ECGR 5105 Homework 1: Gradient Descent

Owen Bailey-Waltz (801488178)

```
# import required packages, load data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
from IPython.display import display

drive.mount('/content/drive/')
file_path = '/content/drive/MyDrive/datasets/D3.csv'
df = pd.DataFrame(pd.read_csv(file_path))
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount,
```

Problem 1: Linear regression, one explanatory variable

```
# assign the arrays used in single-variable fitting
X_1 = df.values[:, 0]
X_2 = df.values[:, 1]
X_3 = df.values[:, 2]
Y = df.values[:, 3]
```

Functions for single-variable gradient descent

The following functions are generic implementations of a single-variable gradient descent algorithm and loss computation designed for use with input arrays rather than matrices. The matrix implementations are featured in Problem 2 alongside the multi-variable implementation.

```
# functions for single-variable gradient descent
def compute_loss_single(X, y, theta):
  H = theta[1] * X + theta[0]
  sq_err = np.square(np.subtract(H, y))
  J = (1 / (2 * len(X))) * np.sum(sq_err)
  return J

def grad_desc_single(X, y, theta, alpha, N):
  m = len(y)
  loss_history = np.zeros(N)

  for i in range(N):
    H = theta[1] * X + theta[0]
    err = np.subtract(H, y)
    inc = [0, 0]
    inc[0] = (alpha / m) * sum(err)
    inc[1] = (alpha / m) * X.dot(err)
    theta[0] -= inc[0]
    theta[1] -= inc[1]
    loss_history[i] = compute_loss_single(X, y, theta)

  return theta, loss_history
```

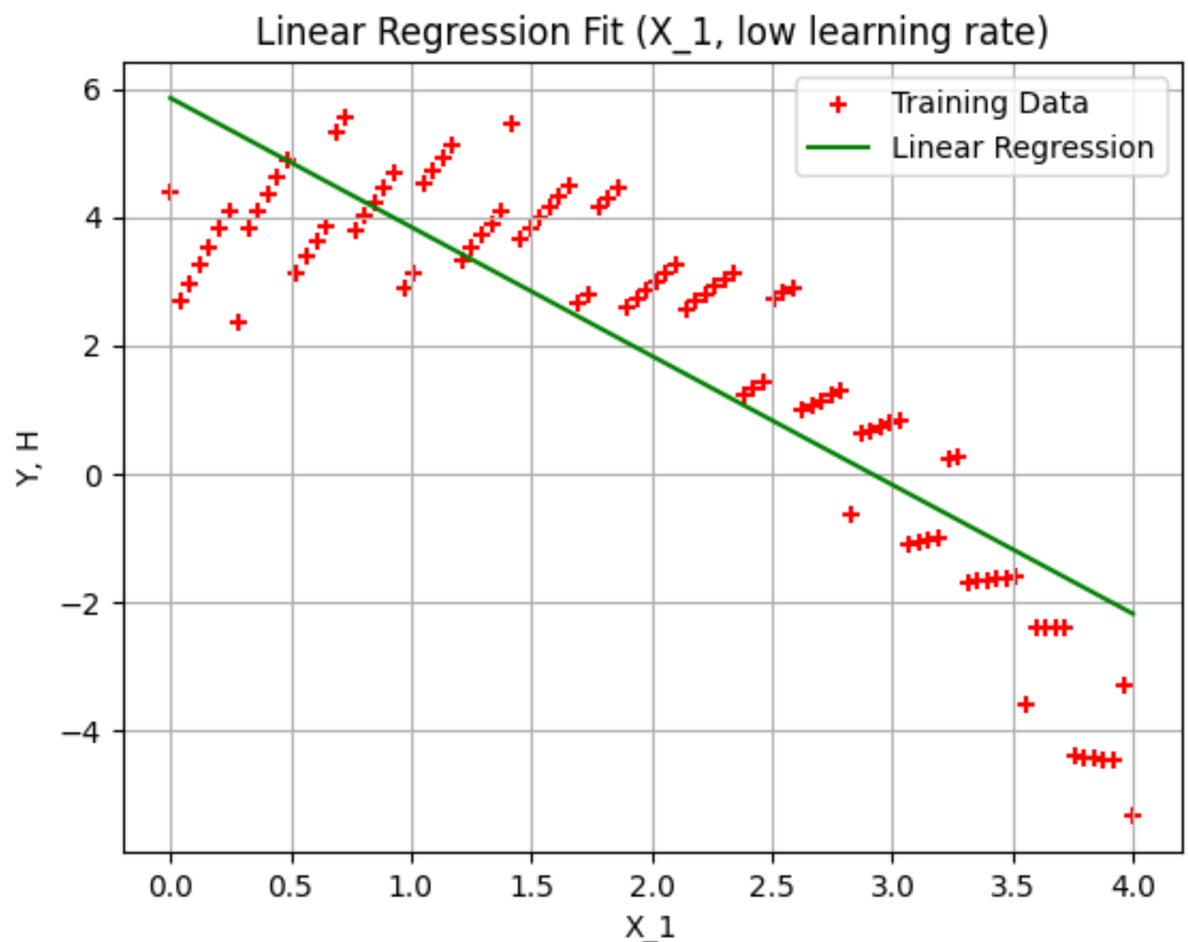
Y vs X_1

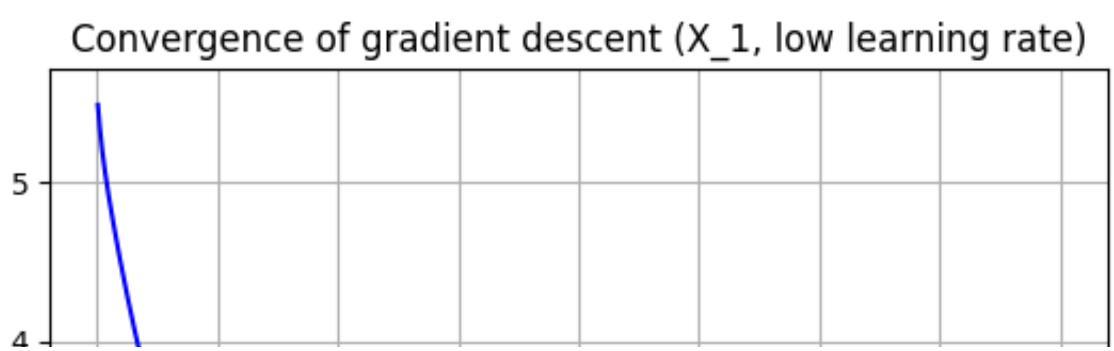
Low learning rate ($\alpha = 0.01$)

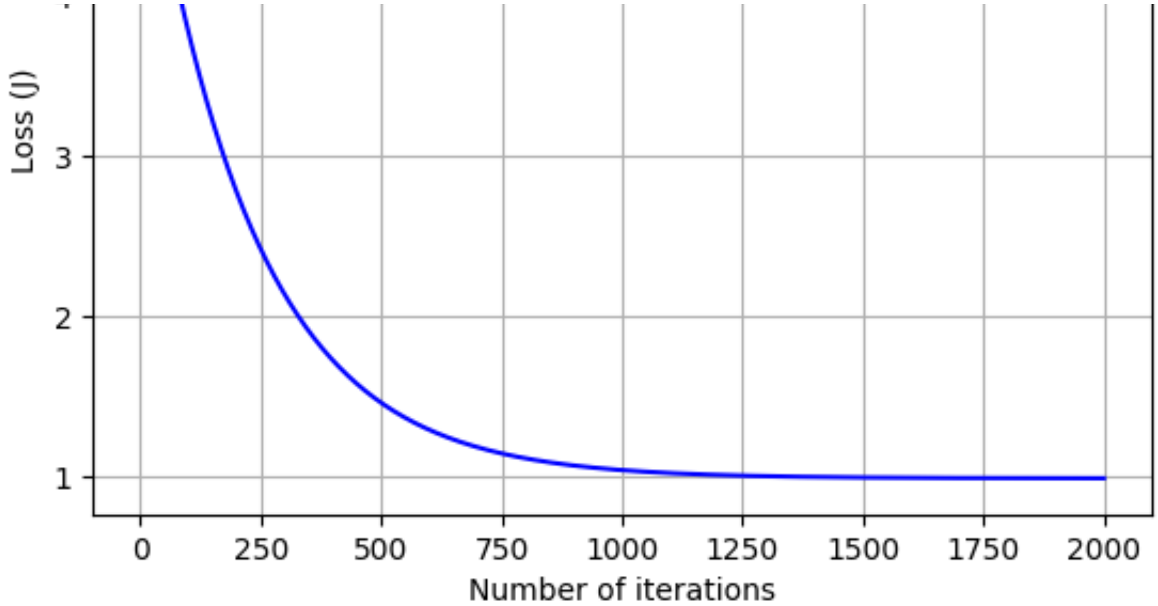
```
alpha = 0.01
theta = [0, 0]
N = 2000

# compute loss
J_1 = compute_loss_single(X_1, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_1, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_1, Y, color='red', marker='+', label='Training Data')
plt.plot(X_1, theta[1] * X_1 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
```

```
plt.xlabel('X_1')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_1, low learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_1, low learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(5.858868600540576), np.float64(-2.
Last loss value: 0.9856018973781949
```





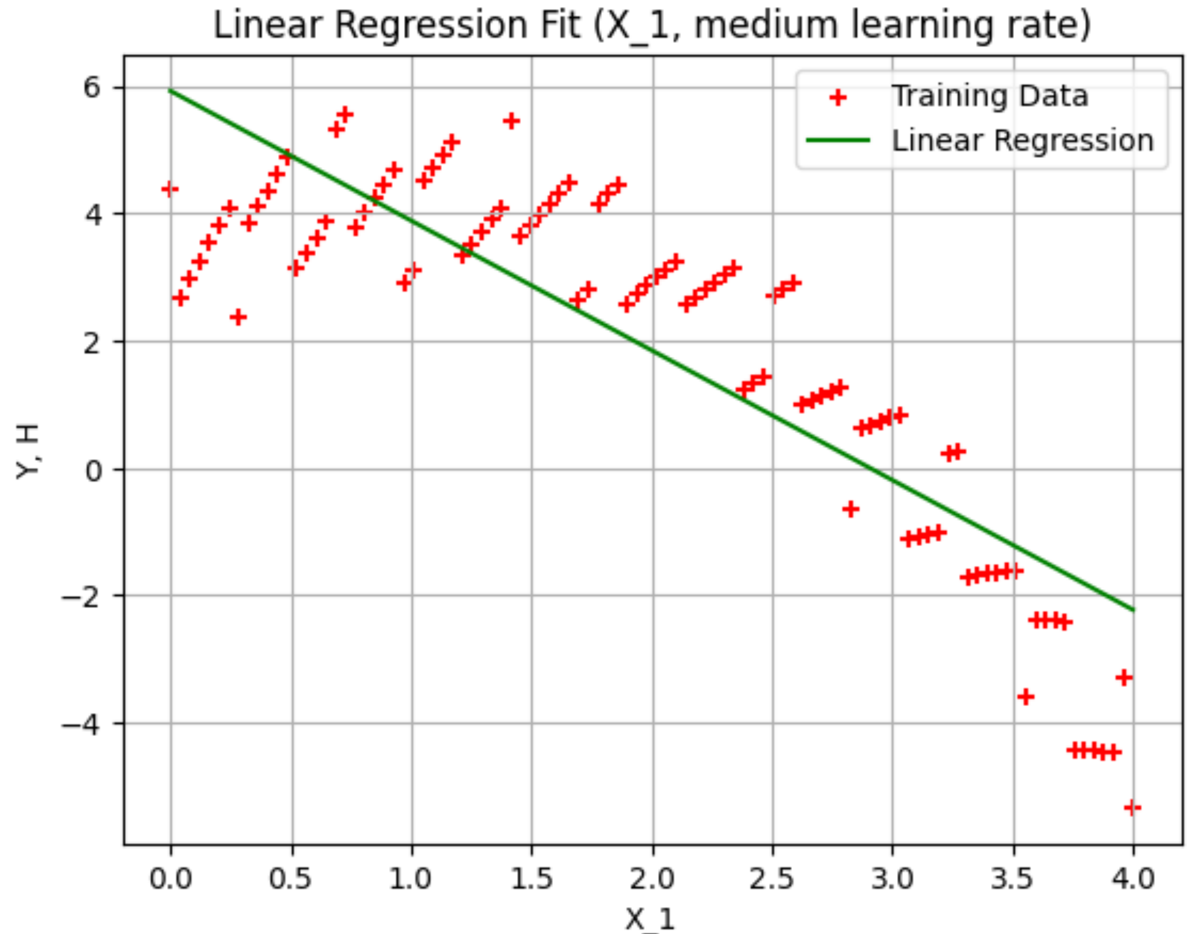


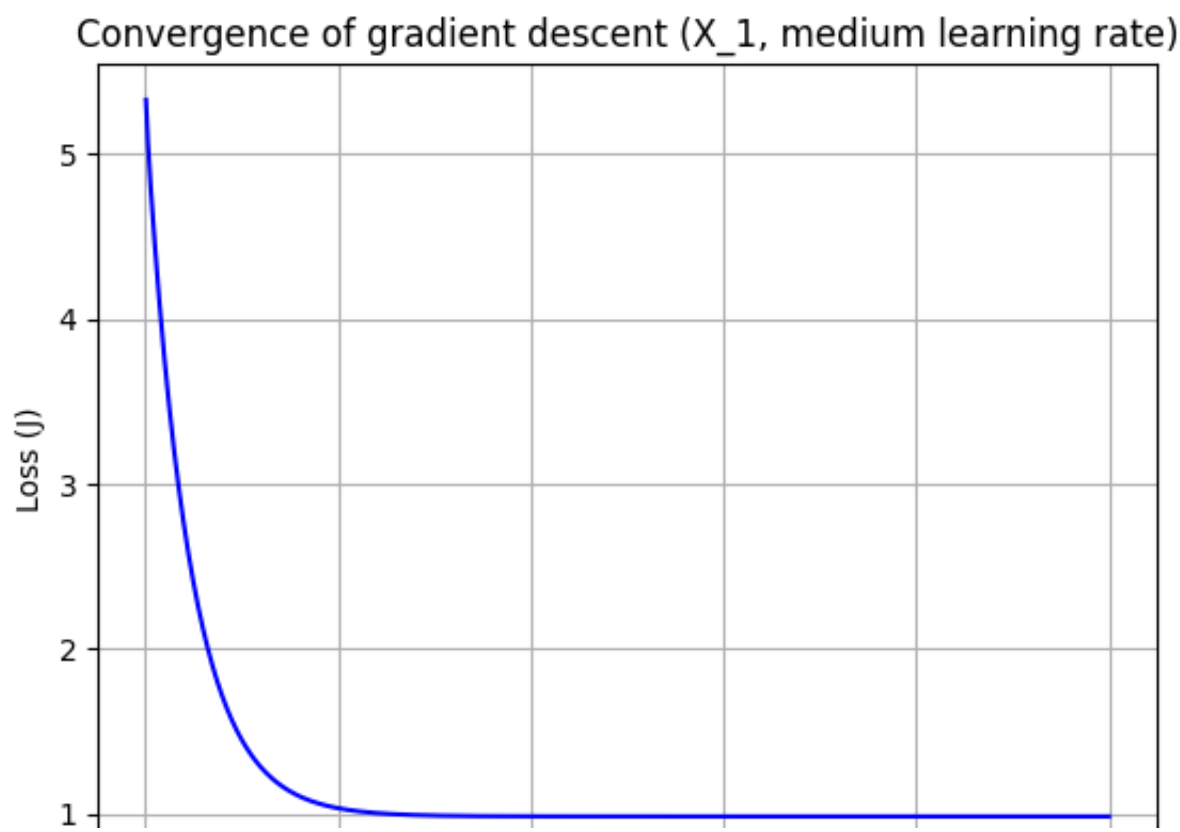
Medium learning rate ($\alpha\ = 0.05$)

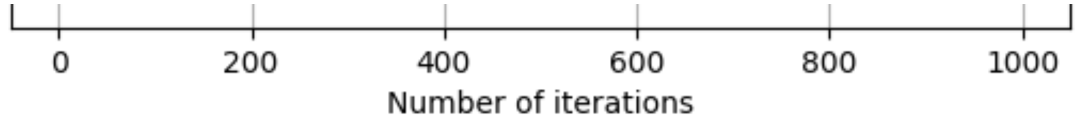
```
alpha = 0.05
theta = [0, 0]
N = 1000

# compute loss
J_1 = compute_loss_single(X_1, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_1, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_1, Y, color='red', marker='+', label='Training Data')
plt.plot(X_1, theta[1] * X_1 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_1')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_1, medium learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_1, medium learning rate)')
```

```
plt.show()
```

```
Final value of parameters: [np.float64(5.927864277730485), np.float64(-2.
Last loss value: 0.9849930834545563
```



