Solution of non-linear equation using Bisection Method

Objective:

To solve the equation using bisection method

Theory:

The bisection method is one of the simplest and most reliable of iterative methods for the solution of nonlinear equations. This method is also known as binary chopping or half-interval method. This is one of the bracketing method

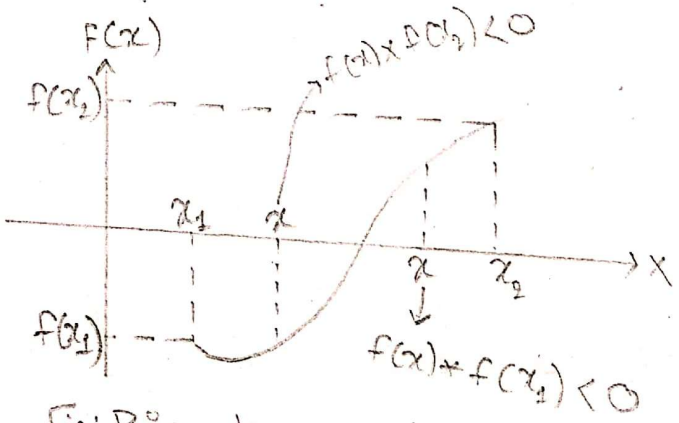


Fig: Bisection Method

Let $f(x)$ be the function defined by the interval $[x_1, x_2]$. Also let another point x to the mid point between x_1 and x_2

i.e. $x = \frac{x_1 + x_2}{2}$

Now, there exists the following conditions:

1. If $f(x) = 0$, we have a root at x
2. If $f(x) * f(x_1) < 0$, there is a root between x and x_1
3. If $f(x) * f(x_2) < 0$, there is a root between x and x_2

Algorithm:

1. Define function $f(x)$ and Error level (E)
2. Input x_1 and x_2 such that $(f(x_1) * f(x_2) < 0)$
3. If $([f(x_1) * f(x_2)] > 0)$
   Print "input right gauses"
   goto step 2
   else
   continue
4. Compute:
   $$x = \frac{x_1 + x_2}{2}$$
   $$\text{Error} = |x_2 - x_1|$$
5. If $([f(x_1) * f(x)] < 0)$ then
   set $x_2 = x$
   else
   set $x_1 = x$
6. If (Error > E)
   goto step 4
   else
   Print x as a root
7. End

Source code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define f(x) (x*x-4*x-10)
#define E (0.0001)
void main()
  {
    float x1,x2,x, Error;
   here:
     printf("Enter initial gouses\n");
     scanf("%f %f",&x1,&x2);
     if (f(x1)*f(x2)>0)
       {
         printf("The root does not exist on those gouses");
         goto here;
       }
     else
     {
      next:
       x=(x1+x2)/2;
     Error = fabs(x1-x2);
     if (f(x)*f(x1)<0)
      {
       x2=x;
      }
     else
       {
         x1=x;
       }
   if(Error>E)
     {
       goto next;
     }
   else
     {
       printf("The root is %f\n",x);
     }
}
```

Output

```
Enter initial gauses
 2
 3
The root does not exist on those gauses
Enter initial gauses
 -2
 -1

