**A project report**

On

**“Finding Roots of an Equation”**

submitted to

**School of Computer Science, Gujarat University**

In partial Fulfilment of the requirement for the degree of

**Master of Science**

In

**Artificial Intelligence and Machine Learning**

By

**Rajan Thakkar – 10017**

**Parth Karkar – 10013**

**Dhyanesh Bhanderi - 10005**

Under Supervision and Guidance of

**Dr. Savita Gandhi**

(Director, School of Computer Science,)



Gujarat University

Ahmedabad-380009

“2020-2022”

**ACKNOWLEDGEMENT**

It is indeed a great pleasure to express my thanks and gratitude to all those who helped me. No serious and lasting achievement or success one can ever achieve without the help of friendly guidance and co-operation of so many people involved in the work.

We are very thankful to our guide **Dr. Savita Gandhi**, the person who makes us to follow the right steps during a project work. we express our deep sense of gratitude to for her guidance. Her valuable and expert suggestions during execution and documentation of our project have indeed helped us a lot.

Thanks to our friends and colleagues who have been a source of inspiration and motivation that helped us during our project work.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Particulars** | **Page No.** |
| **1** | **Introduction** | 1 |
| **2** | **Objectives** | 4 |
| **3** | **Problem Statement** | 5 |
| **4** | **Bisection Method Algorithm** | 6 |
| **5** | **False Position Method Algorithm** | 7 |
| **6** | **Secant Method Algorithm** | 8 |
| **7** | **Newton Raphson Method Algorithm** | 9 |
| **8** | **Time Module** | 10 |
| **9** | **Graph** | 11 |
| **10** | **Result Of program** | 12-18 |
| **11** | **Conclusion** | 19 |
| **12** | **Future Details** | 20 |
| **13** | **References** | 21 |

**Introduction**

* The determination of the roots of an equation is one of the oldest problems in mathematics & there have been many efforts in this regard.
* We learned numerical methods for finding roots of an equations. They represent the values of x that make f(X) equal to zero.
* Thus, we can define the roots of an equation as the value of an equation as the value of x that makes **f(x) = 0.**
* Roots are sometime called the zeros of equation.

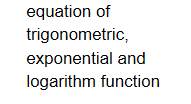
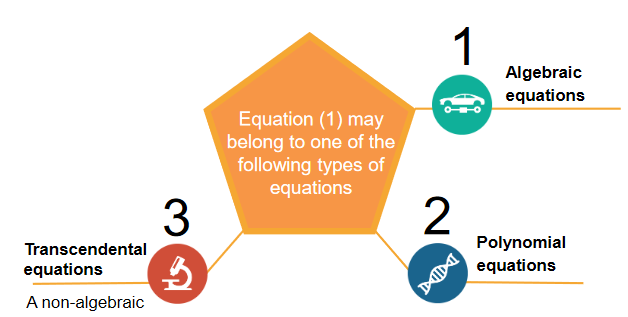
Diagram

Description automatically generated

Root

F(x)=0

**Equation Type**



Example 1: **Algebraic Equation (**An algebraic equation is a mathematical sentence, when two algebraic expressions are related with an equality sign (=).**)**

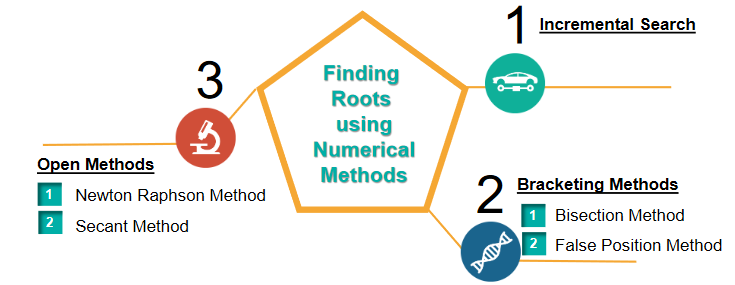
3x+6 -1 = 0

Example 2: **Polynomial Equation (**The equations formed with variables, exponents and coefficients are called as polynomial equations**)**

𝑥2+2𝑥−4=0

Example 3: **Transcendental Equation**

sin2𝑥−3𝑥=0



1. Bracketing Methods (Need two initial estimates that will bracket the root. Always converge.)
   1. Bisection Method
   2. False-Position Method
2. Open Methods (Need one or two initial estimates. May diverge.)
   1. Newton-Raphson Method (Needs the derivative of the function.)
   2. Secant Method

**Objectives**

* Find solutions of Non-Liner equation and Liner Equation (User can Input)
* Derive the formula and follow the algorithms for the solutions of equations using the following methods:
  1. **Bisection**
  2. **False-Position**
  3. **Secant**
  4. **Newton-Raphson**
* Average time Take by all the method in python as well as c++ So, we can predict the performance.
* Graphical representation for better understand

**Problem Statement**

* Write a Python program to find the roots of an equation of any non-liner, liner equation & use four different methods and find the root plus find the average time taken by each method. (Menu driven Program )
  + User will also input the first and second point.
  + User can do new equation (Dynamic input) in the program.
  + Output is view as Tabular format.
  + Graphical representation of Four numerical method
* Note: we also create C++ program for checking which language evaluation is faster.

**Bisection Method Algorithm**

1. Start.
2. Define function f(X).
3. Choose initial guesses x0 and x1 such that f(x0)f(x1)<0.
4. Choose pre specified tolerable error e.(0.0001->4 significant digit)
5. Calculate new approximated root as x2 = (x0 + x1)/2
6. Calculate f(x0)f(x2)
   1. if f(x0)f(x2) < 0 then x0 = x0 and x1 = x2
   2. if f(x0)f(x2) > 0 then x0 = x2 and x1 = x1
   3. if f(x0)f(x2) = 0 then go to (8)
7. if |f(x2)| > e then go to (5) otherwise go to (8)
8. Display x2 as root
9. Stop

**Diagram

Description automatically generated**

**False-Position Method Algorithm**

1. Start
2. Define function f(x)
3. Choose initial guesses x0 and x1 such that f(x0)f(x1) < 0
4. Choose pre-specified tolerable error e.
5. Calculate new approximated root as:

x2 = x1 \* f(x2) – x2\*f(x1) / (f(x2) – f(x1))

1. Calculate f(x0)f(x2)
   1. if f(x0)f(x2) < 0 then x0 = x0 and x1 = x2
   2. if f(x0)f(x2) > 0 then x0 = x2 and x1 = x1
   3. if f(x0)f(x2) = 0 then go to (8)
2. if |f(x2)|>e then go to (5) otherwise go to (8)
3. Display x2 as root.
4. Stop

