Assignment Overview

This project involves building and evaluating a polynomial using a binary tree structure, where each polynomial term is represented by nodes. Key functionalities include constructing polynomials from specified roots, symbolic differentiation, and numerical evaluation. Additionally, the project includes a Python script to visualize the polynomial and its derivatives, enhancing understanding of the polynomial's behavior.

The code consists of four main components:

1. **main.cpp** - The main entry point, handling test cases and invoking the evaluation and differentiation processes.
2. **polynomial.cpp** - Implements the core functionality of polynomial differentiation, evaluation, and construction.
3. **polynomial.hpp** - Defines the data structures for nodes and the binary tree, as well as helper methods.
4. **plot_polynomials.py** - A Python script for reading evaluation data and plotting the polynomial and its derivatives.

1. main.cpp

The **main.cpp** file initializes test cases with two sets of polynomial roots. It uses the saveEvaluations method to generate evaluation files containing the polynomial's value and its derivatives at specific points. Exception handling is implemented to catch and display errors during file operations or polynomial evaluations.

Key Features:

- Initializes polynomials using root values and triggers evaluation saving.
- Outputs success messages upon generating files for easy debugging.
- Catches and displays errors related to polynomial generation or file handling, enhancing user feedback.

2. polynomial.cpp

This file implements core methods for **PolynomialBinaryTree**, including symbolic differentiation, evaluation, and polynomial construction from specified zeros. Differentiation employs symbolic rules such as the product rule, while evaluation computes polynomial values recursively.

Key Features:

- **Differentiation**: Uses the product rule for terms with multiplication and handles various node types (constants, variables, operators).
- **Evaluation**: Computes polynomial values based on recursive traversal of nodes.

- **Polynomial Construction**: Builds polynomials as binary trees from specified roots, structuring each root as a term.

Challenges and Solutions:

- Managing pointers during differentiation required careful memory handling to prevent accidental modification of existing nodes.
- Recursive evaluation ensured flexibility for any polynomial structure but demanded efficient memory use for large trees.

3. polynomial.hpp

This file contains the declarations for the **Node** class and **PolynomialBinaryTree** class. Node represents each polynomial term or operation, and PolynomialBinaryTree provides methods for polynomial creation, differentiation, and evaluation.

Key Features:

- **Node Structure**: Each node represents constants, variables, or operators, enabling polynomial representation as binary trees.
- **Exception Safety**: The PolynomialBinaryTree includes error handling, especially for invalid inputs, such as an empty vector of roots.

Challenges and Solutions:

- Constructing robust nodes required specialized constructors to initialize constants, variables, and operators correctly.
- Memory management for the Node class was essential for stability, as each Node's lifecycle affects tree traversal accuracy.

4. plot_polynomials.py

This Python script reads polynomial evaluations and plots the polynomial and its derivatives. Each derivative is plotted with distinct colors, labels, and annotations for zeros based on the filename. This visualization helps analyze polynomial behavior.

Key Features:

- Reads data from generated evaluation files and visualizes derivatives up to the fifth order.
- Customizes plot appearance with labels, gridlines, and annotations for clarity.

Challenges and Solutions:

- Adapting the script to handle different numbers of derivatives required careful indexing for consistent color and labeling.

- Adding annotations for zeros and managing plot layout ensured clear visualization without overlap or visual clutter.