Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 01
EXPERIMENT NAME: Determining the Root of a Non-Linear Equation Using Bisection Method.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif			**Dr. Md. Golam Moazzam**
Class Roll: 357			**Professor**
Exam Roll: 191338			**Dept. of Computer Science & Engineering**
Submission Date:			**Jahangirnagar University**
Sept 07, 2021			

Experiment No: 01

Name of the Experiment: Determining the Root of a Non-Linear Equation Using Bisection Method.

Objectives:

- Understanding the process of finding the root of non-linear equation using Bisection Method.
- Executing the implementation of the algorithm of Bisection Method.
- To achieve the accurate root and expected output of a given nonlinear function.
- To be able to interpret the advantages and disadvantages of the Bisection method.

The bisection method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of the equation lies.

Theory:

Bisection method being one of the most comprehensible and definitive iterative methods for the solution of nonlinear equations is also known as ***binary chopping*** or ***half-interval method***, relies on the fact that if f(x) is real and continuous in the interval $a < x < b$, and $f(a)$ and $f(b)$ are of opposite signs that is,

$$f(a).\ f(b) < 0$$

then there is at least one real root in the interval between a and b.

Let $x_1 = a$ and $x_2 = b$ Let us also define another point x_0 to be the midpoint between a and b. That is, there now exists the following three conditions:

- If $f(x_0) = 0$, we have a root at x_0.
- If $f(x_0).\ f(x_1) < 0$, there is a root between x_0 and x_1.
- If $f(x_0).\ f(x2) < 0$, there is a root between x_0 and x_2.

Testing the sign of the function at midpoint, we can deduce which part of the interval contains the root. This is illustrated in the following figure:

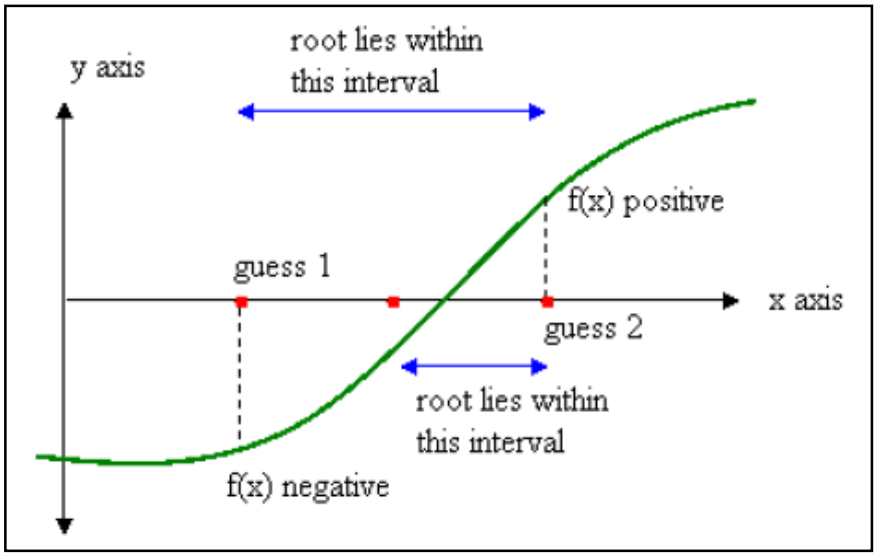

Fig 01: Graphical Depiction of Bisection Method.

Algorithm:

Bisection Method:

1. Deciding initial values for x1 and x2 and stopping criterion, E.
2. Computing $f_1 = f(x_1)$ and $f_2 = f(x_2)$.
3. If *f1 x f2 > 0*, then x_1 and x_2 do not bracket any root and go to step 7; otherwise continue.
4. Compute $x_0 = (x_1 + x_2)/2$ and compute $f_0 = f(x_0)$.
5. If $f_1 x f_0 < 0$ then

   set $x_2 = x_0$.

   else

   set $x_1 = x_0$
   set $f_1 = f_0$

6. If absolute value of $(x_2 - x_1)/ x_2$ is less than error E, then
   Root = $(x_2 - x_1)/ 2$
   Write the value of root
   Go to step 7
   else
   Go to step 4

7. Stop.

Coding in C:

```
#include<stdio.h>
#include<math.h>
#define EPS 0.000001
#define F(x) (x)*(x)+(x)-2
void Bisect(float *a,float *b,float *root,int *s,int *count)
{
    float x1,x2,x0,f0,f1,f2;
    x1=*a;
    x2=*b;
    f1=F(x1);
    f2=F(x2);
    if(f1*f2>0)
    {
        *s=0;
        return ;
    }
    else
    {
        *count = 0;
        while(1)
        {
            *count=*count+1;
            x0=(x1+x2)/2.0;
            f0=F(x0);
            if(f0==0)
            {
                *s=1;
                *root=x0;
                return ;
            }
            if(f1*f0<0)
            {
                x2=x0;
            }
            else
            {
                x1=x0;
                f1=f0;
            }
            if(fabs((x2-x1)/x2)<EPS)
            {
                *s=1;
                *root=(x1+x2)/2.0;
                return;
            }
            else
            {
                continue;
            }
        }
    }
}
```

```c
int main()
{
    int s,count;
    float a,b,root;
    printf("\n");
    printf("F(x) = x^2 + x - 2\n\n");
    printf("\n");
    printf("-----------  BISECTION METHOD SOLUTION  -----------\n");
    printf("\n");
    printf("Enter the starting values:\n");
    scanf("%f %f",&a,&b);
    Bisect(&a,&b,&root,&s,&count);
    if(s==0)
    {
        printf("\n");
        printf("Starting points do not bracket any root\n");
        printf("Check whether they bracket EVEN roots\n");
        printf("\n");
    }
    else
    {
        printf("\nRoot = %f \n",root);
        printf("F(root) = %f\n",F(root));
        printf("\n");
        printf("Iterations = %d\n",count);
        printf("\n");
    }
    return 0;
}
```

Output:

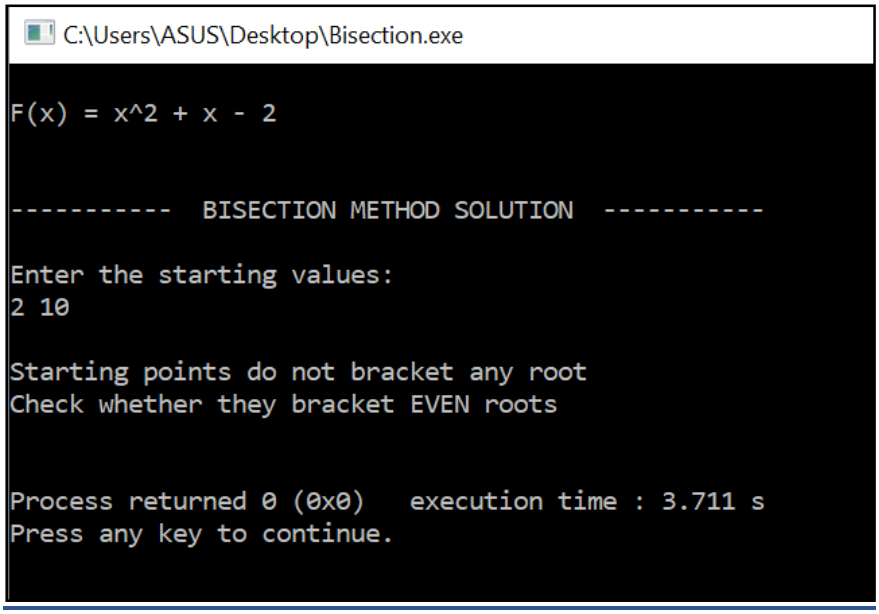

Fig 02: Output obtained when the starting points do not bracket any root.