The root is -1.7416
```

Solution of non Linear equation using False Position Method

Objective:

To solve the equation using False Position Method

Theory:

False Position method is one of the bracketing method. This method is also known as regula-falsi or the method of chords

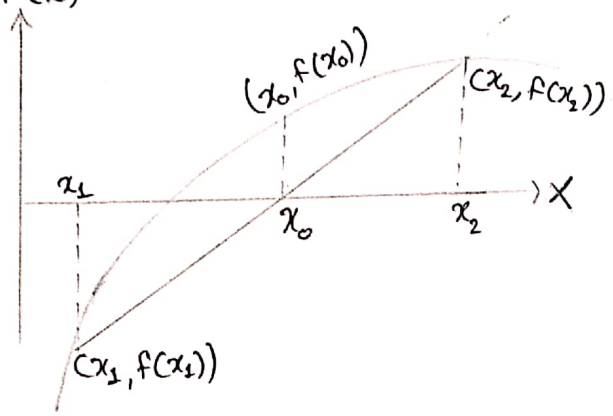


Fig: Illustration of false position method

A graphical depiction of the false position method is shown in above figure. The equation of the line joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ is given by

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

From above figure,

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

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

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

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

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

∴ This equation is known as the false position formula.

Algorithm:

1. Define function $f(x)$ and specify Error (E)
2. Input initial value x_0 and x_1
   Such that $f(x_0) * f(x_1) < 0$,
   Otherwise
   Print message " input right gauses"
   goto step 2
3. Compute:

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

$$\text{Error} = \left| \frac{x_2 - x_0}{x_2} \right|$$

4. If $(f(x_2) * f(x_1) < 0)$
   then
   replace $x_2 = x_0$
   else
   replace $x_1 = x_0$
5. if (Error $> E$)
   goto step 3
   else
   Print x_2 as a root

Source code

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define f(x) (x*x-2*x-5)
#define E (0.0001)
void main()
  {
   float x0,x1,x2, Error;
   up:
      printf("Enter initial gouses\n");
      scanf("%f %f", &x0,&x1);
      if(f(x0)*f(x1)>0)
        {
         printf("The root does not exist on those gauses\n");
         goto up;
        }
      else
      {
       next:
          x2 = (x0 - ((f(x0)*(x1-x0))/(f(x1)-f(x0))));
          Error = fabs((x2-x0)/x2);
          if(f(x2)*f(x1)<0)
            {
              x0=x2;
            }
          else
            {
              x0 =x1;
            }
        if(Error>E)
          {
            goto next;
          }
        else
            {
              printf("The root is %f\n", x2);
            }
}
```

Output:

```
Enter initial gauses
1
2
The root does not exist on those gauses
Enter initial gauses
-1
-2
The root is -1.4494
```

Solution of non linear equation using Secant Method

Objective:

To solve the equation using secant method

Theory

Secant method is open end method uses two initial estimates to compute the root of given equation

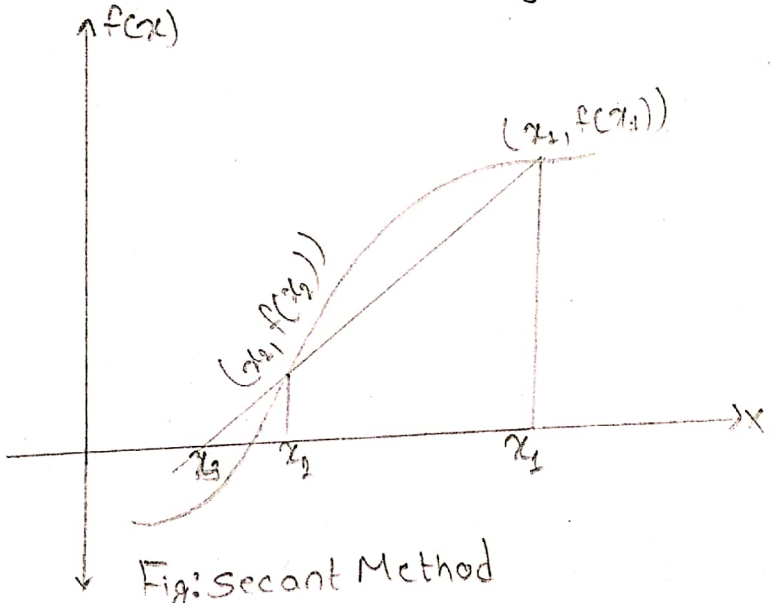


Fig: Secant Method

It can use two points x_1 and x_2 as shown in figure above as starting values although they do not bracket the root

The approximate value of the root is calculated by:

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

Algorithm:

