

Course Code : CC489

Prepared by :

Mostafa Adel 6189

Mahmoud Hossam 6174

numerical analysis

Project 1

Table of Contents

**User manual** .......................................................................................................**2**

**Bisection Method** ….........................................................................................…..**5**

**False – Position Method** .......................................................................................**9**

**Analysis & comparison between Bisection and False position** ...........................**14**

**Fixed – Point Method** ………………….......................................................................**15**

**Analysis of Fixed – Point Method**  .......................................................................**19**

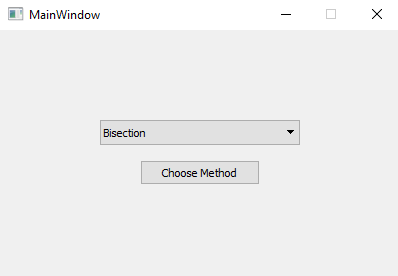
**Newton – Raphson Method** …………......................................................................**20**

**Secant Method** …………........................…………........................…………....................**14**

User manual.

1. main menu :

This is the main menu of the program where you can choose the root finding method that you want from the drop down menu then press the “choose method” button to confirm your choice and proceed to the input window



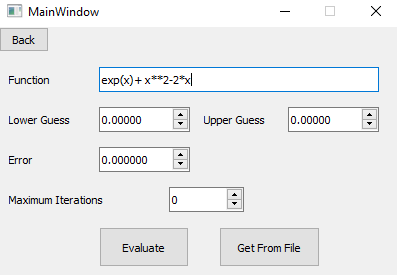
2.Input menu :

The following is the input menu where you enter the Function and the data needed to find the root or you can read your data from a file by clicking on the “Get From File” button

here you enter the function as shown in the function box , the are some rules for entering the function :

1. To enter an exponential function you write “exp( )” and inside the bracket you write the power of the exponential , example : e^x 🡪 exp (x)
2. To enter a power you write “\*\*” , example : x^2 🡪 x\*\*2
3. To multiply a const with a variable you must enter “\*” between them , example : 2x 🡪2\*x
4. If the method chosen is the fixed-point method then the g(x) is the function to enter not f(x)

In these two boxes you enter the upper and lower values of the interval if it’s a method that needs an interval like the bisection method , or they can act as 2 initial values like in the secant method as for a method that takes a single value like the fixed – point method it takes the average of the 2 values.



Here in the error box you can specify the accuracy that you want to reach before the program stops , if you left it as zero it will be set to 0.00001 by default.

Here you enter the maximum iterations that the program can reach before it stops further approximation , if it’s left zero it will be set to 50 iterations by default.

Finally you press the “Evaluate” Button to evaluate the root.

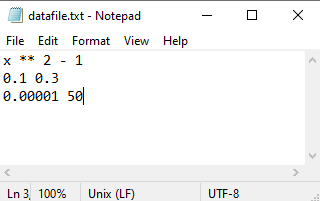
Note : if you choose to read data from a file then you must enter the data in the following format :

“Function”

“Lower guess” “Upper guess”

“Error” “Maximum Iterations”

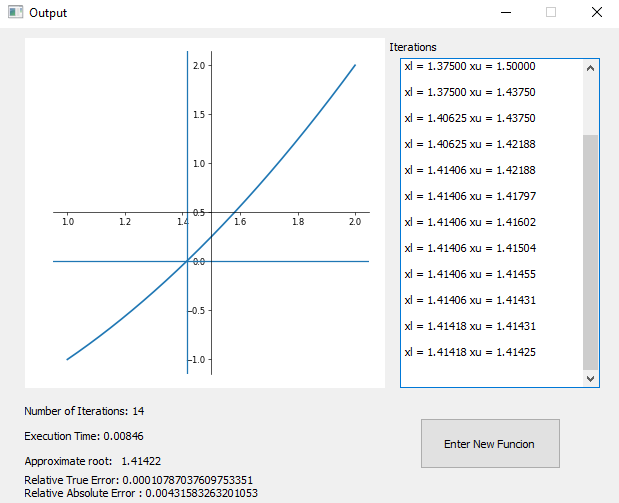
This an example of a valid data file to read from :



3.output menu :

Plot of the graphs

The values at each Iteration



time spent in execution

Relative and absolute error

Number of iterations used

pseudocode for each method and sample runs.

Bisection method

Pseudocode :

Bisection(xLower,xUpper,es,maxIterations,function)

{

if function(xLower)\*function(xUpper) > 0

output("Wrong Input")

return

for i:0 to maxIterations

{

xr = (xLower+xUpper)/2

ea = abs(xLower-xUpper)

test = function(xLower)\*function(xUpper)

if test < 0

xUpper = xr

else

xLower = xr

if test == 0

ea = 0

if ea < es

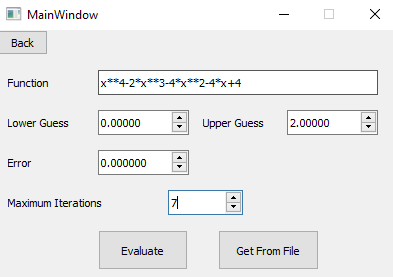
break

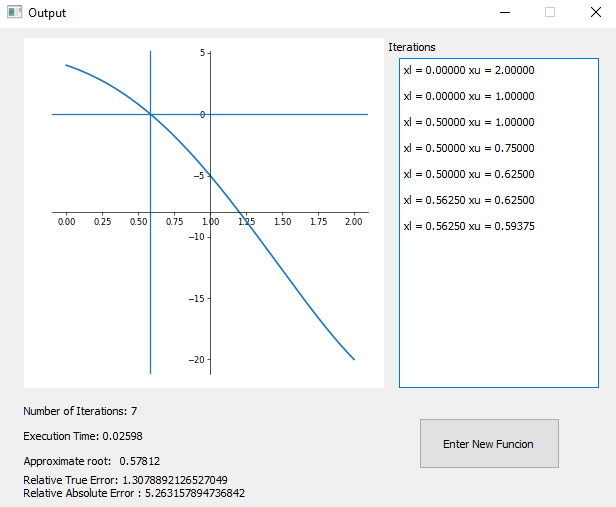
}

}

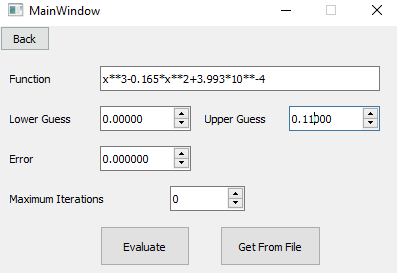
Sample runs :

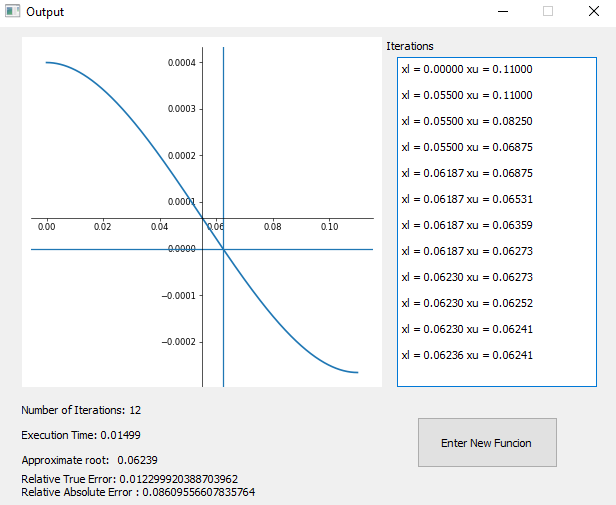
1) x4 - 2x3 - 4x2 – 4x + 4 = 0 , [ 0 , 2 ]



