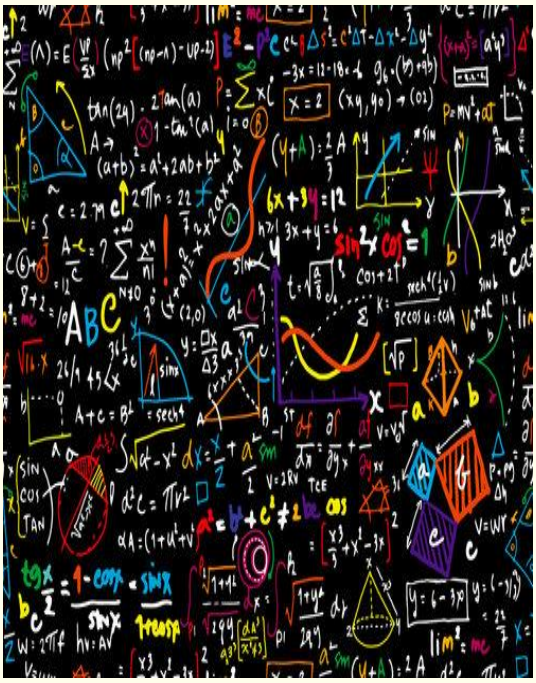


NUMERICAL COMPUTATIONS

Systems Of Non-Linear Equations

NUMERICAL COMPUTATIONS – PHASE TWO

This Project aims to provide means of solving systems of non-linear equations with different approaches to yield the most accurate roots.

Prepared By

Moustafa Esam El-Sayed Amer	21011364
Ahmed Mostafa Elmorsi Amer	21010189
Ahmed Ayman Ahmed Abdallah	21010048
Ebrahim Alaa Eldin Ebrahim	21010017
Ali Hassan Ali Mohamed	21010837
Ahmed Youssef Sobhy Elgoerany	21010217

Design

Object Oriented concepts

- Each Method is a Solver Class itself that gets called through the GUI.
- In addition to a parser that deals with non-linear equations as Sympy expressions and gets it parsed to Solvers.
- Step-by-step String shown at each Solver.

Implementation Notes

- You can find GitHub repository link to view all source codes here GitHub Repo RootFinder
- You will find that the source codes are divided into solvers classes, each class for a method, with a parser class for parsing nonlinear equations using sympy.

GUI

- The GUI is designed using DearPyGUI, which is a fast and powerful Graphical User Interface Toolkit for Python.

Used Data Structures

- Sympy Library is used in the parser to parse the equation to all the methods and if required it passes the first and second derivatives of the functions too, with the symbol x as the valid variable used.
- Used round_to_sf function for fixed precision arithmetic, function is defined in the solver class.
- Python List to store the **Step-by-step** solution steps in each method.
- **NB**
    - This is applied to all methods, **i.e** no method will have other data structures except if explicitly defined.

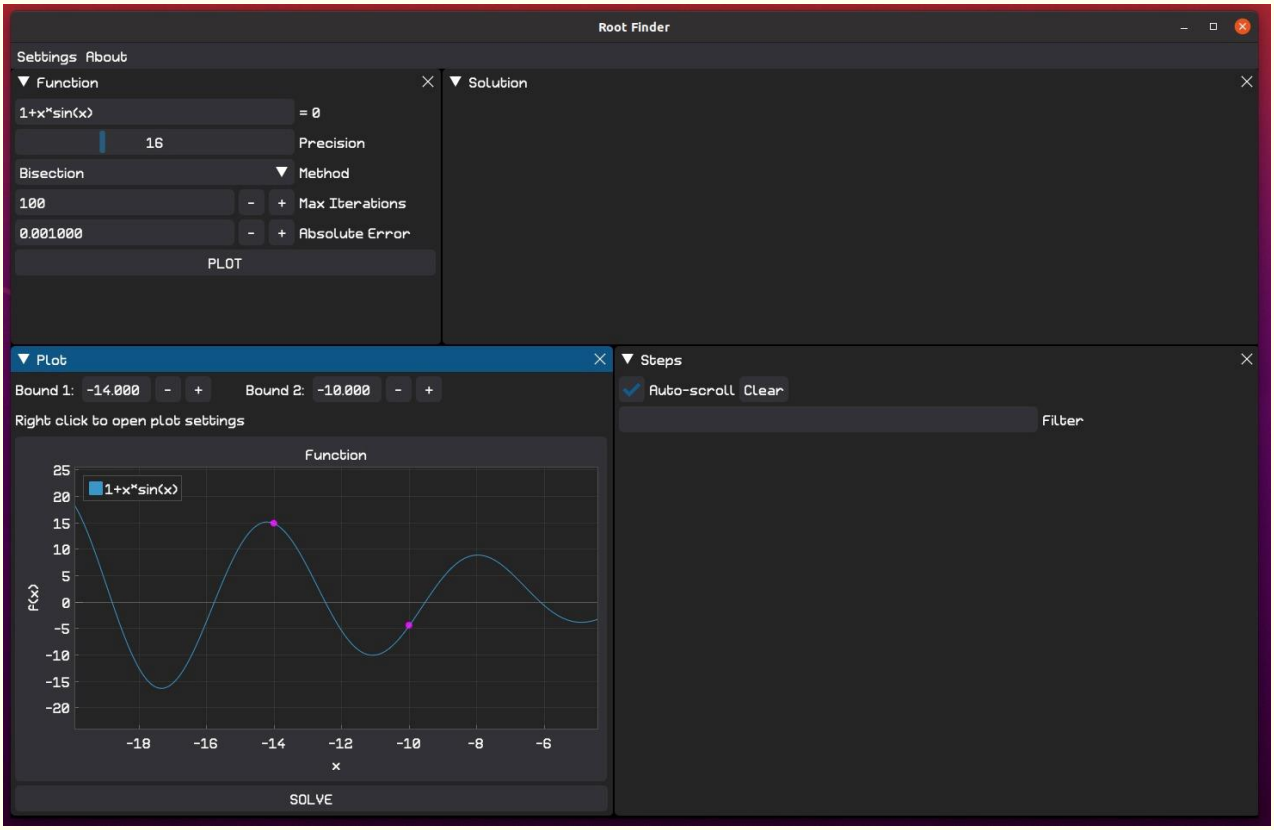
Assumptions

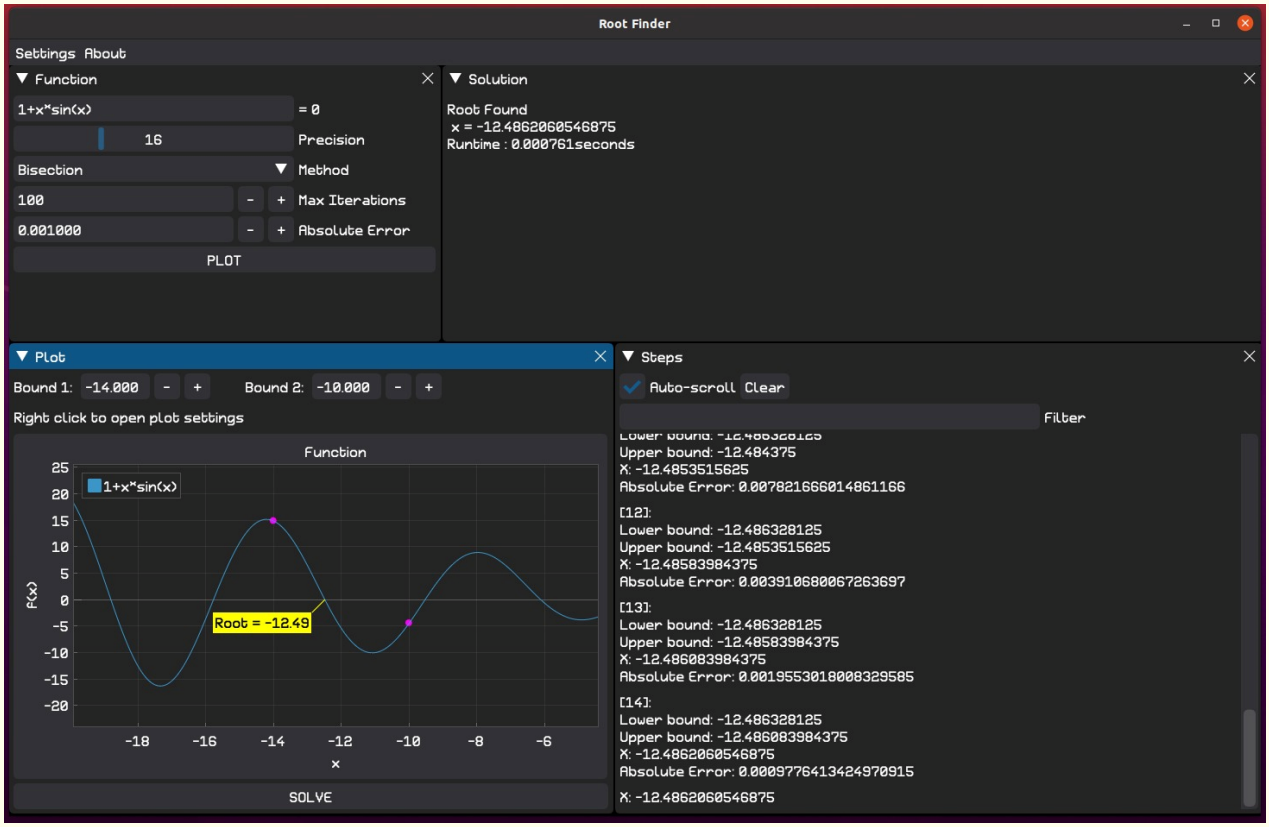
- The user will enter the equations in the form
    - $5x^2 - 5x = 0$ written as 5*x**2-5*x
    - x^5 written as x**5
    - $Sinx$ written as sin(x)
    - $\sqrt{x}$ written as sqrt(x)
    - e^x written as exp(x)

- Default precision value = 16, with the flexibility of being edited in a simple graphical slider bar.
- Termination Factors
    - The relative error converges to the required value.
    - $f(x_r) = 0,\ x_r\ is\ the\ value\ of\ the\ root\ at\ the\ new\ iteration$
    - The method iterates 100 times, that is assumed to be the endpoint to determine divergence cases, the user may have the facility to decrease the number of iterations, for example if I want to know the output of the first 6 iterations only.

- Threshold for divergence is for the root to reach 1×10^{10} or 1×10^{-10}

User Interface


Root Finder
Settings About
Function
1+x*sin(x)
= 0
16
Precision
Bisection
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
Plot
Bound 1: -14.000
Bound 2: -10.000
Right click to open plot settings
Function
1+x*sin(x)
f(x)
x
SOLVE
Steps
Auto-scroll
Clear
Filter


Root Finder
Settings About
Function
1+x*sin(x)
= 0
16
Precision
Bisection
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
Root Found
x = -12.4862060546875
Runtime : 0.000761seconds
Plot
Bound 1: -14.000
Bound 2: -10.000
Right click to open plot settings
Function
1+x*sin(x)
Root = -12.49
f(x)
x
SOLVE
Steps
Auto-scroll
Clear
Filter
Upper bound: -12.484375
X: -12.4853515625
Absolute Error: 0.007821666014861166
[12]:
Lower bound: -12.486328125
Upper bound: -12.4853515625
X: -12.48583984375
Absolute Error: 0.0039106800672636 97
[13]:
Lower bound: -12.486328125
Upper bound: -12.48583984375
X: -12.486083984375
Absolute Error: 0.0019553018008329585
[14]:
Lower bound: -12.486328125
Upper bound: -12.486083984375
X: -12.4862060546875
Absolute Error: 0.0009776413424970915
X: -12.4862060546875

Solver Methods

Bracketing Methods

Bisection Method

Description:

The User enters a lower bound x_l and an upper bound x_u, in which the function has an existing root.

It should be verified that $f(x_l) \times f(x_u) < 0$

For each iteration the root is estimated using the formula

$$x_r = \frac{x_l + x_u}{2}$$

After checking for $f(x_r) \neq 0$ (as a termination condition) the upper or lower bounds are replaced with the estimated root according to the following conditions

$$f(x_l) \times f(x_r) < 0 \rightarrow x_u = x_r$$

$$f(x_l) \times f(x_r) > 0 \rightarrow x_l = x_r$$

Pseudocode:

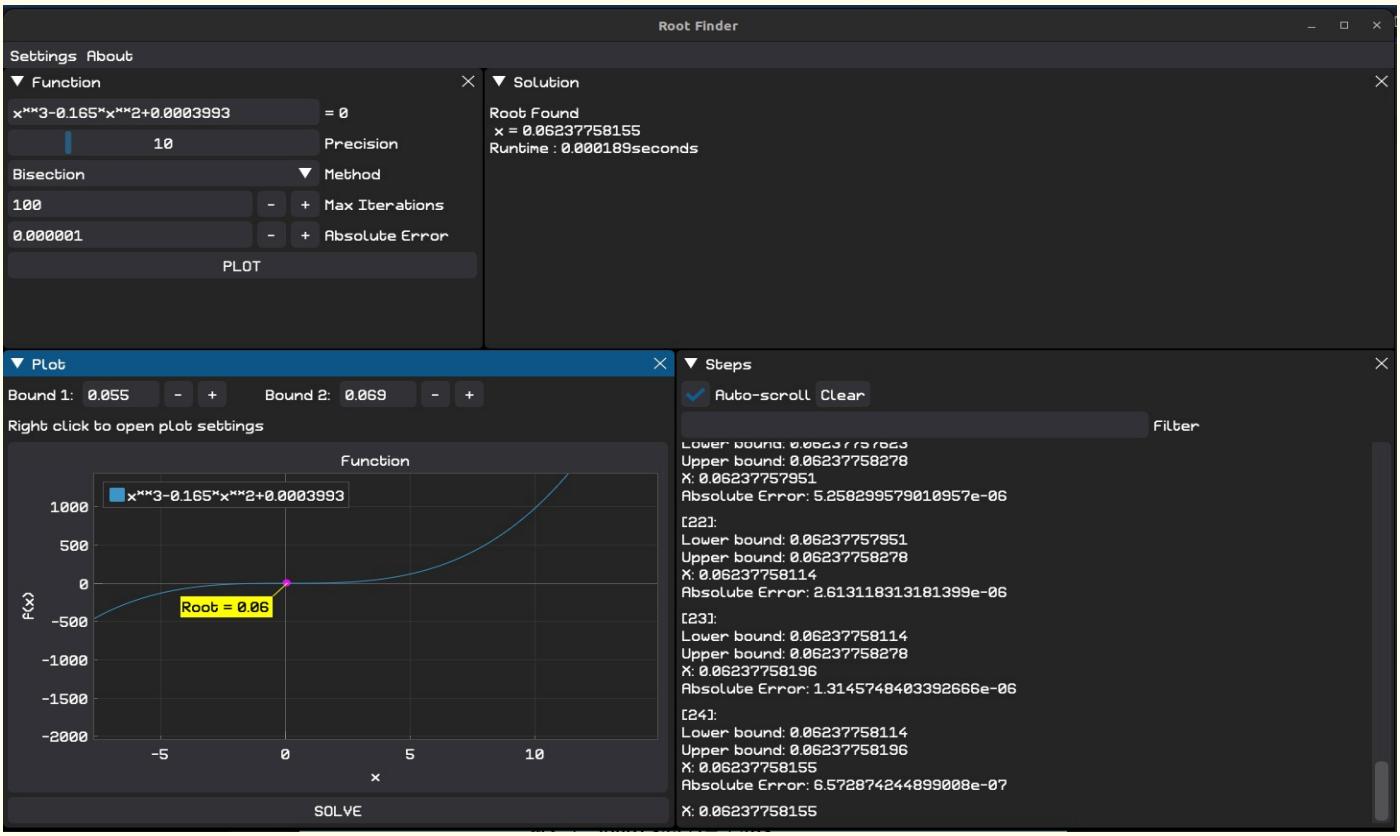
```
function bisection_method(L, U, error):
    step = 0
    L = round_to_sf(L, self.sf)
    U = round_to_sf(U, self.sf)
    error = round_to_sf(error, self.sf)

    if equation(L) * equation(U) >= 0:
        raise ValueError("Bisection method fails.")
    else:
        x_old = round_to_sf((L + U) / 2, self.sf)
        append_step_info(step, L, U, x_old)

        if equation(L) * equation(x_old) < 0:
            U = x_old
        else:
            L = x_old

        x_new = round_to_sf((L + U) / 2, self.sf)
        step += 1
        append_step_info(step, L, U, x_new, absolute_error(x_new, x_old))

        while absolute_error(x_new, x_old) > error and step < max_iterations and equation(x_new) != 0:
            x_old = x_new

            if equation(L) * equation(x_old) < 0:
                U = x_old
            else:
                L = x_old
```

```
    x_new = round_to_sf((L + U) / 2, self.sf)
    step += 1
    append_step_info(step, L, U, x_new, absolute_error(x_new, x_old))

append_final_step_info(x_new)
return x_new
```

Sample Runs:

