**Project Phase 2**

**Team members:**

* رنا محمد علي عطية 21010528
* أمنية طارق ابراهيم السيد 21010298
* نوران أشرف يوسف 21011492
* روان محمد سعيد عبد الفتاح 21010547

**Pseudo-code for each method:**

* **Bisection method**
* **procedure** bisection (fn, xl, xu, fig, eps, iterations):

fl ← precision ( fn.subs(symbol['x'], xl).evalf() )

fu ← precision ( fn.subs(symbol['x'], xu).evalf() )

xr ← precision ( (xl + xu) / 2 )

fr ← precision ( fn.subs(symbol['x'], xr).evalf() )

**if** fl \* fu > 0 **then**

**return** “The function has same sign at end points.”

**end if**

flag ← 0

Ea ← 1

i ← 0

**while** i < iter **and** Ea >eps **do**

**if** fl \* fr < 0 **then**

flag ← 0

xu ← xr

fu ← fr

**else if** fl \* fr > 0 **then**

flag ← 1

xl ← xr

fl ← fr

**end if**

xr ← precision( (xl + xu) / 2 )

fr ← precision( fn.subs(symbol['x'], xr).evalf() )

i ← i + 1

**if** flag == 0 **then**

Ea ← abs(xr - xu)

**else**

Ea ← abs(xr - xl)

**end if**

**if** i >= iter **and** Ea > eps **then**

result ← "Bisection method didn't converge."

**else if** i < iter **and** Ea <= eps **then**

result ← "x = " + str(xr)

**end if**

**return** result

* **False-Position method**
* **procedure** false\_position(fn, xl, xu, fig, eps, iterations):

fl ← precision( fn.subs(symbol['x'], xl).evalf() )

fu ← precision( fn.subs(symbol['x'], xu).evalf() )

**if** fl \* fu > 0 **then**

**return** “The function has same sign at end points.”

**end if**

xr ← precision( (xl \* fu - xu \* fl) / (fu - fl) )

fr ← precision( fn.subs(symbol['x'], xr).evalf() )

flag ← 0

Ea ← 1

i ← 0

**while** i <iter **and** Ea > eps **do**

**if** fl \* fr < 0 **then**

flag ← 0

xu ← xr

fu ← fr

**else if** fl \* fr > 0 **then**

flag ← 1

xl ← xr

fl ← fr

**end if**

xr ← precision( (xl \* fu - xu \* fl) / (fu - fl) )

fr ← precision( fn.subs(symbol['x'], xr).evalf() )

i ← i + 1

**if** flag == 0 **then**

Ea ← abs(xr - xu)

**else**

Ea ← abs(xr - xl)

**end if**

**if** i >= iter **and** Ea > eps **then**

result ← "False-Position method didn't converge."

**else if** i < iter **and** Ea <= eps **then**

result ← "x = " + str(xr)

**end if**

**return** result

* **Fixed Point method**

**A computer screen shot of text

Description automatically generated**

* **Original Newton-Raphson method**
* **procedure** newton\_original (fn, x0, fig, m, eps , iter) :

Ea ← 1

i ← 0

df ← diff (fn, symbol['x'])

**while** i < iter **and** Ea > eps **do**

**if** x0 == 0 **and** fn.subs(symbol['x'], x0).evalf()== 0 **then**

**return** "x = " + str(x0) + "\nThe number of iterations = " + str(i)

**end if**

x1 ← x0 - m \* (precision(fn.subs(symbol['x'], x0).evalf()) ) /

(precision(df.subs(symbol['x'], x0).evalf()) )

x1 ← precision(x1)

i ← i + 1

Ea ← abs( ( x1 - x0 ) / x1 )

x0 = x1

**end while**

**if** i >= iter **and** Ea > eps **then**

result ← "Newton Raphson method didn't converge."

**else if** i < iter **and** Ea <= eps **then**

result ← "x = " + str(x0)

**end if**

**return** result

* **Modified Newton-Raphson method (Second modification)**
* **procedure** newton\_mod2 (fn, x0, fig, eps , iter)) :

Ea ← 1

i ← 0

df ← diff (fn, symbol['x'])

d2f ← diff (df, symbol['x'])

**while** i < iter **and** Ea > eps **do**

**if** x0 == 0 **and** fn.subs(symbol['x'], x0).evalf()== 0 **then**

**return** "x = " + str(x0) + "\nThe number of iterations = " + str(i)

**end if**

value ← precision(fn.subs(symbol['x'], x0).evalf()) \*

precision(df.subs(symbol['x'], x0).evalf())

value2 ← ( precision(df.subs(symbol['x'], x0).evalf()) )\*\*2

value3 ← precision(fn.subs(symbol['x'], x0).evalf()) \*

precision(d2f.subs(symbol['x'], x0).evalf())

x1 ← x0 - ( precision(value) ) / ( precision(value2) - precision(value3) )

x1 ← precision(x1)

i ← i + 1

Ea ← abs( ( x1 - x0 ) / x1 )

x0 ← x1

**end while**

**if** i >= iter **and** Ea >eps **then**

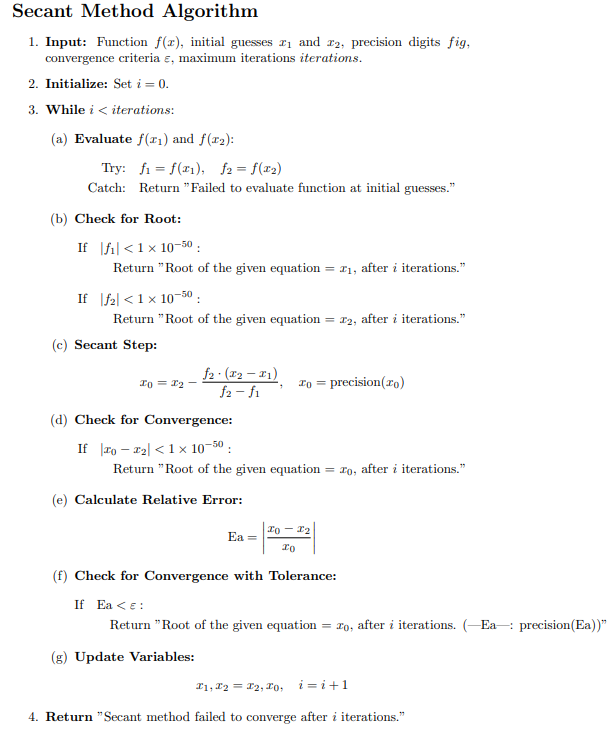
result ← "Newton Raphson method didn't converge."

**else if** i <iter **and** Ea <=eps **then**

result ← "x = " + str(x0)

