

# NUMERICAL METHODS

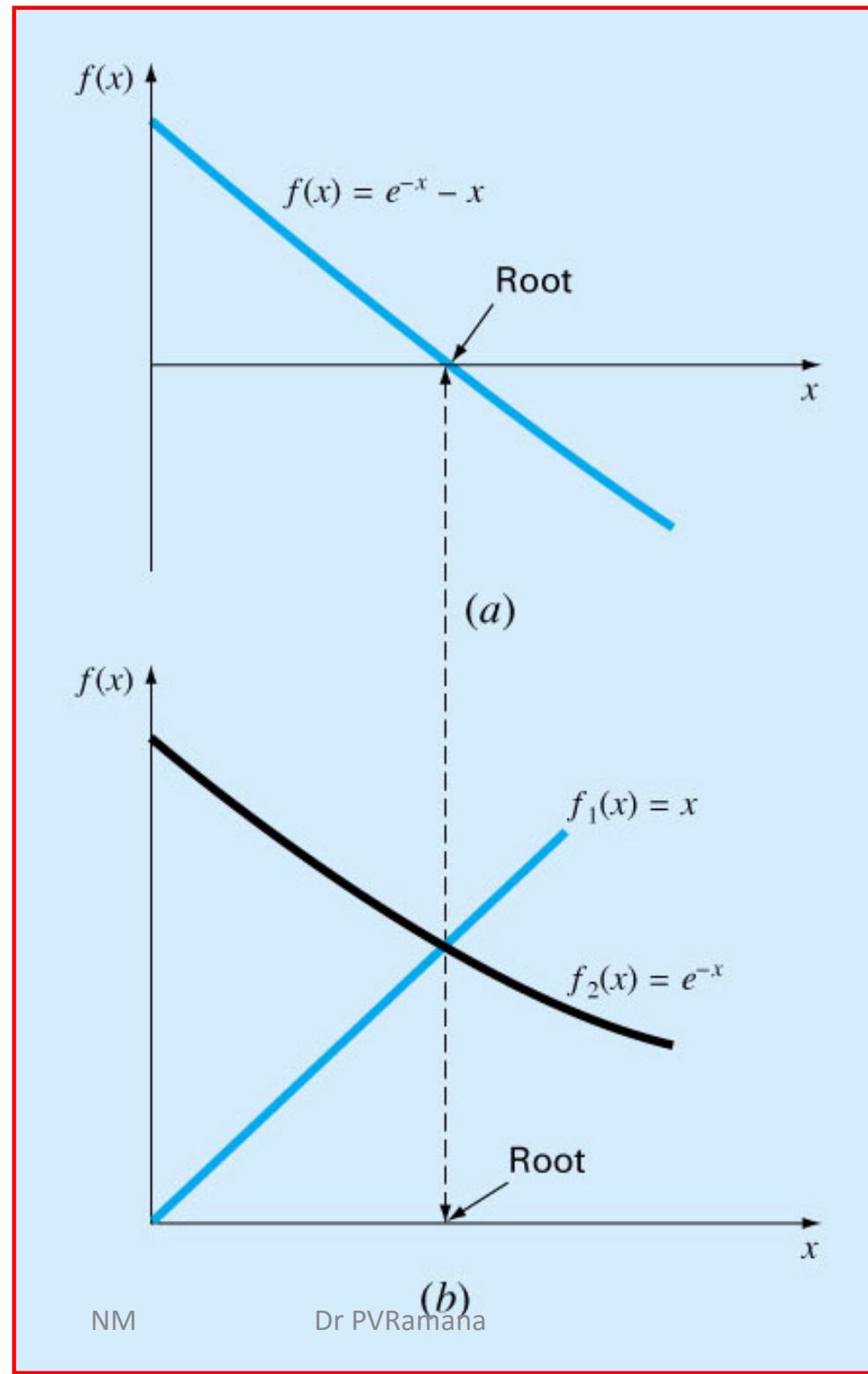
## Lecture 2

Dr. P V Ramana

# Simple Fixed- Point Iteration

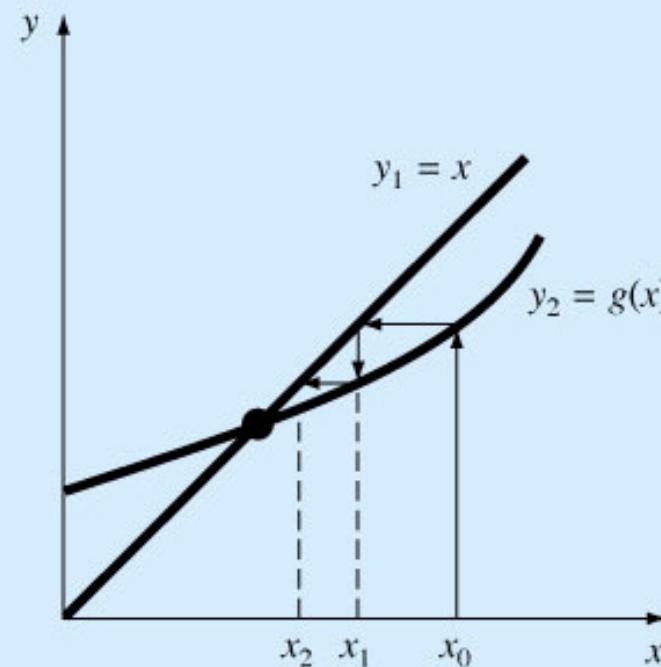
Two Alternative  
Graphical Methods

$$f(x) = f_1(x) - f_2(x) = 0$$

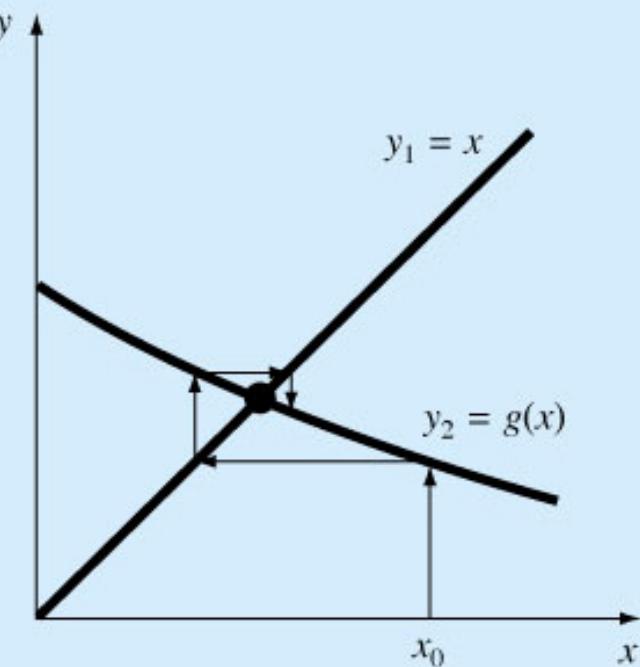


# Fixed-Point Iteration

Convergent

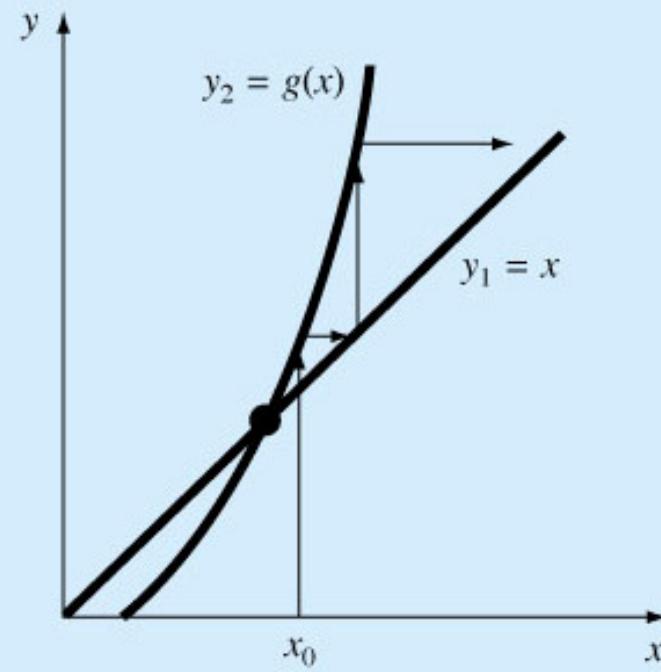


(a)



(b)

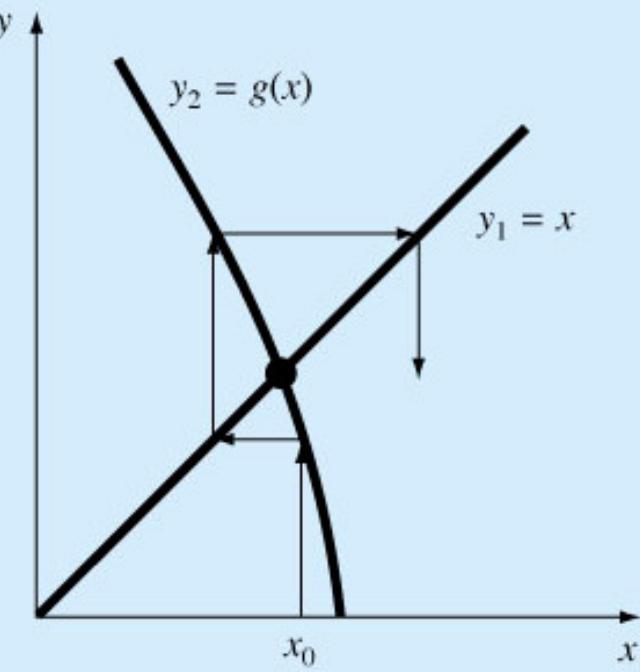
Divergent



NM

(c)

Dr PV Ramana



(d)

# *Newton's Method*



**King of  
the root-  
finding  
methods**

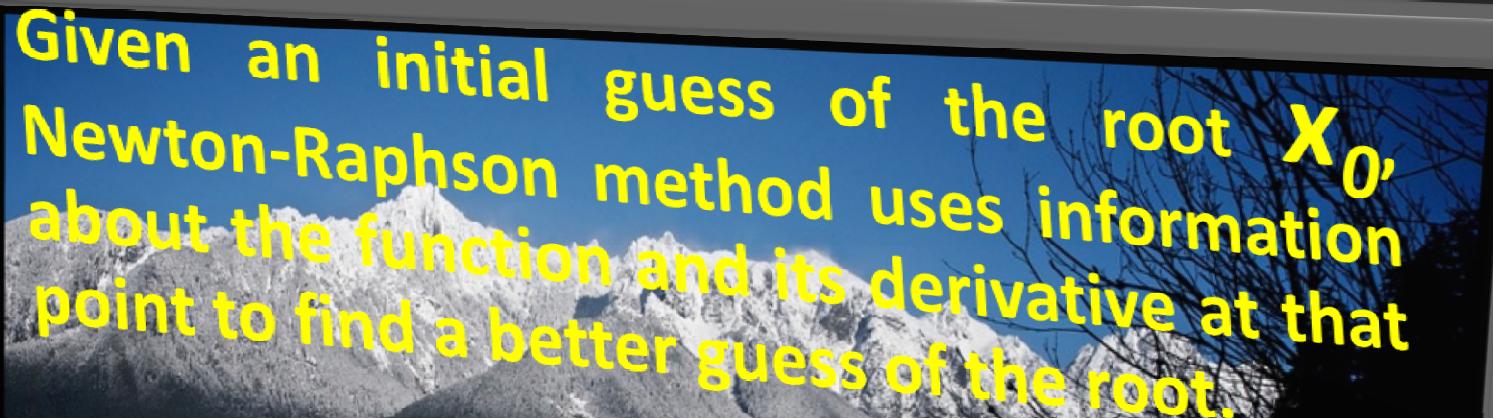
**Newton-  
Raphson  
method**

**Based on  
Taylor  
series  
expansion**

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(\xi)}{2!}(x_{i+1} - x_i)^2 + \Lambda$$

# Newton-Raphson Method

(Also known as Newton's Method)



Given an initial guess of the root  $x_0$ , Newton-Raphson method uses information about the function and its derivative at that point to find a better guess of the root.

