Numerical Analysis and Computer Applications

Root Finder & Interpolation

Developed by :

Eshraq Ibrahim (12)

Rowan Adel (25)

Hanan Samir (20)

Nancy Abdalkareem (69)

Yomna Eldawy (71)

Table of contents:

Overview

The program consists of two parts. The first part is Root Finder program where user enters an equation to find out its solution by various methods, one of the following methods can be chosen: Bisection, False-Position, Fixed-Point, Newton-Raphson, Secant, Birge-Vieta and General algorithm developed by the team. The program gets the nearest root to the initial point entered by the user.

The second part is a program for querying the values of specific points using interpolation which takes as an input the polynomial order, sample point(s), corresponding value(s), the interpolation technique to use (Newton – Lagrange) and the query point(s).

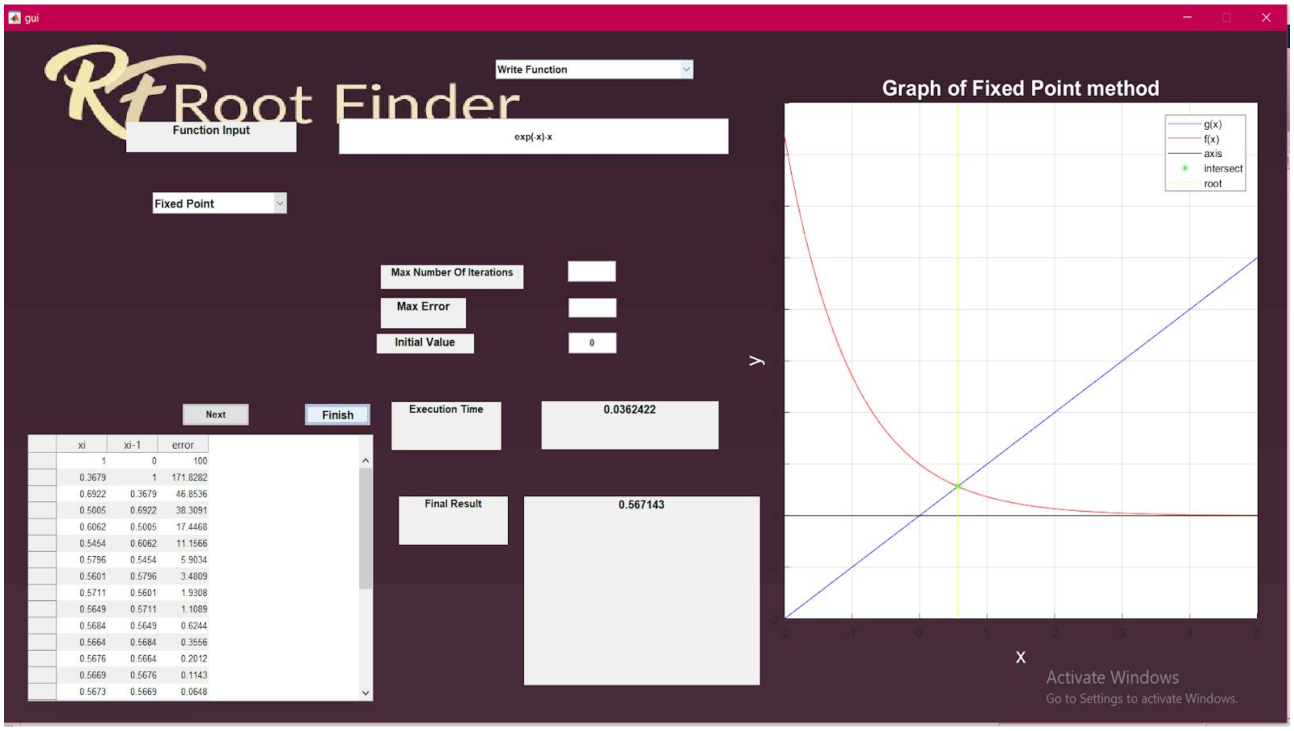
The aim is to compare and analyse the behaviour of the different numerical methods.

The GUI is divided into two parts one for each part

Part I

GUI:

Designed for the Root Finder Program. The user can enter input either through the GUI or by reading from a file. A drop down list to choose the method, text boxes for the initial points, the max number of iterations and the tolerance. Where the max number of iterations and the tolerance are not necessarily added by the user; they have initial values 50 and 0.00001 respectively. The Roots and Errors are shown in a UITable and a graph to show the roots along with the number of iterations.The Execution time and the final root also appear in text box . The General Algorithm needs only the equation and it generates the initial points on its own.



Function Input: exp(-x)-x

Max Number Of Iterations:

Max Error:

Initial Value: 0

Execution Time: 0.0362422

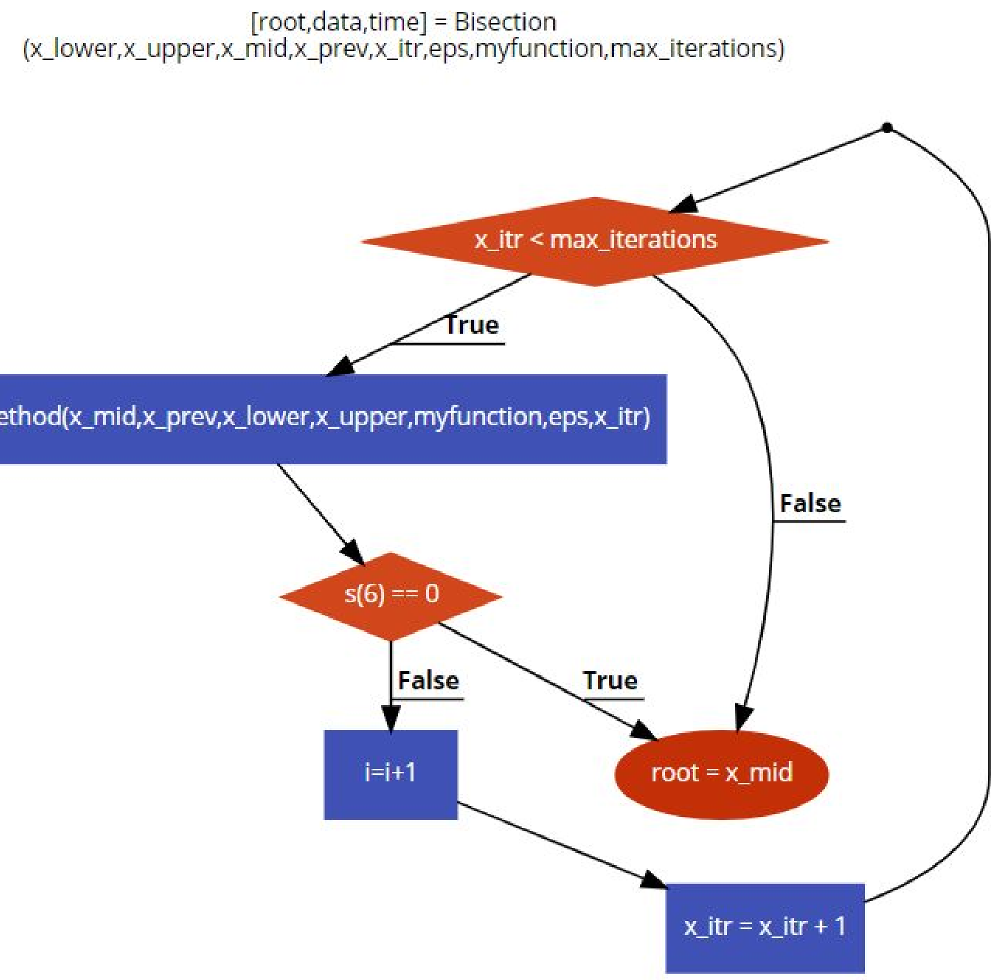
Final Result: 0.567143

xi	xi-1	error
1	0	100
0.3679	1	171.8282
0.6922	0.3679	46.8536
0.5005	0.6922	38.3091
0.6062	0.5005	17.4468
0.5454	0.6062	11.1566
0.5796	0.5454	5.9034
0.5601	0.5796	3.4809
0.5711	0.5601	1.9308
0.5649	0.5711	1.1089
0.5684	0.5649	0.6244
0.5664	0.5684	0.3556
0.5676	0.5664	0.2012
0.5669	0.5676	0.1143
0.5673	0.5669	0.0648

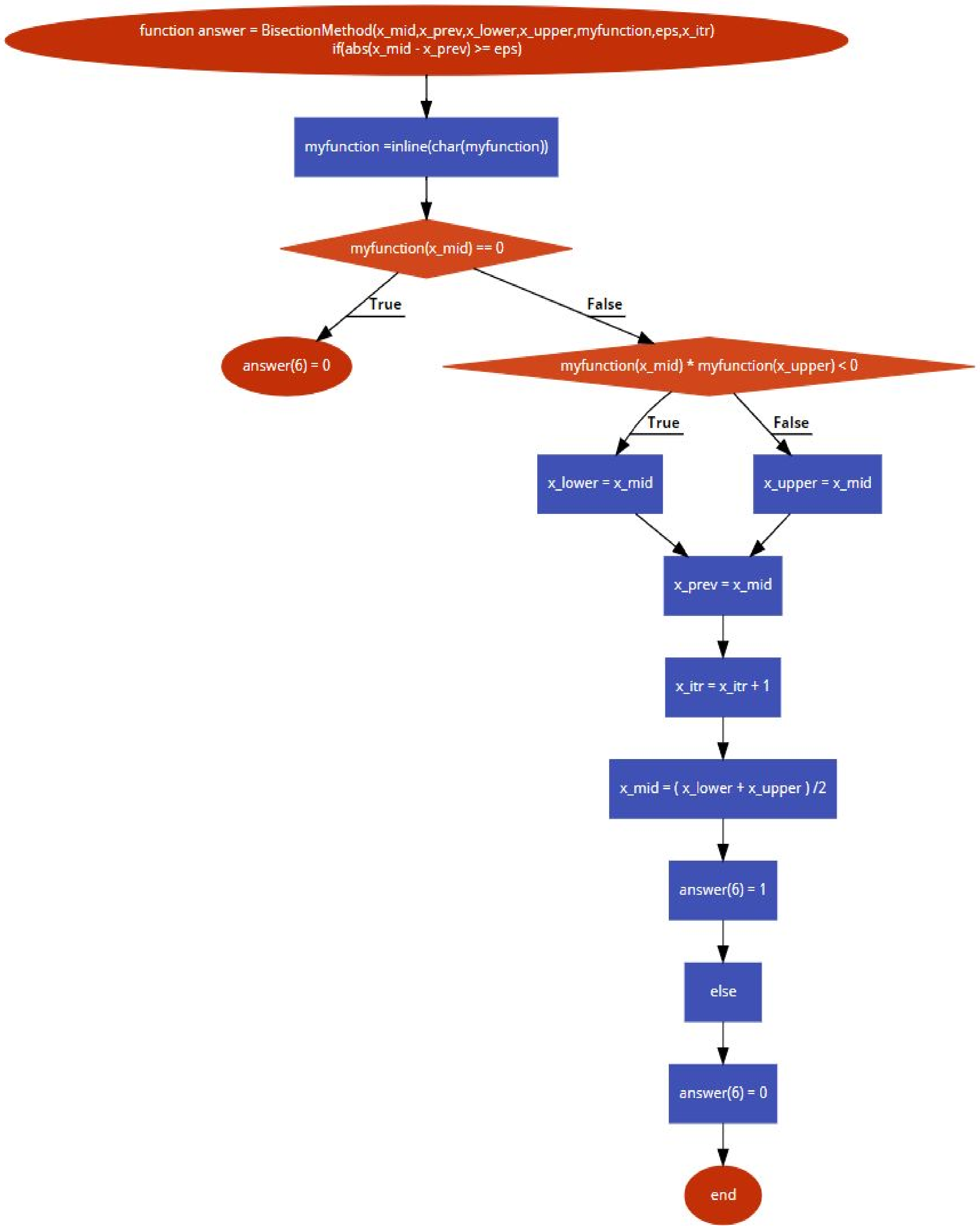
Bisection Method

a. Flow Chart

Fast Bisection flowchart that uses slow bisection method


[root,data,time] = Bisection
(x_lower,x_upper,x_mid,x_prev,x_itr,eps,myfunction,max_iterations)
x_itr < max_iterations
True
False
s = BisectionMethod(x_mid,x_prev,x_lower,x_upper,myfunction,eps,x_itr)
s(6) == 0
False
True
i=i+1
root = x_mid
x_itr = x_itr + 1

Slow Bisection that get x_mid iteration by iteration


function answer = BisectionMethod(x_mid,x_prev,x_lower,x_upper,myfunction,eps,x_itr)
if(abs(x_mid - x_prev) >= eps)
myfunction =inline(char(myfunction))
myfunction(x_mid) == 0
True
False
answer(6) = 0
myfunction(x_mid) * myfunction(x_upper) < 0
True
False
x_lower = x_mid
x_upper = x_mid
x_prev = x_mid
x_itr = x_itr + 1
x_mid = (x_lower + x_upper) /2
answer(6) = 1
else
answer(6) = 0
end

b. Built in data structures or functions used

   i. Data structures

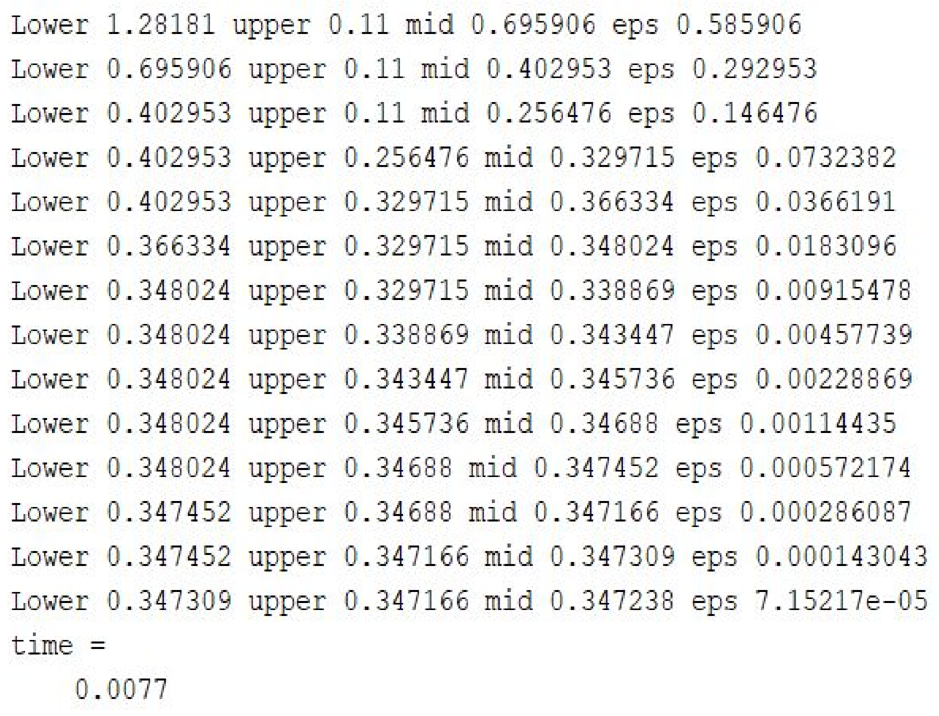
      1. Vector to store data from slow bisection to be used in fast

   ii. Built in functions

      1. Inline : to convert the string function to a character array.
      2. Tic toc : to get the execution time for the method .

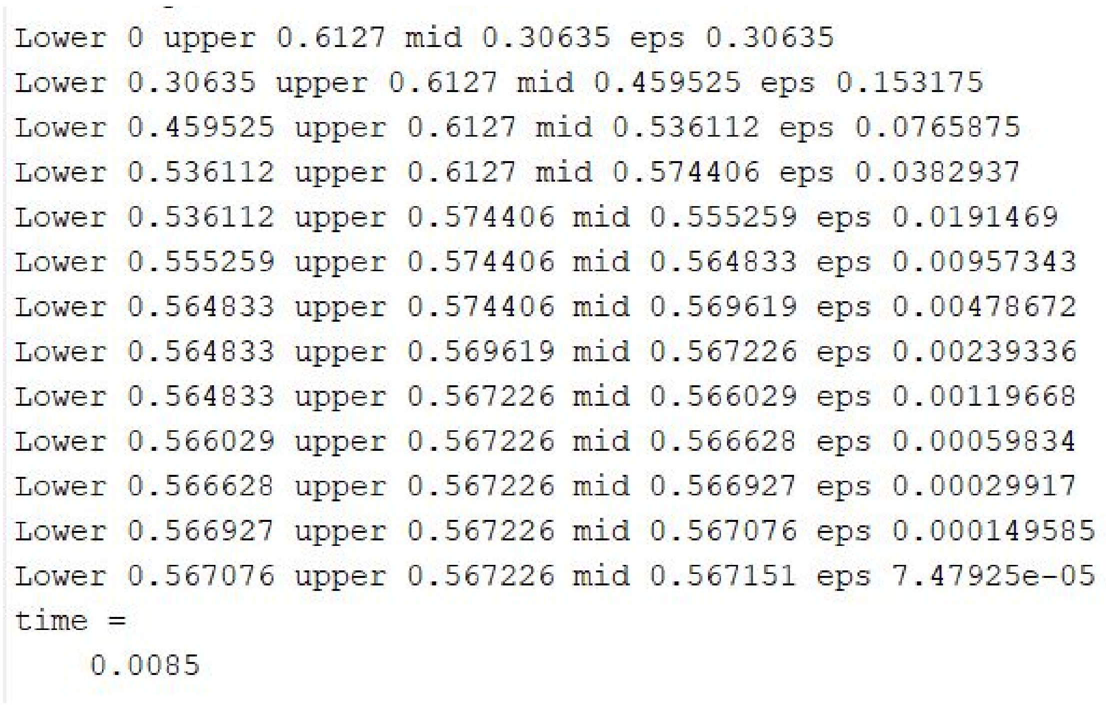
c. Equations and Analysis

   i. x^3-3*x+1, xlower = 1.5 , xUpper = 0.11



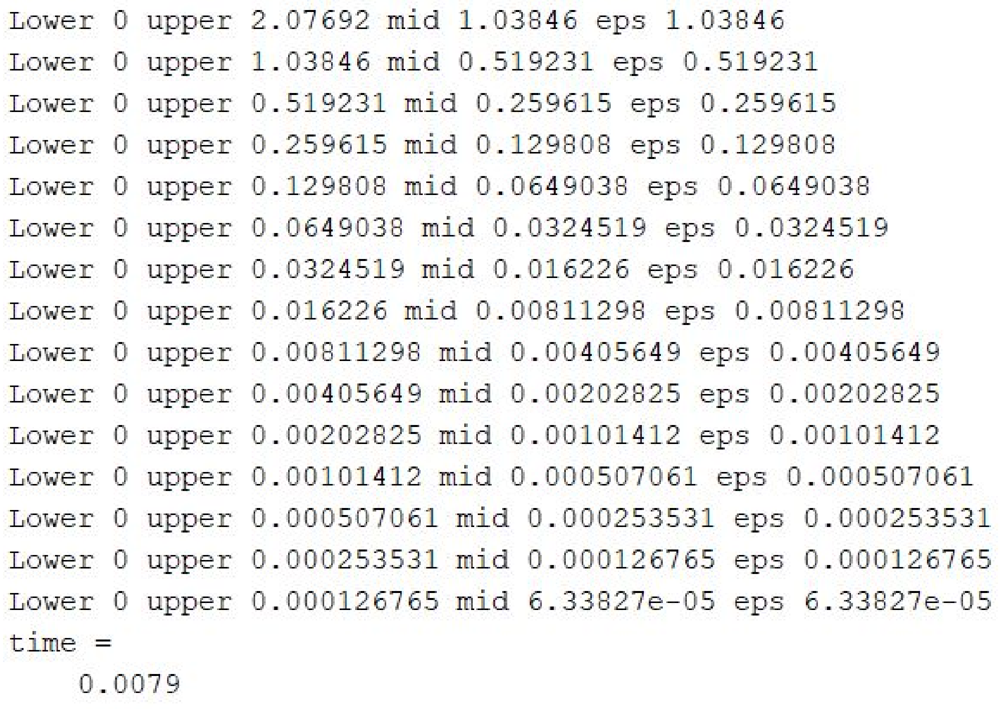
```
Lower 1.28181 upper 0.11 mid 0.695906 eps 0.585906
Lower 0.695906 upper 0.11 mid 0.402953 eps 0.292953
Lower 0.402953 upper 0.11 mid 0.256476 eps 0.146476
Lower 0.402953 upper 0.256476 mid 0.329715 eps 0.0732382
Lower 0.402953 upper 0.329715 mid 0.366334 eps 0.0366191
Lower 0.366334 upper 0.329715 mid 0.348024 eps 0.0183096
Lower 0.348024 upper 0.329715 mid 0.338869 eps 0.00915478
Lower 0.348024 upper 0.338869 mid 0.343447 eps 0.00457739
Lower 0.348024 upper 0.343447 mid 0.345736 eps 0.00228869
Lower 0.348024 upper 0.345736 mid 0.34688 eps 0.00114435
Lower 0.348024 upper 0.34688 mid 0.347452 eps 0.000572174
Lower 0.347452 upper 0.34688 mid 0.347166 eps 0.000286087
Lower 0.347452 upper 0.347166 mid 0.347309 eps 0.000143043
Lower 0.347309 upper 0.347166 mid 0.347238 eps 7.15217e-05
time =
    0.0077
```

   ii. exp(-x)-x , x lower = 0, x Upper = 1



```
Lower 0 upper 0.6127 mid 0.30635 eps 0.30635
Lower 0.30635 upper 0.6127 mid 0.459525 eps 0.153175
Lower 0.459525 upper 0.6127 mid 0.536112 eps 0.0765875
Lower 0.536112 upper 0.6127 mid 0.574406 eps 0.0382937
Lower 0.536112 upper 0.574406 mid 0.555259 eps 0.0191469
Lower 0.555259 upper 0.574406 mid 0.564833 eps 0.00957343
Lower 0.564833 upper 0.574406 mid 0.569619 eps 0.00478672
Lower 0.564833 upper 0.569619 mid 0.567226 eps 0.00239336
Lower 0.564833 upper 0.567226 mid 0.566029 eps 0.00119668
Lower 0.566029 upper 0.567226 mid 0.566628 eps 0.00059834
Lower 0.566628 upper 0.567226 mid 0.566927 eps 0.00029917
Lower 0.566927 upper 0.567226 mid 0.567076 eps 0.000149585
Lower 0.567076 upper 0.567226 mid 0.567151 eps 7.47925e-05
time =
    0.0085
```

iii. x^5-11*x^4+46*x^3-90*x^2+81*x-27 , x lower = 0 , x upper = 2

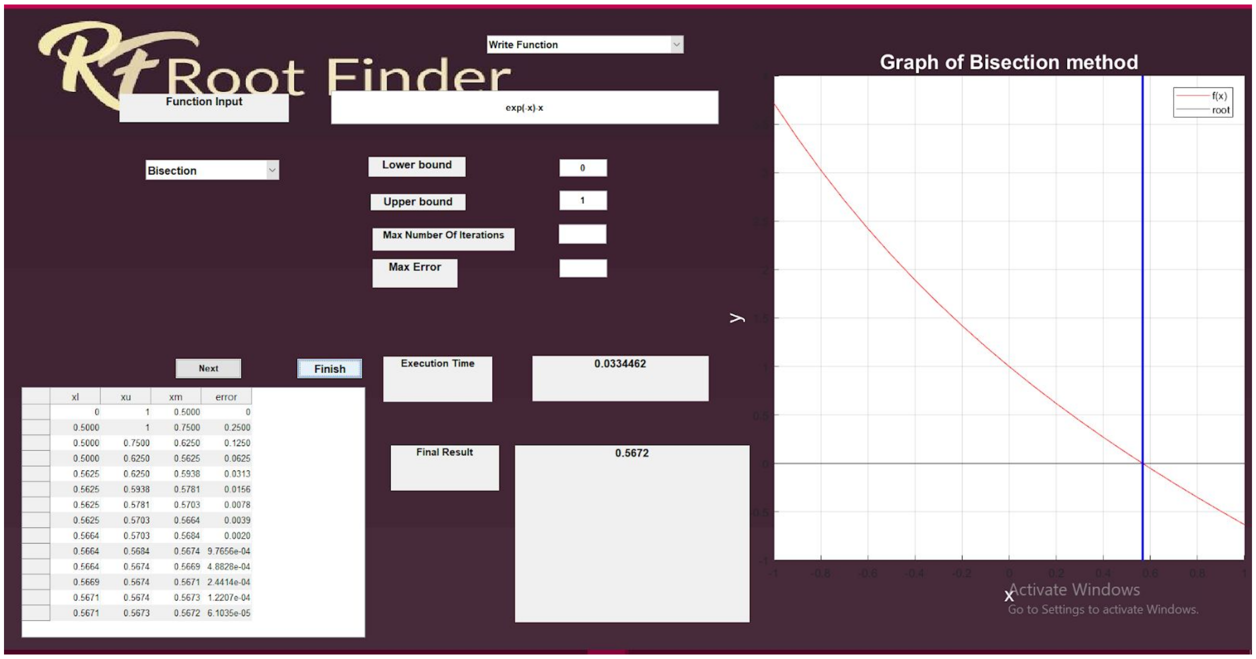


```
Lower 0 upper 2.07692 mid 1.03846 eps 1.03846
Lower 0 upper 1.03846 mid 0.519231 eps 0.519231
Lower 0 upper 0.519231 mid 0.259615 eps 0.259615
Lower 0 upper 0.259615 mid 0.129808 eps 0.129808
Lower 0 upper 0.129808 mid 0.0649038 eps 0.0649038
Lower 0 upper 0.0649038 mid 0.0324519 eps 0.0324519
Lower 0 upper 0.0324519 mid 0.016226 eps 0.016226
Lower 0 upper 0.016226 mid 0.00811298 eps 0.00811298
Lower 0 upper 0.00811298 mid 0.00405649 eps 0.00405649
Lower 0 upper 0.00405649 mid 0.00202825 eps 0.00202825
Lower 0 upper 0.00202825 mid 0.00101412 eps 0.00101412
Lower 0 upper 0.00101412 mid 0.000507061 eps 0.000507061
Lower 0 upper 0.000507061 mid 0.000253531 eps 0.000253531
Lower 0 upper 0.000253531 mid 0.000126765 eps 0.000126765
Lower 0 upper 0.000126765 mid 6.33827e-05 eps 6.33827e-05
time =
    0.0079
```

d. Problematic Functions
    i. Function changes sign but root does not exist
        1. Example : y = 1 / x
    ii. Cannot handle multiple even roots as upper and lower bounds for this root will have the same sign

e. Sample Runs

- In Fast Mode



Function Input: exp(-x)-x

Bisection

Lower bound: 0

Upper bound: 1

Max Number Of Iterations

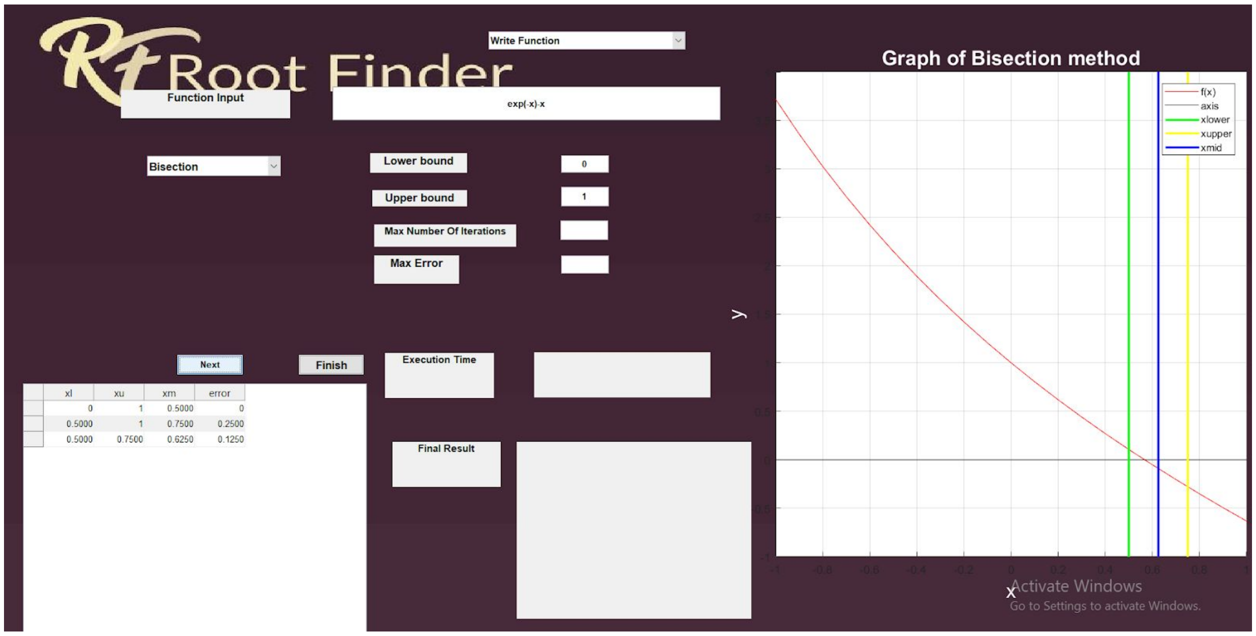
Max Error

Execution Time: 0.0334462

Final Result: 0.5672

	xl	xu	xm	error
	0	1	0.5000	0
	0.5000	1	0.7500	0.2500
	0.5000	0.7500	0.6250	0.1250
	0.5000	0.6250	0.5625	0.0625
	0.5625	0.6250	0.5938	0.0313
	0.5625	0.5938	0.5781	0.0156
	0.5625	0.5781	0.5703	0.0078
	0.5625	0.5703	0.5664	0.0039
	0.5664	0.5703	0.5684	0.0020
	0.5664	0.5684	0.5674	9.7656e-04
	0.5664	0.5674	0.5669	4.8828e-04
	0.5669	0.5674	0.5671	2.4414e-04
	0.5671	0.5674	0.5673	1.2207e-04
	0.5671	0.5673	0.5672	6.1035e-05

- In Slow Mode



Function Input: exp(-x)-x

Bisection

Lower bound: 0

Upper bound: 1

Max Number Of Iterations:

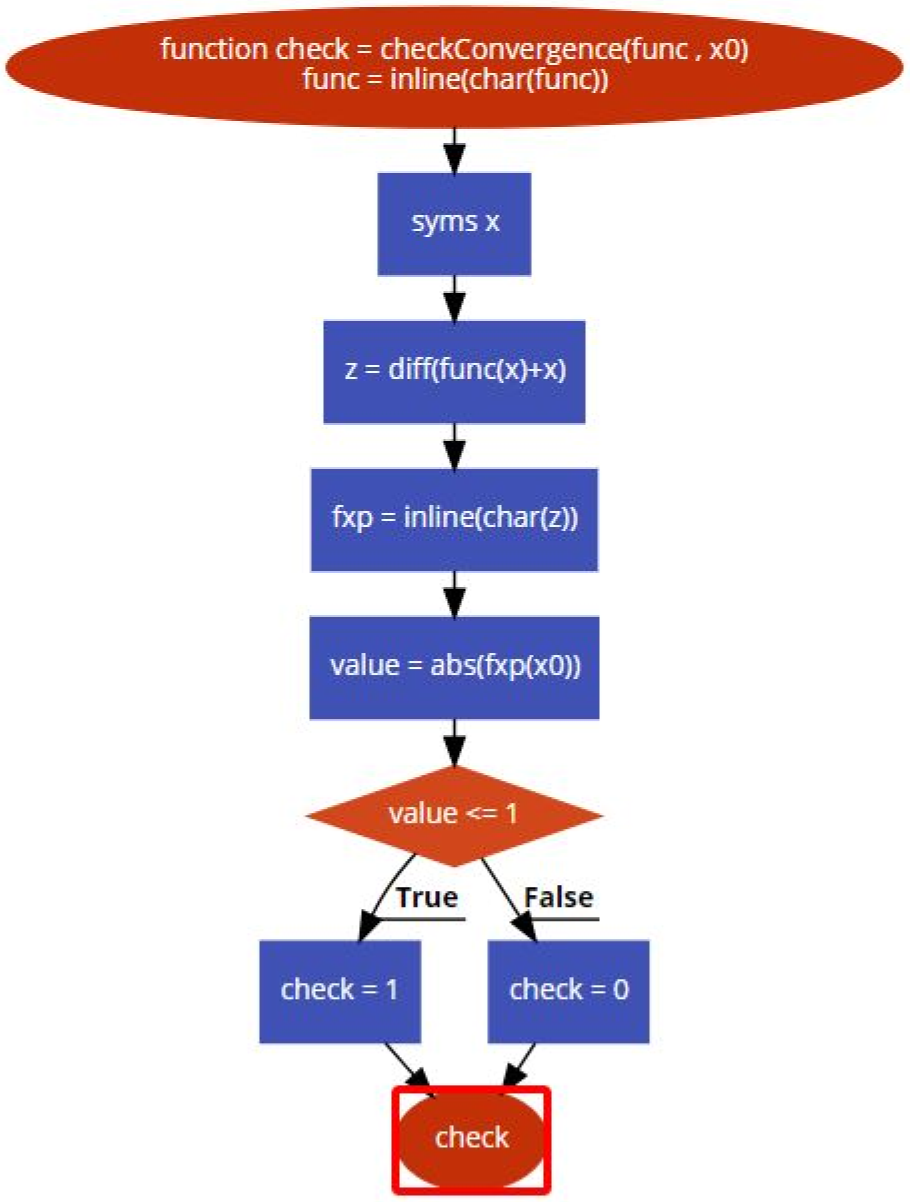
Max Error:

	xl	xu	xm	error
	0	1	0.5000	0
	0.5000	1	0.7500	0.2500
	0.5000	0.7500	0.6250	0.1250

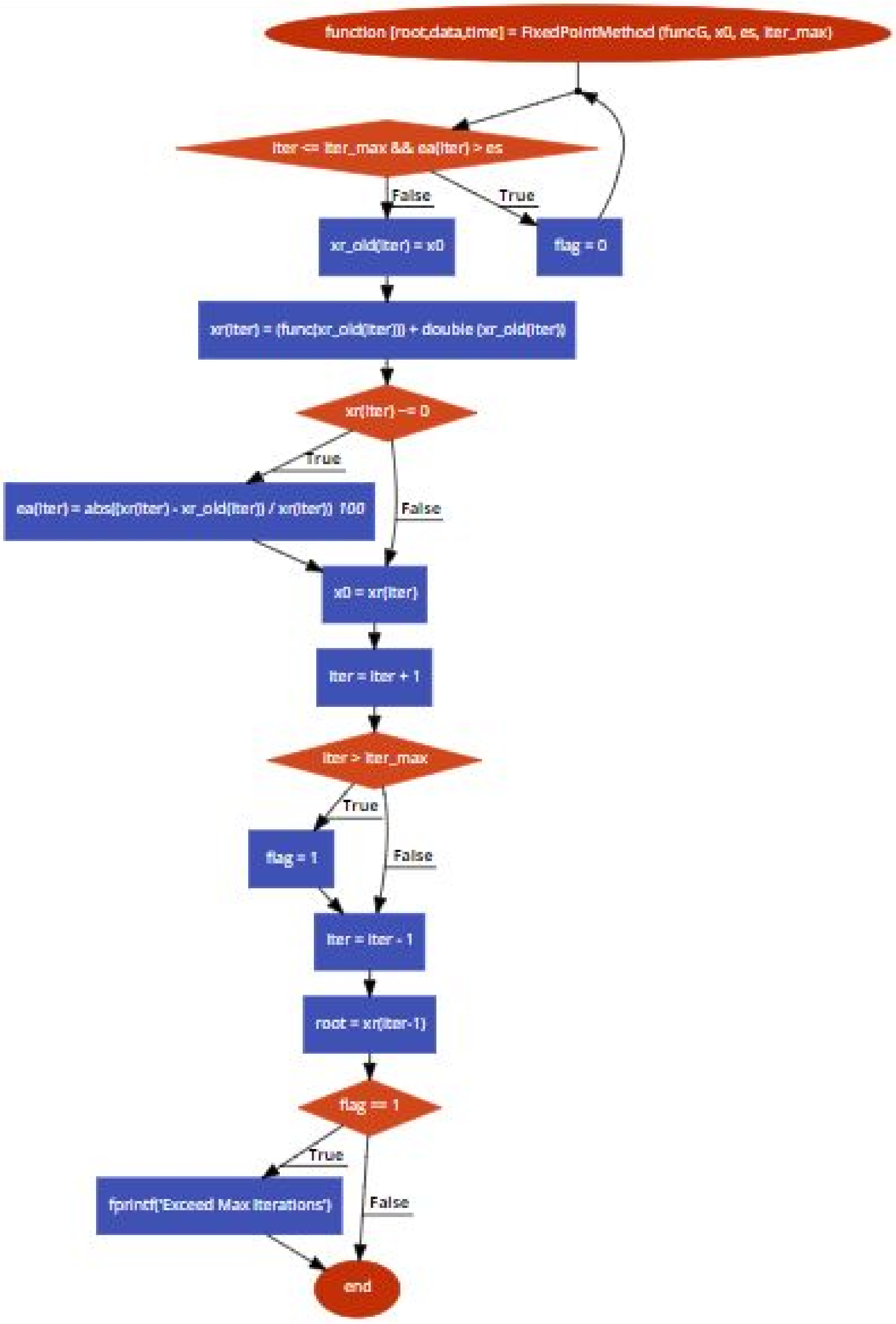
Fixed Point

1. Flow Chart

   Flow Chart for the check of convergence :

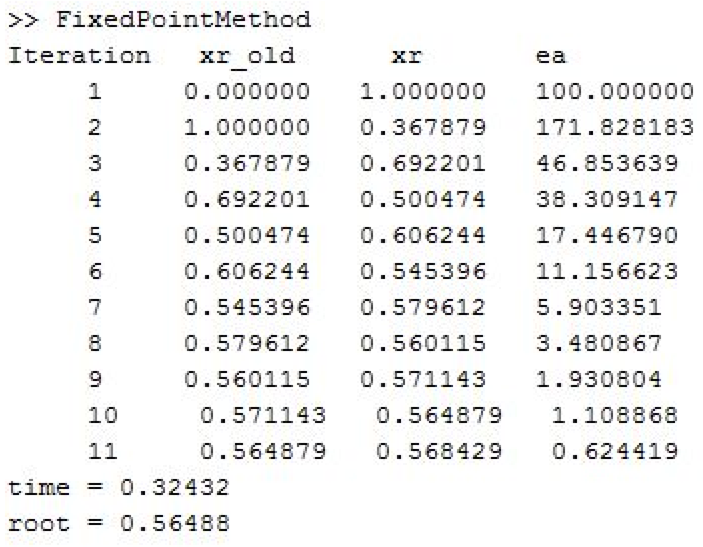


Flow Chart for the implementation of the method :


function [root,data,time] = FixedPointMethod (funcG, x0, es, iter_max)
iter <= iter_max && ea(iter) > es
False
True
xr_old(iter) = x0
flag = 0
xr(iter) = (func(xr_old(iter)) + double (xr_old(iter))
xr(iter) ~= 0
True
ea(iter) = abs((xr(iter) - xr_old(iter)) / xr(iter)) 100
False
x0 = xr(iter)
iter = iter + 1
iter > iter_max
True
flag = 1
False
iter = iter - 1
root = xr(iter-1)
flag == 1
True
fprintf('Exceed Max Iterations')
False
end

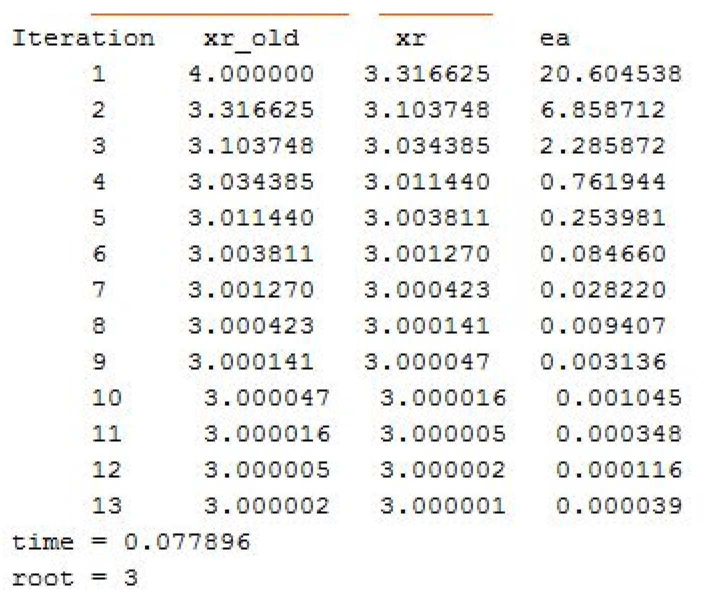
2. Built in data structures or functions used
    a. Inline : to convert the string function to a function.
    b. Diff : to get the derivative of the function.
    c. Abs : to get the absolute value to a value.
    d. Tic toc : to get the execution time for the method .
    e. Subs : to substitute in a function with a given value and get the result.
    f. Vectors : to store the data (current root , previous root and the approximate relative error).
3. Equations and Analysis
    a. exp(-x)-x = 0 with initial guess = 0 and eps = 1 :

Time = 0.32432



```
>> FixedPointMethod
Iteration   xr_old      xr        ea
     1     0.000000   1.000000   100.000000
     2     1.000000   0.367879   171.828183
     3     0.367879   0.692201   46.853639
     4     0.692201   0.500474   38.309147
     5     0.500474   0.606244   17.446790
     6     0.606244   0.545396   11.156623
     7     0.545396   0.579612   5.903351
     8     0.579612   0.560115   3.480867
     9     0.560115   0.571143   1.930804
     10     0.571143   0.564879   1.108868
     11     0.564879   0.568429   0.624419
time = 0.32432
root = 0.56488
```

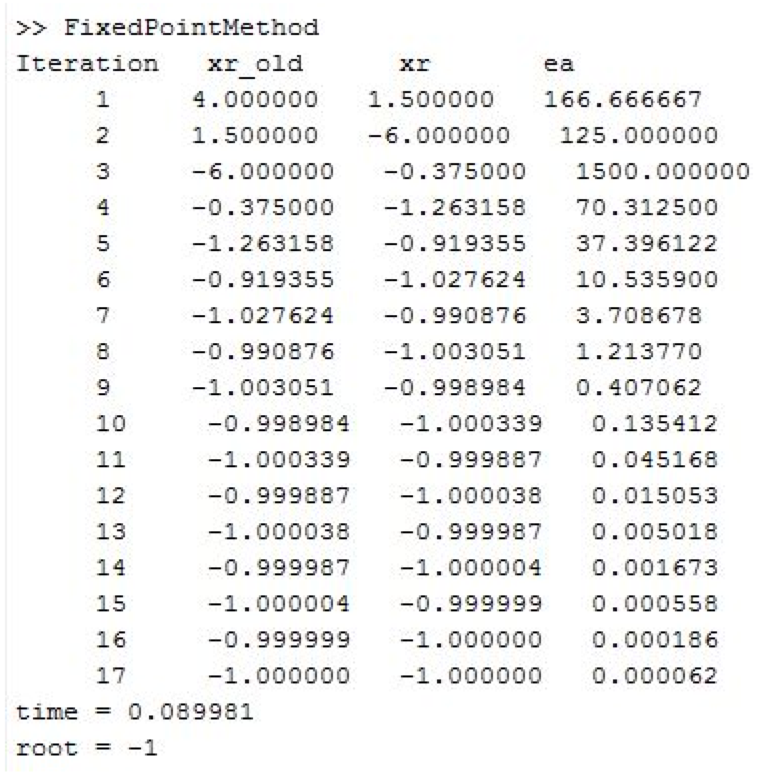
    b. sqrt(2*x + 3) - x = 0 with initial guess = 4 and eps = 0.0001 :
    Time = 0.077896



```
Iteration   xr_old      xr        ea
     1     4.000000   3.316625   20.604538
     2     3.316625   3.103748   6.858712
     3     3.103748   3.034385   2.285872
     4     3.034385   3.011440   0.761944
     5     3.011440   3.003811   0.253981
     6     3.003811   3.001270   0.084660
     7     3.001270   3.000423   0.028220
     8     3.000423   3.000141   0.009407
     9     3.000141   3.000047   0.003136
     10     3.000047   3.000016   0.001045
     11     3.000016   3.000005   0.000348
     12     3.000005   3.000002   0.000116
     13     3.000002   3.000001   0.000039
time = 0.077896
root = 3
```

c. (3/(x - 2)) - x = 0 with initial guess = 4 and eps = 0.0001

Time = 0.089981



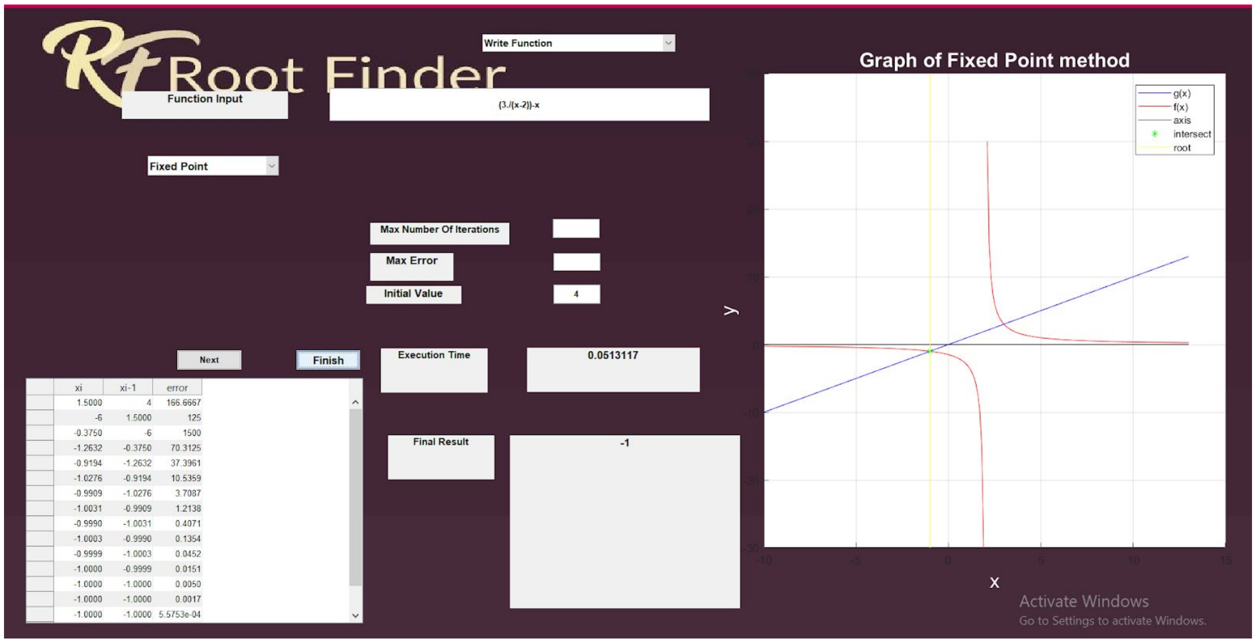
```
>> FixedPointMethod
Iteration   xr_old      xr        ea
     1     4.000000   1.500000   166.666667
     2     1.500000   -6.000000   125.000000
     3     -6.000000   -0.375000   1500.000000
     4     -0.375000   -1.263158   70.312500
     5     -1.263158   -0.919355   37.396122
     6     -0.919355   -1.027624   10.535900
     7     -1.027624   -0.990876   3.708678
     8     -0.990876   -1.003051   1.213770
     9     -1.003051   -0.998984   0.407062
     10     -0.998984   -1.000339   0.135412
     11     -1.000339   -0.999887   0.045168
     12     -0.999887   -1.000038   0.015053
     13     -1.000038   -0.999987   0.005018
     14     -0.999987   -1.000004   0.001673
     15     -1.000004   -0.999999   0.000558
     16     -0.999999   -1.000000   0.000186
     17     -1.000000   -1.000000   0.000062
time = 0.089981
root = -1
```

f. Problematic Functions

- ((x^2 - 3)/2) - x with initial guess x0 = 4 using the Fixed Point Method diverges because the absolute value of the derivative of the function at this point > 1 .

g. Sample Runs

- **In Fast Mode**



(3./(x-2))-x

Max Number Of Iterations:

Max Error:

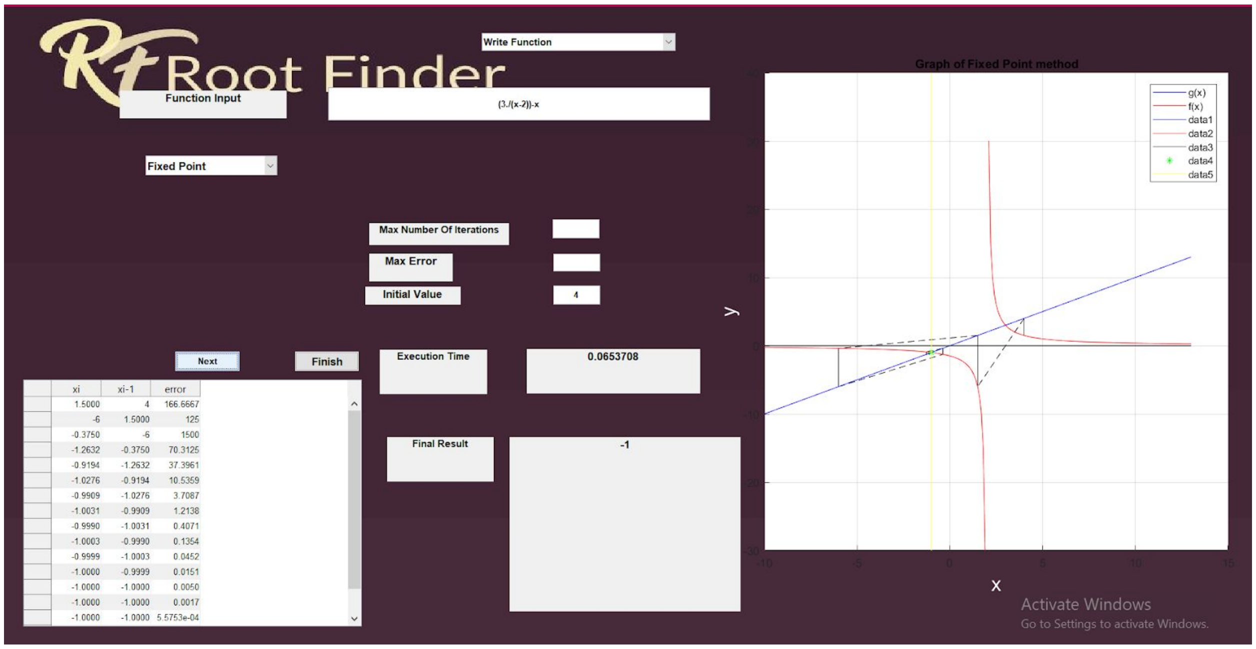
Initial Value: 4

Execution Time: 0.0513117

Final Result: -1

xi	xi-1	error
1.5000	4	166.6667
-6	1.5000	125
-0.3750	-6	1500
-1.2632	-0.3750	70.3125
-0.9194	-1.2632	37.3961
-1.0276	-0.9194	10.5359
-0.9909	-1.0276	3.7087
-1.0031	-0.9909	1.2138
-0.9990	-1.0031	0.4071
-1.0003	-0.9990	0.1354
-0.9999	-1.0003	0.0452
-1.0000	-0.9999	0.0151
-1.0000	-1.0000	0.0050
-1.0000	-1.0000	0.0017
-1.0000	-1.0000	5.5753e-04

- In Slow Mode



Function Input: (3./(x-2))-x

Max Number Of Iterations:

Max Error:

Initial Value: 4

Execution Time: 0.0653708

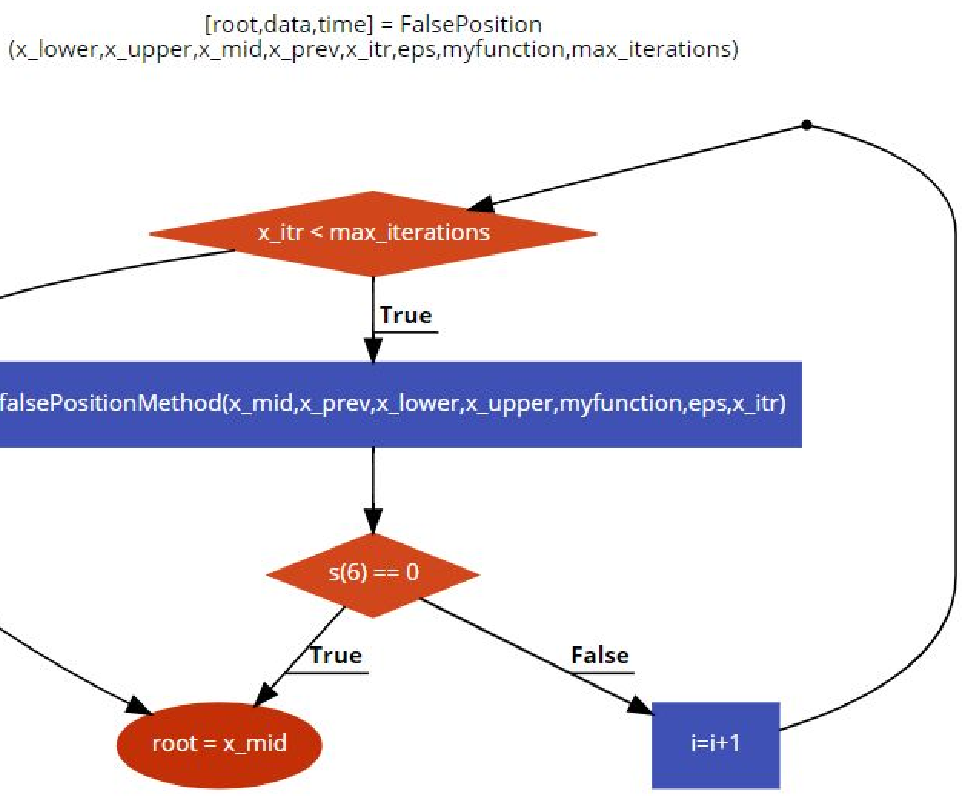
Final Result: -1

xi	xi-1	error
1.5000	4	166.6667
-6	1.5000	125
-0.3750	-6	1500
-1.2632	-0.3750	70.3125
-0.9194	-1.2632	37.3961
-1.0276	-0.9194	10.5359
-0.9909	-1.0276	3.7087
-1.0031	-0.9909	1.2138
-0.9990	-1.0031	0.4071
-1.0003	-0.9990	0.1354
-0.9999	-1.0003	0.0452
-1.0000	-0.9999	0.0151
-1.0000	-1.0000	0.0050
-1.0000	-1.0000	0.0017
-1.0000	-1.0000	5.5753e-04

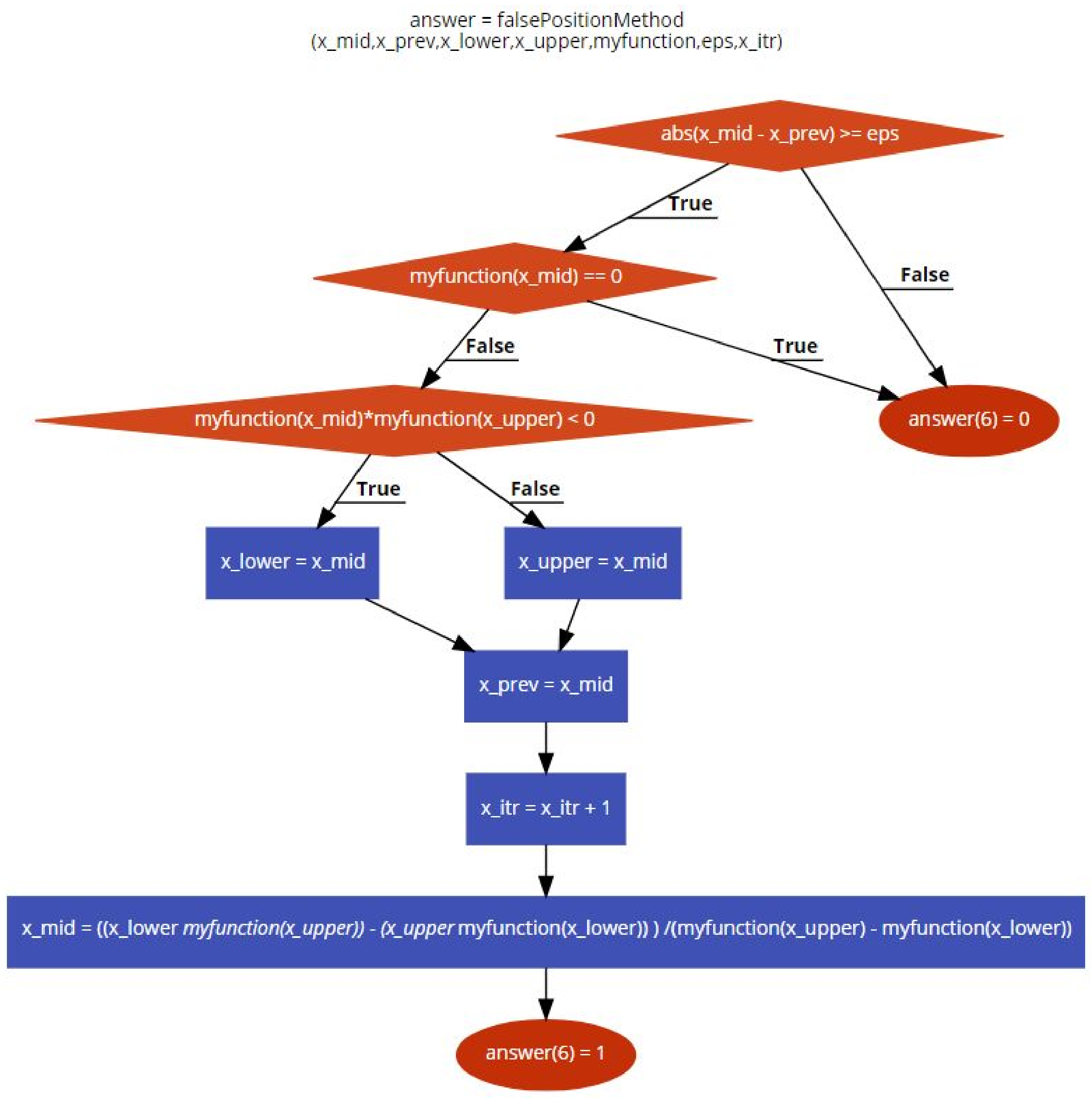
False Position

1. Flow Chart

Fast Bisection flowchart that uses slow bisection method


[root,data,time] = FalsePosition
(x_lower,x_upper,x_mid,x_prev,x_itr,eps,myfunction,max_iterations)
x_itr < max_iterations
True
s = falsePositionMethod(x_mid,x_prev,x_lower,x_upper,myfunction,eps,x_itr)
False
s(6) == 0
True
False
root = x_mid
i=i+1

Slow Bisection that get x_mid iteration by iteration


answer = falsePositionMethod (x_mid,x_prev,x_lower,x_upper,myfunction,eps,x_itr)
abs(x_mid - x_prev) >= eps
True
False
myfunction(x_mid) == 0
False
True
myfunction(x_mid)*myfunction(x_upper) < 0
answer(6) = 0
True
False
x_lower = x_mid
x_upper = x_mid
x_prev = x_mid
x_itr = x_itr + 1
x_mid = ((x_lower myfunction(x_upper)) - (x_upper myfunction(x_lower))) /(myfunction(x_upper) - myfunction(x_lower))
answer(6) = 1

h. Built in data structures or functions used

   i. Data Structures

      1. Vector to store data from slow bisection to be used in fast.

   ii. Built in functions

      1. Inline : to convert the string function to a character array.
      2. Tic toc : to get the execution time for the method.

i. Equations and Analysis

i. x^3-3*x+1, xlower = 1.5 , xUpper = 0.11

```
Lower 1.28181 upper 0.11 mid 0.667649 eps 0.614163
Lower 0.667649 upper 0.11 mid 0.381937 eps 0.285712
Lower 0.381937 upper 0.11 mid 0.34976 eps 0.0321766
Lower 0.34976 upper 0.11 mid 0.347463 eps 0.00229685
Lower 0.347463 upper 0.11 mid 0.347308 eps 0.000155631
Lower 0.347308 upper 0.11 mid 0.347297 eps 1.0506e-05
time =
    0.3387
```

j. Problematic Functions

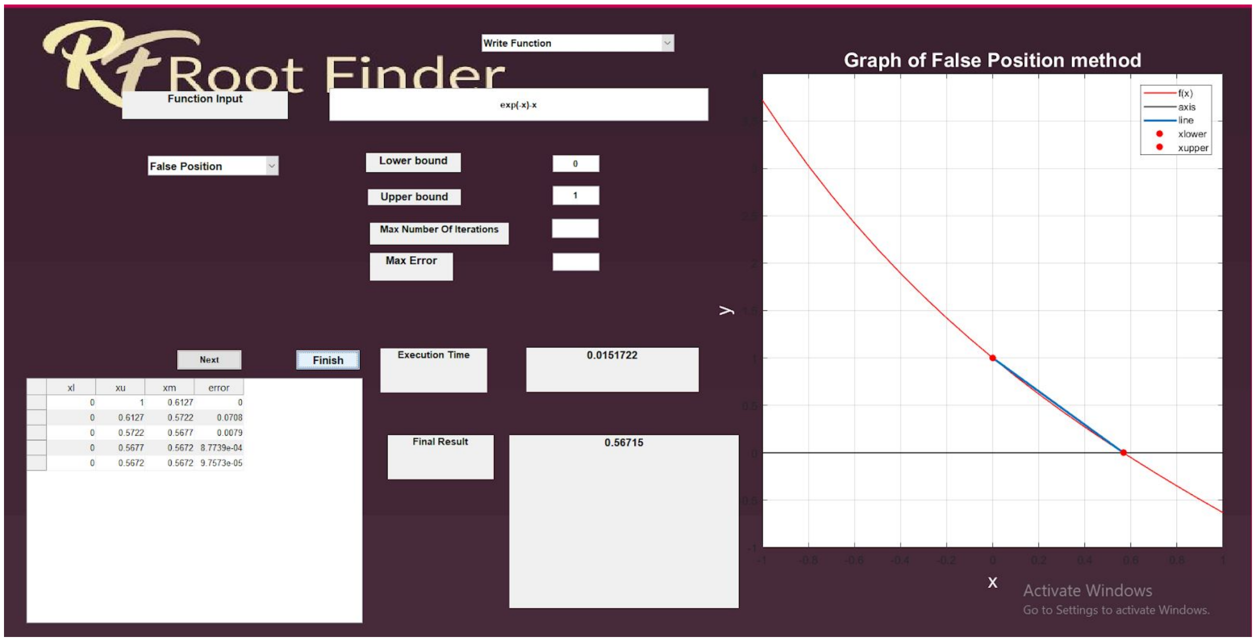
i. Function changes sign but root does not exist

1. Example : y = 1 / x

ii. Cannot handle multiple even roots as upper and lower bounds for this root will have the same sign

k. Sample Runs

- In Fast Mode



Function Input: exp(-x)-x

False Position

Lower bound: 0

Upper bound: 1

Max Number Of Iterations:

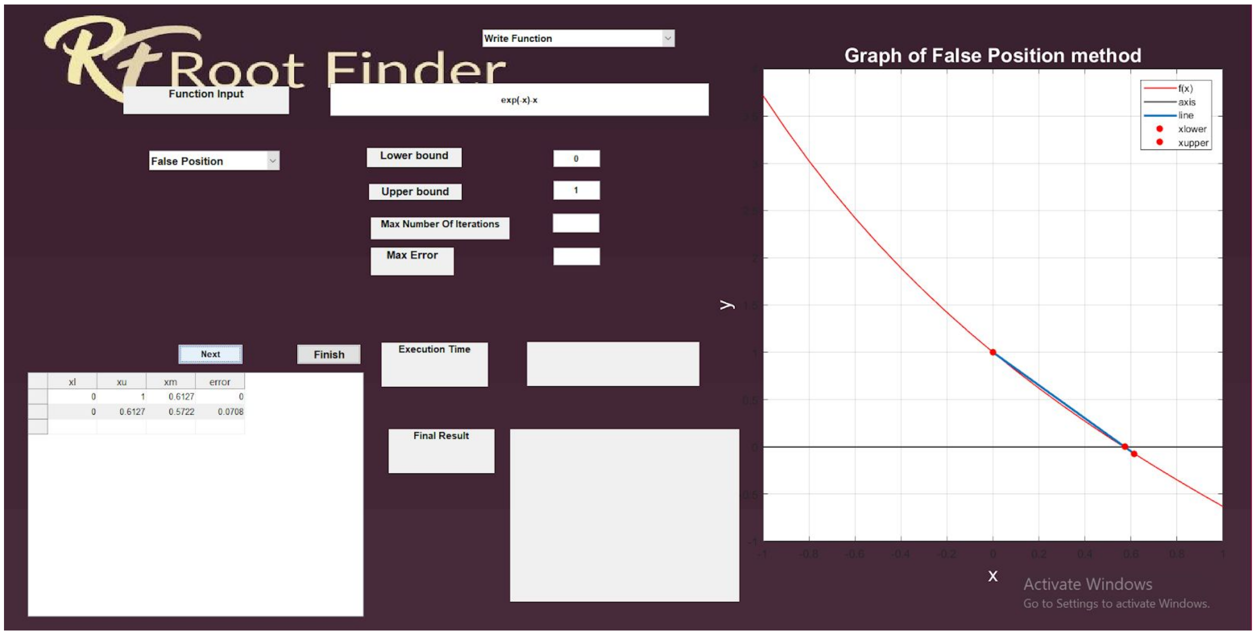
Max Error:

Execution Time: 0.0151722

Final Result: 0.56715

xl	xu	xm	error
0	1	0.6127	0
0	0.6127	0.5722	0.0708
0	0.5722	0.5677	0.0079
0	0.5677	0.5672	8.7739e-04
0	0.5672	0.5672	9.7573e-05

- In Slow Mode



Function Input: exp(-x)-x

False Position

Lower bound: 0

Upper bound: 1

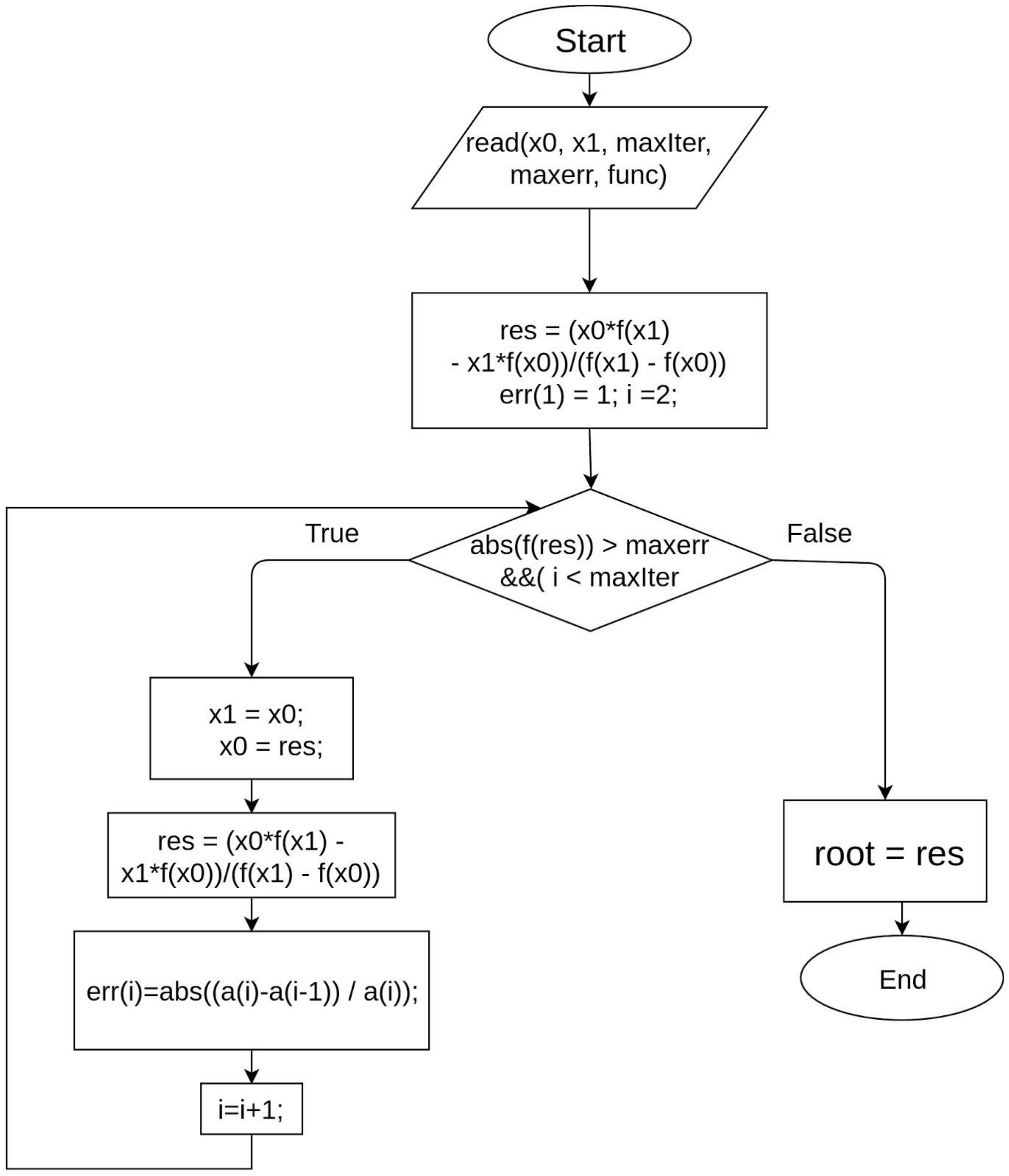
Max Number Of Iterations:

Max Error:

xl	xu	xm	error
0	1	0.6127	0
0	0.6127	0.5722	0.0708

Secant

a. Flow Chart


Start
read(x0, x1, maxIter, maxerr, func)
res = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0)) err(1) = 1; i =2;
abs(f(res)) > maxerr &&(i < maxIter
True
False
x1 = x0; x0 = res;
res = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0))
err(i)=abs((a(i)-a(i-1)) / a(i));
i=i+1;
root = res
End

b. Built in data structures or functions used

   i. Data Structures:

      a matrix (2d array) to store the value of x for each iteration

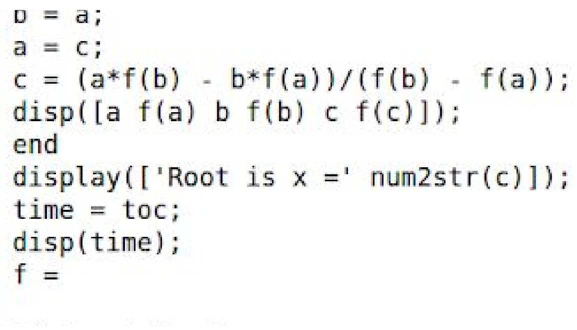
   ii. Functions:

      abs() to calculate the absolute value of the error

c. Equations and Analysis

   i. $f(x) = x^2 - 2$

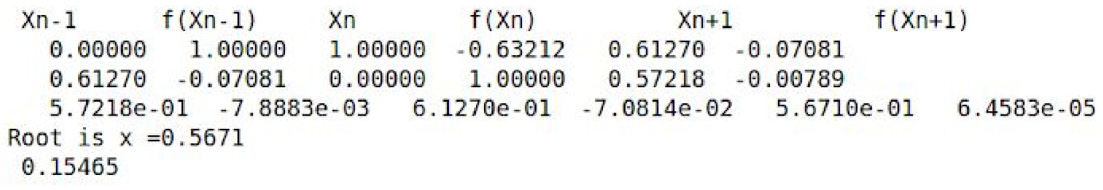
      Execution time = 0.10269 s



```
b = a;
a = c;
c = (a*f(b) - b*f(a))/(f(b) - f(a));
disp([a f(a) b f(b) c f(c)]);
end
display(['Root is x =' num2str(c)]);
time = toc;
disp(time);
f =
```

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

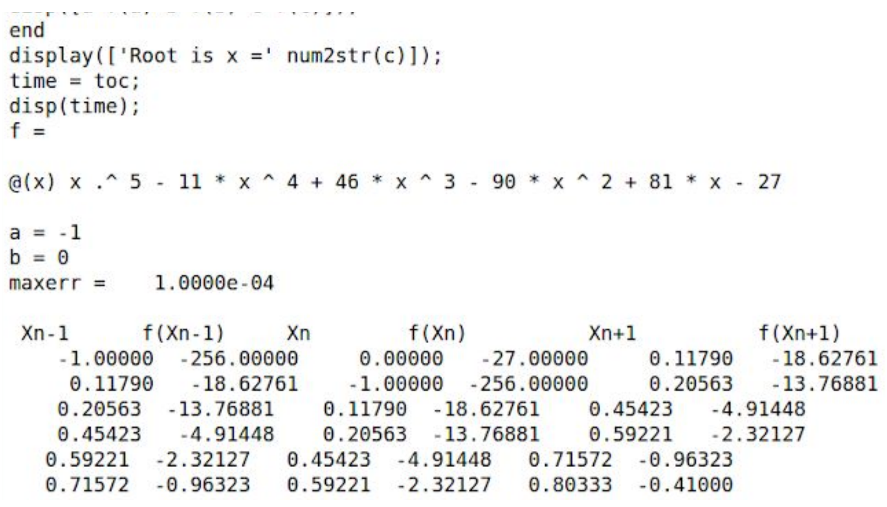
      Execution time = 0.15465 s



```
 Xn-1      f(Xn-1)     Xn        f(Xn)         Xn+1          f(Xn+1)
   0.00000   1.00000   1.00000  -0.63212   0.61270  -0.07081
   0.61270  -0.07081   0.00000   1.00000   0.57218  -0.00789
   5.7218e-01  -7.8883e-03   6.1270e-01  -7.0814e-02   5.6710e-01   6.4583e-05
Root is x =0.5671
 0.15465
```

a. f(x) = $x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27$

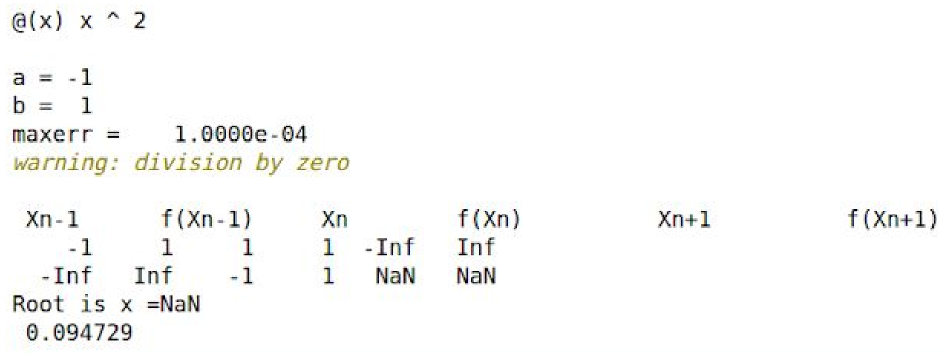
Execution time = 0.11762 s



```
end
display(['Root is x =' num2str(c)]);
time = toc;
disp(time);
f =

@(x) x .^ 5 - 11 * x ^ 4 + 46 * x ^ 3 - 90 * x ^ 2 + 81 * x - 27

a = -1
b = 0
maxerr =    1.0000e-04

 Xn-1      f(Xn-1)     Xn        f(Xn)          Xn+1          f(Xn+1)
    -1.00000  -256.00000     0.00000   -27.00000     0.11790   -18.62761
     0.11790   -18.62761    -1.00000  -256.00000     0.20563   -13.76881
    0.20563  -13.76881    0.11790  -18.62761    0.45423   -4.91448
    0.45423   -4.91448    0.20563  -13.76881    0.59221   -2.32127
   0.59221  -2.32127   0.45423  -4.91448   0.71572  -0.96323
   0.71572  -0.96323   0.59221  -2.32127   0.80333  -0.41000
```

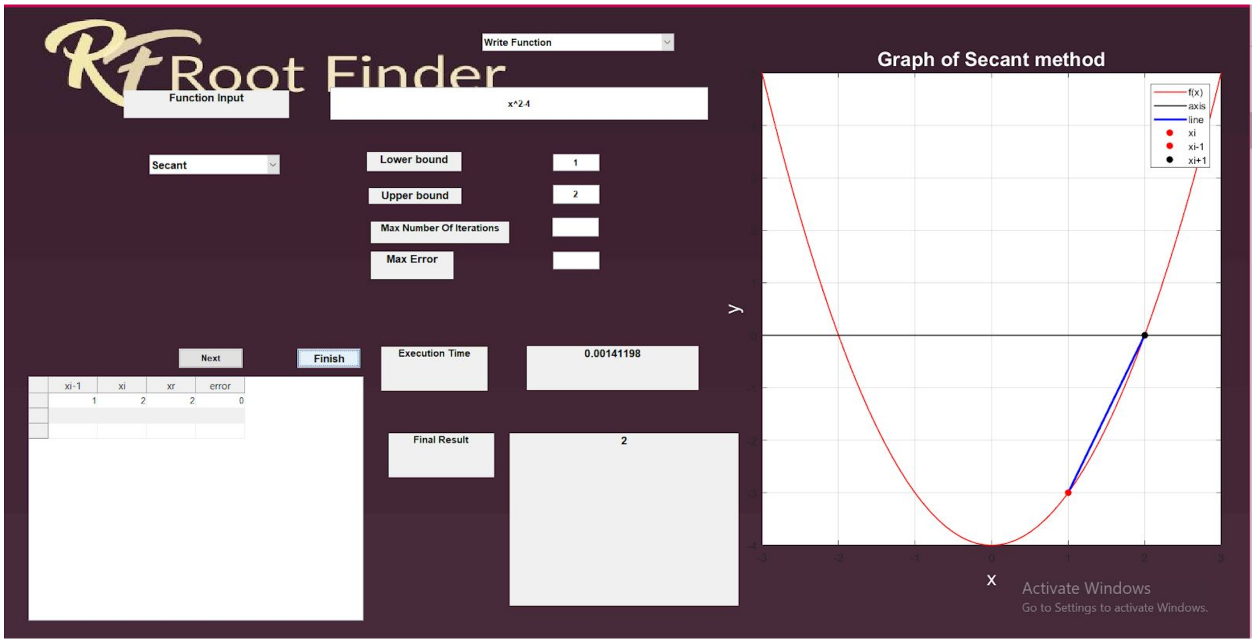
2. Problematic Functions
3. Division by zero problem occurs when slope = 0

   Example : f = x^2



```
@(x) x ^ 2

a = -1
b =  1
maxerr =    1.0000e-04
warning: division by zero

 Xn-1      f(Xn-1)      Xn        f(Xn)          Xn+1          f(Xn+1)
    -1       1      1      1  -Inf   Inf
  -Inf    Inf     -1      1   NaN   NaN
Root is x =NaN
 0.094729
```

4. Sample Runs
5. In Fast Mode



Function Input: x^2-4

Secant

Lower bound: 1

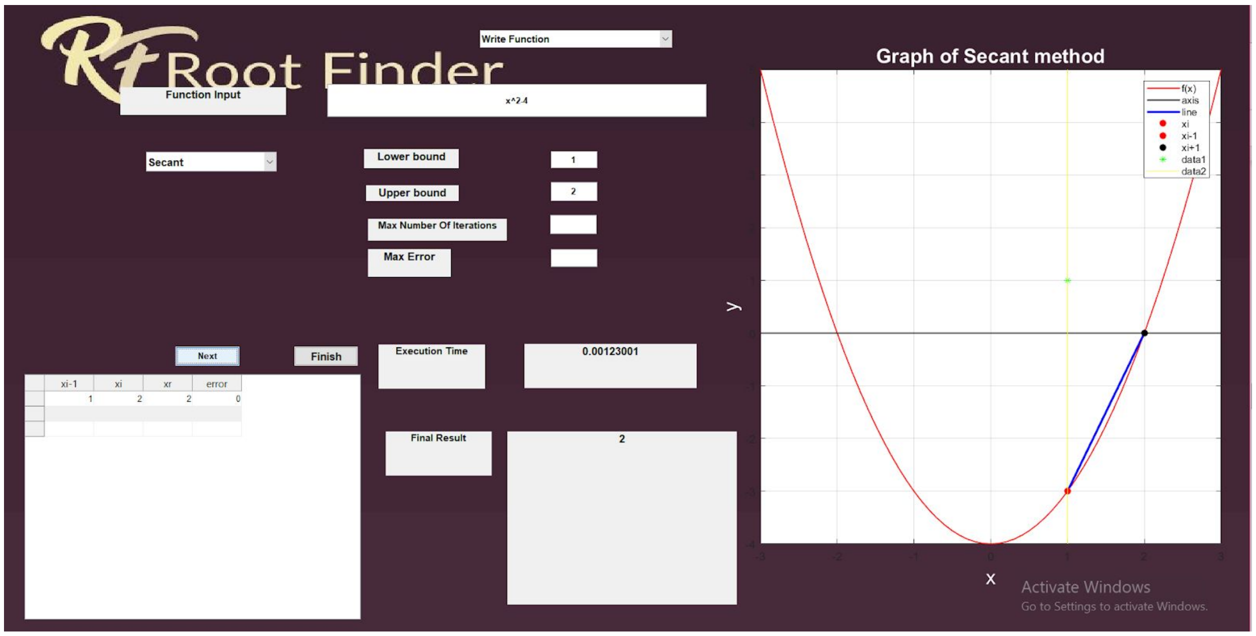
Upper bound: 2

Execution Time: 0.00141198

Final Result: 2

xi-1	xi	xr	error
1	2	2	0

- In Slow Mode



Function Input: x^2-4

Secant

Lower bound: 1

Upper bound: 2

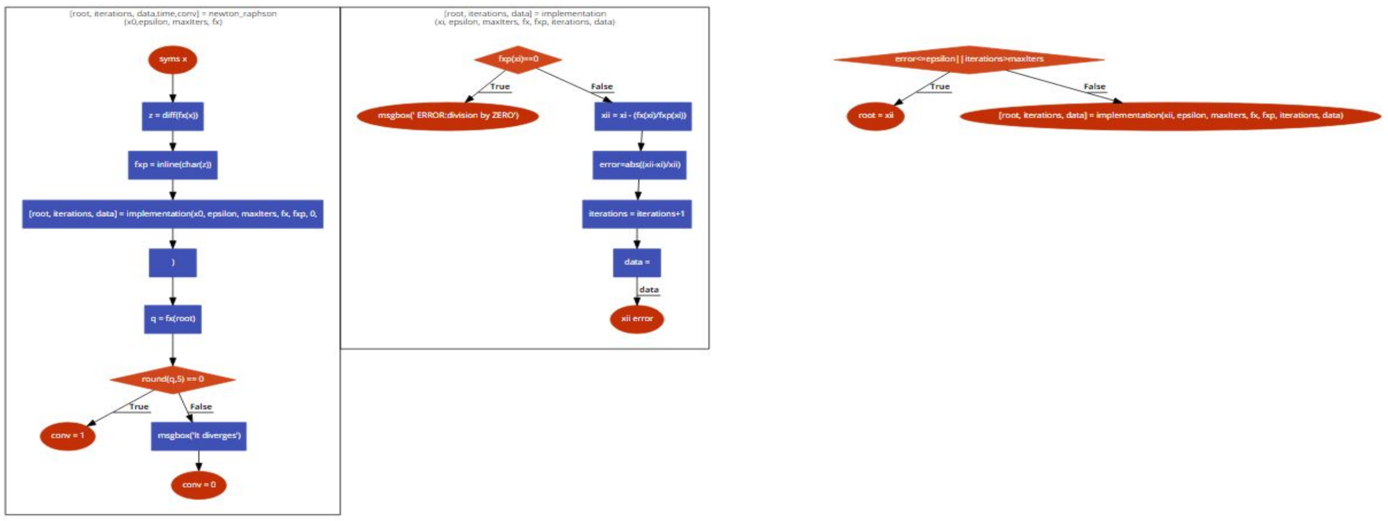
Execution Time: 0.00123001

Final Result: 2

xi-1	xi	xr	error
1	2	2	0

Newton-Raphson

l. Flow Chart


[root, iterations, data,time,conv] = newton_raphson (x0,epsilon, maxIters, fx)
syms x
z = diff(fx(x))
fxp = inline(char(z))
[root, iterations, data] = implementation(x0, epsilon, maxIters, fx, fxp, 0,
)
q = fx(root)
round(q,5) == 0
True
False
conv = 1
msgbox('It diverges')
conv = 0
[root, iterations, data] = implementation (xi, epsilon, maxIters, fx, fxp, iterations, data)
fxp(xi)==0
True
False
msgbox(' ERROR:division by ZERO')
xii = xi - (fx(xi)/fxp(xi))
error=abs((xii-xi)/xii)
iterations = iterations+1
data =
data
xii error
error<=epsilon||iterations>maxIters
True
False
root = xii
[root, iterations, data] = implementation(xii, epsilon, maxIters, fx, fxp, iterations, data)

m. Built in data structures or functions used

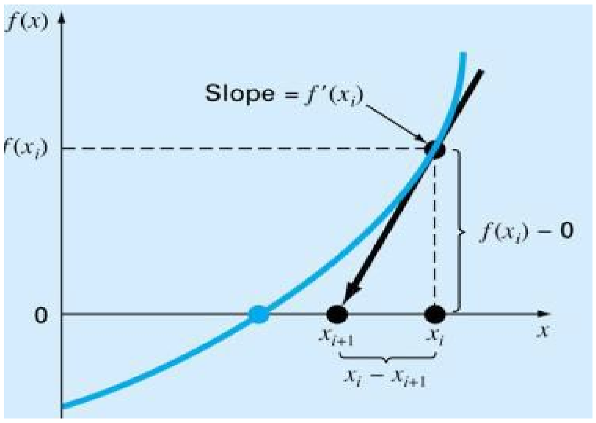
- Data Structures : array (2D) to store xi values and errors.

- Functions:

- abs() to calculate the absolute value of the error
- diff() to calculate the differentiation of a function
- inline() to convert from string to function

n. Equations and Analysis

1) It make use of the slope of the function to predict the location of the root, as shown below:



2) Evaluate the derivative of the function.

3) Use an initial guess of the root to estimate the new value of the root as

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

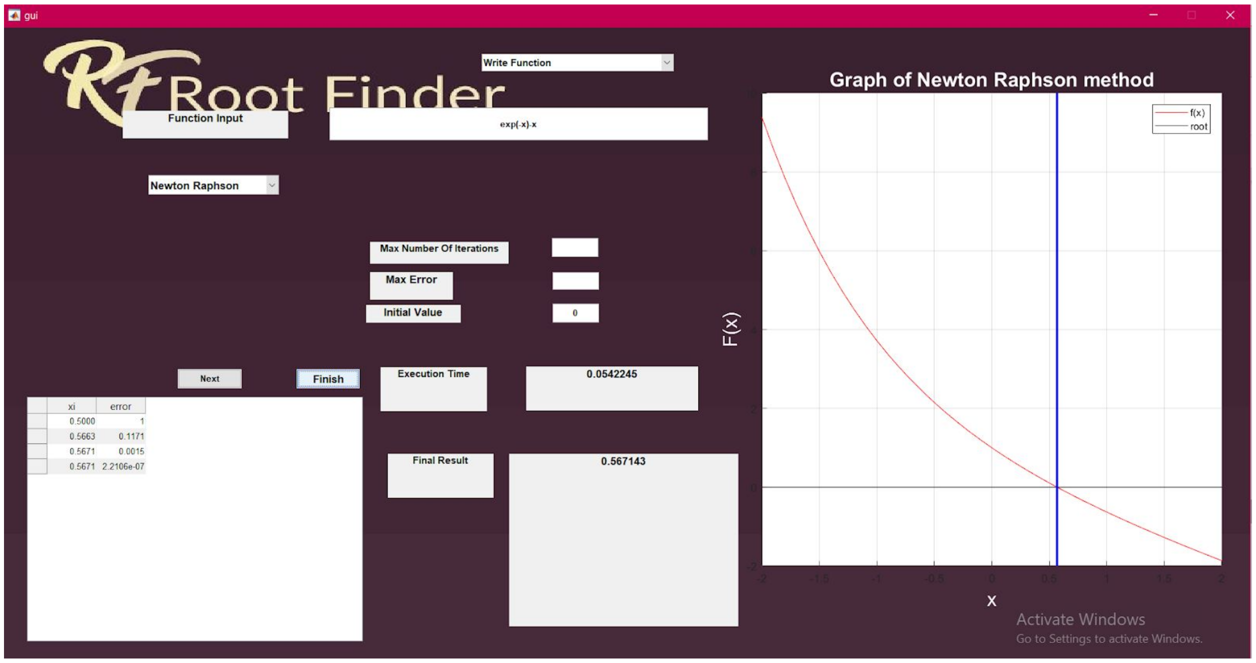
4) Iterate until the absolute relative error is acceptable or the number of iterations has exceeded the maximum number of iterations allowed.

o. Problematic Functions

- X^2 - 4 = 0 at initial point = 0 using NEWTON-RAPHSON Method , As it is division by zero

p. Sample Runs

- In Fast Mode:



Function Input: exp(-x)-x

Newton Raphson

Max Number Of Iterations:

Max Error:

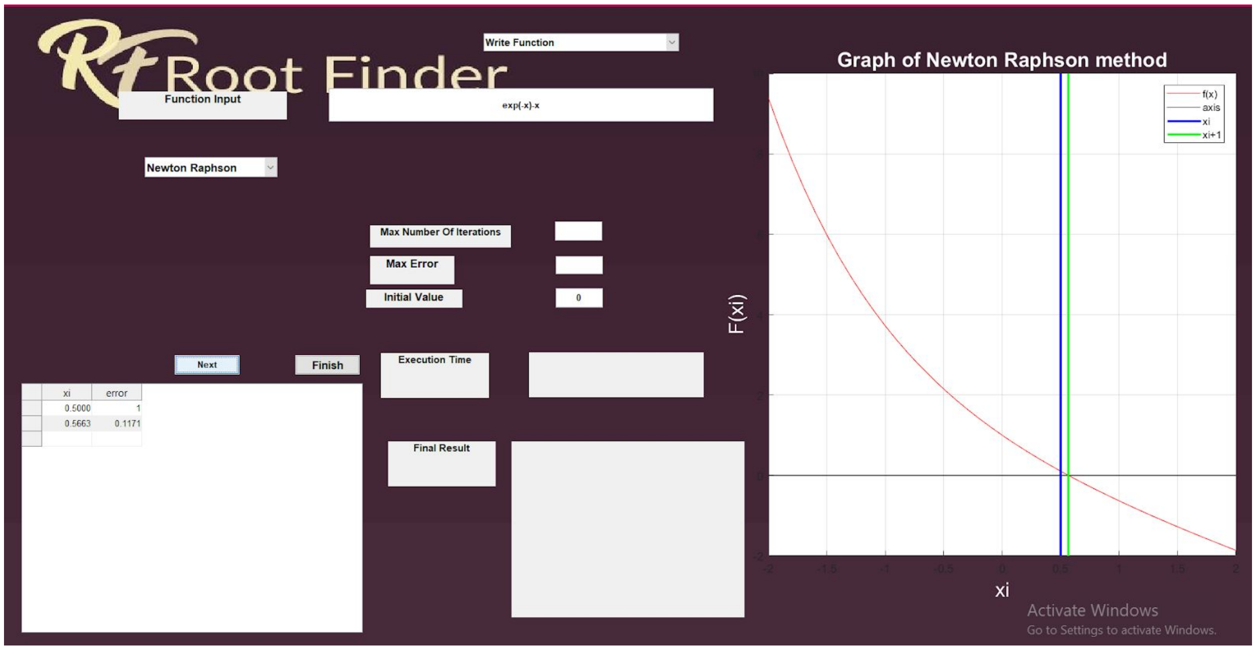
Initial Value: 0

Execution Time: 0.0542245

Final Result: 0.567143

	xi	error
	0.5000	1
	0.5663	0.1171
	0.5671	0.0015
	0.5671	2.2106e-07

- In Slow mode



Function Input: exp(-x)-x

Newton Raphson

Max Number Of Iterations:

Max Error:

Initial Value: 0

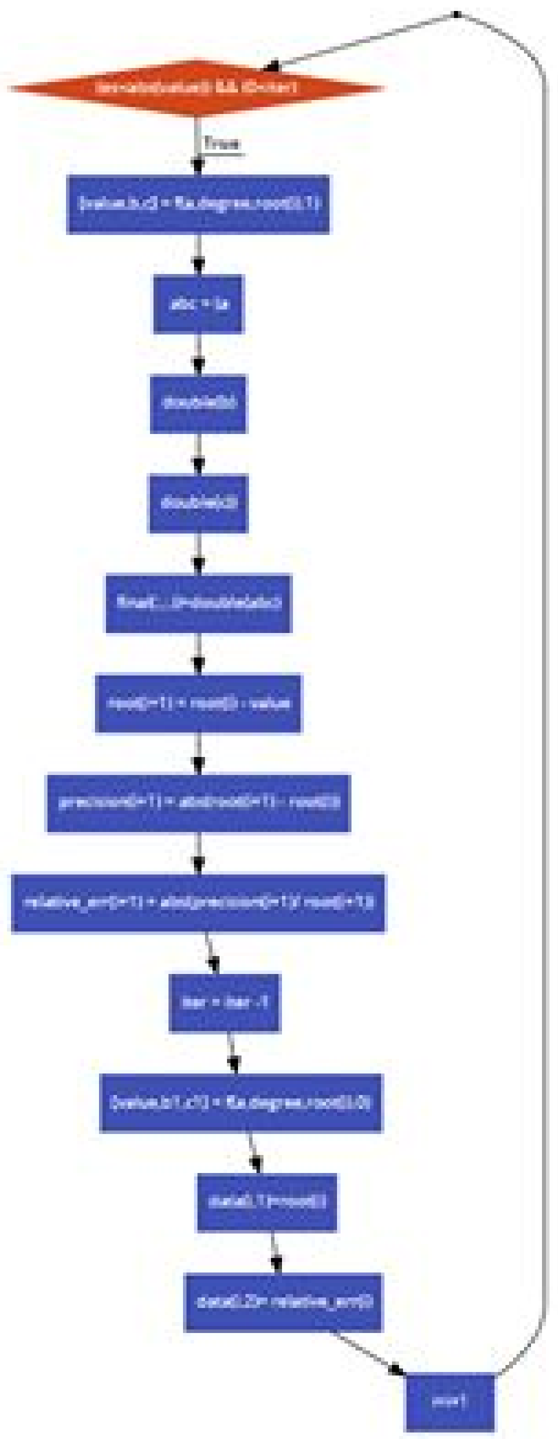
Execution Time:

Final Result:

	xi	error
	0.5000	1
	0.5663	0.1171

Birge-Vieta

q. Flow Chart


True
[value,b,c] = ffa,degree,root(i),1)
double(b)
double(c)
final(:,:,i)=double(abc)
root(i+1) = root(i) - value
precision(i+1) = abs(root(i+1) - root(i))
relative_err(i+1) = abs(precision(i+1)/ root(i+1))
iter = iter -1
[value,b1,c1] = ffa,degree,root(i),0)
data(i,1)=root(i)
data(i,2)= relative_err(i)
i=i+1

r. Built in data structures or functions used

Used Data structures:

- 1d arrays to store a,b,c and root for each iteration.
- 3d matrix.
- 2d matrix.

Used functions:

- Birge_Vieta function:

Its parameters are initial value,max iterations,max error and function and returns 3d matrix (final) that contains the tables of iterations and returns also 2d matrix(data) that contains xi of all iterations and achieved relative error in each iteration.

- evalin(symengine, fun) to convert string to symbolic function.
- coeffs() to get the coefficient of the symbolic function.

s. Equations and Analysis

Newton Raphson method with f(x) and f(x) evaluated using Horner's method.

Used Equations :

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_mx^m = (x-r)\left(b_1 + b_2x + \dots + b_mx^{m-1}\right) + b_0 = (x-r)h(x) + b_0$$

$$b_m = a_m$$
$$b_j = a_j + rb_{j+1} \qquad j = m-1, m-2, \dots, 1, 0$$

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

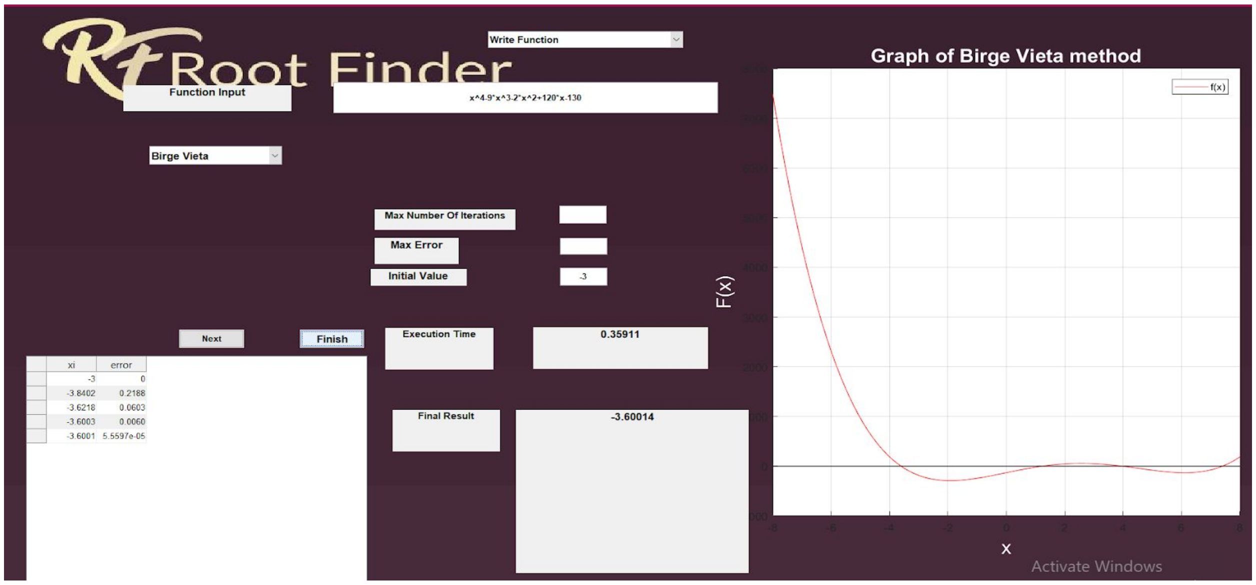
$$h(x) = b_1 + b_2x + \dots + b_mx^{m-1} = (x-r)\left(c_2 + c_3x + \dots + c_mx^{m-2}\right) + c_1$$

$$c_m = b_m$$
$$c_j = b_j + rc_{j+1} \qquad j = m-1, m-2, \dots, 1$$

$$f(r) = b_0$$
$$f'(r) = h(r) = c_1$$

t. Sample Runs

- In Fast Mode



Function Input: x^4-9*x^3-2*x^2+120*x-130

Birge Vieta

Max Number Of Iterations:

Max Error:

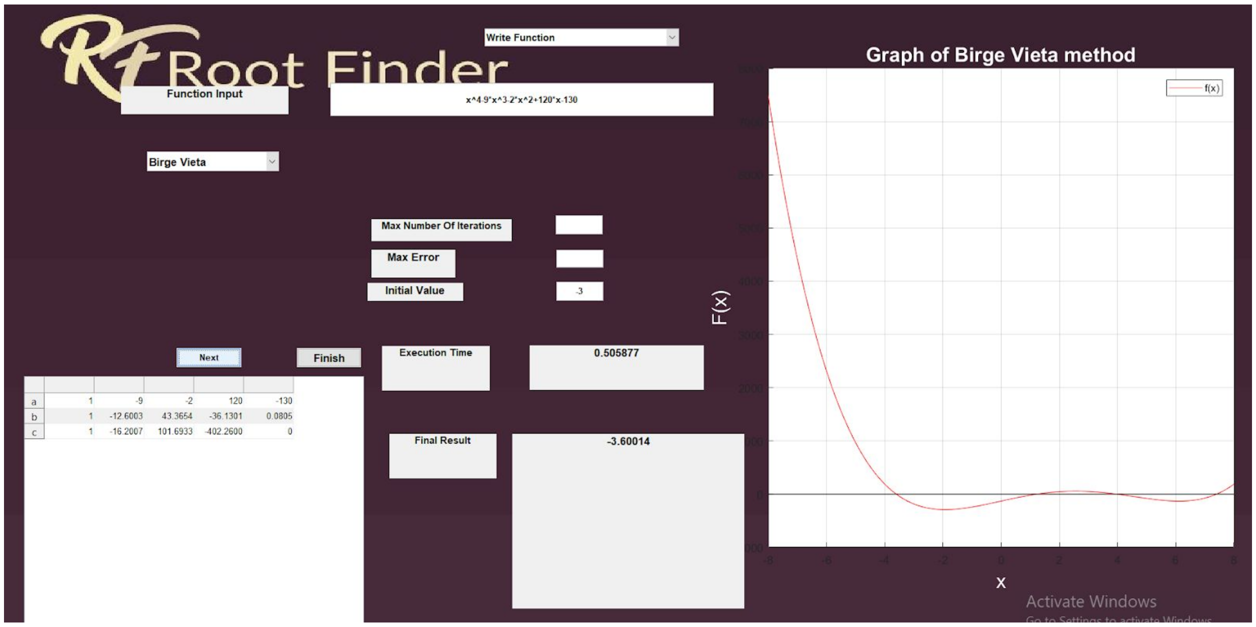
Initial Value: -3

Execution Time: 0.35911

Final Result: -3.60014

xi	error
-3	0
-3.8402	0.2188
-3.6218	0.0603
-3.6003	0.0060
-3.6001	5.5597e-05

- In Slow Mode



Function Input: x^4-9*x^3-2*x^2+120*x-130

Birge Vieta

Max Number Of Iterations:

Max Error:

Initial Value: -3

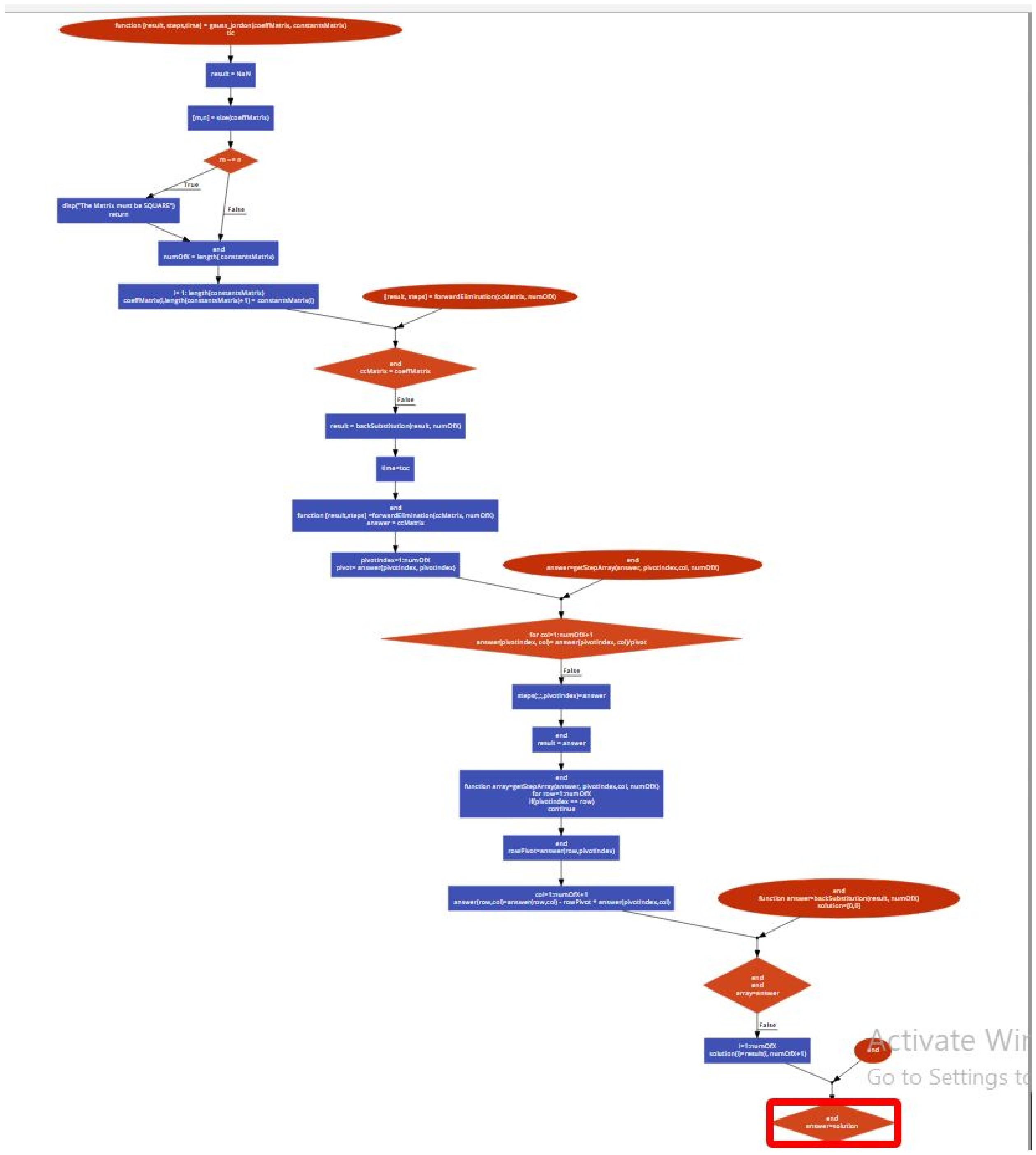
Execution Time: 0.505877

Final Result: -3.60014

a	1	-9	-2	120	-130
b	1	-12.6003	43.3654	-36.1301	0.0805
c	1	-16.2007	101.6933	-402.2600	0

Gauss-Jordan

a. Flow Chart



function [result, steps,time] = gauss_jordon(coeffMatrix, constantsMatrix)
tic
result = NaN
[m,n] = size(coeffMatrix)
m ~= n
True
disp("The Matrix must be SQUARE")
return
False
end
numOfX = length(constantsMatrix)
i= 1: length(constantsMatrix)
coeffMatrix(i,length(constantsMatrix)+1) = constantsMatrix(i)
[result, steps] = forwardElimination(ccMatrix, numOfX)
end
ccMatrix = coeffMatrix
False
result = backSubstitution(result, numOfX)
time=toc
end
function [result,steps] =forwardElimination(ccMatrix, numOfX)
answer = ccMatrix
pivotIndex=1:numOfX
pivot= answer(pivotIndex, pivotIndex)
end
answer=getStepArray(answer, pivotIndex,col, numOfX)
for col=1:numOfX+1
answer(pivotIndex, col)= answer(pivotIndex, col)/pivot
False
steps(:,:,pivotIndex)=answer
end
result = answer
end
function array=getStepArray(answer, pivotIndex,col, numOfX)
for row=1:numOfX
if(pivotIndex == row)
continue
end
rowPivot=answer(row,pivotIndex)
col=1:numOfX+1
answer(row,col)=answer(row,col) - rowPivot * answer(pivotIndex,col)
end
function answer=backSubstitution(result, numOfX)
solution=[0,0]
end
end
array=answer
False
i=1:numOfX
solution(i)=result(i, numOfX+1)
end
end
answer=solution
Activate Wi
Go to Settings t

b.Built in data structures or functions used

i. Data Structures:

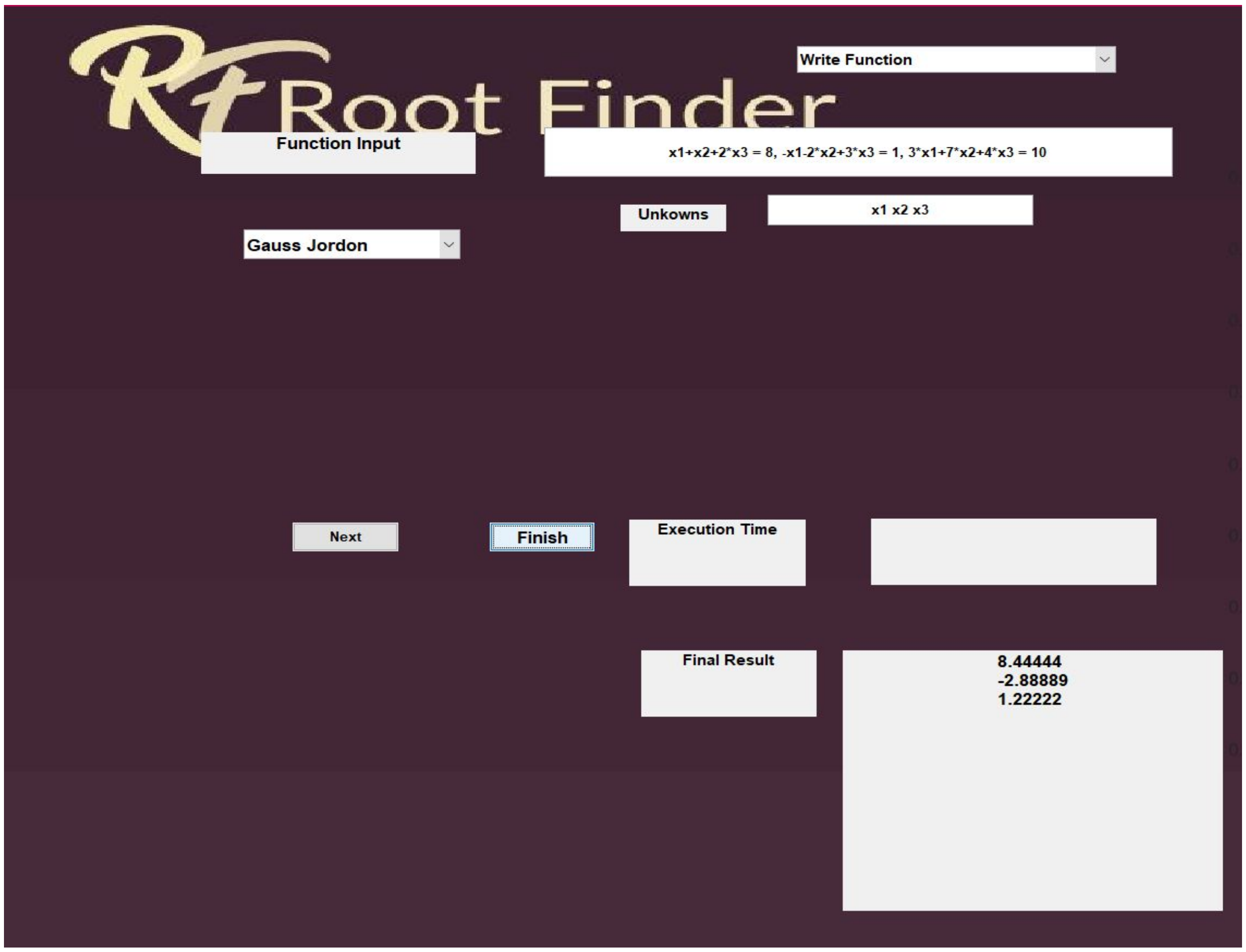
a matrix (2d array) to store the value of coefficients

c.Equations and Analysis

a) Elimination is applied to all equations(excluding the pivot equation) instead of just the subsequent equations.
b) All rows are normalized by dividing them by their pivot elements.
c) No back substitution is required.

d.Sample Runs

- In Fast mode



Function Input: x1+x2+2*x3 = 8, -x1-2*x2+3*x3 = 1, 3*x1+7*x2+4*x3 = 10

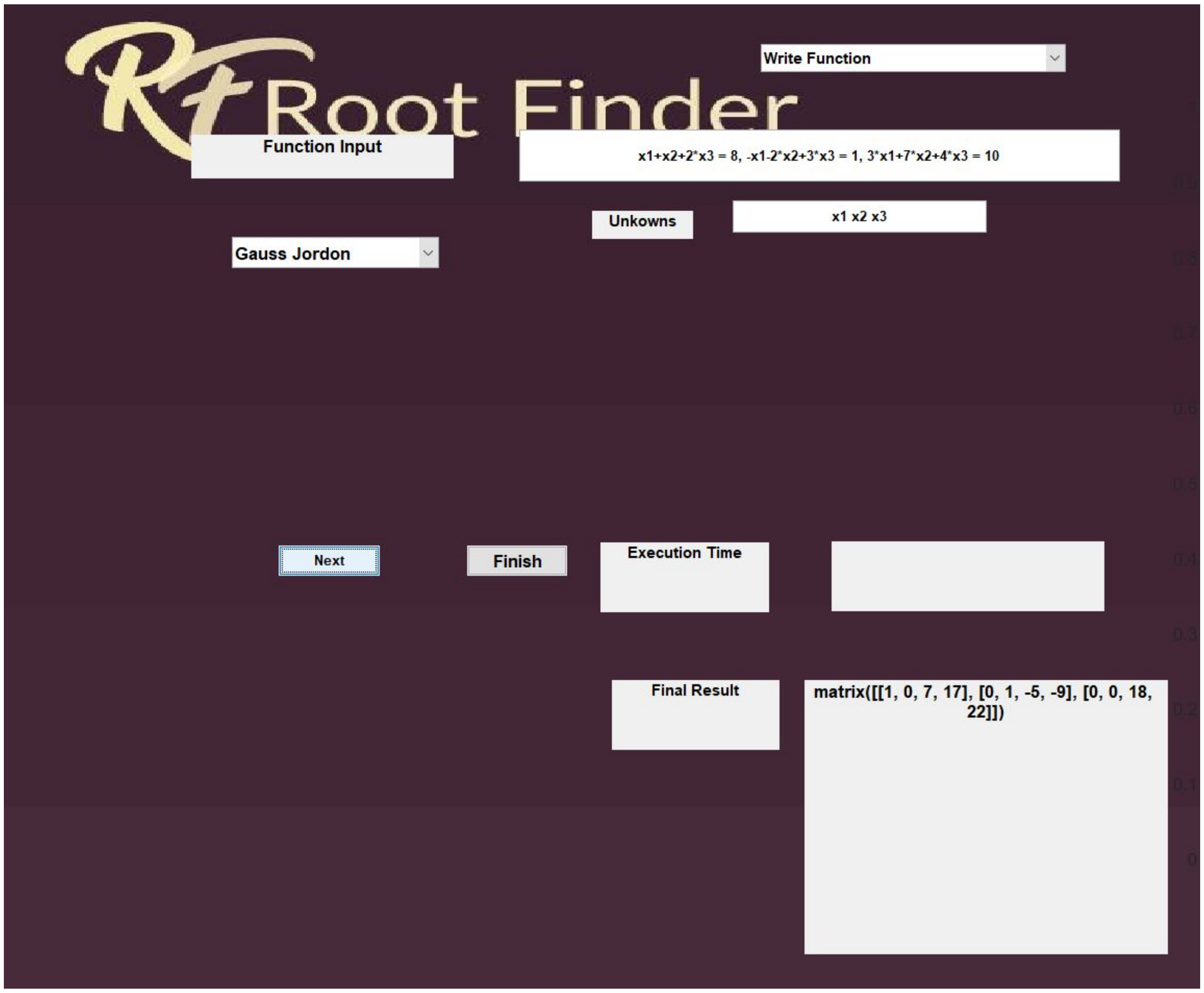
Unkowns: x1 x2 x3

Gauss Jordon

Execution Time

Final Result:
```
8.44444
-2.88889
1.22222
```

- In Slow Mode



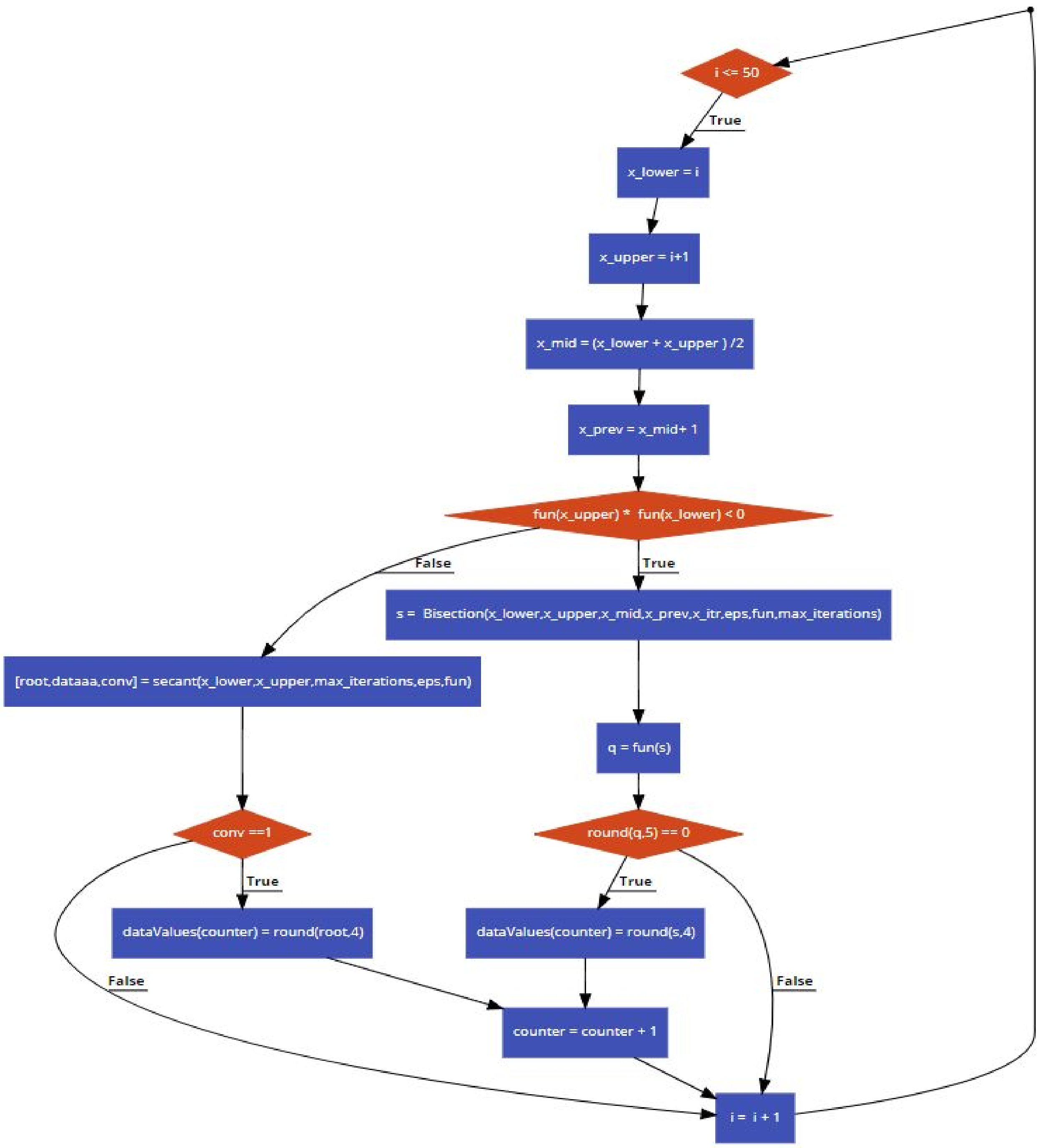
Gauss Jordon

x1+x2+2*x3 = 8, -x1-2*x2+3*x3 = 1, 3*x1+7*x2+4*x3 = 10

x1 x2 x3

matrix([[1, 0, 7, 17], [0, 1, -5, -9], [0, 0, 18, 22]])

General Method



i <= 50
True
x_lower = i
x_upper = i+1
x_mid = (x_lower + x_upper) /2
x_prev = x_mid+ 1
fun(x_upper) * fun(x_lower) < 0
False
True
s = Bisection(x_lower,x_upper,x_mid,x_prev,x_itr,eps,fun,max_iterations)
[root,dataaa,conv] = secant(x_lower,x_upper,max_iterations,eps,fun)
q = fun(s)
conv ==1
round(q,5) == 0
True
True
dataValues(counter) = round(root,4)
dataValues(counter) = round(s,4)
False
False
counter = counter + 1
i = i + 1

u. Flow Chart

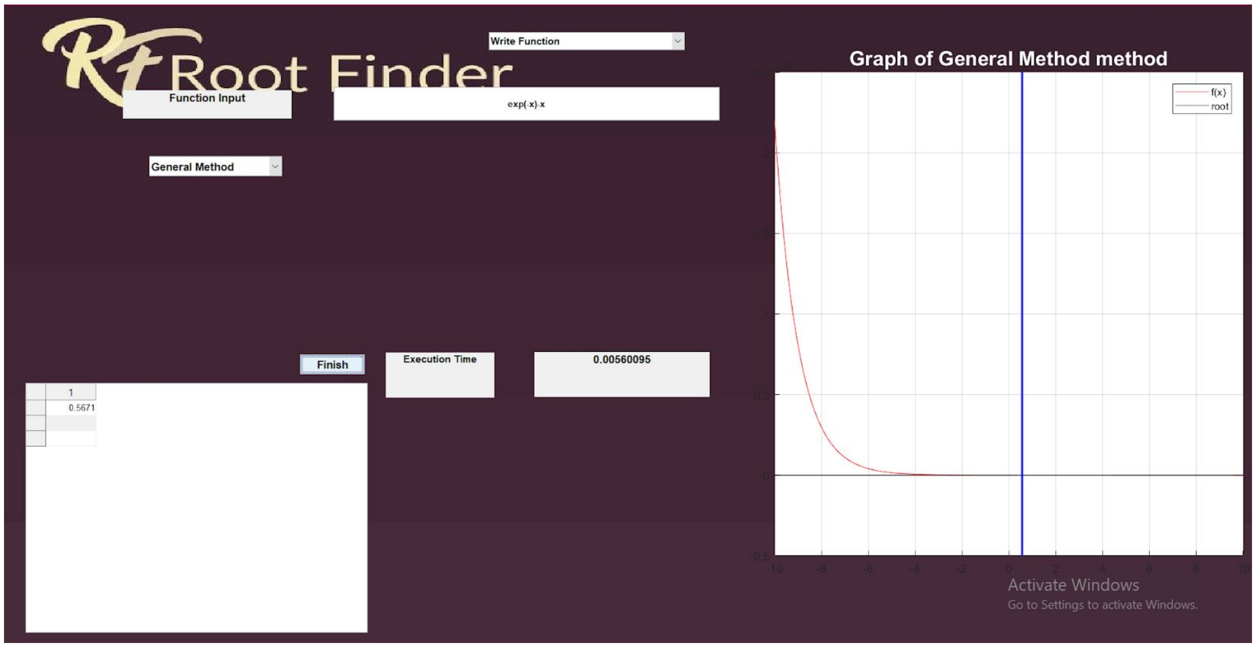
v. Built in data structures or functions used

i. Built in functions

1. Inline : to convert the string function to a function.
2. Unique : to remove duplicate roots.
3. Tic toc : to get the execution time for the method.

ii. Data structures

1. Vector to store roots



exp(-x)-x

Execution Time

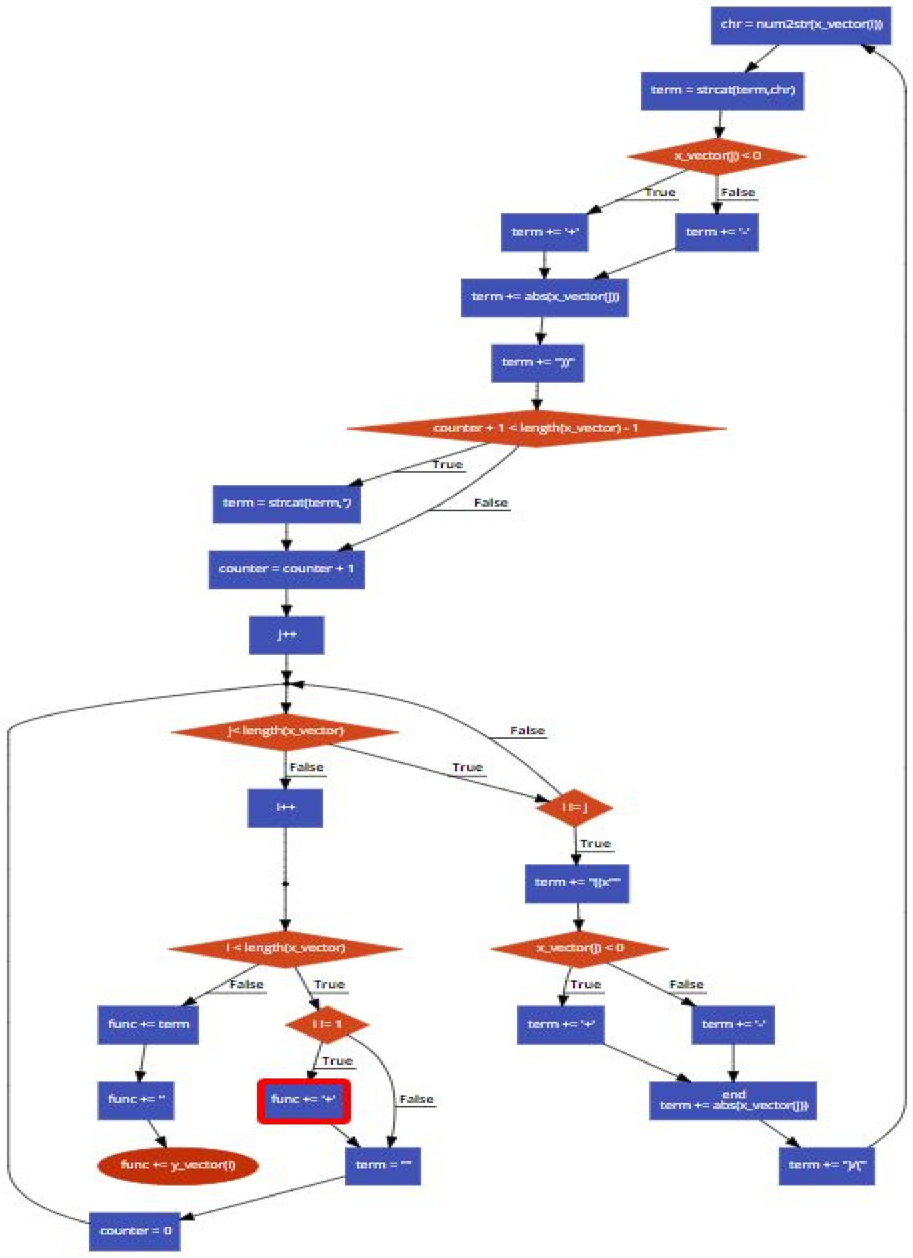
0.00560095

1
0.5671

Part II

1. Lagrange

a. Flow Chart

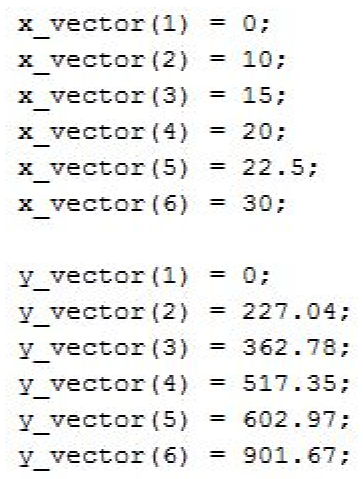


b. Built in data structures or functions used

- Tic toc : to calculate the execution time of the method.
- Strcat : to append the equation of the interpolation into a string .
- Subs : to substitute in the equation with a value and get the result .
- Num2str : to convert a number to a string to append it in the string of the equation.
- Sym : to convert the equation string to an expression .
- Simplify : to simplify the equation to a simple one .
- Char : to convert the simplified expression to a char array .
- Sort : to sort the ‘X’ points to draw the function within the interval of the points.

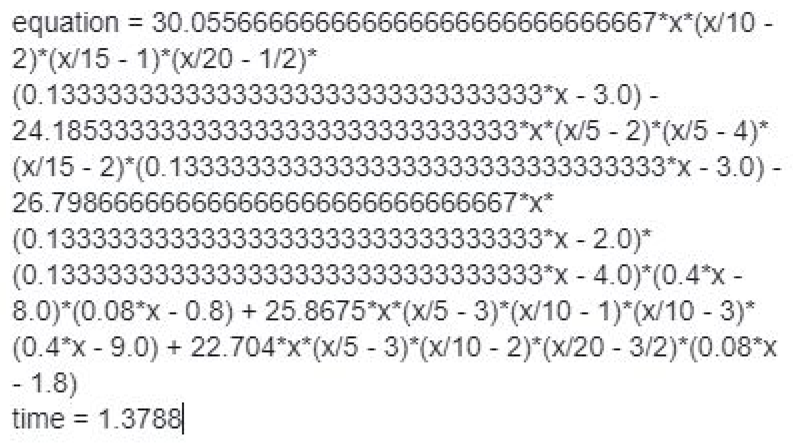
c. Equations and Analysis

Example 1 :



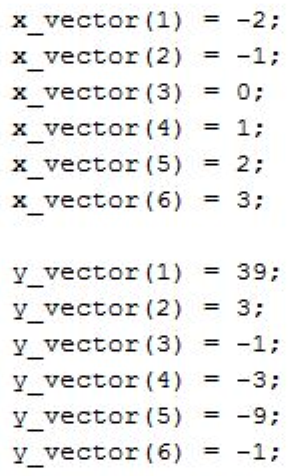
```
x_vector(1) = 0;
x_vector(2) = 10;
x_vector(3) = 15;
x_vector(4) = 20;
x_vector(5) = 22.5;
x_vector(6) = 30;

y_vector(1) = 0;
y_vector(2) = 227.04;
y_vector(3) = 362.78;
y_vector(4) = 517.35;
y_vector(5) = 602.97;
y_vector(6) = 901.67;
```

Time = 1.3788



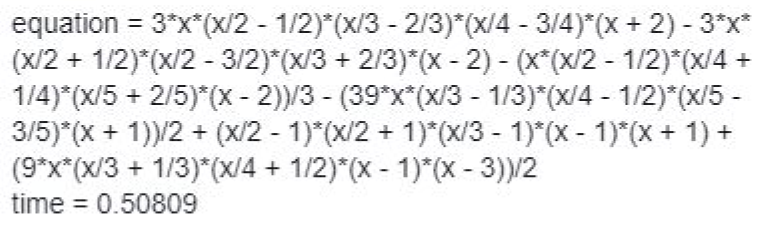
```
equation = 30.055666666666666666666666666667*x*(x/10 -
2)*(x/15 - 1)*(x/20 - 1/2)*
(0.13333333333333333333333333333333*x - 3.0) -
24.185333333333333333333333333333*x*(x/5 - 2)*(x/5 - 4)*
(x/15 - 2)*(0.13333333333333333333333333333333*x - 3.0) -
26.798666666666666666666666666667*x*
(0.13333333333333333333333333333333*x - 2.0)*
(0.13333333333333333333333333333333*x - 4.0)*(0.4*x -
8.0)*(0.08*x - 0.8) + 25.8675*x*(x/5 - 3)*(x/10 - 1)*(x/10 - 3)*
(0.4*x - 9.0) + 22.704*x*(x/5 - 3)*(x/10 - 2)*(x/20 - 3/2)*(0.08*x
- 1.8)
time = 1.3788
```

Example 2 :



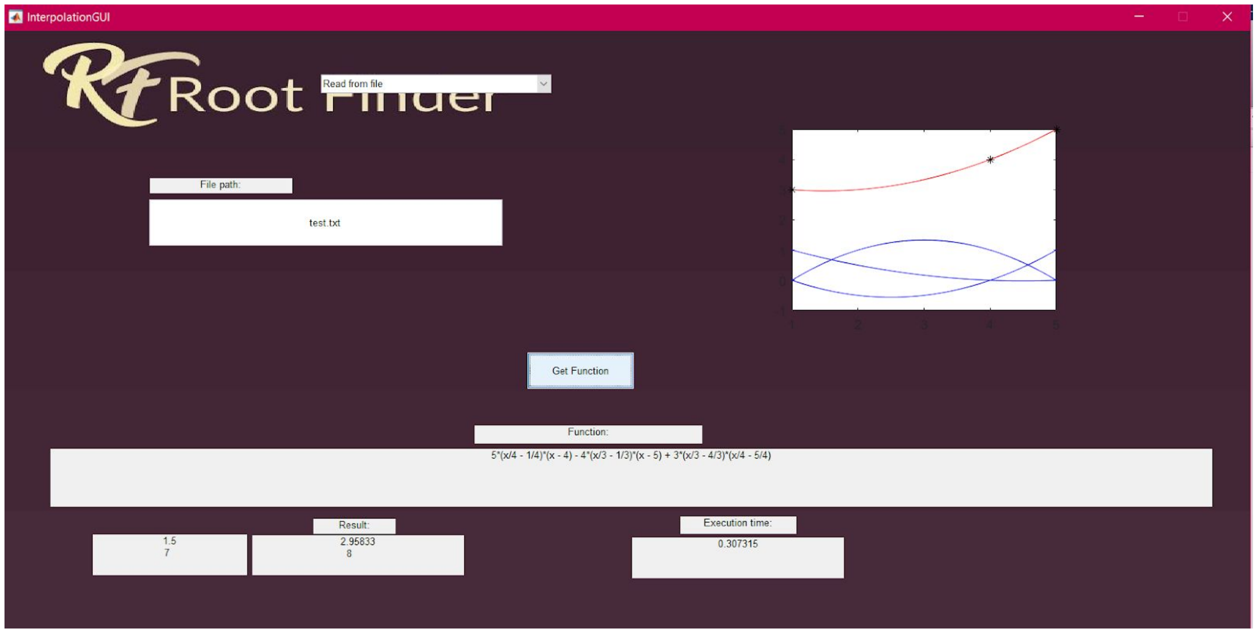
```
x_vector(1) = -2;
x_vector(2) = -1;
x_vector(3) = 0;
x_vector(4) = 1;
x_vector(5) = 2;
x_vector(6) = 3;

y_vector(1) = 39;
y_vector(2) = 3;
y_vector(3) = -1;
y_vector(4) = -3;
y_vector(5) = -9;
y_vector(6) = -1;
```

Time = 0.50809



```
equation = 3*x*(x/2 - 1/2)*(x/3 - 2/3)*(x/4 - 3/4)*(x + 2) - 3*x*
(x/2 + 1/2)*(x/2 - 3/2)*(x/3 + 2/3)*(x - 2) - (x*(x/2 - 1/2)*(x/4 +
1/4)*(x/5 + 2/5)*(x - 2))/3 - (39*x*(x/3 - 1/3)*(x/4 - 1/2)*(x/5 -
3/5)*(x + 1))/2 + (x/2 - 1)*(x/2 + 1)*(x/3 - 1)*(x - 1)*(x + 1) +
(9*x*(x/3 + 1/3)*(x/4 + 1/2)*(x - 1)*(x - 3))/2
time = 0.50809
```

d. Sample Runs



File path:

test.txt

Function:

5*(x/4 - 1/4)*(x - 4) - 4*(x/3 - 1/3)*(x - 5) + 3*(x/3 - 4/3)*(x/4 - 5/4)

Result:

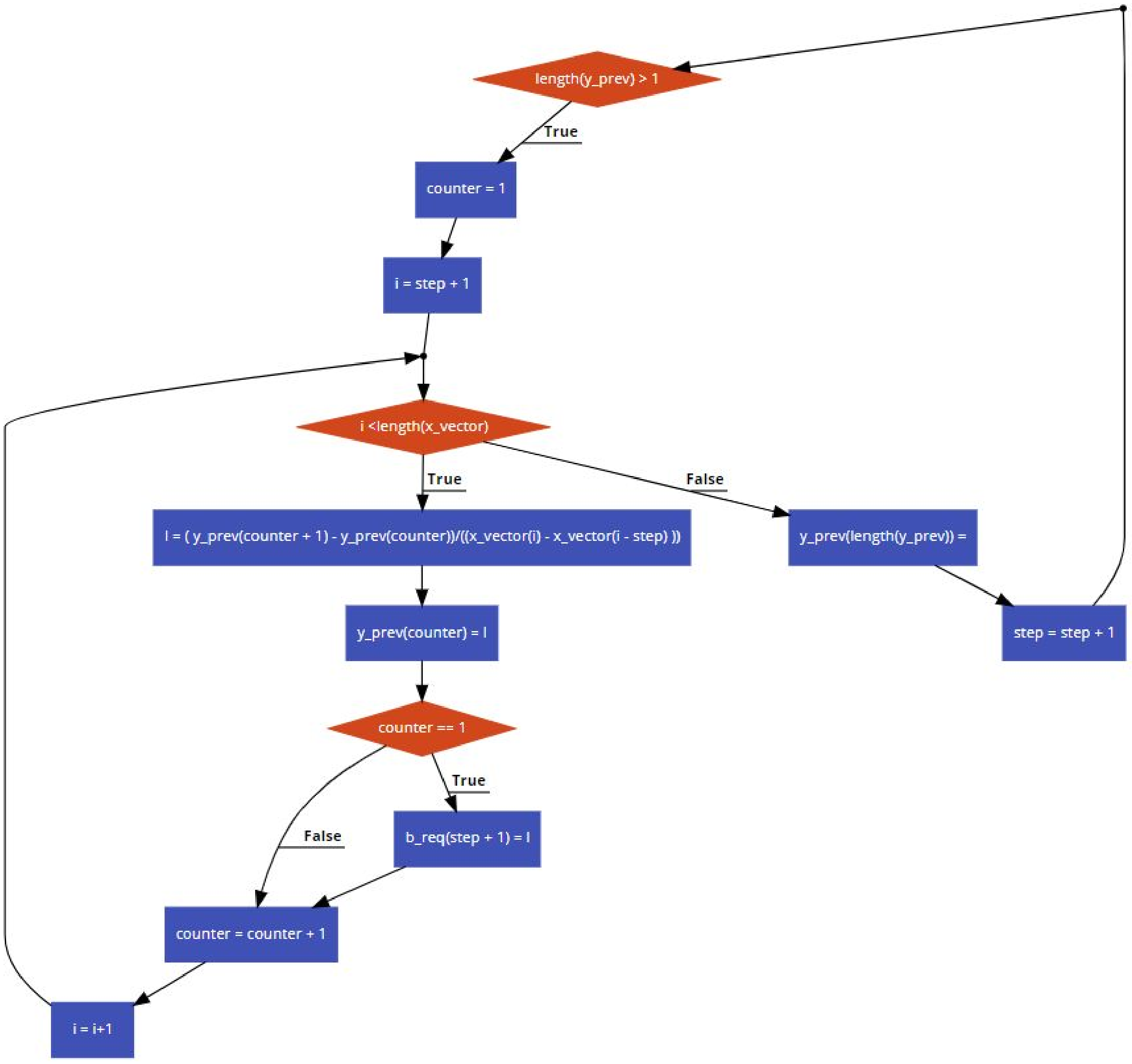
```
1.5    2.95833
7      8
```

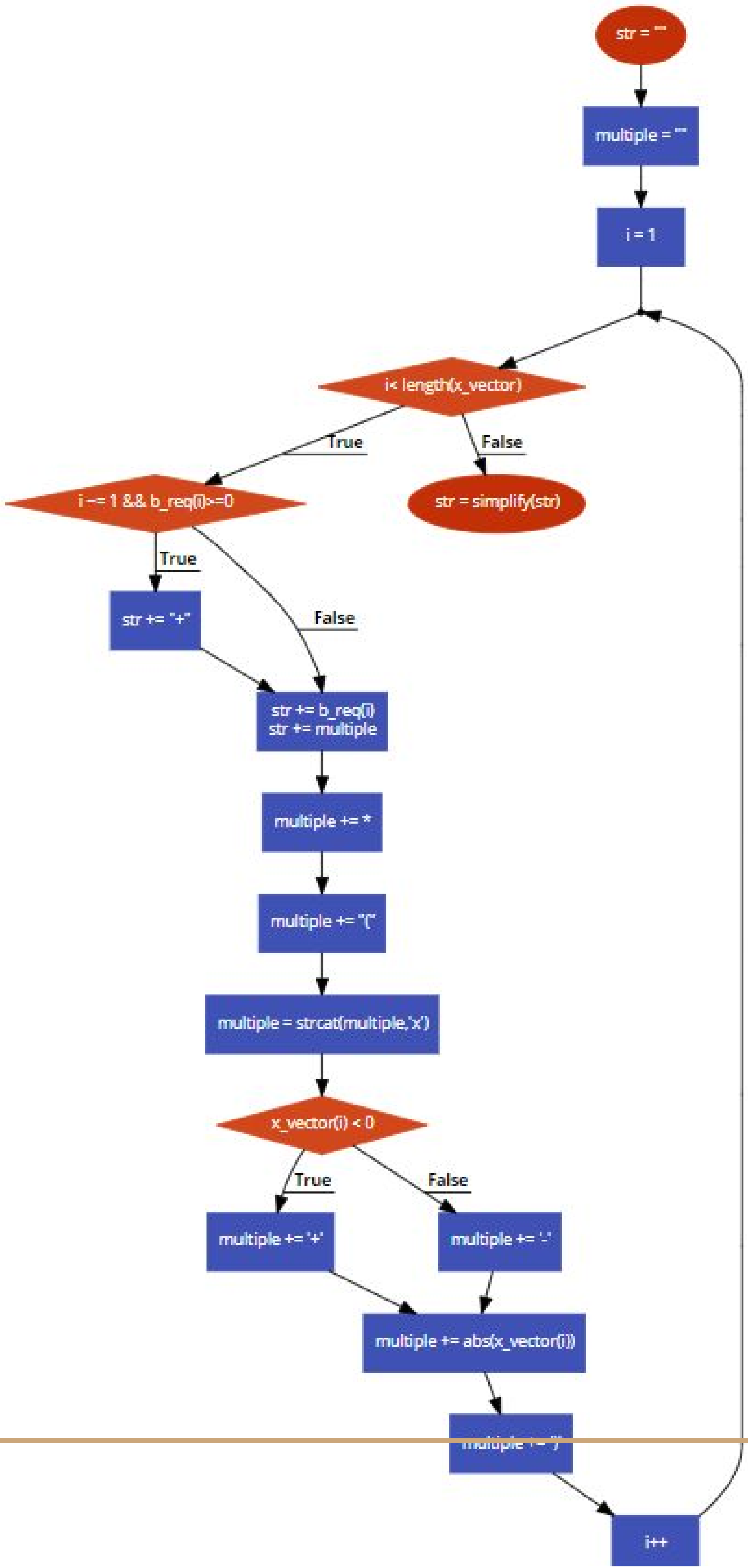
Execution time:

0.307315

2. Newton

a. Flow Chart



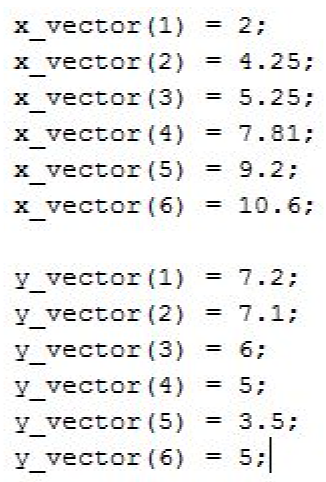

str = ""
multiple = ""
i = 1
i< length(x_vector)
True
False
str = simplify(str)
i ~= 1 && b_req(i)>=0
True
False
str += "+"
str += b_req(i)
str += multiple
multiple += *
multiple += "("
multiple = strcat(multiple,'x')
x_vector(i) < 0
True
False
multiple += '+'
multiple += '-'
multiple += abs(x_vector(i))
multiple += ')'
i++

b. Built in data structures or functions used

- Tic toc : to calculate the execution time of the method.
- Strcat : to append the equation of the interpolation into a string .
- Subs : to substitute in the equation with a value and get the result .
- Num2str : to convert a number to a string to append it in the string of the equation.
- Sym : to convert the equation string to an expression .
- Simplify : to simplify the equation to a simple one .
- Char : to convert the simplified expression to a char array .
- Sort : to sort the ‘X’ points to draw the function within the interval of the points.

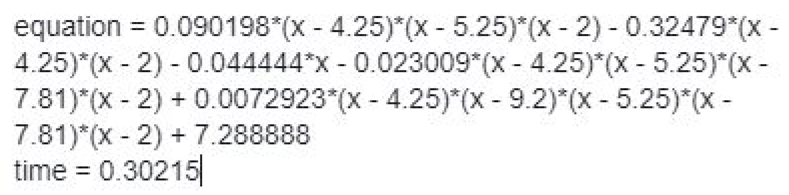
c. Equations and Analysis

Example 1 :



```
x_vector(1) = 2;
x_vector(2) = 4.25;
x_vector(3) = 5.25;
x_vector(4) = 7.81;
x_vector(5) = 9.2;
x_vector(6) = 10.6;

y_vector(1) = 7.2;
y_vector(2) = 7.1;
y_vector(3) = 6;
y_vector(4) = 5;
y_vector(5) = 3.5;
y_vector(6) = 5;
```

Time = 0.30215



```
equation = 0.090198*(x - 4.25)*(x - 5.25)*(x - 2) - 0.32479*(x -
4.25)*(x - 2) - 0.044444*x - 0.023009*(x - 4.25)*(x - 5.25)*(x -
7.81)*(x - 2) + 0.0072923*(x - 4.25)*(x - 9.2)*(x - 5.25)*(x -
7.81)*(x - 2) + 7.288888
time = 0.30215
```

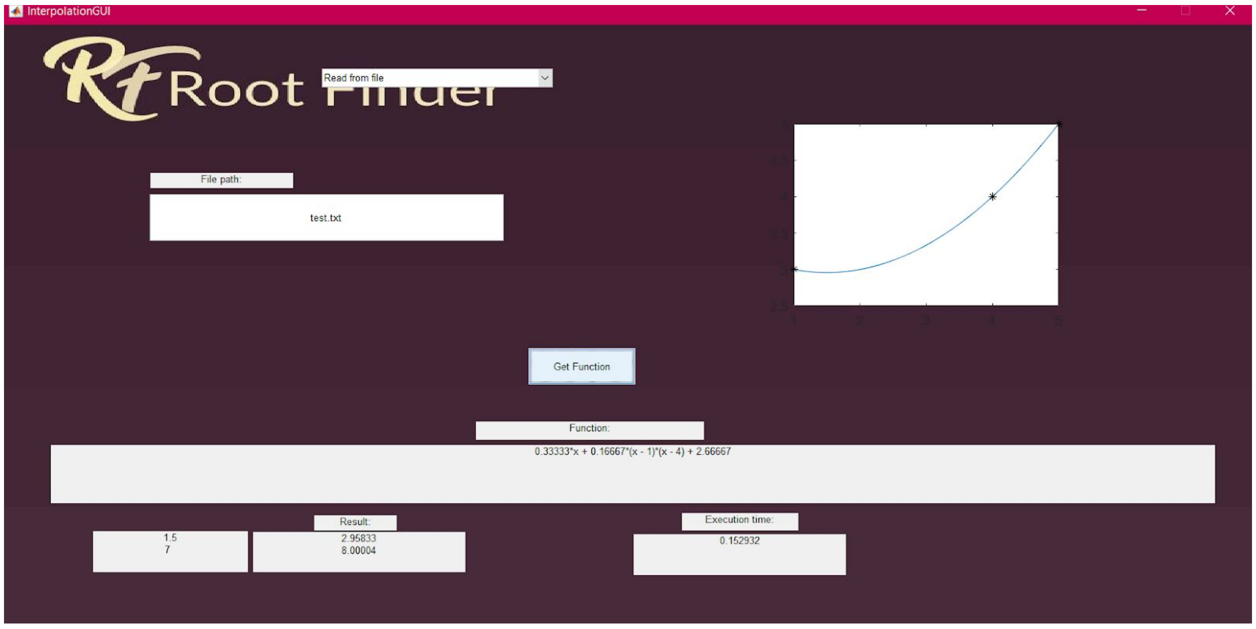
Example 2 :

```
x_vector(1) = -2;
x_vector(2) = -1;
x_vector(3) = 0;
x_vector(4) = 1;
x_vector(5) = 2;
x_vector(6) = 3;

y_vector(1) = 39;
y_vector(2) = 3;
y_vector(3) = -1;
y_vector(4) = -3;
y_vector(5) = -9;
y_vector(6) = -1;
```

Time = 0.26078

```
equation = 16*(x + 1)*(x + 2) - 36*x - 5*x*(x + 1)*(x + 2) + x*(x
- 1)*(x + 1)*(x + 2) - 33
time = 0.26078
```

d. Sample Runs



File path:

test.txt

Function:

0.33333*x + 0.16667*(x - 1)*(x - 4) + 2.66667

Result:

1.5	2.95833
7	8.00004

Execution time:

0.152932