Numerical Methods Lab
(PCCS-391)
EE-I & EE-II

Dr. A. De
Dr. P. Panja
Mrs. S. Mandal

Chapter 1

Root Finding Method

1.1 Introduction

In this chapter we are going to learn numerical Solutions of non linear(Algebraic and Transcendental) equations. The methods are bisection, Regula-Falsi and Newton-Raphson Method. Generally when analytical/mathematical methods fail to give a solution or it is very difficult to find a real root of nonlinear equation then we apply the numerical methods. The process of finding numerical solution of nonlinear equation $f(x) = 0$ involves finding a location of a point $x = \alpha$ (if α is a real and simple root of $f(x) = 0$) on x-axis where the graph of the function $f(x)$ intersects the x-axis.

1.2 Bisection Method

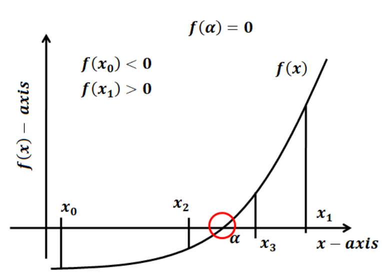
Let us consider an equation of the form $f(x) = 0$.The method of finding a real and simple roots of an equation involve the following:

Step-1. First of all find an interval $[x_0, x_1]$ in which a real and simple root of the equation $f(x) = 0$ exists (This is generally done either by graphical method or by finding two points a and b such that either $f(x_0) > 0, f(x_1) < 0$ or $f(x_0) < 0, f(x_1) > 0$ or precisely $f(x_0).f(x_1) < 0$).

Step-2. Find the point find the midpoint x_2 of $[x_0, x_1]$ by $x_2 = \frac{x_0+x_1}{2}$.

Step-3. If $f(c)$=near to zero then the root is x_2 and exit the process.

Step-4. If $f(x_2)$ is not sufficiently close to zero and if $f(x_2).f(x_0) < 0$ then the root lies in $[x_0, x_2]$ and set $x_1 = x_2$ and repeat **Step 2**. if $f(x_0).f(x_2) > 0$ then the root lies in $[x_2, x_1]$ and set $x_0 = x_2$ and repeat **Step 2**.



1.2.1 Working formula

The iteration formula for bisection method is

$$c_n = \frac{a_n + b_n}{2} \tag{1.1}$$

1.2.2 Algorithm

Step-1. Start

Step-2. Define $f(x)$

Step-3. Read a, b, e # $[a, b]$ *is the initial interval and* e *is the error limits*

Step-4. do *until* $\mid f(c) \mid < e$
$c \leftarrow \frac{a+b}{2}$
if $f(a) * f(c) < 0$ then
$b \leftarrow c$
else
$a \leftarrow c$

Step-5. Print c

Step-6. Stop

1.2.3 Assignment

Write a Code in C(or MATLAB or PYTHON) to implement the Bisection Method and use it to find a root of $3x - \cos x - 1 = 0$, between $x = 0.0$ and $x = 1.0$ correct upto 3 decimal places.

1.2.4 Code in C

```
#include<stdio.h>
#include<math.h>
#define f(x) (3*x-cos(x)-1.0)
int main(){
            float a,b,c,e;
            printf(''Enter the values of a,b,e'');
            scanf(''%f%f%f'',&a,&b,&e);
            do
                {
                c=(a+b)/2;
                if(f(a)*f(c)<0)
                    {
                    b=c;
                    }
                else
                    {
                    a=c;
                    }

                } while(fabs(f(c))>e);
```

```
        printf(''The root of the equation is %.3f'',c);
    return 0;
    }
```

1.2.5 Input/Output

```
Enter the values of a,b,e
0
1
0.0001
The root of the equation is 0.607
```

1.2.6 Advantages and Disadvantages

The advantages of bisection method are

- It is a simple method
- One function needed to evaluate in each iteration.
- Size of the interval containing the root of the equation is reduced by 50% in each iteration.
- The bisection method is always convergent. Since the method brackets the root, the method is guaranteed to converge.
- The error of approximation can be controlled.

The disadvantages of bisection method are

- This method is very slow.
- Since the iteration formula does not depend on the function, to get a value with higher accuracy, large number of iteration is needed.

1.3 Regula Falsi Method

Let us consider an equation of the form $f(x) = 0$.The Regula-Falsi method of finding a real and simple roots of an equation involve the following:

Step-1. First of all find an interval $[x_0, x_1]$ in which a real and simple root of the equation $f(x) = 0$ exists (This is generally done either by graphical method or by finding two points ax_0 and x_1 such that either $f(x_0) > 0, f(x_1) < 0$ or $f(x_0) < 0, f(x_1) > 0$ or precisely $f(x_0).f(x_1) < 0$).

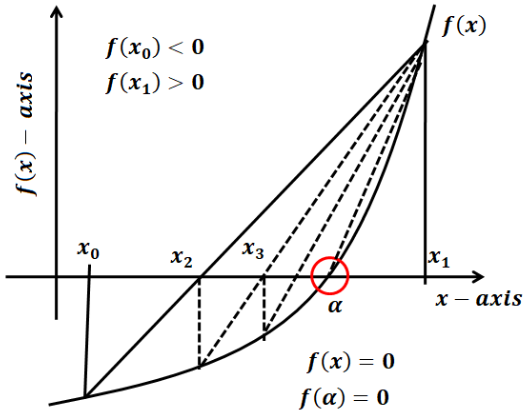
Step-2. Draw a chord (straight line) passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

Step-3. Find the point x_2, where the straight line meets the x-axis.

Step-4. From the point x_2 find the ordinate $f(x_2)$

Step-5. If $f(x_2)$=near to zero then the root is x_2 and exit the process.