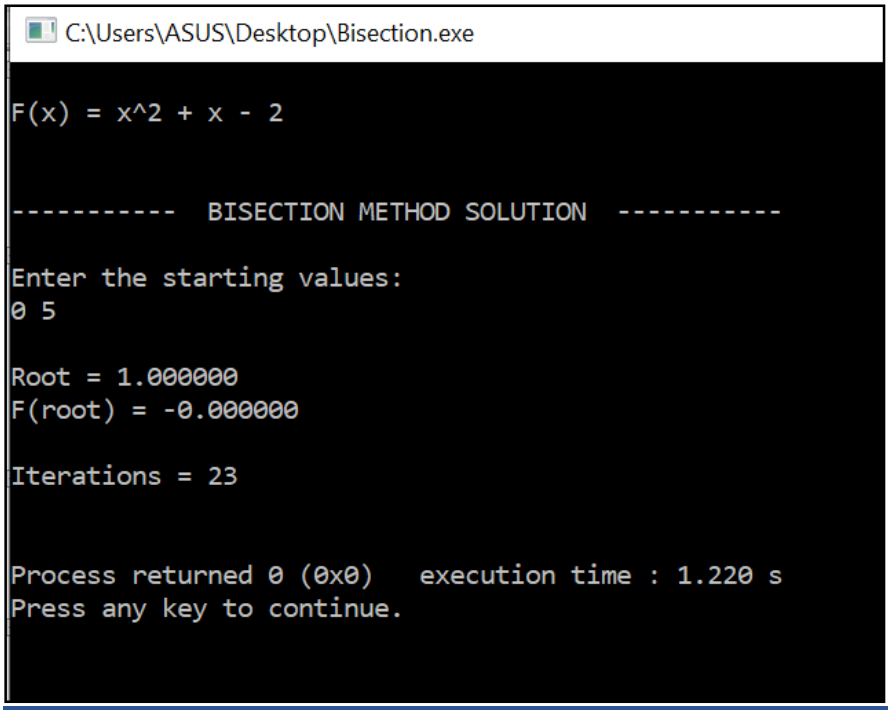

Fig 03: Output Obtained.

Discussion:

From the above Bisection method, we can say that bisection method is linearly convergent. Since the convergence is slow to achieve a high degree of accuracy, a large number of iterations may be needed. However, the bisection algorithm is guaranteed to converge. In bisection method, the interval between x_1 and x_2 is divided into two equal halves, irrespective of location of the root. It may be possible that the root is closer to one end than the other.

After applying this bracketing iterative method, we finally get the expected result which is the root of the polynomial equation. Thus using bisection method, we can find roots of nonlinear equation as it is difficult to solve normally we use numerical mathematics.

Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 02

EXPERIMENT NAME: Determining the root of a non-linear equation using False Position Method.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif			**Dr. Md. Golam Moazzam**
Class Roll: 357			**Professor**
Exam Roll: 191338			**Dept. of Computer Science & Engineering**
Submission Date:			**Jahangirnagar University**
Sept 06, 2021			

Experiment No: 02

Name of the Experiment: Determining the root of a non-linear equation using False Position Method.

Objectives:

- Understanding the process of finding the root of non-linear equation using False Position Method.
- Executing the implementation of False Position Method.
- To achieve the accurate root and expected output of a given nonlinear function.
- To be able to interpret the advantages and disadvantages of the false position method.

Theory:

The False position method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of the equation lies.

The basic principle of false position method is illustrated in the following figure.

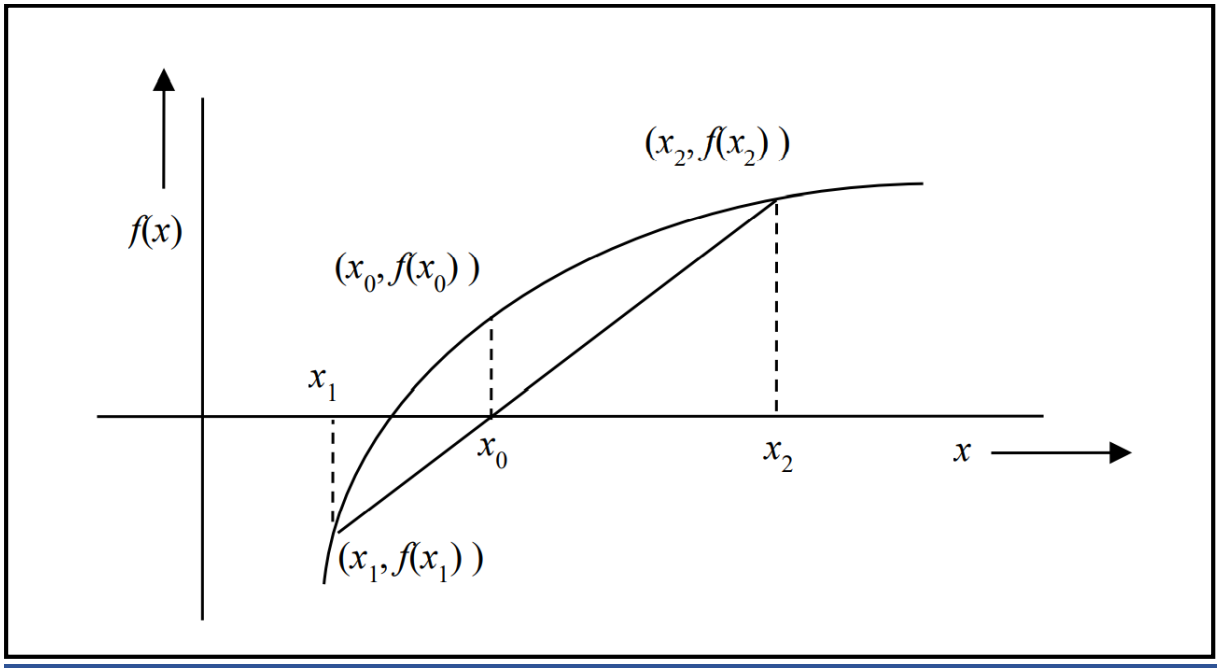

Fig 01: Graphical Depiction of False Position Method.

Let us assume that the root lines between x_1 and x_2.

Let us join the points x_1 and x_2 by a straight line. The point of intersection of this line with the x-axis (x_0) gives an improved estimation of the root and is called the false position of the root.

This point then replaces one of the initial guesses that has a function value of the same sign as $f(x_0)$. This process is repeated with the new values of x_1 and x_2.

Since this method uses the false position of the root repeatedly, it is called *the false position method.*

We know that equation of the line joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ is given by

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{y - f(x_1)}{x - x_1}$$

Since the line intersects the x-axis at x_0, when x = x_0, y = 0, we have

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{0 - f(x_1)}{x_0 - x_1}$$

Or,

$$x_0 - x_1 = \frac{-f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

Therefore,

$$x_0 = x_1 - \frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

This equation is known as the **false position formula**.

