In this project, you will derive the general formulas for the slope and y-intercept of a least squares line, implement these calculations in Python, and verify results with sample data. This project explores mathematical derivation, Python implementation, and comparison with numerical optimization methods like gradient descent.

Requirements: Submit your project in PDF format, including code snippets and explanations.

Due Date: December 1st at 5PM

Preparation: Complete `PartI_differentiation.ipynb` and `PartII_autograd.ipynb`

Tasks:
Assume that you have n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, with the equation of the least squares line given by $y = b + mx$.

1. *Create Example Data*

   To create a synthetic dataset, let's generate x-values and corresponding y-values with some added noise. Use the code below to create a set of points that should roughly lie on a line. You may choose different values for the true slope and intercept.

```python
import numpy as np
import random

# Set a seed for reproducibility
np.random.seed(42)

# Generate x-values
x = np.linspace(0, 10, 50)
# Define the true slope and intercept
true_m = 2
true_b = 5
# Generate y-values with some noise
y = true_b + true_m * x + np.random.normal(0, 1, len(x)
    )
```

   Write Python code to plot the example data and plot the line given by $y = \text{true\_b} + \text{true\_m} \times x$.

2. *Modeling the Least Squares Line*

   (a) For each data point (x_i, y_i), show that the corresponding point on the least squares line has a y-coordinate of $b + mx_i$.

   (b) Implement a Python function, `y_predicted(x, b, m)`, that calculates $b+mx$ for a given x, b, and m.

3. *Calculating Squared Distances*

   (a) Show that for each data point (x_i, y_i), the square of the vertical distance from it to the point on the line is $(y_i - (b + mx_i))^2$.

(b) Write a Python function, `squared_distance(y, y_pred)`, to compute $(y - y_{\text{predicted}})^2$ for values y and $y_{\text{predicted}}$.

4. *Defining and Minimizing the Cost Function*

(a) Define the cost function $f(b, m)$ as the sum of all n squared distances:

$$f(b, m) = \sum_{i=1}^{n} (y_i - (b + mx_i))^2.$$

Show that the partial derivatives $\frac{\partial f}{\partial b}$ and $\frac{\partial f}{\partial m}$ are:

$$\frac{\partial f}{\partial b} = -2 \sum_{i=1}^{n} (y_i - (b + mx_i)),$$

$$\frac{\partial f}{\partial m} = -2 \sum_{i=1}^{n} (y_i - (b + mx_i)) \cdot x_i.$$

(b) Write three Python functions

i. `sum_of_squared_distances(x, y, b, m)`

ii. `partial_derivative_b`

iii. `partial_derivative_m`

to compute $f(b, m)$, $\frac{\partial f}{\partial b}$, and $\frac{\partial f}{\partial m}$.

(c) Write python code to compute $\frac{\partial f}{\partial b}$, and $\frac{\partial f}{\partial m}$ using Symbolic differentiation and Automatic differentiation. Check your answers and compare the computation time.

5. *Setting Up and Solving the System of Equations*

(a) Show that setting $\frac{\partial f}{\partial b} = 0$ and $\frac{\partial f}{\partial m} = 0$ results in the linear system:

$$nb + \left(\sum_{i=1}^{n} x_i \right) m = \sum_{i=1}^{n} y_i,$$

$$\left(\sum_{i=1}^{n} x_i \right) b + \left(\sum_{i=1}^{n} x_i^2 \right) m = \sum_{i=1}^{n} x_i y_i.$$

Explain why solving for b and m minimizes the cost function.

(b) Write a Python function, `solve_least_squares(x, y)`, to solve for b and m using this system of linear equations.

6. *Deriving Explicit Formulas for b and m*

(a) Solve the equations above to derive explicit formulas for b and m:

$$b = \frac{\left(\sum_{i=1}^{n} x_i^2\right) \sum_{i=1}^{n} y_i - \left(\sum_{i=1}^{n} x_i\right) \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2},$$

$$m = \frac{n \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right) \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}.$$

(b) Implement `solve_least_squares_formula(x, y)` in Python to calculate b and m directly from these formulas. Compare with `solve_least_squares(x, y)` on example data.

7. *Comparing with Gradient Descent*

   (a) Implement gradient descent to minimize $f(b, m)$ by iteratively updating b and m using the partial derivatives.

   (b) Compare results from `solve_least_squares`, `solve_least_squares_formula`, and gradient descent on the example dataset. Discuss any differences in convergence and accuracy.

8. Based on your study of `PartI_differentiation.ipynb` and `PartII_autograd.ipynb`,

   (a) discuss the advantage and limitations of numerical differentiation, symbolic differentiation and automatic differentiation

   (b) summarize the connection of multivariate chain rule and backpropagation, and discuss why is backward differentiation efficient for optimizing neural networks