$f(x) = x^3 - 0.165x^2 + 0.0003993$

Bounds are 0.055, 0.069 with a root at 0.06237

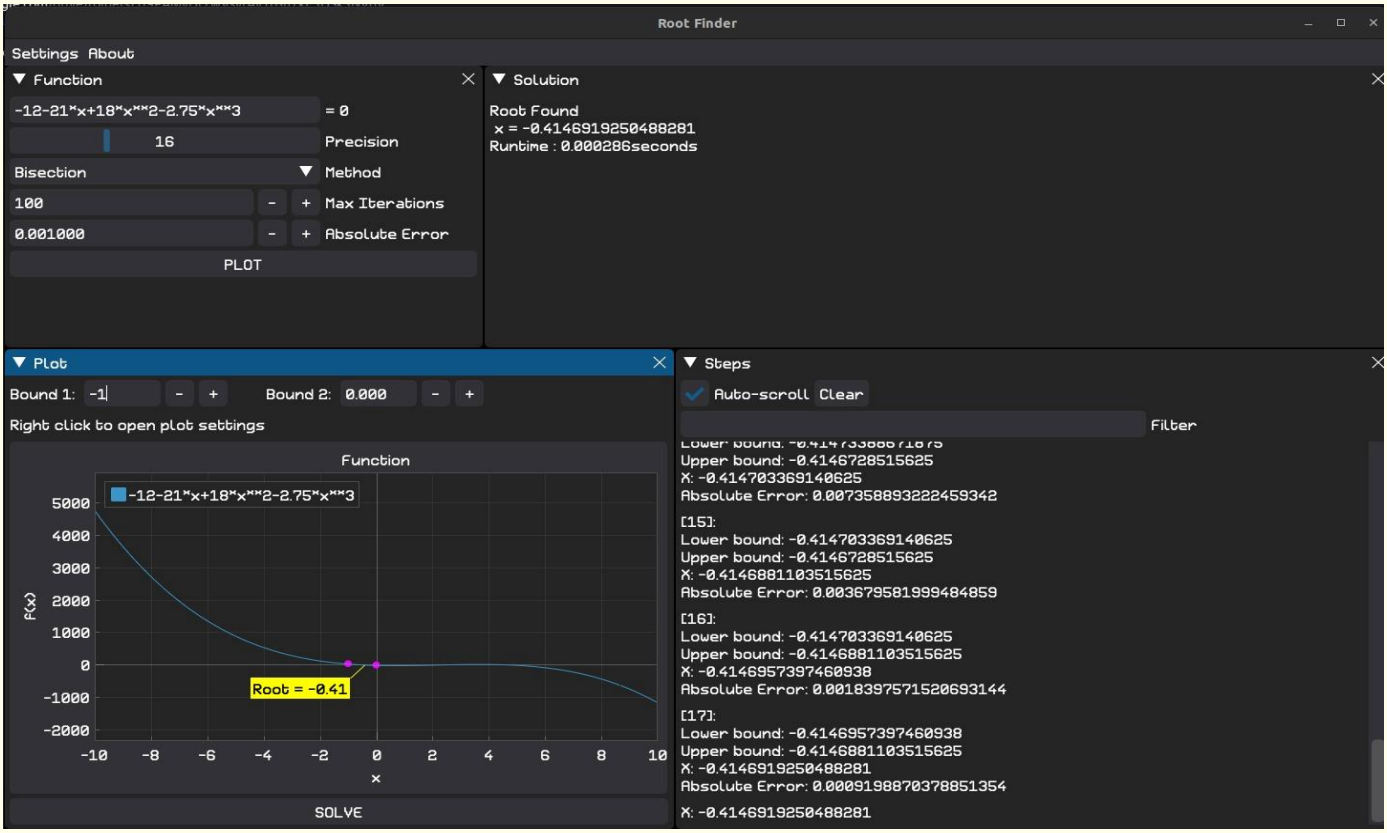


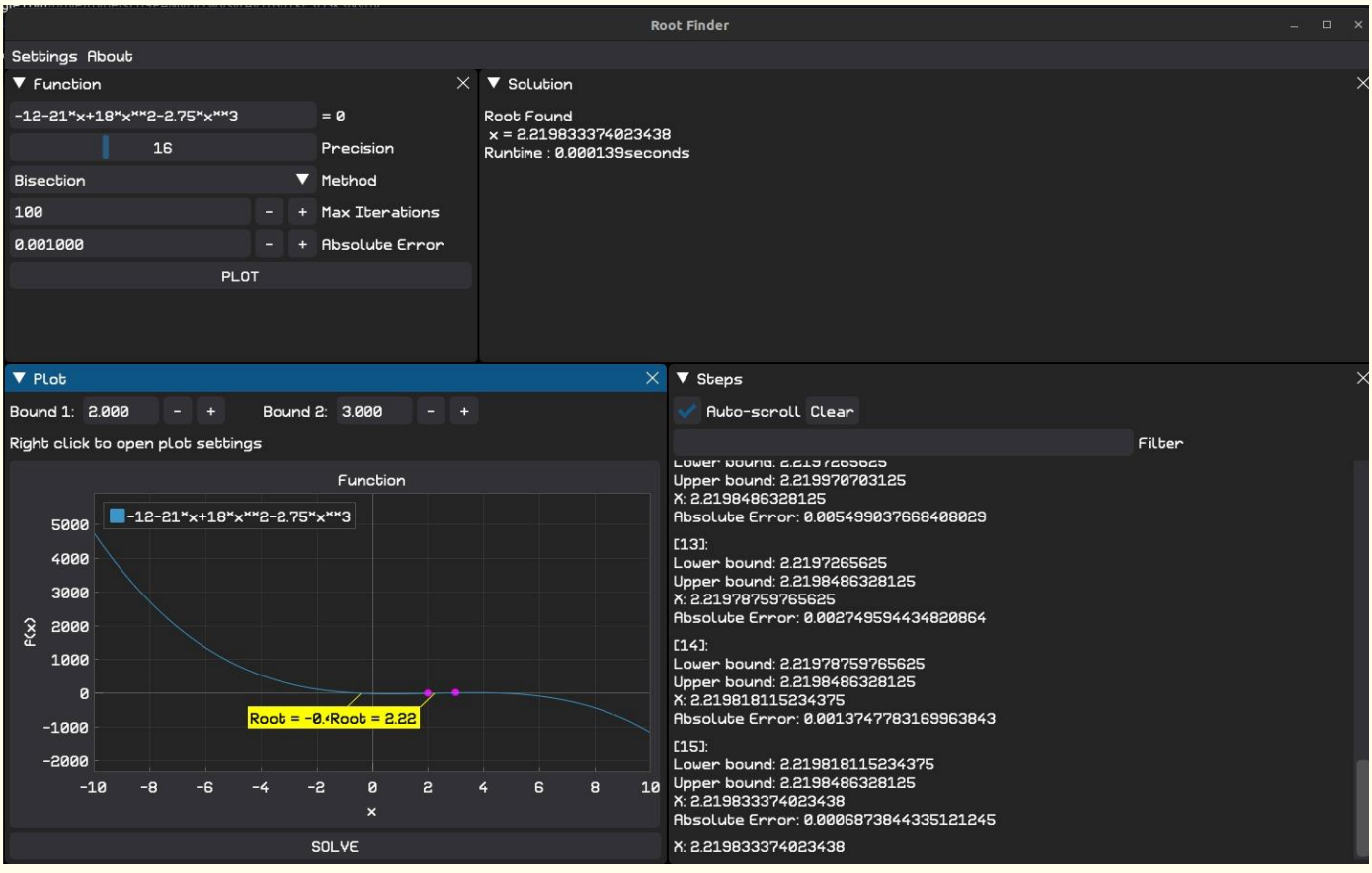
$f(x) = -12 - 21x + 18x^2 - 2.75x^3$

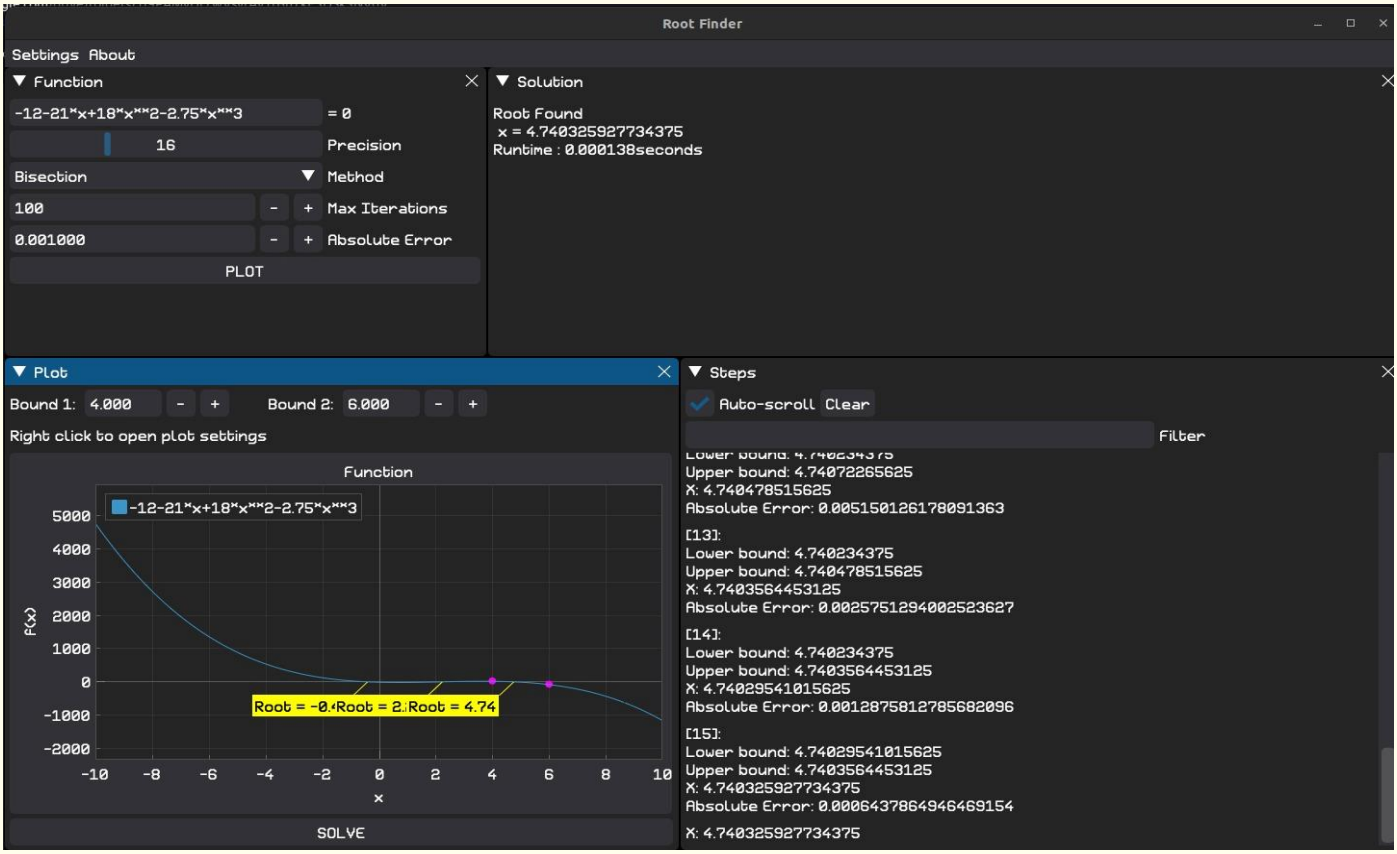
Case 1: bounds are -1, 0 with one root at -0.4147

Case 2: bounds are 2, 3 with one root at 2.2198

Case 3: bounds are 4, 6 with one root at 4.7403


Root Finder
Settings About
Function
-12-21*x+18*x**2-2.75*x**3
= 0
16
Precision
Bisection
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
Root Found
x = -0.4146919250488281
Runtime : 0.000286seconds
Plot
Bound 1: -1
Bound 2: 0.000
Right click to open plot settings
Function
-12-21*x+18*x**2-2.75*x**3
f(x)
x
Root = -0.41
SOLVE
Steps
Auto-scroll
Clear
Filter
Upper bound: -0.4146728515625
X: -0.414703369140625
Absolute Error: 0.007358893222459342
[15]:
Lower bound: -0.414703369140625
Upper bound: -0.4146728515625
X: -0.4146881103515625
Absolute Error: 0.003679581999484859
[16]:
Lower bound: -0.414703369140625
Upper bound: -0.4146881103515625
X: -0.4146957397460938
Absolute Error: 0.0018397571520693144
[17]:
Lower bound: -0.4146957397460938
Upper bound: -0.4146881103515625
X: -0.4146919250488281
Absolute Error: 0.0009198870378851354
X: -0.4146919250488281


Root Finder
Settings About
Function
-12-21*x+18*x**2-2.75*x**3
= 0
16
Precision
Bisection
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
Root Found
x = 2.219833374023438
Runtime : 0.000139seconds
Plot
Bound 1: 2.000
Bound 2: 3.000
Right click to open plot settings
Function
-12-21*x+18*x**2-2.75*x**3
f(x)
x
Root = -0.4Root = 2.22
SOLVE
Steps
Auto-scroll
Clear
Filter
Upper bound: 2.219970703125
X: 2.21984863281250
Absolute Error: 0.005499037668408029
[13]:
Lower bound: 2.2197265625
Upper bound: 2.2198486328125
X: 2.219787597656250
Absolute Error: 0.002749594434820864
[14]:
Lower bound: 2.219787597656250
Upper bound: 2.2198486328125
X: 2.219818115234375
Absolute Error: 0.0013747783169963843
[15]:
Lower bound: 2.219818115234375
Upper bound: 2.2198486328125
X: 2.219833374023438
Absolute Error: 0.0006873844335121245
X: 2.219833374023438



False Position Method

Description:

The User enters a lower bound x_l and an upper bound x_u, in which the function has an existing root.

It should be verified that $f(x_l) \times f(x_u) < 0$

For each iteration the root is estimated using the formula

$$x_r = \frac{x_l \times f(x_u) - x_u \times f(x_l)}{f(x_u) - f(x_l)}$$

After checking for $f(x_r) \neq 0$ (as a termination condition) the upper or lower bounds are replaced with the estimated root according to the following conditions

$$f(x_l) \times f(x_r) < 0 \rightarrow x_u = x_r$$

$$f(x_l) \times f(x_r) > 0 \rightarrow x_l = x_r$$

Pseudocode:

```
function false_position_method(L, U, error):
    step = 0
```

```
    L = round_to_sf(L, self.sf)
    U = round_to_sf(U, self.sf)
    error = round_to_sf(error, self.sf)

    if equation(L) * equation(U) >= 0:
        raise ValueError("False position method fails.")
    else:
        x_old = round_to_sf((L * equation(U) - U * equation(L)) / (equation(U) - equation(L)), self.sf)
        append_step_info(step, L, U, x_old)

        if equation(L) * equation(x_old) < 0:
            U = x_old
        else:
            L = x_old

        x_new = round_to_sf((L * equation(U) - U * equation(L)) / (equation(U) - equation(L)), self.sf)
        step += 1
        append_step_info(step, L, U, x_new, absolute_error(x_new, x_old))

        while absolute_error(x_new, x_old) > error and step < max_iterations and equation(x_new) != 0:
            x_old = x_new

            if equation(L) * equation(x_old) < 0:
                U = x_old
            else:
                L = x_old

            x_new = round_to_sf((L * equation(U) - U * equation(L)) / (equation(U) - equation(L)), self.sf)
            step += 1
            append_step_info(step, L, U, x_new, absolute_error(x_new, x_old))

        append_final_step_info(x_new)
        return x_new
```

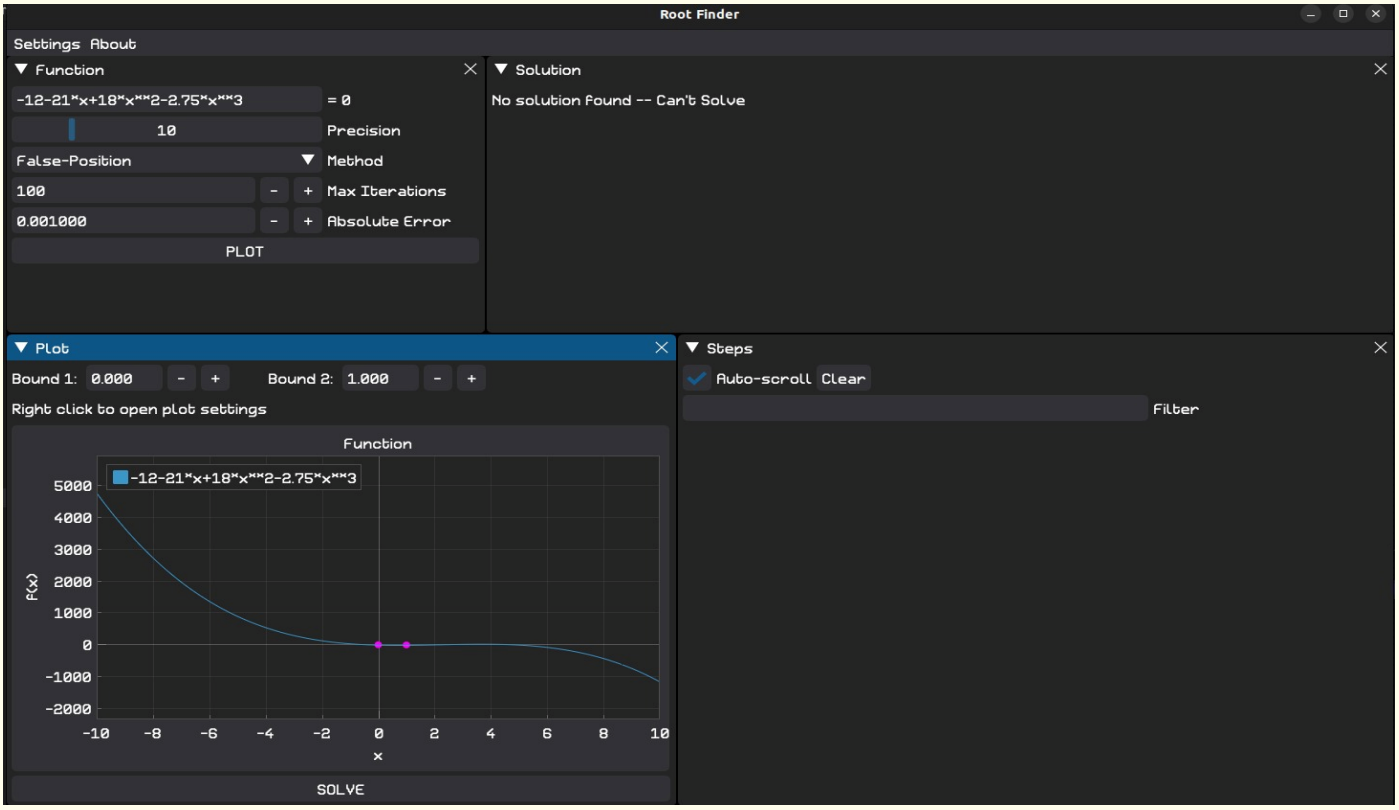
Sample Runs:

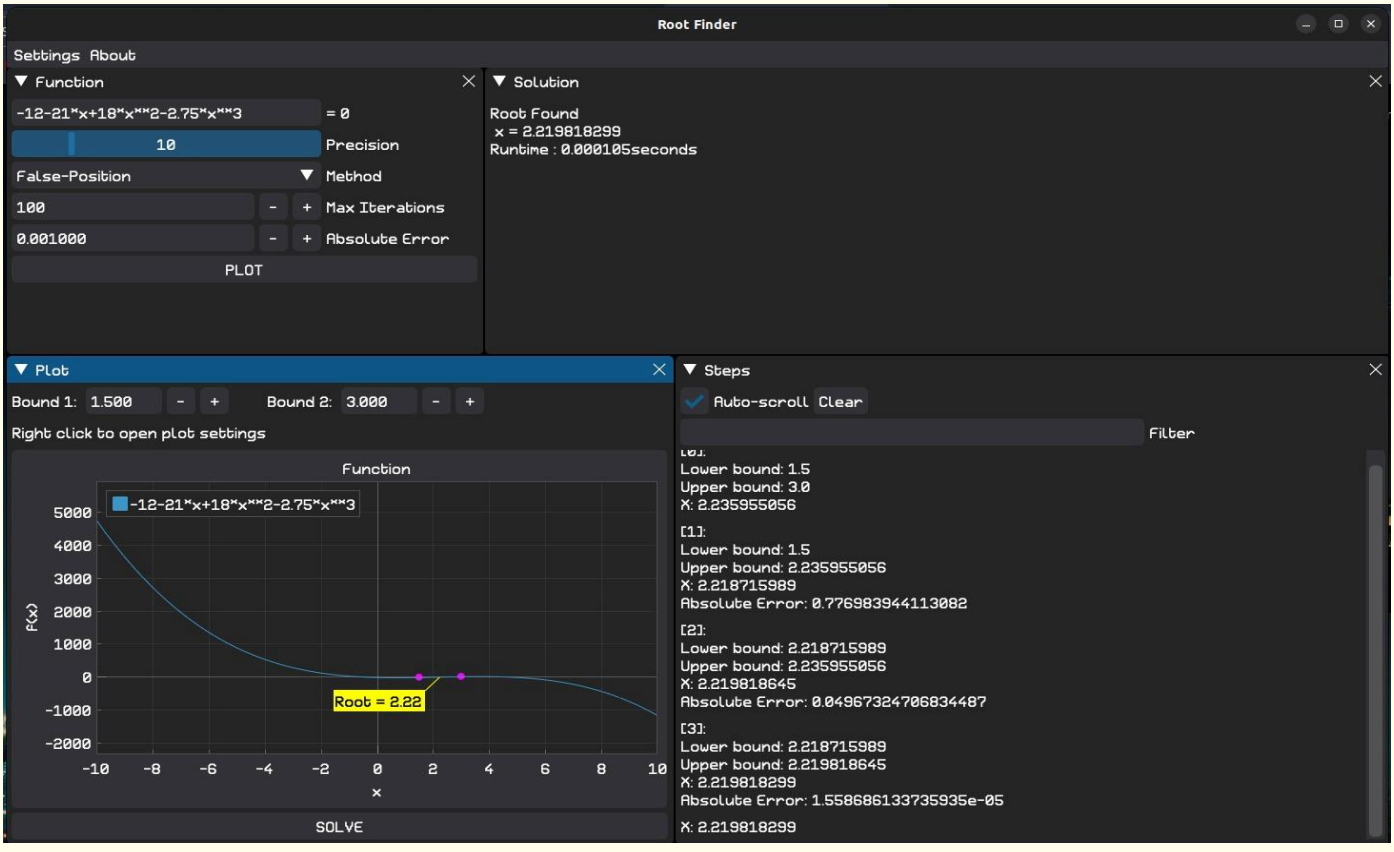
$f(x) = -12 - 21x + 18x^2 - 2.75x^3$

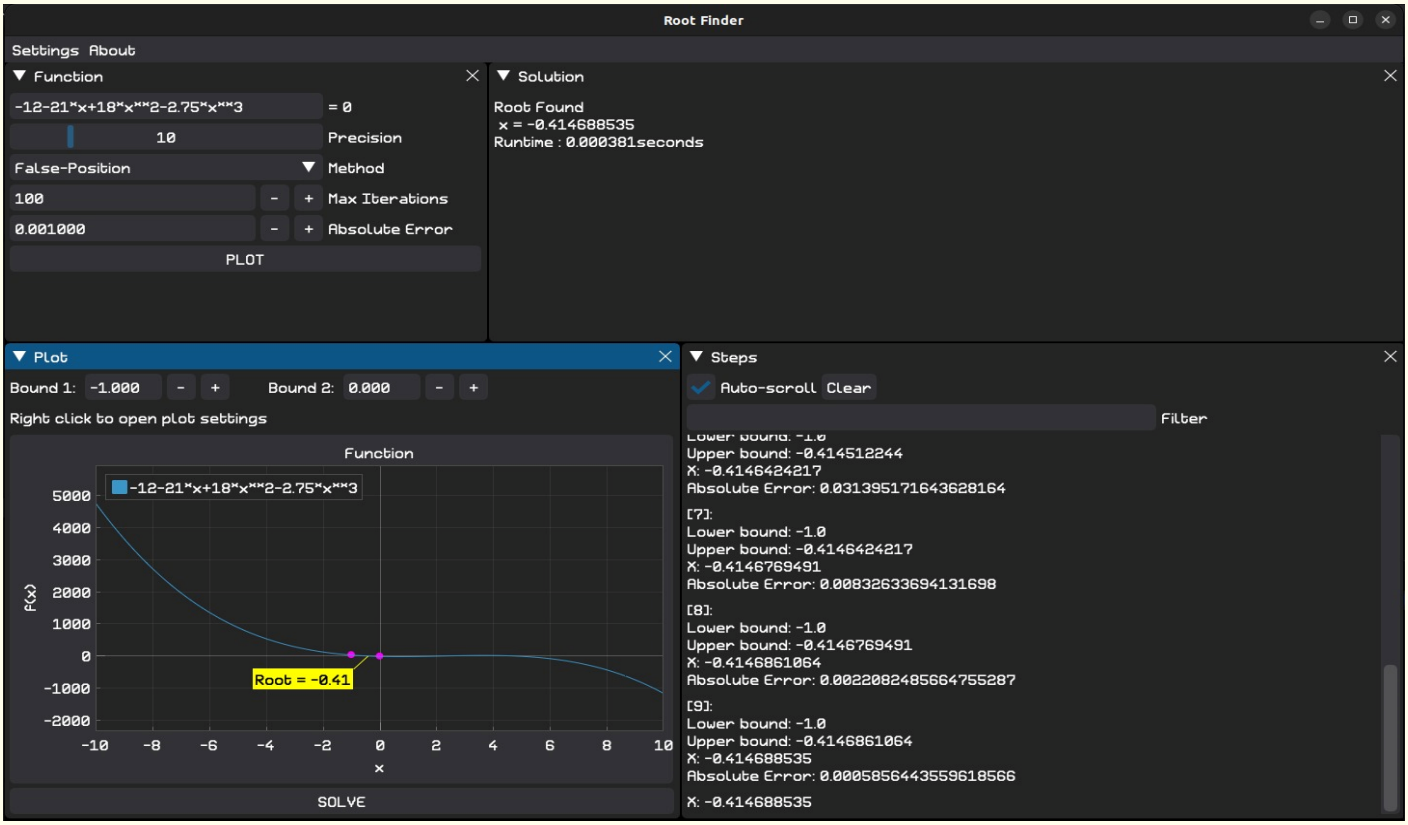
Case 1: bounds are 0, 1 where no root exists in this region

Case 2: bounds are 1.5, 3 with one root at 2.219

Case 3: bounds are -1, 0 with one root at -0.4147


Root Finder
Settings About
Function
-12-21*x+18*x**2-2.75*x**3
= 0
10
Precision
False-Position
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
No solution found -- Can't Solve
Plot
Bound 1: 0.000
Bound 2: 1.000
Right click to open plot settings
Function
f(x)
x
SOLVE
Steps
Auto-scroll
Clear
Filter


Root Finder
Settings About
Function
-12-21*x+18*x**2-2.75*x**3
= 0
10
Precision
False-Position
Method
100
Max Iterations
0.001000
Absolute Error
PLOT
Solution
Root Found
x = 2.219818299
Runtime : 0.000105seconds
Plot
Bound 1: 1.500
Bound 2: 3.000
Right click to open plot settings
Function
f(x)
x
Root = 2.22
SOLVE
Steps
Auto-scroll
Clear
Filter
Lower bound: 1.5
Upper bound: 3.0
X: 2.235955056
[1]:
Lower bound: 1.5
Upper bound: 2.235955056
X: 2.218715989
Absolute Error: 0.776983944113082
[2]:
Lower bound: 2.218715989
Upper bound: 2.235955056
X: 2.219818645
Absolute Error: 0.04967324706834487
[3]:
Lower bound: 2.218715989
Upper bound: 2.219818645
X: 2.219818299
Absolute Error: 1.558686133735935e-05
X: 2.219818299



Open Methods

Fixed-Point Iteration

Description:

The user enters the $g(x)$ with $f(x)$, to calculate the next iteration x according to the formula

$$x_{i+1} = g(x_i)$$

Where $g(x) = x$, obtained by reformulating $f(x)$ with taking into consideration error bounds based upon the chosen $g(x)$.

