BISECTION METHOD ALGORITHM, FLOWCHART AND CODE IN C

Bisection method is an iterative implementation of the *'Intermediate Value Theorem'* to find the real roots of a nonlinear function. According to the theorem "If a function f(x)=0 is continuous in an interval (a, b), such that f(a) and f(b) are of opposite nature or opposite signs, then there exists at least one or an odd number of roots between a and b."

Using C program for bisection method is one of the simplest computer programming approach to find the solution of nonlinear equations. It requires two initial guesses and is a closed bracket method. Bisection method never fails!

The programming effort for Bisection Method in C language is simple and easy. The convergence is linear, slow but steady. The overall accuracy obtained is very good, so bisection method is more reliable in comparison to the *Newton Raphson method* or the *Regula-Falsi method*.

Features of Bisection Method:

- Type – closed bracket
- No. of initial guesses – 2
- Convergence – linear
- Rate of convergence – slow but steady
- Accuracy – good
- Programming effort – easy
- Approach – middle point

Bisection Method Algorithm:

1. Start
2. Read x0, x1, err

   *Here x1 and x2 are initial guesses

   e is the absolute error i.e. the desired degree of accuracy*
3. Compute: f1 = f(x0) and f2 = f(x1)
4. If (f0*f1) > 0, then display initial guesses are wrong and goto (11).

   Otherwise continue.
5. X2 = (x0 + x1)/2
6. If ([(x1 – x2)] > e), then display x2 and goto (11).

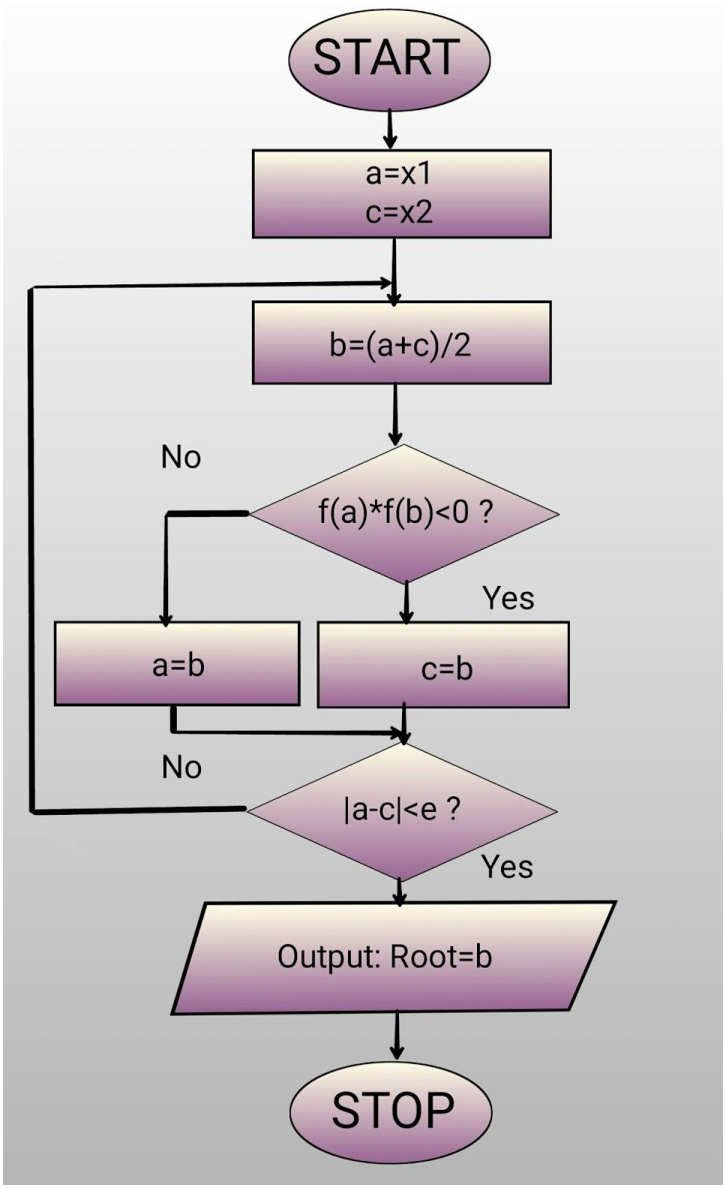
   * Here [ ] refers to the modulus sign. *
7. Else, f2 = f(x2)
8. If ((f2*f0) > 0), then x0 = x2 and f0 = f2.
9. Else, x1 = x2 and f1 = f2.

10. Goto (5).

    Now the loop continues with new values.

11. Stop

Bisection Method Flowchart:



Below is a source code in **C program for bisection method** to find a root of the nonlinear function ***x^3 – 4*x –* 9**. The initial guesses taken are a and b. The calculation is done until the following condition is satisfied:

|a-b| < 0.0005 OR If (a+b)/2 < 0.0005 (or both equal to zero) where, (a+b)/2 is the middle point value.

