2021

Root Finding Report

[AREEJ SALAHUDDIN 6389]
[SOHAILA HAZEM 6388]
[MANAR ABDELKADER 6485]

Contents

Overview

We want to find the roots or an approximation of them for a nonlinear equation by the use of iterative methods :

1) **Graphical method:**
   - By plotting the function and finding its intersections with the x-axis.
2) **Bracketing methods**
   - Bisection method.
   - False Position.
3) **Open methods**
   - Fixed Point.
   - Newton-Raphson.
   - Secant.

The aim of this project is to implement different numerical methods and to compare and analyze their behavior.

A Detailed Analysis of Each Method

1. Bisection Method:

- ***Behavior:***
  1. Choose two points(lower point Xl and upper point Xu) guesses for the root such that the function changes sign over the interval. This is checked by f(Xu)*f(Xl)<0
  2. An estimate of the root Xr is determined by Xr= (Xl+Xu)/2
  3. Looping over the maximum number of iterations or efficient error reached, if fn(Xl)*fn(Xr) < 0 the root lies in the lower subinterval so Xu=Xr but if if fn(Xl)*fn(Xr) > 0 the root lies in the upper subinterval so Xl=Xr.

- ***Pseudocode:***

```
function  root = bisection(xl, xu, es, imax);

if ((exp(-xl) - xl)*(exp(-xu) - xu))>0      % if guesses do not bracket, exit
   disp('no bracket')
   return
end

for i=1:1:imax

  xr=(xu+xl)/2;                % compute the midpoint   xr
  ea = abs((xu-xl)/xl);        % approx. relative error

  test= (exp(-xl) - xl) * (exp(-xr) - xr);   % compute    f(xl)*f(xr)

  if (test < 0)    xu=xr;
  else    xl=xr;
  end

  if (test == 0) ea=0;   end
  if (ea < es)  break;   end

end
```

- ***Data-Structure and Built in Functions:***
  1) Array to store the table values: (iteration, xl, xu, f(xl), f(xu), Xr, f(Xr) and ea)
  2) Built-in functions :
     - abs() -> To compute absolute of | Xr - Xr_old|.

- ***Conclusion:***

Bisection method is

-easy

-always finds a root,

-number of iterations required to attain an absolute error can be computed prior

But also

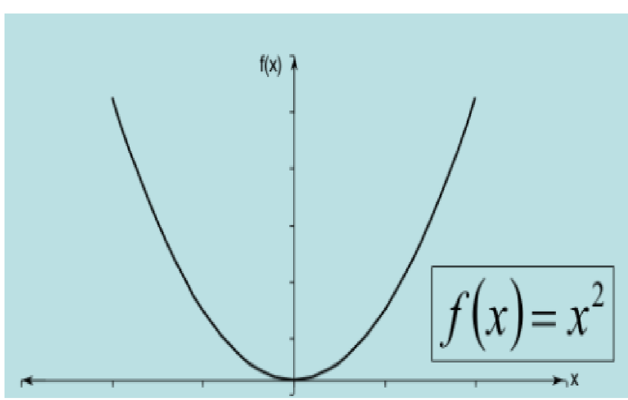
-Slow

-Need to find initial guesses for Xl and Xu

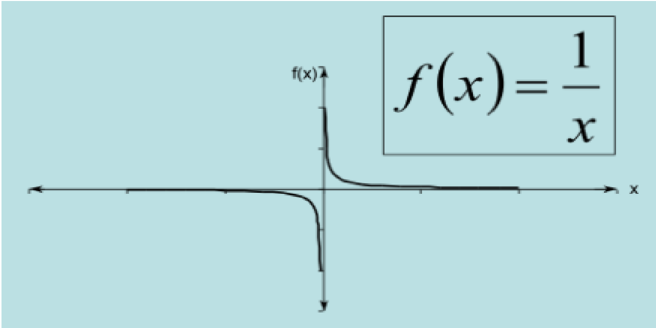
-No account is taken of the fact that if f(xl) is closer to zero, it is likely that root is closer to Xl(and similarly Xu.

- ***Drawbacks:***

1) If a function is such that it just touches the x-axis it will be unable to find the lower and upper guesses.



2) If a function changes sign but root does not exist.



