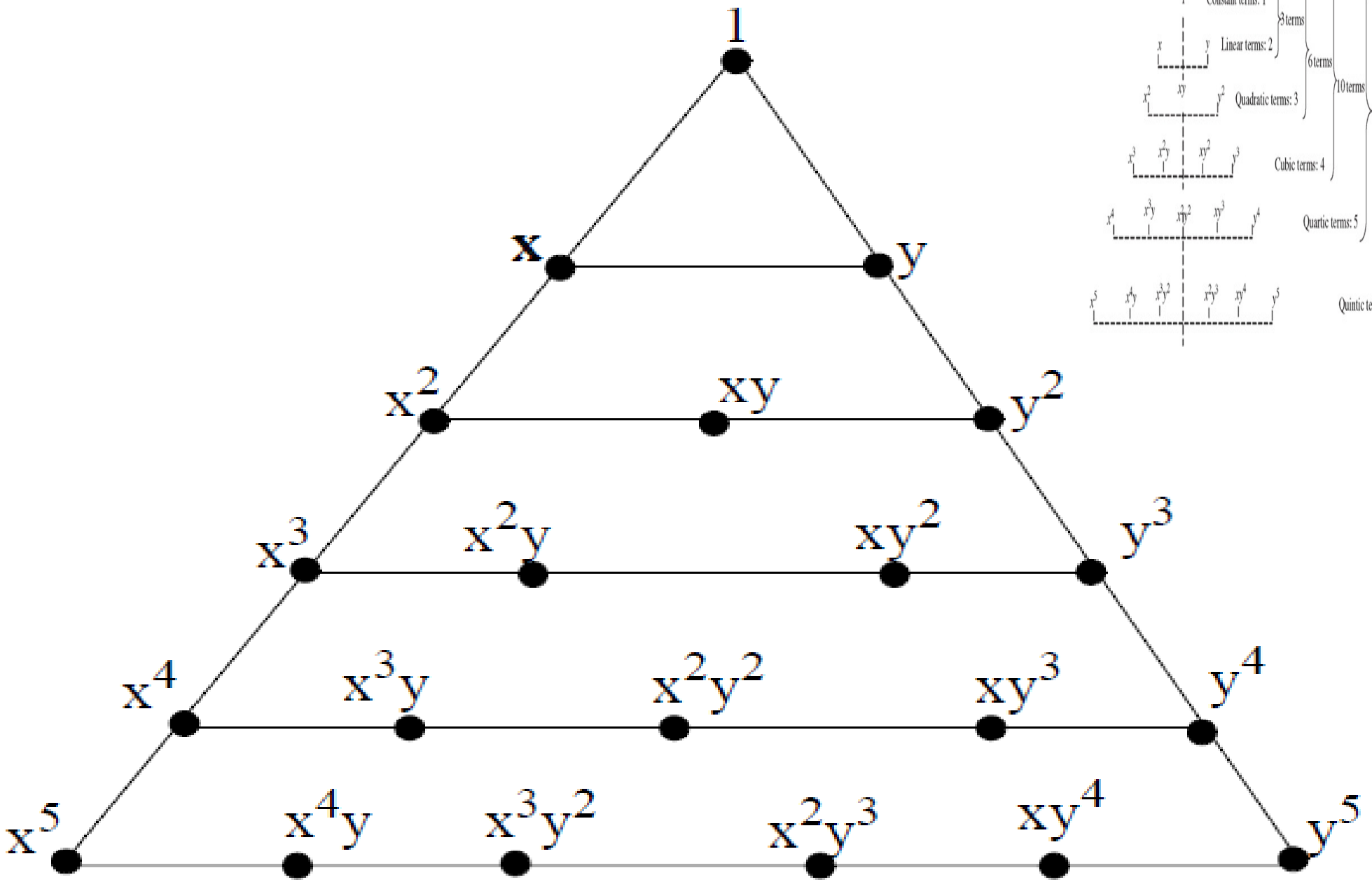
$$(x+y)^3 = 1x^3 + 3x^2y + 3xy^2 + 1y^2$$ — 3rd row

$$(x+y)^4 = 1x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + 1y^4$$ — 4th row

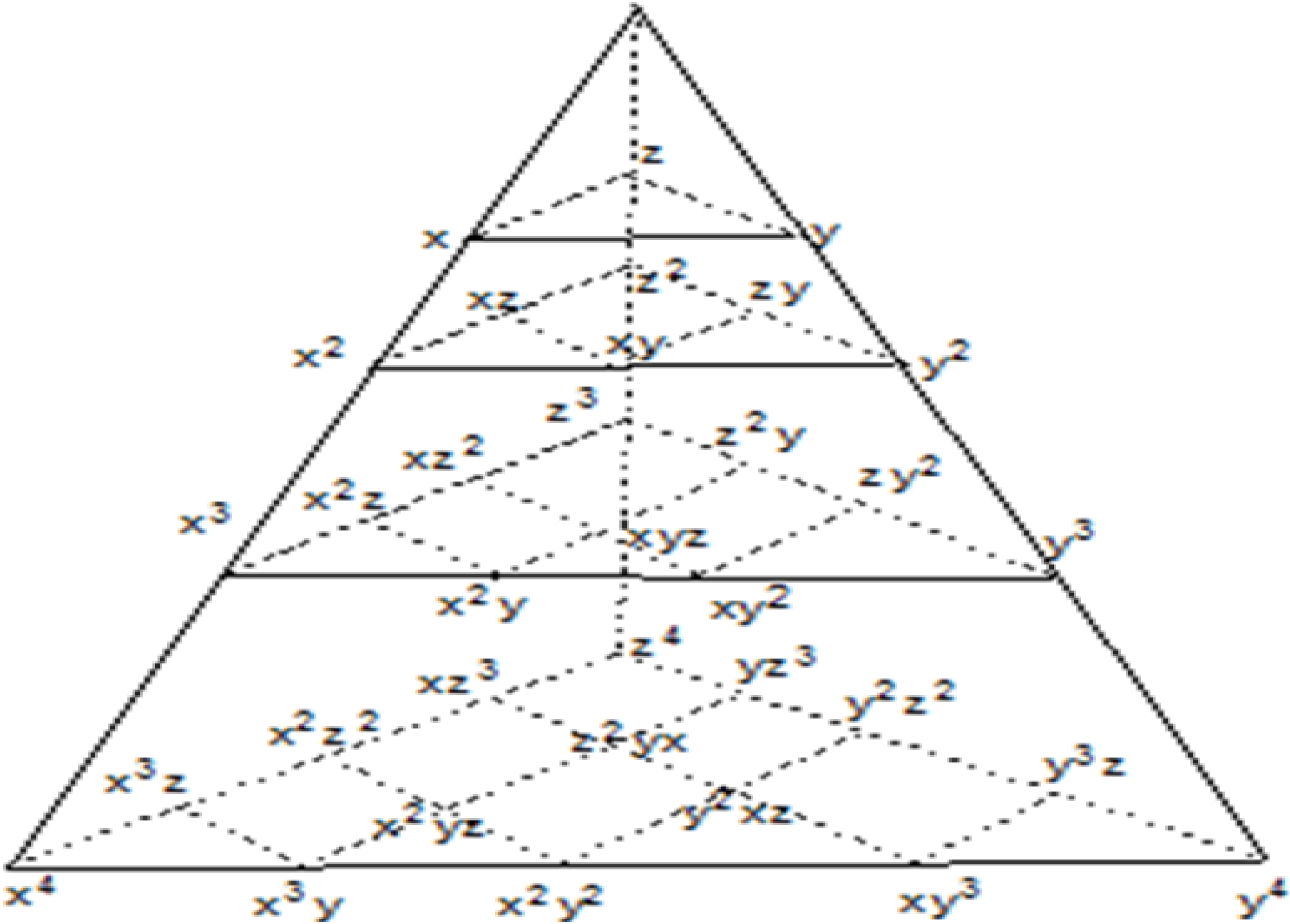
$$(x+y)^5 = 1x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + 1y^5$$ — 5th row

Pascal's Triangle

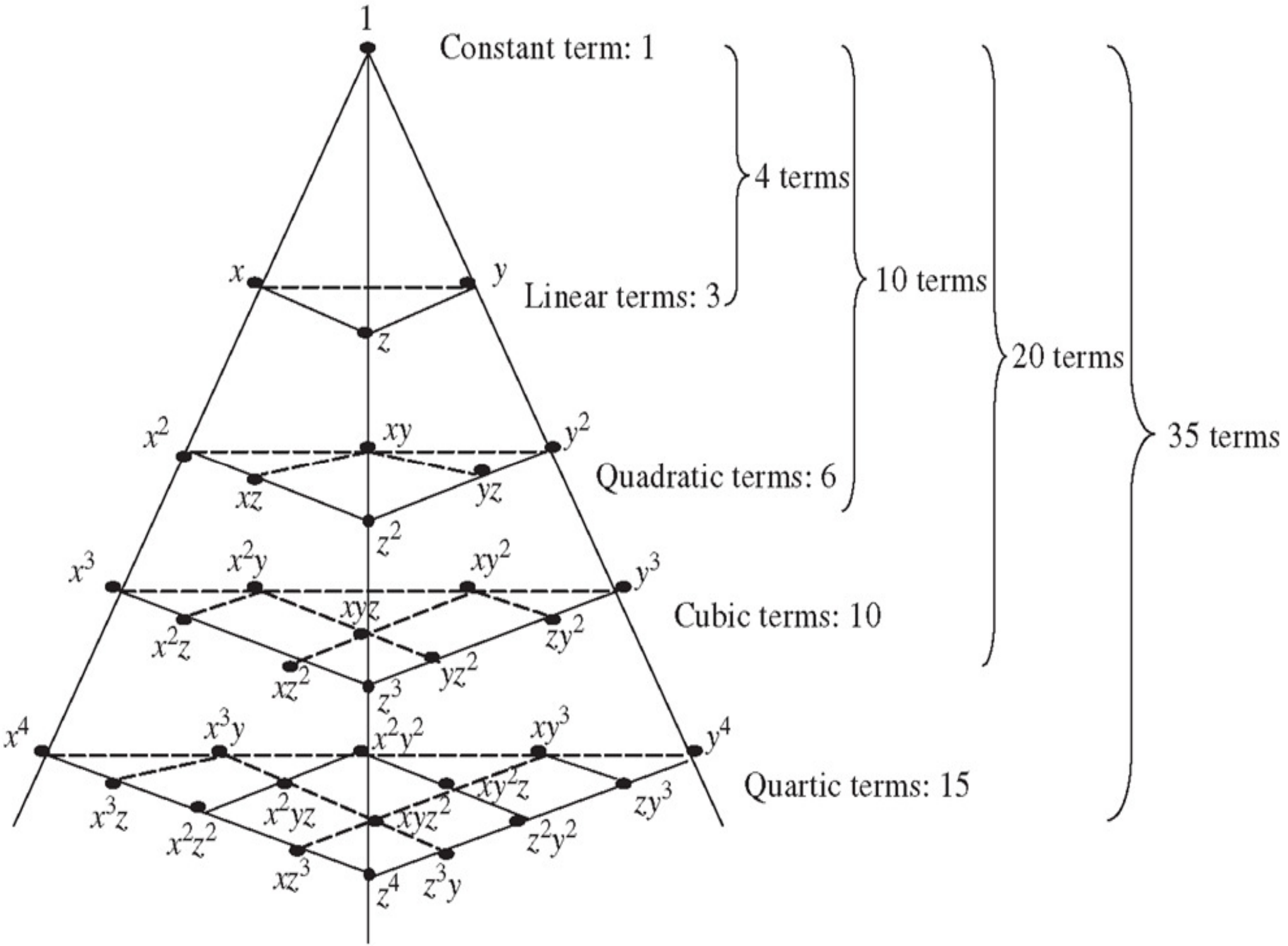
- **Used for determining a complete set of polynomial terms in two dimensions.**

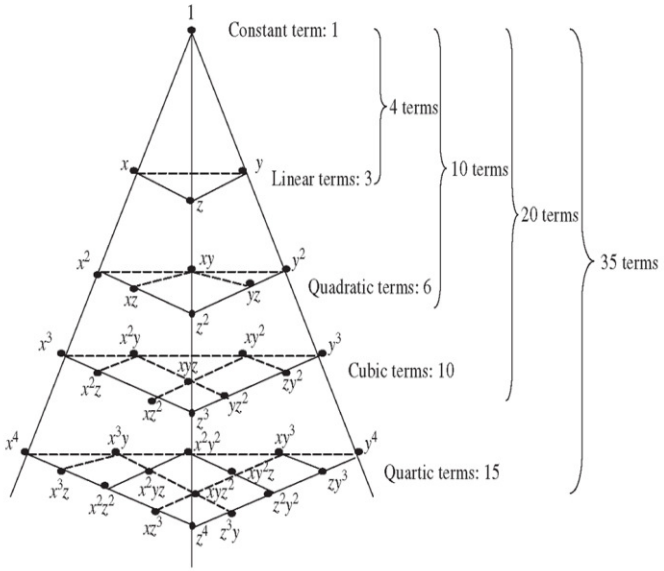


2D Pascal's Triangle



3D Pascal's Triangle


1
Constant term: 1
x
y
z
Linear terms: 3
4 terms
x^2
xy
y^2
xz
yz
z^2
Quadratic terms: 6
10 terms
x^3
x^2y
xy^2
y^3
xyz
x^2z
zy^2
xz^2
yz^2
z^3
Cubic terms: 10
20 terms
x^4
x^3y
x^2y^2
xy^3
y^4
x^3z
x^2yz
xyz^2
xy^2z
zy^3
x^2z^2
z^2y^2
xz^3
z^4
z^3y
Quartic terms: 15
35 terms



$$(x+y+z)^0 = 1$$

$$(x+y+z)^1 = 1x + 1y + 1z$$

$$(x+y+z)^2 = 1x^2 + 1y^2 + 1z^2 + 2xy + 2yz + 2zx$$

$$(x+y+z)^3 = 1x^3 + 1y^3 + 1z^3 + 3xy(x+y) + 3xz(x+z) + 3zy(z+y)$$

$$\vdots$$

$$(x+y+z)^n = \sum_{0<=(i+j)<=n} \frac{n!}{i!\,j!(n-i-j)!} x^i y^j z^{(n-i-j)}$$

Systems of Nonlinear Equations

- Locate the roots of a set of simultaneous nonlinear equations:

$$
\begin{aligned}
&f_1(x_1, x_2, x_3, \mathrm{K}\ , x_n) = 0\\
&f_2(x_1, x_2, x_3, \mathrm{K}\ , x_n) = 0\\
&\mathbf{M}\\
&f_n(x_1, x_2, x_3, \mathrm{K}\ , x_n) = 0
\end{aligned}
$$

Example:

$$x^2 + xy = 10 \quad \Rightarrow \quad u(x, y) = x^2 + xy - 10 = 0$$

$$y + 3xy^2 = 57 \quad \Rightarrow \quad v(x, y) = y + 3xy^2 - 57 = 0$$

- **First-order** Taylor series expansion of a function with more than one variable:

$$u_{i+1} = u_i + \frac{\partial u_i}{\partial x}(x_{i+1} - x_i) + \frac{\partial u_i}{\partial y}(y_{i+1} - y_i) = 0$$

$$v_{i+1} = v_i + \frac{\partial v_i}{\partial x}(x_{i+1} - x_i) + \frac{\partial v_i}{\partial y}(y_{i+1} - y_i) = 0$$

- The root of the equation occurs at the value of x and y where u_{i+1}=0 and v_{i+1}=0 Rearrange to solve for x_{i+1} and y_{i+1}

$$\frac{\partial u_i}{\partial x} x_{i+1} + \frac{\partial u_i}{\partial y} y_{i+1} = -u_i + \frac{\partial u_i}{\partial x} x_i + \frac{\partial u_i}{\partial y} y_i$$

$$\frac{\partial v_i}{\partial x} x_{i+1} + \frac{\partial v_i}{\partial y} y_{i+1} = -v_i + \frac{\partial v_i}{\partial x} x_i + \frac{\partial v_i}{\partial y} y_i$$

$$\begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = -\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- Since x_i, y_i, u_i, and v_i are all known at the i^{th} iteration, this represents a set of two linear equations with two unknowns, x_{i+1} and y_{i+1}
- You may use several techniques to solve these equations

Newton's Method for Systems of Non Linear Equations

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \left(F'\right)^{-1}(x_i) \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = -\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Given: X_0 an initial guess of the root of $F(x) = 0$

Newton's Iteration

$$X_{k+1} = X_k - \left[F'(X_k)\right]^{-1} F(X_k)$$

$$F(X) = \begin{bmatrix} f_1(x_1, x_2, \ldots) \\ f_2(x_1, x_2, \ldots) \\ \mathrm{M} \end{bmatrix}, \quad F'(X) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \mathrm{M} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \\ \mathrm{M} & & \end{bmatrix}$$

Example 1

$$\begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = -\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- Solve the following system of equations:

$$y + x^2 - 0.5 - x = 0$$
$$x^2 - 5xy - y = 0$$
$$\text{Initial guess } \; x = 1, \;\; y = 0$$

$$X_{k+1} = X_k - [F'(X_k)]^{-1} F(X_k)$$

$$F = \begin{bmatrix} y + x^2 - 0.5 - x \\ x^2 - 5xy - y \end{bmatrix}, \; F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x - 5y & -5x - 1 \end{bmatrix}, \; X_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Solution Using Newton's Method

Iteration 1 :

$$F = \begin{bmatrix} y + x^2 - 0.5 - x \\ x^2 - 5xy - y \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} =, \quad F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x - 5y & -5x - 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & -6 \end{bmatrix}$$

$$X_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 2 & -6 \end{bmatrix}^{-1} \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.25 \\ 0.25 \end{bmatrix}$$

$$X_{k+1} = X_k - [F'(X_k)]^{-1} F(X_k)$$

$$\begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = -\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Iteration 2 :

$$F = \begin{bmatrix} 0.0625 \\ -0.25 \end{bmatrix}, \quad F' = \begin{bmatrix} 1.5 & 1 \\ 1.25 & -7.25 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 1.25 \\ 0.25 \end{bmatrix} - \begin{bmatrix} 1.5 & 1 \\ 1.25 & -7.25 \end{bmatrix}^{-1} \begin{bmatrix} 0.0625 \\ -0.25 \end{bmatrix} = \begin{bmatrix} 1.2332 \\ 0.2126 \end{bmatrix}$$

Example 1b

$$\begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = -\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \frac{\partial u_i}{\partial x} & \frac{\partial u_i}{\partial y} \\ \frac{\partial v_i}{\partial x} & \frac{\partial v_i}{\partial y} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- Solve the following system of equations:

$$y + x^2 - 1 - x = 0$$

$$x^2 - 2y^2 - y = 0$$

$$\text{Initial guess } \; x = 0, \;\; y = 0$$

$$F = \begin{bmatrix} y + x^2 - 1 - x \\ x^2 - 2y^2 - y \end{bmatrix}, \; F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x & -4y - 1 \end{bmatrix}, \; X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$X_{k+1} = X_k - [F'(X_k)]^{-1} F(X_k)$$

Example

Solution

Iteration	0	1	2	3	4	5
X_k	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}-1\\0\end{bmatrix}$	$\begin{bmatrix}-0.6\\0.2\end{bmatrix}$	$\begin{bmatrix}-0.5287\\0.1969\end{bmatrix}$	$\begin{bmatrix}-0.5257\\0.1980\end{bmatrix}$	$\begin{bmatrix}-0.5257\\0.1980\end{bmatrix}$

$$X_{k+1} = X_k - [F'(X_k)]^{-1} F(X_k)$$

$$F = \begin{bmatrix} y + x^2 - 1 - x \\ x^2 - 2y^2 - y \end{bmatrix}, \quad F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x & -4y - 1 \end{bmatrix}, \quad X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Newton's Method (Review)

Assumptions: $f(x)$, $f'(x)$, x_0 *are available,*

$$f'(x_0) \neq 0$$

Newton's Method new estimate:

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

Problem:

$f'(x_i)$ is not available,

or difficult to obtain analytically.

Convergence of Newton's Method

- **The error of the Newton-Raphson method (for single root) can be estimated from**

$$f(x) = f(x_k) + (x - x_k) f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(\eta)$$

$$f(x^*) = 0 = f(x_k) + (x^* - x_k) f'(x_k) + \frac{1}{2}(x^* - x_k)^2 f''(\eta)$$

$$\Rightarrow x^* = \underbrace{x_k - \frac{f(x_k)}{f'(x_k)}}_{x_{k+1}} - (x^* - x_k)^2 \frac{f''(\eta)}{2 f'(x_k)}$$

$$\frac{x^* - x_{k+1}}{(x^* - x_k)^2} = \frac{-f''(x^*)}{2 f'(x^*)} = \frac{\varepsilon_{k+1}}{\varepsilon_k^2}$$

Quadratic convergence

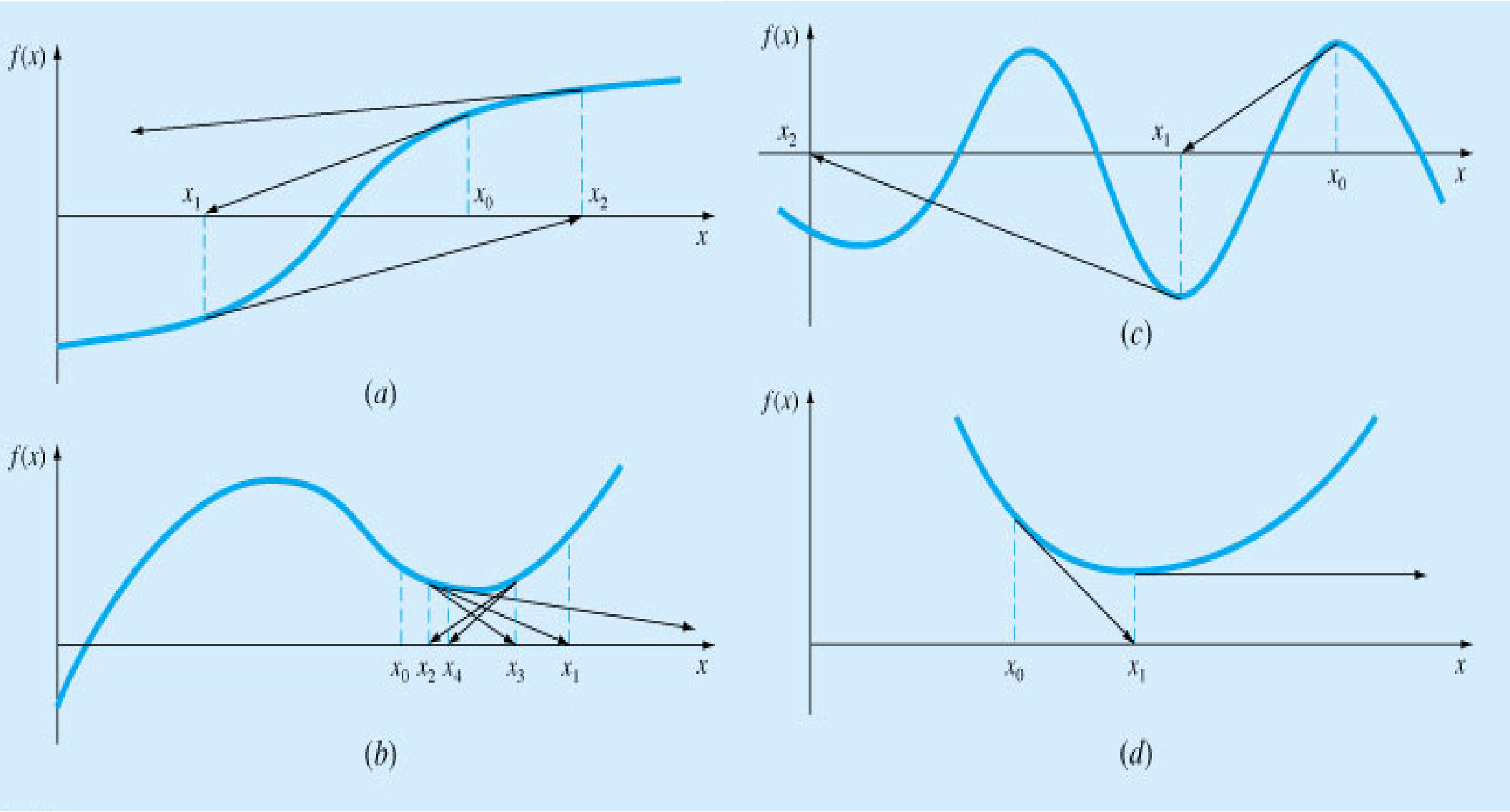
Newton-Raphson Method

Although Newton-Raphson converges very rapidly, it may diverge and fail to find roots

- **If an inflection point (f”=0) is near the root**
- **If there is a local minimum or maximum (f’=0)**
- **If there are multiple roots**
- **If a zero slope is reached**

Open Method, Convergence not guaranteed

Newton-Raphson Method



Examples of poor convergence

Bungee Jumper Problem

- Newton-Raphson method
- Need to evaluate the function and its derivative

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

$$\frac{df(m)}{dt} = \frac{1}{2}\sqrt{\frac{g}{mc_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - \frac{g}{2m} t \sec h^2\left(\sqrt{\frac{gc_d}{m}}t\right)$$