Step-6. If $f(x_2)$ is not sufficiently close to zero and if $f(x_2).f(x_0) < 0$ then the root lies in $[x_0, x_2]$ and set $x_1 = x_2$ and repeat **Step 2**. if $f(x_0).f(x_2) > 0$ then the root lies in $[x_2, x_1]$ and set $x_0 = x_2$ and repeat **Step 2**.



1.3.1 Working formula

There are two approaches to find the iteration formula for Regula-Falsi method **Trigonometric Approach** and **Geometric Approach**.

Trigonometric Approach(Property of similar triangle) Here, Figure 1.1 graphically represents the Regula Falsi Method. According to the figure with the information about the points $x_0(A)$ and $x_1(B)$ we need to find the point $x_2(C)$. Precisely, we need to find the distance AC from two similar triangle ΔAPC and ΔBCQ. Since ΔAPC and ΔBCQ are similar so

$$\frac{\overline{AP}}{\overline{AC}} = \frac{\overline{BQ}}{\overline{CB}} \text{ or, } \frac{\overline{AP}}{\overline{AC}} = \frac{\overline{BQ}}{\overline{AB} - \overline{AC}} \text{ or, } \frac{\overline{AB} - \overline{AC}}{\overline{AC}} = \frac{\overline{BQ}}{\overline{AP}} \text{ or, } \frac{\overline{AB}}{\overline{AC}} - 1 = \frac{\overline{BQ}}{\overline{AP}}$$

$$. \quad \text{or, } \frac{\overline{AB}}{\overline{AC}} = \frac{\overline{BQ} + \overline{AP}}{\overline{AP}} \text{ or, } \overline{AC} = \frac{\overline{AB} \times \overline{AP}}{\overline{AP} + \overline{BQ}} \text{ or, } \overline{AC} = \frac{(x_1 - x_0) \mid f(x_0) \mid}{\mid f(x_0) \mid + \mid f(x_1) \mid}$$

$$\text{or, } \quad x_2 = \overline{OC} = \overline{OA} + \overline{AC} = x_0 + \frac{(x_1 - x_0) \mid f(x_0) \mid}{\mid f(x_0) \mid + \mid f(x_1) \mid} = \frac{x_1 \mid f(x_0) \mid + x_0 \mid f(x_1) \mid}{\mid f(x_0) \mid + \mid f(x_1) \mid}$$

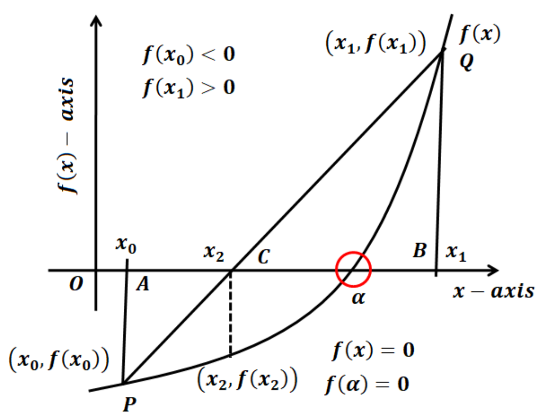


Figure 1.1: Graphical Representation of Regula-Falsi Method

Geometric Approach:

Here we first find the equation of the straight line passing through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ which is given by

$$\frac{y - f(x0)}{f(x_1) - f(x_0)} = \frac{x - x_0}{x_1 - x_0} \tag{1.2}$$

. This line (1.2) meets the $x-$ axis at $x = x_2$(say). Since on x-axis, $y = 0$ so

$$\frac{0 - f(x0)}{f(x_1) - f(x_0)} = \frac{x_2 - x_0}{x_1 - x_0}$$
$$\text{or,} \quad x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0) = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

So, the iteration formula for Regula Falsi method in 1st approach is given by

$$x_{n+2} = \frac{x_{n+1} \mid f(x_n) \mid + x_n \mid f(x_{n+1}) \mid}{\mid f(x_n) \mid + \mid f(x_{n+1}) \mid} \tag{1.3}$$

and in the second approach is given by

$$x_{n+2} = \frac{x_n f(x_{n+1}) - x_{n+1} f(x_n)}{f(x_{n+1}) - f(x_n)} \tag{1.4}$$

1.3.2 Algorithm

Step-1. Start

Step-2. Define $f(x)$

Step-3. Read x_0, x_1, e # $[x_0, x_1]$ *is the initial interval and e is the error limits*

Step-4. do *until* $\mid f(x_2) \mid < e$
$x_2 \leftarrow \frac{x_1 |f(x_0)| + x_0 |f(x_1)|}{|f(x_0)| + |f(x_1)|}$
if $f(x_0) * f(x_2) < 0$ then
$x_1 \leftarrow x_2$
else
$x_0 \leftarrow x_2$

Step-5. Print x_2

Step-6. Stop

1.3.3 Assignment

Write a Code in C(or MATLAB or PYTHON) to implement the Regula Falsi Method and use it to find a root of $3x - \cos x - 1 = 0$, between $x = 0.0$ and $x = 1.0$ correct upto 4 decimal places.

1.3.4 Code in C

```
#include<stdio.h>
#include<math.h>
#define f(x) (3*x-cos(x)-1.0)
int main(){
            float x0,x1,x2,e;
            printf(``Enter the values of x0,x1,e'');
            scanf(``%f%f%f'',&x0,&x1,&e);
            do
                {
                x2=(x1*fabs(f(x0))+x0*fabs(f(x1))/(fabs(f(x0))+fabs(f(x1)));
```

```
        if(f(x0)*f(x2)<0)
            {
            x1=x2;
            }
        else
            {
            x0=x2;
            }

        } while(fabs(f(x2))>e);
        printf(``The root of the equation is %.4f'',x2);
    return 0;
    }
```

1.3.5 Input/Output

```
Enter the values of x0,x1,e
0
1
0.00001
The root of the equation is 0.6071
```

1.3.6 Advantages and Disadvantages

The advantages of Regula-Falsi method are

- The method do not require derivative of the function
- The method is always convergent.
- The method has linear rate of convergence.