2. False Position Method (Regula-Falsi):

- ***Behavior:***
  1. Choose two points(lower point Xl and upper point Xu) guesses for the root such that the function changes sign over the interval. This is checked by f(Xu)*f(Xl)<0
  2. An estimate of the root Xr is determined by

     Xr =(Xl*fn(Xu) - Xu*fn(Xl)) / (fn(Xu) - fn(Xl))
  3. Looping over the maximum number of iterations or efficient error reached, if fn(Xl)*fn(Xr) < 0 the root lies in the lower subinterval so Xu=Xr but if if fn(Xl)*fn(Xr) > 0 the root lies in the upper subinterval so Xl=Xr.

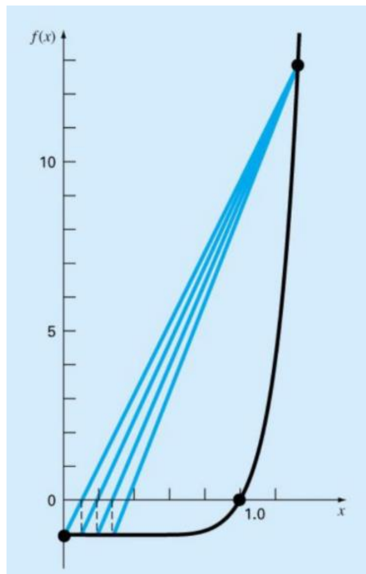
- ***Pseudocode:***

```
function [root] = regulaFalsi(f, l, u, eps, iter)
        if(f(l) * f(u) > 0)
                //No solution
        //end if
        else
                root = (l * f(u) - u * f(l)) / (f(u) - f(l))
                if(f(u) * f(root) < 0)
                        l = root
                        //end if
                else
                        u = root
                i = 0;
                while(relative error > eps and i < iter)
                        i++
                        calculate the next root
                //end while
//end function
```

- ***Data-Structure and Built in Functions:***
    1) Array to store the table values: (iteration, xl, xu, f(xl), f(xu), Xr, f(Xr) and ea)
    2) Built-in functions :
        - abs() -> To compute absolute of | Xr - Xr_old|.

- ***Conclusion:***
    1) Always converges to the root.
    2) Faster than Bisection Method

- ***Drawbacks:***
    1) Has a very slow convergence rate when a part is parallel to the x-axis.



3. Fixed Point Method:

- ***Behavior:***
    1) Starting with one initial value (X_old)
    2) If |g'(X_old)| > 1 then it won't converge.
    3) Looping over the maximum number of iterations or efficient error reached, Xr=X_old.

- ***Pseudocode:***

```
double FixedPt(double x0, double es, int iter_max) {
  double xr = x0;   // Estimated root
  double xr_old;    // Keep xr from previous iteration
  int iter = 0;     // Keep track of # of iterations

  do {
    xr_old = xr;
    xr = g(xr_old);  // g(x) has to be supplied
    if (xr != 0)
      ea = fabs((xr - xr_old) / xr) * 100;

    iter++;
  } while (ea > es && iter < iter_max);

  return xr;
}
```

- ***Data-Structure and Built in Functions:***

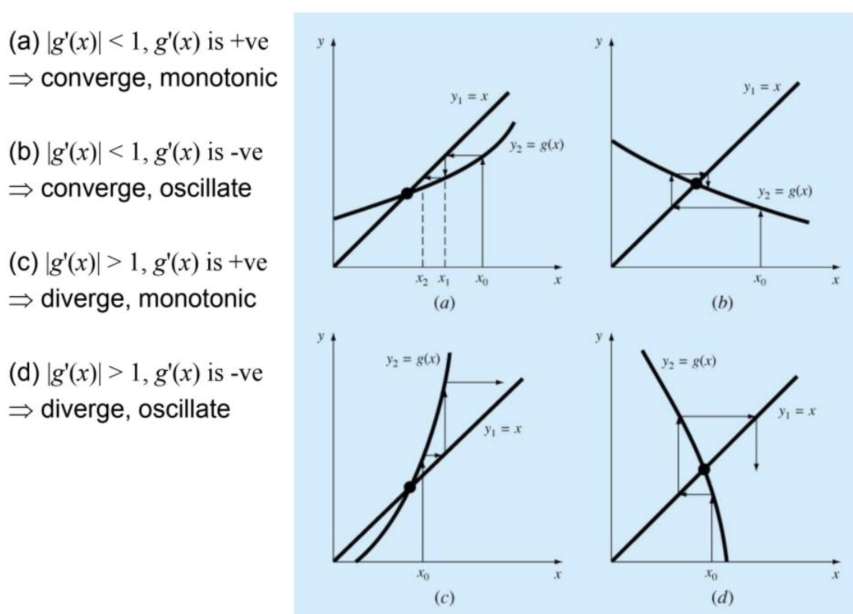
    1) Array to store the table values: (iteration, xr_old, Xr, g(Xr_old), g(Xr) and ea)
    2) Built-in functions :
        - Syms -> to make 'x' a symbolic variable to get the derivative
        - abs() -> To compute absolute of | Xr - Xr_old|.

- ***Conclusion:***

  To find the root for a function f(x) we construct a formula Xi+1=g (Xi) to predict the root iteratively until X converges to a root.

- ***Drawbacks:***

    1) Point X might diverge away from the root due to bad construction of formula g(x).
    2) Even if function passes the convergence condition it may not converge.
    3) In some functions it takes a lot of iterations to converge.



- ***Suggestions:***

  Change the function of g

4. Newton-Raphson Method:

- ***Behavior:***
  1) Use the slope of the function to predict the location of root
  2) Get the derivative of the function
  3) Use initial guess of the root to estimate new value of the root.

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

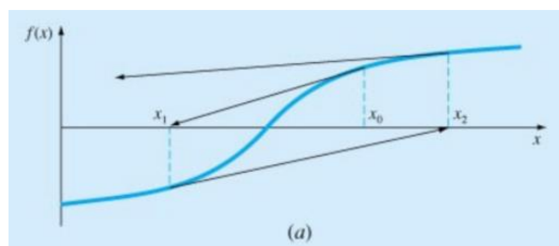
  4) Iterate until the absolute relative error is acceptable or the number of iterations has exceeded the maximum number of iterations allowed.

- ***Pseudocode:***

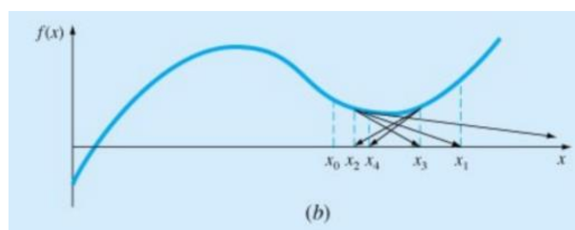
```
function [root] = newtonRaphson(f, x0, eps, iter)
        //calculate the derivative
        derivative = differentiate(f)
        if(derivative(x0) == 0)
                return initial point as the root
        //end if
        else
                root = x0 - f(x0) / derivative(x0)
                i = 0;
                while(derivative(root) != 0 and i < iter and relative error > eps)
                        i++
                        calculate the next root
                //end while
//end function
```

- ***Data-Structure and Built in Functions:***
  1) Array to store the table values: (iteration, xr_old, Xr, f(Xr_old), f(Xr) and ea)
  2) Built-in functions :
     - Syms -> to make ‘x’ a symbolic variable to get the derivative
     - matlabFunction -> to convert the symbolic expression or function f to a MATLAB function
     - abs() -> To compute absolute of | Xr - Xr_old|.
     - diff() -> To differentiate f(x).

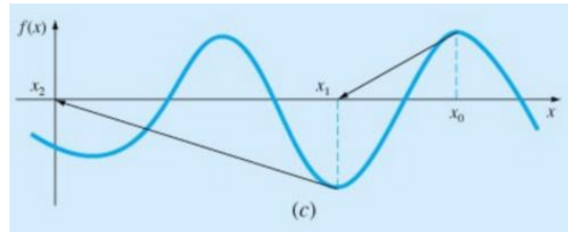
- ***Conclusion:***
  1) Newton-Raphson method converges quadratically when it converges except when the root has a multiplicity
  2) Usually Converges when the initial point is near the root
- ***Drawbacks:***
  1) It diverges if there's an inflection point at the vicinity of the root.



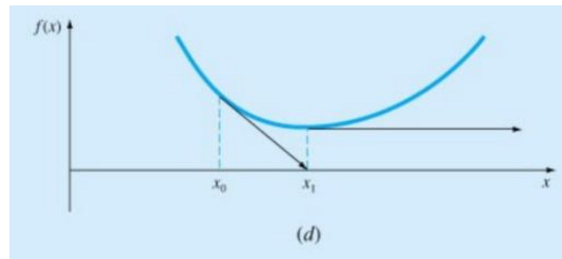
  2) A local maximum or minimum causes oscillations.



3) It may jump from one location close to one root to a location that's several roots away.



4) A Zero slope causes division by Zero



- ***Suggestions:***
  1) Although there's no general convergence criteria for that method, but it can deal well according to good initial points, good knowledge of functions and graphical analysis, and good software that detects slow convergence or divergence.
  2) Use Modified Newton-Raphson Method that uses multiplicity of Roots

5. Secant Method:

- ***Behavior:***

  This method approximates the derivatives by finite divided difference.

  1) Starting with two initial values x_i-1 and xi we find values of f(x_i-1) and f(xi)
  2) Use initial guess of the root to estimate new value of the root.

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

  3) Iterate until the absolute relative error is acceptable or the number of iterations has exceeded the maximum number of iterations allowed.

- ***Pseudocode:***

```
function root = secant(f, x0, x1, eps, maxit)
        for i = 1 to maxit
                root = x1 - f(x1) * (x0 - x1) / (f(x0) - f(x1))
                if(abs((root - x1) / x1) < eps)
                        break;
                //end if
                x0 = x1
                x1 = root
        //end for
//end function
```

- ***Data-Structure and Built in Functions:***
  1) Array to store the table values: (x0, x1, x, f(x1), f(x0), f(x) and ea)
  2) Built-in functions :
     - abs() -> To compute absolute of | Xr - Xr_old|

- ***Conclusion:***
  1) Special case of False position method. Both Methods use the same expression
  2) Difference is where the initial values are replaces by the new estimate
  3) Converges faster than a linear rate

$$\text{Secant:} \qquad x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

$$\text{False position:} \quad x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

- ***Drawbacks:***

  May not converge
- ***Suggestions:***

  Modify method that the function will need only one initial; δ is a float number, that is a crucial factor in the performance of that method.

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

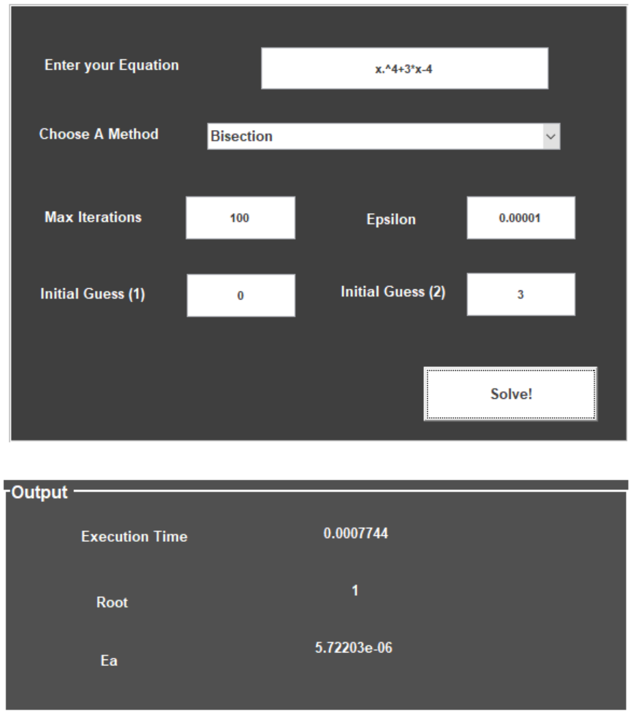
  1) If δ is too small, a subtractive cancellation can happen in the denominator

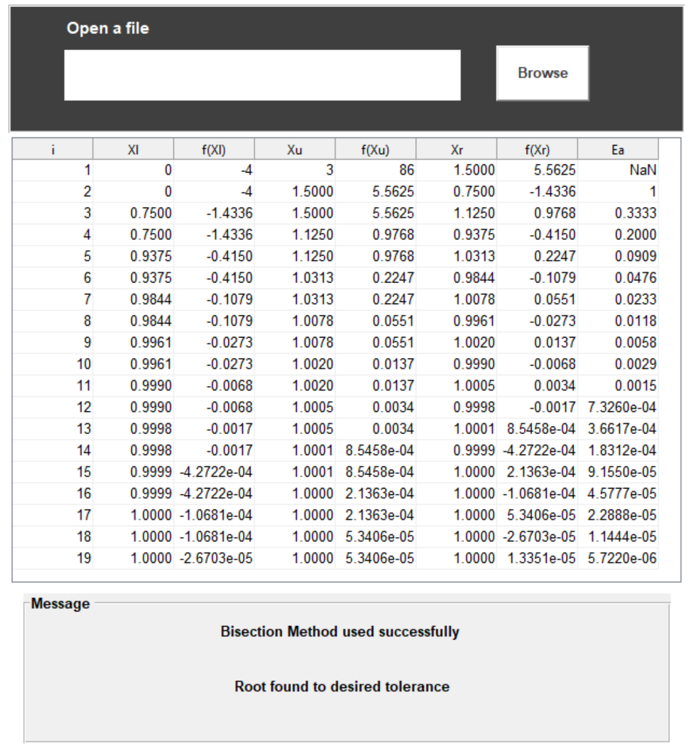
2) If δ is too big, that method often becomes divergent
3) If δ is selected properly, this method provides a good alternative for cases when developing two initial guess is inconvenient.