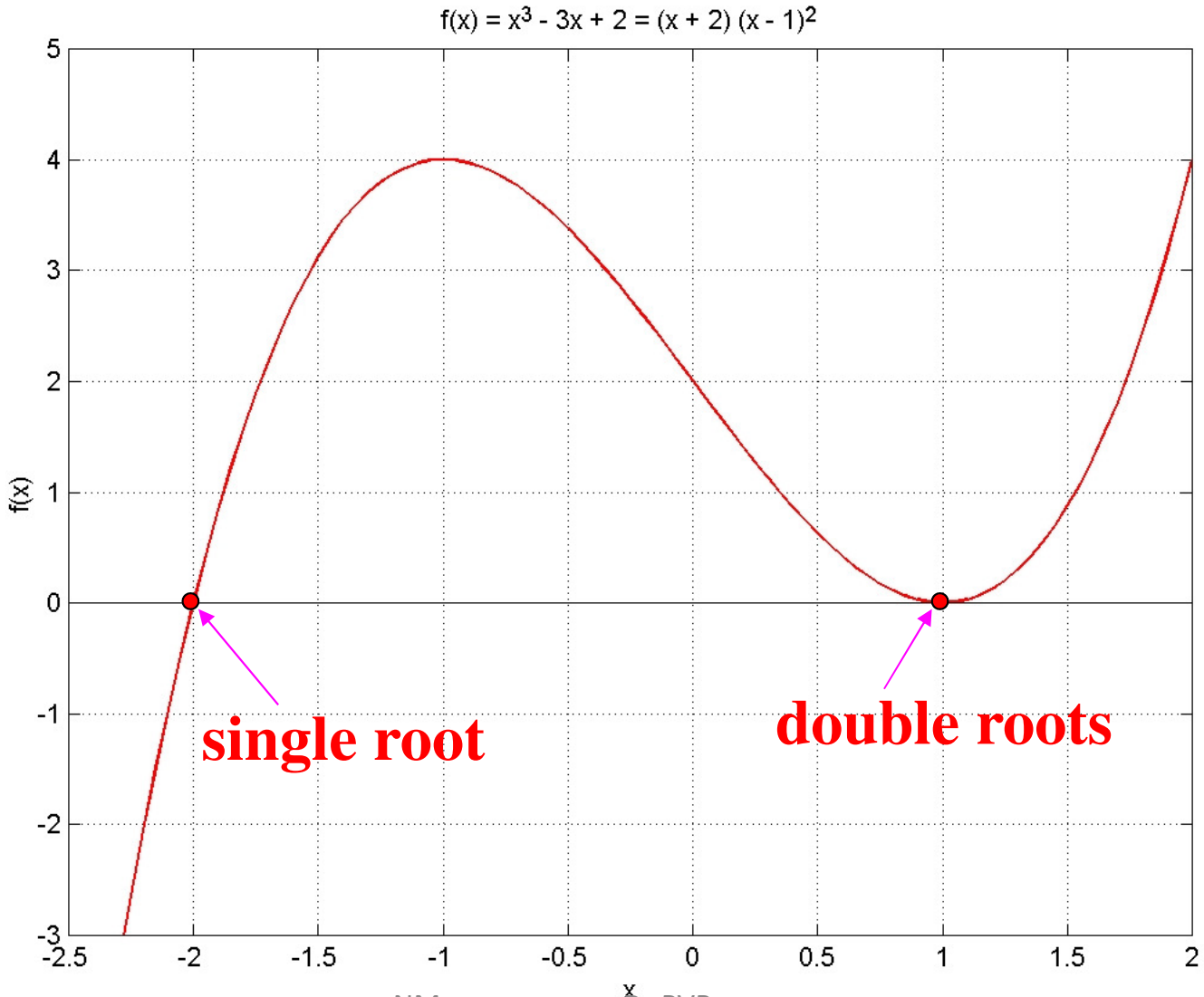
Given c_d = 0.25 kg/m, v = 36 m/s, t = 4 s, and g = 9.81 m^2/s, determine the mass of the bungee jumper

Bungee Jumper Problem

```
>> y=inline('sqrt(9.81*m/0.25)*tanh(sqrt(9.81*0.25/m)*4)-36','m')

y =

     Inline function:

     y(m) = sqrt(9.81*m/0.25)*tanh(sqrt(9.81*0.25/m)*4)-36

>> dy=inline('1/2*sqrt(9.81/(m*0.25))*tanh(sqrt(9.81*0.25/m)*4)-
9.81/(2*m)*4*sech(sqrt(9.81*0.25/m)*4)^2','m')

dy =

     Inline function:

     dy(m) = 1/2*sqrt(9.81/(m*0.25))*tanh(sqrt(9.81*0.25/m)*4)-
9.81/(2*m)*4*sech(sqrt(9.81*0.25/m)*4)^2

>> format short; root = newtraph(y,dy,140,0.00001)

root =

  142.7376
```

Multiple Roots

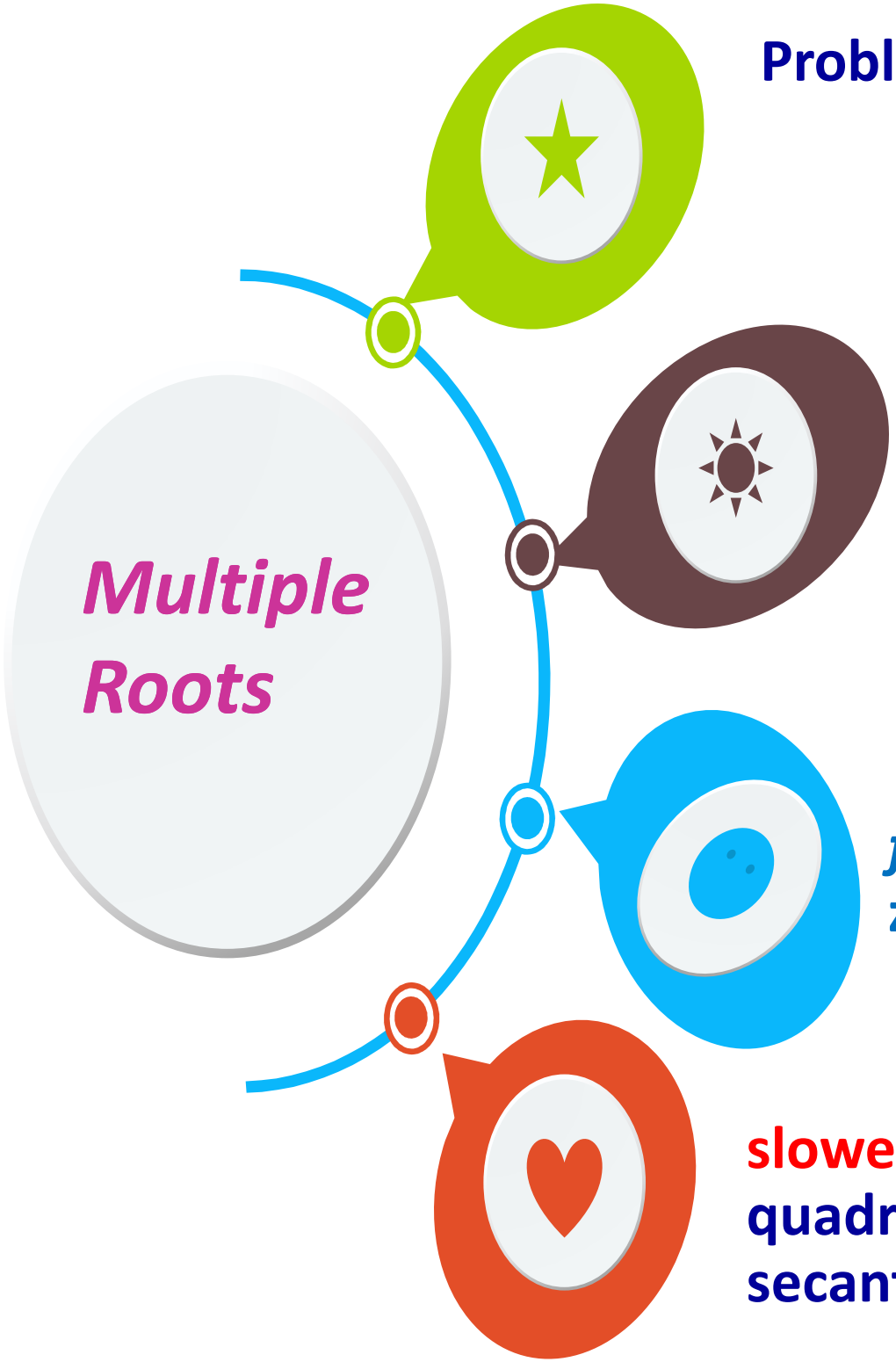
- **A multiple root (double, triple, etc.) occurs where the function is tangent to the x axis**



Multiple Roots

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

- **Problems with multiple roots**
- **The function does not change sign at even multiple roots (i.e., m = 2, 4, 6, …)**
- ***f*′(*x*) goes to zero - need to put a zero check for *f*(*x*) in program**
- **slower convergence (linear instead of quadratic) of Newton-Raphson and secant methods for multiple roots**



Problems with multiple roots

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

The function does not change sign at even multiple roots (i.e., m = 2, 4, 6, …)

f′(*x*) goes to zero - need to put a zero check for *f*(*x*) in program

slower convergence (linear instead of quadratic) of Newton-Raphson and secant methods for multiple roots

Modified Newton-Raphson Method

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

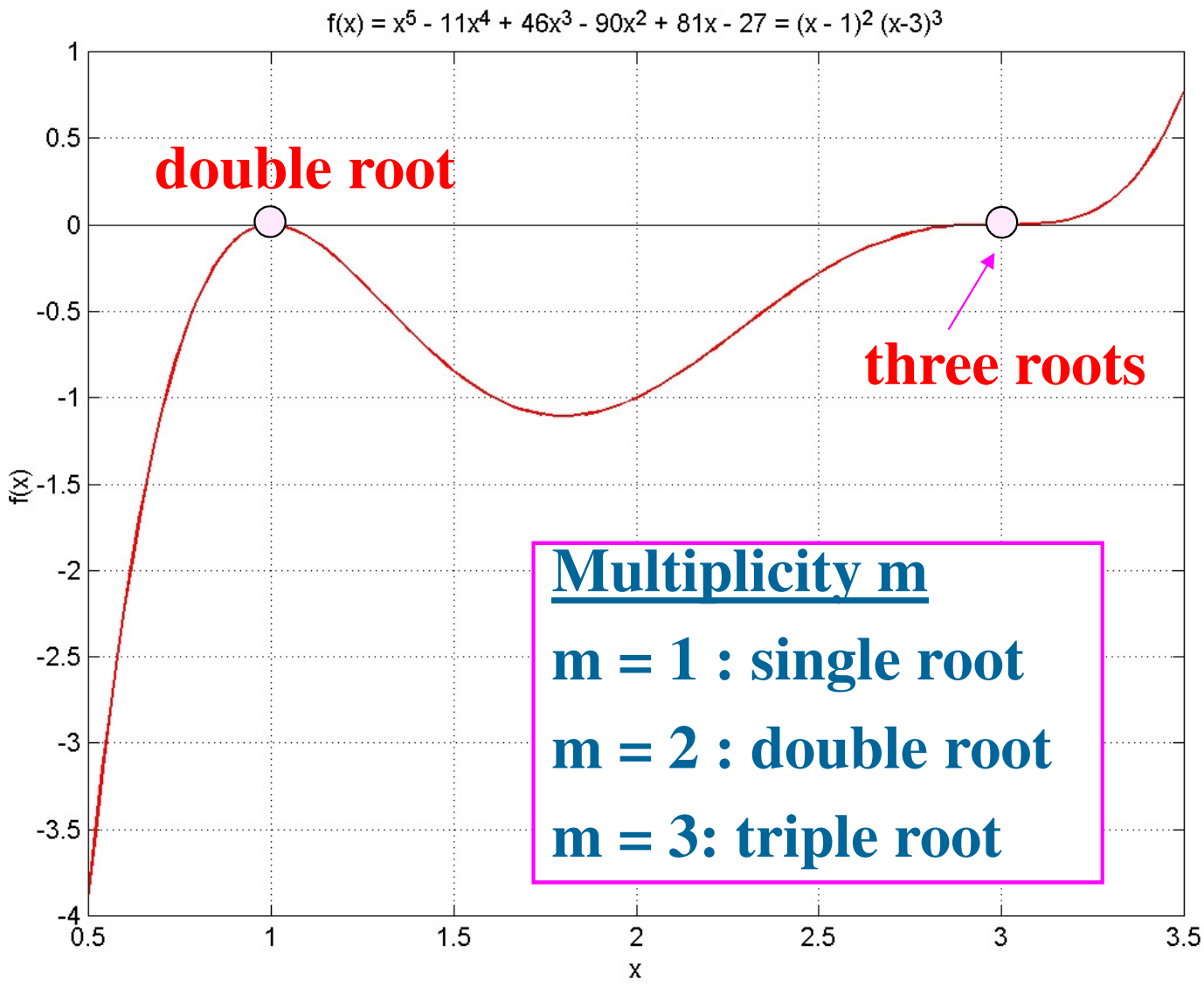
- **When the multiplicity of the root is known**

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

- **Double root : m = 2**
- **Triple root : m = 3**
- **Simple, but need to know the multiplicity m**
- **Maintain quadratic convergence**

Multiple Root with Multiplicity m

$$f(x)=x^5-11x^4+46x^3-90x^2+81x-27$$



Multiplicity m

m = 1 : single root

m = 2 : double root

m = 3: triple root

Modified Newton's Method

Can be used for both single and multiple roots

(m = 1: original Newton's method)

m =1: single root

m=2, double root

m=3 triple root

etc.

```
function [x, f] = multiple1(func, dfunc)

% Find multiple root near xguess using the modified Newton's method
% Multiplicity m of the root is given -- m = 1: single root
% m = 2: double root; m = 3: triple root, etc.
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

m = input('enter multiplicity of the root = ');
xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1) - m * f(i-1) / dfdx(i-1);
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     if abs(x(i) - x(i-1)) < es
        disp('Newton 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;
disp('   step                x                    f                  df/dx')
out = [k; x; f; dfdx]';
fprintf('%5.0f  %20.14f  %21.15f  %21.15f\n',out)
```

Original Newton's method
m = 1

```
» multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 1
enter initial guess x1 = 1.3
allowable tolerance tol = 1.e-6
maximum number of iterations max = 100
Newton method has converged
  step          x                        y
   1    1.30000000000000   -0.442170000000004
   2    1.09600000000000   -0.063612622209021
   3    1.04407272727272   -0.014534428477418
   4    1.02126549372889   -0.003503591972482
   5    1.01045853297516   -0.000861391389428
   6    1.00518770530932   -0.000213627276750
   7    1.00258369467652   -0.000053197123947
   8    1.00128933592285   -0.000013273393044
   9    1.00064404356011   -0.000003315132176
  10    1.00032186610620   -0.000000828382262
  11    1.00016089418619   -0.000000207045531
  12    1.00008043738571   -0.000000051755151
  13    1.00004021625682   -0.000000012938003
  14    1.00002010751461   -0.000000003234405
  15    1.00001005358967   -0.000000000808605
  16    1.00000502663502   -0.000000000202135
  17    1.00000251330500   -0.000000000050527
  18    1.00000125681753   -0.000000000012626
  19    1.00000062892307   -0.000000000003162
```