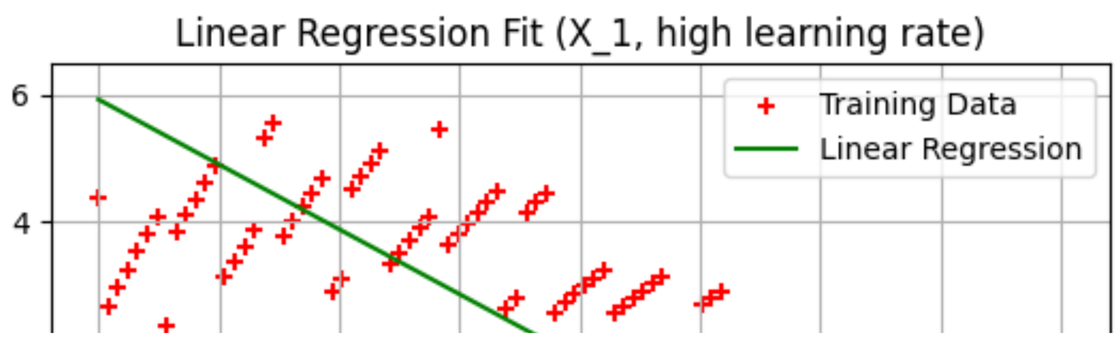


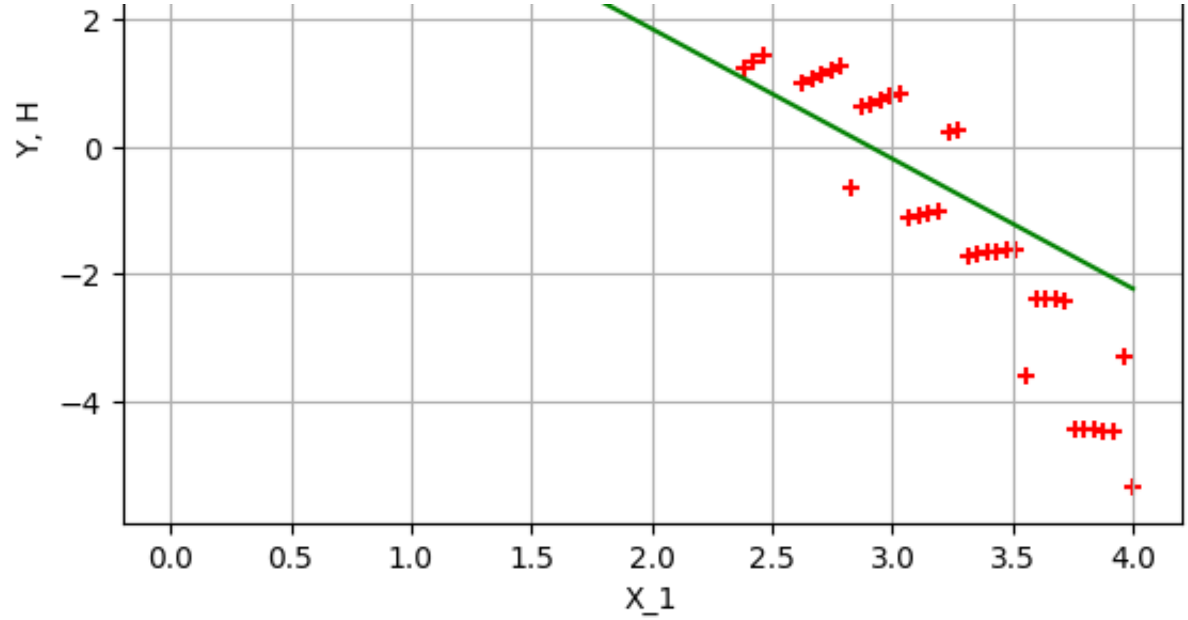


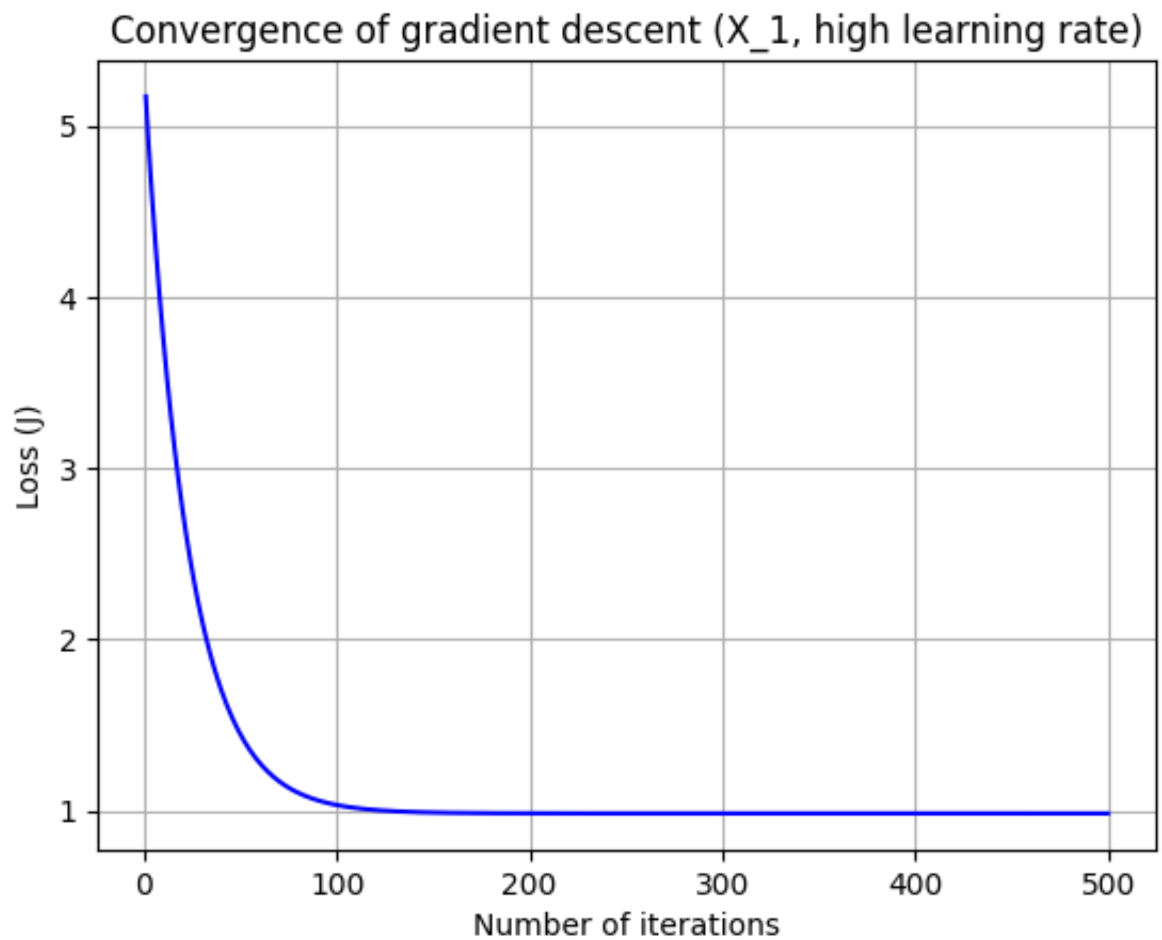
High learning rate ($\alpha = 0.1$)

```
alpha = 0.1
theta = [0, 0]
N = 500

# compute loss
J_1 = compute_loss_single(X_1, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_1, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_1, Y, color='red', marker='+', label='Training Data')
plt.plot(X_1, theta[1] * X_1 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_1')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_1, high learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_1, high learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(5.927869426861156), np.float64(-2.
Last loss value: 0.9849930833467424
```







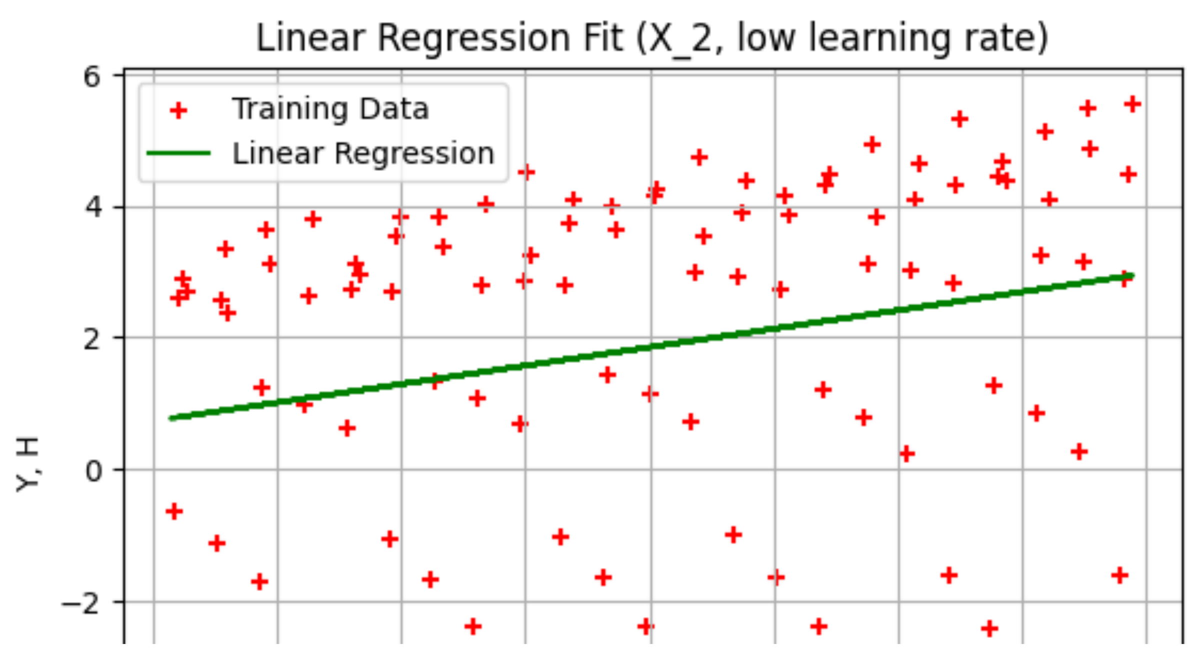
Y vs X_2

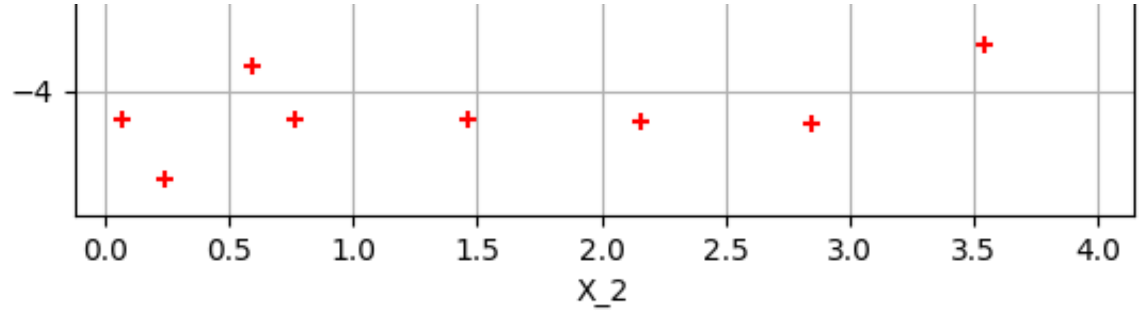
Low learning rate ($\alpha = 0.01$)

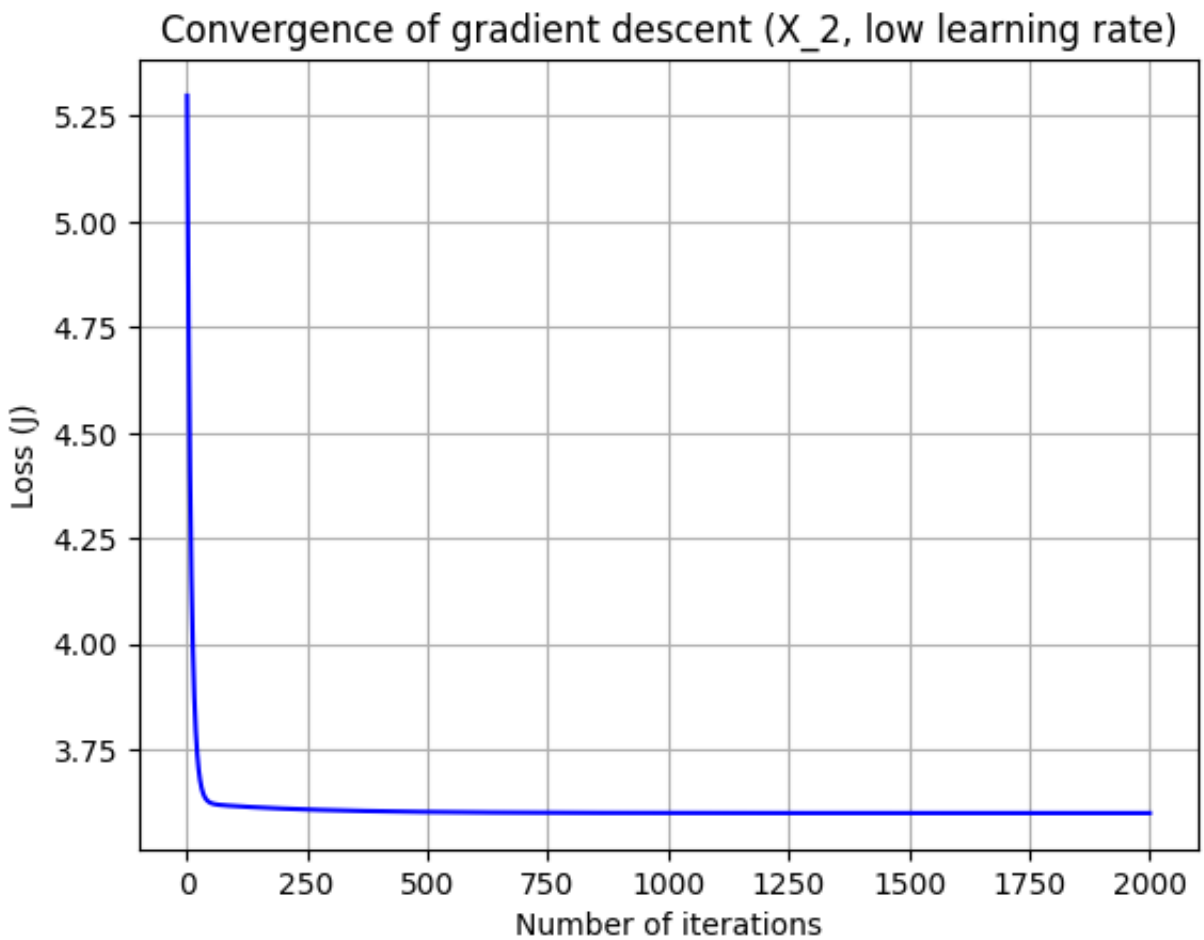
```
alpha = 0.01
```

```
alpha = 0.01
theta = [0, 0]
N = 2000

# compute loss
J_2 = compute_loss_single(X_2, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_2, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_2, Y, color='red', marker='+', label='Training Data')
plt.plot(X_2, theta[1] * X_2 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_2')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_2, low learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_2, low learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(0.7307249777814308), np.float64(0.
Last loss value: 3.599369649815171
```