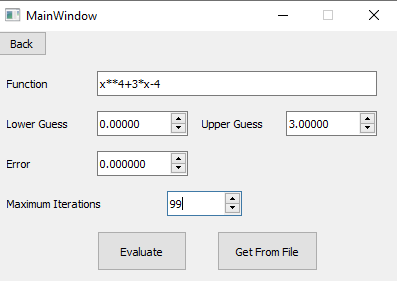


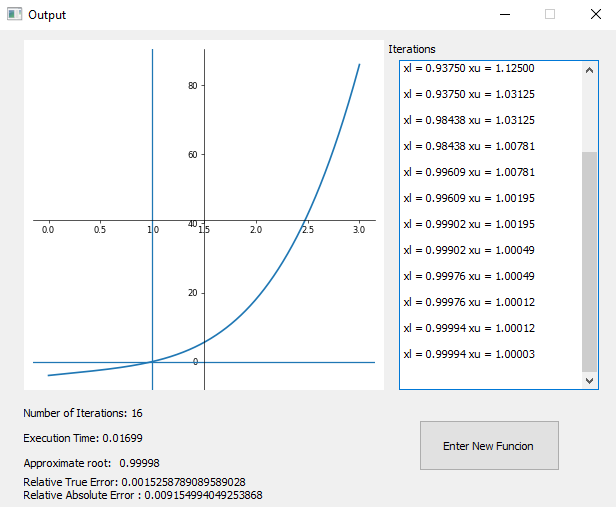
2) x3 - 0.165x2 + 3.993\*10-4 = 0 , [ 0 , 0.11 ]





3) x4 + 3x - 4 = 0 , [ 0 , 3 ]





False-Position method

Pseudocode :

falsePosition(xLower,xUpper,es,maxIterations,function)

{

a[0] = xLower

b[0] = xUpper

ya[0] = function(a[0])

yb[0] = function(b[0])

if ya[0] \* yb[0] > 0.0:

output("Func same sign at both ends")

return

iteration = 0

for i: 0 to maxIterations

{

iteration += 1

x[i] = b[i] - yb[i] \* (b[i] - a[i]) / (yb[i] - ya[i])

y[i] = function(x[i])

if y[i] == 0

output("0 found")

break

else if 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]

ea = abs(x[i] - x[i+1])

if i > 1 and ea < es

output("Method Converged")

break

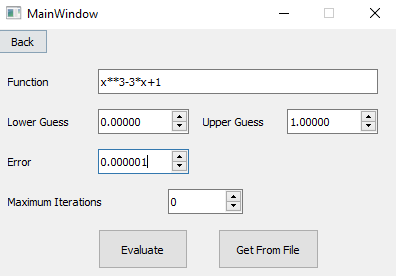
}

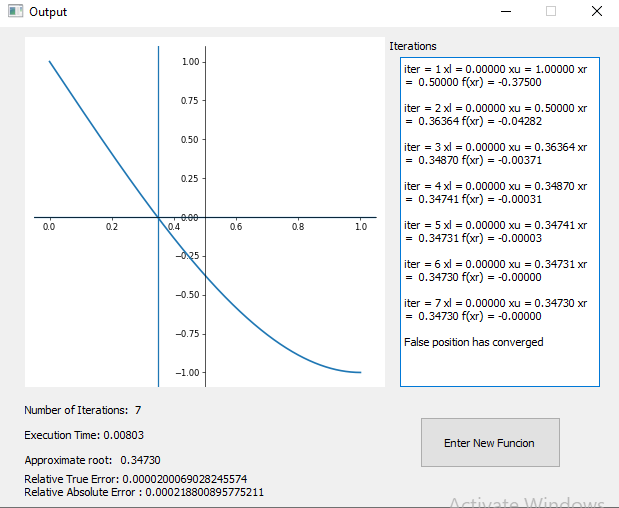
if iterations >= maxIterations

output("Zero might not be Found")}

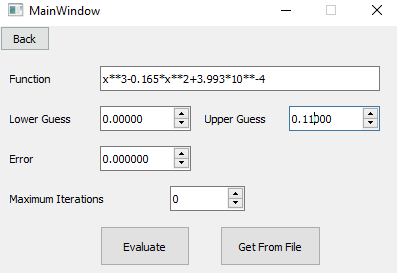
Sample runs :