Modified Newton's Method
m = 2

```
» multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 2
enter initial guess x1 = 1.3
allowable tolerance tol = 1.e-6
maximum number of iterations max = 100
Newton method has converged
  step          x                        y
   1    1.30000000000000   -0.442170000000004
   2    0.89199999999999   -0.109259530656779
   3    0.99229251101321   -0.000480758689392
   4    0.99995587111371   -0.000000015579900
   5    0.99999999853944   -0.000000000000007
   6    1.00000060664549   -0.000000000002935
```

Double root : m = 2

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

Modified Newton's Method with u = f / f´

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

- **A more general modified Newton-Raphson method for multiple roots**

$$u = \frac{f(x)}{f'(x)}$$

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

$$f(x) = (x - x^*)^m$$

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

$$u(x) = \frac{f(x)}{f'(x)} = \frac{x - x^*}{m}$$

- ***u*(*x*) contains only single roots even though *f*(*x*) may have multiple roots**

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$$

```
function [x, f] = multiple2(func, dfunc, ddfunc)

% Find multiple root near xguess using the modified Newton's method.
% Multiplicity of the root is not known a priori
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       ddfunc    name of second derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
d2fdx2(1) = feval(ddfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1)-f(i-1)*dfdx(i-1)/(dfdx(i-1)^2-f(i-1)*d2fdx2(i-1));
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     d2fdx2(i) = feval(ddfunc, x(i));
     if abs(x(i)-x(i-1)) < es
        disp('Newton 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;
disp('   step         x              f           df/dx          d2f/dx2')
out=[k; x; f; dfdx; d2fdx2];
fprintf('%5.0f  %17.14f  %20.15f %20.15f %20.15f\n',out)
```

Modified Newton's method with $u = f / f'$

```
function f = multi_func(x)
% Exact solutions: x = 1 (double) and 2 (triple)
f = x.^5 - 11*x.^4 + 46*x.^3 - 90*x.^2 + 81*x - 27;
```

```
function f_pr = multi_pr(x)
% First derivative f'(x)
f_pr = 5*x.^4 - 44*x.^3 + 138*x.^2 - 180*x + 81;
```

```
function f_pp = multi_pp(x)
% Second-derivative f''(x)
f_pp = 20*x.^3 - 132*x.^2 + 276*x - 180;
```

```
>> [x, f] = multiple2('multi_func','multi_dfunc','multi_ddfunc');
enter initial guess: xguess = 0
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 100
Newton method has converged
```

Double root at x = 1

step	x	f	df/dx	d2f/dx2
1	0.00000000000000	-27.000000000000000	81.000000000000000	-180.000000000000000
2	1.28571428571429	-0.411257214255940	-2.159100374843831	-0.839650145772595
3	1.08000000000002	-0.045298483200014	-1.061683200000175	-10.690559999999067
4	1.00519480519482	-0.000214210129556	-0.082148747927818	-15.627914214305775
5	1.00002034484531	-0.000000003311200	-0.000325502624349	-15.998535200938932
6	1.00000000031772	0.000000000000000	-0.000000005083592	-15.999999977123849
7	1.00000000031772	0.000000000000000	-0.000000005083592	-15.999999977123849

Original Newton's method (m = 1)

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

```
>> [x,f] = multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 1
enter initial guess: xguess = 10
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 200
Newton method has converged
```

Triple Root at x = 3

step	x	f	df/dx
1	10.00000000000000	27783.000000000000000	18081.000000000000000
2	8.46341463414634	9083.801268988610900	7422.201416184873800
3	7.23954576295397	2966.633736828044700	3050.171568370705200
4	6.26693367529599	967.245352637683710	1255.503689063504700
5	5.49652944545325	314.604522684684700	517.982397606370110
6	4.88916416791005	101.981559887686160	214.391058318088990
7	4.41348406871311	32.905501521441806	89.118850798301651
8	4.04425240530314	10.553044477409856	37.250604948102705
9	3.76095379868689	3.358869623128157	15.675199755246240
10	3.54667457573766	1.059579469957555	6.646809147676663
…	……..	……	…..
130	2.99988506446967	-0.000000000006168	0.000000158497869
131	2.99992397673381	-0.000000000001762	0.000000069347379
132	2.99994938715307	-0.000000000000426	0.000000030737851
133	2.99996325688118	-0.000000000000085	0.000000016199920
134	2.99996852018682	0.000000000000000	0.000000011891075
135	2.99996852018682	0.000000000000000	0.000000011891075

Modified Newton's method

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

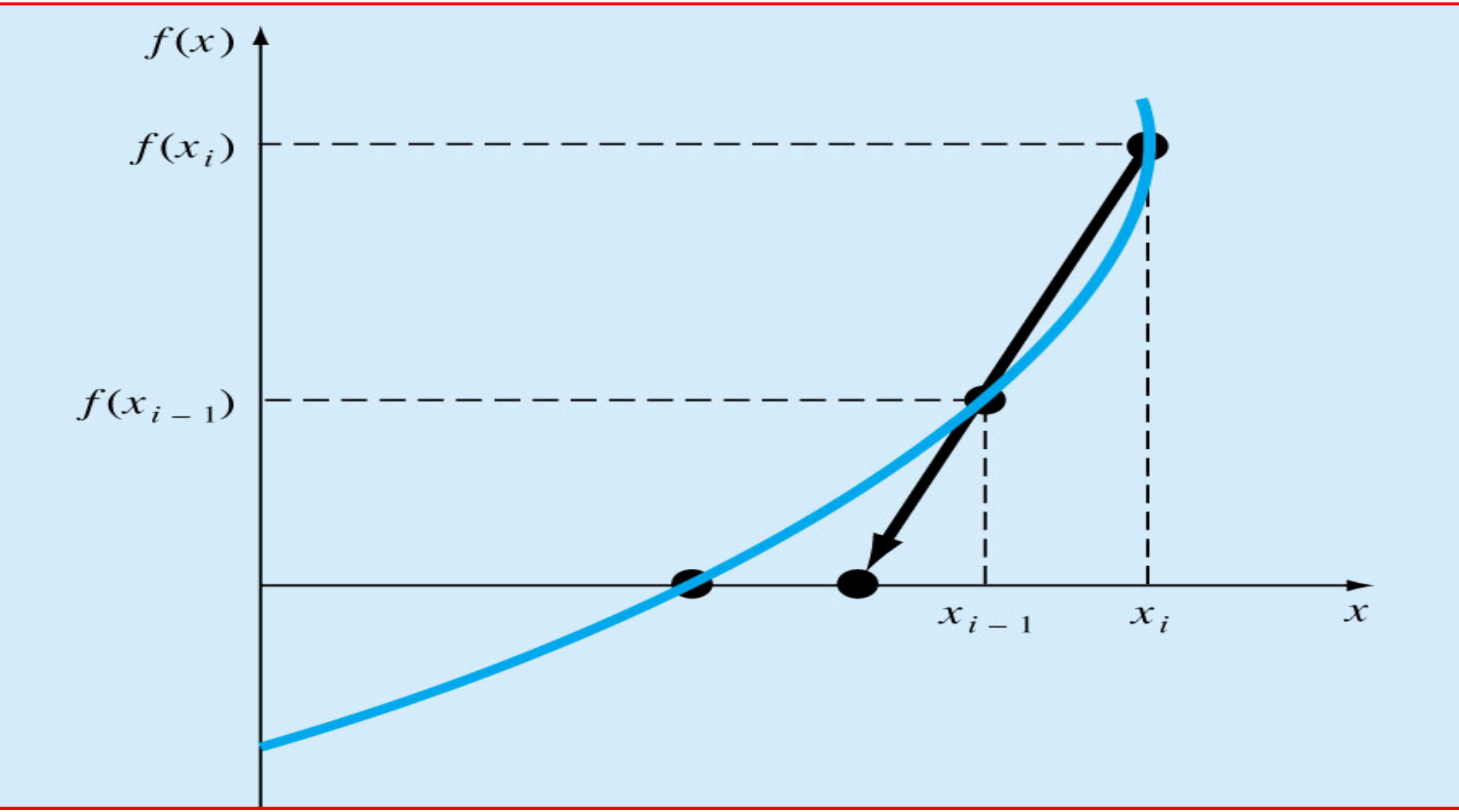
```
>> [x,f] = multiple2('multi_func','multi_dfunc','multi_ddfunc');
enter initial guess: xguess = 10
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 100
Newton method has converged
```

Triple root at x = 3

step	x	f	df/dx	d2f/dx2
1	10.00000000000000	27783.000000000000000	18081.000000000000000	9380.000000000000000
2	2.42521994134897	-0.385717068699165	1.471933198691602	-1.734685930313219
3	2.80435435817775	-0.024381150764611	0.346832001230098	-3.007964394244482
4	2.98444590681717	-0.000014818785758	0.002843242444783	-0.361760865258020
5	2.99991809093254	-0.000000000002188	0.000000080500286	-0.001965495593481
6	2.99999894615774	-0.000000000000028	0.000000000013529	-0.000025292161013
7	2.99999841112323	0.000000000000000	0.000000000030582	-0.000038132921304

- **Original Newton-Raphson method required 135 iterations**
- **Modified Newton's method converged in only 7 iterations**

Secant Method



- **Use secant line instead of tangent line at $f(x_i)$**

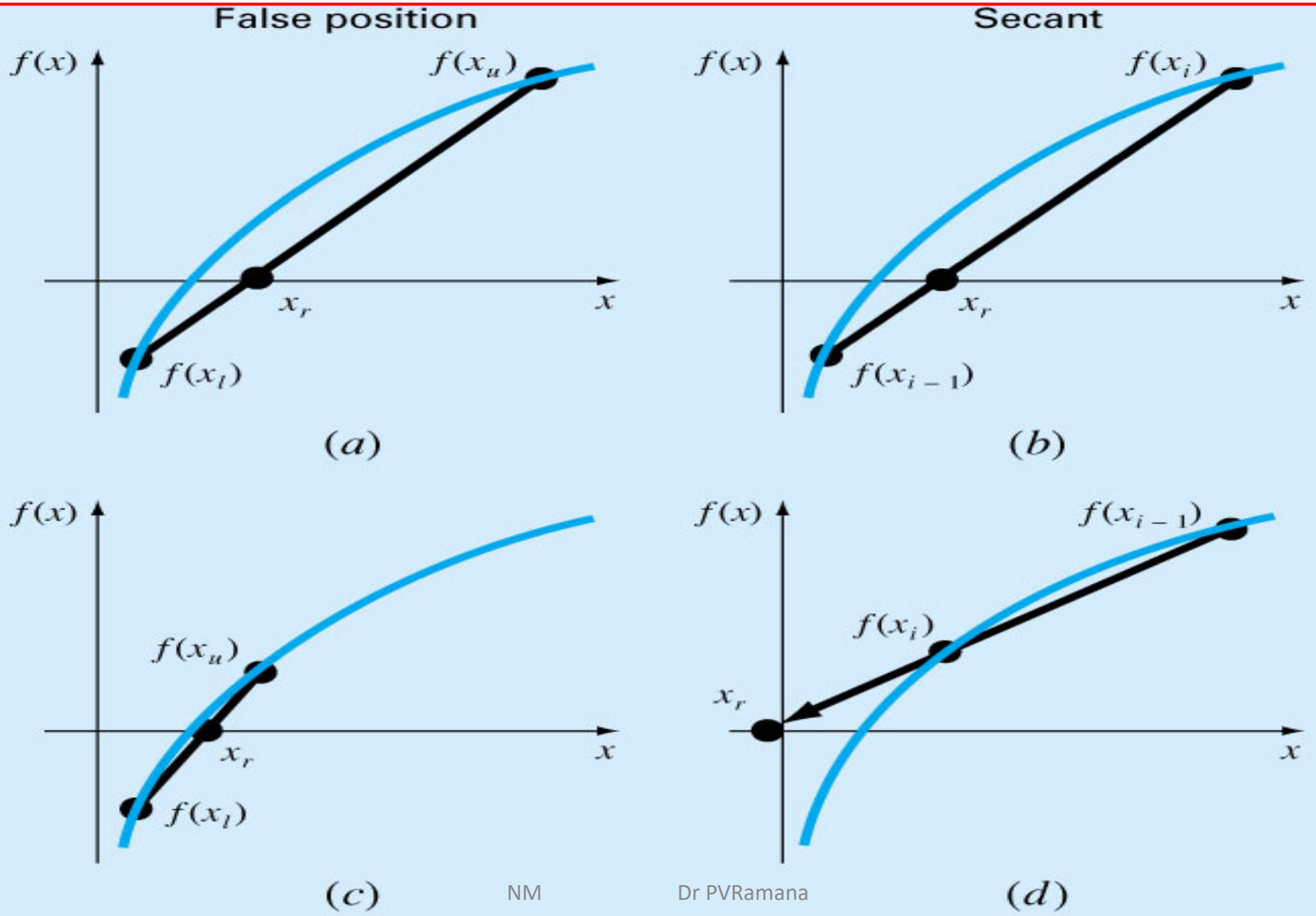
Secant Method

- The formula for the secant method is

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

- Notice that this is very similar to the false position method in form
- Still requires two initial estimates
- But it doesn't bracket the root at all times - there is no sign test

False-Position and Secant Methods



Algorithm for Secant method

- **Open Method**

1. **Begin with any two endpoints $[a, b] = [x_0, x_1]$**
2. **Calculate x_2 using the secant method formula**

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

3. **Replace x_0 with x_1, replace x_1 with x_2 and repeat from (2) until convergence is reached**

- **Use the two most recently generated points in subsequent iterations (not a bracket method!)**

Secant Method

- **Advantage of the secant method -**
- **It can converge even faster and it doesn't need to bracket the root**

- **Disadvantage of the secant method -**
- **It is not guaranteed to converge!**
- **It may diverge (fail to yield an answer)**

The Secant Method

- If derivative *f'(x)* can not be computed analytically then one need to compute it numerically (*backward finite divided difference method*)

RESULT: N-R ➔becomes➔ SECANT METHOD

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

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

Secant : $$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \qquad i = 1,2,3,\text{K}$$

The Secant Method

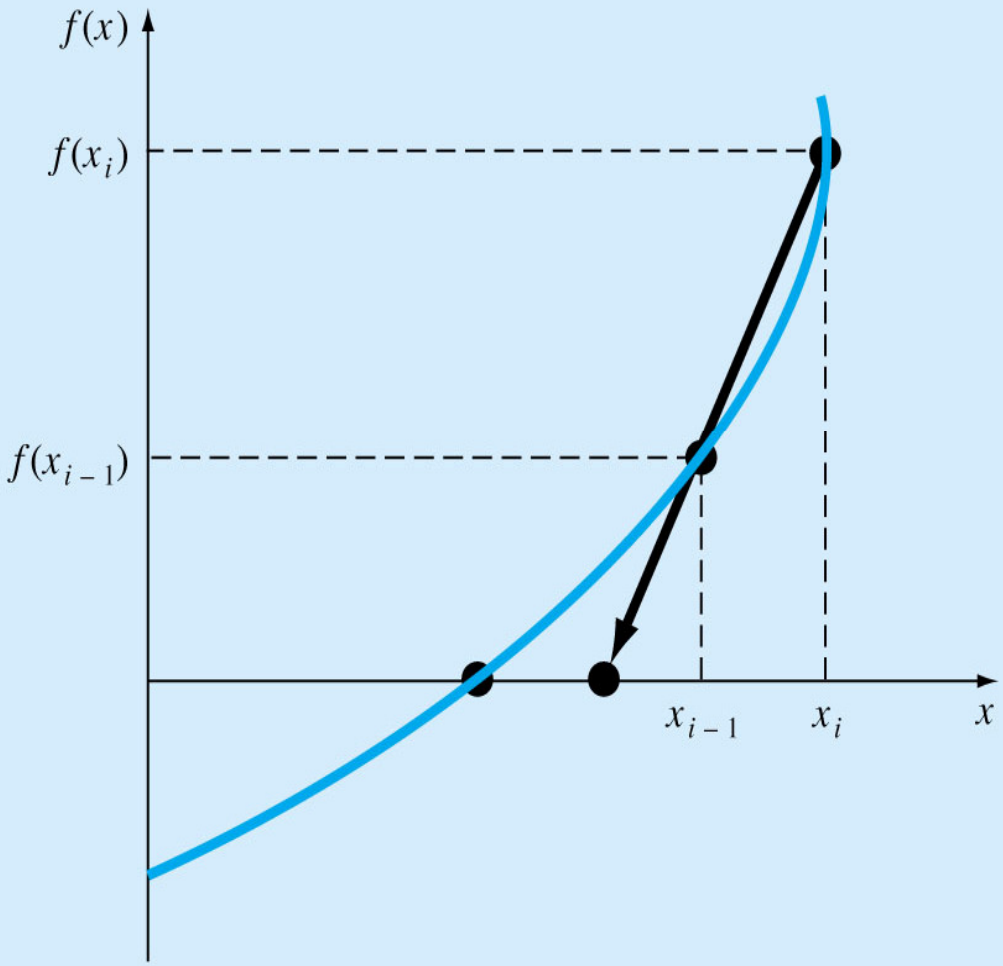
$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

- **Requires two initial estimates x_o, x_1.**

  However, it is not 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 Method

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

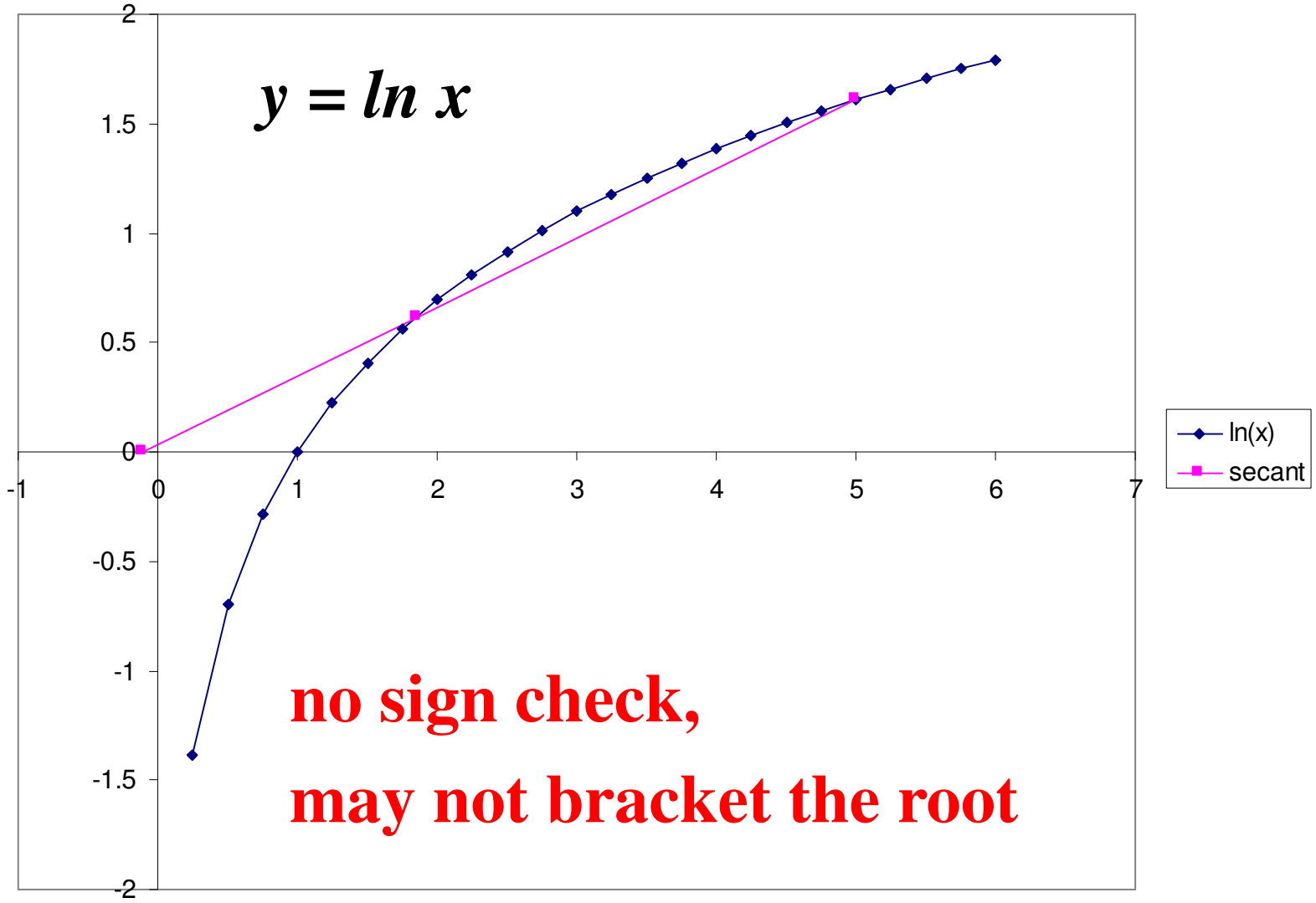
Original Secant : $$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Use a small perturbation fraction δ to compute

$$f'(x_i) = \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$ **in the original N - R formula :**

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

Convergence not Guaranteed



Secant method

```
» [x1 f1]=secant('my_func',0,1,1.e-15,100);
secant method has converged
```

step	x	f
1.0000	0	1.0000
2.0000	1.0000	-1.0000
3.0000	0.5000	-0.3750
4.0000	0.2000	0.4080
5.0000	0.3563	-0.0237
6.0000	0.3477	-0.0011
7.0000	0.3473	0.0000
8.0000	0.3473	0.0000
9.0000	0.3473	0.0000
10.0000	0.3473	0.0000

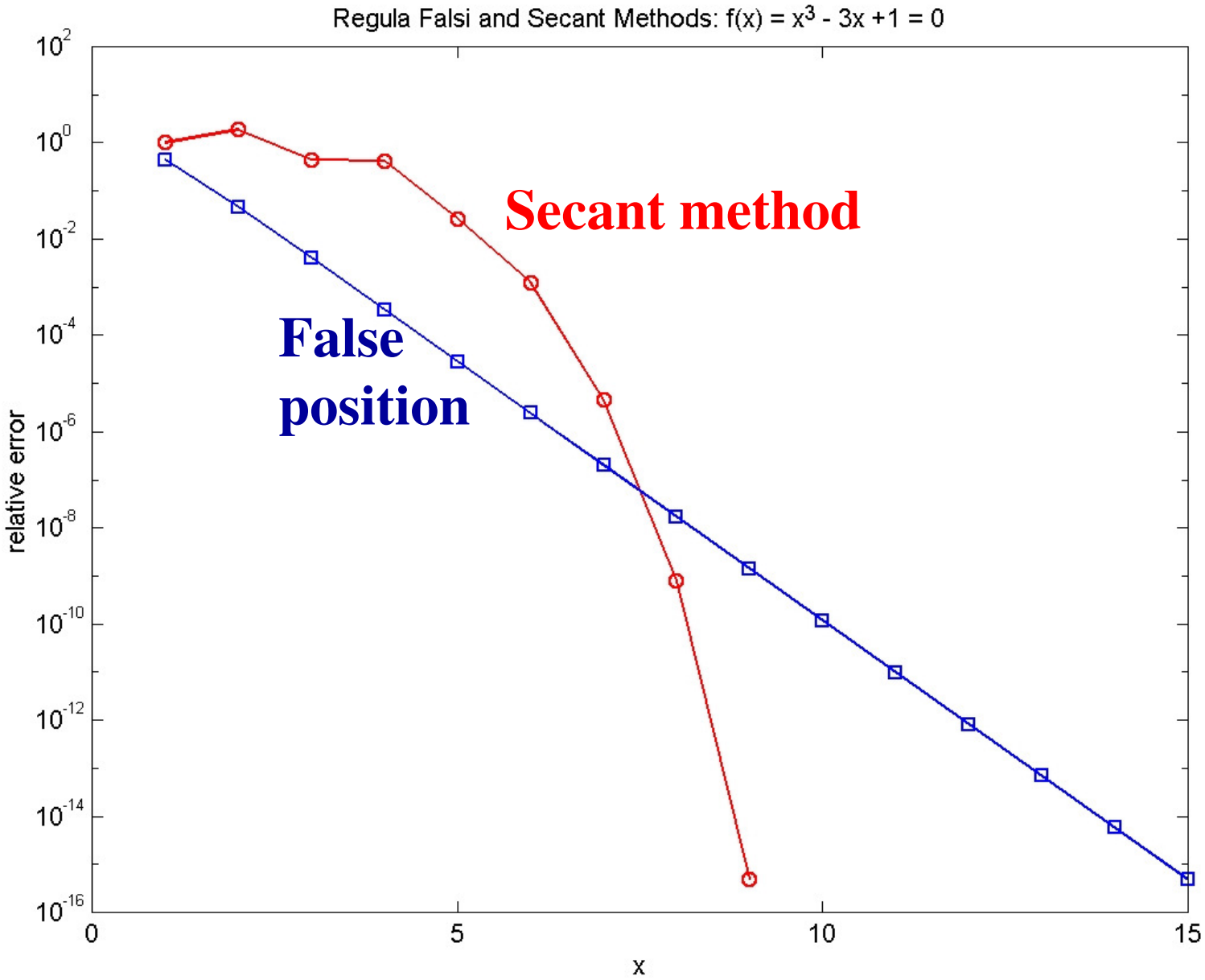
$$f(x) = x^3 - 3x + 1 = 0$$

False position method

```
» [x2 f2]=false_position('my_func',0,1,1.e-15,100);
false_position method has converged
```

step	xl	xu	x	f
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

Secant method may converge even faster and it doesn't need to bracket the root



Secant Method

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

if x_i *and* x_{i-1} are two initial points :

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

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

Secant Method

Assumptions :