The disadvantages of Regula Falsi method are

- This method is also very slow.
- Initial intervals should be chosen closer to the root for faster convergence.

1.4 Newton Raphson Method

This iterative method of determining the roots of the function is termed after **Issac Newton** and **Joseph Raphson**. According to this method, the X-intercept of the tangent drawn at the initial guess point is the better approximation for the root of the function.
Let us consider an equation of the form $f(x) = 0$.The Newton-Raphson method of finding a real and simple roots of an equation involve the following:

Step-1. First of all find a point x_0 on x-axis (called initial guess)

Step-2. Draw a tangent to the curve $y = f(x)$ at the point$(x_0, f(x_0))$.

Step-3. Find the point x_1 (1st approximation), where the tangent to the curve line meets the x-axis.

Step-4. From the point x_1 find the ordinate $f(x_1)$

Step-5. If $f(x_1)$=near to zero then the root is x_1 and exit the process.

Step-6. If $f(x_1)$ is not sufficiently close to zero, then set $x_0 = x_1$ and repeat **Step 2**.

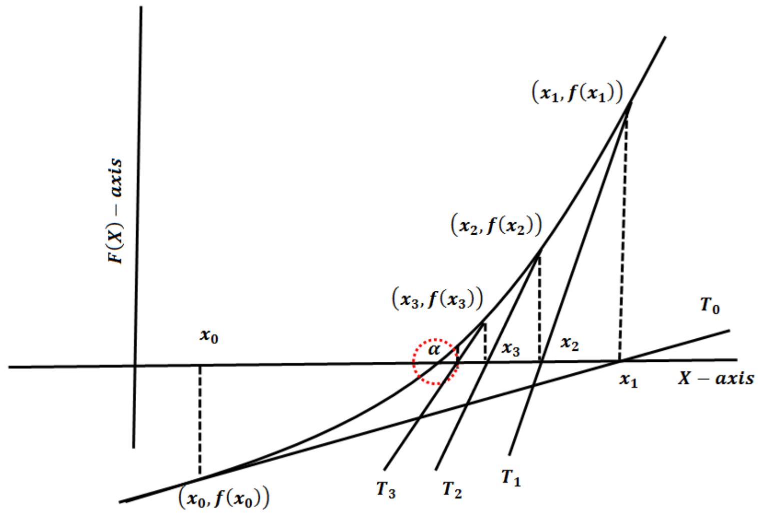


Figure 1.2: Graphical Representation of Newton Raphson's Method

1.4.1 Working formula

There are two approaches to find the iteration formula for Newton-Raphson method **Geometric Approach** and **Calculus approach**.

Geometric Approach(Equation of a tangent to a curve)
Here, Figure 1.3 graphically represents the Newton-Raphson's Methods. According to the figure we shall take an initial approximation to the root of $f(x) = 0$ as x_0. Now we shall find the tangent drawn at the point $(x_0, f(x_0))$, where slope of the tangent at the point is $f'(x_0)$. So the equation of the tangent is given by

$$(y - f(x_0)) = f'(x_0)(x - x_0). \tag{1.5}$$

Let the intercept of the tangent with the x-axis be $x = x_1$. As, $y = 0$ on x-axis, so putting $y = 0$, and $x = x_1$ in (1.5) we obtain the second approximation to the root as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.6}$$

provided $f'(x_0) \neq 0$

If x_1 is not the root or $f(x_1)$ is not sufficiently close to zero, we shall draw tangent again at $(x_1, f(x_1)$ and proceeding in the same way, step by step we obtain the next approximations x_2,x_3, $\cdots$ as

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \quad x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}, \cdots \tag{1.7}$$

Differential calculus Approach:
Here also we shall take the initial approximation to the root as x_0. If x_0 is not the root or $f(x_0)$ is not sufficiently close to zero, we assume that $x_1 = x_0 + h$ is the root. So, $f(x_1) = f(x_0 + h) = 0$. Here, if we

