**Phase 2: Numerical project**

**Names and ids:**

Name1: البراء مصطفى محمد

ID1: 22010636

Name2: فاروق اشرف فاروق

ID2: 22011012

Name3: احمد محمد منصور شعبان

ID3: 22010576

Name4: ايمن ابراهيم محمد قطب

ID4: 22010656

Name5: عبد الرحمن السيد صديق

ID5: 22010870

**Pseudocode**

**1. Bisection**

Algorithm Bisection(f(x), a, b, tolerance, maxIterations)

Input: f(x) - function, a, b - interval endpoints, tolerance, maxIterations

Output: root, iterations, error

If a ≥ b Then

Return Error "b must be greater than a"

fa = f(a)

fb = f(b)

If fa \* fb ≥ 0 Then

Return Error "method may fail"

previous\_c = a

iteration = 0

While iteration < maxIterations

c = (a + b)/2

fc = f(c)

If c ≠ 0 Then

relative\_error = |c - previous\_c|/|c| \* 100

Else

relative\_error = |c - previous\_c| \* 100

If relative\_error ≤ tolerance Then

Break

If fc = 0 Then

Break

Else If fa \* fc < 0 Then

b = c

Else

a = c

fa = fc

previous\_c = c

iteration = iteration + 1

Return root = c, iterations, relative\_error

**2. False position**

Algorithm FalsePosition (f(x), a, b, tolerance, maxIterations)

Input: f(x) - function, a, b - interval endpoints, tolerance, maxIterations

Output: root, iterations, error

If a ≥ b Then

Return Error "b must be greater than a"

fa = f(a)

fb = f(b)

If fa \* fb ≥ 0 Then

Return Error "method may fail"

previous\_c = a

iteration = 0

While iteration < maxIterations

c = (a\*fb - b\*fa)/ (fb - fa)

fc = f(c)

If c ≠ 0 Then

relative\_error = |c - previous\_c|/|c| \* 100

Else

relative\_error = |c - previous\_c| \* 100

If relative\_error ≤ tolerance Then

Break

If fc = 0 Then

Break

Else If fa \* fc < 0 Then

b = c

fb = fc

Else

a = c

fa = fc

previous\_c = c

iteration = iteration + 1

Return root = c, iterations, relative\_error

**3. fixed point**

Algorithm FixedPoint (g(x), x0, tolerance, maxIterations)

Input: g(x) - function, x0 - initial guess, tolerance, maxIterations

Output: root, iterations, error

iteration = 0

relative\_error = ∞

While iteration < maxIterations

x1 = g(x0)

If x1 ≠ 0 Then

relative\_error = |x1 - x0|/|x1| \* 100

Else

relative\_error = |x1 - x0| \* 100

If relative\_error ≤ tolerance Then

Break

If |x1| > 1e10 Then

Return Error "Method is diverging"

x0 = x1

iteration = iteration + 1

Return root = x1, iterations, relative\_error

**4. Secant**

Algorithm Secant(f(x), x0, x1, tolerance, maxIterations)

Input: f(x) - function, x0,x1 - initial guesses, tolerance, maxIterations

Output: root, iterations, error

f0 = f(x0)

f1 = f(x1)

iteration = 0

While iteration < maxIterations

If |f1 - f0| < 1e-15 Then

Return Error "Division by zero"

x2 = x1 - f1\*(x1 - x0)/(f1 - f0)

f2 = f(x2)

If x2 ≠ 0 Then

relative\_error = |x2 - x1|/|x2| \* 100

Else

relative\_error = |x2 - x1| \* 100

If relative\_error ≤ tolerance Or |f2| < tolerance Then

Break

x0 = x1

x1 = x2

f0 = f1

f1 = f2

iteration = iteration + 1

Return root = x2, iterations, relative\_error

**5. Modified Secant**

Algorithm ModifiedSecant(f(x), x0, delta, tolerance, maxIterations)

Input: f(x) - function, x0 - initial guess, delta - small increment, tolerance, maxIterations

Output: root, iterations, error

iteration = 0

relative\_error = ∞

While iteration < maxIterations

fx0 = f(x0)

fx0\_delta = f(x0 + delta)

If |fx0\_delta - fx0| < 1e-15 Then

Return Error "Division by zero"

x1 = x0 - delta \* fx0/(fx0\_delta - fx0)

If x1 ≠ 0 Then

relative\_error = |x1 - x0|/|x1| \* 100

Else

relative\_error = |x1 - x0| \* 100

If relative\_error ≤ tolerance Then

Break

x0 = x1

iteration = iteration + 1

Return root = x1, iterations, relative\_error

**6. Newton**

Algorithm NewtonRaphson (f(x), f'(x), xi, tolerance, maxIterations)

Input: f(x) - function, f'(x) - derivative, xi - initial guess, tolerance, maxIterations

Output: root, iterations, error

iteration = 0

f\_xi = f(xi)

df\_xi = f'(xi)

If f\_xi = 0 Then

Return xi

While iteration < maxIterations

If df\_xi = 0 Then

Return Error "Division by zero"

next\_xi = xi - f\_xi/df\_xi

f\_xi = f(next\_xi)