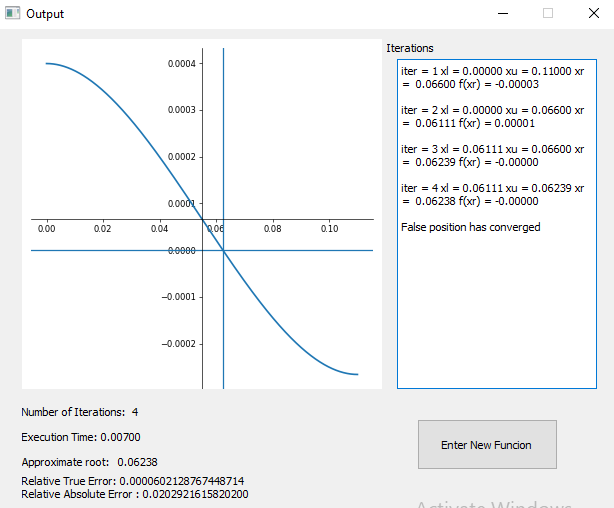
1) x3 - 3x + 1 = 0 , [ 0 , 1 ]



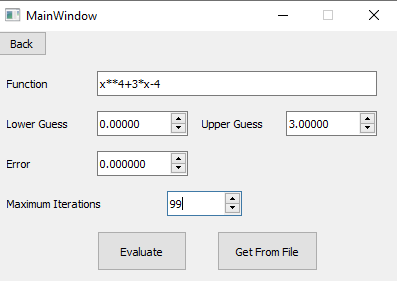


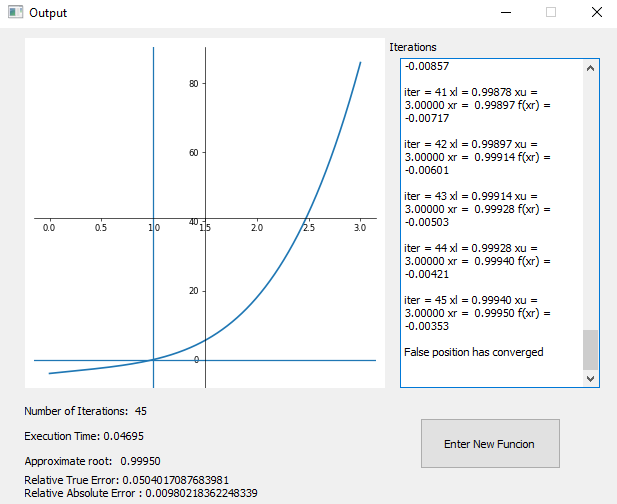
2) x3 - 0.165x2 + 3.993\*10-4 = 0 , [ 0 , 0.11 ]





3) x4 + 3x - 4 = 0 , [ 0 , 3 ]





**Analysis & comparison :**

we notice that the False-position method usually converges faster than the bisection method

example : f(x) = x3 - 0.165x2 + 3.993\*10-4 = 0 , [ 0 , 0.11 ]

in this case the bisection method converged in 12 iteration while in the False-position method it converged in only 4 iterations

but sometimes it may converge much slower than the bisection method

example : f(x) = x4 + 3x - 4 = 0 , [ 0 , 3 ]

we can see that in the bisection method it converged in 16 iterations while in the False – position method it converged in 46 iterations

so we can conclude that the False-position method is a fast converging method in most cases but it shouldn’t be used all the time as it sometimes may converge slowly

Fixed-Point method

Pseudocode :

fixedPoint(xLower,xUpper,es,maxIterations,function)

