Brige Vieta

It is a method to find a single root for a given function.

It's formula is the same with "Newton Raphson Method":

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

The difference is how to substitute in the formula for the values F(Xi) and F'(Xi).

To understand this difference, we must know "Horner Method".

-Horner Method:

Horner's Method

- For polynomial of degree m

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = \sum_{k=0}^{m} a_k x^k$$

- Divide by $(x\text{-}r)$

$$\frac{f(x)}{x-r} = \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m}{x-r} = b_1 + b_2 x + \ldots + b_m x^{m-1} + \frac{b_0}{x-r}$$

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = (x-r)\left(b_1 + b_2 x + \ldots + b_m x^{m-1}\right) + b_0$$

- b_0 is $f(r)$

Now we must know the relations between b's and a's:

Horner's Method

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = (x-r)\left(b_1 + b_2 x + \ldots + b_m x^{m-1}\right) + b_0$$

$$\begin{aligned} f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m &= \left(b_1 x + b_2 x^2 + \ldots + b_{m-1} x^{m-1} + b_m x^m\right) \\ &\quad + \left(-rb_1 - rb_2 x - \ldots - rb_m x^{m-1}\right) + b_0 \\ &= b_0 - rb_1 + x(b_1 - rb_2) + x^2(b_2 - rb_3) + \ldots + x^{m-1}(b_{m-1} - rb_m) + b_m x^m \end{aligned}$$

$$\left.\begin{aligned} b_m &= a_m \\ b_{m-1} &= a_{m-1} + rb_m \\ &\vdots \\ b_2 &= a_2 + rb_3 \\ b_1 &= a_1 + rb_2 \\ b_0 &= a_0 + rb_1 = f(r) \end{aligned}\right\} m\times,\ m+ \qquad \begin{aligned} b_m &= a_m \\ b_j &= a_j + rb_{j+1} \end{aligned} \qquad j = m-1, m-2, \ldots, 1, 0$$

From the previous we knew that to determine the value of the function at some point ‘r’

We can find it using iterative formula.

And we need ‘m’ operations of multiplication operations and ‘m’ operations of addition operations.

Where ‘m’ is the maximum power of the given equation.

-Example for Horner Method:

Horner’s Method

$$f_1(x) = b_1 + b_2 x + \ldots + b_m x^{m-1}$$

$$f(x) = x^2 - 3x + 2$$

$f(1)$

i	a_i	b_i
2	1	1
1	-3	-2
0	2	0

$$f_1(x) = x - 2$$

$f(2)$

i	a_i	b_i
3	1	1
2	-3	-1
1	0	-2
0	4	0

$$f_1(x) = x^2 - x - 2$$

$$f_1(x) = x^2 - x - 2$$

$f(2)$

i	a_i	b_i
2	1	1
1	-1	1
0	-2	0

2 is a root (i.e. a double root of original equation)

$$f_2(x) = x + 1$$

-What is the benefits of Horner in Brige VIeta?

From Horner, we manage to determine the value of a polynomial function for any value ‘r’

In minimum number of operation.

Then we can find the value of F(Xi) like this way.

-Then how to calculate the value of F’(Xi)?

This is the answer:

Birge-Vieta Method

- NR method with $f(x)$ and $f(x)$ evaluated using Horner’s method
- Once a root is found, reduce order of polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = (x-r)\left(b_1 + b_2 x + \ldots + b_m x^{m-1}\right) + b_0 = (x-r)h(x) + b_0$$

$$b_m = a_m$$

$$b_j = a_j + r b_{j+1} \qquad j = m-1, m-2, \ldots, 1, 0$$

$$f'(x) = h(x) + (x-r)h'(x)$$

$$f'(r) = h(r)$$

$$h(x) = b_1 + b_2 x + \ldots + b_m x^{m-1} = (x-r)\left(c_2 + c_3 x + \ldots + c_m x^{m-2}\right) + c_1$$

$$c_m = b_m$$

$$c_j = b_j + r c_{j+1} \qquad j = m-1, m-2, \ldots, 1$$

$$f(r) = b_0$$

$$f'(r) = h(r) = c_1$$

-The Procedure Brige Vieta:

Inputs:(func,Xo,tolerance,max iteration).

Where Xo is the start value for X.

Outputs:

1-Condition: equals 0 for any error happens like division by 0.

2-Table: contains nine columns and number of rows equals the number of iterations

Note every iteration has number of rows equals the maximum power+1.

(iteration , power from maximum power to 0 , a , b , c , x , error , time)

Where column a contains values from ao to am.

Note : a,b and c are specified above.

And b,c as the same like a , but c start from c1 not c0.

-How Does The Algorithm Work?

It contains a loop for each loop :

We calculate the values of the arrays (b,c)the values of the array a does not change throw the program.

Then calculate the value of Xi+1 ,error and time.

Then substitute in the form for values of F(X)=bo and F'(x)=C1.

Then we find new value to estimate root(Xi+1).

After this we calculate time and the absolute approximate error.

-Order of convergence is the same with newton it is a quadratic order.

Error Analysis

$$|\delta_{i+1}| \cong \frac{f''(\alpha)}{2f'(\alpha)}\delta_i^{\,2}$$

Where delta i+1 =absolute true error in the current iteration , delta I = absolute true error in the previous iteration.

-Advantages Of Brige Vieta:

1- it uses the same formula of newton with minimum number of mathematical operations.

2-it is fast .

-Pitfalls:

1-solve only polynomial equations.

2-we are not sure that it will converge to the root .

To be sure that it will converge the value of F’’(Xi)/2F’(Xi) smaller than 1.

3- if F’(Xi)=0 ,then we divide by zero.

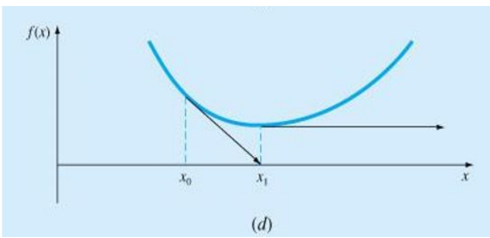


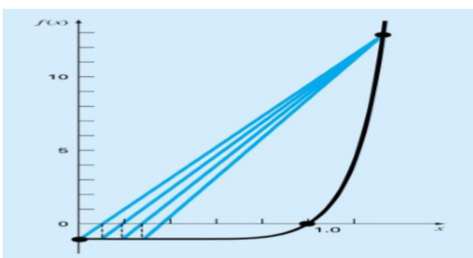
Figure (d)

A zero slope causes division by zero.

4-if the function is very flat ,it will converge but it will be very slow.

- Sometimes slow

$$f(x) = x^{10} - 1$$



iteration	x_i
0	0.5
1	51.65
2	46.485
3	41.8365
4	37.65285
5	33.8877565
...	...
40	1.002316024
41	1.000023934
42	1.000000003
43	1.000000000

46

5-the root is inflection point ,maximum or minimum point.

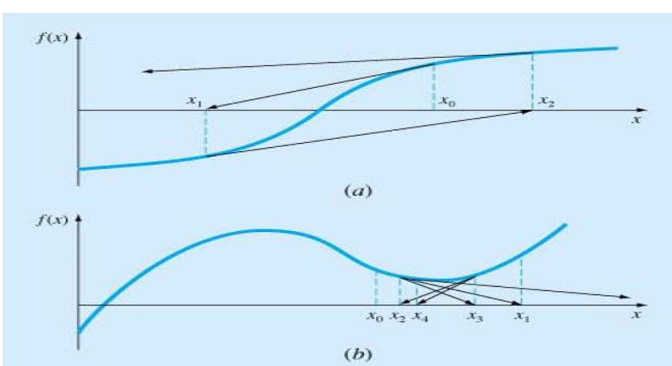


Figure (a)

An inflection point ($f''(x)=0$) at the vicinity of a root causes divergence.

Figure (b)

A local maximum or minimum causes oscillations.

6- it may jump from one location close to one root to a location that is several roots away.

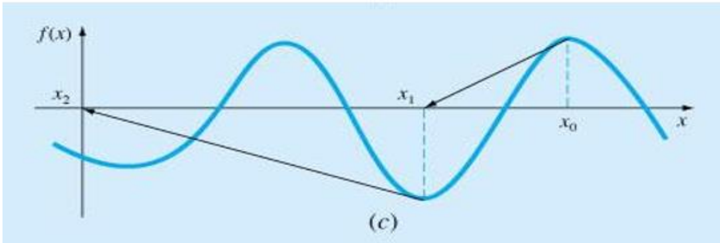


Figure (c)
It may jump from one location close to one root to a location that is several roots away.

-Pseudo Code For Algorithm.

Brige_Vieta(func,Xo,tolerance,maxIteration){

```
help=parsing(func,Xo)

fxIndex=help(1,2)+1

If (help(fxIndex-1,5)=0)

table='Error division by 0';
   condition=0;
else

numOfRows=fxIndex;

 previous=x0;
   next=previous-(help(fxIndex,4)/help(fxIndex-1,5));
   error=next-previous;
   previous=next;
   iterations=1;
   flag2=0;
       while (flag2==0&&iterations<maxIterations&&flag==1){
                      if (abs(error)<=tolerance)
                               flag2=1;
          counter=2;
           help(1,1)=iterations;
         help(1,6)=previous;
         help(1,7)=error;
         time=toc;
         help(1,8)=time;
       while(counter<=numOfRows){
         help(counter,1)=iterations;
         help(counter,6)=previous;
         help(counter,4)=help(counter,3)+previous*help(counter-1,4);
```

```
        help(counter,5)=help(counter,4)+previous*help(counter-1,5);
        help(counter,7)=error;
        time=toc;
        help(counter,8)=time;
        counter=counter+1;
}

    if (help(fxIndex-1,5)==0)
        table='Error division by 0';
        flag=0;
    else
        help(fxIndex,5)=help(fxIndex-1,5);
        hornerMatrix=[hornerMatrix;help];
        next=previous-(help(fxIndex,4)/help(fxIndex-1,5));
        error=next-previous;
        previous=next;
        iterations=iterations+1;
         table=hornerMatrix;
          condition=flag;
}
}
```

-Sample Runs

```
>> func='x^3-x^2-10x+7';
>>
>> [table,condition] = Birge_Vieta(func,1,1e-4,20)

table =

         0    3.0000    1.0000    1.0000    1.0000    1.0000    1.0000         0
         0    2.0000   -1.0000         0    1.0000    1.0000    1.0000         0
         0    1.0000  -10.0000  -10.0000   -9.0000    1.0000    1.0000         0
         0         0    7.0000   -3.0000   -9.0000    1.0000    1.0000         0
    1.0000    3.0000    1.0000    1.0000    1.0000    0.6667   -0.3333    0.1479
    1.0000    2.0000   -1.0000   -0.3333    0.3333    0.6667   -0.3333    0.1479
    1.0000    1.0000  -10.0000  -10.2222  -10.0000    0.6667   -0.3333    0.1479
    1.0000         0    7.0000    0.1852  -10.0000    0.6667   -0.3333    0.1480
    2.0000    3.0000    1.0000    1.0000    1.0000    0.6852    0.0185    0.1480
    2.0000    2.0000   -1.0000   -0.3148    0.3704    0.6852    0.0185    0.1480
    2.0000    1.0000  -10.0000  -10.2157   -9.9619    0.6852    0.0185    0.1480
    2.0000         0    7.0000    0.0003   -9.9619    0.6852    0.0185    0.1480
    3.0000    3.0000    1.0000    1.0000    1.0000    0.6852    0.0000    0.1480
    3.0000    2.0000   -1.0000   -0.3148    0.3704    0.6852    0.0000    0.1480
    3.0000    1.0000  -10.0000  -10.2157   -9.9619    0.6852    0.0000    0.1480
    3.0000         0    7.0000    0.0000   -9.9619    0.6852    0.0000    0.1480


condition =

     1
```

****

```
>> func='x^3-x-4';
[table,condition] = Birge_Vieta(func,1,1e-4,20)

table =

         0    3.0000    1.0000    1.0000    1.0000    1.0000    1.0000         0
         0    2.0000         0    1.0000    2.0000    1.0000    1.0000         0
         0    1.0000   -1.0000         0    2.0000    1.0000    1.0000         0
         0         0   -4.0000   -4.0000    2.0000    1.0000    1.0000         0
    1.0000    3.0000    1.0000    1.0000    1.0000    3.0000    2.0000    0.0007
    1.0000    2.0000         0    3.0000    6.0000    3.0000    2.0000    0.0007
    1.0000    1.0000   -1.0000    8.0000   26.0000    3.0000    2.0000    0.0007
    1.0000         0   -4.0000   20.0000   26.0000    3.0000    2.0000    0.0007
    2.0000    3.0000    1.0000    1.0000    1.0000    2.2308   -0.7692    0.0008
    2.0000    2.0000         0    2.2308    4.4615    2.2308   -0.7692    0.0008
    2.0000    1.0000   -1.0000    3.9763   13.9290    2.2308   -0.7692    0.0008
    2.0000         0   -4.0000    4.8703   13.9290    2.2308   -0.7692    0.0008
    3.0000    3.0000    1.0000    1.0000    1.0000    1.8811   -0.3497    0.0008
    3.0000    2.0000         0    1.8811    3.7622    1.8811   -0.3497    0.0008
    3.0000    1.0000   -1.0000    2.5386    9.6158    1.8811   -0.3497    0.0008
    3.0000         0   -4.0000    0.7754    9.6158    1.8811   -0.3497    0.0008
    4.0000    3.0000    1.0000    1.0000    1.0000    1.8005   -0.0806    0.0008
    4.0000    2.0000         0    1.8005    3.6010    1.8005   -0.0806    0.0008
    4.0000    1.0000   -1.0000    2.2417    8.7252    1.8005   -0.0806    0.0008
    4.0000         0   -4.0000    0.0362    8.7252    1.8005   -0.0806    0.0008
    5.0000    3.0000    1.0000    1.0000    1.0000    1.7963   -0.0041    0.0008
    5.0000    2.0000         0    1.7963    3.5927    1.7963   -0.0041    0.0008
    5.0000    1.0000   -1.0000    2.2268    8.6804    1.7963   -0.0041    0.0008
    5.0000         0   -4.0000    0.0001    8.6804    1.7963   -0.0041    0.0008
    6.0000    3.0000    1.0000    1.0000    1.0000    1.7963   -0.0000    0.0009
    6.0000    2.0000         0    1.7963    3.5926    1.7963   -0.0000    0.0009
    6.0000    1.0000   -1.0000    2.2268    8.6803    1.7963   -0.0000    0.0009
```

```
    6.0000         0   -4.0000    0.0000    8.6803    1.7963   -0.0000    0.0009


condition =

     1
```

