Python Programming Assignment

February 12, 2025

Instructions

- Solve each problem using **Python functions**.
- You may use built-in functions, loops, and `numpy` where necessary.
- Add **comments** to explain your code.

1 Problem 1: Normalizing a List of Numbers

Normalization is a way to **scale numbers** so they have a **mean of 0** and a **standard deviation of 1**. This makes data easier to compare.

Task

Write a function `normalize(data)` that:

1. Computes the **mean** μ of `data`.
2. Computes the **standard deviation** σ.
3. Returns a **new list** where each element is calculated as:

$$x' = \frac{x - \mu}{\sigma}$$

Example

```
>>> normalize([1, 2, 3, 4, 5])
[-1.26, -0.63, 0.0, 0.63, 1.26]
```

2 Problem 2: Finding the Best Line Through a Set of Points

A line can be defined as:

$$y = mx + b$$

where m is the **slope** and b is the **intercept**. Given a set of points, we want to find the **best line** that fits them.

Task

Write a function `best_fit_line(points)` that:

- Accepts a list of **(x, y) coordinates**.
- Computes the best **slope** m and **intercept** b using:

$$m = \frac{N\sum(xy) - \sum x \sum y}{N\sum x^2 - (\sum x)^2}$$

$$b = \frac{\sum y - m\sum x}{N}$$

Example

```
>>> best_fit_line([(1, 2), (2, 2.8), (3, 3.6)])
m = 0.8, b = 1.2
```

3 Problem 3: Checking Stability of a System

A **system is stable** if the largest **eigenvalue** of a matrix is ≤ 1.

Task

Write a function `is_stable(matrix)` that:

- Computes the **eigenvalues** of a square matrix.
- Returns `True` if **all eigenvalues have absolute value** ≤ 1, otherwise `False`.

Example

```
>>> is_stable([[0.5, 0.2], [0.1, 0.7]])
True  # Eigenvalues < 1     Stable
>>> is_stable([[2, -1], [-1, 2]])
False  # Large eigenvalues     Unstable system
```

4 Problem 4: Comparing Two Types of Error

Different **error measures** tell us how close our predictions are to actual values.

Task

Write a function `compare_errors(y_true, y_pred)` that:

- Computes the **Squared Error**:

$$SE = \sum (y_{\text{true}} - y_{\text{pred}})^2$$

- Computes the **Logarithmic Error** (assuming $y_{\text{pred}} \neq 0$):

$$LE = -\sum y_{\text{true}} \log(y_{\text{pred}})$$

Example

```
>>> compare_errors([2, 4, 6], [2.1, 3.8, 5.9])
Squared Error: 0.06
Logarithmic Error: 0.01
```