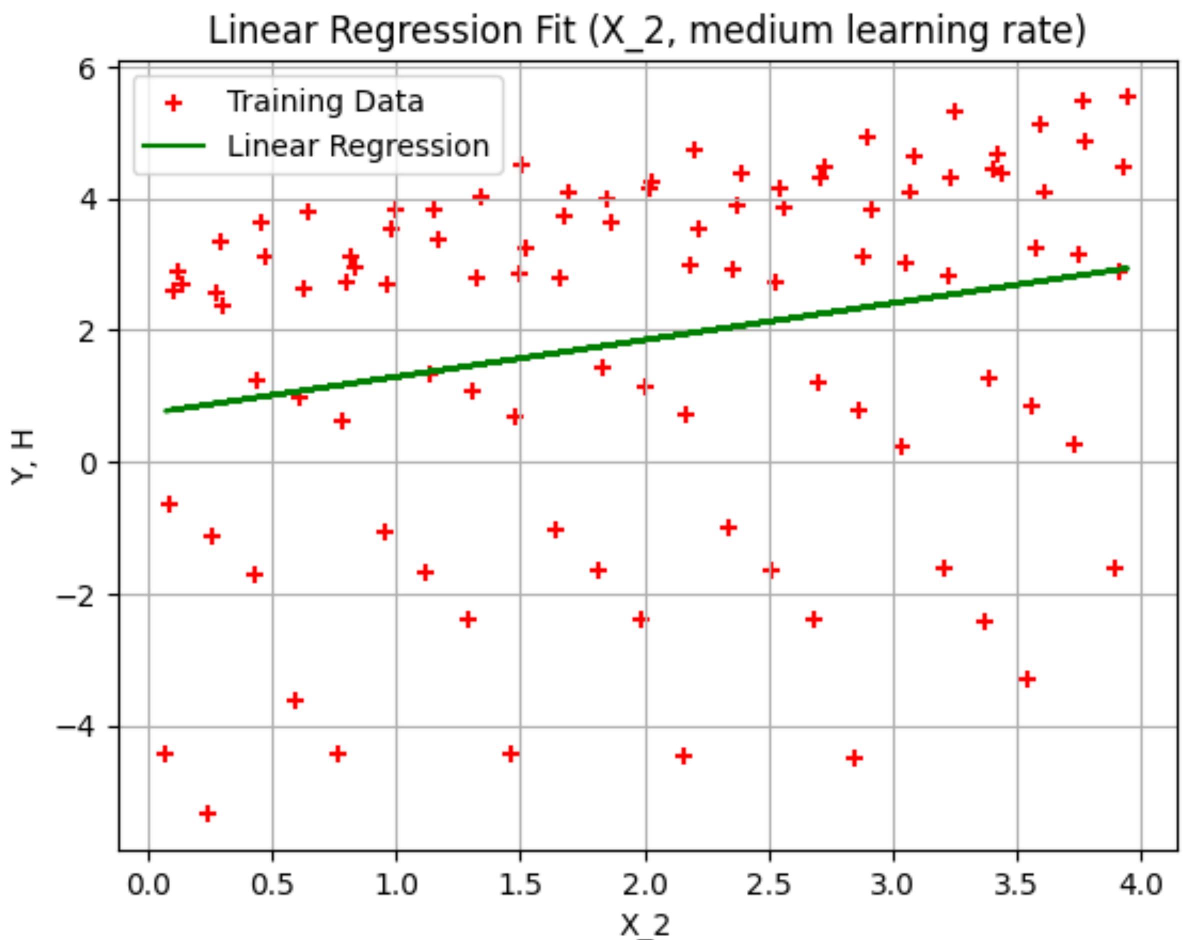


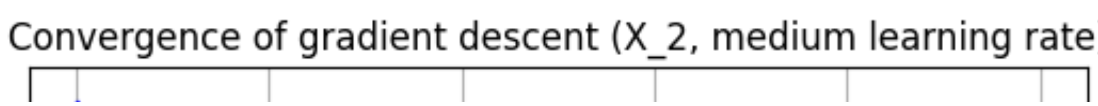
Medium learning rate ($\alpha = 0.05$)

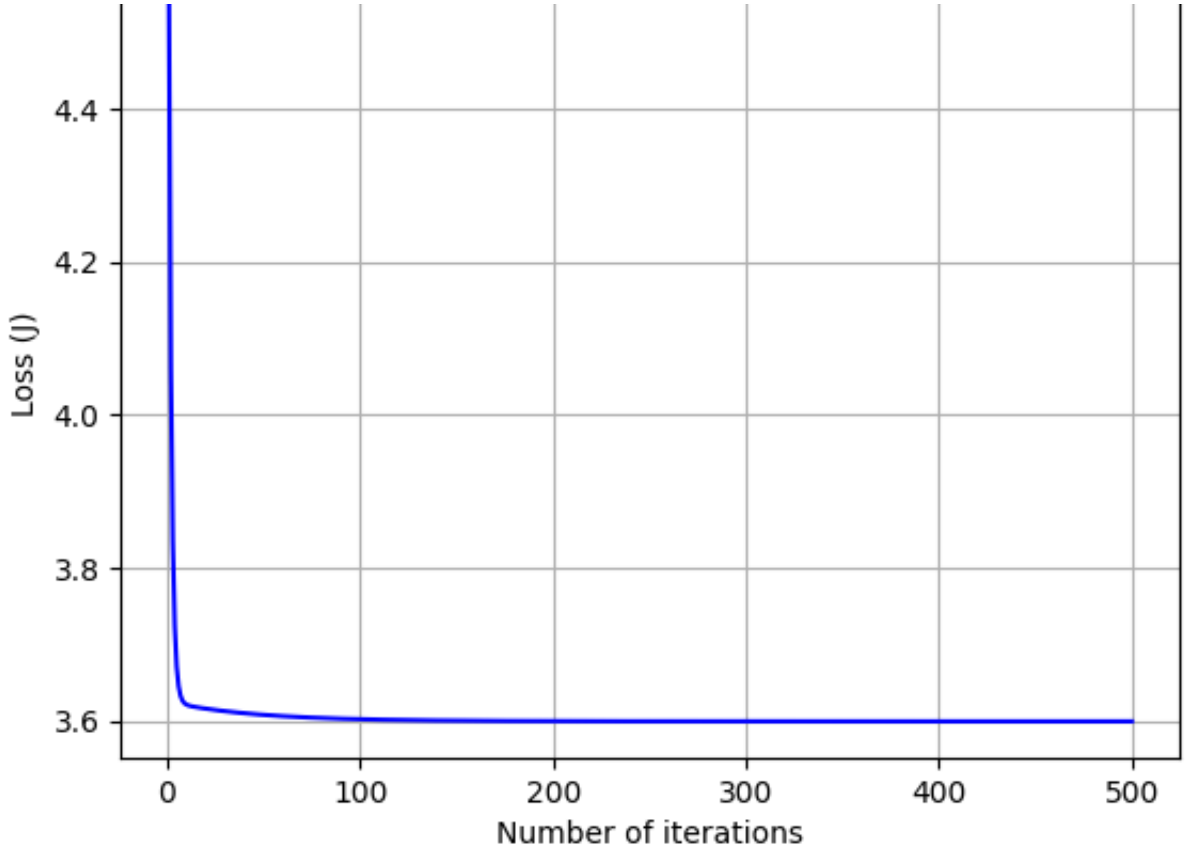
```
alpha = 0.05
theta = [0, 0]
N = 500

# compute loss
J_2 = compute_loss_single(X_2, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_2, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))
```

```
# plot the linear regression fit and the loss over time
plt.scatter(X_2, Y, color='red', marker='+', label='Training Data')
plt.plot(X_2, theta[1] * X_2 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_2')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_2, medium learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_2, medium learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(0.7343436185963006), np.float64(0.
Last loss value: 3.599366394184093
```