****

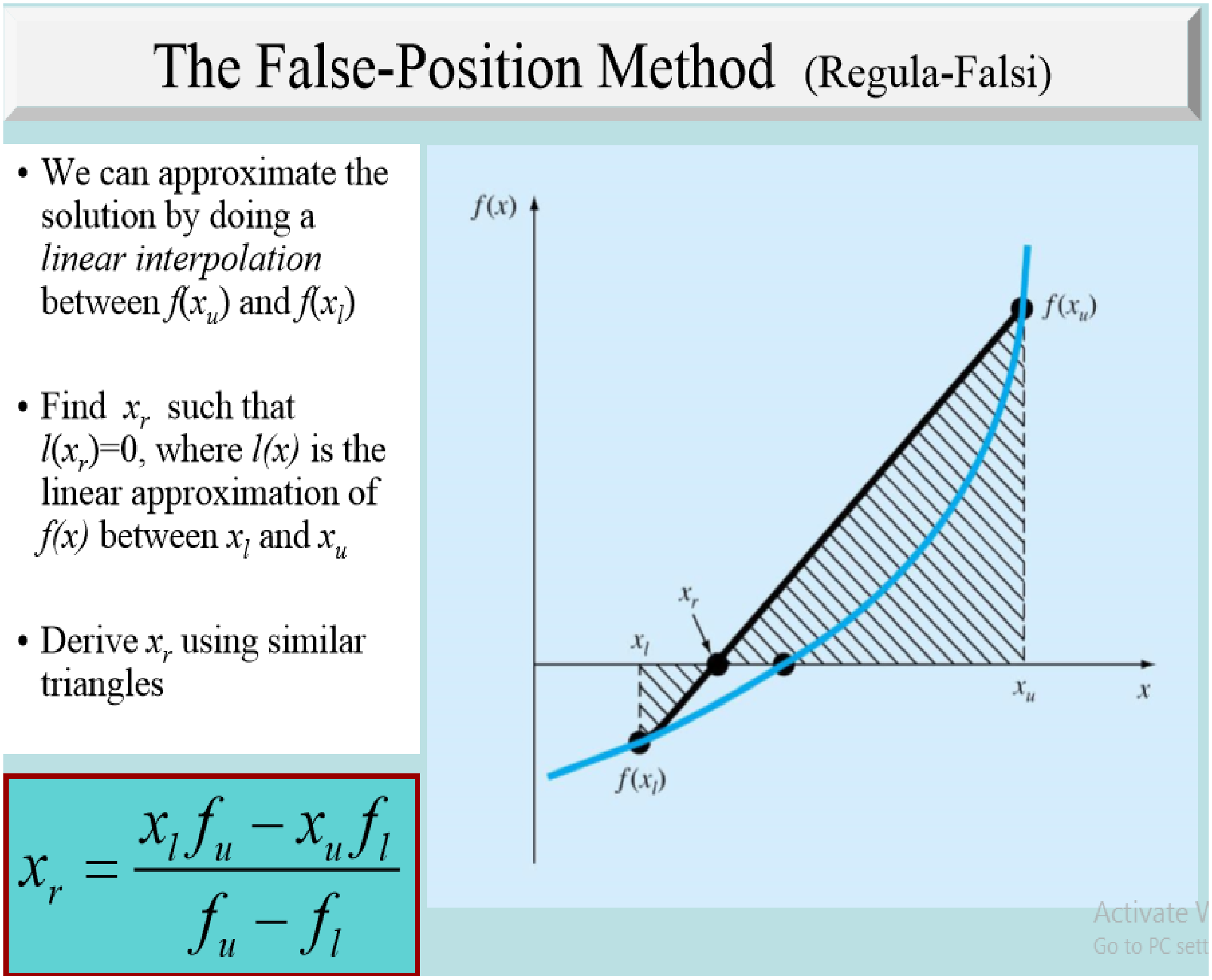
```
>> func='5x^5+x^2-x-4';
[table,condition] = Birge_Vieta(func,1,1e-4,20)

table =

         0    5.0000    5.0000    5.0000    5.0000    1.0000    1.0000         0
         0    4.0000         0    5.0000   10.0000    1.0000    1.0000         0
         0    3.0000         0    5.0000   15.0000    1.0000    1.0000         0
         0    2.0000    1.0000    6.0000   21.0000    1.0000    1.0000         0
         0    1.0000   -1.0000    5.0000   26.0000    1.0000    1.0000         0
         0         0   -4.0000    1.0000   26.0000    1.0000    1.0000         0
    1.0000    5.0000    5.0000    5.0000    5.0000    0.9615   -0.0385    0.0008
    1.0000    4.0000         0    4.8077    9.6154    0.9615   -0.0385    0.0008
    1.0000    3.0000         0    4.6228   13.8683    0.9615   -0.0385    0.0008
    1.0000    2.0000    1.0000    5.4450   18.7799    0.9615   -0.0385    0.0008
    1.0000    1.0000   -1.0000    4.2356   22.2932    0.9615   -0.0385    0.0008
    1.0000         0   -4.0000    0.0727   22.2932    0.9615   -0.0385    0.0008
    2.0000    5.0000    5.0000    5.0000    5.0000    0.9583   -0.0033    0.0009
    2.0000    4.0000         0    4.7914    9.5828    0.9583   -0.0033    0.0009
    2.0000    3.0000         0    4.5915   13.7745    0.9583   -0.0033    0.0009
    2.0000    2.0000    1.0000    5.3999   18.5998    0.9583   -0.0033    0.0009
    2.0000    1.0000   -1.0000    4.1746   21.9984    0.9583   -0.0033    0.0009
    2.0000         0   -4.0000    0.0005   21.9984    0.9583   -0.0033    0.0009
    3.0000    5.0000    5.0000    5.0000    5.0000    0.9583   -0.0000    0.0009
    3.0000    4.0000         0    4.7913    9.5826    0.9583   -0.0000    0.0009
    3.0000    3.0000         0    4.5913   13.7739    0.9583   -0.0000    0.0009
    3.0000    2.0000    1.0000    5.3996   18.5985    0.9583   -0.0000    0.0009
    3.0000    1.0000   -1.0000    4.1742   21.9964    0.9583   -0.0000    0.0009
    3.0000         0   -4.0000    0.0000   21.9964    0.9583   -0.0000    0.0009

condition =

     1
```

****

False Position Method

It is also known as "Regula-Falsi" ,and it is considered an iterative method of "Bracketing Method".

And this is the derivation of the formula.



-False Position Procedure:

Inputs and Outputs:

A-It takes five inputs (The equation , Xl , Xu , tolerance , maximum iterations)

1-The Equation : Is the equation we want to find one root to it in the interval from Xl to Xu.

2-Xl:is lower bound of the interval.

3-Xu:is the upper bound of the interval.

4-Tolerance: the allowed absolute for the solution.

5-Maximum iterations : is the maximum number of the iterations after which we can stop even if the error greater than the tolerance

B- it gives as two outputs:

1-condition : it work as a Boolean variable if condition =0 this indicate some error has happened

*Like division by zero or the value of F(Xu)*F(Xl) greater than 0 ,and when this happen we do not continue to iterate.*

Where F(Xu) indicates the value of the given equation at X=Xu.

Where F(Xl) indicates the value of the given equation at X=Xl.

2-table :

This table contains nine columns and number iteration equal to the number of rows.

The nine columns(the number of iterations , Xl , Xu , Xr , F(Xl) , F(Xu) ,F(xr) , error,time).

-Xr: is the estimate value for the root where Xl<= Xr<=Xu.

And we can calculate it from :

$$x_r = \frac{x_l f_u - x_u f_l}{f_u - f_l}$$

Error: is the approximate absolute error ,E=Xr i- Xr i-1.

Error = the estimate value of the root of the current iteration - the estimate value of the root of the previous iteration.

Time = it is the time taken from the start of the program to the time to estimate the value of Xr for each iteration.

-How does the algorithm work?

It works in a simply way, there is a loop.

For each loop we calculate the value of Xr from the given formula.

And calculate error as mentioned above.

Give new value to Xu and Xl and we give them the values from this condition:

```
if (func(previous)*fXu<0)
    xLower=previous;
    fXl=func(xLower);
else
    xUpper=previous;
    fXu=func(xUpper);
end
```

Where previous= the value of the Xr in the previous iteration.

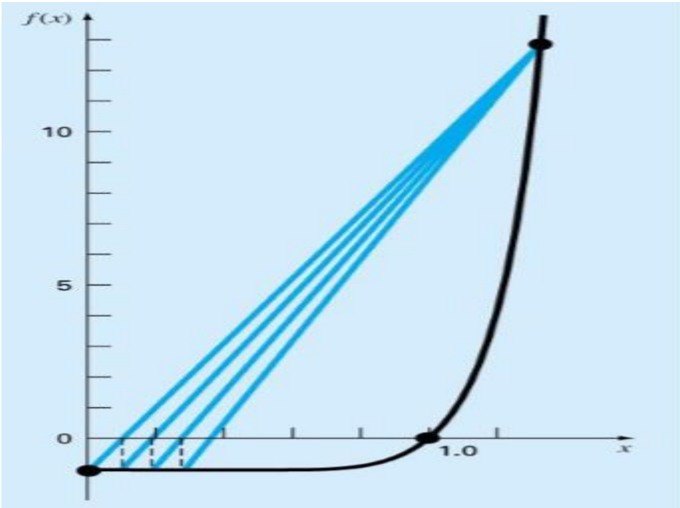
What is the advantage of this method?

It is a fast method.

Always converges for a single root.

Pitfalls of False Position.

1-when the give function is a flat function and this means one of the bound is stuck like this:



And this make it slow to converge to the root.

We can solve this problem by using "Bisection method" in the first time

Because this will make the interval smaller.

Xr = (Xl + Xu)/2.

*2- We can't solve it if (F(Xl)*F(Xu)<1)*

Because the formula dependent on this condition.

-Pseudo Code

```
falsePosition( funcStr,xl,xu,tolerance,maxIterations){
   xLower=xl;
   xUpper=xu;
   converter='@(x)';
   fallStr=strcat(converter,funcStr);
   func = str2func(fallStr);
   fXu=func(xUpper);
   fXl=func(xLower);
   flag=1;
   if (fXu*fXl>0)
      table='Erro f(Xl)*f(Xu)>0';
      condition=0;
   else
              tic;
              previous=(xLower*fXu-xUpper*fXl)/(fXu-fXl);
              time=toc;
              iterations=1;
```

```
          matrix=[iterations,xLower,xUpper,previous,fXl,fXu,func(previous)
         ,previous ,time];
          error=1000;
          while(abs(error)>tolerance&&iterations<maxIterations&&flag==1){
                if (func(previous)*fXu<0)
                   xLower=previous;
                   fXl=func(xLower);
                else
                   xUpper=previous;
                   fXu=func(xUpper);
                if (fXu-fXl==0)
                   flag=0;
                   table='Error Division by 0';
                else
                 xr=(xLower*fXu-xUpper*fXl)/(fXu-fXl);
                 time=toc;
                 error=xr-previous;
                 previous=xr;
                 iterations=iterations+1;
                 help=[iterations,xLower,xUpper,previous,fXl,fXu,func(previous),error,time];
                 matrix=[matrix ; help];
          }

table=matrix;
condition=flag;


}
```

Sample Runs.

```
>> func='x^3-x-4';
>> [table,condition]=falsePosition( func,1,2,1e-4,20)

table =

    1.0000    1.0000    2.0000    1.6667   -4.0000    2.0000   -1.0370    1.6667    0.0000
    2.0000    1.6667    2.0000    1.7805   -1.0370    2.0000   -0.1361    0.1138    0.0009
    3.0000    1.7805    2.0000    1.7945   -0.1361    2.0000   -0.0160    0.0140    0.0010
    4.0000    1.7945    2.0000    1.7961   -0.0160    2.0000   -0.0019    0.0016    0.0010
    5.0000    1.7961    2.0000    1.7963   -0.0019    2.0000   -0.0002    0.0002    0.0010
    6.0000    1.7963    2.0000    1.7963   -0.0002    2.0000   -0.0000    0.0000    0.0010


condition =

     1
```

****

```
>> func='3x^4+6.1x^3-2x^2+3x+2';
[table,condition]=falsePosition( func,-1,0,1e-4,20)

table =

    1.0000   -1.0000         0   -0.2469   -6.1000    2.0000    1.0567   -0.2469    0.0000
    2.0000   -1.0000   -0.2469   -0.3581   -6.1000    1.0567    0.4384   -0.1112    0.0001
    3.0000   -1.0000   -0.3581   -0.4011   -6.1000    0.4384    0.1587   -0.0430    0.0001
    4.0000   -1.0000   -0.4011   -0.4163   -6.1000    0.1587    0.0543   -0.0152    0.0001
    5.0000   -1.0000   -0.4163   -0.4215   -6.1000    0.0543    0.0182   -0.0052    0.0002
    6.0000   -1.0000   -0.4215   -0.4232   -6.1000    0.0182    0.0061   -0.0017    0.0002
    7.0000   -1.0000   -0.4232   -0.4238   -6.1000    0.0061    0.0020   -0.0006    0.0002
    8.0000   -1.0000   -0.4238   -0.4240   -6.1000    0.0020    0.0007   -0.0002    0.0002
    9.0000   -1.0000   -0.4240   -0.4240   -6.1000    0.0007    0.0002   -0.0001    0.0002


condition =

     1
```

```
>> func='3x^4+6.1x^3-2x^2+3x+2';
[table,condition]=falsePosition( func,1,2,1e-4,20)

table =

Erro f(Xl)*f(Xu)>0


condition =

     0
```

