Numerical Report

<u>Analyzing the behavior of numerical methods</u>

<u>Contents:</u>

TEAM

Name: Mayar Ahmed Ramadan **ID:**6055

Name: Menna Abd El Meguid **ID:**6538

❖ BISECTION METHOD:

A)The code of the bisection method:

```
from sympy import* #library for symbolic mathematics to be full featured computer algebra
import os.path
def func(expr, value, x):
    return expr.subs(x, value)
# Prints root of func(x)
# with error of EPSILON
def bisection(a, b, expr, maxIteration, Epsilon, x):
    print("In bisectiooon")
    file = open("../View/values.txt", "w")
    if os.path.isfile('../Viewss/values.txt'):
        print("File exist")
    else:
        print("File not exist")
    if (func(expr,a, x) * func(expr,b, x) >= 0):
        print("You have not assumed right a and b\n")
        return
    c = a
    step = 1
    while (step < maxIteration):
        # Find middle point
        print('Iteration(%d): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-
mid)' % step)
        print("\t\t\t %.6f \t %.6f \t  %.6f \t %.6f \t %.6f \t %.6f \n" % (
        a, func(expr, a, x), b, func(expr, b, x), c, func(expr, c, x)))
        file.write("%.6f %.6f %.6f %.6f %.6f %.6f \n" % (
            a, func(expr, a, x), b, func(expr, b, x), c, func(expr, c, x)))
        c_old=c
        c = (a+b)/2
        if(abs((c-c_old)/c)<Epsilon):
            break
        # Check if middle point is root
        if (func(expr,c, x) == 0.0):
            break
        # Decide the side to repeat the steps
        if (func(expr,c, x)*func(expr,a, x) < 0):
            b = c
        else:
            a = c
        step +=1
    file.close()
    finalIteration = step
    return "%d ): %.6f"%(finalIteration,c)

# Main code
def mainFunc(function, maxIteration, epsilon, a, b):
     # the possible variable names must be known beforehand...
    expr = sympify(function)
    x = var('x')

    return bisection(a, b, expr, maxIteration, epsilon, x)
```

B)Sample runs &Analysis

Now we will choose some examples to test the function:

1) first function:

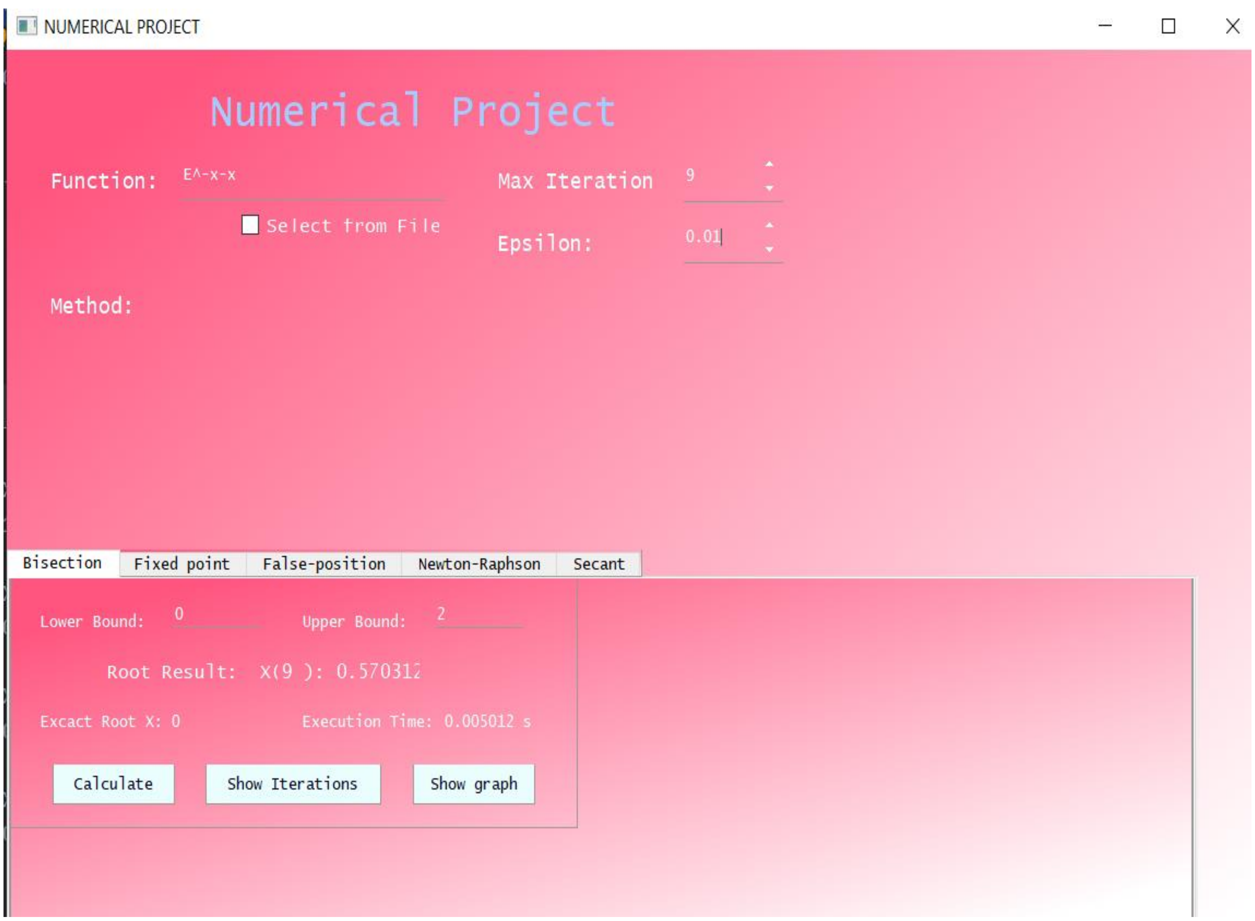
E^-x-x

-lower bound: 0

-upper bound:2

-number of itrations:9

-Epsilon:10^-2



Iterations:

```
In bisectiooon
File not exist
Iteration(1): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.000000    1.000000     2.000000   -1.864665   0.000000    1.000000

Iteration(2): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.000000    1.000000     1.000000   -0.632121   1.000000    -0.632121

Iteration(3): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.500000    0.106531     1.000000   -0.632121   0.500000    0.106531

Iteration(4): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.500000    0.106531     0.750000   -0.277633   0.750000    -0.277633

Iteration(5): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.500000    0.106531     0.625000   -0.089739   0.625000    -0.089739

Iteration(6): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.562500    0.007283     0.625000   -0.089739   0.562500    0.007283

Iteration(7): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.562500    0.007283     0.593750   -0.041498   0.593750    -0.041498

Iteration(8): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
             0.562500    0.007283     0.578125   -0.017176   0.578125    -0.017176
```

C)Observation :

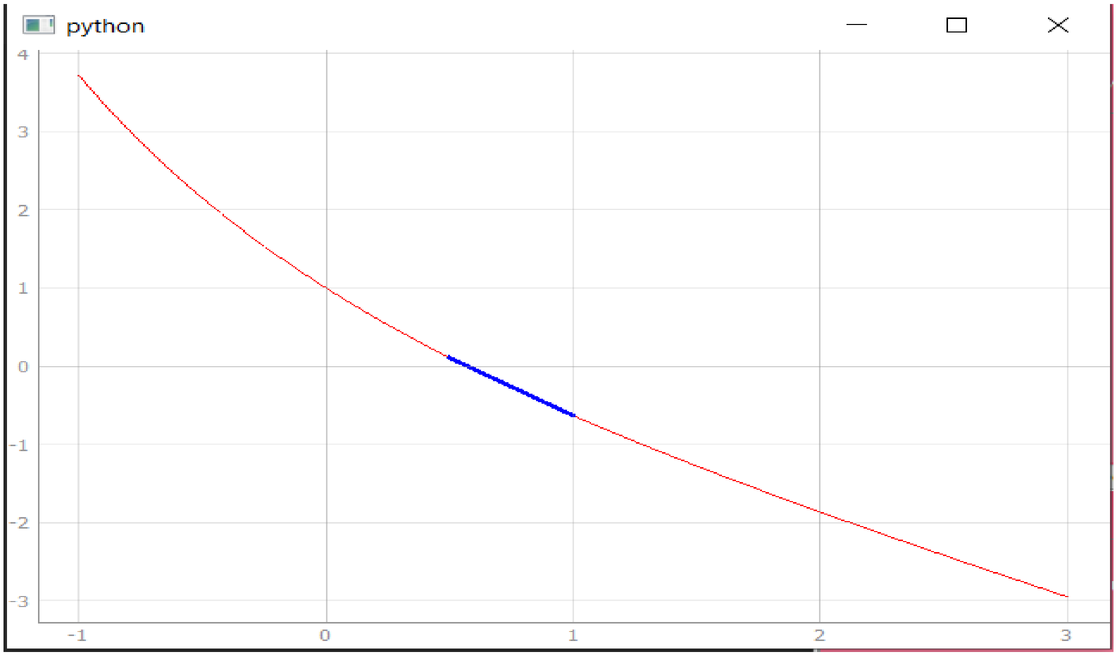
As the number of iterations increase we reaches value near the exact solution.

Bisection method can be done here on this function as the :

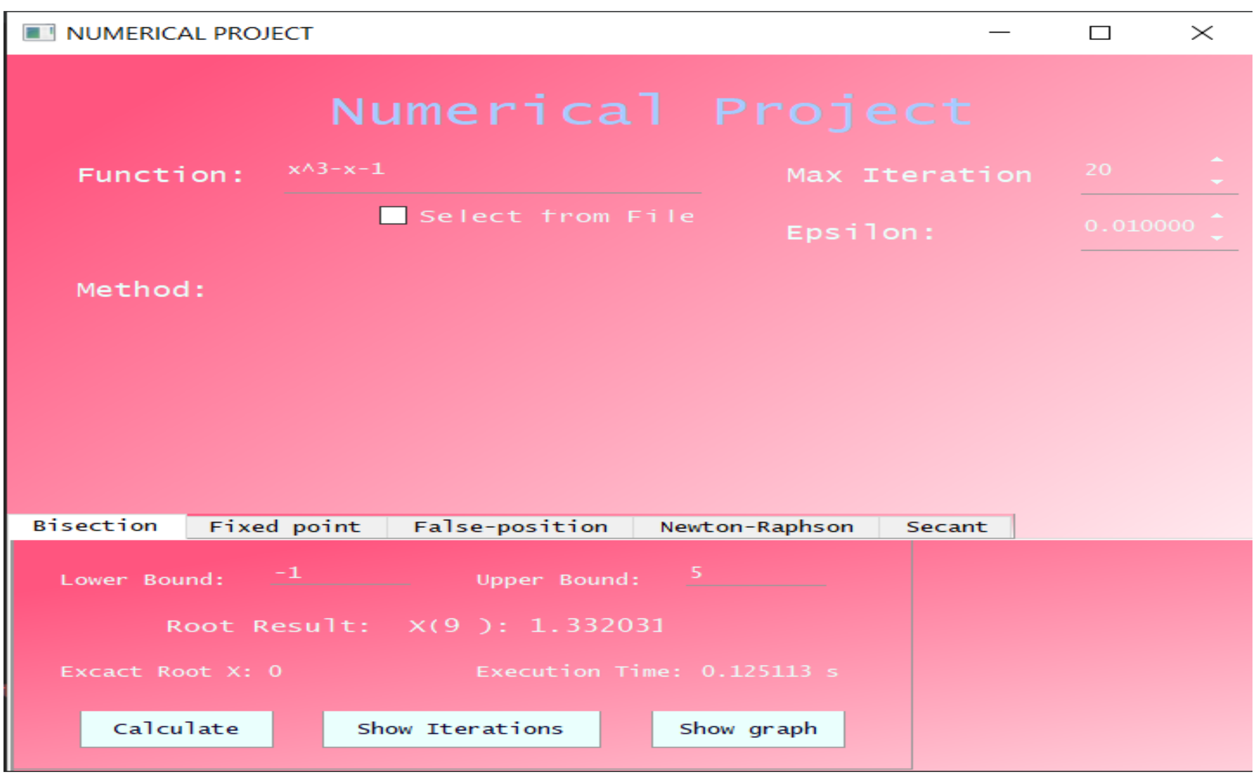
f(x-lower)*f(x-upper)<0

and here we can notice that we have stopped at the 8^{th} iteration despite we have entered 9 iterations , as the relative error became less than the tolerance.

the graph of the bisection method of the previous function:



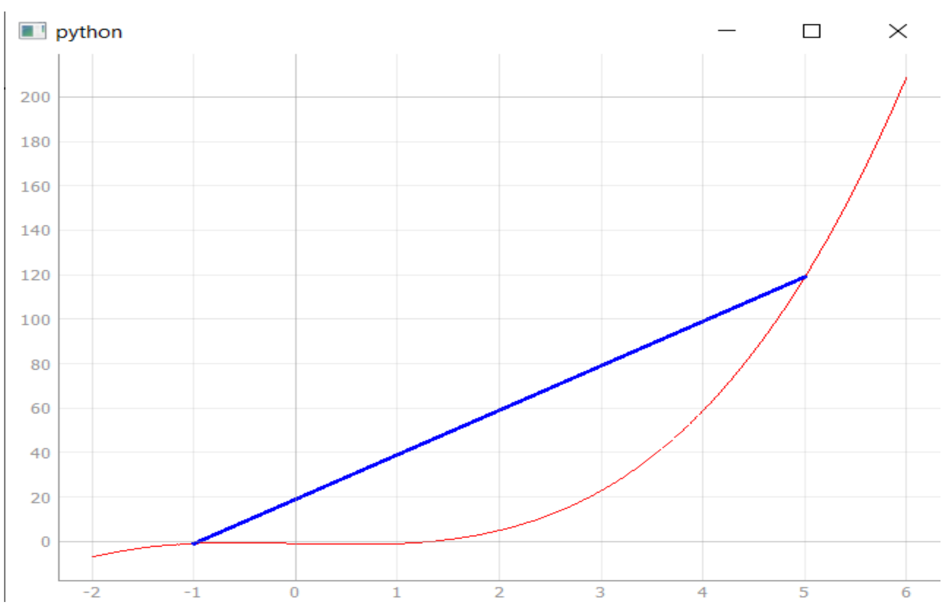
2)second equation:

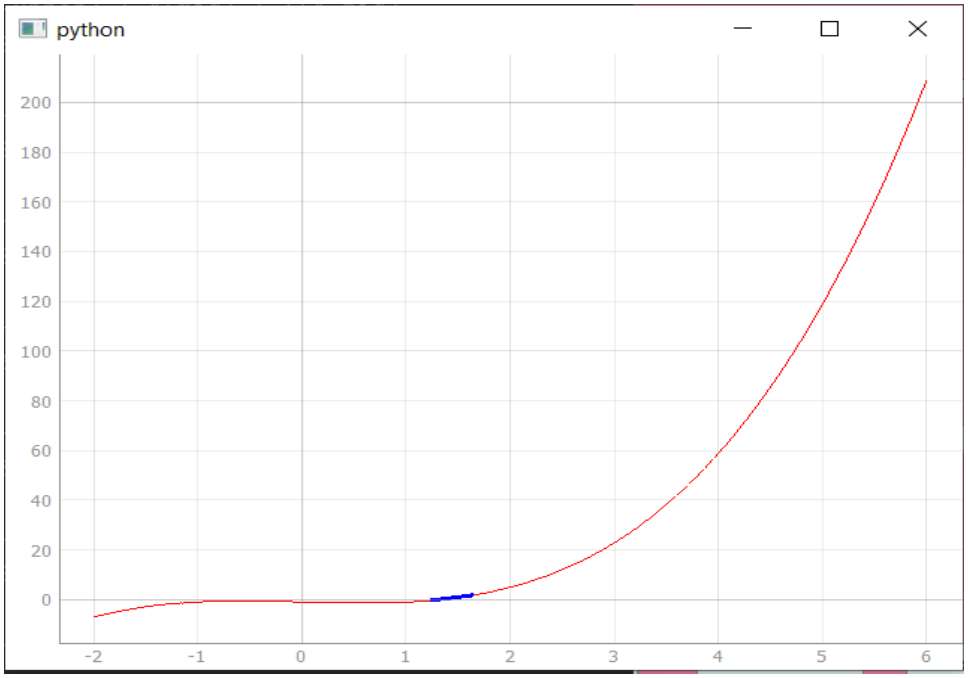


Iterations:

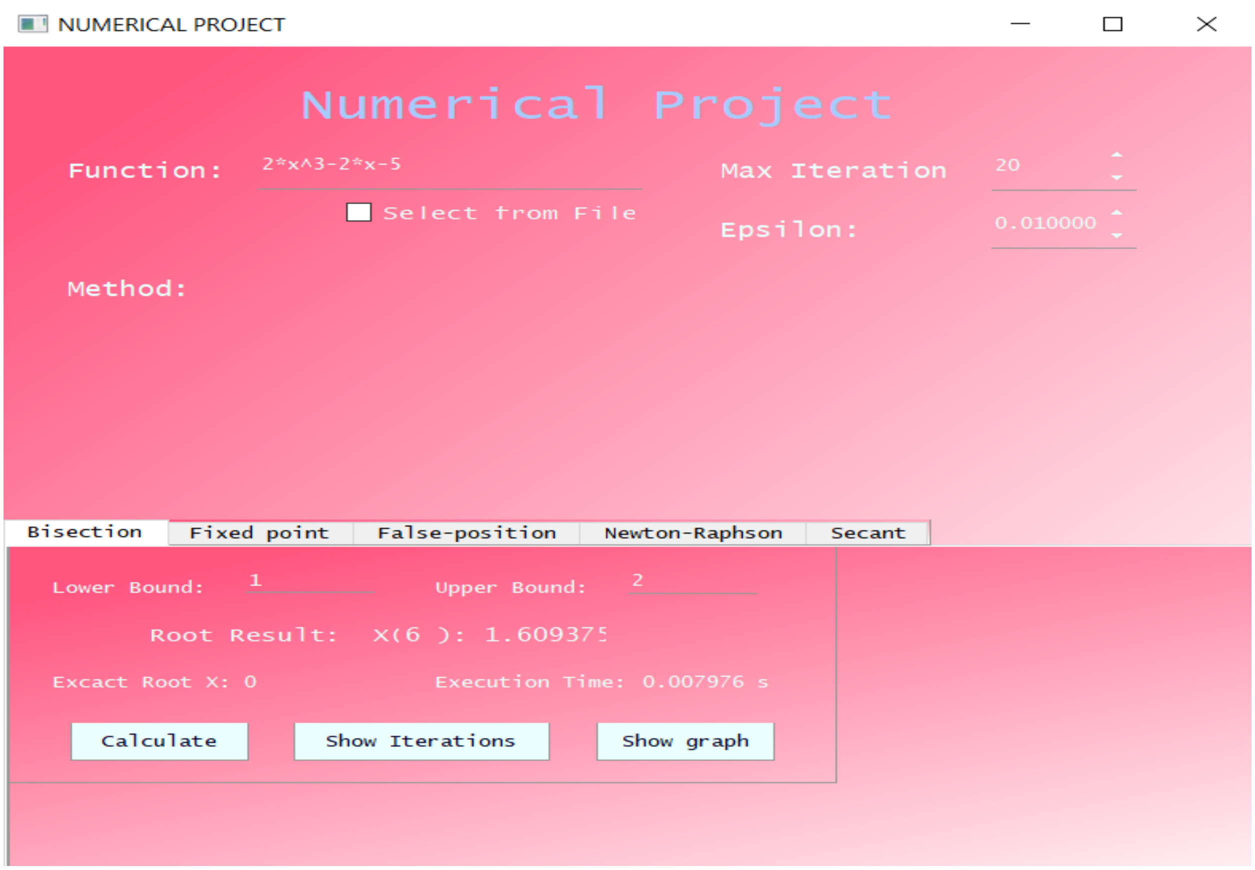
```
Iteration(1): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           -1.000000   -1.000000    5.000000   119.000000      -1.000000   -1.0000

Iteration(2): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           -1.000000   -1.000000    2.000000   5.000000    2.000000    5.000000

Iteration(3): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           0.500000    -1.375000    2.000000   5.000000    0.500000    -1.375000

Iteration(4): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           1.250000    -0.296875    2.000000   5.000000    1.250000    -0.296875

Iteration(5): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           1.250000    -0.296875    1.625000   1.666016    1.625000    1.666016

Iteration(6): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           1.250000    -0.296875    1.437500   0.532959    1.437500    0.532959

Iteration(7): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
           1.250000    -0.296875    1.343750   0.082611    1.343750    0.082611

Iteration(8): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
```

GRAPH FOR THIS FUNCTION:





3)Third function:

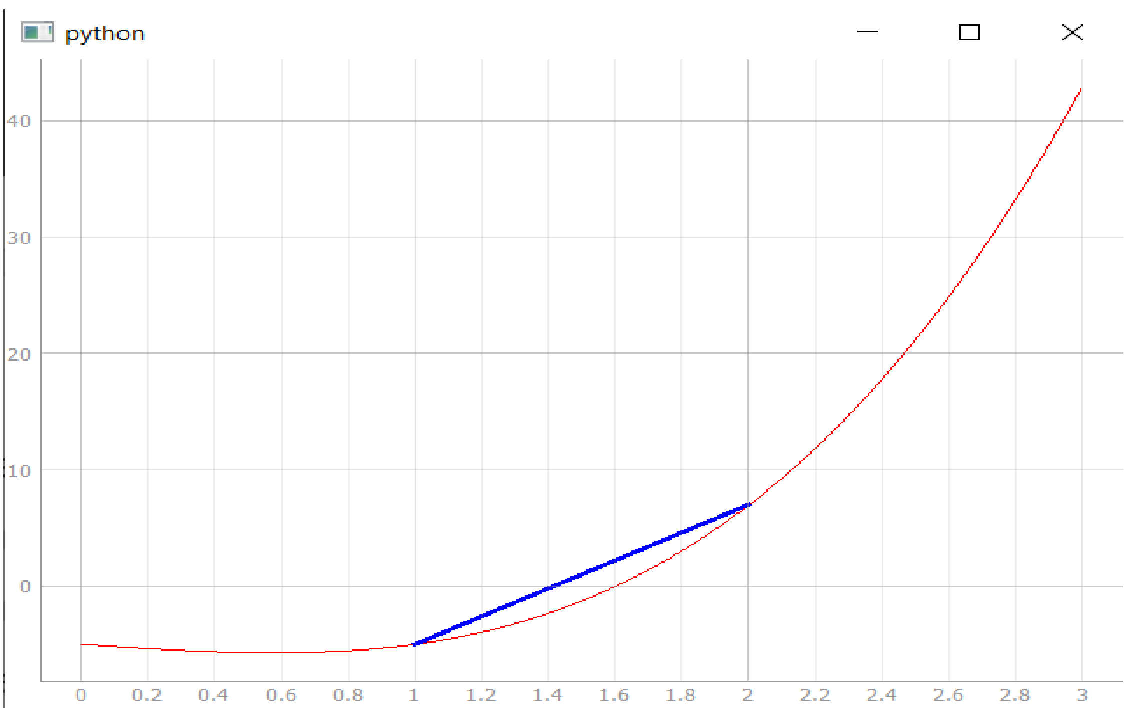


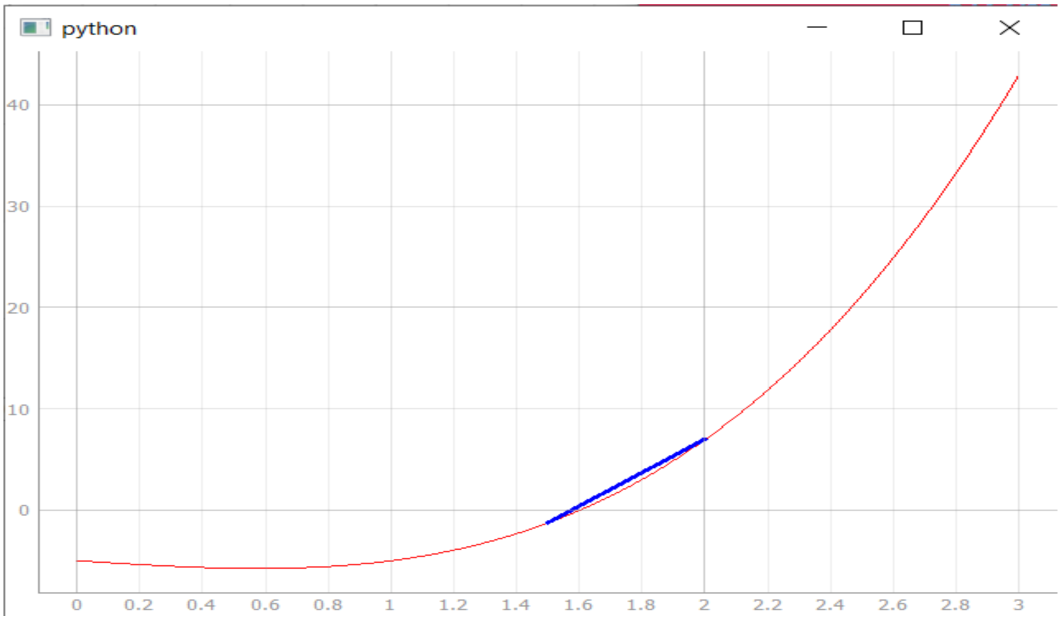
Iterations:

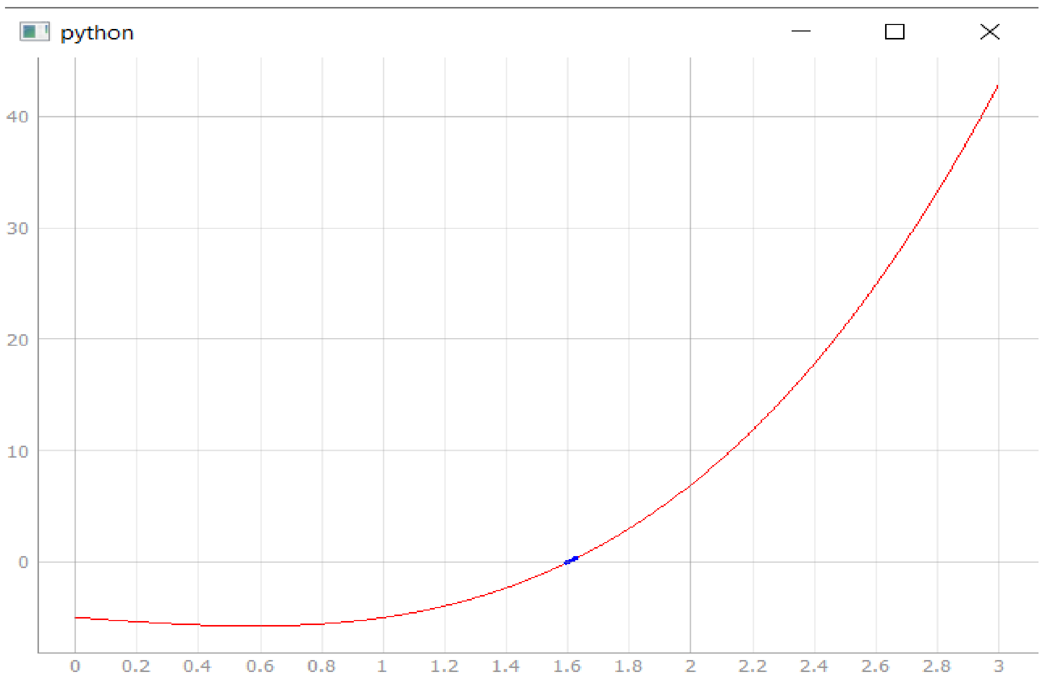
```
Iteration(1): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.000000    -5.000000    2.000000   7.000000     1.000000    -5.000000

Iteration(2): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.500000    -1.250000    2.000000   7.000000     1.500000    -1.250000

Iteration(3): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.500000    -1.250000    1.750000   2.218750     1.750000    2.218750

Iteration(4): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.500000    -1.250000    1.625000   0.332031     1.625000    0.332031

Iteration(5): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.562500    -0.495605    1.625000   0.332031     1.562500    -0.495605

Iteration(6): X(lower) | f(X-lower) | X(upper) | f(X-upper) | X(mid) | f(X-mid)
              1.593750    -0.091125    1.625000   0.332031     1.593750    -0.091125
```

Graphs of this function:

Dynamic graph of Iterations :







Reaching exact root as number of iterations increases

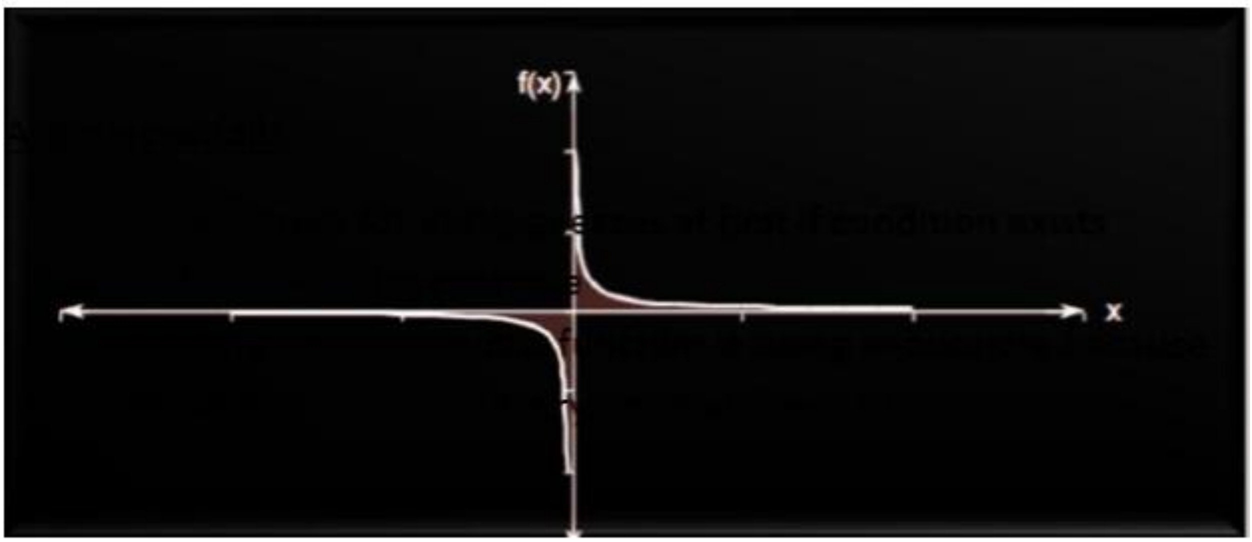
D)Pitfalls:

1. Slow method .

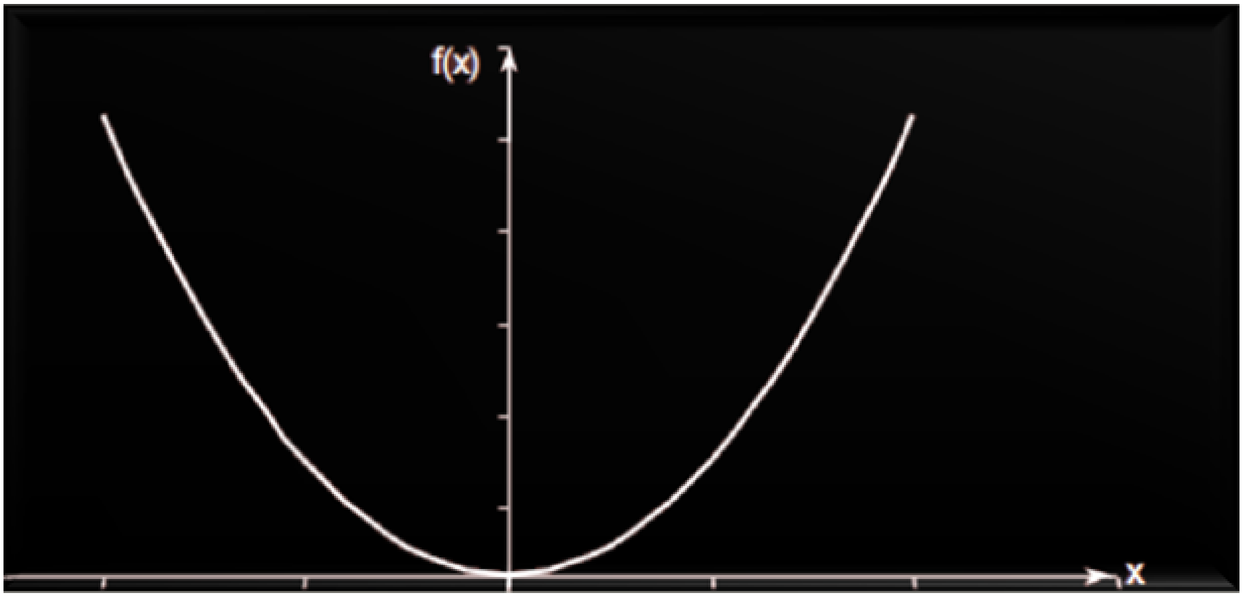
2. Need to find suitable initial guesses (xl ,xu)."sometimes it is impossible to find the initial guesses "

3. No account is taken of the fact that $f\ (\ x1)$ is closer to zero ,it is likely that the root is closer to $x1$.

4. Doesn't work with non-continuous functions even the initial guesses are correct .



- Functions that changes sign but does not have roots.



If the function just touches the x-axis , we will be unable to find the initial guesses.

- **Avoiding pitfalls :**

1. we can check for initial guesses at first . If condition exists continue ,else notify the user of the error.

2. Avoiding non-continuous functions is impossible because we have to check for every point at the given interval.

- **Divergence and convergence** : -It depends on the choice of the initial guesses :

-If initial guesses are correct then, it converges with number of iterations equals :

$$k \geq \log_2 \left| \frac{L_o}{x_l * \mathcal{E}_{es}} \right|$$

-If initial guesses are not correct then, it diverges.

- **FIXED POINT METHOD:**

A) The code of the fixed point method:

```python
# Fixed Point Iteration Method
# Importing math to use sqrt function
import math
from sympy import *
#function of f(x) el3aedya
#we use exp.subs to substitute the x with the real value we will take from user
def func(expr, value, x):
    return expr.subs(x, value)

# Re-writing f(x)=0 to x = g(x)
def funconverte(expr, value, x):
    return expr.subs(x, value)

# Implementing Fixed Point Iteration Method
#x0--> initial guess
#e-->tolerance
#N-->number of itration
#x-->el x elgedeeda
def fixedPointIteration(expr, gx, x0, e, N, x):
    print('\n\n*** FIXED POINT ITERATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        x1 = funconverte(gx, x0, x)
        print('Iteration-%d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1, func(expr, x1, x)))
        x0 = x1
        step = step + 1

        if step > N:
            flag = 0
            break

        condition = abs(func(expr, x1, x)) > e

    if flag == 1:
        # print('\nRequired root is: %0.8f' % x1)
        return "%d ): %.6f" % (step, x1)
    else:
        print('\nNot Convergent.')

# Main code
def mainFunc(function, g_x, N, e, x0):
    x = var('x')  # the possible variable names must be known beforehand...
    expr = sympify(function)
    g_x = sympify(g_x)
    return fixedPointIteration(expr, g_x,x0, e, N,x)
```

B)Sample runs & Analysis :

1) first function :