With an initial guess x_0.

Pseudocode:

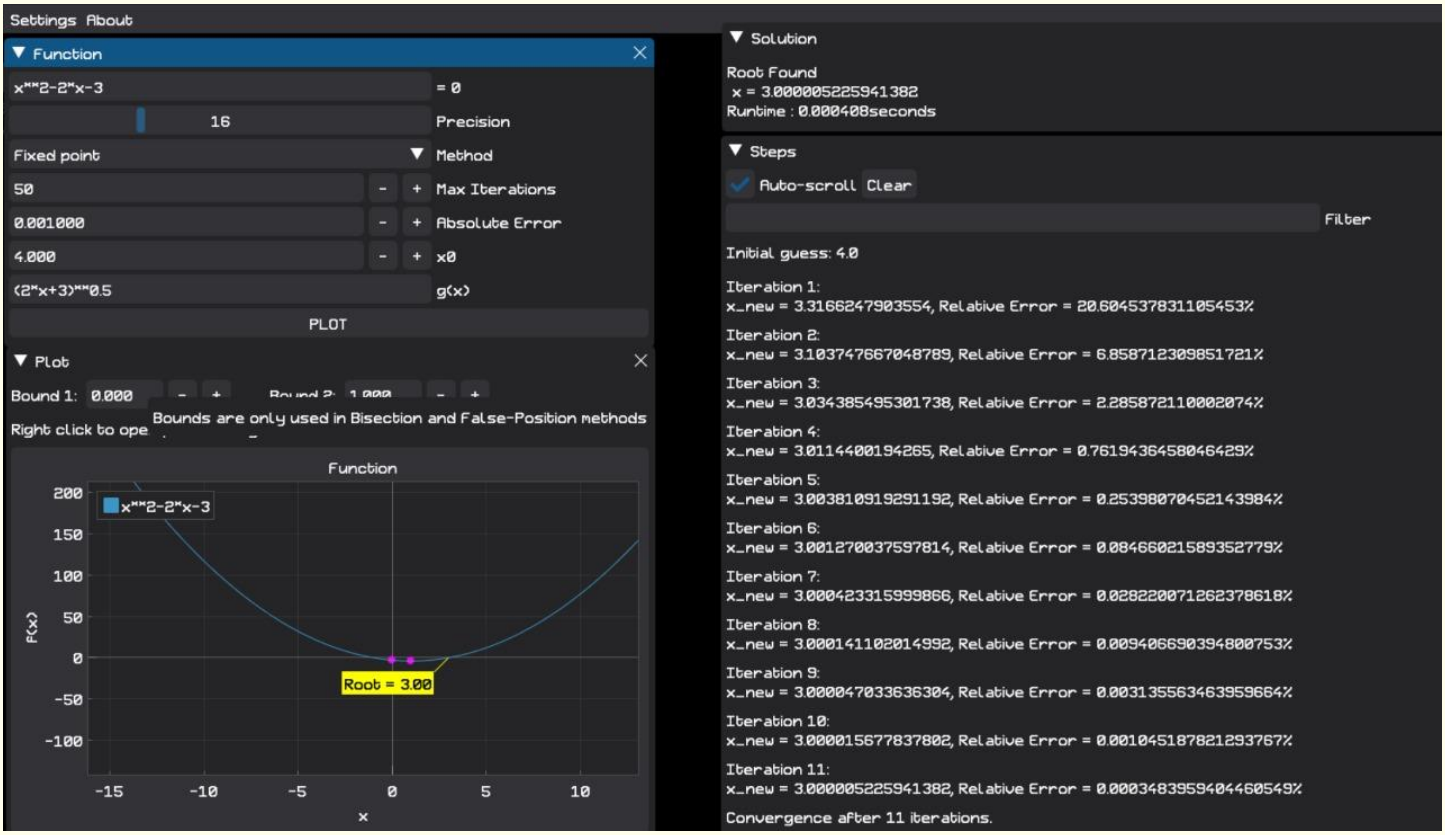
```
function fixed_point():
    threshold = 1e10
    x0 = round_to_sf(self.x0, self.sf)
    self.steps.append(f"Initial guess: {x0}")

    for i in range(self.maxiter):
```

```
        # Calculate the next approximation
        x_new = round_to_sf(self.g(x0), self.sf)

        # Calculate the relative error
        epsilon_a = abs((x_new - x0) / x_new) * 100 if x_new != 0 else float('inf')

        self.steps.append(f"Iteration {i + 1}:\nx_new = {x_new}, Relative Error = {epsilon_a}%")

        # Check for convergence
        if epsilon_a <= self.tol or self.f(x_new) == 0:
            self.steps.append(f"Convergence after {i + 1} iterations.")
            self.steps.append(f"Root: {x_new}")
            return x_new

        if abs(x_new) > threshold:
            self.steps.append("Diverge!!")
            return None

        # Update the current point
        x0 = x_new

    self.steps.append(f"Diverge!!")
    return None
```

Sample Runs:

$$f(x) = x^2 - 2x - 3$$

$$g(x) = +\sqrt{2x + 3}$$



Newton Raphson Method

Original Newton Raphson

Description:

Given the function, we use its first derivative to estimate the value of the new root according to the following formula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \qquad f'(x_i) = \frac{df(x_i)}{dx}$$

Given x_0, the new approximate is estimated and so on.

Assumptions:

In order to use Original Newton Raphson, we use the first modified stating multiplicity of root $m = 1$.

PseudoCode:

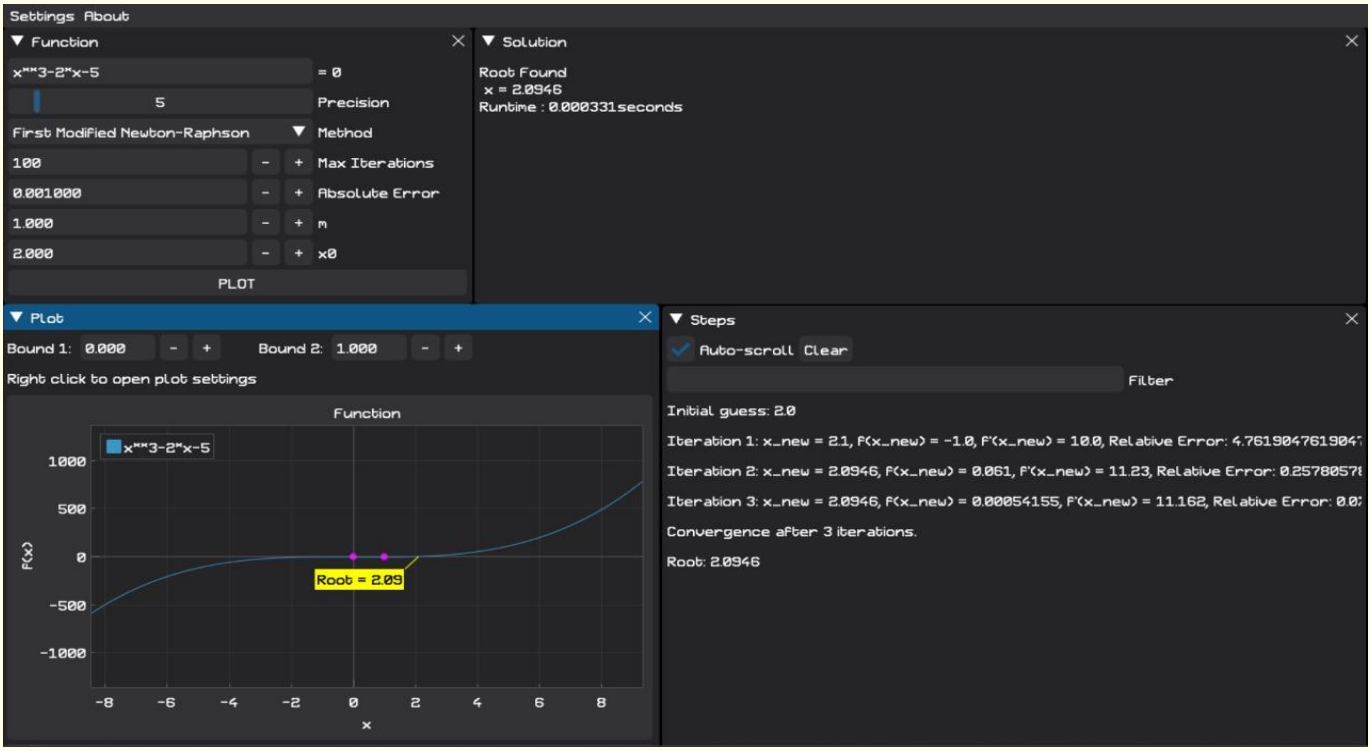
```
function solve():
    threshold = 1e10
    sf = self.sf
    x0 = round_to_sf(self.x0, sf)
    func = self.func
    df = self.df
    self.steps.append(f"Initial guess: {x0}")
```

```
for i in range(self.maxiter):
    # Evaluate function and its derivative
    f_x0 = round_to_sf(func(x0), sf)
    df_x0 = round_to_sf(df(x0), sf)

    if df_x0 == 0:
        self.steps.append(f"Division by zero at iteration {i + 1}, derivative is zero.")
        break

    # Calculate the next approximation for the root
    x_new = round_to_sf(x0 - self.m * (f_x0 / df_x0), sf)

    # Calculate the relative error
    epsilon_a = abs(((x_new - x0) / x_new) * 100) if x_new != 0 else float('inf')
    self.steps.append(f"Iteration {i + 1}: x_new = {x_new}, f(x_new) = {round_to_sf(f_x0, sf)}, f'(x_new) = {round_to_sf(df_x0, sf)}, Relative Error: {epsilon_a}%")

    # Check for convergence
    if epsilon_a <= self.tol or func(x_new) == 0:
        self.steps.append(f"Convergence after {i + 1} iterations.")
        self.steps.append(f"Root: {x_new}")
        return x_new

    if abs(x_new) > threshold:
        self.steps.append("Diverge!!")
        return None

    # Update the current point
    x0 = x_new

self.steps.append(f"No convergence after {self.maxiter} iterations.")
return None
```

Sample Runs:

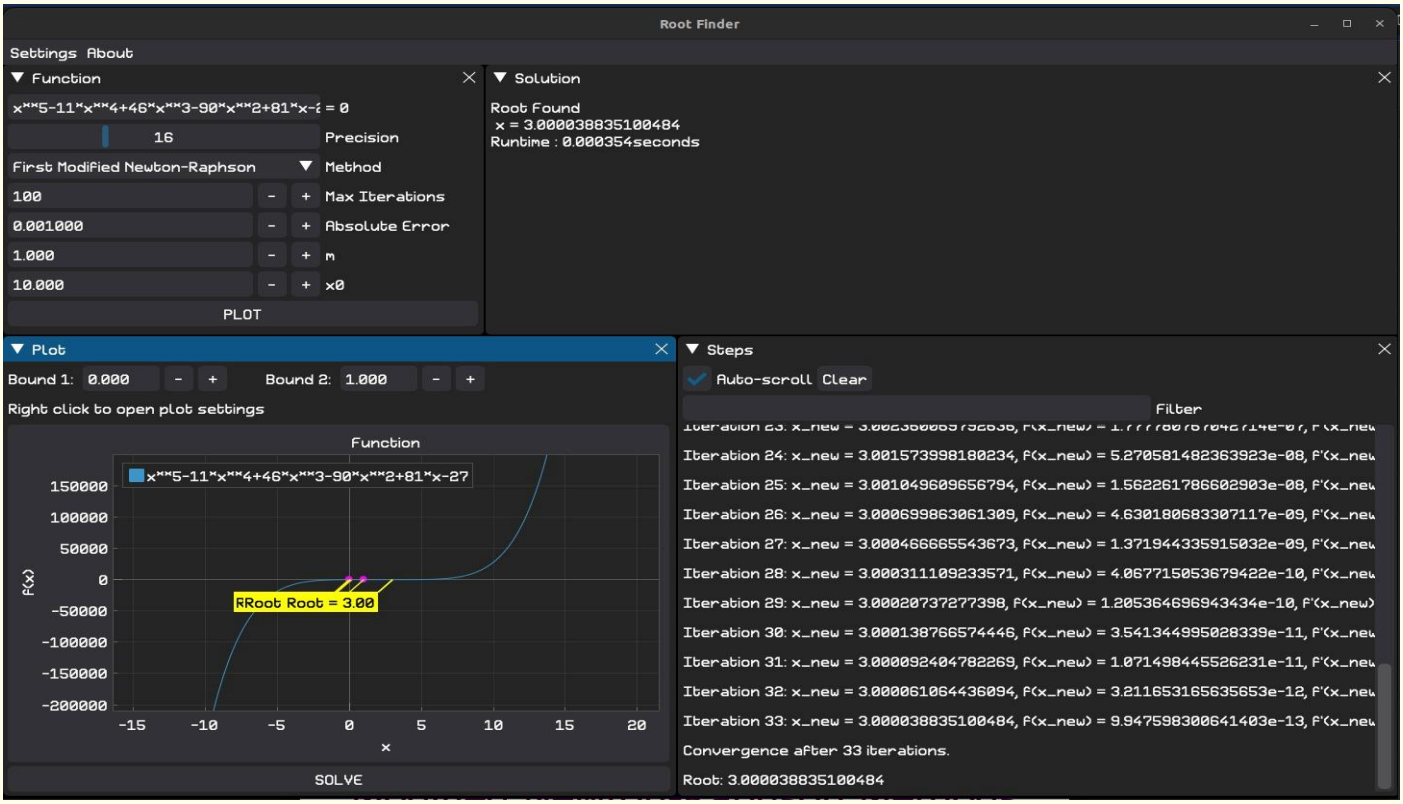
$f(x) = x^3 - 2x - 5, m = 1, x_0 = 2$

Yields a root of value 2.0946 using 5 significant figures



$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27,\ m = 1, x_0 = 10$

Yields a root of value 3.0000388 using 16 significant figures.



First Modified Newton Raphson

Description:

Used to solve equations that have roots of multiplicity m, at a specific point, since bracketing methods don't work for such equations. Original Newton Raphson and Secant work with such functions but with slower rates of convergence, thus we preferably use the modified Newton Raphson methods First or Second.

Given the function, we use its first derivative and the multiplicity of the root to estimate the value of the new root according to the following formula

$$x_{i+1} = x_i - m \times \frac{f(x_i)}{f'(x_i)}, \qquad f'(x_i) = \frac{df(x_i)}{dx}$$

Given x_0 and m, the new approximate is estimated and so on.

Assumptions:

This Method only works when the multiplicity of the root m is known, accordingly we might face some limitations in its usage.

PseudoCode:

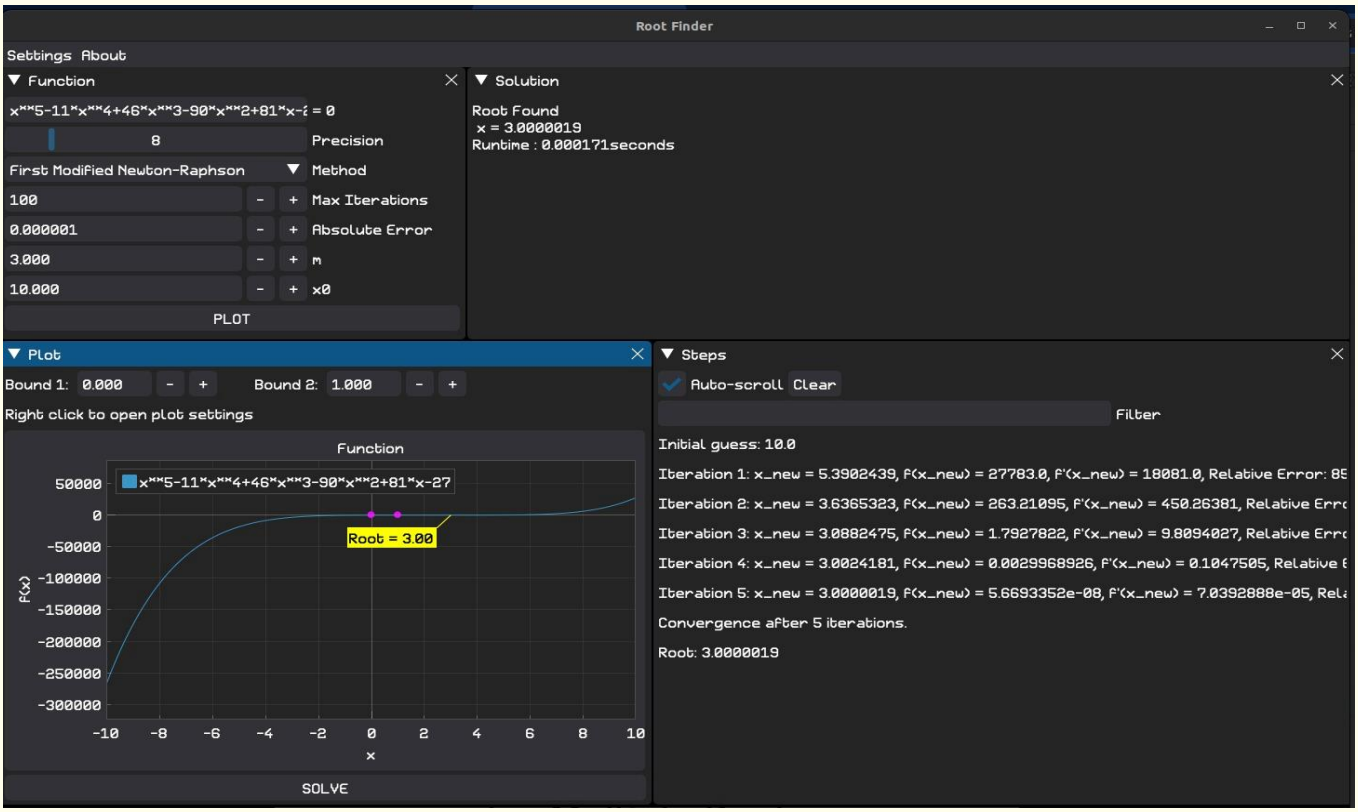
Same as for Original Newton Raphson.

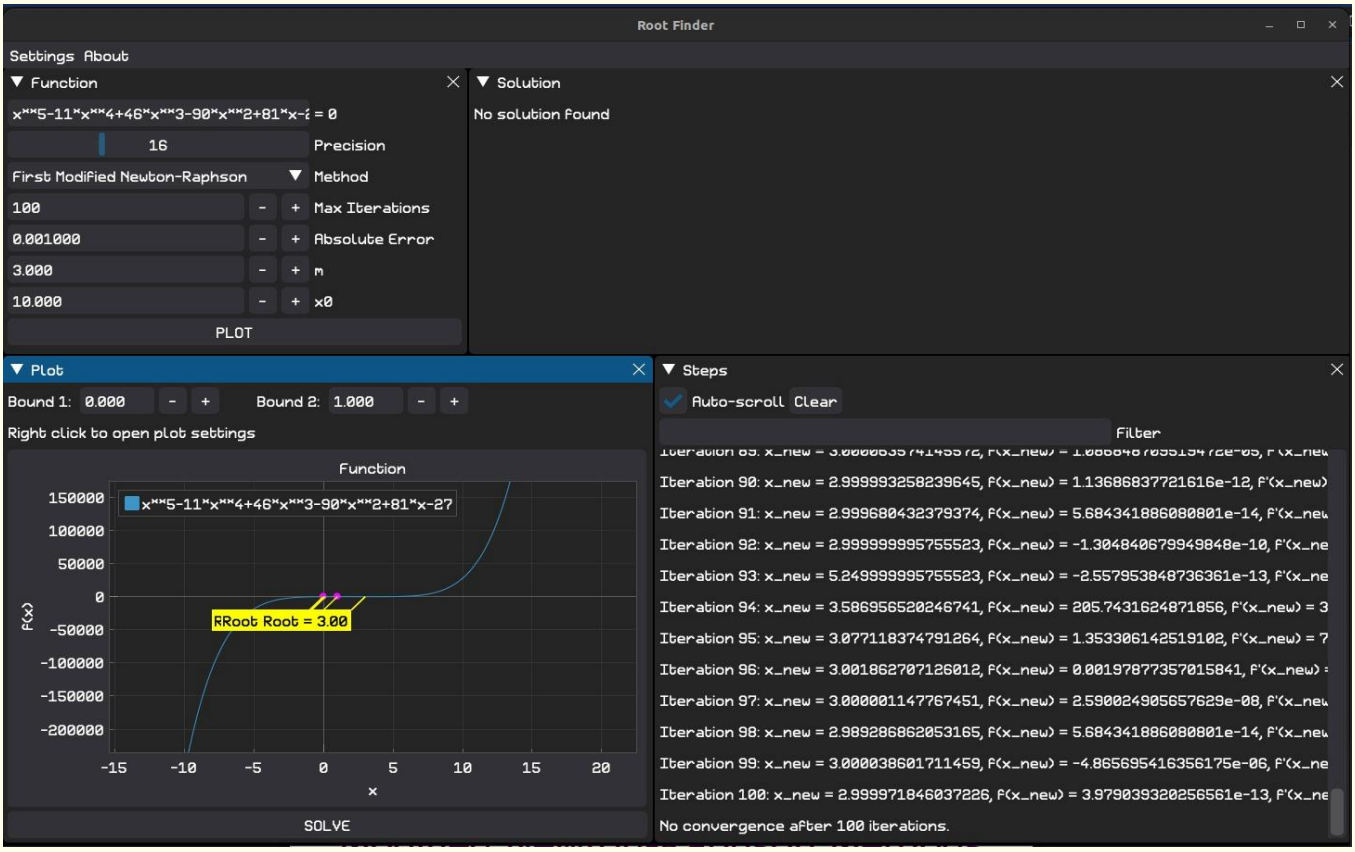
Sample Runs:

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27,\ m = 3, x_0 = 10$$

Yields a root of value 3.0000019 using 8 significant figures.

