Root Finder Program

Name 1: Abdelrahman Salem Mohamed

ID: 6309

Name 2: Reem Abdelhalim

ID: 6114

Course: Numerical Analysis

Instructor: Zeinab

Contents

Objective

The aim of this assignment is to compare and analyze the behavior of numerical methods studied in class {Bisection, False-position, Fixed point, Newton Raphson, Secant}.

Description

It is required to implement a root finder program which takes as an input the equation expression and choose which methods to apply to find the root and its required parameters.

Program specs

Code written in python and GUI implemented using Tkinter, simple user interface just to see the results in proper format and readable way .

Procedures

The program start with window and user should enter **valid** mathematical expression function as shown in figure 1.

Figure 1

And if the user enter invalid expression format Error window is show as presented in figure 2.

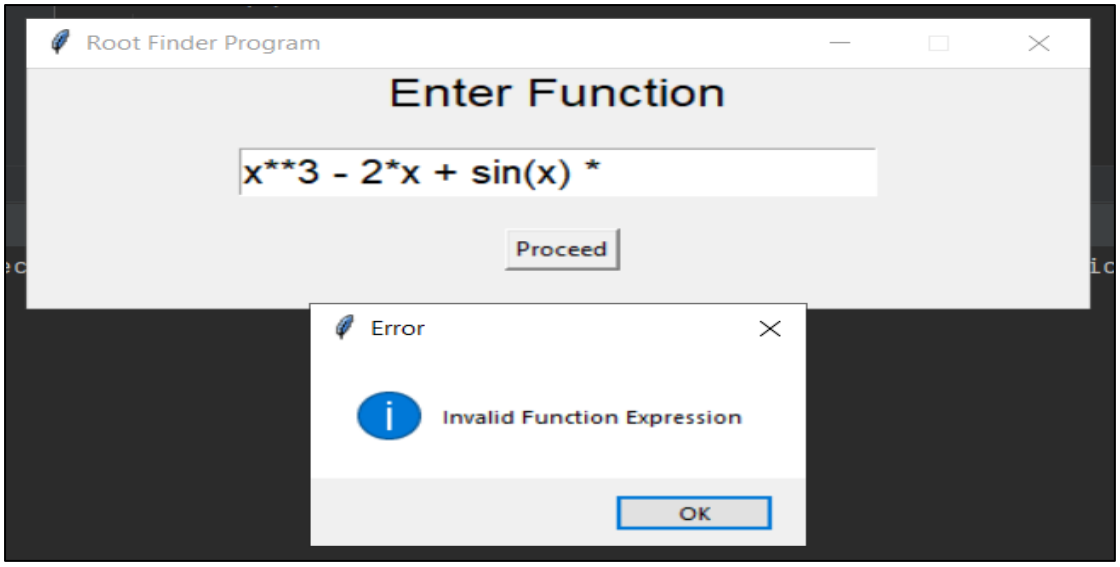


Figure 2

Let's assume the user enter the following valid expression and click proceed as shown in figure 3.

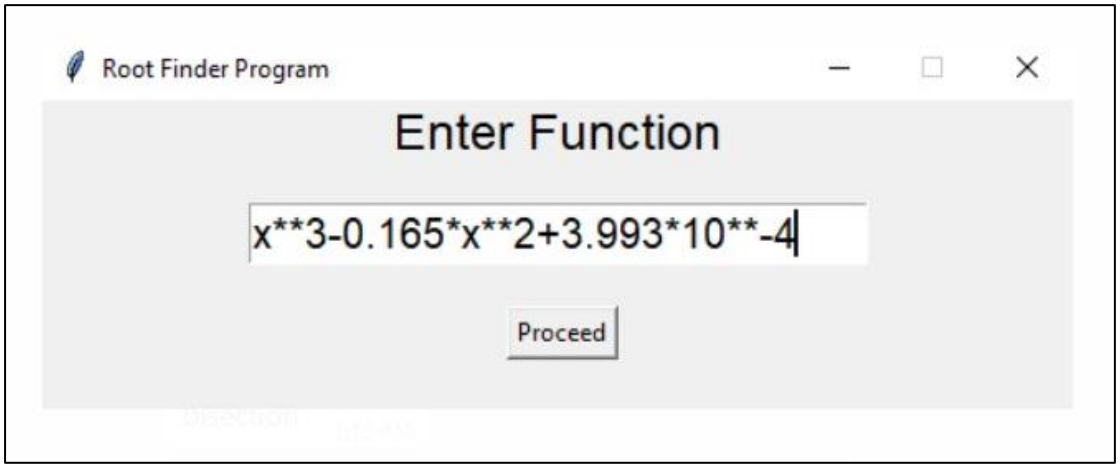


Figure 3

The user will be directed to the next window to select all available root find methods {Bisection, False position, Fixed Point, Newton & Secant} as shown in figure 4.

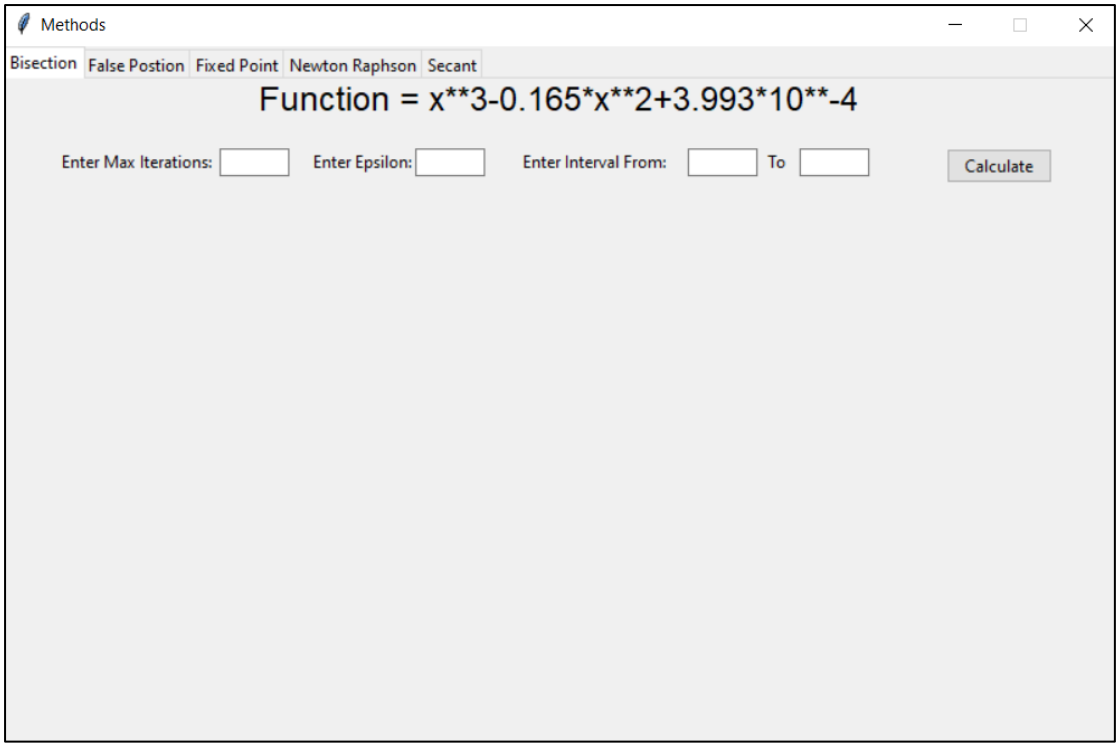
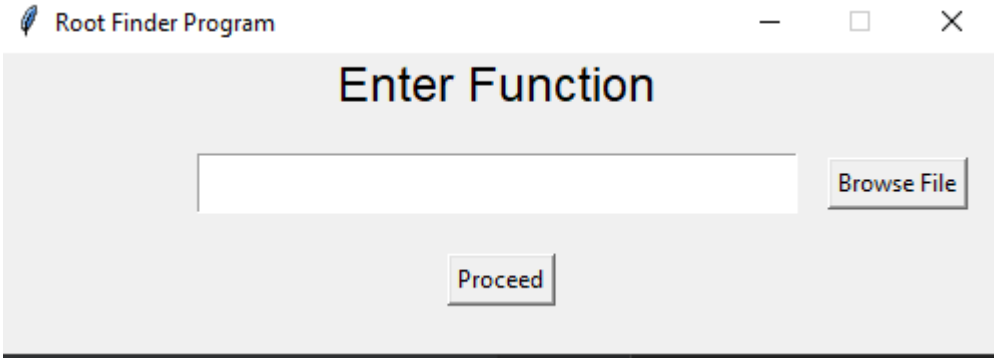


Figure 4

For every method there are couple of parameters the user can enter to apply on the algorithm and from all methods Max iterations and epsilon (approximate absolute relative error) are common and if the user ignores any of them the default pre-defined value will be used, after enter the algorithm parameter then the user can get the results of each iteration and display each calculated epsilon.

Read from file

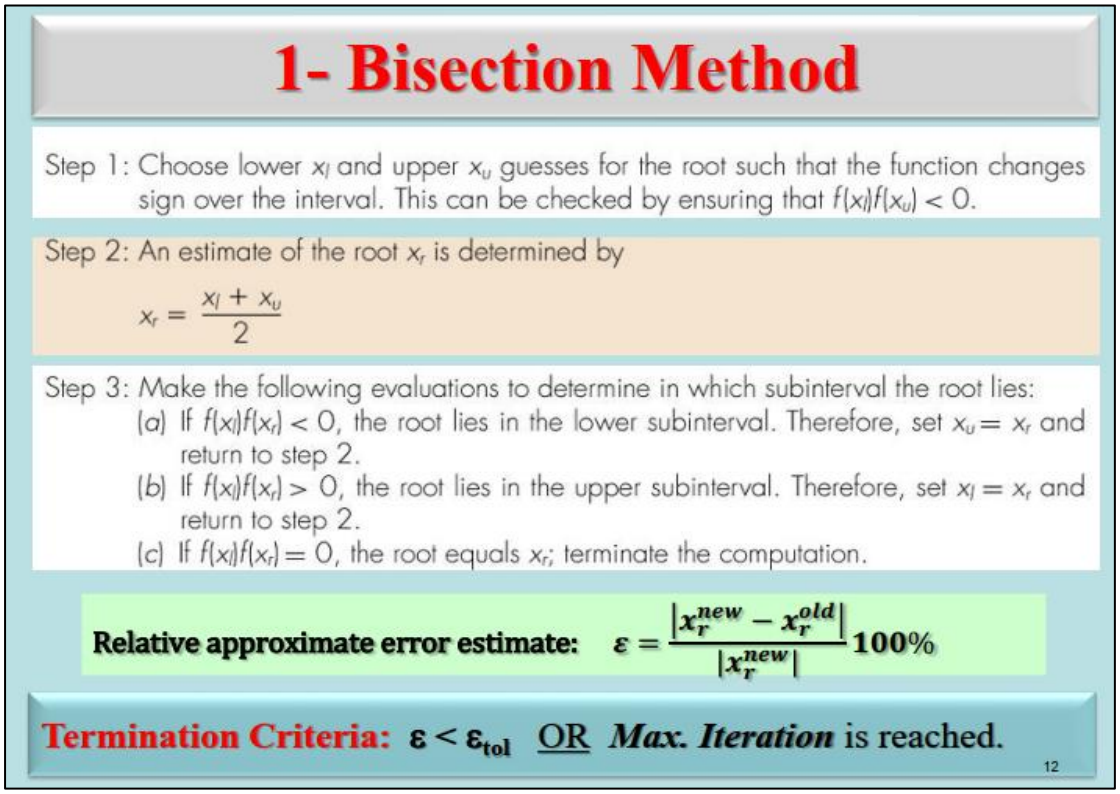


Click on browse file button to choose text file to read equation from.

