Welcome to Python Programming!

Teaching Assistant of Introduction to Numerical Analysis Class

Class Date: 2024-2025

Instructor: Dr. Mina Zarei

Class Link: Telegram group.

Exercise class time: **Monday at 9:30 P002 Class** (determined by student vote)

Session 1 (Install Python in VS Code)

Follow the instructions in the GitHub link.

link: www.github.com/AliSeif96/Introduction-to-Numerical-Analysis/).

---

session 2 (Basic training)

1. Variables and Data Types

2. Taking Output and Input from the User

3. Conditionals (if, elif, else)

4. Loops

5. Functions

---

session 3 (Basic training 2)

6. Lists

7. Arrays

8. Matrices

9. Using `NumPy`

---

session 4 (Root finding)

10.1. Bisection Method (روش تصنیف)

10.2. False Position Method (روش نا بجایی)

10.3. Newton-Raphson Method (روش نیوتن رافسون)

10.4. Secant Method (روش وتری)

10.5. Fixed-Point Iteration (روش تکرار ساده)

10.6. Brent's Method

In [3]:

```python
import math  # Importing the math library for mathematical functions

def bisection_method(f, a, b, tol=1e-6, max_iter=100): # (روش تصنیف)
    """
    The bisection method is a bracketing method that halves the interval [a, b]
    in which the root lies until the desired tolerance is achieved.
    """
    for _ in range(max_iter):  # Loop to perform iterations up to max_iter
        c = (a + b) / 2  # Midpoint of the interval
        if f(c) == 0 or (b - a) / 2 < tol:  # Check if root is "found" or tolerance is met
            return c  # Return the root
        if f(a) * f(c) < 0:  # Root lies in the "left" subinterval
            b = c  # Update upper bound
        else:  # Root lies in the "right" subinterval
            a = c  # Update lower bound
    return c  # Return the approximate root

def false_position_method(f, a, b, tol=1e-6, max_iter=100): #(روش نا بجایی)
    """
    The false position method improves on bisection by using a linear
    approximation to estimate the root within the interval [a, b].
    """
    for _ in range(max_iter):  # Loop to perform iterations up to max_iter
        c = b - (f(b) * (b - a)) / (f(b) - f(a))  # Compute the root approximation
        if f(c) == 0 or abs(f(c)) < tol:  # Check if root is "found" or "tolerance" is met
            return c  # Return the root
        if f(a) * f(c) < 0:  # Root lies in the "left" subinterval
            b = c  # Update upper bound
        else:  # Root lies in the "right" subinterval
            a = c  # Update lower bound
    return c  # Return the approximate root

def newton_raphson_method(f, df, x0, tol=1e-6, max_iter=100): #(روش نیوتن رافسون)
    """
    The Newton-Raphson method uses the derivative of the function to
    iteratively find the root with quadratic convergence near the root.
    """
    x = x0  # Initial guess for the root
    for _ in range(max_iter):  # Loop to perform iterations up to max_iter

        x_new = x - f(x) / df(x)  # Update using Newton-Raphson formula
        if abs(x_new - x) < tol:  # Check if tolerance is met
            return x_new  # Return the root
        x = x_new  # Update the current guess
    return x  # Return the approximate root

def secant_method(f, x0, x1, tol=1e-6, max_iter=100): #(روش وتری)
    """
    The secant method approximates the derivative by using two initial points
    and iteratively refines the root approximation.
    """
    for _ in range(max_iter):  # Loop to perform iterations up to max_iter
        if f(x1) - f(x0) == 0:  # Check to avoid division by zero
            return x1  # Return the current guess
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))  # Compute the root approximation
        if abs(x2 - x1) < tol:  # Check if tolerance is met
            return x2  # Return the root
        x0, x1 = x1, x2  # Update the previous two points
    return x1  # Return the approximate root

def fixed_point_iteration(g, x0, tol=1e-6, max_iter=100): # (روش تکرار ساده)
    """
    Fixed-point iteration rewrites the equation as x = g(x) and uses iteration
    to refine the root approximation, depending on the choice of g(x).
    """
    x = x0  # Initial guess for the root
    for _ in range(max_iter):  # Loop to perform iterations up to max_iter
        x_new = g(x)  # Update using the fixed-point iteration formula
        if abs(x_new - x) < tol:  # Check if tolerance is met
            return x_new  # Return the root
        x = x_new  # Update the current guess
    return x  # Return the approximate root

def brent_method(f, a, b, tol=1e-6):
    """
    Brent's method combines bisection, secant, and inverse quadratic interpolation
    to efficiently and robustly find the root within the interval [a, b].
    """
    from scipy.optimize import brentq  # Import Brent's method from scipy
    return brentq(f, a, b, xtol=tol)  # Find and return the root

    #from scipy import optimize
    #return optimize.brentq(f, a, b, xtol=tol)


def main():
    # Example function: f(x) = x^3 - x - 2                                        def f(x):return x**3 - x - 2
    f = lambda x: x**3 - x - 2  # Define the function
    df = lambda x: 3 * x**2 - 1  # Derivative of f(x)
    g = lambda x: (x + 2)**(1/3)  # Rearranged form for fixed-point iteration

    # Interval and initial guesses
    a, b = 1, 2  # Define the interval [a, b]
    x0, x1 = 1.5, 2  # Define initial guesses for open methods

    print("Root found using 10.1.Bisection Method:", bisection_method(f, a, b))  # Call bisection method
    print("Root found using 10.2.False Position Method:", false_position_method(f, a, b))  # Call false position method
    print("Root found using 10.3.Newton-Raphson Method:", newton_raphson_method(f, df, x0))  # Call Newton-Raphson method
    print("Root found using 10.4.Secant Method:", secant_method(f, x0, x1))  # Call secant method
    print("Root found using 10.5.Fixed-Point Iteration:", fixed_point_iteration(g, x0))  # Call fixed-point iteration
    print("Root found using 10.6.Brent's Method:", brent_method(f, a, b))  # Call Brent's method

if __name__ == "__main__":
    main()  # Execute the main function
```

```
Root found using 10.1.Bisection Method: 1.5213804244995117
Root found using 10.2.False Position Method: 1.5213796360454928
Root found using 10.3.Newton-Raphson Method: 1.5213797068045751
Root found using 10.4.Secant Method: 1.5213797068044876
Root found using 10.5.Fixed-Point Iteration: 1.5213796792714414
Root found using 10.6.Brent's Method: 1.5213799532716044
```

In [ ]:

In [ ]:

In [ ]: