IE406 Machine Learning

Lab Assignment - 2

Group 14

201901466: Miti Purohit

202001430: Aryan Shah

Question 1

Implement polynomial regression as a special case of linear regression. First generate some data. Now, we want to learn a polynomial function of degree p on this dataset, i.e. y = q0 + q1 × x1 + q2 × x2 + . . . qp × xp.

We can use the linear regression implementations for doing so, by transforming the dataset and creating the matrix X containing columns corresponding to x0, x1, x2, . . . , xp. Using any of your implementations above learn the regression coefficients for p = 5 and p = 4. How close are your coefficients for p = 5 to the ones used to generate the data?

Answer

code

[1]:
```
## Miti Purohit - 201901466
## Aryan Shah - 202001430

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
%matplotlib inline
from sklearn.metrics import mean_squared_error as mse
from sklearn.linear_model import LinearRegression
from scipy import optimize
import imageio
import os
#import numdifftools as nd
from mpl_toolkits import mplot3d
from sklearn.preprocessing import MinMaxScaler
```

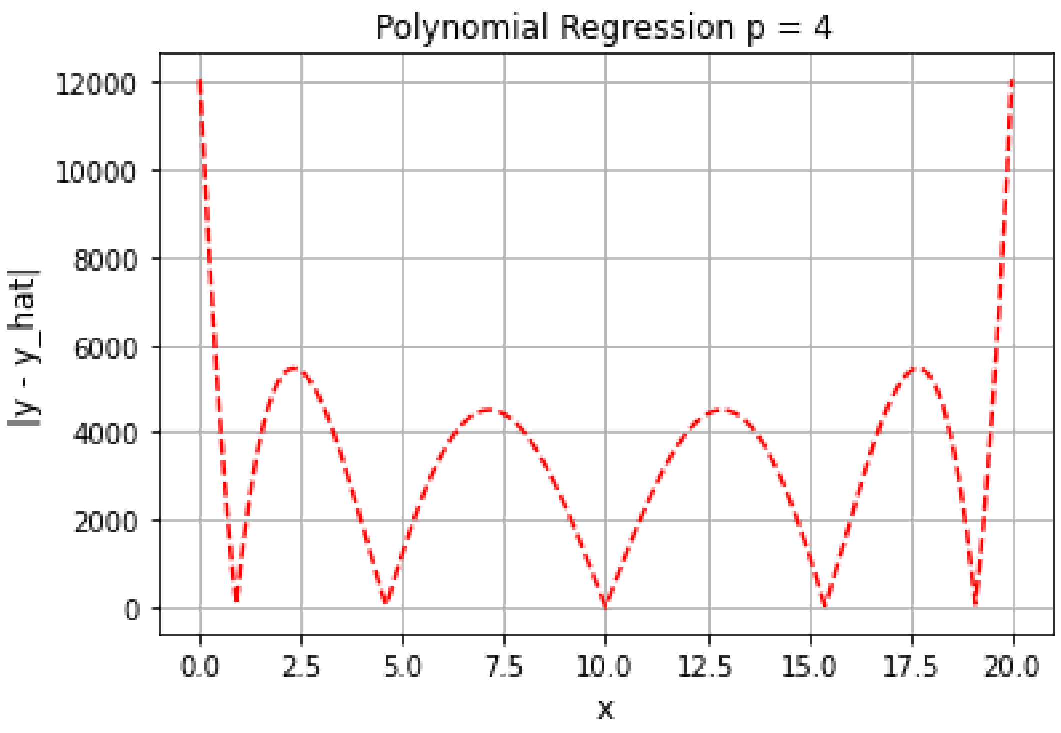
[2]:
```
# function to calculate polynomial regression
def polynomialRegression(X,y):
    lin_reg = LinearRegression()
    lin_reg.fit(X,y)
    theta = lin_reg.coef_
    theta[0] = lin_reg.intercept_
    return theta

def driverFunction(p,x,y,df_ans):
    X = [x**(i) for i in range(p+1)]
    X = np.array(X).reshape(p+1, y.shape[0]).transpose()
    ans_LR = polynomialRegression(X,y)
    df_ans.append(ans_LR)
    y_hat_LR.append(np.matmul(X,np.transpose(ans_LR)))
    print(f'RMSE for p={p} Linear Regression: {np.sqrt(mse(y,y_hat_LR[-1]))}')

    plt.plot(x,np.abs(y-y_hat_LR[-1]),'r--')
    plt.grid()
    plt.xlabel('x',fontsize=12)
    plt.ylabel('|y - y_hat|',fontsize=12)
    plt.title(f'Polynomial Regression p = {p}',fontsize=12)
    plt.show()
```

[3]:
```
x = np.arange(0, 20.1, 0.1)
np.random.seed(0)
theta0 = np.random.randn()
real_theta = [(-1 + theta0)*1e5, -300, 8, -100, 3, 1]
y = 1*x**5 + 3*x**4 - 100*x**3 + 8*x**2 -300*x - 1e5 + theta0*1e5

df_ans = []; y_hat_NE = []; y_hat_LR= []
df_ans.append([i for i in range(6)])
df_ans.append(real_theta)
```