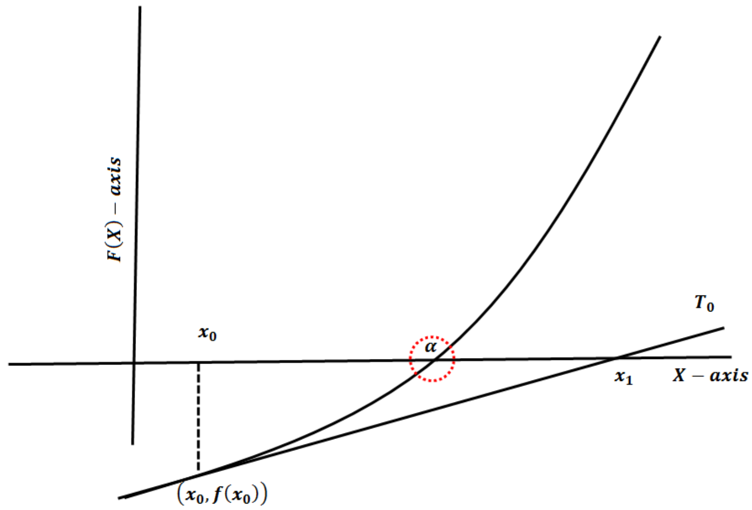


Figure 1.3: Graphical Representation of Newton-Raphson Method

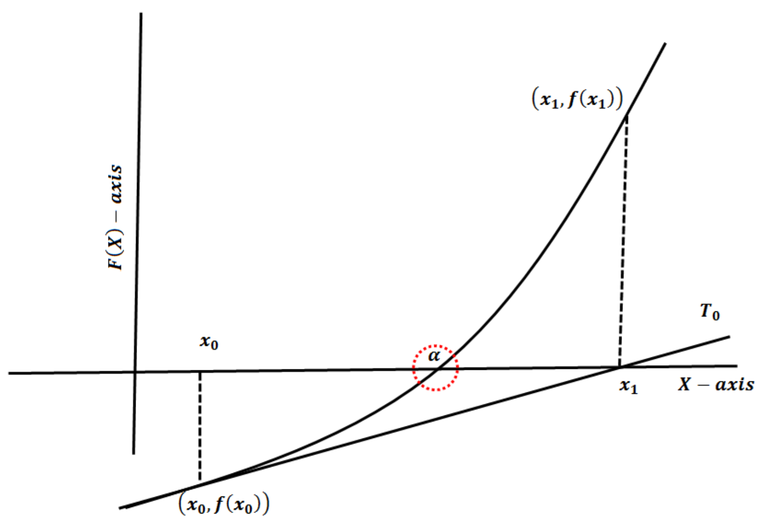


Figure 1.4: Graphical Representation of Newton-Raphson Method

can find h, then we obtain x_1 easily. Now by Taylor's theorem we have

$$\begin{aligned} 0 = f(x_1) = f(x_0 + h) &= f(x_0) + \frac{hf'(x_0)}{1!} + \frac{h^2 f''(x_0)}{2!} + \frac{h^3 f'''(x_0)}{3!} + \cdots \\ \Rightarrow 0 &= f(x_0) + hf'(x_0), \quad \text{[neglecting 2nd and higher degree of small values]} \\ \Rightarrow h &= -\frac{f(x_0)}{f'(x_0)} \end{aligned}$$

Therefore the 1st approximation x_1 is given by

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.8}$$

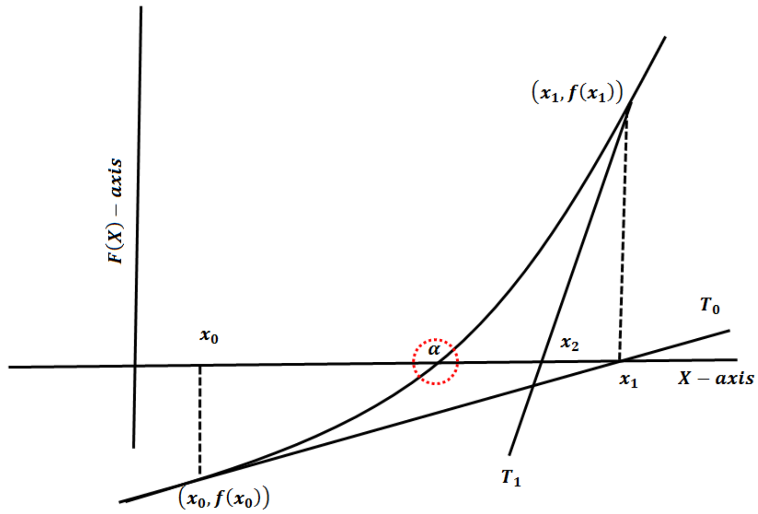


Figure 1.5: Graphical Representation of Newton-Raphson Method

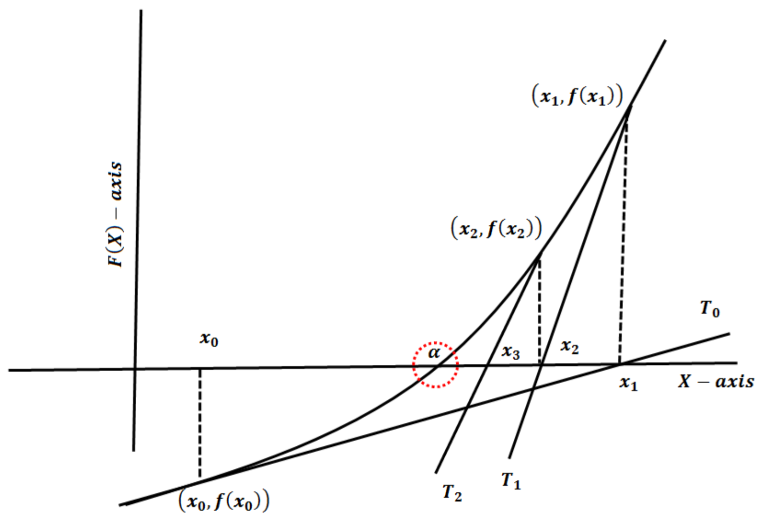


Figure 1.6: Graphical Representation of Newton-Raphson Method

If x_1 is not the root or $f(x_1)$ is not sufficiently close to zero, we shall proceeding in the same way to obtain x_2,x_3, $\cdots$ as

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \quad x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}, \cdots \tag{1.9}$$

So, the iteration formula for Newton Raphson method is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{1.10}$$

Provided the $f'(x) \neq 0$ exists.

1.4.2 Algorithm

Step-1. Start

Step-2. Define $f(x)$

Step-3. Define $df(x)$ # *derivative of f(x)*

Step-4. Read x_0, e # x_0 *is the initial interval and* e *is the error limits*

Step-5. do *until* $| f(x_1) | < e$
$x_1 \leftarrow x_0 - f(x_0)/df(x_0)$
$x_0 \leftarrow x_1$

Step-6. Print x_1

Step-7. Stop

1.4.3 Assignment

Write a Code in C(or MATLAB or PYTHON) to implement the Newton-Raphson Method and use it to find a root of $3x - \cos x - 1 = 0$, near to 0 correct upto 4 decimal places.