## Assumptions:

$f(x)$  is continuous and the first derivative is known

An initial guess  $x_0$  such that  $f'(x_0) \neq 0$  is given

# Derivation of Newton's Method

*Given* :  $x_i$  an initial guess of the root of  $f(x) = 0$

*Question* : How do obtain a better estimate  $x_{i+1}$ ?

---

---

---

---

Taylor Therorem :  $f(x + h) \approx f(x) + f'(x)h$

Find  $h$  such that  $f(x + h) = 0$ .

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

Newton – Raphson Formula

$$\text{A new guess of the root : } x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

# *Newton-Raphson Method*

Truncate the Taylor series to get

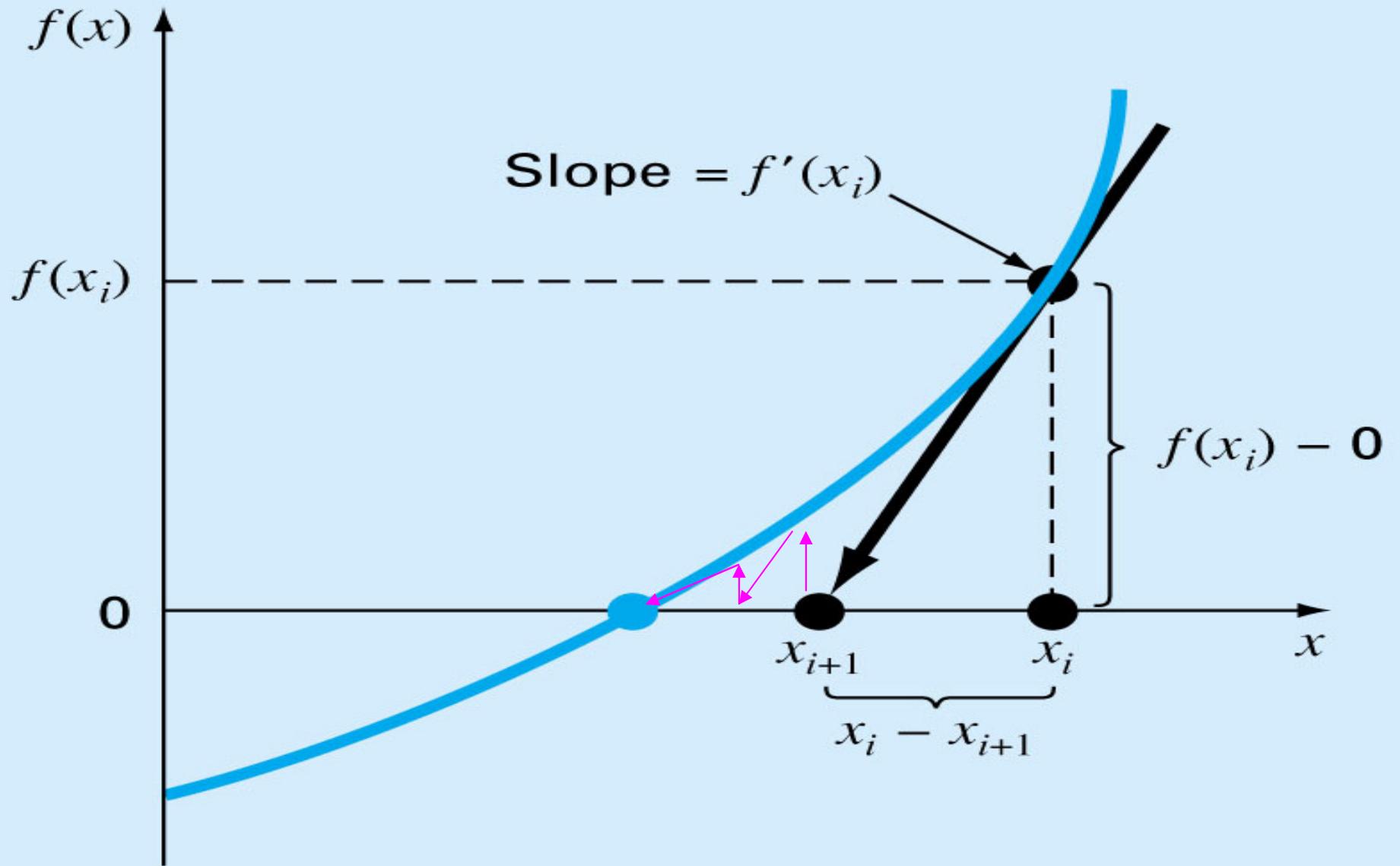
$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

At the root,  $f(x_{i+1}) = 0$ , so

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

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

# Newton-Raphson Method



## ***Newton's Method***

Note that an evaluation of the derivative (slope) is required

You may have to do this numerically

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

**Open Method – Convergence depends on the initial guess (not guaranteed)**

However, Newton method can converge very quickly (quadratic convergence)

# Newton's Method

Given  $f(x)$ ,  $f'(x)$ ,  $x_0$

Assumption  $f'(x_0) \neq 0$

---

for  $i = 0:n$

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

end

C FORTRAN PROGRAM

$$F(X) = X^{**3} - 3*X^{**2} + 1$$

$$FP(X) = 3*X^{**2} - 6*X$$

$$X = 4$$

DO 10 I=1,5

$$X = X - F(X)/FP(X)$$

PRINT \*, X

10 CONTINUE

STOP

END

# Newton's Method

Given  $f(x)$ ,  $f'(x)$ ,  $x_0$

Assumption  $f'(x_0) \neq 0$

for  $i = 0 : n$

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

end

F.m

```
function [F]=F(X)
F=X^3-3*X^2+1
```

FP.m

```
function [FP]=FP(X)
FP=3*X^2-6*X
```

% MATLAB PROGRAM

$X = 4$

for  $i = 1 : 5$

$X = X - F(X) / FP(X)$

end

# Example

Find a zero of the function  $f(x) = x^3 - 2x^2 + x - 3$ ,  $x_0 = 4$

$$f'(x) = 3x^2 - 4x + 1$$

Iteration 1 :  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 4 - \frac{33}{33} = 3$

Iteration 2 :  $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 3 - \frac{9}{16} = 2.4375$

Iteration 3 :  $x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 2.4375 - \frac{2.0369}{9.0742} = 2.2130$

# Example

k (Iteration)	$x_k$	$f(x_k)$	$f'(x_k)$	$x_{k+1}$	$ x_{k+1} - x_k $
0	4	33	33	3	1
1	3	9	16	2.4375	0.5625
2	2.4375	2.0369	9.0742	2.2130	0.2245
3	2.2130	0.2564	6.8404	2.1756	0.0384
4	2.1756	0.0065	6.4969	2.1746	0.0010

Iteration 1 :  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 4 - \frac{33}{33} = 3$

Iteration 2 :  $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 3 - \frac{9}{16} = 2.4375$

Iteration 3 :  $x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 2.4375 - \frac{2.0369}{9.0742} = 2.2130$

$$f(x) = x^3 - 2x^2 + x - 3$$

$$f'(x) = 3x^2 - 4x + 1$$

# Convergence Analysis

Theorem :

Let  $f(x)$ ,  $f'(x)$  and  $f''(x)$  be continuous at  $x \approx r$  where  $f(r) = 0$ . If  $f'(r) \neq 0$  then there exists  $\delta > 0$

such that  $|x_0 - r| \leq \delta \Rightarrow \frac{|x_{k+1} - r|}{|x_k - r|^2} \leq C$

$$C = \frac{1}{2} \frac{\max_{|x_0 - r| \leq \delta} |f''(x)|}{\min_{|x_0 - r| \leq \delta} |f'(x)|}$$

A nighttime photograph of the Seattle skyline, featuring the Space Needle and various illuminated buildings.

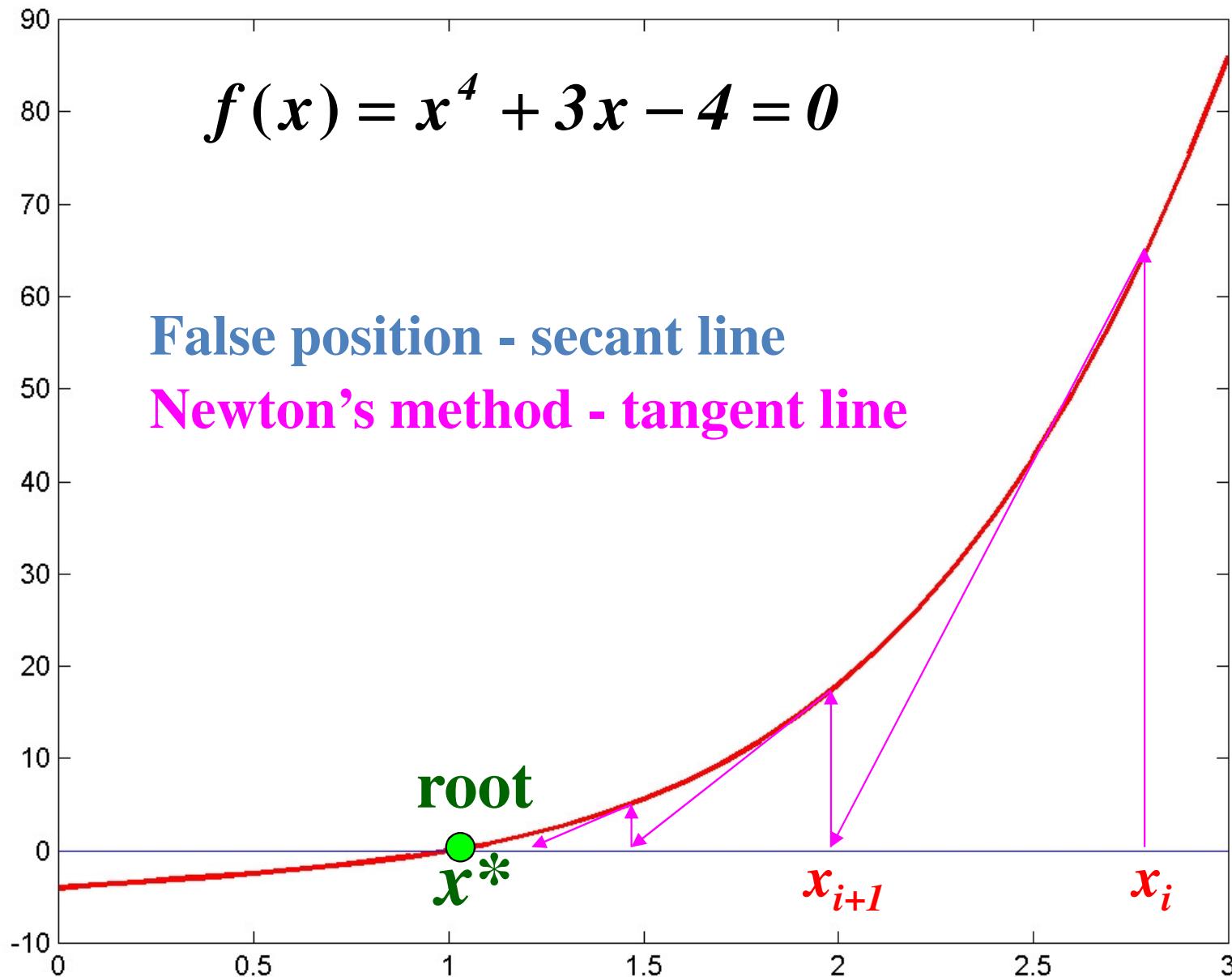
# Convergence Analysis

## Remarks

When the guess is close enough to a simple root of the function then Newton's method is guaranteed to converge quadratically.

