Name: Rezwan Ahmad ID:20-44273-3 Section: C  
**Linear regression and gradient descent Assignment**

simple.py:

## Simple Regression Exercise

import argparse

import sys

import numpy as np

from matplotlib import pyplot as plt

import numpy.linalg as la

# Compute the sample mean and standard deviations for each feature (column)

# across the training examples (rows) from the data matrix X.

def mean\_std(X):

  mean = np.mean(X, axis=0)

  std = np.std(X, axis=0)

  return mean, std

# Standardize the features of the examples in X by subtracting their mean and

# dividing by their standard deviation, as provided in the parameters.

def standardize(X, mean, std):

  S = (X - mean) / std

  return S

# Read data matrix X and labels t from text file.

def read\_data(file\_name):

  data=np.loadtxt(file\_name)

  X = data[:, 0:1]  # first column

  t = data[:, 1]  # second column

  return X, t

# Implement gradient descent algorithm to compute w = [w0, w1].

def train(X, t, eta, epochs):

  costs=[]

  ep=[]

  w = np.zeros(X.shape[1])

  #  Use 'compute\_gradient' function below to find gradient of cost function and update w each epoch.

  #  Compute and append cost and epoch number to variables costs and ep every 10 epochs.

  for epoch in range(epochs):

    gradient = compute\_gradient(X, t, w)

    w = w - eta \* gradient

    if epoch % 10 == 0:

      cost = np.sum((X @ w - t) \*\* 2) / (2 \* len(X))

      costs.append(cost)

      ep.append(epoch)

  return w,ep,costs

# Compute RMSE on dataset (X, t).

def compute\_rmse(X, t, w):

  predictions = X @ w

  errors = predictions - t

  squared\_errors = errors \*\* 2

  mean\_squared\_error = np.mean(squared\_errors)

  rmse = np.sqrt(mean\_squared\_error)

  return rmse

# Compute objective function (cost) on dataset (X, t).

def compute\_cost(X, t, w):

  predictions = X @ w

  errors = predictions - t

  squared\_errors = errors \*\* 2

  cost = np.mean(squared\_errors) / 2

  return cost

# Compute gradient of the objective function (cost) on dataset (X, t).

def compute\_gradient(X, t, w):

  grad = np.zeros(w.shape)

  predictions = X @ w

  errors = predictions - t

  grad = X.T @ errors / len(X)

  return grad

# BONUS: Implement stochastic gradient descent algorithm to compute w = [w0, w1].

def train\_SGD(X, t, eta, epochs):

#  YOUR CODE here:

  costs=[]

  ep=[]

  w = np.zeros(X.shape[1])

  #  YOUR CODE here. Implement stochastic gradient descent to compute w for given epochs.

  #  Compute and append cost and epoch number to variables costs and ep every 10 epochs.

  return w,ep,costs

##======================= Main program =======================##

parser = argparse.ArgumentParser('Simple Regression Exercise.')

parser.add\_argument('-i', '--input\_data\_dir',

                    type=str,

                    default='D:\AIUB\Academics\Semester 10\ML\Final Assignment\Part 1\linear\_regression\data\simple',

                    #default='../data/simple/',  #this line is not working

                    help='Directory for the simple houses dataset.')

FLAGS, unparsed = parser.parse\_known\_args()

# Read the training and test data.

Xtrain, ttrain = read\_data(FLAGS.input\_data\_dir + "/train.txt")

Xtest, ttest = read\_data(FLAGS.input\_data\_dir + "/test.txt")

# Compute the mean and standard deviation of the training data.

mean, std = mean\_std(Xtrain)

# Standardize the training and test features using the mean and std computed over training.

Xtrain = standardize(Xtrain, mean, std)

Xtest = standardize(Xtest, mean, std)

# Add the bias feature (a column of ones) as the first column of the training and test examples.

Xtrain = np.hstack((np.ones((Xtrain.shape[0], 1)), Xtrain))

Xtest = np.hstack((np.ones((Xtest.shape[0], 1)), Xtest))

# Computing parameters for each training method for eta=0.1 and 200 epochs

eta=0.1

epochs=200

w,eph,costs=train(Xtrain,ttrain,eta,epochs)