1.4.4 Code in C

```
#include<stdio.h>
#include<math.h>
#define f(x) (3*x-cos(x)-1.0)
#define df(x) (3+sin(x))
int main(){
        float x0,x1,e;
        printf(``Enter the values of x0,e'');
        scanf(``%f%f'',&x0,&e);
        do
          {
          x1=x0-f(x0)/df(x0);
          x0=x1;
          } while(fabs(f(x1))>e);
          printf(``The root of the equation is %.4f'',x1);
        return 0;
        }
```

1.4.5 Input/Output

```
Enter the values of x0,e
0
0.00001
The root of the equation is 0.6071
```

1.4.6 Advantages and Disadvantages

The advantages of Newton-Raphson method are

- Its a self starting method
- The method has faster than Bisection or Regula-Falsi method.
- Its an open ended method

The disadvantages of Newton-Raphson method are

- The method require derivative of the function, which is sometimes complicated to calculate.
- The method is conditionally convergent.
- Initial approximation should be chosen closer to the root for faster convergence.
- The method fails if derivative of $f(x)$ becomes zero at some point

Chapter 2

Interpolation with Equal & Unequal Intervals

2.1 Introduction

Suppose a function $f(x)$ is known at only $(n+1)$ points (called arguments) $x_0, x_1, \cdots, x_n$. There is no other information available about $f(x)$. That is, the only information we have is

$$f(x_i) = f_i = y_i \quad , i = 0, 1, 2, \cdots, n. \tag{2.1}$$

The problem of interpolation is to compute the value of $f(x)$, at least approximate for an argument $x = \alpha$ (say) which is not among $x_0, x_1, \cdots, x_n$.
The term **Interpolation** is used when $x = \alpha$ lies between the smallest (x_0) and largest (x_n) of the interpolating points $x_0, x_1, \cdots, x_n$ and the term **Extrapolation** is used when $x = \alpha$ lies slightly outside the interpolating points.
In this chapter we are going to code three different interpolation formula namely,

(i) Newton's Forward Interpolation formula

(ii) Newton's Backward Interpolation formula

(iii) Lagrange's Interpolation formula.

2.2 Newton's Forward Interpolation Formula

If the given arguments $x_0, x_1, \cdots, x_n$ are equi-spaced and the unknown value $x = \alpha$ lies towards the beginning of the arguments, that is between x_0 and x_1 we generally apply the forward interpolation formula.

2.2.1 Working Formula

The working formula for Newton's forward interpolation is

$$f(\alpha) = f(x_0) + \frac{u}{1!}\Delta f(x_0) + \frac{u(u-1)}{2!}\Delta^2 f(x_0) + \cdots + \frac{u(u-1)(u-2)\cdots(u-n+1)}{n!}\Delta^n f(x_0) \tag{2.2}$$

where $u = \frac{\alpha - x_0}{h}$, $h =$ Step-length, where $x_i = x_0 + ih$
$\alpha =$ unknown argument
$\Delta^i f(x_0) = i-$th forward difference
$n + 1 =$ Number of arguments.

2.2.2 Algorithm

Step-1. Start

Step-2. Read n # *n=number of arguments-1*

Step-3. Read α # α= *unknown argument*

Step-4. for $i = 0(1)n$,
read x_i's # x_i = *interpolating point*

Step-5. for $i = 0(1)n$,
read f_i's # $f_i = f(x_i)$ *values of the function at* x_i*'s*

Step-6. for $i = 0(1)n$,
set $t_{i,0} \leftarrow f_i$ # *initialize the first column of the difference table* $t_{i,j}$

Step-7. for $j = 1(1)n$ and
for $i = 0(1)n - j$
$t_{i,j} \leftarrow t_{i+1,j-1} - t_{i,j-1}$ # *calculate the difference table column wise*

Step-8. Set $S \leftarrow t_{0,0}$ and $p \leftarrow 1.0$

Step-9. Set $h \leftarrow x_1 - x_0$ # *calculate the step length between the arguments*

Step-10. Set $u \leftarrow \frac{\alpha - x_0}{h}$

Step-11. for $i = 1(1)n$
Set $p \leftarrow p * \frac{(u-i+1)}{i}$
Set $S \leftarrow S + p * t_{0,i}$ # $t_{0,i}$ *are the terms of the first row of the difference table*

Step-12. Print s

Step-13. Stop

2.2.3 The Difference Table

Table 2.1: The Forward Difference Table

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
x_0	$f(x_0)$	$\Delta f(x_0) = f(x_1) - f(x_0)$	$\Delta^2 f(x_0) = \Delta f(x_1) - \Delta f(x_0)$	$\Delta^3 f(x_0) = \Delta^2 f(x_1) - \Delta^2 f(x_0)$
x_1	$f(x_1)$	$\Delta f(x_1) = f(x_2) - f(x_1)$	$\Delta^2 f(x_1) = \Delta f(x_2) - \Delta f(x_1)$	
x_2	$f(x_2)$	$\Delta f(x_2) = f(x_3) - f(x_2)$		
x_3	$f(x_3)$			

Table 2.2: The Table $t_{i,j}$ for Computation

$t_{0,0}$	$t_{0,1}$	$t_{0,2}$	$t_{0,3}$
$t_{1,0}$	$t_{1,1}$	$t_{1,2}$	
$t_{2,0}$	$t_{1,2}$		
$t_{3,0}$			

Note: Here the "for" loops to be taken as "≤" type

2.2.4 Assignment-4

Write a code in C/PYTHON/MATLAB for implementing Newton's Forward Interpolation formula and use it to find the value of $f(0.12)$ from the following table

x	0.10	0.15	0.20	0.25	0.30
$f(x_0)$	0.1003	0.1015	0.2027	0.2553	0.3039

2.2.5 Code in C

```
#include<stdio.h>
int main(){
        int i,j,k,n;
        float x[20],f[20],t[20][20],a,h,s,p,u;
        printf("Enter th evalue of n:");
        scanf("%d",&n);
        printf("Enter the unknown point a:");
        scanf("%f",&a);
        printf("Enter the arguments xi's:\n");
        for(i=0;i<=n;i++)
            {
            scanf("%f",&x[i]);
            }
        printf("Enter the functional values fi's:\n");
        for(i=0;i<=n;i++)
            {
            scanf("%f",&f[i]);
        }
        for(i=0;i<=n;i++)
            {
            t[i][0]=f[i];
        }
        for(j=1;j<=n;j++)
            {
            for(i=0;i<=n-j;i++)
                {
                t[i][j]=t[i+1][j-1]-t[i][j-1];
                }
        }
        s=t[0][0];
        p=1.0;
        h=x[1]-x[0];
        u=(a-x[0])/h;
        for(i=1;i<=n;i++)
            {
            p=p*(u-i+1)/i;
```

```
            s=s+p*t[0][i];
            }
        printf("The value is %f",s);
        return 0;
}
```

2.2.6 Input/Output

```
Enter th evalue of n: 4
Enter the unknown point a:0.12
Enter the arguments xi's:
0.10
0.15
0.20
0.25
0.30
Enter the functional values fi's:
0.1003
0.1511
0.2027
0.2553
0.3039
The value is 0.120753
```

2.3 Newton's Backward Interpolation Formula

If the given arguments $x_0, x_1, \cdots, x_n$ are equi-spaced and the unknown value $x = \alpha$ lies towards the end of the set of the arguments, that is between x_{n-1} and x_n we generally apply the forward interpolation formula.

2.3.1 Working Formula

The working formula for Newton's backward interpolation is

$$f(\alpha) = f(x_n) + \frac{v}{1!}\nabla f(x_n) + \frac{v(v+1)}{2!}\nabla^2 f(x_n) + \cdots + \frac{v(v+1)(v+2)\cdots(v+n-1)}{n!}\nabla^n f(x_0) \tag{2.3}$$

where $v = \frac{\alpha - x_n}{h}$, $h =$ Step-length, where $x_{n-i} = x_n - ih$
$\alpha =$ unknown argument
$\nabla^i f(x_n) = i-$th backward difference
$n+1 =$ Number of arguments. As we know that $\nabla^i f(x_n) = \Delta^i f(x_{n-i})$ In terms of forward notation,

$$f(\alpha) = f(x_n) + \frac{v}{1!}\Delta f(x_{n-1}) + \frac{v(v+1)}{2!}\Delta^2 f(x_{n-2}) + \cdots + \frac{v(v+1)(v+2)\cdots(v+n-1)}{n!}\Delta^n f(x_0) \tag{2.4}$$

2.3.2 Algorithm

Step-1. Start

Step-2. Read n *# n=number of arguments-1*

Step-3. Read α *#* α= *unknown argument*

Step-4. for $i = 0(1)n$,
read x_i's *#* x_i = *interpolating point*

Step-5. for $i = 0(1)n$,
read f_i's # $f_i = f(x_i)$ *values of the function at* x_i*'s*

Step-6. for $i = 0(1)n$,
set $t_{i,0} \leftarrow f_i$ # *initialize the first column of the difference table* $t_{i,j}$

Step-7. for $j = 1(1)n$ and
for $i = 0(1)n - j$
$t_{i,j} \leftarrow t_{i+1,j-1} - t_{i,j-1}$ # *calculate the difference table column wise*

Step-8. Set $S \leftarrow t_{n,0}$ and $p \leftarrow 1.0$

Step-9. Set $h \leftarrow x_1 - x_0$ # *calculate the step length between the arguments*

Step-10. Set $u \leftarrow \frac{\alpha - x_0}{h}$

Step-11. for $i = 1(1)n$
Set $p \leftarrow p * \frac{(v+i-1)}{i}$
Set $S \leftarrow S + p * t_{n-i,i}$ # $t_{n-i,i}$ *are the terms of the off diagonal elements of the difference table*

Step-12. Print s

Step-13. Stop

2.3.3 The Difference Table

Table 2.3: The Forward Difference Table

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
x_0	$f(x_0)$	$\Delta f(x_0) = f(x_1) - f(x_0)$	$\Delta^2 f(x_0) = \Delta f(x_1) - \Delta f(x_0)$	$\Delta^3 f(x_0) = \Delta^2 f(x_1) - \Delta^2 f(x_0)$
x_1	$f(x_1)$	$\Delta f(x_1) = f(x_2) - f(x_1)$	$\Delta^2 f(x_1) = \Delta f(x_2) - \Delta f(x_1)$	
x_2	$f(x_2)$	$\Delta f(x_2) = f(x_3) - f(x_2)$		
x_3	$f(x_3)$			

Table 2.4: The Table $t_{i,j}$ for Computation

$t_{0,0}$	$t_{0,1}$	$t_{0,2}$	$t_{0,3}$
$t_{1,0}$	$t_{1,1}$	$t_{1,2}$	
$t_{2,0}$	$t_{2,1}$		
$t_{3,0}$			

Note: Here the “for” loops to be taken as “≤” type

2.3.4 Assignment-6

Write a code in C/PYTHON/MATLAB for implementing Newton’s Backward Interpolation formula and use it to find the value of $f(0.29)$ from the following table

x	0.10	0.15	0.20	0.25	0.30
$f(x_0)$	0.1003	0.1015	0.2027	0.2553	0.3039

2.3.5 Code in C

```
#include<stdio.h>
int main(){
        int i,j,k,n;
        float x[20],f[20],t[20][20],a,h,s,p,u;
        printf("Enter th evalue of n:");
        scanf("%d",&n);
        printf("Enter the unknown point a:");
        scanf("%f",&a);
        printf("Enter the arguments xi's:\n");
        for(i=0;i<=n;i++)
            {
            scanf("%f",&x[i]);
            }
        printf("Enter the functional values fi's:\n");
        for(i=0;i<=n;i++)
            {
            scanf("%f",&f[i]);
        }
        for(i=0;i<=n;i++)
            {
            t[i][0]=f[i];
        }
        for(j=1;j<=n;j++)
            {
            for(i=0;i<=n-j;i++)
                {
                t[i][j]=t[i+1][j-1]-t[i][j-1];
                }
        }
        s=t[n][0];
        p=1.0;
        h=x[1]-x[0];
        u=(a-x[n])/h;
        for(i=1;i<=n;i++)
            {
            p=p*(v+i-1)/i;
```

```
            s=s+p*t[n-i][i];
            }
        printf("The value is %f",s);
        return 0;
}
```

2.3.6 Input/Output

```
Enter th evalue of n: 4
Enter the unknown point a:0.29
Enter the arguments xi's:
0.10
0.15
0.20
0.25
0.30
Enter the functional values fi's:
0.1003
0.1511
0.2027
0.2553
0.3039
The value is 0.294915
```

2.4 Lagrange's Interpolation Formula

If the given arguments $x_0, x_1, \cdots, x_n$ not necessarily equi-spaced and the unknown value $x = \alpha$ lies at anywhere between the set of the arguments, that is between x_0 and x_n we generally apply the Lagrange's interpolation formula.

2.4.1 Working Formula

The working formula for Lagrange's interpolation is

$$\begin{aligned} f(\alpha) &= \frac{(\alpha - x_1)(\alpha - x_2)\cdots(\alpha - x_n)}{(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n)} f(x_0) + \frac{(\alpha - x_0)(\alpha - x_2)\cdots(\alpha - x_n)}{(x_1 - x_0)(x_1 - x_2)\cdots(x_1 - x_n)} f(x_1) \\ &+ \frac{(\alpha - x_0)(\alpha - x_1)\cdots(\alpha - x_n)}{(x_2 - x_0)(x_2 - x_1)\cdots(x_2 - x_n)} f(x_2) + \cdots + \frac{(\alpha - x_0)(\alpha - x_1)\cdots(\alpha - x_{n-1})}{(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})} f(x_n) \end{aligned}$$

In compact form we write

$$f(x) = \sum_{r=0}^{n} \frac{\omega(x)}{(x - x_r)\omega'(x_r)} f(x_r)$$

$\omega(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$ Here $n + 1$ = Number or arguments.
α = Unknown argument
x_i = Interpolating Points or arguments
f_i = The functional values or entries

2.4.2 Algorithm

Step-1. Start

Step-2. Read n # *n=number of arguments-1*

Step-3. Read α # α= *unknown argument*

Step-4. for $i = 0(1)n$,
read x_i's # x_i = *interpolating point*

Step-5. for $i = 0(1)n$,
read f_i's # $f_i = f(x_i)$ *values of the function at* x_i*'s*

Step-6. Set $s = 0.0$

Step-7. for $i = 0(1)n$
Set $p = 1.0$
for $j = 0(1)n$
if $i \neq j$
$p \leftarrow p * \frac{a-x_j}{(x_i-x_j)}$
Set $s \leftarrow s + p * f(x_i)$

Step-8. Print s

Step-9. Stop

2.4.3 Assignment-6

Write a code in C/PYTHON/MATLAB for implementing Lagrange's Interpolation formula and use it to find the value of $f(0.656)$ from the following table

x	0.654	0.658	0.659	0.661
$f(x)$	2.8156	2.8182	2.8189	2.8202

2.4.4 Code in C

```
#include<stdio.h>
int main(){
    int i,j,k,n;
    float x[20],f[20],a,s,p;
    printf("Enter th evalue of n:");
    scanf("%d",&n);
    printf("Enter the unknown point a:");
    scanf("%f",&a);
    printf("Enter the arguments xi's:\n");
    for(i=0;i<=n;i++)
        {
        scanf("%f",&x[i]);
        }
    printf("Enter the functional values fi's:\n");
    for(i=0;i<=n;i++)
```

```
        {
        scanf("%f",&f[i]);
        }
    s=0.0;
    for(i=0;i<=n;i++)
        {
        p=1.0;
        for(j=0;j<=n;j++)
            {
            if(i!=j)
            {
            p=p*(a-x[j])/(x[i]-x[j]);
            }
            }
        s=s+p*f[i];
        }
    printf("The value is %f",s);
    return 0;
    }
```

2.4.5 Input/Output

```
Enter th evalue of n: 3
Enter the unknown point a:0.656
Enter the arguments xi's:
0.654
0.658
0.659
0.661
Enter the functional values fi's:
2.8156
2.8182
2.8189
2.8202
The value is 2.816814
```

Chapter 3

Numerical Integration

3.1 Trapezoidal Rule

3.1.1 Working Formula

The simple Trapezoidal rule for the interval $[a, a+h]$ is given by

$$\int_a^b f(x)dx = \int_a^{a+h} f(x)dx = \frac{h}{2}[f(a) + f(a+h)]$$

For n number of intervals $[a, a+nh]$ the composite rule will be given by

$$\int_a^b f(x)dx = \int_a^{a+nh} f(x)dx = \frac{h}{2}[f(a) + f(a+nh) + 2(f(a+h) + f(a+2h) + \cdots + f(a+nh))]$$

3.1.2 Algorithm

Step-1. Start

Step-2. define f(x)

Step-3. Read a, b, n # *a=lower limit, b=upper limit, n number of intervals*

Step-4. Set $h \leftarrow (b-a)/n$

Step-5. Set $s \leftarrow 0.0$

Step-6. do
Set $s \leftarrow s + (h/2) * (f(a) + f(a+h))$
Set $a \leftarrow a + h$
Until $(a \geq b)$

Step-7. Print s

Step-8. Stop

3.1.3 Assignment

Write a code in C/MATLAB/PYTHON to implement Trapezoidal Rule and use it to evaluate $\int_0^1 \frac{1}{1+x^2}dx$ correct up to 5 decimal places taking 10 sub intervals.

3.1.4 Code in C

```
#include<stdio.h>
#define f(x) 1/(1+x*x)
int main()
    {
        int n;
        float a,b,h,s;
        printf("Enter a, b and n");
        scanf("%f%f%d",&a,&b,&n);
        h=(b-a)/n;
        s=0.0;
        do
            {
                s=s+h/2.0*(f(a)+f((a+h)));
                a=a+h;
        }while(a<b);
        printf("The value of Integral is %.5f",s);
        return 0;
    }
```

3.1.5 Input/Output

```
Enter a, b and n
0
1
10
The value of Integral is 0.78498
```

3.2 Simpson's 1/3rd Rule

3.2.1 Working Formula

The simple Simpson's 1/3rd rule for the interval $[a, a+2h]$ is given by

$$\int_a^b f(x)dx = \int_a^{a+2h} f(x)dx = \frac{h}{3}[f(a) + 4f(a+h) + f(a+2h)]$$

For n number of intervals (where n is an even number) $[a, a+nh]$ the composite rule will be given by

$$\begin{aligned}\int_a^b f(x)dx = \int_a^{a+nh} f(x)dx &= \frac{h}{3}[f(a) + f(a+nh) + 4(f(a+h) + f(a+3h) + \cdots + f(a+(n-1)h)) \\ &\quad +2(f(a+2h) + f(a+4h) + \cdots + f(a+(n-2)h))]\end{aligned}$$

3.2.2 Algorithm

Step-1. Start

Step-2. define f(x)

Step-3. Read a, b, n # *a=lower limit, b=upper limit, n number of intervals*

Step-4. Set $h \leftarrow (b-a)/n$

Step-5. Set $s \leftarrow 0.0$

Step-6. do
Set $s \leftarrow s + (h/3) * (f(a) + 4f(a+h) + f(a+2h))$
Set $a \leftarrow a + 2h$
Until $(a \geq b)$

Step-7. Print s

Step-8. Stop

3.2.3 Assignment

Write a code in C/MATLAB/PYTHON to implement Simpson's 1/3rd Rule and use it to evaluate $\int_0^1 \frac{1}{1+x^2}dx$ correct up to 5 decimal places taking 10 sub intervals.

3.2.4 Code in C

```
#include<stdio.h>
#define f(x) 1/(1+x*x)
int main()
    {
        int n;
        float a,b,h,s;
        printf("Enter a, b and n");
        scanf("%f%f%d",&a,&b,&n);
        h=(b-a)/n;
        s=0.0;
        do
            {
                s=s+h/3.0*(f(a)+4*f((a+h))+f((a+2*h)));
                a=a+2*h;
        }while(a<b);
        printf("The value of Integral is %.5f",s);
        return 0;
    }
```

3.2.5 Input/Output

```
Enter a, b and n
0
1
```

```
10
The value of Integral is 0.78540
```

3.3 Weddle's Rule

3.3.1 Working Formula

The simple Weddle's rule for the interval $[a, a + 6h]$ is given by

$$\int_a^b f(x)dx = \int_a^{a+6h} f(x)dx = \frac{3h}{10}[f(a) + 5f(a+h) + f(a+2h) + 6f(a+3h) + f(a+4h) + 5f(a+5h) + f(a+6h)]$$

For n number of intervals (where n is multiple of six) $[a, a + nh]$ the composite rule will be given by

$$\begin{aligned}\int_a^b f(x)dx = \int_a^{a+nh} f(x)dx = \frac{3h}{10}[&f(a) + f(a+nh) + 5(f(a+h) + f(a+7h) + \cdots + f(a+(n-5)h)) \\ &+(f(a+2h) + f(a+8h) + \cdots + f(a+(n-4)h)) \\ &+6(f(a+3h) + f(a+9h) + \cdots + f(a+(n-3)h)) \\ &+(f(a+4h) + f(a+10h) + \cdots + f(a+(n-2)h)) \\ &+(f(a+5h) + f(a+11h) + \cdots + f(a+(n-1)h)) \\ &+2(f(a+6h) + f(a+12h) + \cdots + f(a+(n-6)h))]\end{aligned}$$

3.3.2 Algorithm

Step-1. Start

Step-2. define f(x)

Step-3. Read a, b, n # *a=lower limit, b=upper limit, n number of intervals*

Step-4. Set $h \leftarrow (b - a)/n$

Step-5. Set $s \leftarrow 0.0$

Step-6. do
Set $s \leftarrow s + (3 * h/10) * (f(a) + 5 * f(a + h) + f(a + 2 * h) + 6 * f(a + 3 * h) + f(a + 4 * h) + 5 * f(a + 5 * h) + f(a + 6 * h))$
Set $a \leftarrow a + 6 * h$
Until $(a \geq b)$

Step-7. Print s

Step-8. Stop

3.3.3 Assignment

Write a code in C/MATLAB/PYTHON to implement Weddle's Rule and use it to evaluate $\int_0^1 \frac{1}{1+x^2}dx$ correct up to 5 decimal places taking 12 sub intervals.

3.3.4 Code in C

```
#include<stdio.h>
#define f(x) 1/(1+x*x)
int main()
    {
        int n;
        float a,b,h,s;
        printf("Enter a, b and n");
        scanf("%f%f%d",&a,&b,&n);
        h=(b-a)/n;
        s=0.0;
        do
            {
                s=s+3*h/10.0*(f(a)+5*f((a+h))+f((a+2*h))+6*f((a+3*h))+f((a+4*h))+5*f((
                a=a+6*h;
        }while(a<b);
        printf("The value of Integral is %.5f",s);
        return 0;
    }
```

3.3.5 Input/Output

```
Enter a, b and n
0
1
12
The value of Integral is 0.78540
```