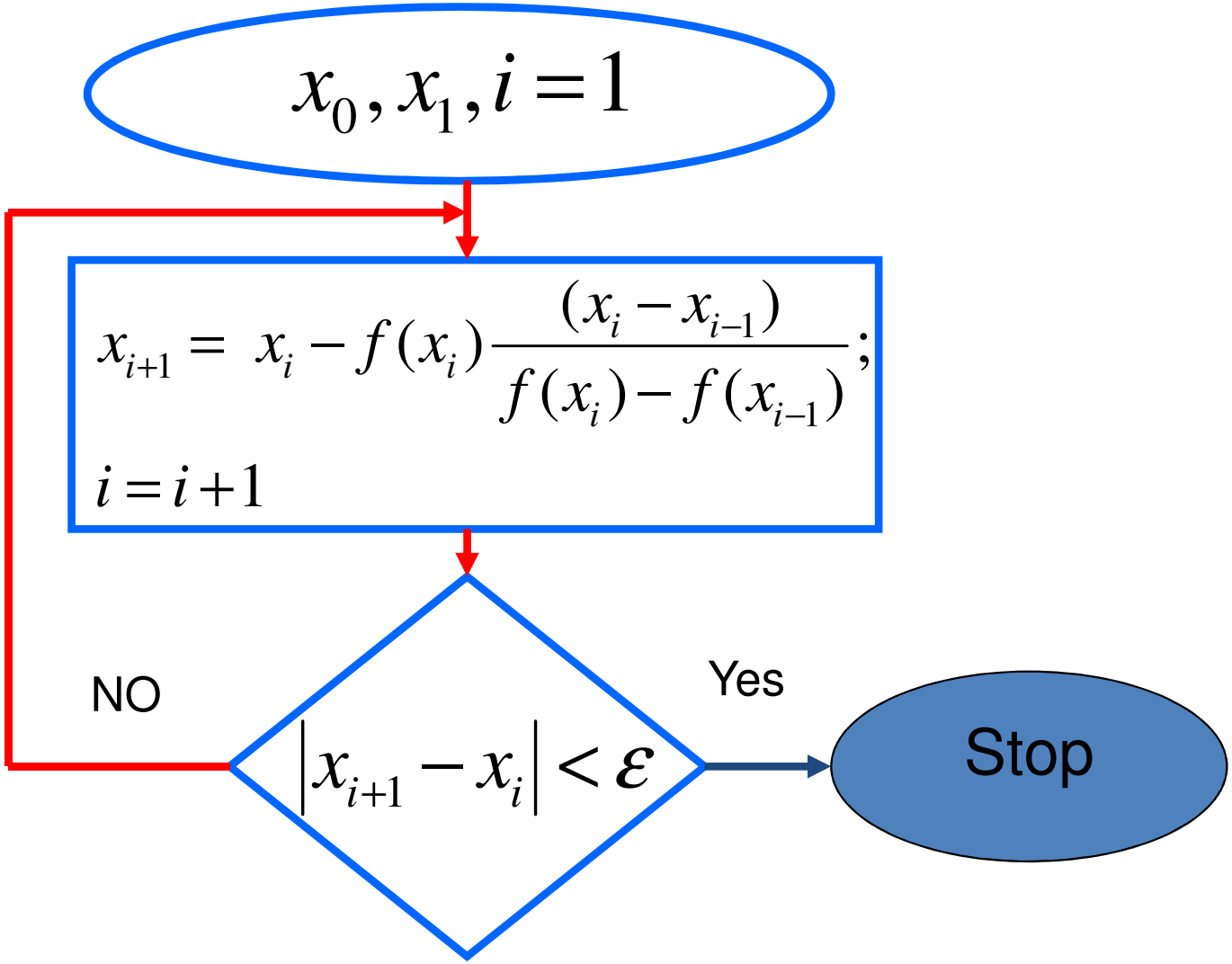
Two initial points x_i *and* x_{i-1}

such that $f(x_i) \neq f(x_{i-1})$

New estimate (Secant Method) :

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

Secant Method - Flowchart



Secant Method

$$f(x) = x^2 - 2x + 0.5$$

$$x_0 = 0$$

$$x_1 = 1$$

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

Example 1

Use Secant method to find the root of :

$$f(x) = x^6 - x - 1$$

Two initial points $x_0 = 1$ *and* $x_1 = 1.5$

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

Solution

i	x_i	$f(x_i)$
0	1.0000	-1.0000
1	1.5000	8.8906
2	1.0506	-0.7062
3	1.0836	-0.4645
4	1.1472	0.1321
5	1.1331	-0.0165
6	1.1347	-0.0005

$$f(x)=x^6-x-1; x_0=1 \; and \; x_1=1.5$$

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

Similarity & Difference b/w **F-P** & **Secant** Methods

False-point method

- Starting two points.
- Similar formula

$x_r = x_u - f(x_u)(x_u - x_l)/(f(x_u) - f(x_l))$

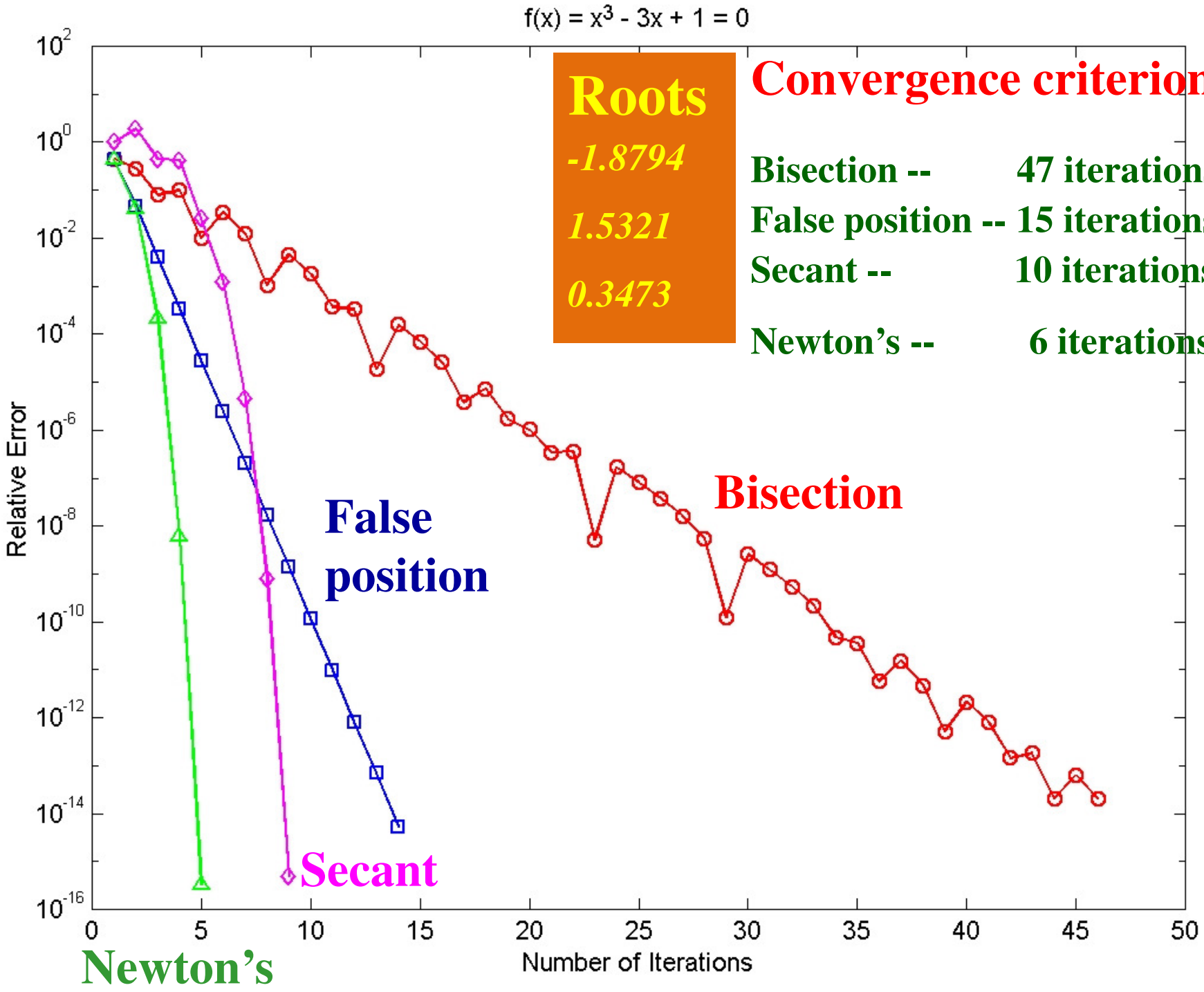
- Next iteration: points replace-ment: if $f(x_r)*f(x_l) < 0$, then $xu = x_r$ else $xl = x_r$.
- Require bracketing
- Always converge

Secant method

- **Starting two points.**
- **Similar formula**

$x_3 = x_2 - f(x_2)(x_2 - x_1)/(f(x_2) - f(x_1))$

- **Next iteration: points replace-ment: always x1 = x2 & x2 =x3.**
- **no requirement of bracketing.**
- **Faster convergence**
- **May not converge**


f(x) = x3 - 3x + 1 = 0
Roots
-1.8794
1.5321
0.3473
Convergence criterion 10 -14
Bisection -- 47 iterations
False position -- 15 iterations
Secant -- 10 iterations
Newton's -- 6 iterations
Bisection
False position
Secant
Newton's
Relative Error
Number of Iterations

Modified Secant Method

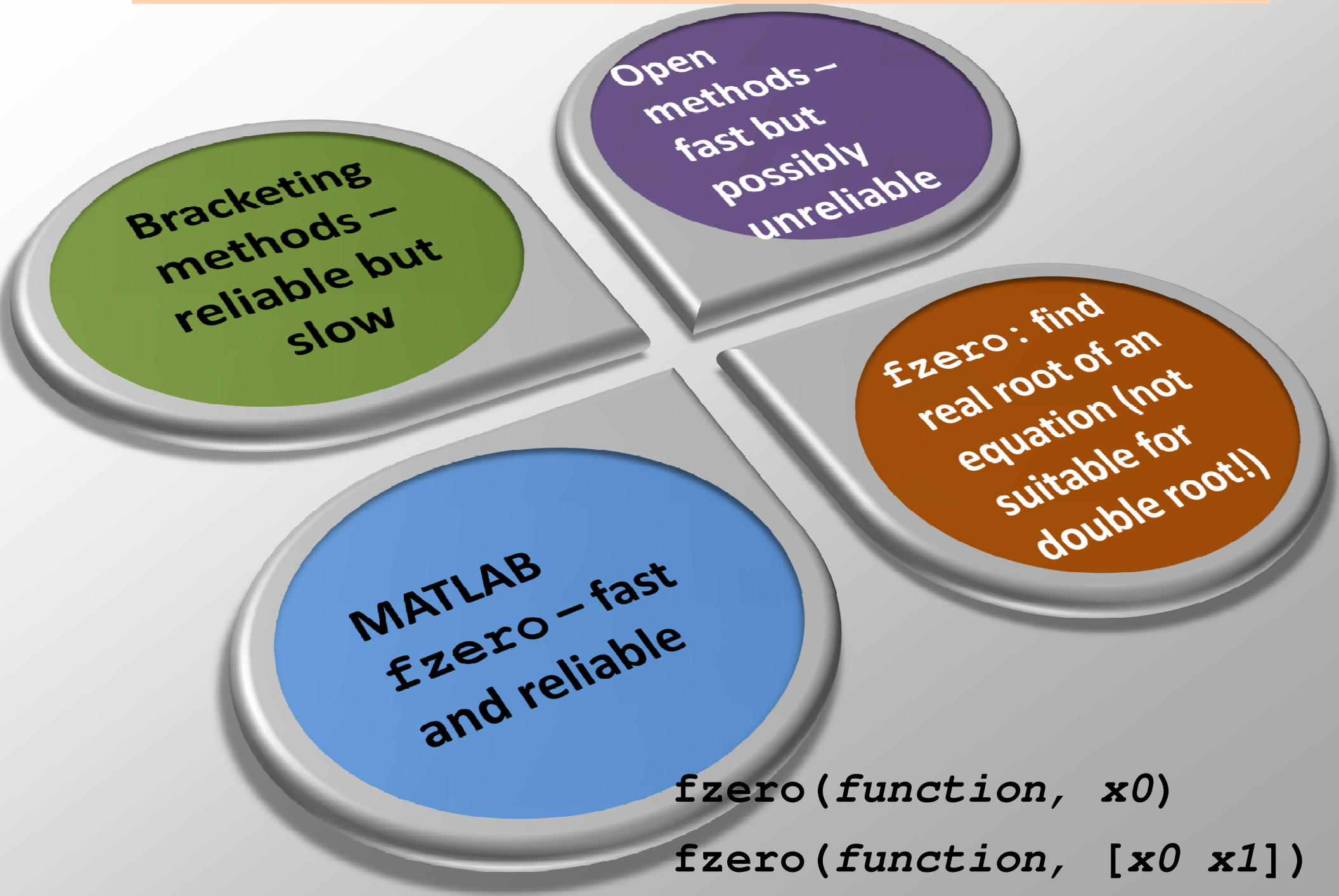
- Use **fractional perturbation** instead of two arbitrary values to estimate the derivative

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

- δ is a small perturbation fraction (e.g., $\delta x_i / x_i = 10^{-6}$)

