Alexandria University

Faculty of Engineering

Computer and Systems Engineering Dept.

Numerical Analysis

Project Phase 2

نور الدين حسن محمد قباري حسن ID: 19016802
عمر مصطفى حسن متولى صفا ID: 19016099
مصطفى عبد الحليم عبد المحسن عبد الحليم الطوى ID: 19016662
مصطفى فراج مصطفى محمد فراج ID: 19016666
جمال عبد الحميد ناصف نويصر ID: 19015550

➢ Pseudo Code
Bracketing Methods

```
class Bracketing:
    BEGIN get_plot():
        interval ← (x_upper − x_lower) / 100
        initialize empty list x_values
        initialize empty list fx_values
        BEGIN FOR 101 times, index i:
            add x lower + i * interval to x_values
            add f(x lower + i * interval) to fx_values
        END FOR
        fig_size ← (10,5)
        fig ← new figure(fig_size)
        plot x_values, fx_values
        plot x-axis
        plot y-axis
        plot grid
        limit plot x-axis to range [x_lower, x_upper]
        RETURN fig
    END get_plot()
    BEGIN get_f_x():
        func_lower ← f(x_lower),
        func_upper ← f(x_upper)
        RETURN func_lower, func_upper
    END get_f_x()
```

```
BEGIN check_tolerance():
    approx_rel_error ← |(x_middle − x_middle_old) / x_middle|
    x_middle_old ← x_middle
    BEGIN IF (approx_rel_error ≤ tolerance):
        RETURN true
    ELSE:
        RETURN false
    END IF
END check_tolerance()
class Bisection inherits from Bracketing:
BEGIN solve(i):
    func_lower, func_upper ← get_f_x()
    BEGIN IF (product of func_lower, func_upper is positive):
        THROW error("Does not converge")
    END IF
    x_middle ← (x_upper + x_lower) / 2
    BEGIN IF (check_tolerance()):
        RETURN x_middle
    END IF
    func_middle ← f(x_middle)
    BEGIN IF (i = max iterations – 1):
        RETURN x_middle
    END IF
    BEGIN IF (func_middle * func_lower is negative):
```

```
            x_lower = $x_{middle}$
            solve(i+1)
        ELSE IF (func_middle * func_lower is positive):
            $x_{upper}$ = $x_{middle}$
            solve(i+1)
        ELSE:
            RETURN $x_{middle}$
        END IF
class FalsePosition inherits from Bracketing:
    BEGIN solve(i):
        func_lower, func_upper ← get_f_x()
        BEGIN IF (product of func_lower, func_upper is positive):
            THROW error("Does not converge")
        END IF
        $x_{middle} \leftarrow f(x_{upper}) \times \frac{x_{upper}-x_{lower}}{f(x_{upper})-f(x_{lower})}$
        BEGIN IF (check_tolerance()):
            RETURN $x_{middle}$
        END IF
        func_middle ← $f(x_{middle})$
        BEGIN IF (i = max iterations – 1):
            RETURN $x_{middle}$
        END IF
        BEGIN IF (func_middle * func_lower is negative):
            $x_{lower}$ = $x_{middle}$
```

```
    solve(i+1)
ELSE IF (func_middle * func_lower is positive):
    x_upper = x_middle
    solve(i+1)
ELSE:
    RETURN x_middle
END IF
```

Fixed point Method

```
class FixedPointIteration:

  Begin evaluate(value,function):
      x=value
      return f(x)

End evaluate

   Begin getg_x():

       f→self.equation
       getting f'(x)

       get ∫ f'(x)dx

       constant → simplify(f'(x) -  ∫ f'(x)dx)

       if constant == 0:
          constant→-10
       subtracting function from constant
       Divide function by X
       g(x) = -constant / f
       return f

   End getg_x

   begin fixedPointIteration(self):

    Try:
      x=symbols('x')
      if equation has one X:
         solve for X

         Plot graph

        return the answer,  graph


      x0→self.initial
      count→0
      error→[]
      get g(x) as String

      Begin while(iterations less than maximum iterations):
         evaluate Xi+1
         compute relative approximate error

         if(count>=1):
              if(error is increasing with each iteration):
               return "diverge"
```

```
      if(error is less than tolerance):
        plot the graph

        return root,  graph

      Xi→Xi+1
      incrementing count

    End While

  Catch Division by zero: return “change the initial guess”

  Catch TypeError: return” Math error”

End  fixedPointIteration
```

Secant Method

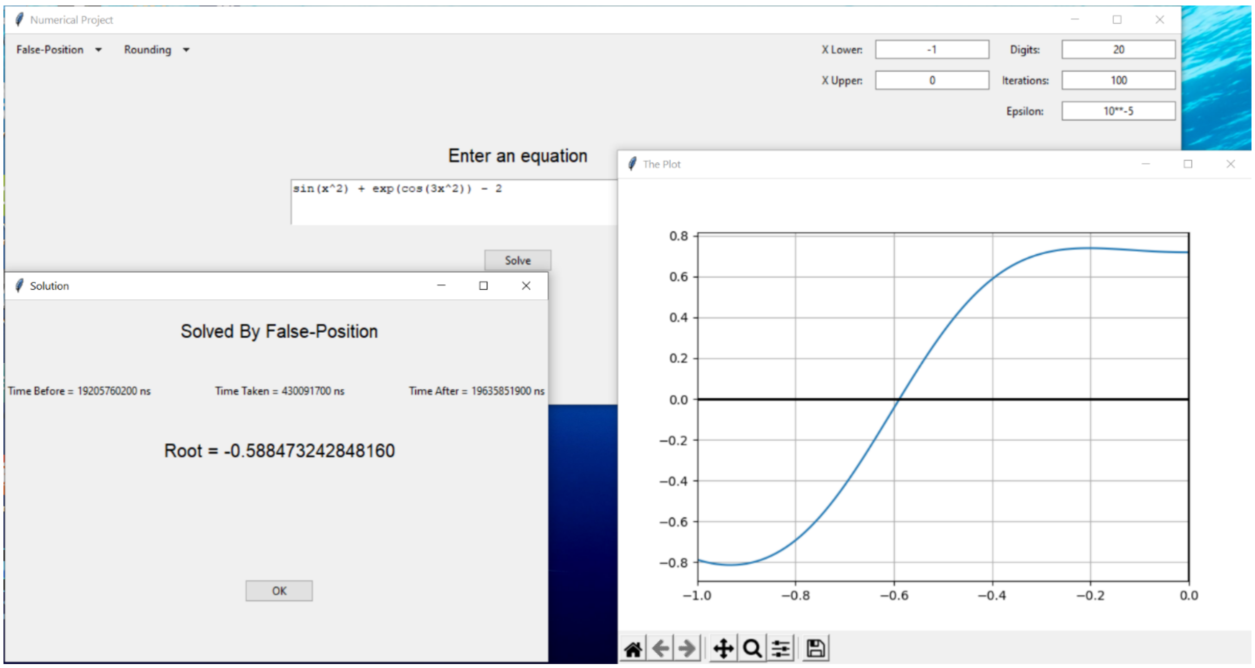
```
Begin class Secant:

oldY ← function(oldX)
y ← function(x)

Begin for iteration  = 0 to ITERATIONS:
    newX ← converter.convert(x - converter.convert(y * converter.convert(converter.convert(x - oldX) / converter.convert(y - oldY))))
    function_plotter(function, functionDerivative, newX)

    eps ← getAbsoluteRelativeError(newX, x)
    Begin if(eps <= EPSILON):
        Break
    End if

    oldX ← x
    oldY ← y
    x ← newX
    y ← function(newX)
End for
```

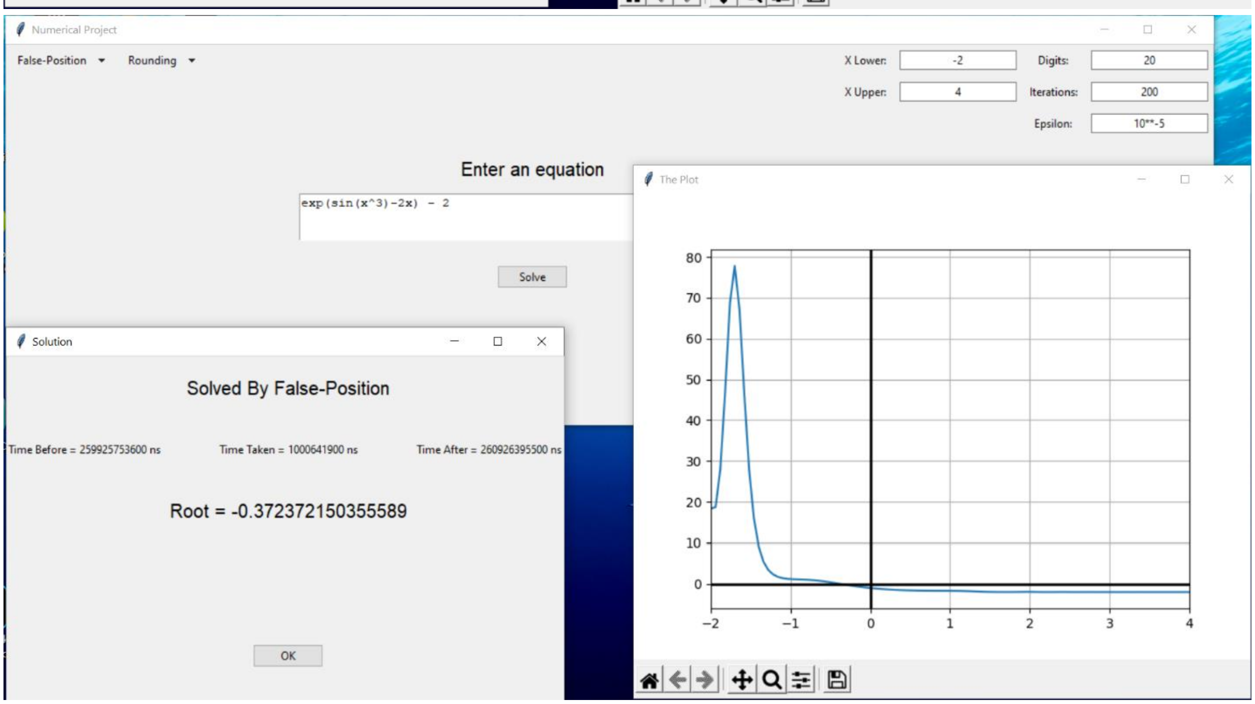
Newton Raphson Method

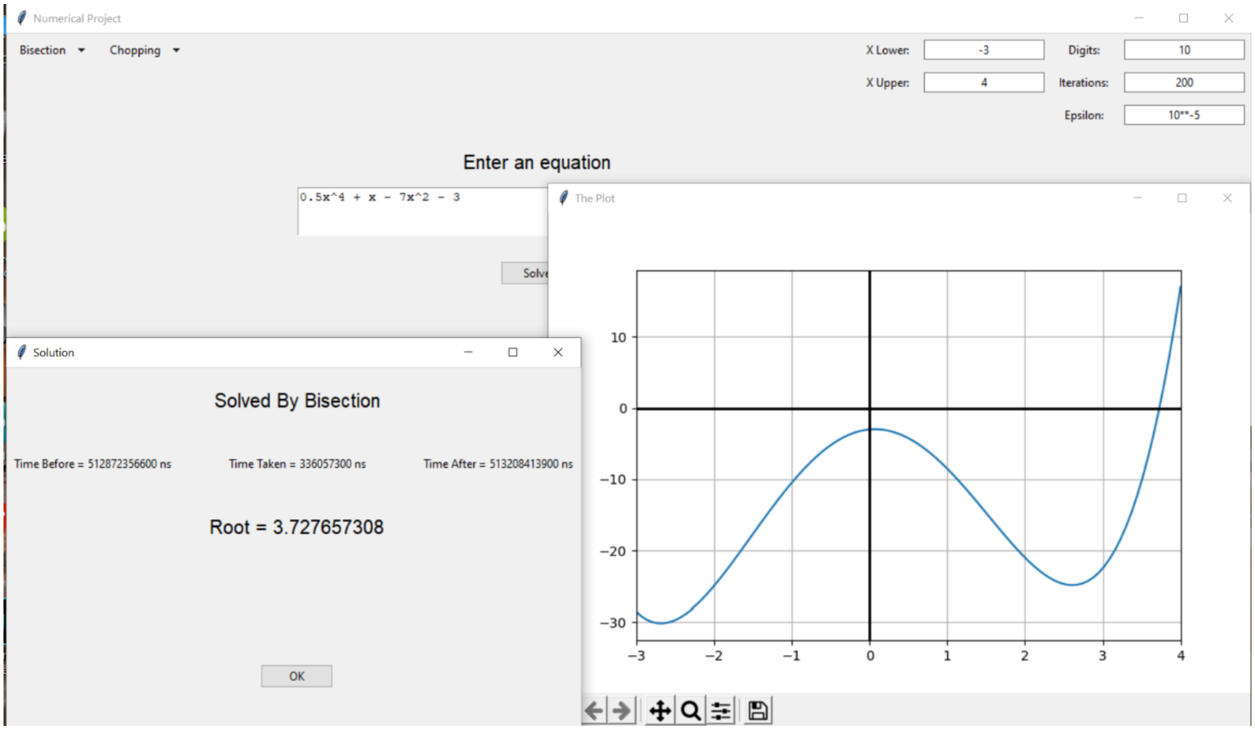
```
class NewtonRaphson:

    NewtonRaphson(function, initialGuess, iterations, eps, floatConverter)

    getApproximateRelativeError( newValue, oldValue):

      if newValue equals 0:
            return infinity
      else:
            return abs( (newValue - oldValue) / newvalue ) * 100

    getDerivative(function):

          derivative = function.differentiate(x)
          return derivative

     solve():

          functionDerivative = getDerivative(function)

          for itertion = 0 to iterations:

             newX = currentX - function(currentX) / functionDerivative(currentX)
             functionPlotter(function, functionDerivative, newX)
             relativeError = getApproximateRelativeError(newX, currentX)
             if relativeError < eps:
                    return newX
             if abs(newX - currentX) > 10^5:
                    return " The method diverges"
             currentX = newX

        return "The method exceeds the iteration before reaching the accuracy"


      functionPlotter(function, derivative, point):

        let range be from point-5 to point+5
        plot function(x) where x belongs to range
        plot derivative(point)*(x - point) + function(point) where x belongs to range
        plot a vetrical line at point from y = 0 to y = function(point)
        plot (point, function(point))
```

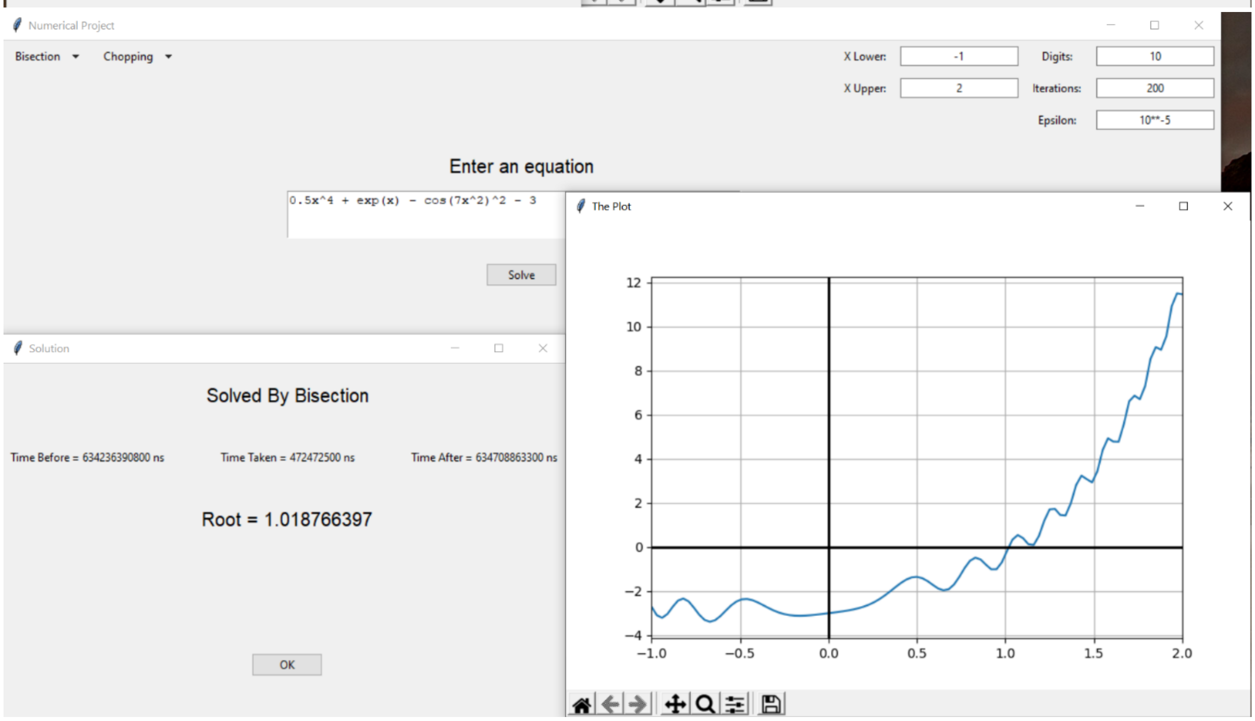
➢ Sample runs

Bracketing Methods

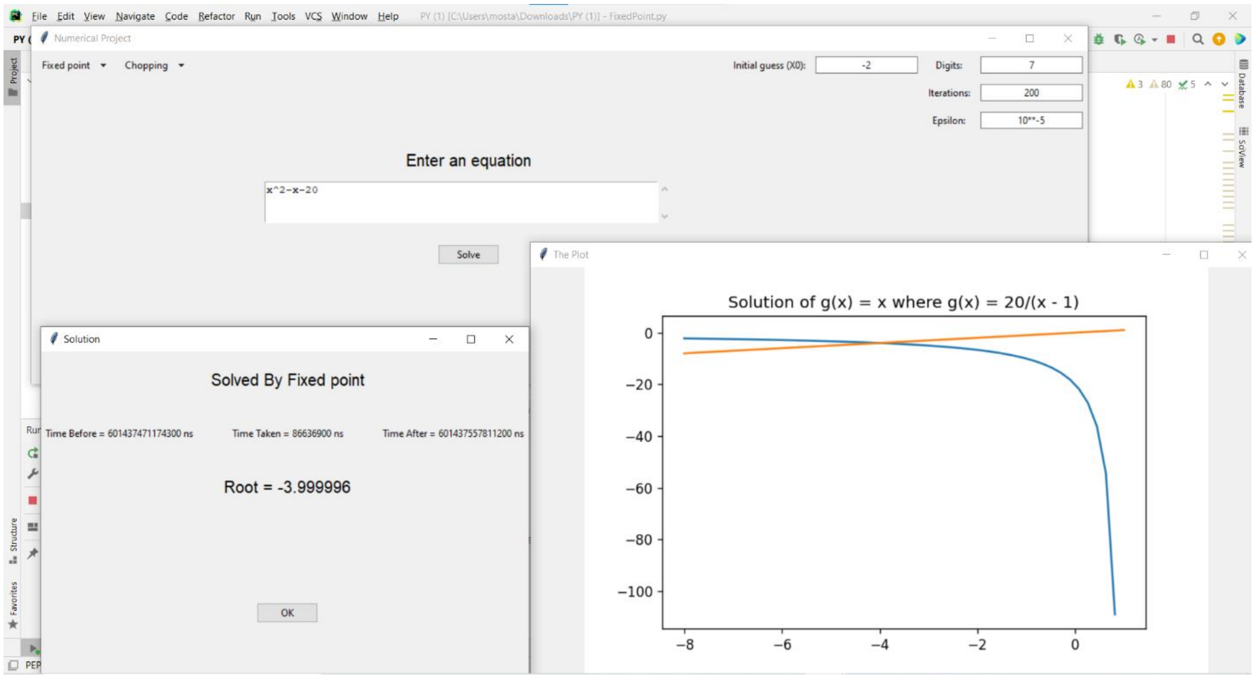


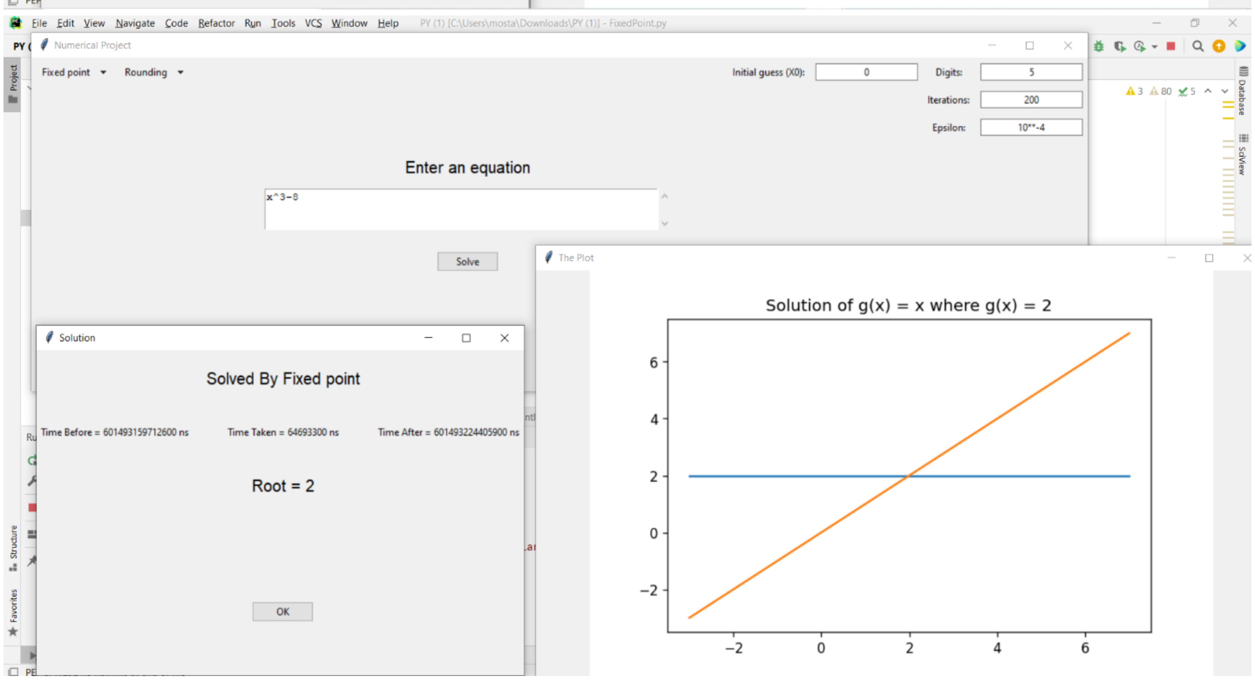


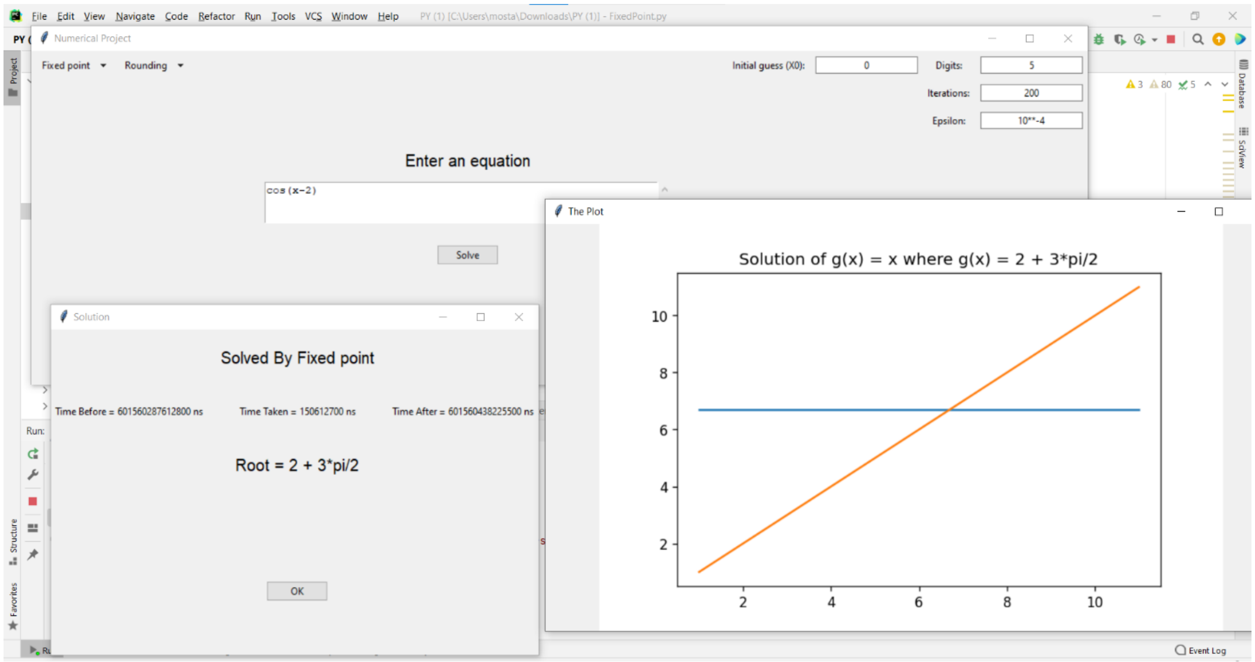

Numerical Project
Bisection
Chopping
X Lower: -3
X Upper: 4
Digits: 10
Iterations: 200
Epsilon: 10**-5
Enter an equation
0.5x^4 + x - 7x^2 - 3
The Plot
Solution
Solved By Bisection
Time Before = 512872356600 ns
Time Taken = 336057300 ns
Time After = 513208413900 ns
Root = 3.727657308
OK

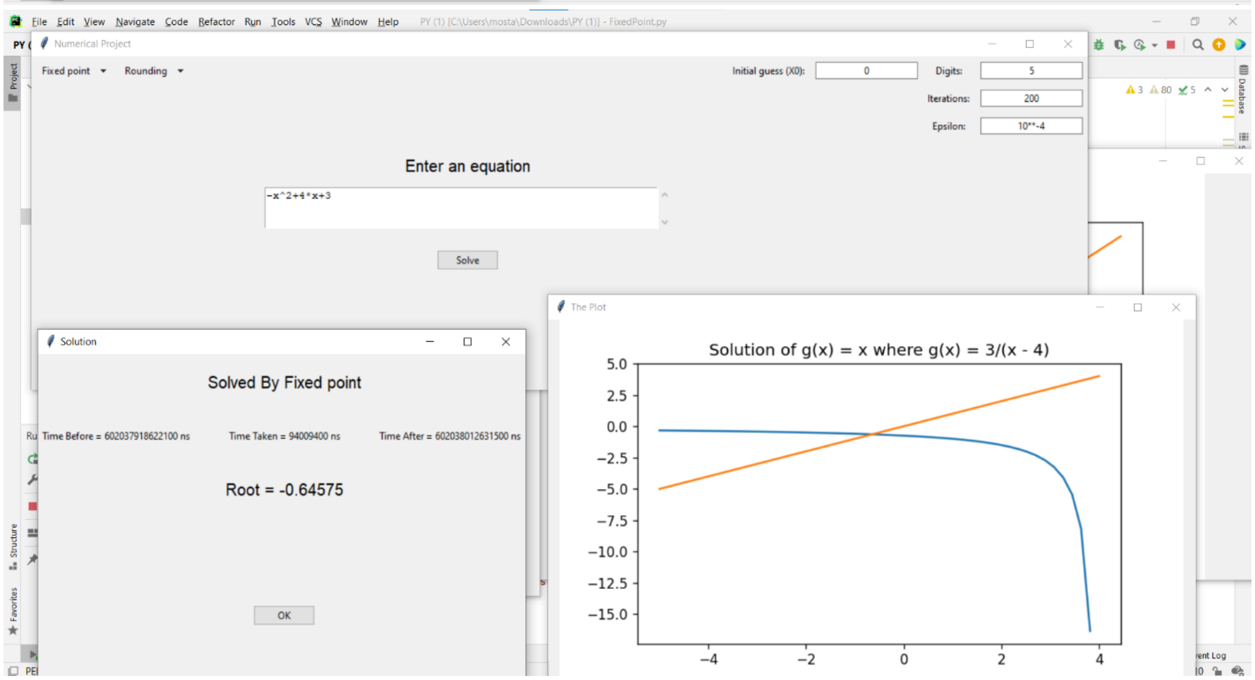

Numerical Project
Bisection
Chopping
X Lower: -1
X Upper: 2
Digits: 10
Iterations: 200
Epsilon: 10**-5
Enter an equation
0.5x^4 + exp(x) - cos(7x^2)^2 - 3
Solve
The Plot
Solution
Solved By Bisection
Time Before = 634236390800 ns
Time Taken = 472472500 ns
Time After = 634708863300 ns
Root = 1.018766397
OK

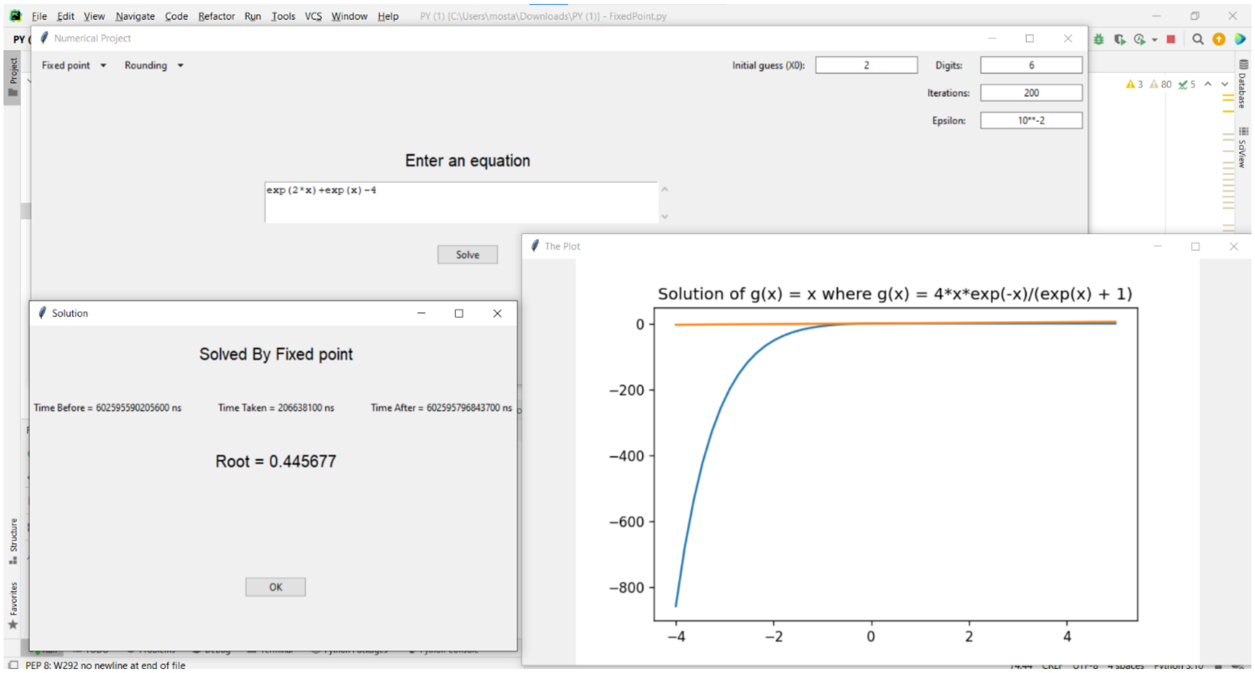
Fixed Point Method



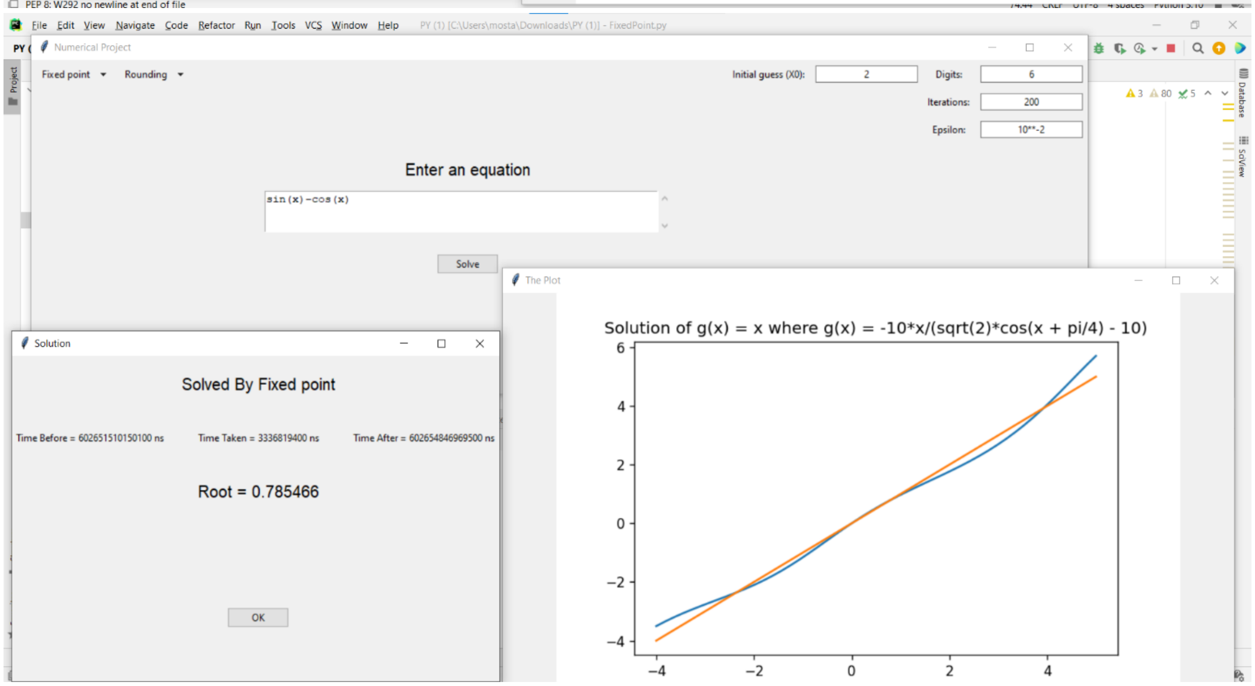



Numerical Project
Fixed point
Rounding
Initial guess (X0): 0
Digits: 5
Iterations: 200
Epsilon: 10**-4
Enter an equation
cos(x-2)
Solve
Solution
Solved By Fixed point
Time Before = 601560287612800 ns
Time Taken = 150612700 ns
Time After = 601560438225500 ns
Root = 2 + 3*pi/2
OK
The Plot
Solution of g(x) = x where g(x) = 2 + 3*pi/2

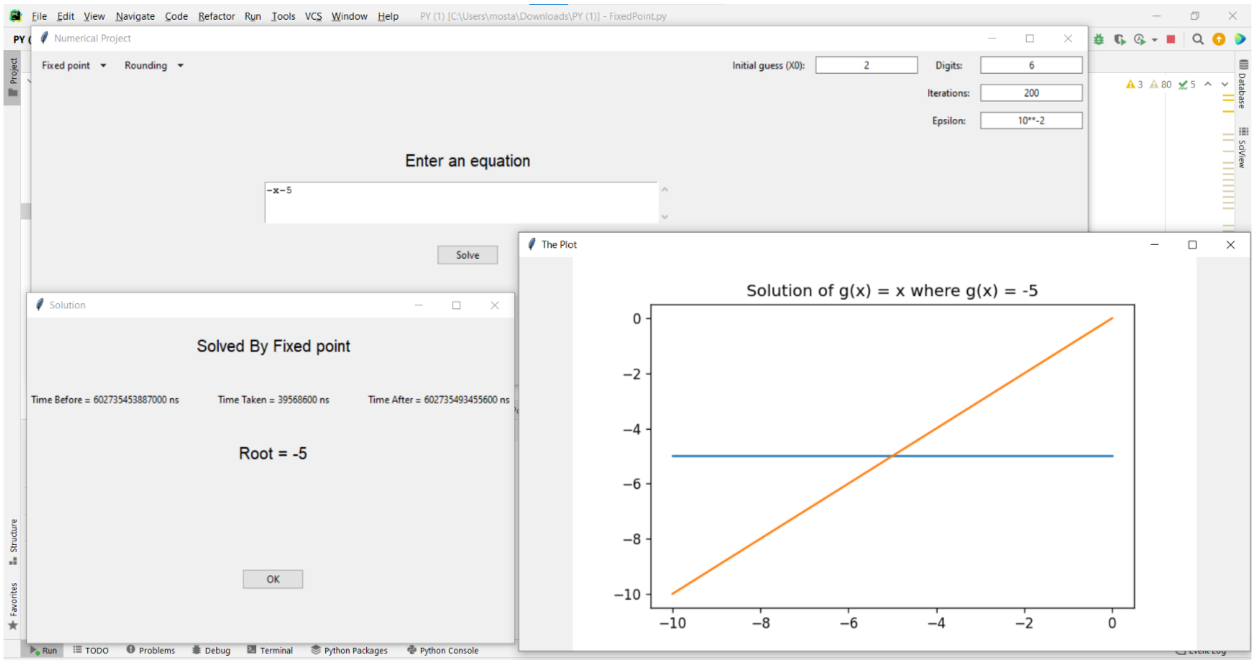

Numerical Project
Fixed point
Rounding
Initial guess (X0): 0
Digits: 5
Iterations: 200
Epsilon: 10**-4
Enter an equation
-x^2+4*x+3
Solve
Solution
Solved By Fixed point
Time Before = 602037918622100 ns
Time Taken = 94009400 ns
Time After = 602038012631500 ns
Root = -0.64575
OK
The Plot
Solution of g(x) = x where g(x) = 3/(x - 4)



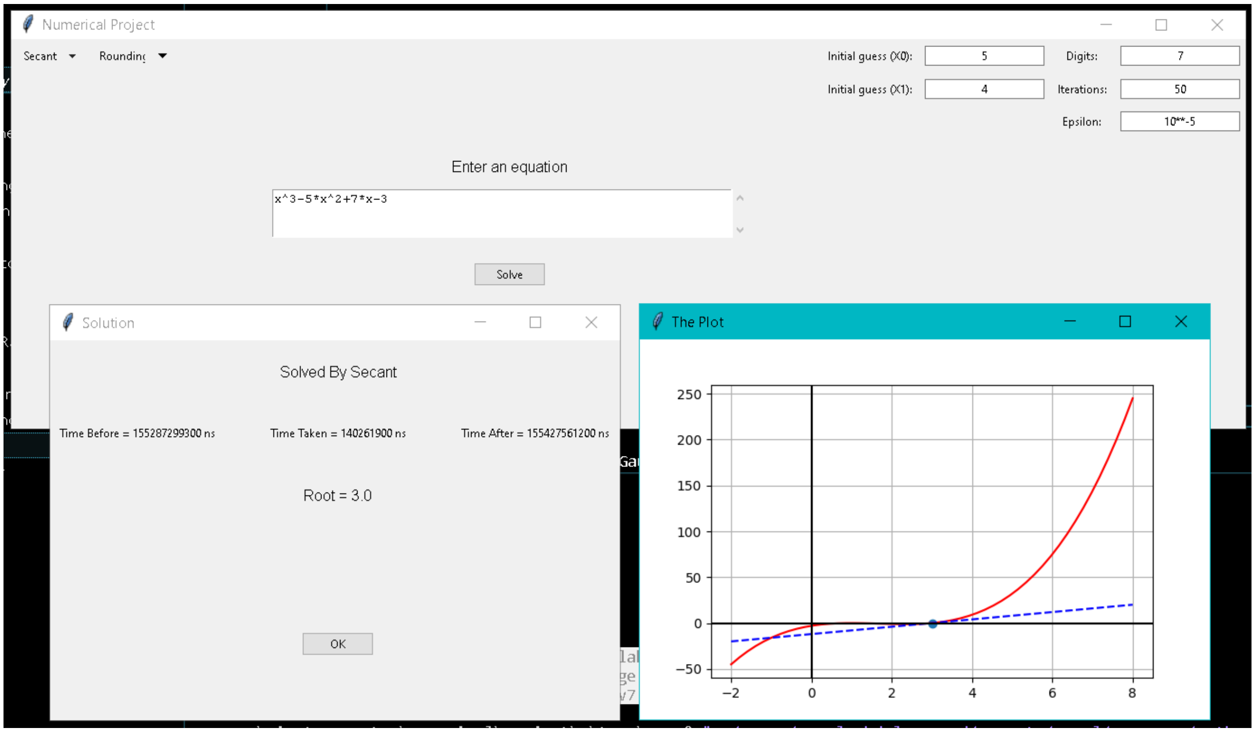
Numerical Project
Fixed point
Rounding
Initial guess (X0):
2
Digits:
6
Iterations:
200
Epsilon:
10**-2
Enter an equation
exp(2*x)+exp(x)-4
Solve
Solution
Solved By Fixed point
Time Before = 602595590205600 ns
Time Taken = 206638100 ns
Time After = 602595796843700 ns
Root = 0.445677
OK
The Plot
Solution of g(x) = x where g(x) = 4*x*exp(-x)/(exp(x) + 1)
0
−200
−400
−600
−800
−4
−2
0
2
4
PEP 8: W292 no newline at end of file

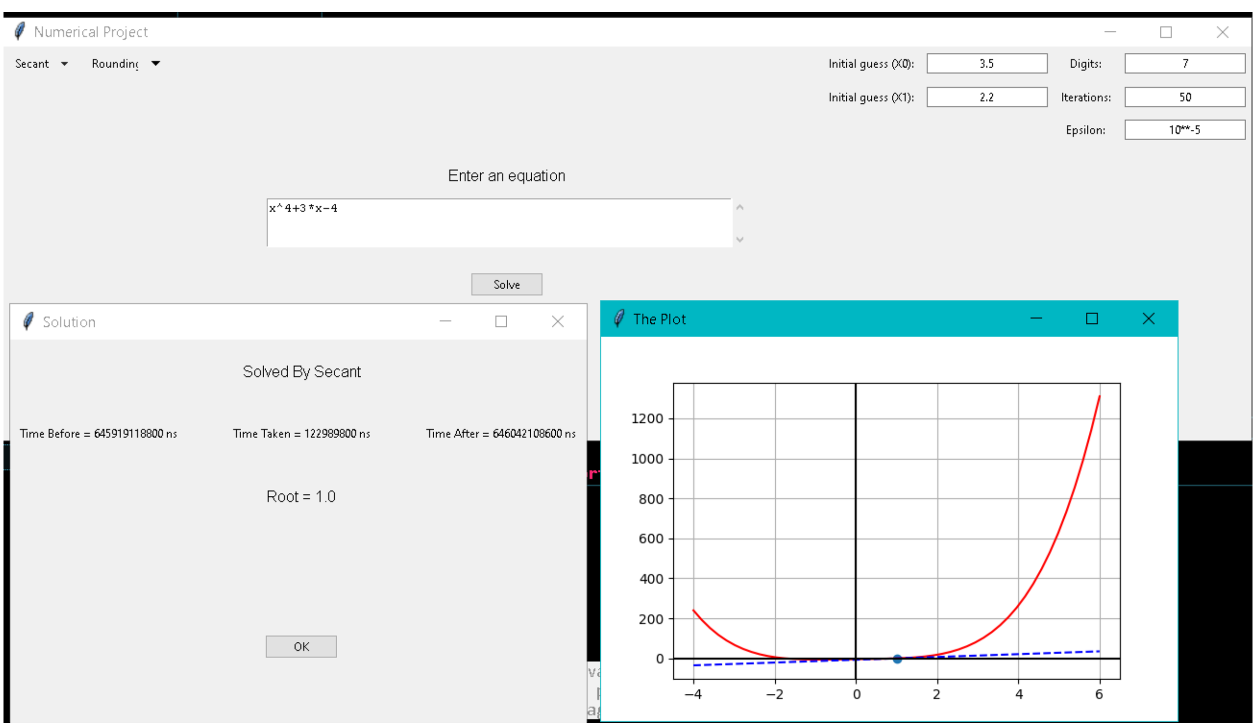


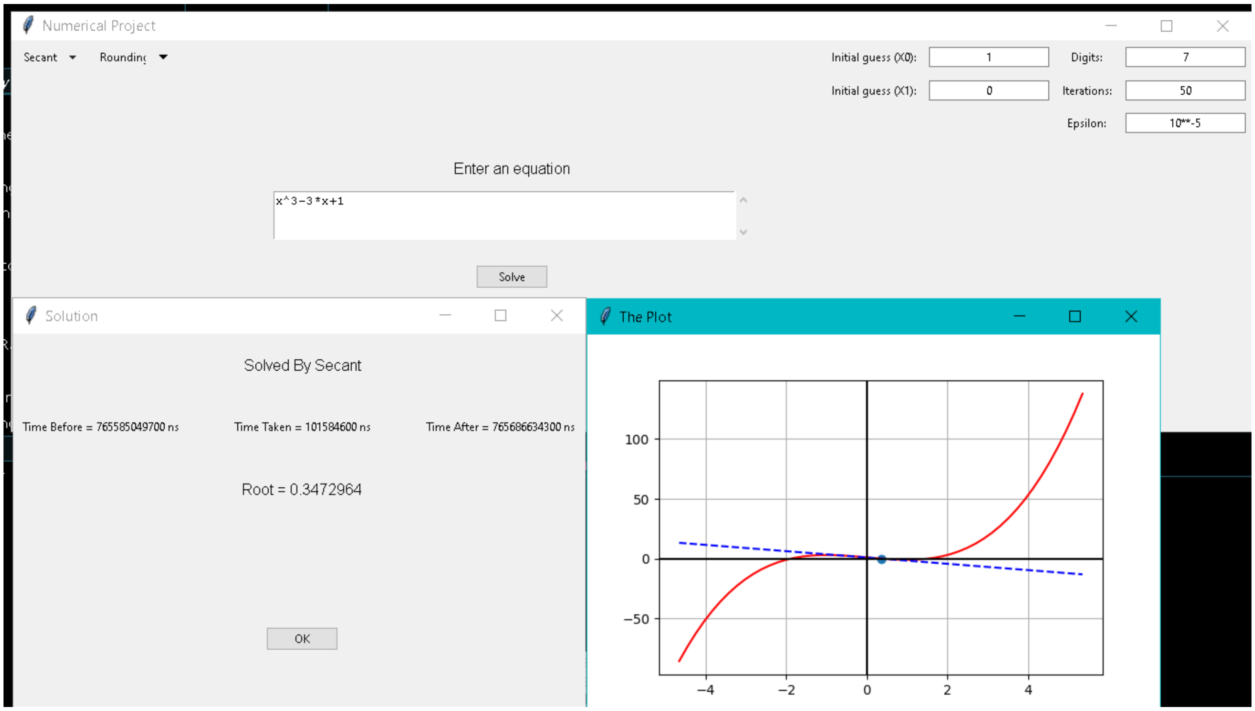
Numerical Project
Fixed point
Rounding
Initial guess (X0):
2
Digits:
6
Iterations:
200
Epsilon:
10**-2
Enter an equation
sin(x)-cos(x)
Solve
Solution
Solved By Fixed point
Time Before = 602651510150100 ns
Time Taken = 3336819400 ns
Time After = 602654846969500 ns
Root = 0.785466
OK
The Plot
Solution of g(x) = x where g(x) = -10*x/(sqrt(2)*cos(x + pi/4) - 10)
6
4
2
0
−2
−4
−4
−2
0
2
4


Numerical Project
Fixed point
Rounding
Initial guess (X0):
2
Digits:
6
Iterations:
200
Epsilon:
10**-2
Enter an equation
-x-5
Solve
Solution
Solved By Fixed point
Time Before = 602735453887000 ns
Time Taken = 39568600 ns
Time After = 602735493455600 ns
Root = -5
OK
The Plot
Solution of g(x) = x where g(x) = -5

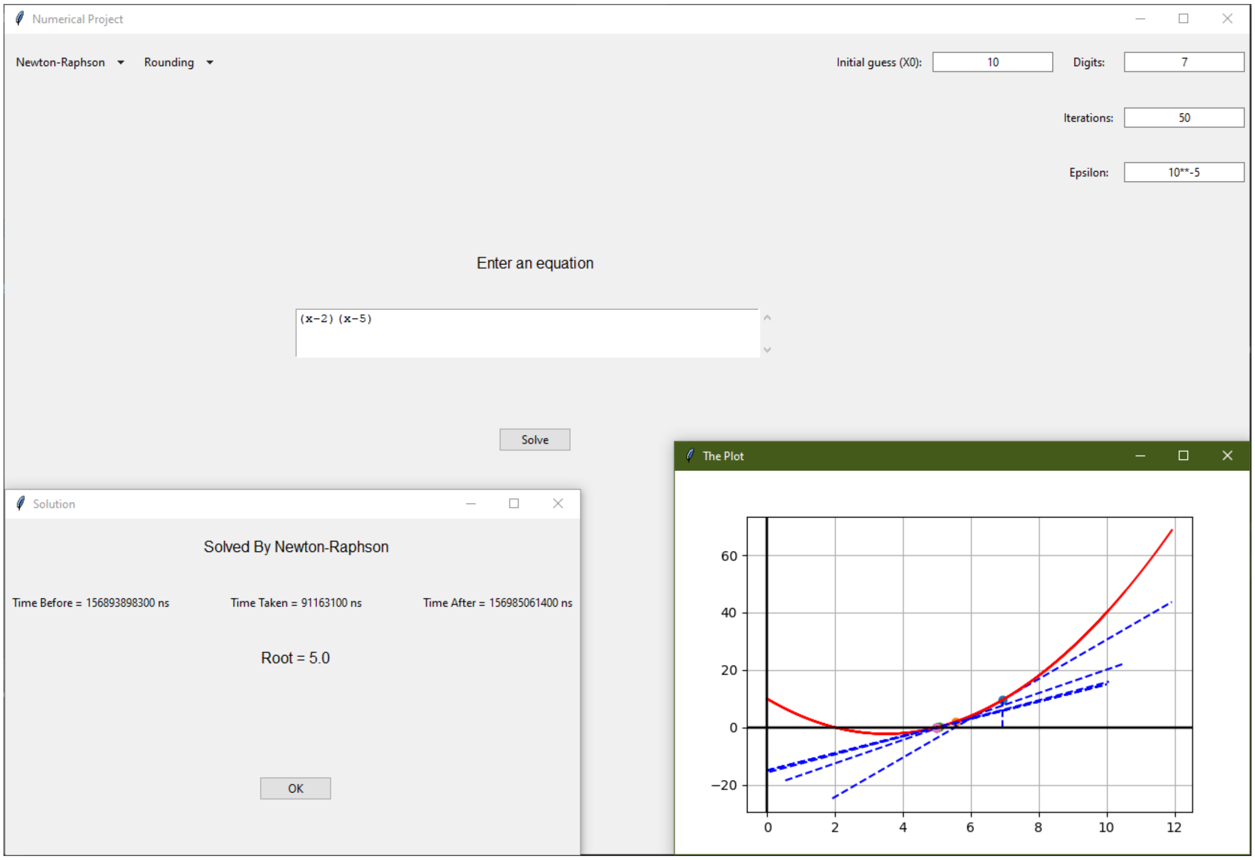
Secant Method

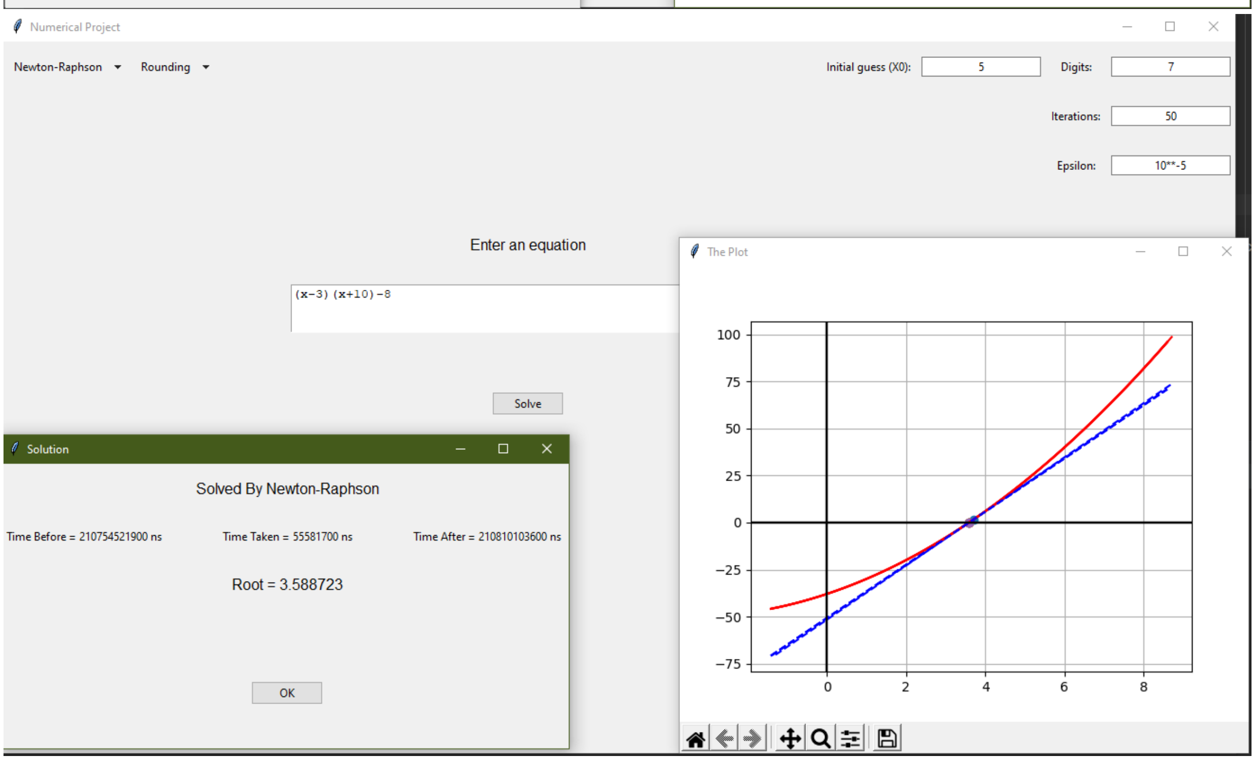


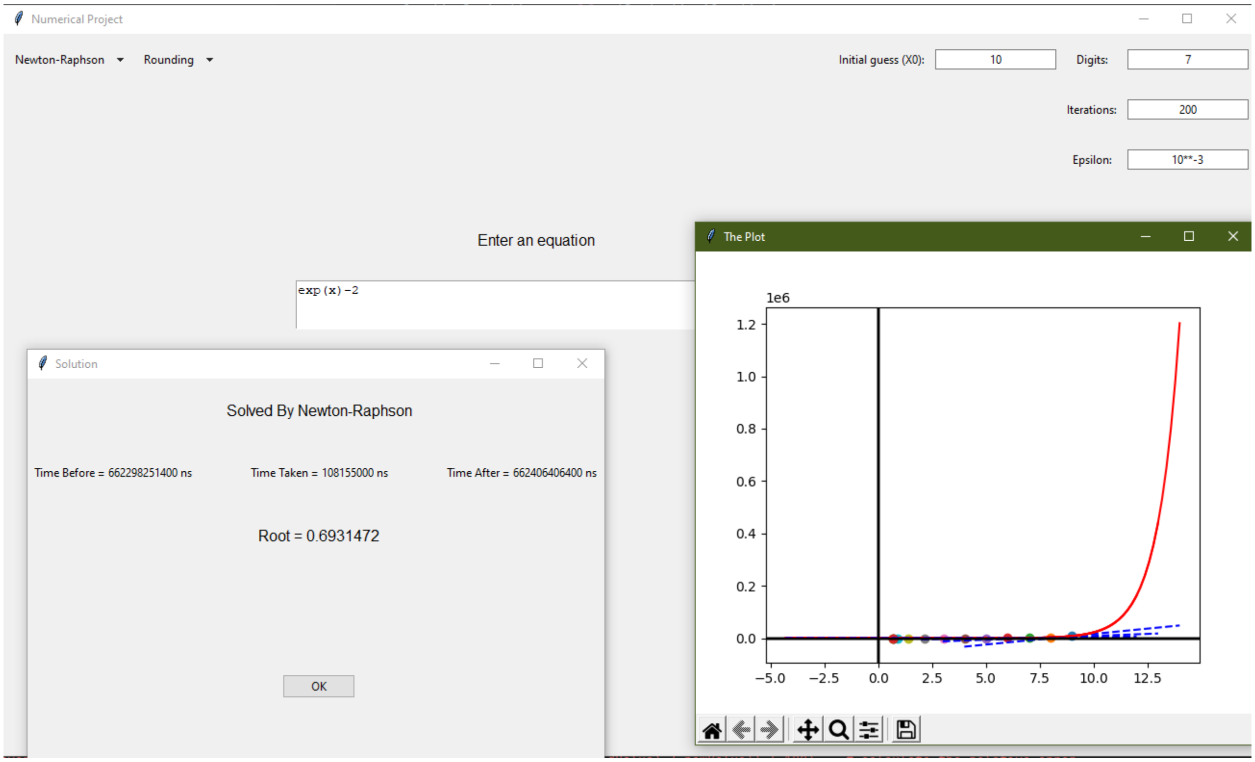


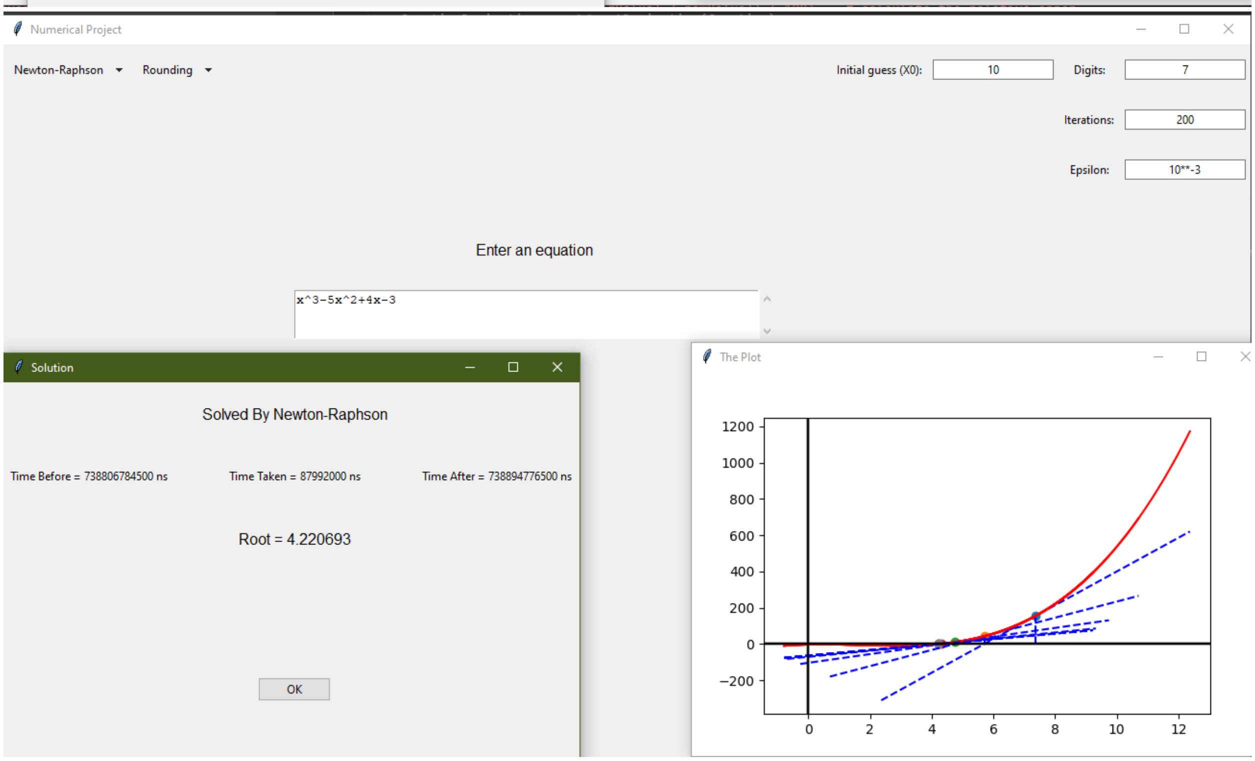

Numerical Project
Secant
Rounding
Initial guess (X0): 1
Digits: 7
Initial guess (X1): 0
Iterations: 50
Epsilon: 10**-5
Enter an equation
x^3-3*x+1
Solve
Solution
Solved By Secant
Time Before = 765585049700 ns
Time Taken = 101584600 ns
Time After = 765686634300 ns
Root = 0.3472964
OK
The Plot

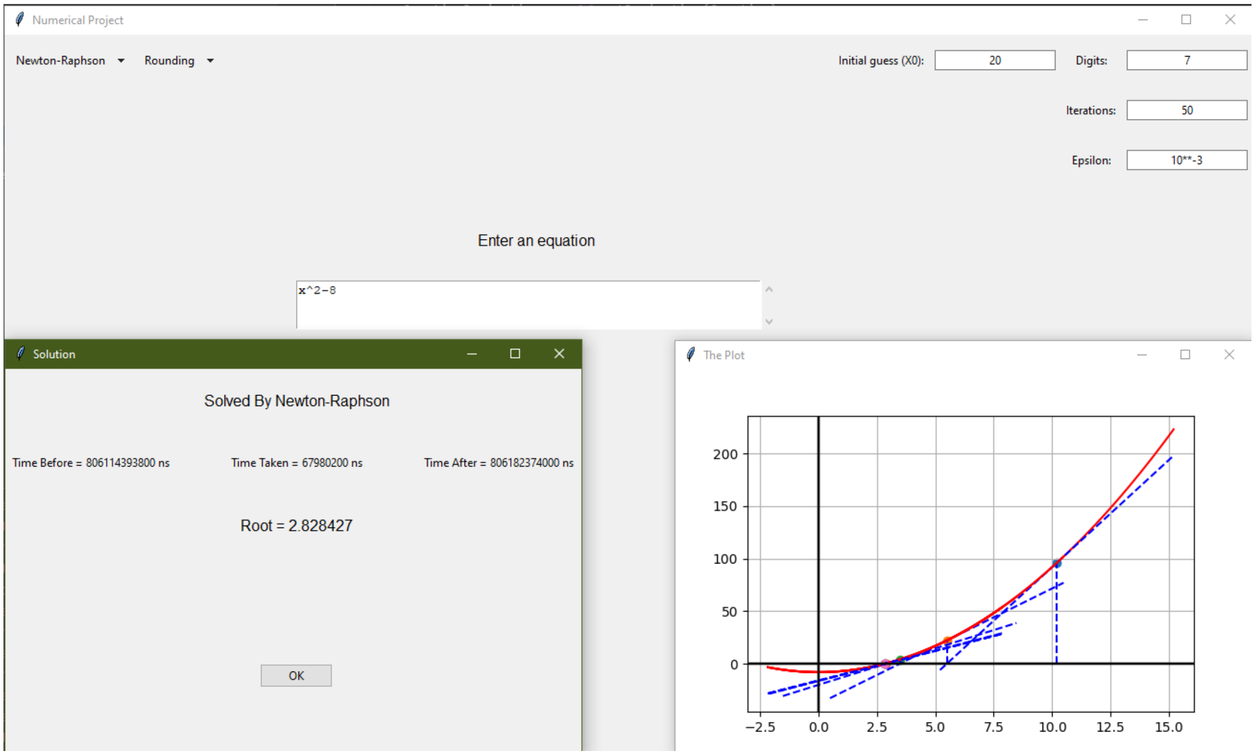
Newton Raphson Method






Numerical Project
Newton-Raphson
Rounding
Initial guess (X0): 10
Digits: 7
Iterations: 200
Epsilon: 10**-3
Enter an equation
exp(x)-2
Solution
Solved By Newton-Raphson
Time Before = 662298251400 ns
Time Taken = 108155000 ns
Time After = 662406406400 ns
Root = 0.6931472
OK
The Plot
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5


Numerical Project
Newton-Raphson
Rounding
Initial guess (X0): 10
Digits: 7
Iterations: 200
Epsilon: 10**-3
Enter an equation
x^3-5x^2+4x-3
Solution
Solved By Newton-Raphson
Time Before = 738806784500 ns
Time Taken = 87992000 ns
Time After = 738894776500 ns
Root = 4.220693
OK
The Plot
1200
1000
800
600
400
200
0
−200
0
2
4
6
8
10
12


Numerical Project
Newton-Raphson
Rounding
Initial guess (X0):
20
Digits:
7
Iterations:
50
Epsilon:
10**-3
Enter an equation
x^2-8
Solution
Solved By Newton-Raphson
Time Before = 806114393800 ns
Time Taken = 67980200 ns
Time After = 806182374000 ns
Root = 2.828427
OK
The Plot

Comparison	Bisection	False Position
Time Complexity	For each iteration Calculating x_m is O(1) calculating $f(x_r), f(x_l),$ $f(x_m)$ is O(1) calculating error is O(1) Total Time complexity for each iteration is O(1) Maximum Number of iterations $\frac{\log L_0 - \log E_a}{\log 2}$	For each iteration Calculating x_m is O(1) calculating $f(x_r), f(x_l),$ $f(x_m)$ is O(1) calculating error is O(1) Total Time complexity for each iteration is O(1)
Convergence	Always Converge	Always Converge

Comparison	Fixed Point	Secant
Time Complexity	Calculating $g(x)$ is O(1) For each iteration calculating new x is O(1) calculating error is O(1) Total Time complexity for each iteration is O(1)	For each iteration calculating new x is O(1) calculating error is O(1) Total Time complexity for each iteration is O(1)
Convergence	A sufficient condition for convergence is $\lvert g\acute{(}x)\rvert < 1$	• Not always converge • Has linear convergence
Best Error	0	0
Approximate Error		

Comparison	Newton Raphson
Time Complexity	Calculating derivative O(1) For each iteration calculating new x is O(1) calculating error is O(1) Total Time complexity for each iteration is O(1)
Convergence	• Convergence depends on function nature and accuracy of initial guess. • Converge quadratically
Best Error	0
Approximate Error	

- Data structures used

1-D Lists:

1-D lists are chosen as they allow easy storage and accessing of multiple values in specific order and allows duplicates.

Used for:

- Storing error values in fixed-point method to check if error values are increasing or not.
- Storing x values and f(x) values during plotting.

- Assumptions

General Assumptions:

- Exponential functions are input as $exp(h(x))$.
- Sine and Cosine functions are input as $sin(h(x))$ and $cos(h(x))$.

Assumptions for Bracketing:

- $f(x_{lower}) \times f(x_{upper}) \leq 0$ for bracketing methods to function correctly.
- Only the interval $[x_{lower}, x_{upper}]$ is plotted for bracketing methods.

Assumptions for Fixed Point:

- g(x) is obtained by:

  f(x) = 0

  let f(x) = h(x) + c

  h(x) + c = 0

  h(x) = -c

  $x \frac{h(x)}{x}$ = -c

  x = $\frac{-c*x}{h(x)}$ = g(x)

- in fixed point iteration, if a function is entered without a constant, a dummy constant is added to prevent breakdown of the program. However, this does not change the results or the curve.