#wsgd,ephsgd,costssgd=train\_SGD(Xtrain,ttrain,eta,epochs)

# Print model parameters.

print('Params GD: ', w)

#print('Params SGD: ', wsgd)

# Print cost and RMSE on training data.

print('Training RMSE: %0.2f.' % compute\_rmse(Xtrain, ttrain, w))

print('Training cost: %0.2f.' % compute\_cost(Xtrain, ttrain, w))

# Print cost and RMSE on test data.

print('Test RMSE: %0.2f.' % compute\_rmse(Xtest, ttest, w))

print('Test cost: %0.2f.' % compute\_cost(Xtest, ttest, w))

# Plotting epochs vs. cost for gradient descent methods

plt.figure(figsize=(10, 6))

plt.xlabel('epochs')

plt.ylabel('cost')

plt.yscale('log')

plt.plot(eph, costs, 'bo-', label='train\_jw\_gd')

plt.legend()

plt.savefig('gd\_cost\_simple.png')

plt.show()

plt.close()

# Plotting linear approximation for each training method

plt.figure(figsize=(10, 6))

plt.xlabel('Floor sizes')

plt.ylabel('House prices')

plt.plot(Xtrain[:, 1], ttrain, 'bo', label='Training data')  # Xtrain[:, 1] to exclude the bias feature

plt.plot(Xtest[:, 1], ttest, 'g^', label='Test data')  # Xtest[:, 1] to exclude the bias feature

plt.plot(Xtrain[:, 1], w[0] + w[1]\*Xtrain[:, 1], 'b', label='GD')  # Xtrain[:, 1] to exclude the bias feature

plt.legend()

plt.savefig('train-test-line.png')

plt.show()

plt.close()

multiple.py:

## Multiple Regression Exercise

import argparse

import sys

import numpy as np

from matplotlib import pyplot as plt

import numpy.linalg as la

# Compute the sample mean and standard deviations for each feature (column)

# across the training examples (rows) from the data matrix X.

def mean\_std(X):

  mean = np.zeros(X.shape[1])

  std = np.ones(X.shape[1])

  ## Your code here. Hint: You can use numpy to compute mean and std.

  mean = np.mean(X, axis=0)

  std = np.std(X, axis=0)

  return mean, std

# Standardize the features of the examples in X by subtracting their mean and

# dividing by their standard deviation, as provided in the parameters.

def standardize(X, mean, std):

  S = np.zeros(X.shape)

  S = (X - mean) / std

  return S

# Read data matrix X and labels t from text file.

def read\_data(file\_name):

  data=np.loadtxt(file\_name)

  X = data[:, :3]

  t = data[:, 3]

  return X, t

# Implement gradient descent algorithm to compute w = [w0, w1, ..].

def train(X, t, eta, epochs):

  costs=[]

  ep=[]

  w = np.zeros(X.shape[1])

  for epoch in range(epochs):

    grad = compute\_gradient(X, t, w)

    w = w - eta \* grad

    if epoch % 10 == 0:

      cost = compute\_cost(X, t, w)

      costs.append(cost)

      ep.append(epoch)

  return w,ep,costs

# Compute RMSE on dataset (X, t).

def compute\_rmse(X, t, w):

  N = X.shape[0]

  rmse = np.sqrt(1/N \* np.sum((t - X.dot(w))\*\*2))

  return rmse

# Compute objective function (cost) on dataset (X, t).

def compute\_cost(X, t, w):

  N = X.shape[0]

  cost = 1/N \* np.sum((t - X.dot(w))\*\*2)

  return cost

# Compute gradient of the objective function (cost) on dataset (X, t).

def compute\_gradient(X, t, w):

  N = X.shape[0]

  grad = np.zeros(w.shape)

  grad = -2/N \* X.T.dot(t - X.dot(w))

  return grad

##======================= Main program =======================##

parser = argparse.ArgumentParser('Multiple Regression Exercise.')

parser.add\_argument('-i', '--input\_data\_dir',

                    type=str,

                    default='D:\AIUB\Academics\Semester 10\ML\Final Assignment\Part 1\linear\_regression\data\multiple',

                    #default='../data/multiple',  #this line is not working

                    help='Directory for the multiple regression houses dataset.')

FLAGS, unparsed = parser.parse\_known\_args()

# Read the training and test data.

Xtrain, ttrain = read\_data(FLAGS.input\_data\_dir + "/train.txt")

Xtest, ttest = read\_data(FLAGS.input\_data\_dir + "/test.txt")

#  Standardize the training and test features using the mean and std computed over \*training\*.

mean, std = mean\_std(Xtrain)

Xtrain = standardize(Xtrain, mean, std)

Xtest = standardize(Xtest, mean, std)

#  Make sure you add the bias feature to each training and test example.

#  The bias features should be a column of ones addede as the first columns of training and test examples

Xtrain = np.hstack((np.ones((Xtrain.shape[0], 1)), Xtrain))

Xtest = np.hstack((np.ones((Xtest.shape[0], 1)), Xtest))

# Computing parameters for each training method for eta=0.1 and 200 epochs

eta=0.1

epochs=200

w,eph,costs=train(Xtrain,ttrain,eta,epochs)

# Print model parameters.

print('Params GD: ', w)

# Print cost and RMSE on training data.

print('Training RMSE: %0.2f.' % compute\_rmse(Xtrain, ttrain, w))

print('Training cost: %0.2f.' % compute\_cost(Xtrain, ttrain, w))

# Print cost and RMSE on test data.

print('Test RMSE: %0.2f.' % compute\_rmse(Xtest, ttest, w))

print('Test cost: %0.2f.' % compute\_cost(Xtest, ttest, w))

# Plotting Epochs vs. cost for Gradient descent methods

plt.figure(figsize=(10, 6))

plt.xlabel(' epochs')

plt.ylabel('cost')

plt.yscale('log')

plt.plot(eph, costs, 'bo-', label='train\_Jw\_gd')

plt.title('Epochs vs. Cost for Gradient Descent')

plt.legend()

plt.grid(True)

plt.savefig('gd\_cost\_multiple.png')

plt.show()

plt.close()

Results (simple.py):  
  
Params GD: [254449.99982048 93308.92004027]

Training RMSE: 64083.51.

Training cost: 2053348364.32.

Test RMSE: 65773.19.

Test cost: 2163056350.22.

Figure 1:

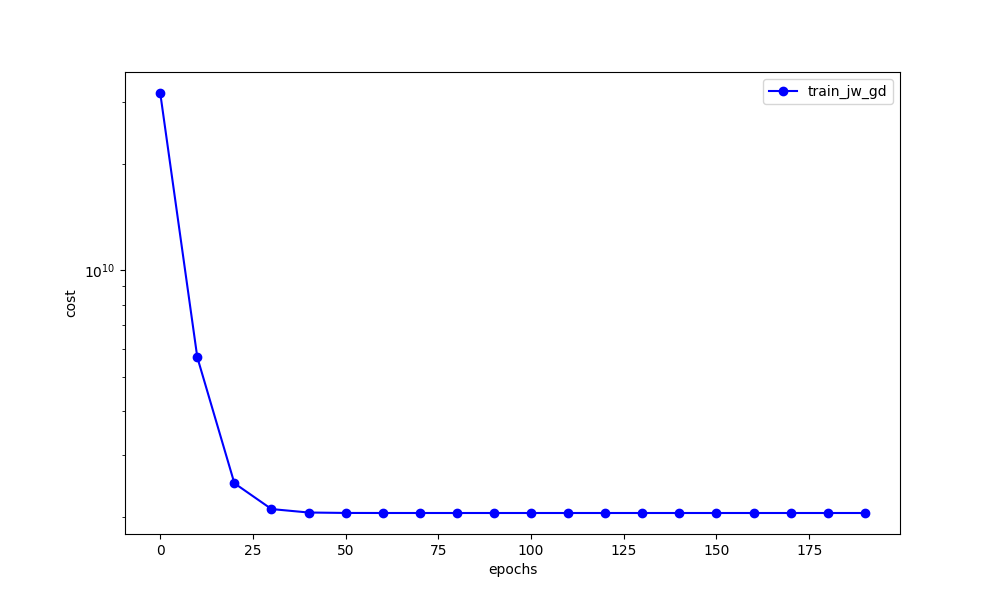
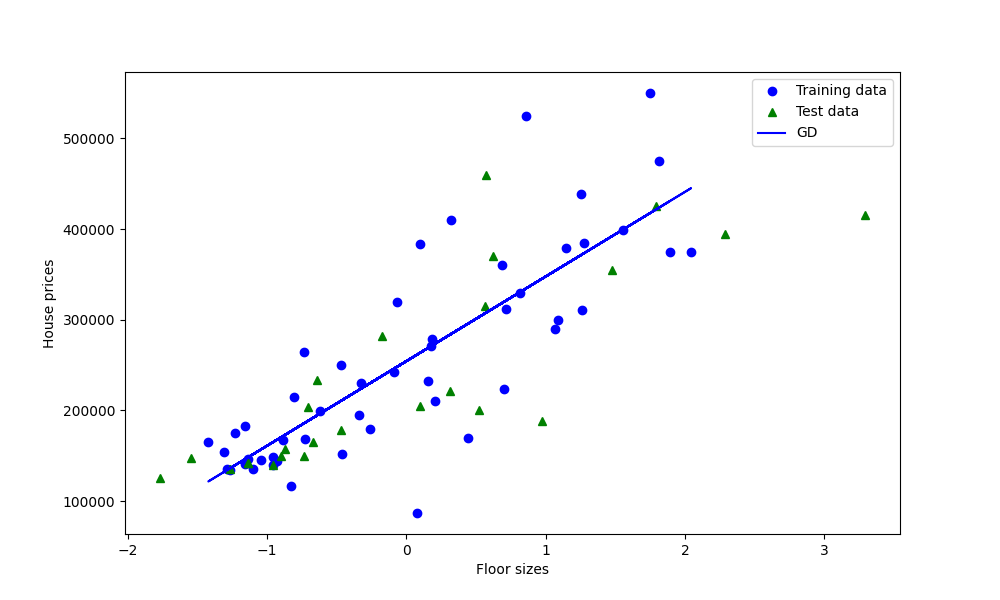


Figure 2:



Results (multiple.py):

Params GD: [254450. 78097.10330981 24424.60846106 2079.70655195]

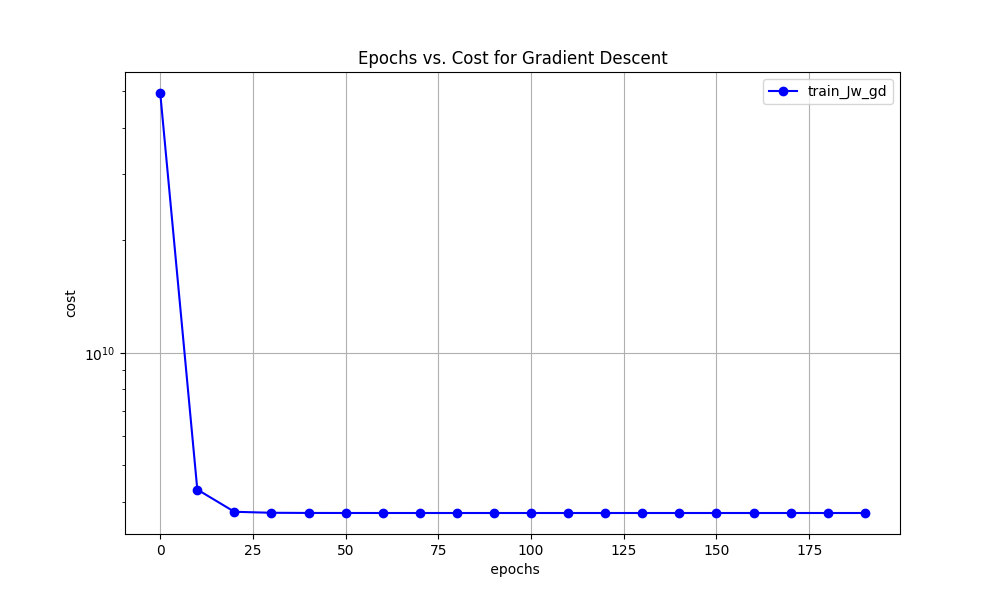
Training RMSE: 61070.62.

Training cost: 3729620373.97.

Test RMSE: 58481.32.

Test cost: 3420064336.43.

Figure 1:



**Report:**

**Introduction:**

This report describes the application and assessment of linear regression and gradient descent techniques using the Athens houses dataset. The main goal is to forecast house prices based on different attributes. These attributes include floor area, number of bedrooms, and construction year. The implementation is segmented into two sections: one for simple regression and another for multiple regression.

**Implemented Code Overview:**

The simple regression model predicts house prices based on their floor size. The model is trained using the gradient descent algorithm, which is run for 200 epochs with a learning rate of 0.1. The training and test data are standardized using the mean and standard deviation computed over the training data. This standardization is crucial as it improves the convergence of the gradient descent algorithm. After training, the model parameters are printed, and the Root Mean Square Error (RMSE) and the objective function values on the training and test data are reported.

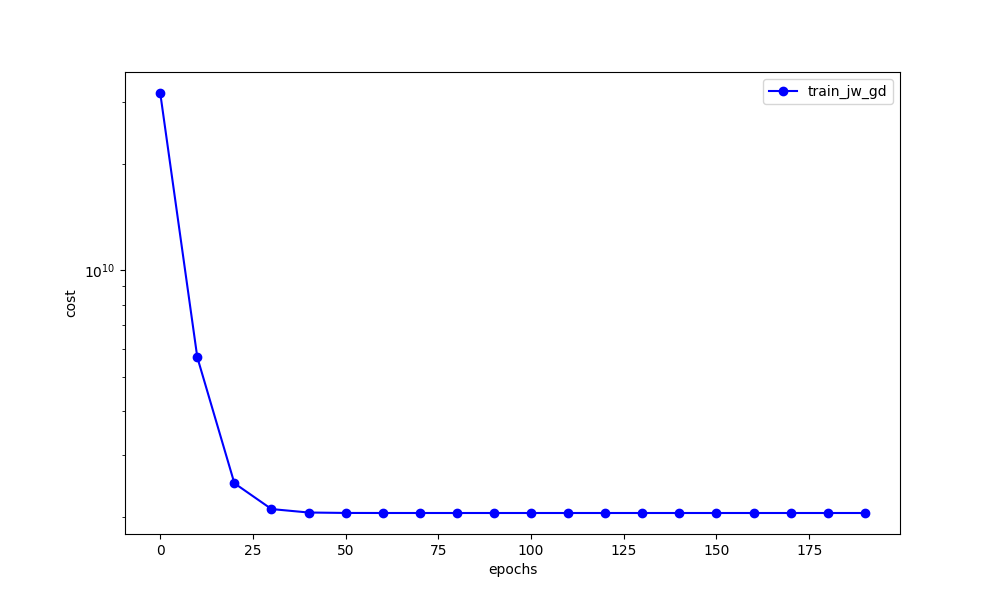
The multiple regression model predicts house prices based on their floor size, number of bedrooms, and year of construction. Similar to the simple regression model, the multiple regression model is trained using the gradient descent algorithm for 200 epochs with a learning rate of 0.1. The training and test data are standardized using the mean and standard deviation computed over the training data. After training, the model parameters are printed, and the RMSE and the objective function values on the training and test data are reported.

**Results and Observations:**

The results of the simple and multiple regression models are presented in this section.

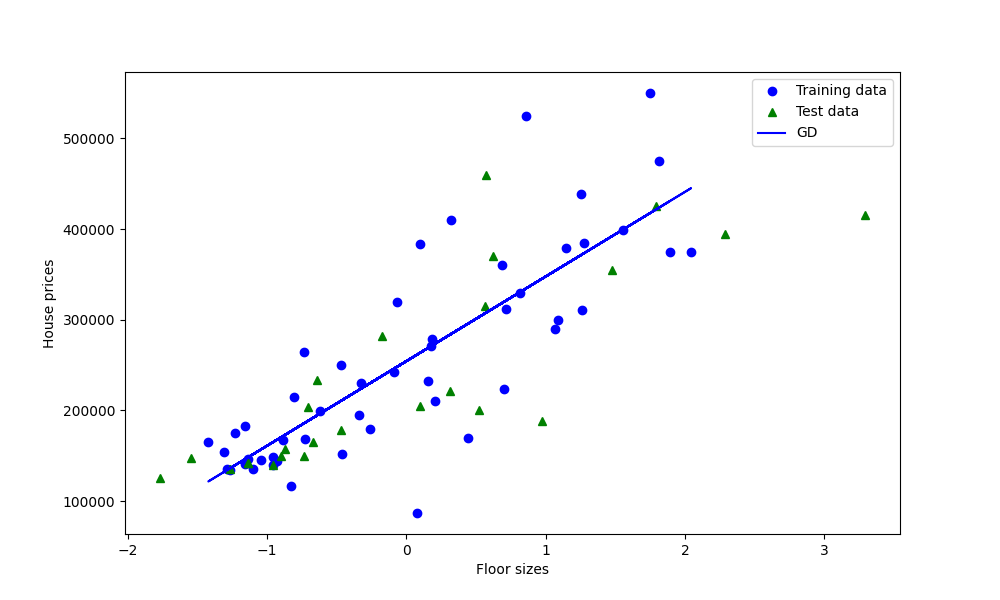
For the simple regression model, two key observations were made:

Epochs vs. Cost for Gradient Descent Methods: This observation is represented by the first figure.



The figure shows how the cost function changes over the epochs during the training of the model.

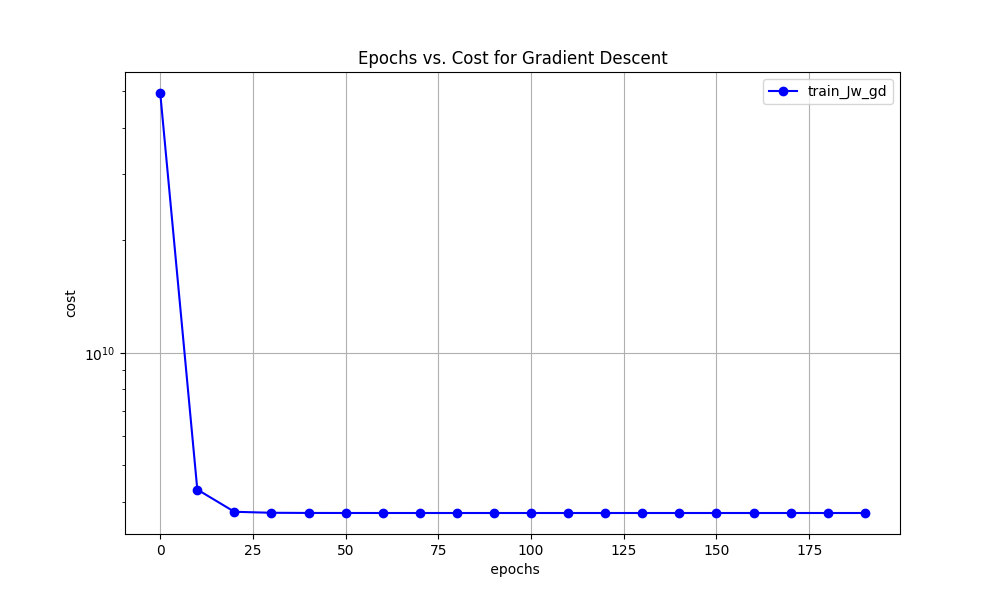
Linear Approximation for Each Training Method: This observation is represented by the second figure.



The figure shows the linear approximation fitted to the training data. Discuss how well the model fits the training data and any patterns or trends you observe.

For the multiple regression model, the key observation was made on how the cost function changes over the epochs during the training of the model:

Epochs vs. Cost for Gradient Descent Methods: This observation is represented by the third figure.



Scrutinizes the trends and patterns noted within this plot. Interprets these findings in relation to the model's overall performance.

**Conclusion:**

In conclusion, the implementation of the linear regression and gradient descent algorithms on the Athens houses dataset was successful. The models were able to predict house prices with reasonable accuracy. The results demonstrate the effectiveness of these algorithms in solving regression problems. The exercise also highlights the importance of feature scaling in improving the convergence of the gradient descent algorithm. Future work could explore the use of other optimization algorithms, the impact of different learning rates, and the inclusion of additional features in the model.