Diagram

Description automatically generated

**Secant Method Algorithm**

1. Start
2. Define function as f(x)
3. Input initial guesses (x0 and x1), tolerable error (e) and maximum iteration (N)
4. Initialize iteration counter i = 1
5. If f(x0) = f(x1) then print "Mathematical Error" and go to (11) otherwise go to (6)
6. Calculate x2 = x1 - (x1-x0) \* f(x1) / ( f(x1) - f(x0) )
7. Increment iteration counter i = i + 1
8. If i >= N then print "Not Convergent" and goto (11) otherwise goto (9)
9. If |f(x2)| > e then set x0 = x1, x1 = x2 and goto (5) otherwise goto (10)
10. Print root as x2
11. Stop

Chart, line chart

Description automatically generated

**Newton Raphson Method Algorithm**

1. Start
2. Define function as f(x)
3. Define first derivative of f(x) as g(x)
4. Input initial guess (x0), tolerable error (e) and maximum iteration (N
5. Initialize iteration counter i = 1
6. If g(x0) = 0 then print "Mathematical Error" and goto (12) otherwise goto (7)
7. Calculate x1 = x0 - f(x0) / g(x0)
8. Increment iteration counter i = i + 1
9. If i >= N then print "Not Convergent" and go to (12) otherwise go to (10)
10. If |f(x1)| > e then set x0 = x1 and go to (6) otherwise go to (11)
11. Print root as x1
12. Diagram

    Description automatically generatedStop

**Time Module**

* Time Module to calculate the execution time of every method every iteration and after calculate average of all iteration of every method.
* We use perf\_counter() method to calculate the time.
  + Basically, it will work like counter when we start function than we called this function and when function over then we store value of end time.
  + Then we difference two of them and we find the total time take by that function

import time

start\_time = time.perf\_counter()

if f(x1) \* f(x2) < 0.0:

...

else:

…

end\_time = time.perf\_counter()

diff=end\_time - start\_time

print(f"Execution Time is : {diff:0.6f} seconds")

**Graph Plotting**

import matplotlib.pyplot as plt

# matplotlib library which have pyplot module so that we can use graph in python

def graph(ls):

plt.plot(\*zip(\*sorted(ls.items())))

# sorted() sorted by key, return a list of tuples

# Zip() unpack a list of pairs into two tuples

# items() Function show keys and values

plt.xlabel('x - axis(New point)')

plt.ylabel('y - axis(New point value)')

plt.title('Numerical Methods ')

plt.show() #Show() use for display the graph

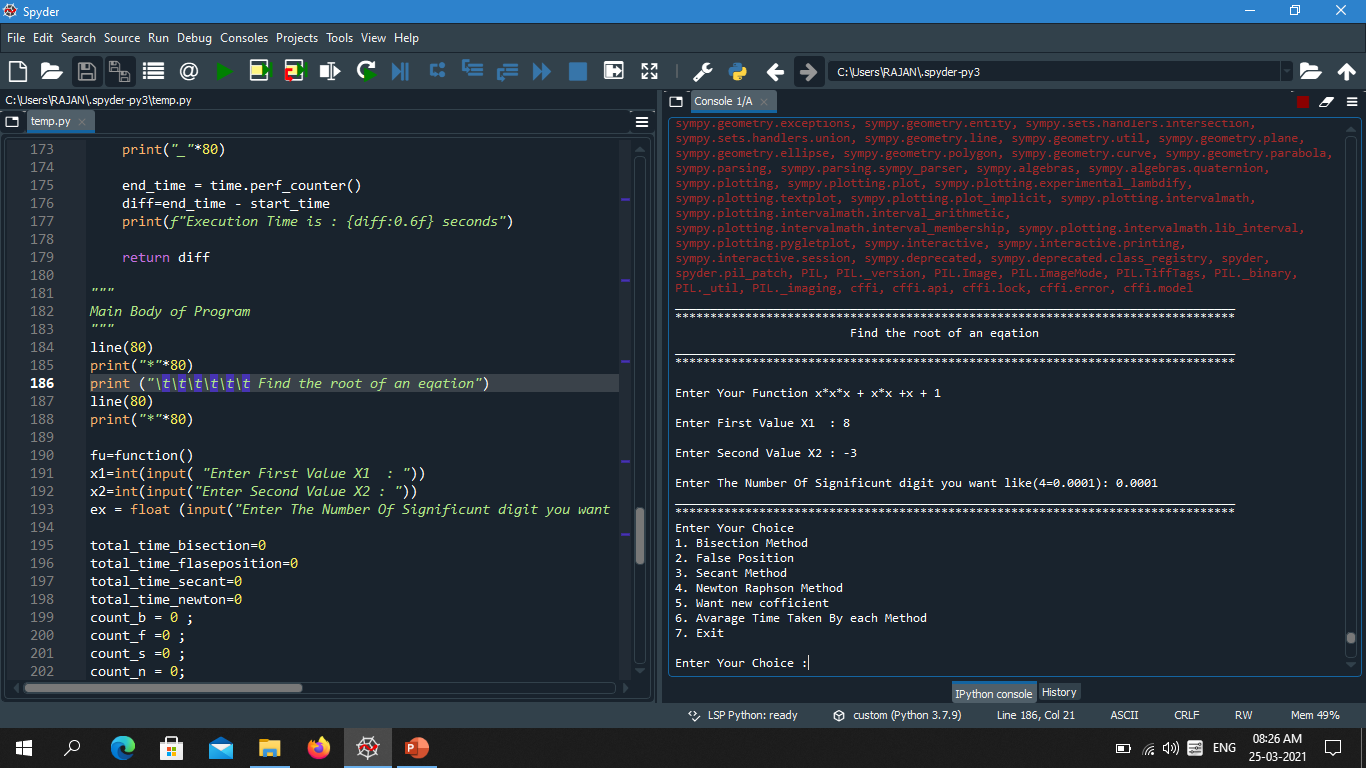
Above is the function which have one argument that is dictionary (ls) which have new point as key and their point’s function evolution as value.

ls = dict()

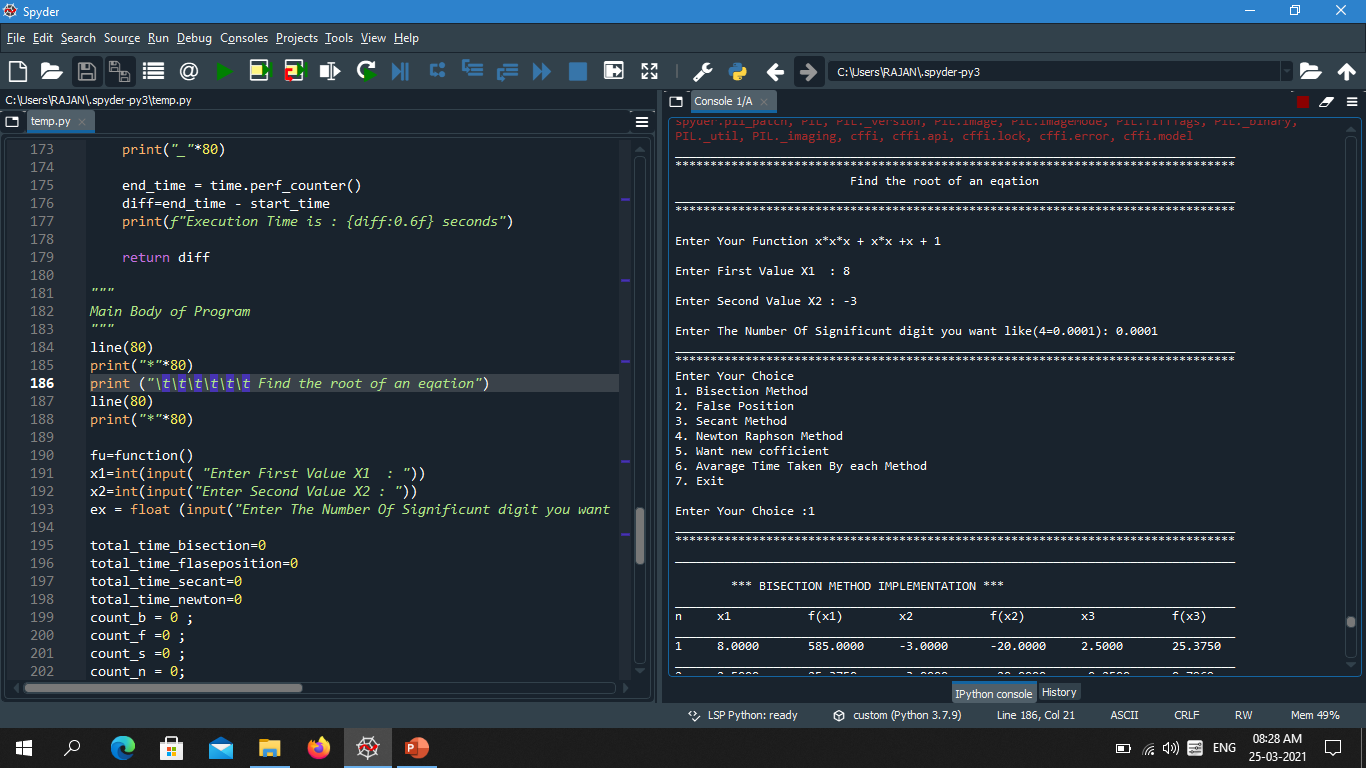
ls.update({x3:f(fu,x3)})

so, this is the way to show the graph of each of the numerical method.

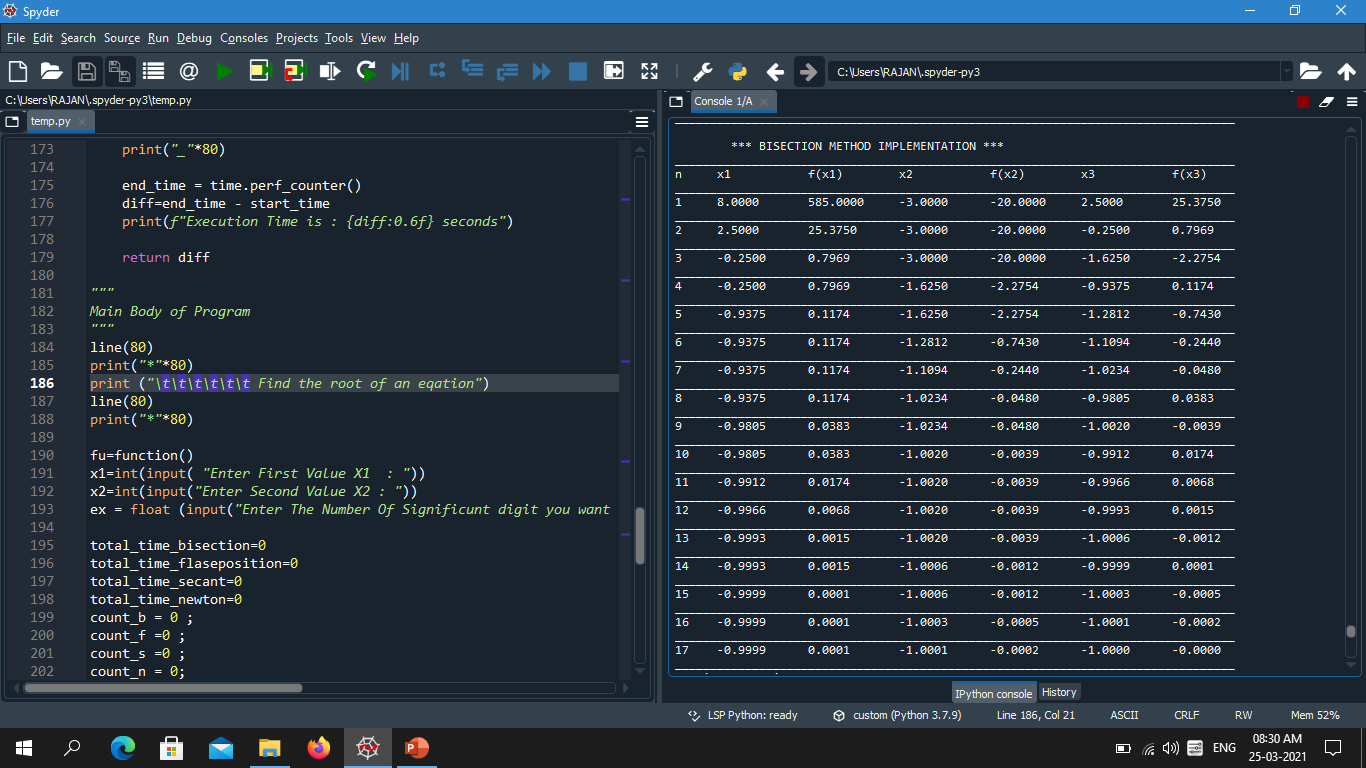
**Input Screen (which user can give any equation)**