Algorithm:

False Position Method:

1. Start
2. Read values of x_1, x_2 and e

   /*Here x_1 and x_2 are the two initial guesses e is the degree of accuracy or the absolute error i.e. the stopping criteria*/
3. Compute function values $f(x_1)$ and $f(x_2)$
4. Check whether the product of f(x_1) and f(x_2) is negative or not. If it is positive take another initial guesses. If it is a negative value then go to step 5.
5. Determine:

$$x_0 = x_1 - \frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

6. Check whether the product of $f(x_1)$and $f(x_0)$is negative or not. If it is negative, then assign $x_2 = x_0$, f(x_2) = f(x_0); If it is positive, assign $x_1 = x_0$, f(x_1)=f(x_0);
7. Check whether the value of f(x_0) is greater than e or not.

   If yes, goto step 5.

   If no, goto step 8.

   /* Here the value e=0.000001 is the desired degree of accuracy, and hence the stopping criteria. */
8. Display the root as x.

Coding in C:

```c
#include<stdio.h>
#include<math.h>
#define EPS 0.000001
#define F(x) (x)*(x) - (x) - 2
int cnt;
float root;

int fal(float a, float b)
{
    float x1, x2, x0, f0, f1, f2;
    x1 = a;
    x2 = b;
    f1 = F(x1);
    f2 = F(x2);
    if(f1 * f2 > 0)
    {
        return 0;
    }
    cnt = 1;
    while(1)
    {
        x0 = x1 - (f1 * (x2 - x1) / (f2 - f1));
        f0 = F(x0);
        if(f1 * f0 < 0)
        {
            x2 = x0;
            f2 = f0;
        }
        else
        {
            x1 = x0;
            f1 = f0;
        }

        if(fabs(f0) <EPS)
        {
            break;
        }
        else
        {
            cnt++;
        }
    }
    root=x0;
    return 1;
}
int main()
{
    int s;
    float a, b;
    printf("\nF(x) = (x)*(x) - (x) - 2\n\n");
    printf("SOLUTION BY FALSE POSITION METHOD\n\n");
    printf("Input starting with values : ");
    scanf("%f %f",&a,&b);
    printf("\n");
    s = fal(a, b);
    if(s == 0)
    {
```

```
        printf("\nStarting points do not bracket any root\n\n");
    }
    else
    {
        printf("\nRoot = %.6f\n",root);
        printf("F(root) = %.6f\n",F(root));
        printf("\nNO. OF ITERATIONS = %d\n",cnt - 1);
    }
    return 0;
}
```

Output:

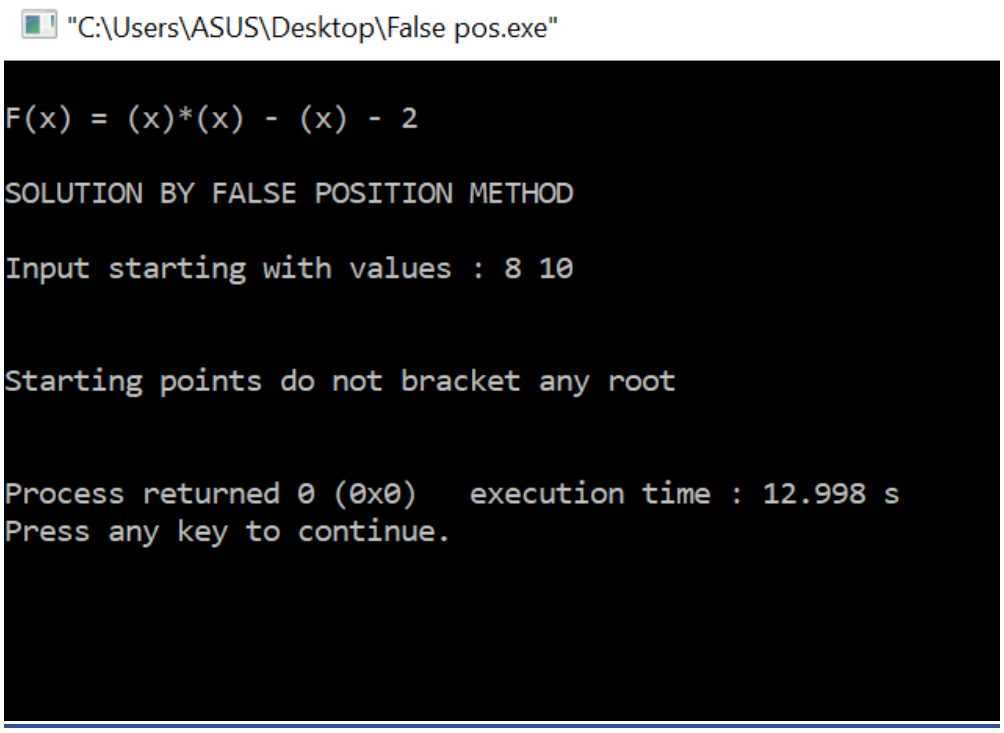

Fig 02: Output obtained when the starting points do not bracket any root.

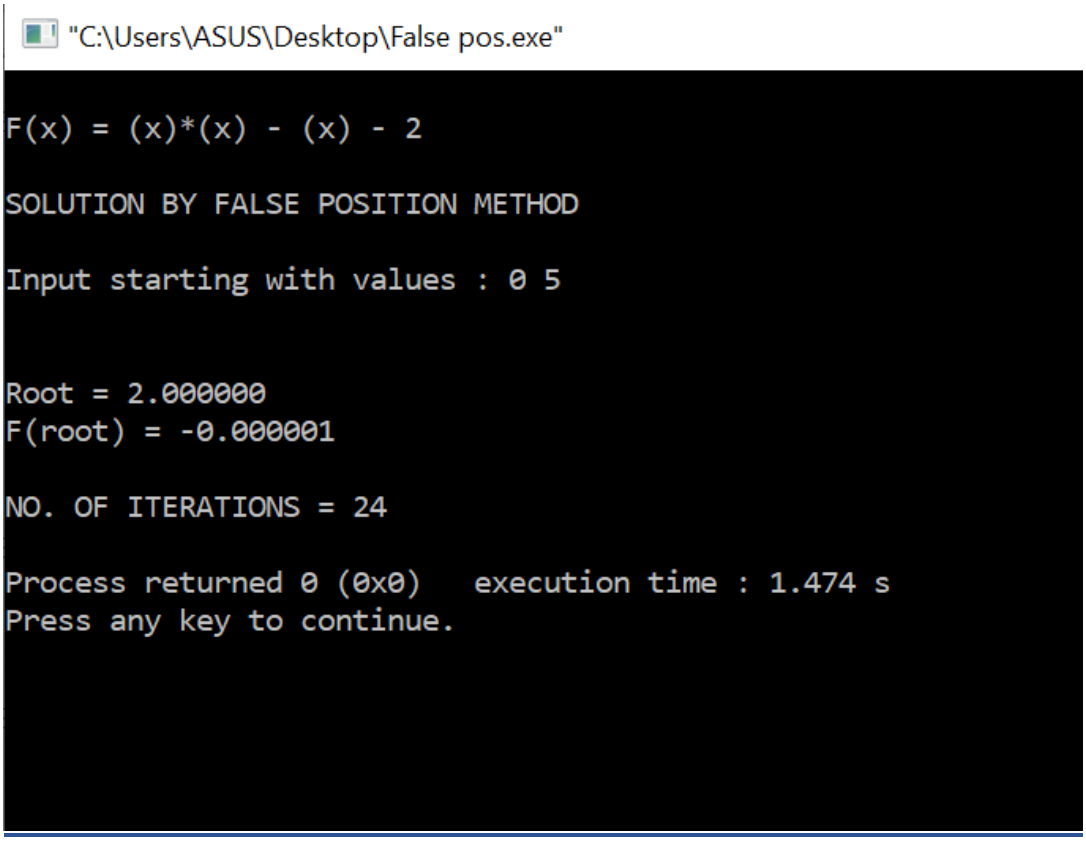

Fig 03: Output Obtained.

Discussion:

False Position method is linearly convergent as the initial guesses bracket the root. As the convergence rate is very slow, it takes many iterations to find a root. That is, as it is a trial and error method, in some cases it may take large time span to calculate the correct root and thereby slowing down the process. However, it is guaranteed to find root of a nonlinear equation by false position method.

Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 03

EXPERIMENT NAME: Determining the root of a non-linear equation using Newton-Raphson Method.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif **Class Roll: 357** **Exam Roll: 191338** **Submission Date:** **Sept 05, 2021**			**Dr. Md. Golam Moazzam** **Professor** **Dept. of Computer Science & Engineering** **Jahangirnagar University**

Experiment No: 03

Name of the Experiment: Determining the root of a non-linear equation using Newton-Raphson Method.

Objectives:

- Understanding the process of finding the root of non-linear equation using Newton-Raphson Method.
- Executing the implementation of the algorithm of Newton-Raphson Method.
- To achieve the accurate root and expected output of a given nonlinear function.
- To be able to interpret the advantages and disadvantages of Newton-Raphson Method.

Theory:

Let us assume that x_1 is an approximate root of $f(x) = 0$. Now, a tangent at the curve $f(x)$ at $x = x_1$.

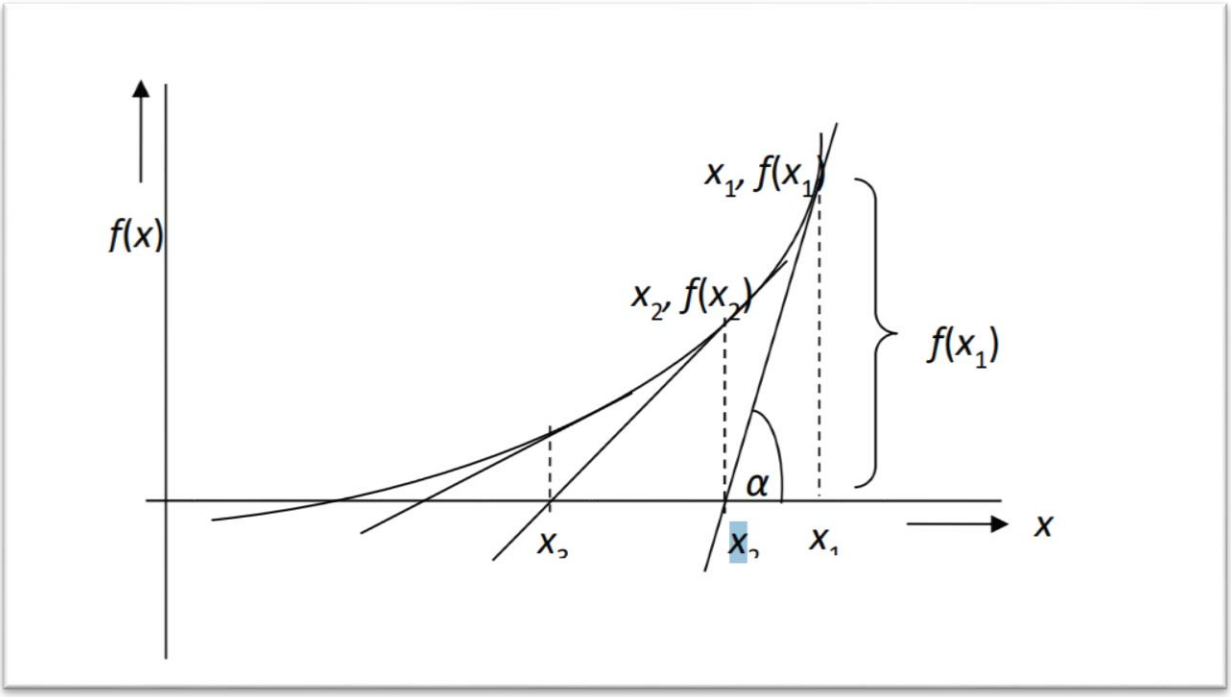

Fig 01: Graphical Depiction of Newton-Raphson Method.

The point of intersection of this tangent with the x-axis gives the second approximation to the root. Let the point of intersection be x_2.

The slope of the tangent is given by,

$$\tan \alpha = \frac{f(x_1)}{x_1 - x_2} = f'(x_1)$$

where $f'(x_1)$is the slope of *f(x)* at x = x_1. Solving for x_2 we obtain

$$x_2 = \frac{f(x_1)}{f'(x_1)}$$

This is called the Newton-Raphson formula.

The next approximation would be

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

That is in general,

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

This method of successive approximation is called the Newton-Raphson method. The process will be terminated when the difference between two successive values is within a prescribed limit.

Algorithm:

Newton-Raphson Method:

1. Assign an initial value of x, say x_0.
2. Evaluate $f(x_0)$ and $f'(x_0)$.
3. Find the improved estimate of x_0

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

4. Check the accuracy of the latest estimate.
   Compare relative error to a predefined value E.
   If $|\frac{x_1 - x_0}{x_{01}}| \leq E$ stop. Otherwise continue.
5. Replace x_0 by x_1 and repeat step 3 and step 4

Coding in C:

```
#include<stdio.h>
#include<math.h>
#define EPS 0.000001
#define MAXIT 20
#define F(x) (x)*(x)-3*(x)+2
#define FD(x) 2*x-3
void New_Raph(float x0, float fx, float fdx, float xn, int count)
{
    count=1;
    while(1)
    {
        fx=F(x0);
        fdx=FD(x0);
        xn= x0-(fx / fdx);
        if((fabs(xn-x0)/ xn) <EPS)
        {
            printf("Root = %f\n",xn);
            printf("Function Value = %f\n",F(xn));
            printf("Number of Iterations = %d\n",count);
            break;
        }
        else
        {
            x0 = xn;
            count=count+1;
```

```
            if(count<MAXIT)
            {
                continue;
            }
            else
            {
                printf("SOLUTION DOES NOT CONVERGE\n");
                printf("IN %d ITERATIONS\n",MAXIT);
            }
        }
    }
}
int main()
{
    int count;
    float x0,xn,fx,fdx;
    printf("The function is: x^2 - 3x +2\n");
    printf("Function derivative of first order is: 2x-3\n\n");
    printf("Input Initial value of x\n");
    scanf("%f",&x0);
    printf("\n");
    printf("SOLUTION BY NEWTON RAPHSON METHOD\n");
    printf("-------------------------------------\n");
    New_Raph(x0,fx,fdx,xn,count);
}
```

Output:

