PHY1112 Lab 12

Digging up the roots of a function

April 2nd, 2024

|  |  |  |
| --- | --- | --- |
| Part | 1 | Total |
| Points | 20 | 20 |
| Score |  |  |

Objectives

1. Implement the bisection method to find the roots of a function.
2. Test it on functions where we can find the roots analytically.

Part 1: Bisection – getting to the root of a function, one half at a time!

1. (10 points) Write a python function that implements the bisection root finding algorithm as we discussed in class.

The inputs are :

* A function handle that defines a function whose root we would like to find
* Initial boundary points and , such that
* The desired tolerance,

The output is:

* The found root

The function should:

* check the signs and before your main loop begins. if one of them is zero, return the root; if not, and their signs are not different, print an error message and return NaN or none.

Hint: You may need to use, np.sign(), np.ceil(), and/or np.log2()

This will not be vectorized. Rather, it will use if and for loops similar to what we did in Unit 1.

*def* bisection\_root(*f*, *a*, *b*, *tolerance*):

    '''

    (float, float, function) -> float

    This function finds the root of a function f(x) using the bisection method.

    Preconditions: tolerance is positive.

    '''

    # Check if a and b are ordered correctly (i.e b > a). If they are not, swap them and continue.

    if a >= b:

        a, b = b, a

        print("a and b have been swapped due to ValueError: a >= b")

    # Check if f(a) and f(b) have the same sign. If they do, return None.

    if np.sign(f(a)) == np.sign(f(b)):

        print("ValueError: f(a) and f(b) have the same sign. Please choose different a and b.")

        return None

    # Check if f(a) or f(b) is zero. If they are, return the corresponding value (this is the root).

    if f(a) == 0:

        return a

    if f(b) == 0:

        return b

    # Calculate the root using the bisection method within tolerance.

    while abs(b - a) > tolerance:

        # Calculate the midpoint of the interval.

        c = (a + b) / 2

        # Check if the midpoint is the root.

        if f(c) == 0:

            return c

        # Check if the root is in the left half of the interval.

        elif f(a) \* f(c) < 0:

            # Update the interval if the root is within the bounds.

            b = c

        else:

            # Update the interval.

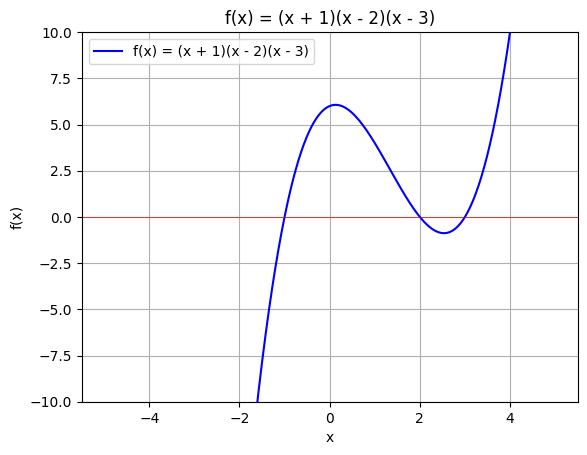
            a = c

    # Return the root.

    return c

1. (5 points) Plot the function . Choosing = , run your bisection function from question 1 for each of the following initial boundary points, and take a snapshot of your results and include it here.
   1. and
   2. and
   3. and
   4. and
   5. and

Find all the roots analytically and compare them to the answers your code has given for parts i. to v. above. Discuss your findings.



**Figure 1.** A graph of the function f(x) = (x+1)(x-2)(x-3) over the interval [-5, 5].

A computer screen with white text

Description automatically generated

Analytical roots:

[-5, -2]: None

[-2, 1]: -1

[1, 2.5]: 2

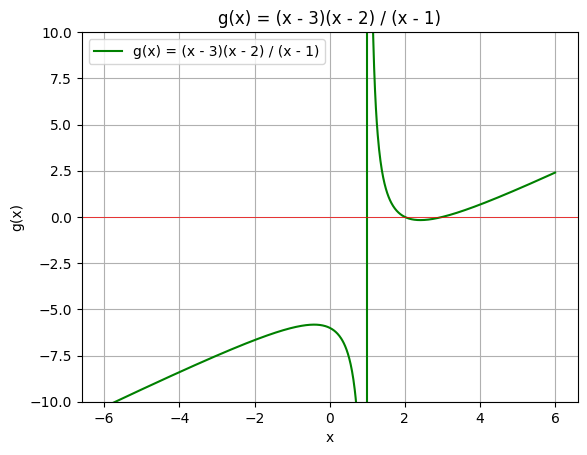
[2.5, 3]: 3

[1, 4]: 2, 3

Each root found with bisection is within the tolerance (1e-5) of the analytical value unless the bound is on the root (exact) or there are multiple roots within the interval (thus returns None).

1. (5 points) Consider the function . Choosing = , run your bisection function from question 1 for each of the following initial boundary points, and take a snapshot of your results and include it here.
   1. and
   2. and
   3. and
   4. and
   5. and

Find all the roots analytically and compare them to the answers your code has given for parts i. to v. above. Discuss your findings.



**Figure 2.** An incomplete graph of the function g(x) = (x-3)(x-2)/(x-1). Note the asymptote that should be at x = 1 is displayed as a vertical line connecting the endpoints that should go to infinity.

A computer screen with white text

Description automatically generated

Analytical roots:

[-2, 0]: None

[0, 1.5]: None, Asymptote at x = 1

[1.5, 2.75]: 2

[2.75, 4]: 3

[4, 5.5]: None

The roots found (aside from at x = 1) are all within tolerance (1e-5) of the analytical roots, however due to the nature of the function, there is thought to be a root at x = 1 which is in fact an asymptote.

**CODE:**

'''

Filename:       lab\_12.py

Author:         Patrick Geraghty

Date Created:   2024-04-02

Date Modified:  2024-04-02

Description:    Lab 12 - Functions

'''

import numpy as np

import matplotlib.pyplot as plt

# Part 1.

*def* bisection\_root(*f*, *a*, *b*, *tolerance*):

    '''

    (float, float, function) -> float

    This function finds the root of a function f(x) using the bisection method.

    Preconditions: tolerance is positive.

    '''

    # Check if a and b are ordered correctly (i.e b > a). If they are not, swap them and continue.

    if a >= b:

        a, b = b, a

        print("a and b have been swapped due to ValueError: a >= b")

    # Check if f(a) and f(b) have the same sign. If they do, return None.

    if np.sign(f(a)) == np.sign(f(b)):

        print("ValueError: f(a) and f(b) have the same sign. Please choose different a and b.")

        return None

    # Check if f(a) or f(b) is zero. If they are, return the corresponding value (this is the root).

    if f(a) == 0:

        return a

    if f(b) == 0:

        return b

    # Calculate the root using the bisection method within tolerance.

    while abs(b - a) > tolerance:

        # Calculate the midpoint of the interval.

        c = (a + b) / 2

        # Check if the midpoint is the root.

        if f(c) == 0:

            return c

        # Check if the root is in the left half of the interval.

        elif f(a) \* f(c) < 0:

            # Update the interval if the root is within the bounds.

            b = c

        else:

            # Update the interval.

            a = c

    # Return the root.

    return c

# Part 2.

*def* f(*x*):

    '''

    (float) -> float

    This function defines the function f(x) = (x + 1)(x - 2)(x - 3).

    Preconditions: None.

    '''

    return (x + 1) \* (x - 2) \* (x - 3)

*def* f\_x\_plot():

    '''

    () -> None

    This function plots the function f(x) = (x + 1)(x - 2)(x - 3) over the interval [-5, 5].

    Preconditions: None.

    '''

    # Define the x values.

    x = np.linspace(-5, 5, 1000)

    # Define the y values.

    y = f(x)

    # Plot the function.

    plt.figure(1)

    plt.plot(x, y, *label*="f(x) = (x + 1)(x - 2)(x - 3)", *color*="blue")

    plt.axhline(0, *color*="red", *linewidth*=0.5)

    plt.xlabel("x")

    plt.ylabel("f(x)")

    plt.ylim(-10, 10)

    plt.title("f(x) = (x + 1)(x - 2)(x - 3)")

    plt.grid()

    plt.legend()

    plt.show()

*def* f\_x\_root\_snapper():

    '''

    () -> None

    This function finds the roots of the function f(x) = (x + 1)(x - 2)(x - 3) using the bisection method for varying endpoints.

    Preconditions: None.

    '''

    print("Roots of f(x) = (x + 1)(x - 2)(x - 3) on:")

    print("[-5, -2]:", bisection\_root(f, -5, -2, 1e-5))

    print()

    print("[-2, 1]:", bisection\_root(f, -2, 1, 1e-5))

    print()

    print("[1, 2.5]:", bisection\_root(f, 1, 2.5, 1e-5))

    print()

    print("[2.5, 3]:", bisection\_root(f, 2.5, 3, 1e-5))

    print()

    print("[1, 4]:", bisection\_root(f, 1, 4, 1e-5))

# Part 3.

*def* g(*x*):

    '''

    (float) -> float

    This function defines the function g(x) = (x - 3)(x - 2) / (x - 1)

    Preconditions: None.

    '''

    return (x - 3) \* (x - 2) / (x - 1)

*def* g\_x\_plot():

    '''

    () -> None

    This function plots the function g(x) = (x - 3)(x - 2) / (x - 1) over the interval [-6, 6].

    Preconditions: None.

    '''

    # Define the x values.

    x = np.linspace(-6, 6, 1000)

    # Define the y values.

    y = g(x)

    # Plot the function.

    plt.figure(2)

    plt.plot(x, y, *label*="g(x) = (x - 3)(x - 2) / (x - 1)", *color*="green")

    plt.axhline(0, *color*="red", *linewidth*=0.5)

    plt.xlabel("x")

    plt.ylabel("g(x)")

    plt.ylim(-10, 10)

    plt.title("g(x) = (x - 3)(x - 2) / (x - 1)")

    plt.grid()

    plt.legend()

    plt.show()

*def* g\_x\_root\_snapper():

    '''

    () -> None

    This function finds the roots of the function g(x) = (x - 3)(x - 2) / (x - 1) using the bisection method for varying endpoints.

    Preconditions: None.

    '''

    print("Roots of g(x) = (x - 3)(x - 2) / (x - 1) on:")

    print("[-2, 0]:", bisection\_root(g, -2, 0, 1e-5))

    print()

    print("[0, 1.5]:", bisection\_root(g, 0, 1.5, 1e-5))

    print()

    print("[1.5, 2.75]:", bisection\_root(g, 1.5, 2.75, 1e-5))

    print()

    print("[2.75, 4]:", bisection\_root(g, 2.75, 4, 1e-5))

    print()

    print("[4, 5.5]:", bisection\_root(g, 4, 5.5, 1e-5))

*def* main():

    f\_x\_plot()

    f\_x\_root\_snapper()

    g\_x\_plot()

    g\_x\_root\_snapper()