

ROOT FINDING BY NUMERICAL METHODS

Presented By
Lalita

Nonlinear Equation Solvers

```
graph TD; A[Nonlinear Equation Solvers] --> B[Bracketing]; A --> C[Graphical]; A --> D[Open Methods]; B --> E["Bisection  
False Position  
(Regula-Falsi)"]; D --> F["Newton Raphson  
Secant"];
```

Bracketing

Bisection
False Position
(Regula-Falsi)

Graphical

Open Methods

Newton Raphson
Secant

Bracketing Methods

- In bracketing methods, the method starts with an interval that contains the root and a procedure is used to obtain a smaller interval containing the root.
- Examples of bracketing methods:
 - Bisection method
 - False position method

Open Methods

- In the open methods, the method starts with one or more initial guess points. In each iteration, a new guess of the root is obtained.
- Open methods are usually more efficient than bracketing methods.
- They may not converge to a root.

Examples of Open methods:

Newton Raphson method

Secant method



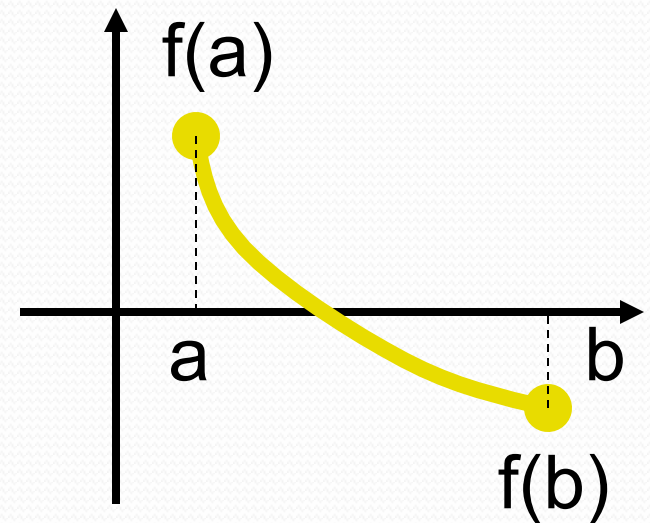
Bisection method

Introduction

- The **Bisection method** is one of the simplest methods to find a zero of a nonlinear function.
- It is also called **interval halving** method.
- To use the Bisection method, one needs an initial interval that is known to contain a zero of the function.
- The method systematically reduces the interval. It does this by dividing the interval into two equal parts, performs a simple test and based on the result of the test, half of the interval is thrown away.
- The procedure is repeated until the desired interval size is obtained.

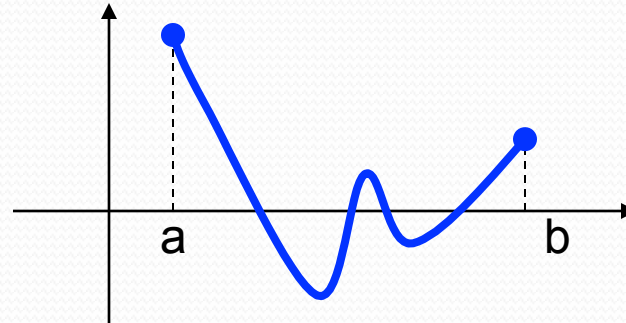
Intermediate Value Theorem

- Let $f(x)$ be defined on the interval $[a,b]$.
- Intermediate value theorem:
if a function is continuous and $f(a)$ and $f(b)$ have different signs then the function has at least one zero in the interval $[a,b]$.

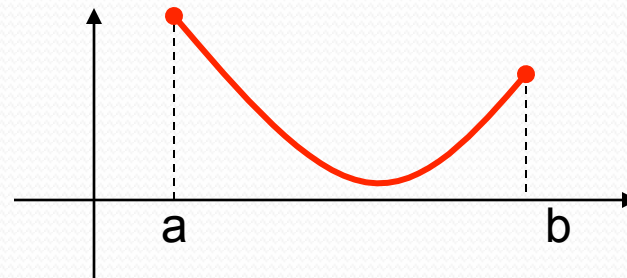


Examples

- If $f(a)$ and $f(b)$ have the same sign, the function may have an even number of real zeros or no real zeros in the interval $[a, b]$.
- Bisection method can not be used in these cases.



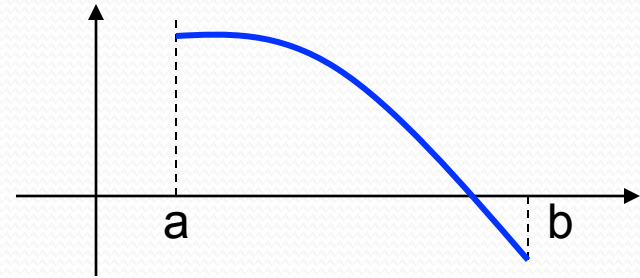
The function has four real zeros



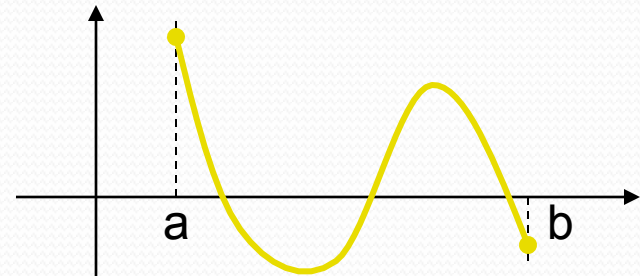
The function has no real zeros

Two More Examples

- If $f(a)$ and $f(b)$ have different signs, the function has at least one real zero.
- Bisection method can be used to find one of the zeros.



The function has one real zero



The function has three real zeros

Bisection Method

- If the function is continuous on $[a,b]$ and $f(a)$ and $f(b)$ have different signs, Bisection method obtains a new interval that is half of the current interval and the sign of the function at the end points of the interval are different.
- This allows us to repeat the Bisection procedure to further reduce the size of the interval.

Bisection Method

Assumptions:

Given an interval $[a,b]$

$f(x)$ is continuous on $[a,b]$

$f(a)$ and $f(b)$ have opposite signs.

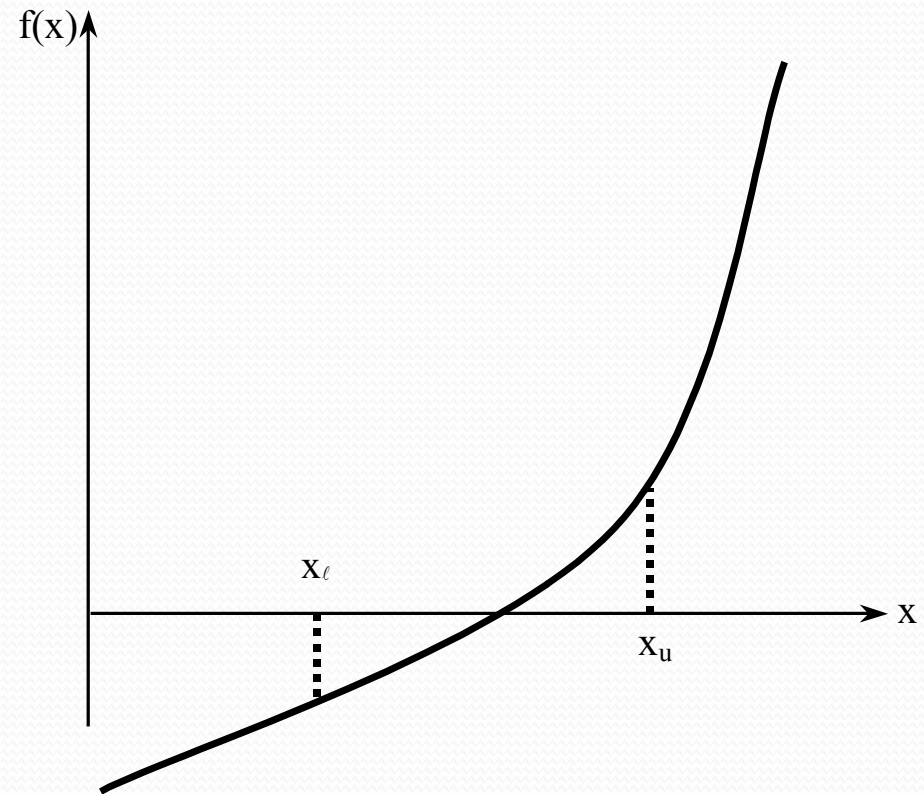
These assumptions ensure the existence of at least one zero in the interval $[a,b]$ and the bisection method can be used to obtain a smaller interval that contains the zero.