{

xr = (xLower + xUpper)/2

iterations = 0

while True

{

tempX = xr

xr = function(xr)

if xr != 0

ea = abs(xLower-xUpper)

iterations += 1

if ea < es or iterations > maxIterations

break

}

if iterations > maxIterations

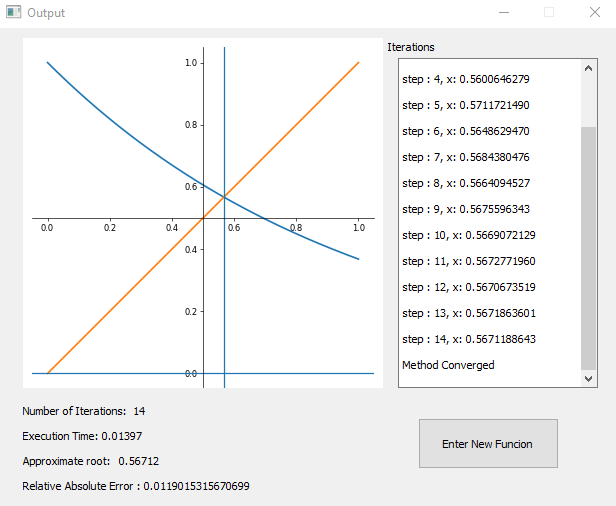
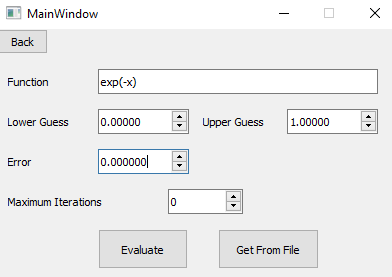
output("Method Diverged")

}

Sample runs :

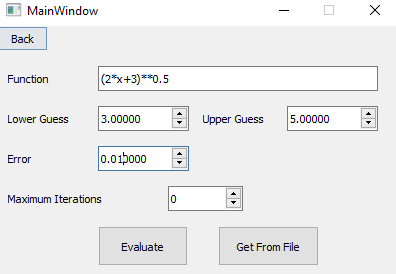
1) f(x) = e-x - x = 0 , g(x) = e-x , x0 = 0.5

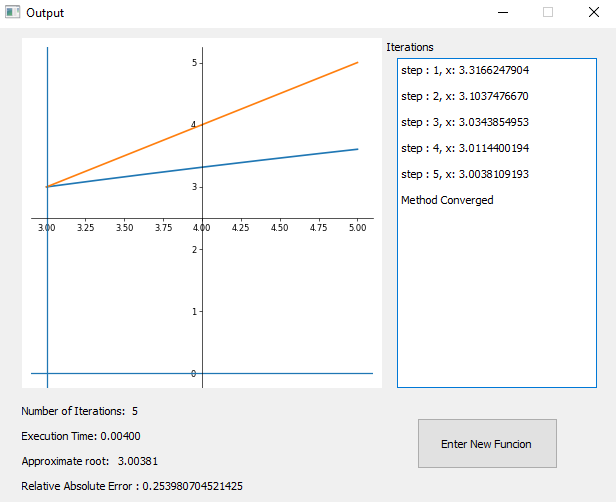
Note : the program takes the average of the upper and lower guess as the initial value so will enter 0 and 1 to get x0 = 0.5.



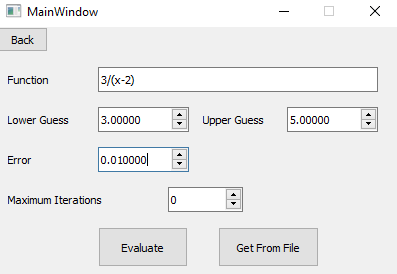
2) f(x) = x2 - 2x - 3 , x0 = 4

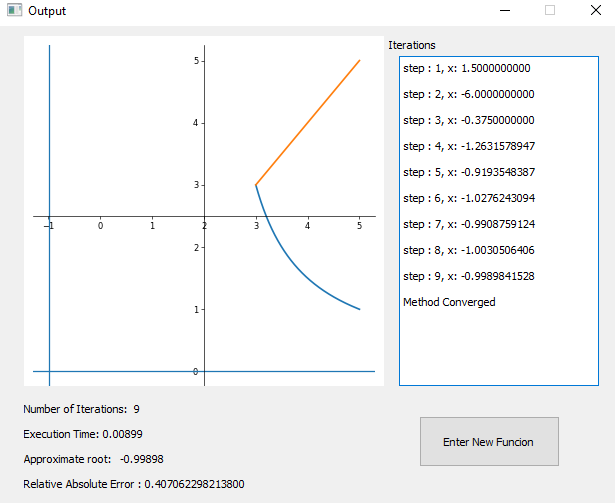
a) g1(x) = (2x+3)



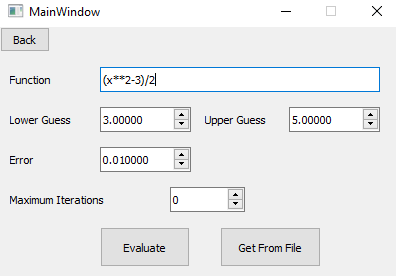


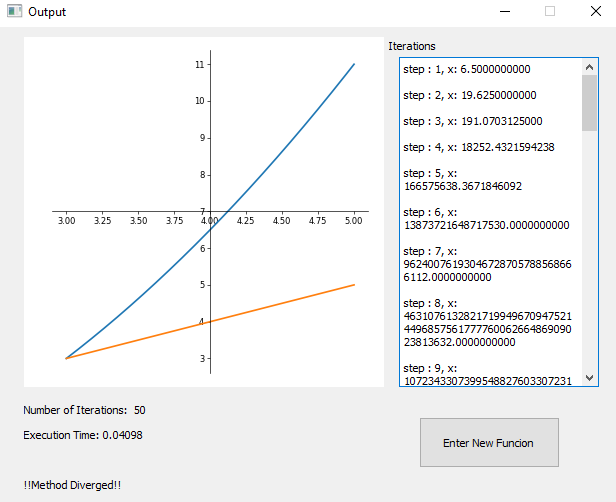
b) g2(x) = 3/(x-2)





c) g3(x) = (x2-3)/2





**Analysis**

It was shown in the previous sample runs that the fixed-point method can diverge quickly but this can widely differ when changing the g(x)

for example we used three different g(x) functions to evaluate the function :

f(x) = x2 - 2x - 3 , x0 = 4

we evaluated using :

g1(x) = (2x+3)

g2(x) = 3/(x-2)

g3(x) = (x2-3)/2

the results :

g1(x) converged fast within 5 iterations only

g2(x) also converged but slower than g(x) , it converged within 9 iterations

g3(x) diverge

we conclude that not every g(x) converges and if it converges , then the speed of convergence may differ from a g(x) to another.

So we must choose g(x) wisely before evaluating the root.

Newton Raphson’s method

Pseudocode :

function newtonRaphson(xLower,xUpper,error,maxIterations,function)

{

xGuess = (xLower+xUpper)/2

dFunction = diffrentiate(function)

xr[maxIterations]

fx[maxIterations]

dfx[maxIterations]

itrations = 0

for i : 1 to maxIterations

{

iterations += 1

xr[i] = xr[i-1] - fx[i-1]/dfx[i-1]

fx[i] = function(xr[i])

dfx[i] = dFunction(xr[i])

ea = abs(xr[i]-xr[i-1])

if ez < es :

output("Converged")