Sample Runs

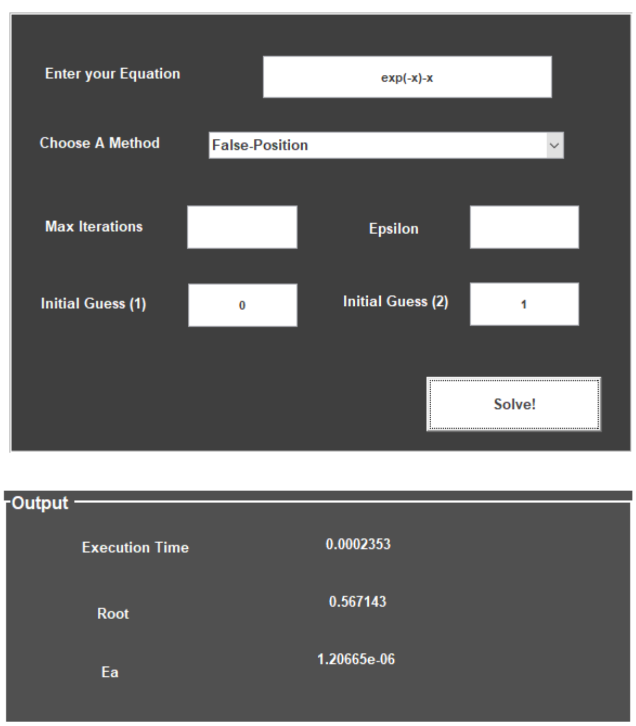
Default Max Iterations = 50

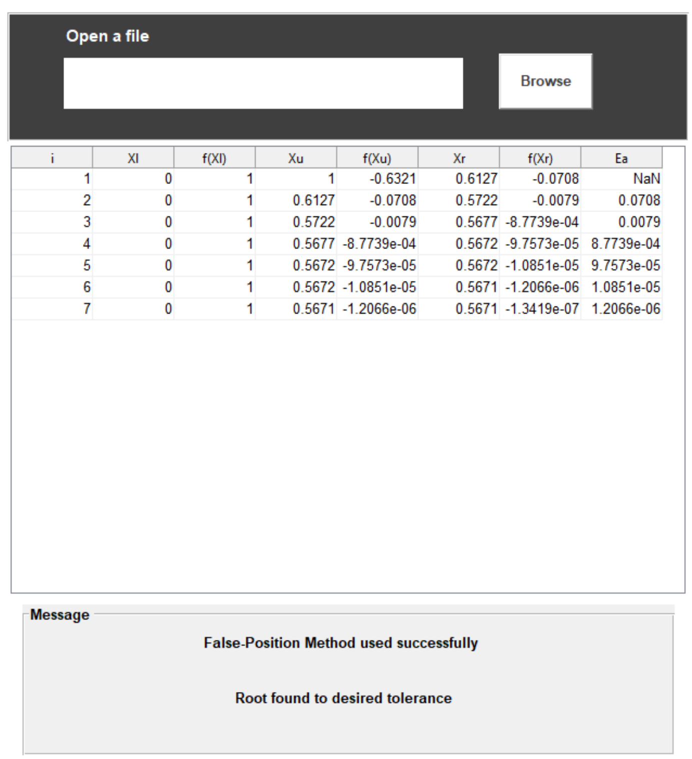
Default Epsilon = 0.00001





i	Xl	f(Xl)	Xu	f(Xu)	Xr	f(Xr)	Ea
1	0	-4	3	86	1.5000	5.5625	NaN
2	0	-4	1.5000	5.5625	0.7500	-1.4336	1
3	0.7500	-1.4336	1.5000	5.5625	1.1250	0.9768	0.3333
4	0.7500	-1.4336	1.1250	0.9768	0.9375	-0.4150	0.2000
5	0.9375	-0.4150	1.1250	0.9768	1.0313	0.2247	0.0909
6	0.9375	-0.4150	1.0313	0.2247	0.9844	-0.1079	0.0476
7	0.9844	-0.1079	1.0313	0.2247	1.0078	0.0551	0.0233
8	0.9844	-0.1079	1.0078	0.0551	0.9961	-0.0273	0.0118
9	0.9961	-0.0273	1.0078	0.0551	1.0020	0.0137	0.0058
10	0.9961	-0.0273	1.0020	0.0137	0.9990	-0.0068	0.0029
11	0.9990	-0.0068	1.0020	0.0137	1.0005	0.0034	0.0015
12	0.9990	-0.0068	1.0005	0.0034	0.9998	-0.0017	7.3260e-04
13	0.9998	-0.0017	1.0005	0.0034	1.0001	8.5458e-04	3.6617e-04
14	0.9998	-0.0017	1.0001	8.5458e-04	0.9999	-4.2722e-04	1.8312e-04
15	0.9999	-4.2722e-04	1.0001	8.5458e-04	1.0000	2.1363e-04	9.1550e-05
16	0.9999	-4.2722e-04	1.0000	2.1363e-04	1.0000	-1.0681e-04	4.5777e-05
17	1.0000	-1.0681e-04	1.0000	2.1363e-04	1.0000	5.3406e-05	2.2888e-05
18	1.0000	-1.0681e-04	1.0000	5.3406e-05	1.0000	-2.6703e-05	1.1444e-05
19	1.0000	-2.6703e-05	1.0000	5.3406e-05	1.0000	1.3351e-05	5.7220e-06





i	Xl	f(Xl)	Xu	f(Xu)	Xr	f(Xr)	Ea
1	0	1	1	-0.6321	0.6127	-0.0708	NaN
2	0	1	0.6127	-0.0708	0.5722	-0.0079	0.0708
3	0	1	0.5722	-0.0079	0.5677	-8.7739e-04	0.0079
4	0	1	0.5677	-8.7739e-04	0.5672	-9.7573e-05	8.7739e-04
5	0	1	0.5672	-9.7573e-05	0.5672	-1.0851e-05	9.7573e-05
6	0	1	0.5672	-1.0851e-05	0.5671	-1.2066e-06	1.0851e-05
7	0	1	0.5671	-1.2066e-06	0.5671	-1.3419e-07	1.2066e-06

f(x) = exp(-x) - x, g(x) = f(x) + x = exp(-x)

Enter your Equation: exp(-x)

Choose A Method: Fixed Point

Max Iterations: 20

Epsilon: 0.01

Initial Guess (1): 0

Solve!

Output

Execution Time: 0.0014517

Root: 0.568429

Ea: 0.00624419

Open a file

Browse

i	Xr_old	Xr	g(Xr_old)	g(Xr)	Ea
1	0	1	1	0.3679	NaN
2	1	0.3679	0.3679	0.6922	1.7183
3	0.3679	0.6922	0.6922	0.5005	0.4685
4	0.6922	0.5005	0.5005	0.6062	0.3831
5	0.5005	0.6062	0.6062	0.5454	0.1745
6	0.6062	0.5454	0.5454	0.5796	0.1116
7	0.5454	0.5796	0.5796	0.5601	0.0590
8	0.5796	0.5601	0.5601	0.5711	0.0348
9	0.5601	0.5711	0.5711	0.5649	0.0193
10	0.5711	0.5649	0.5649	0.5684	0.0111
11	0.5649	0.5684	0.5684	0.5664	0.0062

Message

Fixed-Point Method used successfully

Root found to desired tolerance

Enter your Equation: exp(-x)-x

Choose A Method: Newton-Raphson

Max Iterations:

Epsilon:

Initial Guess (1): 0

Solve!

Output

Execution Time: 0.0579412

Root: 0.567143

Ea: 2.21064e-07

Open a file

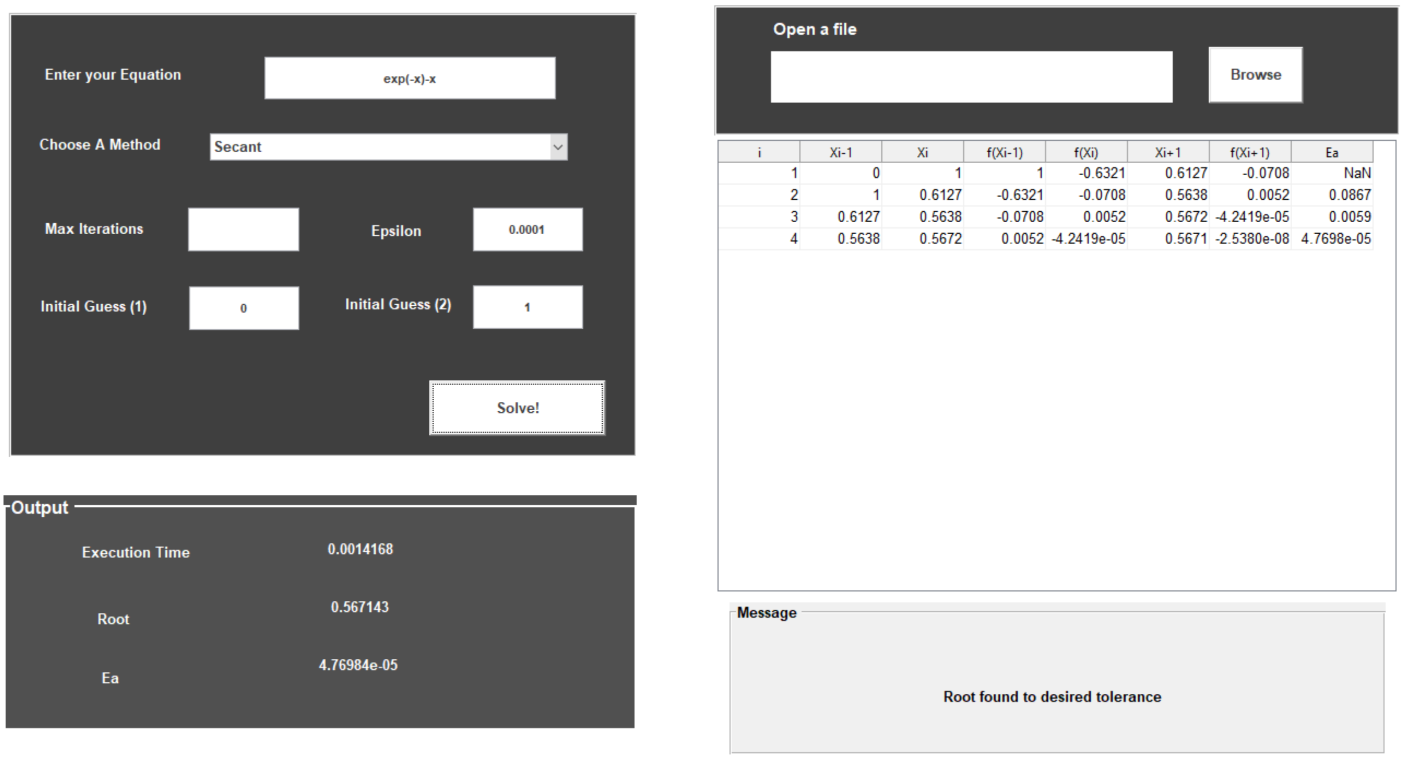
Browse

i	Xr_old	Xr	f(Xr_old)	f(Xr)	Ea
1	0	0.5000	1	0.1065	NaN
2	0.5000	0.5663	0.1065	0.0013	0.1171
3	0.5663	0.5671	0.0013	1.9648e-07	0.0015
4	0.5671	0.5671	1.9648e-07	4.5519e-15	2.2106e-07

Message

Newton-Raphson Method used successfully

Root found to desired tolerance



i	Xi-1	Xi	f(Xi-1)	f(Xi)	Xi+1	f(Xi+1)	Ea
1	0	1	1	-0.6321	0.6127	-0.0708	NaN
2	1	0.6127	-0.6321	-0.0708	0.5638	0.0052	0.0867
3	0.6127	0.5638	-0.0708	0.0052	0.5672	-4.2419e-05	0.0059
4	0.5638	0.5672	0.0052	-4.2419e-05	0.5671	-2.5380e-08	4.7698e-05

Problematic Functions

f(x) = x^2 - 4

with initial point = 0

using Newton-Raphson is a problem as it will cause division by zero.