

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 np.random.seed(42)
5 m = 2000
6 n = 2
7
8 X = np.random.rand(m, n)
9 X[:, 0] = 1
10 true_theta = np.array([2, 3])
11 y = X.dot(true_theta) + np.random.randn(m) * 0.3
12
13 theta = np.zeros(n)
14 learning_rate = 0.01
15 num_iterations = 500
16
17 def compute_cost(X, y, theta):
18     m = len(y)
19     predictions = X.dot(theta)
20     errors = predictions - y
21     cost = (1 / (2 * m)) * np.sum(errors ** 2)
22     return cost

```

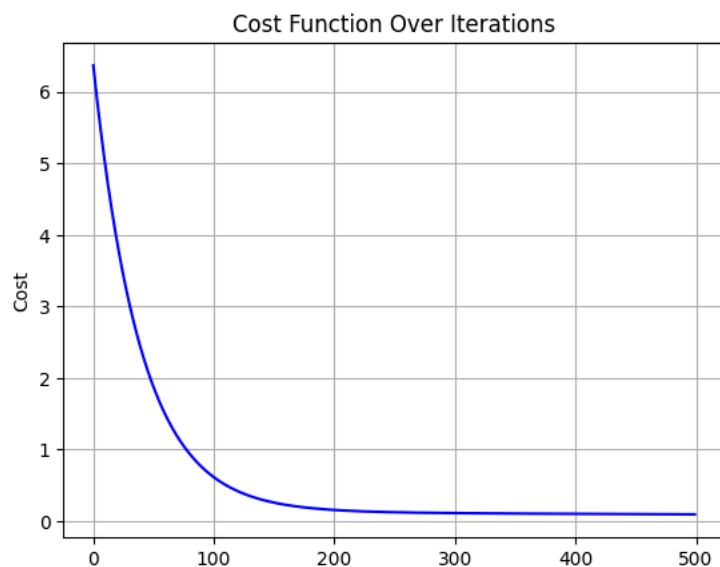
✓ Batch GD

```

1 def batch_gradient_descent(X, y, theta, learning_rate, num_iterations):
2     m = len(y)
3     cost_history = np.zeros(num_iterations)
4     for i in range(num_iterations):
5         predictions = X.dot(theta)
6         errors = predictions - y
7         gradients = (1 / m) * X.T.dot(errors)
8         theta = theta - learning_rate * gradients
9         cost_history[i] = compute_cost(X, y, theta)
10    return theta, cost_history
11
12 theta, cost_history = batch_gradient_descent(X, y, theta, learning_rate, num_iterations)
13
14 print("Theta:", theta)
15 print("Final cost:", cost_history[-1])
16
17 plt.plot(range(num_iterations), cost_history, color='blue')
18 plt.xlabel('Number of iterations')
19 plt.ylabel('Cost')
20 plt.title('Cost Function Over Iterations')
21 plt.grid(True)
22 plt.show()

```

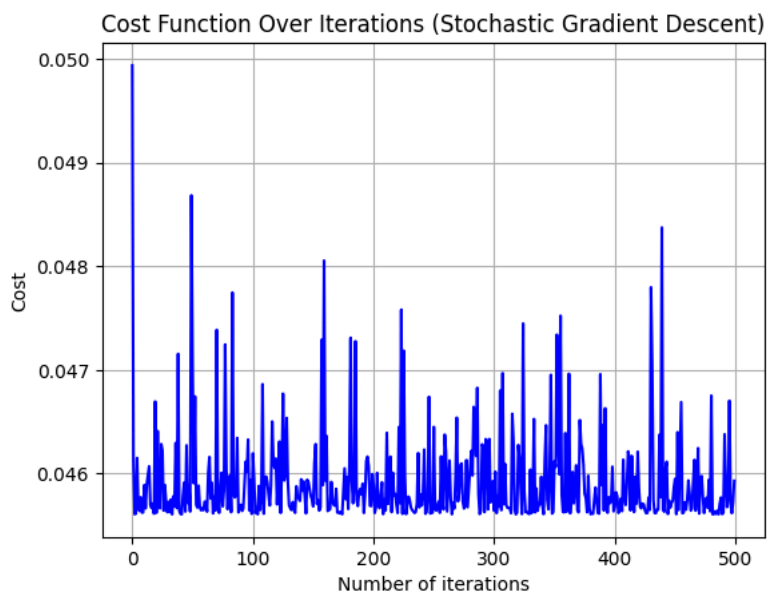
→ Theta: [2.56986733 1.91440276]
 Final cost: 0.09499329707514159



✓ SGD

```
1 def stochastic_gradient_descent(X, y, theta, learning_rate, num_iterations):
2     m = len(y)
3     cost_history = np.zeros(num_iterations)
4
5     for i in range(num_iterations):
6         for j in range(m):
7             index = np.random.randint(m)
8             X_i = X[index:index + 1]
9             y_i = y[index:index + 1]
10
11             predictions = X_i.dot(theta)
12             errors = predictions - y_i
13             gradients = X_i.T.dot(errors)
14             theta = theta - learning_rate * gradients
15
16         cost_history[i] = compute_cost(X, y, theta)
17
18     return theta, cost_history
19
20 theta, cost_history = stochastic_gradient_descent(X, y, theta, learning_rate, num_iterations)
21
22 print("Theta:", theta)
23 print("Final cost:", cost_history[-1])
24
25 plt.plot(range(num_iterations), cost_history, color='blue')
26 plt.xlabel('Number of iterations')
27 plt.ylabel('Cost')
28 plt.title('Cost Function Over Iterations (Stochastic Gradient Descent)')
29 plt.grid(True)
30 plt.show()
31
```

→ Theta: [1.96770767 3.00259837]
Final cost: 0.0459303724047596



✓ Mini Batch GD

```
1 def mini_batch_gradient_descent(X, y, theta, learning_rate, num_iterations, batch_size):
2     m = len(y)
3     cost_history = np.zeros(num_iterations)
4     for i in range(num_iterations):
5         indices = np.random.permutation(m)
6         X_shuffled = X[indices]
7         y_shuffled = y[indices]
8         for start in range(0, m, batch_size):
9             end = min(start + batch_size, m)
10             X_mini_batch = X_shuffled[start:end]
11             y_mini_batch = y_shuffled[start:end]
12             predictions = X_mini_batch.dot(theta)
```

```

13         errors = predictions - y_mini_batch
14         gradients = (1 / len(y_mini_batch)) * X_mini_batch.T.dot(errors)
15         theta = theta - learning_rate * gradients
16         cost_history[i] = compute_cost(X, y, theta)
17     return theta, cost_history
18
19 batch_size = 64
20 theta, cost_history = mini_batch_gradient_descent(X, y, theta, learning_rate, num_iterations, batch_size)
21 print("Theta:", theta)
22 print("Final cost:", cost_history[-1])
23
24 plt.plot(range(num_iterations), cost_history, color='blue')
25 plt.xlabel('Number of iterations')
26 plt.ylabel('Cost')
27 plt.title('Cost Function Over Iterations (Mini-Batch Gradient Descent)')
28 plt.grid(True)
29 plt.show()

```

→ Theta: [1.99291503 3.00228372]
 Final cost: 0.04560672672255235