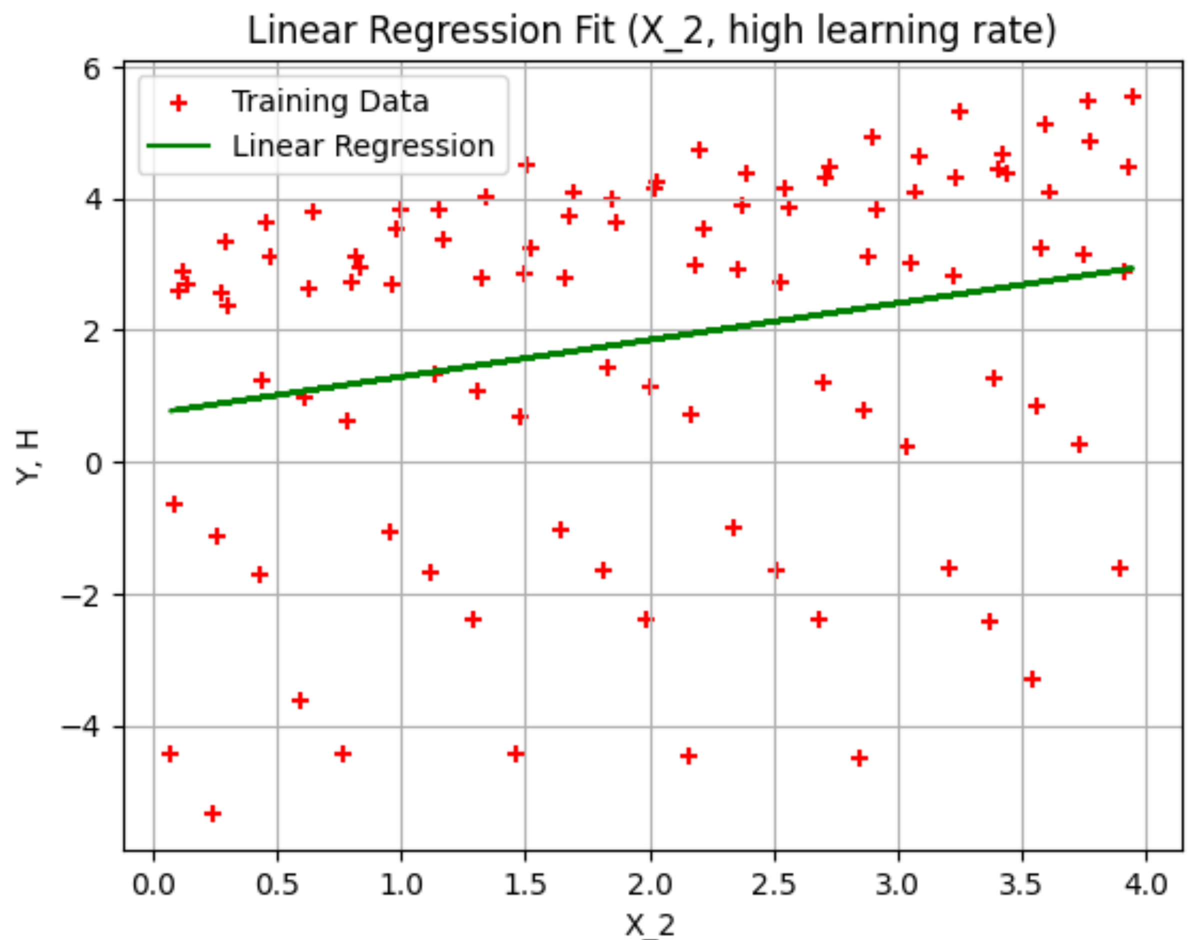


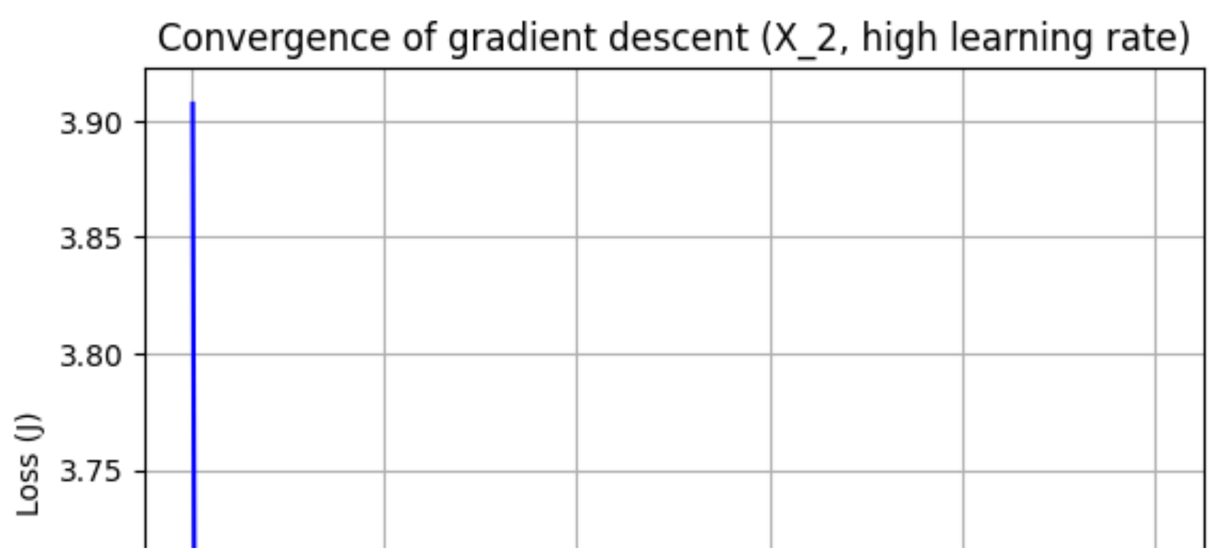
High learning rate ($\alpha = 0.1$)

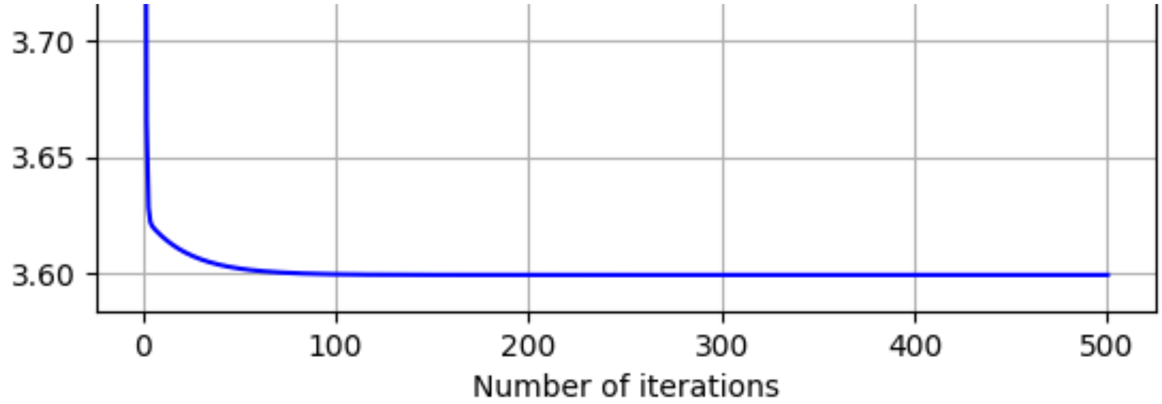
```
alpha = 0.1
theta = [0, 0]
N = 500

# compute loss
J_2 = compute_loss_single(X_2, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_2, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_2, Y, color='red', marker='+', label='Training Data')
plt.plot(X_2, theta[1] * X_2 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_2')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_2, high learning rate)')
plt.legend()
plt.show()
```

```
plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_2, high learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(0.7360542889293448), np.float64(0.
Last loss value: 3.599366018172853
```







Double-click (or enter) to edit

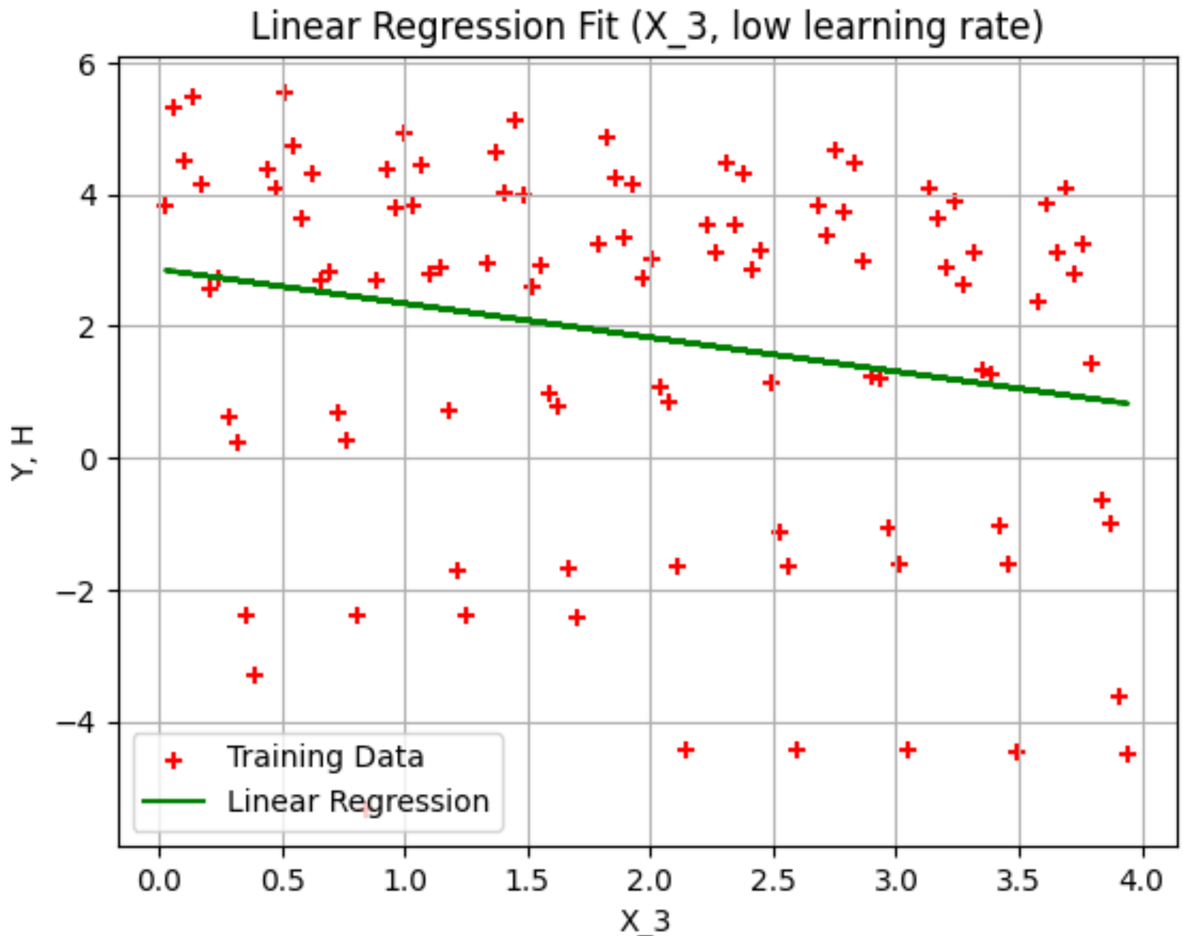
Y vs X_3

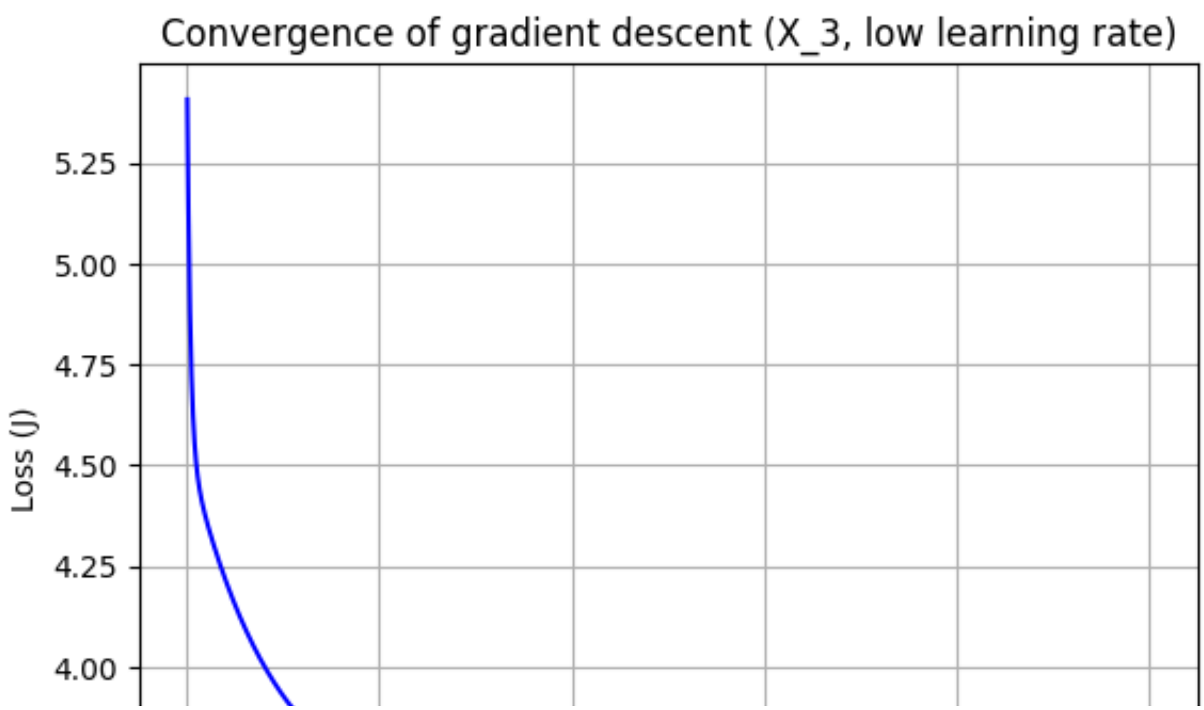
Low learning rate ($\alpha = 0.01$)

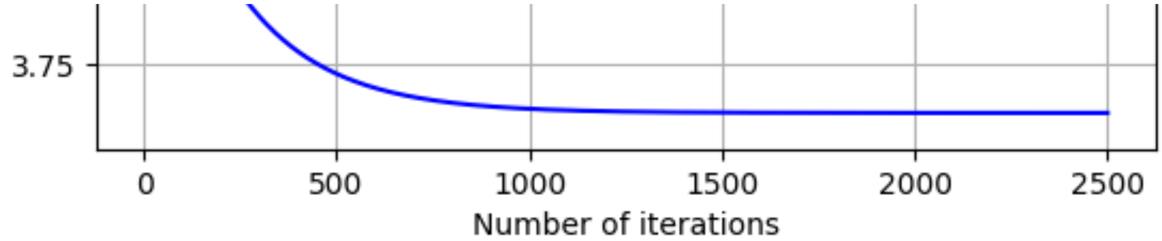
```
alpha = 0.01
theta = [0, 0]
N = 2500

# compute loss
J_3 = compute_loss_single(X_3, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_3, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_3, Y, color='red', marker='+', label='Training Data')
plt.plot(X_3, theta[1] * X_3 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_3')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_3, low learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
```

```
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_3, low learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(2.861845085919815), np.float64(-0.
Last loss value: 3.629463048034272
```



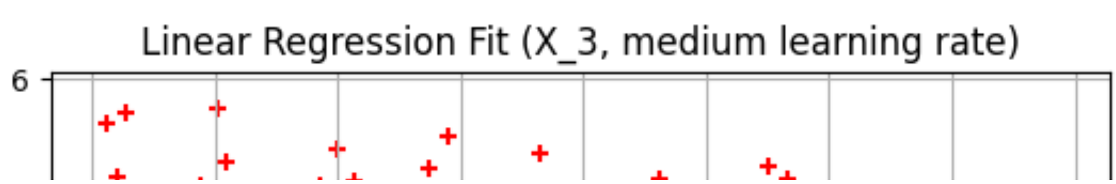


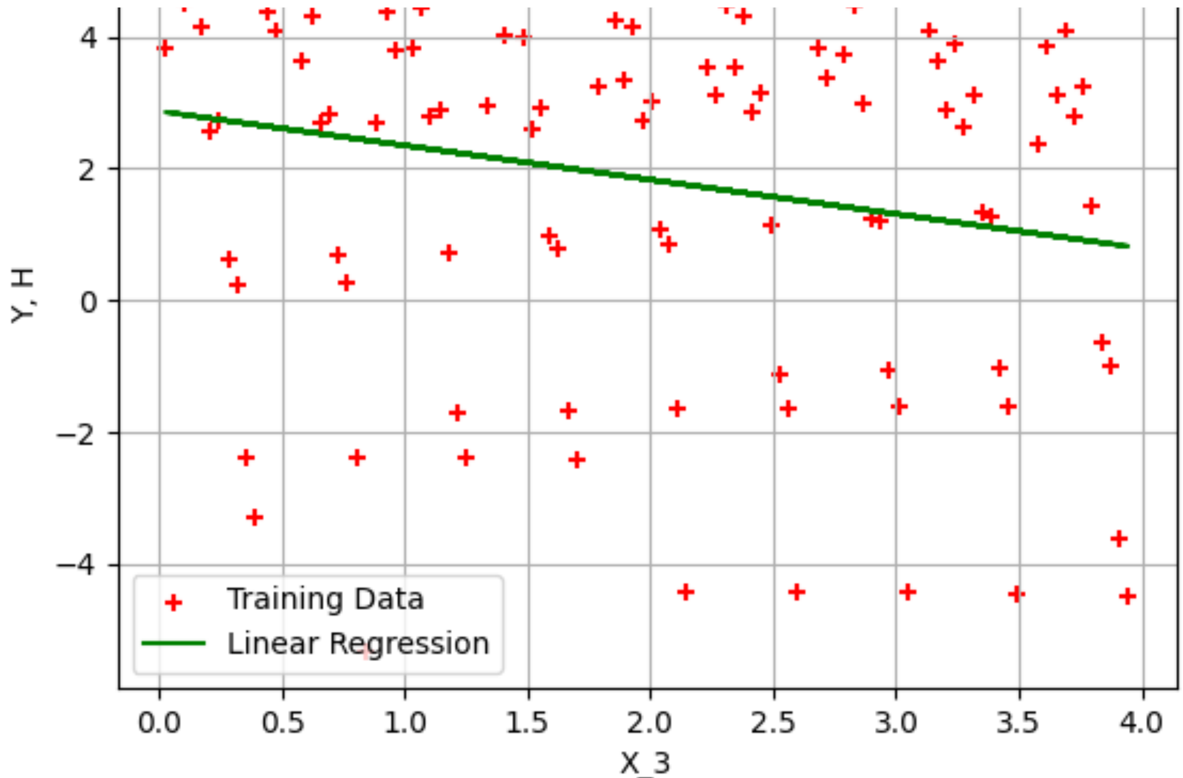


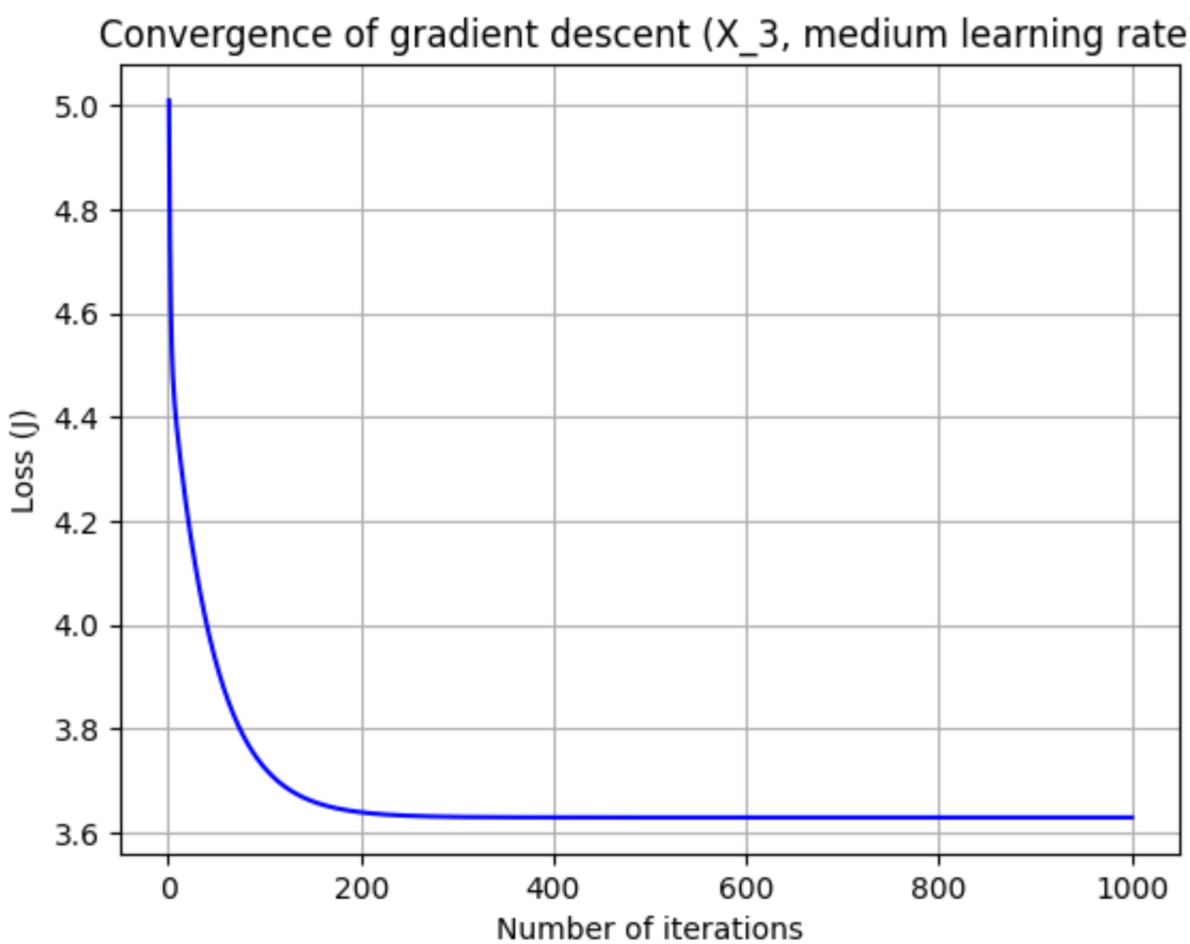
Medium learning rate ($\alpha\ = 0.05$)

```
alpha = 0.05
theta = [0, 0]
N = 1000

# compute loss
J_3 = compute_loss_single(X_3, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_3, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_3, Y, color='red', marker='+', label='Training Data')
plt.plot(X_3, theta[1] * X_3 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_3')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_3, medium learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_3, medium learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(2.8713893505679553), np.float64(-0
Last loss value: 3.629451124747377
```