Yields a root of value 2.999971846037226 using 16 significant figures, though the error oscillates at larger significant figures and it doesn't converge.





Second Modified Newton Raphson

Description:

Given the function, we use its first and second derivatives to estimate the value of the new root according to the following formula

$$x_{i+1} = x_i - \frac{f(x_i) \times f'(x_i)}{(f'(x_i))^2 - f(x_i) \times f''(x_i)}, \qquad f'(x_i) = \frac{df(x_i)}{dx}, \qquad f''(x_i) = \frac{df'(x_i)}{dx}$$

Given x_0, no need for multiplicity of root m, the new approximate is estimated and so on.

PseudoCode:

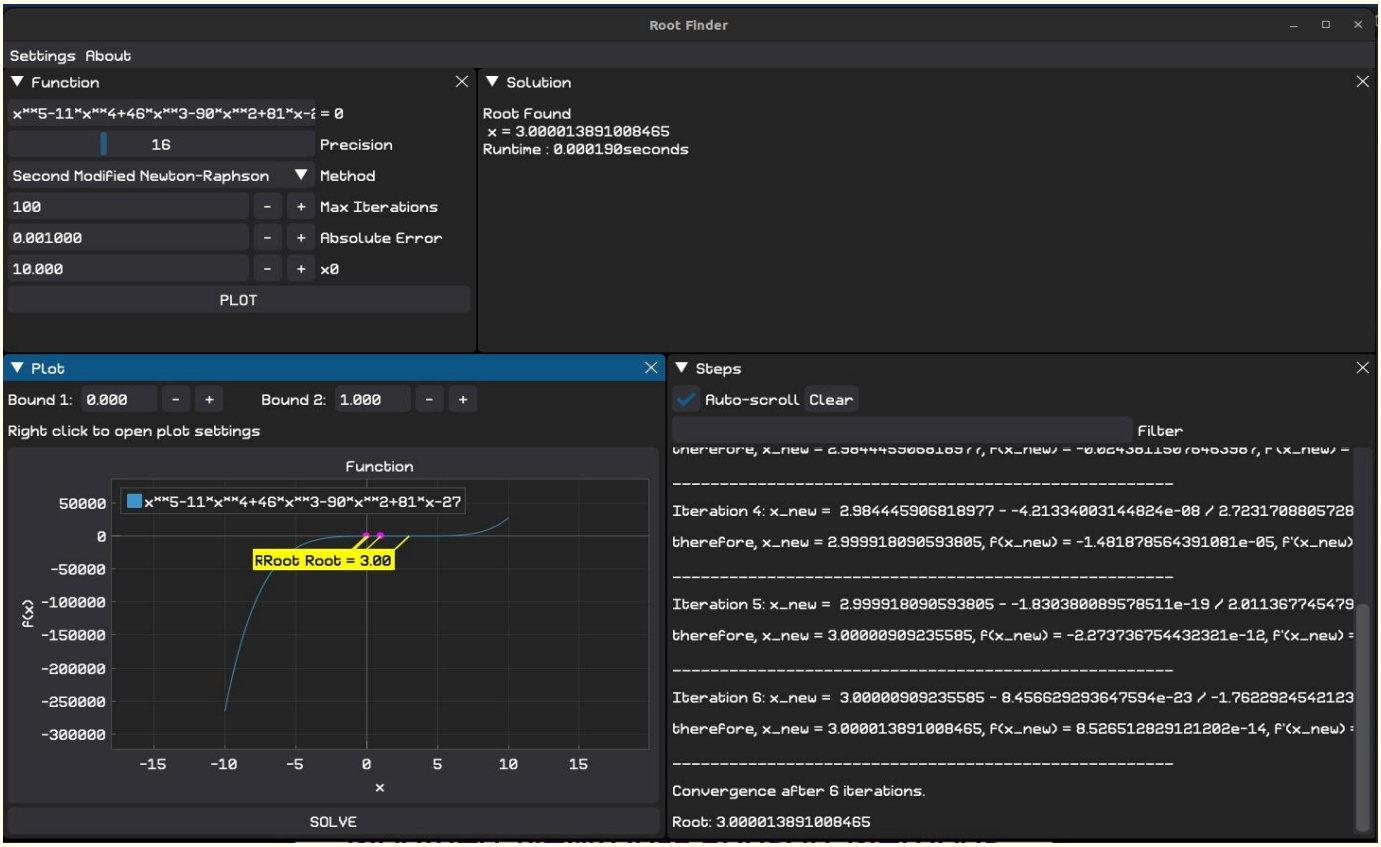
```
function second_modified_NR():
    threshold = 1e10
    sf = self.sf
    x0 = round_to_sf(self.x0, sf)
    func = self.func
    df = self.df
    sdf = self.sdf  # Second derivative function
    self.steps.append(f"Initial guess: {x0}")
```

```
for i in range(self.maxiter):
    # Evaluate function, its first derivative, and second derivative
    f_x0 = round_to_sf(func(x0), sf)
    df_x0 = round_to_sf(df(x0), sf)
    sdf_x0 = round_to_sf(sdf(x0), sf)

    if f_x0 == 0:
        self.steps.append(f"Convergence after {i} iterations.")
        self.steps.append(f"Root: {x0}")
        return x0

    if df_x0 * df_x0 == f_x0 * sdf_x0:
        self.steps.append(f"Division by zero at iteration {i + 1}, derivative is zero.")
        return None

    numerator = round_to_sf(f_x0 * df_x0, sf)
    denominator = round_to_sf(df_x0**2 - f_x0 * sdf_x0, sf)

    # Calculate the next approximation for the root using the second derivative
    x_new = round_to_sf(x0 - (numerator) / (denominator), sf)
    self.steps.append(f"Iteration {i + 1}: x_new =  {x0} - {numerator} / {denominator}")

    # Calculate the relative error
    epsilon_a = abs(((x_new - x0) / x_new) * 100) if x_new != 0 else float('inf')
    self.steps.append(
        f"therefore, x_new = {x_new}, f(x_new) = {f_x0}, f'(x_new) = {df_x0}, f''(x_new) = {sdf_x0}, Relative Error: {epsilon_a}%\n"
    )
    self.steps.append("_____")

    # Check for convergence
    if abs(x_new) > threshold:
        self.steps.append("Diverge!!")
        return None

    if epsilon_a <= self.tol or func(x_new) == 0:
        self.steps.append(f"Convergence after {i + 1} iterations.")
        self.steps.append(f"Root: {x_new}")
        return x_new

    # Update the current point
    x0 = x_new

self.steps.append(f"No convergence after {self.maxiter} iterations.")
return None
```

Sample Runs:

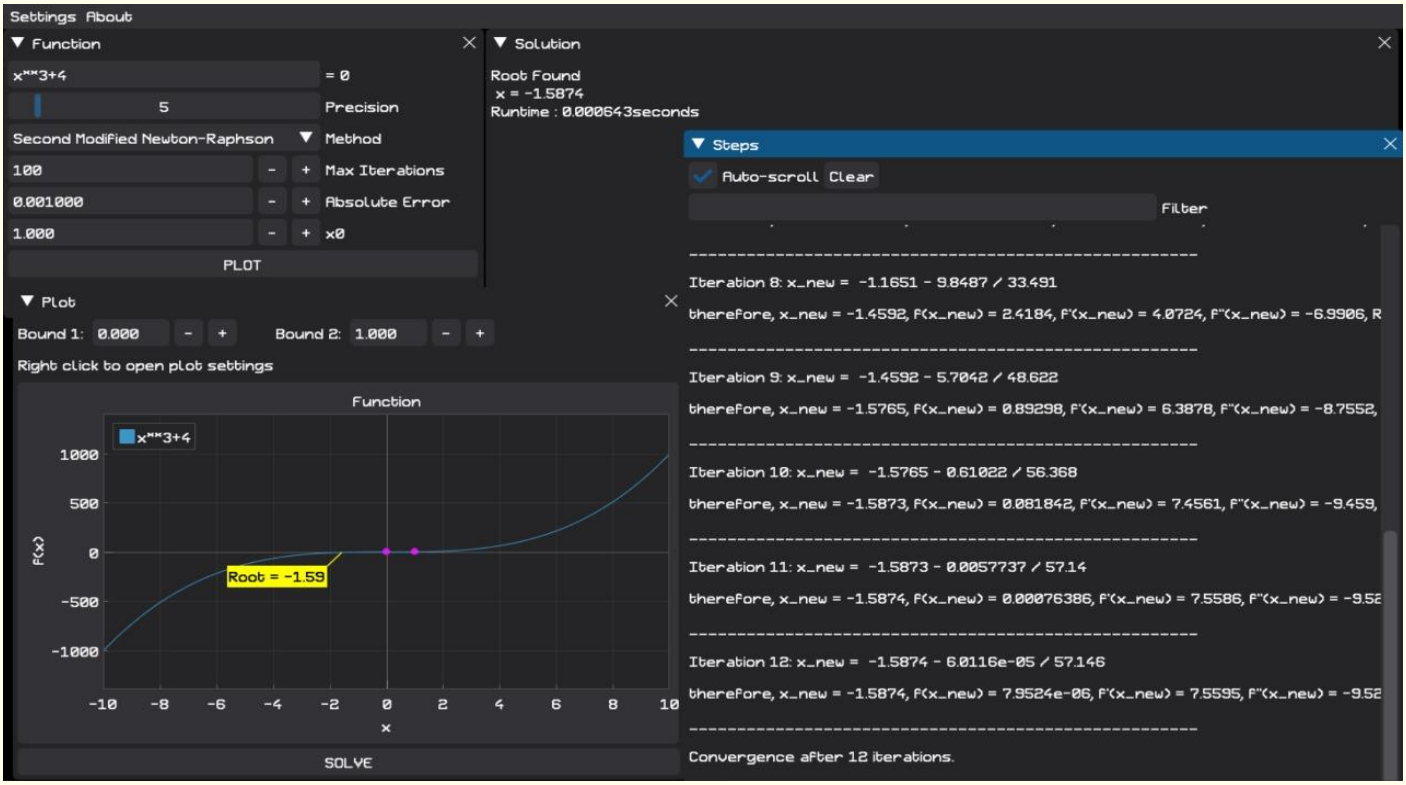
$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27,\ x_0 = 10$

Yields a root of value 3.0000014 using 16 significant figures.



$f(x) = x^3 + 4, \qquad x_0 = 1$

Yields a root of value -1.5874 using 5 significant figures after 12 iterations.



Secant Method

Description:

Given $x_i \; and \; x_{i-1}$, the value for the estimated root will follow the following formula

$$x_{i+1} = x_i - f(x_i) \times \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Note:

The derivative is substituted, by taking the rate of change between each x_i.

PseudoCode

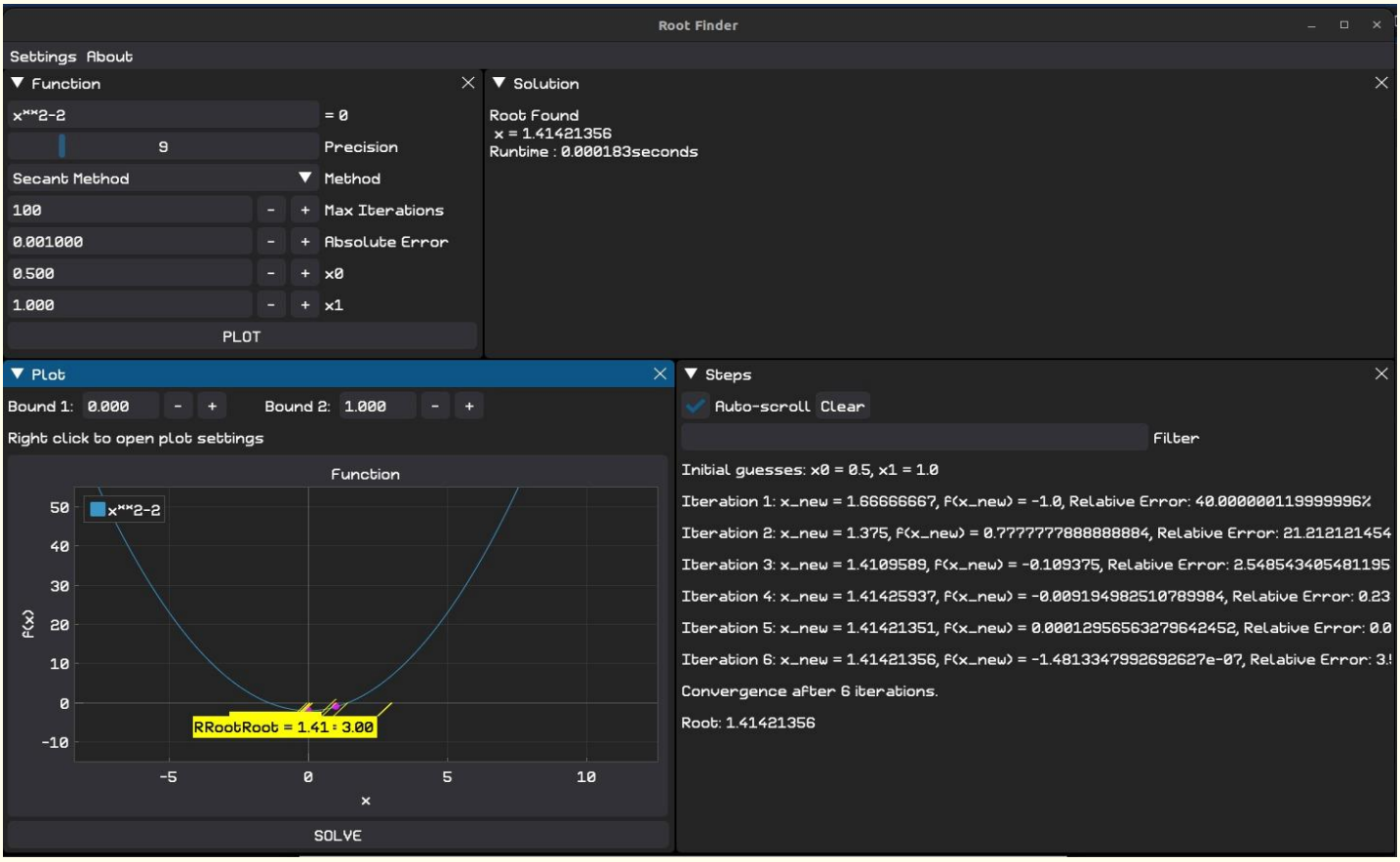
```
function solve():
    threshold = 1e10
    x0 = self.x0
    x1 = self.x1
    tol = self.tol
    max_iter = self.max_iter
    func = self.func
    sf = self.sf

    self.steps.append(f"Initial guesses: x0 = {x0}, x1 = {x1}")

    for i in range(max_iter):
```

```
    f_x0 = func(x0)
    f_x1 = func(x1)
    x0_minus_x1 = round_to_sf(x0 - x1, sf)

    # Check if any intermediate calculation is not valid
    if any(np.isinf(val) or np.isnan(val) for val in [f_x1, f_x0, x0_minus_x1]):
        self.steps.append("Error: intermediate calculation resulted in inf or nan at iteration {}".format(i + 1))
        return None

    if f_x1 - f_x0 == 0:
        self.steps.append(f"Division by zero at iteration {i + 1}, cannot continue.")
        return None

    x_new = round_to_sf(x1 - f_x1 * (x0_minus_x1) / (f_x0 - f_x1), sf)
    epsilon_a = abs((x_new - x1) / x_new) * 100 if x_new != 0 else float('inf')

    self.steps.append(f"Iteration {i + 1}: x_new = {x_new}, f(x_new) = {f_x1}, Relative Error: {epsilon_a}%")

    if epsilon_a <= tol or self.func(x_new) == 0:
        self.steps.append(f"Convergence after {i + 1} iterations.")
        self.steps.append(f"Root: {x_new}")
        return x_new

    if abs(x_new) > threshold:
        self.steps.append("Diverge!!")
        return None

    x0 = x1
    x1 = x_new

self.steps.append(f"No convergence after {max_iter} iterations.")
return None
```

Sample Runs

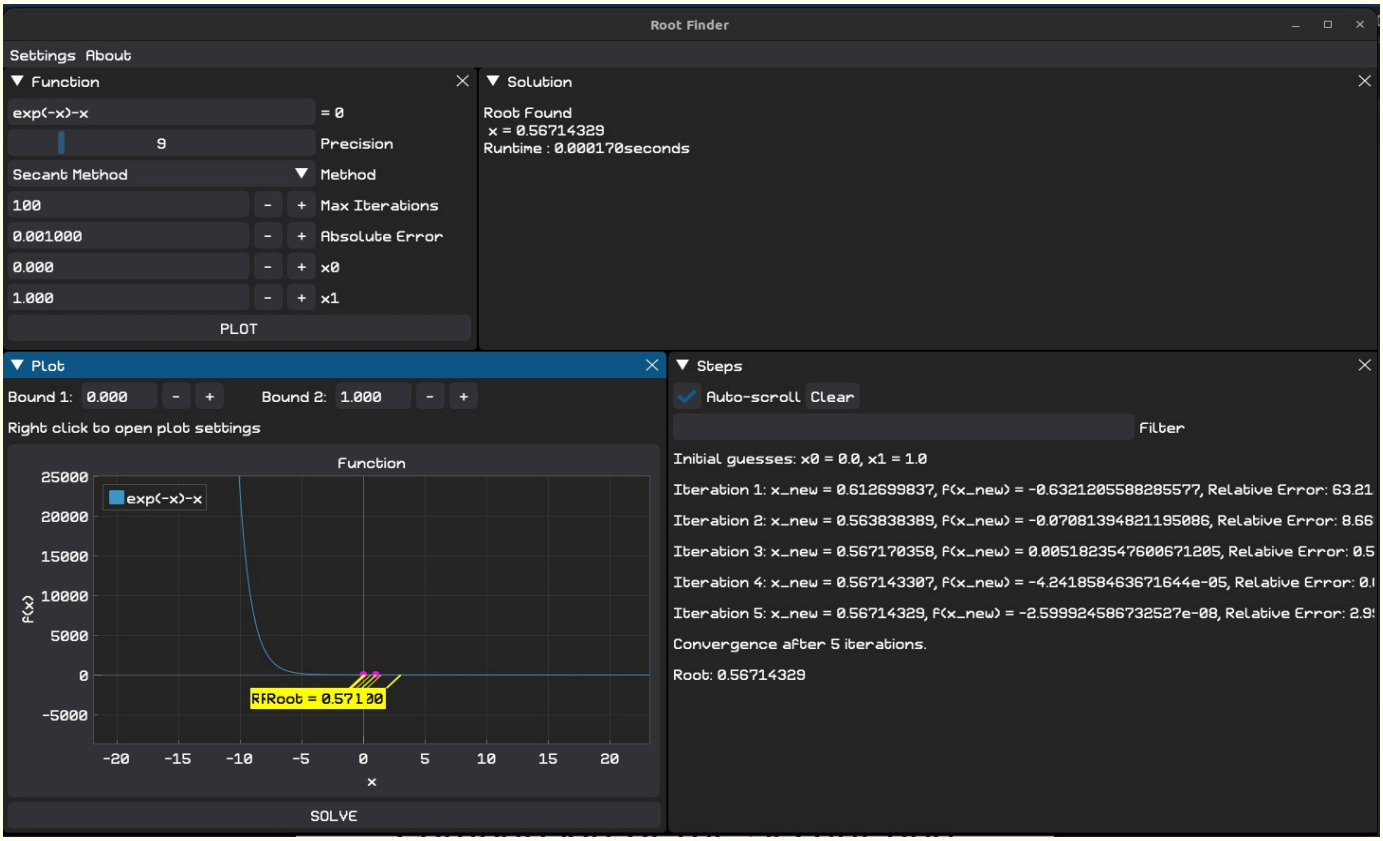
$f(x) = x^2 - 2\,, \qquad x_0 = 0.5, \qquad x_1 = 1$

Yields a root of value 1.41421356 using 9 significant figures.