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

MATLAB Function: `fzero`




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

```
>> root=fzero('multi_func',-10)
root =
   2.99997215011186
>> root=fzero('multi_func',1000)
root =
   2.99996892915965
>> root=fzero('multi_func',[-1000 1000])
root =
   2.99998852581534
>> root=fzero('multi_func',[-2 2])
??? Error using ==> fzero
The function values at the interval endpoints must differ in sign.

>> root=multiple2('multi_func','multi_dfunc','multi_ddfunc');
enter initial guess: xguess = -1
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 100
Newton method has converged
   step          x                    f                    df/dx                d2f/dx2
    1  -1.00000000000000  -256.000000000000000  448.000000000000000 -608.000000000000000
    2   1.54545454545455    -0.915585746130091   -1.468752134417059    5.096919609316331
    3   1.34838709677419    -0.546825709009667   -2.145926990290406    1.190673693397343
    4   1.12513231297383    -0.103193137485462   -1.484223690473826   -8.078669685137726
    5   1.01327476262380    -0.001381869252874   -0.206108293568889  -15.056858096928863
    6   1.00013392759869    -0.000000143464007   -0.002142195919063  -15.990358504281403
    7   1.00000001342083     0.000000000000000   -0.000000214733291  -15.999999033700192
    8   1.00000001342083     0.000000000000000   -0.000000214733291  -15.999999033700192
```

fzero unable to find the double root of

$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$

Roots

3 times 3

2 times 1

fzero and optimset functions

```
>> options=optimset('display','iter');
>> [x fx]=fzero('manning',50,options)
 Func-count        x              f(x)          Procedure
     1              50         -14569.8          initial
     2         48.5858           -14062          search
     3         51.4142         -15078.3          search
     4              48         -13851.9          search
     5              52         -15289.1          search

   ...             ...              ...

    18         27.3726         -6587.13          search
    19         72.6274         -22769.2          search
    20              18          -3457.1          search
    21              82         -26192.5          search
    22         4.74517           319.67          search

   Looking for a zero in the interval [4.7452, 82]

    23         5.67666           110.575          interpolation
    24         6.16719          -5.33648          interpolation
    25         6.14461         0.0804104          interpolation
    26         6.14494      5.51498e-005          interpolation
    27         6.14494     -1.47793e-012          interpolation
    28         6.14494      3.41061e-013          interpolation
    29         6.14494     -5.68434e-013          interpolation
Zero found in the interval: [4.7452, 82].
x =
    6.14494463443476
fx =
     3.41060513648481e-013
```

Search in both directions $x_0 \pm \Delta x$ for sign change

Find sign change after 22 iterations

Switch to secant (linear) or inverse quadratic interpolation to find root

Modified Secant Method

In this modified Secant method, only one initial guess is needed :

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

$$x_{i+1} = x_i - \frac{f(x_i)}{\dfrac{f(x_i + \delta_i x_i) - f(x_i)}{\delta_i x_i}} = \quad x_i - \frac{\delta_i x_i f(x_i)}{f(x_i + \delta_i x_i) - f(x_i)}$$

Problem : How to select δ_i ?

If not selected properly, the method may diverge.

Example 1

Roots

0.8719 + 0.80i(2)

0.3912 +1.35i(2)

-1.1053

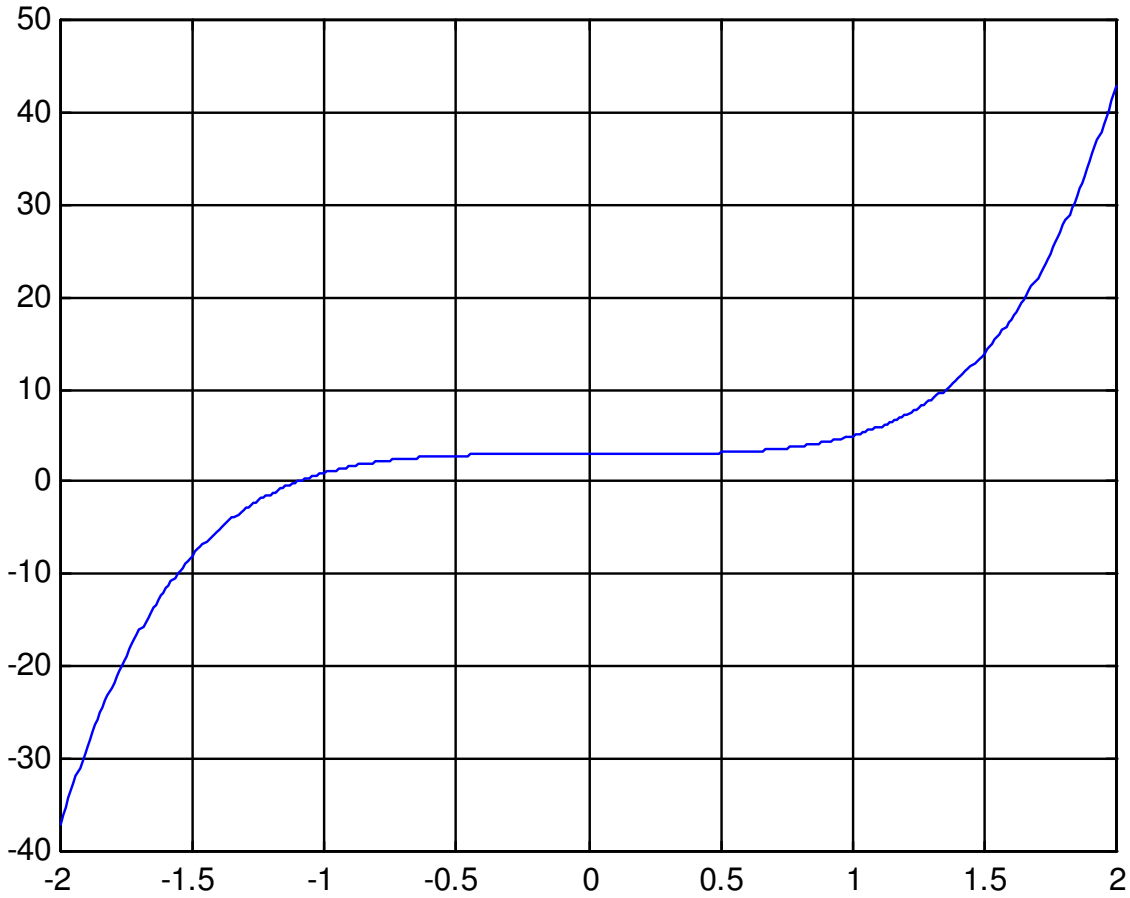
Find the roots of :

$$f(x) = x^5 + x^3 + 3$$

Initial points

$$x_0 = -1 \; and \; x_1 = -1.1$$

$$with \;\; error < 0.001$$



Example 1

x(i)	f(x(i))	x(i+1)	|x(i+1)-x(i)|
-1.0000	1.0000	-1.1000	0.1000
-1.1000	0.0585	-1.1062	0. 0062
-1.1062	0.0102	-1.1052	0.0009
-1.1052	0.0001	-1.1052	0.0000

Convergence Analysis

- The rate of convergence of the Secant method is super linear:

$$\frac{|x_{i+1} - r|}{|x_i - r|^{\alpha}} \leq C, \qquad \alpha \approx 1.62$$

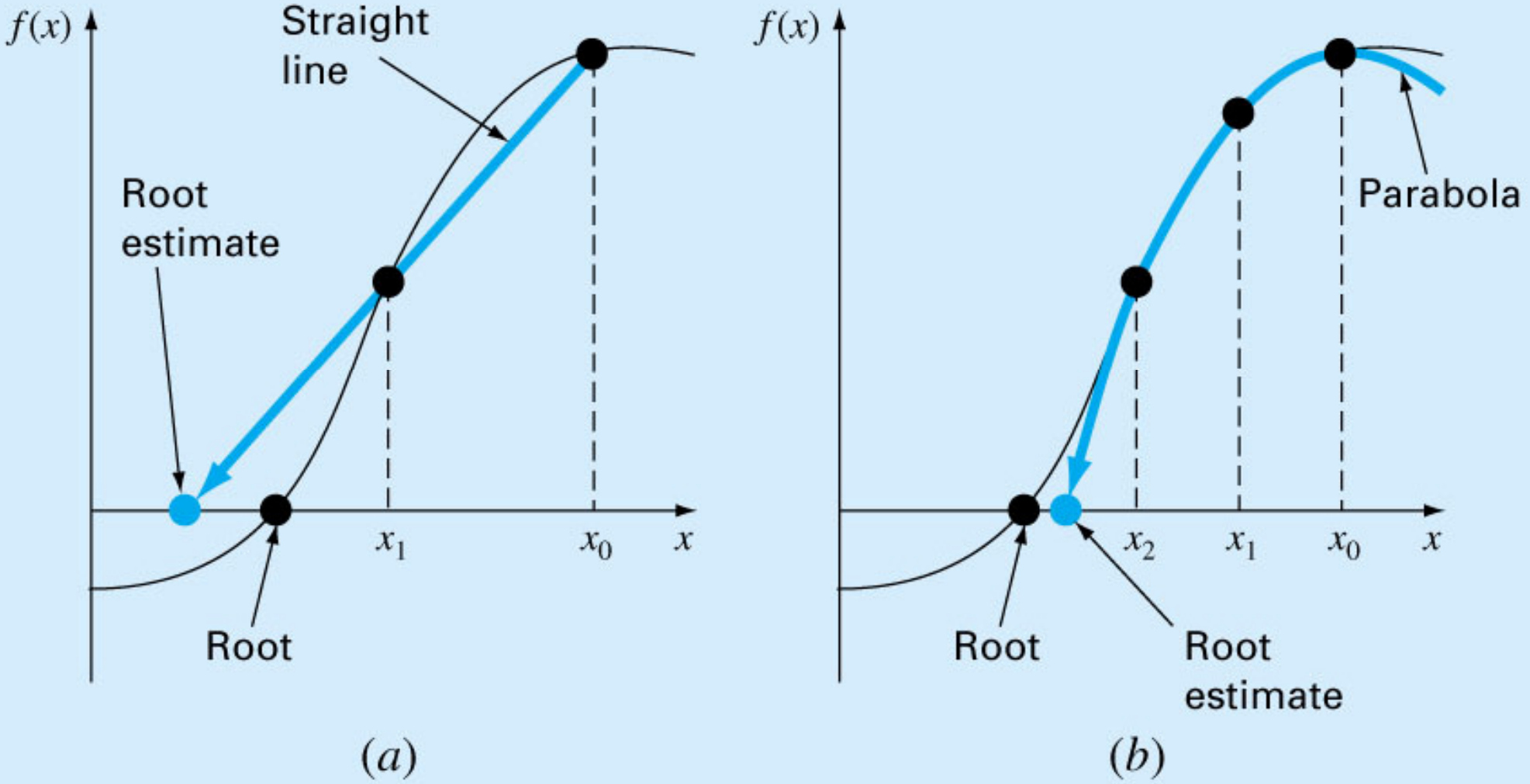
$r : root \quad x_i : \text{estimate of the root at the } i^{\text{th}} \text{ iteration.}$

- It is better than Bisection method but not as good as Newton's method.

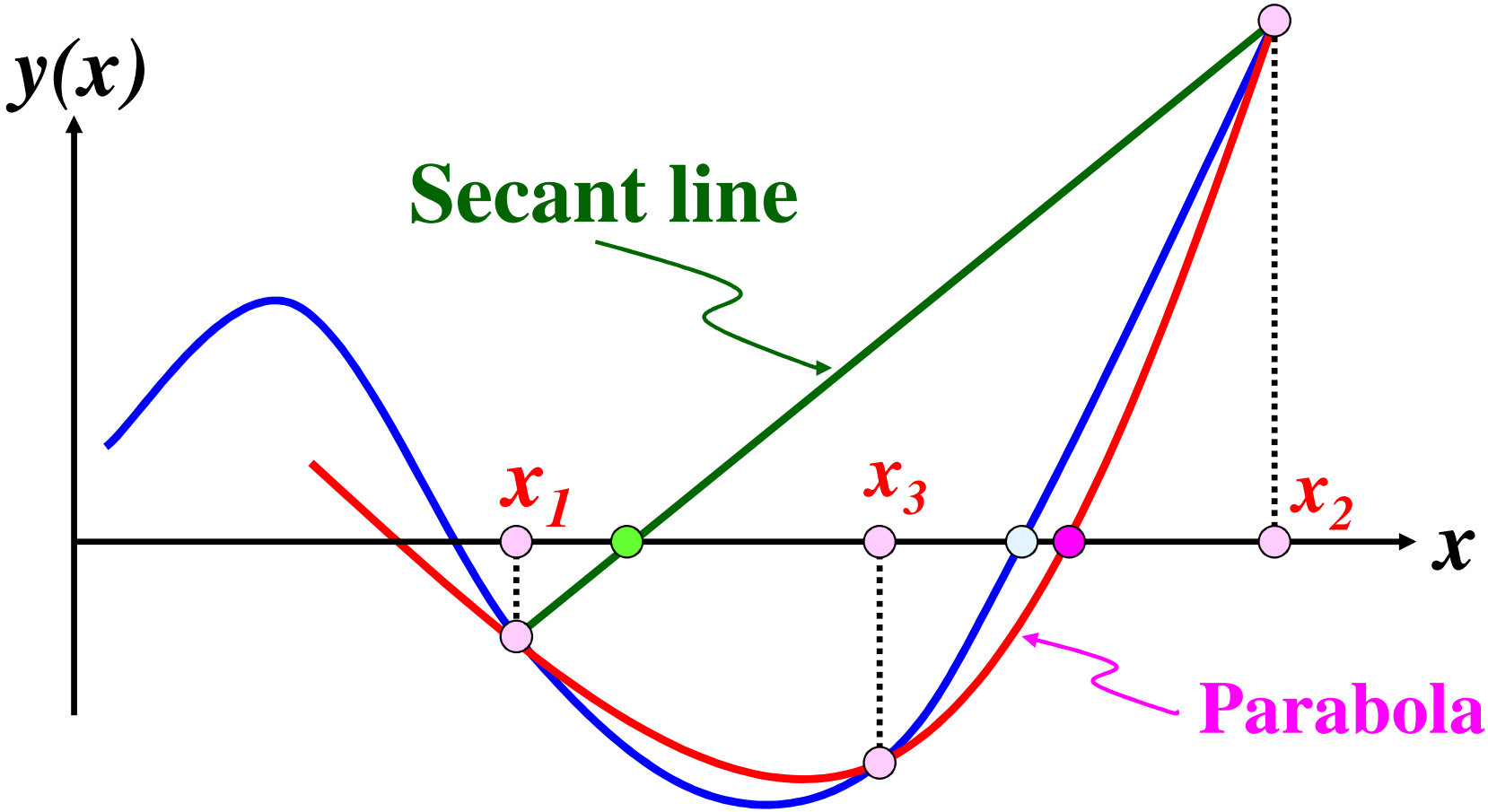
Summary

Method	Pros	Cons
Bisection	- Easy, Reliable, Convergent - One function evaluation per iteration - No knowledge of derivative is needed	- Slow - Needs an interval [a,b] containing the root, i.e., $f(a)f(b)<0$
Newton	- Fast (if near the root) - Two function evaluations per iteration	- May diverge - Needs derivative and an initial guess x_0 such that $f'(x_0)$ is nonzero
Secant	- Fast (slower than Newton) - One function evaluation per iteration - No knowledge of derivative is needed	- May diverge - Needs two initial points guess x_0, x_1 such that $f(x_0)- f(x_1)$ is nonzero