Result

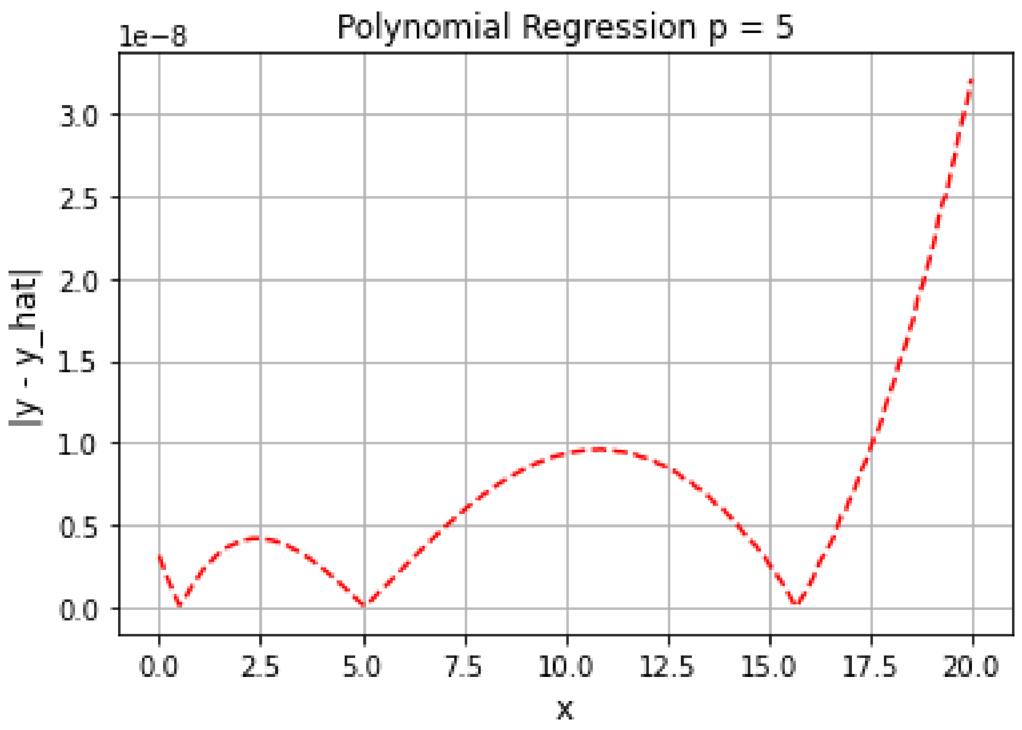
[4]:
```
driverFunction(4,x,y,df_ans)
```

```
RMSE for p=4 Linear Regression: 3922.723748085212
```



[5]:
```
driverFunction(5,x,y,df_ans)
```

```
RMSE for p=5 Linear Regression: 9.274475585343036e-09
```



[6]:
```
ans = pd.DataFrame(df_ans).T
ans.columns= ['Degree','Real ','LR p=4','LR p=5']
ans
```

[6]:
```
   Degree          Real         LR p=4        LR p=5
0     0.0  76405.234597   88479.758597  76405.234597
1     1.0   -300.000000 -19066.341289   -300.000000
2     2.0      8.000000   6641.833333      8.000000
3     3.0   -100.000000   -987.794444   -100.000000
4     4.0      3.000000     53.000000      3.000000
5     5.0      1.000000           NaN      1.000000
```

Observation

The values of the coefficients obtained when p=5 are almost equal to the real coefficients. The root mean squared error when p=5 is almost negligible but for p=4 it is quite significant.

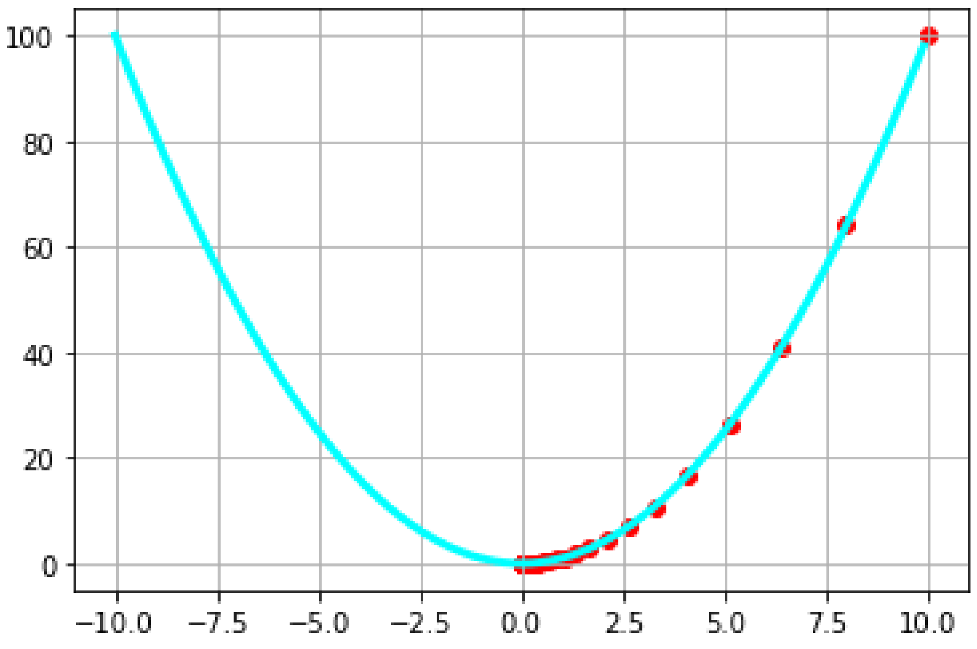
Question 2

Find minima of following functions using Gradient Descent method with learning rate 0.01 and 0.1 and different number of iterations. Try choosing a large value of learning rate and test the convergence. For L5(), use the data file.

