Non Linear Equation Solver Using Python

Name – Tsering Wangchu
Roll No- 244161007
Course-Programming with Python
Course Instructor- Neeraj Kumar Sharma

√5

Introduction

Have you ever wondered how Python finds the root of any function behind the scenes? Finding the root is a fundamental problem in mathematics and computer science. It involves finding the value of x where a given function equals zero, commonly known as the root of the equation.

$$x^2 - 5 = 0$$

The root of this simple non linear equation is √5.
So how does python finds it ?

In build Function in python and implementation behind it

index.py > ...

```python
import math
#finding square root of any number
number=5
root=math.sqrt(number)
```

It simply solving the nonlinear equation

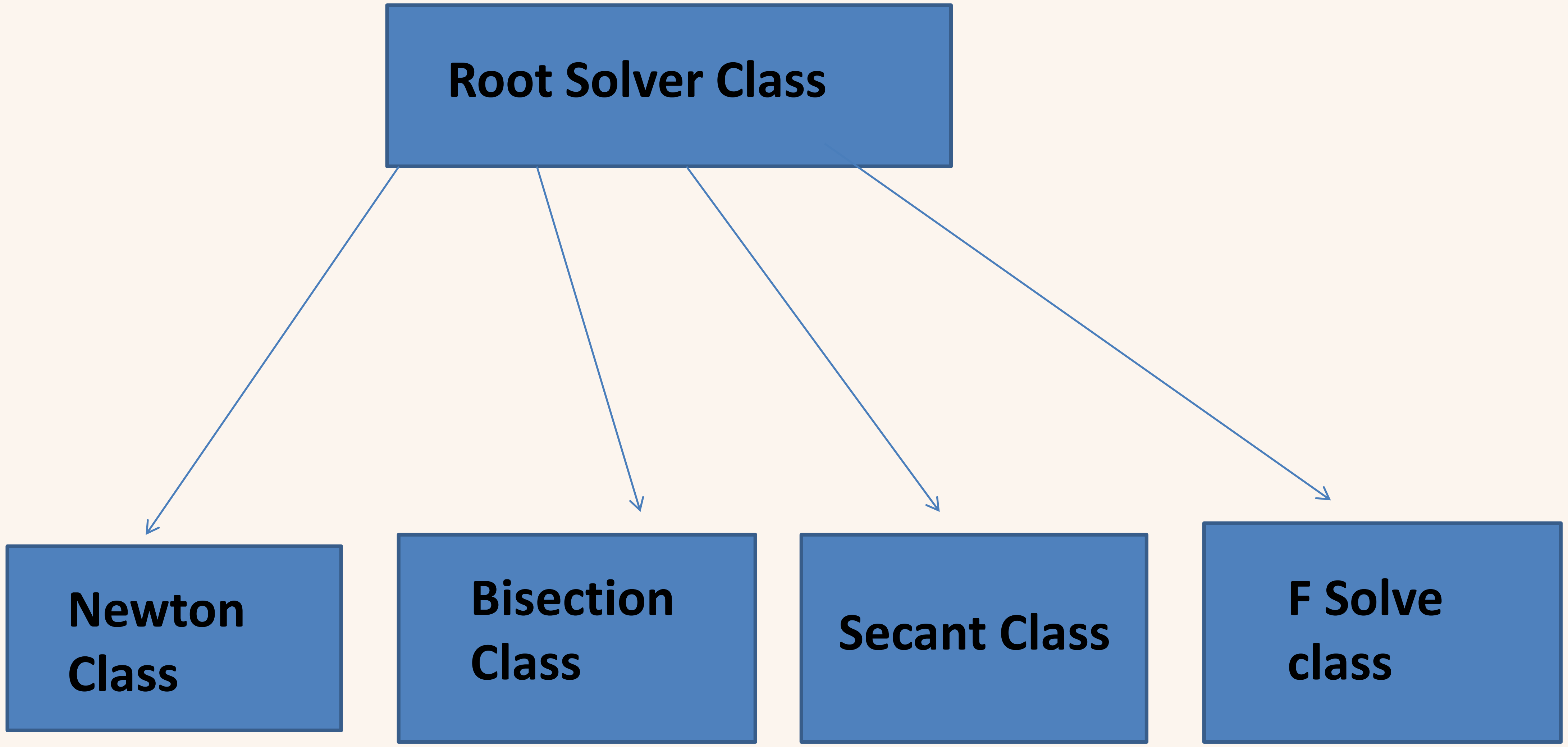
$$x^2 - 5 = 0$$

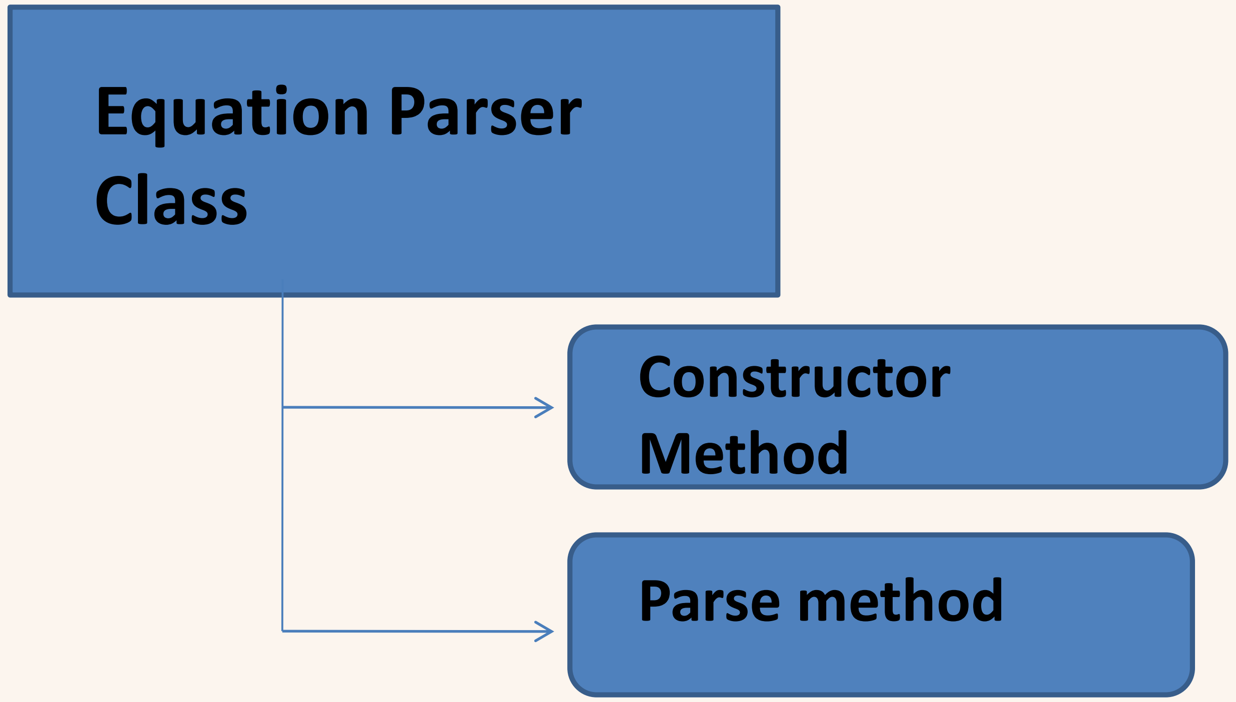
Newton's Method: An iterative technique that approximates the root by using both the function and its derivative. With just an initial guess, it quickly converges to the root.

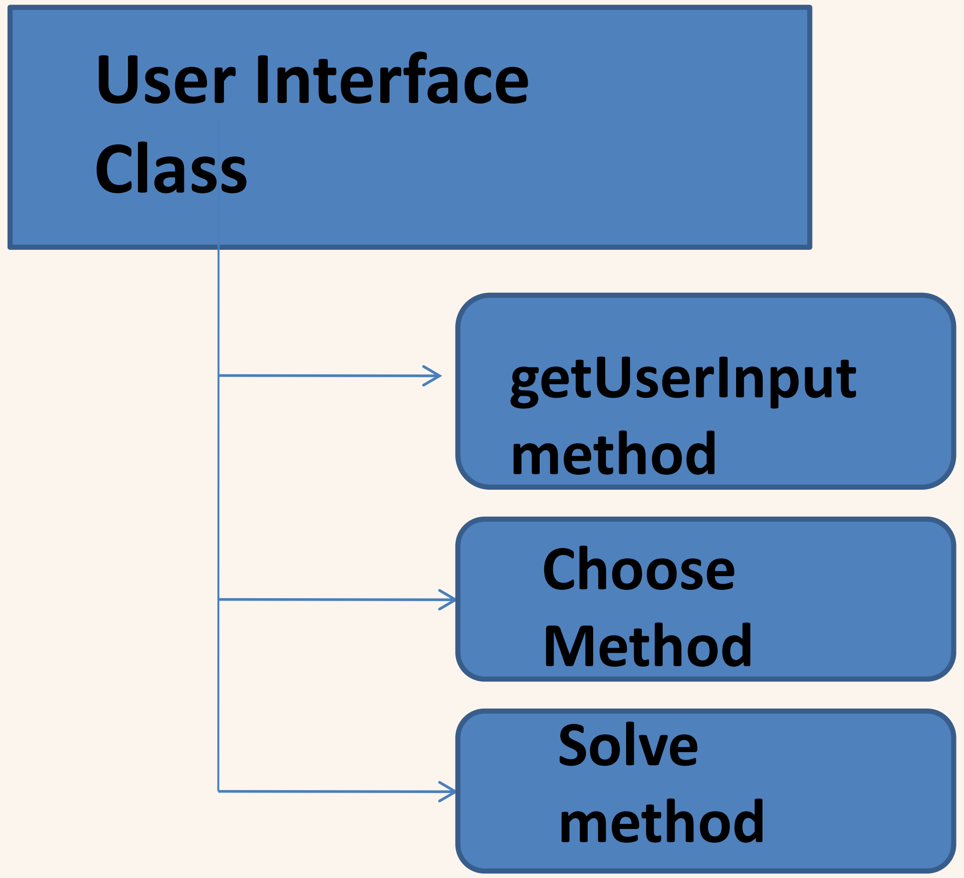
Bisection Method: This method works by dividing an interval in half repeatedly, narrowing the range until the root is pinpointed. It is guaranteed to find a root as long as the function changes signs at the endpoints.

Secant Method: A derivative-free approach that approximates the root using two initial guesses, iteratively improving the estimate of the root without requiring the actual derivative of the function.

Workflow of my Project

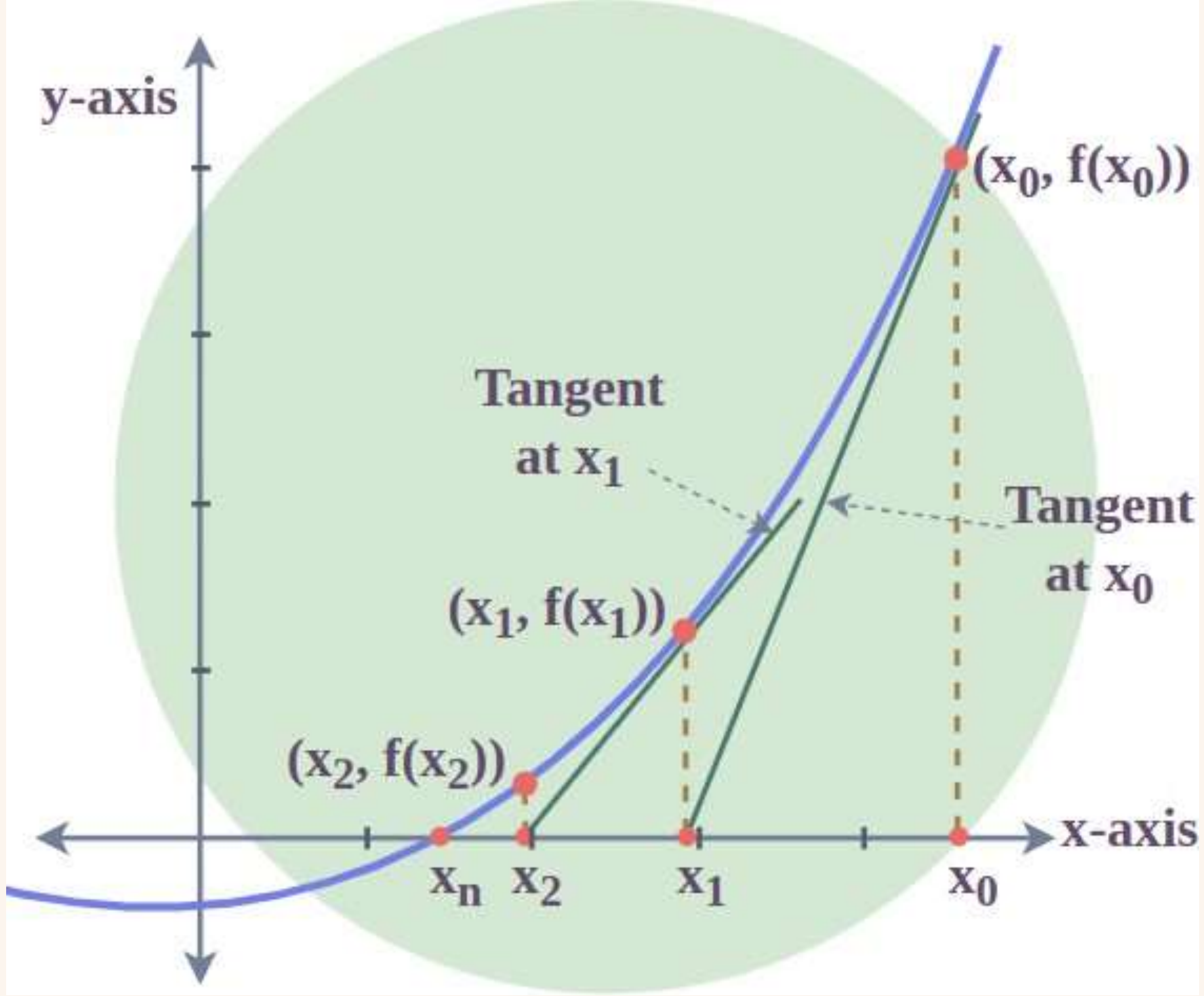






Root Solver is the parent class and other class (Newton ,Bisection , Secant) inherits Root Solver class.

Technique 1: Newton’s Method



$$Xi + 1 = Xi + \frac{f(Xi)}{f'(Xi)}$$

Pseudo Code

Input: A differentiable function f

Initial guess: c tolerance :tol

A limit N for maximum number of iteration.

Output Approximate solution c of $f(x)=0$ satisfying $|f(c)| \leq$ tol.

IT=0

while($|f(c)|$>**tol**) **and** (IT≤N) **do**

$c=c-f(c)/f'(c)$

IT=IT+1

Implementation of Newton Method

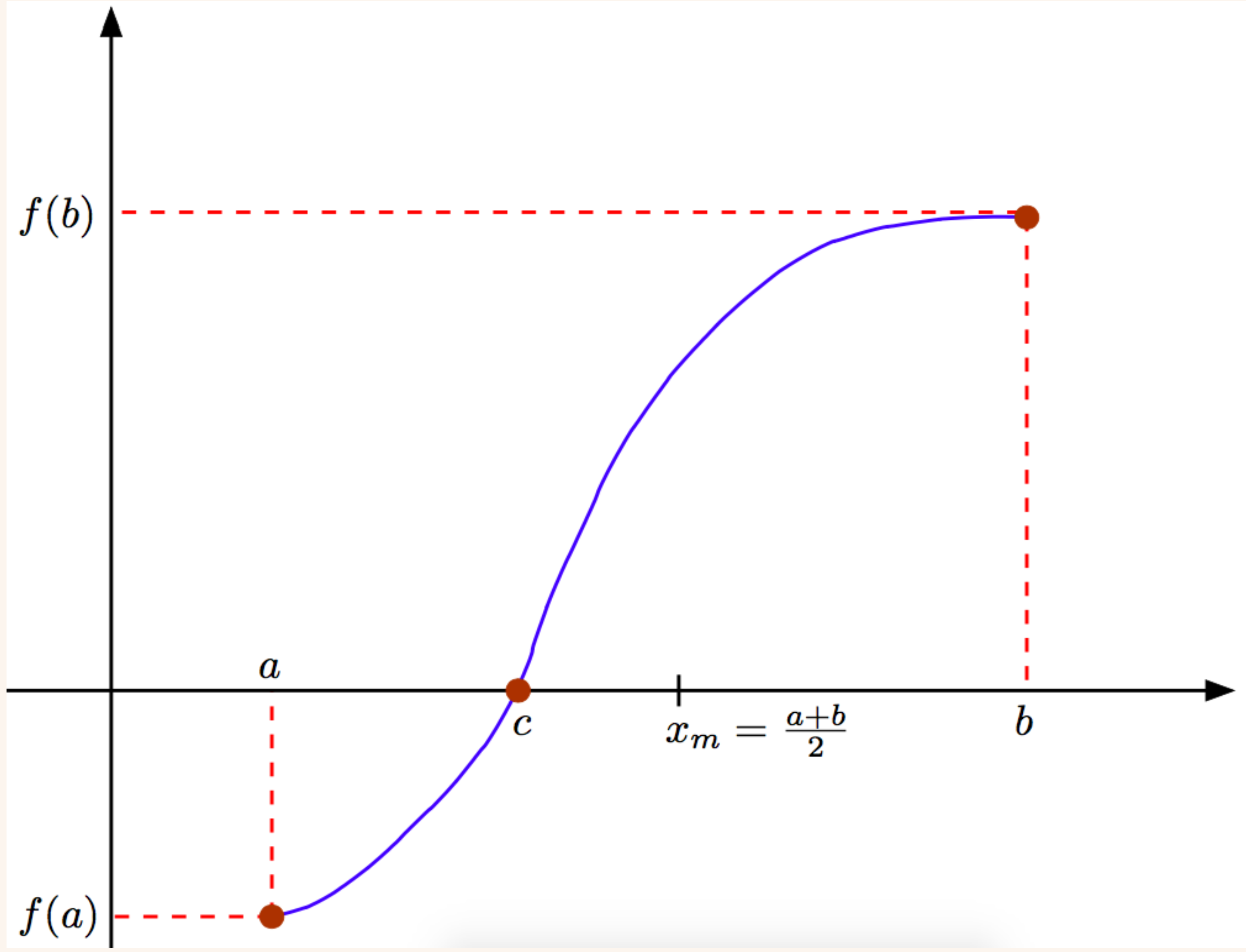
```python
class NewtonMethod(RootSolver):

    def __init__(self, func, derivative, x0, tol=1e-6, max_iter=100):
        super().__init__(func)
        self.derivative = derivative
        self.x0 = x0
        self.tol = tol
        self.max_iter = max_iter

    def solve(self):
        x = self.x0
        for _ in range(self.max_iter):
            fx = self.func(x)
            fx_prime = self.derivative(x)
            if abs(fx) < self.tol:
                return x
            x = x - fx / fx_prime
        return x
```

Constructor taking the arguments ,a function, its derivative, tolerance, max_iteration

Calling the constructor of the parent class Root Solver

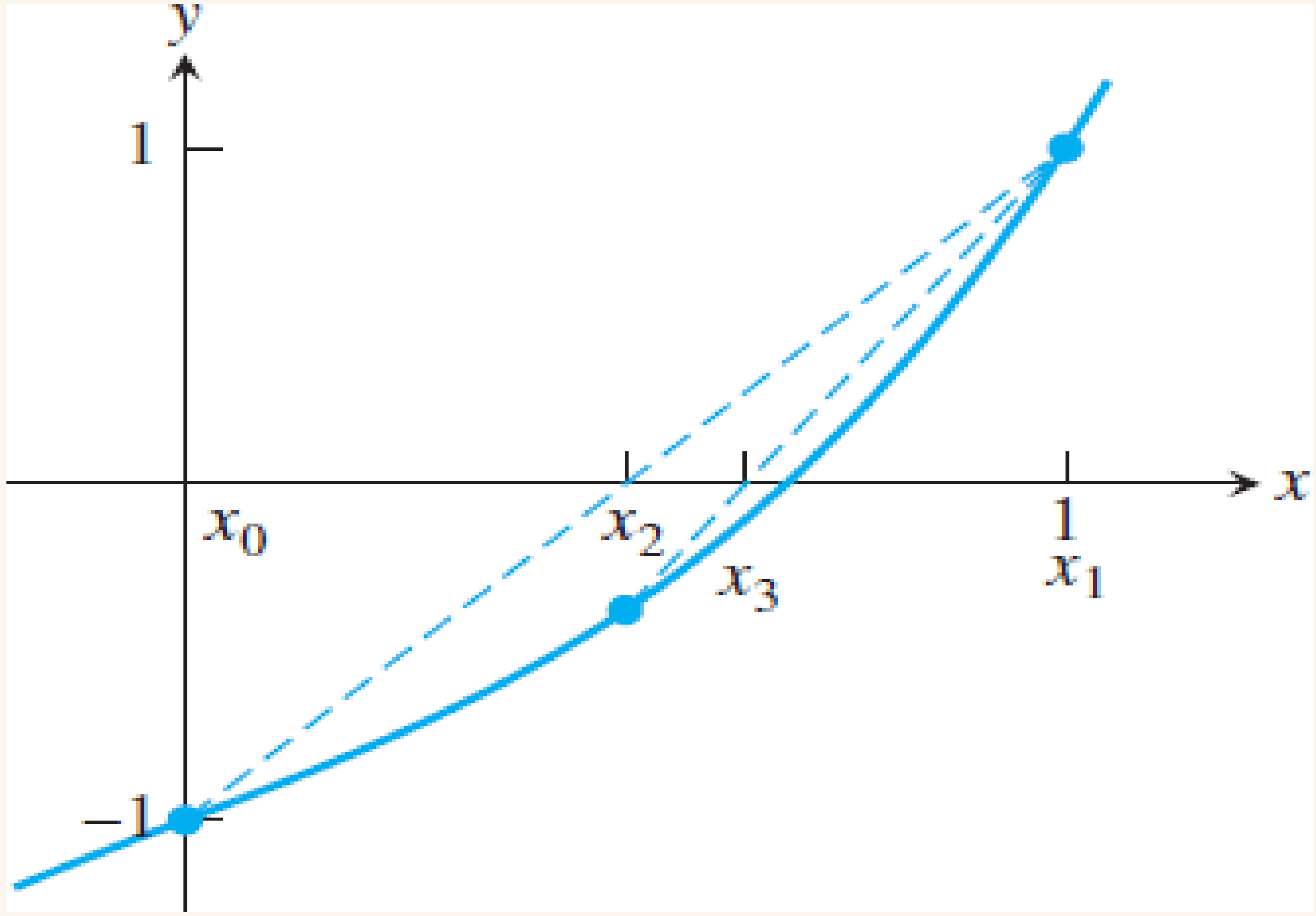
Newton’s Iteration to find the next point

Technique 2: Bisection Method



$$xm := \frac{am + bm}{2}$$

Technique 3: Secant Method



$$Xn+1 = Xn - \frac{f(Xn)(Xn - Xn-1)}{f(Xn) - f(Xn-1)}$$

Implementation

```python
class BisectionMethod(RootSolver):
    def __init__(self, func, a, b, tol=1e-6, max_iter=100):

        super().__init__(func)
        self.a = a
        self.b = b
        self.tol = tol
        self.max_iter = max_iter

    def solve(self):

        if self.func(self.a) * self.func(self.b) >= 0:
            # Check if the function changes sign
            raise ValueError("Function has the same signs at the endpoints.")

        for _ in range(self.max_iter):
            c = (self.a + self.b) / 2  # Midpoint
            if abs(self.func(c)) < self.tol:  # Check if the function value at c is close to
                return c
            elif self.func(c) * self.func(self.a) < 0:  # Root is between a and c
                self.b = c
            else:  # Root is between c and b
                self.a = c
        return (self.a + self.b) / 2  # Return the midpoint after max iterations
```

```python
class SecantMethod(RootSolver):
    def __init__(self, func, x0, x1, tol=1e-6, max_iter=100):
        super().__init__(func)
        self.x0 = x0
        self.x1 = x1
        self.tol = tol
        self.max_iter = max_iter

    def solve(self):

        for _ in range(self.max_iter):
            fx0 = self.func(self.x0)  # Evaluate the function at x0
            fx1 = self.func(self.x1)  # Evaluate the function at x1
            if abs(fx1) < self.tol:  # If the value of the function is small enough
                return self.x1
            # Secant method iteration
            x2 = self.x1 - fx1 * (self.x1 - self.x0) / (fx1 - fx0)
            self.x0, self.x1 = self.x1, x2
        return self.x1  # Return the last value if max iterations are reached
```

Equation parser and User Interface Class functionality

Equation Parser class

- ➢SymPy: SymPy is a Python library for symbolic mathematics. It allows for algebraic manipulation, differentiation, integration, equation solving, and other symbolic calculations
- ➢Converts a string equation into a callable function for numerical evaluation
- ➢Defines x as a symbolic variable with sp.symbols('x').
- ➢Uses sp.sympify to turn equation_str into a symbolic expression.
- ➢Uses sp.lambdify to convert the expression into a callable function with NumPy support.

User Interface Class

- ➢Get user method displays a welcome message and Prompts the user to enter an equation as a string (using 'x' as the variable).
- ➢Asks the user to select a solving method from four options:
  - 1: Newton’s Method
  - 2: Bisection Method
  - 3: Secant Method
  - 4: SciPy’s fsolve Method
- ➢Returns the user's method choice.

Project Output

```
TERMINAL
Welcome to the Nonlinear Equation Solver!
Enter the equation to solve (use 'x' as the variable): x^2 +3*x -5
Choose a method (1 = Newton's Method, 2 = Bisection, 3 = Secant, 4 = fsolve): 1
Enter the initial guess: 3
Enter the derivative of the function: 2*x +3
Root of the equation: 1.1925824049286264
PS C:\Users\user\Desktop\NonLinearEquationsUsingPython> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c:/Users/user/Desktop/NonLinearEquationsUsingPython/project.py
Welcome to the Nonlinear Equation Solver!
Enter the equation to solve (use 'x' as the variable): x^2+3*x-5
Choose a method (1 = Newton's Method, 2 = Bisection, 3 = Secant, 4 = fsolve): 2
Enter the left interval a: 1
Enter the right interval b: 2
Root of the equation: 1.192582368850708
PS C:\Users\user\Desktop\NonLinearEquationsUsingPython> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c:/Users/user/Desktop/NonLinearEquationsUsingPython/project.py
Welcome to the Nonlinear Equation Solver!
Enter the equation to solve (use 'x' as the variable): x^2+3*x-5
Choose a method (1 = Newton's Method, 2 = Bisection, 3 = Secant, 4 = fsolve): 3
Enter the first initial guess: 2
Enter the second initial guess: 4
Root of the equation: 1.192582442152445
PS C:\Users\user\Desktop\NonLinearEquationsUsingPython> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c:/Users/user/Desktop/NonLinearEquationsUsingPython/project.py
Welcome to the Nonlinear Equation Solver!
Enter the equation to solve (use 'x' as the variable): x^2+3*x-5
Choose a method (1 = Newton's Method, 2 = Bisection, 3 = Secant, 4 = fsolve): 4
Enter the initial guess: 2
```

x^2 +3*x-5=0

Options to choose a method

Choose any random Number as initial Guess

Provide the derivate of the equation (applicable only in Newton method)

Choose a and b such that f(a)*f(b) <0 (only in bisection method)

Solved using sciPy python library

Conclusion

Mathematical Derivation of Root-Finding Techniques: Implementing Newton-Raphson, Bisection, Secant, and fsolve methods highlighted the different approaches and applications for each method, illustrating why certain techniques are better suited for specific types of nonlinear equations.

Object-Oriented Programming in Python: By structuring our solution with classes, we created a flexible and reusable codebase, making it easy to add, modify, or switch between algorithms.

Symbolic Computation: Using SymPy allowed for parsing symbolic equations into callable functions, demonstrating the versatility of Python's scientific libraries in solving complex computational problems.