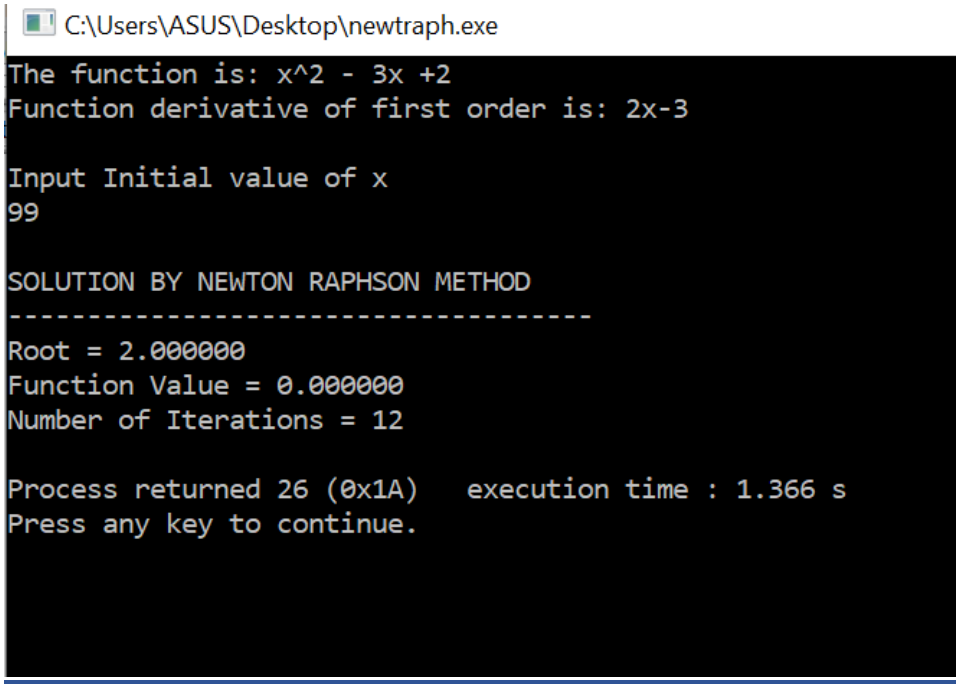

Fig 02: Output Obtained.

Discussion:

From the above experiment, we can observe that, complications will arise if the derivative is zero.

In such case, a new initial value for x must be chosen to continue the procedure. If the initial guess is too far away from the required root, the process may converge to some other root.

A particular value in the iteration sequence may repeat, resulting in an infinite loop. This occurs when the tangent to the curve $f(x)$ at $x = x_{i+1}$ cuts the x-axis again at $x = x_i$.

This method is much convenient for determining roots of a non-linear equation because this method converges quickly.

Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 04

EXPERIMENT NAME: Determining The Root of a Non-Linear Equation Using Secant Method.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif			**Dr. Md. Golam Moazzam**
Class Roll: 357			**Professor**
Exam Roll: 191338			**Dept. of Computer Science & Engineering**
Submission Date:			**Jahangirnagar University**
Sept 04, 2021			

Experiment No: 04

Name of the Experiment: Determining the root of a non-linear equation using secant method.

Objectives:

- Understanding the process of finding the root of non-linear equation using Scant Method.
- Executing the implementation of the algorithm of Scant Method.
- To achieve the accurate root and expected output of a given nonlinear function.
- To be able to interpret the advantages and disadvantages of Scant Method.

Theory:

Secant method uses two initial estimates but does not require that they must bracket the root. The basic concept is illustrated in the following figure:

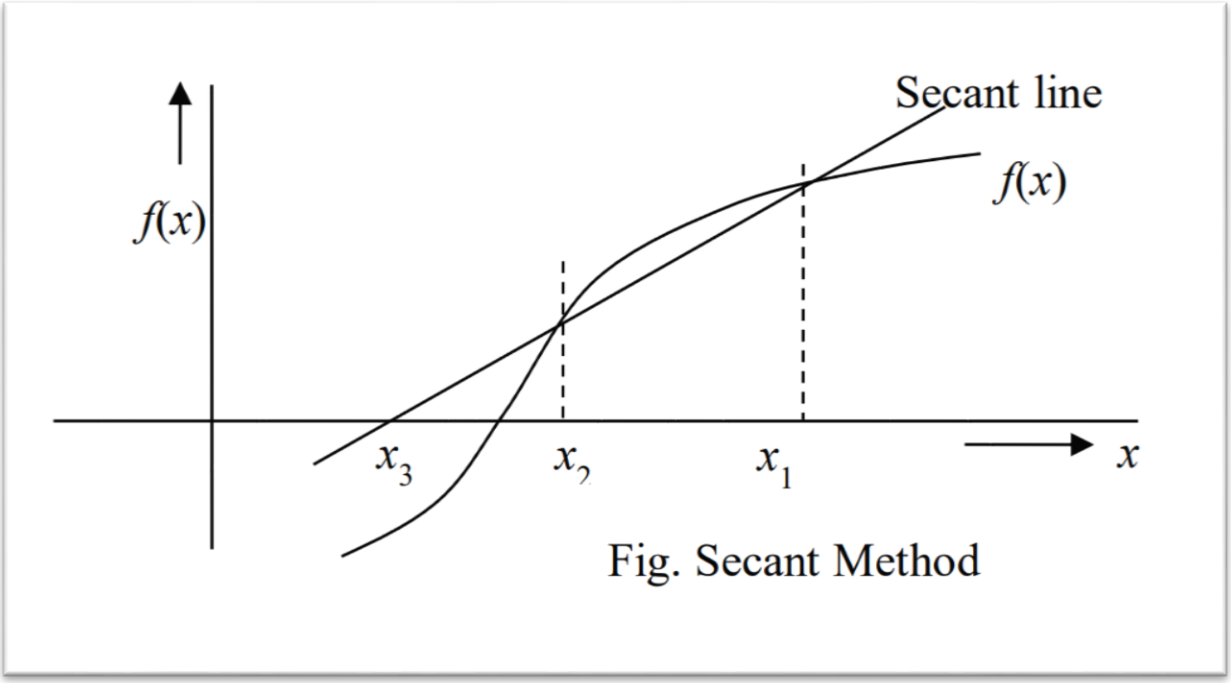

Fig 01: Graphical Depiction of Secant Method.

Consider the points x1 and x2 as starting values. They don't bracket the root. Slope of the secant line passing through x1 and x2 is given by

$$\frac{f(x_1)}{x_1 - x_3} = \frac{f(x_2)}{x_2 - x_3}$$

or,

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

or,

$$x_3[f(x_2) - f(x_1)] = f(x_2)x_1 - f(x_1)x_2$$

Therefore,

$$x_3 = x_2 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)}$$

In general form,

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

Algorithm:

Secant Method:

1. Choose two initial points x_1 and x_2, accuracy level required, E.
2. Compute f1=f(x_1) and f2=f(x_2).
3. Compute

$$x_3 = x_2 - \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1}$$

4. Test for accuracy of x_3.
   If $|\frac{x_3 - x_2}{x_3}| > E$, then Set x_1=x_2 and f_1=f_2 Set x_2=x_3 and f_2=f_3 Go to step 3
   Otherwise, Set root = x_3 Print results.
5. Stop.

