****UNIVERSITY OF BOHOL

Tagbilaran City, Bohol, Philippines

COLLEGE OF ENGINEERING, TECHNOLOGIES,

ARCHITECTURE AND FINE ARTS

COMPUTER ENGINEERING NUMERICAL METHODS

CPEP 221

**NUMERICAL PLOTTING USING PYTHON**

**Submitted by:**

BALMOCINA, NIÑA JEAN G.

**Introduction**

This project is a standalone Python-based application designed to support the exploration and application of classical numerical methods for solving nonlinear equations. It allows users to input a mathematical function, visualize its behavior graphically, and apply several root-finding techniques to approximate the roots. The application is built with a focus on usability, educational value, and clarity, making it suitable for students, instructors, and self-learners in numerical analysis.

The application supports a range of root-finding algorithms, displays both tabular and graphical outputs for iterative methods, and provides an intuitive interface that makes the learning of algorithmic behavior more interactive and accessible.

**Scope and Limitations**

**Scope**

**The main functionalities covered by this application include:**

* Function Input: Accepts mathematical expressions involving a single variable (e.g., polynomials, trigonometric functions, exponentials).
* Visualization: Plots the function over a user-defined range, highlighting root approximations and iteration steps.
* Numerical Methods Implemented:
  + Incremental Search
  + Bisection Method
  + False Position (Regula Falsi) Method
  + Newton-Raphson Method
  + Secant Method
* Tabular Output: Presents each step of the root-finding iterations with key values such as current approximation, function value, and error.
* Dark Mode: Offers theme customization for better visual comfort.
* Cross-Platform GUI: Built using Tkinter for compatibility on Windows, macOS, and Linux.

**Limitations**

Despite its utility, the application has certain limitations:

* Only supports single-variable equations.
* Convergence depends on good initial guesses for methods like Newton-Raphson and Secant.
* Derivatives must be computable and valid for Newton-Raphson.
* Static graph updates—no animated iteration plotting.
* Basic error checking; invalid or complex input may cause failures.
* Not optimized for performance with high-complexity or high-precision needs.

**Objective**

The primary goal of this project is to deepen understanding of both numerical analysis and Python application development. It integrates symbolic computation, numerical algorithms, and interface design into a cohesive tool for learning and experimentation. Through hands-on implementation of multiple algorithms, the project reinforces foundational concepts in mathematics, programming, and user interface interaction.

**Features**

* Graphical visualization with labeled roots
* Custom function input with validation
* Dynamic input fields based on method selection
* Scrollable results table for iterative outputs
* Toggle for light/dark mode
* Clear/reset functions for restarting sessions

**Input and Output Requirements**

Input Parameters

* Function Expression: f(x) = ...
* x\_min, x\_max: Range for plotting and searching
* Tolerance: Desired precision for iterative methods
* Max Iterations: Safety limit to prevent infinite loops
* Step Size: For Incremental Search
* Initial Guesses:
  + One for Newton-Raphson
  + Two for Secant Method

**Output Results**

* Root Approximations: Final values where f(x) approaches 0
* Function Evaluations: f(x) at each step
* Graph: f(x) plot with roots and intervals highlighted
* Iteration Table: Values of x, f(x), errors, and remarks per step

**Key Formulas Used**

**Graphical Method**

Formula:  
*None explicitly*, but relies on detecting sign changes and linear interpolation:

Description:  
 Detects roots by plotting the function and finding where it crosses the x-axis. Uses interpolation to estimate root locations visually.

**Incremental Search**

* **Step Update**
* **Sign Change Check:**

**Bisection Method**

* **Midpoint:**
* **Error:**

**False Position Method**

* **Estimate:**
* **Error:**

**Newton-Raphson**

* **Update Rule:**
* **Error:**

**Secant Method**

* **Update Rule:**
* **Error:**

**System Design**

**File Structure**

* Single Script File: All functionalities are integrated into one Python file named Project.py.
* Self-Contained GUI: Tkinter is used for building the interface without external dependencies.

**Interface Layout**

* Top: Function input field and method selection dropdown
* Middle: Dynamic fields for method-specific parameters
* Right Panel: Matplotlib graph canvas
* Bottom: Buttons for solving, plotting, and clearing
* Results: Scrollable table displaying iteration data

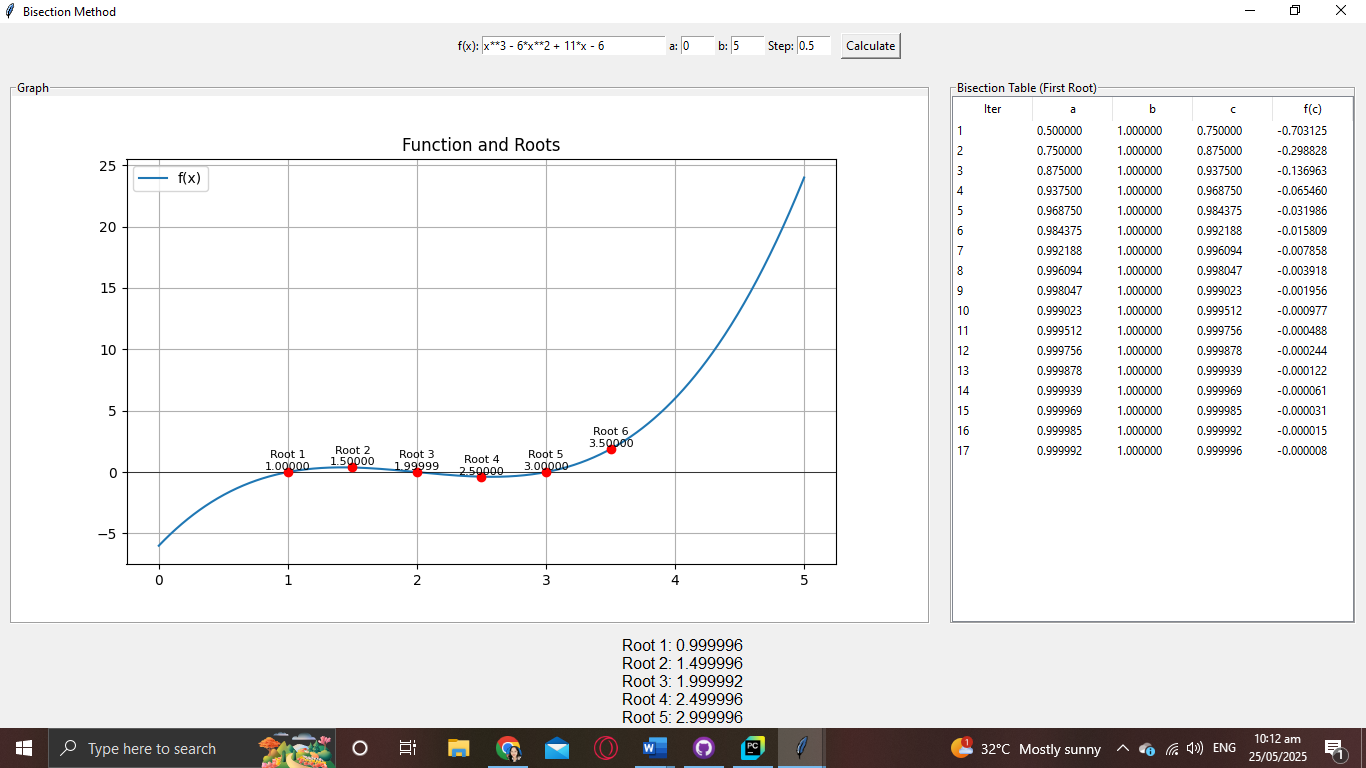
**Modules and Libraries Used**

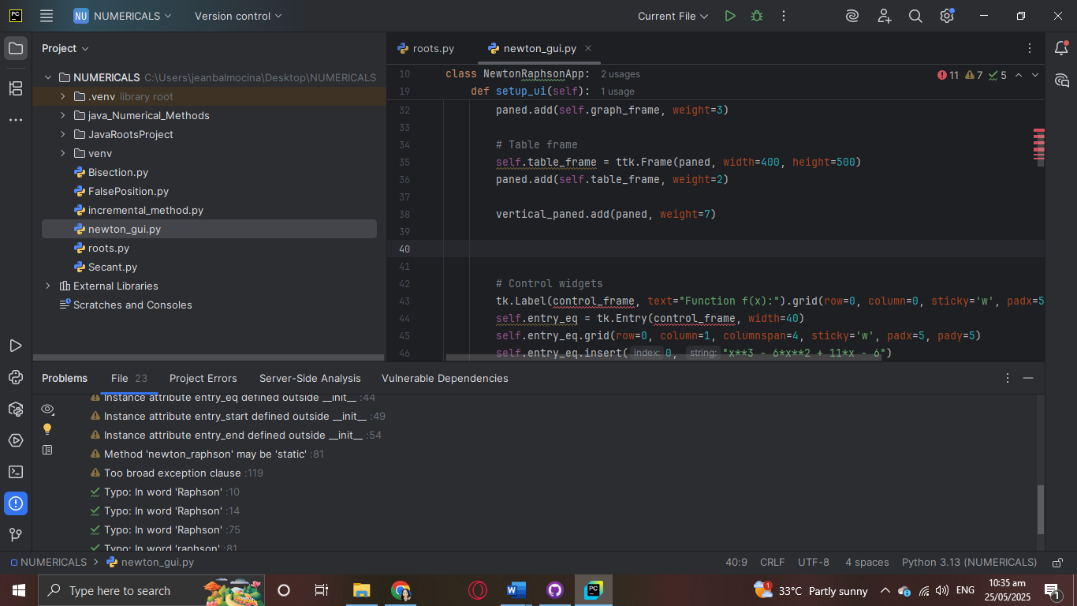
* numpy: Array and numerical calculations
* sympy: Symbolic computation and derivatives
* matplotlib: Function plotting and graphical output
* pandas: Table formatting and data management
* tkinter: Core GUI framework
* re: Input validation via regular expressions
* platform, os: OS detection and file handling

**Main Function Descriptions**

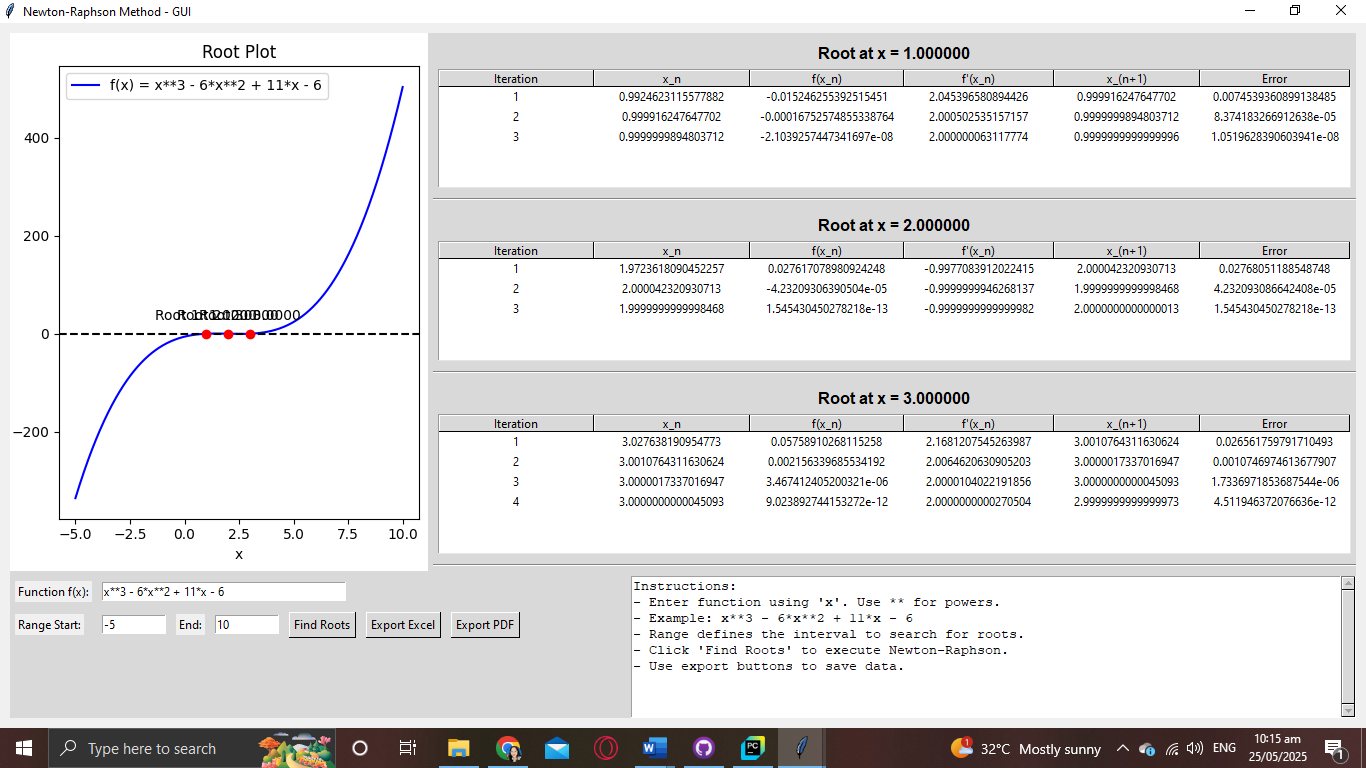
* parse\_function\_input(input\_str): Converts string expressions into callable functions.
* incremental\_search(...): Implements scanning with step size to locate sign changes.
* bisection\_method(...): Applies interval halving to locate root.
* regula\_falsi\_method(...): Uses secant line for interpolation.
* newton\_raphson\_method(...): Iterates using function and derivative.
* secant\_method(...): Iterates based on finite difference approximation.
* plot\_function\_on\_canvas(...): Displays function plot with annotations.
* display\_table(...): Generates iteration table from results.
* solve\_equation (): Central logic for input processing and result handling.
* toggle\_dark\_mode (): Switches UI theme.
* clear\_all (): Resets all fields and outputs.

**Testing and debugging**

Initial output.



Final output.



**UNIVERSITY OF BOHOL**

CITY OF TAGBILARAN

**CURRICULUM VITAE**

**I. PERSONAL INFORMATIONS**



Name : Niña Jean G. Balmocina

Address : Basiao, Garcia Hernandez, Bohol

Birthdate : September 9, 2005

Nationality : Filipino

Religion : Roman Catholic

Civil Status : Single

Parents : Pedro L. Balmocina

Julia G. Balmocina

**II. EDUCATIONAL BACKGROUND**

Tertiary : University of Bohol, College of Engineering,

Technology, Architecture and Fine Arts

Dr. Cecilio Putong Street, Cogon Tagbilaran City, Bohol

2023-Present

Secondary : Tabuan National High School

Tabuan, Garcia Hernandez, Bohol

S.Y. 2017-2023

Elementary : Tabuan Elementary School

Tabuan, Garcia Hernandez, Bohol

S.Y. 2011-2017

**Future Development**

**Recommendations**

* **Incorporate Advanced Root-Finding Methods**  
  Introduce additional algorithms such as Brent’s Method or the Modified Newton-Raphson to improve accuracy and handle edge cases more effectively.
* **Strengthen Input Validation**  
  Implement robust checks for user inputs to catch syntax errors, undefined expressions, or incompatible parameter values before computation begins.
* **Add Complex Number Support**  
  Extend capabilities to compute and represent complex roots, either visually with separate plots or through formatted output panels.
* **Improve Graph Interaction**  
  Allow users to interact directly with the graph — such as clicking on points to view details, dragging to zoom, and resetting the view with a single click.
* **Enable Method Performance Comparison**  
  Let users run multiple numerical methods side-by-side and visualize differences in convergence speed, iteration count, and final results.
* **Implement Data Export Features**  
  Provide tools to save iteration logs, result summaries, and plots in formats like CSV, Excel, or PDF for external use or analysis.
* **Support for Simultaneous Equations**  
  Add features to solve systems of nonlinear equations and display their solutions either numerically or graphically.
* **Create Platform-Independent Executables**  
  Bundle the application into standalone installers for Windows, macOS, and Linux to ensure easy installation without requiring a Python environment.
* **Expand to Mobile and Web Interfaces**  
  Adapt the app for mobile use with frameworks like Kivy, or build a web-based version using Django, Flask, or Streamlit for browser-based interaction.
* **Add Testing and Maintenance Tools**  
  Introduce automated tests and integrate continuous deployment tools to keep the project stable, maintainable, and ready for future upgrades.

**Project Cost**

|  |  |
| --- | --- |
| Expenses | Costs |
| Printing | 110 |
| Bookbinding | 150 |
| Total | 260 |

**Glossary**

***Root-Finding*** *A category of numerical techniques used to identify values of xxx for which a function f(x)=0f(x) = 0f(x)=0.*

***Incremental Search*** *A preliminary method that steps through a function’s domain to locate intervals where a sign change occurs, indicating the presence of a root.*

***Bisection Method*** *A bracketing technique that repeatedly divides an interval in half and selects the subinterval containing a sign change to converge on a root.*

***False Position Method (Regula Falsi)*** *A bracketing method similar to Bisection, but uses a linear approximation between interval endpoints to estimate the root more effectively.*

***Newton-Raphson Method*** *An open method that uses the function’s derivative to trace tangent lines and rapidly approach a root from an initial guess.*

***Secant Method*** *An open method that approximates the derivative by connecting two prior estimates with a secant line, eliminating the need for symbolic differentiation.*

***Multiple Roots Detection*** *A feature that allows the program to identify and display more than one root, particularly helpful for functions with several valid solutions.*

***Function Parser*** *A system that translates mathematical expressions entered as strings into executable or symbolic forms for evaluation and differentiation.*

***Tkinter*** *The default GUI library in Python, used to design and manage interface elements like buttons, labels, input fields, and windows.*

***SymPy*** *A Python package for symbolic computation used to interpret mathematical functions, compute derivatives, and simplify expressions.*

***Matplotlib*** *A Python plotting library used to generate function graphs, iteration visuals, and annotate roots on coordinate plots within the GUI.*

***NumPy*** *A Python library optimized for array-based numerical operations, essential for efficiently evaluating function values over ranges.*

***Pandas*** *A powerful data manipulation library used to organize iteration results into structured tables for display and export.*

***ScrolledText*** *A scrollable text widget from Tkinter used to present real-time logs, iteration data, and error messages inside the application window.*

***FigureCanvasTkAgg*** *A component that connects Matplotlib plots to the Tkinter interface, enabling interactive chart displays within the GUI.*

***Dark Mode*** *A visual theme that modifies the application's appearance using darker backgrounds and lighter text to reduce eye strain.*

***PDF/Excel Export*** *Functionality allowing users to save computed results and plots in portable file formats for documentation or academic submission.*

***Dynamic Equation Input*** *A flexible input field that lets users define their own functions using standard mathematical syntax for evaluation.*

***Standalone Executable*** *A version of the application packaged for deployment without requiring Python installation, typically created using tools like PyInstaller.*

**Bibliography**

Kaw, A. (n.d.). *Chapter 03.03: Bisection Method of Solving a Nonlinear Equation*. Holistic Numerical Methods. Retrieved from <https://nm.mathforcollege.com/nmsims/03NLE/03NLE_Bisection_Method/bisection-method_en.html>