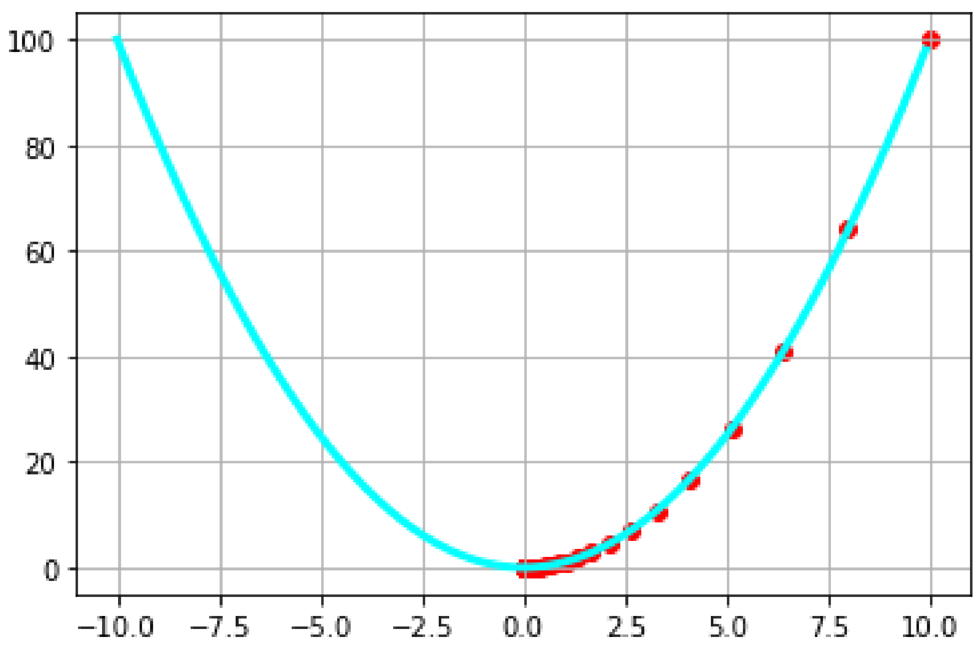
a. L1() = 2

[25]:
```
def L(theta):
    return theta*theta

def gradientDescent(iterations,alpha,startingVal):

    gradientPlot = [startingVal]
    for i in range(iterations - 1):
        gradientPlot.append(gradientPlot[-1] - alpha*2*gradientPlot[-1]);
    gradientPlot = np.asarray(gradientPlot)
    return gradientPlot

def driverFunc(alpha,iterations):
    theta = np.arange(-10,10,0.1)
    L_theta = []
    for t in theta:
        L_theta.append(L(t))

    iterPlot = gradientDescent(iterations,alpha,10)
    print('Minimum value of L() is', L(iterPlot[-1]), ' at  = ', iterPlot[-1])
    plt.figure()
    plt.plot(theta, L_theta, 'cyan', linewidth=3)
    plt.scatter(iterPlot,L(iterPlot),c='red')
    plt.grid()
    plt.show()

driverFunc(0.1,50)
driverFunc(0.1,100)
driverFunc(0.01,100)
driverFunc(0.01,1000)
```

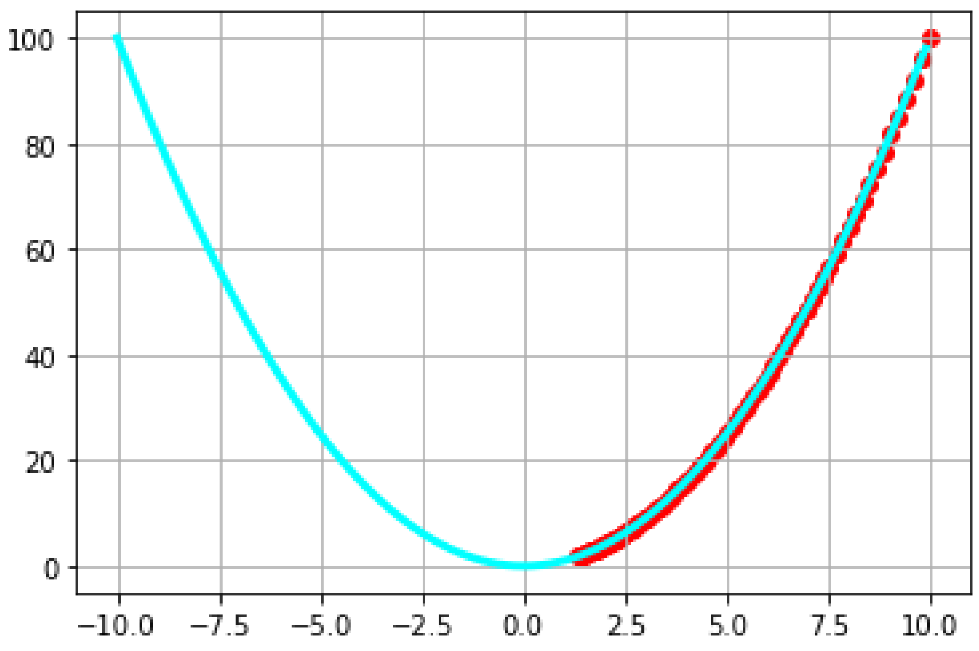
```
Minimum value of L() is 3.182868713022636e-08  at  =  0.00017840596158824503
```



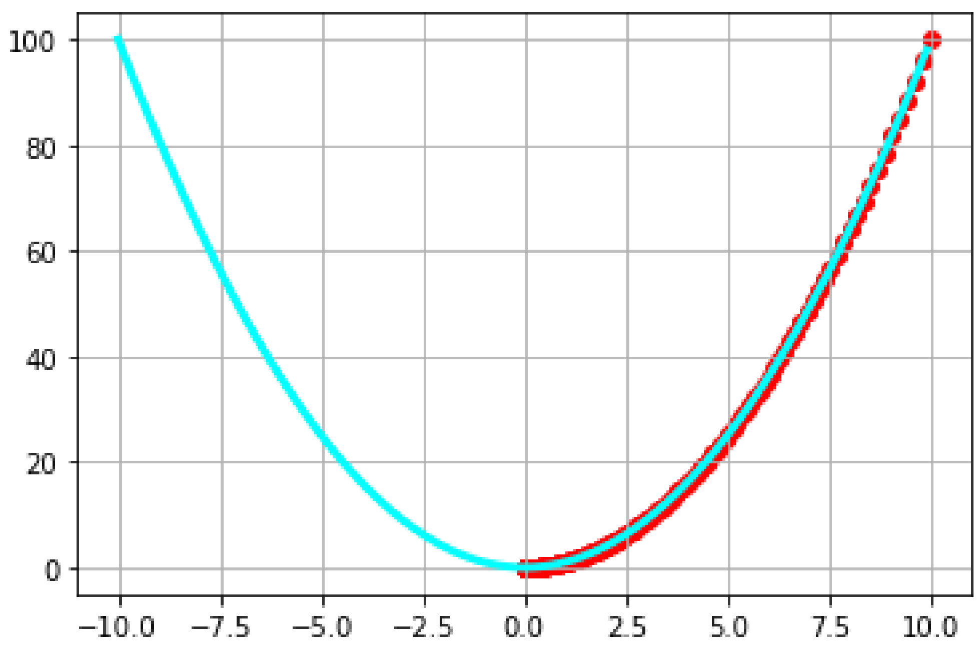
```
Minimum value of L() is 6.483618076376561e-18  at  =  2.5462949704181095e-09
```



```
Minimum value of L() is 1.8313147236278182  at  =  1.353260774436257
```



```
Minimum value of L() is 2.9491661031387704e-16  at  =  1.7173136298122048e-08
```



b. L2() = 12 + 22

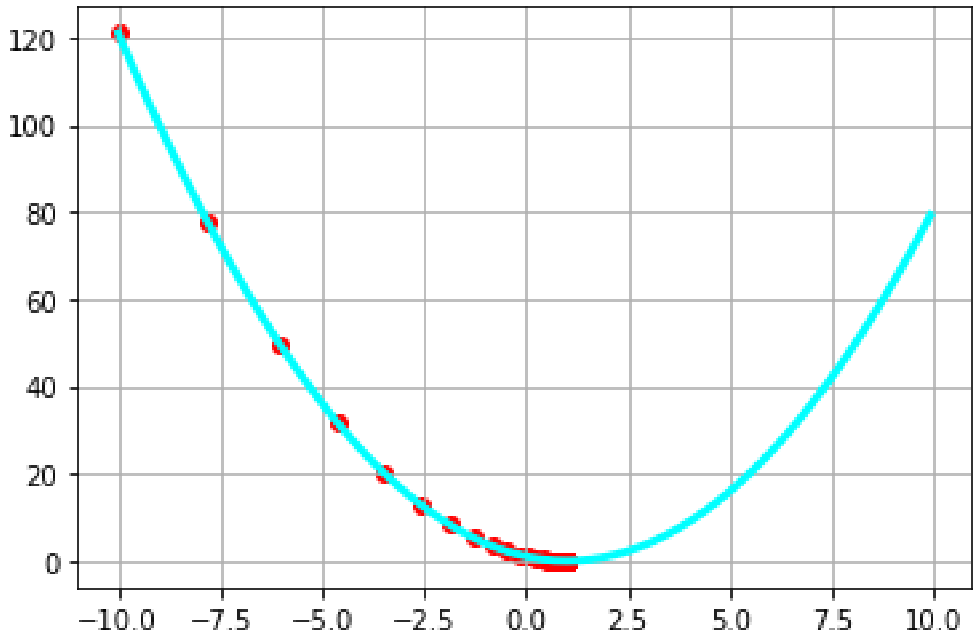
```
[26]: def L(theta1, theta2):
          return theta1*theta1 + theta2*theta2

      def gradientDescent(iterations,alpha,startingVal):

          gradientPlot1 = [startingVal]
          gradientPlot2 = [startingVal]

          for i in range(iterations - 1):
              gradientPlot1.append(gradientPlot1[-1] - alpha*2*gradientPlot1[-1])
              gradientPlot2.append(gradientPlot2[-1] - alpha*2*gradientPlot2[-1])

          return np.asarray(gradientPlot1) , np.asarray(gradientPlot2)

      def driverFunc(alpha,iterations):
          theta1 = np.arange(-10,10,0.1)
          theta2 = np.arange(-10,10,0.1)

          ans1,ans2  = gradientDescent(iterations,alpha,10)

          print('Minimum value of L(1, 2) is', L(ans1[-1],ans2[-1]),' at 1 = ',␣
       ↪ans1[-1],' at 2 = ', ans2[-1])

      driverFunc(0.1,100)
      driverFunc(0.01,100)
      driverFunc(0.01,1000)
```

```
Minimum value of L(1, 2) is 1.2967236152753122e-17  at 1 =
2.5462949704181095e-09  at 2 =  2.5462949704181095e-09
Minimum value of L(1, 2) is 3.6626294472556364  at 1 =  1.353260774436257  at
2 =  1.353260774436257
Minimum value of L(1, 2) is 5.898332206277541e-16  at 1 =
1.7173136298122048e-08  at 2 =  1.7173136298122048e-08
```

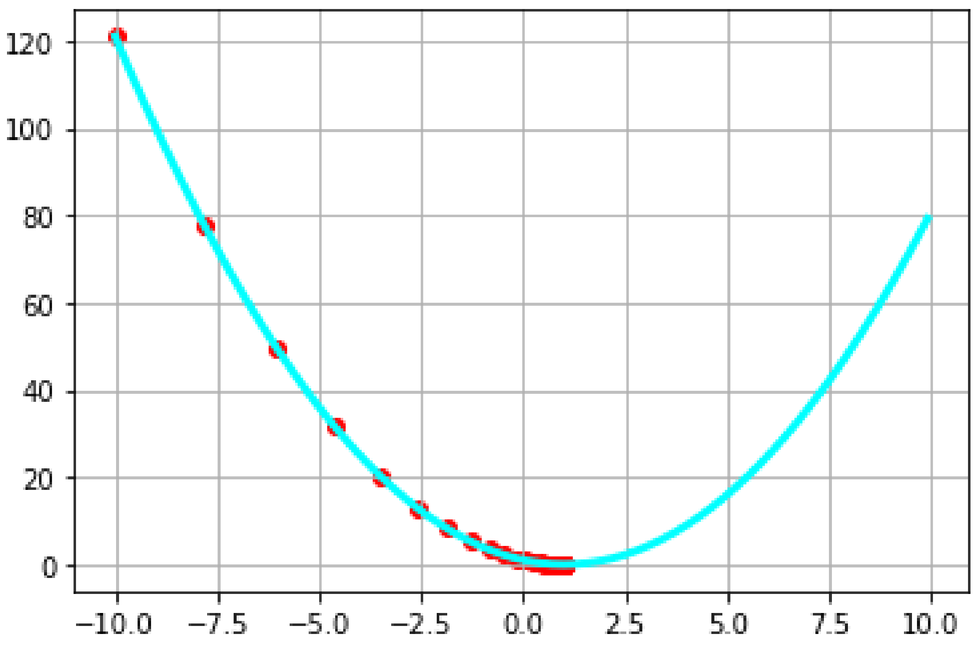
c. L3() = (-1)2

```
[27]: def L(theta):
          return (theta-1)**2


      def gradientDescent(iterations,alpha,startingVal):

          gradientPlot = [startingVal]
          for i in range(iterations - 1):
              gradientPlot.append(gradientPlot[-1] - alpha*2*(gradientPlot[-1]-1))
          gradientPlot = np.asarray(gradientPlot)
          return gradientPlot

      def driverFunc(alpha,iterations):
          theta = np.arange(-10,10,0.1)
          L_theta = []
          for t in theta:
              L_theta.append(L(t))

          iterPlot = gradientDescent(iterations,alpha,-10)
          print('Minimum value of L() is', L(iterPlot[-1]), ' at  = ', iterPlot[-1])
          plt.figure()
          plt.plot(theta, L_theta, 'cyan', linewidth=3)
          plt.scatter(iterPlot,L(iterPlot),c='red')
          plt.grid()
          plt.show()

      driverFunc(0.1,50)
      driverFunc(0.1,100)
      driverFunc(0.01,1000)
```

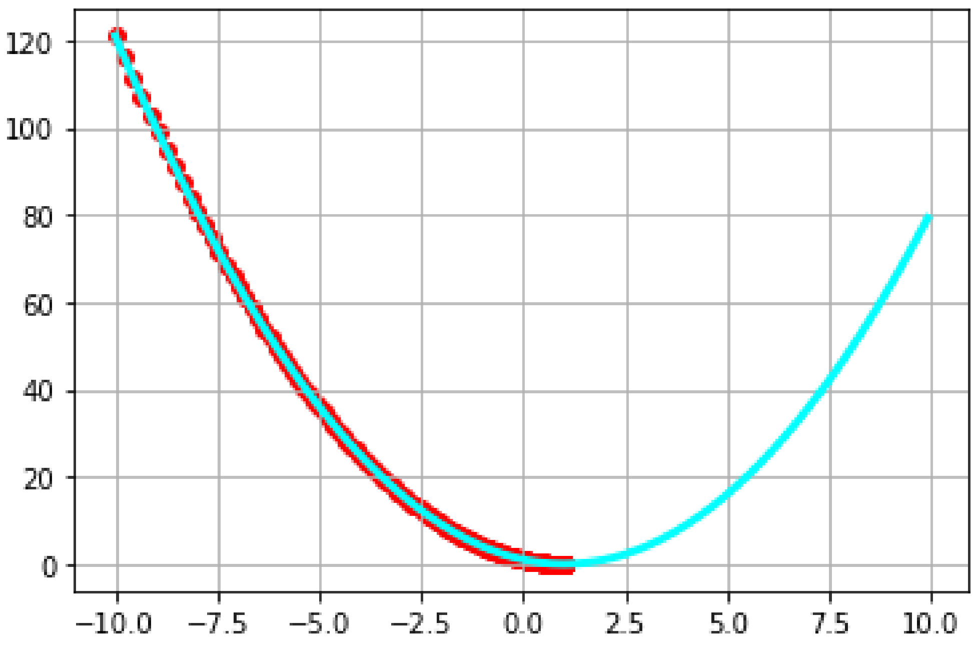
```
Minimum value of L() is 3.8512711427578704e-08  at  =  0.9998037534422529
```



```
Minimum value of L() is 7.845178011795908e-18  at  =  0.9999999971990755
```



```
Minimum value of L() is 3.5684910052415354e-16  at  =  0.99999998110955
```



d. L4() = 2(1-1)2 + 2(2-1)2

[28]:
```
def L(theta1, theta2):
    return 2*((theta1-1)**2 + (theta2-1)**2)

def gradientDescent(iterations,alpha,startingVal):

    gradientPlot1 = [startingVal]
    gradientPlot2 = [startingVal]

    for i in range(iterations - 1):
        gradientPlot1.append(gradientPlot1[-1] - alpha*4*(gradientPlot1[-1]-1))
        gradientPlot2.append(gradientPlot2[-1] - alpha*4*(gradientPlot2[-1]-1))

    return np.asarray(gradientPlot1) , np.asarray(gradientPlot2)

def driverFunc(alpha,iterations):

    theta1 = np.arange(-10,10,0.1)
    theta2 = np.arange(-10,10,0.1)

    ans1,ans2  = gradientDescent(iterations,alpha,10)

    print('Minimum value of L(1, 2) is', L(ans1[-1],ans2[-1]),' at 1 = ',␣
 ↪ans1[-1],' at 2 = ', ans2[-1])

driverFunc(0.1,50)
# driverFunc(0.1,100)
driverFunc(0.01,50)
driverFunc(0.01,100)
```

```
Minimum value of L(1, 2) is 5.879861387323954e-20  at 1 =  1.0000000001212421
at 2 =  1.0000000001212421
Minimum value of L(1, 2) is 5.930971649595571  at 1 =  2.217679314269111  at
2 =  2.217679314269111
Minimum value of L(1, 2) is 0.10005738583696015  at 1 =  1.158159243989215
at 2 =  1.158159243989215
```

e. L5() = (y(i)– (0 + 1. x(i)))2 m i=1

[29]:
```
def L(x, y, theta0, theta1):
    return np.sum(np.square(y - (theta0 + theta1*x)))

def derivativeFunc1(x, y, theta_0, theta_1):
    return np.sum(y - (theta_0 + (theta_1*x)))

def derivativeFunc2(x, y, theta_0, theta_1):
    ans = 0
    for i in range(len(x)):
        ans = ans + x[i]*(y[i] - (theta_0 + theta_1 * x[i] ))
    return ans


def gradientDescent(iterations,alpha,startingVal1,startingVal2,x,y):

    gradientPlot1 = [startingVal1]
    gradientPlot2 = [startingVal2]

    for i in range(iterations - 1):
        gradientPlot1.append(gradientPlot1[i] +␣
 ↪alpha*2*derivativeFunc1(x,y,gradientPlot1[i],gradientPlot2[i]))
        gradientPlot2.append(gradientPlot2[i] +␣
 ↪alpha*2*derivativeFunc2(x,y,gradientPlot1[i],gradientPlot2[i]))

    return np.asarray(gradientPlot1) , np.asarray(gradientPlot2)
```

```
def driverFunc(alpha,iterations):

    theta0 = np.arange(-50,50,0.5)
    theta1 = np.arange(-1,1,0.01)
    L_theta0_theta1 = np.zeros((len(theta1),len(theta0)))
    data = pd.read_excel(r'C:\Users\HP\Desktop\Group_14_Lab_2\data.xlsx')
    X = data.x
    Y= data.y


    for i in range(len(theta1)):
        for j in range(len(theta0)):
            L_theta0_theta1[i][j] = L(data.x, data.y, theta0[j], theta1[i])


    ans1,ans2  = gradientDescent(iterations,alpha,50,0,X, Y)
    print('Minimum value of L(1, 2) is', L(data.x,data.y,ans1[-1],ans2[-1]),' at 1␣
 ↪= ', ans1[-1],' at 2 = ', ans2[-1])

driverFunc(1e-11,1000)
driverFunc(1e-12,1000)
driverFunc(1e-12,5000)
```

```
Minimum value of L(1, 2) is 1576.3391560950045  at 1 =  49.999997117838255
at 2 =  -0.008851835542843913
Minimum value of L(1, 2) is 3588.012370125311  at 1 =  49.99999767923025  at
2 =  -0.007344317765876041
Minimum value of L(1, 2) is 1576.3405601331215  at 1 =  49.999997166533326
at 2 =  -0.0088505764975341
```

Observation

From the graph we can observe that if we increase the learning rate, we take bigger steps towards the minima. Thus, we are able to converge quickly.

Moreover, if the learning rate is lower, then we do not reach the minima in some cases. In such situations we need to increase the number of iterations for the model to converge to the minima.

Question 3

Find minimum of the function L() = mi=1 (y(i)– (0 + 1. x(i)))2 using the Stochastic Gradient Descent method (Take the data from the data file.) Choose different learning rates and number of iterations.

[30]:
```
def L(x, y, theta0, theta1):
    return np.sum(np.square(y - (theta0 + theta1*x)))

def derivativeFunc1(x, y, theta_0, theta_1):
    ans = 0
    for i in range(len(x)):
        ans = ans + (y[i] - (theta_0 + theta_1 * x[i] ) )
    return ans

def derivativeFunc2(x, y, theta_0, theta_1):
    ans = 0
    for i in range(len(x)):
        ans = ans + x[i]*(y[i] - (theta_0 + theta_1 * x[i] ) )
    return ans


def gradientDescent(iterations,alpha,startingVal1,startingVal2,x,y,k):
    gradientPlot1 = [startingVal1]
    gradientPlot2 = [startingVal2]
    # k = 5
    for i in range(iterations - 1):
        tempIndex = np.random.randint(0,len(x),k)
        tempX = []
        tempY = []
        for j in tempIndex:
            tempX.append(x[j])
            tempY.append(y[j])
        gradientPlot1.append(gradientPlot1[i] +␣
 ↪alpha*2*derivativeFunc1(tempX,tempY,gradientPlot1[i],gradientPlot2[i]))
        gradientPlot2.append(gradientPlot2[i] +␣
 ↪alpha*2*derivativeFunc2(tempX,tempY,gradientPlot1[i],gradientPlot2[i]))

    return np.asarray(gradientPlot1) , np.asarray(gradientPlot2)


def driverFunc(alpha,iterations,k):

    theta0 = np.arange(-50,50,0.5)
    theta1 = np.arange(-1,1,0.01)
    data = pd.read_excel(r'C:\Users\HP\Desktop\Group_14_Lab_2\data.xlsx')
    ans1,ans2  = gradientDescent(iterations,alpha,50,0,data.x, data.y,k)
    print('Minimum value of L(1, 2) is', L(data.x,data.y,ans1[-1],ans2[-1]),' at 1␣
 ↪= ', ans1[-1],' at 2 = ', ans2[-1])

driverFunc(1e-11,10000,5)
driverFunc(1e-12,10000,5)
driverFunc(1e-12,5000,5)
driverFunc(1e-11,5000,1)
driverFunc(1e-12,500000,1)
```

```
Minimum value of L(1, 2) is 1576.381481992516  at 1 =  49.9999971714277  at
2 =  -0.008844920841848306
Minimum value of L(1, 2) is 12187.76527951023  at 1 =  49.99999829781062  at
2 =  -0.005389488209515883
Minimum value of L(1, 2) is 28722.459896942473  at 1 =  49.999998952966976
at 2 =  -0.003314030627328275
Minimum value of L(1, 2) is 12041.6709857974  at 1 =  49.99999829275912  at
2 =  -0.00541340499194373
Minimum value of L(1, 2) is 1576.3454318298009  at 1 =  49.99999716498985  at
2 =  -0.008849173147827443
```

Observation

As opposed to standard gradient descent wherein the samples are selected in a group, in stochastic gradient the data samples are selected randomly.

It can be observed that as we increase the number of iterations and the step size/learning rate the minimum value decreases.

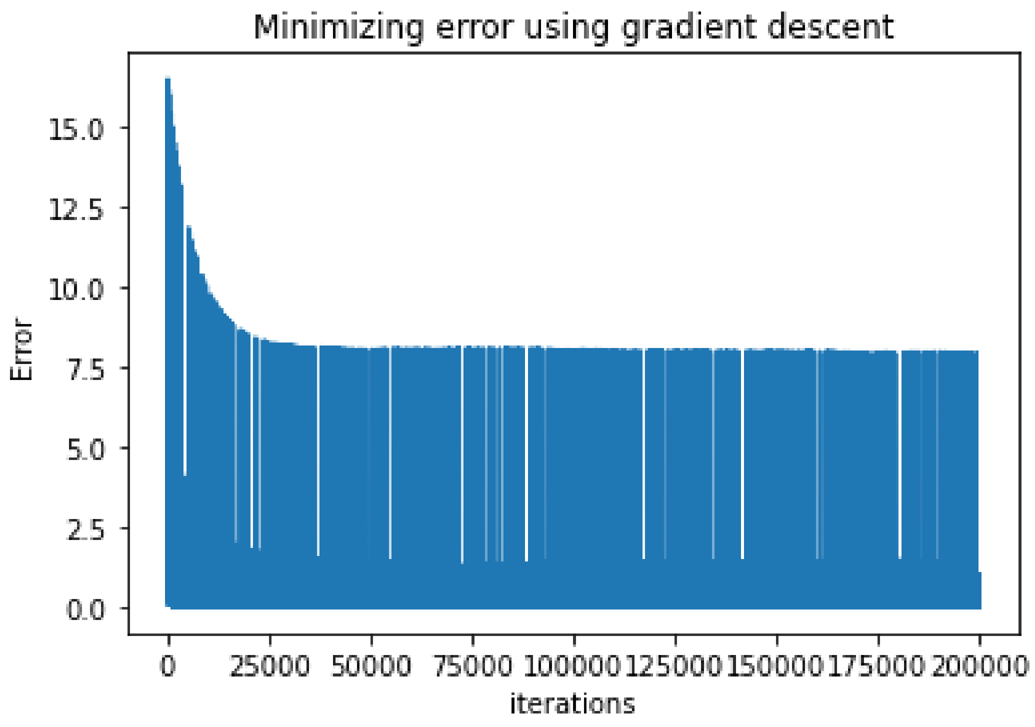
Question 4

Use optimal set of features found in question 6 (e) of assignment 1 for real estate price prediction regression problem and apply Stochastic Gradient Descent method for the same using these features.

[31]:
```
data = pd.read_excel(r'C:\Users\HP\Desktop\Group_14_Lab_2\realEstate.xlsx')
```

```
X = np.array(data.iloc[:,1:6])
Y = np.array(data.iloc[:,7])
m = np.size(Y)

ones = np.ones((m))
X = np.insert(X, 0, ones, axis=1)

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X[:,0] = 1

def s_gradient_descent(alpha,theta, itr,X,Y,m):
  i = 0
  diff = []
  while(i<itr):
    i = i+1;
    idx = np.random.choice(np.arange(len(X)), 1, replace=False)
    x_sample = X[idx]
    y_sample = Y[idx]
    error = (np.matmul(x_sample, theta)-y_sample)
    diff.append(np.sum(error**2)/(2*m))
    grad = np.matmul(np.transpose(x_sample),error)
    theta = theta - alpha*(grad)/m
  return theta,diff

alpha = 0.03
itr = 200000

theta = np.array([[0],[0],[0],[0],[0],[0]])
theta_opt,diff = s_gradient_descent(alpha,theta,itr,X,Y,m)
print('theta: ',theta_opt)

itr = np.arange(1,itr+.1,1)
plt.plot(itr,diff)
plt.xlabel('iterations')
plt.ylabel('Error')
plt.title('Minimizing error using gradient descent')
plt.show()
```

```
theta:  [[22.81924306]
 [ 6.34444629]
 [-3.84737912]
 [-9.57869074]
 [18.0505661 ]
 [16.55960195]]
```



Observation

From the graph it can be inferred that the error decreases as we increase the number of iterations. Beyond a certain number of iterations there is now significant decrease in error.

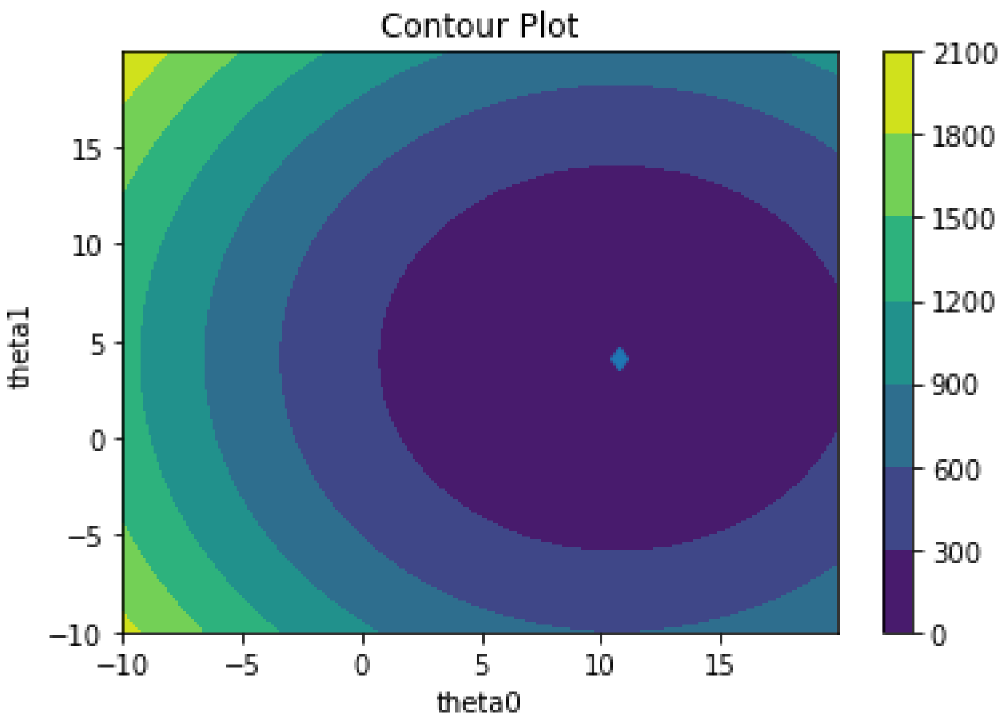
Question 5

The following question is to aid your understanding of gradient descent with the help of some visualization. Consider the data X and Y as given below.

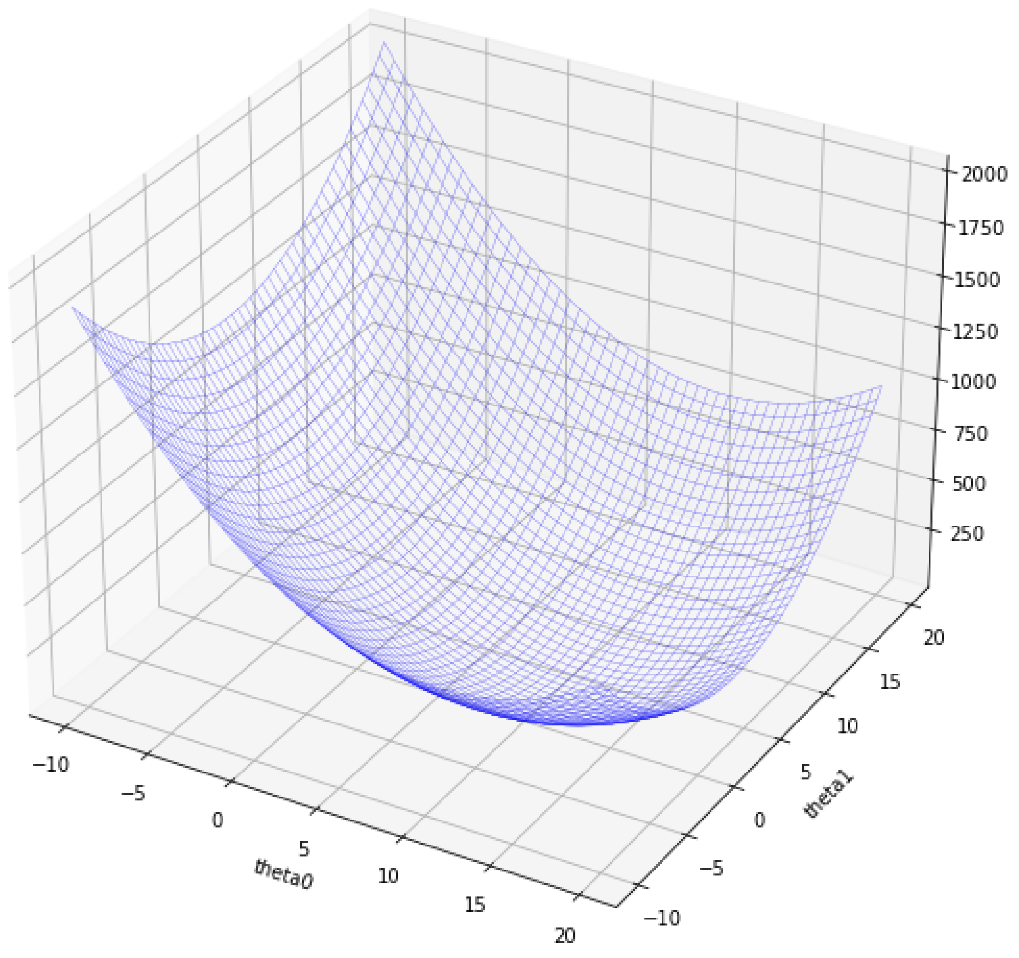
(a) Create a contour plot in the 0 and 1 space of residual sum of squares.

[32]:
```python
# 5a
x = np.array([[1],[3],[6]])
y = np.array(([6],[10],[16]))
mu = np.mean(x)
sig = np.std(x)
x = (x - mu)/sig
theta0 = np.arange(-10,20,0.1)
theta1 = np.arange(-10,20,0.1)
xx, yy = np.meshgrid(theta0, theta1)
L = np.zeros((np.size(theta1),np.size(theta0)))
for i in range(np.size(theta1)):
    for j in range(np.size(theta0)):
        L[i][j] = np.sum(np.square(y - (theta0[j] + (theta1[i]*x))))
minima = np.argwhere(L == np.min(L))
plt.contourf(xx, yy,L)
plt.colorbar()
plt.plot(theta0[minima[0,1]],theta1[minima[0,0]],'d')
plt.title('Contour Plot')
plt.xlabel('theta0')
plt.ylabel('theta1')
plt.show()
```

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')
ax.plot_wireframe(xx, yy, L, color='blue', linewidth=0.2)
plt.title('Wireframe Plot')
plt.xlabel('theta0')
plt.ylabel('theta1')
plt.show()
```



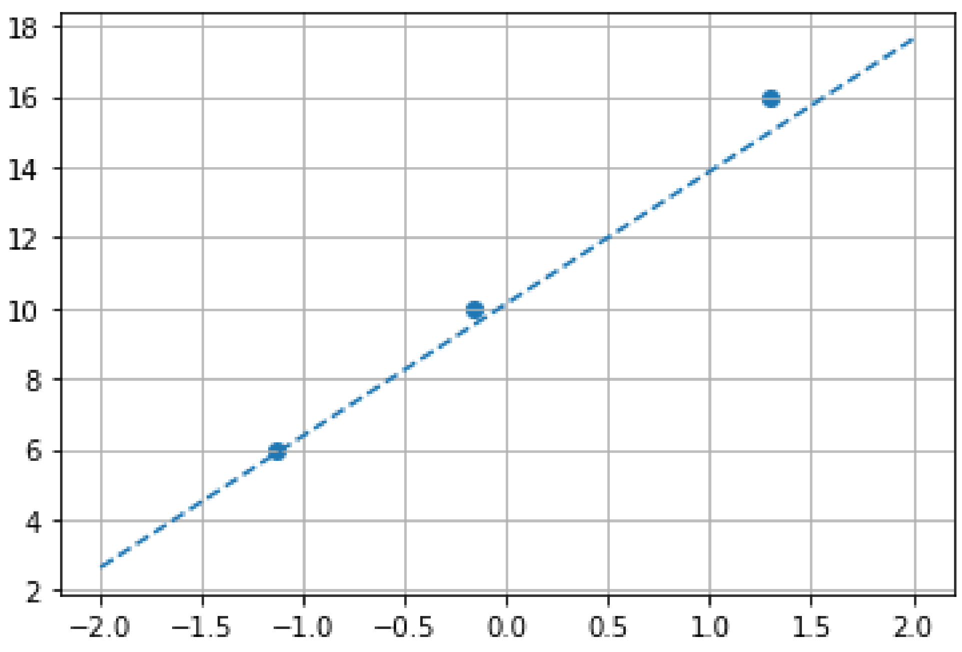
Wireframe Plot



(b) Create a Matplotlib animation where the plot contains two columns: the first one being the contour plot and the second one being the linear regression fit on the data. The different frames in the animation correspond to different iterations of gradient descent applied on the dataset to learn 0 and 1. For each iteration, draw the current value of q0 and q1 on the contour plot and also an arrow to the next q0 and q1 as learnt by gradient update rule. Correspondingly draw the y = 0 + 1 × x line on the other subplot showing the scatter plot. The overall title of the plot shows the iteration number and the residual sum of squares. You are free to use any gradient descent implementation i.e. your own or using libraries like scikit learn.

[33]:
```
#5b
temp = np.copy(x)
x2 = np.ones((np.size(temp,0),2))
x2[:,1] = temp[:,0]
iterations = 60
alpha = 0.01
theta = np.array([[-10],[-10]])
theta0_vals = [-10]
theta1_vals = [-10]
for i in range(iterations - 1):
 y_hat = np.dot(x2,theta)
 theta = theta - (alpha*2)*np.dot(x2.T,(y_hat - y))
 theta0_vals.append(theta[0,0])
 theta1_vals.append(theta[1,0])

print('the value of theta is:',theta)
plt.figure()
axes = plt.gca()
x_val = np.linspace(-2,2,100)
y_val = theta[0,0] + theta[1,0] * x_val
plt.plot(x_val, y_val, '--')
plt.scatter(x, y)
plt.grid()
plt.plot()
plt.show()
#GIF MAKER
filenames = []
for i in range(iterations):
 # plot the line chart
 plt.contourf(xx, yy,L)
 plt.colorbar()
 for j in range(i+1):
  plt.plot(theta0_vals[j],theta1_vals[j],'ro')
 plt.title('Contour Plot')
 plt.xlabel('theta0')
 plt.ylabel('theta1')

  # create file name and append it to a list
 filename = f'{i}.png'
 filenames.append(filename)

 # save frame
 plt.savefig(filename)
 plt.close()# build gif

with imageio.get_writer('Ques5.gif', mode='I') as writer:
 for filename in filenames:
   image = imageio.imread(filename)
   writer.append_data(image)

# Remove files
for filename in set(filenames):
 os.remove(filename)
```

```
the value of theta is: [[10.12986507]
 [ 3.74312252]]
```



Observation

In a contour plot, for points on the same contour, the value of the function is same. As observed in the G.I.F. file included, after applying gradient descent the coefficients start getting optimised with each iteration. Eventually, the model converges and we get a good estimation of the coefficients.