Secant and Muller's Method



Muller's Method



- Fit a parabola (quadratic) to exact curve
- Find both real and complex roots ($x^2 + rx + s = 0$)

MATLAB Function: `roots`

- **Recast the root evaluation task as an eigenvalue problem (Chapter 20)**
- **Zeros of a n^{th}-order polynomial**

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \Lambda + c_2 x^2 + c_1 x + c_0$$

$$coefficient\ vector\ \ c = [c_n, c_{n-1}, \Lambda, c_2, c_1, c_0]$$

```
r = roots(c)
```
- roots

```
c = poly(r)
```
- inverse function

Roots of Polynomial

- Consider the 6th-order polynomial

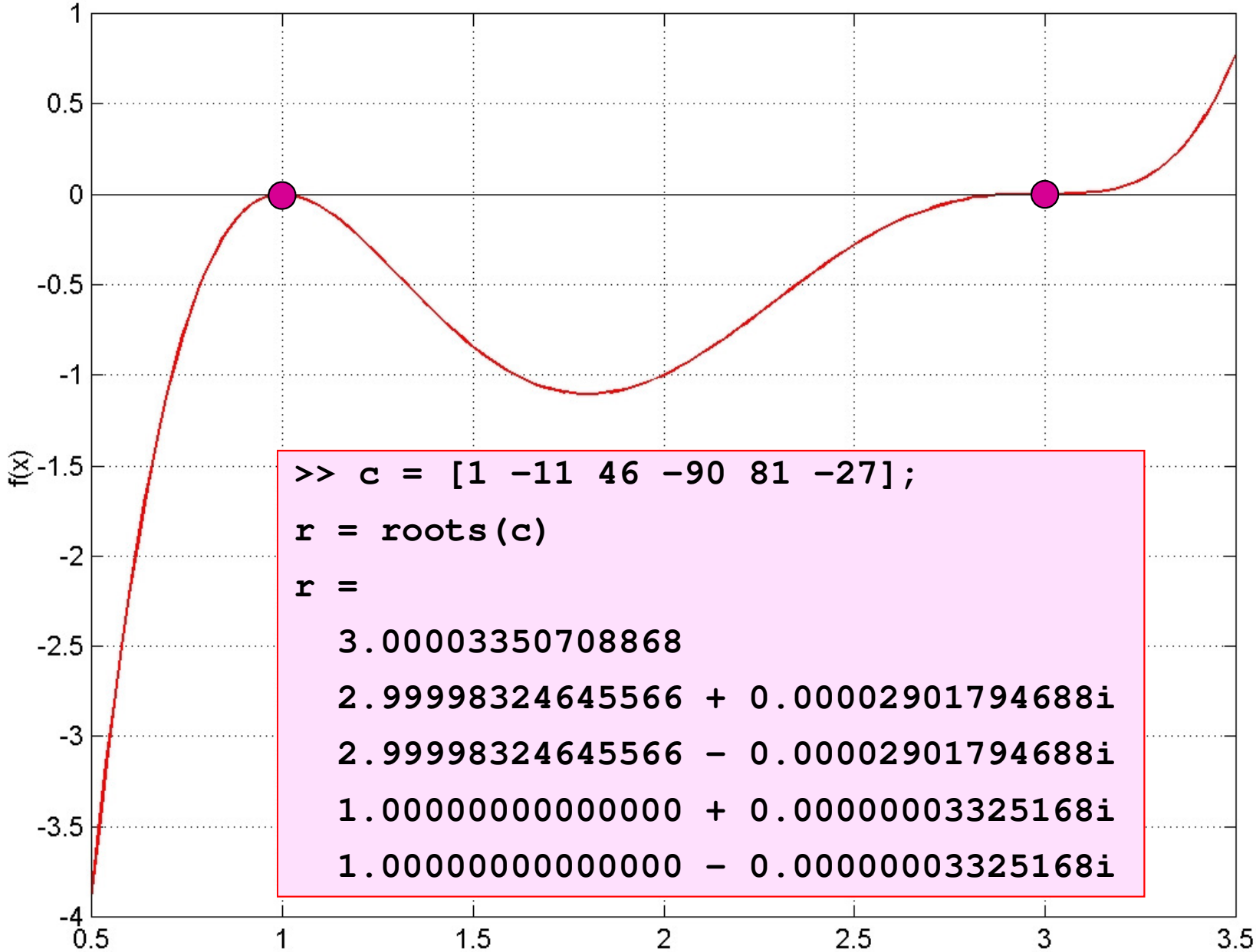
$$f(x) = x^6 - 6x^5 + 14x^4 + 10x^3 - 111x^2 + 56x + 156 = 0$$

$$x_r = -1, -2, 2, 3, 2 \pm 3i$$ **(4 real and 2 complex roots)**

```
>> c = [1 -6 14 10 -111 56 156];
>> r = roots(c)
r =
   2.0000 + 3.0000i
   2.0000 - 3.0000i
   3.0000
   2.0000
  -2.0000
  -1.0000
>> polyval(c, r), format long g
ans =
      1.3642420526593e-012 + 4.50484094471904e-012i
      1.3642420526593e-012 - 4.50484094471904e-012i
     -1.30739863379858e-012
       1.4210854715202e-013
       7.105427357601e-013
       5.6843418860808e-014
```

Verify $f(x_r) = 0$

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = (x-1)^2(x-3)^3$$




```
>> c = [1 -11 46 -90 81 -27];
r = roots(c)
r =
   3.00003350708868
   2.99998324645566 + 0.00002901794688i
   2.99998324645566 - 0.00002901794688i
   1.00000000000000 + 0.00000003325168i
   1.00000000000000 - 0.00000003325168i
```


Thank You!

Roots of Polynomials

- The roots of polynomials such as

$$f_n(x) = a_o + a_1 x + a_2 x^2 + \mathrm{K} + a_n x^n$$

Follow these rules:

1. For an *n*th order equation, there are n real or complex roots.
2. If *n* is odd, there is at least one real root.
3. If complex root exist in conjugate pairs (that is, $\lambda+\mu i$ and $\lambda-\mu i$), where $i=sqrt(-1)$.

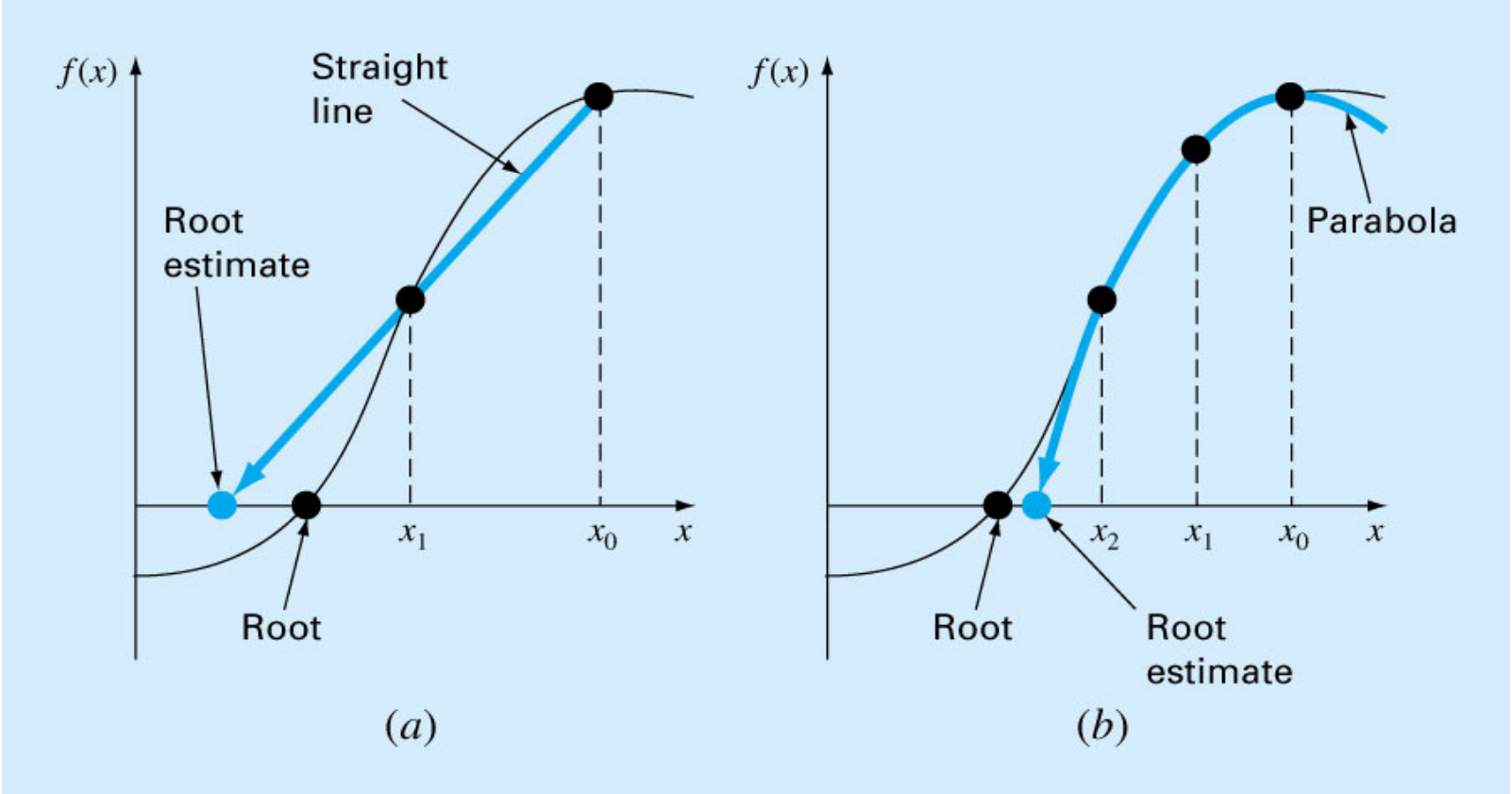
Conventional Methods

- The efficacy of bracketing and open methods depends on whether the problem being solved involves complex roots. If only real roots exist, these methods could be used. However,
  - Finding good initial guesses complicates both the open and bracketing methods, also the open methods could be susceptible to divergence.
- Special methods have been developed to find the real and complex roots of polynomials – Müller and Bairstow methods.

Müller Method

- Müller's method obtains a root estimate by projecting a parabola to the x axis through three function values.

Figure 7.3



Müller Method

- The method consists of deriving the coefficients of parabola that goes through the three points:

1. Write the equation in a convenient form:

$$f_2(x) = a(x - x_2)^2 + b(x - x_2) + c$$

2. The parabola should intersect the three points $[x_o, f(x_o)]$, $[x_1, f(x_1)]$, $[x_2, f(x_2)]$. The coefficients of the polynomial can be estimated by substituting three points to give

$$f(x_o) = a(x_o - x_2)^2 + b(x_o - x_2) + c$$
$$f(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$$
$$f(x_2) = a(x_2 - x_2)^2 + b(x_2 - x_2) + c$$

3. These equations can be solved for three unknowns, *a, b, c*. Since two of the terms in the 3rd equation are zero, it can be immediately solved for $c=f(x_2)$.

$$f(x_o) - f(x_2) = a(x_o - x_2)^2 + b(x_o - x_2)$$
$$f(x_1) - f(x_2) = a(x_1 - x_2)^2 + b(x_1 - x_2)$$

If

$h_o = x_1 - x_o \qquad h_1 = x_2 - x_1$

$$\delta_o = \frac{f(x_1) - f(x_o)}{x_1 - x_o} \qquad \delta_1 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$(h_o + h_1)b - (h_o + h_1)^2 a = h_o\delta_o + h_1\delta_1$$

$$h_1 b - h_1^2 a = h_1\delta_1$$

Solved for a and *b*

$$a = \frac{\delta_1 - \delta_o}{h_1 + h_o} \qquad b = ah_1 + \delta_1 \qquad c = f(x_2)$$

- Roots can be found by applying an alternative form of quadratic formula:

$$x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

- The error can be calculated as

$$\varepsilon_a = \left| \frac{x_3 - x_2}{x_3} \right| 100\%$$

- ±term yields two roots, the sign is chosen to agree with *b*. This will result in a largest denominator, and will give root estimate that is closest to x_2.

- Once x_3 is determined, the process is repeated using the following guidelines:
  1. If only real roots are being located, choose the two original points that are nearest the new root estimate, x_3.
  2. If both real and complex roots are estimated, employ a sequential approach just like in secant method, x_1, x_2, and x_3 to replace x_o, x_1, and x_2.