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.

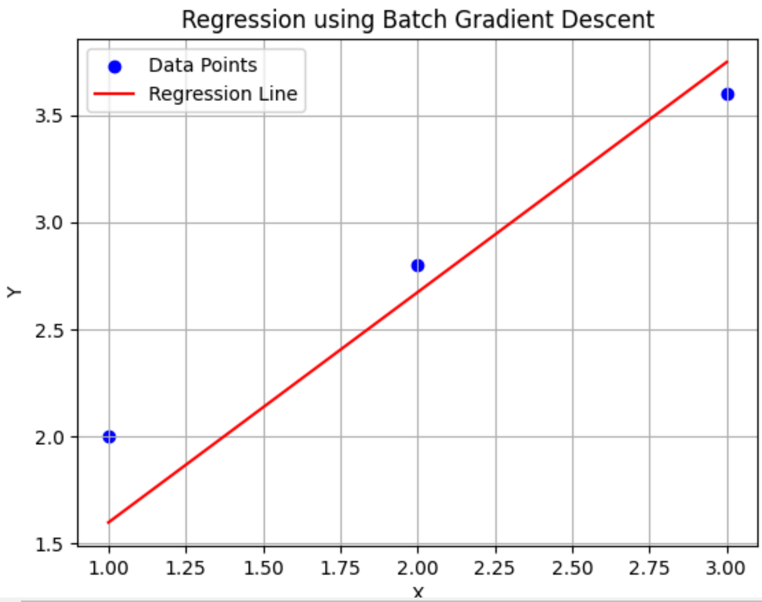
```
m, b = gradient_descent(X, y, m, b, learning_rate, num_iterations)

# 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: Gradient descent with convergence check

In this part, we define a function that will perform gradient descent until the error falls below a specified tolerance. The Mean Squared Error (MSE) will be calculated to assess the fit of the model, and the gradients for m and b will be computed as before and updated based on the gradients and learning rate. The loop will continue until the error is less than the specified tolerance.

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

# Initial regression coefficients
m = 0.0  # slope
b = 0.0  # intercept

# Function to perform gradient descent until convergence
def gradient_descent_until_convergence(X, y, m, b, learning_rate, tolerance):
    n = len(y)
    iteration = 0
    error = float('inf')  # Initial error

    while error > tolerance:
        y_pred = m * X + b

        # Calculating the MSE
        error = (1/n) * sum((y - y_pred) ** 2)

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

        # Updating parameters
        m -= learning_rate * dm
        b -= learning_rate * db

        # Print iteration details
        print(f"Iteration {iteration + 1}: m = {m}, b = {b}, error = {error}")

        iteration += 1

    return m, b

# Setting a tolerance level for convergence
tolerance = 0.1
# Setting a learning rate
learning_rate = 0.1

# Running the gradient descent function until convergence
m, b = gradient_descent_until_convergence(X, y, m, b, learning_rate, tolerance)
```

```
Iteration 1: m = 0.6133333333333333, b = 0.27999999999999997, error = 8.266666666666666
Iteration 2: m = 0.8844444444444444, b = 0.4093333333333333, error = 1.695940740740741
Iteration 3: m = 1.0031703703703703, b = 0.4715111111111111, error = 0.39136151440329225
Iteration 4: m = 1.0540553086419753, b = 0.5037259259259259, error = 0.13129822895290366
Iteration 5: m = 1.0747509794238683, b = 0.5225422716049383, error = 0.07843488637065488
```

Part 3: Gradient descent for different learning rates

Let us now run the gradient descent function for a specified number of iterations, with different learning rates.

```
import numpy as np

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

# Initial regression coefficients
m = 0.0  # slope
b = 0.0  # intercept

# Function to perform gradient descent for a fixed number of iterations
def gradient_descent(X, y, m, b, learning_rate, iterations):
    n = len(y)
    for i in range(iterations):
        y_pred = m * X + b
        error = (1/n) * sum((y - y_pred) ** 2)
        dm = (-1/n) * sum(X * (y - y_pred))
        db = (-1/n) * sum(y - y_pred)
        m -= learning_rate * dm
        b -= learning_rate * db
        print(f"Iteration {i+1}: m = {m}, b = {b}, error = {error}")
    return m, b

# Running gradient descent for different learning rates and iterations
for learning_rate in [0.1, 0.9, 0.01]:
    iterations = 100
    print(f"\nLearning Rate: {learning_rate}")
    m, b = gradient_descent(X, y, m, b, learning_rate, iterations)
```

```
Learning Rate: 0.1
Iteration 1: m = 0.6133333333333333, b = 0.27999999999999997, error = 8.266666666666666
Iteration 2: m = 0.8844444444444444, b = 0.4093333333333333, error = 1.695940740740741
Iteration 3: m = 1.0031703703703703, b = 0.4715111111111111, error = 0.39136151440329225
Iteration 4: m = 1.0540553086419753, b = 0.5037259259259259, error = 0.13129822895290366
Iteration 5: m = 1.0747509794238683, b = 0.5225422716049383, error = 0.07843488637065488
Iteration 6: m = 1.082025401371742, b = 0.5353378485596708, error = 0.06669807942336473
Iteration 7: m = 1.0833459776863283, b = 0.5453989834293553, error = 0.06314819483252626
Iteration 8: m = 1.0820380580801707, b = 0.5541898895491542, error = 0.06125129842139242
Iteration 9: m = 1.0795823197329268, b = 0.5623632889782045, error = 0.05971075662643603
Iteration 10: m = 1.0766379127285868, b = 0.5702104961337987, error = 0.05826871501911052
Iteration 11: m = 1.0734981208951533, b = 0.5778618639747015, error = 0.056873366156232875
Iteration 12: m = 1.0702932916824748, b = 0.5853760533982008, error = 0.05551378569134261
Iteration 13: m = 1.067081211551013, b = 0.5927797897218857, error = 0.05418717350743006
Iteration 14: m = 1.0638873548828298, b = 0.6000855684394946, error = 0.052892355976104546
Iteration 15: m = 1.060722808916277, b = 0.6072995406189792, error = 0.05162849683867541
Iteration 16: m = 1.0575922566315519, b = 0.6144250247738259, error = 0.05039484117022606
Iteration 17: m = 1.0544975319153957, b = 0.6214640709701329, error = 0.0491906642530646
Iteration 18: m = 1.051439202827518, b = 0.6284181574900405, error = 0.048015261099987896
Iteration 19: m = 1.0484172766766682, b = 0.6352885011755328, error = 0.04686794404803696
Iteration 20: m = 1.0454315139924497, b = 0.6420761957226458, error = 0.045748041960792386
Iteration 21: m = 1.0424815683181106, b = 0.6487822733518913, error = 0.04465489975736398
Iteration 22: m = 1.039567048432614, b = 0.65540773235308, error = 0.0435878780136503
Iteration 23: m = 1.0366875460267782, b = 0.6619535494312492, error = 0.04254635258522802
Iteration 24: m = 1.0338426479946985, b = 0.6684206852827687, error = 0.04152971424165297
Iteration 25: m = 1.0310319418739522, b = 0.6748100871555521, error = 0.04053736830997442
Iteration 26: m = 1.0282550182349974, b = 0.6811226900652065, error = 0.039568734326863794
Iteration 27: m = 1.0255114717122906, b = 0.6873594174116864, error = 0.038623245699072314
Iteration 28: m = 1.0228009014308843, b = 0.6935211813280596, error = 0.03770034937200743
Iteration 29: m = 1.020122911164193, b = 0.6996088829090767, error = 0.03679950550622829
Iteration 30: m = 1.0174771093724209, b = 0.7056234123853304, error = 0.035920187161671994
Iteration 31: m = 1.014863109188225, b = 0.7115656492723132, error = 0.03506187998942448
Iteration 32: m = 1.0122805283792573, b = 0.7174364625074369, error = 0.03422408193085776
Iteration 33: m = 1.0097289893007833, b = 0.7232367105808417, error = 0.03340630292395475
Iteration 34: m = 1.0072081188442494, b = 0.7289672416626009, error = 0.0326080646166528
Iteration 35: m = 1.0047175483844129, b = 0.7346288937274909, error = 0.03182890008703572
Iteration 36: m = 1.0022569137261887, b = 0.7402224946778592, error = 0.031068353570212415
Iteration 37: m = 0.9998258550517288, b = 0.7457488624648355, error = 0.030325980191721592
Iteration 38: m = 0.9974240168679549, b = 0.7512088052080061, error = 0.029601345707306537
Iteration 39: m = 0.9950510479546414, b = 0.7566031213136145, error = 0.02889402624890832
Iteration 40: m = 0.9927066013130859, b = 0.7619325995913248, error = 0.028203608076727754
Iteration 41: m = 0.9903903341153808, b = 0.7671980193695751, error = 0.027529687337212783
Iteration 42: m = 0.9881019076542881, b = 0.7724001506095415, error = 0.0268718698268276
Iteration 43: m = 0.9858409872937121, b = 0.7775397540177297, error = 0.026229770761467545
Iteration 44: m = 0.9836072424197672, b = 0.7826175811572142, error = 0.025603014551383047
Iteration 45: m = 0.9814003463924329, b = 0.7876343745575394, error = 0.02499123458148202
Iteration 46: m = 0.9792199764977897, b = 0.7925908678232989, error = 0.024394072996881695
Iteration 47: m = 0.9770658139008281, b = 0.7974877857414111, error = 0.02381118049358497
Iteration 48: m = 0.9749375435988261, b = 0.8023258443871044, error = 0.023242216114158505
Iteration 49: m = 0.9728348543752864, b = 0.8071057512286287, error = 0.02268684704829256
Iteration 50: m = 0.970757438754427, b = 0.8118282052307085, error = 0.02214474843812695
Iteration 51: m = 0.9687049929562194, b = 0.8164938969567522, error = 0.021615603188228572
Iteration 52: m = 0.9666772168519665, b = 0.8211035086698332, error = 0.02109910178010929
Iteration 53: m = 0.9646738139204155, b = 0.8256577144324565, error = 0.020594942091176186
```