Variables:

- i – a counter variable which keeps track of the no. of iterations performed
- x2 – the value of root at the nth iteration
- x0, x1 – the limits within which the root lies
- err – allowed error
- x1 – the value of root at (n+1)th iteration

f(x) = x^3 – 4*x – 9

Source Code for Bisection Method in C:

```
1. #include<stdio.h>                    /*including header files*/
2. #include<conio.h>
3. #include<math.h>
4. #define err 0.0005                    /*defining the error term*/
5. #define f(x) x*x*x-4*x-9              /*defining the function f(x)*/
6. void main()
7. {
8. int i=1;                        /*declaring variables*/
9. float x0,x1,x2;
10.     double f0,f1,f2;
11.     printf("\nEnter the value of x0: ");
12.     scanf("%f",&x0);      /*generating input/output statements*/
13.     printf("\nEnter the value of x1: ");
14.     scanf("%f",&x1);
15.     printf("\n-------------------------------------------------
   ------------\n");
16.     printf("\nIteration x0\t   x1\t   x2\t   f0\t   f1\t
   f2\n");                          /*generating the tabular form*/
17.     printf("\n-------------------------------------------------
   ----------\n");
18.     do                        /*initialization of do loop*/
19.     {
```

```
20.      f0=f(x0);
21.      f1=f(x1);
22.      x2=(x0+x1)/2;
23.      f2=f(x2);       /*assigning of values*/
24.      printf("\n%d\t%f %f %f %lf %lf %lf",i,x0,x1,x2,f0,f1,f2);
   /*values of the process*/
25.      if(f0*f2<0)     /*the main process*/
26.      x1=x2;
27.      else
28.      x0=x2;
29.      i++;
30.      }while(fabs(f2)>err);   /*check if the error term is met*/
31.      printf("\n-------------------------------------\n");
32.      printf("Approximate Root: %f",x2); /*final answer*/
33.      getch();
34.      }
```

Input/Output:

```
Enter the value of x0: 2

Enter the value of x1: 3

-------------------------------------------------------------

Iteration x0       x1       x2       f0       f1       f2

-------------------------------------------------------------

1       2.000000 3.000000 2.500000 -9.000000 6.000000 -3.375000
2       2.500000 3.000000 2.750000 -3.375000 6.000000 0.796875
3       2.500000 2.750000 2.625000 -3.375000 0.796875 -1.412109
4       2.625000 2.750000 2.687500 -1.412109 0.796875 -0.339111
5       2.687500 2.750000 2.718750 -0.339111 0.796875 0.220917
6       2.687500 2.718750 2.703125 -0.339111 0.220917 -0.061077
7       2.703125 2.718750 2.710938 -0.061077 0.220917 0.079423
8       2.703125 2.710938 2.707031 -0.061077 0.079423 0.009049
9       2.703125 2.707031 2.705078 -0.061077 0.009049 -0.026045
10      2.705078 2.707031 2.706055 -0.026045 0.009049 -0.008506
11      2.706055 2.707031 2.706543 -0.008506 0.009049 0.000270
-------------------------------------
Approximate Root: 2.706543_
```

REGULA FALSI Method ALGORITHM, FLOWCHART AND CODE IN C

Regula Falsi method, also known as the **false position method**, is the oldest approach to find the real root of a function. It is a closed bracket method and closely resembles the bisection method.

The **C Program for regula falsi method** requires two initial guesses of opposite nature. Like the secant method, interpolation is done to find the new values for successive iterations, but in this method one interval always remains constant.

The programming effort for **Regula Falsi or False Position Method in C** language is simple and easy. The convergence is of first order and it is guaranteed. In manual approach, the method of false position may be slow, but it is found superior to the bisection method.

Features of Regula Falsi Method:

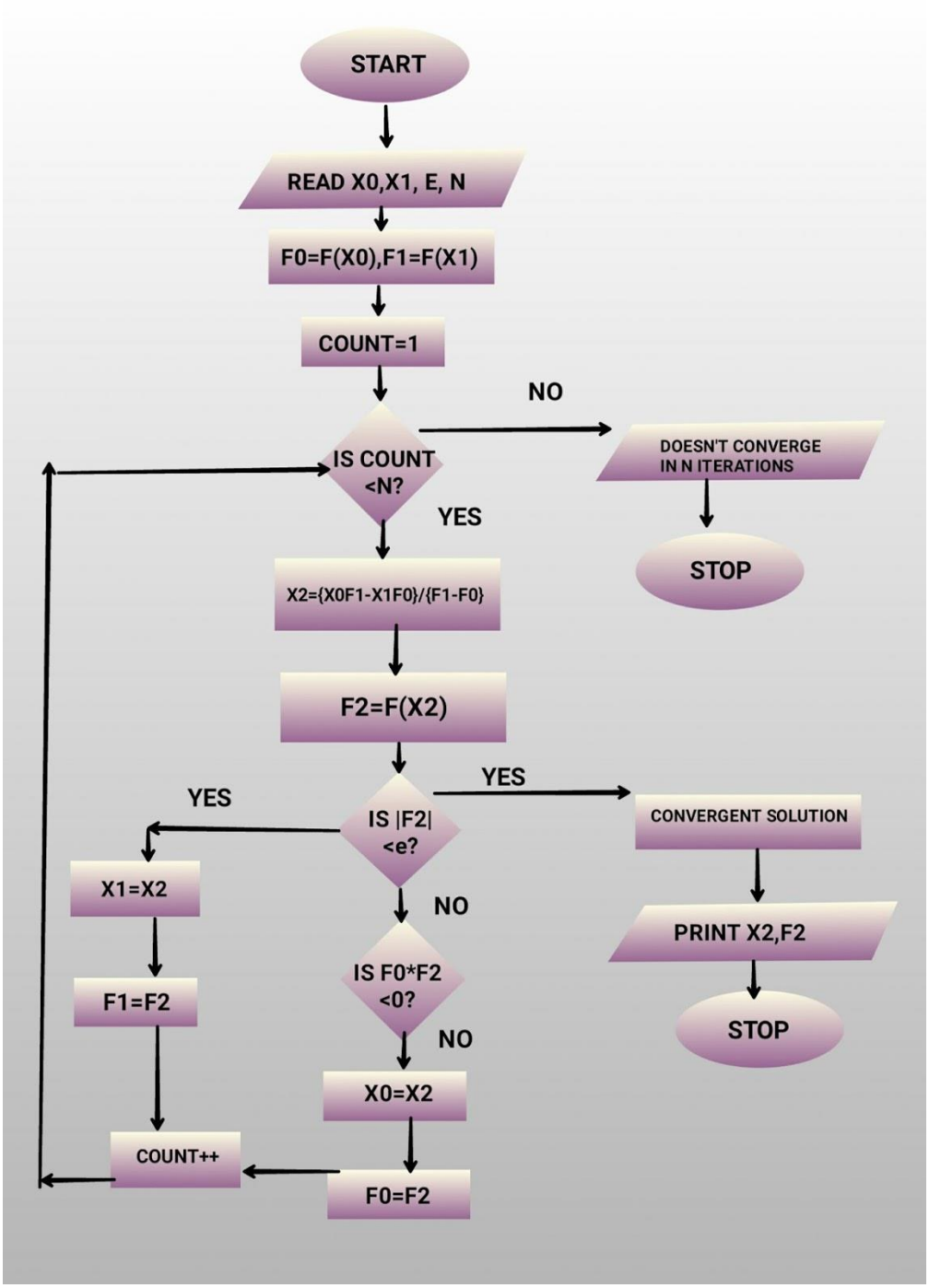
- Type – closed bracket
- No. of initial guesses – 2
- Convergence – linear
- Rate of convergence – slow
- Accuracy – good
- Programming effort – easy
- Approach – interpolation

Regula Falsi Method Algorithm:

1. Start
2. Read values of x0, x1, e and N
   *Here x0 and x1 are the two initial guesses
   e is the degree of accuracy or the absolute error i.e. the stopping criteria*
3. N is the count value.
4. Computer function values f(x0) and f(x1)
5. Check whether the product of f(x0) and f(x1) is negative or not.
   If it is positive take another initial guesses.
   If it is negative then goto step 5.
6. Determine:
   x = [x0*f(x1) – x1*f(x0)] / (f(x1) – f(x0))
7. Check whether the product of f(x1) and f(x) is negative or not.
   If it is negative, then assign x0 = x;
   If it is positive, assign x1 = x;
8. Check whether the value of f(x) is greater than 0.00001 or not.
   If yes, goto step 5.
   If no, goto step 8.
   Here the value 0.00001 is the desired degree of accuracy, and hence the stopping criteria.
9. Display the root as x.
10. Stop

er.sarbesh@gmail.com

Regula Falsi Method Flowchart:



Below is a short and simple source code in **C program for regula falsi method** to find the root of *cos(x) – x*e^x*. Here, x0 and x1 are the initial guesses taken.

Variables:

- itr − a counter which keeps track of the no. of iterations performed

- maxmitr – maximum number of iterations to be performed
- x0, x1 – the limits within which the root lies
- x2 – the value of root at nth iteration
- x3 – the value of root at (n+1)th iteration
- allerr – allowed error
- x – value of root at nth iteration in the regula function
- f(x0), f(x1) – the values of f(x) at x0 and x1 respectively

f(x) = x^3-2*x-5

Source Code for Regula Falsi Method in C:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>              /*included header files*/
#define esp 0.0001            /*defining error term*/
#define f(x) x*x*x-2*x-5    /*defining the function*/
void main()
{
float x0,x1,x2,f1,f2,f0;
int count=1;
     do
     {
     printf("\n Enter the value of x0: ");
     scanf("%f",&x0);              /*loop to get x0*/
     }while(f(x0)>0);
     do
     {
     printf("\n Enter the value of x1: ");
     scanf("%f",&x1);           /*loop to get x1*/
     }while(f(x1)<0);
     printf("\n
**************************************************** \n");
     printf("  n  x0\t    x1\t       x2\t       f0\t       f1\t
f2");
     printf("\n
**************************************************** \n");
/*formatting the output*/
     do
     {
     f0=f(x0);
     f1=f(x1);
```

er.sarbesh@gmail.com

```
27.      x2=x0-((f0*(x1-x0))/(f1-f0));          /*working formula
   loop*/
28.      f2=f(x2);
29.      printf("\n %d  %f  %f  %f  %f  %f
   %f",count,x0,x1,x2,f0,f1,f2);
30.      if(f0*f2<0)
31.      x1=x2;
32.      else
33.      x0=x2;
34.      count++;
35.      }while(fabs(f2)>esp);
36.      printf("\n
   ************************************************** \n");
37.      printf("\n\n Approximate Root = %f",x2);  /*final answer*/
38.      getch();
39.      }
```

Input/Output:

```
Enter the value of x0: 2

Enter the value of x1: 3

*********************************************************
 n   x0      x1          x2              f0              f1              f2
*********************************************************

1   2.000000   3.000000   2.058824   -1.000000   16.000000   -0.390799
2   2.058824   3.000000   2.081264   -0.390799   16.000000   -0.147203
3   2.081264   3.000000   2.089639   -0.147203   16.000000   -0.054677
4   2.089639   3.000000   2.092740   -0.054677   16.000000   -0.020203
5   2.092740   3.000000   2.093884   -0.020203   16.000000   -0.007450
6   2.093884   3.000000   2.094306   -0.007450   16.000000   -0.002745
7   2.094306   3.000000   2.094461   -0.002745   16.000000   -0.001010
8   2.094461   3.000000   2.094518   -0.001010   16.000000   -0.000372
9   2.094518   3.000000   2.094539   -0.000372   16.000000   -0.000137
10   2.094539   3.000000   2.094547   -0.000137   16.000000   -0.000050
*********************************************************


Approximate Root = 2.094547
```

NEWTON RAPHSON METHOD ALGORITHM, FLOWCHART AND CODE IN C

Newton Raphson method, also called the Newton's method, is the fastest and simplest approach of all methods to find the real root of a nonlinear function. It is an open bracket approach, requiring only one initial guess. This method is quite often used to improve the results obtained from other iterative approaches.

The convergence is fastest of all the root-finding methods we have discussed in Code with C. The algorithm and flowchart for Newton Raphson method given below is suitable for not only find the roots of a nonlinear equation, but the roots of algebraic and *transcendental equations* as well.

The overall approach of Newton's method is more useful in case of large values the first derivative of f(X) i.e f'(X). The iterative formula for Newton Raphson method is:

[highlight color="yellow"]Xn+1 = Xn – f(Xn)/f'(Xn)[/highlight]

Features of Newton's Method:

- Type – open bracket
- No. of initial guesses – 1
- Convergence – quadratic
- Rate of convergence – faster
- Accuracy – good
- Programming effort – easy
- Approach – Taylor's series

Limitations of Newton-Raphson Method:

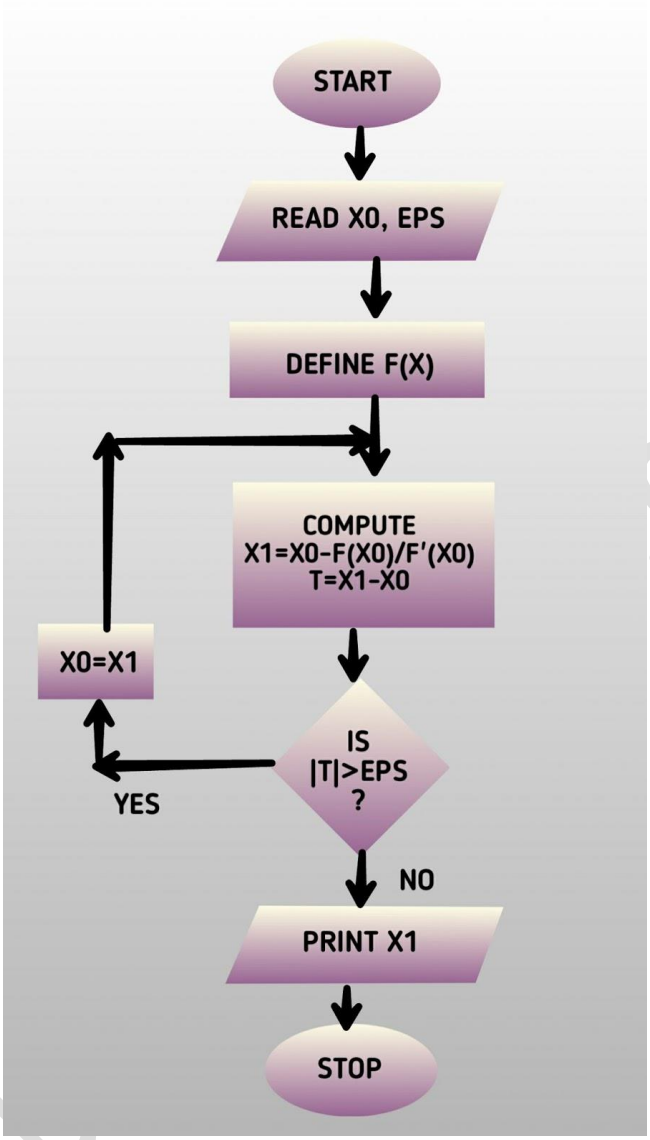
- Finding the f'(x) i.e. the first derivative of f(x) can be difficult if f(x) is complicated.
- When f'(xn) tends to zero i.e. the first derivative of f(xn) tends to zero, Newton-Raphson method gives no solution.
- Near local maxima or local minima, there is infinite oscillation resulting in slow convergence.
- In Newton's method C program, if the initial guess is far from the desired root, then the method may converge to some other roots.
- If root jumping occurs, the intended solution is not obtained.

Newton Raphson Method Algorithm:

STEP 1: Read x0 (the initial guess) and eps (the allowable error)
STEP 2: Define f(x)
STEP 3: Compute x1=x0-f(x0)/f'(x0) and t = x1-x0.
STEP 4: If |t|>eps, then replace x0 by x1 and goto STEP 3, otherwise print the root is x1 and terminate the Algorithm.

er.sarbesh@gmail.com

Newton Raphson Method Flowchart:



Below is a very short and simple source code in C program for Newton's method to find the root of **x^2+2*x – 2**

.**Variables:**

- itr – a counter which keeps track of the no. of iterations performed
- z=df(x) – the derivative of f(x) with respect to x
- x0 – the value of root at nth iteration
- xn – the value of root at (n+1)th iteration
- err – allowed error

f(x) = x^2+2*x – 2

Source Code for Newton Raphson Method in C:

```
2. #include<stdio.h>
```

```
#include<conio.h>
#include<math.h>                    /*header files included*/
float f(float x)                   /*declaring the function f(x)*/
{
float y;
y=x*x+2*x-2;
return y;
}
float f1(float x)          /*declaring the function f'(x)*/
{
float z;
z=2*x+2;
return z;
}
void main()
{
int n=0;
float x0,xn,err=1,h;
printf("\nEnter the initial value of x = "); /*asking for the
intial value of x*/
scanf("%f",&x0);
printf("\n n\t xn \t\t f(xn)\t\t f'(xn)\t\tx(n+1)\n");
/*designing the table*/
while(fabs(err)>.000001)
{
h=-(f(x0)/f1(x0));                    /*working formula*/
xn=x0+h;
err=xn-x0;
printf("\n %d\t %.10f\t %.10f\t %.10f\t
%.10f\n",n,x0,f(x0),f1(x0),xn);  /*formatting values in output*/
x0=xn;
n++;
}
printf("\n One real root of the given equation is %.5f (correct
upto 5 decimal places)",xn);      /*printing the final answer*/
getch();
}
```

Input/Output:

```
Enter the initial value of x = 0
 n       xn              f(xn)           f'(xn)          x(n+1)
 0       0.0000000000    -2.0000000000   2.0000000000    1.0000000000
 1       1.0000000000    1.0000000000    4.0000000000    0.7500000000
 2       0.7500000000    0.0625000000    3.5000000000    0.7321428657
 3       0.7321428657    0.0003189071    3.4642858505    0.7320508361
 4       0.7320508361    0.0000000988    3.4641017914    0.7320508361
 One real root of the given equation is 0.73205 (correct upto 5 decimal places)
Process returned 13 (0xD)   execution time : 7.080 s
Press any key to continue.
```

SECANT METHOD ALGORITHM, FLOWCHART AND CODE IN C

Secant method is the most effective approach to find the root of a function. It is based on *Newton-Raphson method*, and being free from derivative it can be used as an alternative to Newton's method. The C program for Secant method requires two initial guesses, and the method overall is open bracket type. Also, the secant method is an improvement over the *Regula-Falsi method* as approximation is done by a secant line during each iterative operation.

The programming effort for Secant Method in C is a bit tedious, but it's the most effective of all other method to find the root of a function. Here, at each iteration, two of the most recent approximations of the root are utilized to find out the next approximation. Also, in secant method, it is not mandatory that the interval should contain the root.

The secant method is faster than the bisection method as well as the regula-falsi method. The rate of convergence is fast; once the secant method converges, its rate of convergence is 1.618, which is quite high. Although convergence is not guaranteed in this method, this method is the most economical one giving definitely rapid rate of convergence at low cost.

Features of Secant Method:

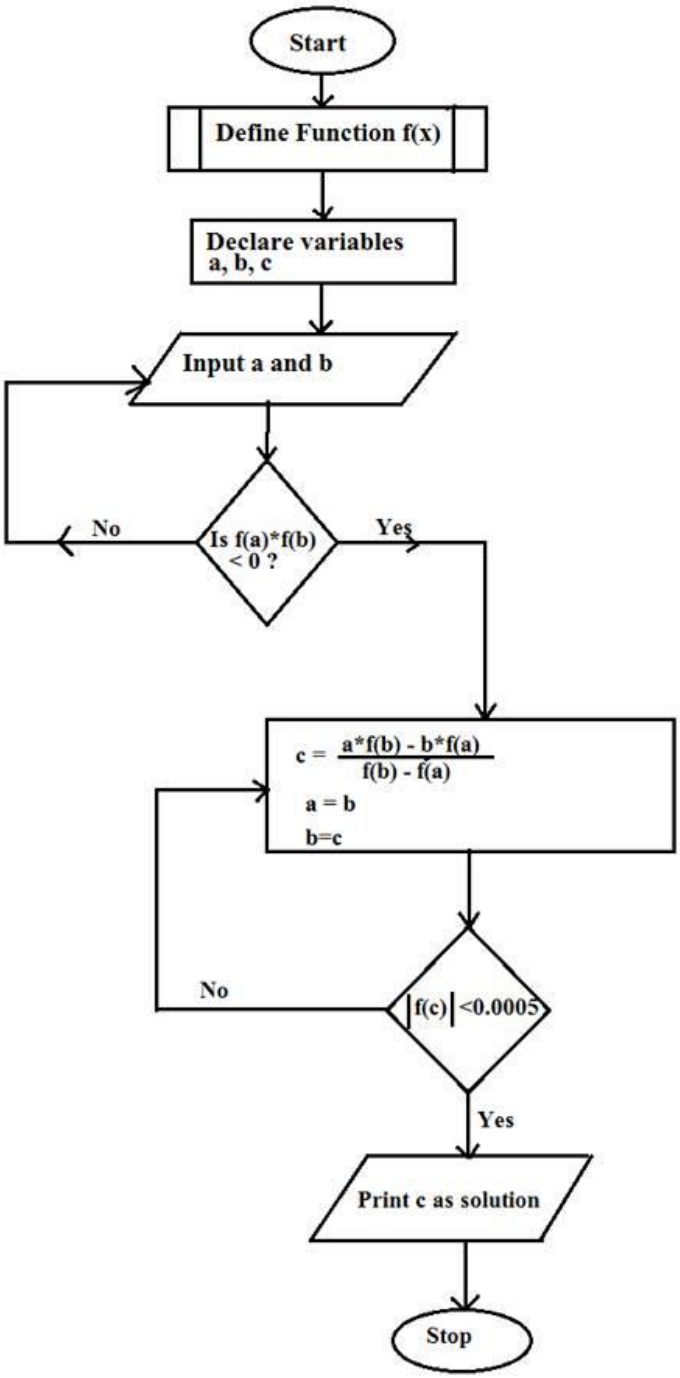
- Type – open bracket
- No. of initial guesses – 2
- Convergence – *super linear*
- Rate of convergence – faster
- Accuracy – good
- Programming effort – tedious
- Approach – interpolation

Secant Method Algorithm:

1. Start

2. Get values of x0, x1 and e
   *Here x0 and x1 are the two initial guesses
   e is the stopping criteria, absolute error or the desired degree of accuracy*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0*f(x1) – x1*f(x0)] / [f(x1) – f(x0)]
5. Test for accuracy of x2
   If [(x2 – x1)/x2] > e, *Here [ ] is used as modulus sign*
   then assign x0 = x1 and x1 = x2 and goto step 4
6. Else, goto step 7
7. Display the required root as x2.
8. Stop

Secant Method Flowchart:



Below is a short and simple C programming source code for Secant method to find the root of x^3-8x-5

f(x) = x^3-8x-5

Source Code for Secant Method in C:

```
#include<stdio.h>
float f(float x)
{
return(x*x*x-8*x-5); // f(x)= x^3-8x-5
}
float main()
{
float x0,x1,x2,err;
int count=1,itr;
    printf("\n\nEnter the values of x0 and x1:\n"); //(a,b) must contain the solution.
    scanf("%f%f",&x0,&x1);
    printf("Enter the values of allowed error and maximun number of iterations:\n");
    scanf("%f %d",&err,&itr);
    do
    {
    if(f(x0)==f(x1))
    {
    printf("\nSolution cannot be found as the values of a and b are same.\n");
    return;
    }
    x2=(x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
    x0=x1;
    x1=x2;
    printf("Iteration No-%d    x=%f\n",count,x2);
    count++;
    if(count==itr)
    {
    break;
    }
    } while(fabs(f(x2))>err);
    printf("\n The required solution is %.4f\n",x2);
    }
```

er.sarbesh@gmail.com

Input/Output:

```
Enter the values of a and b:
3
3.5
Enter the values of allowed error and maximun number of iterations:
0.00005
20
Iteration No-1    x=3.084211
Iteration No-2    x=3.097876
Iteration No-3    x=3.100451
Iteration No-4    x=3.100432

 The required solution is 3.1004

Process returned 34 (0x22)   execution time : 16.286 s
Press any key to continue.
```

When the values for a and b, the initial guesses, are entered same, the solution can't be obtained. In such case, enter different values for a and b such that f(a) and f(b) are not equal.

```
Enter the values of a and b:
3
3
Enter the values of allowed error and maximun number of iterations:
0.005
20

Solution cannot be found as the values of a and b are same.

Process returned 0 (0x0)   execution time : 8.891 s
Press any key to continue.
```

FIXED POINT ITERATION METHOD ALGORITHM, FLOWCHART AND CODE IN C

Fixed point iteration method is commonly known as the *iteration method*. It is one of the most common methods used to find the real roots of a function. The C program for fixed point iteration method is more particularly useful for locating the real roots of an equation given in the form of an infinite series. So, this method can be used for finding the solution of arithmetic series, geometric series, Taylor's series and other forms of infinite series.

This method is linearly convergent with somewhat slower rate of convergence, similar to the *bisection method*. It is based on modification approach to find the fixed point. It is commonly referred to as simple enclosure method or open bracket method.

Like other methods to find the root of a function, the programming effort for Iteration Method in C is easy, short and simple. Just like the *Newton-Raphson method*, it requires only one initial guess, and the equation is solved by the assumed approximation.

Iterations and modifications are successively continued with the updated approximations of the guess. Iterative method gives good accuracy overall just like the other methods.

Features of Fixed Point Iteration Method:

- Type – open bracket
- No. of initial guesses – 1
- Convergence – linear
- Rate of convergence – fast
- Accuracy – good
- Programming effort – easy
- Approach – modification

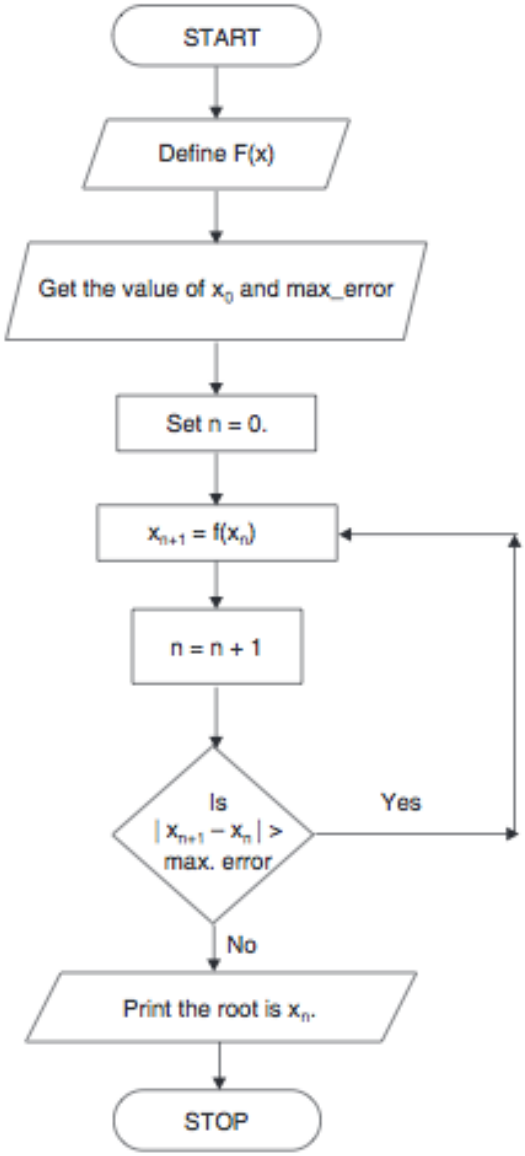
Fixed Point Iteration Method Algorithm:

1. Start
2. Read values of x0 and err.
   *Here x0 is the initial approximation
   err is the absolute error or the desired degree of accuracy, also the stopping criteria*
3. If [gd(x)]>=1, display "The equation is not convergent "and go to step 7
4. *Here [ ] refers to the modulus sign & gd(x) refers to derivative of g(x)*
5. Calculate x1 = g(x0)

1. If [x1 – x0] <= e, goto step 6.
   Here [ ] refers to the modulus sign
2. Else, assign x0 = x1 and goto step 4.
3. Display x1 as the root.
4. Stop

er.sarbesh@gmail.com

Fixed Point Iteration Method Flowchart:



Below is a source code in C program for iteration method to find the root of $e^{-x}-x$. The desired degree of accuracy in the program can be achieved by continuing the iteration i.e. by increasing the maximum number of iterations.

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

Source Code for Fixed Point Iteration Method in C:

```
1. #include<conio.h>
2. #include<stdio.h>
3. #include<math.h>
4. #define g(x) (exp(-x))
```

```
#define gd(x) -(exp(-x))
#define e 0.001
int main(void)
{
float x0,x1,f1, error ;
    int i=1;
    printf("Using Fixed Point Method / Iterative Method to find
  root of exp(-x)-x=0\n");
    printf("Enter value for x0\n");
    printf("x0: ");
    scanf("%f",&x0);
    if(fabs(gd(x0))>1)
    {
    printf("\nThe equation is not convergent");
    goto c;
    }
    else
    {
    printf("steps \tx0  \tx1=g(x0) \tError\n");
    printf("\n-----------------------------------------\n");
    b:x1=g(x0);
    error=fabs(x1-x0);
    printf("%d \t%.6f \t%.6f \t%.6f\n",i,x0,x1,error);
    i++;
    }
    if(fabs((x1-x0))<=e)
    {
    printf("\n=========================================\n");
    printf("The root is %.6f",x0);
    goto c;
    }
    else
    {
    x0=x1;
    goto b;
    }
    c:
    getch();
    }
```

er.sarbesh@gmail.com

Input/Output:

```
Using Fixed Point Method / Iterative Method to find root of exp(-x)-x=0
Enter value for x0
x0: 1.1
steps    x0        x1=g(x0)          Error

-----------------------------------------
1        1.100000          0.332871          0.767129
2        0.332871          0.716863          0.383992
3        0.716863          0.488282          0.228581
4        0.488282          0.613680          0.125398
5        0.613680          0.541355          0.072325
6        0.541355          0.581959          0.040604
7        0.581959          0.558803          0.023157
8        0.558803          0.571893          0.013091
9        0.571893          0.564456          0.007438
10       0.564456          0.568670          0.004214
11       0.568670          0.566278          0.002391
12       0.566278          0.567634          0.001356
13       0.567634          0.566865          0.000769

=========================================
The root is 0.567634
Process returned 0 (0x0)   execution time : 18.967 s
Press any key to continue.
```

HORNER'S RULE ALGORITHM, FLOWCHART AND CODE IN C

According to Horne's rule the quadratic (i.e. degree 2) polynomial

$$y = c_1 x^2 + c_2 x + c_3$$

can be rearranged as

$$y = (c_1 x + c_2) x + c_3$$

The same pattern can be applied to a cubic (degree 3) polynomial

$$y = c_1 x^3 + c_2 x^2 + c_3 x + c_4$$
$$y = (c_1 x^2 + c_2 x + c_3) x + c_4$$

Factoring the quadratic term inside the parenthesis gives

$$y = ((c_1 x + c_2) x + c_3) x + c_4$$

This pattern is called Horner's rule for evaluating a polynomial. For hand calculation of low degree, it makes sense to use direct computation of the polynomial in its standard form. To evaluate a polynomial in a computer program, Horner's rule makes more sense, especially if speed and accuracy are important and the degree of the polynomial is large.

Horner's method Algorithm

Step 1: Start

Step 2: Input (integer / float) the coefficients of a polynomial of degree *n*

Step 3:Read all the coeff. till n=0

Step 4: Input x (integer / float)

Step 5: Read x

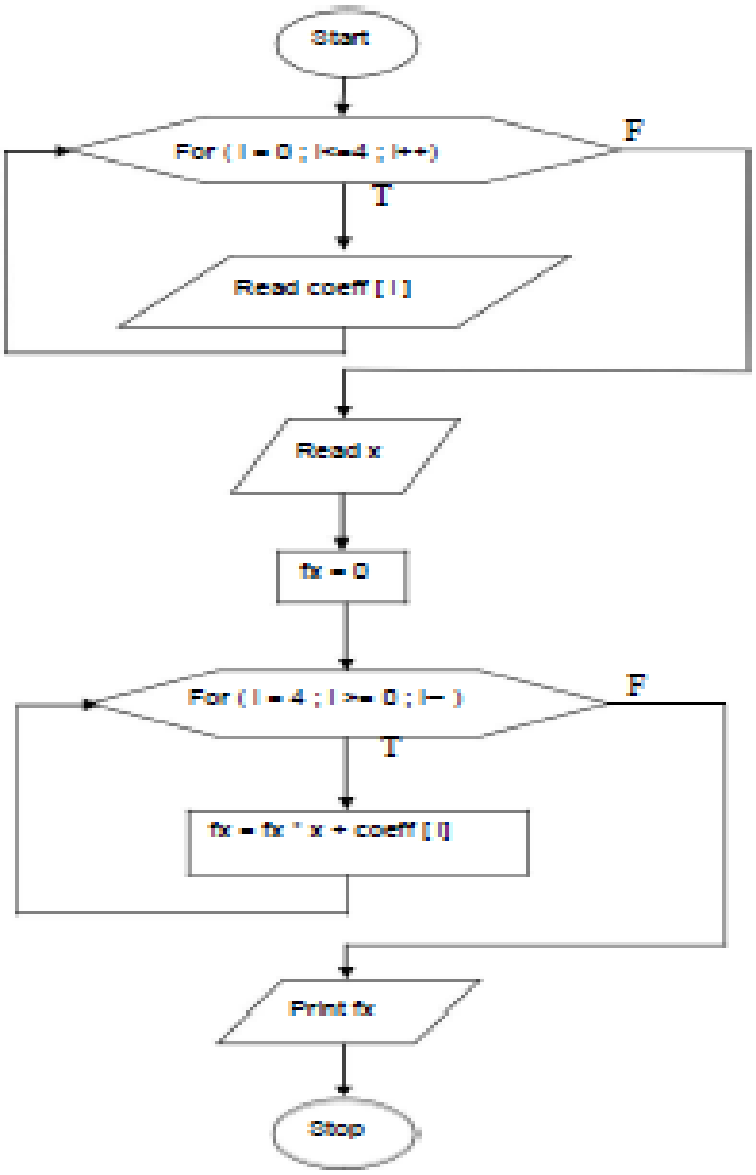
Step 6: [Initialize] fx = 0 and calculate sum=(sum+a[i])*x

Step 7: Repeat step 6 until *I* is greater than or equal to 0

Step 8: [Output] Print the value of fx

Step 9: Stop

Horner's method Flowchart



Below is a source code in **C program for Horner's method** to evakluate the polynomial,

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x^0 + a_0$$

Source Code for Horner's Method in C:

```
#include <stdio.h>
int main()
{
float a[100],sum=0,x;
int n,i;
printf("\nEnter degree of the polynomial X :: ");
scanf("%d",&n);
printf("\nEnter coefficient's of the polynomial X :: \n");
for(i=n;i>=0;i--)
    {
    printf("\nEnter Coefficient of [ X^%d ] :: ",i);
    scanf("%f",&a[i]);
    }
    printf("\nEnter the value of X :: ");
    scanf("%f",&x);
    for(i=n;i>0;i--)
    {
    sum=(sum+a[i])*x;
    }
    sum=sum+a[0];
    printf("\nValue of the polynomial is = [ %f ]\n",sum);
    return 0;
    }
```

Input/Output:

```
Enter degree of the polynomial X :: 3

Enter coefficient's of the polynomial X ::

Enter Coefficient of [ X^3 ] :: 4

Enter Coefficient of [ X^2 ] :: 0

Enter Coefficient of [ X^1 ] :: -3

Enter Coefficient of [ X^0 ] :: 9

Enter the value of X :: 1.5

Value of the polynomial is = [ 18.000000 ]

Process returned 0 (0x0)   execution time : 28.250 s
Press any key to continue.
```

THE END

sarbesh@gmail.com