E^-x-x

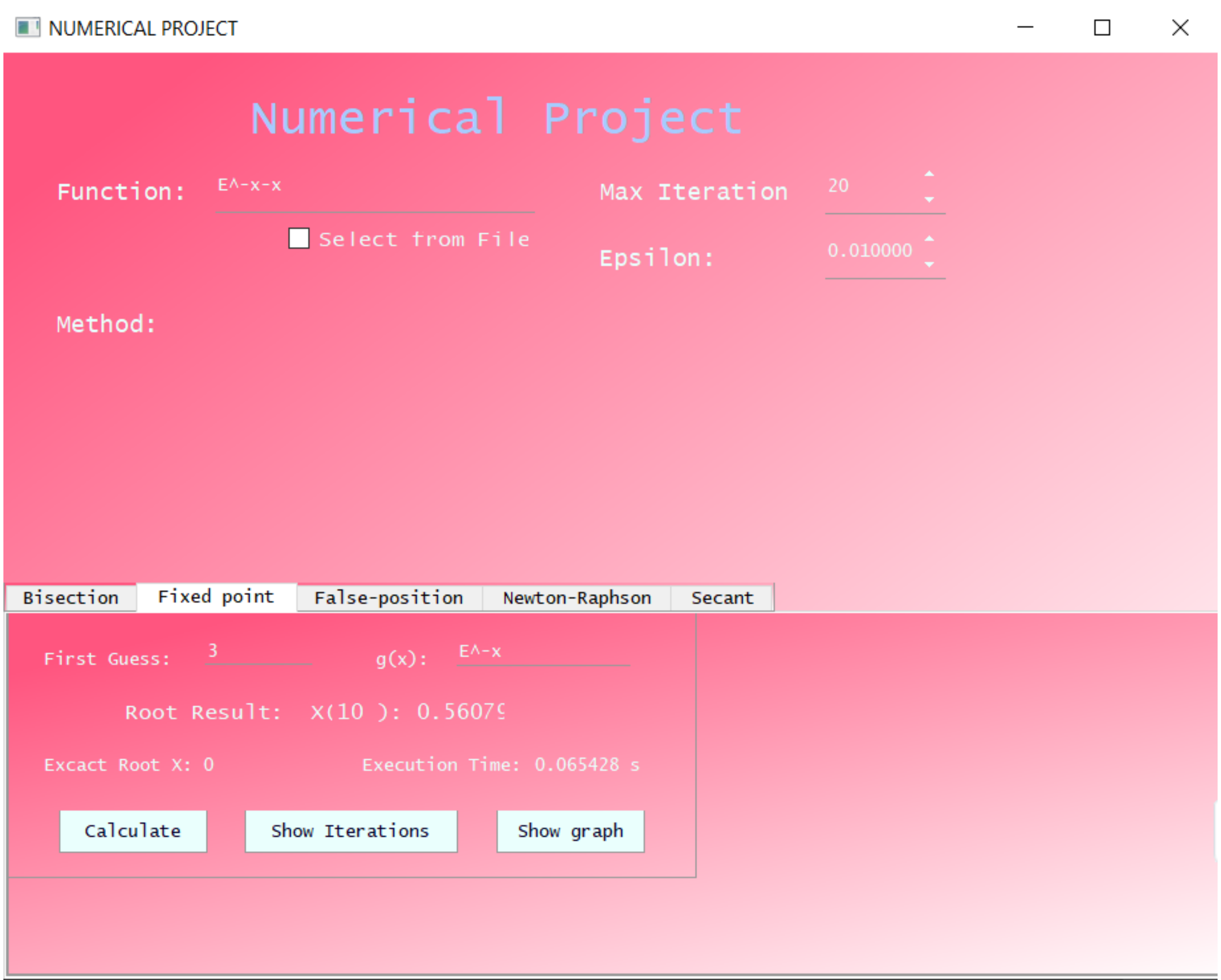
-lower bound: 0

-upper bound:2

-number of itrations:9

-Epsilon:10^-2

magic function: E^-x

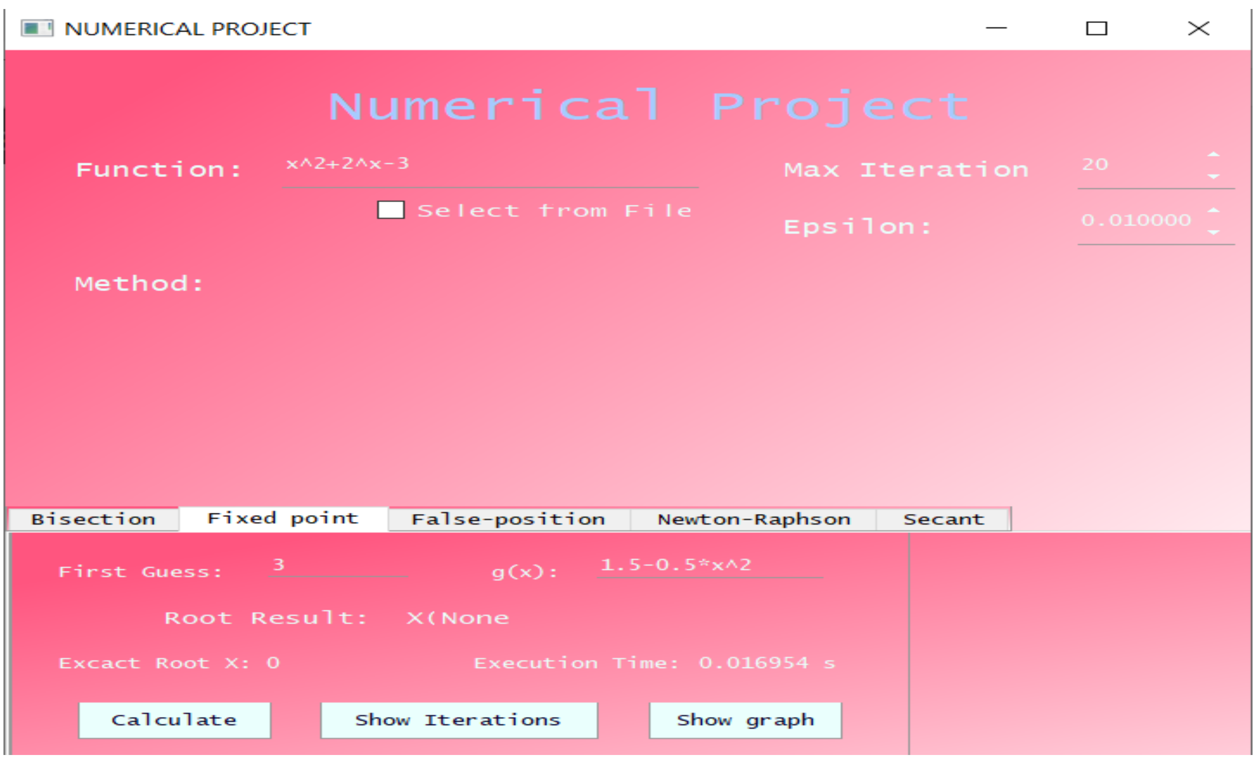


Iterations:

```
*** FIXED POINT ITERATION ***
Iteration-1, x1 = 0.049787 and f(x1) = 0.901645
Iteration-2, x1 = 0.951432 and f(x1) = -0.565244
Iteration-3, x1 = 0.386188 and f(x1) = 0.293455
Iteration-4, x1 = 0.679643 and f(x1) = -0.172845
Iteration-5, x1 = 0.506798 and f(x1) = 0.095624
Iteration-6, x1 = 0.602422 and f(x1) = -0.054937
Iteration-7, x1 = 0.547484 and f(x1) = 0.030919
Iteration-8, x1 = 0.578403 and f(x1) = -0.017610
Iteration-9, x1 = 0.560793 and f(x1) = 0.009963
```

2)second function :

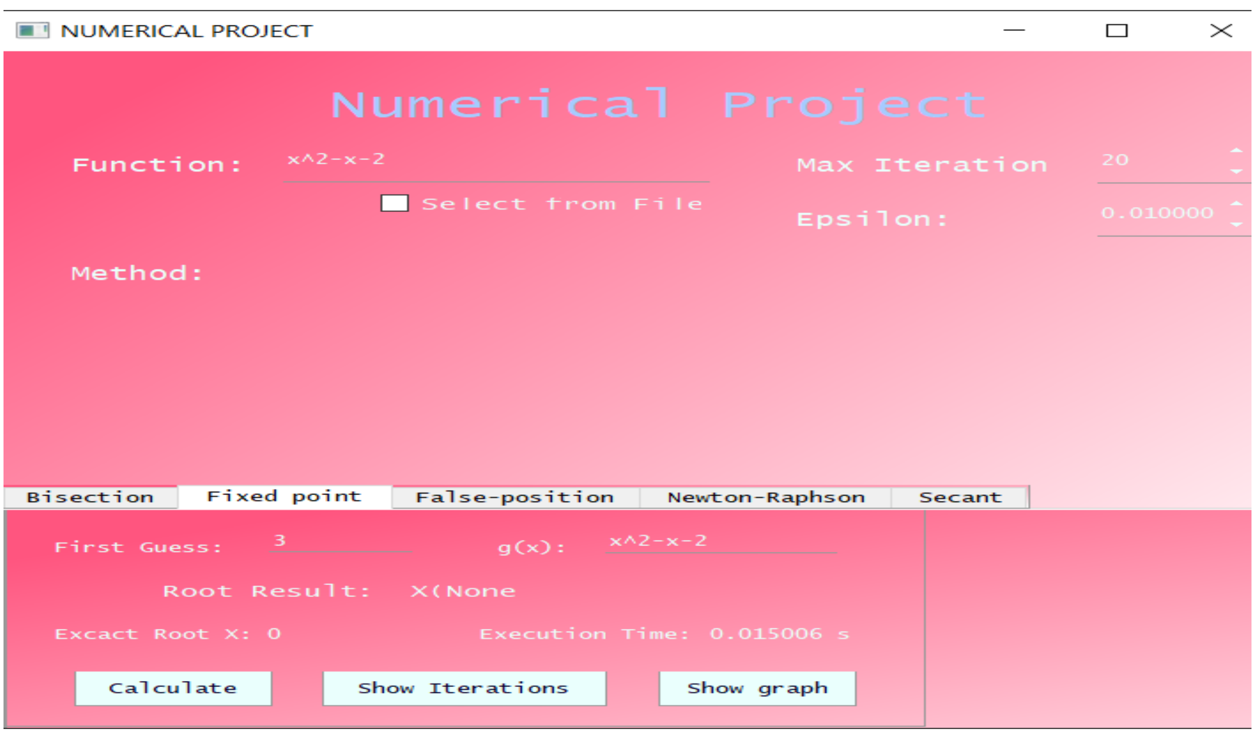
Now we will choose an equation that will diverge:



Iterations:

```
*** FIXED POINT ITERATION ***
Iteration-1, x1 = -3.000000 and f(x1) = 6.125000
Iteration-2, x1 = -3.000000 and f(x1) = 6.125000
Iteration-3, x1 = -3.000000 and f(x1) = 6.125000
Iteration-4, x1 = -3.000000 and f(x1) = 6.125000
Iteration-5, x1 = -3.000000 and f(x1) = 6.125000
Iteration-6, x1 = -3.000000 and f(x1) = 6.125000
Iteration-7, x1 = -3.000000 and f(x1) = 6.125000
Iteration-8, x1 = -3.000000 and f(x1) = 6.125000
Iteration-9, x1 = -3.000000 and f(x1) = 6.125000
Iteration-10, x1 = -3.000000 and f(x1) = 6.125000
Iteration-11, x1 = -3.000000 and f(x1) = 6.125000
Iteration-12, x1 = -3.000000 and f(x1) = 6.125000
Iteration-13, x1 = -3.000000 and f(x1) = 6.125000
Iteration-14, x1 = -3.000000 and f(x1) = 6.125000
Iteration-15, x1 = -3.000000 and f(x1) = 6.125000
Iteration-16, x1 = -3.000000 and f(x1) = 6.125000
Iteration-17, x1 = -3.000000 and f(x1) = 6.125000
Iteration-18, x1 = -3.000000 and f(x1) = 6.125000
Iteration-19, x1 = -3.000000 and f(x1) = 6.125000
Iteration-20, x1 = -3.000000 and f(x1) = 6.125000
```

3)Third Function:



Iterations:

```
*** FIXED POINT ITERATION ***
Iteration-1, x1 = 7.000000 and f(x1) = 40.000000
Iteration-2, x1 = 47.000000 and f(x1) = 2160.000000
Iteration-3, x1 = 2207.000000 and f(x1) = 4868640.000000
Iteration-4, x1 = 4870847.000000 and f(x1) = 23725145626560.000000
Iteration-5, x1 = 23725150497407.000000 and f(x1) = 562882766124587916017532928.000000
Iteration-6, x1 = 562882766124611624237006848.000000 and f(x1) = 316837008400094212218388701986061763188239223924719616.000000
Iteration-7, x1 = 316837008400094212218388701986061763188239223924719616.000000 and f(x1) = 100385689891921378312632548688778251824188071189375211467520615229474709028045860342097
Iteration-8, x1 = 100385689891921378312632548688778251824188071189375211467520615229474709028045860342097678567429997779222528.000000 and f(x1) = 100772867350770053963511317203866
Iteration-9, x1 = 1007728673507700539635113172038675285881841645052077687125157578078235611948605336870275456861393858116301056958181492324208571887581952904868093050549672746797
Iteration-10, x1 = inf and f(x1) = inf
Iteration-11, x1 = inf and f(x1) = inf
Iteration-12, x1 = inf and f(x1) = inf
Iteration-13, x1 = inf and f(x1) = inf
Iteration-14, x1 = inf and f(x1) = inf
Iteration-15, x1 = inf and f(x1) = inf
Iteration-16, x1 = inf and f(x1) = inf
Iteration-17, x1 = inf and f(x1) = inf
Iteration-18, x1 = inf and f(x1) = inf
Iteration-19, x1 = inf and f(x1) = inf
Iteration-20, x1 = inf and f(x1) = inf
```

C)Observation

As the number of iterations increase we reaches value near the exact solution .

D) Pit falls :

- Finding the magical formula that will converge is the only pit fall .

- **Avoiding Pit falls : -**By finding the first derivative then substitute by the initial guess ,If the result is greater than 1 then it will converge .

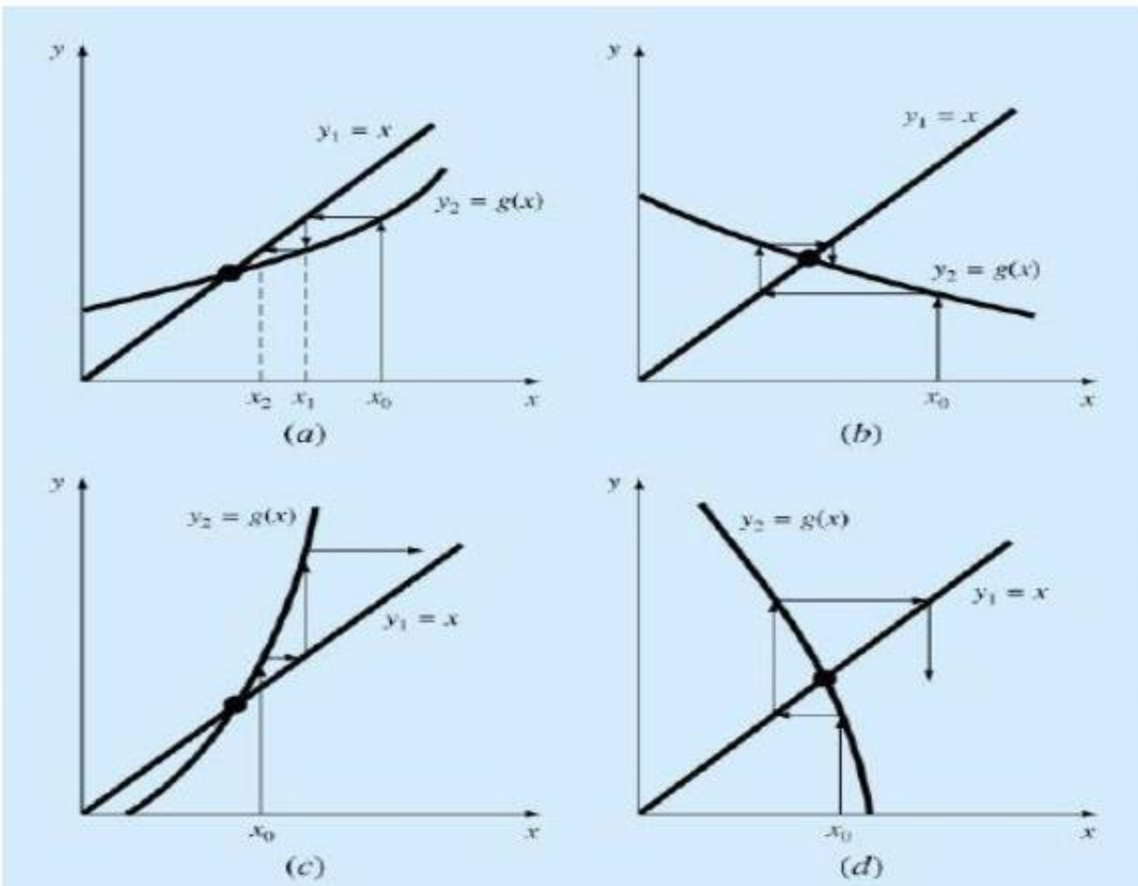
- **Divergence and convergence :**

a. $g'(x) < 1$, $g'(\, x)$ is positive . ------> converge , monotonic.