****

```
>> func='8x^5-3x^4+6.1x^3-2x^2+3x+2';
[table,condition]=falsePosition( func,-1,2,1e-4,20)

table =

    1.0000   -1.0000    2.0000   -0.7822  -20.1000  256.8000   -7.9563   -0.7822    0.0000
    2.0000   -0.7822    2.0000   -0.6986   -7.9563  256.8000   -5.1980    0.0836    0.0015
    3.0000   -0.6986    2.0000   -0.6451   -5.1980  256.8000   -3.8181    0.0535    0.0016
    4.0000   -0.6451    2.0000   -0.6063   -3.8181  256.8000   -2.9751    0.0388    0.0016
    5.0000   -0.6063    2.0000   -0.5765   -2.9751  256.8000   -2.4034    0.0298    0.0016
    6.0000   -0.5765    2.0000   -0.5526   -2.4034  256.8000   -1.9897    0.0239    0.0016
    7.0000   -0.5526    2.0000   -0.5330   -1.9897  256.8000   -1.6766    0.0196    0.0016
    8.0000   -0.5330    2.0000   -0.5165   -1.6766  256.8000   -1.4316    0.0164    0.0017
    9.0000   -0.5165    2.0000   -0.5026   -1.4316  256.8000   -1.2353    0.0140    0.0017
   10.0000   -0.5026    2.0000   -0.4906   -1.2353  256.8000   -1.0747    0.0120    0.0017
   11.0000   -0.4906    2.0000   -0.4802   -1.0747  256.8000   -0.9413    0.0104    0.0017
   12.0000   -0.4802    2.0000   -0.4712   -0.9413  256.8000   -0.8292    0.0091    0.0017
   13.0000   -0.4712    2.0000   -0.4632   -0.8292  256.8000   -0.7338    0.0080    0.0017
   14.0000   -0.4632    2.0000   -0.4562   -0.7338  256.8000   -0.6519    0.0070    0.0018
   15.0000   -0.4562    2.0000   -0.4500   -0.6519  256.8000   -0.5812    0.0062    0.0018
   16.0000   -0.4500    2.0000   -0.4444   -0.5812  256.8000   -0.5197    0.0055    0.0018
   17.0000   -0.4444    2.0000   -0.4395   -0.5197  256.8000   -0.4658    0.0049    0.0018
   18.0000   -0.4395    2.0000   -0.4351   -0.4658  256.8000   -0.4185    0.0044    0.0019
   19.0000   -0.4351    2.0000   -0.4311   -0.4185  256.8000   -0.3767    0.0040    0.0019
   20.0000   -0.4311    2.0000   -0.4276   -0.3767  256.8000   -0.3397    0.0036    0.0019


condition =

     1
```

```
>> pi=3.141592654;
 func='sin(x)+0.5';
[table,condition]=falsePosition( func,-1*pi/2,2*pi,1e-4,20)

table =

    1.0000   -1.5708    6.2832    2.3562   -0.5000    0.5000    1.2071    2.3562    0.0000
    2.0000   -1.5708    2.3562   -0.4206   -0.5000    1.2071    0.0917   -2.7768    0.0013
    3.0000   -1.5708   -0.4206   -0.5988   -0.5000    0.0917   -0.0637   -0.1782    0.0013
    4.0000   -0.5988   -0.4206   -0.5258   -0.0637    0.0917   -0.0019    0.0731    0.0013
    5.0000   -0.5258   -0.4206   -0.5237   -0.0019    0.0917   -0.0001    0.0021    0.0014
    6.0000   -0.5237   -0.4206   -0.5236   -0.0001    0.0917   -0.0000    0.0001    0.0014


condition =

     1
```

Bisection Method

1- Overview:

This is a bracketing method, and always converge, it works by getting the mid of an interval then substitute in the function, and if the result of this mid is multiplied value of the function at the upper bound or the lower one, and the result is positive then, that mean the mid and the bound are both under the X-axis or above it then there is no root will be found between that bound and the mid, so we update the that bound to be the mid, and the process continues until we find root.

2- Pseudo-Code:

```
For I = 1: numberOfIterations
        mid = (upper+lower)/2
        test = f(mid)
        if (test <= eps)
         iterations = i
         root = mid
         break;
        end
        if f(mid)*f(upper) > 0 then
         upper = mid
        else
         lower = mid
        end

end
```

3- Pitfalls:

1- Slow.
2- Need to find initial guesses for upper and lower bound.
3- No account is taken of the fact that if f(lower bound) is closer to zero, it is likely that root is closer to lower bound.

4- Examples:

```
f =

x^4-2*x^3-4*x^2+4*x+4

>> [x y]=BiSection(f,-2,-1)

x =

     0


y =

    1.0000   -2.0000   -1.5000   -1.0000  100.0000    0.0000
    2.0000   -1.5000   -1.2500   -1.0000    0.2500    0.0013
    3.0000   -1.5000   -1.3750   -1.2500    0.1250    0.0020
    4.0000   -1.5000   -1.4375   -1.3750    0.0625    0.0037
    5.0000   -1.4375   -1.4063   -1.3750    0.0313    0.0044
    6.0000   -1.4375   -1.4219   -1.4063    0.0156    0.0051
    7.0000   -1.4219   -1.4141   -1.4063    0.0078    0.0057
    8.0000   -1.4219   -1.4180   -1.4141    0.0039    0.0063
    9.0000   -1.4180   -1.4160   -1.4141    0.0020    0.0069
   10.0000   -1.4160   -1.4150   -1.4141    0.0010    0.0075
   11.0000   -1.4150   -1.4146   -1.4141    0.0005    0.0080
   12.0000   -1.4146   -1.4143   -1.4141    0.0002    0.0086
   13.0000   -1.4143   -1.4142   -1.4141    0.0001    0.0092
   14.0000   -1.4143   -1.4142   -1.4142    0.0001    0.0098
   15.0000   -1.4142   -1.4142   -1.4142    0.0000    0.0104
   16.0000   -1.4142   -1.4142   -1.4142    0.0000    0.0109
   17.0000   -1.4142   -1.4142   -1.4142    0.0000    0.0115
```

```
f =

x^3-x-1

>> [x y]=BiSection(f,1,2)

x =

     0


y =

    1.0000    1.0000    1.5000    2.0000  100.0000    0.0000
    2.0000    1.0000    1.2500    1.5000    0.2500    0.0033
    3.0000    1.2500    1.3750    1.5000    0.1250    0.0048
    4.0000    1.2500    1.3125    1.3750    0.0625    0.0063
    5.0000    1.3125    1.3438    1.3750    0.0313    0.0075
    6.0000    1.3125    1.3281    1.3438    0.0156    0.0079
    7.0000    1.3125    1.3203    1.3281    0.0078    0.0083
    8.0000    1.3203    1.3242    1.3281    0.0039    0.0088
    9.0000    1.3242    1.3262    1.3281    0.0020    0.0092
   10.0000    1.3242    1.3252    1.3262    0.0010    0.0096
   11.0000    1.3242    1.3247    1.3252    0.0005    0.0100
   12.0000    1.3247    1.3250    1.3252    0.0002    0.0105
   13.0000    1.3247    1.3248    1.3250    0.0001    0.0109
   14.0000    1.3247    1.3248    1.3248    0.0001    0.0113
   15.0000    1.3247    1.3247    1.3248    0.0000    0.0117
   16.0000    1.3247    1.3247    1.3247    0.0000    0.0122
   17.0000    1.3247    1.3247    1.3247    0.0000    0.0126
```

```
f =

exp(-1*x)-x

>> [x y]=BiSection(f,0,1)

x =

     0


y =

    1.0000         0    0.5000    1.0000  100.0000    0.0000
    2.0000    0.5000    0.7500    1.0000    0.2500    0.0012
    3.0000    0.5000    0.6250    0.7500    0.1250    0.0017
    4.0000    0.5000    0.5625    0.6250    0.0625    0.0022
    5.0000    0.5625    0.5938    0.6250    0.0313    0.0026
    6.0000    0.5625    0.5781    0.5938    0.0156    0.0031
    7.0000    0.5625    0.5703    0.5781    0.0078    0.0035
    8.0000    0.5625    0.5664    0.5703    0.0039    0.0040
    9.0000    0.5664    0.5684    0.5703    0.0020    0.0044
   10.0000    0.5664    0.5674    0.5684    0.0010    0.0048
   11.0000    0.5664    0.5669    0.5674    0.0005    0.0053
   12.0000    0.5669    0.5671    0.5674    0.0002    0.0057
```

Fixed Point Method

1- Overview:

This is an open method, that may converge or diverge, it works by generating the magic function g(x) from the input function f(x), then iterate with initial point until g(x) is equal to the input x.

2- Pseudo-Code:

```
For I = 1: numberOfIterations
x2 = g(x1)
if(abs(x2-x1) < eps)
        I = iterations
        Root = x1
End
x1 = x2;

end
```

3- Pitfalls:

1- Guessing the initial point x0.
2- Multiple magic functions.
3- Diverge if $|g'(x)| > 1$
4- Number of iterations can't be known prior.

4- Examples:

Converge:

```
f =

exp(-1*x)-x

>> [x y]=Fixed(f,0)

x =

     0


y =

    1.0000         0    1.0000    1.0000    0.0008
    2.0000    1.0000    0.3679    0.6321    0.0014
    3.0000    0.3679    0.6922    0.3243    0.0017
    4.0000    0.6922    0.5005    0.1917    0.0019
    5.0000    0.5005    0.6062    0.1058    0.0020
    6.0000    0.6062    0.5454    0.0608    0.0022
    7.0000    0.5454    0.5796    0.0342    0.0023
    8.0000    0.5796    0.5601    0.0195    0.0025
    9.0000    0.5601    0.5711    0.0110    0.0026
   10.0000    0.5711    0.5649    0.0063    0.0028
   11.0000    0.5649    0.5684    0.0035    0.0029
   12.0000    0.5684    0.5664    0.0020    0.0030
   13.0000    0.5664    0.5676    0.0011    0.0032
   14.0000    0.5676    0.5669    0.0006    0.0033
   15.0000    0.5669    0.5673    0.0004    0.0035
   16.0000    0.5673    0.5671    0.0002    0.0036
   17.0000    0.5671    0.5672    0.0001    0.0037
   18.0000    0.5672    0.5671    0.0001    0.0039
   19.0000    0.5671    0.5672    0.0000    0.0040
   20.0000    0.5672    0.5671    0.0000    0.0041
   21.0000    0.5671    0.5671    0.0000    0.0043
   22.0000    0.5671    0.5671    0.0000    0.0044
```

Diverge:

```
>> f = '.95*(x^3)-5.9*(x^2)+10.9*x-6';
>> [ x y] = Fixed(f,3.5)

x =

     1


y =

  1.0e+203 *

    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    5.1338    5.1338    0.0000
```

```
x^2-x-3

>> [ x y] = Fixed(f,2)

x =

     1

    1.0000    2.0000    1.0000    1.0000    0.0008
    2.0000    1.0000   -2.0000    3.0000    0.0014
    3.0000   -2.0000    1.0000    3.0000    0.0017
    4.0000    1.0000   -2.0000    3.0000    0.0018
    5.0000   -2.0000    1.0000    3.0000    0.0022
    6.0000    1.0000   -2.0000    3.0000    0.0028
    7.0000   -2.0000    1.0000    3.0000    0.0032
    8.0000    1.0000   -2.0000    3.0000    0.0036
    9.0000   -2.0000    1.0000    3.0000    0.0039
   10.0000    1.0000   -2.0000    3.0000    0.0041
   11.0000   -2.0000    1.0000    3.0000    0.0043
   12.0000    1.0000   -2.0000    3.0000    0.0046
   13.0000   -2.0000    1.0000    3.0000    0.0048
   14.0000    1.0000   -2.0000    3.0000    0.0051
   15.0000   -2.0000    1.0000    3.0000    0.0053
   16.0000    1.0000   -2.0000    3.0000    0.0055
   17.0000   -2.0000    1.0000    3.0000    0.0058
   18.0000    1.0000   -2.0000    3.0000    0.0060
   19.0000   -2.0000    1.0000    3.0000    0.0062
   20.0000    1.0000   -2.0000    3.0000    0.0065
   21.0000   -2.0000    1.0000    3.0000    0.0067
   22.0000    1.0000   -2.0000    3.0000    0.0070
   23.0000   -2.0000    1.0000    3.0000    0.0072
   24.0000    1.0000   -2.0000    3.0000    0.0075
   25.0000   -2.0000    1.0000    3.0000    0.0077
   26.0000    1.0000   -2.0000    3.0000    0.0080
   27.0000   -2.0000    1.0000    3.0000    0.0082
   28.0000    1.0000   -2.0000    3.0000    0.0084
   29.0000   -2.0000    1.0000    3.0000    0.0087
   30.0000    1.0000   -2.0000    3.0000    0.0089
   31.0000   -2.0000    1.0000    3.0000    0.0092
   32.0000    1.0000   -2.0000    3.0000    0.0094
   33.0000   -2.0000    1.0000    3.0000    0.0096
```

Secant Method

1- Overview:

This is an open method, that may converge or diverge, this method uses the newton method, but it approximates the derivatives by finite divided difference, so it takes 2 point as initial points for iterations.

2- Pseudo-Code:

```
For I = 1: numberOfIterations
x3 = x2 - ( f(x2) * (x1-x2)) / (f(x1) - f(x2))
if(abs(x3-x2) < eps)
        I = iterations
        Root = x3
        Break;
End
x1 = x2
x2 = x3

end
```

3- Pitfalls:

1- 2 points needed to start method.
2- May converge and may diverge.
3- Number of iterations can't be known prior.
4- Division by zero if f(x1) = f(x2).
5- Work slowly with step curves.

4- Examples:

```
f =

x^2-2

>> [err,iterations] = Secant( f,.5,1)

err =

     0


iterations =

    1.0000    0.5000    1.0000    1.6667    0.6667    0.0005
    2.0000    1.0000    1.3750    1.6667    0.2917    0.0011
    3.0000    1.6667    1.4110    1.3750    0.0360    0.0017
    4.0000    1.3750    1.4143    1.4110    0.0033    0.0024
    5.0000    1.4110    1.4142    1.4143    0.0000    0.0031
```

```
f =

x^4-18*x^2+45

>> [err iterations] = Secant( f,1,2)

err =

     0


iterations =

    1.0000    1.0000    2.0000    1.7179    0.2821    0.0006
    2.0000    2.0000    1.7322    1.7179    0.0143    0.0013
    3.0000    1.7179    1.7321    1.7322    0.0002    0.0020
```

```
f =

exp(-1*x)-x

>> [err iterations] = Secant( f,0,1)

err =

     0


iterations =

    1.0000         0    1.0000    0.6127    0.3873    0.0006
    2.0000    1.0000    0.5638    0.6127    0.0489    0.0011
    3.0000    0.6127    0.5672    0.5638    0.0033    0.0017
    4.0000    0.5638    0.5671    0.5672    0.0000    0.0023
```

Newton Raphson Method

Introduction :

Newton's method is an extremely powerful technique in general the convergence is quadratic, first starts with an initial guess which is reasonably close to the true root, then the function is approximated by its tangent line, and one computes the x-intercept of this tangent line. This x-intercept will typically be a better approximation to the function's root than the original guess, and the method can be iterated.

pseudo-code :

- evaluate f'(x)

- get the new value of X_1 using the initial guess X_0 by the equation :

$$X_{i+1} = X_i - \frac{f(Xi)}{f'(Xi)}$$

- do that previous formula till a specific number of iterations or till reach for the desirable error.