If next\_xi ≠ 0 Then

relative\_error = |next\_xi - xi|/|next\_xi| \* 100

Else

relative\_error = |next\_xi - xi| \* 100

If relative\_error ≤ tolerance Or f\_xi = 0 Then

Break

xi = next\_xi

df\_xi = f'(xi)

iteration = iteration + 1

Return root = next\_xi, iterations, relative\_error

**7. Modified 1 Newton**

Algorithm ModifiedNewtonRaphson1(f(x), f'(x), xi, m, tolerance, maxIterations)

Input: f(x) - function, f'(x) - derivative, xi - initial guess, m - multiplier, tolerance, maxIterations

Output: root, iterations, error

iteration = 0

f\_xi = f(xi)

df\_xi = f'(xi)

If f\_xi = 0 Then

Return xi

While iteration < maxIterations

If df\_xi = 0 Then

Return Error "Division by zero"

next\_xi = xi - m \* f\_xi/df\_xi // Only difference from original method is multiplier m

f\_xi = f(next\_xi)

If next\_xi ≠ 0 Then

relative\_error = |next\_xi - xi|/|next\_xi| \* 100

Else

relative\_error = |next\_xi - xi| \* 100

If relative\_error ≤ tolerance Or f\_xi = 0 Then

Break

xi = next\_xi

df\_xi = f'(xi)

iteration = iteration + 1

Return root = next\_xi, iterations, relative\_error

**8. Modified 2 Newton**

Algorithm ModifiedNewtonRaphson2(f(x), f'(x), f''(x), xi, tolerance, maxIterations)

Input: f(x) - function, f'(x) - first derivative, f''(x) - second derivative, xi - initial guess, tolerance, maxIterations

Output: root, iterations, error

iteration = 0

f\_xi = f(xi)

df\_xi = f'(xi)

d2f\_xi = f''(xi)

If f\_xi = 0 Then

Return xi

While iteration < maxIterations

numerator = f\_xi \* df\_xi

denominator = (df\_xi)² - d2f\_xi \* f\_xi

If denominator = 0 Then

Return Error "Division by zero"

next\_xi = xi - numerator/denominator

f\_xi = f(next\_xi)

If next\_xi ≠ 0 Then

relative\_error = |next\_xi - xi|/|next\_xi| \* 100

Else

relative\_error = |next\_xi - xi| \* 100

If relative\_error ≤ tolerance Or f\_xi = 0 Then

Break

xi = next\_xi

df\_xi = f'(xi)

d2f\_xi = f''(xi)

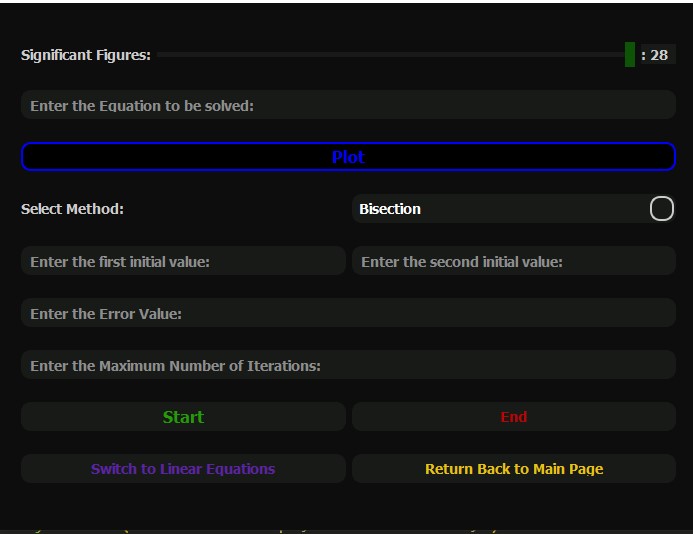
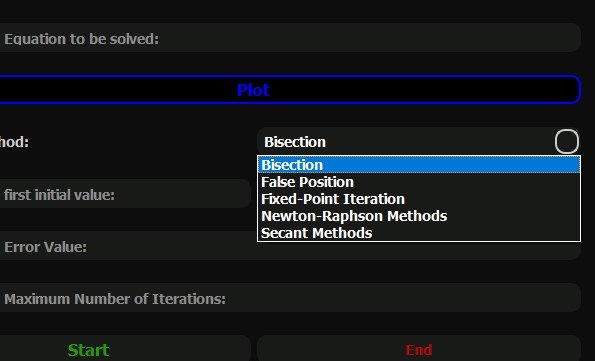
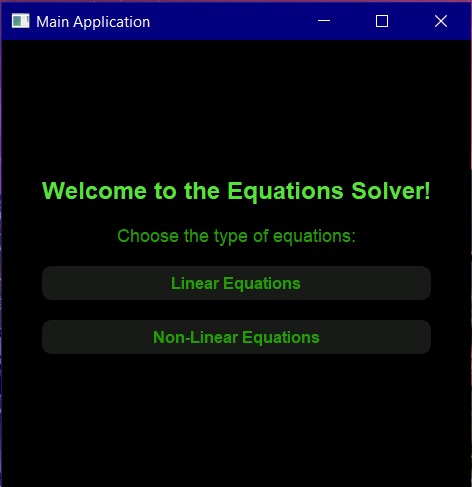
iteration = iteration + 1