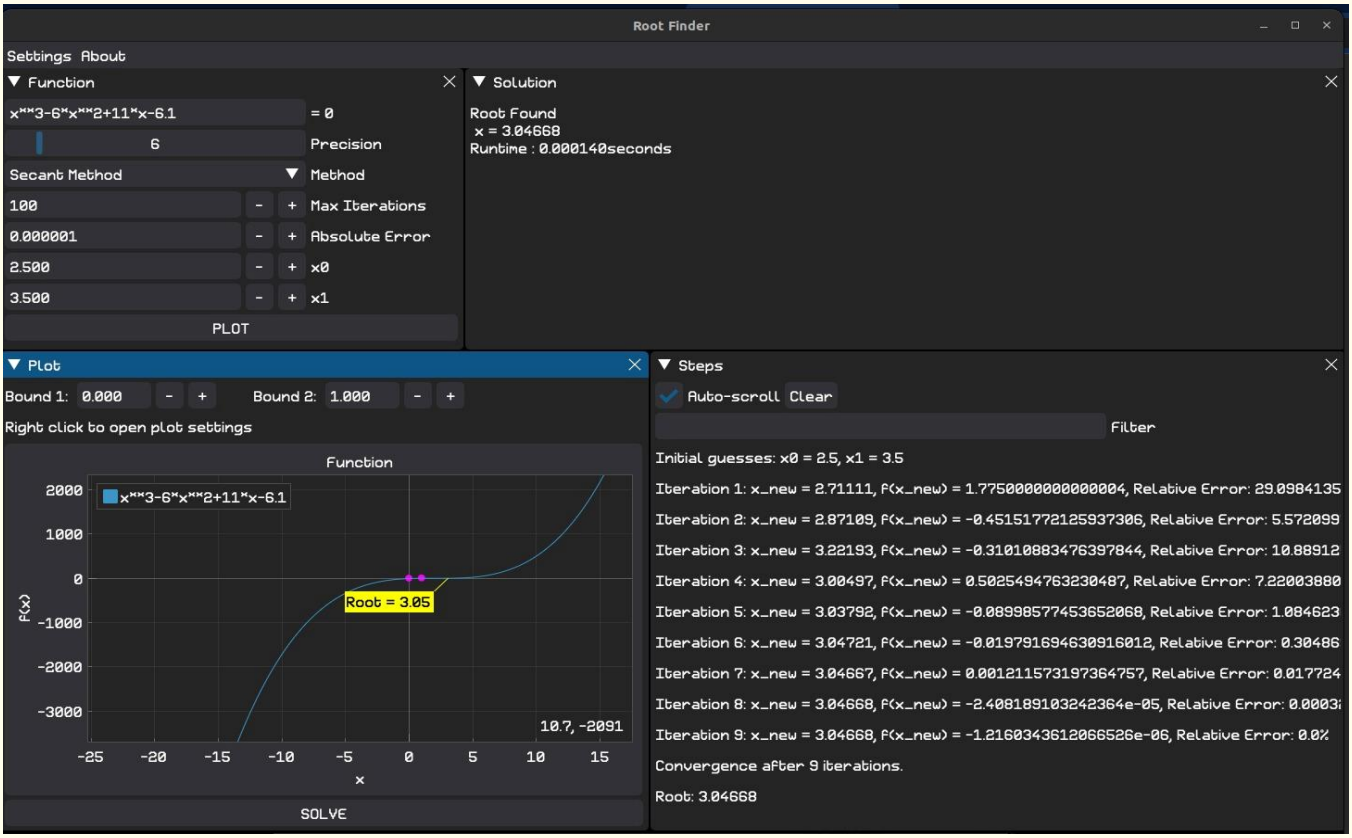
$f(x) = e^{-x} - x\,, \qquad x_0 = 0, \qquad x_1 = 1$

Yields a root of value 0.56714329 using 9 significant figures.



$f(x) = x^3 - 6x^2 + 11x - 6.1\,, \qquad x_0 = 2.5, \qquad x_1 = 3.5$

Yields a root of value 3.04668 using 6 significant figures.



Comparison Between Different Methods

Bisection Method

- Time Complexity O($\log n$), where n is the number of iterations.
- Convergence: It converges -reducing the interval size by half in each iteration- given the circumstances where roots exist, and the function passes the $x - axis$ not only touches it as $f(x) = x^2$
- Minimum Iterations required follows the given formula

$$k \geq \log_2\left(\left|\frac{x_u - x_l}{E_a}\right|\right), \qquad k \text{ is the number of iterations}, \qquad E_a \text{ is the Tolerance error}$$

False Position

- Time complexity O($\log n$) , where n is the number of iterations.
- Convergence: Always converge and mostly faster than *bisection method,* though -since the interval adapts based on the function values- in some cases it might be better using bisection depending on the function itself, its derivative, smoothness, and continuity.
- Best when the function is continuous and changes its state rapidly.

Newton Raphson

Newton Raphson

- Time complexity O(n), where n is the number of iterations.

- Converges rapidly, with the error decreasing with each iteration.
- May fail to converge rapidly, at functions with multiple roots, or with bad initial guesses.

First Modified Newton Raphson

- Similar to Newton Raphson, though it is better regarding functions with roots of multiplicity m.

Second Modified Newton Raphson

- Alike the previous ones, however one improvement is that the multiplicity of a root m is not required to be known previously unlike the *First Modified Newton Raphson*.
- The use of the second derivative can affect the convergence rate positively for certain functions.

Secant

- Time complexity O(n), where n is the number of iterations.
- Converges rapidly relatively equivalent to *Newton Raphson* but with two initial guesses instead of one and doesn't require the derivative of the function since it uses the magnitude difference.
- May diverge for bad choice of initial points.

Test Cases

All Test Cases are solved using 5 Significant Figures, with the default maximum iterations as 50 and error tolerance $\varepsilon_{tol} = 10^{-5}$.

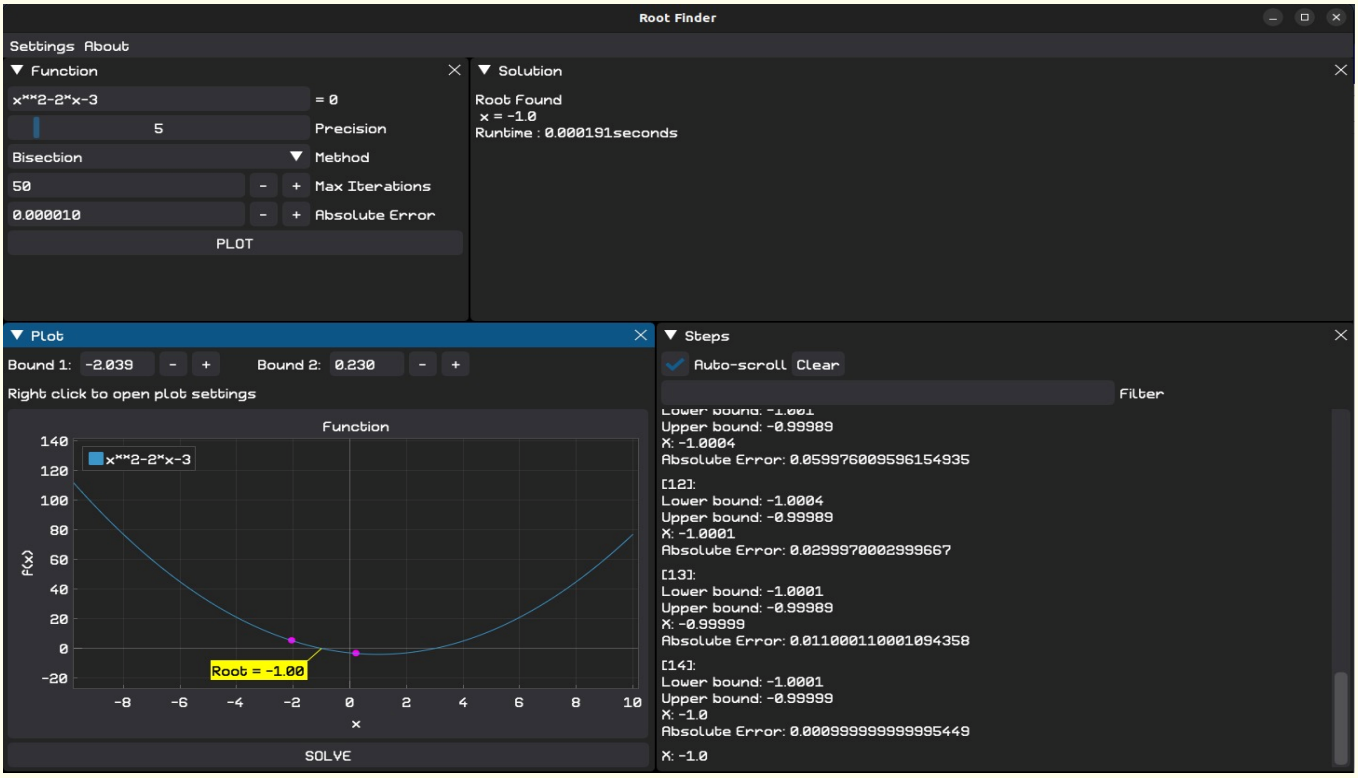
Case 1

$$f(x) = x^2 - 2x - 3$$

Bisection

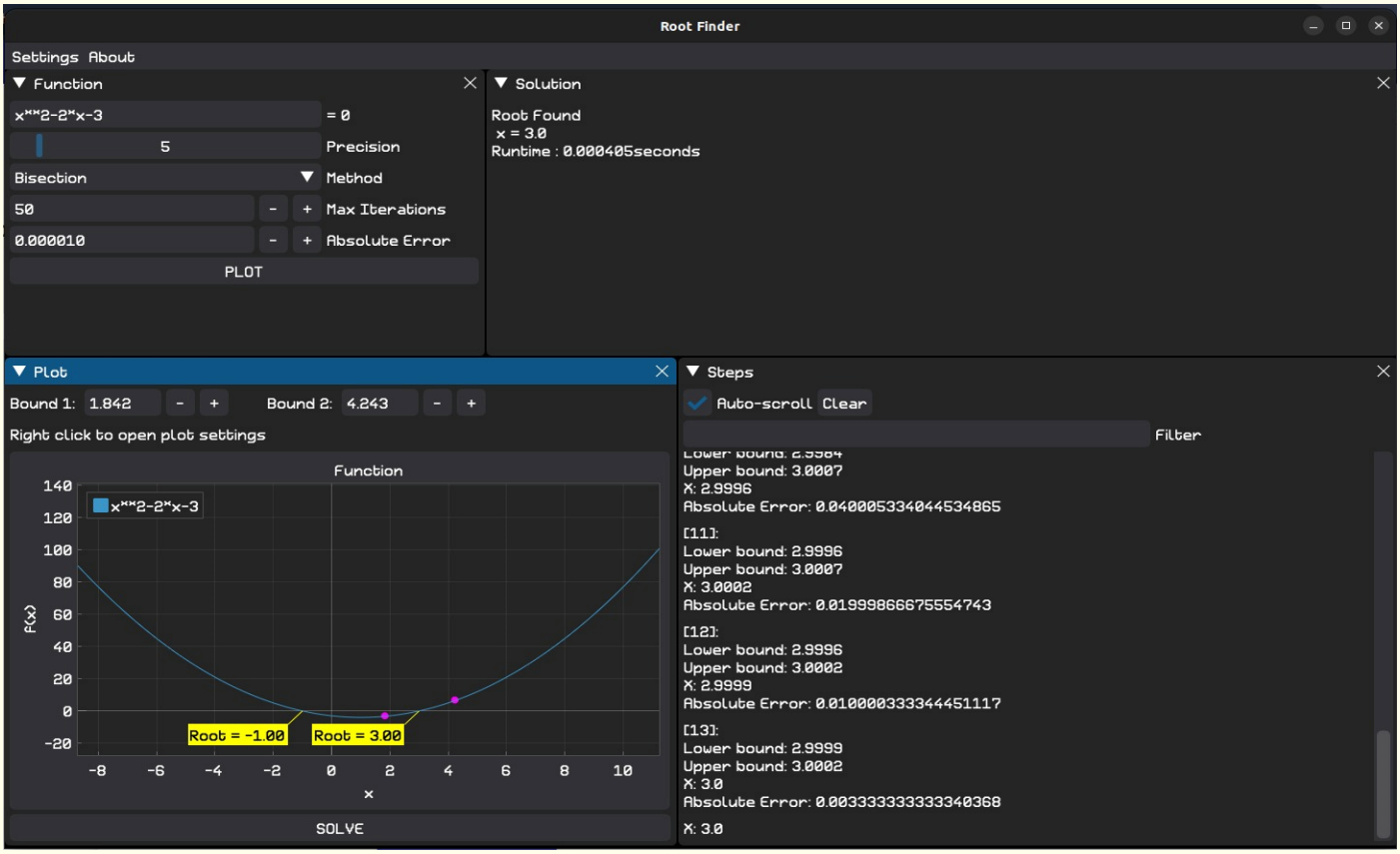
$x_l = -2.039, \quad x_u = 0.230$

Yields a root of value -1.0 after 14 iterations.



$x_l = 1.842, \quad x_u = 4.243$

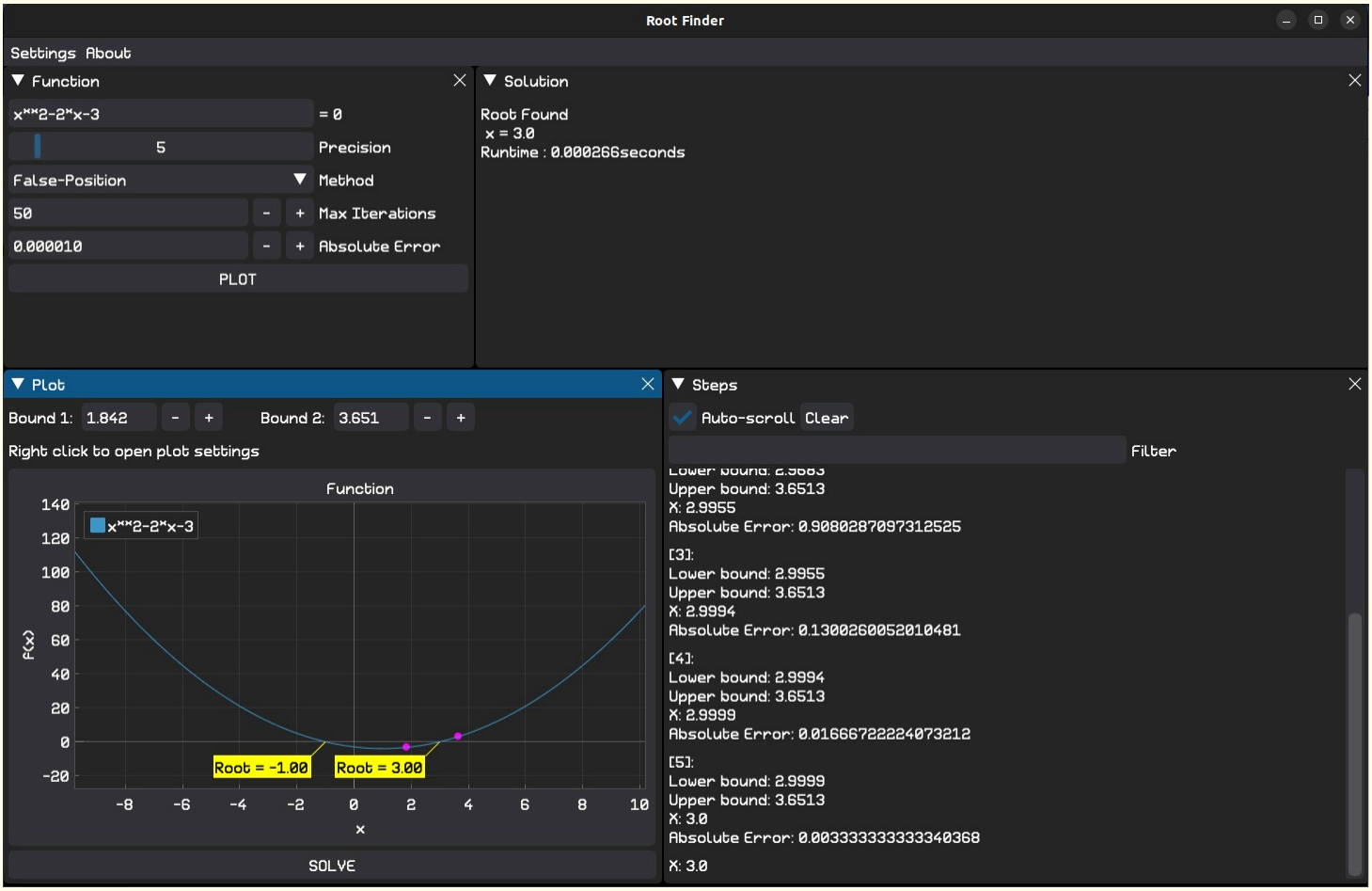
Yields a root of value 3 after 13 iterations.



False Position

$x_l = 1.842, \quad x_u = 3.651$

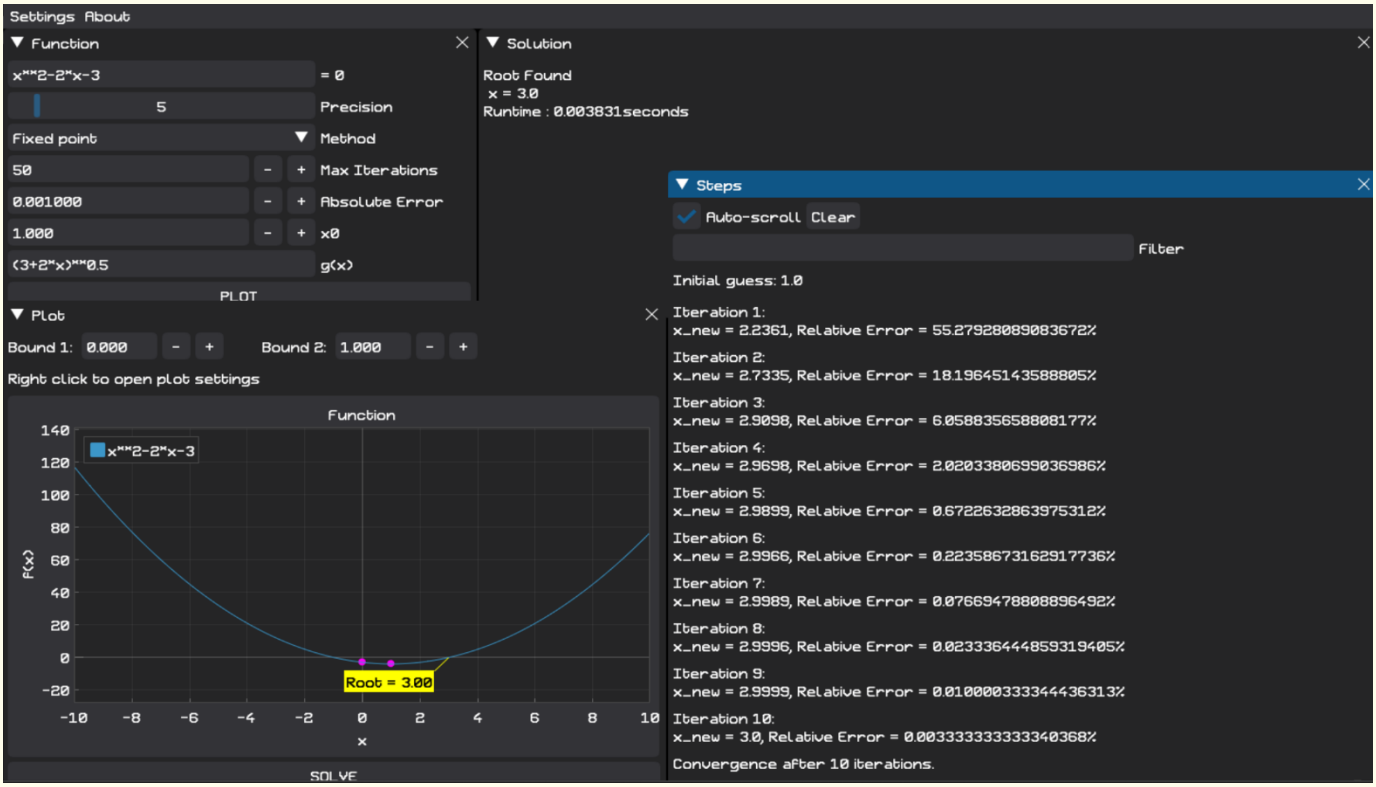
Yields a root of value 3 after 5 iterations.



Fixed Point

$x_0 = 1.0, \quad \varepsilon_{tol} = 10^{-3}, \qquad g(x) = \sqrt{3 + 2x}.$

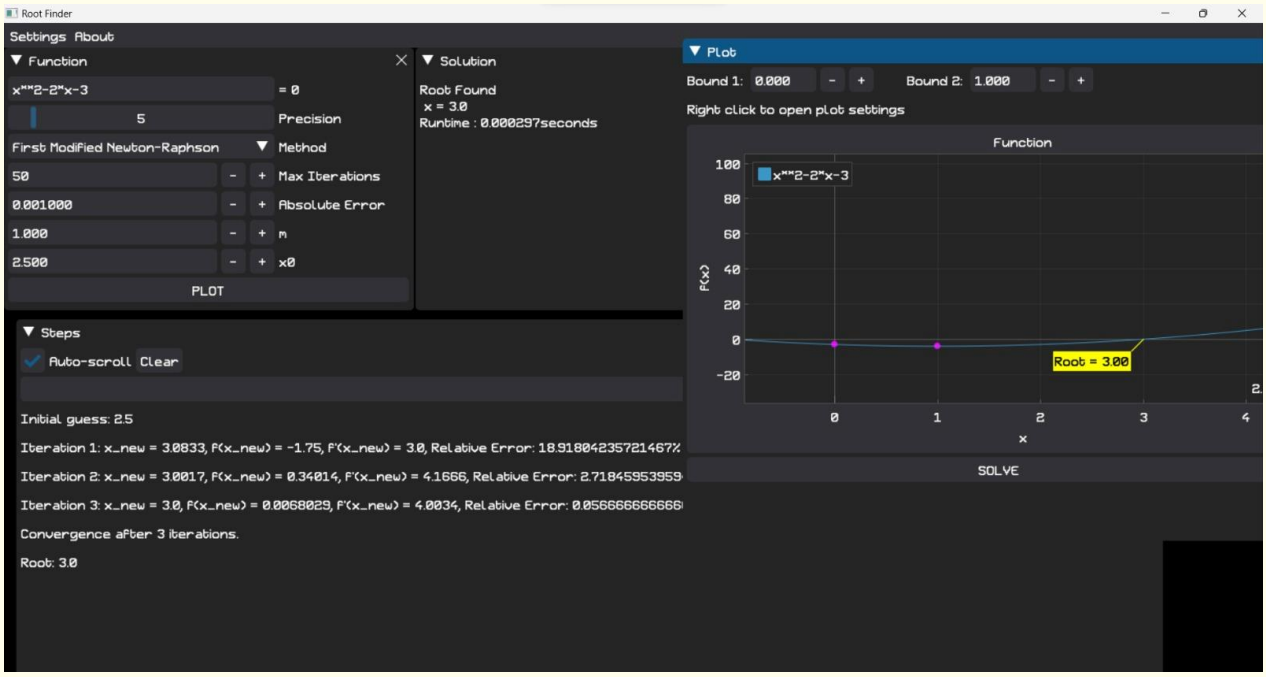
Yields a root of value 3.0 after 10 iterations.



First Modified Newton

$x_0 = 2.5, \quad \varepsilon_{tol} = 10^{-3}, \quad m = 1$

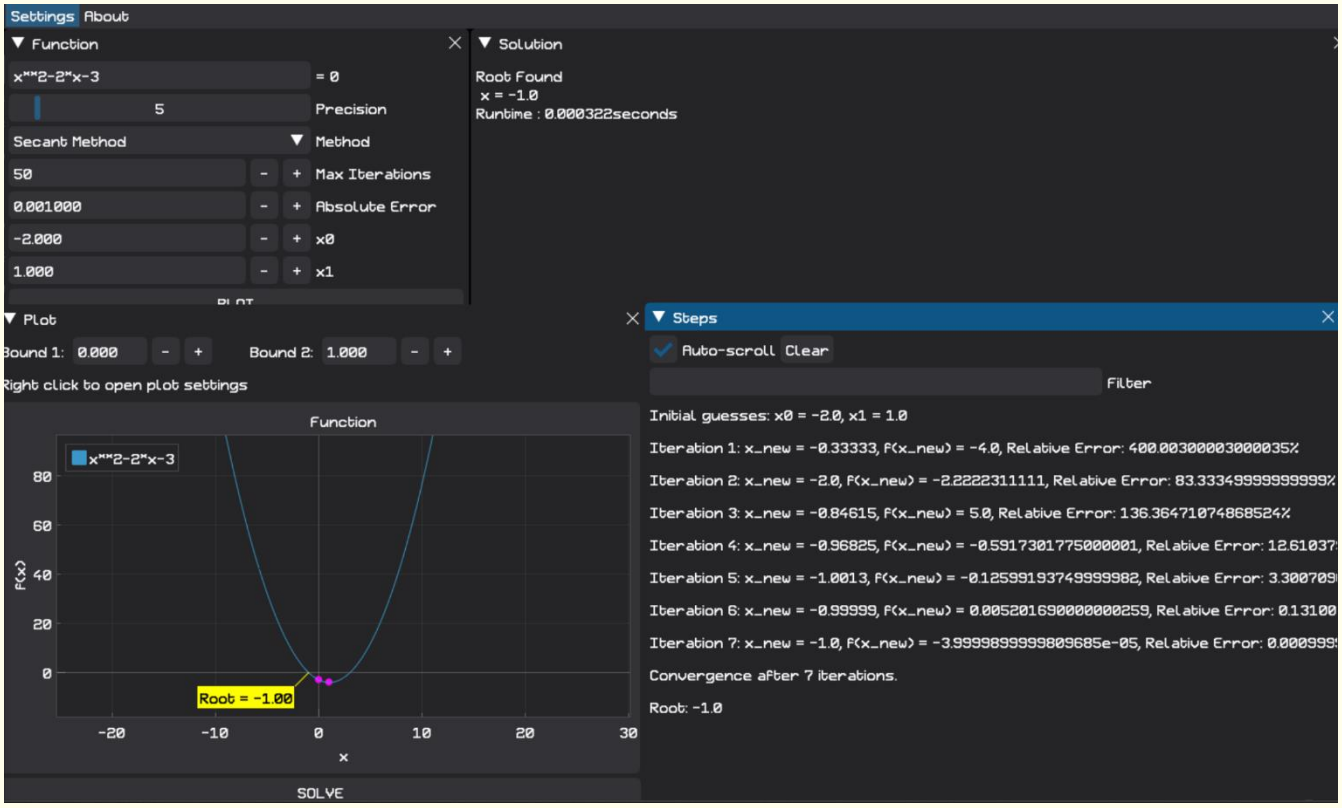
Yields a root of value 3.0 after 3 iterations.



Secant

$x_0 = -2.0, \quad x_1 = 1.0, \quad \varepsilon_{tol} = 10^{-3}.$

Yields a root of value -1 after 7 iterations.



Comparison

Observing the outputs of each method the following table is constructed.

Method	No. of Iterations	Run Time	Root	Initial Guesses
Bisection	14	0.000191	-1.0	$x_l = -2.039, x_u = 0.230$
	13	0.000405	3.0	$x_l = 1.842, x_u = 4.243$
False Position	5	0.000266	3.0	$x_l = 1.842, x_u = 3.651$
Fixed Point	10	0.003831	3.0	$x_0 = 1.0,\ g(x) = \sqrt{3 + 2x}$
Newton Raphson	3	0.000297	3.0	$x_0 = 2.5$
Secant	7	0.000322	-1.0	$x_0 = -2.0,\quad x_1 = 1.0$

Regarding the functions for the same root, throughout testing it is noticed that the number of iterations varies based upon the initial guesses and how relevant are they to the actual value of the root and the method used.

It is concluded that, given the right initial values, false position is faster than bisection and newton Raphson and secant are relatively close for this function.

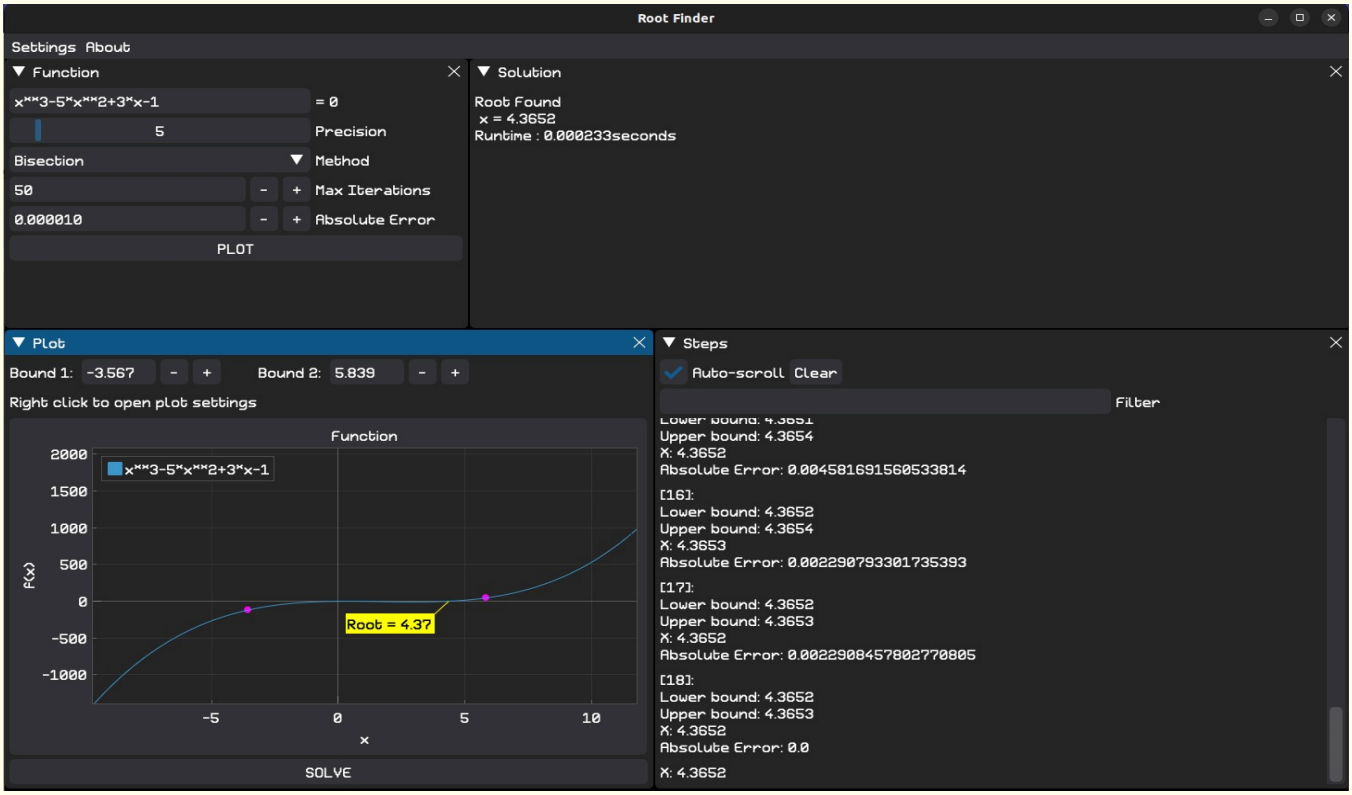
Case 2

$$f(x) = x^3 - 5x^2 + 3x - 1$$

Bisection

$x_l = -3.567, \quad x_u = 5.839$

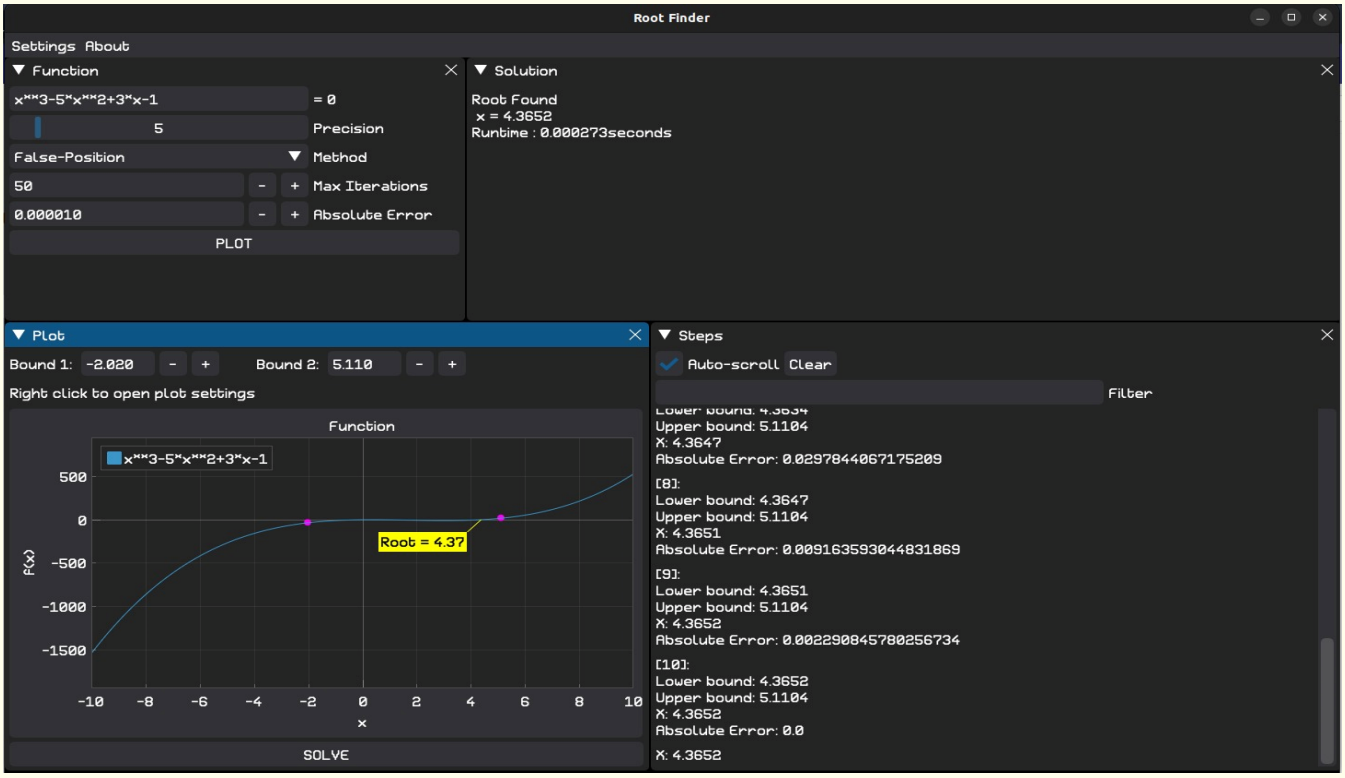
Yields a root of value 4.3652 after 18 iterations.



False Position

$x_l = -2.020, \quad x_u = 5.110$

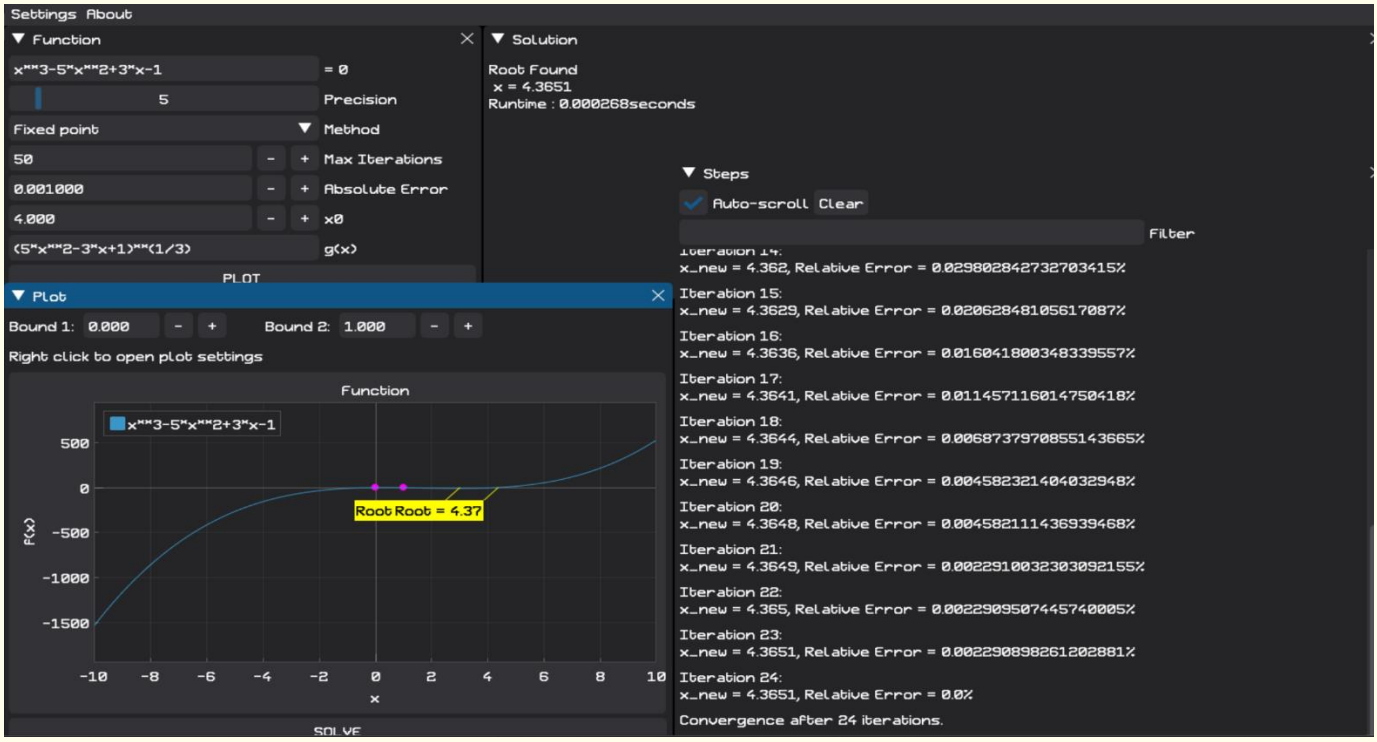
Yields a root of value 4.3652 after 10 iterations.



Fixed Point

$x_0 = 4, \quad \varepsilon_{tol} = 10^{-3}, \quad g(x) = \sqrt[3]{5x^2 - 3x + 1}$

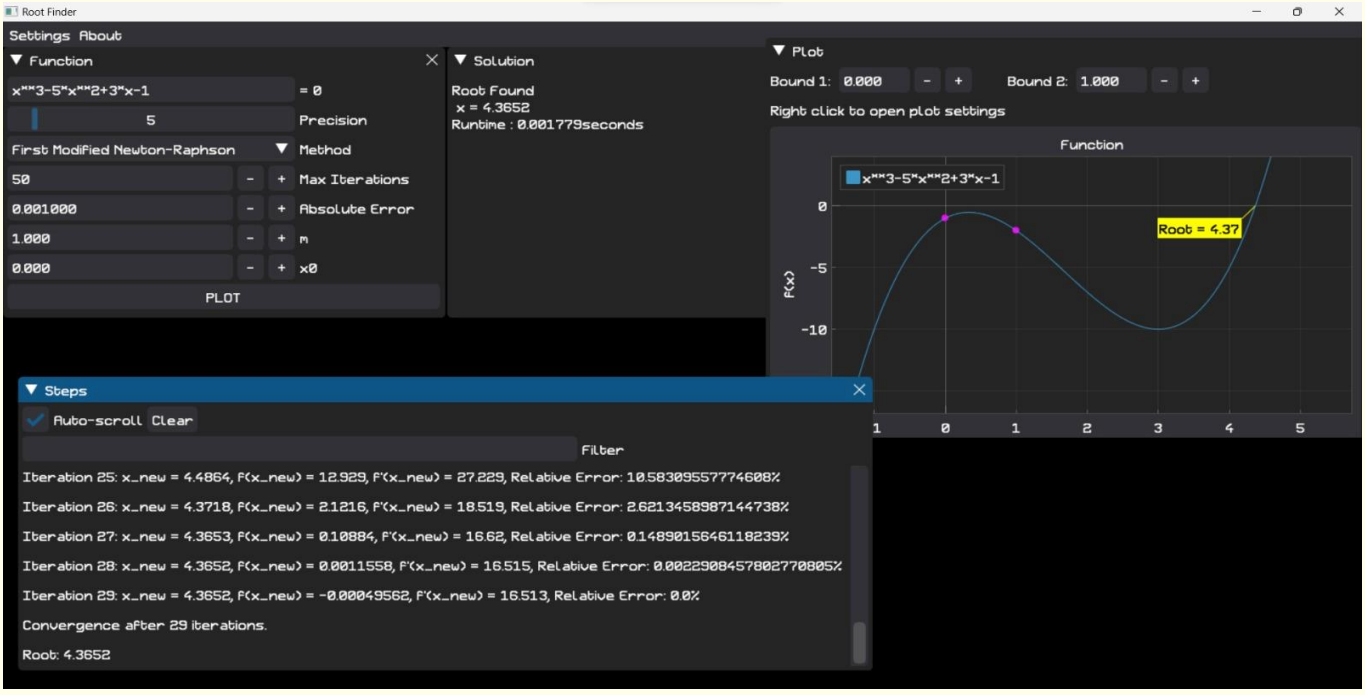
Yields a root of value 4.3651 after 24 iterations.



First Modified Newton

$x_0 = 0, \quad \varepsilon_{tol} = 10^{-3}, \quad m = 1$

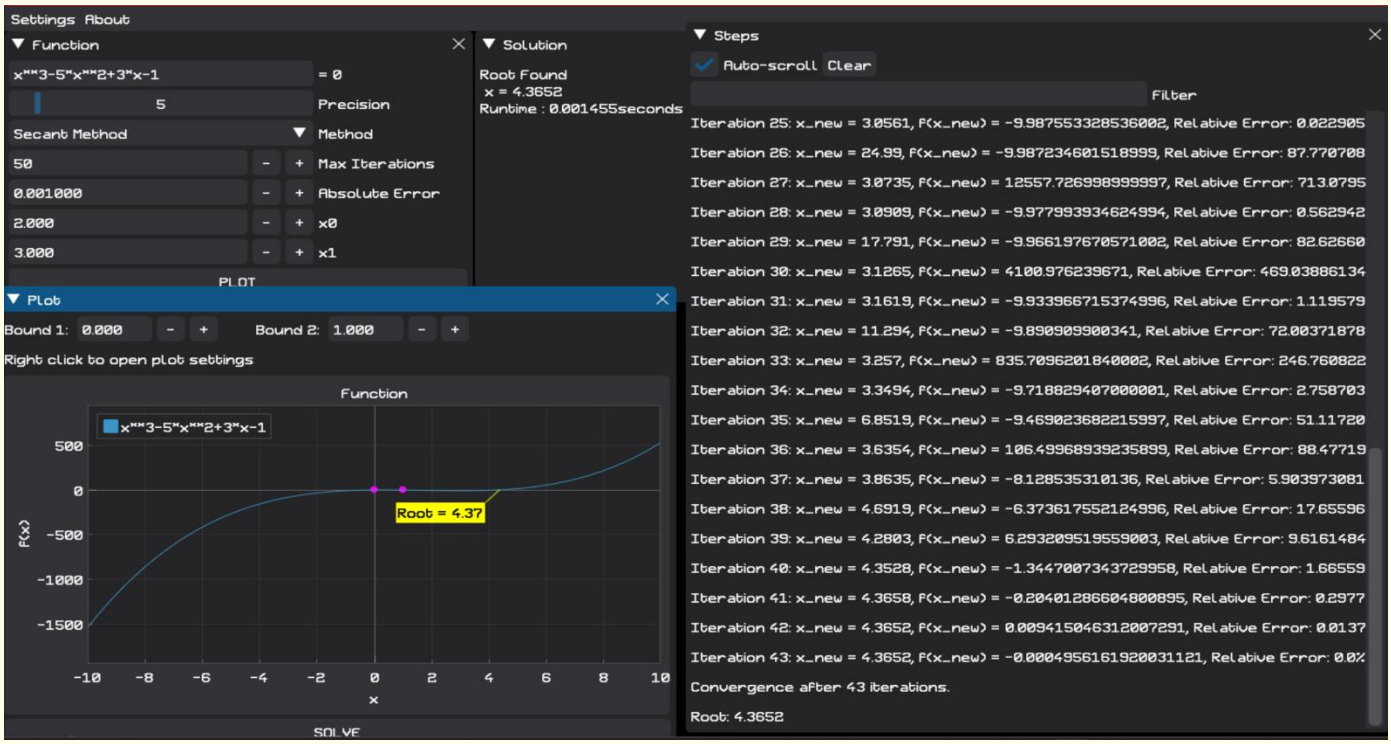
Yields a root of value 4.3652 after 29 iterations.



Secant

$x_0 = 2, \quad x_1 = 3, \quad \varepsilon_{tol} = 10^{-3}$

Yields a root of value 4.3652 after 43 iterations.



Comparison

Observing the outputs of each method the following table is constructed.

Method	No. of Iterations	Run Time	Root	Initial Guesses
Bisection	18	0.000233	4.3652	$x_l = -3.567, \quad x_u = 5.839$
False Position	10	0.000273	4.3652	$x_l = -2.020, \quad x_u = 5.110$
Fixed Point	24	0.000268	4.3651	$x_0 = 4,$ $g(x) = \sqrt[3]{5x^2 - 3x + 1}$
Newton Raphson	29	0.001779	4.3652	$x_0 = 0$
Secant	43	0.001455	4.3652	$x_0 = 2, \quad x_1 = 3$

Like the previous example, the outputs depend mainly on the initial guesses and the method used, for instance *Bisection, False Position and Fixed Point* have relatively close time rates with variations in the number of iterations required for the convergence.

Newton Raphson and Secant methods both took longer for this function, however they still reached exact calculations given the initial values (*not very good ones*).

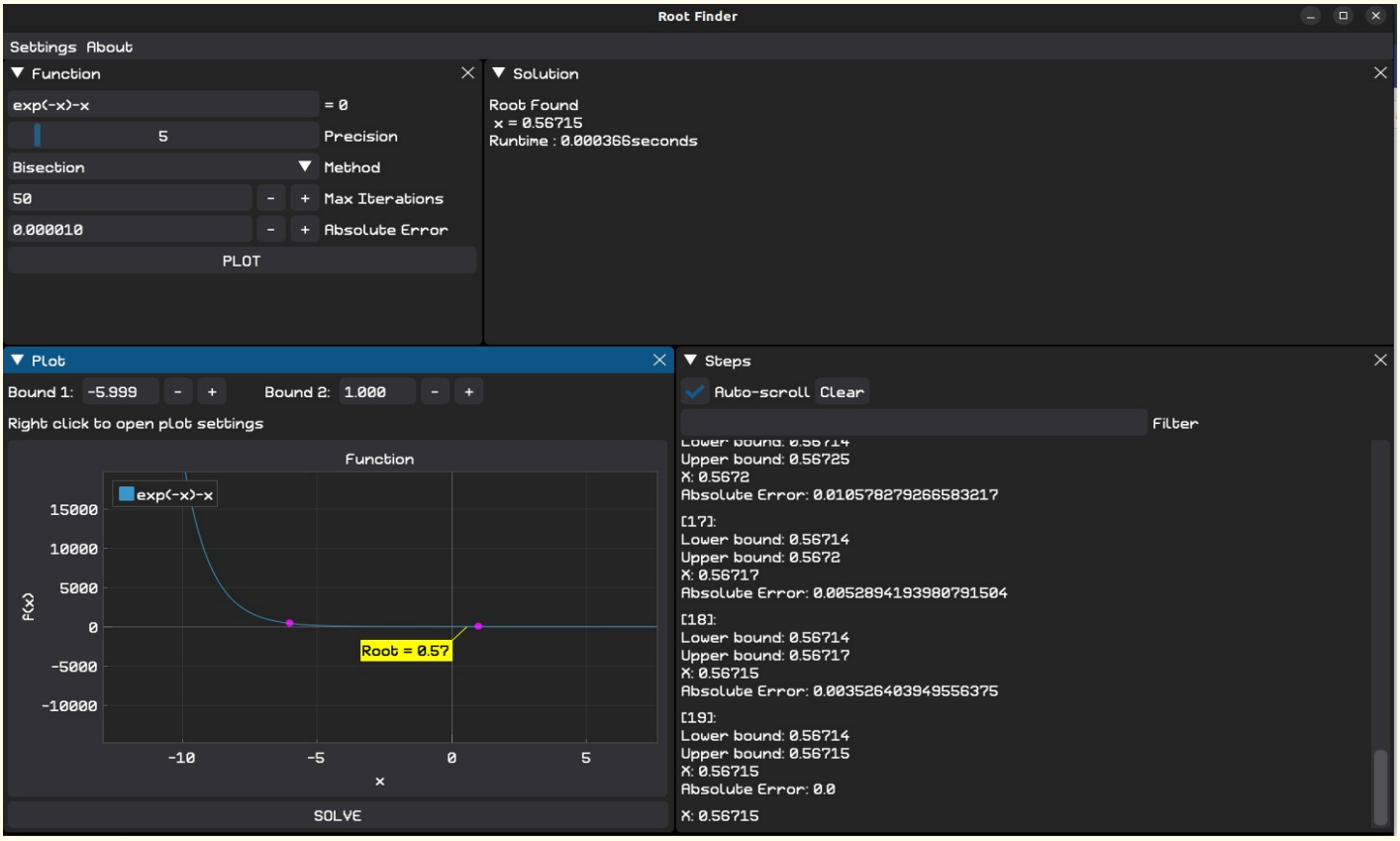
Case 3

$$f(x) = e^{-x} - x$$

Bisection

$x_l = -5.999, \quad x_u = 1$

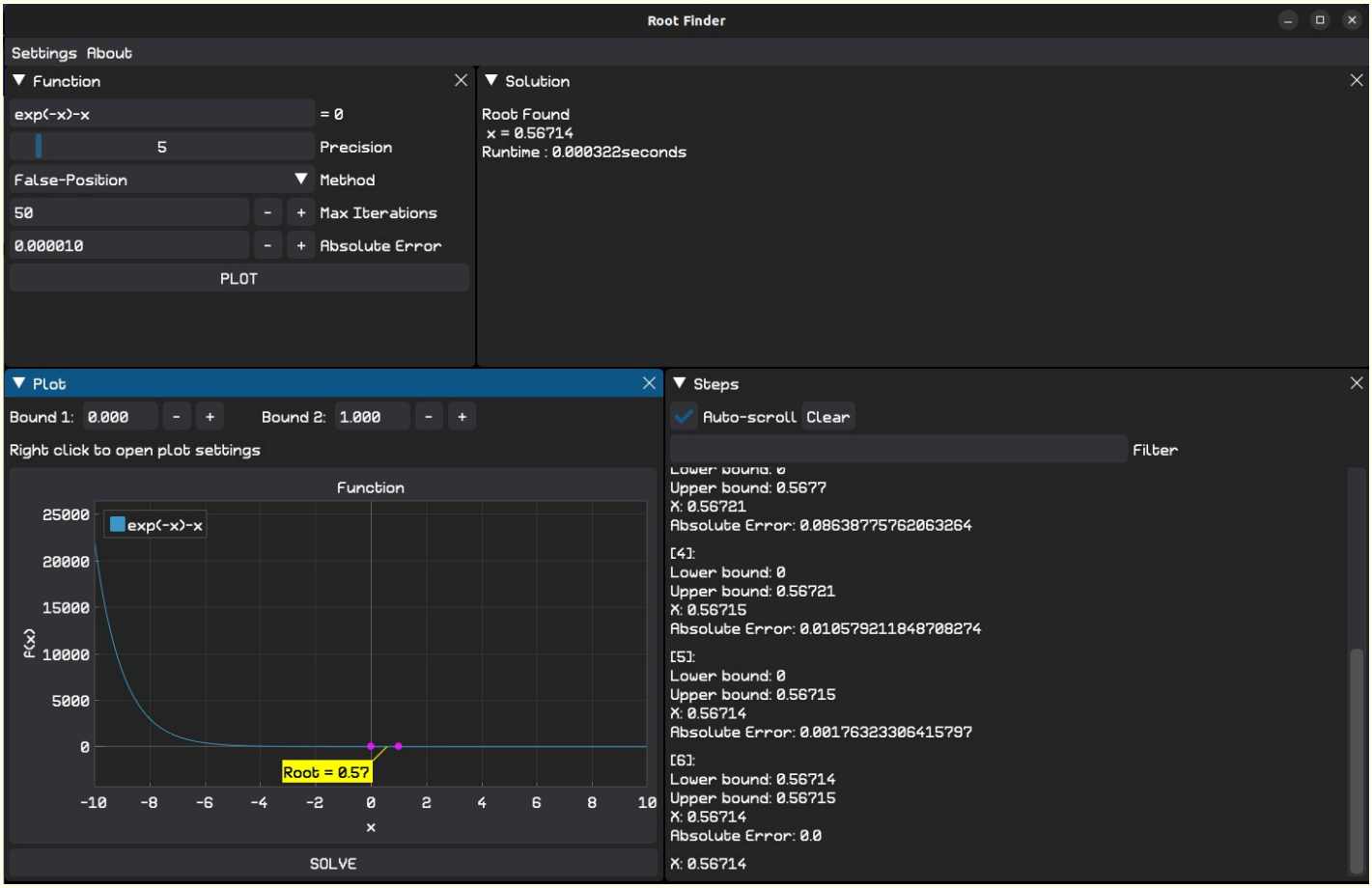
Yields a root of value 0.56715 after 19 iterations.



False Position

$x_l = 0, \quad x_u = 1.000$

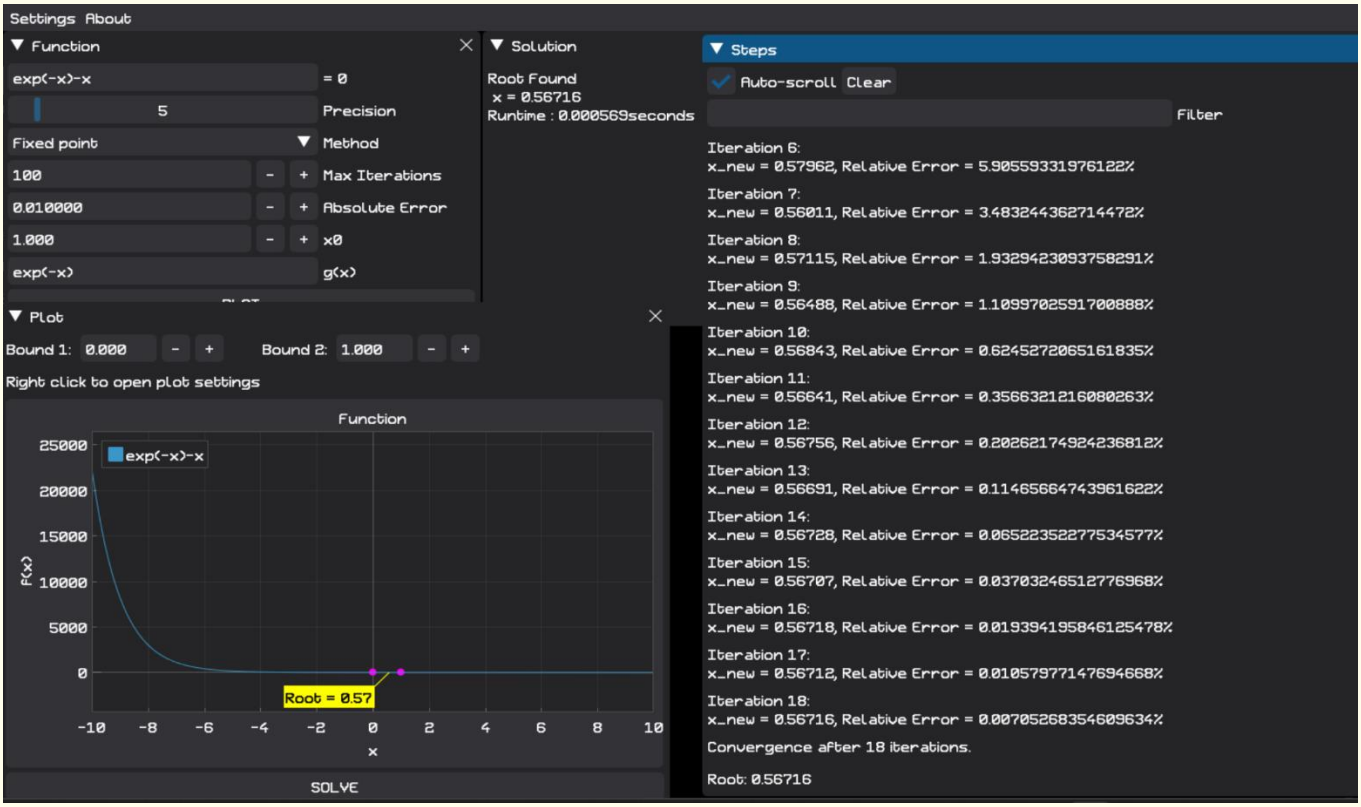
Yields a root of value 0.56714 after 6 iterations.



Fixed Point

$x_0 = 1.000, \quad \varepsilon_{tol} = 10^{-2}, \quad g(x) = e^{-x}$

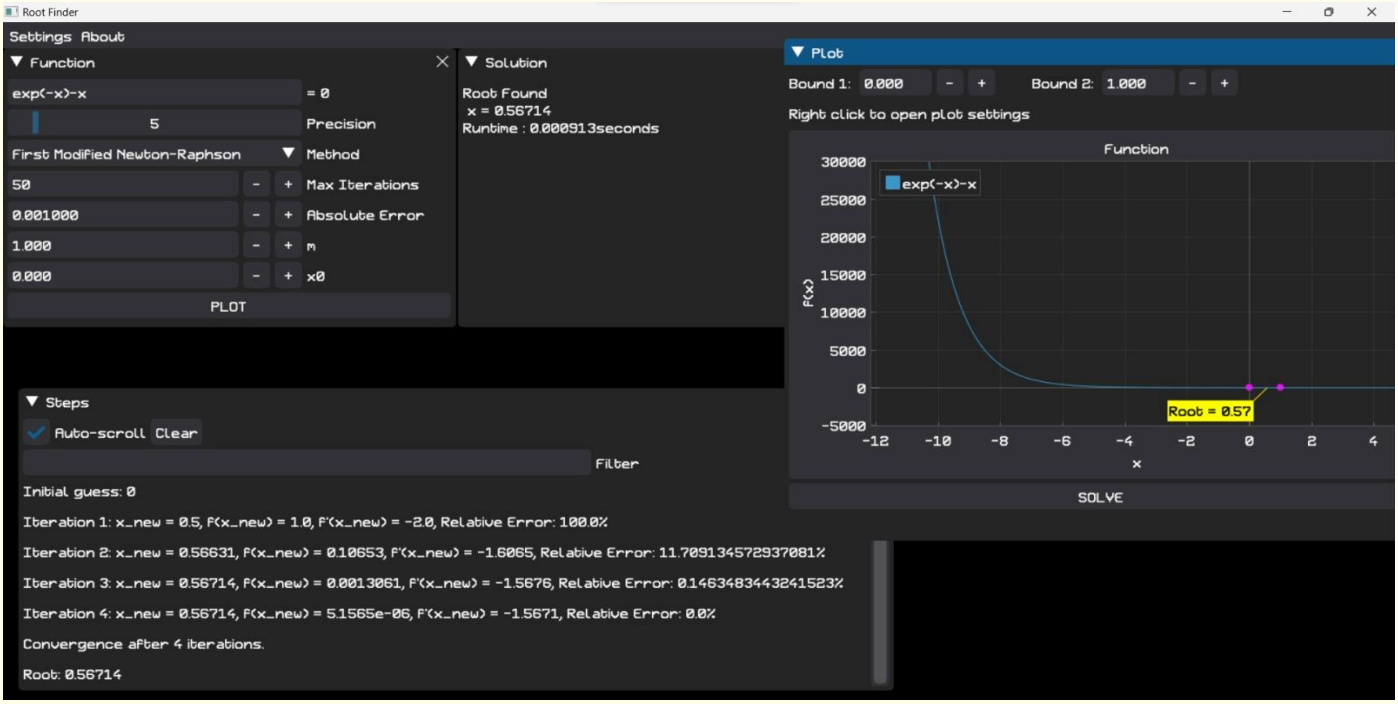
Yields a root of value 0.56716 after 18 iterations.



First Modified Newton

$x_0 = 0, \quad \varepsilon_{tol} = 10^{-3}, \quad m = 1$

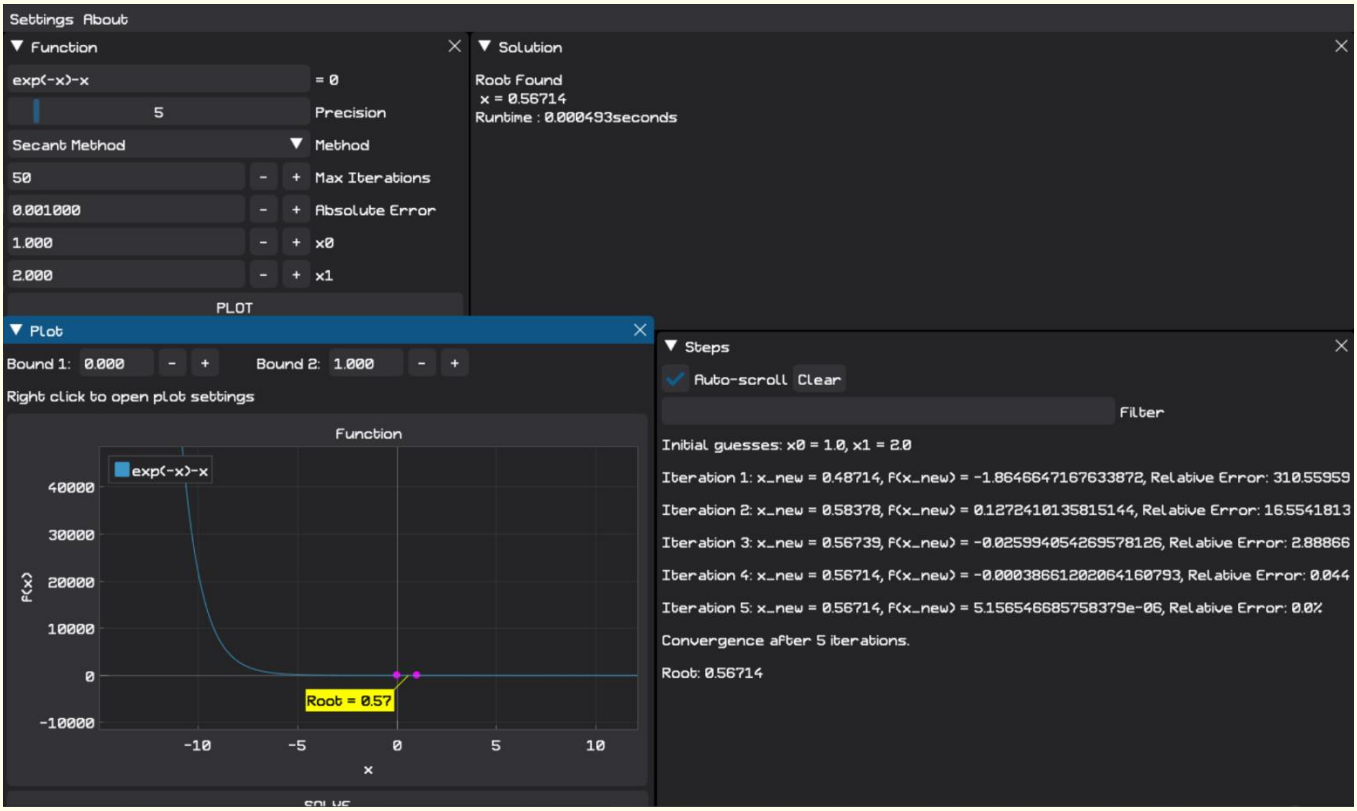
Yields a root of value 0.56714 after 4 iterations.



Secant

$x_0 = 1.000, \quad x_1 = 2.000, \quad \varepsilon_{tol} = 10^{-3}$

Yields a root of value 0.56714 after 5 iterations.



Comparison

Observing the outputs of each method the following table is constructed.

Method	No. of Iterations	Run Time	Root	Initial Guesses
Bisection	19	0.000366	0.56715	$x_l = -5.999, \quad x_u = 1$
False Position	6	0.000322	0.56714	$x_l = 0, \quad x_u = 1.000$
Fixed Point	18	0.000569	0.56716	$x_0 = 1.000, \quad g(x) = e^{-x}$
Newton Raphson	4	0.000913	0.56714	$x_0 = 0$
Secant	5	0.000493	0.56714	$x_0 = 1.000, \quad x_1 = 2.000$

Similarly, the factors are dependent. However, it is observed that *Secant* method is much faster than the *Newton Raphson* with same accuracy given 5 significant figures.

Regarding the other methods their values vary but both *Bisection and False Position* have relatively the same run time, apart from the number of iterations for each, while fixed point is a bit slower, this can be modified by choosing another initial guess or changing $g(x)$.

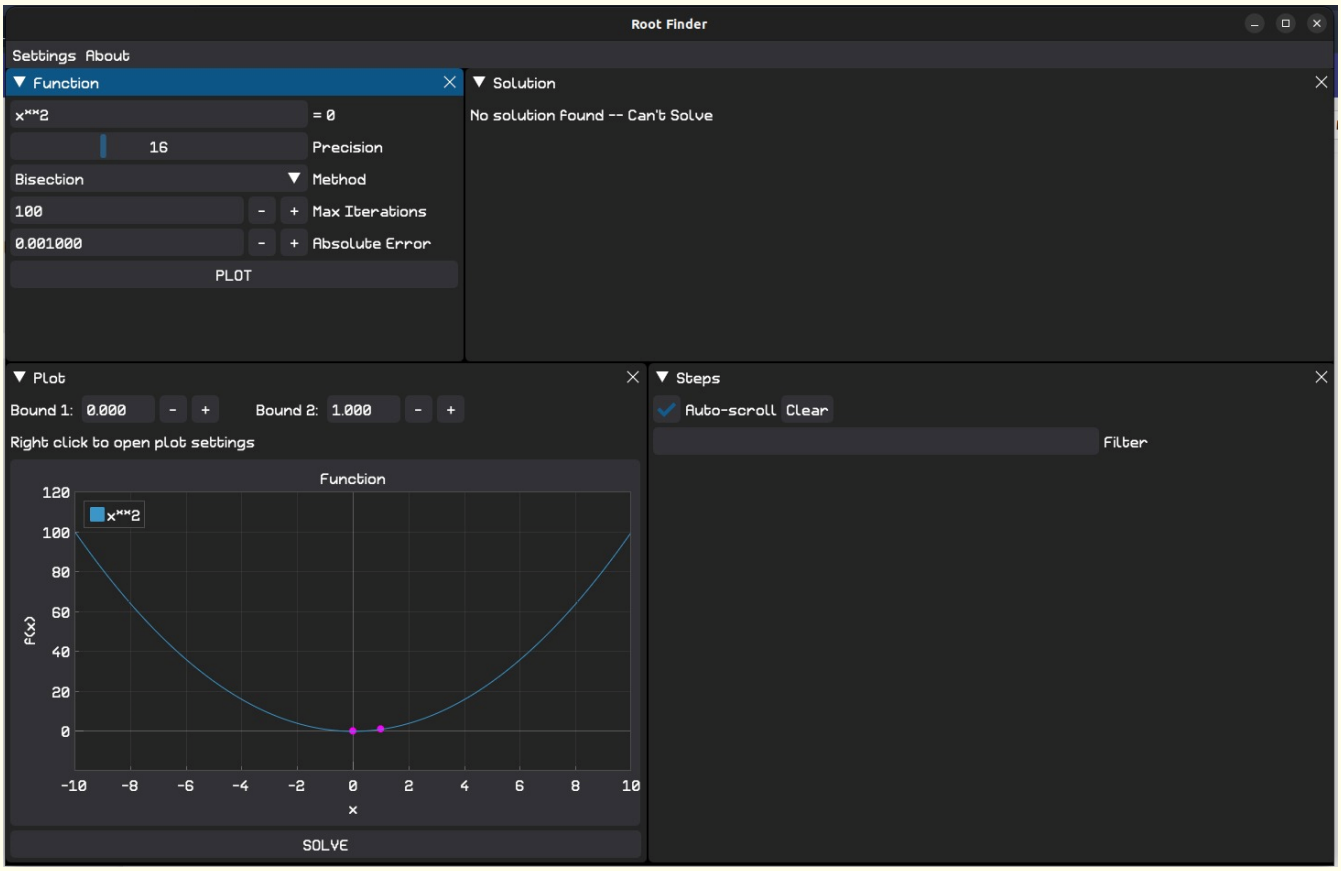
Case 4

$$f(x) = x^2$$

Bisection

$$x_l = ANY, \quad x_u = ANY$$

Yields No Solution since the quadratic function x^2, it is important to note that the function is always non-negative (or positive) except at $x = 0$. The bisection method typically requires the function values at the endpoints of the initial interval to have opposite sign. However, the function doesn't have any negative values, thus selecting an interval with two different signs will be an impossible approach. Accordingly, the bisection method falls for such functions.



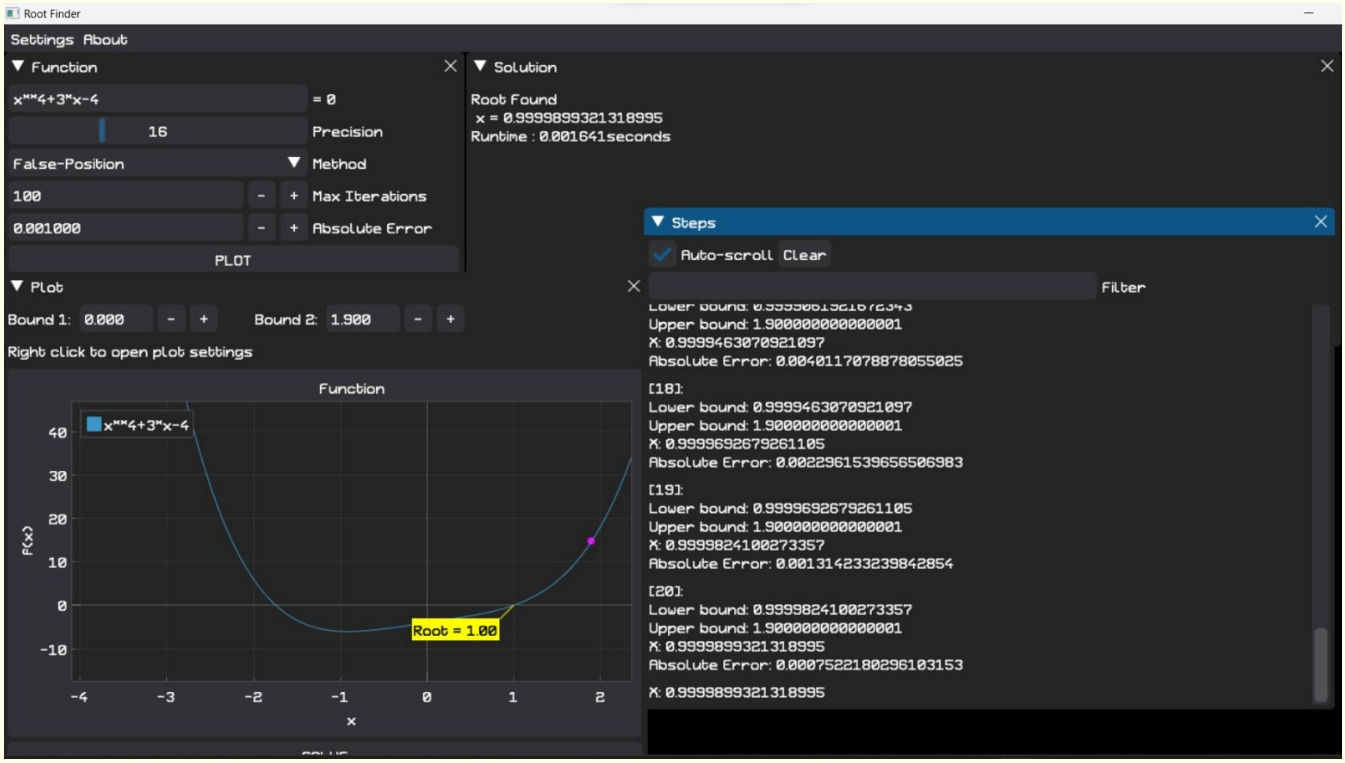
Case 5

$$f(x) = x^4 + 3x - 4$$

False Position

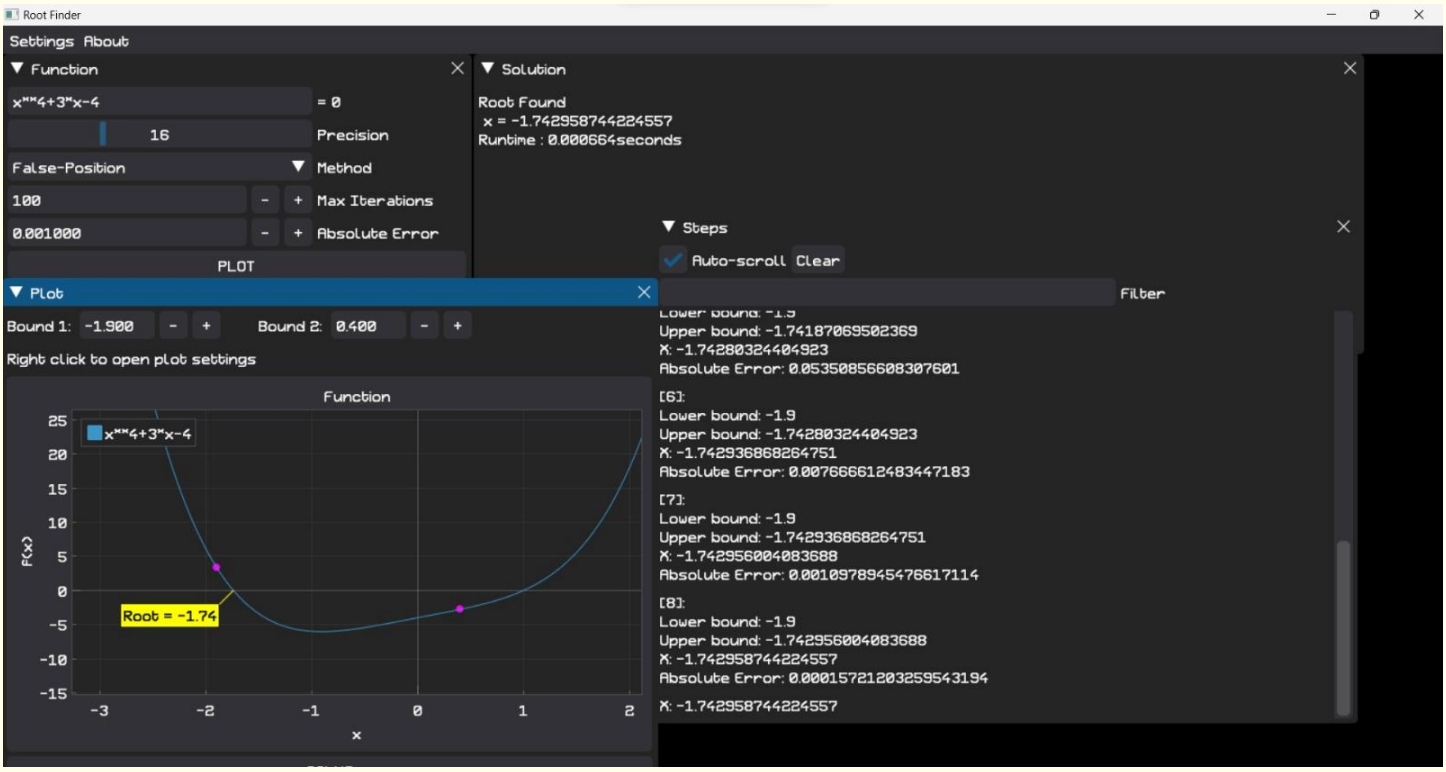
$$x_l = 0, \quad x_u = 1.900, \quad \varepsilon_{tol} = 10^{-3}$$

Yields a root of value 0.9999899321318995 using 16 significant figures after 20 iterations.



$x_l = -1.90, \quad x_u = 0.40, \quad \varepsilon_{tol} = 10^{-3}$

Yields a root of value -1.742958744224557 using 16 significant figures after 8 iterations.



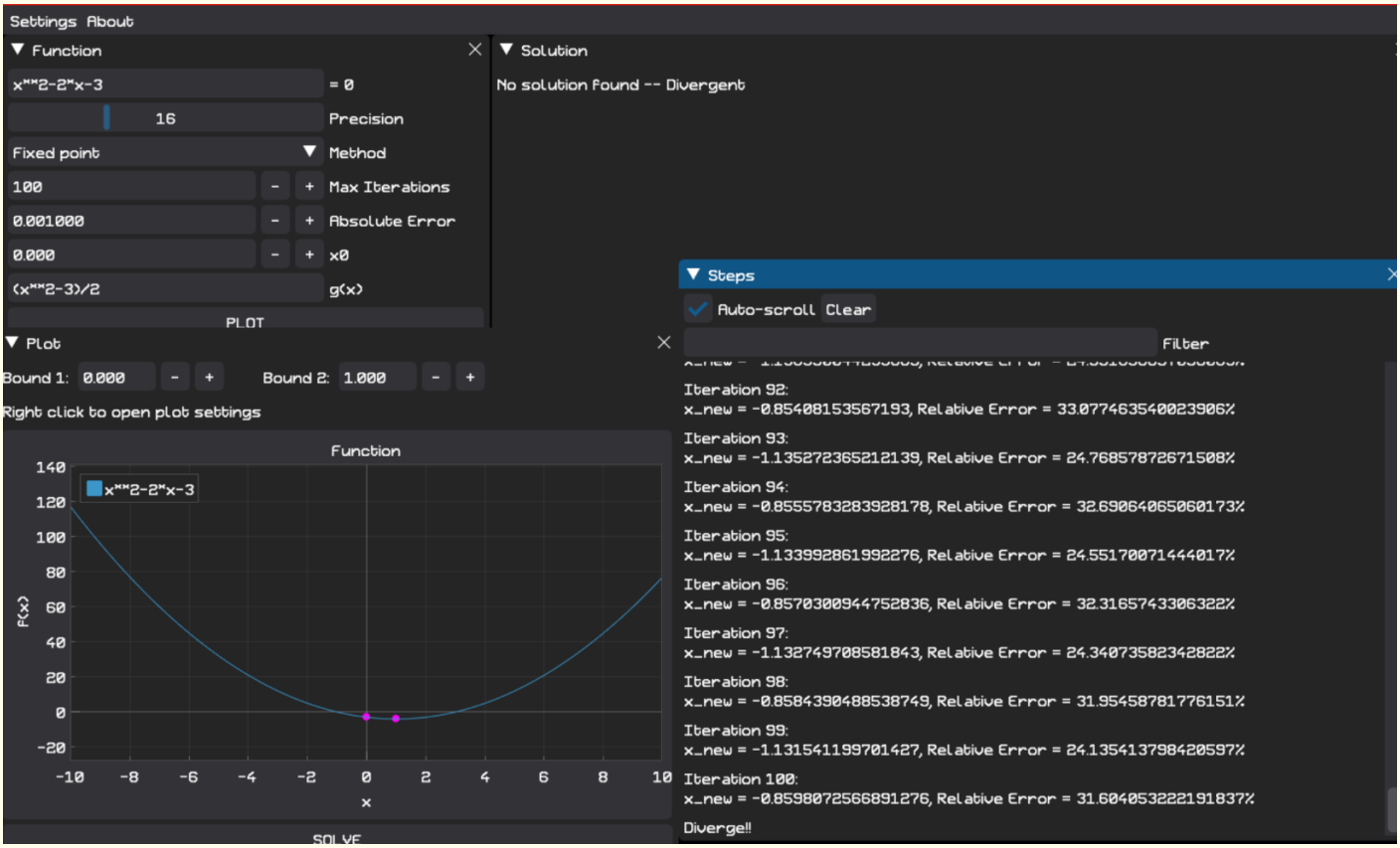
Case 6

$$g(x) = \frac{(x^2 - 3)}{2} = x$$

Fixed Point

$x_0 = 0, \quad f(x) = x^2 - 2x - 3 = 0$

Yields a root of value -0.859807 with 16 significant figures, though it Diverges after 100 iterations.



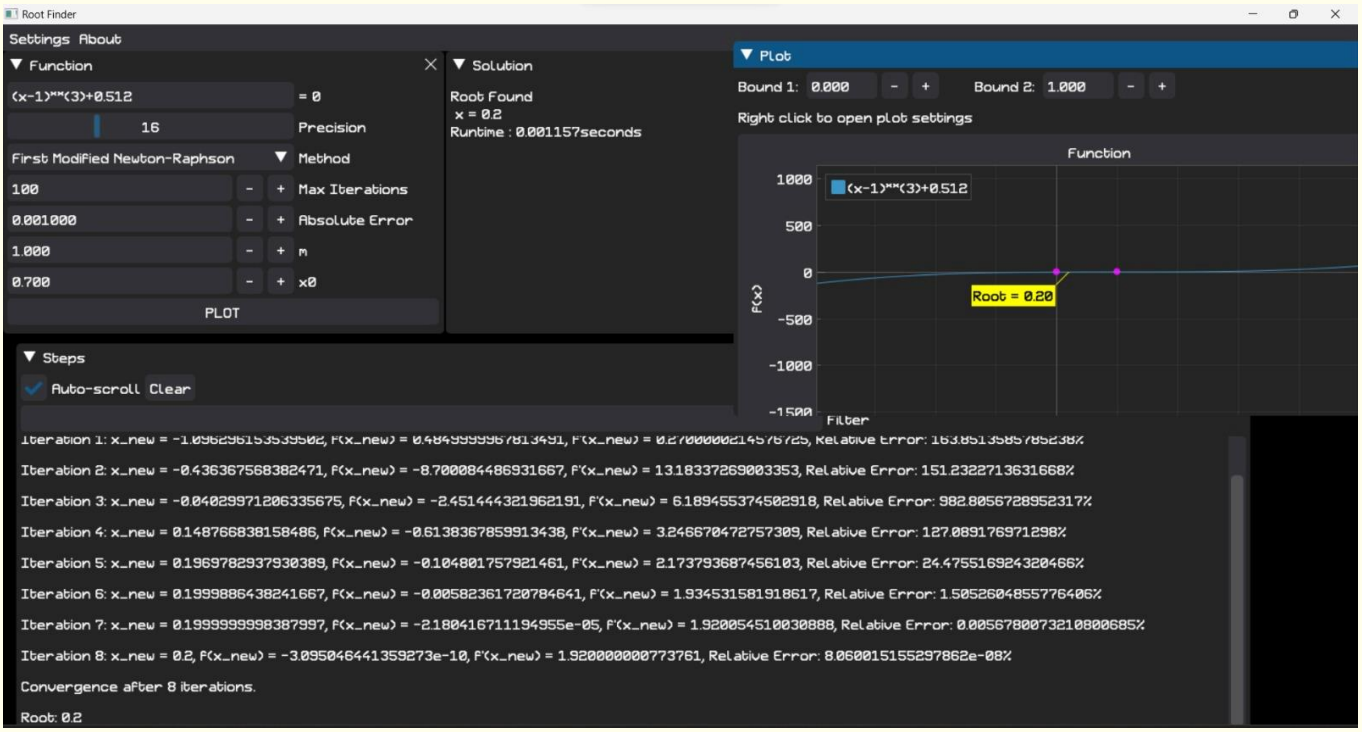
Case 7

$$f(x) = (x-1)^3 + 0.512$$

Newton Raphson

$x_0 = 0.7, \quad \varepsilon_{tol} = 10^{-3}, \quad m = 1$

Yields a root of value 0.2 after 8 iterations.



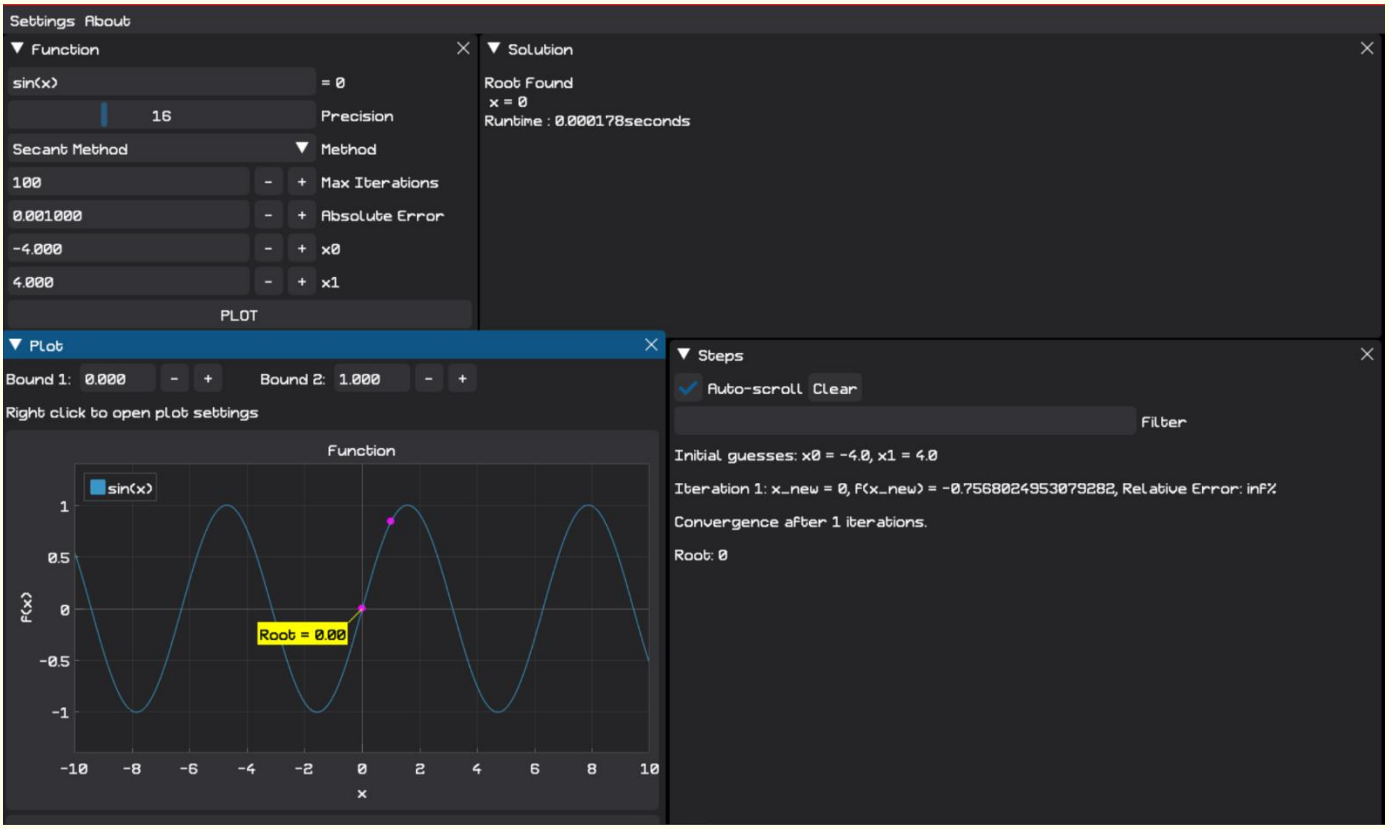
Case 8

$$f(x) = \sin x$$

Secant

$x_0 = -4.0, \quad x_1 = 4.0$

Yields a root of value 0 after 1 iteration.


Settings About
Function
sin(x)
= 0
16
Precision
Secant Method
Method
100
Max Iterations
0.001000
Absolute Error
-4.000
x0
4.000
x1
PLOT
Solution
Root Found
x = 0
Runtime : 0.000178seconds
Plot
Bound 1: 0.000
Bound 2: 1.000
Right click to open plot settings
Function
sin(x)
f(x)
x
Root = 0.00
Steps
Auto-scroll
Clear
Filter
Initial guesses: x0 = -4.0, x1 = 4.0
Iteration 1: x_new = 0, f(x_new) = -0.7568024953079282, Relative Error: inf%
Convergence after 1 iterations.
Root: 0