1. Define function f(x) and specified error (E)
2. Input the initial gauses say x_1 and x_2
3. Compute

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

$$\text{Error} = \left| \frac{x_3 - x_2}{x_3} \right|$$

4. Exchange the value as $x_1 = x_2$ and $x_2 = x_3$
5. If (Error > E)
   then goto step 3

   else
   print x_3 as root
6. End

Source code

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define f(x) (x*x-2*x-5)
#define E (0.0001)
Void main()
 {
  float x0, x1, x2, Error;
  Printf("Enter initial gauses\n");
  Scanf("%f%f", &x0, &x1);
Up:
  x2 = (x1-((f(x1)* (x1-x0))/(f(x1)-f(x0))));
  Error = fabs(x2-x1)/x2);
  if(Error >E)
   {
     x0=x1;
     x1=x2;
   goto up;
   }
 else
   {
    Printf("Root is%f", x2);
   }
}
```

Output:

```
Enter initial gouses:
2
4
Root is 3.4494
```

Solution of non-linear equation using Newton Raphson Method

Objective

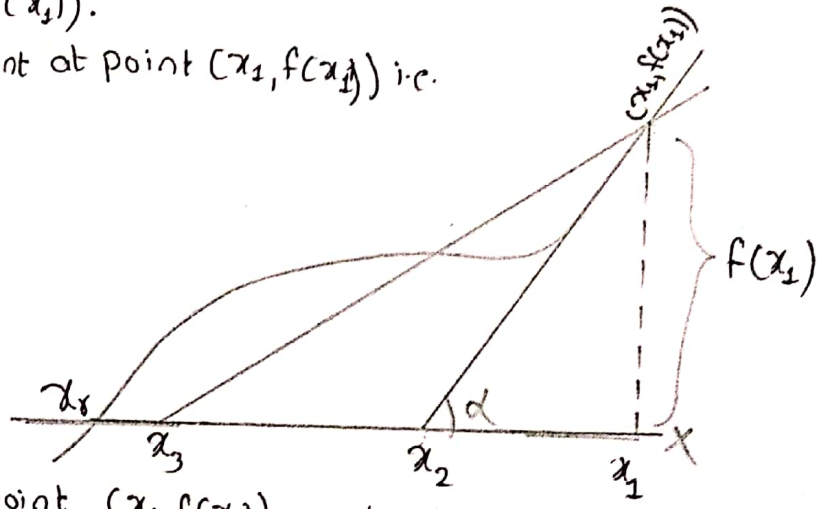
To solve the equation using newton Raphson method

Theory:

Say $f(x)=0$ be an equation initial gauses x such that a tangent be drown at point $(x_1, f(x_1))$.

Now, the slope of tangent at point $(x_1, f(x_1))$ i.e.

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



The first derivative of point $(x_1, f(x_1))$ provides the slope of tangent

so, $\tan\alpha = f'(x)|_{at\ x=x_1}$

$= f'(x_1)$

Now,

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

$$\text{or, } x_1 - x_2 = \frac{f(x_1)}{f'(x_1)}$$

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

for n^{th} step

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

∴ This is approximate formula for Newton Raphson Method

Algorithm

1. Define the function $f(x)$ and $f'(x)$
2. Take the initial gauses x_0
3. Check $f'(x_0) \neq 0$ or not
   if yes goto step 4
   else
   print the message there is no convergence of root
4. compute

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

$$\text{Error} = \left| \frac{x_1 - x_0}{x_1} \right|$$

5. Exchange x_0 by x_1 i.e. $x_0 = x_1$
6. If Error $> E$
   then goto step 3

   else

   print x_1 as root
7. End

Source code:

```
#include<stdio.h>
#include <conio.h>
#include <math.h>
#define f(x) x*x - 3*x + 2
#define g(x) 2*x - 3
#define E (0.0001)
int main()
 {
  float x0,x1,Error;
  Printf("Enter inital gauses \n");
  scanf("%f",&x0);
  up:
    if((g(x0))!=0)
     {
       goto down;
     }
    else
     {
       Printf("There is no convergence of root.");
     }
down:
   x1 = x0 - ((f(x0))/(g(x0)));
   Error = fabs((x1-x0)/x1);
   x0 = x1;
   if (Error > E)
     {
       goto up;
     }
   else
     {
       Printf("Root of f(x) = %f",x1);
     }
return 0;
}
```

Output

Enter inital gauses

3

Root of f(x) = 2.00

Solution of non linear equation using Fixed point iteration method

Objective

To solve the equation using Fixed point iteration method

Theory:

It is also known as, direct substitution method or method of sucessive approximation. It is applicable if the equation $f(x) = 0$ can be expressed in term of $x = g(x)$. If x_0 is the inibal approximation to the root, then the next approximation will be $x_1 = g(x_0)$ and next again will be $x_2 = g(x_1)$ and so on.

where,

$x = g(x)$ is known as fixed point equation

Algorithm

1. Define $g(x)$ and error level (E)
2. Take initial gauses as x_0
3. Compute,

$$x_1 = g(x_0)$$

$$\text{Error} = \left| \frac{x_1 - x_0}{x_1} \right|$$

4. Replace x_0 by x_1 i.e $x_0 = x_1$
5. If Error > E

   goto 'step 3

   else

   print x_0 as root

6. End

Source code

```
#include <stdio.h>
#include<math.h>
#define f(x) x*x-5
#define g(x) ((5/x)+x)/2
#define E (0.0001)
int main()
 {
   float x0, x1, Error;
   Printf("Enter initial gauses \n");
   scanf("%f", &x0);
up:
   x1 = g(x0);
   Error = fabs((x1-x0)/x1);
   if(Error > E)
    {
      x0 = x1;
     goto up;
    }
 else
   {
     Printf("Root of g(x) = %f", x1);
   }
return 0;
}
```

Output

Enter initial gauses

1

Root of g(x) = 2.2360

Computation of missing value from data set using linear interpolation

Objective:

To find out the missing value from given data set using linear interpolation

Theory:

Linear Interpolation

A method of finding the missing value from given data set by approximating linear line expressed as

$$f(x) = a_0 + a_1(x - x_0)$$

Say $(x_0, f(x_0))$ $(x_1, f(x_1))$ be the given dataset such that value at $f(x)$ at x be computed as

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

$$\text{or, } f(x) - f(x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

$$\text{or, } f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Compare,

$$f(x_0) = a_0 + a_1(x - x_0)$$

Where,

$$a_0 = f(x_0)$$

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Algorithm

1. Input number of set of data say n.
2. for i=0 upto n
   input the data set x[i] & y[i]
   end for
3. Input point for interpolation say (p)
4. Compute Lagrange coefficient as
   for i=0 upto n
   say term = 1
   for j=0 upto n
   if (i ≠ j) then

$$term = term * \frac{p - x[j]}{x[i] - x[j]}$$

   end if
   end for
   L[i] = term
   end for
5. Compute the interpolate value as
   Sum = 0
   for i=0 upto n
   Sum = sum + L[i] * y[i]
   end for
6. Print Sum as interpolate value
7 End

Source code:

```
#include <stdio.h>
#include <conio.h>
int main()
  {
    float x[100], y[100], p, sum=0, term, l[100];
    int i,j,n;
    Printf("Enter the number of data:");
    scanf("%d",&n);
    Printf ("Enter data:\n");
    for(i=0; i<n; i++)
      {
        Printf("x[%d] =",i);
        Scanf ("%f", &x[i]);
        Printf("y[%d]=", i);
        Scanf("%f",&y[i]);
      }
  Printf("Enter interpolation point:");
  scanf("%f", &p);
  for(i=0; i<n; i++)
    {
      term=1;
      for(j=0; j<n; j++)
        {
          if(i!=j)
            {
              term = term * (p-x[j])/(x[i] - x[j]);
            }
          l[i] = term;
        }
    }
    for(i=0; i<n; i++)
      {
        sum=sum+ l[i]* y[i];
      }
      Printf("Interpolated value at %.4f is %.4f ", p, sum);
      return 0;
}
```

Output

Enter number of data: 4

Enter data:

x[0] = 0

y[0] = 1

x[1] = 1

y[1] = -1

x[2] = 2

y[2] = -1

x[3] = 3

y[3] = 0

Enter interpolation point: 2.5

Interpolated value at 2.5 is -0.5625