- evaluate the absolute error of the current X_i :

$$E_i = X_{i+1} - X_i$$

- compare that error with the desirable error to decide if it is acceptable or not.

- After finishing specifying the root α .

- evaluate f''(x).

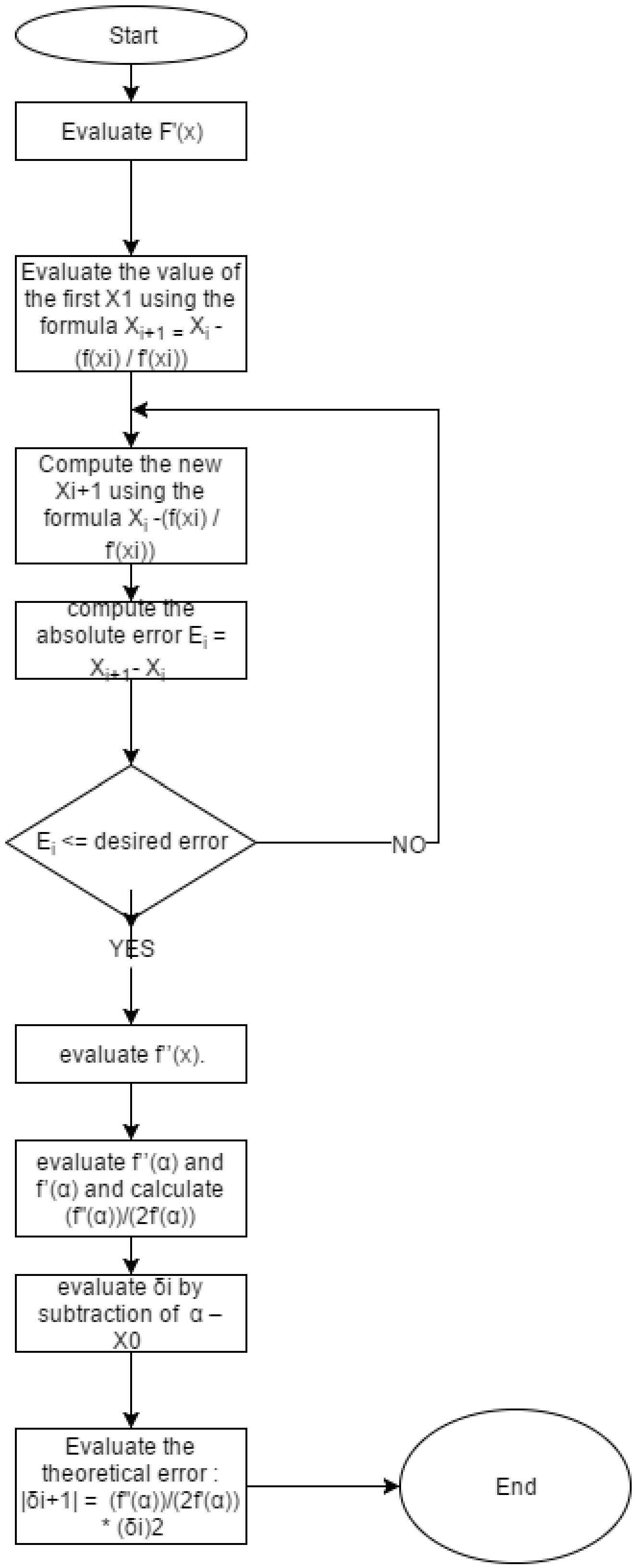
- evaluate f''(α) and f'(α) and calculate $\frac{f''(\alpha)}{2f'(\alpha)}$

- evaluate δ_i by subtraction of $\alpha - X_0$.

- evaluate the theoretical bound of error which equal :

$$|\delta_{i+1}| = \frac{f''(\alpha)}{2f'(\alpha)} * (\delta_i)^2$$

Flow Chart :



Start
Evaluate F'(x)
Evaluate the value of the first X1 using the formula $X_{i+1} = X_i$ - (f(xi) / f'(xi))
Compute the new Xi+1 using the formula X_i -(f(xi) / f'(xi))
compute the absolute error E_i = X_{i+1} - X_i
E_i <= desired error
NO
YES
evaluate f''(x).
evaluate f''(α) and f'(α) and calculate (f''(α))/(2f'(α))
evaluate δi by subtraction of α – X0
Evaluate the theoretical error : |δi+1| = (f''(α))/(2f'(α)) * (δi)2
End

Analysis for the behavior of different examples and GUI samples:

Example :

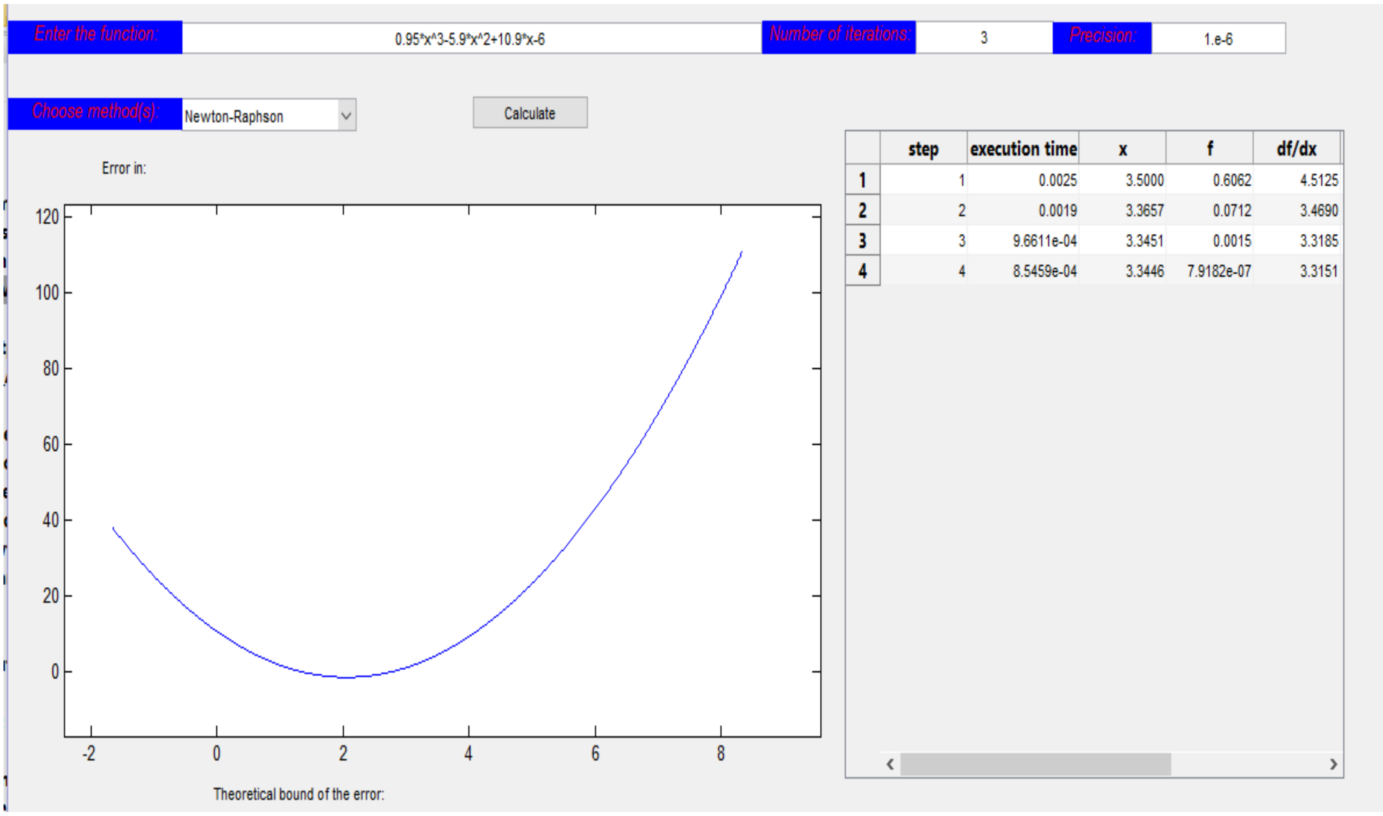
$f(x)= 0.95\ x^3 – 5.9\ x^2 + 10.9\ x – 6$

3 iterations, xi = 3.5

Solution :

Iteration	xi	f(xi)	f(xi+1)	error
1.0000	3.5000	0.6062	4.5125	0
2.0000	3.3657	0.0712	3.4690	0.1343
3.0000	3.3451	0.0015	3.3185	0.0205
4.0000	3.3446	0.0000	3.3151	0.0005

the error = 0.0264



Example :

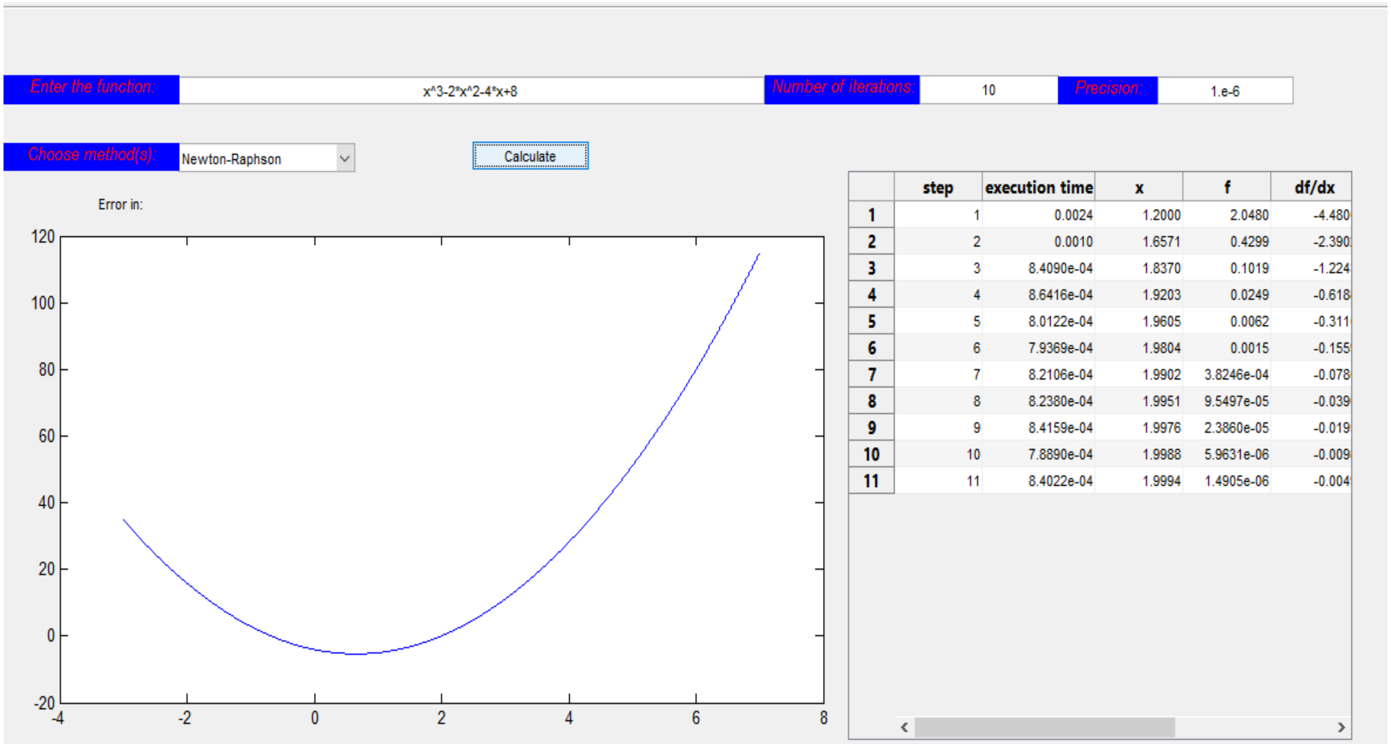
$f(x)= x^3 – 2x^2 - 4x + 8$

10 iterations, xi = 1.2

Solution :

Iteration	xi	f(xi)	f(xi+1)	Aerror			
1.0000	0.1014	1.2000	2.0480	-4.4800	0	0.7994	523.2539
2.0000	0.0011	1.6571	0.4299	-2.3902	0.4571	0.3422	95.9122
3.0000	0.0009	1.8370	0.1019	-1.2243	0.1799	0.1624	21.5923
4.0000	0.0008	1.9203	0.0249	-0.6188	0.0833	0.0791	5.1258
5.0000	0.0008	1.9605	0.0062	-0.3110	0.0403	0.0388	1.2356
6.0000	0.0008	1.9804	0.0015	-0.1559	0.0198	0.0190	0.2962
7.0000	0.0008	1.9902	0.0004	-0.0780	0.0098	0.0092	0.0690

8.0000	0.0008	1.9951	0.0001	-0.0390	0.0049	0.0043	0.0150
9.0000	0.0008	1.9976	0.0000	-0.0195	0.0024	0.0018	0.0027
10.0000	0.0008	1.9988	0.0000	-0.0098	0.0012	0.0006	0.0003
11.0000	0.0008	1.9994	0.0000	-0.0049	0.0006	0	0



	step	execution time	x	f	df/dx
1	1	0.0024	1.2000	2.0480	-4.480
2	2	0.0010	1.6571	0.4299	-2.390
3	3	8.4090e-04	1.8370	0.1019	-1.224
4	4	8.6416e-04	1.9203	0.0249	-0.618
5	5	8.0122e-04	1.9605	0.0062	-0.311
6	6	7.9369e-04	1.9804	0.0015	-0.155
7	7	8.2106e-04	1.9902	3.8246e-04	-0.078
8	8	8.2380e-04	1.9951	9.5497e-05	-0.039
9	9	8.4159e-04	1.9976	2.3860e-05	-0.019
10	10	7.8890e-04	1.9988	5.9631e-06	-0.009
11	11	8.4022e-04	1.9994	1.4905e-06	-0.004

Example :

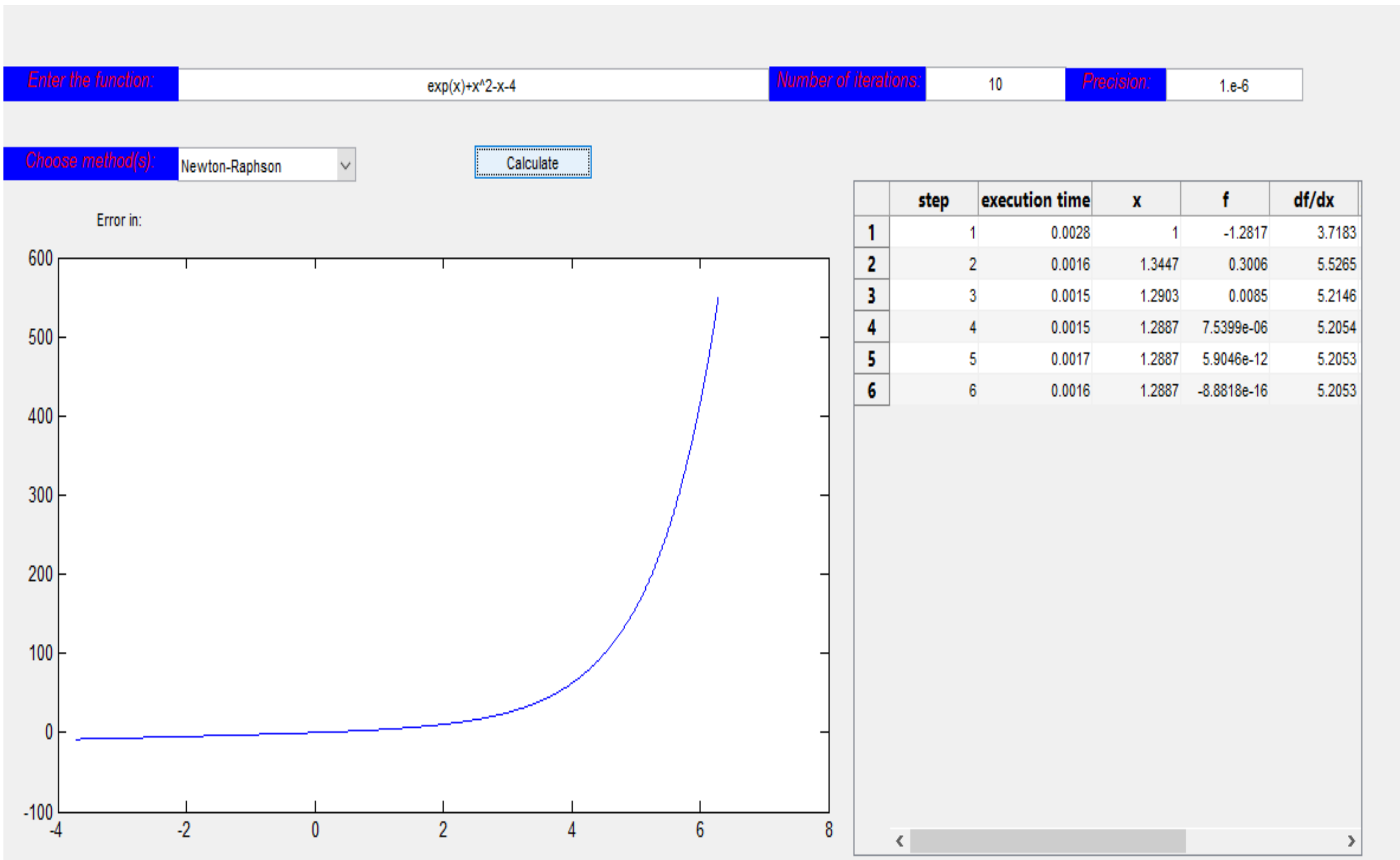
$f(x)= e^x + x^2 - x - 4$

10 iterations, xi = 1

Solution :

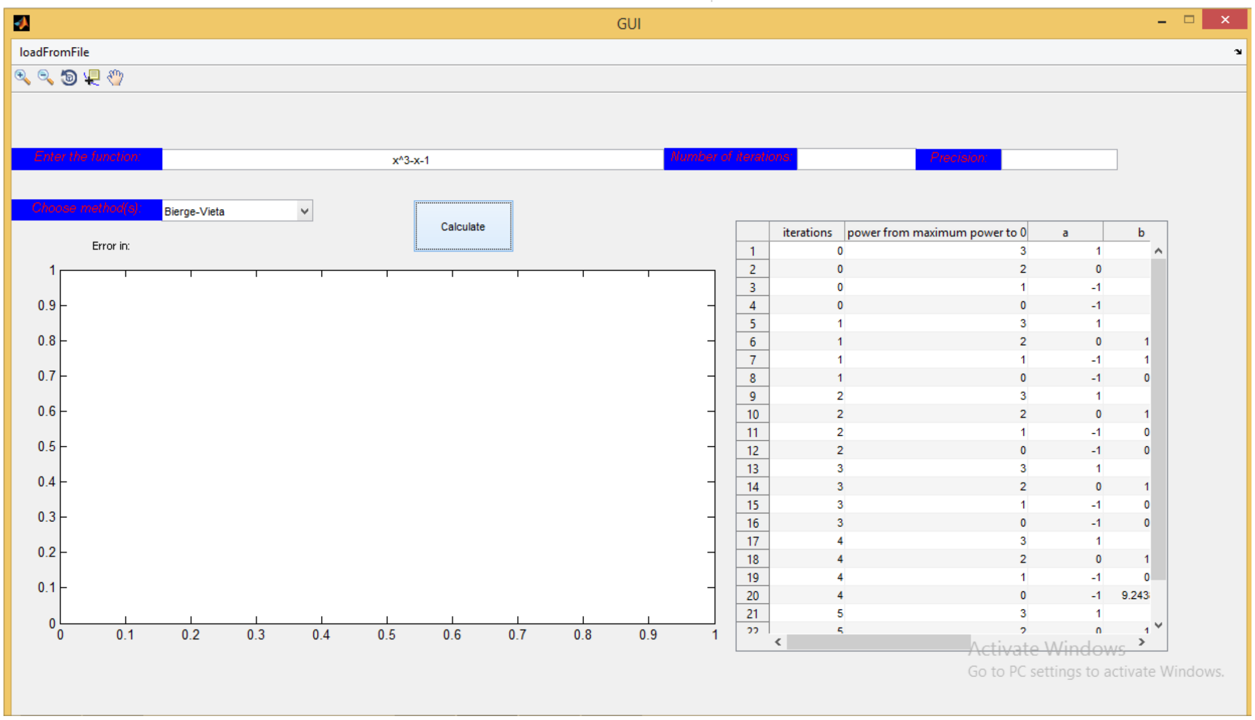
Iteration	xi	f(xi)	f(xi+1)	Aerror			
1.0000	0.1387	1.0000	-1.2817	3.7183	0	0.2887	0.0451
2.0000	0.0018	1.3447	0.3006	5.5265	0.3447	-0.0560	0.0017
3.0000	0.0016	1.2903	0.0085	5.2146	0.0544	-0.0016	0.0000

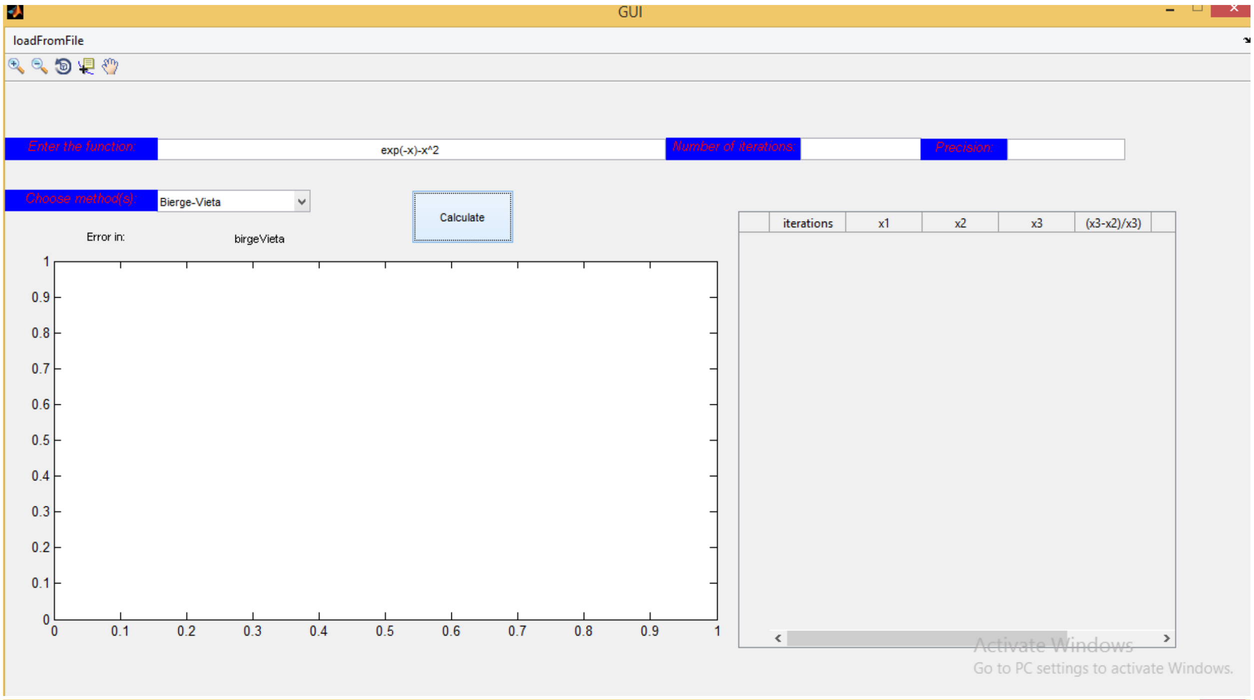
4.0000	0.0018	1.2887	0.0000	5.2054	0.0016	-0.0000	0.0000
5.0000	0.0016	1.2887	0.0000	5.2053	0.0000	0	0
6.0000	0.0015	1.2887	-0.0000	5.2053	0.0000	0	0

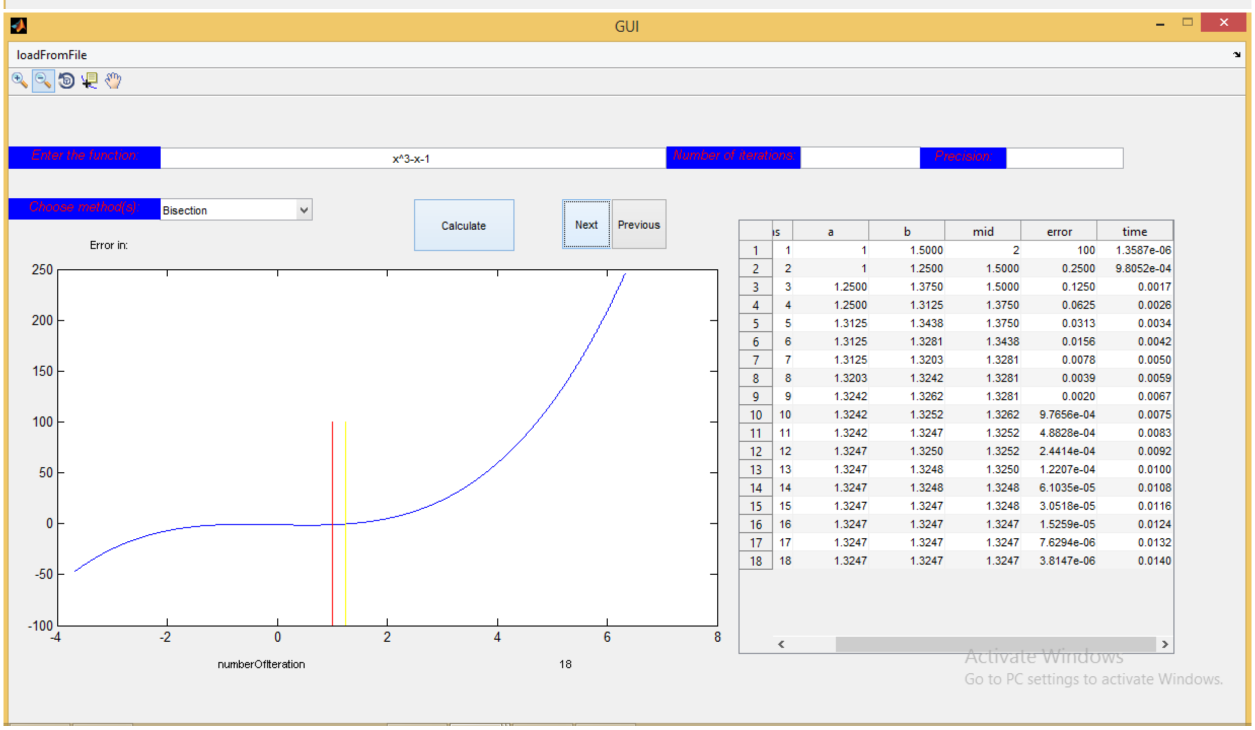


Screen Shots:

iterations	power from maximum power to 0	a	b	c	x	error	time
0.000000	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
0.000000	2.000000	0.000000	1.000000	2.000000	1.000000	1.000000	0.000000
0.000000	1.000000	-1.000000	0.000000	2.000000	1.000000	1.000000	0.000000
0.000000	0.000000	-1.000000	-1.000000	2.000000	1.000000	1.000000	0.000000
1.000000	3.000000	1.000000	1.000000	1.000000	1.500000	0.500000	0.050276
1.000000	2.000000	0.000000	1.500000	3.000000	1.500000	0.500000	0.050288
1.000000	1.000000	-1.000000	1.250000	5.750000	1.500000	0.500000	0.050291
1.000000	0.000000	-1.000000	0.875000	5.750000	1.500000	0.500000	0.050294
2.000000	3.000000	1.000000	1.000000	1.000000	1.347826	-0.152174	0.050315
2.000000	2.000000	0.000000	1.347826	2.695652	1.347826	-0.152174	0.050319
2.000000	1.000000	-1.000000	0.816635	4.449905	1.347826	-0.152174	0.050322
2.000000	0.000000	-1.000000	0.100682	4.449905	1.347826	-0.152174	0.050325
3.000000	3.000000	1.000000	1.000000	1.000000	1.325200	-0.022626	0.050334
3.000000	2.000000	0.000000	1.325200	2.650401	1.325200	-0.022626	0.050338
3.000000	1.000000	-1.000000	0.756156	4.268468	1.325200	-0.022626	0.050341

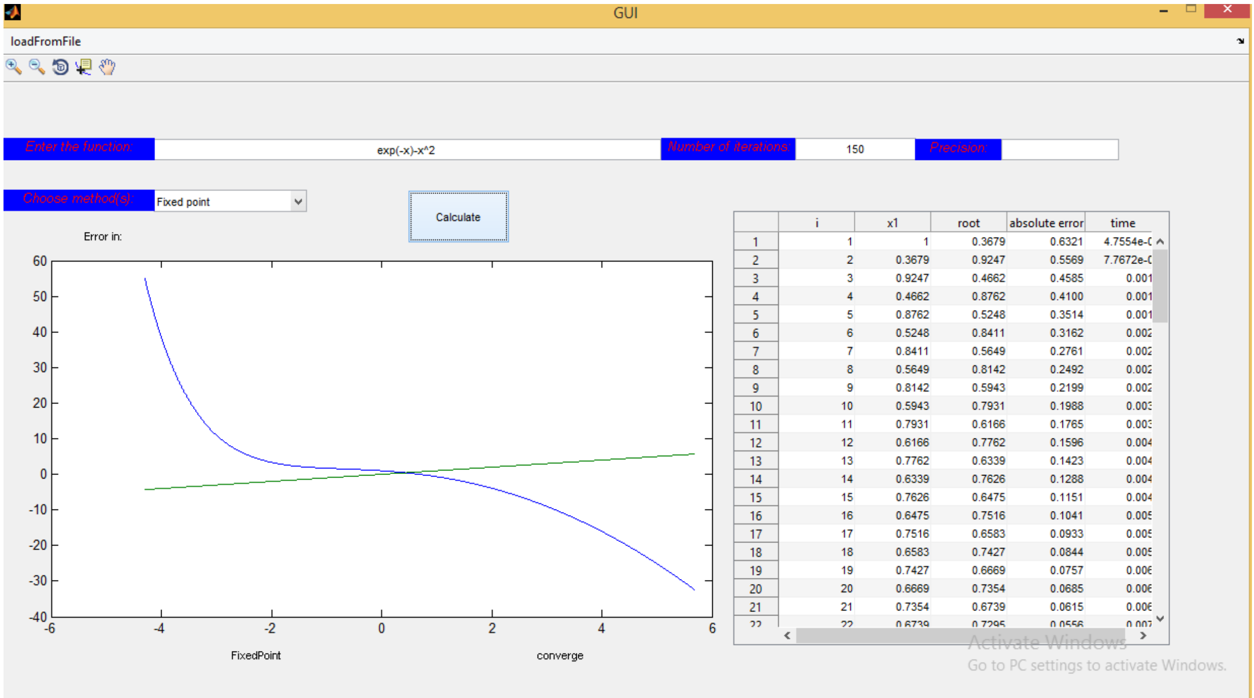




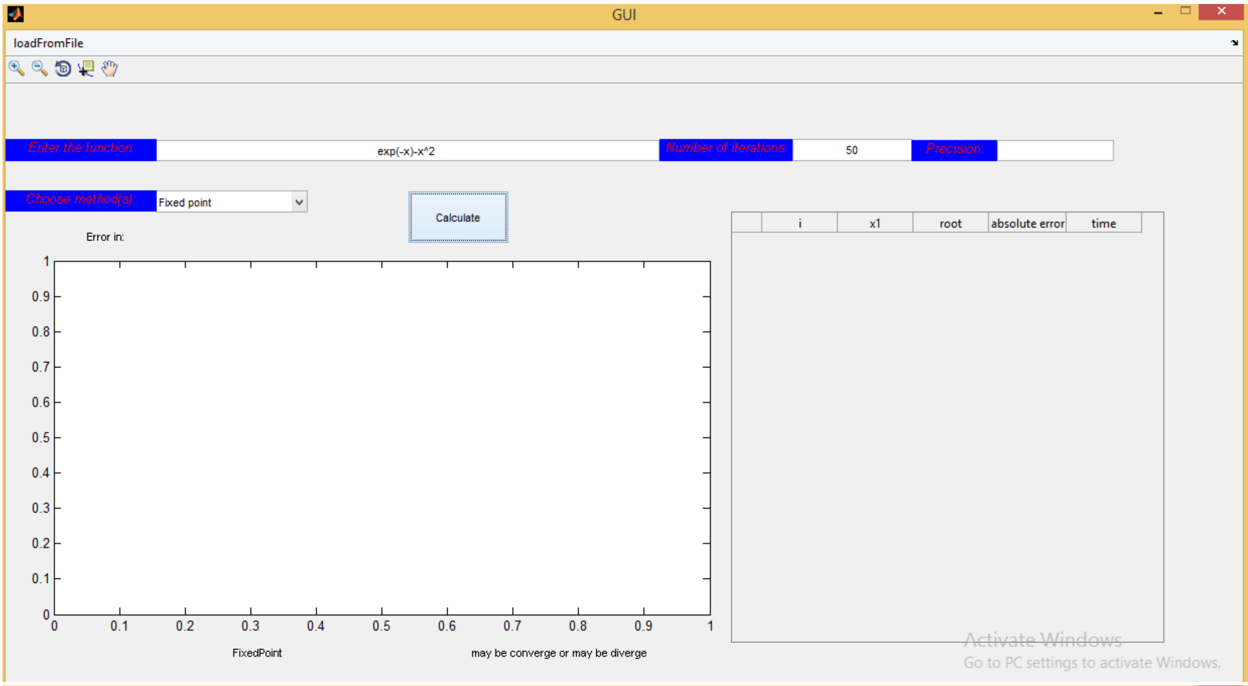


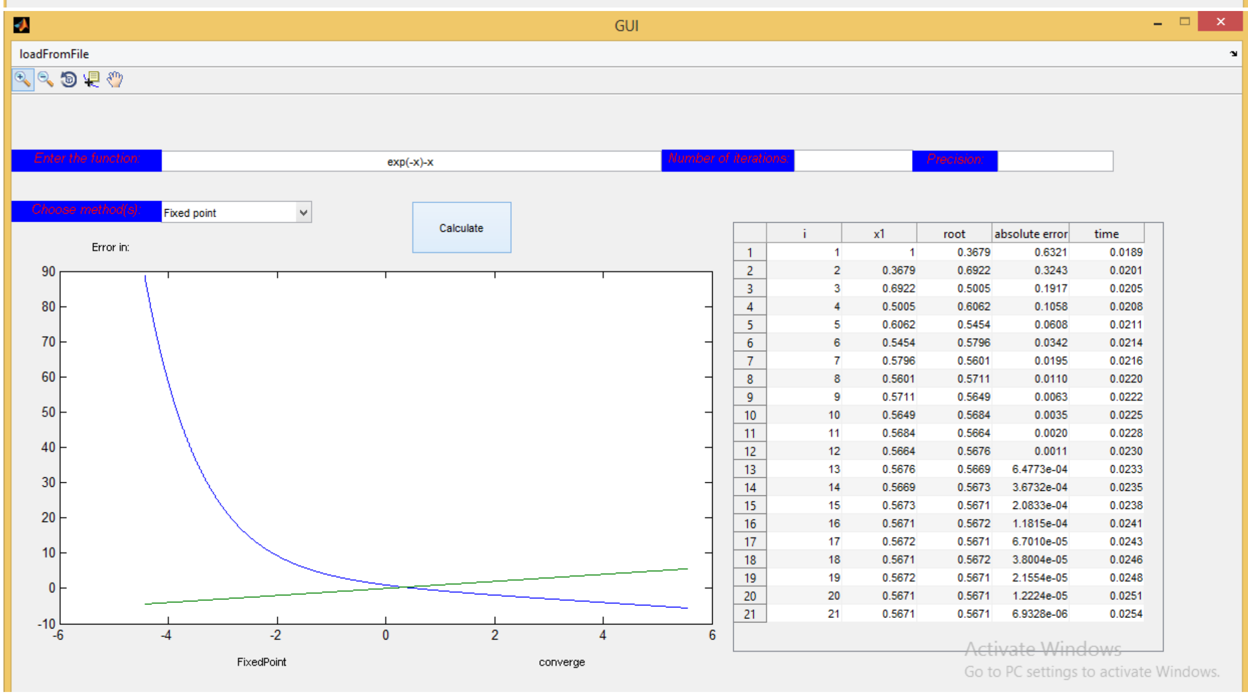
	s	a	b	mid	error	time
1	1	1	1.5000	2	100	1.3587e-06
2	2	1	1.2500	1.5000	0.2500	9.8052e-04
3	3	1.2500	1.3750	1.5000	0.1250	0.0017
4	4	1.2500	1.3125	1.3750	0.0625	0.0026
5	5	1.3125	1.3438	1.3750	0.0313	0.0034
6	6	1.3125	1.3281	1.3438	0.0156	0.0042
7	7	1.3125	1.3203	1.3281	0.0078	0.0050
8	8	1.3203	1.3242	1.3281	0.0039	0.0059
9	9	1.3242	1.3262	1.3281	0.0020	0.0067
10	10	1.3242	1.3252	1.3262	9.7656e-04	0.0075
11	11	1.3242	1.3247	1.3252	4.8828e-04	0.0083
12	12	1.3247	1.3250	1.3252	2.4414e-04	0.0092
13	13	1.3247	1.3248	1.3250	1.2207e-04	0.0100
14	14	1.3247	1.3248	1.3248	6.1035e-05	0.0108
15	15	1.3247	1.3247	1.3248	3.0518e-05	0.0116
16	16	1.3247	1.3247	1.3247	1.5259e-05	0.0124
17	17	1.3247	1.3247	1.3247	7.6294e-06	0.0132
18	18	1.3247	1.3247	1.3247	3.8147e-06	0.0140

i	a	b	mid	error	time
1.000000	1.000000	1.500000	2.000000	100.000000	0.000001
2.000000	1.000000	1.250000	1.500000	0.250000	0.000952
3.000000	1.250000	1.375000	1.500000	0.125000	0.001708
4.000000	1.250000	1.312500	1.375000	0.062500	0.002498
5.000000	1.312500	1.343750	1.375000	0.031250	0.003282
6.000000	1.312500	1.328125	1.343750	0.015625	0.004069
7.000000	1.312500	1.320313	1.328125	0.007813	0.004849
8.000000	1.320313	1.324219	1.328125	0.003906	0.005638
9.000000	1.324219	1.326172	1.328125	0.001953	0.006463
10.000000	1.324219	1.325195	1.326172	0.000977	0.007221
11.000000	1.324219	1.324707	1.325195	0.000488	0.008026
12.000000	1.324707	1.324951	1.325195	0.000244	0.008791
13.000000	1.324707	1.324829	1.324951	0.000122	0.009687
14.000000	1.324707	1.324768	1.324829	0.000061	0.010478
15.000000	1.324707	1.324738	1.324768	0.000031	0.011250


GUI
loadFromFile
Enter the function:
exp(-x)-x^2
Number of iterations:
150
Precision:
Choose method(s):
Fixed point
Calculate
Error in:
FixedPoint
converge
i x1 root absolute error time
1 1 1 0.3679 0.6321 4.7554e-
2 2 0.3679 0.9247 0.5569 7.7672e-
3 3 0.9247 0.4662 0.4585 0.001
4 4 0.4662 0.8762 0.4100 0.001
5 5 0.8762 0.5248 0.3514 0.001
6 6 0.5248 0.8411 0.3162 0.002
7 7 0.8411 0.5649 0.2761 0.002
8 8 0.5649 0.8142 0.2492 0.002
9 9 0.8142 0.5943 0.2199 0.002
10 10 0.5943 0.7931 0.1988 0.003
11 11 0.7931 0.6166 0.1765 0.003
12 12 0.6166 0.7762 0.1596 0.004
13 13 0.7762 0.6339 0.1423 0.004
14 14 0.6339 0.7626 0.1288 0.004
15 15 0.7626 0.6475 0.1151 0.004
16 16 0.6475 0.7516 0.1041 0.005
17 17 0.7516 0.6583 0.0933 0.005
18 18 0.6583 0.7427 0.0844 0.005
19 19 0.7427 0.6669 0.0757 0.006
20 20 0.6669 0.7354 0.0685 0.006
21 21 0.7354 0.6739 0.0615 0.006
Activate Windows
Go to PC settings to activate Windows.

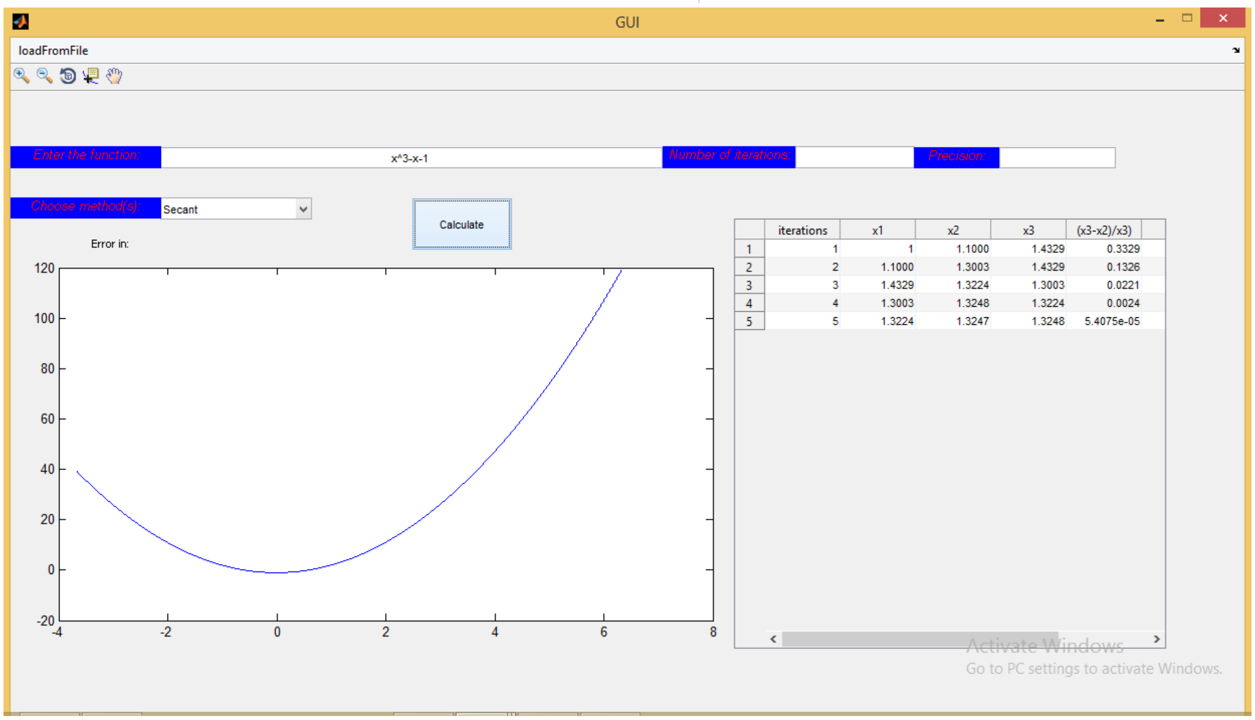
i	x1	root	error	time
1.000000	1.000000	0.367879	0.632121	0.001145
2.000000	0.367879	0.692201	0.324321	0.002001
3.000000	0.692201	0.500474	0.191727	0.002412
4.000000	0.500474	0.606244	0.105770	0.002735
5.000000	0.606244	0.545396	0.060848	0.002980
6.000000	0.545396	0.579612	0.034217	0.003225
7.000000	0.579612	0.560115	0.019497	0.003520
8.000000	0.560115	0.571143	0.011028	0.003768
9.000000	0.571143	0.564879	0.006264	0.004070
10.000000	0.564879	0.568429	0.003549	0.004314
11.000000	0.568429	0.566415	0.002014	0.004612
12.000000	0.566415	0.567557	0.001142	0.004858
13.000000	0.567557	0.566909	0.000648	0.005106
14.000000	0.566909	0.567276	0.000367	0.005350
15.000000	0.567276	0.567068	0.000208	0.005646



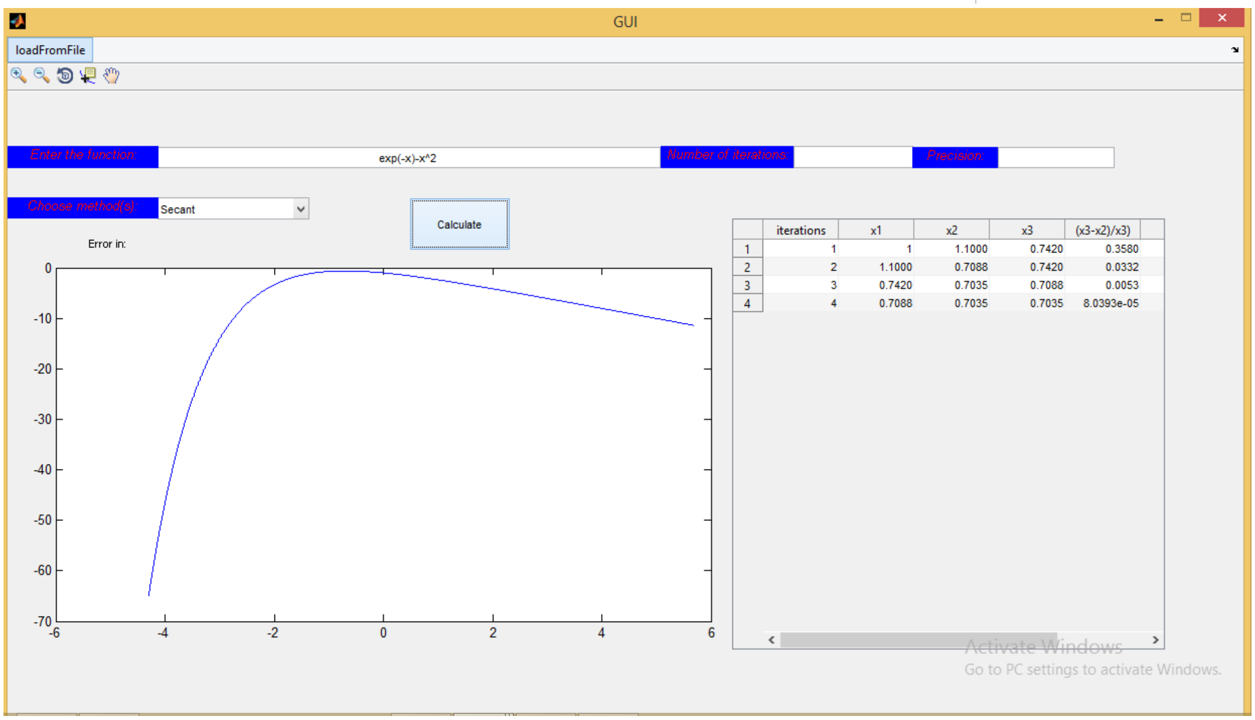


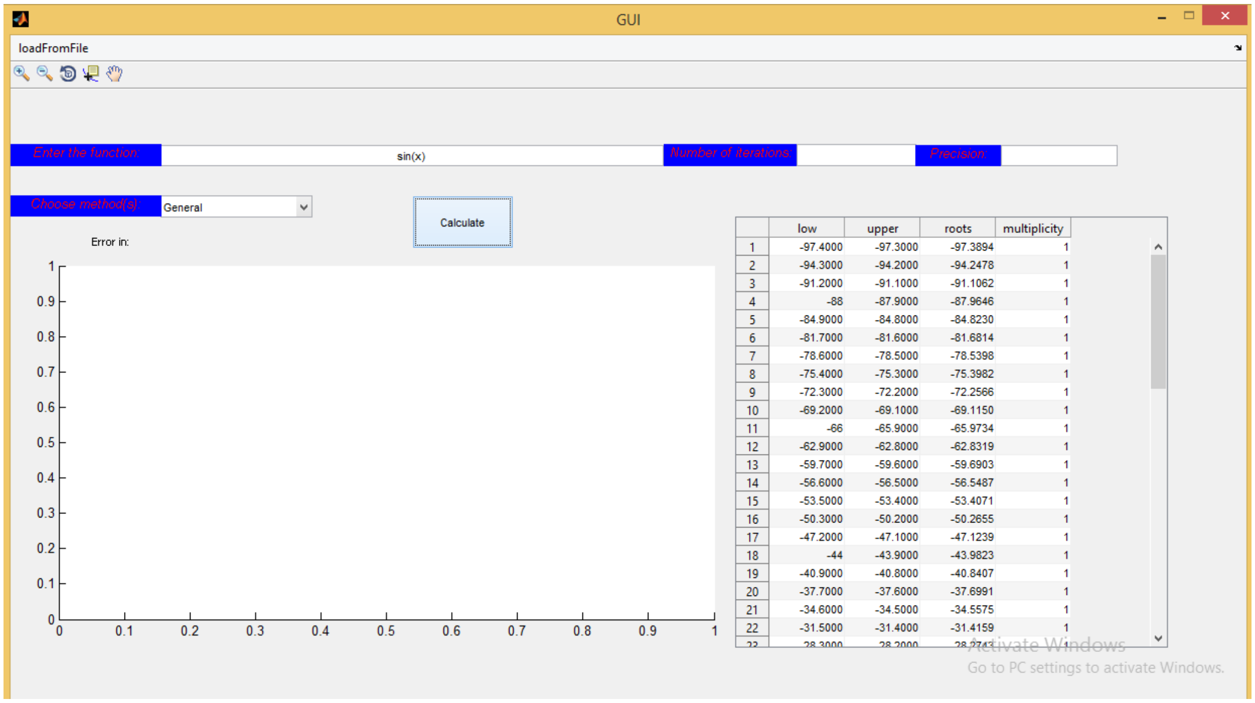
	i	x1	root	absolute error	time
1	1	1	0.3679	0.6321	0.0189
2	2	0.3679	0.6922	0.3243	0.0201
3	3	0.6922	0.5005	0.1917	0.0205
4	4	0.5005	0.6062	0.1058	0.0208
5	5	0.6062	0.5454	0.0608	0.0211
6	6	0.5454	0.5796	0.0342	0.0214
7	7	0.5796	0.5601	0.0195	0.0216
8	8	0.5601	0.5711	0.0110	0.0220
9	9	0.5711	0.5649	0.0063	0.0222
10	10	0.5649	0.5684	0.0035	0.0225
11	11	0.5684	0.5664	0.0020	0.0228
12	12	0.5664	0.5676	0.0011	0.0230
13	13	0.5676	0.5669	6.4773e-04	0.0233
14	14	0.5669	0.5673	3.6732e-04	0.0235
15	15	0.5673	0.5671	2.0833e-04	0.0238
16	16	0.5671	0.5672	1.1815e-04	0.0241
17	17	0.5672	0.5671	6.7010e-05	0.0243
18	18	0.5671	0.5672	3.8004e-05	0.0246
19	19	0.5672	0.5671	2.1554e-05	0.0248
20	20	0.5671	0.5671	1.2224e-05	0.0251
21	21	0.5671	0.5671	6.9328e-06	0.0254

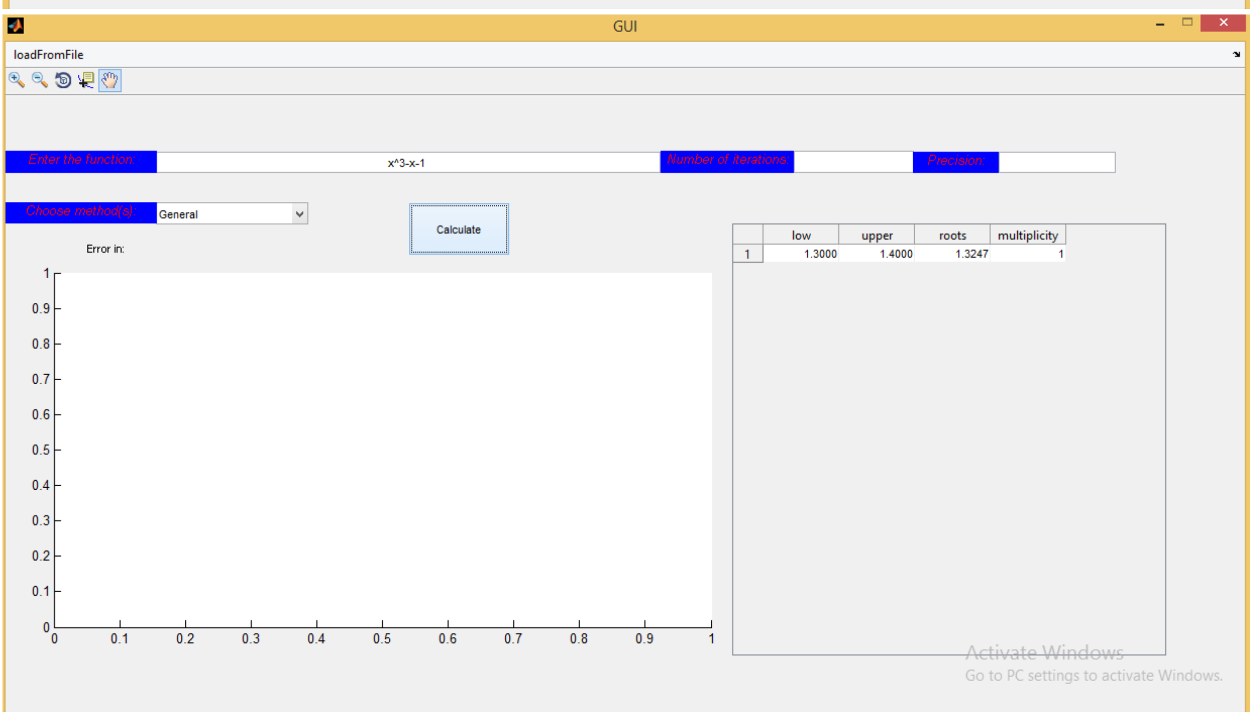
iterations	xl	xu	xr	f(xl)	f(xu)	f(xr)	error	time
1.000000	1.000000	2.000000	1.166667	-1.000000	5.000000	-0.578704	1.166667e+000	2.264492e-006
2.000000	1.166667	2.000000	1.253112	-0.578704	5.000000	-0.285363	8.644537e-002	1.009963e-004
3.000000	1.253112	2.000000	1.293437	-0.285363	5.000000	-0.129542	4.032537e-002	1.915760e-004
4.000000	1.293437	2.000000	1.311281	-0.129542	5.000000	-0.056588	1.784362e-002	2.364129e-004
5.000000	1.311281	2.000000	1.318989	-0.056588	5.000000	-0.024304	7.707482e-003	2.563405e-004
6.000000	1.318989	2.000000	1.322283	-0.024304	5.000000	-0.010362	3.294214e-003	2.762680e-004
7.000000	1.322283	2.000000	1.323684	-0.010362	5.000000	-0.004404	1.401576e-003	2.952897e-004
8.000000	1.323684	2.000000	1.324279	-0.004404	5.000000	-0.001869	5.951679e-004	3.147644e-004
9.000000	1.324279	2.000000	1.324532	-0.001869	5.000000	-0.000793	2.525248e-004	3.342390e-004
10.000000	1.324532	2.000000	1.324639	-0.000793	5.000000	-0.000336	1.071067e-004	3.537136e-004
11.000000	1.324639	2.000000	1.324685	-0.000336	5.000000	-0.000143	4.542186e-005	3.727353e-004
12.000000	1.324685	2.000000	1.324704	-0.000143	5.000000	-0.000060	1.926130e-005	3.922100e-004
13.000000	1.324704	2.000000	1.324712	-0.000060	5.000000	-0.000026	8.167608e-006	4.116846e-004

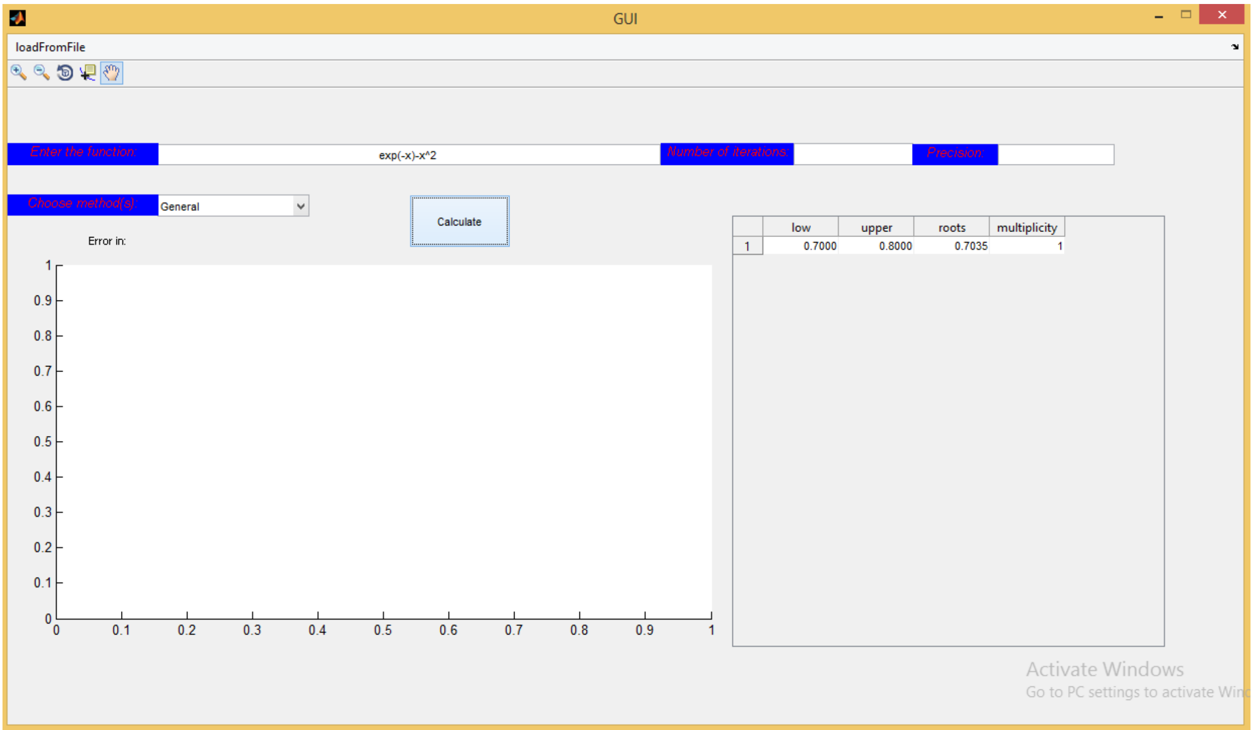


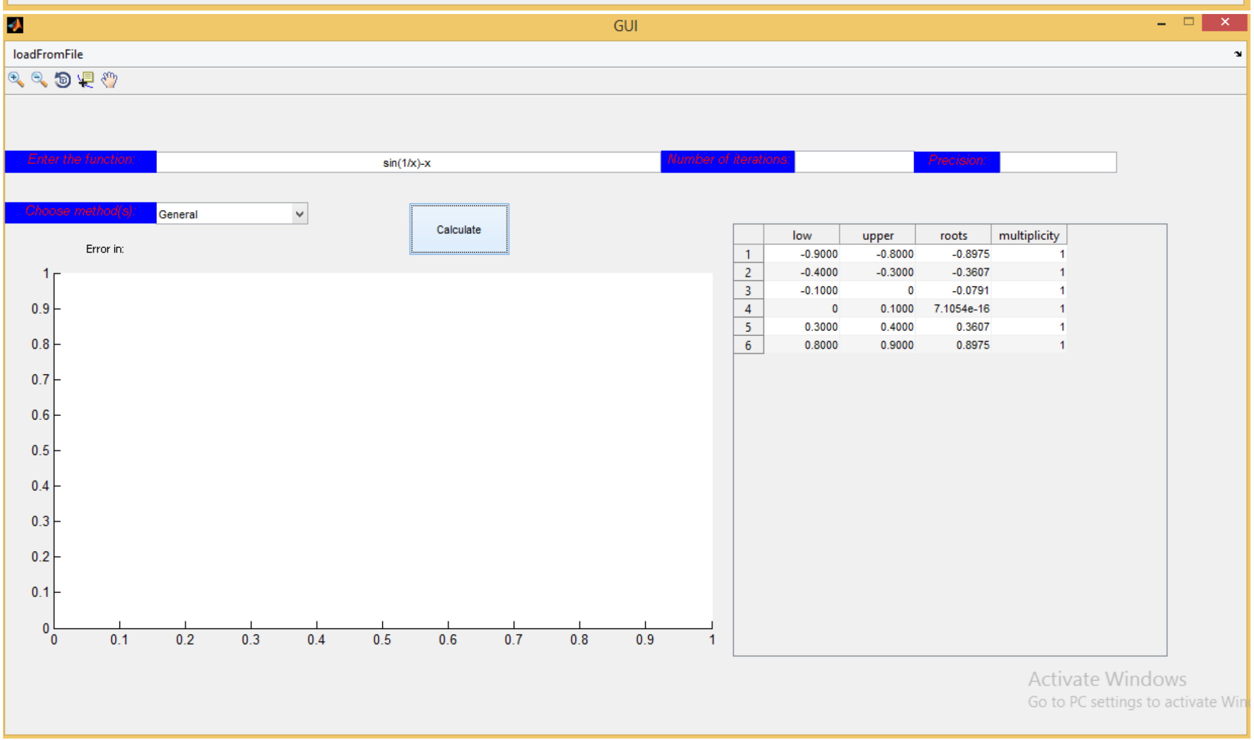
i	x1	x2	x3	error	time
1.000000	1.000000	1.100000	1.432900	0.332900	0.001744
2.000000	1.100000	1.300292	1.432900	0.132608	0.003752
3.000000	1.432900	1.322391	1.300292	0.022099	0.004830
4.000000	1.300292	1.324772	1.322391	0.002381	0.005903
5.000000	1.322391	1.324718	1.324772	0.000054	0.006963




GUI
loadFromFile
Enter the function:
sin(x)
Number of iterations:
Precision:
Choose method(s):
General
Calculate
Error in:
low upper roots multiplicity
1 -97.4000 -97.3000 -97.3894 1
2 -94.3000 -94.2000 -94.2478 1
3 -91.2000 -91.1000 -91.1062 1
4 -88 -87.9000 -87.9646 1
5 -84.9000 -84.8000 -84.8230 1
6 -81.7000 -81.6000 -81.6814 1
7 -78.6000 -78.5000 -78.5398 1
8 -75.4000 -75.3000 -75.3982 1
9 -72.3000 -72.2000 -72.2566 1
10 -69.2000 -69.1000 -69.1150 1
11 -66 -65.9000 -65.9734 1
12 -62.9000 -62.8000 -62.8319 1
13 -59.7000 -59.6000 -59.6903 1
14 -56.6000 -56.5000 -56.5487 1
15 -53.5000 -53.4000 -53.4071 1
16 -50.3000 -50.2000 -50.2655 1
17 -47.2000 -47.1000 -47.1239 1
18 -44 -43.9000 -43.9823 1
19 -40.9000 -40.8000 -40.8407 1
20 -37.7000 -37.6000 -37.6991 1
21 -34.6000 -34.5000 -34.5575 1
22 -31.5000 -31.4000 -31.4159 1
Activate Windows
Go to PC settings to activate Windows.


GUI
loadFromFile
Enter the function:
x^3-x-1
Number of iterations:
Precision:
Choose method(s):
General
Calculate
Error in:
low upper roots multiplicity
1 1.3000 1.4000 1.3247 1
Activate Windows
Go to PC settings to activate Windows.

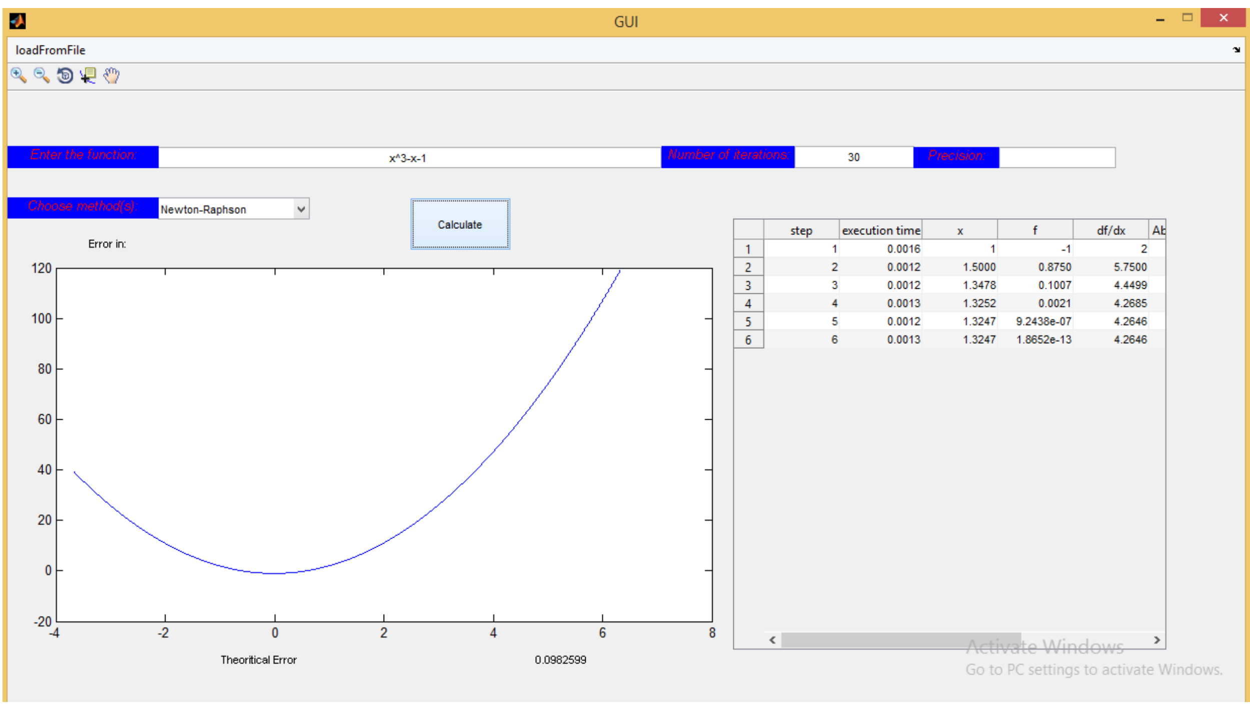

GUI
loadFromFile
Enter the function:
exp(-x)-x^2
Number of iterations:
Precision:
Choose method(s):
General
Calculate
Error in:
low upper roots multiplicity
1 0.7000 0.8000 0.7035 1
Activate Windows
Go to PC settings to activate Win

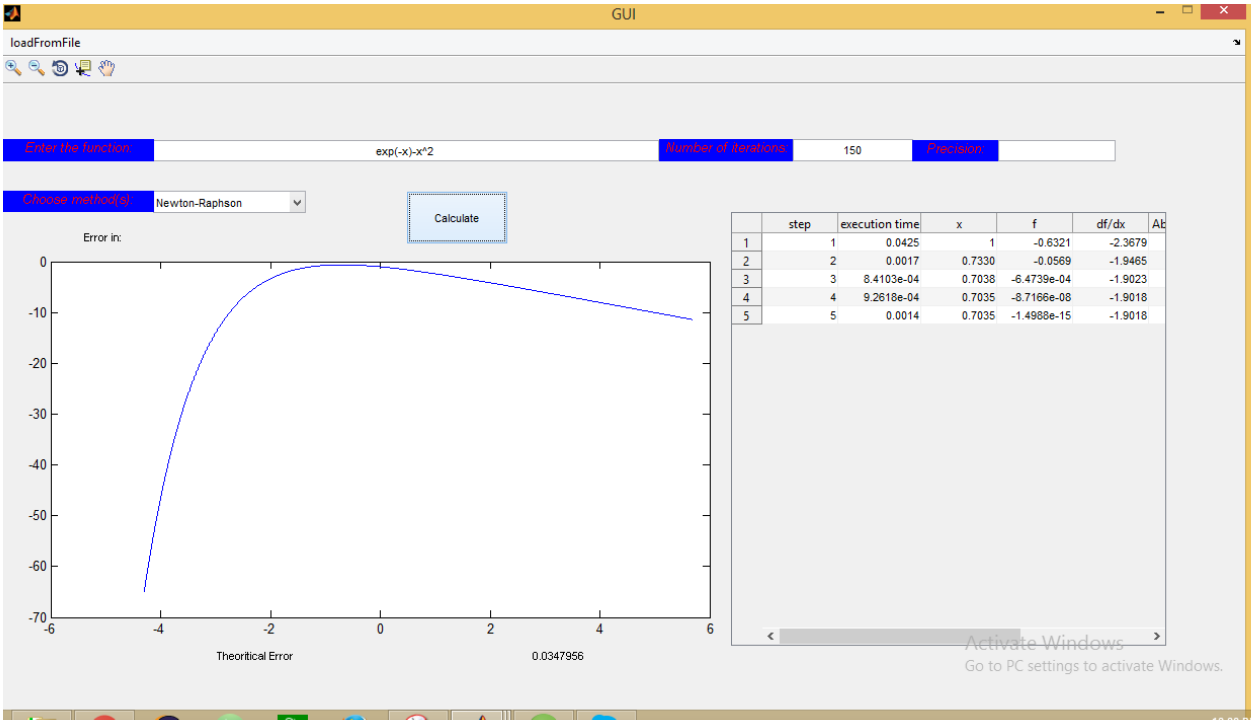

GUI
loadFromFile
Enter the function:
sin(1/x)-x
Number of iterations:
Precision:
Choose method(s):
General
Calculate
Error in:
low upper roots multiplicity
1 -0.9000 -0.8000 -0.8975 1
2 -0.4000 -0.3000 -0.3607 1
3 -0.1000 0 -0.0791 1
4 0 0.1000 7.1054e-16 1
5 0.3000 0.4000 0.3607 1
6 0.8000 0.9000 0.8975 1
Activate Windows
Go to PC settings to activate Win

left	right	root	multiplicity
1.300000	1.400000	1.324718	1.000000

left	right	root	multiplicity
-97.400000	-97.300000	-97.389372	1.000000
-94.300000	-94.200000	-94.247780	1.000000
-91.200000	-91.100000	-91.106187	1.000000
-88.000000	-87.900000	-87.964594	1.000000
-84.900000	-84.800000	-84.823002	1.000000
-81.700000	-81.600000	-81.681409	1.000000
-78.600000	-78.500000	-78.539816	1.000000
-75.400000	-75.300000	-75.398224	1.000000
-72.300000	-72.200000	-72.256631	1.000000
-69.200000	-69.100000	-69.115038	1.000000
-66.000000	-65.900000	-65.973446	1.000000
-62.900000	-62.800000	-62.831853	1.000000
-59.700000	-59.600000	-59.690260	1.000000
-56.600000	-56.500000	-56.548668	1.000000
-53.500000	-53.400000	-53.407075	1.000000

left	right	root	multiplicity
-0.900000	-0.800000	-0.897539	1.000000
-0.400000	-0.300000	-0.360672	1.000000
-0.100000	0.000000	-0.079079	1.000000
0.000000	0.100000	0.000000	1.000000
0.300000	0.400000	0.360672	1.000000
0.800000	0.900000	0.897539	1.000000





iterations	time	x	f	df/dx	Error
1.000000	0.000809	49.000000	117599.000000	7202.000000	0.000000
2.000000	0.000561	33.000000	35903.000000	3266.000000	16.000000
3.000000	0.000609	22.000000	10625.000000	1451.000000	11.000000
4.000000	0.000625	15.000000	3359.000000	674.000000	7.000000
5.000000	0.000554	10.000000	989.000000	299.000000	5.000000
6.000000	0.000611	7.000000	335.000000	146.000000	3.000000
7.000000	0.000551	5.000000	119.000000	74.000000	2.000000
8.000000	0.000607	3.000000	23.000000	26.000000	2.000000
9.000000	0.000673	2.000000	5.000000	11.000000	1.000000
10.000000	0.000609	2.000000	5.000000	11.000000	0.000000