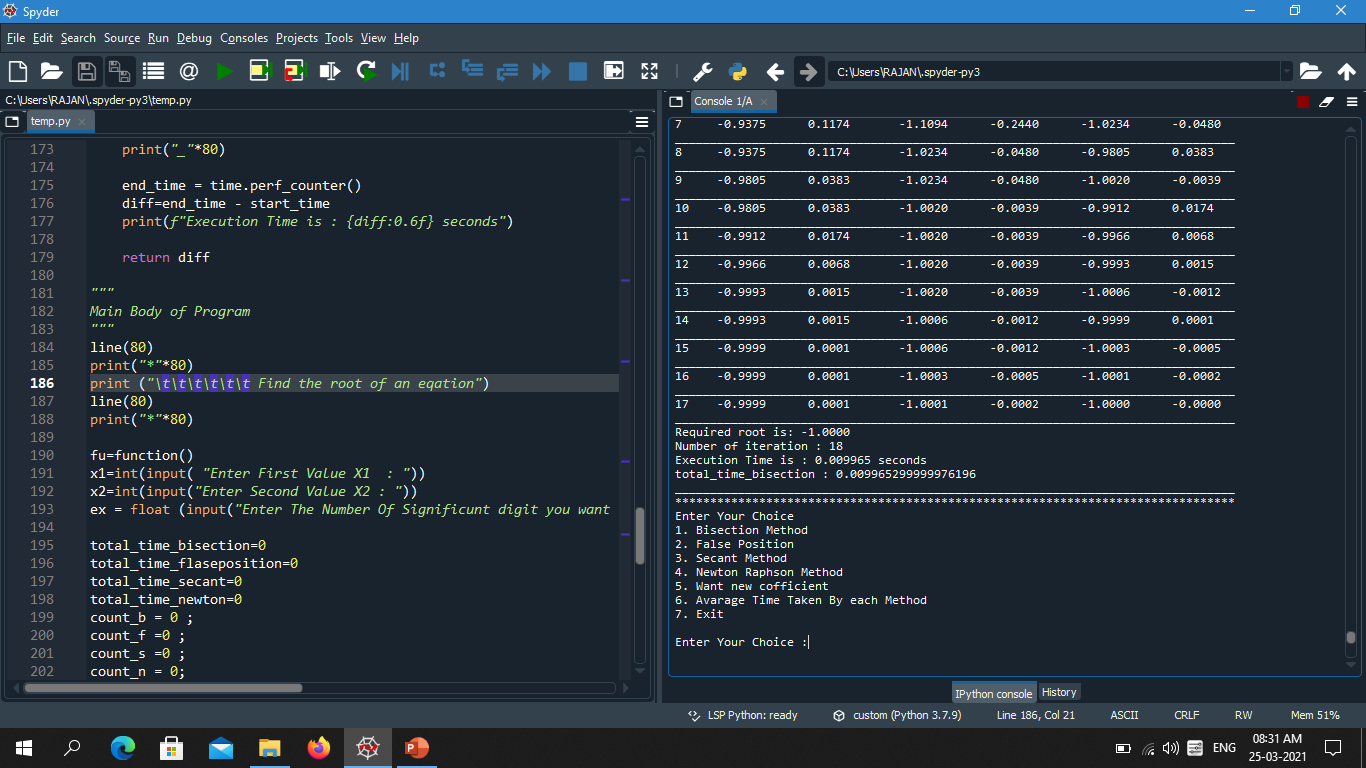
****

**Menu of Different operation**

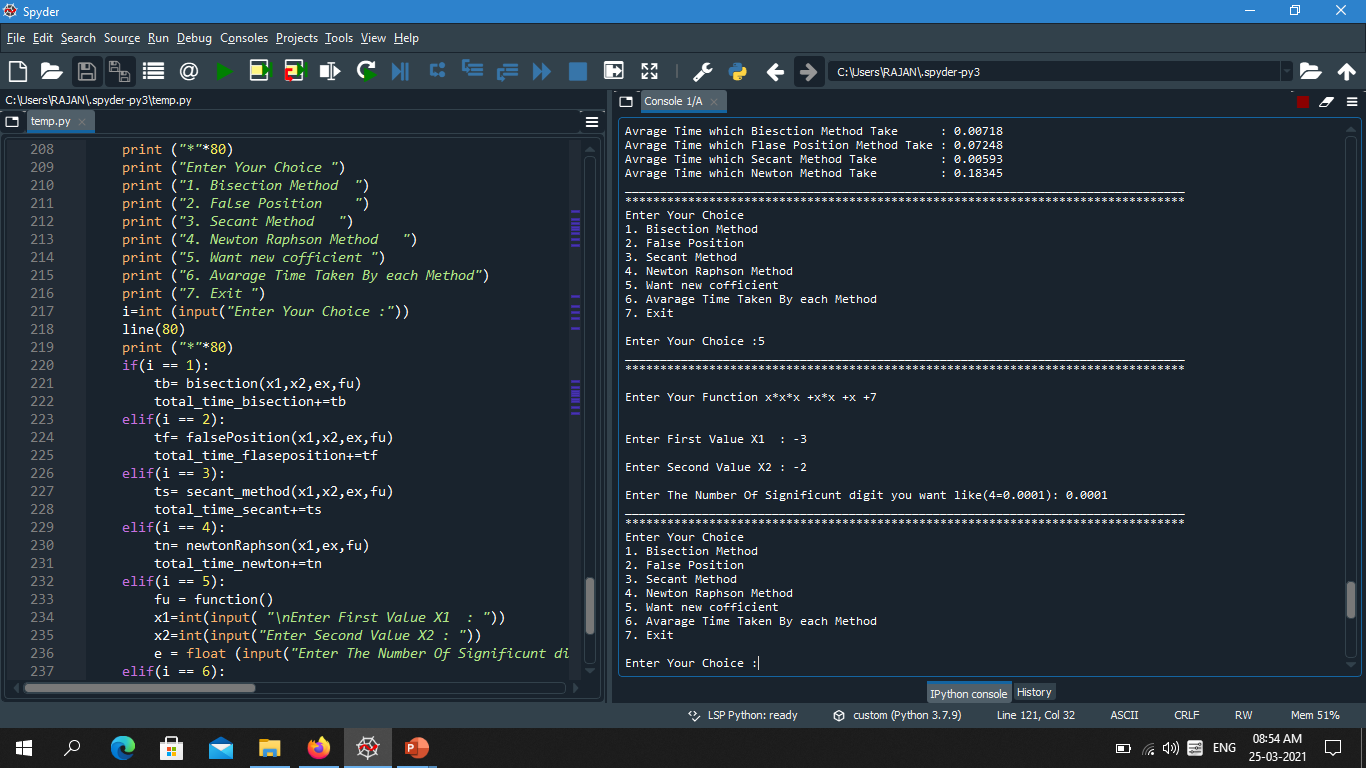
****

**Output of Bisection Method**

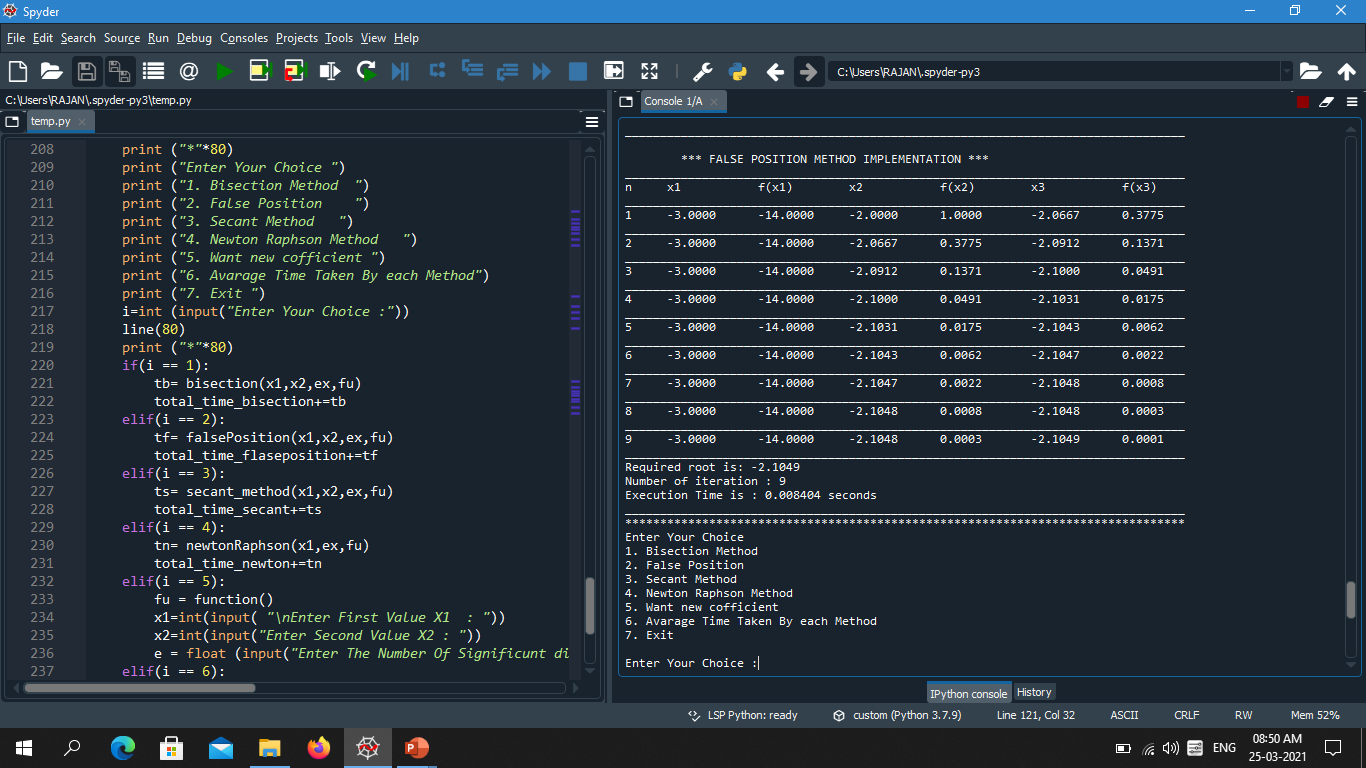
****

****

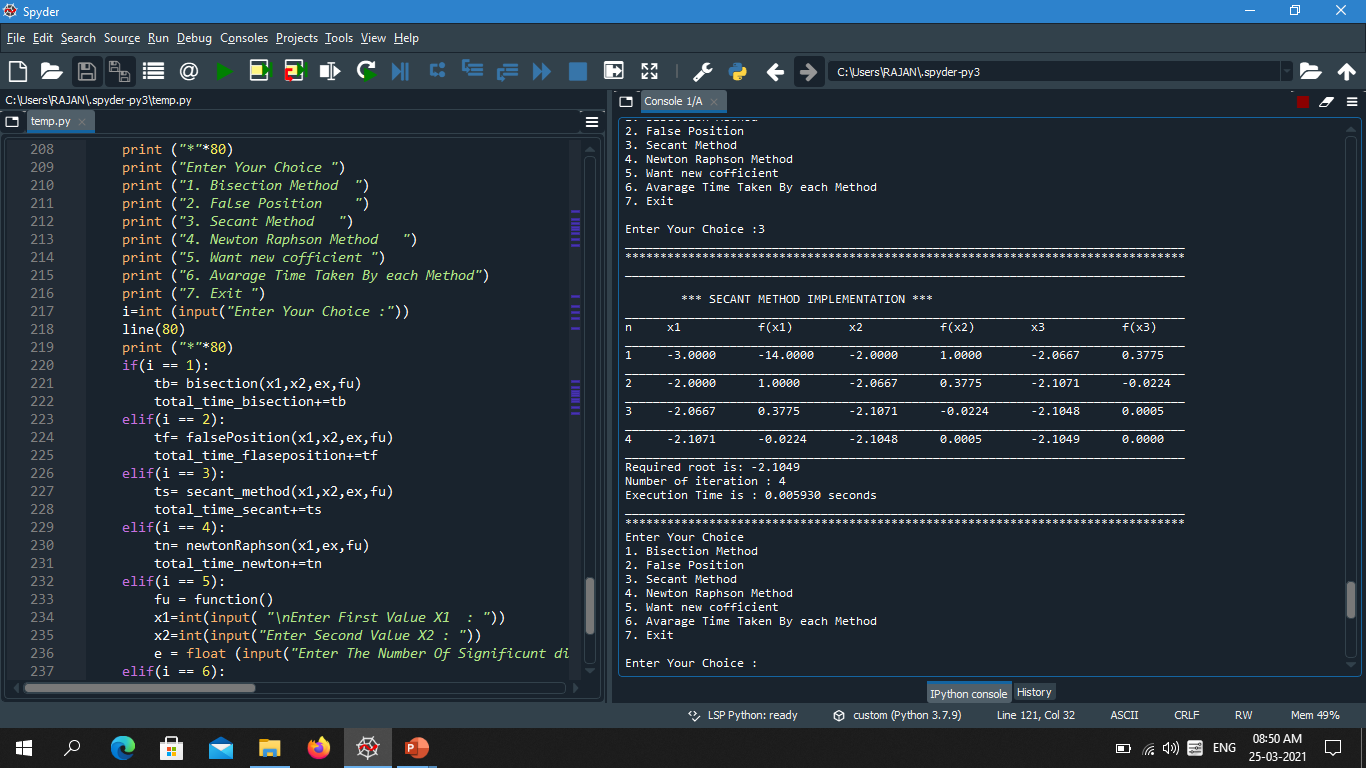
**Enter new coefficient (Dynamic Input)**

****

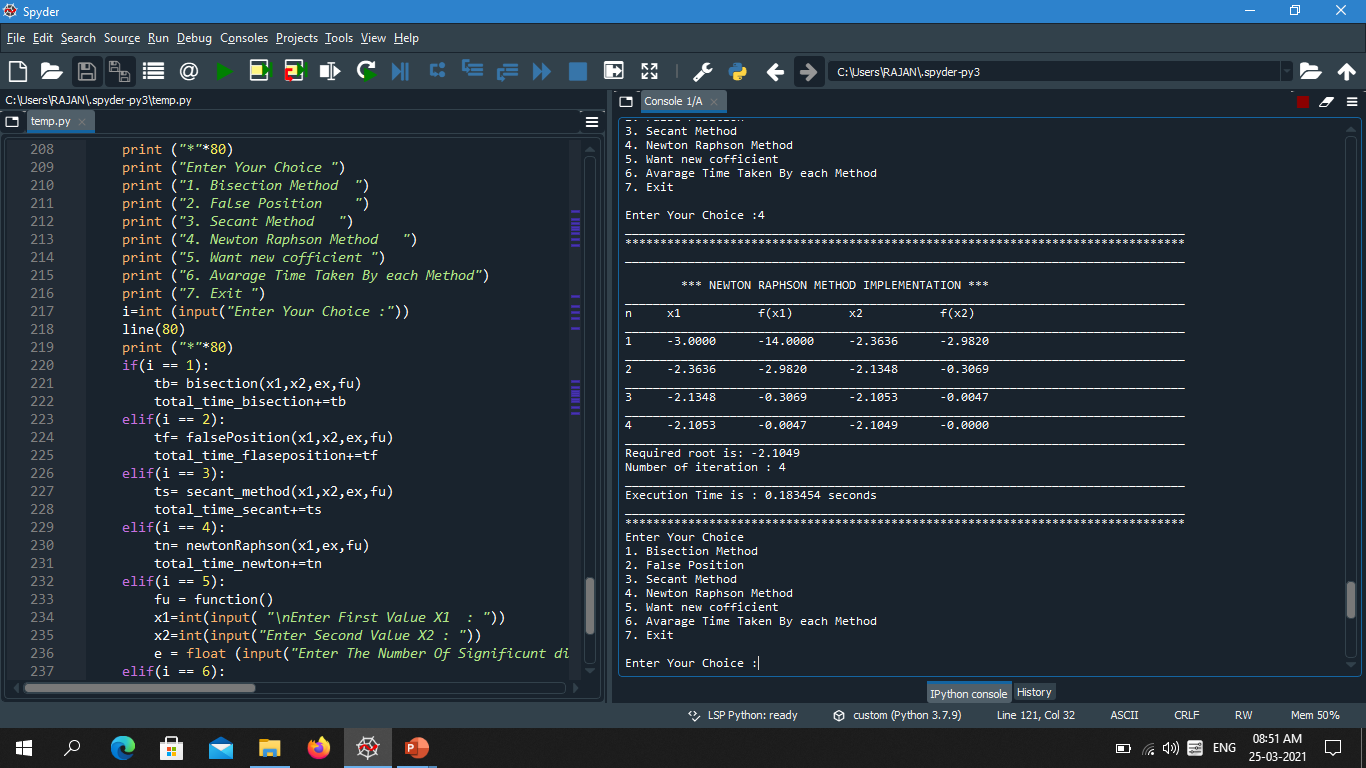
**Output of False Position Method**

****

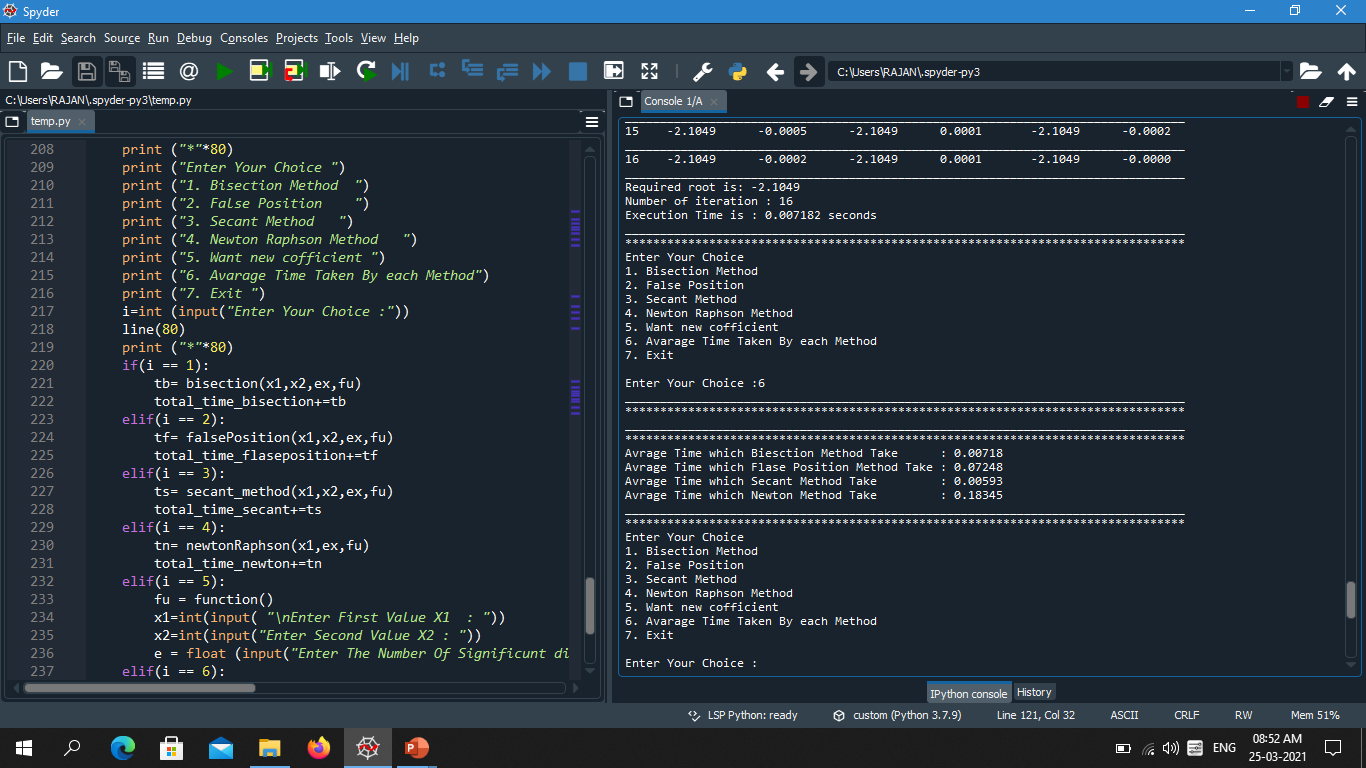
**Output of Secant Method**

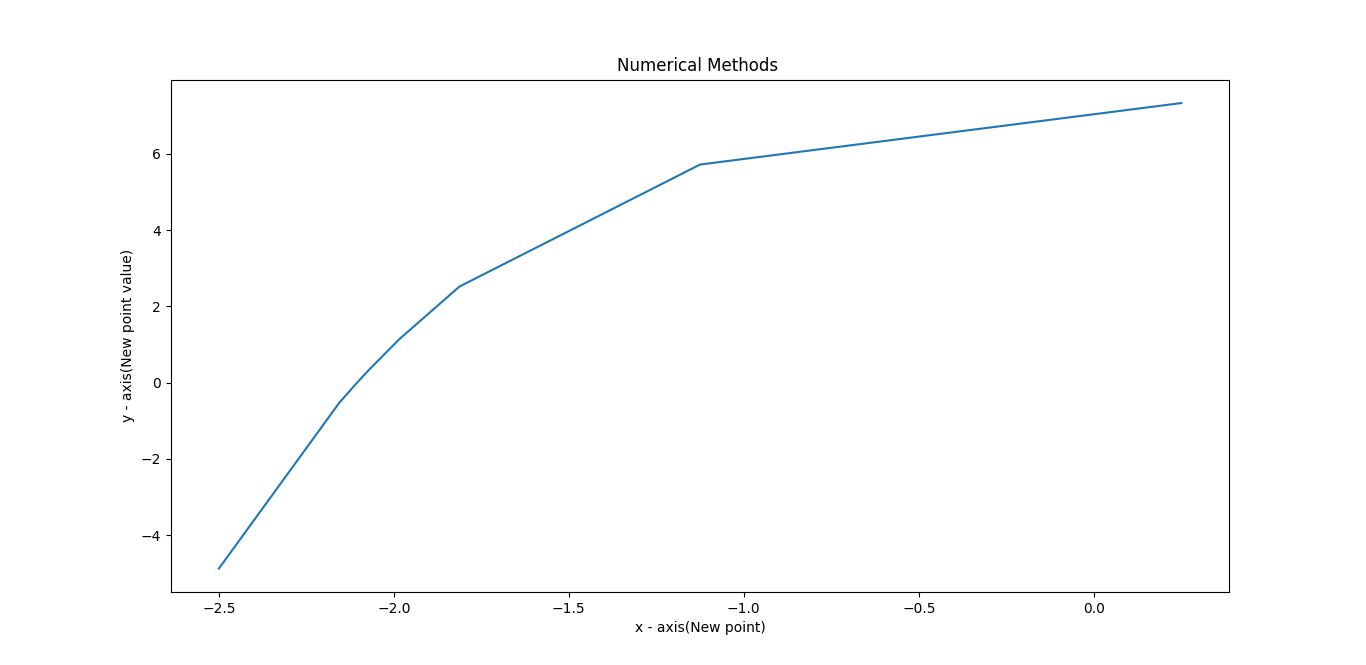
****

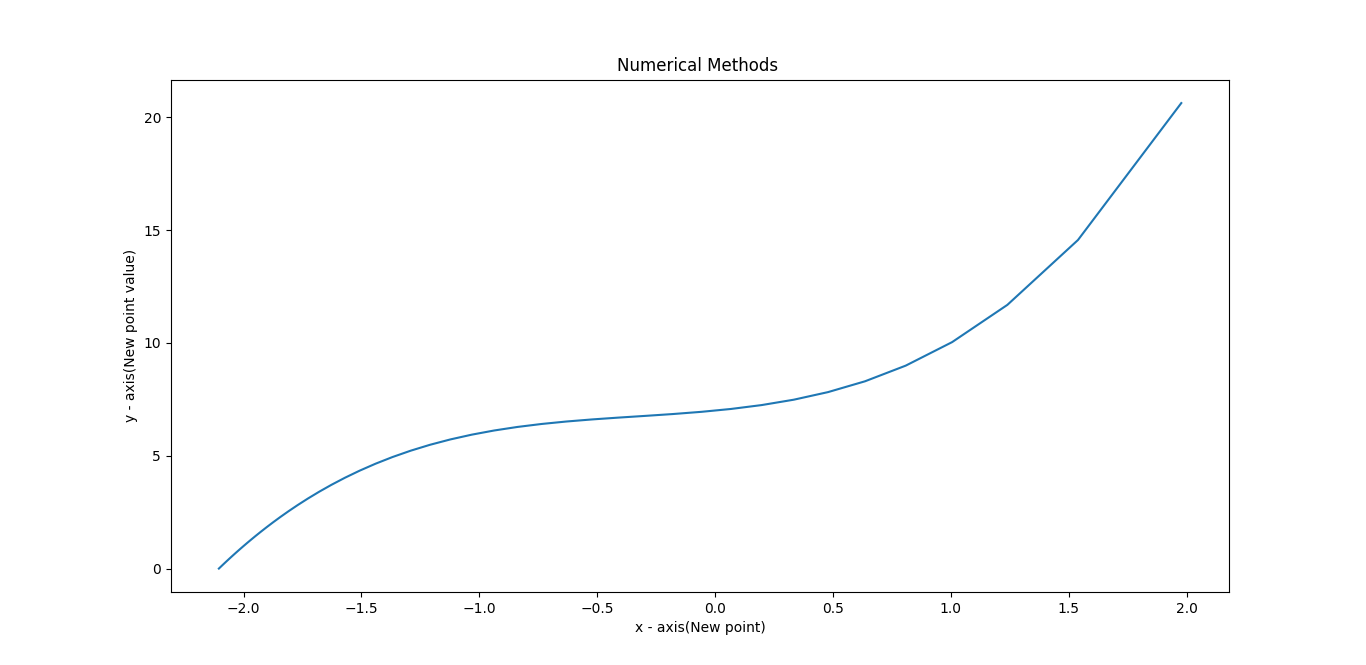
**Output of Newton Raphson Method**

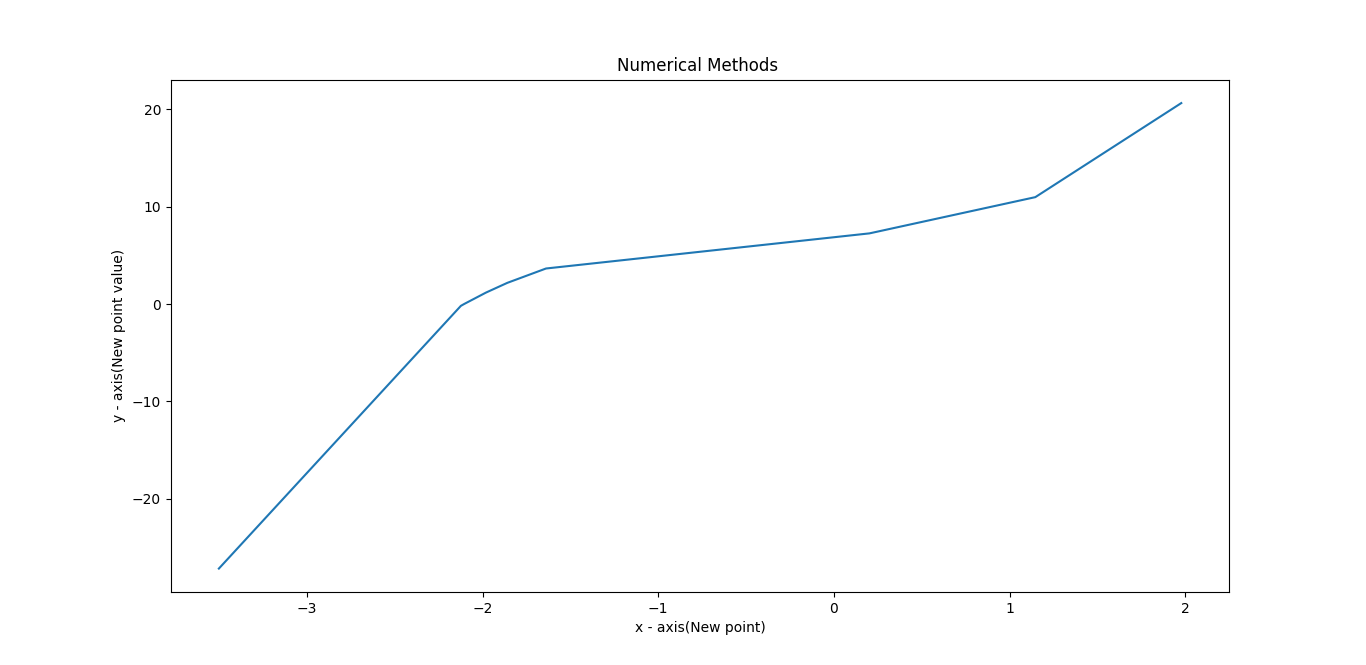
****

**Average Time Calculation**

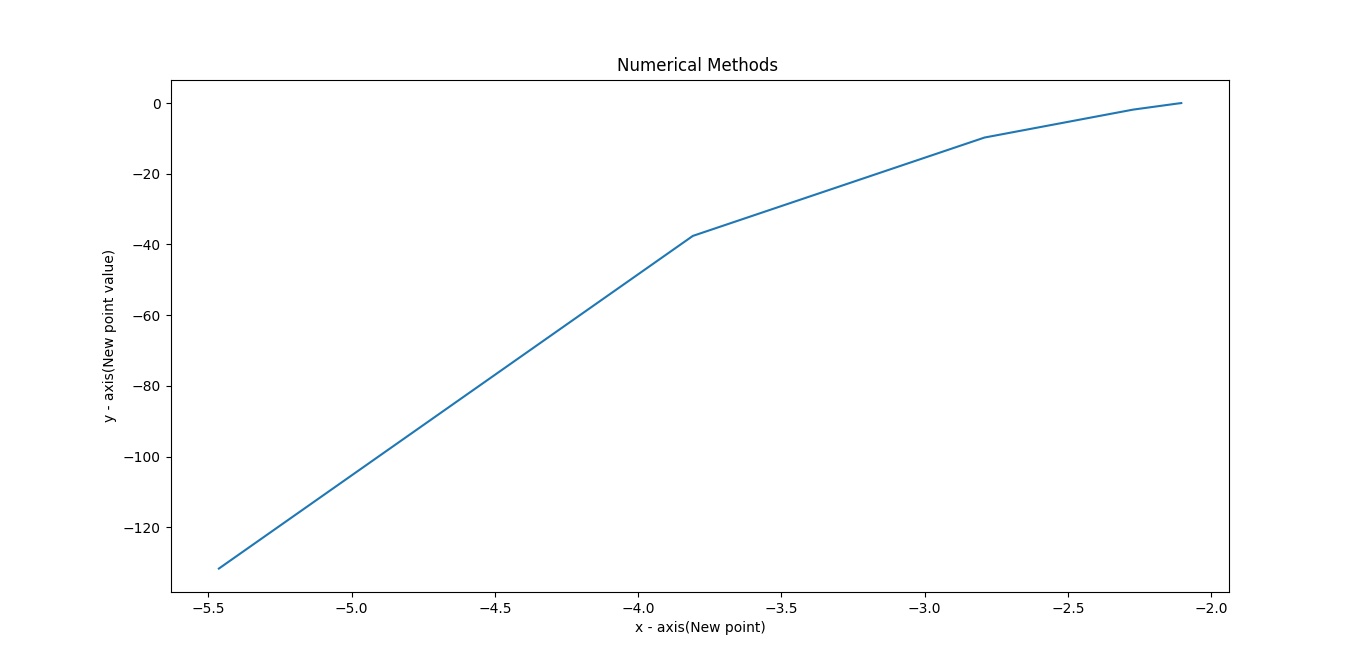
****

**Graph of Bisection Method**

**Graph of False Position Method**

**Graph of Secant Method**

**Graph of Newton Raphson Method**

****

**Conclusion**

* The numerical methods provide a real solution to problems where f (x) = 0.
* Here we are not only programing these methods but also to analyze which method converges who faster than the other method.
* We measure the average time to solve each method (how much time conduct for solving by each method) and the iteration (In how many iterations the method converges.)
* Use dynamic input so that user can input its own choice of function
* We use the graph for new point so that if f(x)=0 than what is the value x we can easily find out by viewing the graph
* In That we also compare two different language which is faster c++ or python.
* We conclude that among all the Newton Raphson is efficient method.
  + Note: calculation time is low in bisection method and give answer in too many iterations but depending upon system newton Raphson is take little bit more time but find the root in few iterations.

**Future Details**

1. Use ML algorithm or method to solve an equation

**References**

* **<https://www.codesansar.com/numerical-methods/> (1)**
* **Introductory methods of numerical analysis , Sastry, S.S. , , PHI Learning Private Limited ,2015 (2)**
* **<https://www.epythonguru.com/2019/12/how-to-find-derivatives-in-python.html> (3)**

**THANK**

**YOU**