SOFTWARE ENGINEERING LAB TASK 6

08-01-2025

HU22CSEN0100287

SAI GANESH ESWARAPRASAD

Aim:

Develop weather modeling using the quadratic model using Waterfall model.

Waterfall Model:

The Waterfall Model is a traditional software development methodology that follows a linear and sequential approach. Each phase must be completed before moving to the next.

Phases:

1. Requirement Analysis

- o Gather all relevant data sources for weather modeling, such as temperature and humidity.

- o Specify the requirements for a quadratic model and the desired outputs.

2. Design

- o Develop a mathematical representation of the quadratic model: $y = ax^2 + bx + c$.

- o Create data structures and flowcharts to outline the process.

3. Implementation

- o Write the code for the quadratic model using a chosen programming language, such as Python.

4. Verification

○ Test the model's predictions by comparing them with historical weather data.

5. Deployment

○ Deliver the final system for practical use.

6. Maintenance

○ Periodically review and update the model to ensure accuracy and relevance.

Steps Of Implementation:

1. Library Imports

imports **NumPy** for numerical computations and **Matplotlib** for creating visualizations.

```
Weather Modeling Prediction Using Waterfall Model Approach

[1]   # Import necessary libraries
Os    import numpy as np
      import matplotlib.pyplot as plt
```
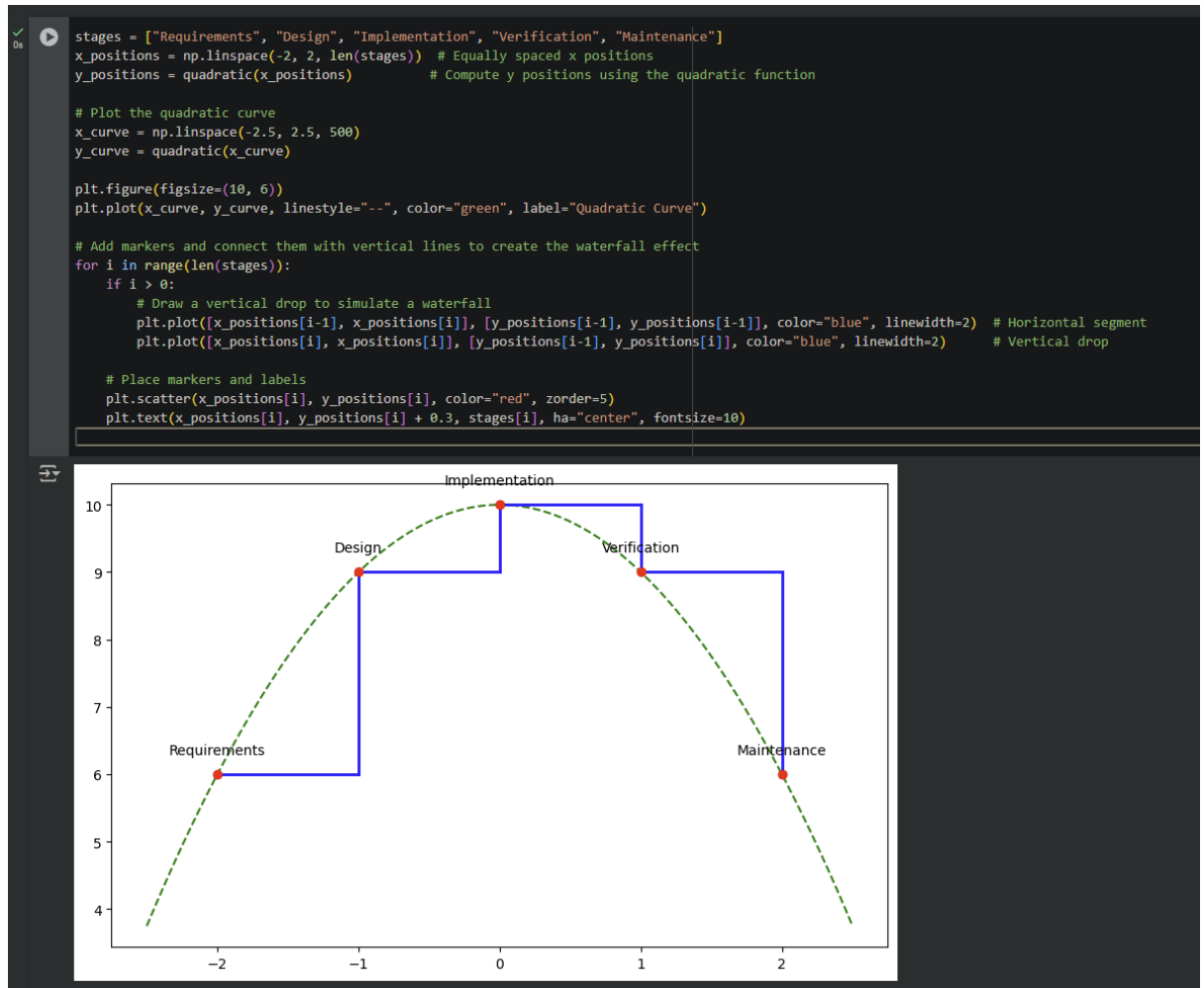
2. Define Quadratic Function

Defines a quadratic function y=ax2+bx+cy = ax^2 + bx + cy=ax2+bx+c, which will model the values for the stages of the Waterfall Model.

```
      # Define the quadratic function
Os    def quadratic(x, a=-1, b=0, c=10):
          return a * x**2 + b * x + c
```

3. Waterfall Model Visualization with Quadratic Curve

This code plots a quadratic curve to represent the stages of the Waterfall Model, with markers and vertical lines creating a waterfall effect between each stage.

```python
stages = ["Requirements", "Design", "Implementation", "Verification", "Maintenance"]
x_positions = np.linspace(-2, 2, len(stages))  # Equally spaced x positions
y_positions = quadratic(x_positions)           # Compute y positions using the quadratic function

# Plot the quadratic curve
x_curve = np.linspace(-2.5, 2.5, 500)
y_curve = quadratic(x_curve)

plt.figure(figsize=(10, 6))
plt.plot(x_curve, y_curve, linestyle="--", color="green", label="Quadratic Curve")

# Add markers and connect them with vertical lines to create the waterfall effect
for i in range(len(stages)):
    if i > 0:
        # Draw a vertical drop to simulate a waterfall
        plt.plot([x_positions[i-1], x_positions[i]], [y_positions[i-1], y_positions[i-1]], color="blue", linewidth=2)  # Horizontal segment
        plt.plot([x_positions[i], x_positions[i]], [y_positions[i-1], y_positions[i]], color="blue", linewidth=2)      # Vertical drop

    # Place markers and labels
    plt.scatter(x_positions[i], y_positions[i], color="red", zorder=5)
    plt.text(x_positions[i], y_positions[i] + 0.3, stages[i], ha="center", fontsize=10)
```

4. Styling and Displaying the Plot

Finalizes the plot by adding titles, axes labels, grid lines, and a legend, then displays the output.

```python
plt.plot(x_curve, y_curve, linestyle="--", color="green", label="Quadratic Curve")
plt.title("Weather Modeling: Waterfall Model Representation Using Quadratic Curve", fontsize=14)
plt.xlabel("Stages of the Waterfall Model", fontsize=12)
plt.ylabel("Model Values", fontsize=12)
plt.axhline(0, linewidth=0.5, linestyle="--", color="black", alpha=0.7)
plt.axvline(0, linewidth=0.5, linestyle="--", color="black", alpha=0.7)
plt.grid(color="gray", linestyle="--", linewidth=0.5, alpha=0.7)
plt.legend()
plt.show()
```


Weather Modeling: Waterfall Model Representation Using Quadratic Curve