```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 16 21:22:39 2021

@author: 11327
"""

import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import csv

# Exercise 1

# Reading csv file for male data
with open("./data/male_train_data.csv", "r") as csv_file:
    reader = csv.reader(csv_file, delimiter=',')
    # Add your code here to process the data into usable form
    i = 0
    data_m = np.array([])
    for line in reader:
        try:
            line[1] = str(float(line[1])/10) # normalizing BMI
        except:
            None
        try:
            line[2] = str(float(line[2])/1000) # normalizing stature_mm
        except:
            None
        if i == 0:
            data_m = np.array(line)
        else:
            data_m = np.row_stack((data_m,np.array(line)))
        i = i+1
    print(data_m[0:11,:])

csv_file.close()

print()

# Reading csv file for female data
with open("./data/female_train_data.csv", "r") as csv_file:
    reader = csv.reader(csv_file, delimiter=',')
    # Add your code here to process the data into usable form
    i = 0
    data_f = np.array([])
    for line in reader:
        try:
            line[1] = str(float(line[1])/10) # normalizing BMI
        except:
            None
        try:
            line[2] = str(float(line[2])/1000) # normalizing stature_mm
        except:
            None
        if i == 0:
            data_f = np.array(line)
        else:
            data_f = np.row_stack((data_f,np.array(line)))
        i = i+1
    print(data_f[0:11,:])
```

```
csv_file.close()



# Exercise 2
# b

data_m = np.around(data_m[1:,1:].astype(np.float), decimals = 3)
data_f = np.around(data_f[1:,1:].astype(np.float), decimals = 3)

X = np.row_stack((data_m,data_f))
# X = X.astype(np.float)

# add a column of 1 to the X
X = np.c_[np.ones(len(X)), X]

# Producing yn
len_m = len(data_m)
len_f = len(data_f)
y = np.concatenate((np.array(np.ones(len_m)), np.array(-1.0*np.ones(len_f))))

# Calculating theta
theta = np.dot(np.dot(np.linalg.pinv(np.dot(X.T,X)),X.T),y)

# c
THETA = cp.Variable(X.shape[1])
objective = cp.Minimize(cp.sum_squares(y - X@THETA))
prob = cp.Problem(objective)

result = prob.solve()
THETA_c = THETA.value

# e: Gradient descent
# Initialize gradient descent
d = X.shape[1]
max_itr = 50000
cost = np.zeros(max_itr)
theta_e = [0.,0.,0.]
XtX = np.dot( np.transpose(X), X)
theta_store = np.zeros((d,max_itr+1))
for i in range(d):
    theta_store[i,0] = 0

# Gradient descent
for itr in range(max_itr):
  dJ     = 2*(np.dot(XtX,theta_e) - np.dot(X.T,y))
  dd     = -1 * dJ
  alpha  = (np.dot(np.dot(y.T,X),dd) - np.dot(np.dot(theta_e,XtX),dd)) / np.sum((np.dot(X,dd))*
  theta_e  = theta_e + alpha*dd
  theta_store[:,itr+1] = theta_e
  cost[itr] = np.linalg.norm(y-np.dot(X, theta_e))**2/X.shape[0]

# f
plt.figure(1)
plt.semilogx(cost,'o',linewidth=8)

# g
beta = 0.9
# Initialize gradient descent
d = X.shape[1]
```

```python
max_itr = 50000
cost = np.zeros(max_itr)
theta_g = [0.,0.,0.]
XtX = np.dot( np.transpose(X), X)
theta_store = np.zeros((d,max_itr+1))
for i in range(d):
    theta_store[i,0] = 0

# Gradient descent
itr = 0
dJ     = 2*(np.dot(XtX,theta_g) - np.dot(X.T,y))
dJ_lasttime = 2*(np.dot(XtX,theta_store[:,itr]) - np.dot(X.T,y))
dd     = -1 * (beta*dJ_lasttime + (1-beta)*dJ)
alpha  = (np.dot(np.dot(y.T,X),dd) - np.dot(np.dot(theta_g,XtX),dd)) / np.sum((np.dot(X,dd))**2
theta_g  = theta_g + alpha*dd
theta_store[:,itr+1] = theta_g
cost[itr] = np.linalg.norm(y-np.dot(X, theta_g))**2/X.shape[0]

for itr in range(1,max_itr):
    dJ     = 2*(np.dot(XtX,theta_g) - np.dot(X.T,y))
    dJ_lasttime = 2*(np.dot(XtX,theta_store[:,itr-1]) - np.dot(X.T,y))
    dd     = -1 * (beta*dJ_lasttime + (1-beta)*dJ)
    alpha  = (np.dot(np.dot(y.T,X),dd) - np.dot(np.dot(theta_g,XtX),dd)) / np.sum((np.dot(X,dd)
    theta_g  = theta_g + alpha*dd
    theta_store[:,itr+1] = theta_g
    cost[itr] = np.linalg.norm(y-np.dot(X, theta_g))**2/X.shape[0]

# h
plt.figure(2)
plt.semilogx(cost,'o',linewidth=8)



# Exercise 3

# a
# i
plt.figure(3)
plt.scatter(data_m[:,0],data_m[:,1],c = 'b',alpha = 0.8,linewidth = 0.5)
plt.scatter(data_f[:,0],data_f[:,1],c = 'r',alpha = 0.8,linewidth = 0.5)
plt.xlabel('bmi')
plt.ylabel('stature_mm')

# ii
xaxis = np.linspace(1,10,100)
yaxis = (-1*theta[0]-theta[1]*xaxis) / theta[2]
plt.plot(xaxis,yaxis)

# b
f_predicted_mm =  (-1*theta[0]-theta[1]*data_f[:,0]) / theta[2]
m_predicted_mm =  (-1*theta[0]-theta[1]*data_m[:,0]) / theta[2]

false_alarm = 0
for i in range(len_f):
    if (data_f[i,1] > f_predicted_mm[i]):
        false_alarm = false_alarm + 1
false_alarm = false_alarm / len_f

Miss = 0
for i in range(len_m):
    if (data_m[i,1] < m_predicted_mm[i]):
```

```
        Miss = Miss + 1
Miss = Miss / len_m

TP = len_m - Miss
FP = Miss
FN = false_alarm
precision = (TP)/(TP+FP)
recall = TP/(TP+FN)



# Exercise 4: Regularization
# a

THETA_4 = []
lambd = np.arange(0.1,10,0.1)

for i in range(len(lambd)):
    THETA = cp.Variable(X.shape[1])
    objective = cp.Minimize(cp.sum_squares(X@THETA - y)+lambd[i]*cp.sum_squares(THETA))
    prob = cp.Problem(objective)

    result = prob.solve()
    THETA_4.append(THETA.value)

x_4a = []
y_4a = []
for i in range(len(lambd)):
    x_4a.append(np.linalg.norm(THETA_4[i], ord=2))
    y_4a.append(np.linalg.norm(X@THETA_4[i] - y, ord=2))
plt.figure(1)
plt.plot(x_4a, y_4a)

plt.figure(2)
plt.plot(lambd, y_4a)

plt.figure(3)
plt.plot(lambd, x_4a)
```