b. $g'(x) < 1$,$g'(\, x\,)$ is negative ------> converge , oscillate.

c. $g'(x) > 1$,$g'\, (x\,)$ is positive . ------> diverge , monotonic.

d. $g'(x) > 1$,$g'\, (x\,)$ is negative . ------> diverge , monotonic



❖ FALSE POSITION METHOD:

A) The code of the false position method:

```
from sympy import var
from sympy import sympify
from sympy import *
import time

def func(expr, value, x):
    return expr.subs(x, value)


# An example function whose solution
# is determined using Bisection Method.
# The function is x^3 - x^2 + 2

# Prints root of func(x) in interval [a, b]
def regulaFalsi(a, b, expr, x,MAX_ITER, EPSILON):
    if func(expr, a, x) * func(expr, b, x) >= 0:
        print("You have not assumed right a and b")
        return -1
    c = a
    # Initialize result
    i = 1
    old_c = int(0)
    start_time = time.time()
    step=0
    while i < MAX_ITER:
        # Find the point that touches x a/xis

        c = (a * func(expr, b, x) - b * func(expr, a, x)) / (func(expr, b, x) - func(expr,
a, x))
        print('Iteration-%d, Xr = %0.6f and f(Xr) = %0.6f' % (step, c, func(expr, c,x)))
        step=step+1
        if abs(abs(c - old_c) / c) < EPSILON:
            break
        # Check if the above found point is root
        if func(expr, c, x) == 0:
            break
        # Decide the side to repeat the steps
        elif func(expr, c, x) * func(expr, a, x) < 0:
            b = c
        else:
            a = c
        old_c = c
        i = i + 1
    end_time = time.time()
    t3 = end_time - start_time
    print("execution time for RegularFalse=", "%.6f" " sec" % t3)

    return "%d ): %.6f" % (i, c)

def mainFunc(function, maxIteration, epsilon, a, b):
    x = var('x')  # the possible variable names must be known beforehand...
    expr = sympify(function)
    return regulaFalsi(a, b, expr, x, maxIteration, epsilon)
```

B)Sample runs & Analysis :

1) first function :

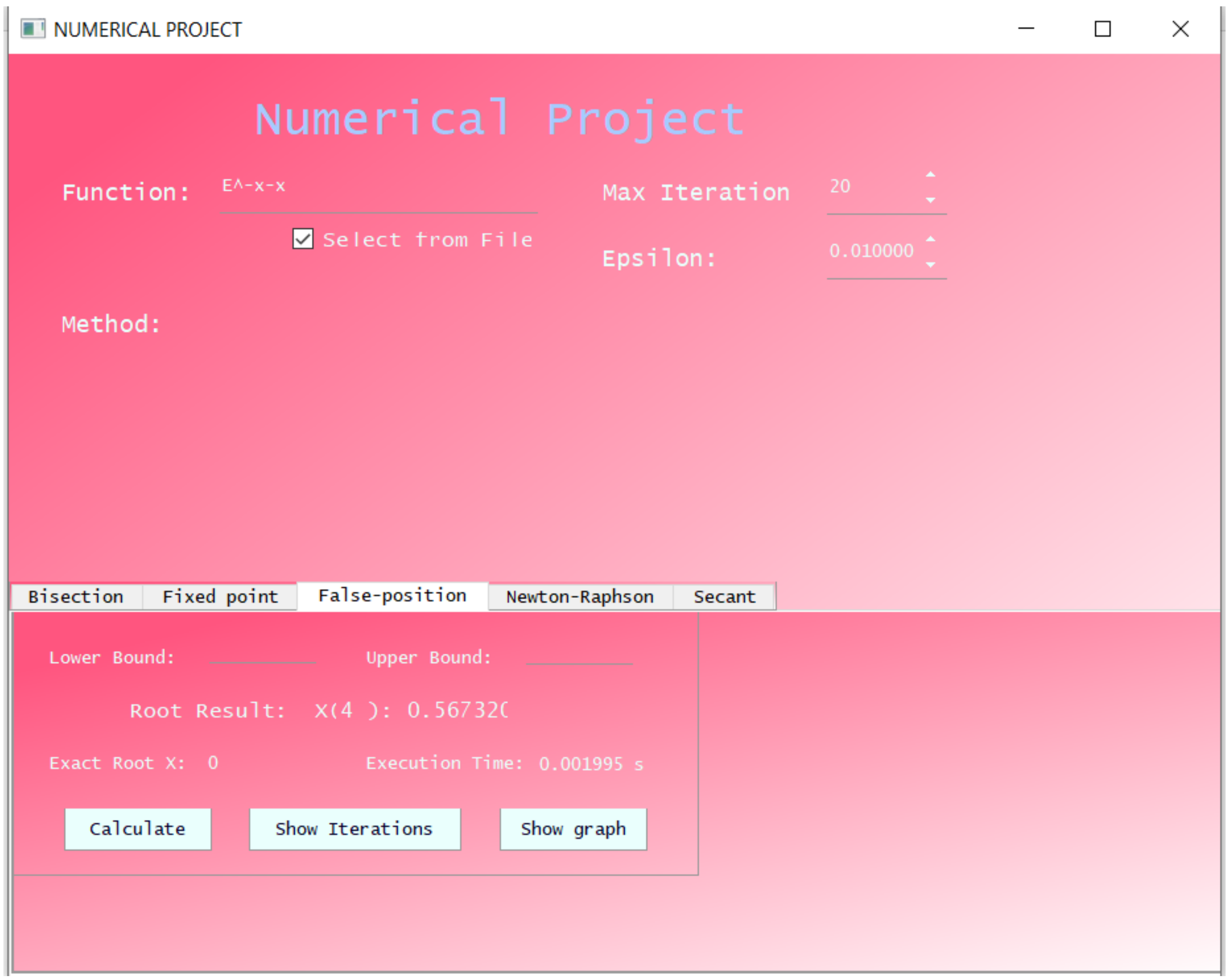
E^-x-x

-lower bound: 0

-upper bound:2

-number of itrations:9

-Epsilon:10^-2

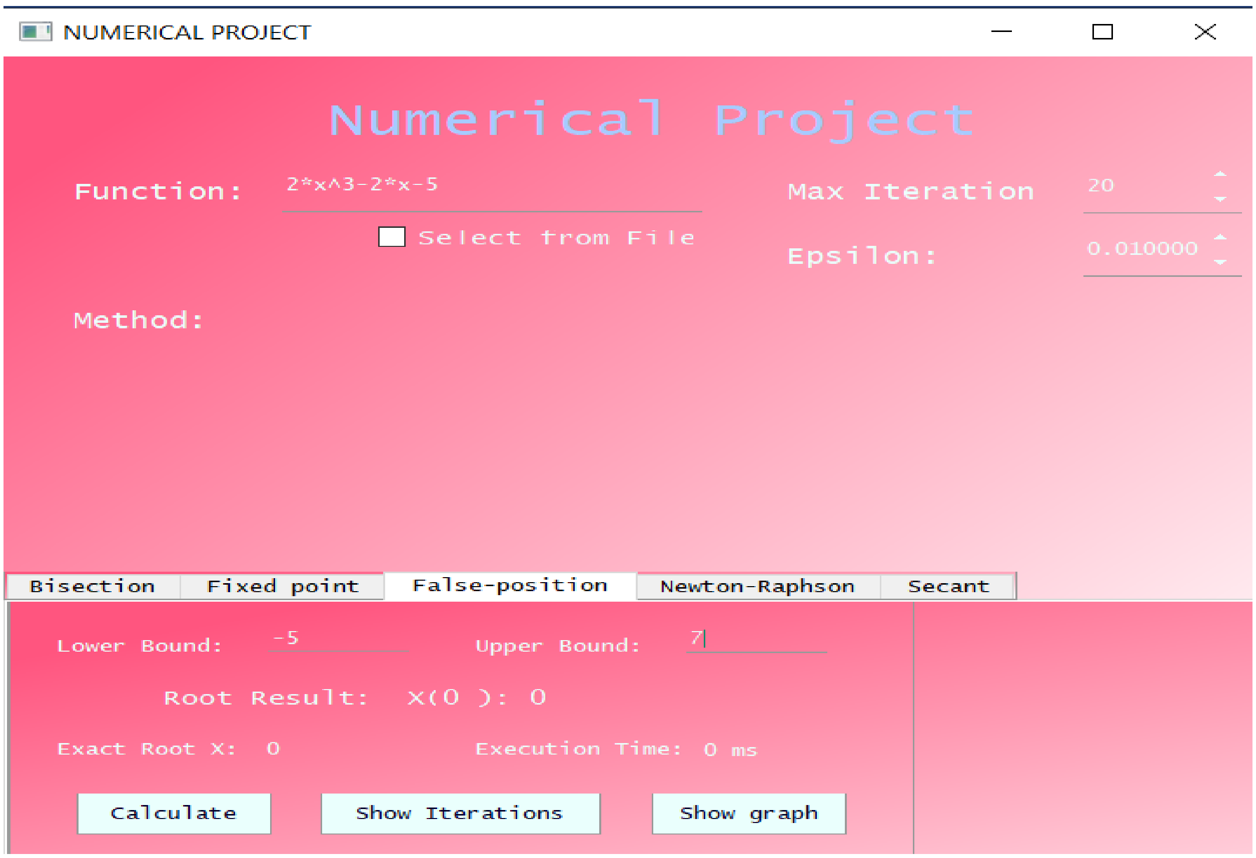


ROOT RESULT: is at x4 =0.56732

Iterations:

```
Iteration-0, Xr = 0.698162 and f(Xr) = -0.200663
Iteration-1, Xr = 0.581480 and f(Xr) = -0.022410
Iteration-2, Xr = 0.568735 and f(Xr) = -0.002494
Iteration-3, Xr = 0.567320 and f(Xr) = -0.000277
execution time for RegularFalse= 0.000997 sec
```

2)second function:

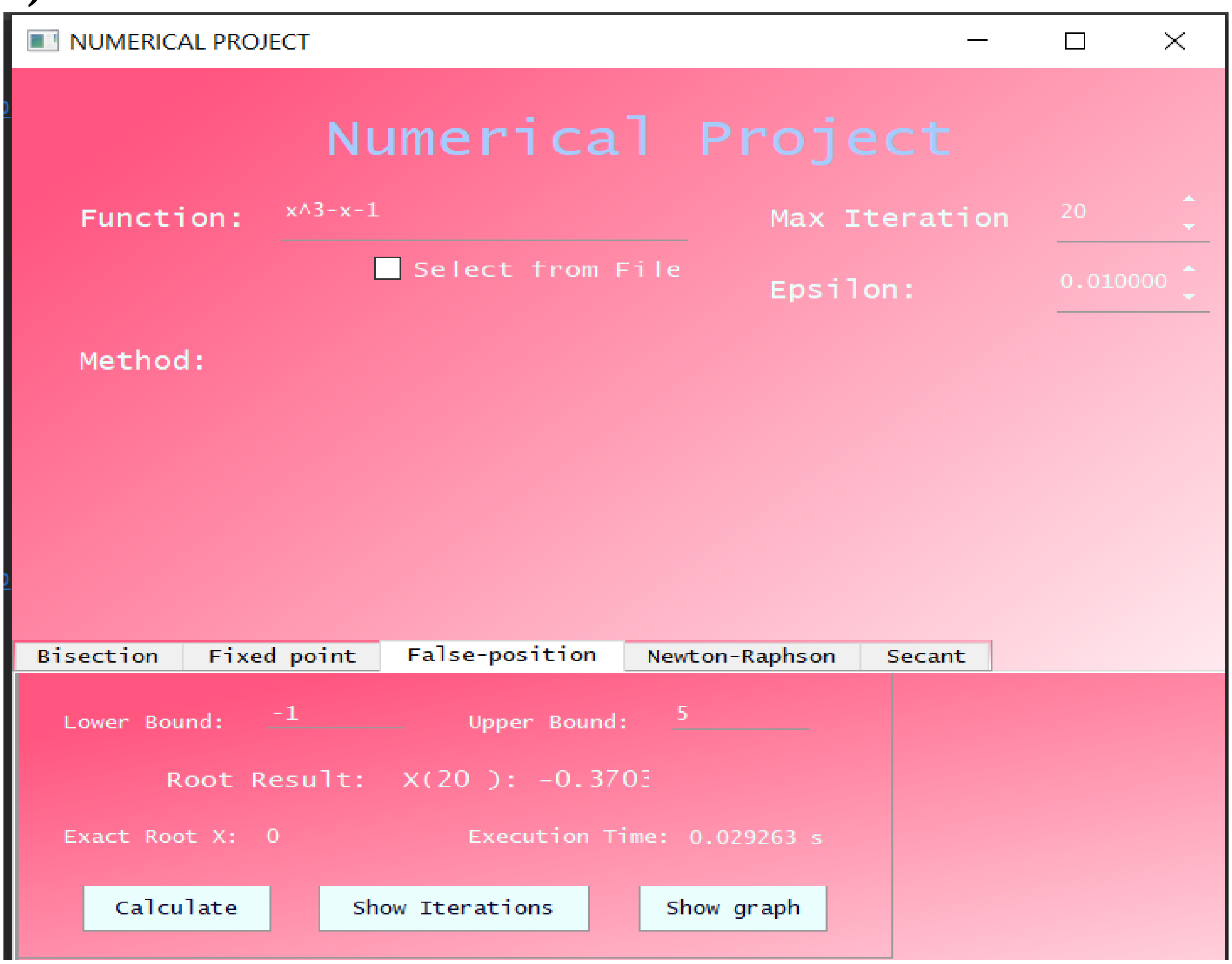


Iterations :

```
The value of the root is :  1.6006
execution time for NewtonRaphson= 0.004989 sec
Iteration-0, Xr = -1.776316 and f(Xr) = -12.656979
Iteration-1, Xr = -1.612878 and f(Xr) = -10.165646
Iteration-2, Xr = -1.483581 and f(Xr) = -8.563599
Iteration-3, Xr = -1.376041 and f(Xr) = -7.458956
Iteration-4, Xr = -1.283409 and f(Xr) = -6.661088
Iteration-5, Xr = -1.201504 and f(Xr) = -6.066001
Iteration-6, Xr = -1.127588 and f(Xr) = -5.612176
Iteration-7, Xr = -1.059772 and f(Xr) = -5.260953
Iteration-8, Xr = -0.996698 and f(Xr) = -4.986859
Iteration-9, Xr = -0.937354 and f(Xr) = -4.772473
Iteration-10, Xr = -0.880965 and f(Xr) = -4.605503
Iteration-11, Xr = -0.826922 and f(Xr) = -4.477054
Iteration-12, Xr = -0.774736 and f(Xr) = -4.380546
Iteration-13, Xr = -0.724008 and f(Xr) = -4.311016
Iteration-14, Xr = -0.674406 and f(Xr) = -4.264660
Iteration-15, Xr = -0.625650 and f(Xr) = -4.238506
Iteration-16, Xr = -0.577498 and f(Xr) = -4.230200
Iteration-17, Xr = -0.529743 and f(Xr) = -4.237835
Iteration-18, Xr = -0.482204 and f(Xr) = -4.259837
execution time for RegularFalse= 0.093667 sec
```

3-Third equation:

3)Third function:



Iterations:

```
Iteration-0, Xr = -0.950000 and f(Xr) = -0.907375
Iteration-1, Xr = -0.904975 and f(Xr) = -0.836181
Iteration-2, Xr = -0.863771 and f(Xr) = -0.780689
Iteration-3, Xr = -0.825553 and f(Xr) = -0.737093
Iteration-4, Xr = -0.789692 and f(Xr) = -0.702770
Iteration-5, Xr = -0.755701 and f(Xr) = -0.675868
Iteration-6, Xr = -0.723195 and f(Xr) = -0.655044
Iteration-7, Xr = -0.691864 and f(Xr) = -0.639315
```

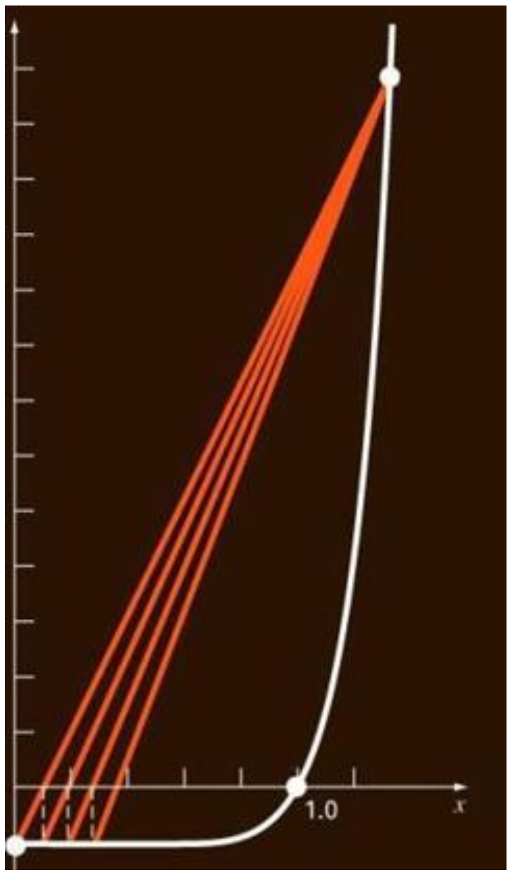
```
Iteration-8, Xr = -0.661449 and f(Xr) = -0.627945
Iteration-9, Xr = -0.631731 and f(Xr) = -0.620383
Iteration-10, Xr = -0.602523 and f(Xr) = -0.616213
Iteration-11, Xr = -0.573661 and f(Xr) = -0.615123
Iteration-12, Xr = -0.544999 and f(Xr) = -0.616879
Iteration-13, Xr = -0.516402 and f(Xr) = -0.621307
Iteration-14, Xr = -0.487750 and f(Xr) = -0.628286
Iteration-15, Xr = -0.458929 and f(Xr) = -0.637729
Iteration-16, Xr = -0.429830 and f(Xr) = -0.649583
Iteration-17, Xr = -0.400351 and f(Xr) = -0.663817
Iteration-18, Xr = -0.370394 and f(Xr) = -0.680421
execution time for RegularFalse= 0.027268 sec
```

C)Observation

As the number of iterations increase we reaches value near the exact solution .

D) Pit falls :

- Works well but not always .
- Cannot detect continuous functions .
-Initial guesses are required .
-one of bounds might get stuck .



• Avoiding Pit falls :

One way to mitigate the “one-sided”
nature of the false position is to have
the algorithm detect when one of the bounds is stuck.
If this occurs,
then the original formula of bisection can be used.

❖ NEWTON RAPTHSON METHOD:

A)the code for newton Raphson method

```
from sympy import var
from sympy import sympify
from sympy import *
import time

def func(expr, value, x):
    return expr.subs(x, value)


# An example function whose solution
# is determined using Bisection Method.
# The function is x^3 - x^2 + 2

# Prints root of func(x) in interval [a, b]
def regulaFalsi(a, b, expr, x,MAX_ITER, EPSILON):
    if func(expr, a, x) * func(expr, b, x) >= 0:
        print("You have not assumed right a and b")
        return -1
    c = a
    # Initialize result
    i = 1
    old_c = int(0)
    start_time = time.time()
    step=0
    while i < MAX_ITER:
        # Find the point that touches x a/xis

        c = (a * func(expr, b, x) - b * func(expr, a, x)) / (func(expr, b, x) - func(expr,
a, x))
        print('Iteration-%d, Xr = %0.6f and f(Xr) = %0.6f' % (step, c, func(expr, c,x)))
        step=step+1
        if abs(abs(c - old_c) / c) < EPSILON:
            break
        # Check if the above found point is root
        if func(expr, c, x) == 0:
            break
        # Decide the side to repeat the steps
        elif func(expr, c, x) * func(expr, a, x) < 0:
            b = c
        else:
            a = c
        old_c = c
        i = i + 1
    end_time = time.time()
    t3 = end_time - start_time
    print("execution time for RegularFalse=", "%.6f" " sec" % t3)

    return "%d ): %.6f" % (i, c)

def mainFunc(function, maxIteration, epsilon, a, b):
    x = var('x')  # the possible variable names must be known beforehand...
    expr = sympify(function)
    return regulaFalsi(a, b, expr, x, maxIteration, epsilon)
```

B)Sample runs & Analysis :

1) first function :

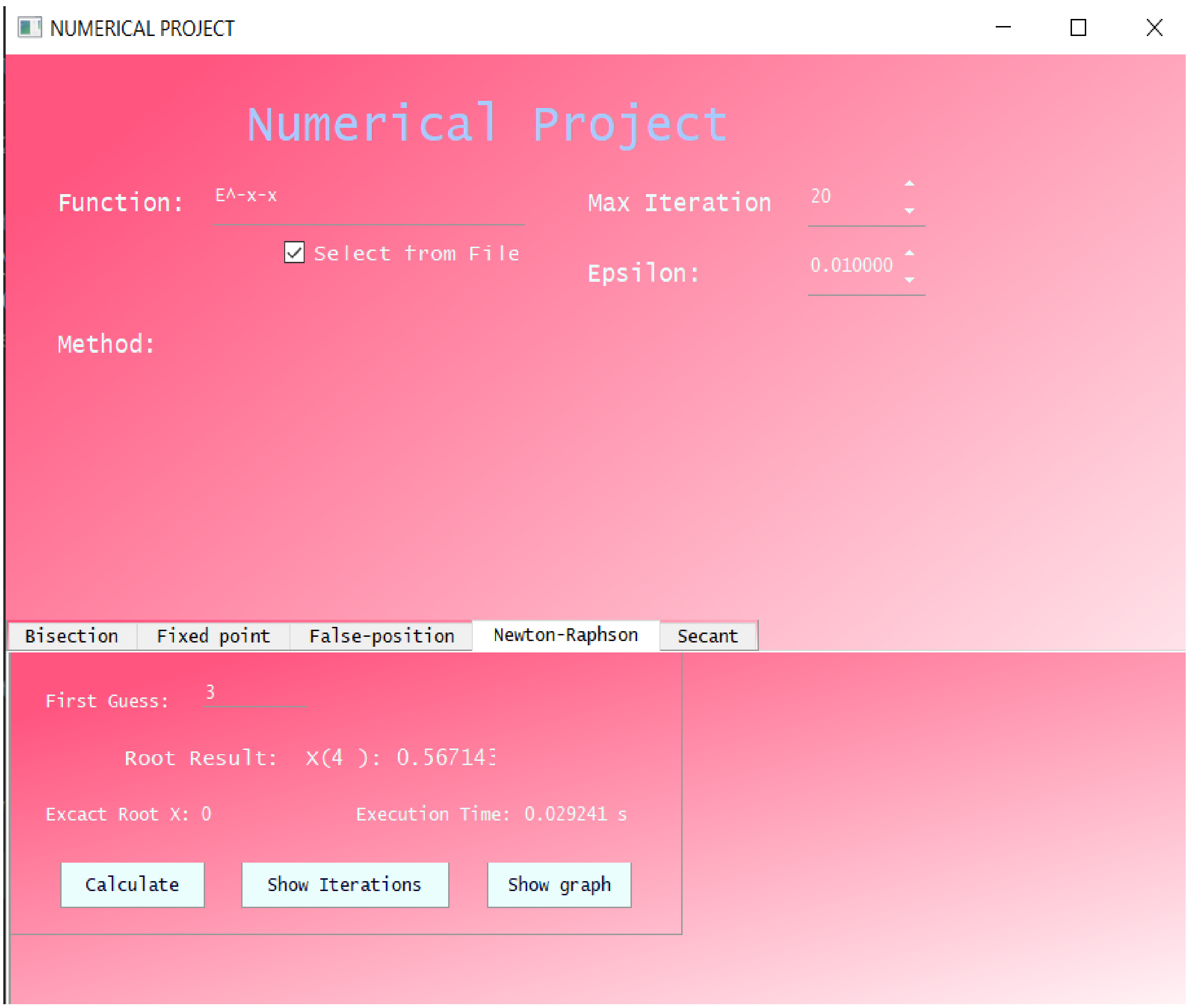
- E^-x-x

-lower bound: 0

-upper bound:2

-number of itrations:9

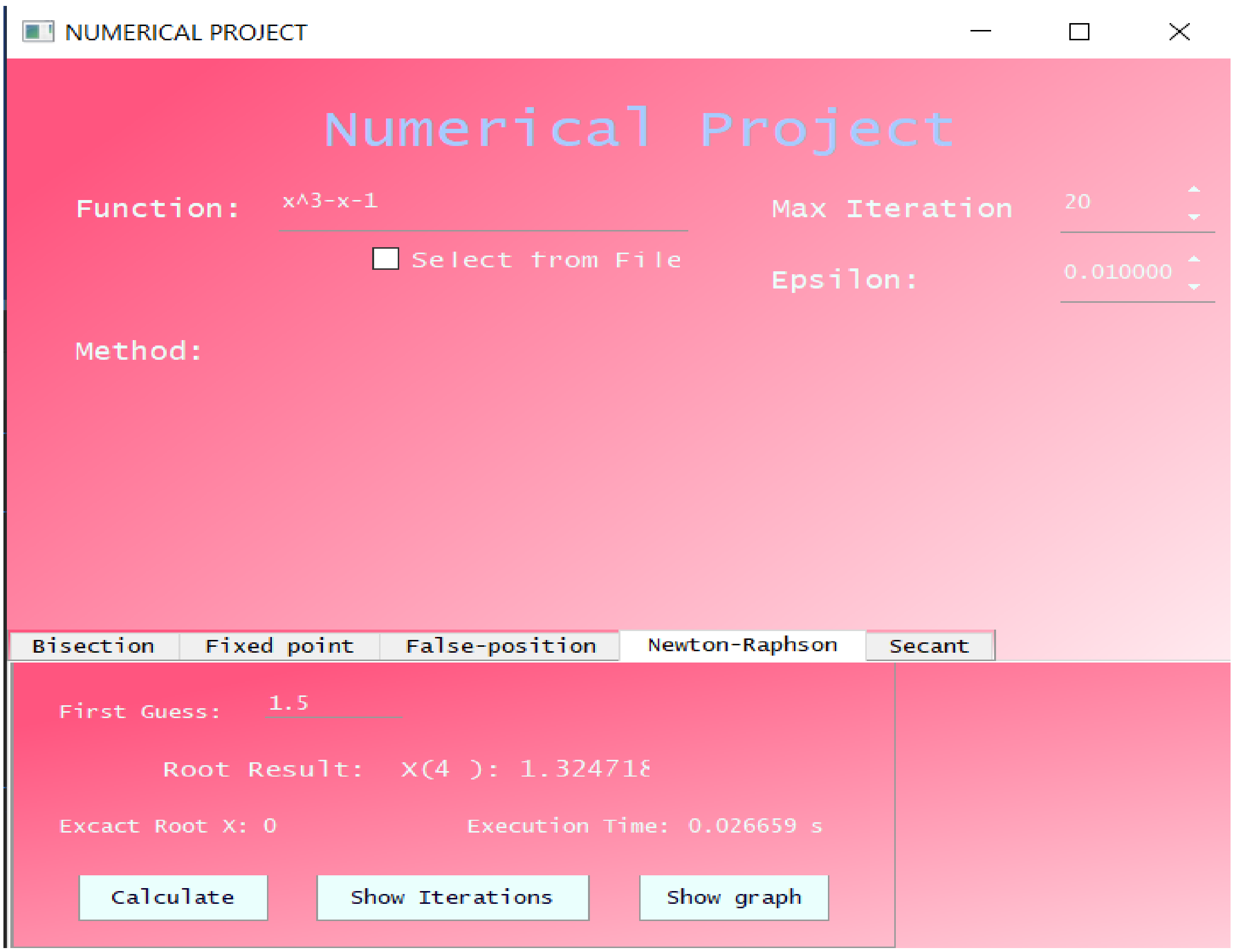
-Epsilon:10^-2



Iterations:

```
Iteration-1, xi = 0.000000 and f(xi) = 1.000000
Iteration-2, xi = 0.500000 and f(xi) = 0.106531
Iteration-3, xi = 0.566311 and f(xi) = 0.001305
The value of the root is :  0.5671
execution time for NewtonRaphson= 0.003960 sec
```

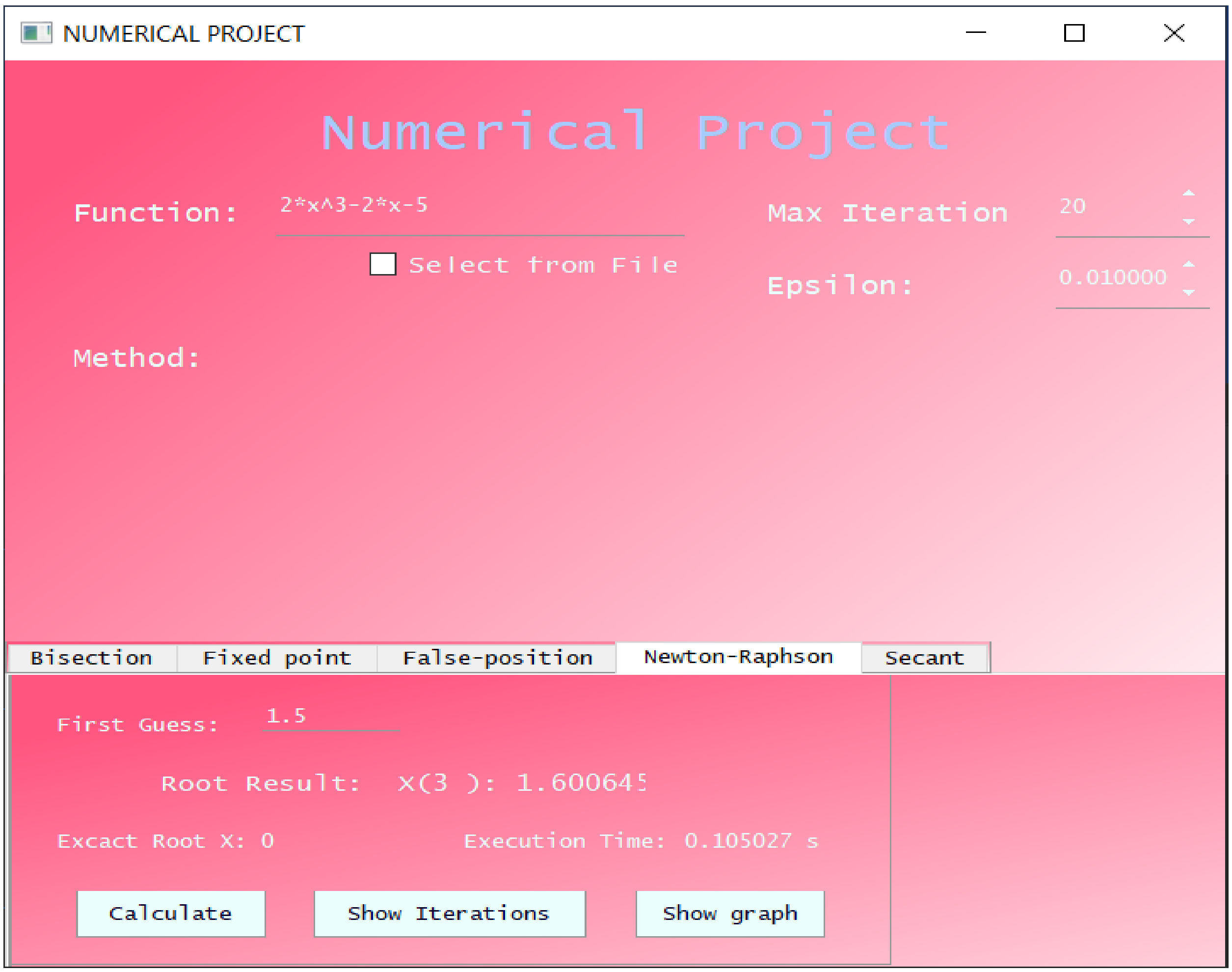
2)second function:



Iterations:

```
Iteration-1, xi = 1.500000 and f(xi) = 0.875000
Iteration-2, xi = 1.347826 and f(xi) = 0.100682
Iteration-3, xi = 1.325200 and f(xi) = 0.002058
The value of the root is :  1.3247
execution time for NewtonRaphson= 0.003988 sec
```

3)Third function:



Iterations:

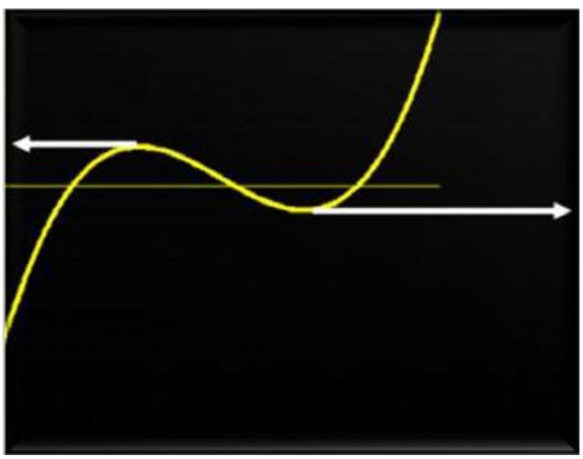
```
Iteration-1, xi = 1.500000 and f(xi) = -1.250000
Iteration-2, xi = 1.608696 and f(xi) = 0.108901
The value of the root is :  1.6006
execution time for NewtonRaphson= 0.004989 sec
```

C)Observation

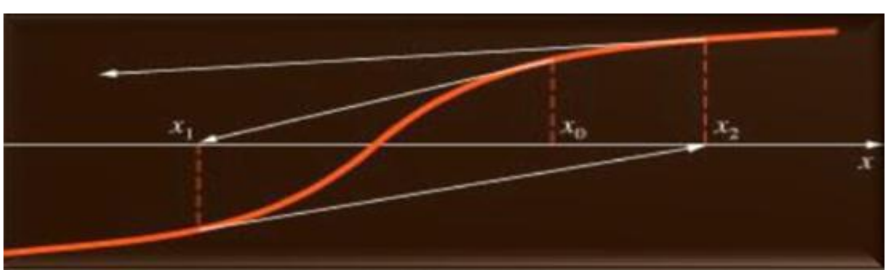
As the number of iterations increase we reaches value near the exact solution .

D)Pit Falls :

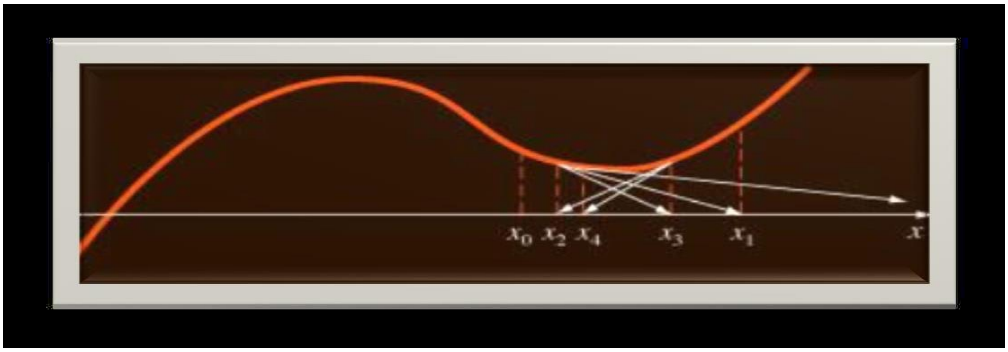
- Division by zero



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



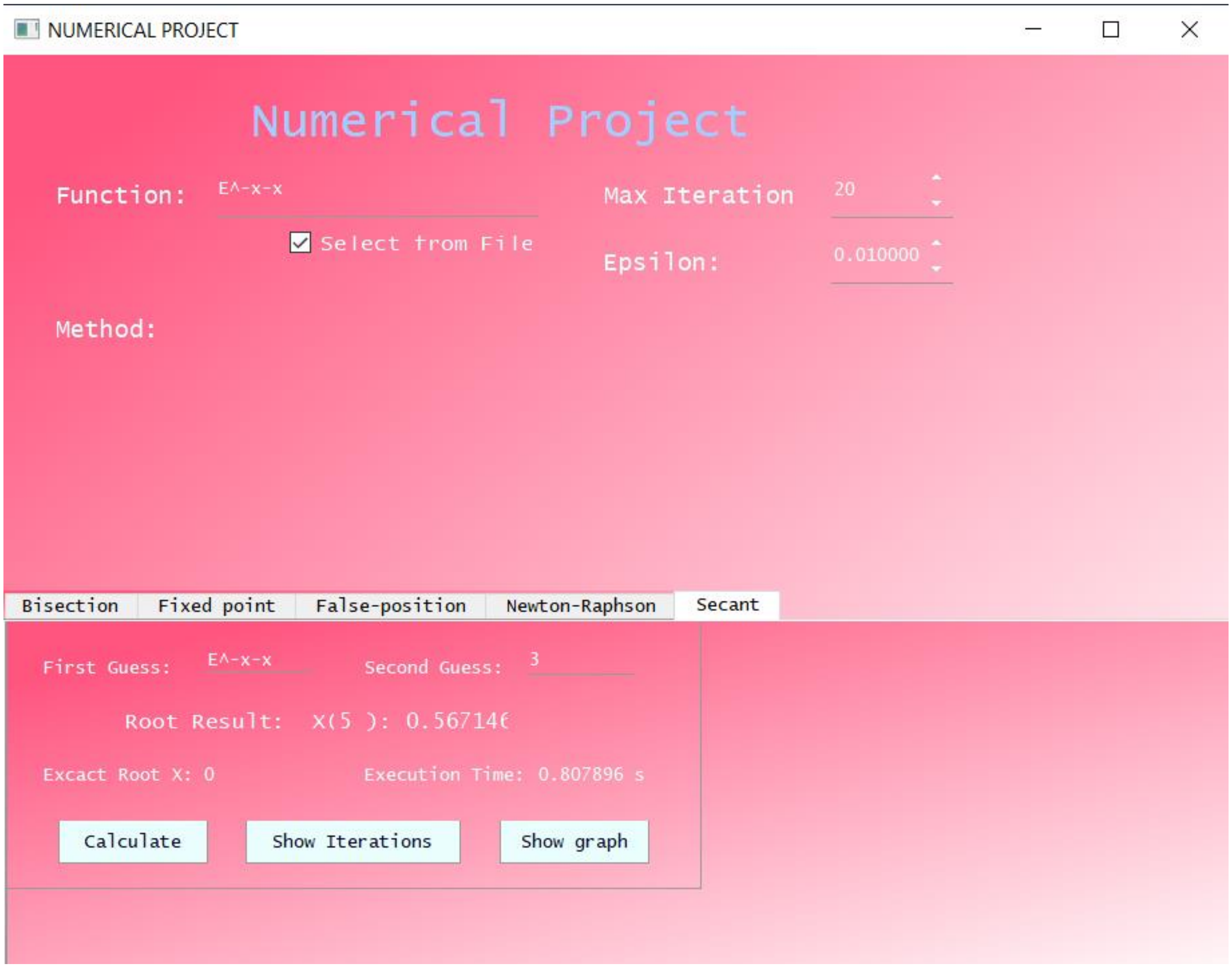
-A local maximum or minimum causes oscillations.



❖ SECANT METHOD:

```python
from sympy import *
import time

# Defining Function
def f(y, Function):
    x = var('x')
    return Function.subs(x, y)


# Implementing Secant Method

def secant(x0, x1, e, N, Function):
    print('\n\n*** SECANT METHOD IMPLEMENTATION *')
    step = 1
    condition = True
    start_time = time.time()
    while condition:
        if f(x0, Function) == f(x1, Function):
            print('Divide by zero error!')
            break

        x2 = x0 - (x1 - x0) * f(x0, Function) / (f(x1, Function) - f(x0, Function))
        fx = f(x2, Function)
        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (step, x2, f(x2, Function)))
        x0 = x1
        x1 = x2
        step = step + 1

        if step > N:
            print('Not Convergent!')
            break

        condition = abs((x1 - x0) / x1) >= e
    end_time = time.time()
    t4 = end_time - start_time
    print("execution time for Secant=", "%.6f" " sec" % t4)
    return "%d ): %.6f" % (step, x2)


def mainFunc(expr, N, e, x0, x1):
    Function = sympify(expr)
    x = var('x')
    sol = solve(Function, x)
    if sol == LambertW(1):
        sol = 0.5671432904097

    # Starting Secant Method
    return secant(x0, x1, e, N, Function)
```

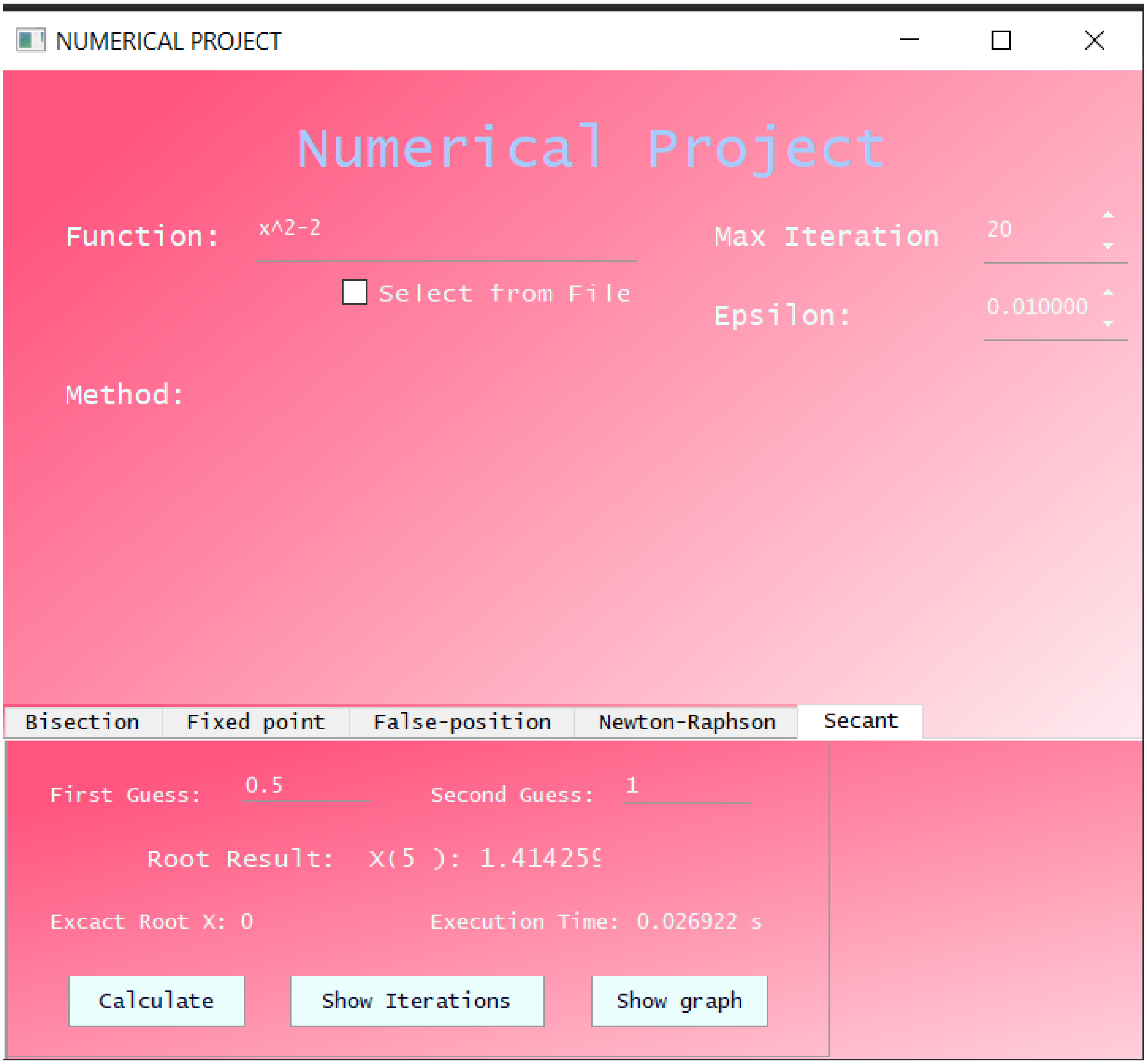
1)first function:



Iterations:

```
Iteration-2, x2 = 0.541172 and f(x2) = 0.040893
Iteration-3, x2 = 0.567749 and f(x2) = -0.000949
Iteration-4, x2 = 0.567146 and f(x2) = -0.000004
execution time for Secant= 0.001991 sec
```

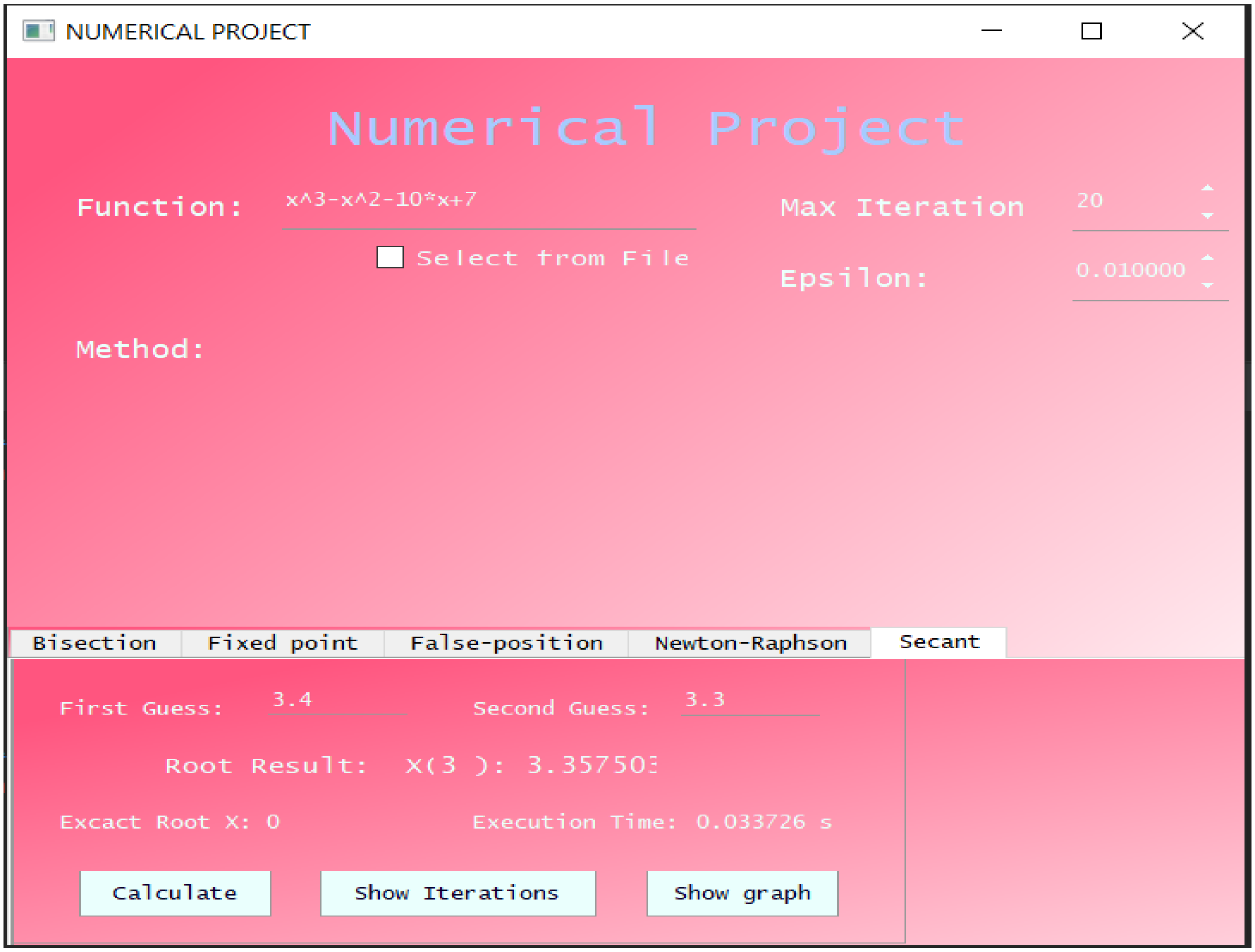
2)second equation:



Iterations:

```
Iteration-2, x2 = 1.375000 and f(x2) = -0.109375
Iteration-3, x2 = 1.410959 and f(x2) = -0.009195
Iteration-4, x2 = 1.414259 and f(x2) = 0.000130
execution time for Secant= 0.002991 sec
```

3)third equation:



Iterations:

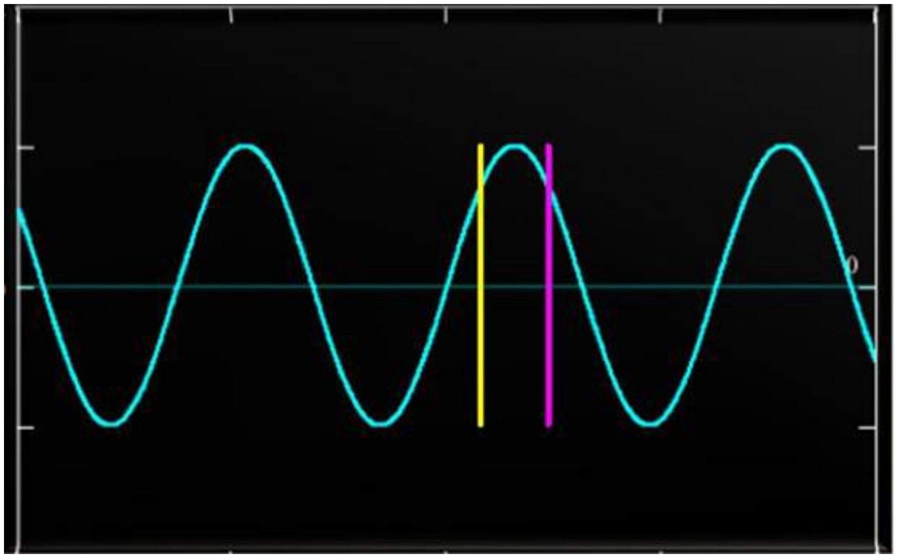
```
*** SECANT METHOD IMPLEMENTATION *
Iteration-1, x2 = 3.356158 and f(x2) = -0.022297
Iteration-2, x2 = 3.357503 and f(x2) = 0.000697
execution time for Secant= 0.003989 sec
```

C)Observation

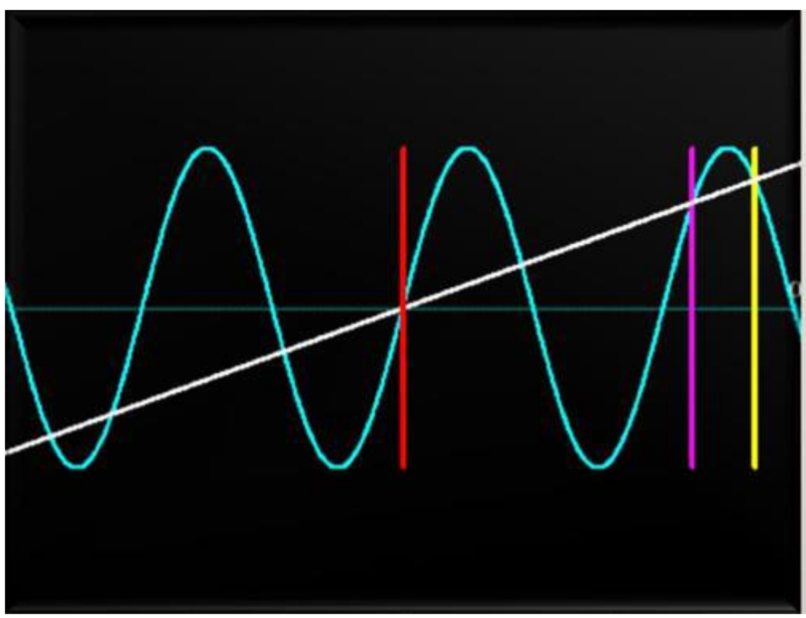
As the number of iterations increase we reaches value near the exact solution .

Pit falls :

- Division by zero .



-Root jumping .



E)Problematic functions:

Problems we have faced :

1-while passing the values (substitution) of

`x`→the value of either the upper or lower bound.

to the main function ,but we have solve it by using **func.sub(x,value).**

2-To make the user have the option to pass the function and its inputs directly from the GUI or to read from file.

We have fixed this by making a check box linked with a condition (if else),and this has worked properly.

F)Data structure:

No data structure used.

G)User manual :

The use can enter the function in the GUI or he can select the function from file then he can choose the desired method entering the initial guess then press calculate to get the results ,the user can also press on graph to see the graph of the function.

NOTE: calculations results are in the output file

NOTE:in each method we print our output in an output file.

Example: