Regression

1.a)

```
import numpy
import matplotlib.pyplot as plt
x = numpy.arange(0, 1, 0.001)
y=[]
for i in x:
 a = 1*i + 10
 y.append(a)
plt.plot(x,y)
```

1.b)

```
import numpy
import matplotlib.pyplot as plt
import random
x = numpy.arange(0, 1, 0.001)
y=[]
for i in x:
   a = 1*i + 10
   y.append(a)
r = []
for j in range(0, 1000):
   b=random.uniform(0, 1)
   r.append(b)
y1 =[]
for k in range(0, 1000):
   c=y[k] +(r[k]*0.1)
   y1.append(c)

plt.scatter(x, y1)
```

1.c)

```
import numpy as np
import random
import matplotlib.pyplot as plt
x=[]
```

```
N=1000
x = np.random.rand(N)
z = np.random.rand(N)

y_cor=x+10+0.1*z
w = []

for i in range(1000):

    k = random.uniform(-5, 7)
    w.append(k)
w.sort()
error=[]
for i in range(1000):
    y_pred=[]
    error.append(0)
    for j in range(1000):
        y_pred.append(w[i]*x[j]+10)
        error[i]+=(y_cor[j]-y_pred[j])**2
    error[i]/=1000

plt.figure(1)
plt.plot(w, error)

min_in=error.index(min(error))

y_best=[]
for i in range(1000):
    y_best.append(w[min_in]*x[i] + 10)
plt.figure(2)
plt.scatter(x,y_cor)
plt.plot(x,y_best, color='orange')
plt.show()
```

Q2 .a)

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 1, 0.001)
y=[]
for i in x:
 a = 1.5*i + 5
 y.append(a)
```

```
plt.plot(x,y)
```

2.b)

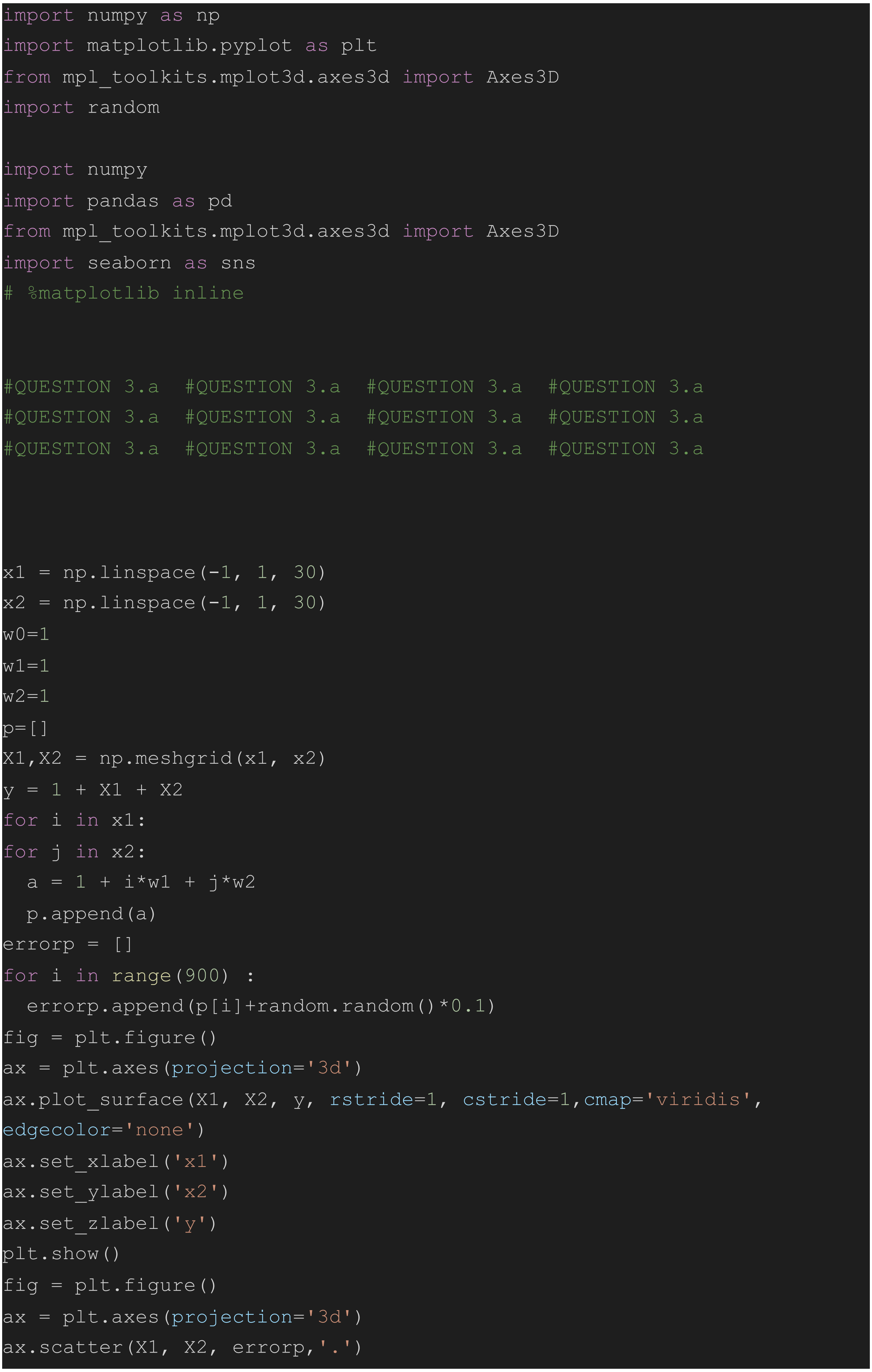
```
import numpy
import matplotlib.pyplot as plt
import random
x = numpy.arange(0, 1, 0.001)
y=[]
for i in x:
   a = 1.5*i + 5
   y.append(a)
r = []
for j in range(0, 1000):
   b=random.uniform(0, 1)
   r.append(b)
y1 =[]
for k in range(0, 1000):
   c=y[k] +(r[k]*0.1)
   y1.append(c)

plt.scatter(x, y1)
```

2.c

```
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd
from mpl_toolkits.mplot3d.axes3d import Axes3D
import seaborn as sns
%matplotlib inline
sns.set()

x = np.arange(0, 1, 0.001)
y=[]
for i in x:
   a = 1*i + 10
   y.append(a)
r = []
for j in range(0, 1000):
   b=random.uniform(0, 1)
   r.append(b)
y1 =[]
for k in range(0, 1000):
   c=y[k] +(r[k]*0.1)
```

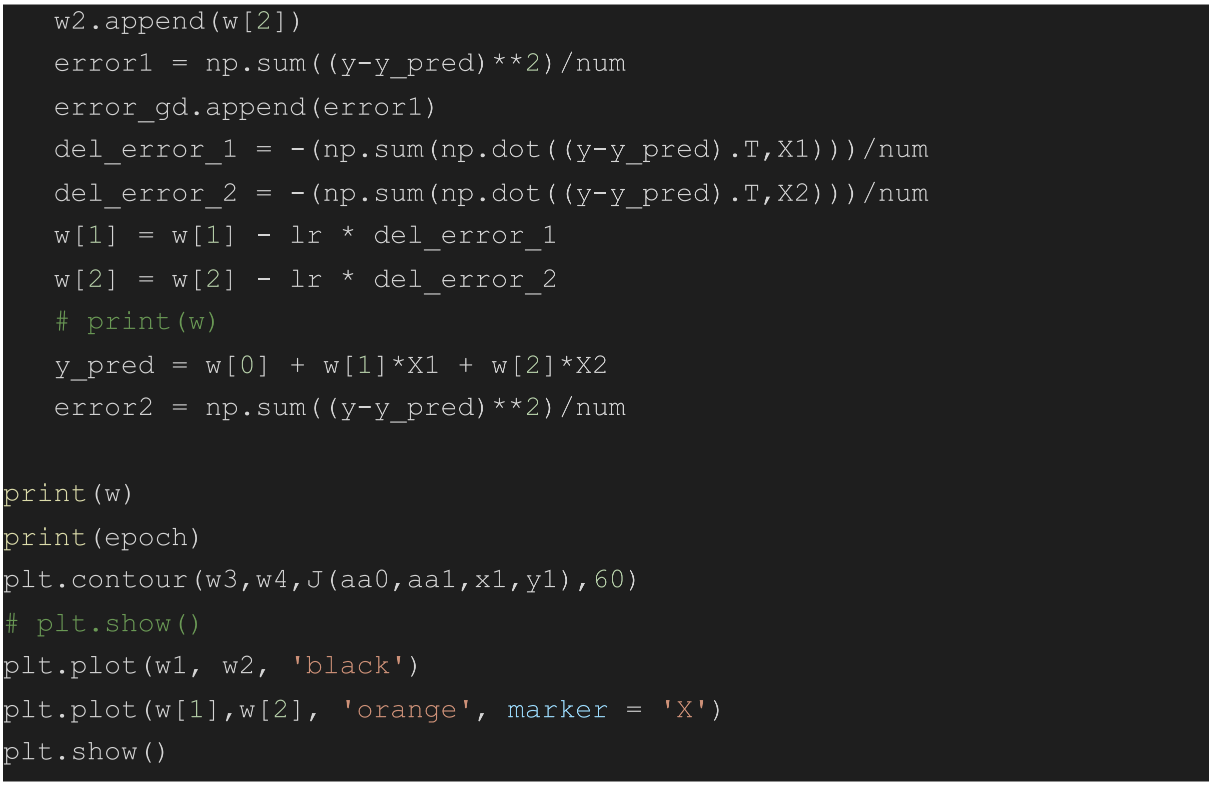
```
    y1.append(c)
def J(w0, w1, x, y1):
    J = 0
    for i in range(1000):
        J += ((w0 + w1*x[i]) - y1[i] )**2
    return J/2000
fig = plt.figure()
ax = fig.add_subplot(1,1,1,projection='3d')
w0 = np.linspace(-10,10,1000)
w1 = np.linspace(-10,10,1000)
aa0, aa1 = np.meshgrid(w0, w1)
ax.plot_surface(aa0, aa1, J(aa0,aa1,x,y1), rstride=1,
cstride=1,cmap='viridis', edgecolor='none')
ax.set_xlabel('w1')
ax.set_ylabel('w0')
ax.set_zlabel('error')

plt.show()
```

Q3



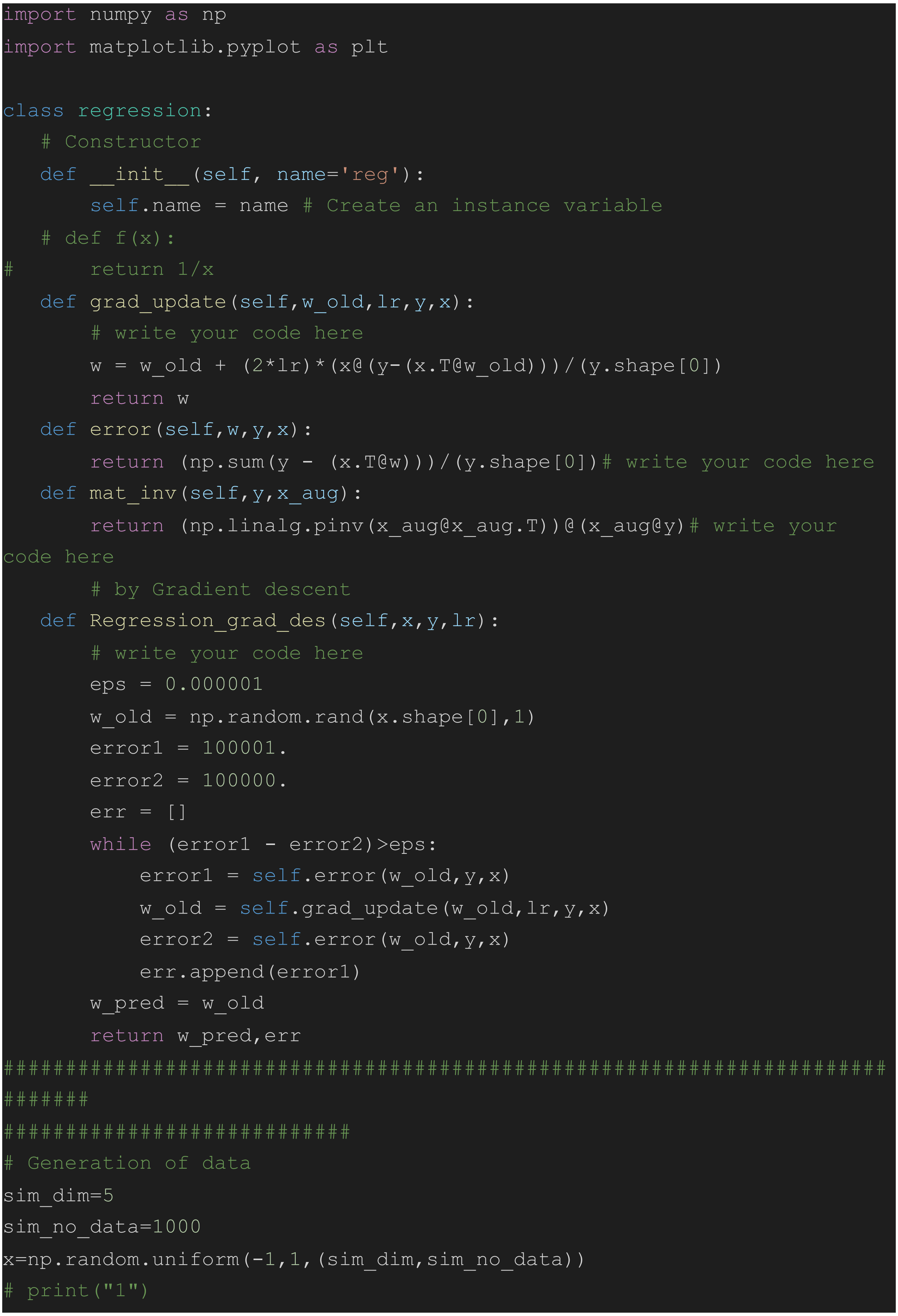
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D
import random

import numpy
import pandas as pd
from mpl_toolkits.mplot3d.axes3d import Axes3D
import seaborn as sns
# %matplotlib inline


#QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a
#QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a
#QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a  #QUESTION 3.a



x1 = np.linspace(-1, 1, 30)
x2 = np.linspace(-1, 1, 30)
w0=1
w1=1
w2=1
p=[]
X1,X2 = np.meshgrid(x1, x2)
y = 1 + X1 + X2
for i in x1:
for j in x2:
  a = 1 + i*w1 + j*w2
  p.append(a)
errorp = []
for i in range(900) :
  errorp.append(p[i]+random.random()*0.1)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X1, X2, y, rstride=1, cstride=1,cmap='viridis',
edgecolor='none')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.show()
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X1, X2, errorp,'.')
```

```
plt.show()



#QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b
#QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b
#QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b  #QUESTION 3.b


sns.set()
x1 = np.linspace(-1, 1, 30)
x2 = np.linspace(-1, 1, 30)
w0=1
w1=1
w2=1
p=[]
for i in x1:
  for j in x2:
      a = 1 + i*w1 + j*w2
      p.append(a)
y1 = []
for i in range(900) :
  y1.append(p[i]+random.random()*0.1)
def J(w1, w2,x1,y1):
  J = 0
  p=[]
  for i in x1:
      for j in x2:
          a = 1 + i*w1 + j*w2
          p.append(a)
  for i in range(900):
      J += (p[i] - y1[i] )**2
  return J/900
fig = plt.figure()
ax = fig.add_subplot(1,1,1,projection='3d')
w1 = np.linspace(-10,10,100)
w2 = np.linspace(-10,10,100)
aa0, aa1 = np.meshgrid(w1, w2)
ax.plot_surface(aa0, aa1, J(aa0,aa1,x1,y1), rstride=1,
cstride=1,cmap='viridis', edgecolor='none')
ax.set_xlabel('w1')
ax.set_ylabel('w2')
ax.set_zlabel('j')
plt.show()
plt.contour(w1,w2,J(aa0,aa1,x1,y1),60)
w3=w1
w4=w2
plt.show()



#QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c
#QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c
#QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c  #QUESTION 3.c

y = w0 + 1*X1 + 2*X2
print(y.shape)
# y = y.reshape(30,1)

# write your code here
xnew = [X1.reshape(900,1),X2.reshape(900,1)]
xnew = np.array(xnew).reshape(900,2)
ynew = y.reshape(900,1)
# xnew = xnew.T
print(xnew.shape)
w = np.array([5.,-6.,-4.])
eps = 0.00001
lr = 0.1
indexs = [1,2]
# w,w_gds,error1,epoch =  gradient_descent(w,eps,xnew,ynew,lr,indexs)
num=900
w = np.array([5.,-6.,-4.])
eps = 0.00001
lr = 0.1
X1 = X1.reshape(900,1)
X2 = X2.reshape(900,1)
y = w[0] + 1*X1 +2*X2
error1 = 1000001.
error2 = 1000000.
error_gd=[]
w1=[]
w2=[]
epoch=0
while abs(error1-error2)>eps:
   epoch+=1
   # print(error1)
   y_pred = w[0] + w[1]*X1 + w[2]*X2
   w1.append(w[1])
```

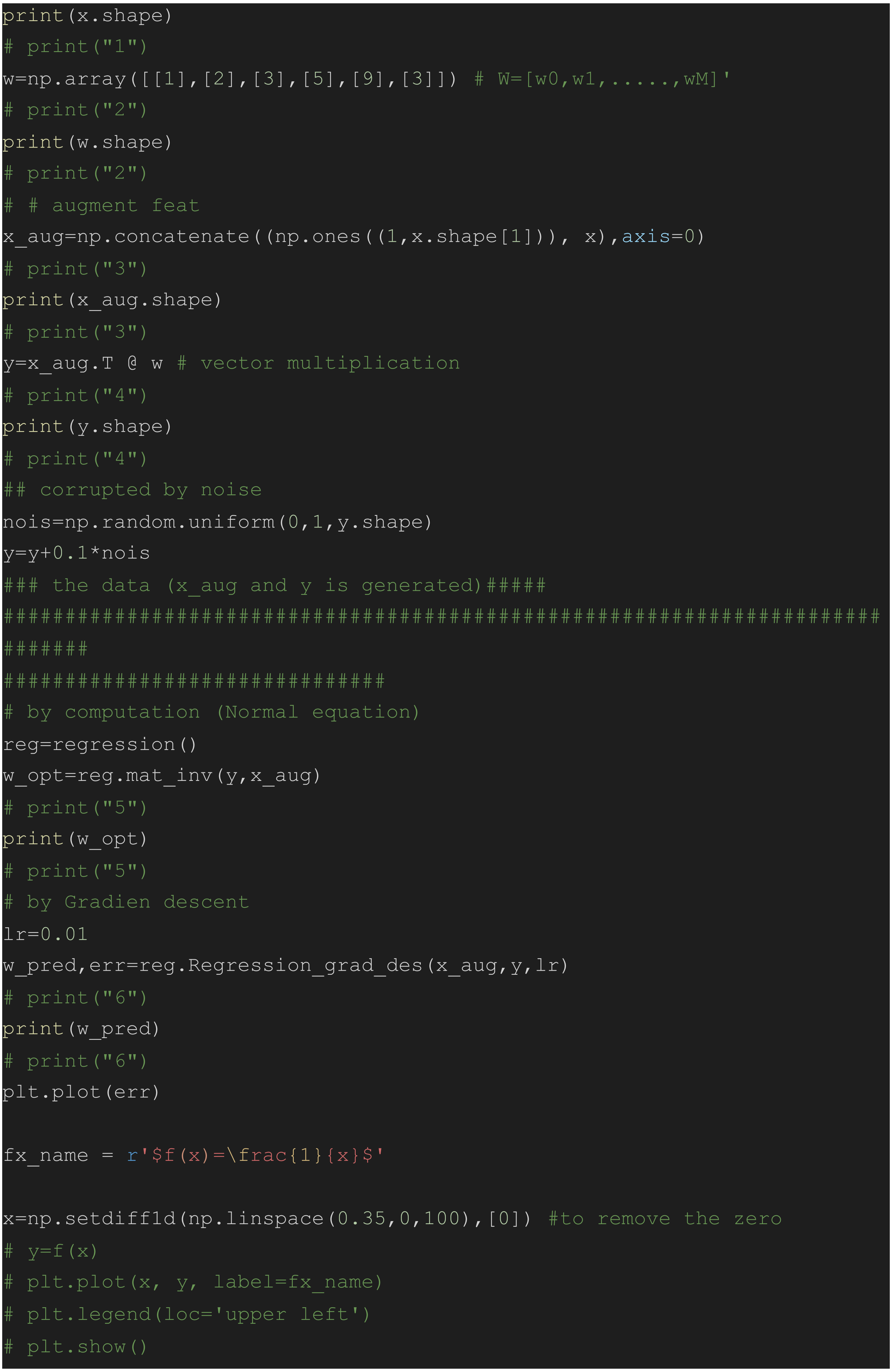


```
    w2.append(w[2])
    error1 = np.sum((y-y_pred)**2)/num
    error_gd.append(error1)
    del_error_1 = -(np.sum(np.dot((y-y_pred).T,X1)))/num
    del_error_2 = -(np.sum(np.dot((y-y_pred).T,X2)))/num
    w[1] = w[1] - lr * del_error_1
    w[2] = w[2] - lr * del_error_2
    # print(w)
    y_pred = w[0] + w[1]*X1 + w[2]*X2
    error2 = np.sum((y-y_pred)**2)/num

print(w)
print(epoch)
plt.contour(w3,w4,J(aa0,aa1,x1,y1),60)
# plt.show()
plt.plot(w1, w2, 'black')
plt.plot(w[1],w[2], 'orange', marker = 'X')
plt.show()
```

Q4



```
import numpy as np
import matplotlib.pyplot as plt

class regression:
    # Constructor
    def __init__(self, name='reg'):
        self.name = name # Create an instance variable
    # def f(x):
#       return 1/x
    def grad_update(self,w_old,lr,y,x):
        # write your code here
        w = w_old + (2*lr)*(x@(y-(x.T@w_old)))/(y.shape[0])
        return w
    def error(self,w,y,x):
        return (np.sum(y - (x.T@w)))/(y.shape[0])# write your code here
    def mat_inv(self,y,x_aug):
        return (np.linalg.pinv(x_aug@x_aug.T))@(x_aug@y)# write your
code here
        # by Gradient descent
    def Regression_grad_des(self,x,y,lr):
        # write your code here
        eps = 0.000001
        w_old = np.random.rand(x.shape[0],1)
        error1 = 100001.
        error2 = 100000.
        err = []
        while (error1 - error2)>eps:
            error1 = self.error(w_old,y,x)
            w_old = self.grad_update(w_old,lr,y,x)
            error2 = self.error(w_old,y,x)
            err.append(error1)
        w_pred = w_old
        return w_pred,err
########################################################################
#######
############################
# Generation of data
sim_dim=5
sim_no_data=1000
x=np.random.uniform(-1,1,(sim_dim,sim_no_data))
# print("1")
```



```
print(x.shape)
# print("1")
w=np.array([[1],[2],[3],[5],[9],[3]]) # W=[w0,w1,.....,wM]'
# print("2")
print(w.shape)
# print("2")
# # augment feat
x_aug=np.concatenate((np.ones((1,x.shape[1])), x),axis=0)
# print("3")
print(x_aug.shape)
# print("3")
y=x_aug.T @ w # vector multiplication
# print("4")
print(y.shape)
# print("4")
## corrupted by noise
nois=np.random.uniform(0,1,y.shape)
y=y+0.1*nois
### the data (x_aug and y is generated)#####
########################################################################
#######
###############################
# by computation (Normal equation)
reg=regression()
w_opt=reg.mat_inv(y,x_aug)
# print("5")
print(w_opt)
# print("5")
# by Gradien descent
lr=0.01
w_pred,err=reg.Regression_grad_des(x_aug,y,lr)
# print("6")
print(w_pred)
# print("6")
plt.plot(err)

fx_name = r'$f(x)=\frac{1}{x}$'

x=np.setdiff1d(np.linspace(0.35,0,100),[0]) #to remove the zero
# y=f(x)
# plt.plot(x, y, label=fx_name)
# plt.legend(loc='upper left')
# plt.show()
```

q5.

```
#### for
degree=0--------------------------------------------------------------
------------------------------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)
y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i]
- 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)
y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

plt.scatter(x,y1,color='red')
x1 = np.ones((100,1))
#x = np.reshape(x,(10,1))
#x = np.append(x,axis=1)
x_transpose = np.transpose(x1)
```

```
x_transpose_dot_x = x_transpose.dot(x1)

temp_1 = np.linalg.inv(x_transpose_dot_x)

temp_2=x_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

y = 17.67+0*x           #Uncomment this when using
Sample Dataset

plt.plot(x,y,color='blue')
plt.show()
#### for
degree=1--------------------------------------------------------------
--------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random

x = np.linspace(-6,6,100)
x = np.array(x)

y=[]
for i in range(0,100):
   y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i]
- 3)
```

```
r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

plt.scatter(x,y1,color='red')

x_bias = np.ones((100,1))

x1 = np.reshape(x,(100,1))

x_new = np.append(x_bias,x1,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new =
x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)

temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)
```

```
ynew = 18.67+3.6*x           #Uncomment this when
using Sample Dataset

plt.plot(x,ynew,color='blue')
plt.show()



#### for
degree=2--------------------------------------------------------------
--------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random

x = np.linspace(-6,6,100)
x = np.array(x)
z = np.linspace(0,1,100)
z = np.array(z)

y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i]
- 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

plt.scatter(x,y1,color='red')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new =
x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)
```

```
ynew = (1.24*pow(x,2))+2.54*x-0.80          #Uncomment
this when using Sample Dataset

plt.plot(x,ynew,color='blue')
plt.show()

#### for
degree=3-------------------------------------------------------------
--------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)

y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i]
- 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

x4=[]
for i in range(0,100):
    x4.append(pow(x[i],3))

plt.scatter(x,y1,color='red')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))
x5 = np.reshape(x4,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)
x_new = np.append(x_new,x5,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new =
x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)
```

```
print(theta)

ynew = (0.24*pow(x,3))+(1.25*pow(x,2))-2.97*x-0.64
#Uncomment this when using Sample Dataset

plt.plot(x,ynew,color='blue')
plt.show()

#### for
degree=4---------------------------------------------------------------
--------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)

y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i]
- 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

x4=[]
for i in range(0,100):
    x4.append(pow(x[i],3))

x6=[]
for i in range(0,100):
    x6.append(pow(x[i],4))

plt.scatter(x,y1,color='red')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))
x5 = np.reshape(x4,(100,1))
x7 = np.reshape(x6,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)
x_new = np.append(x_new,x5,axis=1)
x_new = np.append(x_new,x7,axis=1)

x_new_transpose = np.transpose(x_new)
```

```
x_new_transpose_dot_x_new =
x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)
#ynew =
((-9.64842938e-04)*pow(x,4)+((2.52206499e-01)*pow(x,
3))+((1.26992473e+00)*pow(x,2))-(2.98376091e+00)*x-
5.71438803e-01          #Uncomment this when using
Sample Dataset
plt.plot(x,ynew,color='blue')
plt.show()
```

****
****

6 Salary Prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
%matplotlib inline

import pandas as pd
data =
pd.read_csv(r"C:\Users\Abhay\anaconda3\salary_pred_
data1.csv")
data.head()
data.shape

x = data.iloc[:,:-1]
x.head()

y = data.iloc[:,5]
y.head()



#Distribution of salary
sns.kdeplot(data.Salary).set_title('Salary ($)')
plt.figure(figsize = (10, 3))
sns.boxplot(data.Salary)

data.plot(kind="scatter",x="Years of
experiance",y="Salary",color="blue",figsize=(8,6))
plt.title("Years of Experiance vs Salary")
plt.xlabel("Years of Experiance")
plt.ylabel("Salary")
plt.show()

# Splitting the dataset into the Training set and Test set
# Allocating half of the dataset set train and the other half to test on the model
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 1/10, random_state = 0)


# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)


# Predicting the Test set results
y_pred = regressor.predict(x_test)
#Converting series to array
y_t=y_test.values
#Combining two arrays
y_final = y_t, np.round(y_pred,2)
y_final


result = pd.DataFrame(list(y_final))
result


result = pd.DataFrame(list(y_final))
result = result.transpose()
result.columns = ["Actual Salary", "Predicted Salary"]
result.head(100)
```