**end if**

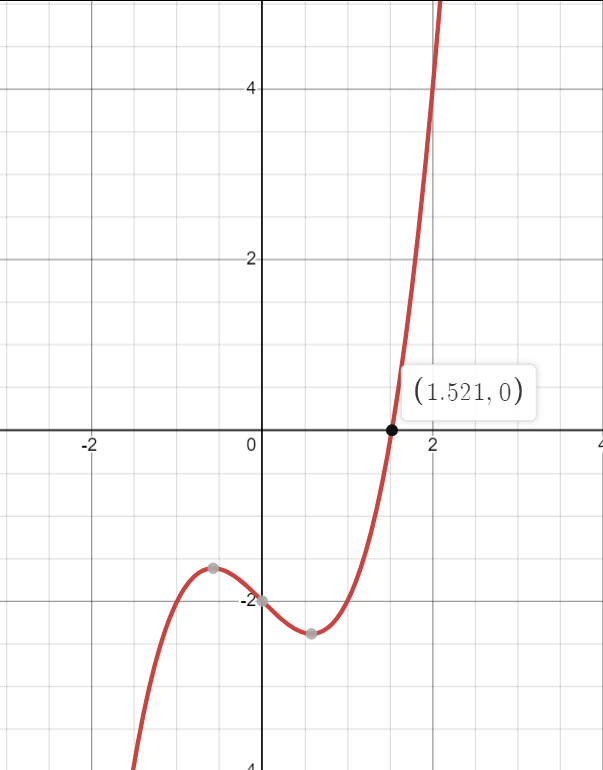
**return** result

* **Secant method**

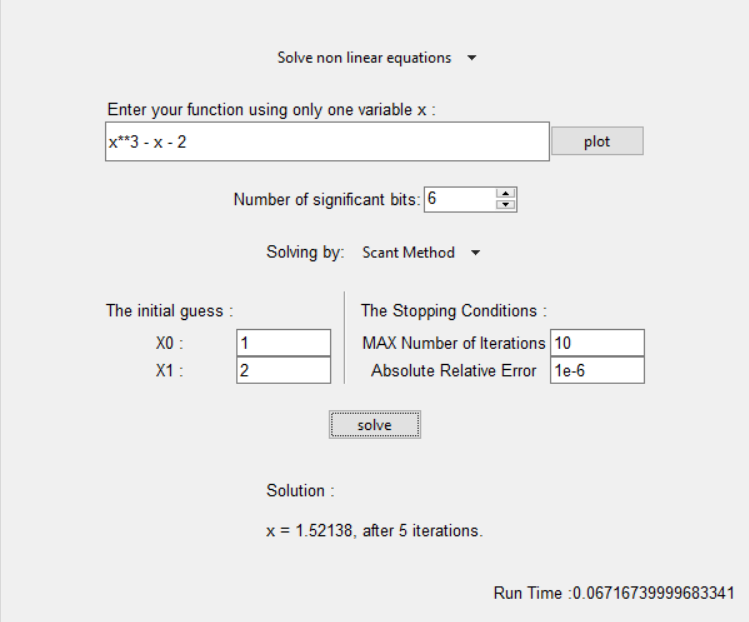
**Sample runs for each method:**

* **Bisection method**
* **False-Position method**
* **Fixed Point method**
* **Original Newton-Raphson method**
* **Modified Newton-Raphson method**
* **Secant method**

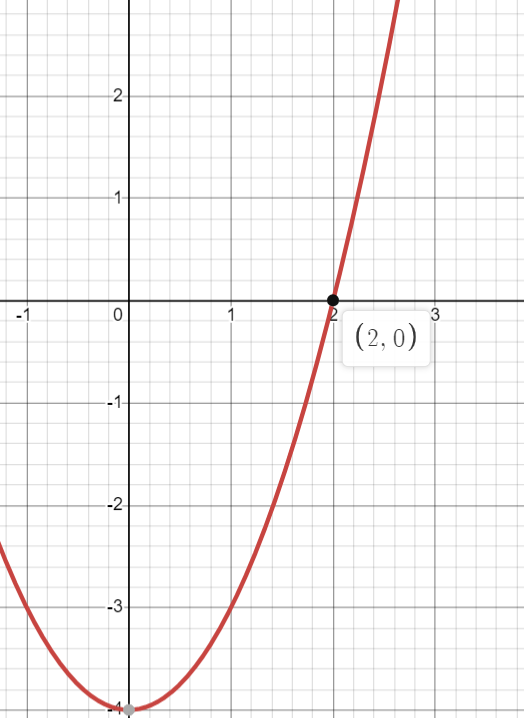
**Expected root for X3-X-2:**

****

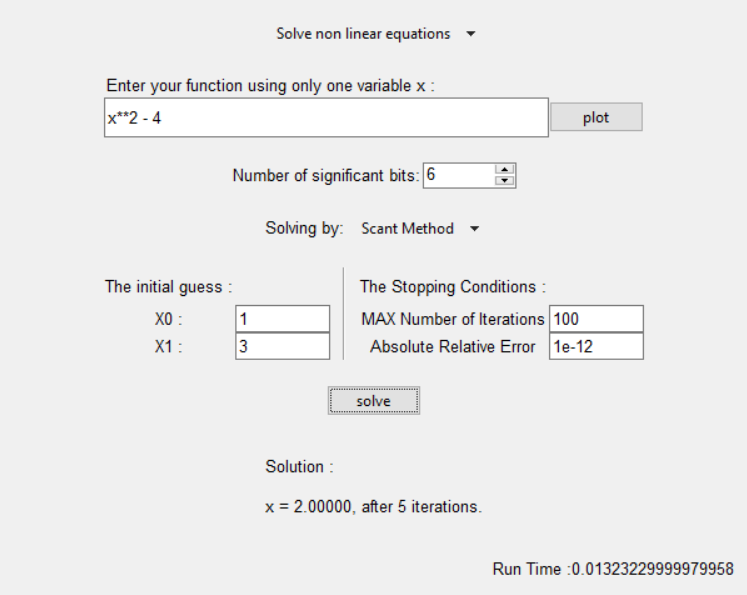
**Calculated from the app:**

****

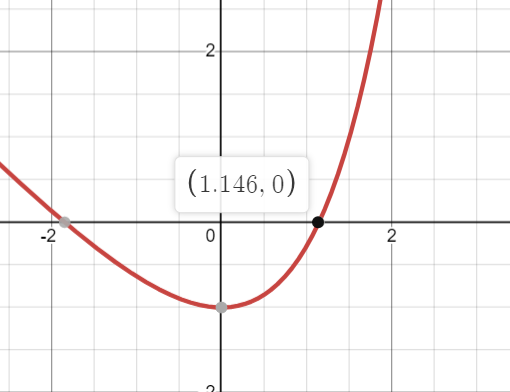
**Expected root for X2- 4 between 1 and 3:**