Methods Algorithm & Pseudocode

Bisection Method

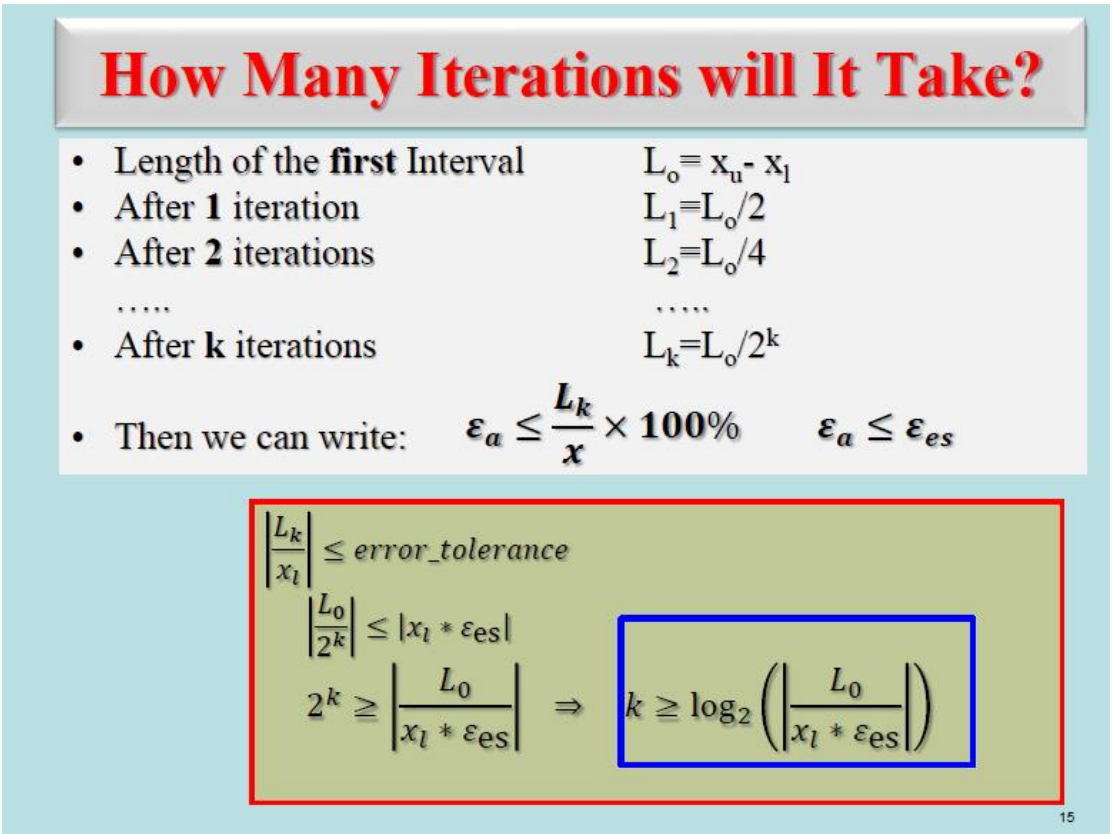
Algorithm



Pseudocode

```
% Bisection Method - simple
% function f(x) = exp(-x) - x = 0   sample call:  bisection(-2, 4, 0.001,500)

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);         % approxate 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

s=sprintf('\n  Root= %f    #Iterations = %d \n', xr, i); disp(s);
```

How many Iteration needed by bisection method:



We used this equation to calculate k, which is maximum number of iterations needed to calculate the root

False Position Method

Algorithm

1. Find a **pair** of values of x, (x_l and x_u), such that:

   $f_l = f(x_l) < 0$ and $f_u = f(x_u) > 0$.

2. Estimate the value of the **root** from the following formula: $x_r = \frac{x_l f_u - x_u f_l}{f_u - f_l}$

   and evaluate $f(x_r)$.

3. Use the **new point**, x_r, to replace one of the **original** points, keeping the **two points** on **opposite sides** of the x-axis.

   If $f(x_r)<0$ then $x_l=x_r$ ==> $f_l=f(x_r)$

   If $f(x_r)>0$ then $x_u=x_r$ ==> $f_u=f(x_r)$

   If $f(x_r)=0$, then you have found the **root** and need go no further!

4. See if the new x_l and x_u are **close** enough for **convergence** to be declared. If they are not, go back to **step 2**.

Pseudocode

```
function [x,y] = false_position(func)

% Find root near x1 using the false position method.
% Input:   func       string containing name of function
%          xl,xu      initial guesses
%          es         allowable tolerance in computed root
%          maxit      maximum number of iterations
% Output:  x          row vector of approximations to root

xl = input('enter lower bound xl = ');
xu = input('enter upper bound xu = ');
es = input('allowable tolerance es = ');
maxit = input('maximum number of iterations maxit = ');

a(1) = xl; b(1) = xu;
ya(1) = feval(func, a(1)); yb(1)=feval(func, b(1));
if ya(1) * yb(1) > 0.0
     error('Function has same sign at end points')
end
for i = 1:maxit
     x(i) = b(i) - yb(i)*(b(i)-a(i))/(yb(i)-ya(i));
     y(i) = feval(func, x(i));
     if y(i) == 0.0
        disp('exact zero found'); break;
     elseif y(i) * ya(i) < 0
        a(i+1) = a(i); ya(i+1) = ya(i);
        b(i+1) = x(i); yb(i+1) = y(i);
     else
        a(i+1) = x(i); ya(i+1) = y(i);
        b(i+1) = b(i); yb(i+1) = yb(i);
     end;
      if((i > 1) & (abs(x(i) - x(i-1)) < es))
          disp('False position method has converged'); break
      end
     iter = i;
end
if(iter >= maxit)
        disp('zero not found to desired tolerance');
end
n=length(x); k=1:n; out = [k'  a(1:n)'  b(1:n)'  x'  y'];
disp('      step      xl          xu        xr        f(xr)')
disp(out)
```

Linear interpolation

45

Fixed Point Method

Algorithm

- Also known as **One-Point Iteration** or **Successive Substitution (Approximation).**
- To find the **root** for $f(x) = 0$, we **reformulate** $f(x) = 0$ so that there is an x on one side of the equation.

$$f(x) = 0 \quad \Leftrightarrow \quad g(x) = x$$

- If we can solve $g(x) = x$, we solve $f(x) = 0$.
  - x is known as the **fixed point** of $g(x)$.
- We solve $g(x) = x$ by computing:

$$x_{i+1} = g(x_i) \quad \text{with } x_0 \text{ given}$$

until x_{i+1} **converges** to x.

12

Pseudocode

```
// x0: Initial guess of the root
// es: Acceptable relative percentage error
// iter_max: Maximum number of iterations allowed
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) must be supplied
      if (xr != 0)
        ea = fabs((xr - xr_old) / xr) * 100;

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

  return xr;
}
```

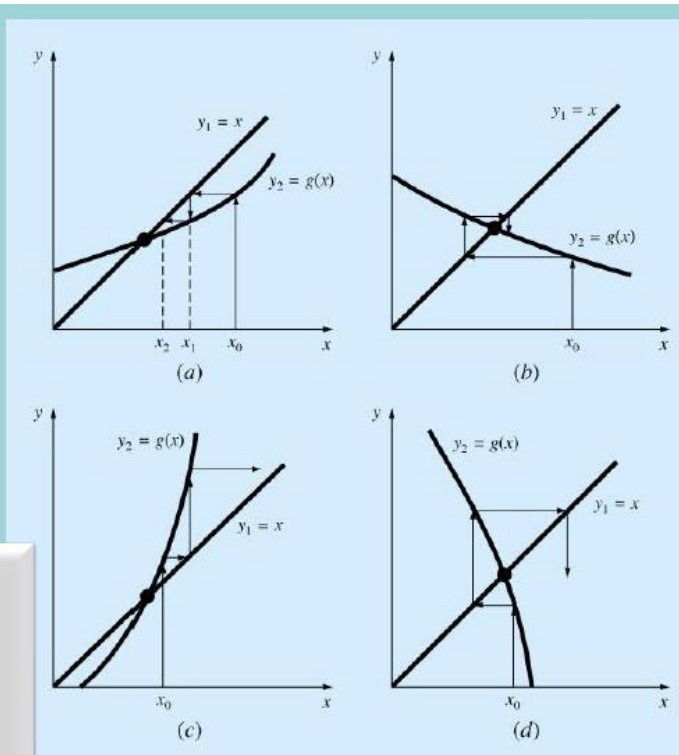
Check convergence of equation:

(a) $|g'(x)| < 1$, $g'(x)$ is +ve
⇒ converge, **monotonic**

(b) $|g'(x)| < 1$, $g'(x)$ is -ve
⇒ converge, **oscillate**

(c) $|g'(x)| > 1$, $g'(x)$ is +ve
⇒ diverge, **monotonic**

(d) $|g'(x)| > 1$, $g'(x)$ is -ve
⇒ diverge, **oscillate**



The **convergence rate** of **fixed-point iteration** is usually **linear** with constant $C = |g'(x)|$,

Newton – Raphson's Method

1- Evaluate $f'(x)$ symbolically.