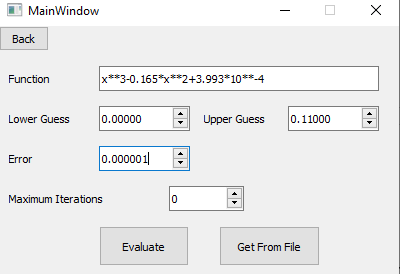
break

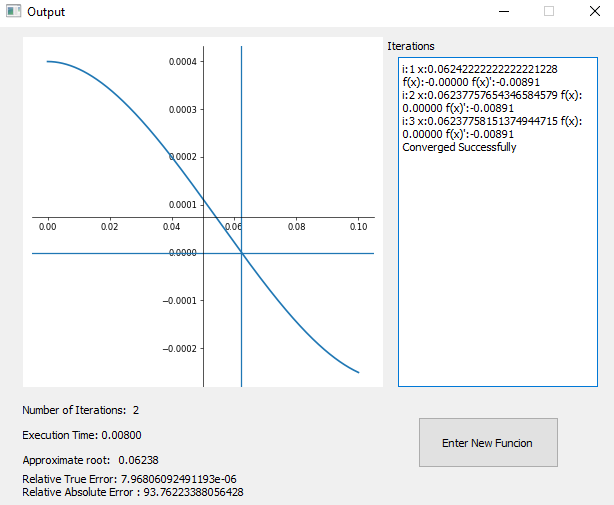
}

}

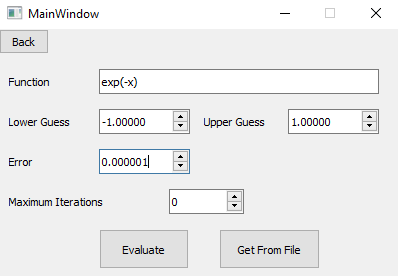
Sample runs :

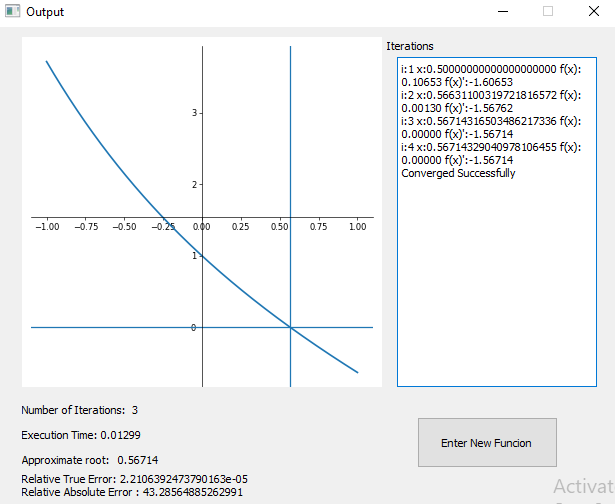
1) x3 - 0.165x2 + 3.993\*10-4 = 0 , x0 = 0.5





2) e-x - x = 0 , x0 = 0





Secant method

Pseudocode :

Secant (x0, x1, error, maxIterations)

{

step = 0

condition = True

x2 = 0

while condition

{

if function(x0) == function (x1)

{

output("Divide by zero error")

break

}

f0 = function(x0)

f1 = function(x1)

x2 = x0 - (x1 - x0) \* f0 /(f1 - f0)

x0 = x1

x1 = x2

step += 1

if step > maxIterations

{

output("Max Iterations reached")

break

}

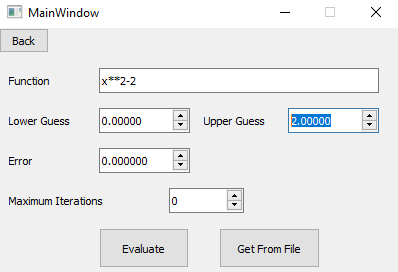
// to do the do while loop which is not in python

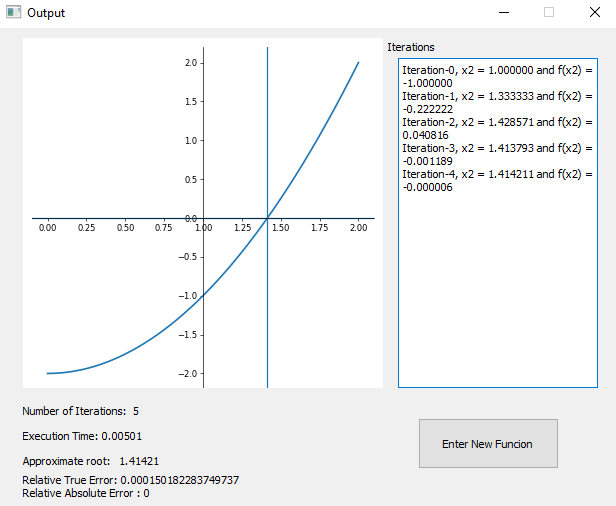
}

}

Sample runs :

1) x2-2 = 0 , x-1 = 0 , x0 = 2





2) x3 - 0.165x2 + 3.993\*10-4 = 0 , x-1 = 0 , x0 = 0.1