****

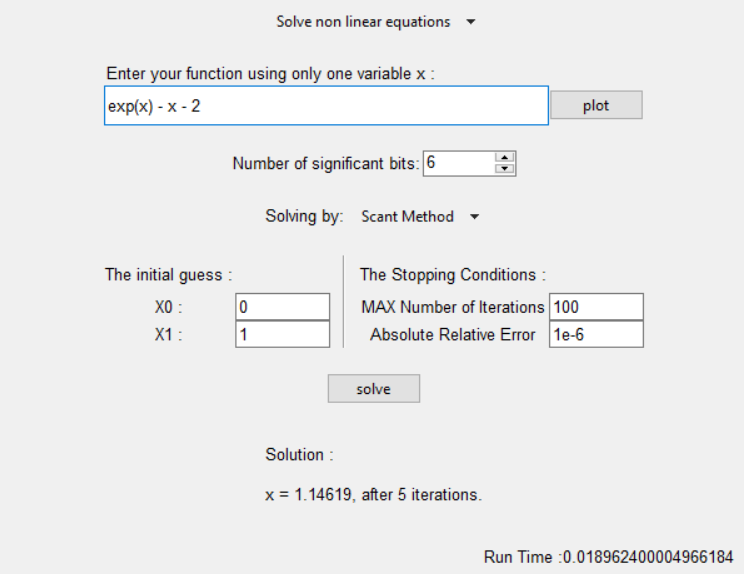
**Calculated from the app:**

****

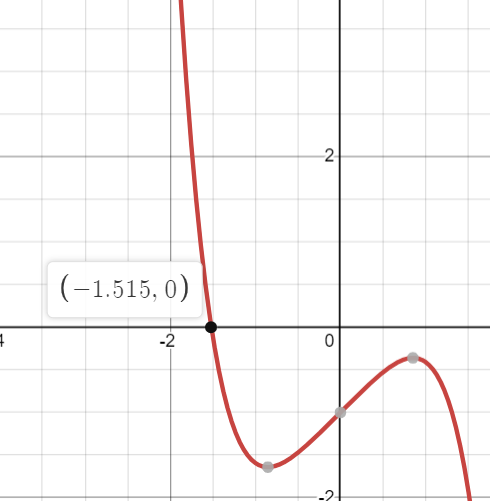
**Expected root for ex - x - 2:**

****

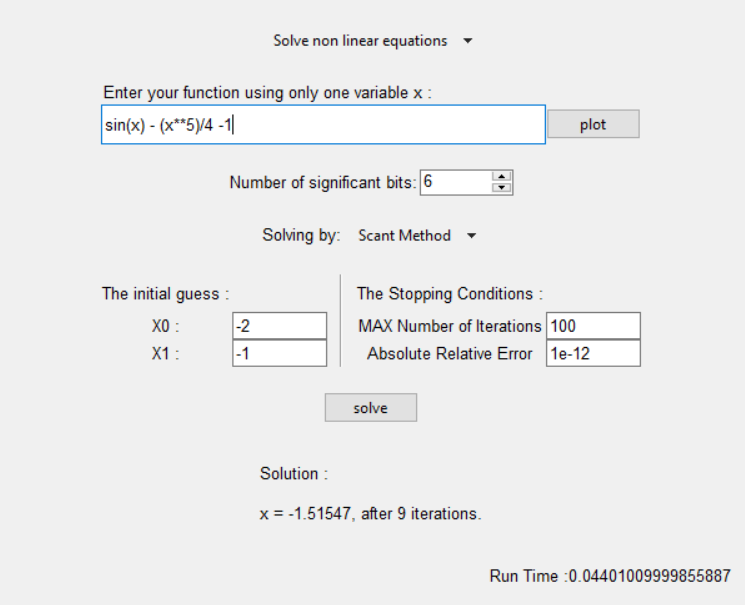
**Calculated from the app:**

****

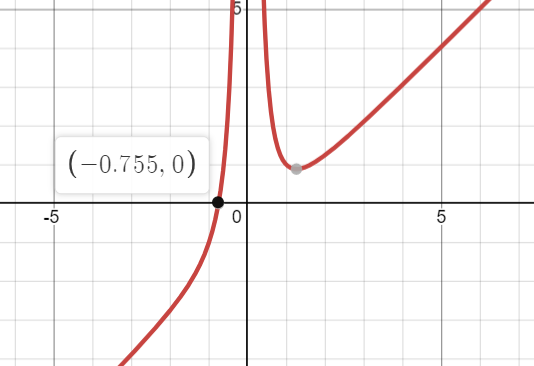
**Expected root for sin(x) - (x\*\*5)/4 -1:**

****

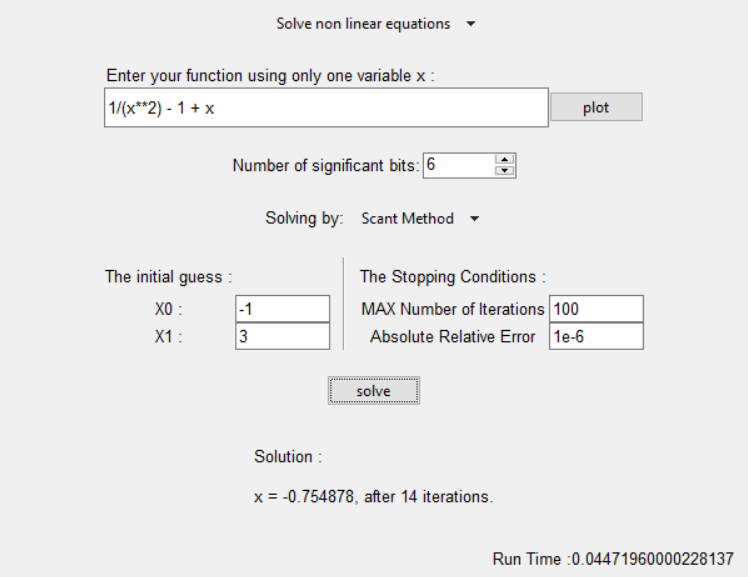
**Calculated from the app:**

****

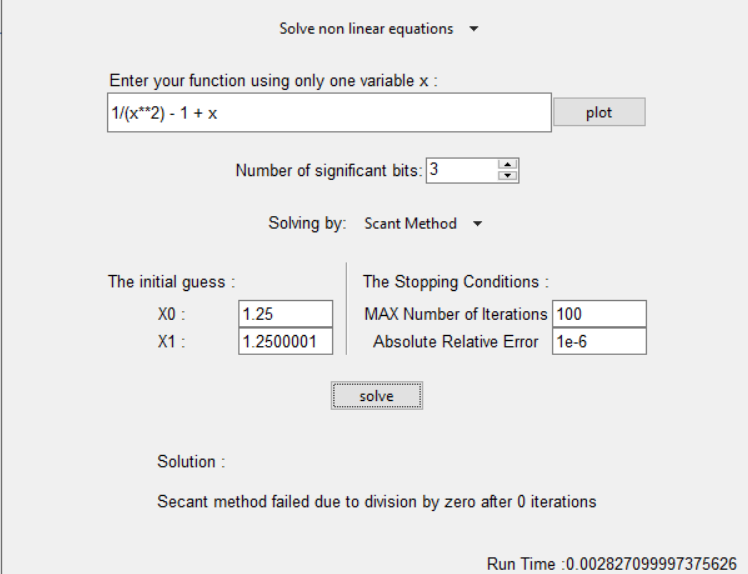
**Expected root for 1/(x\*\*2) - 1 + x:**

****

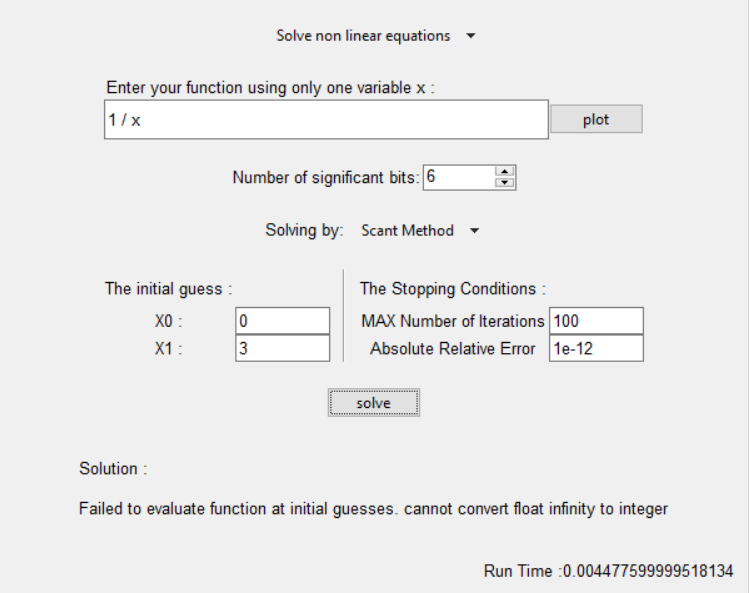
**Calculated from the app:**

****

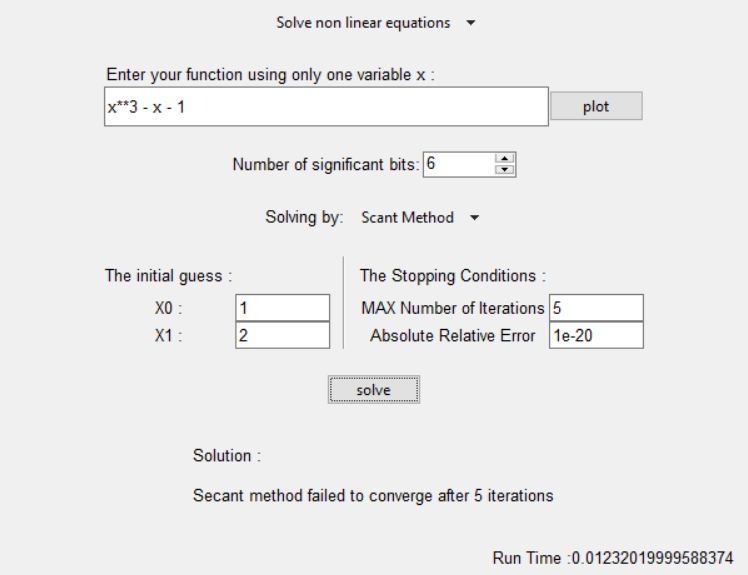
**When initial guesses are very close and precision is small subtractive cancellation occurs, so method terminate due to division by zero:**