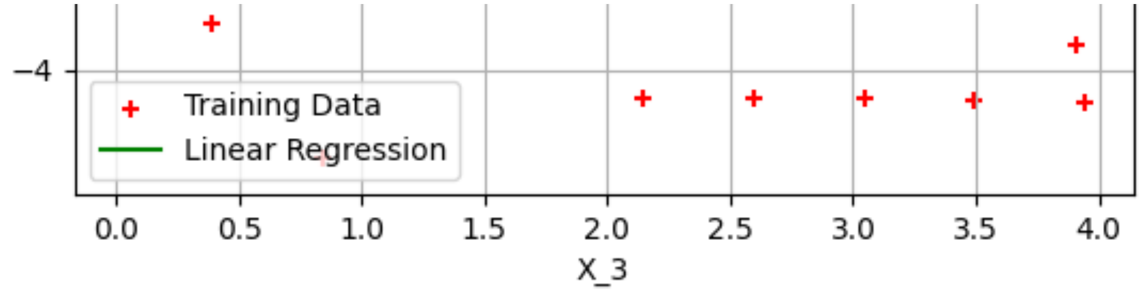


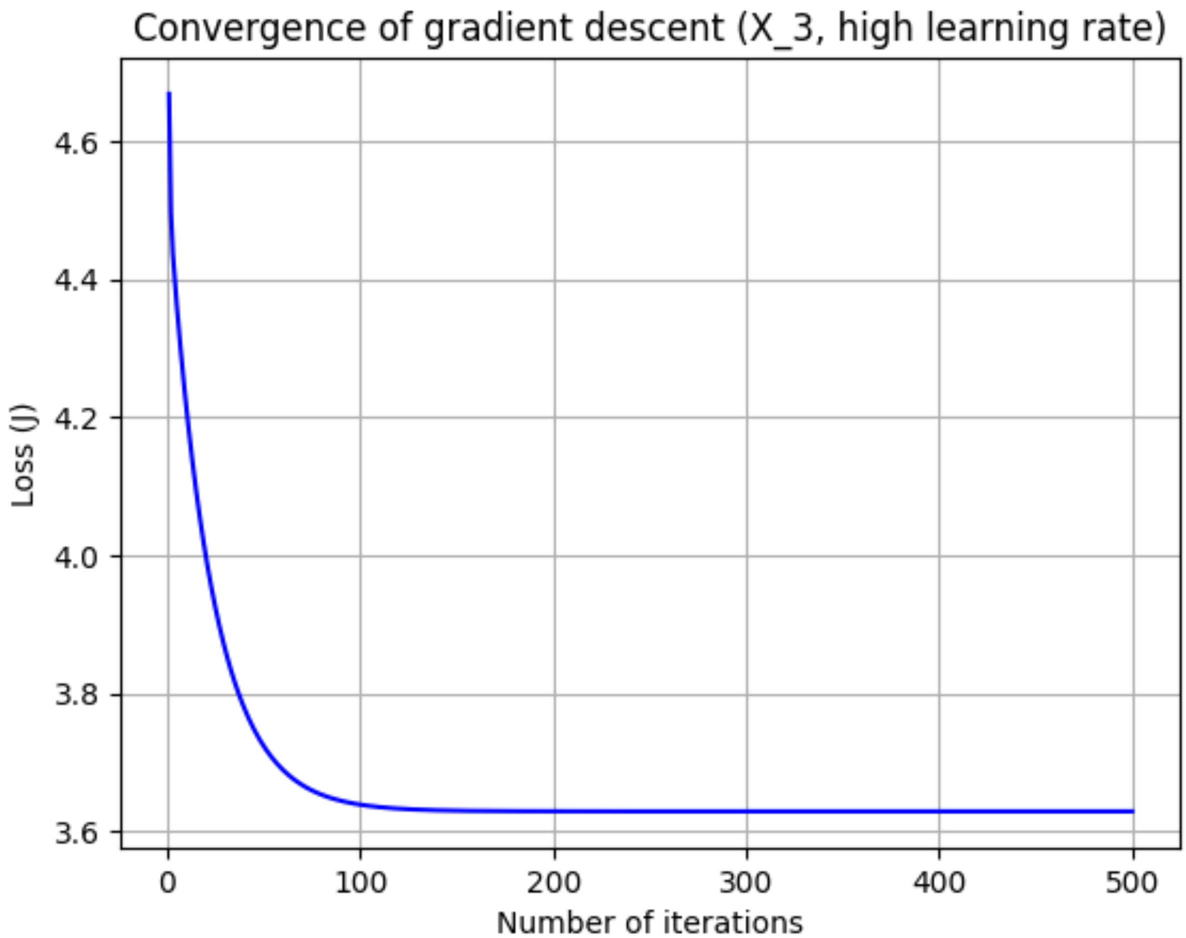


High learning rate ($\alpha = 0.1$)

```
alpha = 0.1
theta = [0, 0]
N = 500

# compute loss
J_3 = compute_loss_single(X_3, Y, theta)

# minimize loss and print new parameters
theta, loss_history = grad_desc_single(X_3, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the linear regression fit and the loss over time
plt.scatter(X_3, Y, color='red', marker='+', label='Training Data')
plt.plot(X_3, theta[1] * X_3 + theta[0], color='green',
         label='Linear Regression')
plt.grid(True)
plt.xlabel('X_3')
plt.ylabel('Y, H')
plt.title('Linear Regression Fit (X_3, high learning rate)')
plt.legend()
plt.show()

plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X_3, high learning rate)')
plt.show()
```

```
Final value of parameters: [np.float64(2.8713914006320294), np.float64(-0
Last loss value: 3.6294511247304646
```





Double-click (or enter) to edit

Problem 2: Linear regression, multiple explanatory variables

```
# assign the array used in multi-variable fitting (reshape existing arra
# horizontally stack them into our input matrix)
X_0 = np.ones((len(Y), 1))
X = np.hstack((X_0, X_1.reshape(len(X_1), 1), X_2.reshape(len(X_2), 1),
               X_3.reshape(len(X_3), 1)))
```

Functions for multi-variable gradient descent

The following functions are generic implementations of a multi-variable gradient descent algorithm and loss computation.

```
def compute_loss(X, y, theta):
  H = X.dot(theta)
  sq_err = np.square(np.subtract(H, y))
  J = (1 / (2 * len(X))) * np.sum(sq_err)
  return J

def grad_desc(X, y, theta, alpha, N):
  m = len(y)
  loss_history = np.zeros(N)

  for i in range(N):
    H = X.dot(theta)
    err = np.subtract(H, y)
    inc = (alpha / m) * X.transpose().dot(err)
    theta -= inc
    loss_history[i] = compute_loss(X, y, theta)

  return theta, loss_history
```

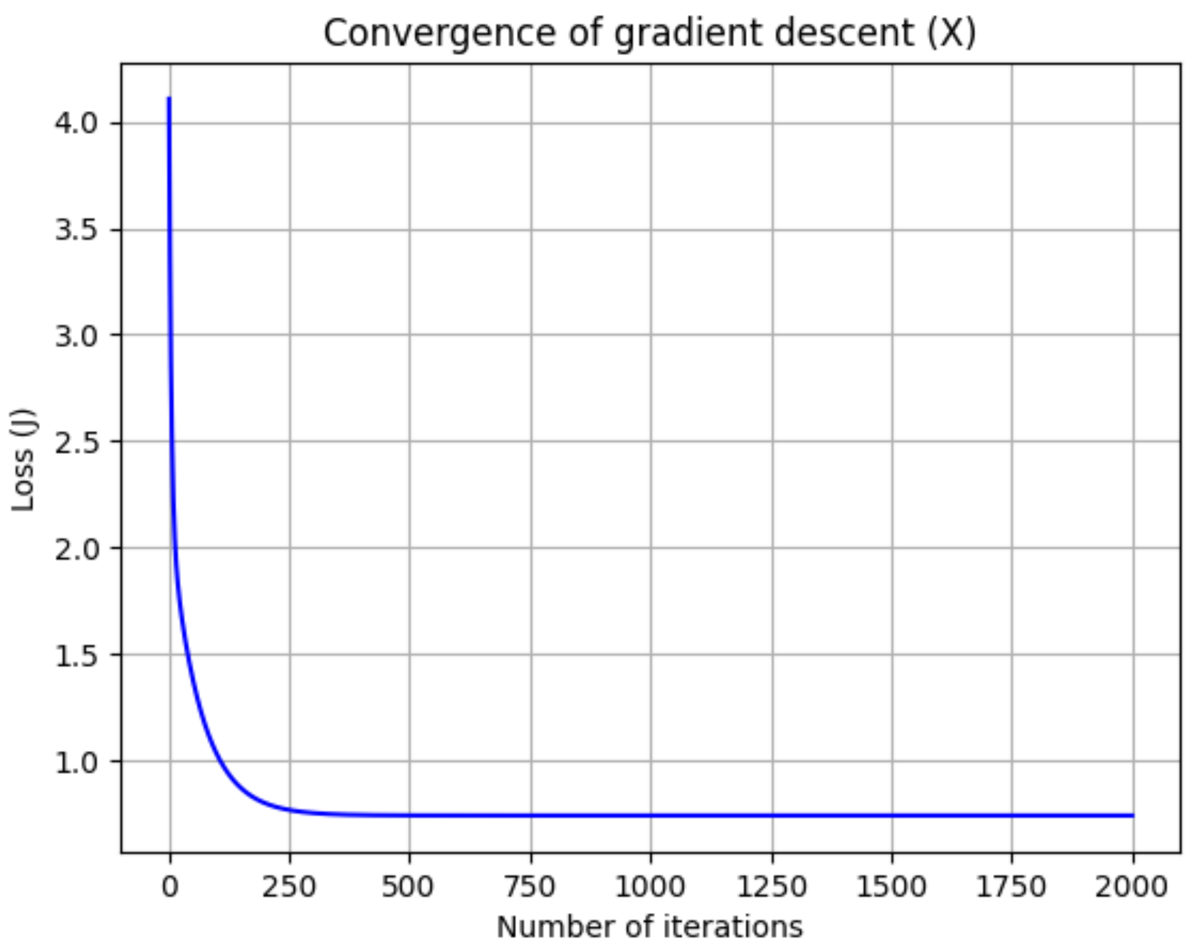
Multi-variable gradient descent implementation at several learning rates

Main submission model ($\alpha = 0.08$, n = 2000)

```
alpha = 0.08
theta = np.zeros(4)
N = 2000

# minimize loss and print new parameters
theta, loss_history = grad_desc(X, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the loss over time
plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
```

```
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X)')
plt.show()

# three training set points picked randomly, then predict some values
foo = np.array([1, 4, 0.24, 0.84]).transpose().dot(theta)
boo = np.array([1, 2.707070707, 2.001616162, 2.488484848]).transpose().d
soo = np.array([1, 1.01010101, 0.813737374, 3.652121212]).transpose().do
print("Predictions for training set inputs (4, 0.24, 0.84)"
      + " (2.707070707, 2.001616162, 2.488484848) & "
      + "(1.01010101, 0.813737374, 3.652121212)")
print("{} {} {}".format(foo, boo, soo))
print("Training set -5.332454989 1.139717238 3.110675304")
print()

foo = np.array([1, 1, 1, 1]).transpose().dot(theta)
boo = np.array([1, 2, 0, 4]).transpose().dot(theta)
soo = np.array([1, 3, 2, 1]).transpose().dot(theta)
print("Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)")
print("{} {} {}".format(foo, boo, soo))
```

```
Final value of parameters: [ 5.3141666  -2.00371919  0.53256343 -0.265601
Last loss value: 0.738464241568312
```



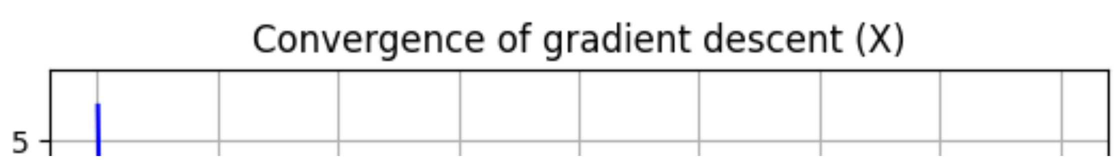

```
Predictions for training set inputs (4, 0.24, 0.84) (2.707070707, 2.00161
-2.7960004258807656 0.2949986427913852 2.753564690826601
```

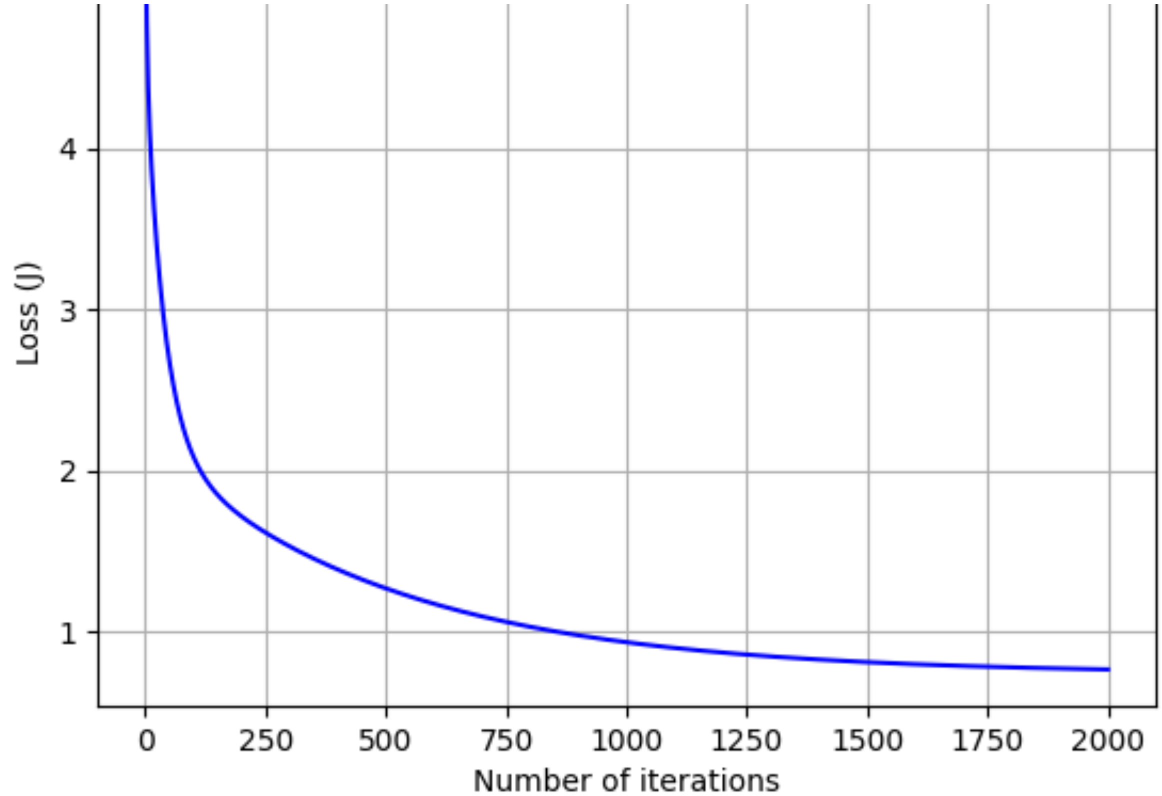
```
Training set -5.332454989 1.139717238 3.110675304

Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)
3.5774090589936436 0.24432109722649975 0.10253411574010246
```

Low learning rate ($\alpha = 0.01$)

```
alpha = 0.01
theta = np.zeros(4)
N = 2000

# minimize loss and print new parameters
theta, loss_history = grad_desc(X, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the loss over time
plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X)')
plt.show()

# three training set points picked randomly, then predict some values
foo = np.array([1, 4, 0.24, 0.84]).transpose().dot(theta)
boo = np.array([1, 2.707070707, 2.001616162, 2.488484848]).transpose().d
soo = np.array([1, 1.01010101, 0.813737374, 3.652121212]).transpose().do
print("Predictions for training set inputs (4, 0.24, 0.84)"
      + " (2.707070707, 2.001616162, 2.488484848) & "
      + "(1.01010101, 0.813737374, 3.652121212)")
print("{} {} {}".format(foo, boo, soo))
print("Training set -5.332454989 1.139717238 3.110675304")
print()

foo = np.array([1, 1, 1, 1]).transpose().dot(theta)
boo = np.array([1, 2, 0, 4]).transpose().dot(theta)
soo = np.array([1, 3, 2, 1]).transpose().dot(theta)
print("Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)")
print("{} {} {}".format(foo, boo, soo))
```

```
Final value of parameters: [ 4.60784132 -1.90393905  0.64927931 -0.162068
Last loss value: 0.7650394625052138
```






```
Predictions for training set inputs (4, 0.24, 0.84) (2.707070707, 2.00161
-2.9882256930544124 0.35004576276776844 2.621118312084609
Training set -5.332454989 1.139717238 3.110675304

Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)
3.1911127254543454 0.15168781022234323 0.03251393005171718
```

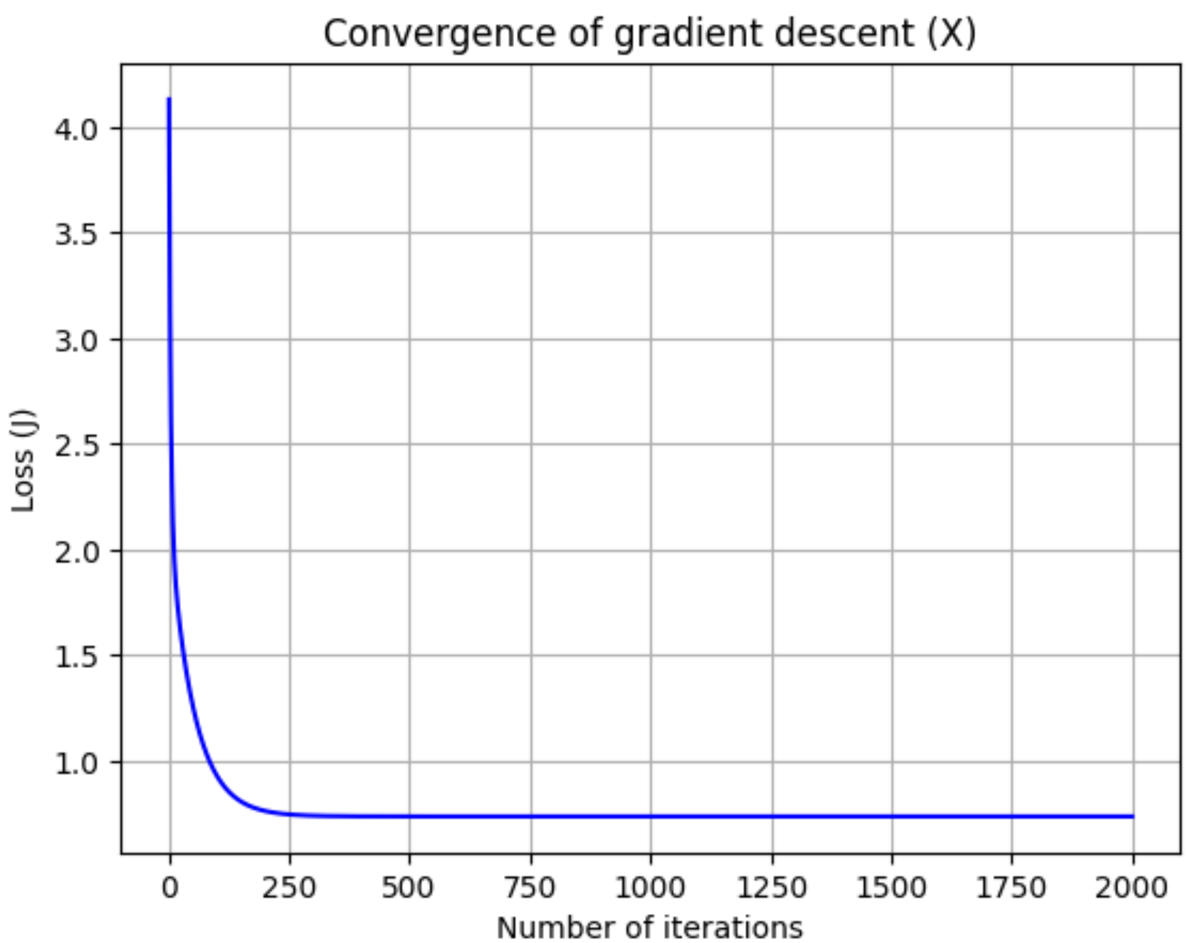
Double-click (or enter) to edit

High learning rate ($\alpha = 0.1$)

```
alpha = 0.1
theta = np.zeros(4)
N = 2000

# minimize loss and print new parameters
theta, loss_history = grad_desc(X, Y, theta, alpha, N)
print("Final value of parameters: {}".format(theta))
print("Last loss value: {}".format(loss_history[len(loss_history) - 1]))

# plot the loss over time
plt.plot(range(1, N + 1), loss_history, color='blue')
plt.grid(True)
plt.xlabel('Number of iterations')
plt.ylabel('Loss (J)')
plt.title('Convergence of gradient descent (X)')
```

```
plt.show()

# three training set points picked randomly, then predict some values
foo = np.array([1, 4, 0.24, 0.84]).transpose().dot(theta)
boo = np.array([1, 2.707070707, 2.001616162, 2.488484848]).transpose().d
soo = np.array([1, 1.01010101, 0.813737374, 3.652121212]).transpose().do
print("Predictions for training set inputs (4, 0.24, 0.84)"
      + " (2.707070707, 2.001616162, 2.488484848) & "
      + "(1.01010101, 0.813737374, 3.652121212)")
print("{} {} {}".format(foo, boo, soo))
print("Training set -5.332454989 1.139717238 3.110675304")
print()

foo = np.array([1, 1, 1, 1]).transpose().dot(theta)
boo = np.array([1, 2, 0, 4]).transpose().dot(theta)
soo = np.array([1, 3, 2, 1]).transpose().dot(theta)
print("Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)")
print("{} {} {}".format(foo, boo, soo))
```

```
Final value of parameters: [ 5.31416716 -2.00371927  0.53256334 -0.265601
Last loss value: 0.7384642415682942
```




```
Predictions for training set inputs (4, 0.24, 0.84) (2.707070707, 2.00161
-2.796000271789023 0.294998598664478 2.7535647969983517
Training set -5.332454989 1.139717238 3.110675304
```

```
Predictions for inputs (1, 1, 1) (2, 0, 4) & (3, 2, 1)
3.5774093686567574 0.24432117148325472 0.10253417186972863
```