Coding in C:

```c
#include<stdio.h>
#include<math.h>
#define EPS 0.000001
#define MAXIT 50
#define F(x) (x)*(x) - 4 * (x) - 10
int cnt;
float root, x1, x2;
int sec(float a, float b)
{
    float x3, f1, f2, err;
    x1 = a;
    x2 = b;
    cnt = 1;

    int c;
    do
    {
        f1 = F(x1);
        f2 = F(x2);
        if(fabs(f1 - f2) <=EPS) return 1;
        x3 = x2 - (f2 * (x2 - x1) / (f2 - f1));
        x1 = x2;
        f1 = f2;
        x2 = x3;
```

```
        f2 = F(x3);
        cnt++;
        if(cnt > MAXIT)
        {
            return 2;
        }
    }
    while(fabs(f1 - f2) >= EPS);
    root = x3;
    return 3;
}
int main()
{
    int s = 0;
    float a, b, x1, x2;
    printf("\n");
    printf("F(x) = (x)*(x) - 4 * (x) - 10\n\n");
    printf("SOLUTION BY SECANT METHOD\n\n");
    printf("Input starting with values : ");
    scanf("%f%f",&a,&b);
    printf("\n");
    s = sec(a, b);
    if(s == 1)
    {
        printf("\nDIVISION BY ZERO\n");
        printf("Last x1 = %.6f\n",x1);
        printf("Last x2 = %.6f\n",x2);
        printf("NO. OF ITERATIONS = %d\n\n",cnt - 1);
    }
    else if(s == 2)
    {

        printf("\nNO COVERGENCE IN %d ITERATIONS\n\n",MAXIT);
    }
    else
    {
        printf("\nRoot = %.6f\n",root);
        printf("F(root) = %.6f\n",F(root));
        printf("NO. OF ITERATIONS = %d\n\n",cnt - 1);
    }
    return 0;
}
```

Output:

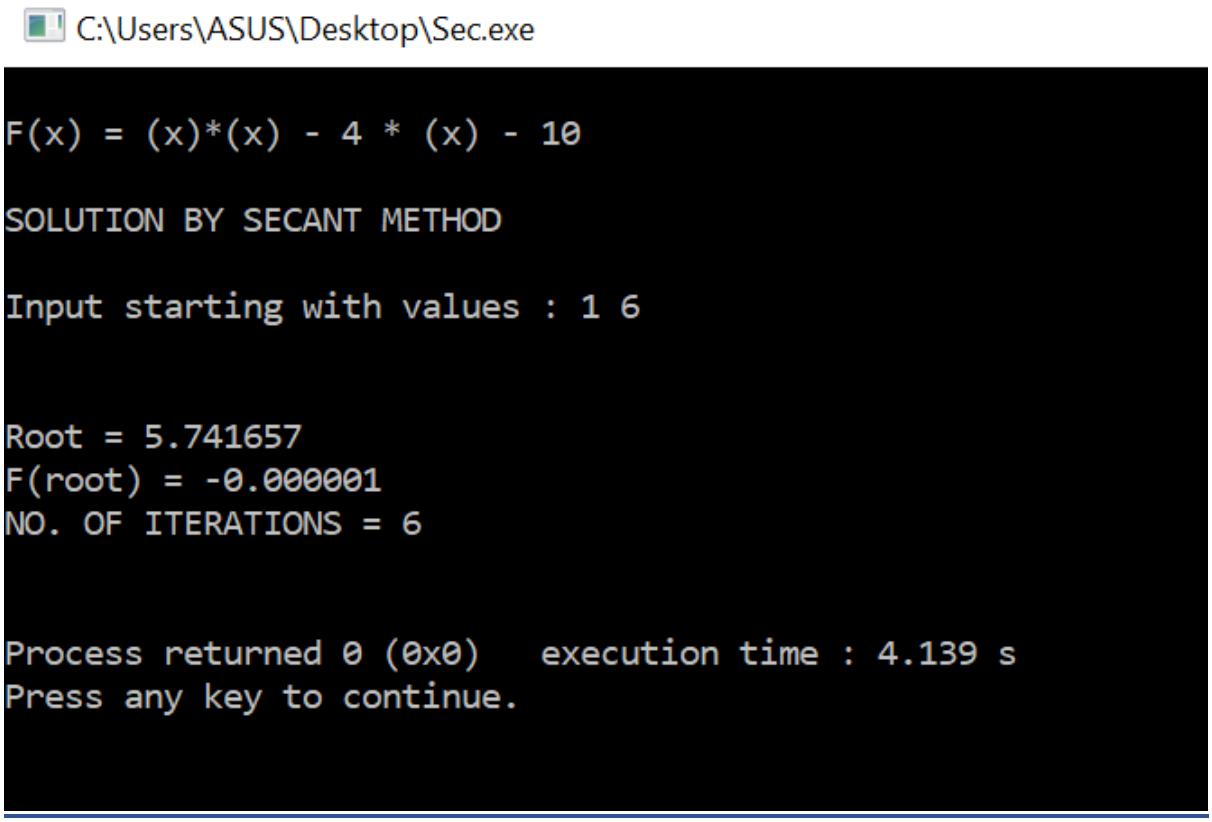

Fig 02: Output Obtained.

Discussion:

From the above experiment, we can observe that, Secant method doesn't need to bracket the root. Its convergence is slower than Newton-Raphson method. It doesn't need derivative to find the root of a nonlinear equation which makes it more convenient than Newton-Raphson method.

As the initial guesses don't bracket the root, it is uncertain whether the method will converge or not.

This method is easier to implement. If time is not the issue, then we can use this method to easily determine a root of a non-linear equation.

Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 05

EXPERIMENT NAME: Determining the solution of system of Linear Equations using Jacobi Iteration Method.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif			**Dr. Md. Golam Moazzam**
Class Roll: 357			**Professor**
Exam Roll: 191338			**Dept. of Computer Science & Engineering**
Submission Date:			**Jahangirnagar University**
Sept 04, 2021			

Experiment No: 05

Name of the Experiment: Determining the solution of system of Linear Equations using Jacobi Iteration Method.

Objectives:

- Understanding the process of solving system of linear equations using Jacobi Iteration Method.
- Executing the implementation of the algorithm of Jacobi Iteration Method.
- To achieve the accurate root and expected output of given system of linear equations.
- To be able to interpret the advantages and disadvantages of Jacobi Iteration Method.

Theory:

In numerical mathematics, the Jacobi-iteration method is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. It is one of simplest iterative algorithms to find roots of linear system of equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges.

Let us consider a system of n equations in n unknowns.

$$a_{11}x_1 + a_{12}x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + a_{2n}x_n = b_2$$

.

.

.

$$a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + a_{nn}x_n = b_n$$

We rewrite the original system as,

$$x_1 = \frac{b_2 - (a_{12}x_2 + a_{13}x_3 + a_{1n}x_n)}{a_{11}}$$

$$x_2 = \frac{b_2 - (a_{21}x_1 + a_{23}x_3 + a_{2n}x_n)}{a_{22}}$$

.

.

.

$$x_n = \frac{b_n - (a_{n1}x_1 + a_{n2}x_2 + a_{nn}x_{n-1})}{a_{nn}}$$

Now we can compute $x_1, x_2, \ldots x_n$ by using initial guesses for these values. These new values are again used to compute the next set of x values. The process can continue till we obtain a desired level of accuracy in the x values.

Algorithm:

Jacobi Iteration Method:

1. Obtain n, aij and bi values.
2. Set x0i = bi / aij for i =1, 2, ...n
3. Set key = 0.
4. For i = 1, 2, ..., n i.
    i. Set sum = bi.
    ii. For j = 1, 2, ... n (j ! = i) Set sum=sum- aij x0j Repeat j
    iii. Set xi = sum/aii
    iv. If key = 0 then,
        If |(xi - x0i)/xi| > error,
            Then Set key=1;

    Repeat i
5. If key = 1 then Set x0i = xi Go to step 3
6. Write results.

Coding in C:

```
#include<stdio.h>
#include<math.h>
#define EPS 0.000001
#define MAXIT 200

void Jacobi(int n,float a[10][10],float b[10], float x[10], int
            *count, int *status)
{
    int i,j,key;
    float sum, x0[10];

    for(i=1; i<=n; i++)
        x0[i]=b[i]/(a[i][i]*1.0);

    *count=1;
    while(1)
    {
        key=0;
        /*Computing values of x[i]
        ------------------------------------------
        x1=(b1-a12 x2-a13 x3-......a1n xn)/ a11
        x2=(b2-a21 x1-a23 x3-......a2n xn)/a22
        */
        for(i=1; i<=n; i++)
        {
            sum=b[i];
            for(j=1; j<=n; j++)
            {
                if(i==j)
                    continue;
                sum=sum-a[i][j]*x0[j];
            }
            x[i]=sum/(a[i][i]*1.0);
            if(key==0)
            {
//Testing for accuracy
                if(fabs((x[i]-x0[i])/x[i])>EPS)
                    key=1;
            }
        }

        if(key==1)
        {
            if(*count==MAXIT)
            {
                *status=2;
                return;
            }

            else
            {
                *status=1;
                for(i=1; i<=n; i++)
                {
                    x0[i]=x[i];
                }
```

```
            }
            *count=*count+1;
        }
        else
        {
            break;
        }
    }
    return;
}
int main()
{
    int i,j,n,count,status;
    float a[10][10],b[10],x[10];
    printf("\nSOLUTION BY JACOBI ITERATION \n");
    printf("-------------------------------------");
    printf("\n What is the size of n of the system? \n");
    scanf("%d",&n);
    printf("\nInput coefficients a(i,j), row by row \n");
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
        {
            scanf("%f",&a[i][j]);
        }
    }
    printf("\nInput vector b\n");
    for(i=1; i<=n; i++)
    {
        scanf("%f",&b[i]);
    }
    Jacobi(n,a,b,x,&count,&status);
    if(status==2)
    {
        printf("\nNo convergence in %d iterations",MAXIT);
        printf("\n\n");
    }
    else

    {
        printf("\nSOLUTION VECTOR X\n\n");
        for(i=1; i<=n; i++)
        {
            printf("X%d = %.6f\n",i,x[i]);
        }
        printf("\n\nIterations = %d ",count);
    }
    return 0;
}
```

Output:

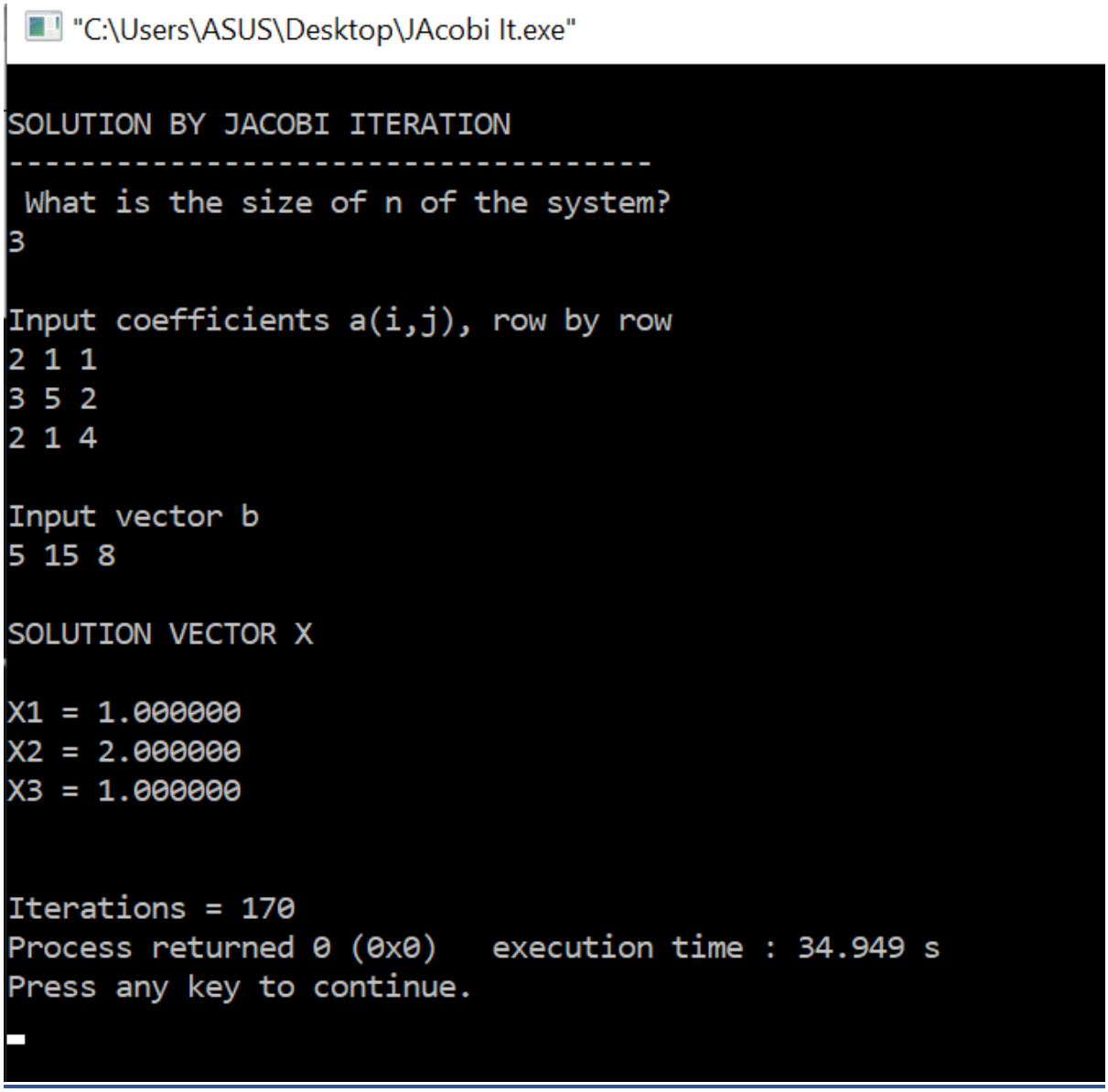

Fig 01: Output Obtained.

Discussion:

As we can see from above experiment, the more the iterations, the more the solution is accurate. Jacobi Iteration method can solve any n by n system of linear equations. This method takes too many iterations and gives the correct output.

Note that the simplicity of this method has both advantages and disadvantages. The method is preferred because it is relatively easy to understand and thus is a good first taste of iterative methods.

Jahangirnagar University
Department of Computer Science and Engineering

LAB REPORT
ON
CSE-206 (NUMERICAL METHODS LAB)

EXPERIMENT No.: 06

EXPERIMENT NAME: Determining the interpolation value at a specified point, given a set of data points, using the Lagrange Interpolation representation.

SUBMITTED BY			SUBMITTED TO
Sumaita Binte Shorif			**Dr. Md. Golam Moazzam**
Class Roll: 357			**Professor**
Exam Roll: 191338			**Dept. of Computer Science & Engineering**
Submission Date:			**Jahangirnagar University**
Sept 03, 2021			

Experiment No: 06

Name of the Experiment: Determining the interpolation value at a specified point, given a set of data points, using the Lagrange Interpolation representation.

Objectives:

- Understanding the process of finding interpolation value at a specified point given a set of data points, using Lagrange Interpolation polynomial representation.
- Executing the implementation of the algorithm of Lagrange Interpolation Polynomial.
- To get the value at a specified point as accurate as possible.
- To be able to interpret the advantages and disadvantages of Lagrange Interpolation Polynomial.

Theory:

Let $x_0, x_1, \dots, x_n$ denote n distinct real numbers and let $f_0, f_1, \dots, f_n$ be arbitrary real numbers. The points (x_0, f_0), (x_1, f_1), (x_2, f_2),... ., (x_n, f_n) can be imagined to be data values connected by a curve.

Any function p(x) satisfying the conditions

$p(x_k) = f_k$ for k = 0, 1, …., n

is called *interpolation function*. An interpolation function is, therefore, a curve that passes through the data points as pointed out.

Let us consider a second order polynomial of the form

$$p_2(x) = b_1(x - x_0)(x - x_1) + b_2(x - x_1)(x - x_2) + b_3(x - x_2)(x - x_0) \dots\dots\dots (1)$$

If (x_0, f_0), (x_1, f_1), and (x_2, f_2) are the three interpolating points, then we have

$$p_2(x_0) = f_0 = b_2(x_0 - x_1)(x_0 - x_2)$$

Or,

$$b_2 = \frac{f_0}{(x_0 - x_1)(x_0 - x_2)}$$

Or,

$$p_2(x_1) = f_1 = b_3(x_1 - x_2)(x_1 - x_0)$$

Or,

$$b_3 = \frac{f_1}{(x_1 - x_2)(x_1 - x_0)}$$

$$p_2(x_2) = f_2 = b_1(x_2 - x_0)(x_2 - x_1)$$

Or,

$$b_1 = \frac{f_2}{(x_2 - x_0)(x_2 - x_1)}$$

Substituting b_1, b_2 and b_3 in equation (1), we get,

$$p_2(x) = \frac{f_2(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} + \frac{f_0(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + \frac{f_1(x - x_2)(x - x_0)}{(x_1 - x_2)(x_1 - x_0)}$$

Equation (2) may be represented as

$$p_2(x) = f_0 l_0(x) + f_1 l_1(x) + f_2 l_2(x) = \sum_0^n f_0 l_0(x)$$

Where,

$$l_i(x) = \prod_{j=0, j!=i}^{2} \frac{x - x_j}{x_i - x_j}$$

In general for n+1 points we obtain n-th degree polynomial as,

$$p_n(x) = \sum_0^n f_i l_i(x)$$

$$l_i(x) = \prod_{j=0, j!=i}^{n} \frac{x - x_j}{x_i - x_j}$$

Algorithm:

Lagrange Interpolation:

1. Start
2. Read number of data n.
3. For I = 1 to n
   Read data xi and fi.
4. Read input xp at which interpolation is required.
5. Set sum = 0.
6. For i = 1 to n
   Set lf = 1.0, where lf is lagrage interpolation factor.
   For j = 1 to n
   if (i!=j) set lf = lf * (xp - x[j]) / (x[i] - x[j])
   sum=sum+(lf * fi)
7. fp=sum
8. Print fp
9. End.

Coding in C:

```
#include<stdio.h>
#define MAX 10
int main()
{
    int n,i,j;
    float x[MAX], f[MAX], fp, lf, sum, xp;
    printf("\nInput number of data points, n : ");
    scanf("%d",&n);
    printf("\nInput data points x(i) and values f(i) [one set per
line]\n\n");
    for(i = 1; i <= n; i++)
    {
        scanf("%f %f",&x[i],&f[i]);
    }
    printf("\nInput x at which interpolation is required : ");
    scanf("%f",&xp);
    printf("\n\n");
    sum = 0.0;
    for(i = 1; i <= n; i++)
    {
        lf = 1.0;
        for(j = 1; j <= n; j++)
        {
            if(i != j) lf = lf * (xp - x[j]) / (x[i] - x[j]);
        }
```

```
            sum += (lf * f[i]);
        }
        fp = sum;
        printf("\tLAGRAGIAN INTERPOLATION\t\n\n");
        printf("Interpolated function value at x = %.6f is %.6f\n",xp,fp);
}
```

Output:

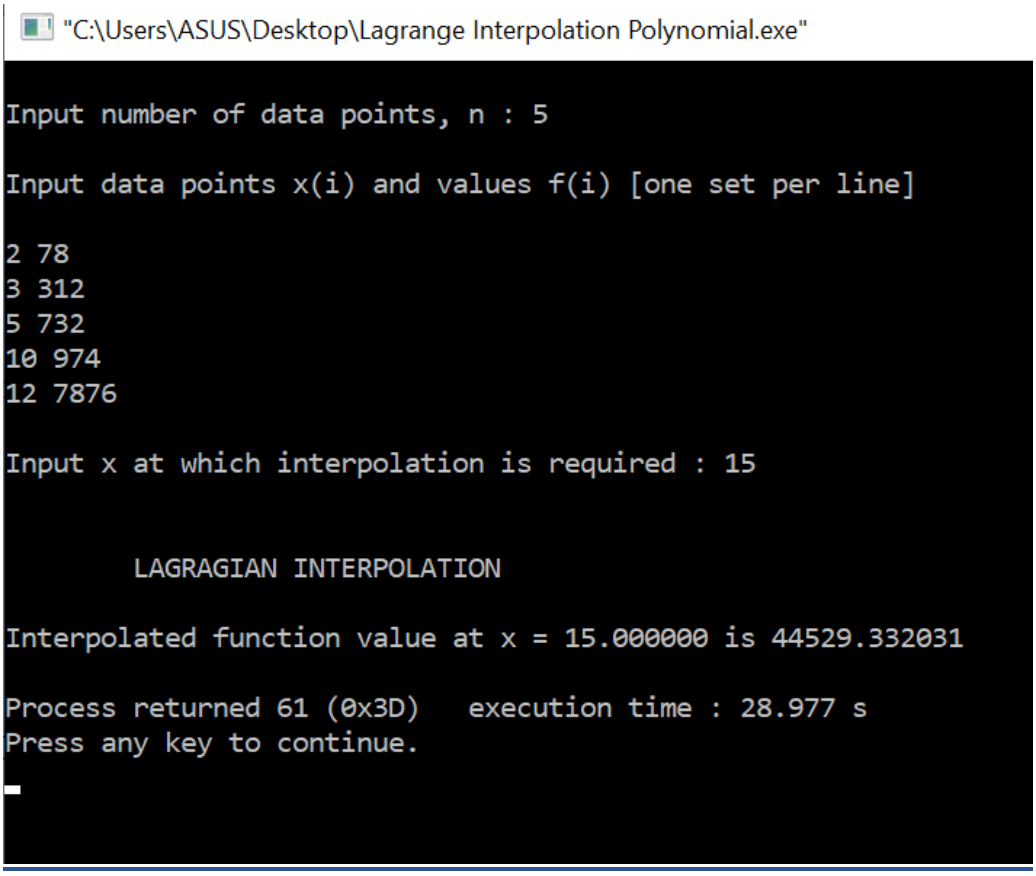

Fig 01: Output Obtained.

Discussion:

Although it's an easy method to implement, it has some advantages and disadvantages.

Advantages:

The Lagrange form of the interpolation polynomial shows the linear character of polynomial interpolation and the uniqueness of the interpolation polynomial. Therefore, it is preferred in proofs and theoretical arguments.

Disadvantages:

It requires 2(n+1) multiplications/divisions and 2n+1 additions and subtractions. If we want to add one more data point, we have to compute the polynomial from the beginning. It does not use the polynomial that has already been computed.

As we can observe from the above experiment, Lagrange Interpolation formula requires 2(n+1) multiplications/divisions and 2n+1 additions and subtractions. If we want to add one more data point, we have to compute the polynomial from the beginning. It does not use the polynomial already computed.
