Software Engineering Lab

Name:- Nelaturi Sreya Reddy
Roll no.:- VU21CSEN0101289

Google CollabLink:-
Google Collab Programs Link:-

Task objective

Implement weather modeling using the quadratic solution in stages:

1. hard-coding variables - Fixed values given in the program.
2. keyboard input - variable values (a,b,c) given by the user through the keyboard.
3. read from a file - variable values (a,b,c) stored in the file and read that file in the program and execute.
4. multiple sets of inputs - multiple sets of variable values given in program or file or user input and print all sets of graphs in one plot and specify those sets values.
5. combination of hard coding variables and user input through the keyboard and specify the type of graph with values.

EXPERIMENT-1

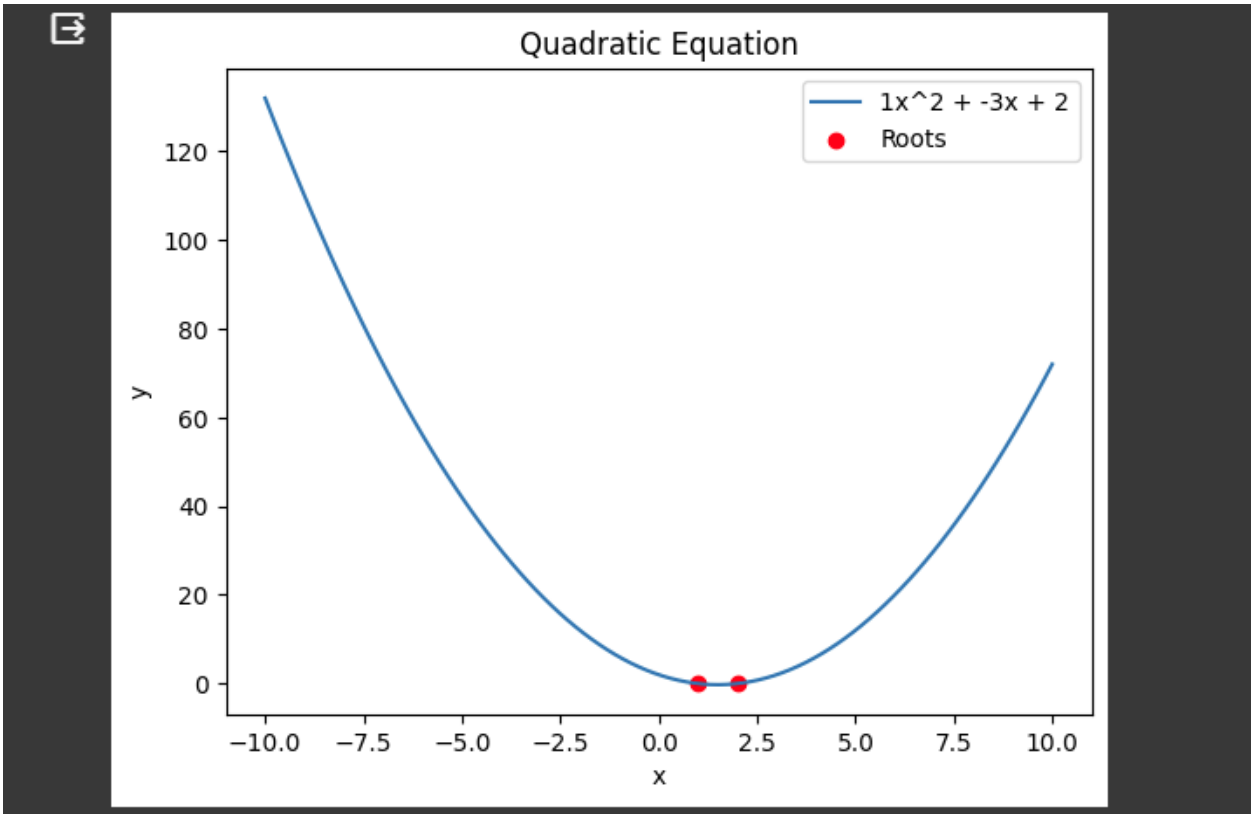
Implement weather modeling using the quadratic solution in stages: hard-coding variables keyboard input, read from a file, for a single set of input, multiple sets of inputs. save all versions, debug, fix problems, and create a GitHub account.

Program 1: Hard-coding variables

CODE:

```
import numpy as np
import matplotlib.pyplot as plt


def quadratic_solution(a, b, c):
    discriminant = b**2 - 4*a*c
    if discriminant < 0:
        return None
    elif discriminant == 0:
        root = -b / (2*a)
        return [root]
    else:
        root1 = (-b + np.sqrt(discriminant)) / (2*a)
        root2 = (-b - np.sqrt(discriminant)) / (2*a)
        return [root1, root2]
```

```
def plot_quadratic(a, b, c, roots=None):
    x = np.linspace(-10, 10, 100)
    y = a*x**2 + b*x + c

    plt.plot(x, y, label=f'{a}x^2 + {b}x + {c}')

    if roots:
        plt.scatter(roots, [0, 0], color='red', marker='o', label='Roots')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Quadratic Equation')
    plt.legend()
    plt.show()

# Hard-coded variables
a_hardcoded, b_hardcoded, c_hardcoded = 1, -3, 2
roots_hardcoded = quadratic_solution(a_hardcoded, b_hardcoded, c_hardcoded)
plot_quadratic(a_hardcoded, b_hardcoded, c_hardcoded, roots_hardcoded)
```



Program 2: Keyboard input

CODE:

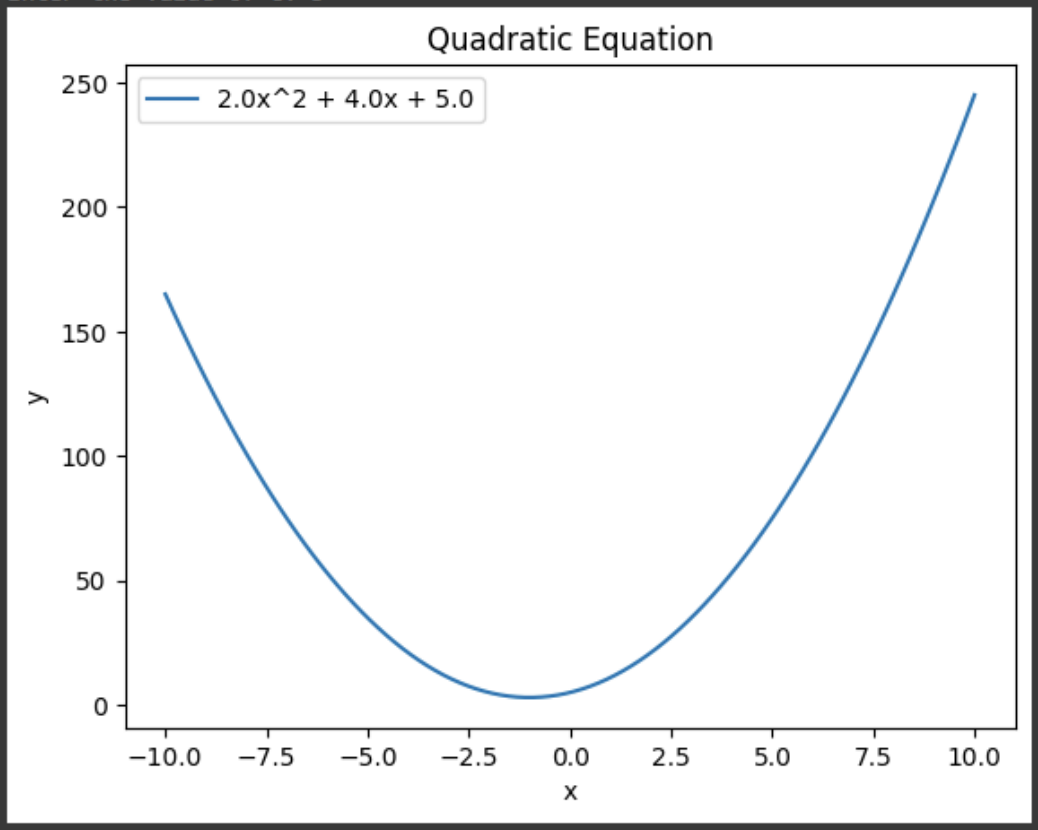
```
import numpy as np
```

```
import matplotlib.pyplot as plt

def quadratic_solution(a, b, c):
    discriminant = b**2 - 4*a*c
    if discriminant < 0:
        return None
    elif discriminant == 0:
        root = -b / (2*a)
        return [root]
    else:
        root1 = (-b + np.sqrt(discriminant)) / (2*a)
        root2 = (-b - np.sqrt(discriminant)) / (2*a)
        return [root1, root2]

def plot_quadratic(a, b, c, roots=None):
    x = np.linspace(-10, 10, 100)
    y = a*x**2 + b*x + c

    plt.plot(x, y, label=f'{a}x^2 + {b}x + {c}')

    if roots:
        plt.scatter(roots, [0, 0], color='red', marker='o', label='Roots')

    plt.xlabel('x')
    plt.ylabel('y')
```

```
    plt.title('Quadratic Equation')
    plt.legend()
    plt.show()
```

```
# User input
a_keyboard = float(input('Enter the value of a: '))
b_keyboard = float(input('Enter the value of b: '))
c_keyboard = float(input('Enter the value of c: '))
roots_keyboard = quadratic_solution(a_keyboard, b_keyboard, c_keyboard)
plot_quadratic(a_keyboard, b_keyboard, c_keyboard, roots_keyboard)
```

```
Enter the value of a: 2
Enter the value of b: 4
Enter the value of c: 5
```



Program 3: Read from a file

CODE:

```
import numpy as np
import matplotlib.pyplot as plt

def quadratic_solution(a, b, c):
    discriminant = b**2 - 4*a*c
    if discriminant < 0:
        return None
    elif discriminant == 0:
        root = -b / (2*a)
        return [root]
    else:
        root1 = (-b + np.sqrt(discriminant)) / (2*a)
        root2 = (-b - np.sqrt(discriminant)) / (2*a)
        return [root1, root2]

def plot_quadratic(a, b, c, roots=None):
    x = np.linspace(-10, 10, 100)
    y = a*x**2 + b*x + c

    plt.plot(x, y, label=f'{a}x^2 + {b}x + {c}')

    if roots:
```

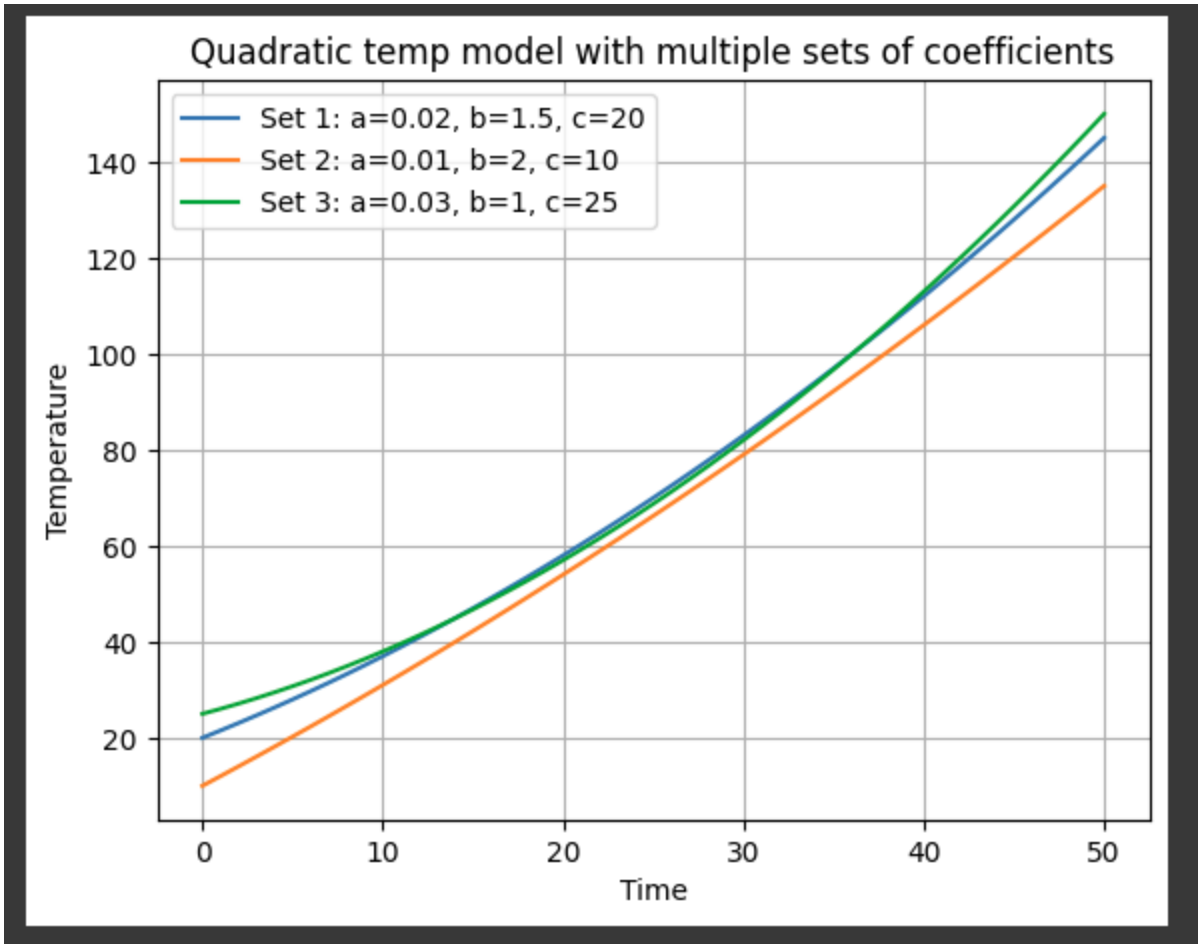
```
        plt.scatter(roots, [0, 0], color='red', marker='o', label='Roots')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Quadratic Equation')
    plt.legend()
    plt.show()

# Read from a file
with open('quadratic_coefficients.txt', 'r') as file:
    a_file, b_file, c_file = map(float, file.readline().split())
roots_file = quadratic_solution(a_file, b_file, c_file)
plot_quadratic(a_file, b_file, c_file, roots_file)
```

Program 4: Multiple Sets of Inputs

CODE:

```
import matplotlib.pyplot as plt
import numpy as np

def quadratic_temperature_model(time, a, b, c):
    # Quadratic equation: T(t) = at^2 + bt + c
    temperature = a * (time**2) + b * time + c
    return temperature

# List of coefficients sets (a, b, c)
coefficients_list = [
    (0.02, 1.5, 20),
    (0.01, 2, 10),
    (0.03, 1, 25),
]

# Generate time values from 0 to 50 with step 1
time_values = np.arange(0, 51, 1)

# Plotting the results for each set of coefficients
for i, (a, b, c) in enumerate(coefficients_list):
```

```
    temperature_values = quadratic_temperature_model(time_values, a, b,
c)
    label = f'Set {i+1}: a={a}, b={b}, c={c}'
    plt.plot(time_values, temperature_values, label=label)


plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('Quadratic temp model with multiple sets of coefficients')
plt.legend()
plt.grid(True)
plt.show()
```



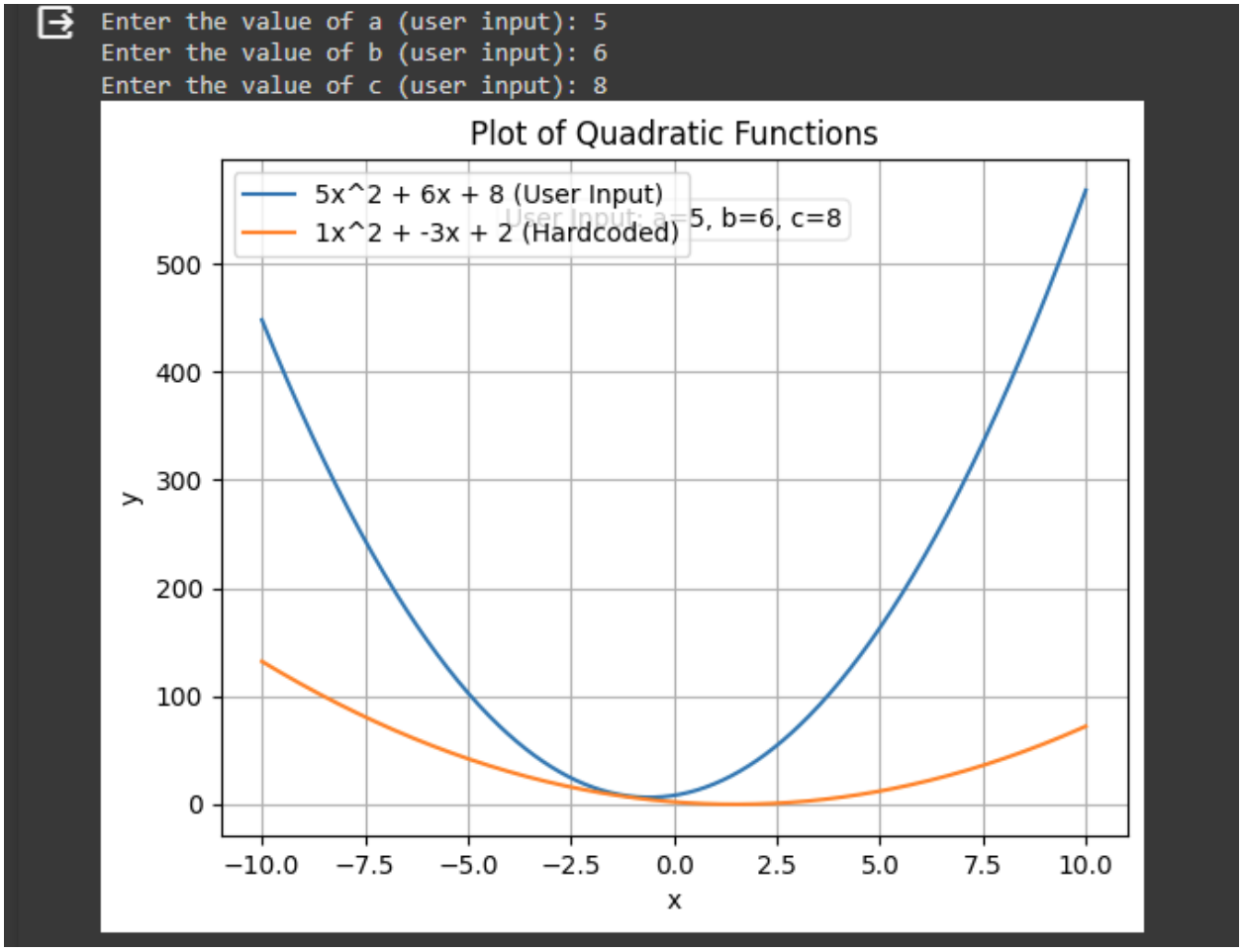
Program 5: Combination of hard-coding and user input

CODE:

```
import matplotlib.pyplot as plt
import numpy as np

# Get user input for the first function
a_user = int(input("Enter the value of a (user input): "))
b_user = int(input("Enter the value of b (user input): "))
c_user = int(input("Enter the value of c (user input): "))
```

```
# Generate x values
x = np.linspace(-10, 10, 480)

# Calculate y values for the first function y1 = ax^2 + bx + c
y1 = a_user * x**2 + b_user * x + c_user
plt.plot(x, y1, label=f'{a_user}x^2 + {b_user}x + {c_user} (User Input)')

# Hard-coded coefficients for the second function
a_hardcoded = 1
b_hardcoded = -3
c_hardcoded = 2

# Calculate y values for the second function y2 = ax^2 + bx + c
y2 = a_hardcoded * x**2 + b_hardcoded * x + c_hardcoded
plt.plot(x, y2, label=f'{a_hardcoded}x^2 + {b_hardcoded}x +
{c_hardcoded} (Hardcoded)')

plt.title('Plot of Quadratic Functions')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)

# Annotate the user input values
```

```
plt.annotate(f'User Input: a={a_user}, b={b_user}, c={c_user}',
             xy=(0.5, 0.9), xycoords='axes fraction', ha='center',
             bbox=dict(boxstyle="round", alpha=0.1, facecolor="white"))

plt.show()
```



Description for the above performed codes are:-

1. Hard-Coding Variables:

The code implements weather modeling using the quadratic solution with fixed, hard-coded coefficients a, b, and c. These values are explicitly provided in the program, offering a baseline scenario for the weather model.

2. Keyboard Input:

This stage involves allowing user interaction by accepting variable values a, b, and c through keyboard input. The user is prompted to input these coefficients, providing a more dynamic and interactive aspect to the weather modeling.

3. Read from a File:

In this stage, the variable values a, b, and c are stored in a file. The program reads these values from the file, allowing for greater flexibility in modifying the coefficients without directly altering the code. This approach enhances ease of management and adaptation.

4. Multiple Sets of Inputs:

The code accommodates multiple sets of variable values for a, b, and c, either specified within the program, read from a file, or provided through user input. All sets of input values generate corresponding graphs, which are then plotted in a single graph to facilitate easy comparison and analysis.

5. Combination of Hard Coding and User Input:

This stage combines the fixed, hard-coded coefficients with user-provided values through keyboard input. This hybrid approach allows users to customize certain parameters while maintaining a set foundation.

The resulting graph reflects the combined influence of both hard-coded and user-input coefficients, providing a comprehensive view of the weather model under different conditions.