MACHINE LEARNING – 2CS501

PRACTICAL 4

Name: Bhanderi Mihir

Roll No.: 19BCE023

Batch No.: A-1

1) GRADIENT DESCENT WITH REGULARIZATION

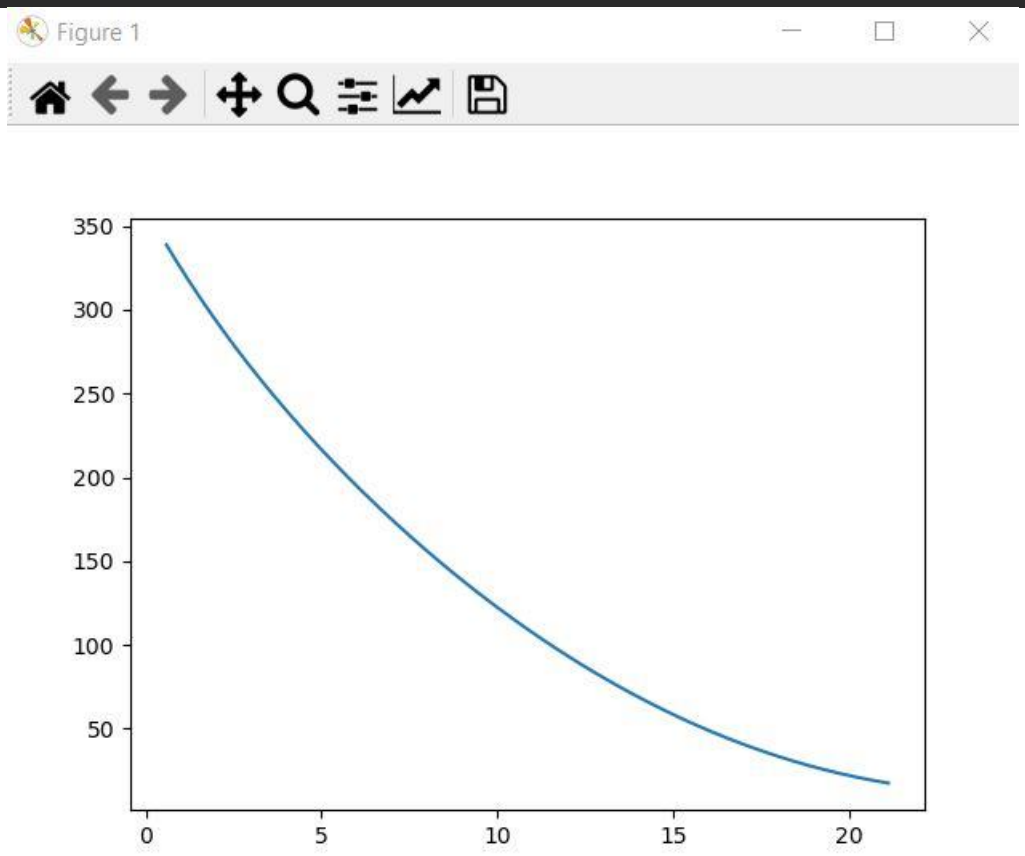
Code:

```
#Gradient Descent with Regularization

import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics, datasets
from sklearn.preprocessing import StandardScaler

def Plot_CF(theta1_Val, Cf):
    plt.plot(theta1_Val, Cf)
    plt.show()



if __name__ == '__main__':
    x, y = datasets.load_boston(return_X_y=True)

    x_train_temp1 = x[0:400, :]
    x_train = np.ones((x_train_temp1.shape[0], x_train_temp1.shape[1] + 1))
    x_train[:, 1:] = x_train_temp1
    y_train = y[:400]
    x_test_temp1 = x[400:506, :]
    x_test = np.ones((x_test_temp1.shape[0], x_test_temp1.shape[1] + 1))
    x_test[:, 1:] = x_test_temp1
    y_test = y[400:506]

    scaler = StandardScaler()
    scaler.fit(x_train[:, 1:])
    x_train[:, 1:] = scaler.transform(x_train[:, 1:])
    x_test[:, 1:] = scaler.transform(x_test[:, 1:])

    alpha = 0.002
    nitterations = 1000



    lamda = [-15,-10,10,100,500,1000]
    for j in lamda:
        m = x_train.shape[0]
        n = x_train.shape[1]
        theta = []
        theta = np.random.uniform(0, 1, n)
```

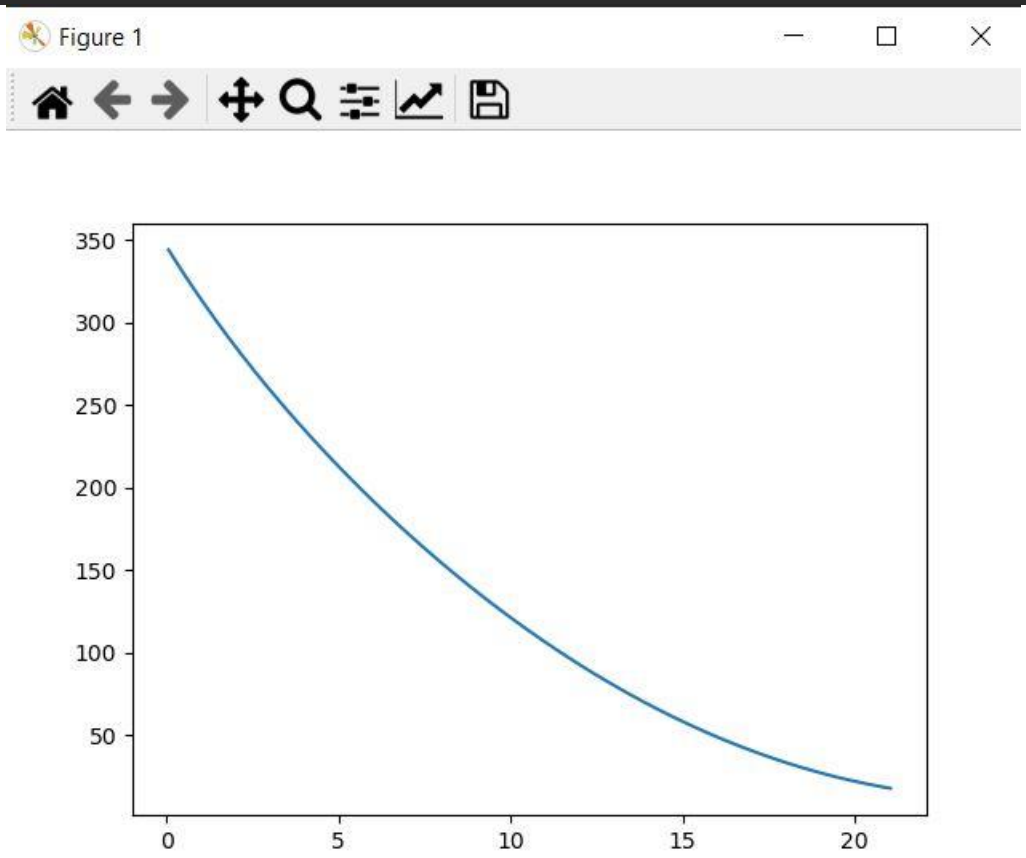
```
        theta1_Val = []
        Cf = list()
        for i in range(nitterations):
            update = np.zeros(x_train.shape[1])
            ypred = np.dot(x_train, theta)
            error = ypred - y_train

            update = np.sum(error * x_train.T,1)

            theta[0] = theta[0] - (1 / m) * alpha * update[0]
            theta[1:] = theta[1:]*(1 - alpha*j/m) - (1 / m) * alpha * update[1:]
            theta1_Val.append(theta[0])
            # Cost-Function
            sum1 = 0
            for k in range(m):
                sum2 = 0
                for i in range(n):
                    sum2 += theta[i] * x_train[k, i]
                sum1 += (sum2 - y_train[k]) ** 2

            Cf.append(sum1 * (1 / (2 * m)))
        print("\n\nLamda value : ", j)
        print("Theta : ", theta)

        #print("CF : ", Cf)
        prediction = np.dot(theta, x_test.T)
        print("MAE: ", metrics.mean_absolute_error(y_true=y_test, y_pred=prediction))
        print("MSE: ", metrics.mean_squared_error(y_true=y_test, y_pred=prediction))
        predytrain = theta[0] + theta[1] * x_train[:, 1]
        predytest = theta[0] + theta[1] * x_test[:, 1]

        print("\nPlotting of Graphs : ")
        print("""        1. Cost Function
        2. EXIT""")
        while True:
            n = int(input("Enter What to Plot : "))
            if n == 1:
                Plot_CF(theta1_Val, Cf)
            else:
                break
```

Output:

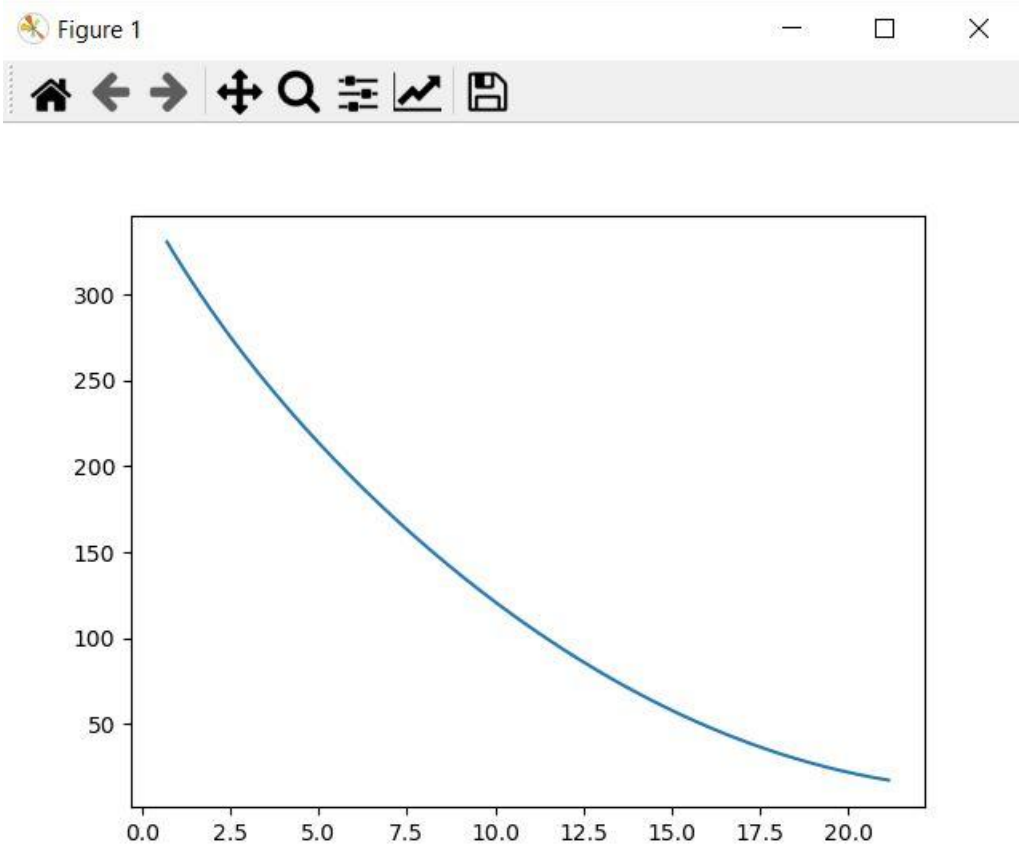
```
Lamda value :  -15
Theta :  [ 2.11358731e+01 -4.99896189e-01  6.15822501e-01 -4.29173704e-01
  9.38298856e-01 -1.94899283e-01  4.07744857e+00  2.08215457e-01
 -1.18062851e+00  3.46482902e-01  2.14866517e-03 -1.40289926e+00
  3.86211395e-01 -2.93386087e+00]
MAE:  3.4871863211923855
MSE:  22.753253190052416

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
```



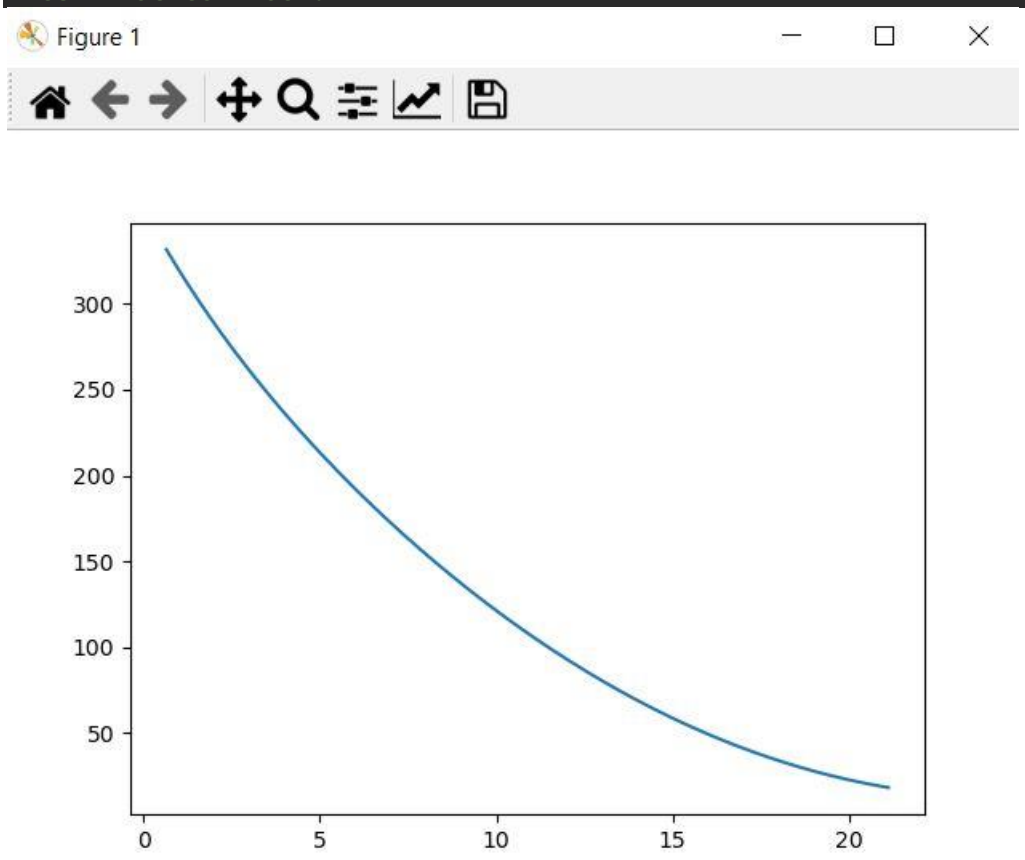
```
Lamda value :  -10
Theta :  [21.18217478 -0.82857956  0.79938978 -0.48061016  0.86893462 -0.11569201
  3.887665    0.28164522 -1.15551859  0.83201805 -0.21175951 -1.53631325
  0.38594588 -2.89095192]
MAE:  3.4999819378935895
MSE:  23.050030661876832

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
```



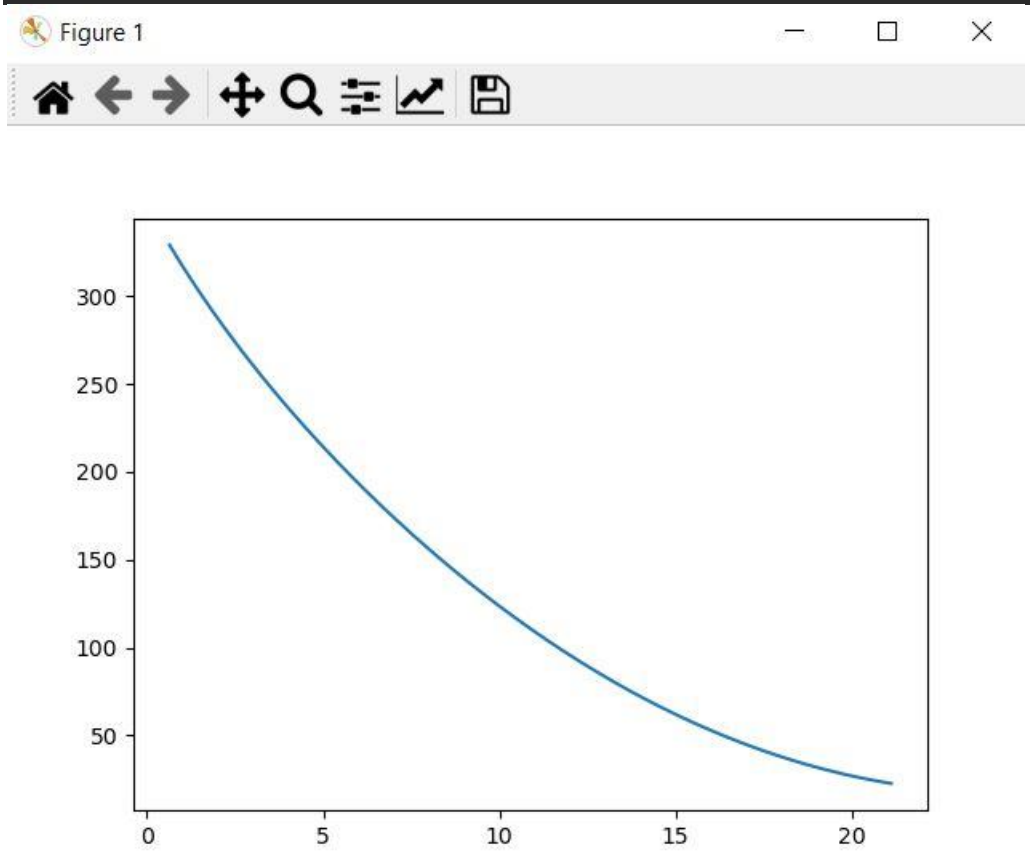
```
Lamda value :  10
Theta :  [21.11571453 -0.69834258  0.63924463 -0.17394759  0.84889403 -0.1801215
  3.93091899 -0.18560462 -1.23394348  0.99685336 -0.6274013  -1.45739386
  0.25539001 -2.71397685]
MAE:  3.4789702824785826
MSE:  21.51418379179827

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
```



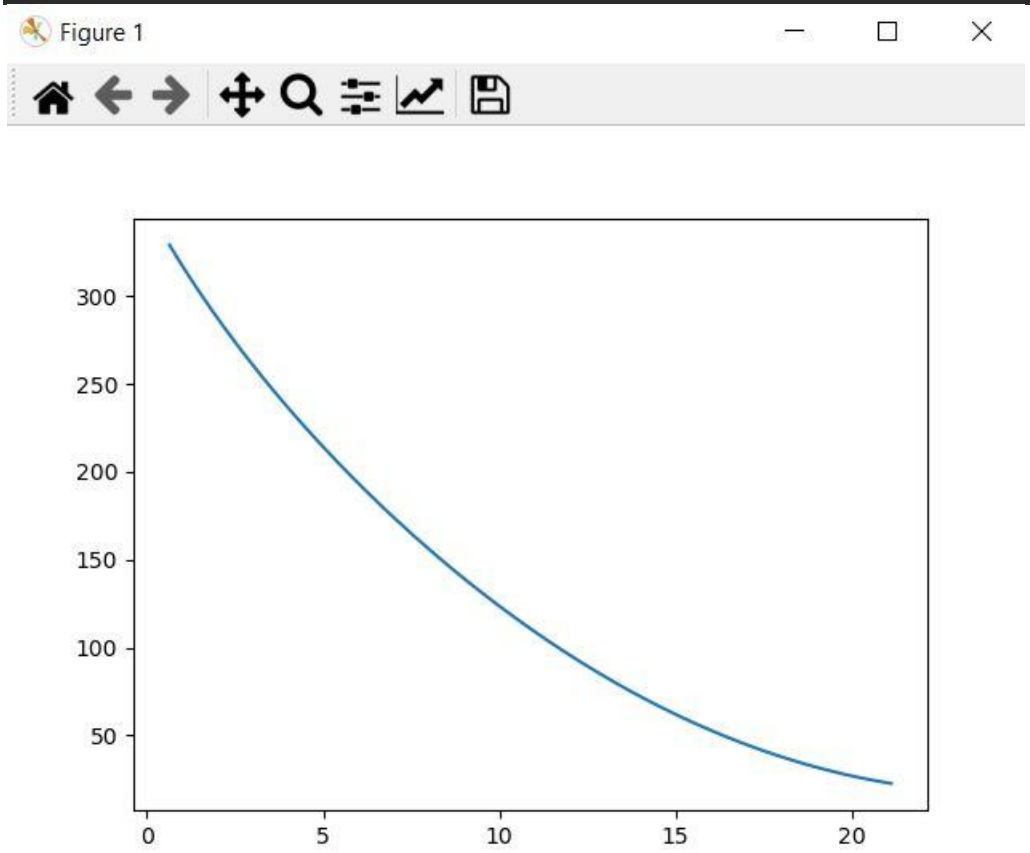
```
Lamda value :  100
Theta :  [ 2.11517152e+01 -4.65503021e-01  6.64602031e-01 -3.21945855e-01
  7.26071851e-01 -1.47736207e-02  3.32387287e+00  1.07535772e-01
 -8.89087706e-01  3.82038919e-01 -3.08512006e-01 -1.26185351e+00
  3.51112359e-01 -2.78414282e+00]
MAE:  3.2524983549075066
MSE:  19.685686942001226

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
```



```
Lamda value :  500
Theta :  [21.12761042 -0.40377998  0.43550672 -0.38769366  0.50913454 -0.27036742
  2.14938949 -0.18412111 -0.5084933   0.21721631 -0.35154178 -1.03575688
  0.21051185 -1.77336196]
MAE:  2.991860981702847
MSE:  14.98089534106851

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
```



```
Lamda value :  1000
Theta :  [ 2.10880659e+01 -3.15899531e-01  3.54741834e-01 -3.81190080e-01
  3.58640958e-01 -2.56573446e-01  1.49866782e+00 -2.07012907e-01
 -2.48152992e-01  1.58529328e-02 -2.88394107e-01 -7.72167346e-01
  1.74915670e-01 -1.27163940e+00]
MAE:  3.1637681531697393
MSE:  15.476894874055557

Plotting of Graphs :
        1. Cost Function
        2. EXIT
Enter What to Plot : 1
```



Observation:

As we increase value Lambda, errors start decreasing till a point after that particular value of lambda error terms also starts increasing.

Also if we don't perform Standardization the particular value of lambda where error term is minimal will be a very large value.

2) NORMAL EQUATION WITH REGULARIZATION

Code:

```
#Normal Equation with Regularization

import numpy as np
from sklearn import metrics, datasets
from sklearn.preprocessing import StandardScaler


if __name__ == '__main__':
    x, y = datasets.load_boston(return_X_y=True)

    x_train_temp1 = x[0:400, :]
    x_train = np.ones((x_train_temp1.shape[0], x_train_temp1.shape[1] + 1))
    x_train[:, 1:] = x_train_temp1

    y_train = y[:400]
    x_test_temp1 = x[400:506, :]
    x_test = np.ones((x_test_temp1.shape[0], x_test_temp1.shape[1] + 1))
    x_test[:, 1:] = x_test_temp1

    y_test = y[400:506]


    m = x_train.shape[0]
    n = x_train.shape[1]

    scaler = StandardScaler()
    scaler.fit(x_train[:, 1:])
    x_train[:, 1:] = scaler.transform(x_train[:, 1:])
    x_test[:, 1:] = scaler.transform(x_test[:, 1:])

    theta = np.zeros(x_train.shape[1])

    temp_matrix = np.zeros((x_train.shape[1], x_train.shape[1]))
    for i in range(1, x_train.shape[1]):
        temp_matrix[i, i] = 200  # Here as we increase lambda errors get decreasing
upto some extent

    XTXi = np.linalg.inv(np.dot(x_train.T, x_train) + temp_matrix)

    XTy = np.dot(x_train.T, y_train)
    theta = np.dot(XTXi, XTy)
    prediction = np.dot(theta, x_test.T)
    print("MAE: ", metrics.mean_absolute_error(y_true=y_test, y_pred=prediction))
    print("MSE: ", metrics.mean_squared_error(y_true=y_test, y_pred=prediction))
```

Output:

```
MAE:  3.8935297339273682
MSE:  22.4346470265702
```

Observation:

Here it is clear that there is drastic change in Error Term as compared to Normal Equation without Regularization