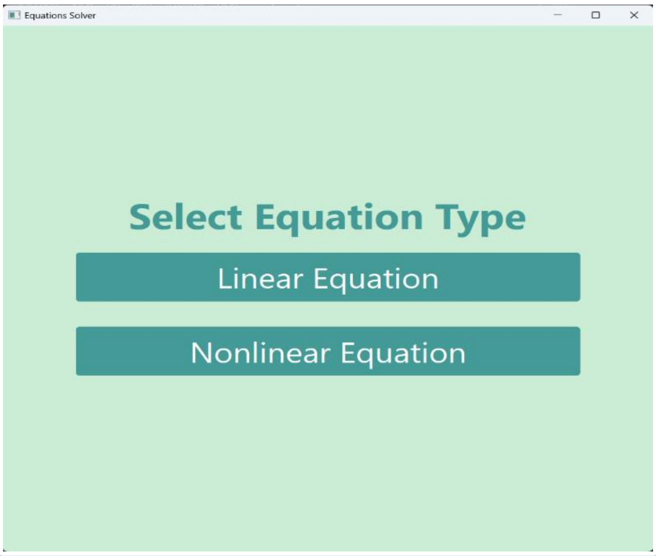
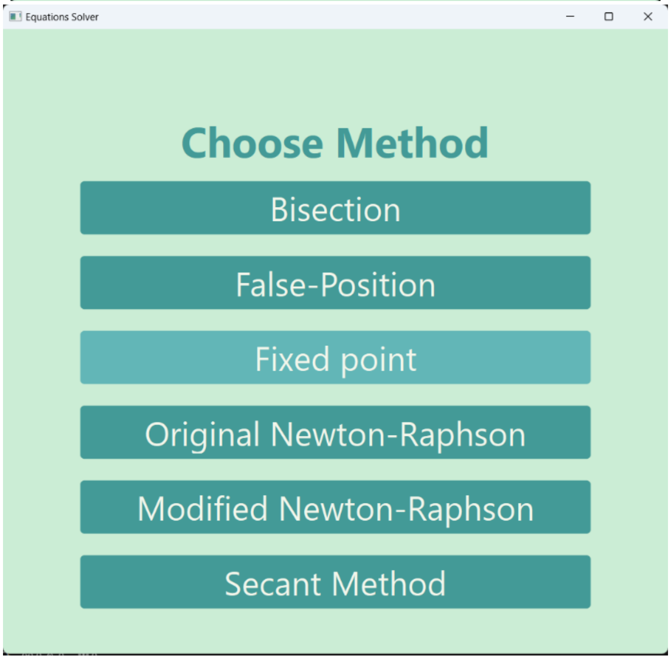




Numerical Project Phase 2

Shahd Mohamed	22010848
Menan Mohamed	22011265
Youssef Ahmed	22011369
Omar Zydan	22010966
Abdelrahman Elsayed	22010869

Pseudo-code for methods

- **Bisection**

```
def solve( func, a, b, tolerance, max_iterations, SF):
  error = 1
  counter = 0

  while error >= tolerance  and  counter <= max_iterations:
    x = (a + b) / 2                          # Compute midpoint
    if func(a)*f(b):                         # no root in this interval
      return "no root"

    counter += 1
    if func(x) == 0:  # Exact root found
     return x
    if func(x) * func(a) < 0:  # Narrow interval
      b = x
    else:
      a = x
```

- ## False-Position

```
def solve(func, a, b, tolerance, max_iterations):
  error = 1
  counter = 0
  while error >= tolerance and counter <= max_iterations:
    x = b - (func(b) * (b - a)) / (func(b) - func(a))          # Compute x using Regula Falsi formula
    counter += 1
    if func(x) == 0:                              # Exact root found
      break
    if func(x) * func(a) < 0:
      b = x
    else:
      a = x
```

- ## Fixed Point

```
def solve( g, x0, tolerance, max_iterations):
  error = 1
  counter = 0
  while error >= tolerance and counter <= max_iterations:
    x1 = g(x0)                    # Compute the next approximation
    counter += 1
    error = abs(x1 - x0)          # Update error
    if error < tolerance:         # Check if the solution has converged
      break

    x0 = x1                       # Update for the next iteration
```

- ## Original-Newton

```
def solve( func, dfunc, x0, tolerance, max_iterations):
  error = 1
  counter = 0
  while error >= tolerance and counter <= max_iterations:
    x1 = x0 - func(x0) / dfunc(x0)                    # Newton-Raphson formula
    counter += 1

    error = abs(x1 - x0)                        # Update error
    if error < tolerance:                       # Check if the solution has converged
      break

    x0 = x1                                     # Update for the next iteration
```

- ## Modified-Newton

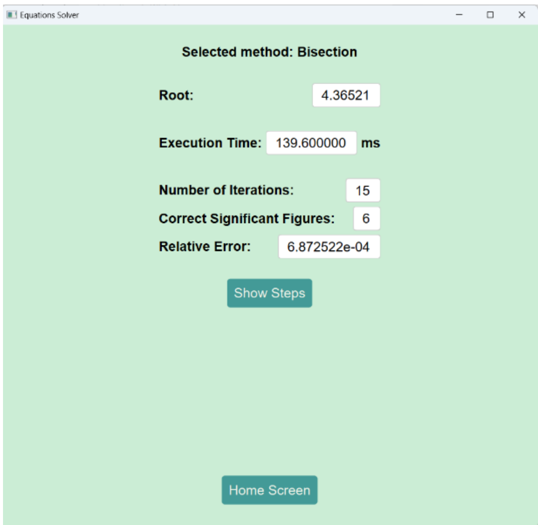
```
def solve( func, x0, delta, tolerance, max_iterations):
  error = 1
  counter = 0

  while error >= tolerance and counter <= max_iterations:
    x1 = x0 - func(x0) * dfunc(x0) / ((dfunc(x0))**2 - func(x0)*ddfunc(x0))
                                                               # Modified Newton formula
    counter += 1
    error = abs(x1 - x0)                        # Update error
    if error < tolerance:                       # Check if the solution has converged
      break

    x0 = x1                                     # Update for the next iteration
```

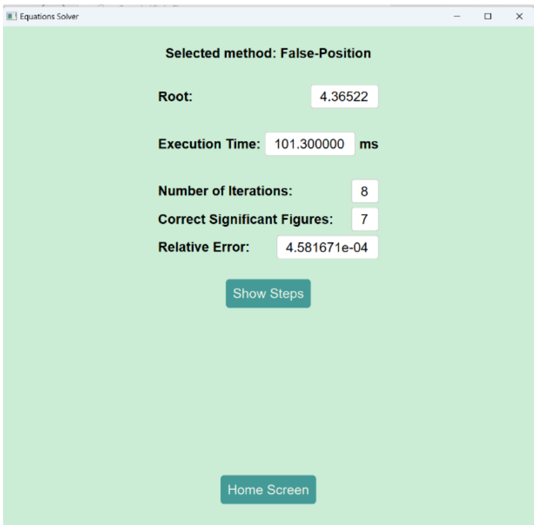
- ## Secant

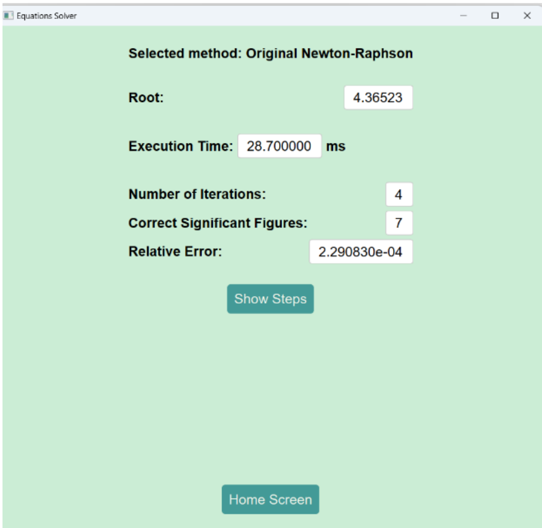
```
def solve( func, x0, x1, tolerance, max_iterations):
  error = 1
  counter = 0
  while error >= tolerance and counter <= max_iterations:
    x2 = x1 - func(x1) * (x1 - x0) / (func(x1) - func(x0))                    # Secant formula
    counter += 1

    error = abs(x2 - x1)                     # Update error
    if error < tolerance:                    # Check if the solution has converged
      break

    x0, x1 = x1, x2                          # Update points for the next iteration
```

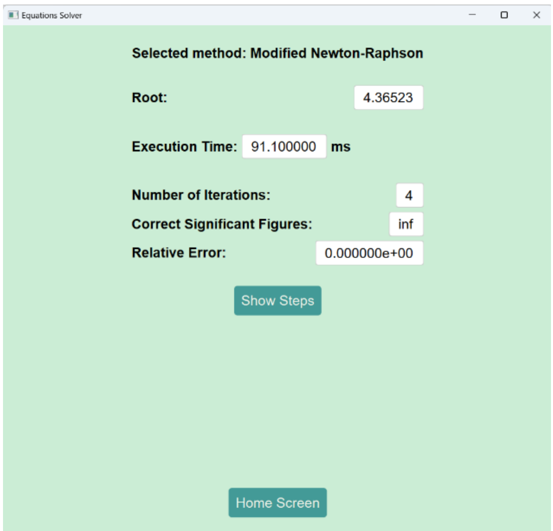
Test Cases

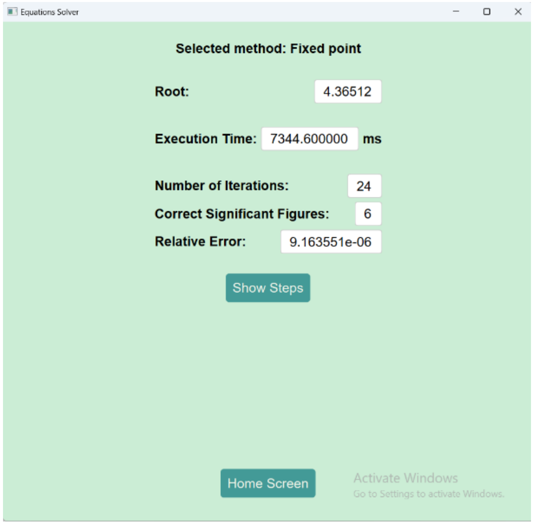
Case 1

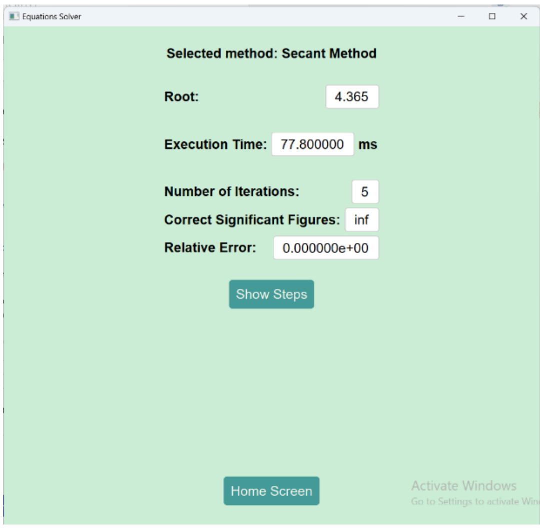










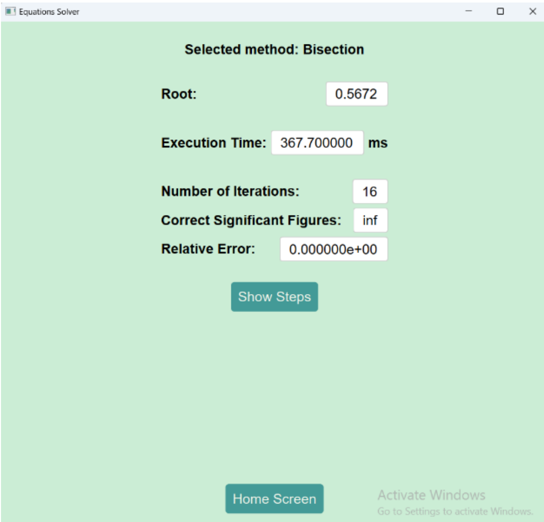


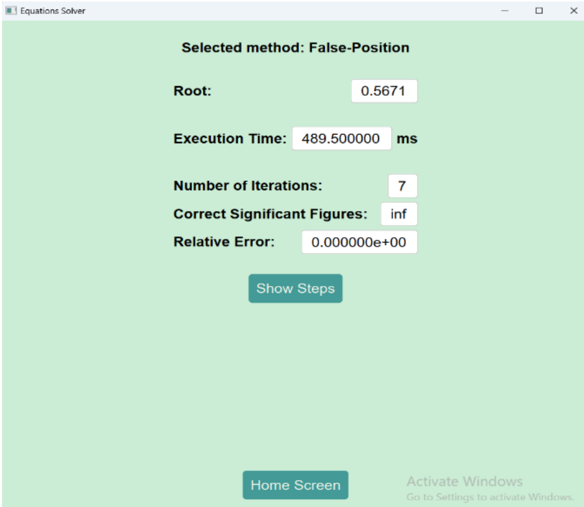
Method	Num iterations	Run time	App.Root	Initial guesses	Relative error
Bisection	15	139	4.36521	4,5	6e-4
False-Position	8	101	4.36522	4,5	4e-4
Original Newton	4	28	4.36523	4	2e-4
Modified Newton	4	91	4.36523	4	0
Fixed Point	24	7344	4.36512	4	9e-6
Secant	5	77	4.365	4,5	0

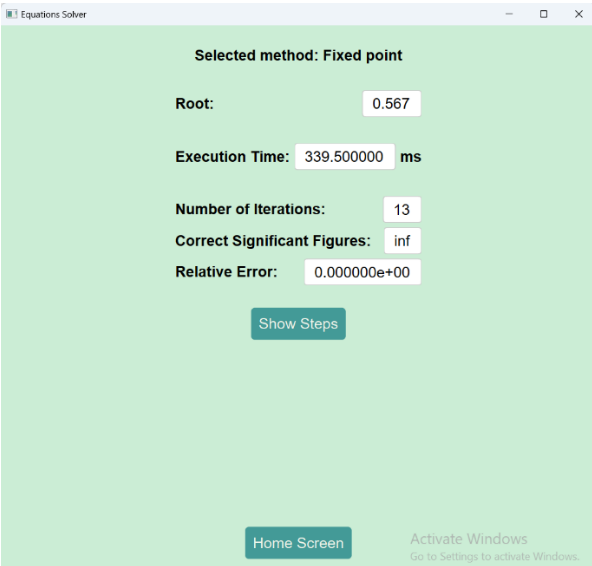
#From table Fixed point takes the largest number of iterations

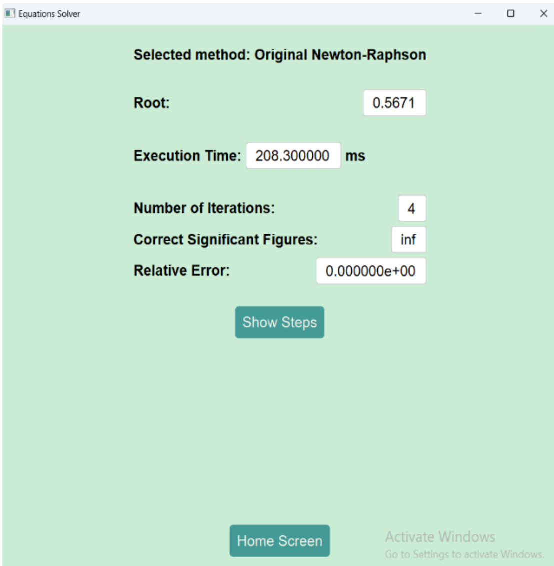
#Newton is the most efficient method to get the roots for this problem

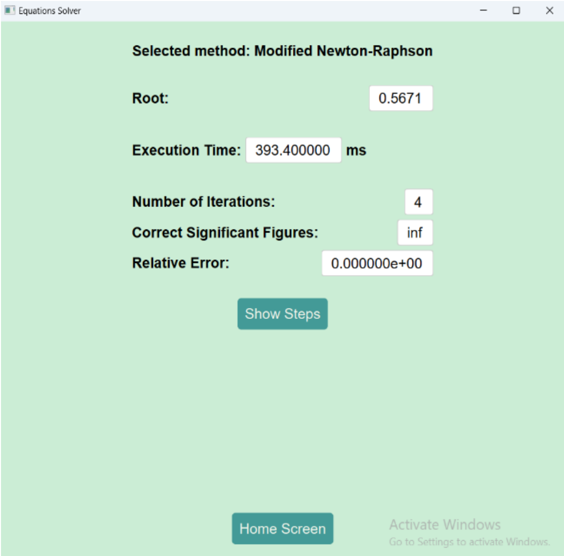
Case 2

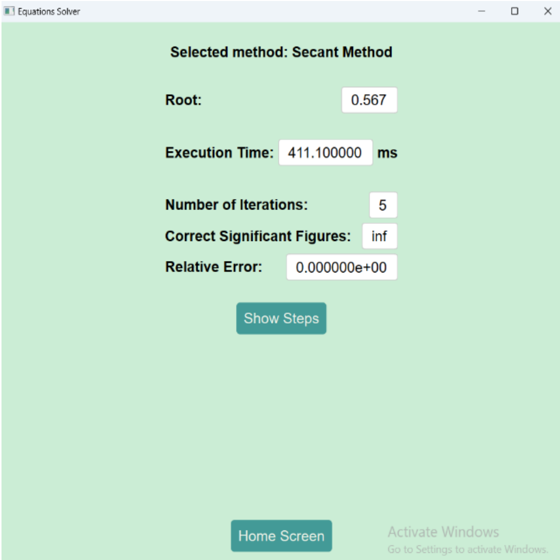

Equations Solver
Selected method: Bisection
Root: 0.5672
Execution Time: 367.700000 ms
Number of Iterations: 16
Correct Significant Figures: inf
Relative Error: 0.000000e+00
Show Steps
Home Screen
Activate Windows
Go to Settings to activate Windows.


Equations Solver
Selected method: False-Position
Root: 0.5671
Execution Time: 489.500000 ms
Number of Iterations: 7
Correct Significant Figures: inf
Relative Error: 0.000000e+00
Show Steps
Home Screen
Activate Windows
Go to Settings to activate Windows.


Equations Solver
Selected method: Fixed point
Root: 0.567
Execution Time: 339.500000 ms
Number of Iterations: 13
Correct Significant Figures: inf
Relative Error: 0.000000e+00
Show Steps
Home Screen
Activate Windows
Go to Settings to activate Windows.


Equations Solver
Selected method: Original Newton-Raphson
Root: 0.5671
Execution Time: 208.300000 ms
Number of Iterations: 4
Correct Significant Figures: inf
Relative Error: 0.000000e+00
Show Steps
Home Screen
Activate Windows
Go to Settings to activate Windows.





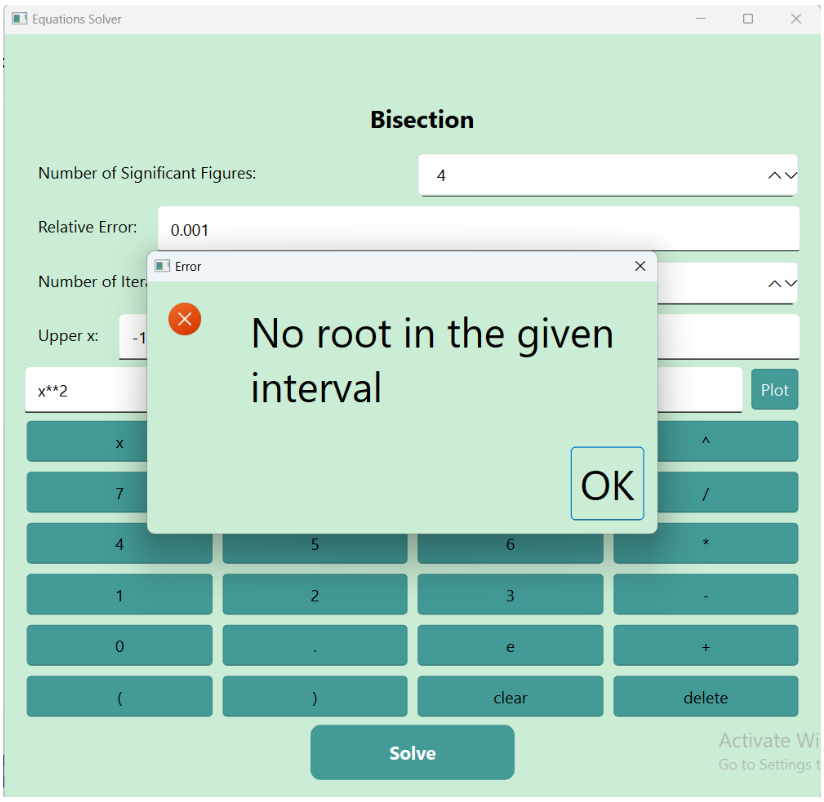
Method	Num iterations	Run time	App.root	Initial guesses	Relative error
Bisection	16	367	0.5672	0,2	0
False-Position	7	489	0.5671	0,2	0
Original Newton	4	208	0.5671	0	0
Modified Newton	4	393	0.5671	0	0
Fixed point	13	339	0.567	0	0
Secant	5	411	0.567	0,2	0

#From table Bisection takes the largest number of iterations

#Newton is the most efficient method to get the roots for this problem

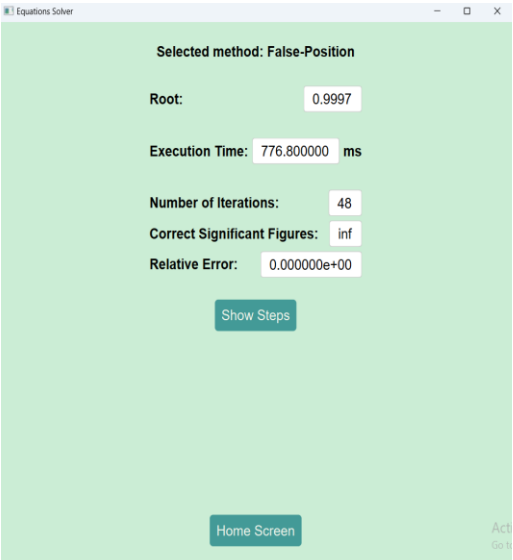
#All relative errors are close to zero

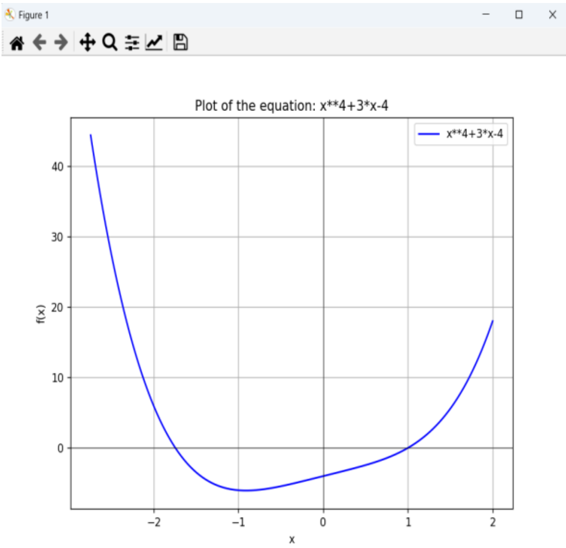
Case 3



The given initial guesses have the same sign which makes Bisection method fails

Case 4

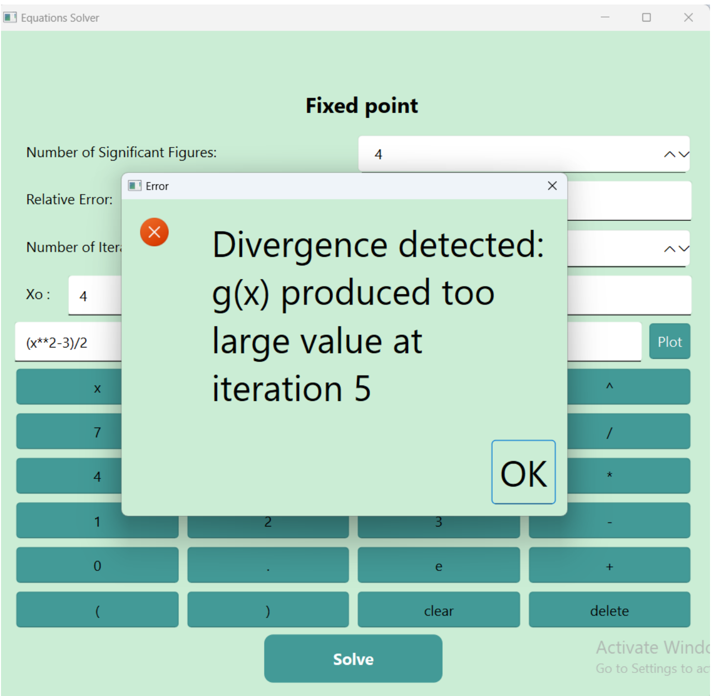




#From graph the result is so close to the true value but takes many iterations

#False-Position works because the initial guesses have different signs

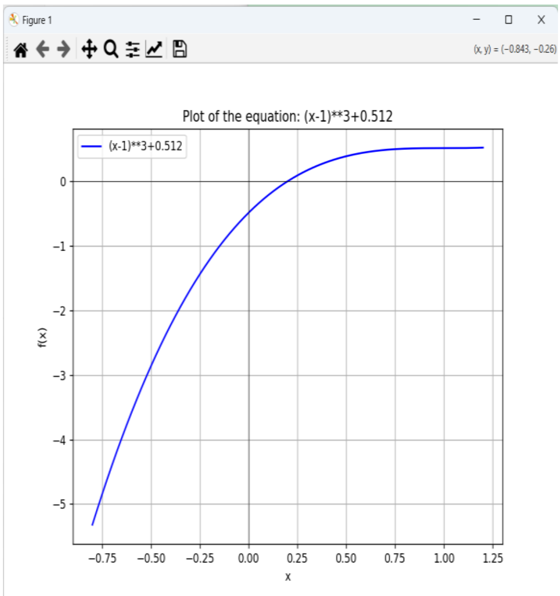
Case 5

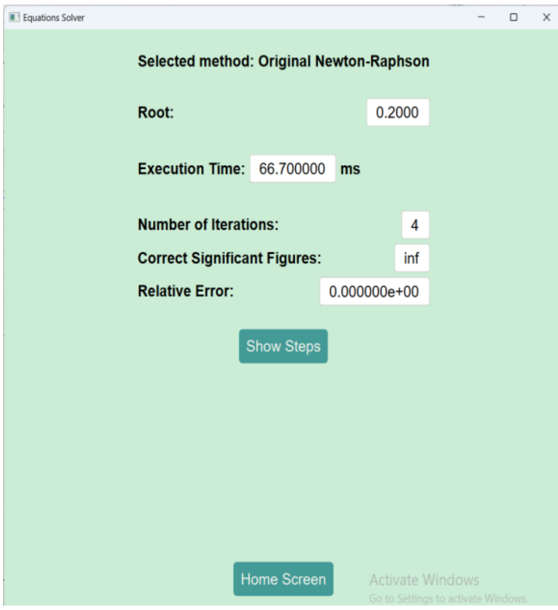


#Becuase the derivative of the given function is greater than one, it diverges

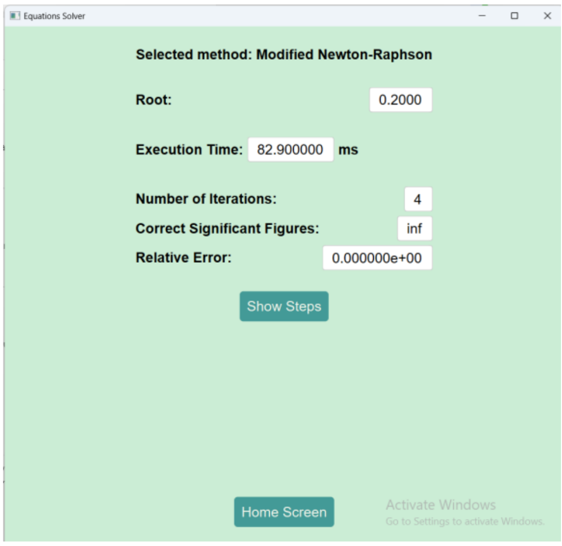
Case 6

Original



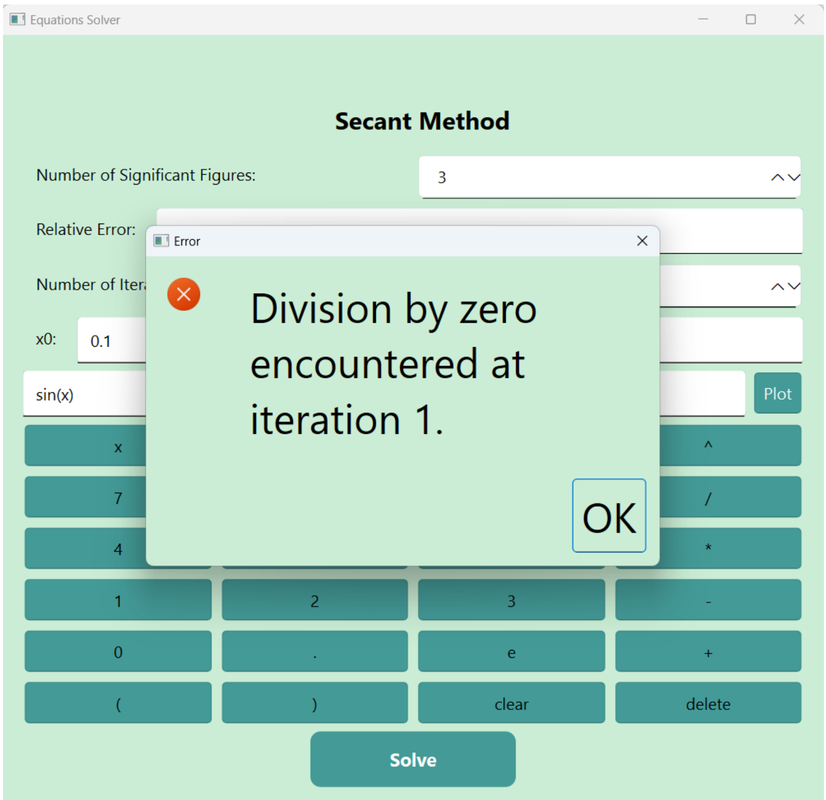


Modified



#Original and Modified method have the same result at the same number of iterations

Case 7



#The denominator in secant formula is very close to zero and the method diverges