Exercise 1

```
import cvxpy as cp
import numpy as np
from itertools import islice
import pandas as pd

# Problem data.
m = 30
n = 20
np.random.seed(1)
A = np.random.randn(m, n)
b = np.random.randn(m)
# Construct the problem.
x = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A@x - b))
constraints = [0 <= x, x <= 1]
prob = cp.Problem(objective, constraints)
# The optimal objective value is returned by `prob.solve()`.
result = prob.solve()
# The optimal value for x is stored in `x.value`.

print(x.value)
# The optimal Lagrange multiplier for a constraint is stored in
# `constraint.dual_value`.
print(constraints[0].dual_value)
```

```
[-1.79109253e-19  2.85112420e-02  2.79973443e-19  3.37658751e-20
 -2.72802659e-19  1.49285011e-01 -9.97212062e-20  8.35373892e-20
  2.46718649e-01  5.78224144e-01 -4.03739462e-19  1.01242860e-03
 -9.28486200e-20  2.26767464e-01 -1.58813677e-19 -8.97232308e-20
 -1.22145726e-19 -1.51509432e-19  1.12060673e-19 -3.48318630e-19]
[ 2.50938945  0.          2.78354615  1.79425782 13.08579183  0.
  0.73716363  3.35344995  0.          0.          8.93825054  0.
  7.02955161  0.          4.71068649  3.18873635  2.06090107 10.08166738
  3.0481157   8.53268239]
```

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
import csv
from google.colab import files


uploaded = files.upload()
# Reading csv file for male data
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving female_test_data.csv to female_test_data.csv
Saving female_train_data.csv to female_train_data.csv
Saving male_test_data.csv to male_test_data.csv
Saving male_train_data.csv to male_train_data.csv
```

```
male_train = pd.read_csv('male_train_data.csv')
male_train['male_bmi'] = male_train['male_bmi']/10
male_train['male_stature_mm'] = male_train['male_stature_mm']/1000
male_train['y'] = 1
male_train.head(10)
```

	index	male_bmi	male_stature_mm	y

```
female_train = pd.read_csv('female_train_data.csv')
female_train['female_bmi'] = female_train['female_bmi']/10
female_train['female_stature_mm'] = female_train['female_stature_mm']/1000
female_train['y'] = -1
female_train.head(10)
```

	index	female_bmi	female_stature_mm	y
0	0	2.82	1.563	-1
1	1	2.22	1.716	-1
2	2	2.71	1.484	-1
3	3	2.81	1.651	-1
4	4	2.55	1.548	-1
5	5	2.30	1.665	-1
6	6	3.56	1.564	-1
7	7	3.11	1.676	-1
8	8	2.46	1.690	-1
9	9	4.30	1.704	-1

```
bmi =  male_train["male_bmi"].values.tolist() + female_train["female_bmi"].values.tolist()
stature =  male_train["male_stature_mm"].values.tolist() + female_train["female_stature_mm"].values.tolist()
y = male_train["y"].values.tolist() + female_train["y"].values.tolist()
```

```
male_test = pd.read_csv('male_test_data.csv')
male_test['male_bmi'] = male_test['male_bmi']/10
male_test['male_stature_mm'] = male_test['male_stature_mm']/1000
male_test['y'] = 1

female_test = pd.read_csv('female_test_data.csv')
female_test['female_bmi'] = female_test['female_bmi']/10
female_test['female_stature_mm'] = female_test['female_stature_mm']/1000
female_test['y'] = -1

bmi_test =  male_test["male_bmi"].values.tolist() + female_test["female_bmi"].values.tolist()
stature_test =  male_test["male_stature_mm"].values.tolist() + female_test["female_stature_mm"].values.tolist()
y_test = male_test["y"].values.tolist() + female_test["y"].values.tolist()

X_test = np.array([bmi_test, stature_test])
X_test = X_test.T
X_2_test = np.append(np.ones([len(X_test),1]), X_test,1)

X_2_test
```

```
array([[1.   , 3.39 , 1.681],
       [1.   , 2.85 , 1.735],
       [1.   , 2.76 , 1.821],
       ...,
       [1.   , 1.66 , 1.575],
       [1.   , 2.28 , 1.542],
       [1.   , 3.16 , 1.669]])
```

```
X = np.array([bmi, stature])
X = X.T
X_2 = np.append(np.ones([len(X),1]), X,1)
X_2
```

```
array([[1.   , 3.   , 1.679],
       [1.   , 2.56 , 1.586],
       [1.   , 2.42 , 1.773],
       ...,
       [1.   , 2.79 , 1.573],
       [1.   , 2.13 , 1.432],
       [1.   , 2.74 , 1.651]])
```

Exercise 2

Using Analytical Solution

```
theta = np.matmul(X_2.T,X_2)
theta = np.linalg.inv(theta)
temp = np.matmul(X_2.T, y)
theta = np.matmul(theta, temp)

print(theta)
```

```
[-10.7017505   -0.12339677   6.67486843]
```

Using CVX

```
theta = cp.Variable(3)
cost = cp.sum_squares(X_2@theta-y)
prob = cp.Problem(cp.Minimize(cost))
prob.solve()
theta.value

print(theta.value)
```

```
[-10.7017505   -0.12339677   6.67486843]
```

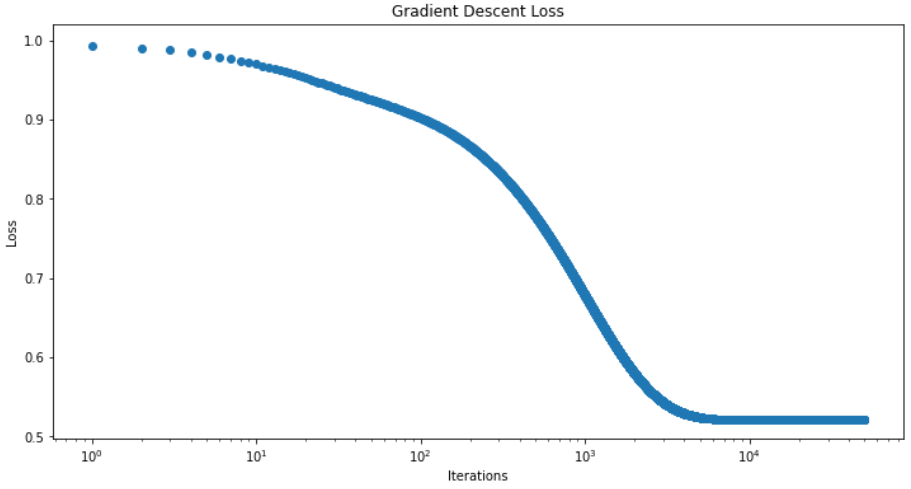
Gradient Descent

```
d = 2
N = X_2.shape[0]
itr = 50000
theta = np.zeros(d+1)
cost  = np.zeros(itr)
XtX = np.dot( np.transpose(X_2), X_2)

# Gradient descent
for itr in range(itr):
  dJ     = np.dot(np.transpose(X_2), np.dot(X_2, theta)-y)
  dd     = dJ
  alpha  = np.dot(dJ, dd) / np.dot(np.dot(XtX, dd), dd)
  theta  = theta - alpha*dd
  cost[itr] = np.linalg.norm(np.dot(X_2, theta)-y)**2/N

print(theta)
```

```
[-10.7017505   -0.12339677   6.67486843]
```

```
# Plotting
fig = plt.figure(figsize = (12,6))
plt.semilogx(cost,'o', linewidth=8)
plt.title("Gradient Descent Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
```

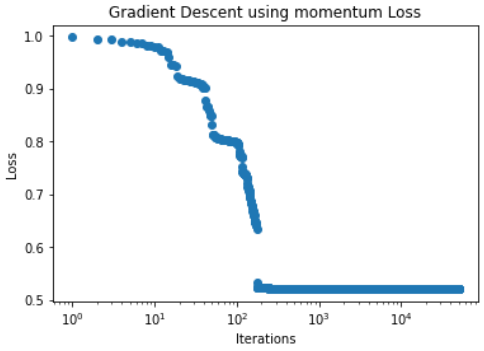
```
Text(0, 0.5, 'Loss')
```



Gradient descent with momentum

```
d = 2
N = X_2.shape[0]
itr = 50000
theta = np.zeros(d+1)
cost  = np.zeros(itr)
dJ_old = np.zeros(d+1)
XtX = np.dot( np.transpose(X_2), X_2)

beta  = 0.9
for itr in range(itr):
  dJ     = np.dot(np.transpose(X_2), np.dot(X_2, theta)-y)
  dd     = beta*dJ_old + (1-beta)*dJ
  alpha  = np.dot(dJ, dd) / np.dot(np.dot(XtX, dd), dd)
  theta  = theta - alpha*dd
  dJ_old = dJ
  cost[itr] = np.linalg.norm(np.dot(X_2, theta)-y)**2/N

print(theta)

plt.semilogx(cost,'o', linewidth=8)
plt.title("Gradient Descent using momentum Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
```

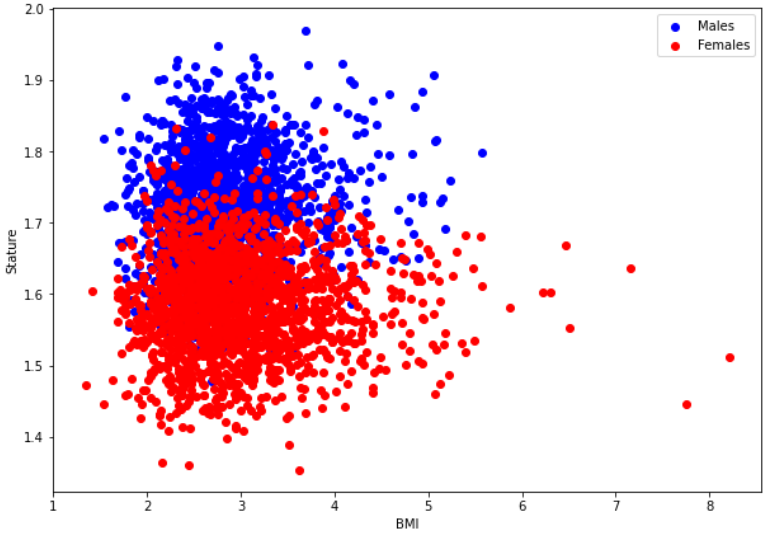
```
[-10.7017505   -0.12339677   6.67486843]
Text(0, 0.5, 'Loss')
```



Exercise 3

```
plt.figure(figsize = (10,7))
plt.scatter(male_train['male_bmi'], male_train['male_stature_mm'], c='b')
plt.scatter(female_train['female_bmi'], female_train['female_stature_mm'], c='r')
plt.xlabel("BMI")
plt.ylabel("Stature")
plt.legend(["Males", "Females"])
```

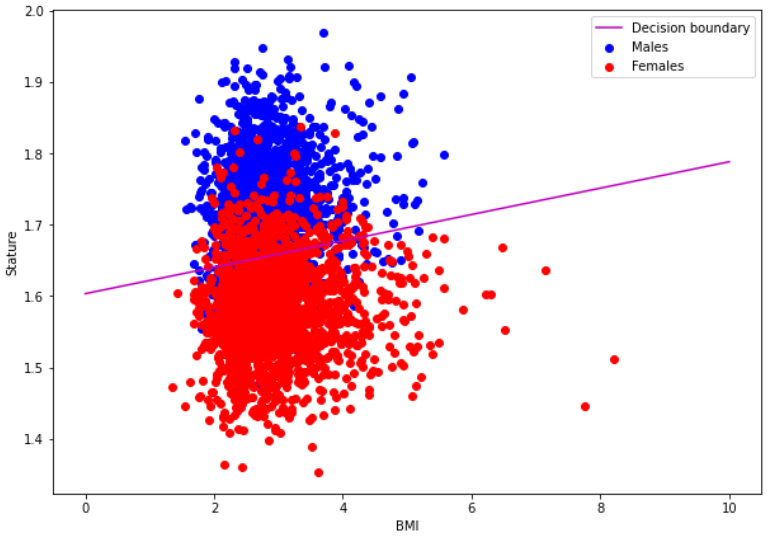
```
<matplotlib.legend.Legend at 0x7f2a79590fd0>
```




```
from numpy.core.function_base import linspace
m = - theta[1]/theta[2]
```

```
c = - theta[0]/theta[2]

x1 = linspace(0,10,100)
x2 = m*x1 + c

plt.figure(figsize = (10,7))
plt.plot(x1, x2, c='m')
plt.scatter(male_train['male_bmi'], male_train['male_stature_mm'], c='b')
plt.scatter(female_train['female_bmi'], female_train['female_stature_mm'], c='r')
plt.xlabel("BMI")
plt.ylabel("Stature")
plt.legend(["Decision boundary","Males", "Females"])
```

```
<matplotlib.legend.Legend at 0x7f2a793eed30>
```




```
y_preds = [np.sign(np.dot(X_2_test[i], theta)) for i in range(X_test.shape[0])]
```

```
FALSE_ALARM_COUNT = 0
MISS_COUNT = 0
TRUE_POSITIVE_COUNT = 0
TRUE_NEGATIVE_COUNT = 0

for i in range(len(y_preds)):
  if((y_test[i] == -1) & (y_preds[i] == 1)):
    FALSE_ALARM_COUNT = FALSE_ALARM_COUNT + 1

  if((y_test[i] == 1) & (y_preds[i] == -1)):
    MISS_COUNT = MISS_COUNT + 1

  if((y_test[i] == 1) & (y_preds[i] == 1)):
    TRUE_POSITIVE_COUNT = TRUE_POSITIVE_COUNT + 1

  if((y_test[i] == -1) & (y_preds[i] == -1)):
    TRUE_NEGATIVE_COUNT = TRUE_NEGATIVE_COUNT + 1

TYPE_1 = FALSE_ALARM_COUNT/(FALSE_ALARM_COUNT + TRUE_NEGATIVE_COUNT)
TYPE_2 = MISS_COUNT/(MISS_COUNT + TRUE_POSITIVE_COUNT)
PRECISION = TRUE_POSITIVE_COUNT/(TRUE_POSITIVE_COUNT + FALSE_ALARM_COUNT)
RECALL = TRUE_POSITIVE_COUNT/(TRUE_POSITIVE_COUNT + MISS_COUNT)

print(f"Type 1 error = {TYPE_1*100:.2f} %")
print(f"Type 2 error = {TYPE_1*100:.2f} %")
print(f"Precision = {PRECISION:.2f}")
print(f"Recall = {RECALL:.2f}")
```

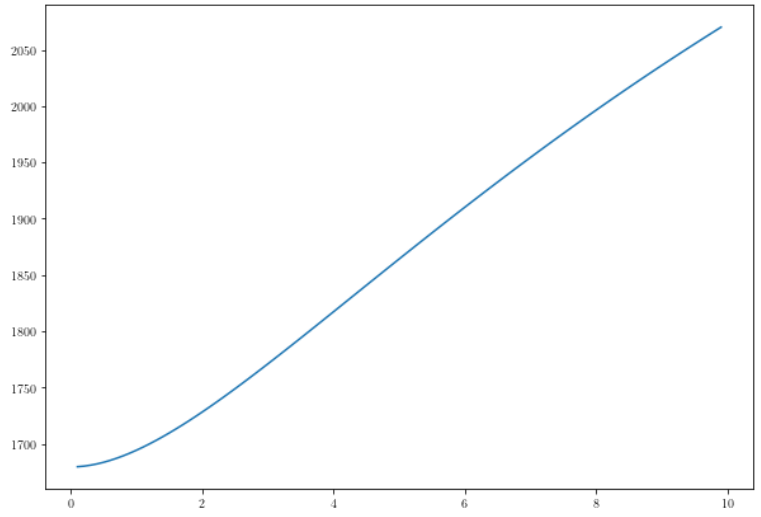
```
Type 1 error = 14.17 %
Type 2 error = 14.17 %
Precision = 0.85
Recall = 0.82
```

```
from pyparsing import cpp_style_comment
lambd_values = np.arange(0.1,10,0.1)
theta = cp.Variable(3)
theta_norm_values = []
Sum_sq_values = []
```

```
for lambd in lambd_values:
  objective   = cp.Minimize( cp.sum_squares(X_2@theta-y) + lambd*cp.sum_squares(theta) )
  prob        = cp.Problem(objective)
  prob.solve()
  theta_ridge = theta.value

  theta_norm_values.append(np.dot(theta_ridge,theta_ridge))
  val = np.matmul(X_2, theta_ridge)-y
  Sum_sq_values.append(np.dot(val, val))
```

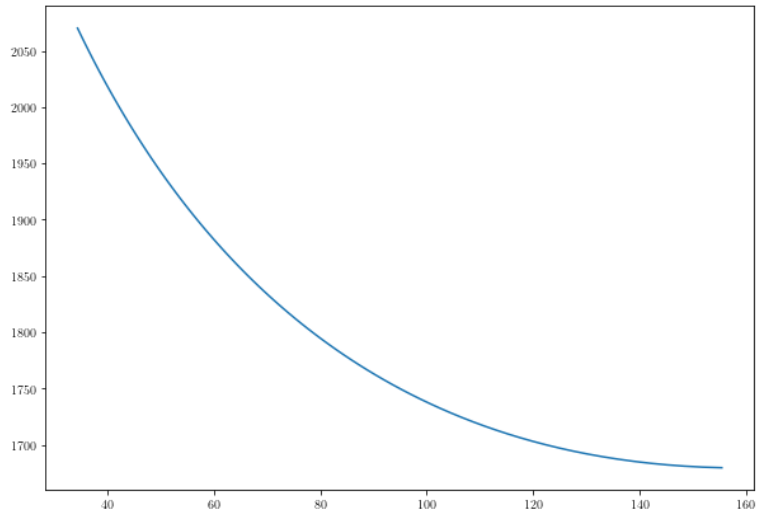
```
import matplotlib
from matplotlib import rc
import matplotlib.pyplot as plt
%matplotlib inline

rc('text', usetex=True)
matplotlib.rcParams['text.latex.preamble'] = [r'\usepackage{amsmath}']

plt.figure(figsize = (10,7))
plt.plot(lambd_values, Sum_sq_values)
```

```
[<matplotlib.lines.Line2D at 0x7f41668698e0>]
```




```
plt.figure(figsize = (10,7))
plt.plot(theta_norm_values, Sum_sq_values)
```

```
[<matplotlib.lines.Line2D at 0x7f41667c8610>]
```




```
plt.figure(figsize = (10,7))
plt.plot(lambd_values, theta_norm_values)
```

```
[<matplotlib.lines.Line2D at 0x7f4166b1c0a0>]
```