****

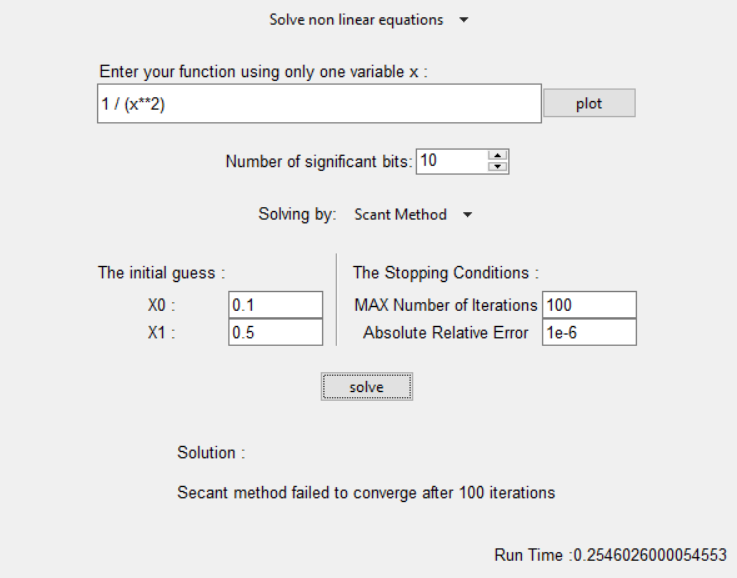
**In case of initial guesses that don’t belong to the domain of the function an error message will appear:**

****

**When the number of iterations is very small the algorithm will terminate before reaching the desired tolerable error:**

****

**In case of no real root, method fails to converge:**

****

**Comparison between different methods:**

* **Bisection method**

**Time Complexity:**

The time complexity equals O(n) where n equals the maximum number of iterations to reach the desired error.

**Convergence and Approximate Errors:**

It will always converge to the true root, but it converges slower than open methods.

It cannot locate even multiple roots (roots of even multiplicity).

* **False-Position method**

**Time Complexity:**

The time complexity equals O(n) where n equals the maximum number of iterations to reach the desired error.

**Convergence and Approximate Errors:**

It will always converge to the true root, and it converges faster than the bisection method but also slower than the open methods.

It cannot locate even multiple roots (roots of even multiplicity).

* **Fixed Point method**

**Time Complexity:**

* O(n) is the time complexity, where n equals the maximum number of iterations.

**Convergence and Approximate Errors:**

* Fixed-point iteration converges if the abs slope of the line of g(x) <1.
* When the method converges, the error is roughly proportional to or less than the error of the previous step, therefore, it is called “linearly convergent.”
* **Original Newton-Raphson method**

**Time Complexity:**

The time complexity equals O(n) where n equals the maximum number of iterations to reach the desired error.

**Convergence and Approximate Errors:**

The method converges quadratically if the multiplicity of the root is 1.

If the multiplicity of the root is greater than 1, then the method converges linearly.

* **Modified Newton-Raphson method (Second modification)**

**Time Complexity:**

The time complexity equals O(n) where n equals the maximum number of iterations to reach the desired error.

**Convergence and Approximate Errors:**

The method converges quadratically if the multiplicity of the root is greater than 1.

If the multiplicity of the root is 1, then the method converges slower than the original method.

* **Secant method**

**Time Complexity:**

* In the worst case, the method has a fixed maximum number of iterations, and each iteration involves a constant number of calculations. Therefore, the time complexity is linear with respect to the number of iterations.
* Time Complexity: O(n), where n is the number of iterations.

**Convergence and Approximate Errors:**

* It converges faster than linear methods (such as the Bisection method, false position and fixed-point method) but slower than quadratic methods.
* The Secant Method is not guaranteed to converge for all functions.
* Depending on the choice of initial points and the behavior of the function, it may diverge (if f(X0)- f(X1) = 0).
* Approximate error in every iteration provides a measure of how much the current approximation has changed relative to the previous one, giving an indication of convergence.

**Data structure used:**

**Dictionary (symbol):**

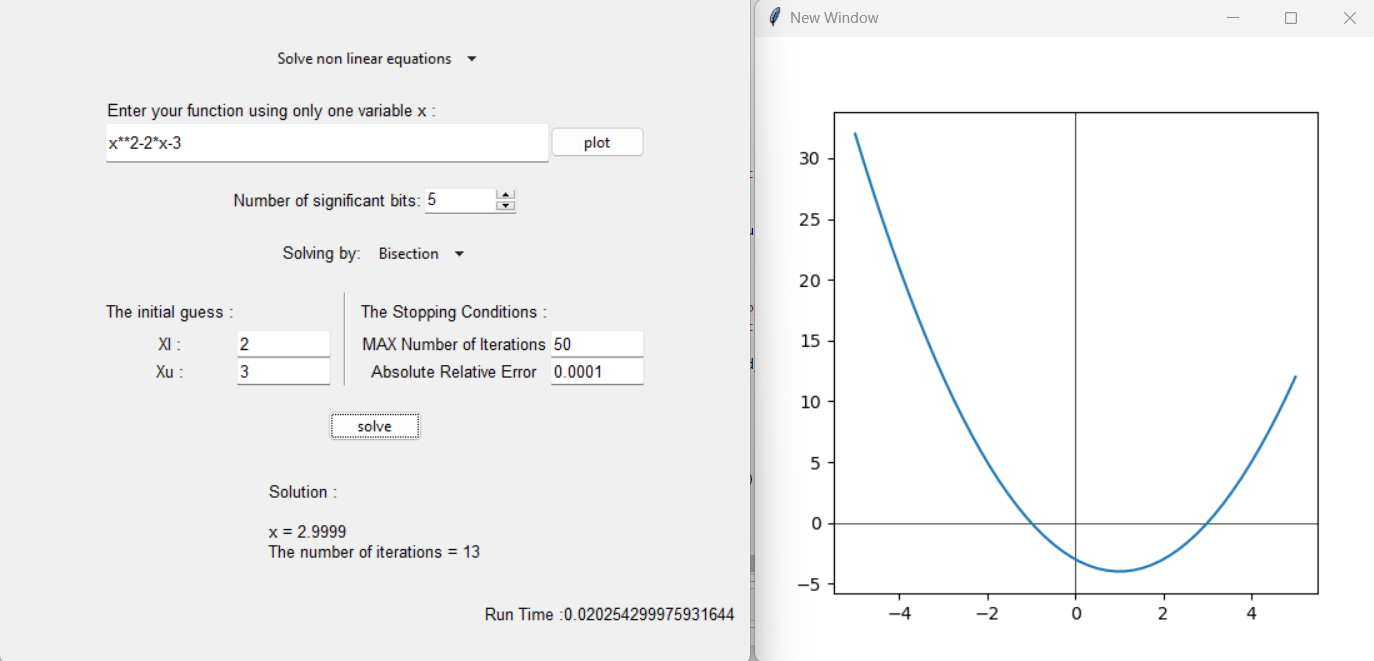
**Usage:**

* store the symbolic variable 'x' using the sympy library.
* Helpful for symbolic expression manipulation, differentiation, and substitution.

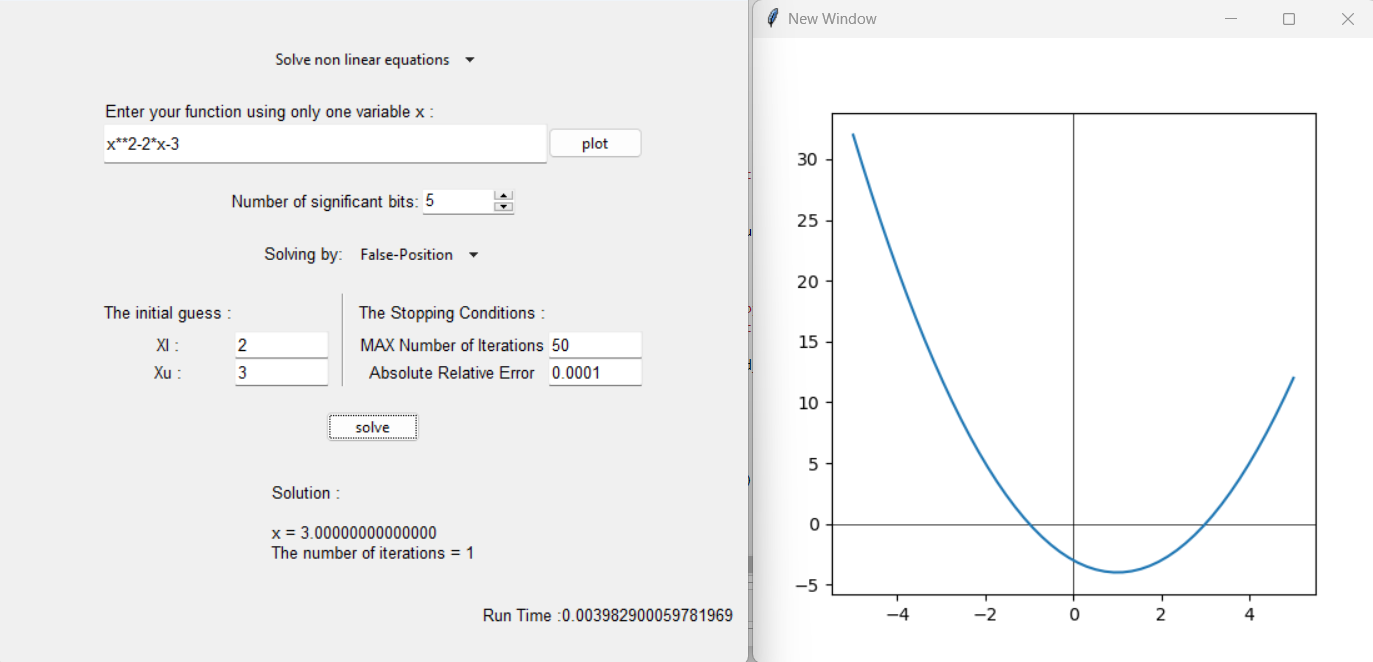
**Test Cases:**

* **Test case 1:**

**Bisection method:**

****

**False position method:**

****

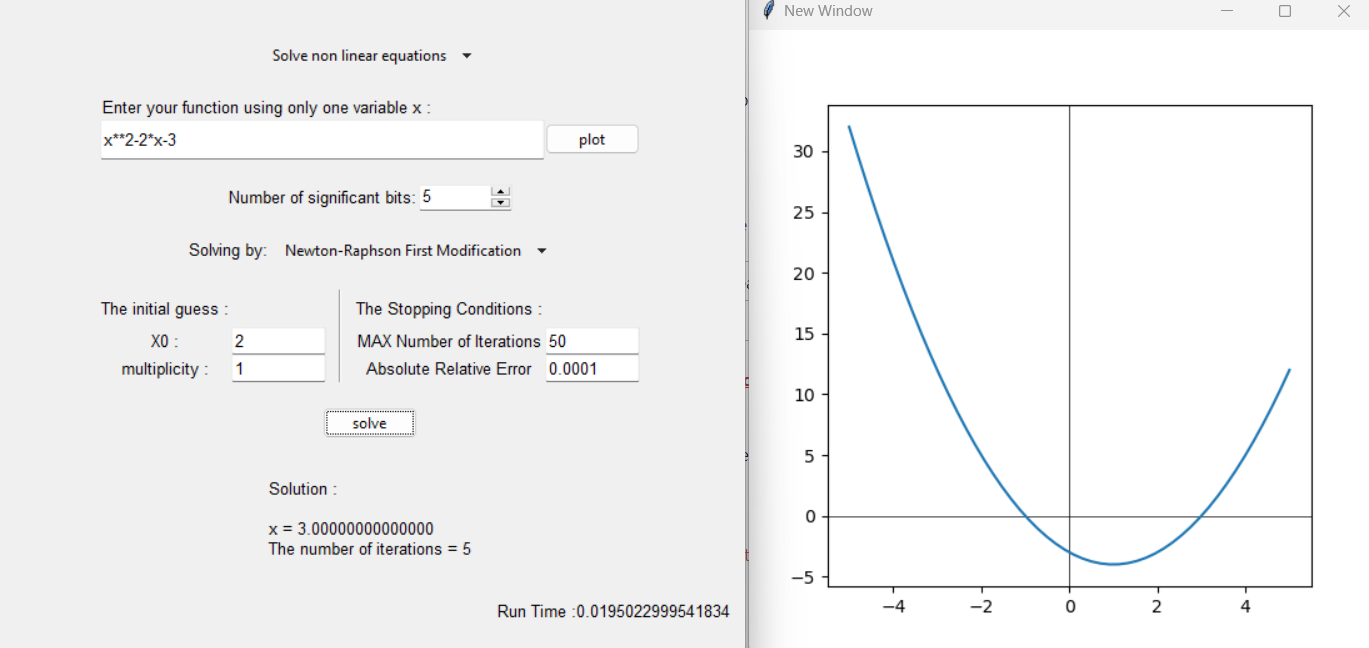
**Fixed point:**

**A screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generated**

**Newton Raphson:**

****

**Secant method:**

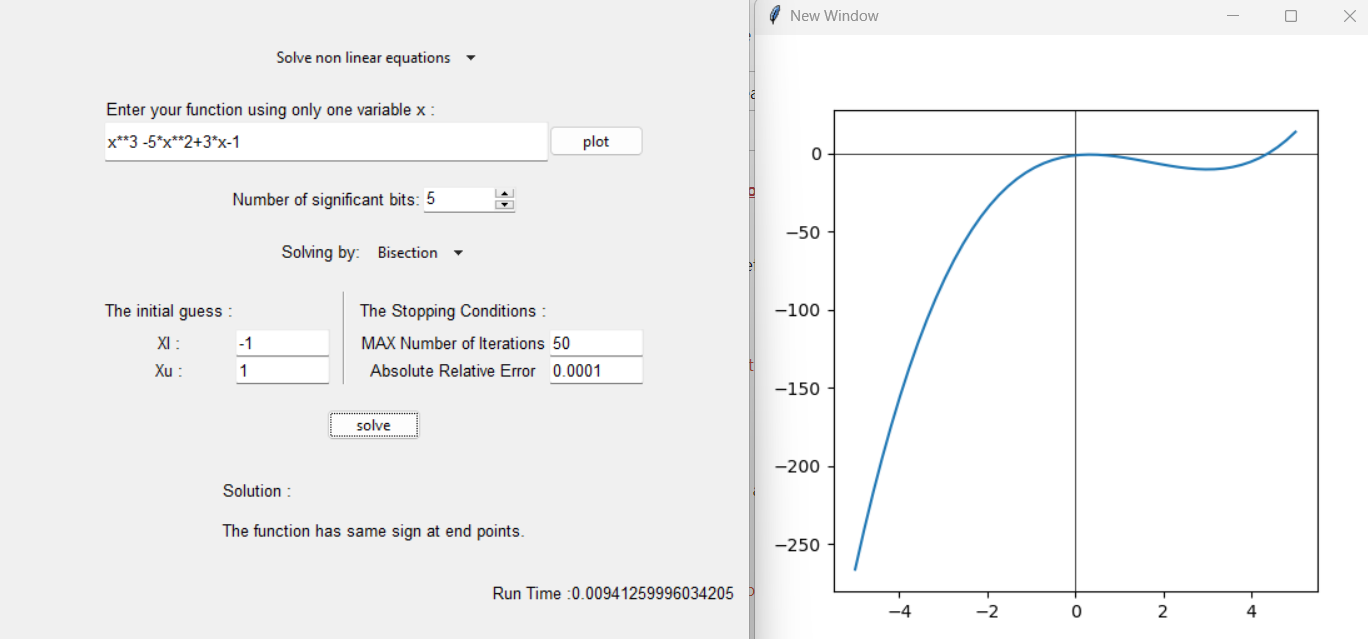
A screenshot of a computer

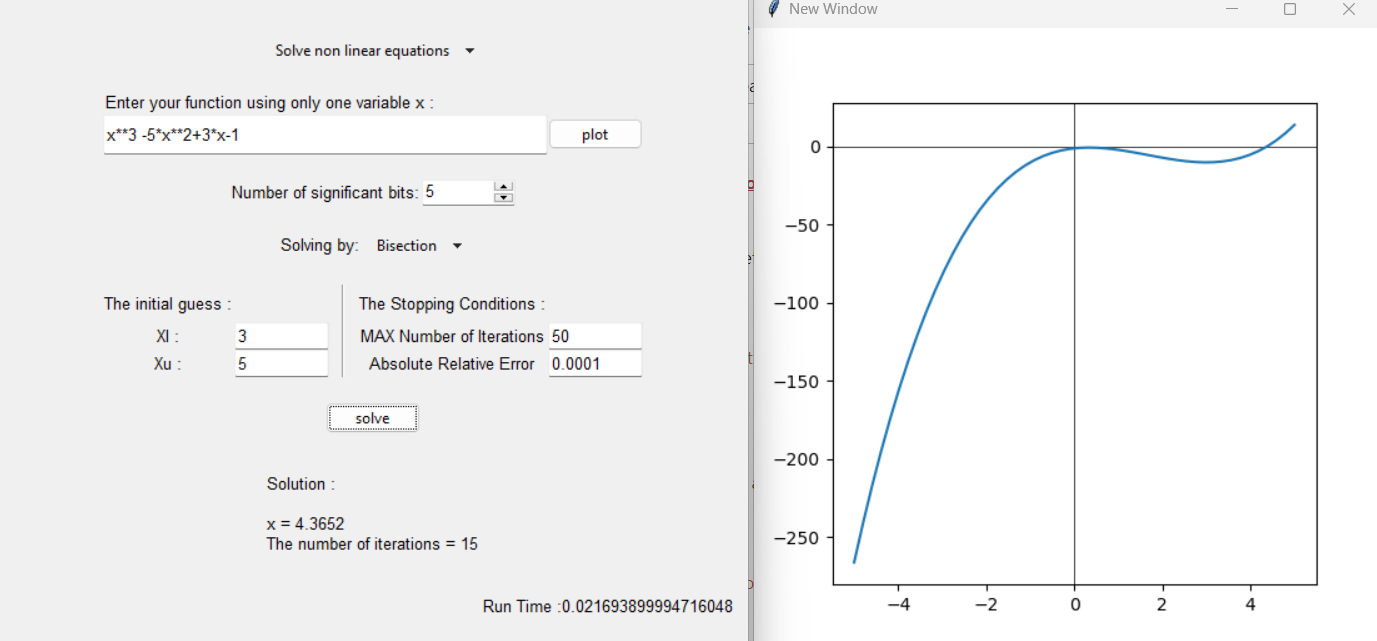
Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Number of Iter. | Run Time | Root | Initial Guesses |
| **Bisection** | 13 | 0.020254 | 2.9999 | X0 = 2 X1 = 3 |
| **False Position** | 1 | 0.003982 | 3.0000 | X0 = 2 X1 = 3 |
| **Fixed Point** | 9 | 0.029936 | 3.0001 | X0 = 2 |
| **Newton Raphson** | 5 | 0.019502 | 3.0000 | X0 = 2 |
| **Secant** | 0 | 0.004299 | 3.0000 | X0 = 2 X1 = 3 |

* **Test case 2**

**Bisection method:**

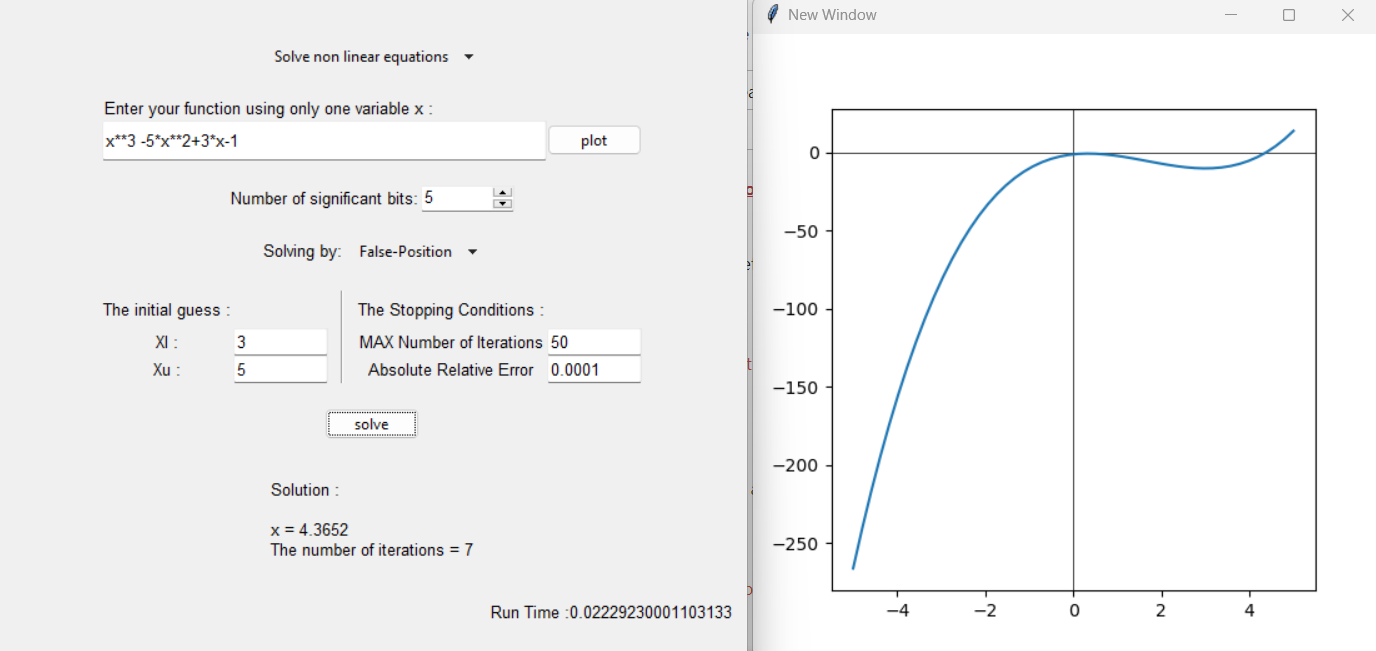
****

****

**False position method:**

**A screenshot of a computer

Description automatically generated**

****

**Fixed point:**

**A screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generated**

**Newton Raphson:**

**A screenshot of a computer

Description automatically generated**

**Secant:**

**A screenshot of a computer

Description automatically generated**

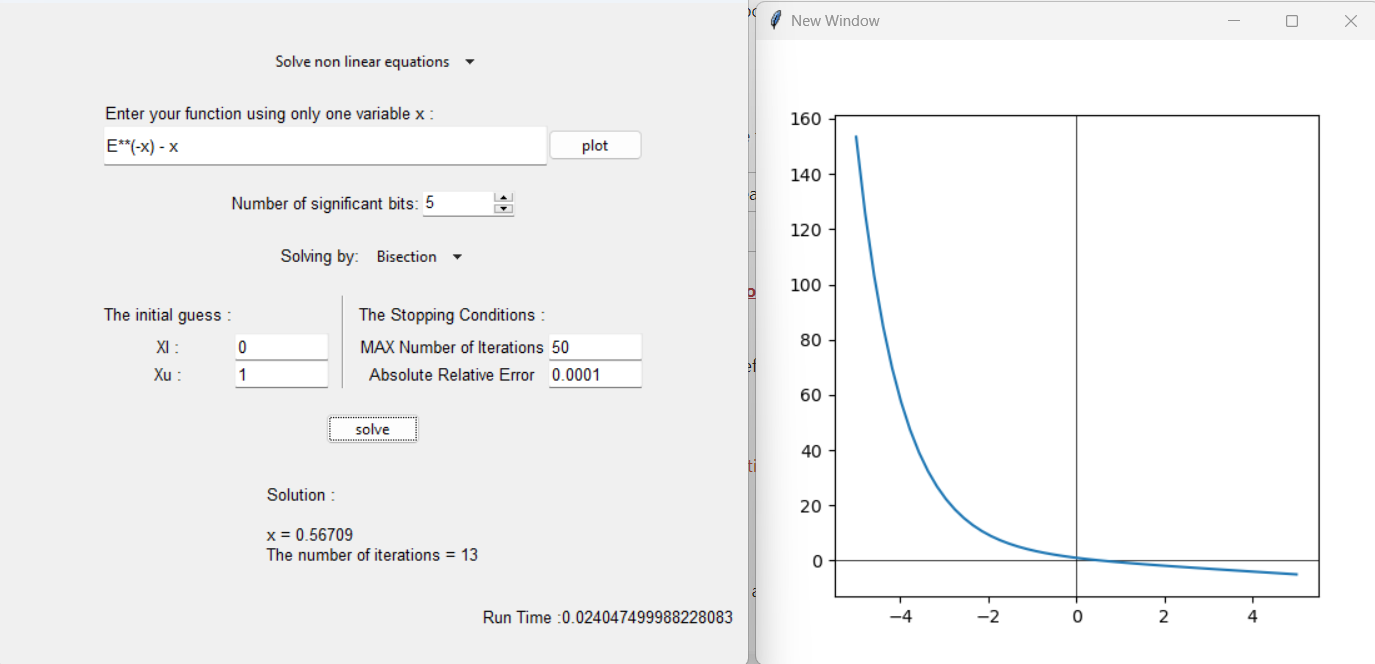
**A screenshot of a computer

Description automatically generated**

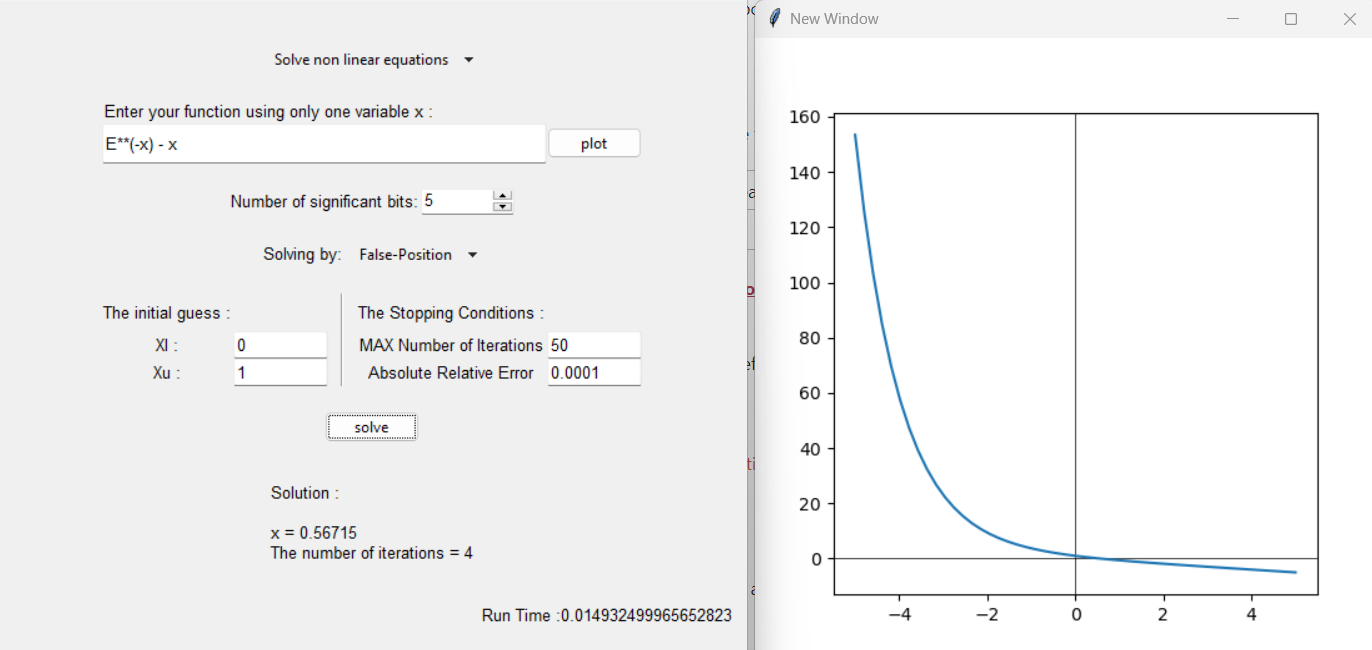
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Number of Iter. | Run Time | Root | Initial Guesses |
| **Bisection** | 15 | 0.021694 | 4.3652 | X0 = 3 X1 = 5 |
| **False Position** | 7 | 0.022292 | 4.3652 | X0 = 3 X1 = 5 |
| **Fixed Point** | 6 | 0.018017 | 4.3652 | X0 = 2 |
| **Newton Raphson** | 4 | 0.020592 | 4.3652 | X0 = 4 |
| **Secant** | 5 | 0.017409 | 4.3651 | X0 = 3 X1 = 5 |

* **Test case 3**

**Bisection method:**

****

**False position method:**

****

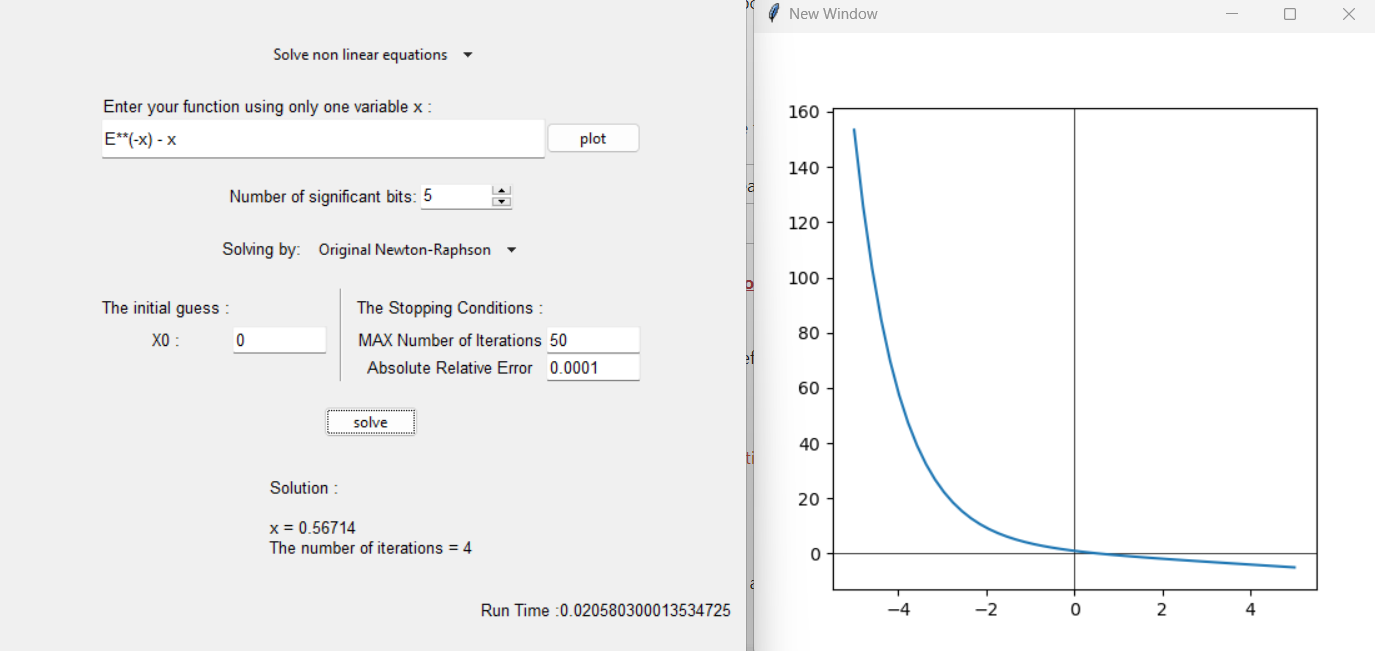
**Fixed point:**

**A screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generated**

**Newton Raphson:**

****

**Secant:**

**A screenshot of a computer

Description automatically generated**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Number of Iter. | Run Time | Root | Initial Guesses |
| **Bisection** | 13 | 0.024047 | 0.56709 | X0 = 0 X1 = 1 |
| **False Position** | 4 | 0.014932 | 0.56715 | X0 = 0 X1 = 1 |
| **Fixed Point** | 19 | 0.037481 | 0.56713 | X0 = 2 |
| **Newton Raphson** | 4 | 0.020580 | 0.56714 | X0 = 0 |
| **Secant** | 3 | 0.014091 | 0.56717 | X0 = 0 X1 = 1 |

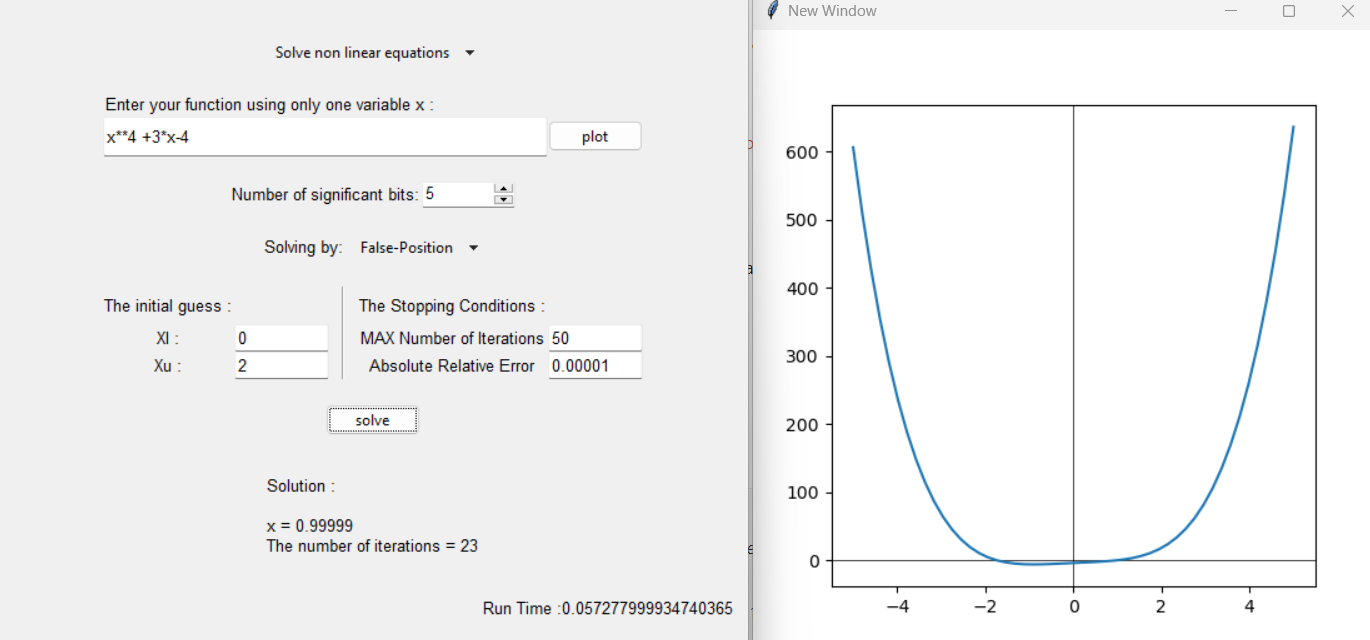
* **Test case 4**

**A screenshot of a computer

Description automatically generated**

**This result appears because the bisection method cannot solve equations that have f(xl) and f(xu) on the same side (have the same sign).**

* **Test case 5**

****

**A screenshot of a computer

Description automatically generated**

**From this result we can notice that by changing the interval the root found by the method changes.**

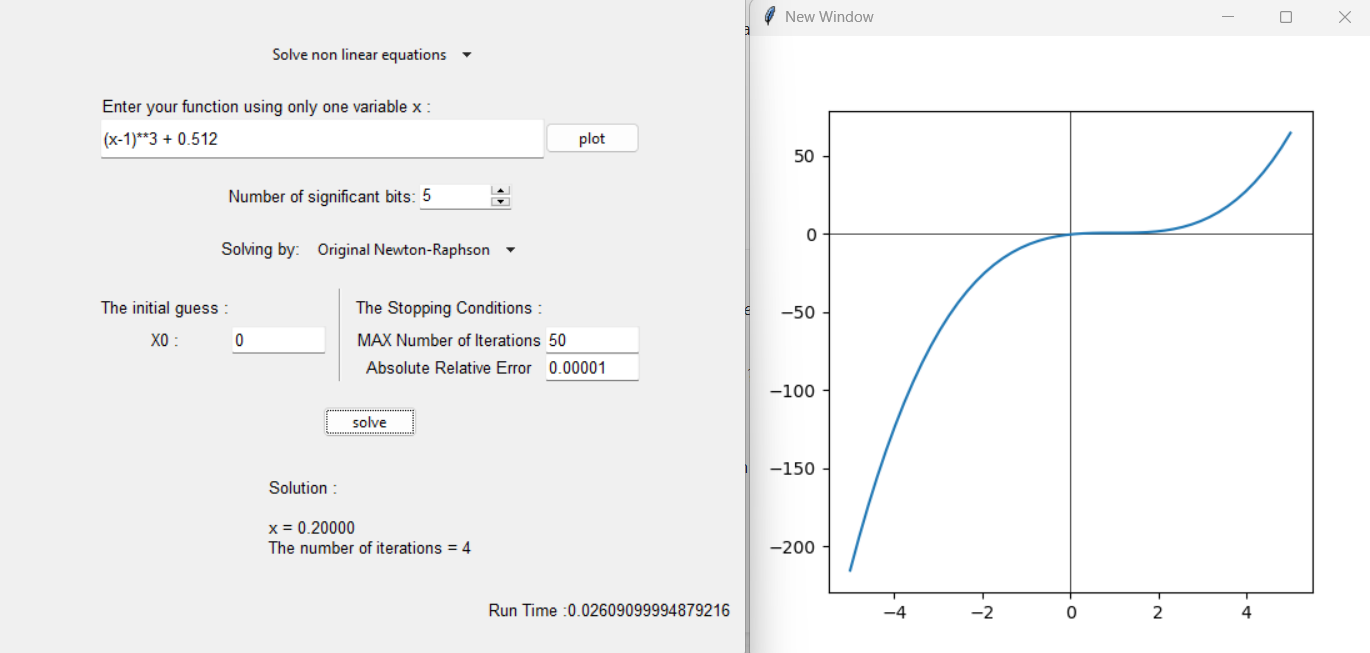
* **Test case 6**

**A screenshot of a computer

Description automatically generated**

We can notice that the system is diverging, as the |g’(x)|> 1

* **Test case 7**

****

* **Test case 8**

**A screen shot of a graph

Description automatically generated**

Secant method successfully found a root of the equation sin(x) at x=0 after only 1 iteration.

**Sample runs for the GUI:**

1. The following window will appear when the app is launched.

A screenshot of a computer

Description automatically generated

1. You can choose the system from the provided list.

A screenshot of a computer

Description automatically generated

1. The following window will appear after choosing the the system.

A screenshot of a computer

Description automatically generated

1. You can choose the the method from the provided list.

A screenshot of a computer

Description automatically generated

1. Enter your attributes, then click solve to solve the system.

A screenshot of a computer

Description automatically generated