Quadratic convergence means that the number of correct digits is nearly doubled at each iteration.

# *Newton-Raphson Method*





**Step 1: Start at the point  $(x_1, f(x_1))$ .**



**Step 2 : The intersection of the tangent of  $f(x)$  at this point and the x-axis.**

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



**Step 3: Examine if  $f(x_2) = 0$  or  $\text{abs}(x_2 - x_1) < \text{tolerance}$ ,**



**Newton Raphson  
Method**

**Step 4: If yes, solution  $x_r = x_2$   
If not,  $x_1 = x_2$ , repeat the  
iteration.**

# Example 1

Use Newton's Method to find a root of :

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

Use the initial point :  $x_0 = 1$ .

Stop after three iterations , or

if  $|x_{k+1} - x_k| < 0.001$  , or

if  $|f(x_k)| < 0.0001$  .

# Five Iterations of the Solution

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

$$f'(x) = 3x^2 - 1$$

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

•	k	x <sub>k</sub>	f(x <sub>k</sub> )	f'(x <sub>k</sub> )	ERROR	
						if  x <sub>k+1</sub> - x <sub>k</sub>   < 0.001, or
						if  f(x <sub>k</sub> )  < 0.0001
•	0	1.0000	-1.0000	2.0000	-	
•	1	1.5000	0.8750	5.7500	0.1522	
•	2	1.3478	0.1007	4.4499	0.0226	
•	3	1.3252	0.0021	4.2685	0.0005	
•	4	1.3247	0.0000	4.2646	0.0000	
•	5	1.3247	0.0000	4.2646	0.0000	

## Example 2

Use Newton's Method to find a root of :

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

Use the initial point :  $x_0 = 1$ .

Stop after three iterations , or

if  $|x_{k+1} - x_k| < 0.001$  , or

if  $|f(x_k)| < 0.0001$  .

## Example 2

Use Newton's Method to find a root of :

$$f(x) = e^{-x} - x, \quad f'(x) = -e^{-x} - 1, \text{ with } x_0 = 1.$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$x_k$	$f(x_k)$	$f'(x_k)$	$\frac{f(x_k)}{f'(x_k)}$
1.0000	- 0.6321	- 1.3679	0.4621
0.5379	0.0461	- 1.5840	- 0.0291
0.5670	0.0002	- 1.5672	- 0.0002
0.5671	0.0000	- 1.5671	- 0.0000

# Example 3

In estimating the root of:  $x - \cos(x) = 0$ , to get more than 13 correct digits:

- 4 iterations of Newton ( $x_0=0.8$ )
- 43 iterations of Bisection method (initial interval [0.6, 0.8])
- 5 iterations of Secant method  
( $x_0=0.6, x_1=0.8$ )

# Example 3

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$f(x) = x - \cos(x)$$

$$f'(x) = \sin(x) + 1$$

Estimates of the root of:  $x - \cos(x) = 0$ .

0.600000000000000

Initial guess

0.74401731944598

1 correct digit

0.73909047688624

4 correct digits

0.73908513322147

10 correct digits

0.73908513321516

14 correct digits

# Example : 4 Slow Convergence

Find the positive roots of :

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

**N-R Formula :**

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9} \quad \text{use } x_0 = 0.5$$

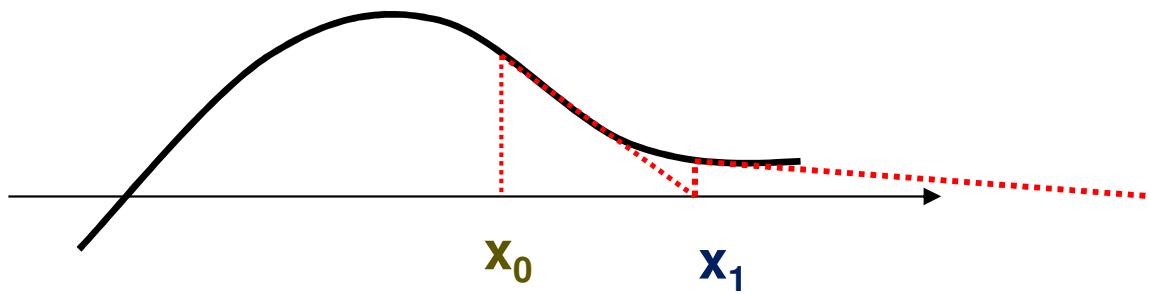
Iteration	x
0	0.5
1	51.65
2	46.485
3	41.8365
4	37.65285
5	33.887565
.	
.	
38	1.083
99999999 99999999	1.0000000

# Problems with Newton's Method

- If the initial guess of the root is far from the root the method may not converge.
- Newton's method converges linearly near multiple zeros {  $f(r) = f'(r) = 0$  }. In such a case, modified algorithms can be used to regain the quadratic convergence.

# Problems with Newton's Method

## - Runaway -



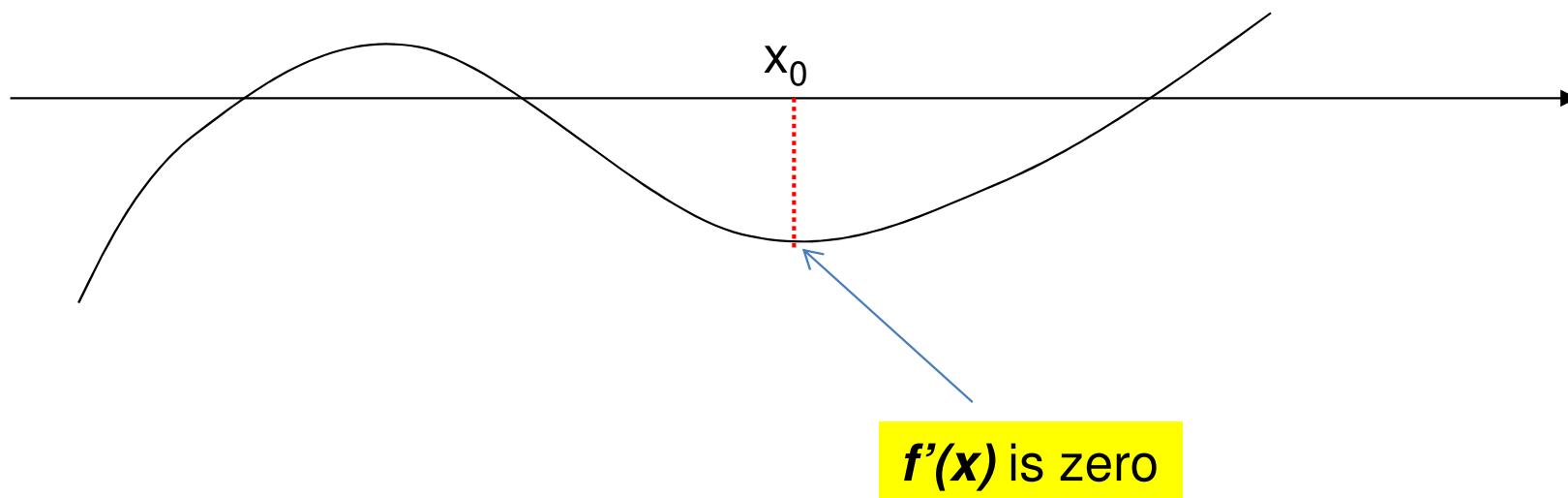
The estimates of the root is going away from the root.

# Problems with Newton's Method

## - Flat Spot -

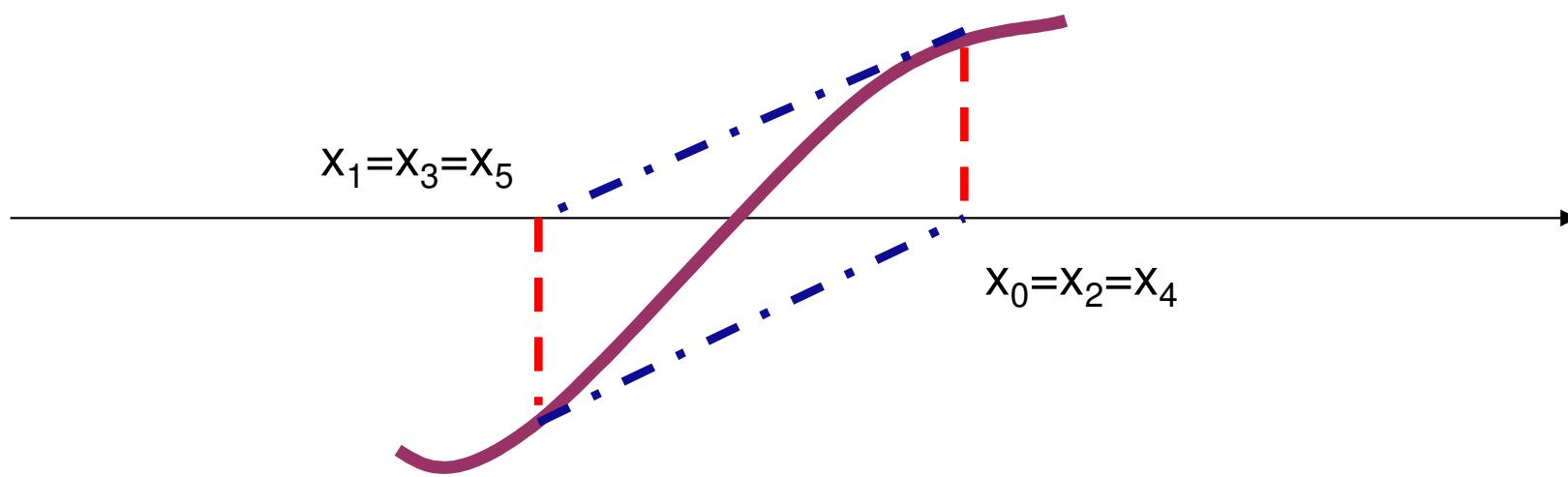
The value of  $f'(x)$  is zero, the algorithm fails.

If  $f'(x)$  is very small then  $x_1$  will be very far from  $x_0$ .

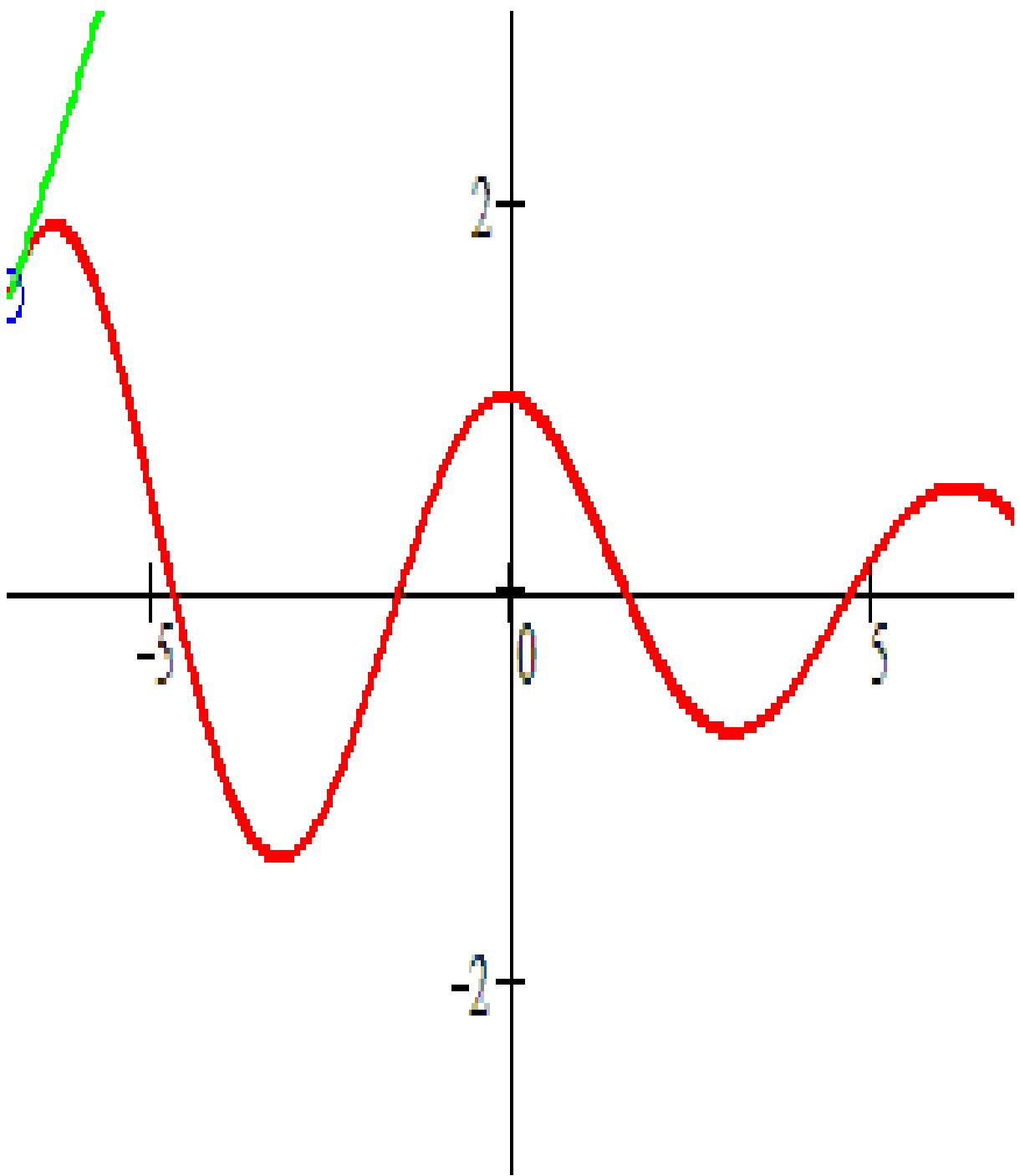


# Problems with Newton's Method

## - Cycle -



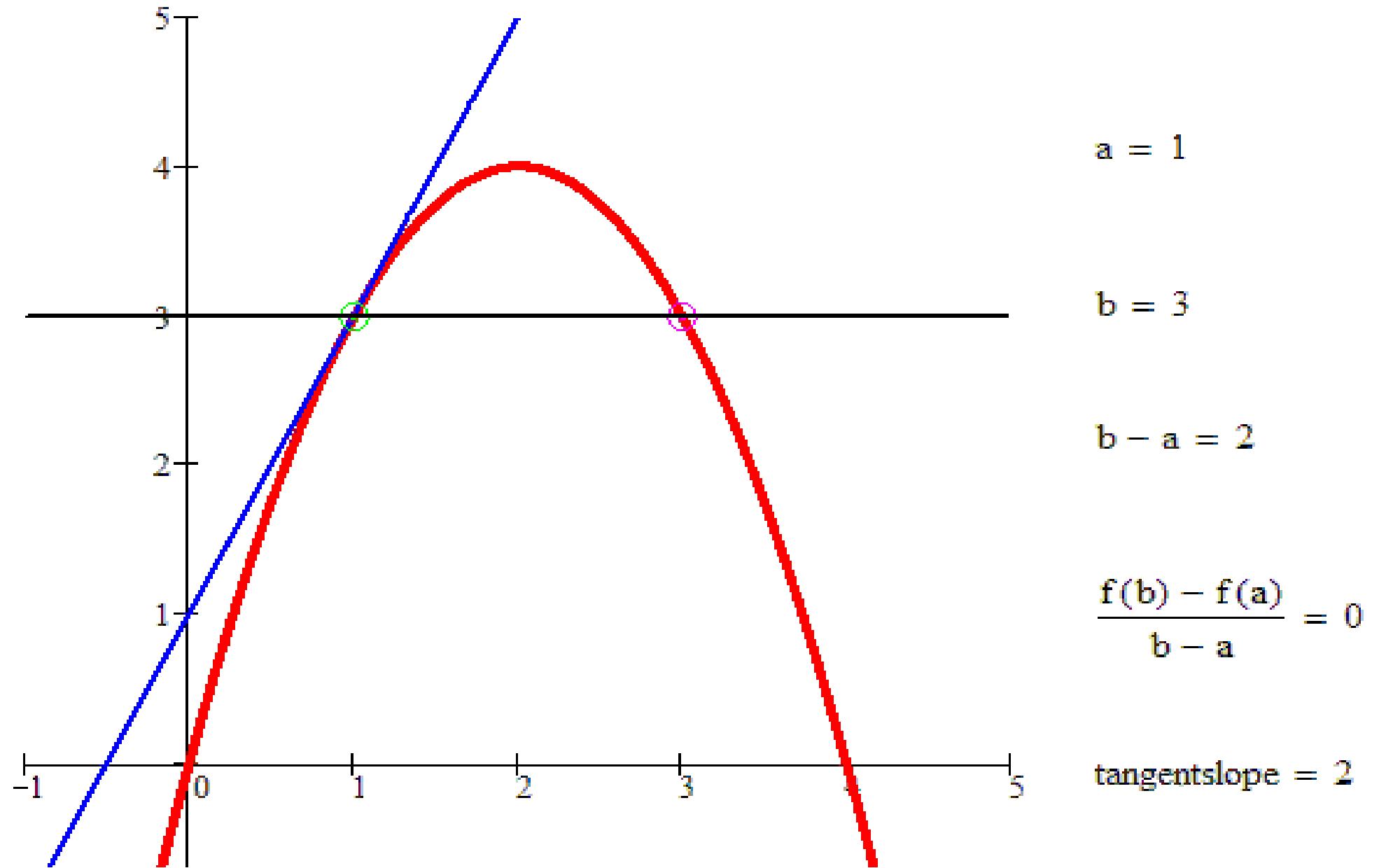
The algorithm cycles between two values  $x_0$  and  $x_1$



When the slope of the tangent line is positive, the function is increasing. When the slope of the tangent line is negative, the function is decreasing.

$$m = 1.171$$

A **secant line** to a function  $f$  goes through the points  $(a, f(a))$  and  $(b, f(b))$ . If we fix  $a$  and let  $b$  approach  $a$ , the slopes of the secant lines will approach the slope of the **tangent line** at  $a$ , if the limit exists.



# Systems of Nonlinear Equations

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

$$f_1(x_1, x_2, x_3, \dots, x_n) = 0$$

$$f_2(x_1, x_2, x_3, \dots, x_n) = 0$$

M

$$f_n(x_1, x_2, x_3, \dots, x_n) = 0$$

Example :

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

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

# 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

$$\begin{aligned}f(x) &= f(x_k) + (x - x_k)f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(\eta) \\f(x^*) = 0 &= f(x_k) + (x^* - x_k)f'(x_k) + \frac{1}{2}(x^* - x_k)^2 f''(\eta) \\ \Rightarrow x^* &= x_k - \frac{f(x_k)}{f'(x_k)} - (x^* - x_k)^2 \frac{f''(\eta)}{2f'(x_k)} \\ &\quad \textcolor{red}{x_{k+1}}\end{aligned}$$

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

Quadratic convergence

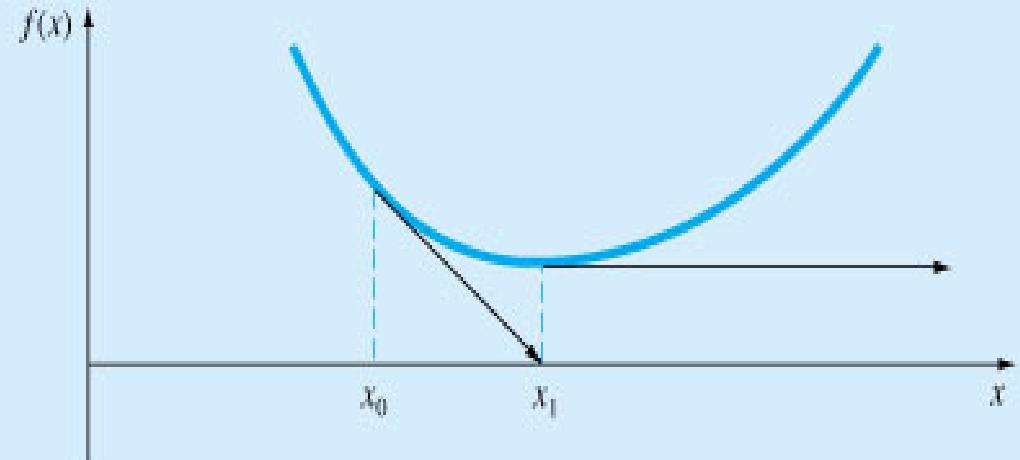
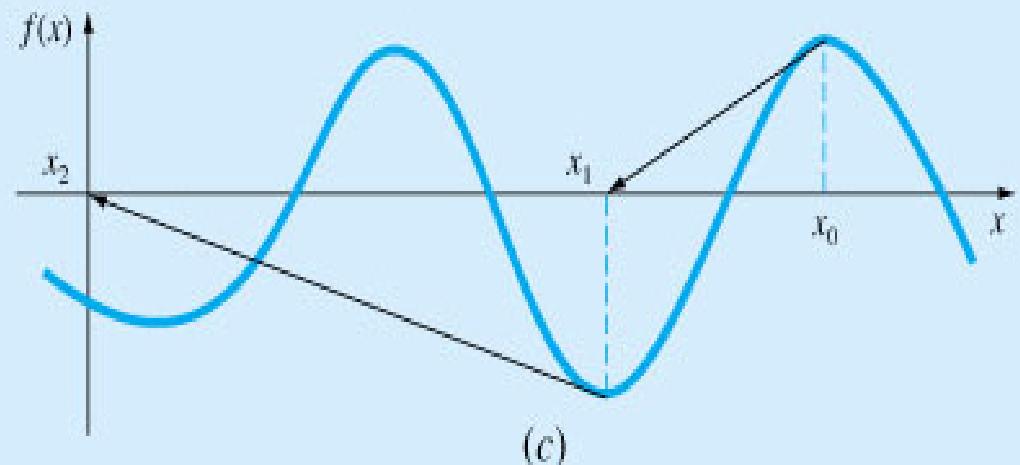
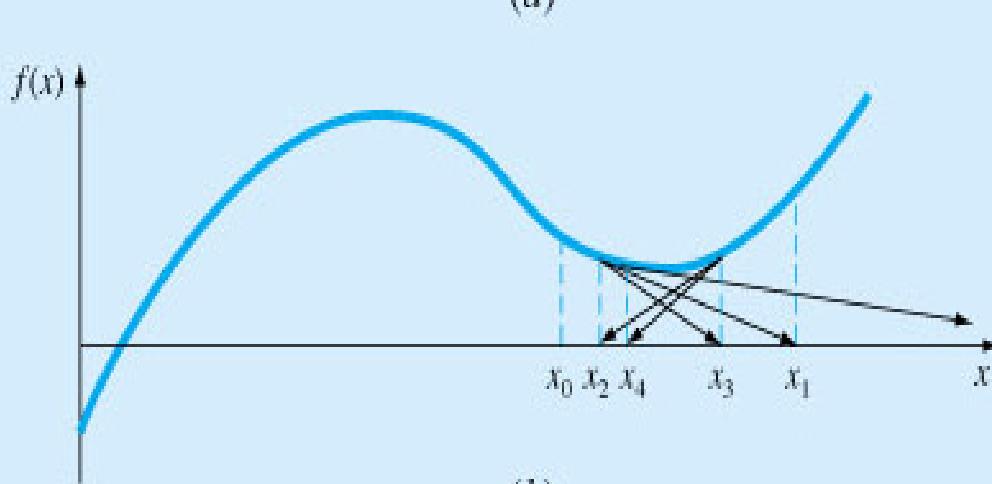
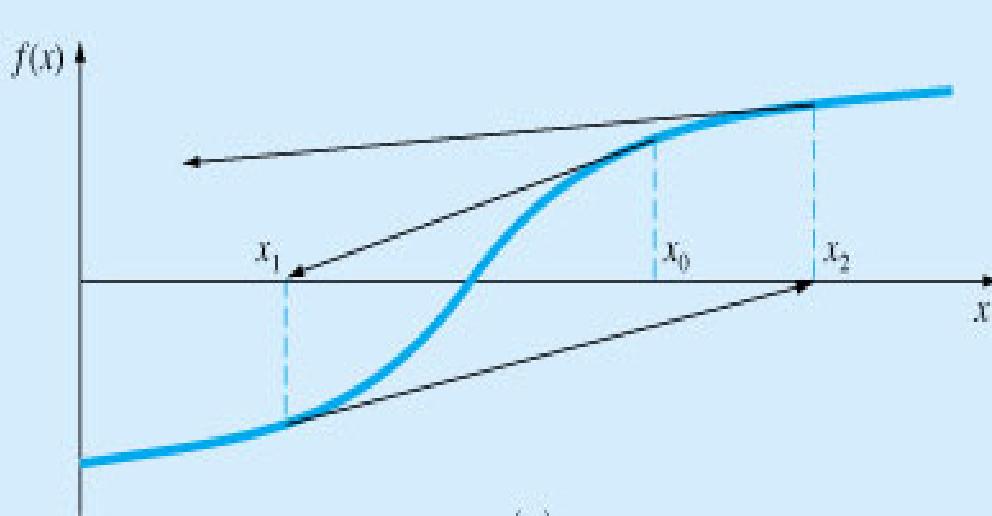
# *Newton-Raphson Method*

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

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

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

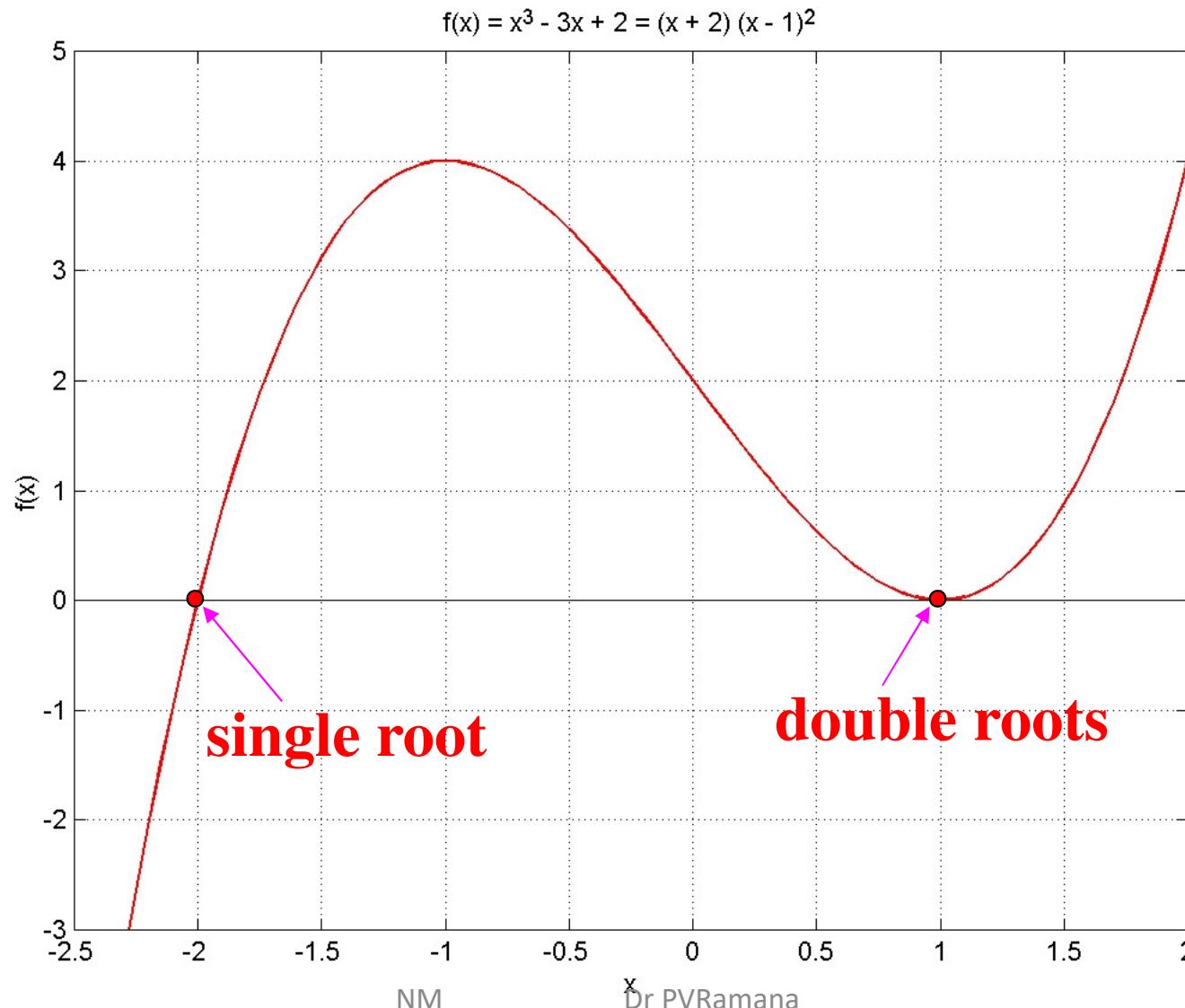
Given  $c_d = 0.25 \text{ kg/m}$ ,  $v = 36 \text{ m/s}$ ,  $t = 4 \text{ s}$ , and  $g = 9.81 \text{ m}^2/\text{s}$ , determine the mass of the bungee jumper

# 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*

- 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

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

## Problems with multiple roots

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

*Multiple Roots*

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

$f'(x)$  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*

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

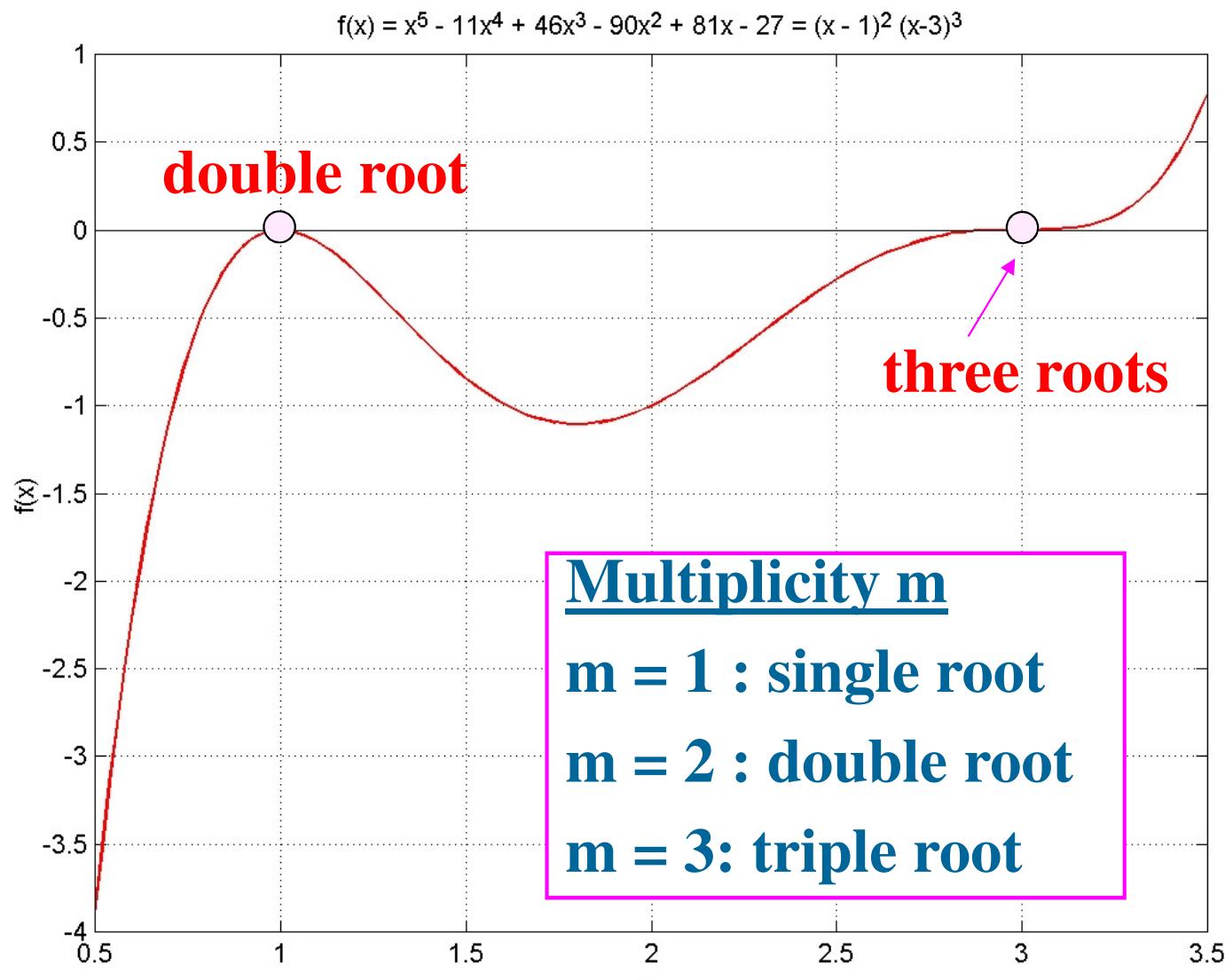
- When the multiplicity of the root is known

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

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

# *Multiple Root with Multiplicity m*

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



# 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	1.300000000000000	-0.442170000000004
2	1.096000000000000	-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	1.300000000000000	-0.442170000000004
2	0.891999999999999	-0.109259530656779
3	0.99229251101321	-0.000480758689392
4	0.99995587111371	-0.000000015579900
5	0.9999999853944	-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 $u = f/f'$**

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

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

- 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]; Dr PV Ramana
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	-27.00000000000000	81.00000000000000	-180.00000000000000
2	1.28571428571429	-0.411257214255940	-2.159100374843831	-0.839650145772595
3	1.08000000000002	-0.045298483200014	-1.061683200000175	-10.6905599999999067
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	-15.99999997123849
7	1.00000000031772	0.000000000000000	0.000000005083592	-15.99999997123849

# *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	27783.0000000000000000	18081.0000000000000000
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	-0.00000000006168	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	NM 0.000000000000000	0.000000011891075

# *Modified Newton's method*

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

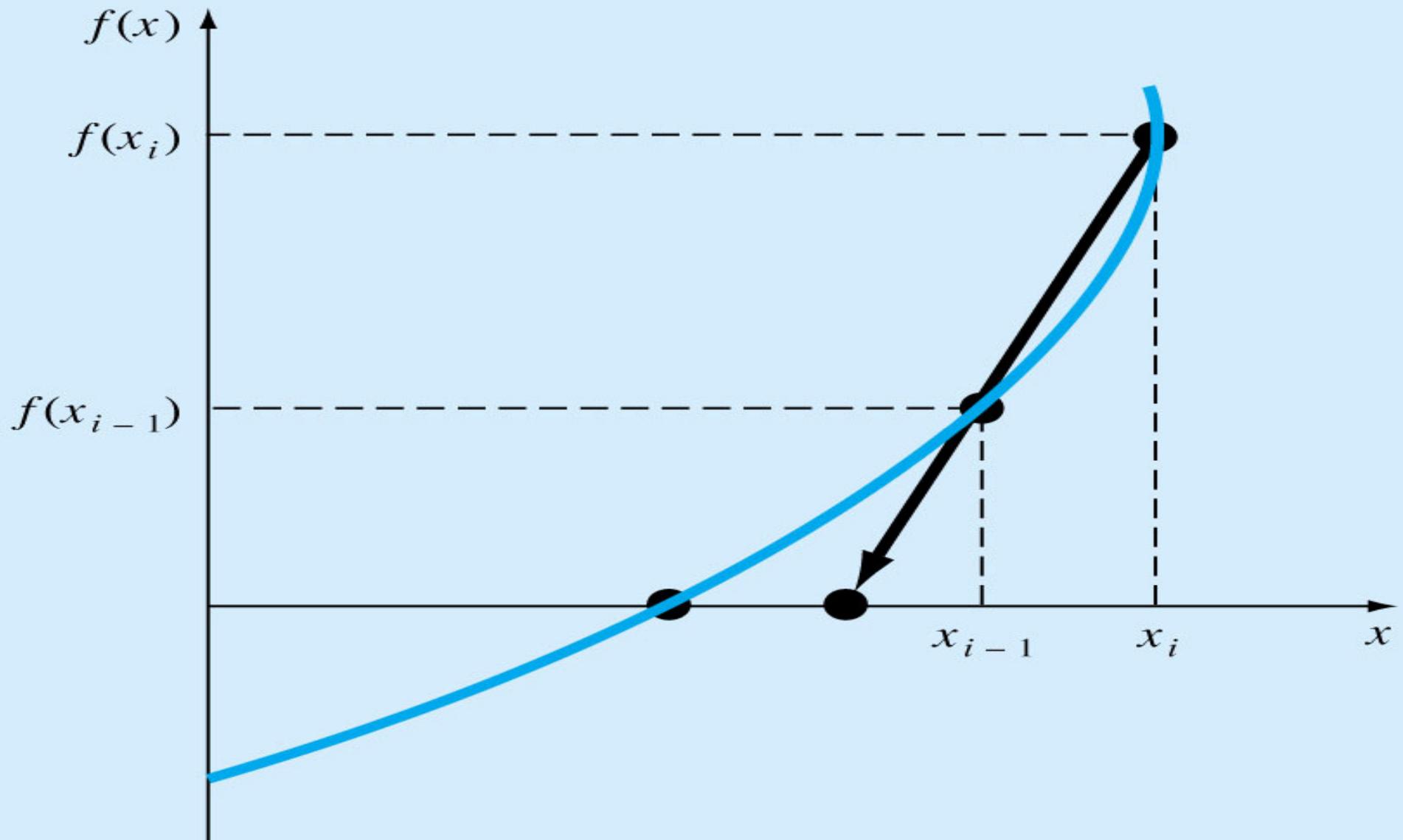
```
>> [x,f] = multiple2('multi_func','multi_dfunc','multi_ddfunc');  
enter initial guess: xguess = 10  
allowable tolerance: es = 1.e-6  
maximum number of iterations: maxit = 100  
Newton method has converged
```

**Triple root at x = 3**

step	x	f	df/dx	d2f/dx2
1	10.00000000000000	27783.0000000000000000	18081.0000000000000000	9380.0000000000000000
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*

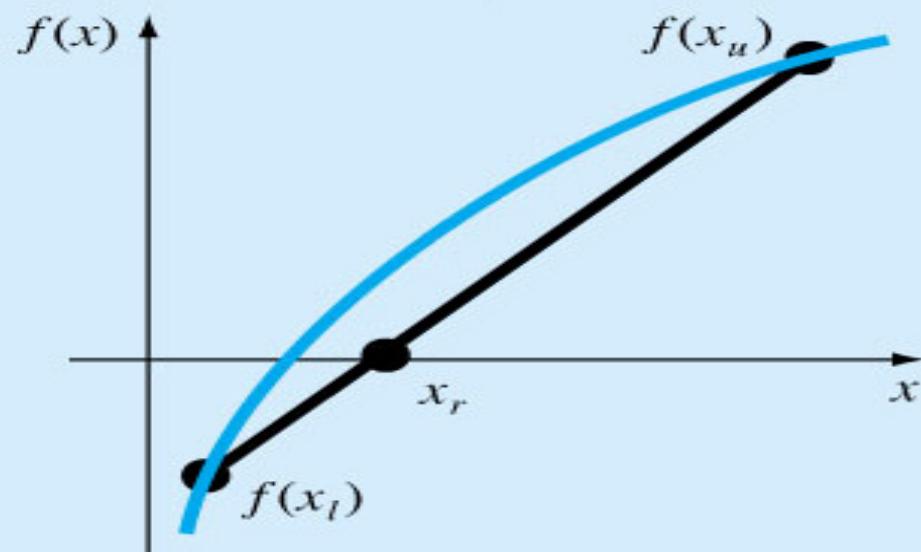
- The formula for the secant method is

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

- 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

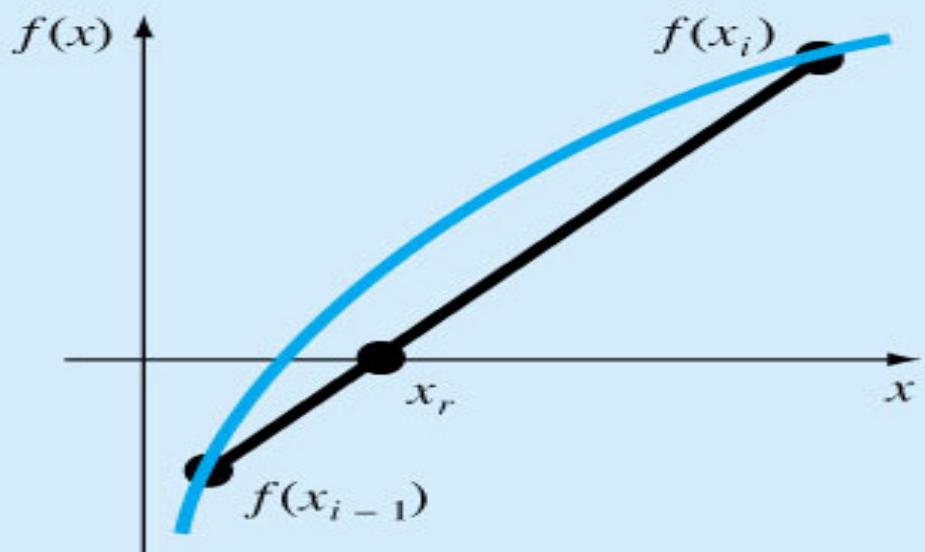
# False-Position and Secant Methods

False position

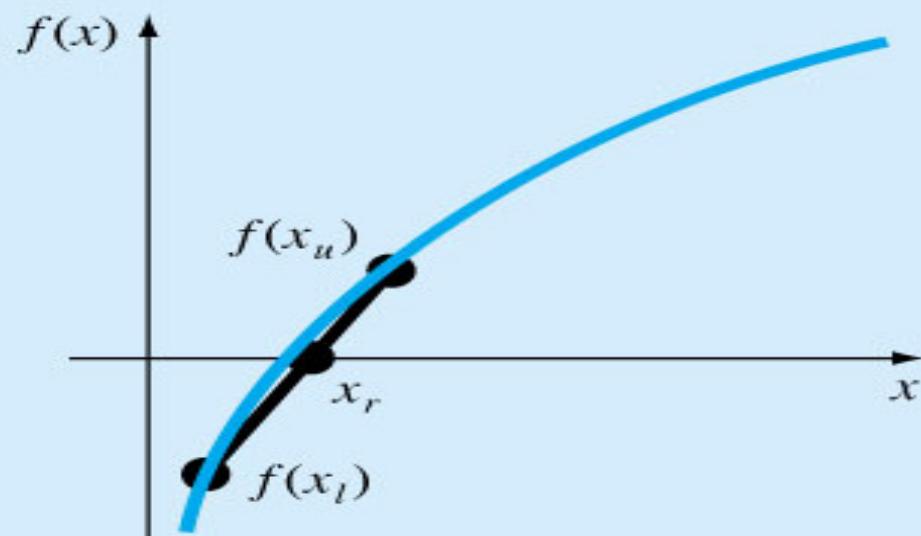


(a)

Secant

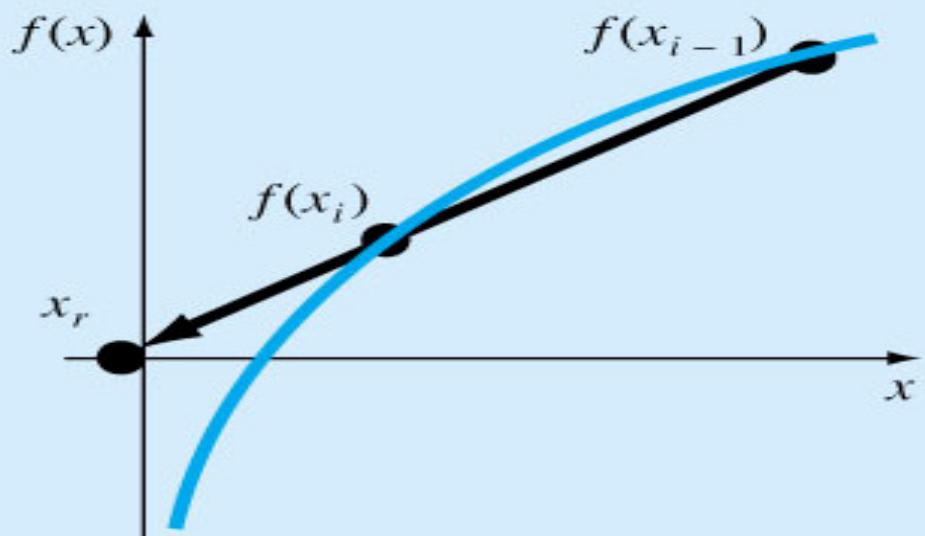


(b)



(c)

NM



(d)

Dr PVRamana

# *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 :**  $x_{i+1} = x_i - f(x_i) \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \quad i = 1, 2, 3, K$

# The Secant Method

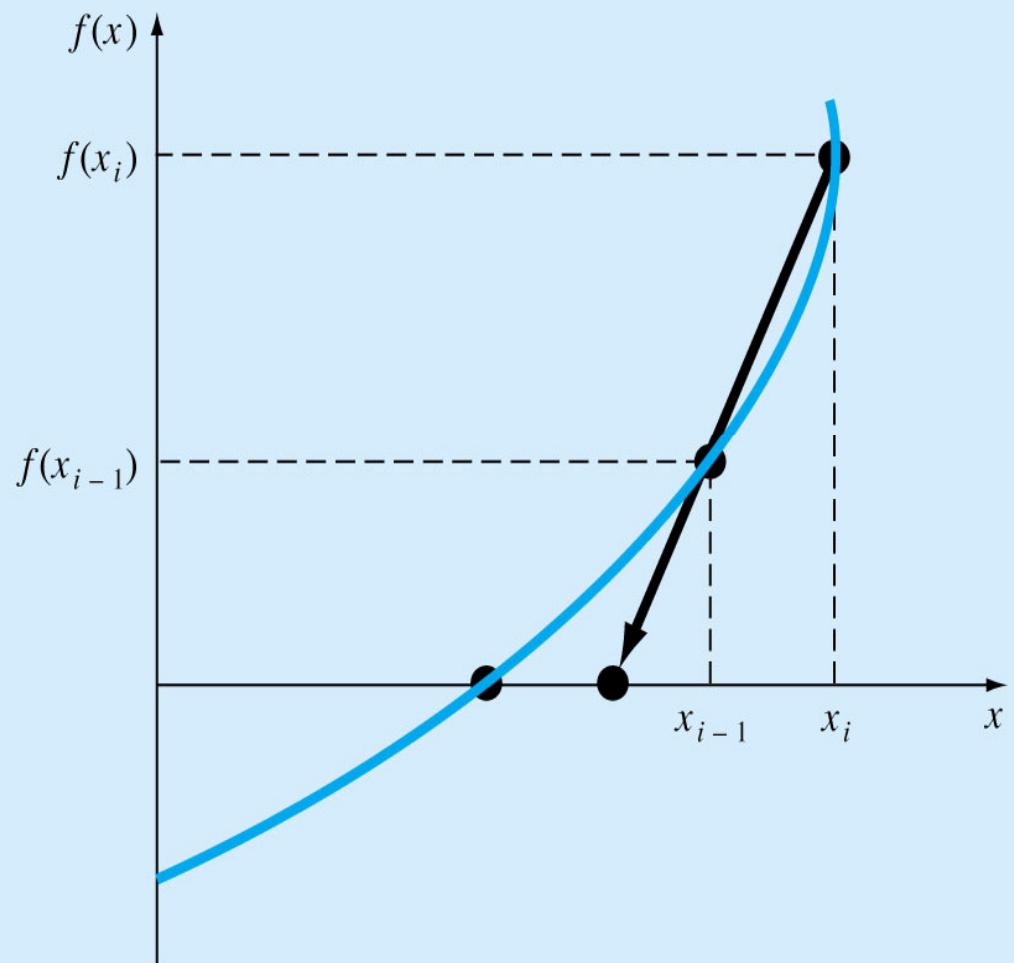
$$x_{i+1} = x_i - f(x_i) \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

- Requires two initial estimates  $x_0, x_1$ .

However, it is not a “bracketing” method.

- *The Secant Method has the same properties as Newton’s method.*

Convergence is not guaranteed for all  $x_0, f(x)$ .



# Modified Secant Method

**Newton - Raphson :**

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

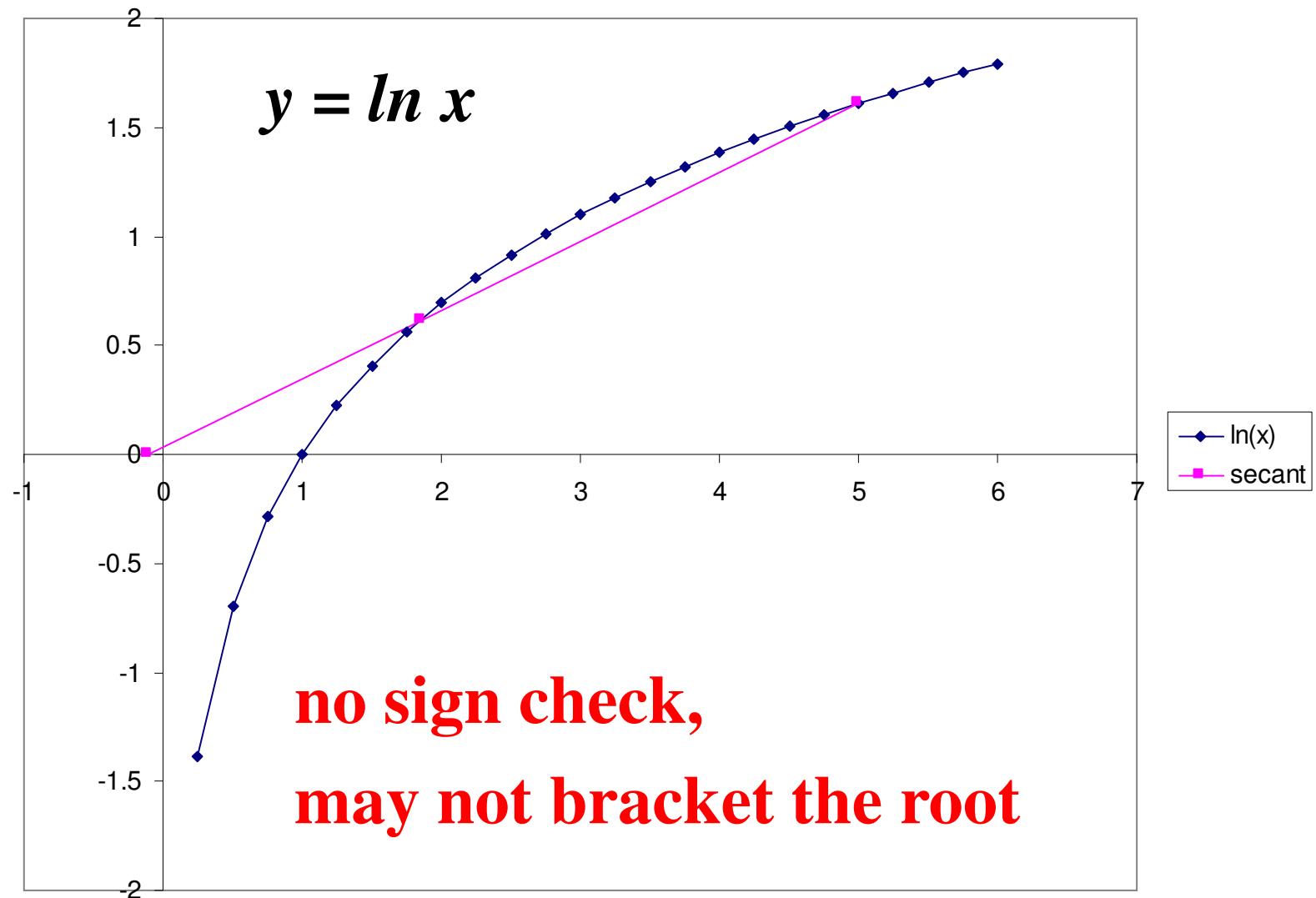
**Original Secant :**  $x_{i+1} = x_i - f(x_i) \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$

**Use a small perturbation fraction  $\delta$  to compute**

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

**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

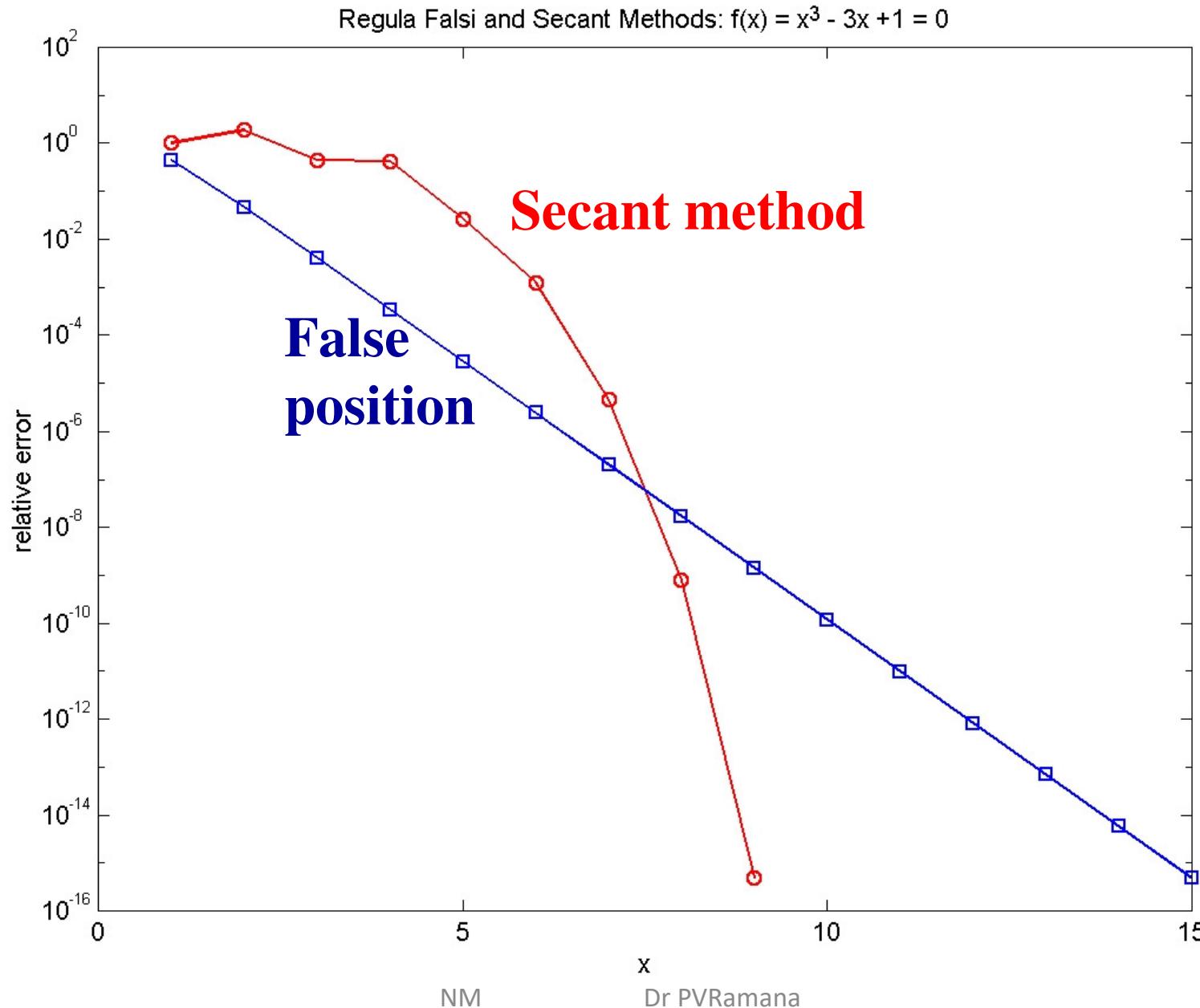
$$f(x) = x^3 - 3x + 1 = 0$$

## 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 :

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

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

# Secant Method

Assumptions :

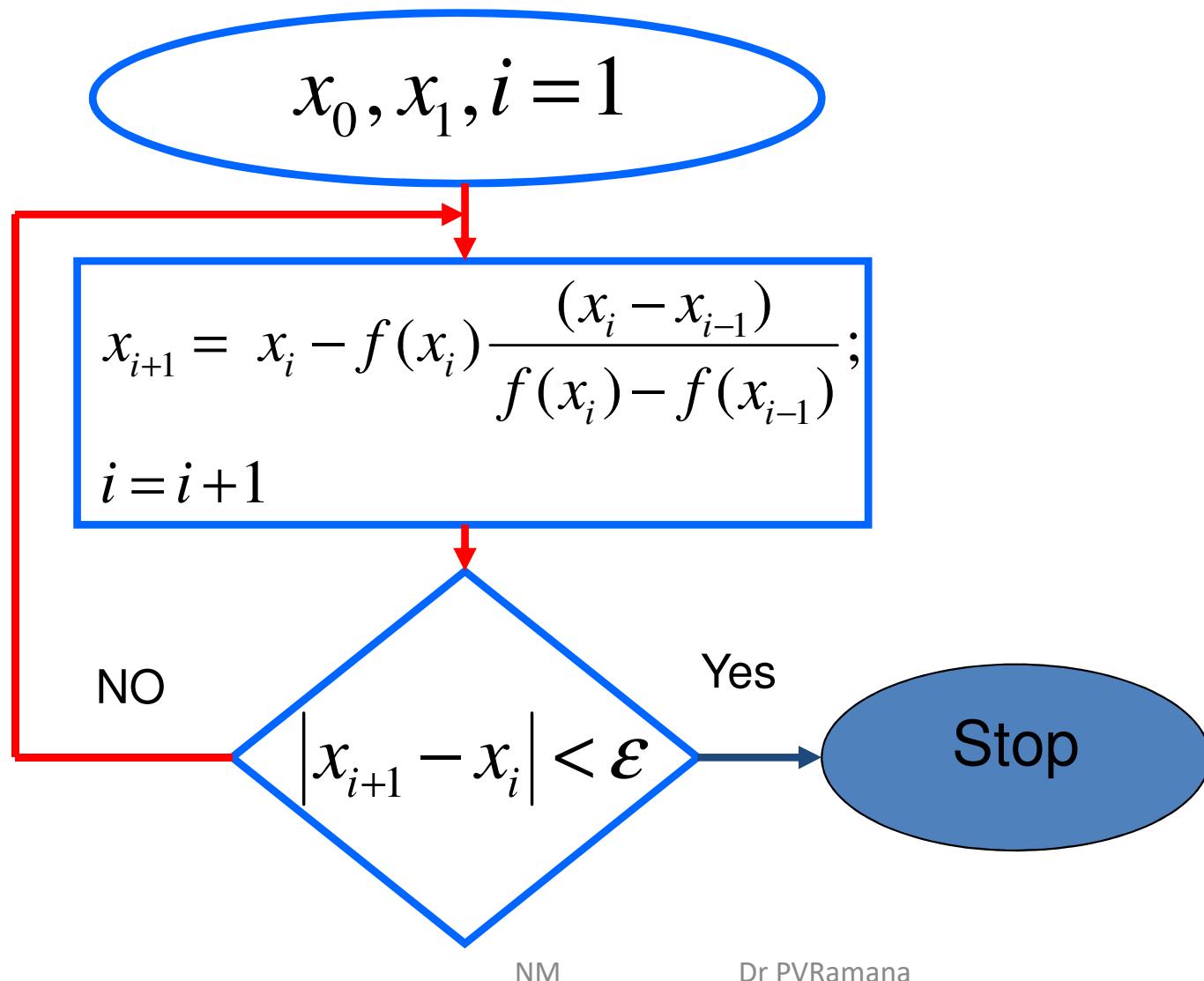
Two initial points  $x_i$  and  $x_{i-1}$

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

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

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

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

## False-point method

- Starting two points.
- Similar formula

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

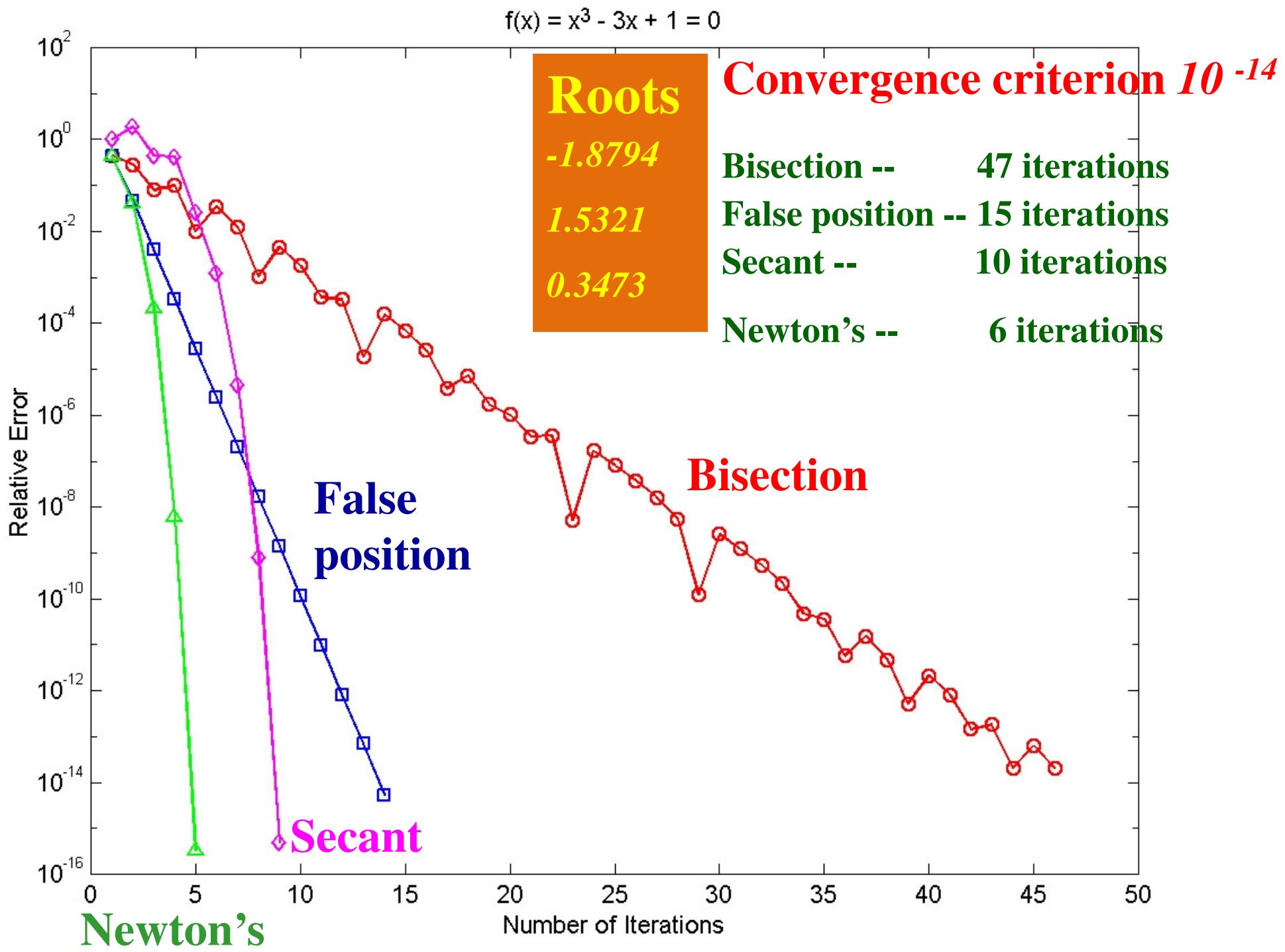
- 
- Next iteration: points replacement: if  $f(x_r) * f(x_l) < 0$ , then  $x_u = x_r$  else  $x_l = x_r$ .
  - Require bracketing
  - Always converge

## 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 replacement: always  $x_1 = x_2$  &  $x_2 = x_3$ .
  - no requirement of bracketing.
  - Faster convergence
  - May not converge



# *Modified Secant Method*

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

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

- $\delta$  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

Bracketing  
methods –  
reliable but  
slow

Open  
methods –  
fast but  
possibly  
unreliable

fzero : find  
real root of an  
equation (not  
suitable for  
double root!)

MATLAB  
fzero – fast  
and reliable

`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	-1.000000000000000	-256.0000000000000	448.0000000000000	-608.0000000000000
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	-15.99999033700192
8	1.00000001342083	NM	-0.000000214733291	-15.99999033700192

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

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

Find sign change after 22 iterations

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 =  
3.410605131648481e-013 NM

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 :

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

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

Problem : How to select  $\delta_i$  ?

If not selected properly, the method may diverge.

# Roots

$$0.8719 + 0.80i(2)$$

$$0.3912 + 1.35i(2)$$

$$-1.1053$$

## Example 1

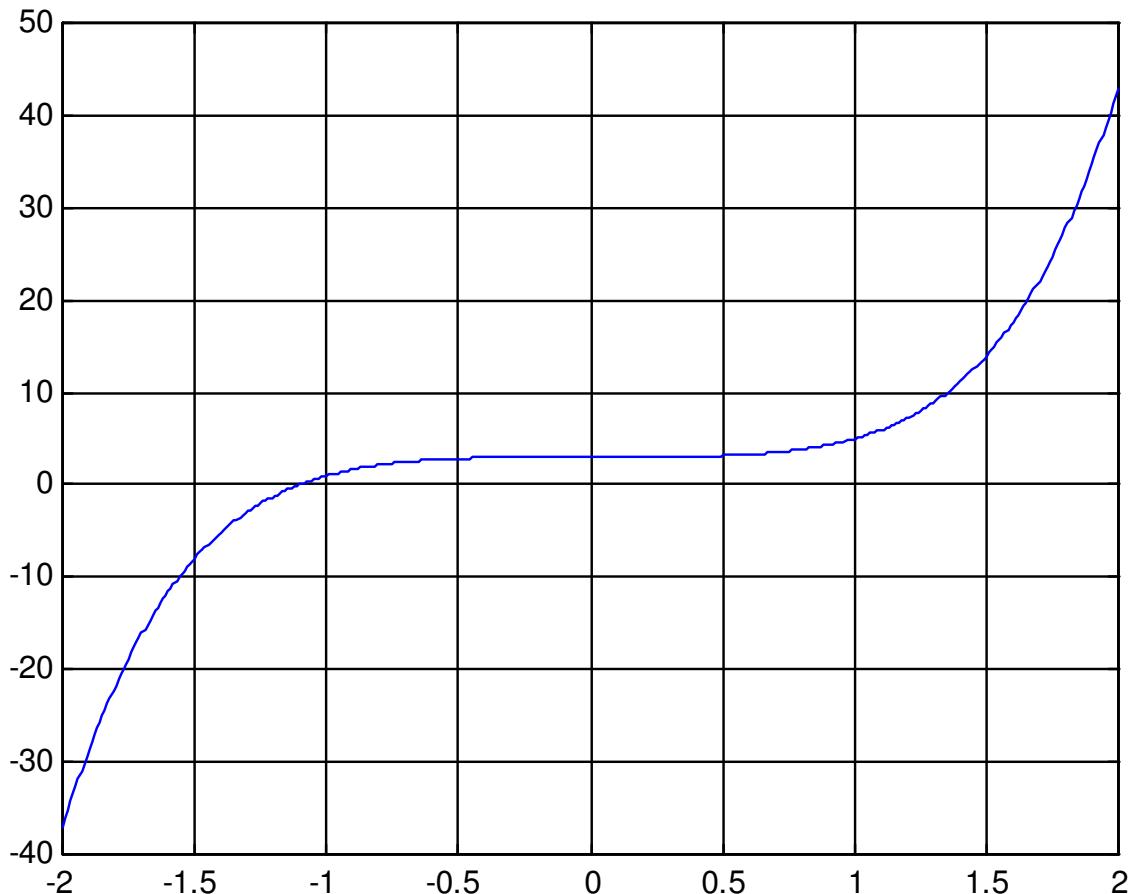
Find the roots of :

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

Initial points

$$x_0 = -1 \text{ and } x_1 = -1.1$$

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:

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

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

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

# Summary

Method	Pros	Cons
Bisection	<ul style="list-style-type: none"><li>- Easy, Reliable, Convergent</li><li>- One function evaluation per iteration</li><li>- No knowledge of derivative is needed</li></ul>	<ul style="list-style-type: none"><li>- Slow</li><li>- Needs an interval <math>[a,b]</math> containing the root, i.e., <math>f(a)f(b) &lt; 0</math></li></ul>
Newton	<ul style="list-style-type: none"><li>- Fast (if near the root)</li><li>- Two function evaluations per iteration</li></ul>	<ul style="list-style-type: none"><li>- May diverge</li><li>- Needs derivative and an initial guess <math>x_0</math> such that <math>f'(x_0)</math> is nonzero</li></ul>
Secant	<ul style="list-style-type: none"><li>- Fast (slower than Newton)</li><li>- One function evaluation per iteration</li><li>- No knowledge of derivative is needed</li></ul>	<ul style="list-style-type: none"><li>- May diverge</li><li>- Needs two initial points guess <math>x_0, x_1</math> such that <math>f(x_0) - f(x_1)</math> is nonzero</li></ul>