Algorithm of Bisection Method

Step 1

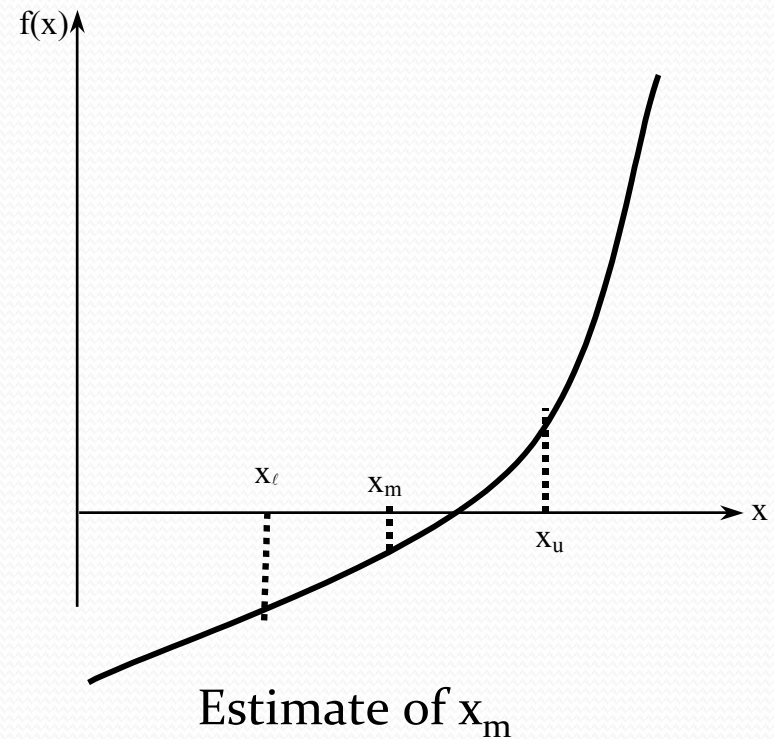
Choose x_ℓ and x_u as two guesses for the root such that $f(x_\ell) f(x_u) < 0$, or in other words, $f(x)$ changes sign between x_ℓ and x_u .



Step 2

Estimate the root, x_m of the equation $f(x) = 0$ as the mid point between x_l and x_u as

$$x_m = \frac{x_l + x_u}{2}$$



Step 3

Now check the following

- a) If $f(x_l)f(x_m) < 0$, then the root lies between x_l and x_m ;
then $x_\ell = x_l$; $x_u = x_m$.
- b) If $f(x_l)f(x_m) > 0$, then the root lies between x_m and x_u ;
then $x_\ell = x_m$; $x_u = x_u$.
- c) If $f(x_l)f(x_m) = 0$; then the root is x_m . Stop the algorithm if this is true.

Step 4

Find the new estimate of the root

$$x_m = \frac{x_l + x_u}{2}$$

Find the absolute relative approximate error

$$|\epsilon_a| = \left| \frac{x_m^{new} - x_m^{old}}{x_m^{new}} \right| \times 100$$

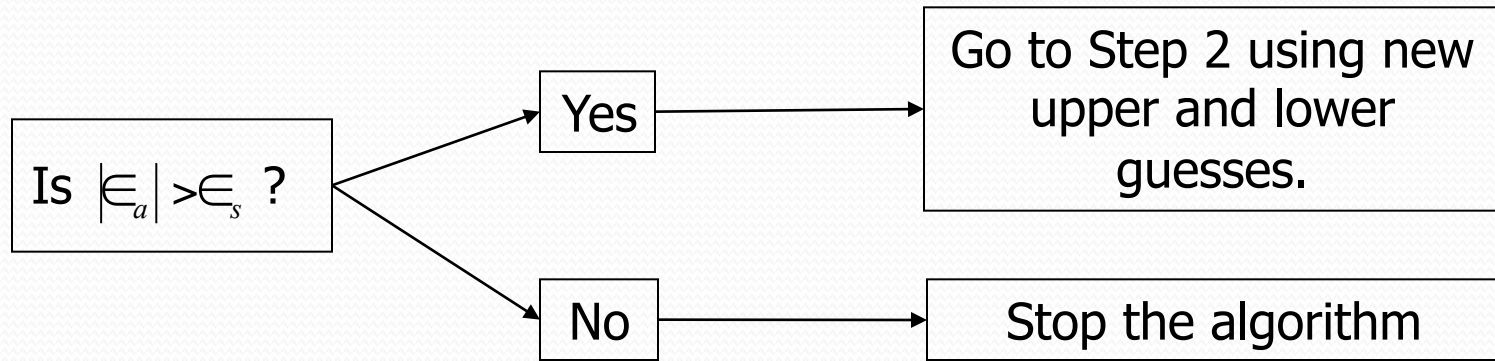
where

x_m^{old} = previous estimate of root

x_m^{new} = current estimate of root

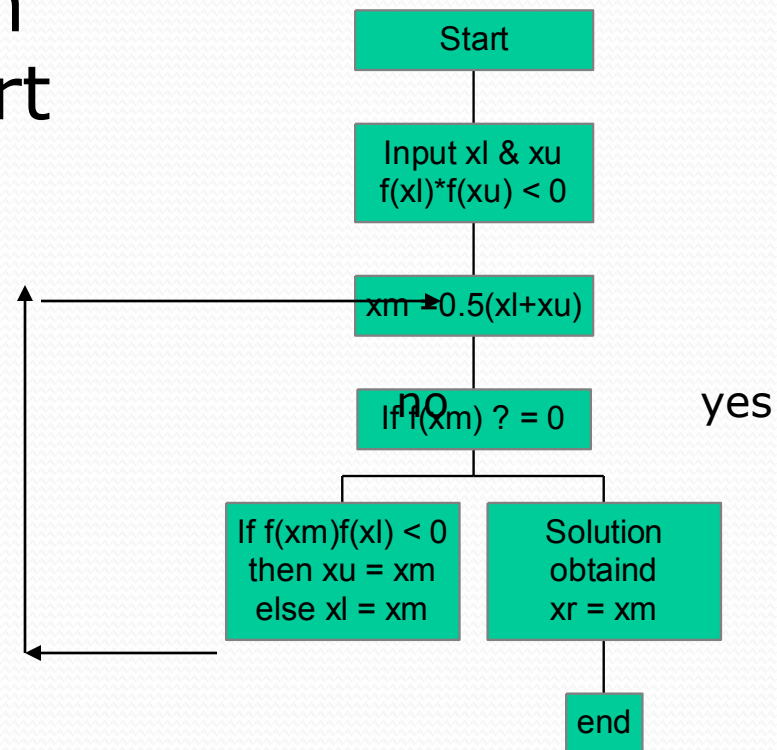
Step 5

Compare the absolute relative approximate error $|\epsilon_a|$ with the pre-specified error tolerance ϵ_s .



Note one should also check whether the number of iterations is more than the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user about it.

Bisection Flowchart



BISECTION METHOD PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define f(x) (x)*(x)*(x)-(x)-4
main()
{
int n;
float x0,x1,x2;
double f0,f1,f2;
printf("enter value of x0 x1,iteration");
scanf("%f%f %d",&x0,&x1,&n);
if((f(x0)*f(x1))<0)
{
for(int i=1;i<=n;i++)
{
x2=((x0+x1)/2);
f0=f(x0);
f1=f(x1);
f2=f(x2);
printf("x0=%f x1=%f x2=%f f0=%lf f1=%lf f2=%lf ",x0,x1,x2,f0,f1,f2);
printf("\n");
```

```
if(f0*f2<0)
x1=x2;
else
x0=x2;

}
printf("App.root=%f",x2);
}
getch();
}
```

C:\Users\DELL\Desktop\final\bis.exe

enter value of x0 x1, iteration 1 2 7

x0=1.000000	x1=2.000000	x2=1.500000	f0=-4.000000	f1=2.000000	f2=-2.125000
x0=1.500000	x1=2.000000	x2=1.750000	f0=-2.125000	f1=2.000000	f2=-0.390625
x0=1.750000	x1=2.000000	x2=1.875000	f0=-0.390625	f1=2.000000	f2=0.716797
x0=1.750000	x1=1.875000	x2=1.812500	f0=-0.390625	f1=0.716797	f2=0.141846
x0=1.750000	x1=1.812500	x2=1.781250	f0=-0.390625	f1=0.141846	f2=-0.129608
x0=1.781250	x1=1.812500	x2=1.796875	f0=-0.129608	f1=0.141846	f2=0.004803
x0=1.781250	x1=1.796875	x2=1.789063	f0=-0.129608	f1=0.004803	f2=-0.062730

App.root=1.789063_

ORDER OF CONVERGENCE OF ITERATIVE METHODS

Convergence of an iterative method is judged by the order at which the error between successive approximations to the root decreases.

An iterative method is said to be k^{th} order convergent if k is the largest positive real number, such that

$$\lim_{i \rightarrow \infty} \left| \frac{e_{i+1}}{e_i^k} \right| \leq A$$

where A is a non-zero finite number called asymptotic error constant and it depends on derivative of $f(x)$ at an approximate root x .

e_i and e_{i+1} are the errors in successive approximations. k^{th} order convergence gives us the idea that in each iteration, the number of significant digits in each approximation increases k times.

The error in any step is proportional to the k^{th} power of the error in the previous step.

PROVE THAT BISECTION METHOD ALWAYS CONVERGES

Let $[p_n, q_n]$ be the interval at n^{th} step of bisection, having a root of the equation $f(x) = 0$. Let x_n be the n^{th} approximation for the root. Then, initially, $p_1 = a$ and $q_1 = b$.

$$\Rightarrow x_1 = \text{first approximation} = \left(\frac{p_1 + q_1}{2} \right)$$

$$\Rightarrow p_1 < x_1 < q_1$$

Now either the root lies in $[a, x_1]$ or in $[x_1, b]$.

$$\therefore \text{ either } [p_2, q_2] = [p_1, x_1] \quad \text{or} \quad [p_2, q_2] = [x_1, q_1]$$

$$\Rightarrow \text{ either } p_2 = p_1, q_2 = x_1 \quad \text{or} \quad p_2 = x_1, q_2 = q_1$$

$$\Rightarrow p_1 \leq p_2, q_2 \leq q_1$$

$$\text{Also, } x_2 = \frac{p_2 + q_2}{2} \text{ so that } p_2 < x_2 < q_2$$

Continuing this way, we obtain that at n^{th} step,

$$x_n = \frac{p_n + q_n}{2}, p_n < x_n < q_n$$

and $p_1 \leq p_2 \leq \dots \leq p_n$ and $q_1 \geq q_2 \geq \dots \geq q_n$

$\therefore \langle p_1, p_2, \dots, p_n, \dots \rangle$ is a bounded, non-decreasing sequence bounded by b and $\langle q_1, q_2, \dots, q_n, \dots \rangle$ is a bounded, non-increasing sequence of numbers bounded by a .

Hence, both these sequences converge.

Let, $\lim_{n \rightarrow \infty} p_n = p$ and $\lim_{n \rightarrow \infty} q_n = q$.

Now, since the length of the interval is decreasing at every step, we get that

$$\lim_{n \rightarrow \infty} (q_n - p_n) = 0 \Rightarrow q = p$$

$$\begin{aligned}
\text{Also,} \quad & p_n < x_n < q_n \\
\Rightarrow & \lim p_n \leq \lim x_n \leq \lim q_n \\
\Rightarrow & p \leq \lim x_n \leq q \\
\Rightarrow & \lim x_n = p = q \tag{2}
\end{aligned}$$

Further, since a root lies in $[p_n, q_n]$, we shall have

$$\begin{aligned}
& f(p_n) \cdot f(q_n) < 0 \\
\Rightarrow & 0 \geq \lim_{n \rightarrow \infty} [f(p_n) \cdot f(q_n)] \\
\Rightarrow & 0 \geq f(p) \cdot f(q) \\
\Rightarrow & 0 \geq [f(p)]^2
\end{aligned}$$

But, $[f(p)]^2 \geq 0$ being a square

$$\begin{aligned}
\therefore \text{ we get} \quad & f(p) = 0 \\
\therefore p \text{ is a root of} \quad & f(x) = 0 \tag{3}
\end{aligned}$$

From (2) and (3), we see that $\langle x_n \rangle$ converges necessarily to a root of equation $f(x) = 0$

The method is not rapidly converging, but it is useful in the sense that it converges surely.

Stopping Criteria

Two common stopping criteria

1. Stop after a fixed number of iterations
2. Stop when the absolute error is less than a specified value

How are these criteria related?

Stopping Criteria

- c_n : is the midpoint of the interval at the n^{th} iteration
(c_n is usually used as the estimate of the root).
- r : is the zero of the function.

After n iterations:

$$|error| = |r - c_n| \leq E_a^n = \frac{b - a}{2^n} = \frac{\Delta x^0}{2^n}$$

Convergence Analysis

Given $f(x)$, a , b , and ε

How many iterations are needed such that: $|x - r| \leq \varepsilon$

where r is the zero of $f(x)$ and x is the bisection estimate (i.e., $x = c_k$)?

$$n \geq \frac{\log(b - a) - \log(\varepsilon)}{\log(2)}$$



Bisection Method

Advantages

- **Simple** and easy to implement
- The bisection algorithm is guaranteed to converge
- **One** function evaluation per iterations
- **No** knowledge of the **derivative** is needed
- The function does **not** have to be **differentiable**

Disadvantage

- **Slow** to converge
- **Good** intermediate approximations may be **discarded**



False Position method

False Position

- This technique is similar to the bisection method except that the next iterate is taken as the line of interception between the pair of x -values and the x -axis rather than at the midpoint.

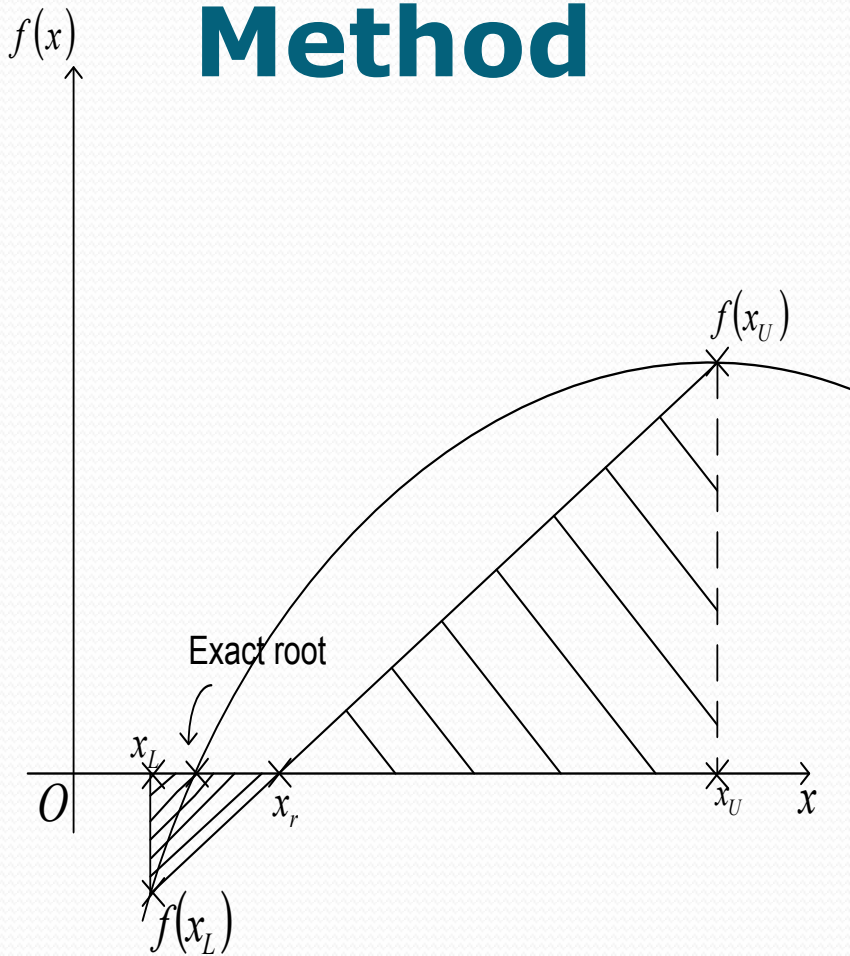
This is slow convergence method and may be thought of as an attempt to improve the convergence characteristic of bisection method. Its also known as the method of linear interpolation.

False-Position (point) Method

Why bother with another method?

- The bisection method is simple and guaranteed to converge (single root)
- But the convergence is slow and non-monotonic!
- Bisection only the sign, not the value $f(x_k)$ itself
- **False-position method takes advantage of function curve shape**
- False position method may converge more quickly

Derivation of False Position Method



$$f(x) = 0 \quad (1)$$

In the Bisection method

$$f(x_L) * f(x_U) < 0 \quad (2)$$

$$x_r = \frac{x_L + x_U}{2} \quad (3)$$

Figure 1 False-Position Method

False-Position Method

Based on two similar triangles, shown in Figure 1, one gets:

$$\frac{f(x_L)}{x_r - x_L} = \frac{f(x_U)}{x_r - x_U} \quad (4)$$

The signs for both sides of Eq. (4) is consistent, since:

$$f(x_L) < 0; x_r - x_L > 0$$

$$f(x_U) > 0; x_r - x_U < 0$$

From Eq. (4), one obtains

$$\begin{aligned}(x_r - x_L)f(x_U) &= (x_r - x_U)f(x_L) \\ x_U f(x_L) - x_L f(x_U) &= x_r \{f(x_L) - f(x_U)\}\end{aligned}$$

The above equation can be solved to obtain the next predicted root x_r , as

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)} \quad (5)$$

The above equation,

$$x_r = x_U - \frac{f(x_U)\{x_L - x_U\}}{f(x_L) - f(x_U)} \quad (6)$$

or

$$x_r = x_L - \frac{f(x_L)}{\left\{ \frac{f(x_U) - f(x_L)}{x_U - x_L} \right\}} \quad (7)$$

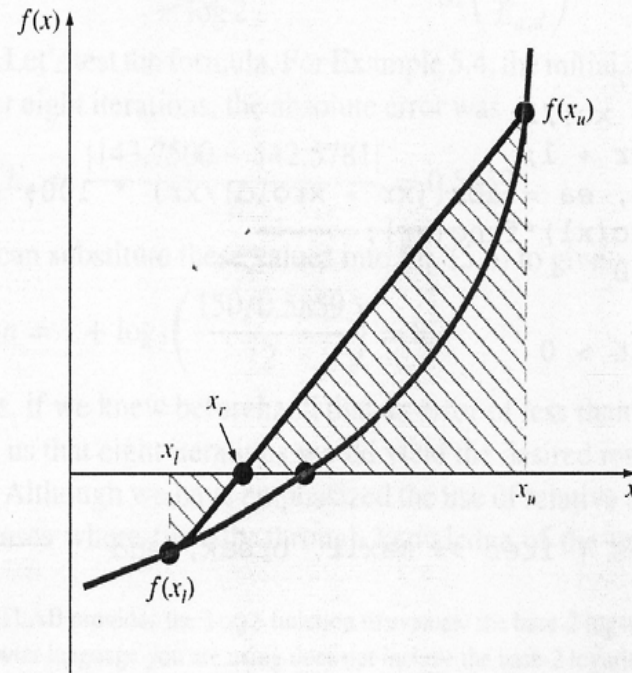


Algorithm of False position method

Step 1

Choose x_L and x_U as two guesses for the root such that

$$f(x_L)f(x_U) < 0$$



Step2

Estimate the root,

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

Step 3

Now check the following

(a) If $f(x_L)f(x_m) < 0$, then the root lies between x_L and x_m ; then $x_L = x_L$ and $x_U = x_m$

(b) If $f(x_L)f(x_m) > 0$, then the root lies between x_m and x_U ; then $x_L = x_m$ and $x_U = x_U$

Step 4

Find the new estimate of the root

$$x_m = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)}$$

Find the absolute relative approximate error as

$$|\epsilon_a| = \left| \frac{x_m^{new} - x_m^{old}}{x_m^{new}} \right| \times 100$$

where

x_m^{new} = estimated root from present iteration

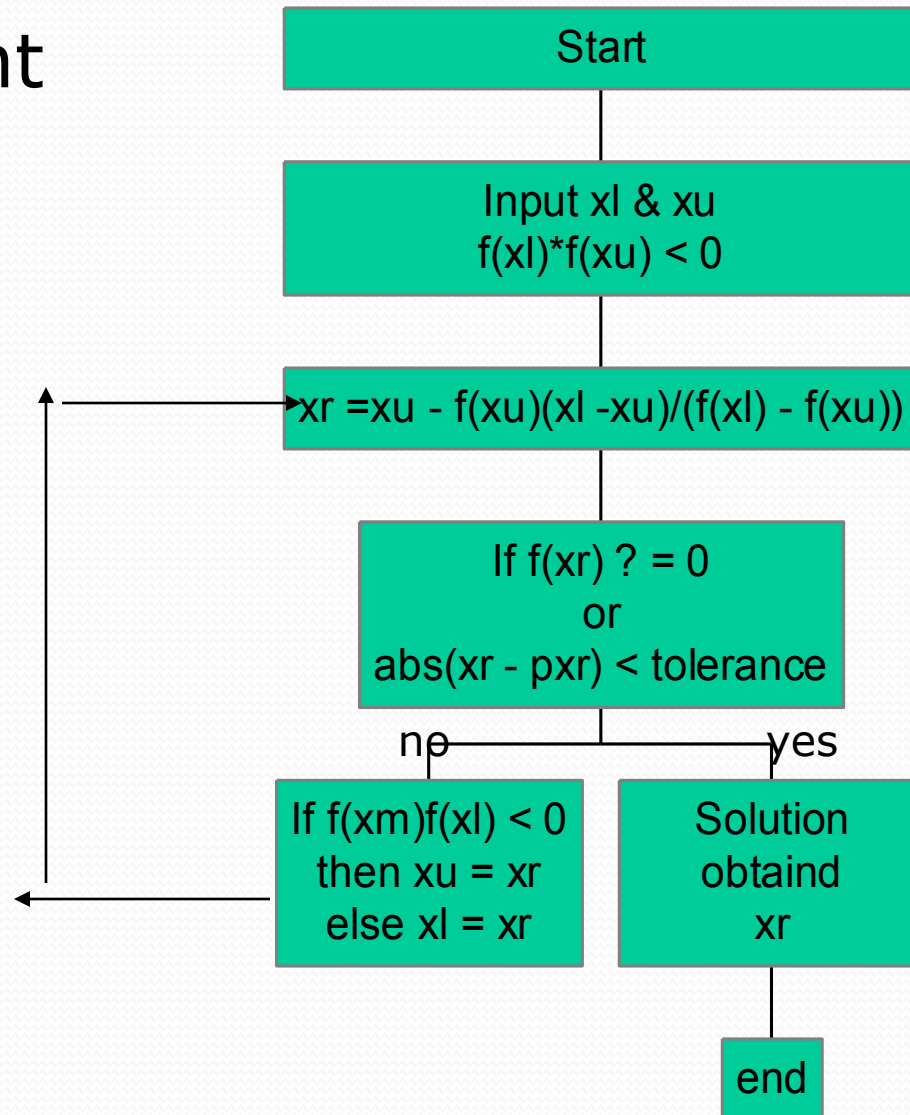
x_m^{old} = estimated root from previous iteration

Step 5

say $\epsilon_s = 10^{-3} = 0.001$. If $|\epsilon_a| > \epsilon_s$, then go to step 3, else stop the algorithm.

▪

False-point Method Flowchart



REGULA FALSI METHOD PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define f(x) (x)*(x)*(x)-(x)-4
main()
{
int n;
float x0,x1,x2;
double f0,f1,f2;
printf("enter value of x0 ,x1 and iteration");
scanf("%f%f %d",&x0,&x1,&n);
if(f(x0)*f(x1)<0)
{
for(int i=1;i<=n;i++)
{
f0=f(x0);
f1=f(x1);
f2=f(x2);
x2=(x0*f1-x1*f0)/(f1-f0);
```

```
printf("x0=%f x1=%f x2=%f f0=%f f1=%f f2=%f\t ",x0,x1,x2,f0,f1,f2);
if(f0*f2<0)
x1=x2;
else
x0=x2;
if(i==n)
printf("\nApp. root=%f",x2);
}
}
getch();
}
```

C:\Users\DELL\Desktop\final\regula.exe

enter value of x_0 , x_1 and iteration 1 2 7

$x_0=1.000000$	$x_1=2.000000$	$x_2=1.666667$	$f_0=-4.000000$	$f_1=2.000000$	$f_2=-4.000000$
$x_0=1.666667$	$x_1=2.000000$	$x_2=1.780488$	$f_0=-1.037037$	$f_1=2.000000$	$f_2=-1.037037$
$x_0=1.780488$	$x_1=2.000000$	$x_2=1.794474$	$f_0=-0.136098$	$f_1=2.000000$	$f_2=-0.136098$
$x_0=1.794474$	$x_1=2.000000$	$x_2=1.796107$	$f_0=-0.016025$	$f_1=2.000000$	$f_2=-0.016025$
$x_0=1.796107$	$x_1=2.000000$	$x_2=1.796297$	$f_0=-0.001863$	$f_1=2.000000$	$f_2=-0.001863$
$x_0=1.796297$	$x_1=2.000000$	$x_2=1.796319$	$f_0=-0.000217$	$f_1=2.000000$	$f_2=-0.000217$
$x_0=1.796319$	$x_1=2.000000$	$x_2=1.796322$	$f_0=-0.000025$	$f_1=2.000000$	$f_2=-0.000025$

App. root=1.796322

CONVERGENCE OF REGULA-FALSI METHOD

If $\langle x_n \rangle$ is the sequence of approximations obtained from

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} f(x_n) \quad (1)$$

and α is the exact value of the root of the equation $f(x) = 0$, then

Let
$$x_n = \alpha + e_n$$

$$x_{n+1} = \alpha + e_{n+1}$$

where e_n, e_{n+1} are the errors involved in n^{th} and $(n + 1)^{\text{th}}$ approximations, respectively.

Clearly, $f(\alpha) = 0$. Hence, (1) gives

$$\alpha + e_{n+1} = \alpha + e_n - \frac{(e_n - e_{n-1})}{f(\alpha + e_n) - f(\alpha + e_{n-1})} \cdot f(\alpha + e_n)$$

OR

$$\begin{aligned}
 e_{n+1} &= \frac{e_{n-1} f(\alpha + e_n) - e_n f(\alpha + e_{n-1})}{f(\alpha + e_n) - f(\alpha + e_{n-1})} \\
 &= \frac{e_{n-1} \left[f(\alpha) + e_n f'(\alpha) + \frac{e_n^2}{2!} f''(\alpha) + \dots \right] - e_n \left[f(\alpha) + e_{n-1} f'(\alpha) + \frac{e_{n-1}^2}{2!} f''(\alpha) + \dots \right]}{\left[f(\alpha) + e_n f'(\alpha) + \frac{e_n^2}{2!} f''(\alpha) + \dots \right] - \left[f(\alpha) + e_{n-1} f'(\alpha) + \frac{e_{n-1}^2}{2!} f''(\alpha) + \dots \right]} \\
 &= \frac{(e_{n-1} - e_n) f(\alpha) + \frac{e_{n-1} e_n}{2!} (e_n - e_{n-1}) f''(\alpha) + \dots}{(e_n - e_{n-1}) f'(\alpha) + \frac{(e_n - e_{n-1})(e_n + e_{n-1})}{2!} f''(\alpha) + \dots} \\
 &= \frac{\frac{e_{n-1} e_n}{2} f''(\alpha) + \dots}{f'(\alpha) + \left(\frac{e_n + e_{n-1}}{2} \right) f''(\alpha) + \dots} \quad | \because f(\alpha) = 0
 \end{aligned}$$

or
$$e_{n+1} = \frac{e_n e_{n-1} f''(\alpha)}{2! f'(\alpha)} \quad (2)$$

(neglecting high powers of e_n, e_{n-1})

Let $e_{n+1} = c e_n^k$, where c is a constant and $k > 0$.

$\therefore e_n = c e_{n-1}^k$

or $e_{n-1} = c^{-1/k} e_n^{1/k}$

\therefore From (2)
$$c e_n^k = \frac{e_n c^{-1/k} e_n^{1/k} f''(\alpha)}{2! f'(\alpha)} = \frac{c^{-1/k}}{2!} e_n^{1+1/k} \cdot \frac{f''(\alpha)}{f'(\alpha)}$$

Comparing the two sides, we get

$$k = 1 + \frac{1}{k} \quad \text{and} \quad c = \frac{c^{-1/k} f''(\alpha)}{2! f'(\alpha)}$$


Now, $k = 1 + \frac{1}{k} \Rightarrow k^2 - k - 1 = 0 \Rightarrow k = 1.618$

Also, $c = c^{-1/k} \cdot \frac{1}{2!} \frac{f''(\alpha)}{f'(\alpha)}$

$$c^{1+\frac{1}{k}} = c^{1.618} = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)}$$

or $c = \left[\frac{f''(\alpha)}{2f'(\alpha)} \right]^{0.618}$

This gives the rate of convergence and $k = 1.618$ gives the order of convergence.



Advantages and Disadvantages of False Position method



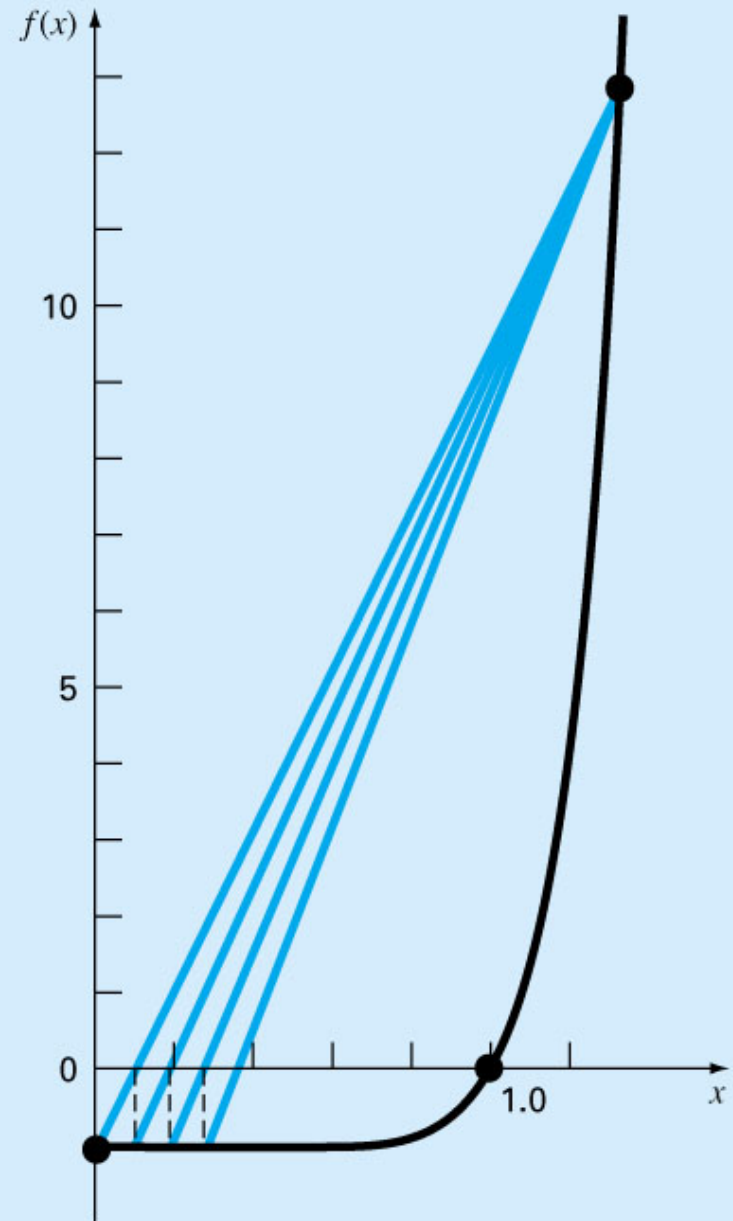
- **Advantages**

- Always converges
- Faster convergence than bisection
- Often superior to bisection

- **Disadvantages**

- Can be VERY slow
- Like Bisection, need an initial interval around the root.

- False-Position Method
 - $f(x)=x^{10}-1$
 - The root is hard to find because the curvature is steep



Newton-Raphson Method

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.

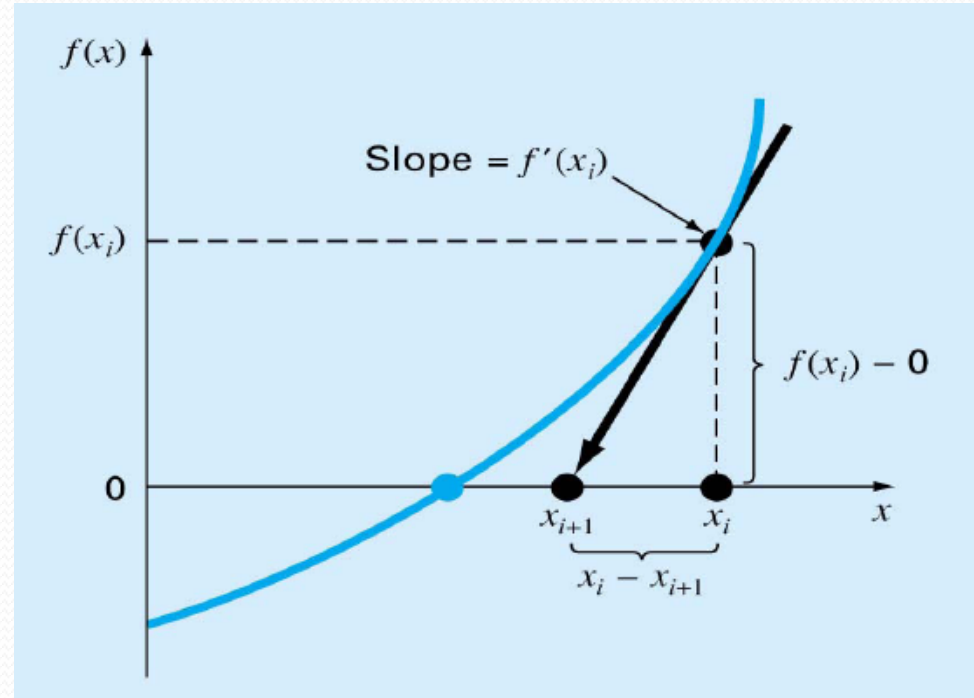
Assumptions:

- $f(x)$ is continuous and the first derivative is known
- An initial guess x_0 such that $f'(x_0) \neq 0$ is given

Newton Raphson Method

- Graphical Depiction -

- If the initial guess at the root is x_i , then 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} .



NEWTON-RAPHSON METHOD

This method is generally used to improve the result obtained by one of the previous methods. Let x_0 be an approximate root of $f(x) = 0$ and let $x_1 = x_0 + h$ be the correct root so that $f(x_1) = 0$.

Expanding $f(x_0 + h)$ by Taylor's series, we get

$$f(x_0) + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + \dots = 0$$

Since h is small, neglecting h^2 and higher powers of h , we get

$$f(x_0) + hf'(x_0) = 0 \quad \text{or} \quad h = -\frac{f(x_0)}{f'(x_0)} \quad (29)$$

A better approximation than x_0 is therefore given by x_1 , where

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Successive approximations are given by x_2, x_3, \dots, x_{n+1} , where

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (30) \quad (n = 0, 1, \dots)$$

which is the Newton-Raphson formula.



Algorithm for Newton-Raphson Method

Step 1

Evaluate $f'(x)$

Step 2

Use an initial guess of the root, x_i , to estimate the new value of the root, x_{i+1} , as

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

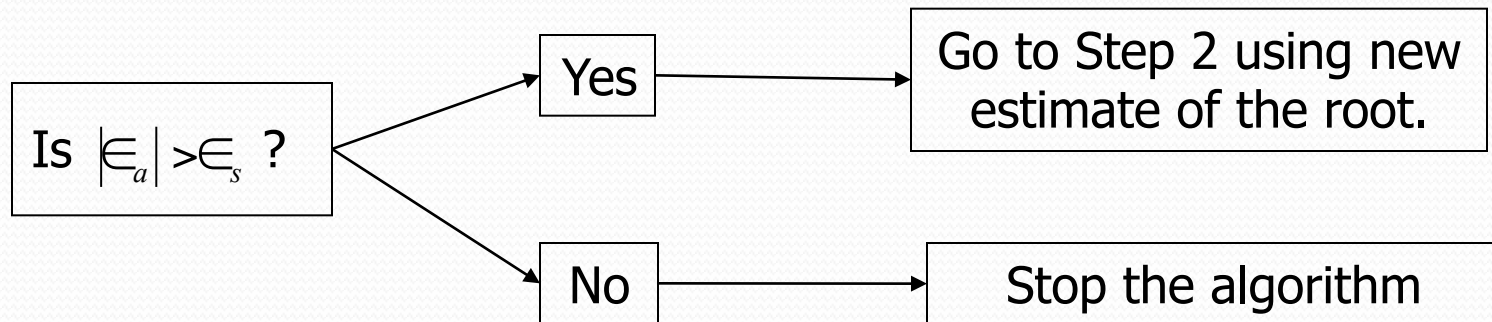
Step 3

Find the absolute relative approximate error $|\epsilon_a|$ as

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

Step 4

Compare the absolute relative approximate error with the pre-specified relative error tolerance .



Also, check if the number of iterations has exceeded the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user.

NEWTON'S RAPHSON METHOD PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define f(x) (x)*(x)*(x)-(x)-4
#define df(x) 3*(x)*(x)-1
main()
{
int n;
float x0,x1;
double f0,df0;
printf("enter value of x0 and iteration ");
scanf("%f %d",&x0,&n);
for(int i=1;i<=n;i++)
{
f0=f(x0);
df0=df(x0);
x1=x0-(f0/df0);
x0=x1;
```



```
printf("x0=%f\t x1=%f\t f0=%lf\t df0=%lf\t\t\t ",x0,x1,f0,df0);  
}  
printf("\n App.root=%f",x1);  
getch();  
}
```

C:\Users\DELL\Desktop\final\newton.exe

enter value of x_0 and iteration 2 4

$x_0=1.818182$	$x_1=1.818182$	$f_0=2.000000$	$df_0=11.000000$
$x_0=1.796613$	$x_1=1.796613$	$f_0=0.192337$	$df_0=8.917356$
$x_0=1.796322$	$x_1=1.796322$	$f_0=0.002527$	$df_0=8.683455$
$x_0=1.796322$	$x_1=1.796322$	$f_0=0.000001$	$df_0=8.680318$

App.root=1.796322

ORDER OF CONVERGENCE

Suppose x_n differs from the root α by a small quantity e_n so that

$$x_n = \alpha + e_n \quad \text{and} \quad x_{n+1} = \alpha + e_{n+1}$$

Then () becomes, $e_{n+1} = e_n - \frac{f(\alpha + e_n)}{f'(\alpha + e_n)}$

$$= e_n - \frac{f(\alpha) + e_n f'(\alpha) + \frac{e_n^2}{2!} f''(\alpha) + \dots}{f'(\alpha) + e_n f''(\alpha) + \dots} \quad (\text{By Taylor's expansion})$$

$$= e_n - \frac{e_n f'(\alpha) + \frac{e_n^2}{2} f''(\alpha) + \dots}{f'(\alpha) + e_n f''(\alpha) + \dots} \quad | \because f(\alpha) = 0$$

$$= \frac{e_n^2 f''(\alpha)}{2[f'(\alpha) + e_n f''(\alpha)]} \quad | \text{ Neglect high powers of } e_n$$

$$= \frac{e_n^2}{2} \cdot \frac{f''(\alpha)}{f'(\alpha) \left\{ 1 + e_n \frac{f''(\alpha)}{f'(\alpha)} \right\}}$$

$$= \frac{e_n^2}{2} \cdot \frac{f''(\alpha)}{f'(\alpha)} \left\{ 1 + e_n \frac{f''(\alpha)}{f'(\alpha)} \right\}^{-1}$$

$$= \frac{e_n^2}{2} \frac{f''(\alpha)}{f'(\alpha)} \left\{ 1 - e_n \frac{f''(\alpha)}{f'(\alpha)} + \dots \right\}$$

$$= \frac{e_n^2}{2} \frac{f''(\alpha)}{f'(\alpha)} - \frac{e_n^3}{2} \left\{ \frac{f''(\alpha)}{f'(\alpha)} \right\}^2 + \dots$$

$$\frac{e_{n+1}}{e_n^2} = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)} - \frac{e_n}{2} \left\{ \frac{f''(\alpha)}{f'(\alpha)} \right\}^2 + \dots$$

$$= \frac{f''(\alpha)}{2f'(\alpha)}$$

(Neglecting terms containing powers of e_n)

Hence by definition, the order of convergence of Newton-Raphson method is 2, i.e., Newton-Raphson method is quadratic convergent.

This also shows that subsequent error at each step is proportional to the square of the previous error and as such the *convergence is quadratic*.

Hence, if at the first iteration we have an answer correct to one decimal place, then it should be correct to two places at the second iteration, and to four places at the third iteration.

This means that the number of correct decimal places at each iteration is almost doubled.

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

Convergence Analysis

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'S ITERATIVE FORMULAE FOR FINDING INVERSE, SQUARE ROOT

1. Inverse. The reciprocal or inverse of a number 'a' can be considered as a root of the equation $\frac{1}{x} - a = 0$, which can be solved by Newton's method.

Since $f(x) = \frac{1}{x} - a, f'(x) = -\frac{1}{x^2}$

∴ Newton's formula gives

$$x_{n+1} = x_n + \frac{\left(\frac{1}{x_n} - a\right)}{\left(-\frac{1}{x_n^2}\right)}$$

$$x_{n+1} = x_n (2 - ax_n)$$

2. Square root. The square root of 'a' can be considered a root of the equation $x^2 - a = 0$, solvable by Newton's method.

Since $f(x) = x^2 - a$, $f'(x) = 2x$

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}$$

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$$

8. Inverse square root. Equation is $\frac{1}{x^2} - a = 0$

Iterative formula is

$$x_{n+1} = \frac{1}{2} x_n (3 - a x_n^2)$$

4. **General formula for p^{th} root.** The p^{th} root of a can be considered a root of the equation $x^p - a = 0$. To solve this by Newton's method, we have

$$f(x) = x^p - a \quad \text{and hence,} \quad f'(x) = px^{p-1}$$

COMPUTER-BASED NUMERICAL AND STATISTICAL TECHNIQUES

\therefore The iterative formula is $x_{n+1} = x_n - \frac{(x_n^p - a)}{px_n^{p-1}}$

$$x_{n+1} = \frac{(p-1)x_n^p + a}{px_n^{p-1}}$$

Also, the general formula for the reciprocal of p^{th} root of a is

$$x_{n+1} = x_n \left(\frac{p+1 - ax_n^p}{p} \right)$$

RATE OF CONVERGENCE OF NEWTON'S SQUARE ROOT FORMULA

Let $\sqrt{a} = \alpha$ so that $a = \alpha^2$. If we write

$$x_n = \alpha \left(\frac{1 + e_n}{1 - e_n} \right)$$

then,

$$x_{n+1} = \alpha \left(\frac{1 + e_{n+1}}{1 - e_{n+1}} \right) \quad (31)$$

Also, by formula, $x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$, we get

$$\begin{aligned} x_{n+1} &= \frac{1}{2} \left[\alpha \left(\frac{1 + e_n}{1 - e_n} \right) + \frac{a}{\alpha \left(\frac{1 + e_n}{1 - e_n} \right)} \right] \\ &= \alpha \left(\frac{1 + e_n^2}{1 - e_n^2} \right) \quad (32) \quad (\because a = \alpha^2) \end{aligned}$$

Comparing (31) and (32), we get $e_{n+1} = e_n^2$
confirming quadratic convergence of Newton's method.

RATE OF CONVERGENCE OF NEWTON'S INVERSE FORMULA

Let $\alpha = \frac{1}{a}$ i.e., $a = \frac{1}{\alpha}$. If we write $x_n = \alpha(1 - e_n)$

then, $x_{n+1} = \alpha(1 - e_{n+1})$

By formula, $x_{n+1} = x_n(2 - ax_n)$, we get

$$x_{n+1} = \alpha(1 - e_n) [2 - a\alpha(1 - e_n)] = \alpha(1 - e_n^2) \quad \because a\alpha = 1$$

Comparing, we get $e_{n+1} = e_n^2$, hence, convergence is quadratic.



Advantages and Drawbacks of Newton Raphson Method

Advantages

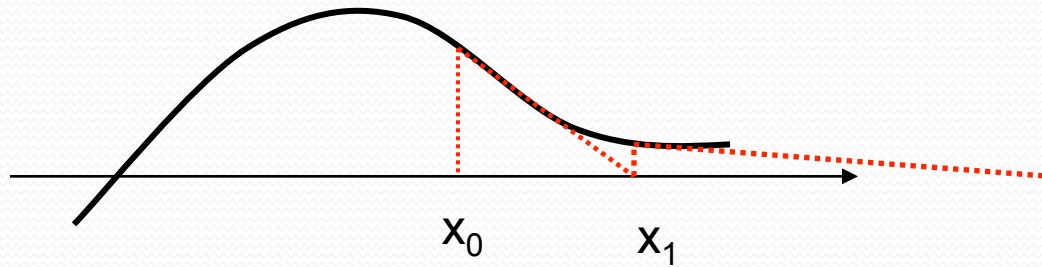
- Converges fast (quadratic convergence), if it converges.
- Requires only one guess

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

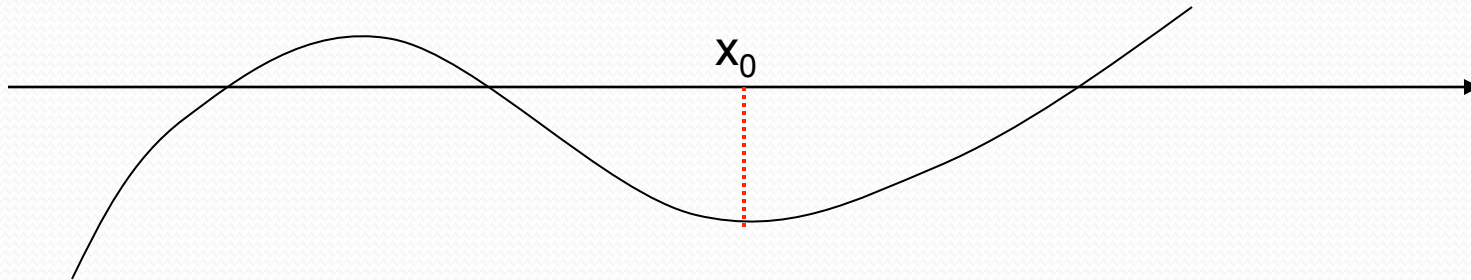
- Runaway -



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

Problems with Newton's Method

- Flat Spot -

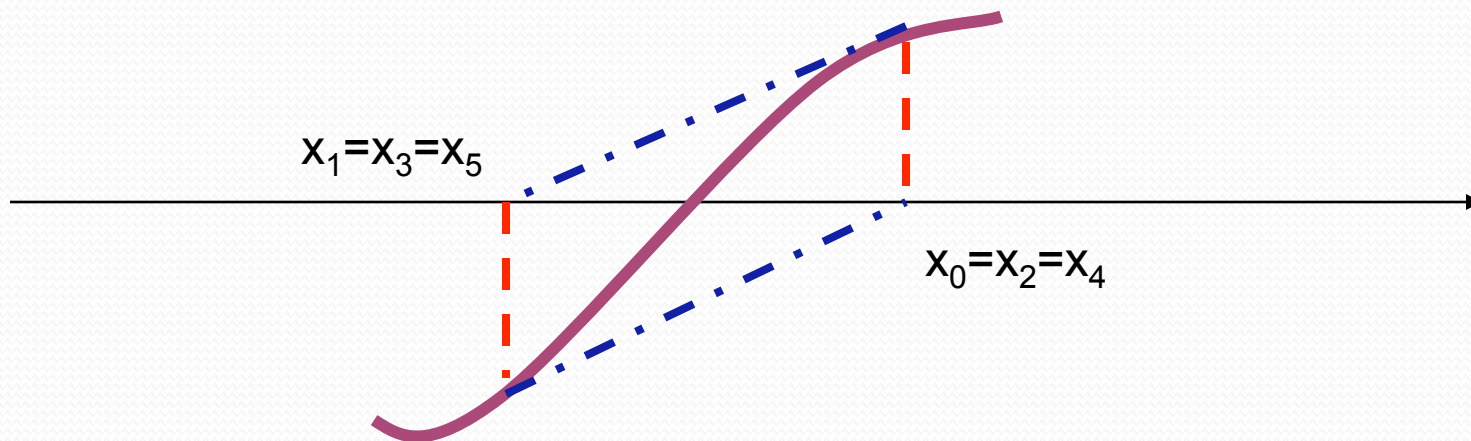


The value of $f'(x)$ is zero, the algorithm fails.

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

Secant Method

Newton's Method (Review)

*Assumptions : $f(x)$, $f'(x)$, x_0 are available,
 $f'(x_0) \neq 0$*

Newton's Method new estimate:

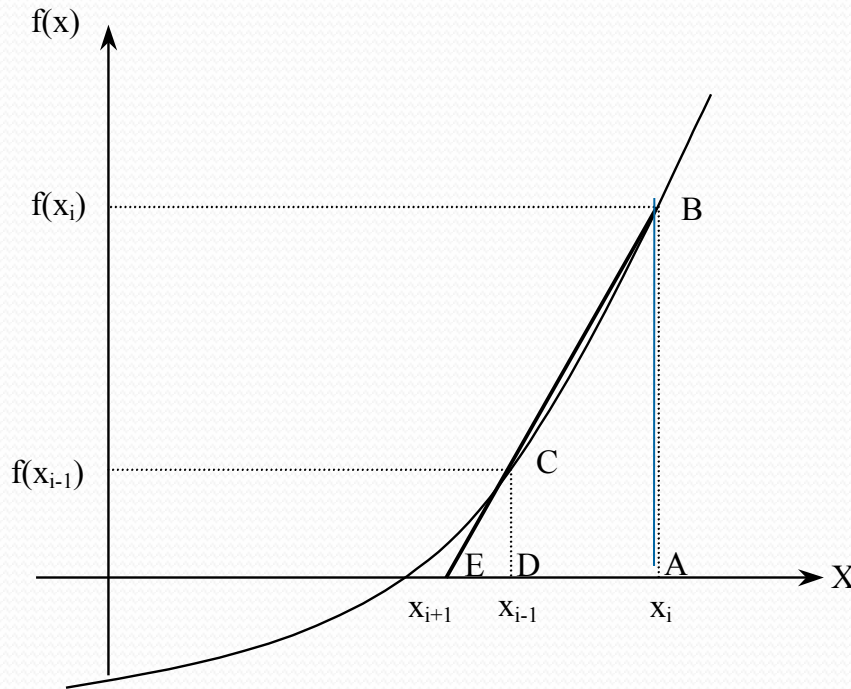
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Problem :

$f'(x_i)$ is not available,
or difficult to obtain analytically.

Secant Method – Derivation

The secant method can also be derived from geometry:



Geometrical representation of the Secant method.

The Geometric Similar Triangles

$$\frac{AB}{AE} = \frac{DC}{DE}$$

can be written as

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

On rearranging, the secant method is given as

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

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

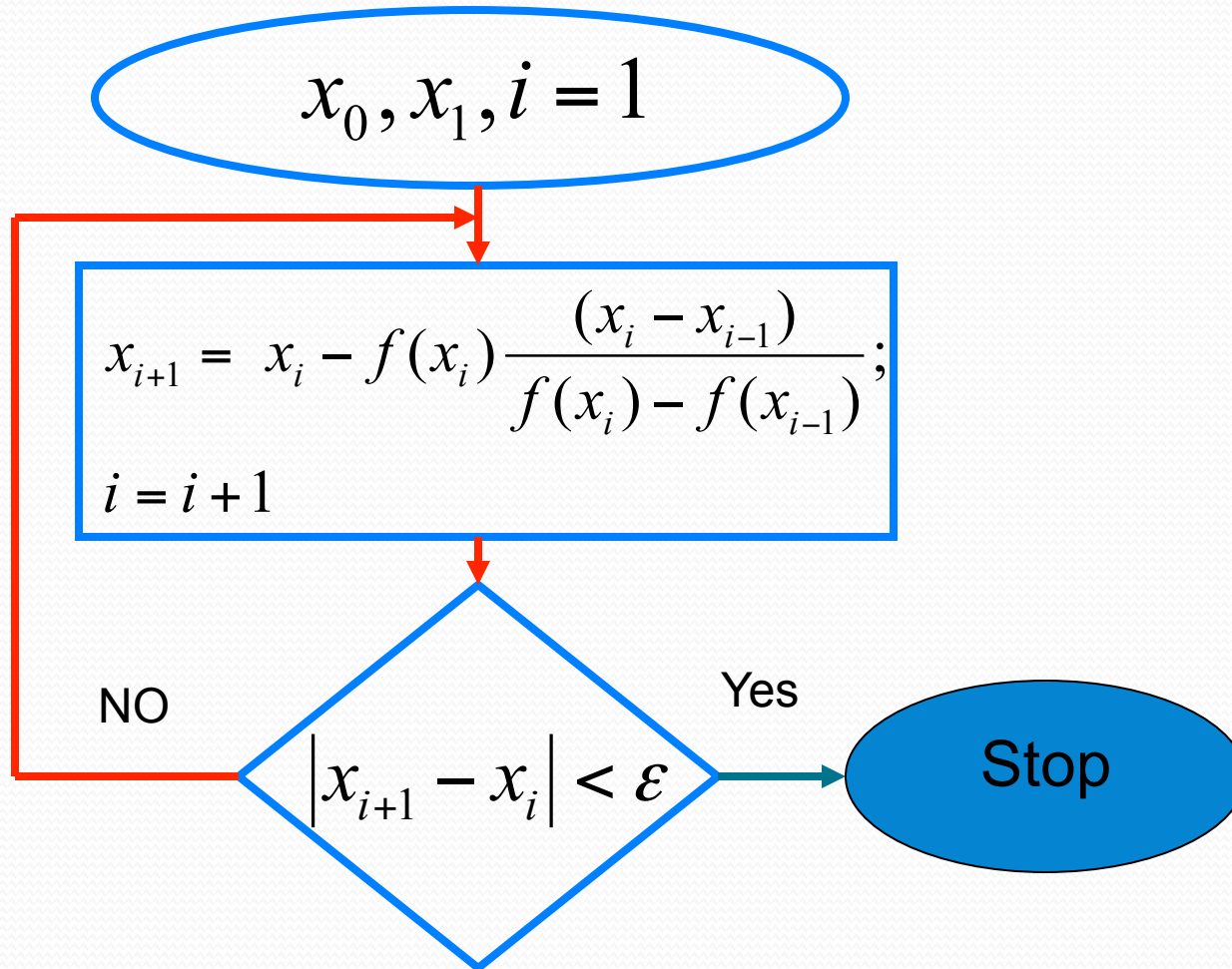
Assumptions :

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





Algorithm for Secant Method

Step 1

Calculate the next estimate of the root from two initial guesses

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

Find the absolute relative approximate error

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

Step 2

Find if the absolute relative approximate error is greater than the prespecified relative error tolerance.

If so, go back to step 1, else stop the algorithm.

Also check if the number of iterations has exceeded the maximum number of iterations.

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^{th} iteration.

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

Advantages of secant method:

It converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.

- It does not require use of the derivative of the function, something that is not available in a number of applications.
- It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

Disadvantages of secant method

1. It may not converge.
2. There is no guaranteed error bound for the computed iterates.
3. It is likely to have difficulty if $f'(\alpha) = 0$. This means the x-axis is tangent to the graph of $y = f(x)$ at $x = \alpha$.
4. Newton's method generalizes more easily to new methods for solving simultaneous systems of nonlinear equations.

Comparison of Root Finding Methods

- Advantages/disadvantages

COMPARISON OF NEWTON'S METHOD WITH REGULA-FALSI METHOD

Regula-Falsi is surely convergent while Newton's method is conditionally convergent. But once Newton's method converges, it converges faster.

In the Falsi method, we calculate only one more value of the function at each step i.e., $f(x^{(n)})$ while in Newton's method, we require two calculations $f(x_n)$ and $f'(x_n)$ at each step.

\therefore Newton's method generally requires fewer iterations but also requires more time for computation at each iteration.

When $f'(x)$ is large near the root the correction to be applied is smaller in the case of Newton's method which is then preferred. If $f'(x)$ is small near the root, the correction to be applied is large and the curve becomes parallel to the x -axis.

In this case the Regula-Falsi method should be applied.

COMPARISON OF ITERATIVE METHODS

1. Convergence in the case of the Bisection method is slow but steady. It is the simplest method and never fails.
2. The method of false position is slow and it is I order convergent. Convergence is guaranteed.
3. Newton's method has the fastest rate of convergence. This method is quite sensitive to starting value. It may diverge if $f'(x) = 0$ during iterative cycle.
4. For locating complex roots, the bisection method cannot be applied. Newton's and Muller's methods are effective.
5. If all the roots of a given equation are required, Lin-Bairstow's method is recommended. After a quadratic factor has been found, this method must be applied on the reduced polynomial.

Comparison of the Secant and False-position method

- Both methods use the **same expression** to compute x_r .

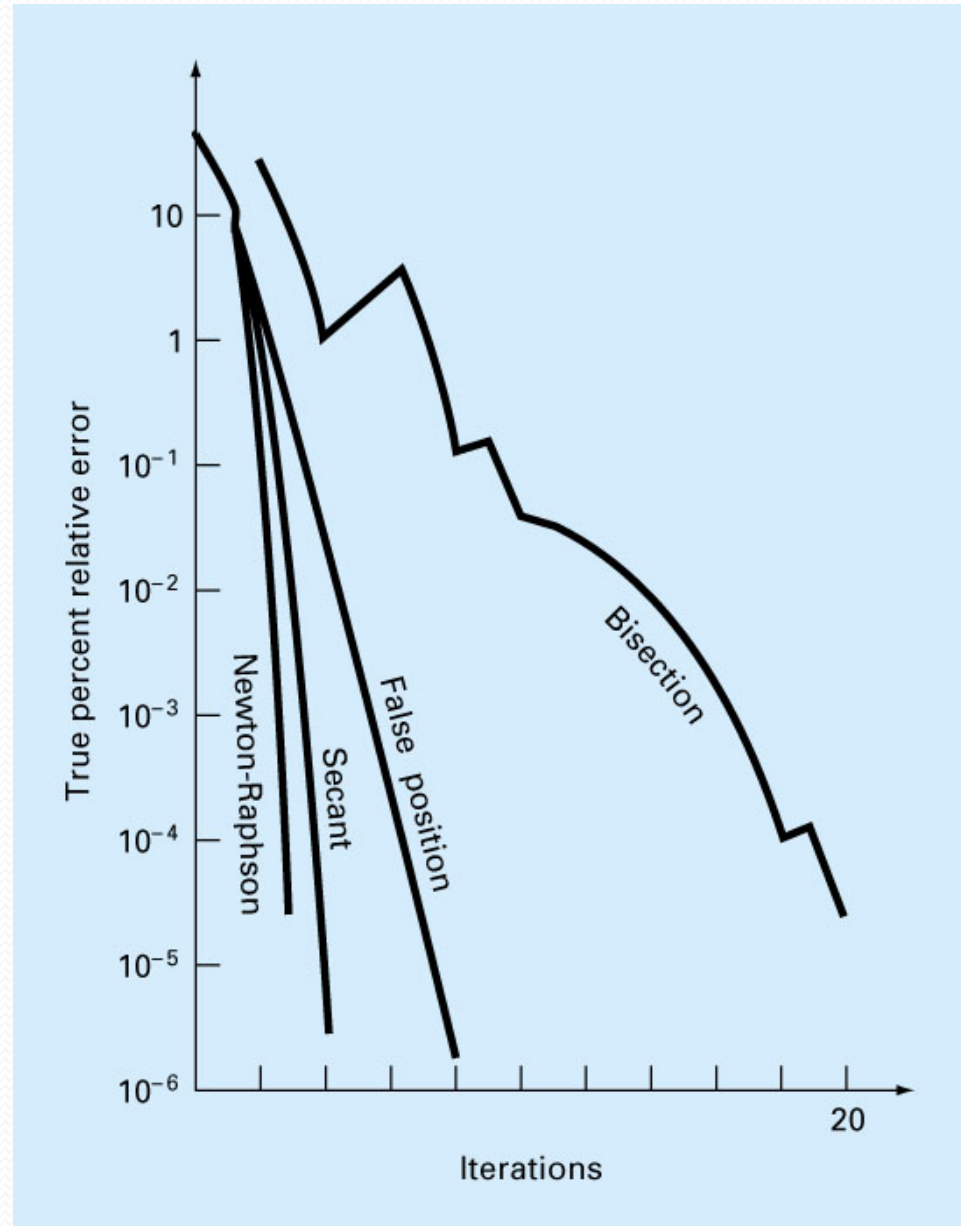
Secant :
$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

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

- They have **different methods** for the replacement of the initial values by the new estimate.

Comparison of the different Root finding method

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



Difficulties when we have multiple roots

- Bracketing methods **do not work for even multiple roots**.
- $f(\alpha) = f'(\alpha) = 0$, so both $f(x_i)$ and $f'(x_i)$ approach zero near the root. This could result in **division by zero**. A zero check for $f(x)$ should be incorporated so that the computation stops before $f'(x)$ reaches zero.
- For multiple roots, Newton-Raphson and Secant methods converge **linearly**, rather than quadratic convergence.

Summary

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



THANKYOU