Part 1: Basic implementation of batch gradient descent

1. Importing necessary libraries

First, we need to import the libraries required for numerical operations and graph plotting.

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

2. Defining the dataset

Then, we define the input (*X*) and output (*y*) values as NumPy arrays.

```
X = np.array([1, 2, 3])
y = np.array([2, 2.8, 3.6])
```

3. Initialising the parameters

We initialise the slope *m* and intercept *b* to zero, set the learning rate, and define the number of iterations for processing.

```
m = 0  # slope
b = 0  # intercept

learning_rate = 0.1
num_iterations = 5
```

4. Implementation of the batch gradient descent technique

Next, we define a function to implement the batch gradient descent technique on the given dataset. This function will calculate the predicted values, compute the gradients for the slope and intercept, and update these parameters iteratively.

```
def gradient_descent(X, y, m, b, learning_rate, num_iterations):
    n = len(y)
    for _ in range(num_iterations):
        y_pred = m * X + b

        dm = (-1/n) * sum(X * (y - y_pred))
        db = (-1/n) * sum(y - y_pred)

        m -= learning_rate * dm
        b -= learning_rate * db

    return m, b
```

5. Running the gradient descent function

We call the gradient descent function and obtain the final values for the slope and intercept after the specified number of iterations.

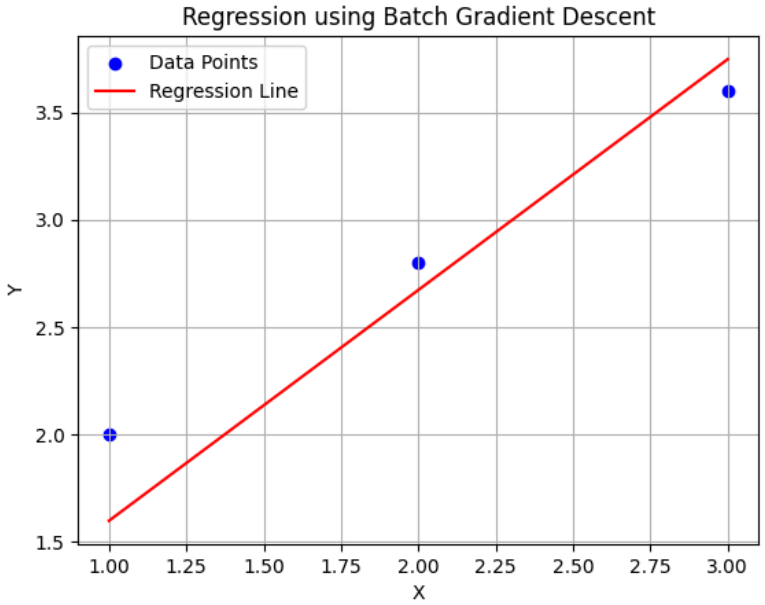
```
m, b = gradient_descent(X, y, m, b, learning_rate, num_iterations)

# Displaying the final coefficients
m, b
```

```
(1.0747509794238683, 0.5225422716049383)
```

6. Visualising the results

Finally, we plot the original data points and the regression line obtained from the gradient descent.

```
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, m * X + b, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Regression using Batch Gradient Descent')
plt.legend()
plt.grid()
plt.show()
```



Part 2: Basic implementation of stochastic gradient descent

1. Importing necessary libraries

First, we need to import the libraries required for numerical operations and graph plotting.

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

2. Defining the dataset

Then, we define the input (X) and output (y) values as NumPy arrays.

```
# Defining the dataset
X = np.array([1, 2, 3])
y = np.array([2, 2.8, 3.6])
```

3. Initialising the parameters

We initialise the slope `m` and intercept `b` to zero, set the learning rate, and define the number of iterations for processing.

```
m = 0  # slope
b = 0  # intercept
learning_rate = 0.1
num_iterations = 5
```

4. Implementation of the stochastic gradient descent technique

Next, we define a function to implement the stochastic gradient descent technique on the given dataset. For this, we modify the (previous) gradient descent function to update parameters for each training example.

```
def stochastic_gradient_descent(X, y, m, b, learning_rate, num_iterations):
    n = len(y)
    for _ in range(num_iterations):
        for i in range(n):
            y_pred = m * X[i] + b
            dm = -X[i] * (y[i] - y_pred)
            db = -(y[i] - y_pred)
            m -= learning_rate * dm
            b -= learning_rate * db
    return m, b
```

5. Running the gradient descent function

We call the gradient descent function and obtain the final values for the slope and intercept after the specified number of iterations.

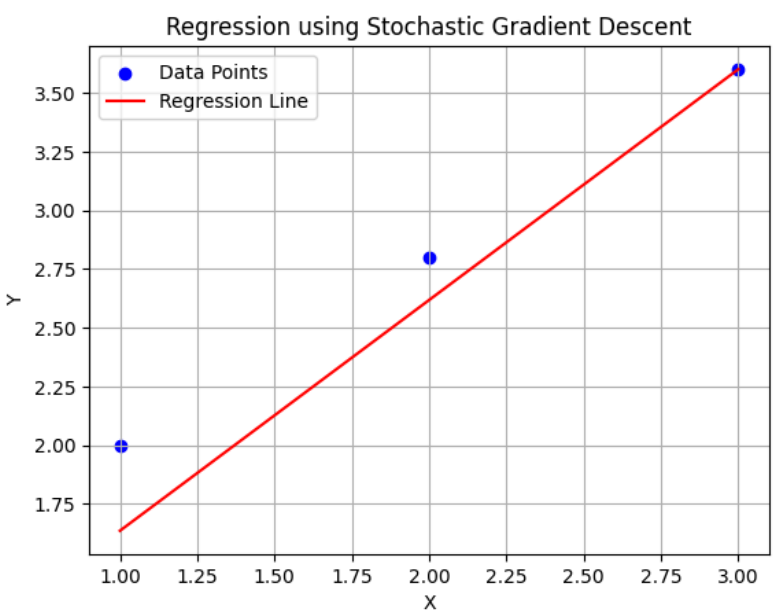
```
m, b = stochastic_gradient_descent(X, y, m, b, learning_rate, num_iterations)

# Displaying the final coefficients
m, b
```

```
(0.982090824636928, 0.653727526089216)
```

6. Visualising the results

Finally, we plot the original data points and the regression line obtained from the gradient descent.

```
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, m * X + b, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Regression using Stochastic Gradient Descent')
plt.legend()
plt.grid()
plt.show()
```



Part 3: Comparison of the two gradient descent methods

We now implement both batch gradient descent and stochastic gradient descent for linear regression, utilising the same dataset to fit a regression line. Each method updates the slope and intercept of the regression line until the Mean Squared Error (MSE) falls below a specified tolerance level, allowing for a visual comparison of the performance of the two approaches.

```
# Step 1
import numpy as np
import matplotlib.pyplot as plt

# Step 2
X = np.array([1, 2, 3])
y = np.array([2, 2.8, 3.6])

# Step 3
m_batch = 0
b_batch = 0
m_stochastic = 0
b_stochastic = 0
learning_rate = 0.1
tolerance = 0.01

# Step 4
# BGD function with convergence check
def gradient_descent(X, y, m, b, learning_rate, tolerance):
    n = len(y)
    error = float('inf')  # Initial error
    while error > tolerance:
        y_pred = m * X + b
        error = (1/n) * sum((y - y_pred) ** 2)  # Mean Squared Error
        dm = (-1/n) * sum(X * (y - y_pred))
        db = (-1/n) * sum(y - y_pred)
        m -= learning_rate * dm
        b -= learning_rate * db
    return m, b
```

```
# SGD function with convergence check
def stochastic_gradient_descent(X, y, m, b, learning_rate, tolerance):
    n = len(y)
    error = float('inf')  # Initial error
    while error > tolerance:
        for i in range(n):  # Iterate over each sample
            y_pred = m * X[i] + b
            error = (1/n) * sum((y - (m * X + b)) ** 2)  # Mean Squared Error
            dm = -X[i] * (y[i] - y_pred)
            db = -(y[i] - y_pred)
            m -= learning_rate * dm
            b -= learning_rate * db
    return m, b

# Step 5
# Running the BGD function
m_batch, b_batch = gradient_descent(X, y, m_batch, b_batch, learning_rate, tolerance)

# Running the SGD function
m_stochastic, b_stochastic = stochastic_gradient_descent(X, y, m_stochastic, b_stochastic, learning_rate, tolerance)

# Step 6
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, m_batch * X + b_batch, color='red', label='Batch Gradient Descent Line')
plt.plot(X, m_stochastic * X + b_stochastic, color='green', label='Stochastic Gradient Descent Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Comparison of Regression Lines with Convergence Check')
plt.legend()
plt.grid()
plt.show()
```