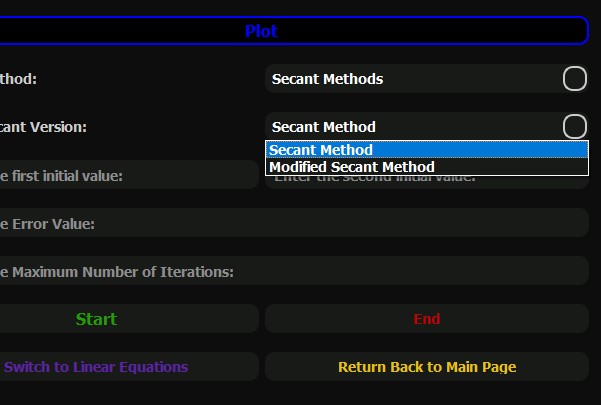
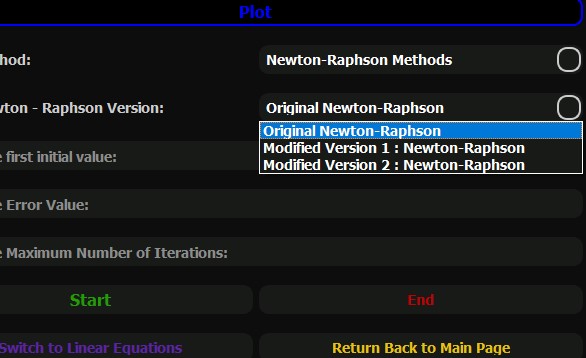
Return root = next\_xi, iterations, relative\_error

**Data structure used**

* **We formed the solution as a JSON object and the iterations (solution steps) as an array in that object.**

**GUI:**

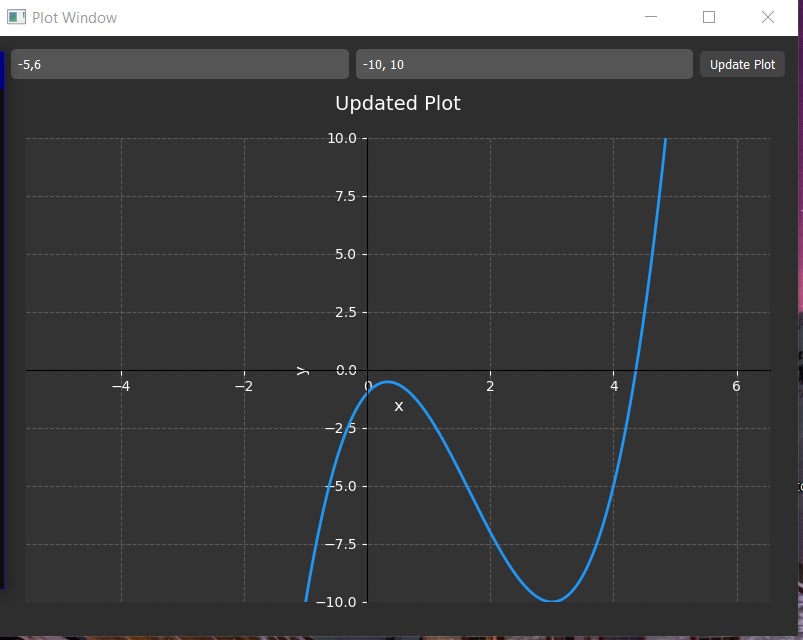
**Start page:  
taking inputs:**

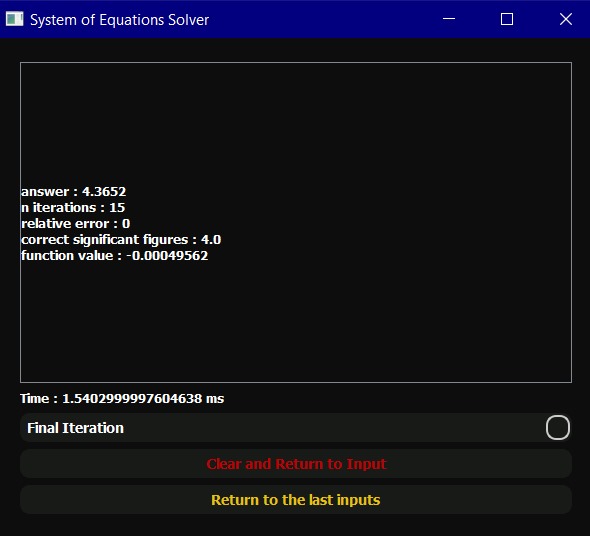
**method type:**

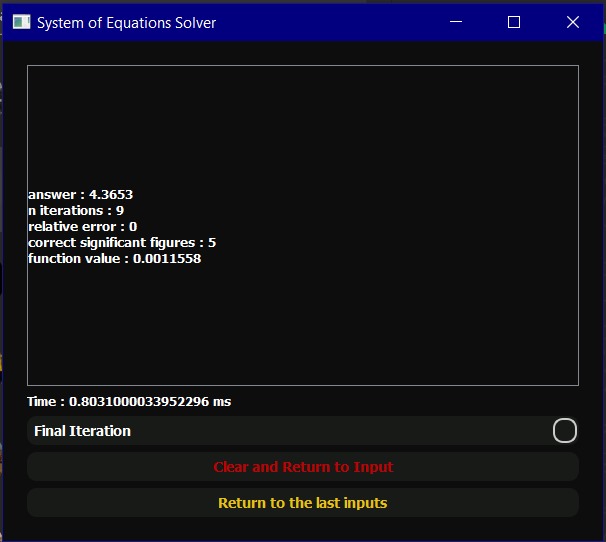
**Given test cases:**

**Test case 1:**

* results:

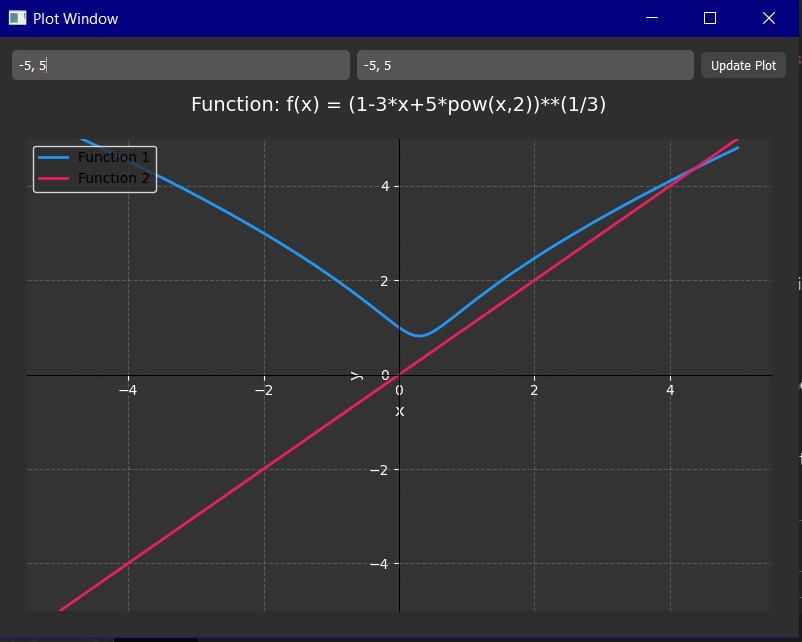
*function plot:*

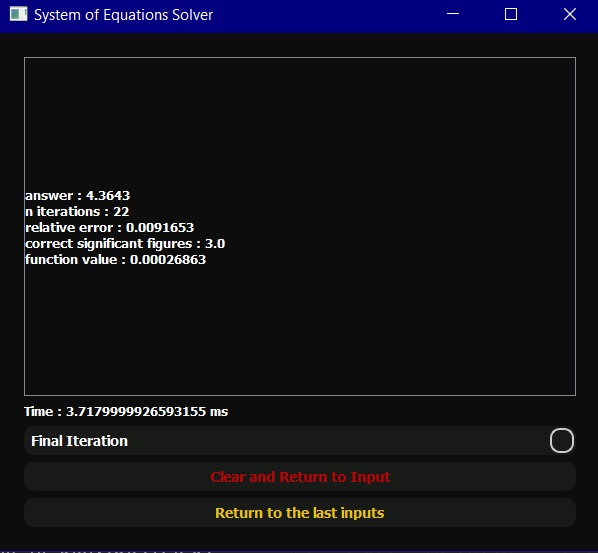
*bisection between [4,5]:*

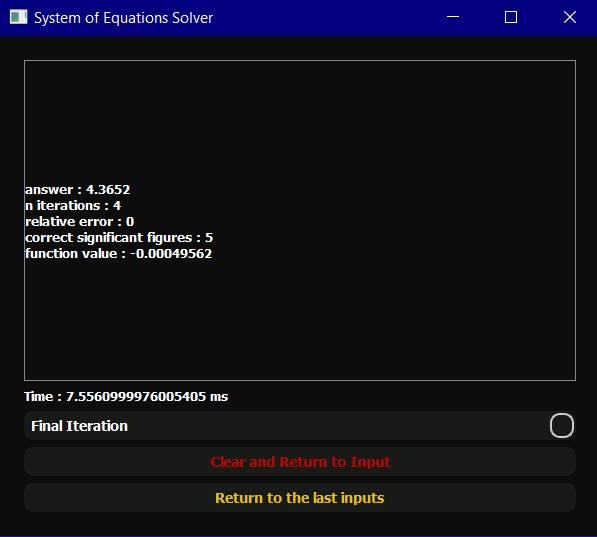
*false position:*

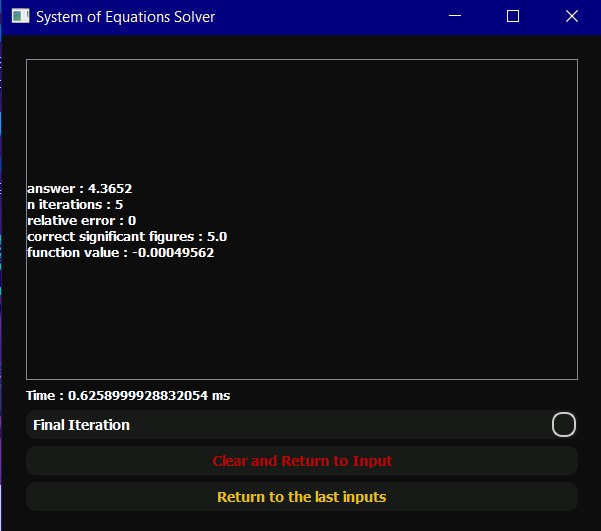
*fixed point:*

*using g(x)=(1-3\*x+5\*pow(x,2)) \*\*(1/3)*

*plot g(x):*

*result:*

*original newton:*

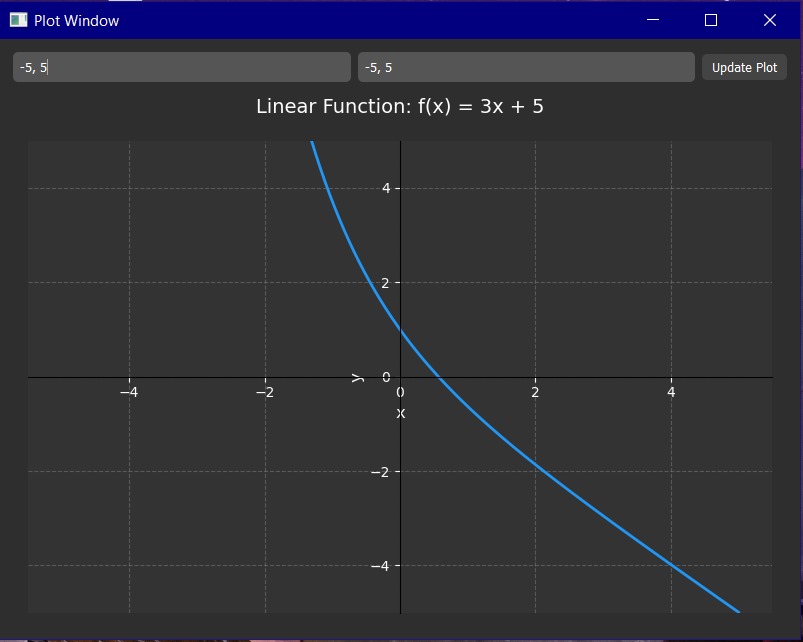
*secant:*

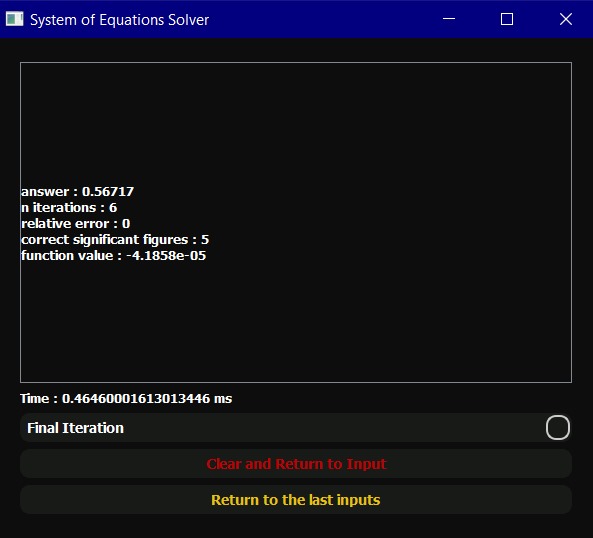
* comment:
* comparison table:

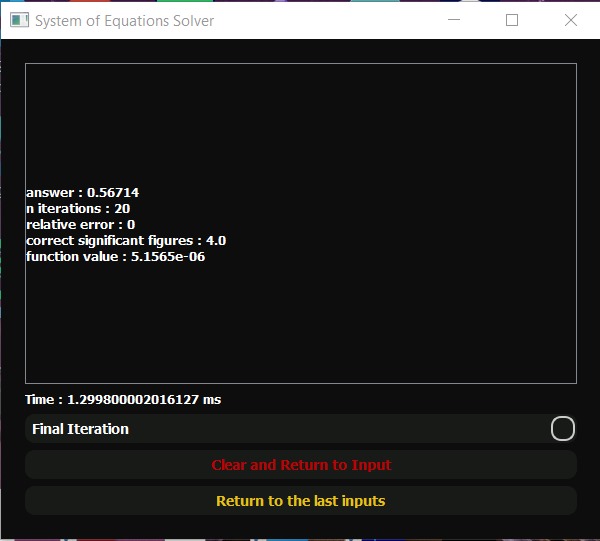
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| method | iterations reached | Run time | App root | Initial guesses | Relative error |
| Bisection |  |  |  |  |  |
| False position |  |  |  |  |  |
| Fixed point |  |  |  |  |  |
| Original newton |  |  |  |  |  |
| Secant |  |  |  |  |  |

**Test case 2:**

* results:

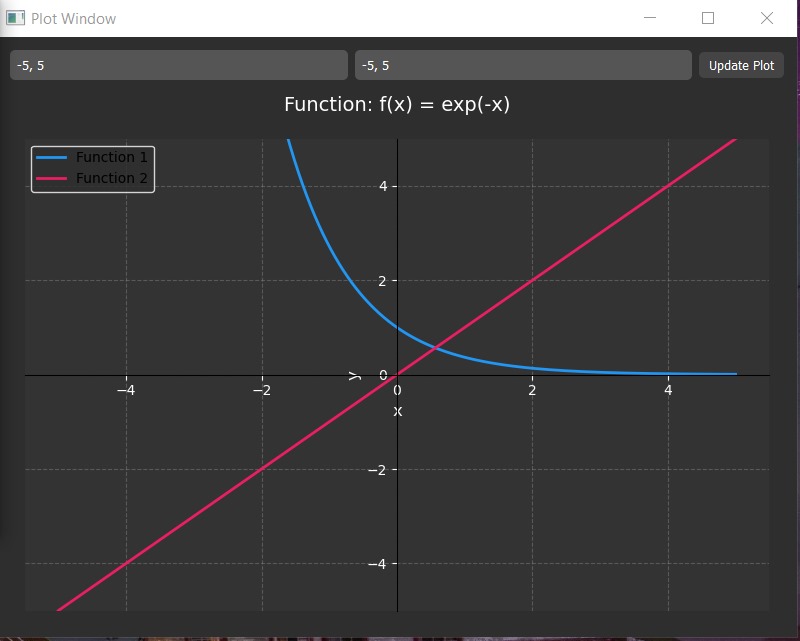
*function plot:*

*bisection between [0,2]:*

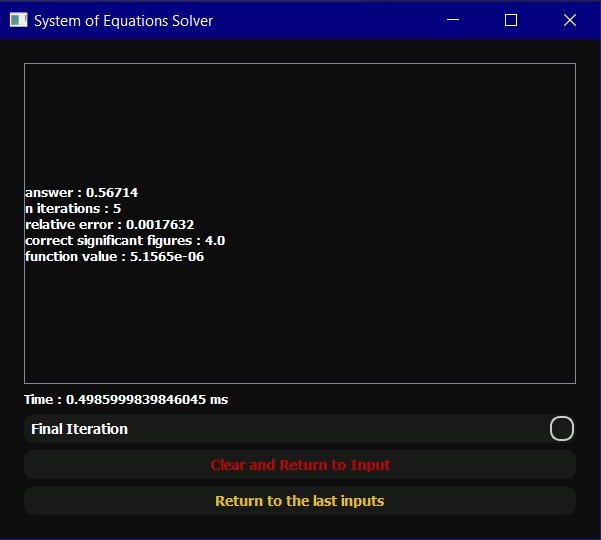
*false position:*

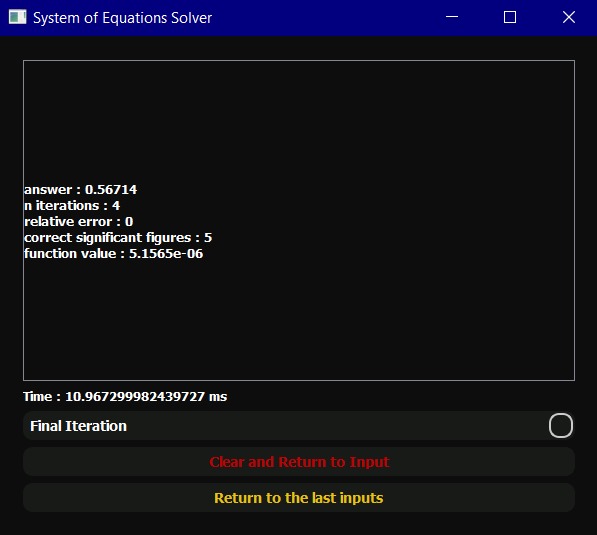
*fixed point:*

*using g(x)= e^(-x)*

*plot g(x):*

*result:*

*original newton:*

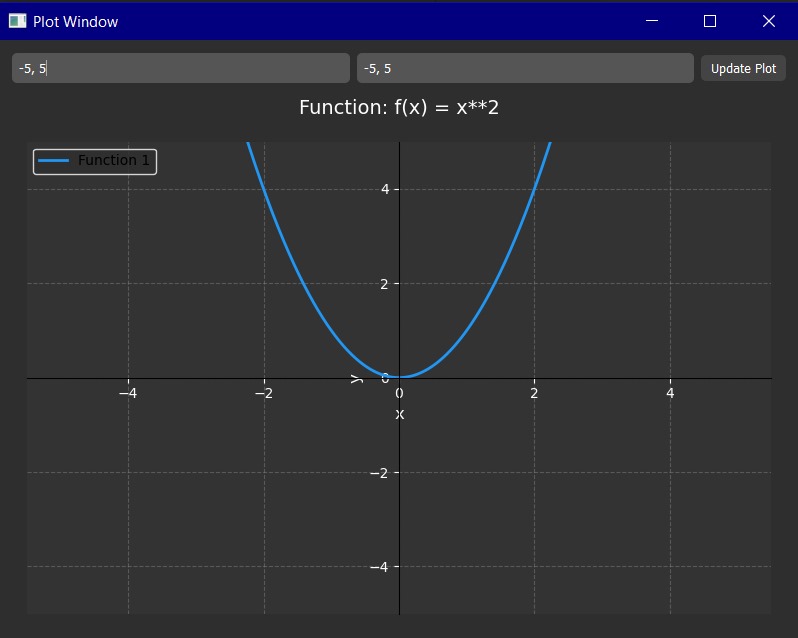
*secant:*

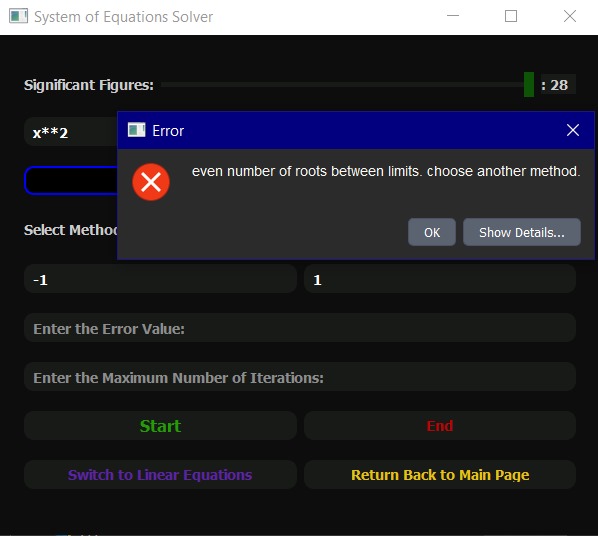
* comment:
* comparison table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| method | iterations reached | Run time | App root | Initial guesses | Relative error |
| Bisection |  |  |  |  |  |
| False position |  |  |  |  |  |
| Fixed point |  |  |  |  |  |
| Original newton |  |  |  |  |  |
| Secant |  |  |  |  |  |

**Test case 3:**

* results:

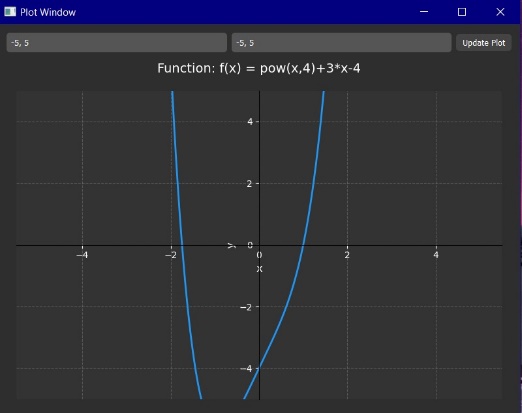
*function plot:*

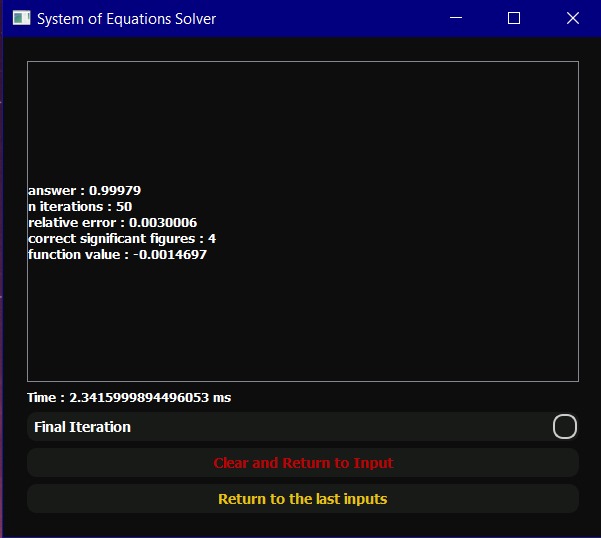
*result:*

* comment:

**Test case 4:**

* results:

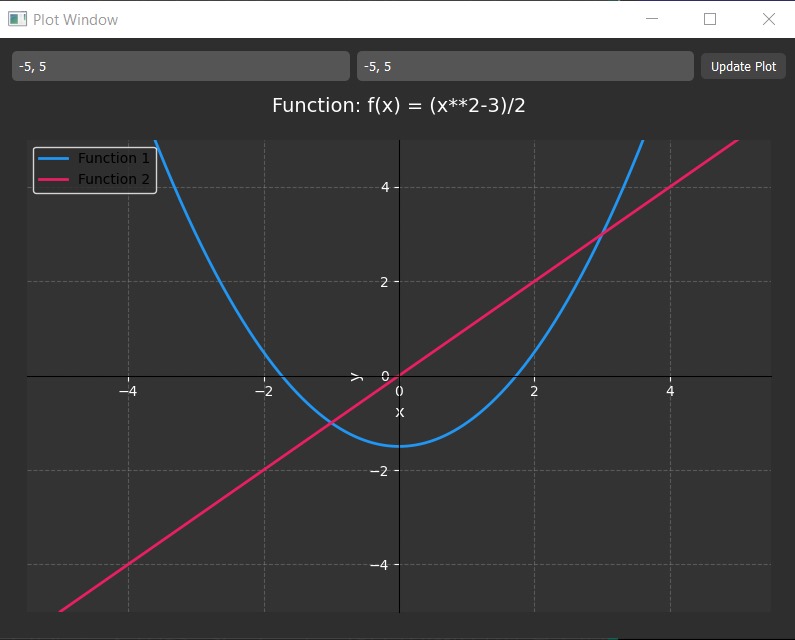
*function plot:*

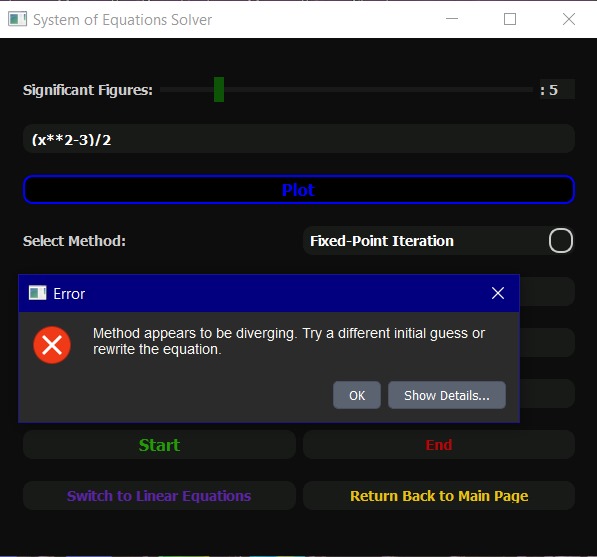
*result:*

* comment:

**Test case 5:**

* results:

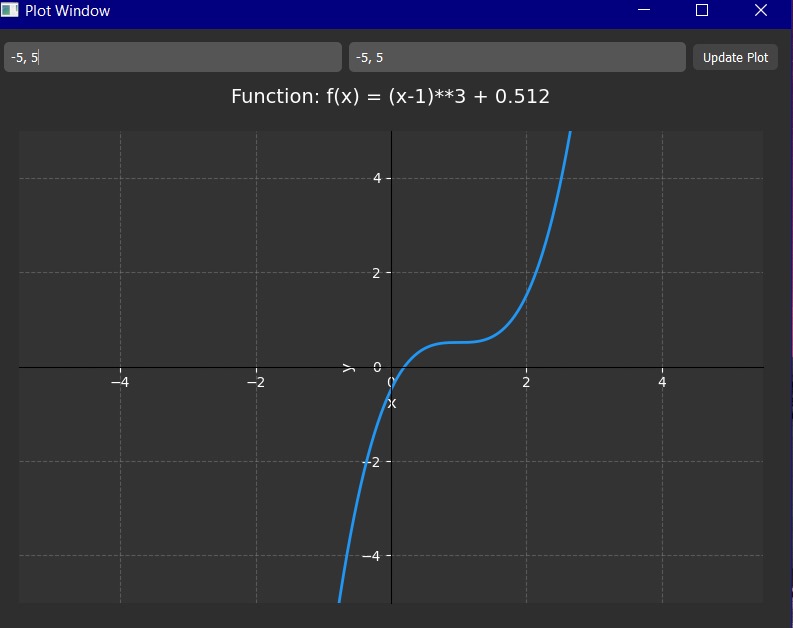
*function plot:*

*result:*

* comment:

**Test case 6:**

* results:

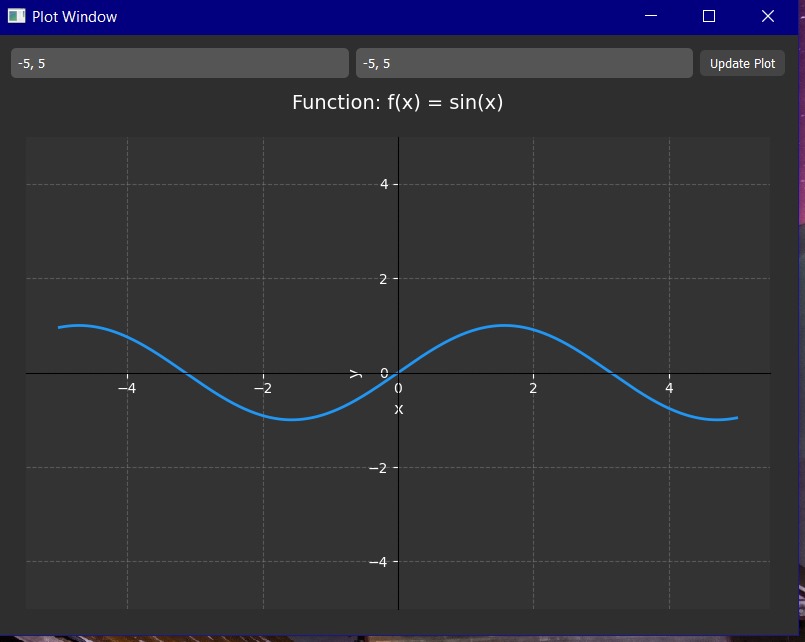
*function plot:*

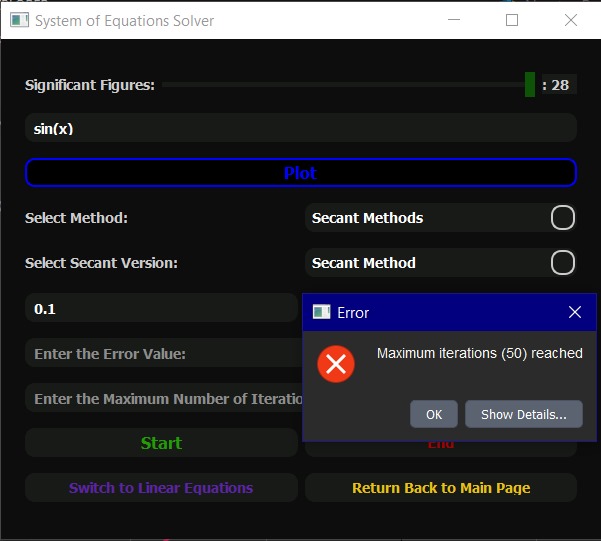
*result:*

* comment:

**Test case 7:**

* results:

*function plot:*

*result:*

* comment: