Lab Manual for Numerical Analysis

Lab No. 3

SOLUTION OF NON-LINEAR EQUATIONS BY BISECTION METHOD & FALSE POSITION METHOD

Software Engineering Department
Bahria University (Karachi Campus)

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 3

Solution of Non-linear Equations by Bisection Method & False Position Method

OBJECTIVE:

This lab aims to teach students the bisection and false position methods for solving nonlinear equations, emphasizing practical applications in scientific and engineering contexts.

Introduction

In the domains of mathematics and science, **equations** play an indispensable role as foundational instruments for modeling and comprehending the intricacies of real-world phenomena. **Linear equations**, by nature, offer a relatively uncomplicated path to solutions. However, the landscape becomes notably more intricate when one encounters **nonlinear equations**. These equations, characterized by their departure from linear relationships between variables, emerge as ubiquitous challenges across a spectrum of disciplines, encompassing physics, engineering, economics, and biology. The quest for solutions to these nonlinear equations frequently stands as a pivotal undertaking, representing a critical stride in untangling the complexities of multifaceted problems and advancing our understanding of the natural world.

Nonlinear equations

A **linear equation** is a mathematical expression of **degree 1**, typically involving variables and coefficients. For instance, an equation like $4x + 5 = 0$, where x is the variable, is considered a linear equation. Linear equations can have one or more variables, and they follow a pattern like $ax + by + cz + ... = d$, where a, b, c, ... are coefficients, and x, y, z, ... are variables.

On the other hand, a **quadratic equation** is a type of nonlinear equation characterized by having a **degree of 2** or **higher**. For example, $x^2 + 3x + 2 = 0$ represents a single-variable nonlinear equation. Nonlinear equations can involve various degrees of complexity, and a **second-degree nonlinear equation** is commonly referred to as a quadratic equation. When the degree goes beyond 2, it is termed as a **cubic equation** or **higher.** Linear equations can often be solved analytically, solving nonlinear equations is generally more challenging and may not always have analytical solutions.

Solution of nonlinear equations

There are two primary categories of methods for finding the roots of algebraic equations: **direct methods** and **indirect (iterative) methods**.

Direct Methods: Direct methods are capable of providing the **exact roots** in a finite number of steps. These methods aim to determine all the roots simultaneously, assuming minimal or **no impact** from **round-off errors**. Among the direct methods, **Elimination Methods** offer a notable advantage, particularly when dealing with large systems of equations.

Indirect (Iterative) Methods: **Indirect methods** operate on the principle of successive approximations. The general approach involves initiating the process with one or more initial approximations to the root and then obtaining a sequence of iterates denoted as "x." These iterates converge, in the limit, to the actual or true solution of the root. Indirect methods typically determine one or two roots at a time, and they are less sensitive to rounding errors. Moreover, these methods exhibit self-correcting characteristics and are relatively straightforward to program, making them suitable for **computer implementation.**

Indirect methods are further categorized into two subtypes:

a. **Bracketing Methods:** Bracketing methods require knowledge of the limits within which the root is located. Examples of bracketing methods include the **Bisection method** and the **False Position method**.

b. **Open Methods:** Open methods, in contrast, necessitate an initial estimation of the solution. The Newton-Raphson method is an illustration of an open method.

Implementation of Bisection Method and False Position Method

In this section, we'll provide Python implementations for both the **Bisection Method** and the **False Position Method**, two powerful techniques for finding roots of equations. These methods offer different approaches to iteratively refine solutions within specified intervals, enabling accurate numerical solutions to a wide range of equations.

a. Bisection Method

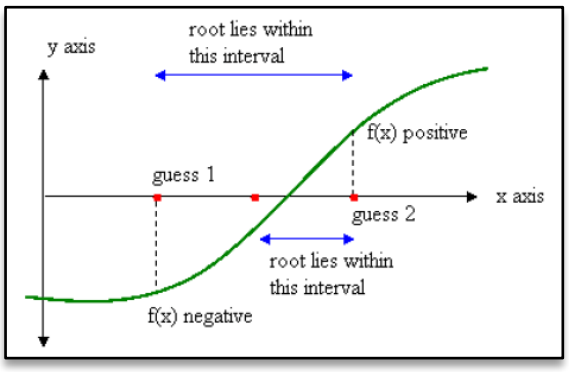
The **bisection method** is also called the **interval halving** method, the **binary search** method or the **dichotomy method**. This method is used to find root of an equation in a given interval that is value of 'x' for which **f(x) = 0**.

It requires that **f** is a **continuous function** defined over an interval **[a, b]**, and the conditions **f(a)** and **f(b)** have opposite signs, meaning that **f(a) * f(b) < 0**. When these conditions are met, it signifies that the interval **[a, b]** "brackets" a root of the function because, according to the **intermediate value theorem**, the continuous function f must have at least one root within this interval (a, b).

To apply the method, we calculate the **midpoint**, denoted as c, using the formula:

$$c = \frac{a + b}{2}$$

Then, we select either the interval **[a, c]** or **[c, b]** based on the sign of **f(c) * f(a)**. If f(c) * f(a) is negative, we choose the interval [a, c]; otherwise, we choose the interval [c, b]. This step is crucial in narrowing down the interval containing the root.



Implementation in Python

```
# Define the function for which you want to find the root
def func(x):
    return x**2 - 4  # Change the equation here

# Function to find the root using Bisection Method
def bisection(a, b, num_iterations):
    if (func(a) * func(b) >= 0):
        print("The chosen interval does not bracket a root.")
        return

    for _ in range(num_iterations):
        root = (a + b) / 2

        if func(root) == 0:
            print("Found the exact root:", root)
            return

        if func(root) * func(a) < 0:
            b = root
        else:
            a = root

    print("The approximate root after", num_iterations,
"iterations is:", root)

# Define the interval
a = 1
b = 3
num_iterations = 10 # Specify the number of iterations

# Find the root of the equation using the Bisection Method
bisection(a, b, num_iterations)
```

b. False Position Method

The **False Position Method**, also known as the **Regula Falsi Method**, is a numerical technique used to approximate the root of a mathematical function within a specified interval. This method is particularly effective when dealing with continuous functions and the goal is to find the value of 'x' for which the function equals zero, i.e., **f(x) = 0**. The method relies on the principles of **linear interpolation** to iteratively narrow down the interval in which the root lies.

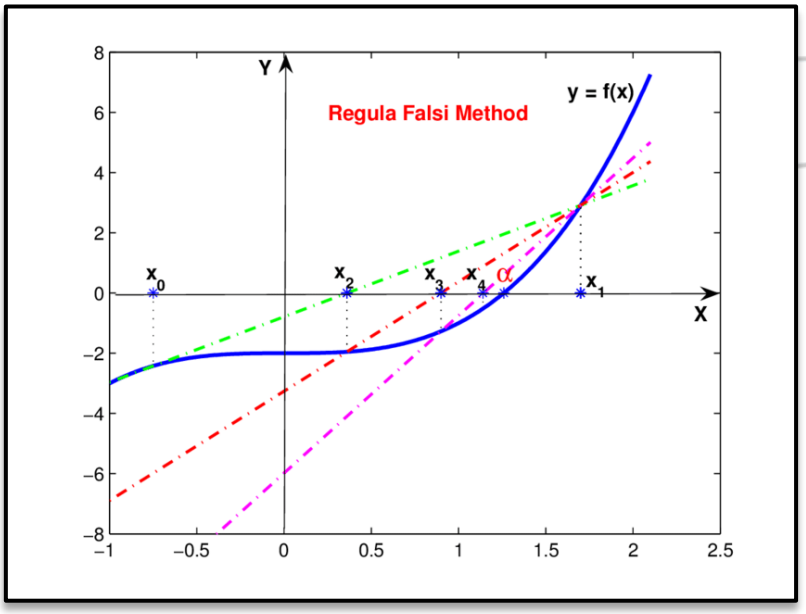
The formula for the False Position Method is as follows:

$$c = a - \frac{f(a) * (b - a)}{f(b) - f(a)}$$

Here, 'a' and 'b' represent the **endpoints** of the interval, **'c'** is the **estimated root**, and **'f(x)'** is the given function. The formula calculates **'c'** as the point where the function crosses the x-axis, and it updates the interval based on the sign of **'f(c)'** to either **[a, c]** or **[c, b]**.

It begins with two initial points, 'a' and 'b', chosen so that 'f(a)' and 'f(b)' have opposite signs, indicating the presence of a root within the interval [a, b]. The method employs a formula to calculate 'c', the point where the function crosses the x-axis. It then checks if 'f(c)' is sufficiently close to zero; if so, 'c' becomes an approximate root, and the process concludes. If 'f(c)' isn't close to zero, the method updates either 'a' or 'b' based on the sign of 'f(c)' to narrow down the interval. If 'f(c)' and 'f(a)' have opposite signs, 'b' is updated to 'c', otherwise 'a' is updated to 'c'. This iterative process of calculating 'c', updating the interval, and checking for proximity to zero continues until 'f(c)' is satisfactorily close to zero or until a predefined maximum number of iterations is reached.

The False Position Method combines simplicity with relatively fast convergence, making it an effective tool for root-finding in various mathematical contexts.



Implementation in Python

```
# Maximum number of iterations allowed
MAX_ITER = 10

# Define a different example function to find the root of
def function(x):
  # Replace this function with your desired equation
  # Example: x^2 - 4
  return (x**2 - 4)

# Function to find the root using the Regula Falsi (False
Position) Method
def regulaFalsi(a, b):
  # Check if the initial values a and b are chosen correctly
  if function(a) * function(b) >= 0:
    print("The chosen values of 'a' and 'b' are not
suitable.")
    return -1

  # Initialize the result as 'a'
  c = a

  # Perform iterations
  for i in range(MAX_ITER):
    # Calculate the point where the function crosses the x-
axis
    c = a - ((function(a)*(b-a))/(function(b)-function(a)))


    # Check if the found point is the root
    if function(c) == 0:
      break

    # Determine which side to continue the search
    elif function(c) * function(a) < 0:
      b = c
    else:
      a = c

  print("The estimated root is: ", '%.4f' %c)

# Driver code to test the root-finding function
# Initial values for the interval [a, b]
a = 1
b = 3
regulaFalsi(a, b)
```

Lab Tasks

1. Write a python program for approximating the roots of following functions using bisection method:
    a. x^3- 9x+1 starting with the interval [2, 4]
    b. 3x = √ (1 + sinx) starting with the interval [0, 1]

2. Write a python program for approximating the roots of following functions using false-position method:

    a. $f(x) = (x-4)^2 (x+2)$ starting with the interval [-2.5, -1.0]
    b. $f(x) = e^x (3.2\sin(x) - 0.5\cos(x))$ starting with the interval [3,4]