2- Use an **initial guess** of the **root**, x_i , to **estimate** the **new value** of the **root**, x_{i+1} , as:

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

3- Find the **absolute relative approximate error** $|\epsilon_a|$ as:

$$|\epsilon_a| = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100$$

Pseudocode

```
function [x, f] = multiple1(func, dfunc)

% Find multiple root near xguess using the modified Newton's method
% Multiplicity m of the root is given -- m = 1: single root
% m = 2: double root; m = 3: triple root, etc.
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

m = input('enter multiplicity of the root = ');
xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1) - m * f(i-1) / dfdx(i-1);
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     if abs(x(i) - x(i-1)) < es
        disp('Newton method has converged'); break;
     end
     iter = i;
end
if (iter >= maxit)
        disp('zero not found to desired tolerance');
end
n = length(x);   k = 1:n;
disp('   step              x                   f                  df/dx')
out = [k; x; f; dfdx];
fprintf('%5.0f  %20.14f  %21.15f  %21.15f\n',out)
```

Secant

Algorithm

It's a slight variation of **Newton-Raphson's** method for functions whose **derivatives** are **difficult** to evaluate.

For these cases, the **derivative** can be **approximated** by a **backward finite divided difference**.

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

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \qquad i = 1,2,3,\ldots$$

Pseudocode

```
function [x, f] = multiple2(func, dfunc, ddfunc)

% Find multiple root near xguess using the modified Newton's method.
% Multiplicity of the root is not known a priori
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       ddfunc    name of second derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
d2fdx2(1) = feval(ddfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1)-f(i-1)*dfdx(i-1)/(dfdx(i-1)^2-f(i-1)*d2fdx2(i-1));
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     d2fdx2(i) = feval(ddfunc, x(i));
     if abs(x(i)-x(i-1)) < es
        disp('Newton method has converged'); break;
     end
     iter = i;
end
if (iter >= maxit)
        disp('zero not found to desired tolerance');
end
n = length(x);   k = 1 : n;
disp('   step        x             f          df/dx          d2f/dx2')
out=[k; x; f; dfdx; d2fdx2];
fprintf('%5.0f  %17.14f  %20.15f %20.15f %20.15f\n',out)
```

Code Result Analysis

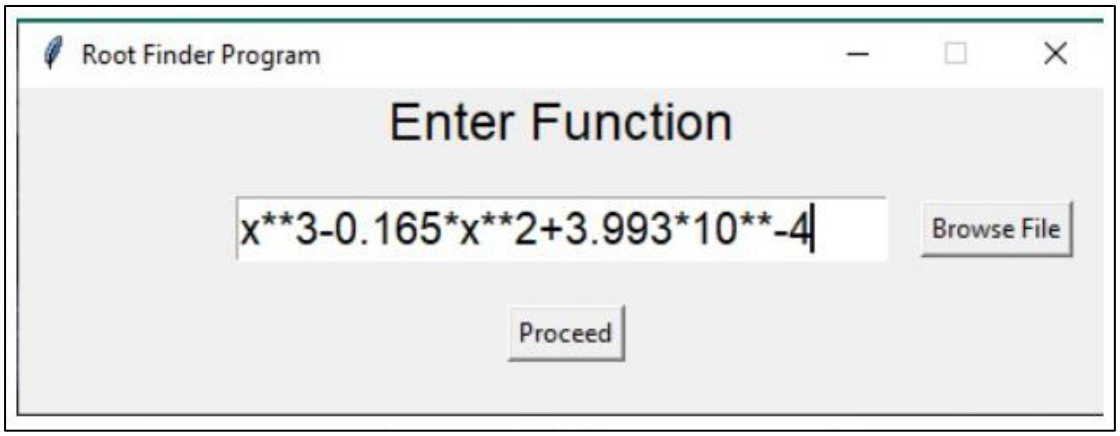
We have picked three function expressions to test on the code from the class lecture based on basic mathematical functions {polynomial, sin, cosine, exponential}.

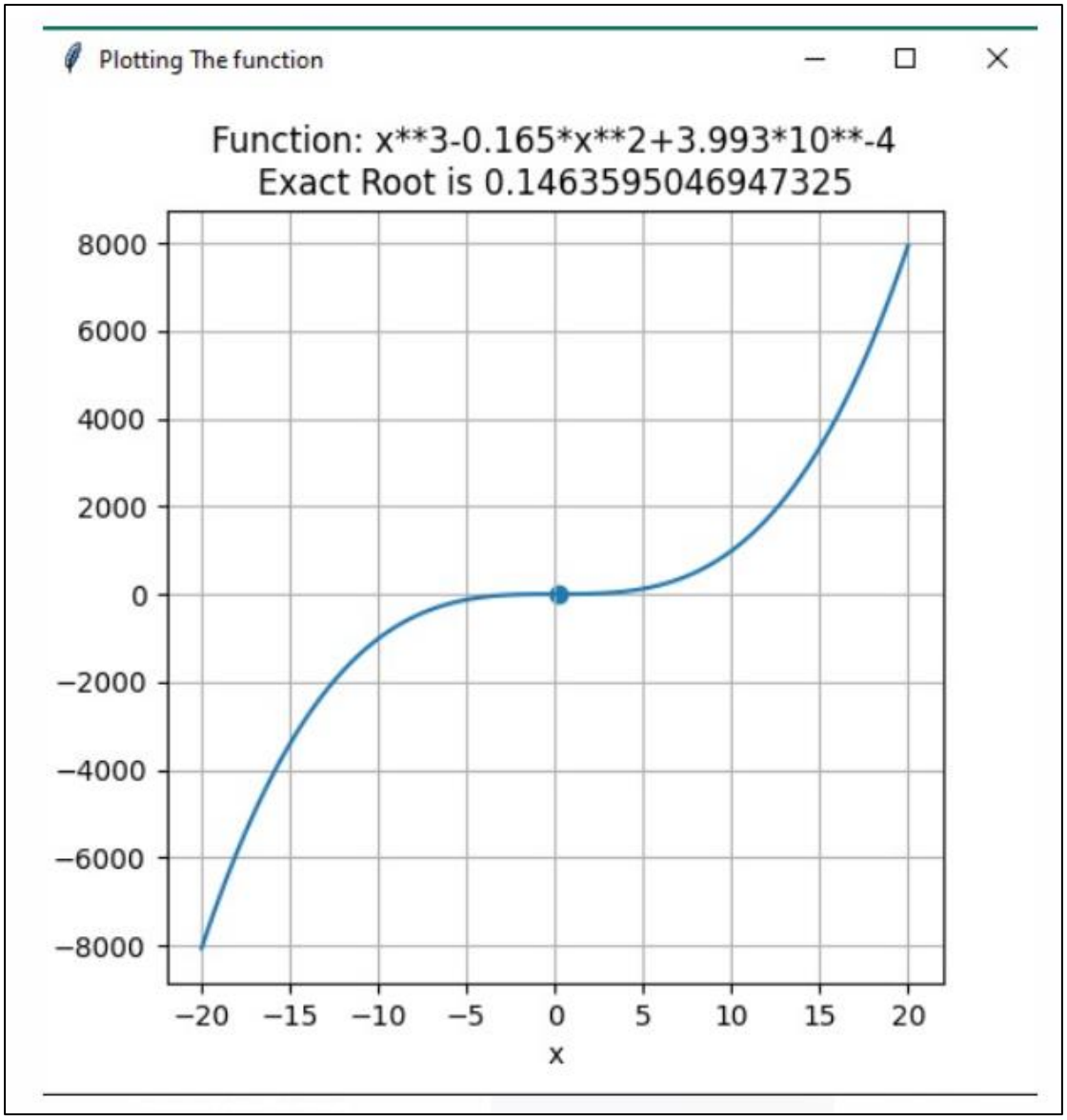
- $x^3 - 0.165x^2 + 3.993 \times 10^{-4}$
- $e^{-x} - x$
- $x + \cos(x)$

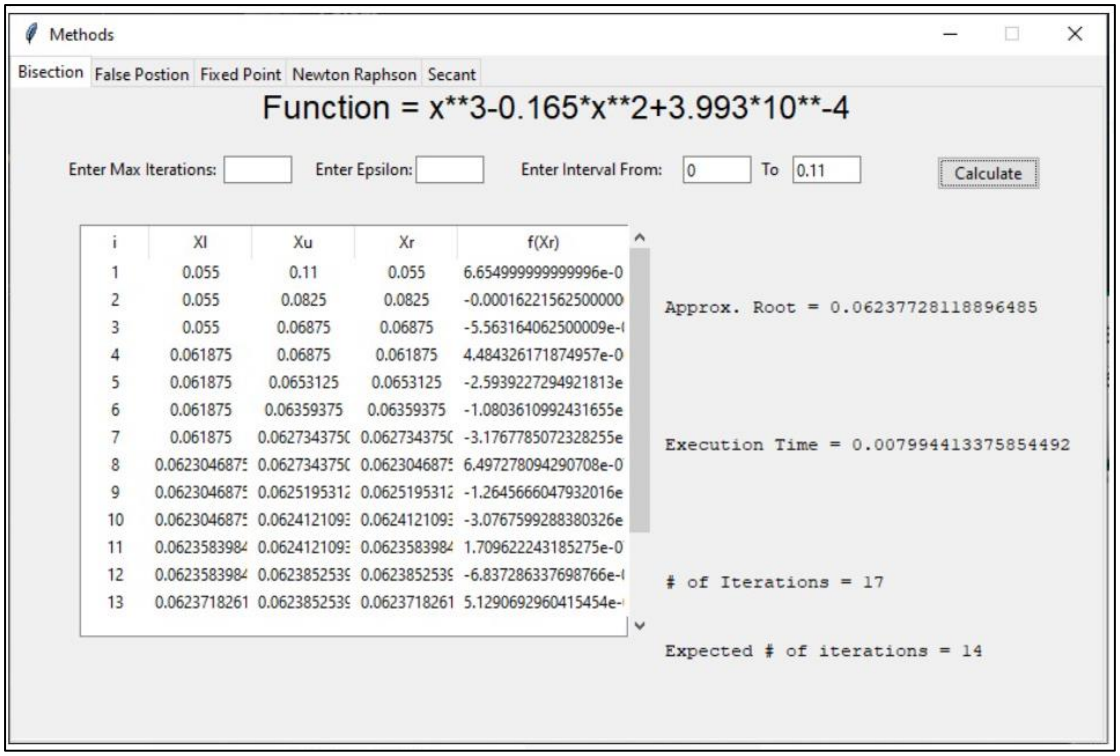
Expression one

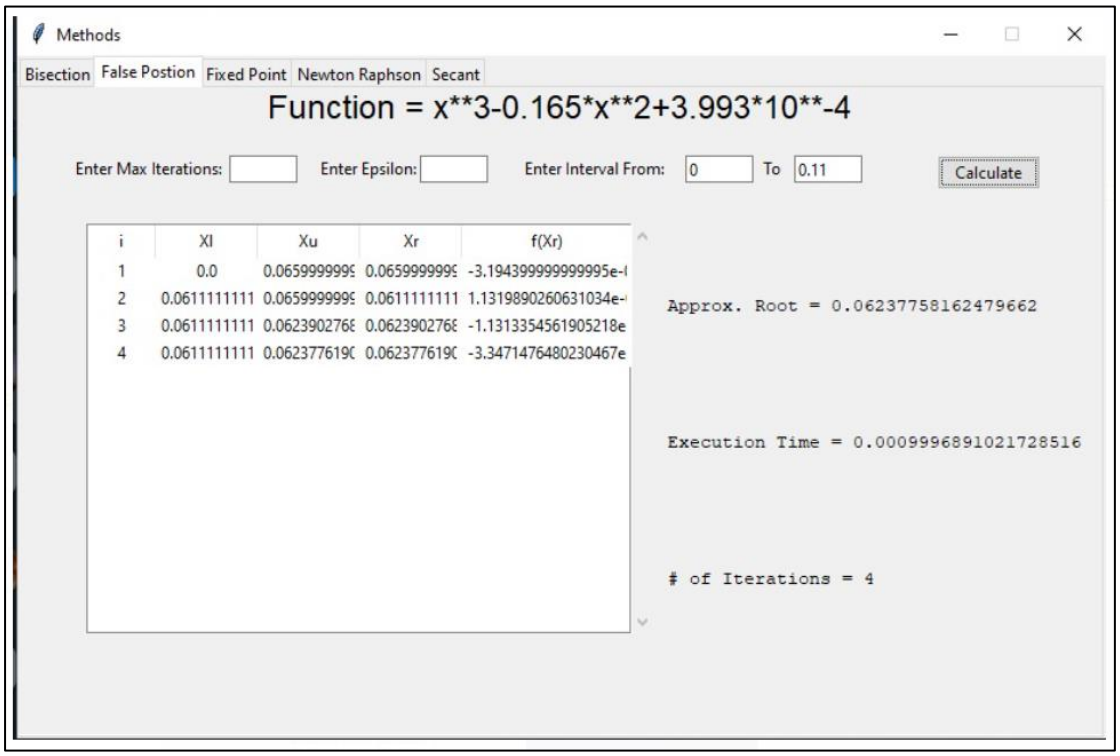
$$x^3 - 0.165x^2 + 3.993 \times 10^{-4}$$

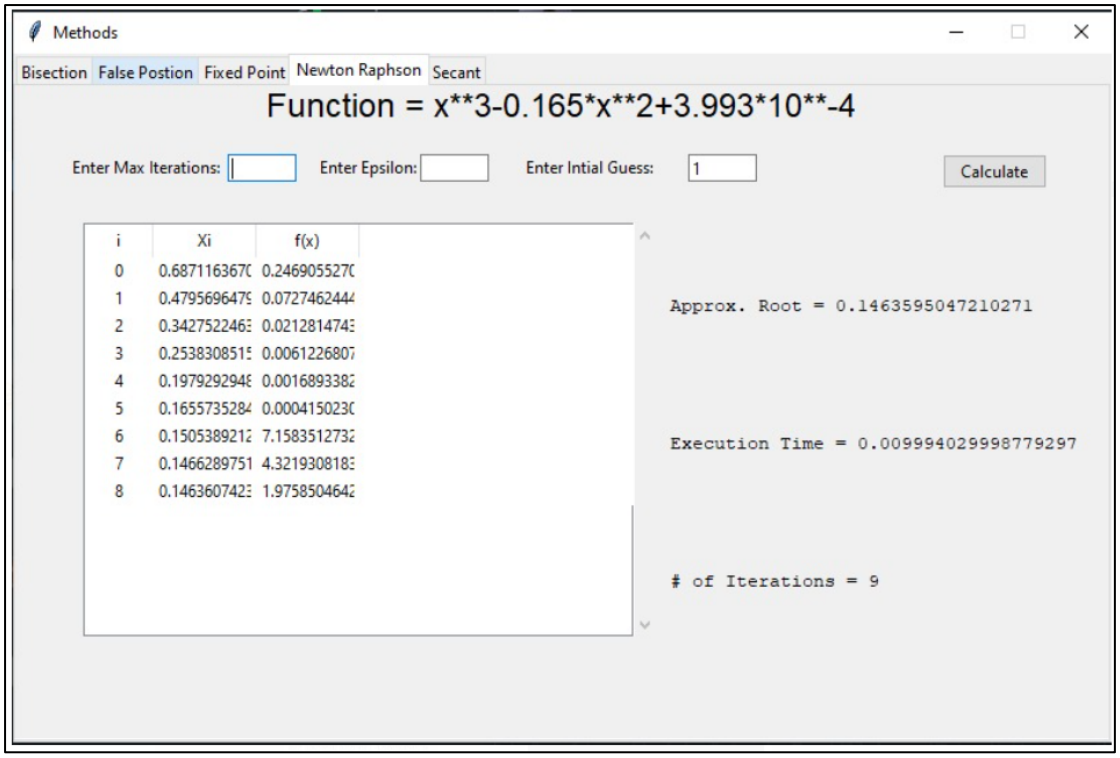
Assuming the user enter the equation in valid way and with no error

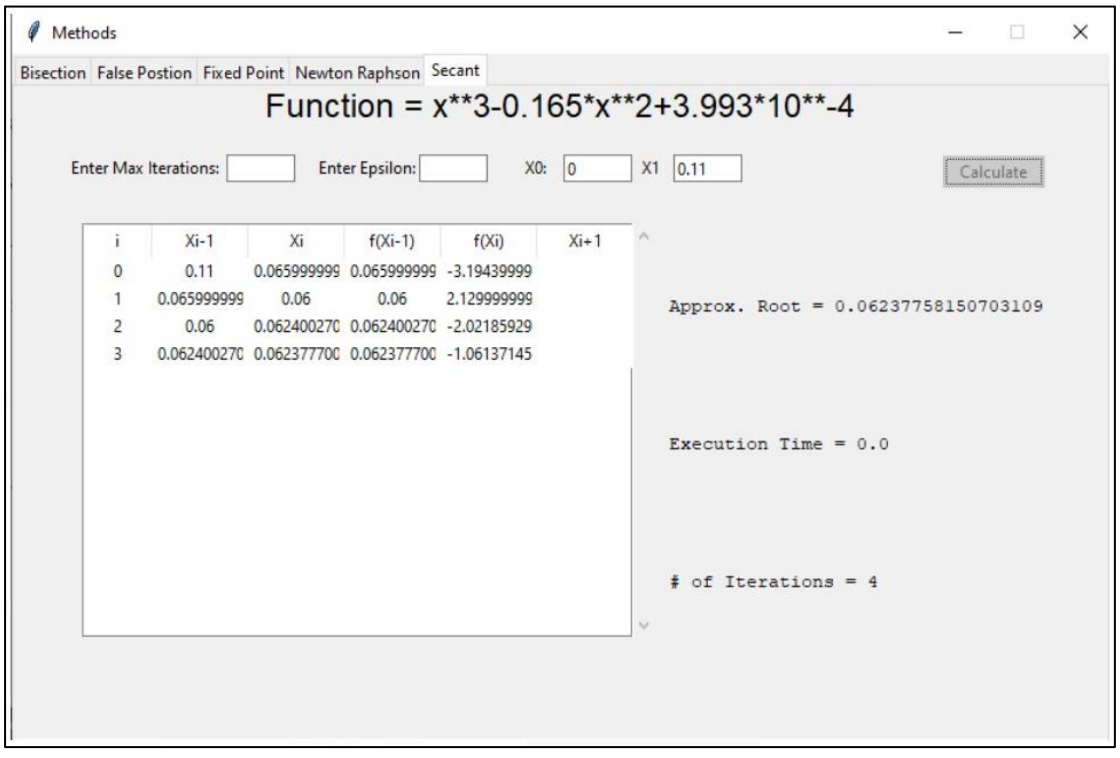



Plotting The function
Function: x**3-0.165*x**2+3.993*10**-4
Exact Root is 0.1463595046947325
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20
x

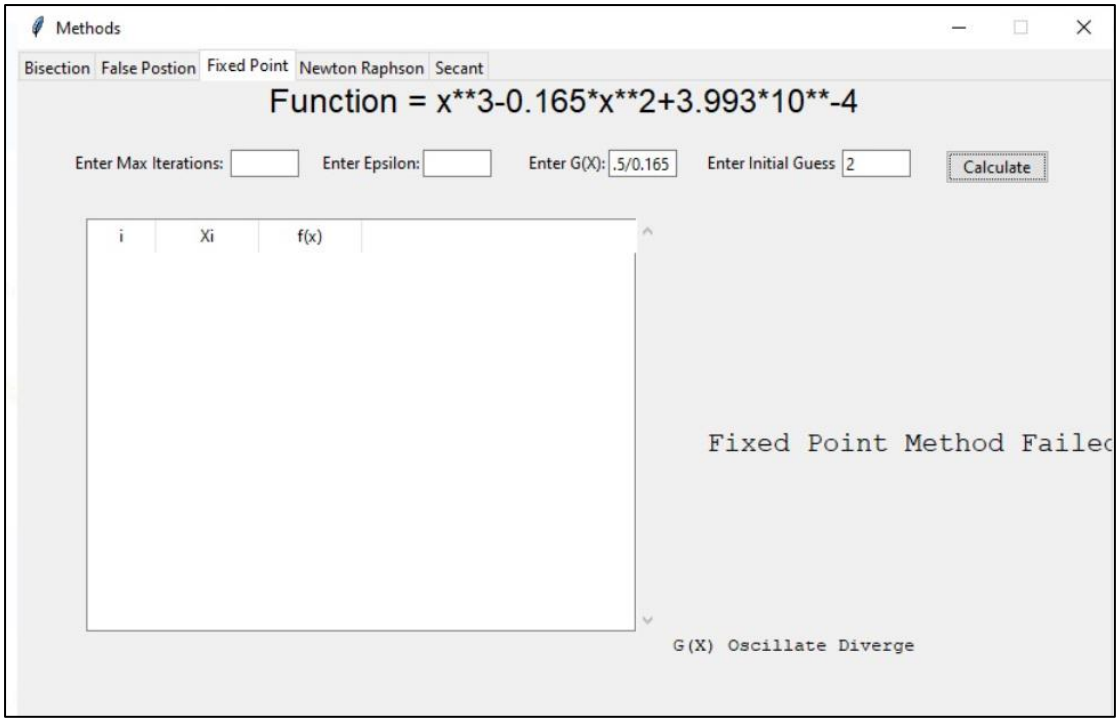

Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x**3-0.165*x**2+3.993*10**-4
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
0
To
0.11
Calculate
i
Xl
Xu
Xr
f(Xr)
1 0.055 0.11 0.055 6.654999999999996e-0
2 0.055 0.0825 0.0825 -0.00016221562500000
3 0.055 0.06875 0.06875 -5.563164062500009e-
4 0.061875 0.06875 0.061875 4.484326171874957e-0
5 0.061875 0.0653125 0.0653125 -2.5939227294921813e
6 0.061875 0.06359375 0.06359375 -1.0803610992431655e
7 0.061875 0.062734375 0.062734375 -3.1767785072328255e
8 0.062304687 0.062734375 0.062304687 6.497278094290708e-0
9 0.062304687 0.062519531 0.062519531 -1.2645666047932016e
10 0.062304687 0.062412109 0.062412109 -3.0767599288380326e
11 0.062358398 0.062412109 0.062358398 1.709622243185275e-0
12 0.062358398 0.062385253 0.062385253 -6.837286337698766e-
13 0.062371826 0.062385253 0.062371826 5.1290692960415454e-
Approx. Root = 0.06237728118896485
Execution Time = 0.007994413375854492
# of Iterations = 17
Expected # of iterations = 14


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x**3-0.165*x**2+3.993*10**-4
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
0
To
0.11
Calculate
i
Xl
Xu
Xr
f(Xr)
1 0.0 0.065999999 0.065999999 -3.194399999999995e-
2 0.061111111 0.065999999 0.061111111 1.1319890260631034e-
3 0.061111111 0.062390276 0.062390276 -1.1313354561905218e
4 0.061111111 0.062377619 0.062377619 -3.3471476480230467e
Approx. Root = 0.06237758162479662
Execution Time = 0.0009996891021728516
# of Iterations = 4



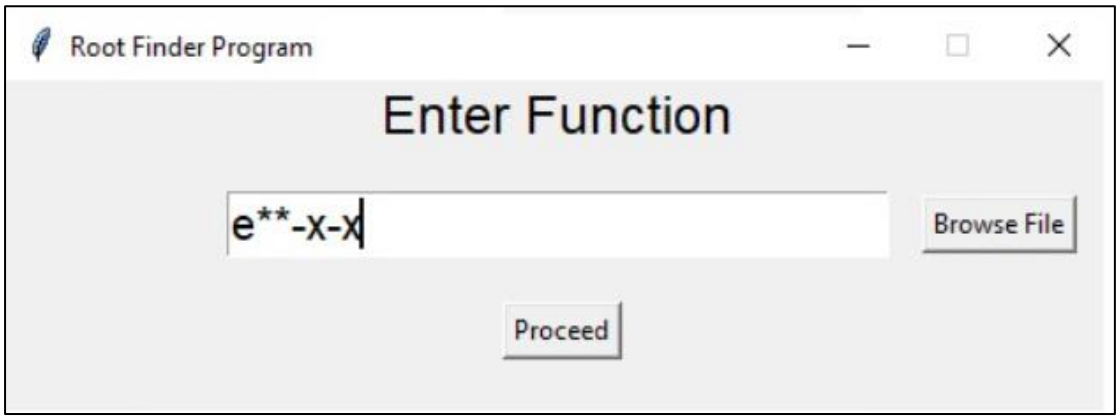


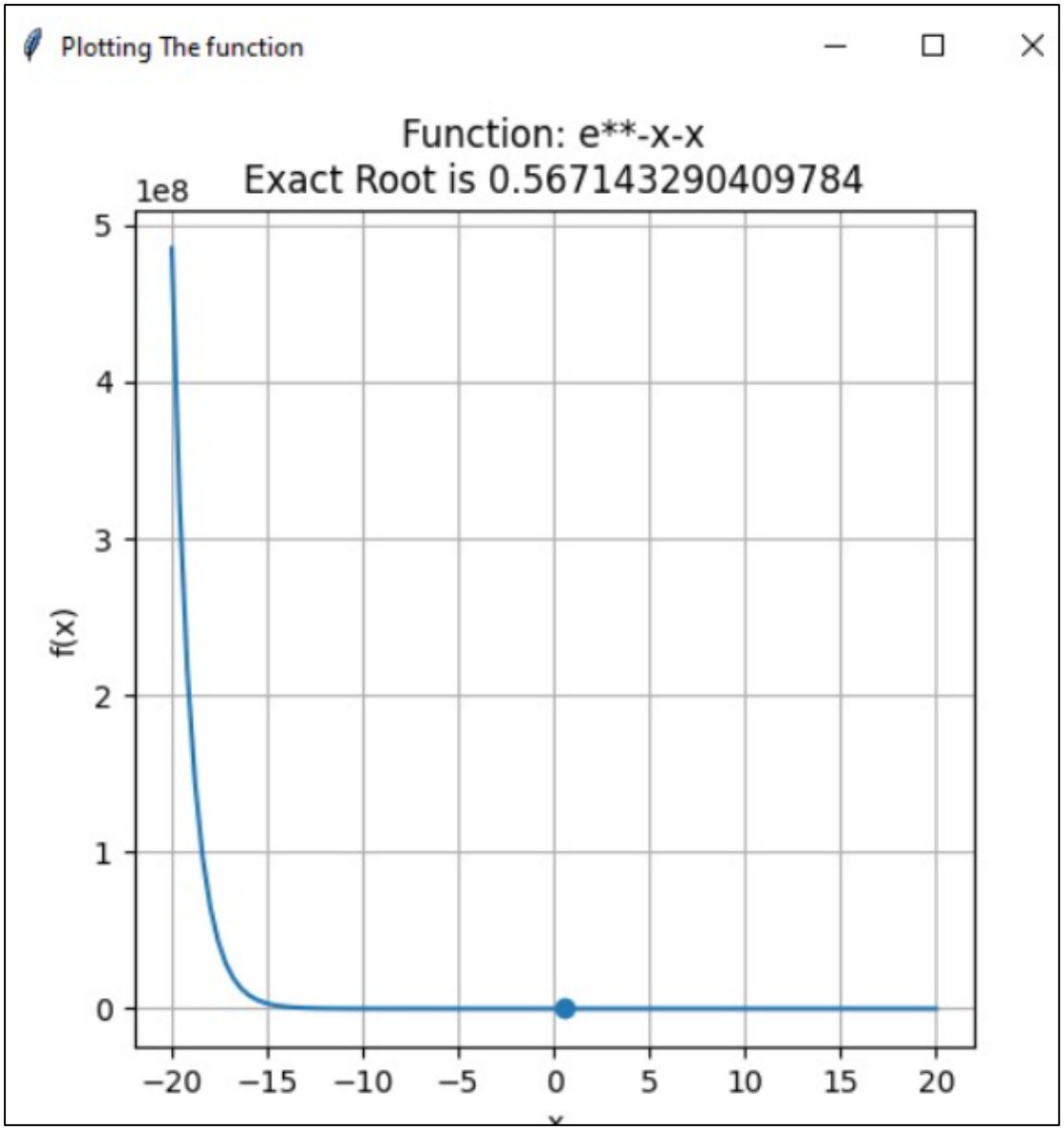
Comment: Since the equation from the third degree and some roots are different → some methods find get the same root bisection, false position & secant but newton method finds another different root depends on the initial guess, unfortunately the fixed-point method required G(x) with result fail case and couldn't find the root due to diverge based on the mean-value theorem .

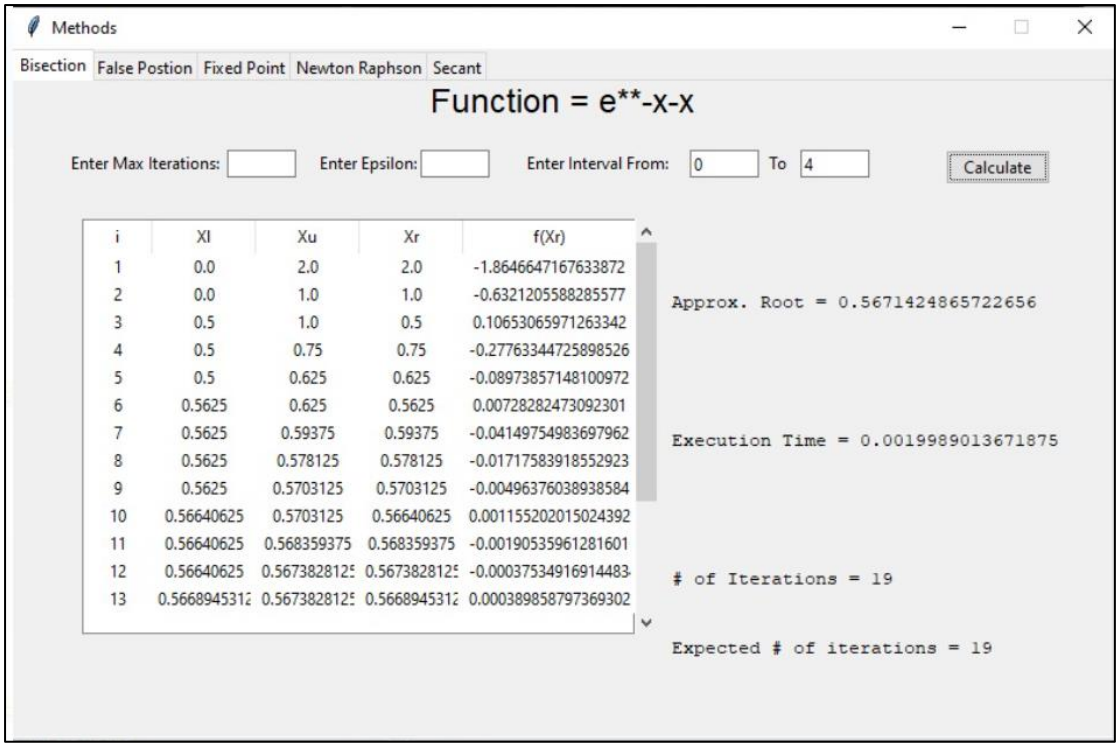


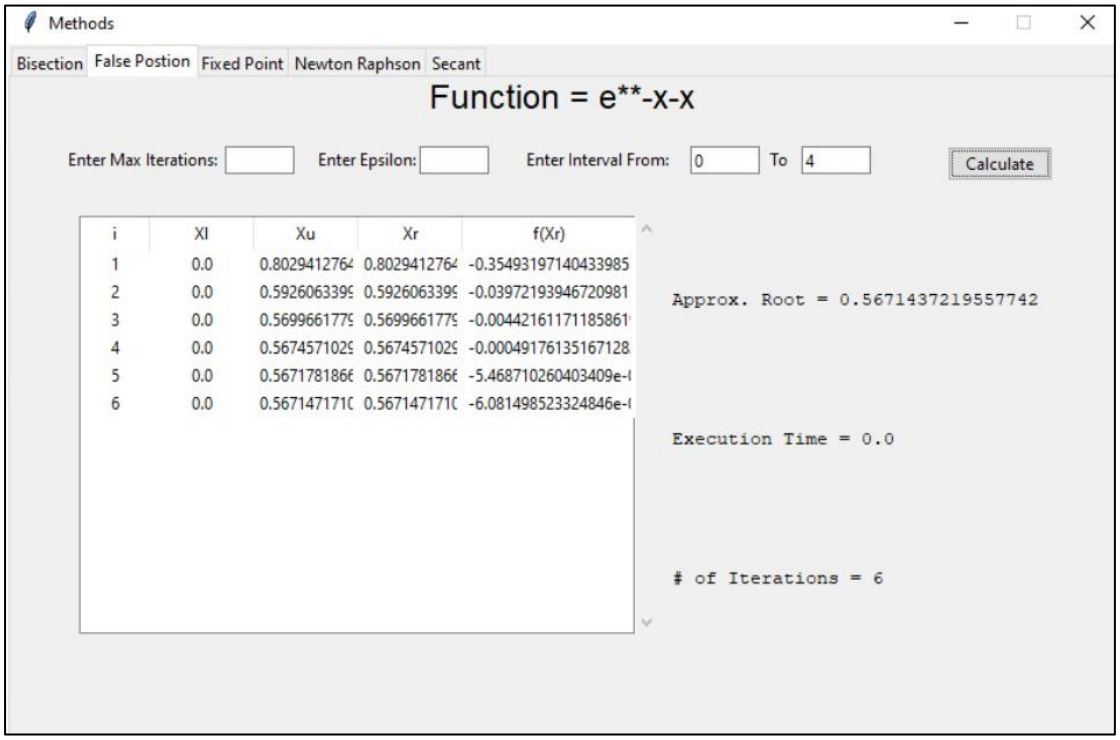
Expression two

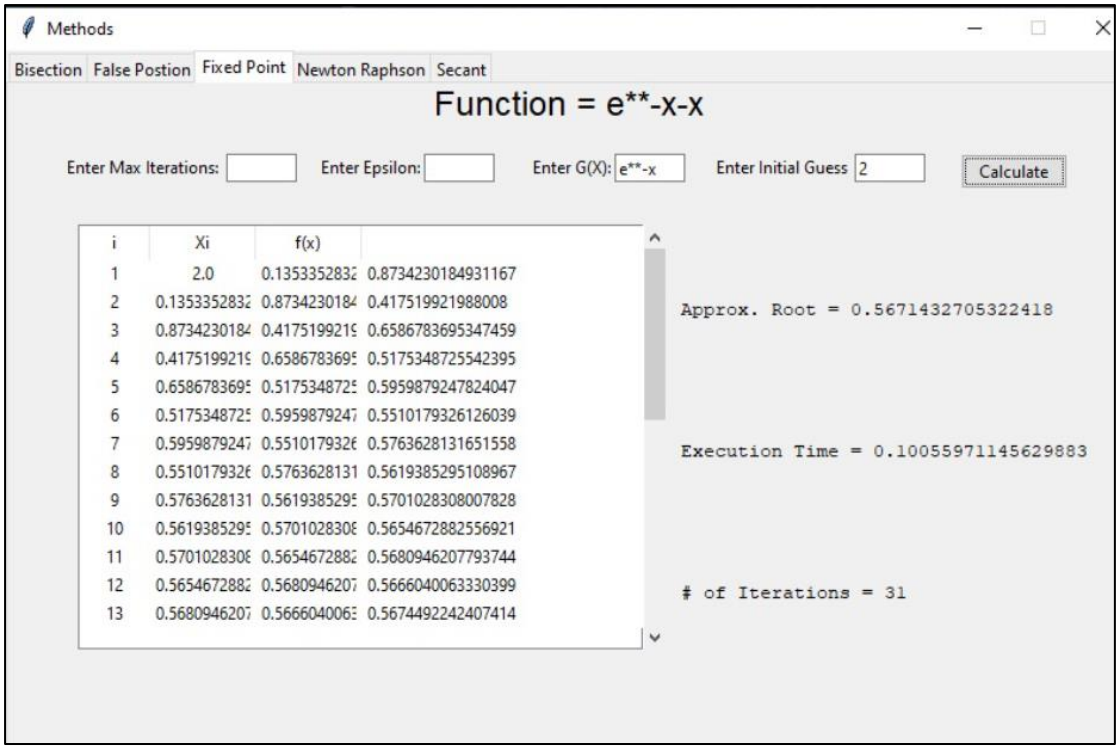
$$e^{-x} - x$$

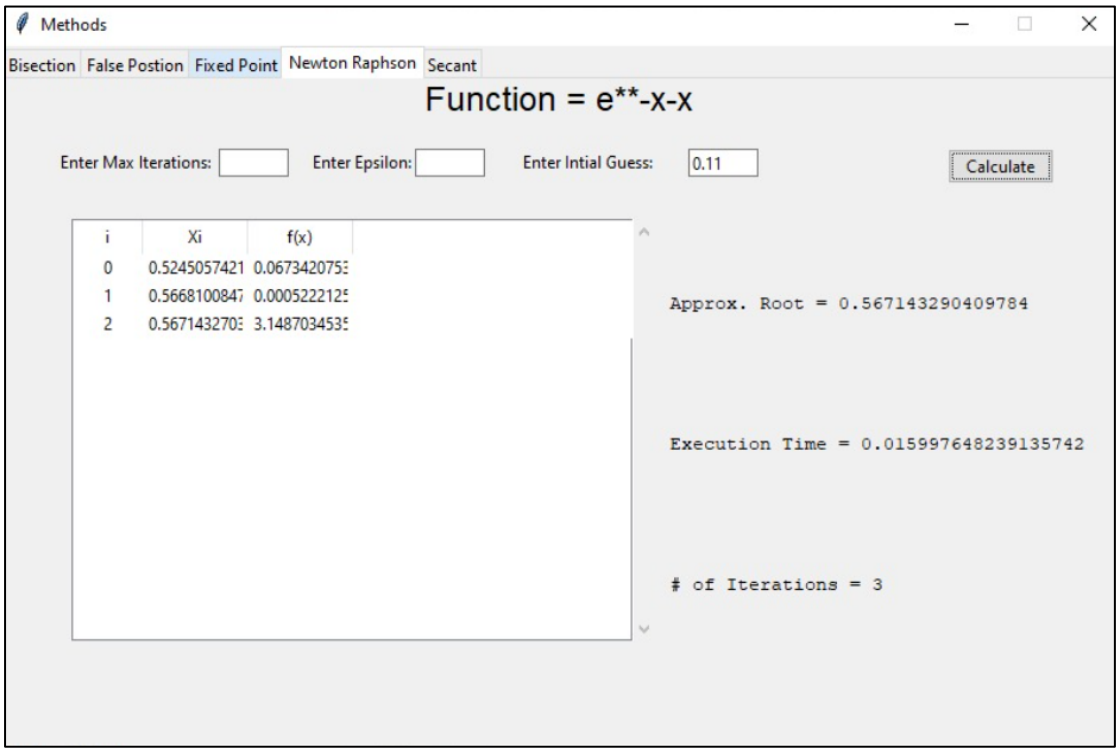


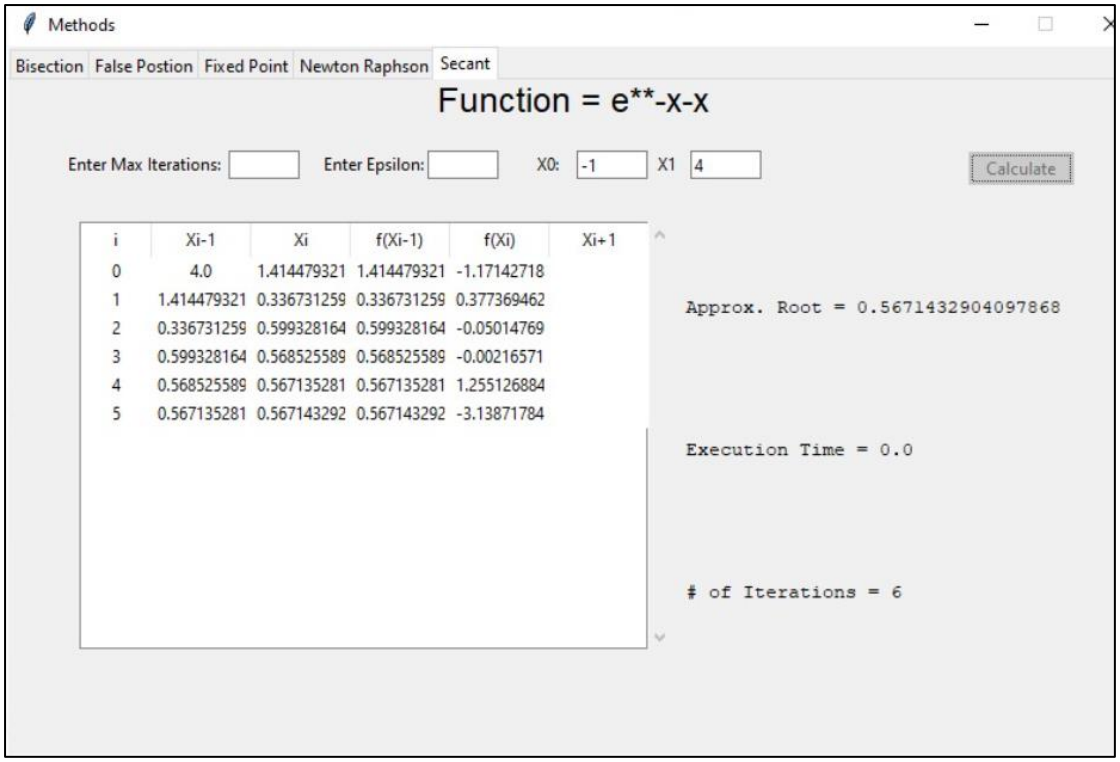

Plotting The function
Function: e**-x-x
Exact Root is 0.567143290409784
1e8
5
4
3
2
1
0
f(x)
−20
−15
−10
−5
0
5
10
15
20
x


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = e**-x-x
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
0
To
4
Calculate
i Xl Xu Xr f(Xr)
1 0.0 2.0 2.0 -1.8646647167633872
2 0.0 1.0 1.0 -0.6321205588285577
3 0.5 1.0 0.5 0.10653065971263342
4 0.5 0.75 0.75 -0.27763344725898526
5 0.5 0.625 0.625 -0.08973857148100972
6 0.5625 0.625 0.5625 0.00728282473092301
7 0.5625 0.59375 0.59375 -0.04149754983697962
8 0.5625 0.578125 0.578125 -0.01717583918552923
9 0.5625 0.5703125 0.5703125 -0.00496376038938584
10 0.56640625 0.5703125 0.56640625 0.001155202015024392
11 0.56640625 0.568359375 0.568359375 -0.00190535961281601
12 0.56640625 0.5673828125 0.5673828125 -0.00037534916914483
13 0.5668945312 0.5673828125 0.5668945312 0.000389858797369302
Approx. Root = 0.5671424865722656
Execution Time = 0.0019989013671875
# of Iterations = 19
Expected # of iterations = 19


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = e**-x-x
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
0
To
4
Calculate
i Xl Xu Xr f(Xr)
1 0.0 0.8029412764 0.8029412764 -0.35493197140433985
2 0.0 0.5926063395 0.5926063395 -0.03972193946720981
3 0.0 0.5699661775 0.5699661775 -0.00442161171185861
4 0.0 0.5674571025 0.5674571025 -0.00049176135167128
5 0.0 0.5671781866 0.5671781866 -5.468710260403409e-
6 0.0 0.5671471710 0.5671471710 -6.081498523324846e-
Approx. Root = 0.5671437219557742
Execution Time = 0.0
# of Iterations = 6


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = e**-x-x
Enter Max Iterations:
Enter Epsilon:
Enter G(X):
e**-x
Enter Initial Guess
2
Calculate
i
Xi
f(x)
1 2.0 0.1353352832 0.8734230184931167
2 0.1353352832 0.8734230184 0.417519921988008
3 0.8734230184 0.4175199219 0.6586783695347459
4 0.4175199219 0.6586783695 0.5175348725542395
5 0.6586783695 0.5175348725 0.5959879247824047
6 0.5175348725 0.5959879247 0.5510179326126039
7 0.5959879247 0.5510179326 0.5763628131651558
8 0.5510179326 0.5763628131 0.5619385295108967
9 0.5763628131 0.5619385295 0.5701028308007828
10 0.5619385295 0.5701028308 0.5654672882556921
11 0.5701028308 0.5654672882 0.5680946207793744
12 0.5654672882 0.5680946207 0.5666040063330399
13 0.5680946207 0.5666040063 0.5674492242407414
Approx. Root = 0.5671432705322418
Execution Time = 0.10055971145629883
# of Iterations = 31

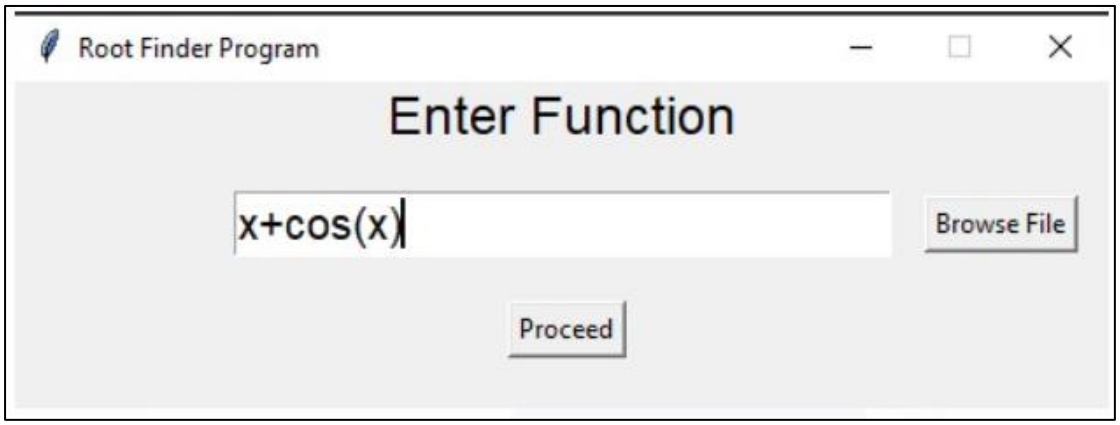

Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = e**-x-x
Enter Max Iterations:
Enter Epsilon:
Enter Intial Guess:
0.11
Calculate
i
Xi
f(x)
0 0.5245057421 0.0673420753
1 0.5668100847 0.0005222125
2 0.5671432703 3.1487034535
Approx. Root = 0.567143290409784
Execution Time = 0.015997648239135742
# of Iterations = 3

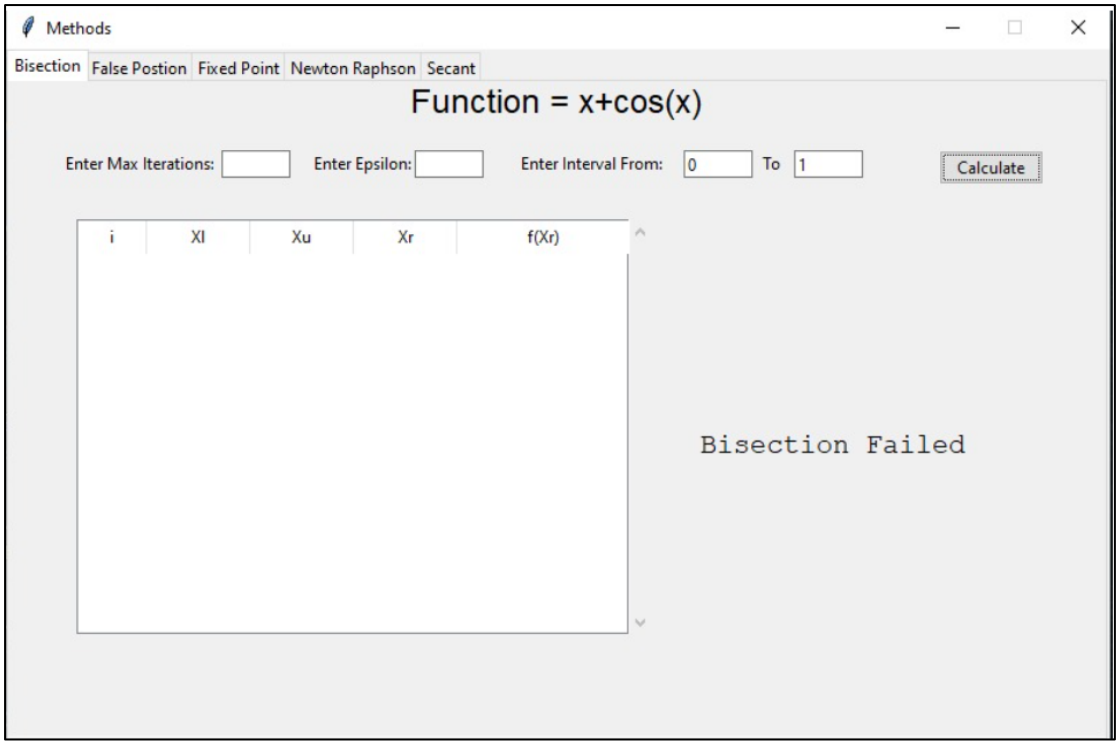


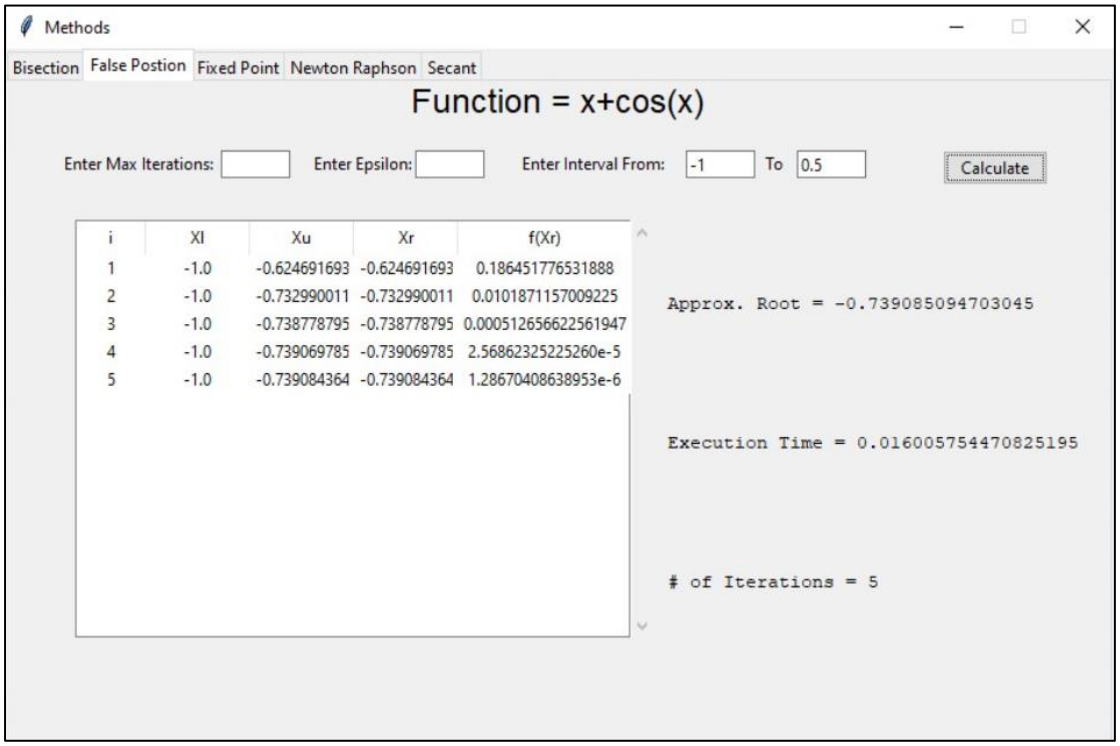
Comment: all the methods returned almost the same root but secant and newton found it faster as we use the graphical plotting for the guessing .

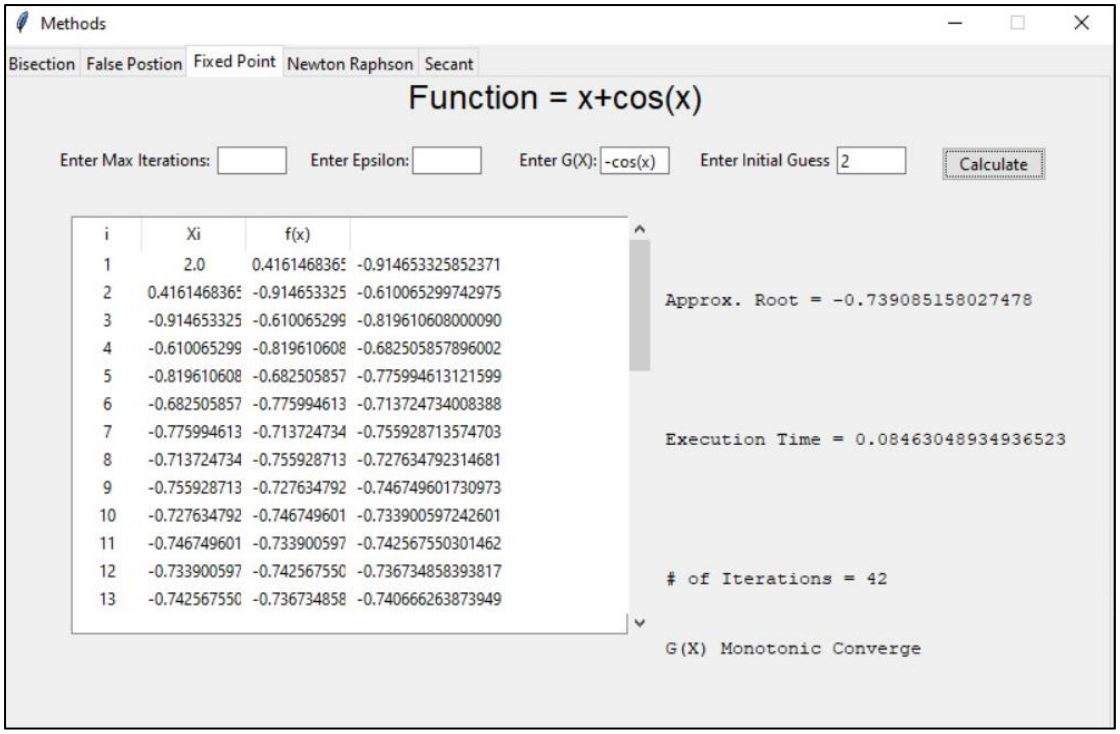
Expression three

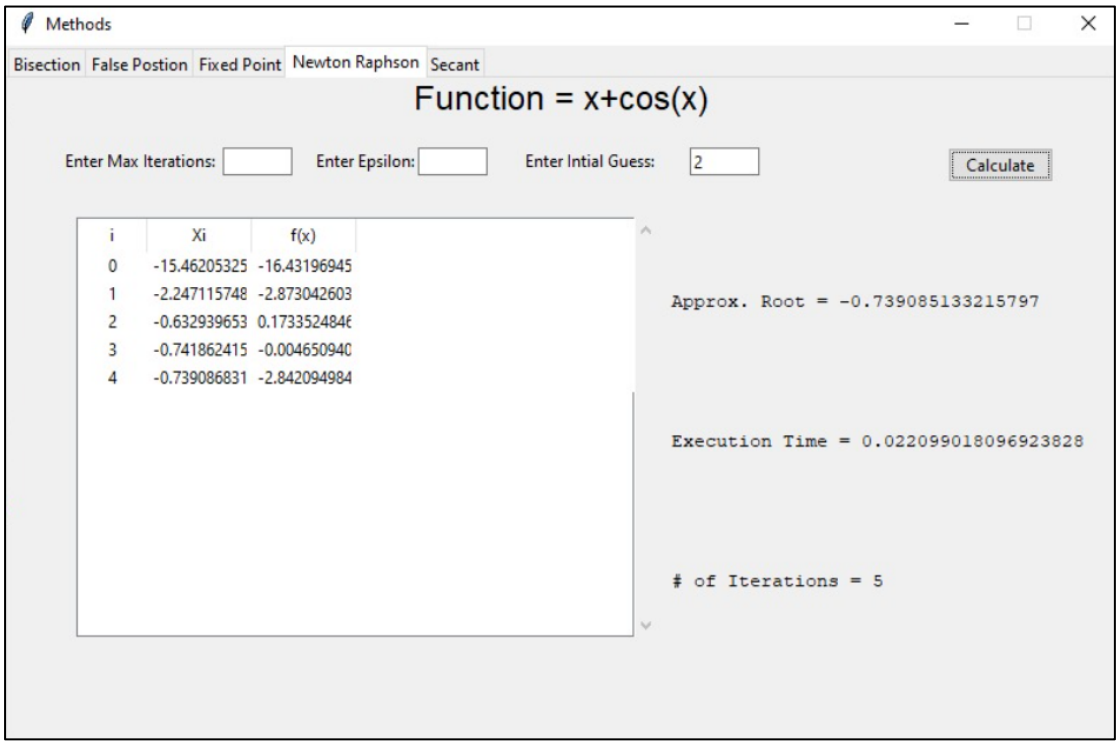
$$x + \cos(x)$$

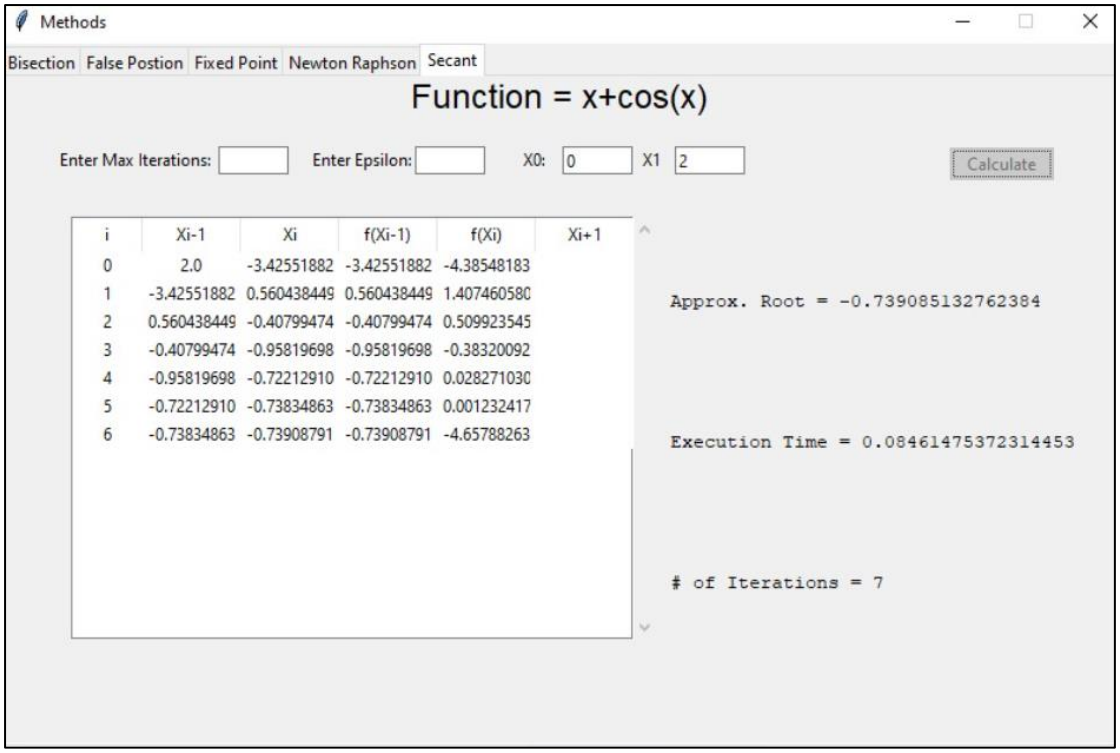



Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x+cos(x)
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
0
To
1
Calculate
i
Xl
Xu
Xr
f(Xr)
Bisection Failed


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x+cos(x)
Enter Max Iterations:
Enter Epsilon:
Enter Interval From:
-1
To
0.5
Calculate
i
Xl
Xu
Xr
f(Xr)
1 -1.0 -0.624691693 -0.624691693 0.186451776531888
2 -1.0 -0.732990011 -0.732990011 0.0101871157009225
3 -1.0 -0.738778795 -0.738778795 0.000512656622561947
4 -1.0 -0.739069785 -0.739069785 2.56862325225260e-5
5 -1.0 -0.739084364 -0.739084364 1.28670408638953e-6
Approx. Root = -0.739085094703045
Execution Time = 0.016005754470825195
# of Iterations = 5


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x+cos(x)
Enter Max Iterations:
Enter Epsilon:
Enter G(X):
-cos(x)
Enter Initial Guess
2
Calculate
i
Xi
f(x)
1 2.0 0.4161468365 -0.914653325852371
2 0.4161468365 -0.914653325 -0.610065299742975
3 -0.914653325 -0.610065299 -0.819610608000090
4 -0.610065299 -0.819610608 -0.682505857896002
5 -0.819610608 -0.682505857 -0.775994613121599
6 -0.682505857 -0.775994613 -0.713724734008388
7 -0.775994613 -0.713724734 -0.755928713574703
8 -0.713724734 -0.755928713 -0.727634792314681
9 -0.755928713 -0.727634792 -0.746749601730973
10 -0.727634792 -0.746749601 -0.733900597242601
11 -0.746749601 -0.733900597 -0.742567550301462
12 -0.733900597 -0.742567550 -0.736734858393817
13 -0.742567550 -0.736734858 -0.740666263873949
Approx. Root = -0.739085158027478
Execution Time = 0.08463048934936523
# of Iterations = 42
G(X) Monotonic Converge


Methods
Bisection
False Postion
Fixed Point
Newton Raphson
Secant
Function = x+cos(x)
Enter Max Iterations:
Enter Epsilon:
Enter Intial Guess:
2
Calculate
i
Xi
f(x)
0 -15.46205325 -16.43196945
1 -2.247115748 -2.873042603
2 -0.632939653 0.1733524846
3 -0.741862415 -0.004650940
4 -0.739086831 -2.842094984
Approx. Root = -0.739085133215797
Execution Time = 0.022099018096923828
# of Iterations = 5



Comment: we have faced problem with plotting the trigonometric function due to the eval python function that has been updated on recent python version that can't compute the lambda expression for plotting so we couldn't predict the interval for the bisection method so no result produced. However, all others methods returned the same root fixed point takes more iteration to find but it converges after all.

Problems we faced

1. Due to some versions update for python we couldn’t plotting any trigonometric functions so the initial guess and any parameters we choose based on the graphical solution was hard to use.
2. We found it really challenging to find the magic function for the fixed-point method to perform the root finder algorithm so we assumed that the user can choose any magic function format can be generated from the expression and embed it as parameter in the algorithm.
3. Some functions for newton – Raphson's method produce division by zero due to the denominator may be number very close from the zero due to the differentiation of the original expression , we tried to prevent dividing by exact zero.

Code

We will provide the code in the submission form and we upload into GitHub link .