**Task1**

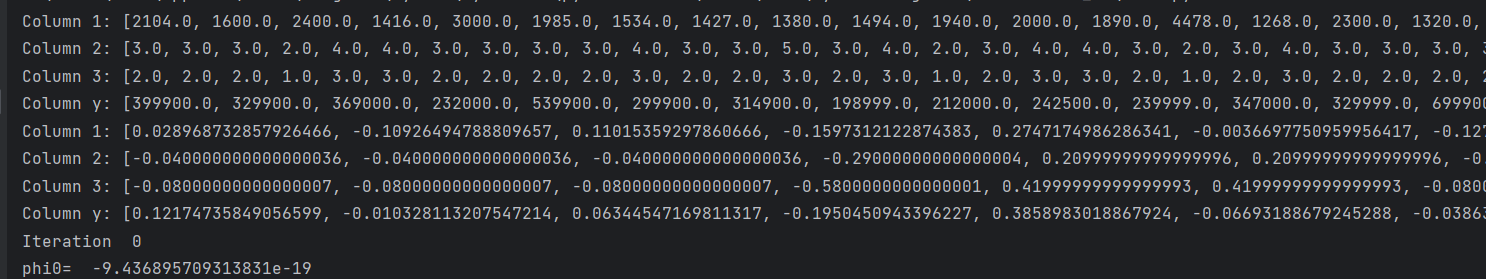
1. **Preprocess data:**

I used z-score method to make data standardized because the one parameter of the data set X is relatively contain large values than other parameters. So it makes large impact of one parameter large on result. So I standardized the data before applying linear regression.

**Code:**

maxvalx1 = max(x1)  
minvalx1 = min(x1)  
meanx1 = sum(x1)/50  
for x in x1:  
 normalx1.append(float((x - meanx1)/(maxvalx1 - minvalx1)))  
maxvalx2 = max(x2)  
minvalx2 = min(x2)  
meanx2 = sum(x2)/50  
for x in x2:  
 normalx2.append(float((x - meanx2)/(maxvalx2 - minvalx2)))  
maxvalx3 = max(x3)  
minvalx3 = min(x3)  
meanx3 = sum(x3)/50  
for x in x3:  
 normalx3.append(float((x - meanx3)/(maxvalx3 - minvalx3)))  
maxvaly = max(y)  
minvaly = min(y)  
meany = sum(y)/50  
for x in y:  
 normaly.append(float((x - meany)/(maxvaly - minvaly)))

**Result:**

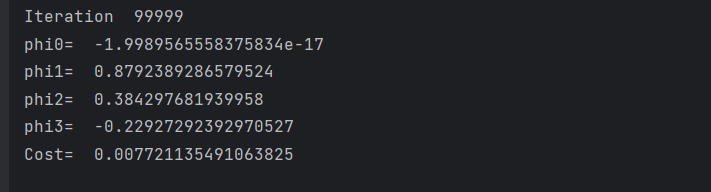


1. **Implement gradient descent algorithm with a learning rate = 0.02**

I apply the linear Gradient Descent Algorithm on the data to calculate the Thethas values. I perform about 100000 iterations on the data to find the convergence values of thethas.

**Code:**

for i in range(100000):  
 sum1 = 0  
 sum2 = 0  
 sum3 = 0  
 sum0 = 0  
 costsum = 0  
 for j in range(50):  
 hyp = (phi0 + phi1 \* normalx1[j] + phi2 \* normalx2[j] + phi3 \* normalx3[j])  
 sum0 = sum0 + (hyp - normaly[j])  
 sum1 = sum1 + (hyp - normaly[j]) \* normalx1[j]  
 sum2 = sum2 + (hyp - normaly[j]) \* normalx2[j]  
 sum3 = sum3 + (hyp - normaly[j]) \* normalx3[j]  
  
 costsum = costsum + (hyp - normaly[j])\*\*2  
  
  
 cost = (1/(2\*50)) \* costsum  
 phi0 -= learing\_rate \* (1/50) \* sum0  
 phi1 -= learing\_rate \* (1/50) \* sum1  
 phi2 -= learing\_rate \* (1/50) \* sum2  
 phi3 -= learing\_rate \* (1/50) \* sum3  
 print("Iteration " , i)  
 print("phi0= ", phi0)  
 print("phi1= ", phi1)  
 print("phi2= ", phi2)  
 print("phi3= ", phi3)  
 print("Cost= ", cost)

**Output:  
**

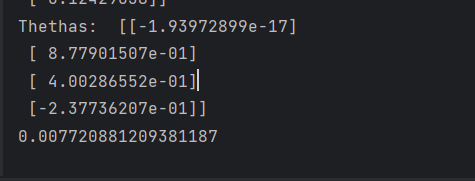
1. **Closed-form solution**

I apply the closed-form solution to the data set to find the Thethas value. First of all I make a matrix from data set X and Y. Then find the values using this calculations.

(X^T \*X)^-1 \* (X^T\*Y). At last I get the thethas values. From which I calculate the cost using predicted Y values and multiply it with X matrix.

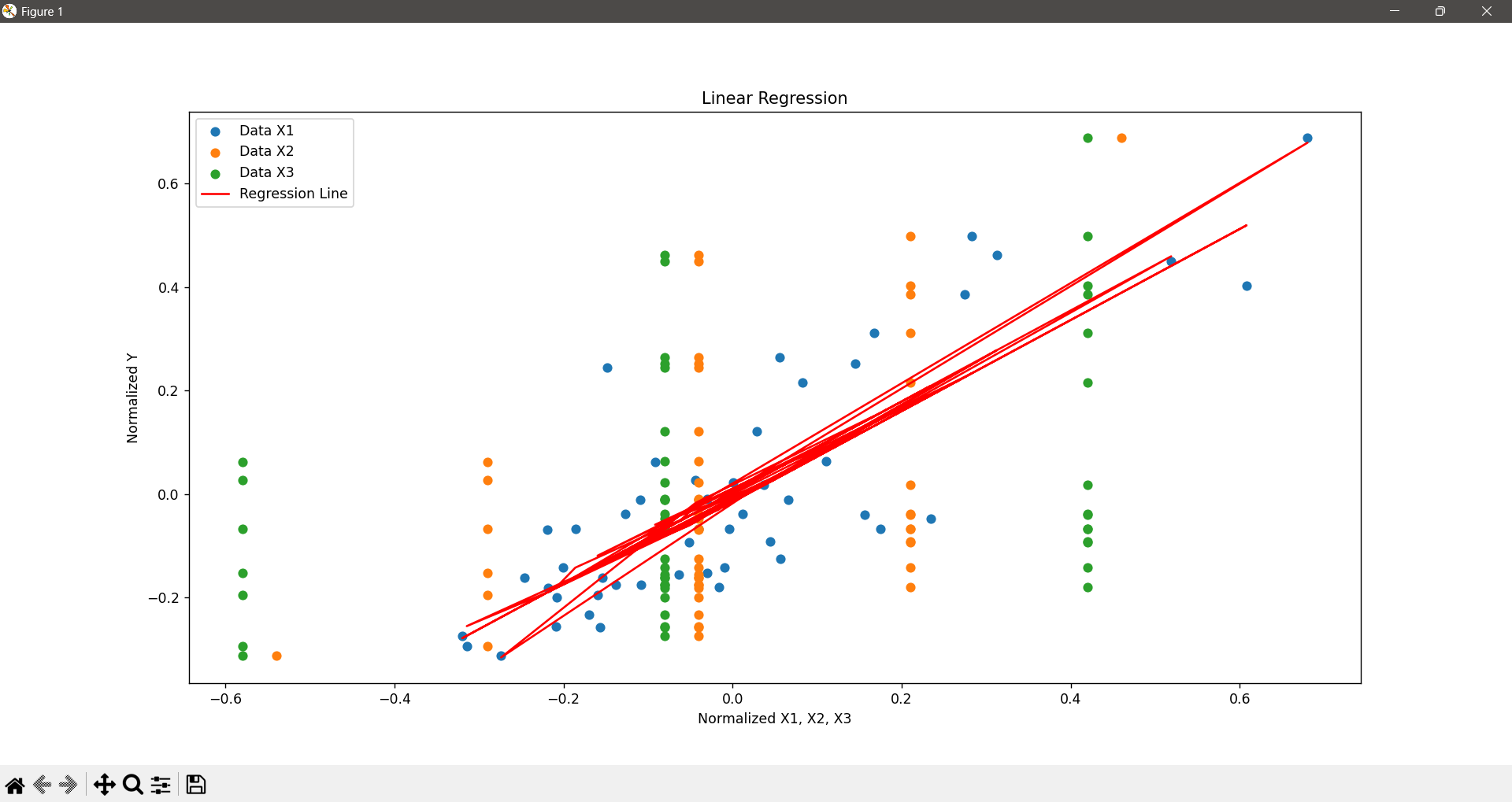
num\_rows = 50  
num\_columns = 4  
# Initialize an empty matrix with zeros  
MatrixA = [[0 for \_ in range(num\_columns)] for \_ in range(num\_rows)]  
  
# Initialize an empty matrix with zeros  
MatrixY = [[0 for \_ in range(1)] for \_ in range(50)]  
  
for i in range(50):  
 MatrixA[i][0] = 1  
 MatrixA[i][1] = normalx1[i]  
 MatrixA[i][2] = normalx2[i]  
 MatrixA[i][3] = normalx3[i]  
  
for i in range(50):  
 MatrixY[i][0] = normaly[i]  
  
mat = np.array(MatrixA)  
mty = np.array(MatrixY)  
print("Matrix A:",mat)  
print("Matrix Y:",mty)  
  
mt = mat.T  
  
res = np.dot(mt,mat)  
  
pseudo\_inverse = np.linalg.inv(res.T @ res) @ res.T  
  
XTY = np.dot(mt,mty)  
  
totalres = np.dot(pseudo\_inverse,XTY)  
  
  
y\_predicted = np.dot(mat, totalres)  
squared\_errors = (mty - y\_predicted) \*\* 2  
mse = (1 / (2 \* 50)) \* np.sum(squared\_errors)  
  
  
  
print("Thethas: ", totalres)  
print(mse)

**Output:**

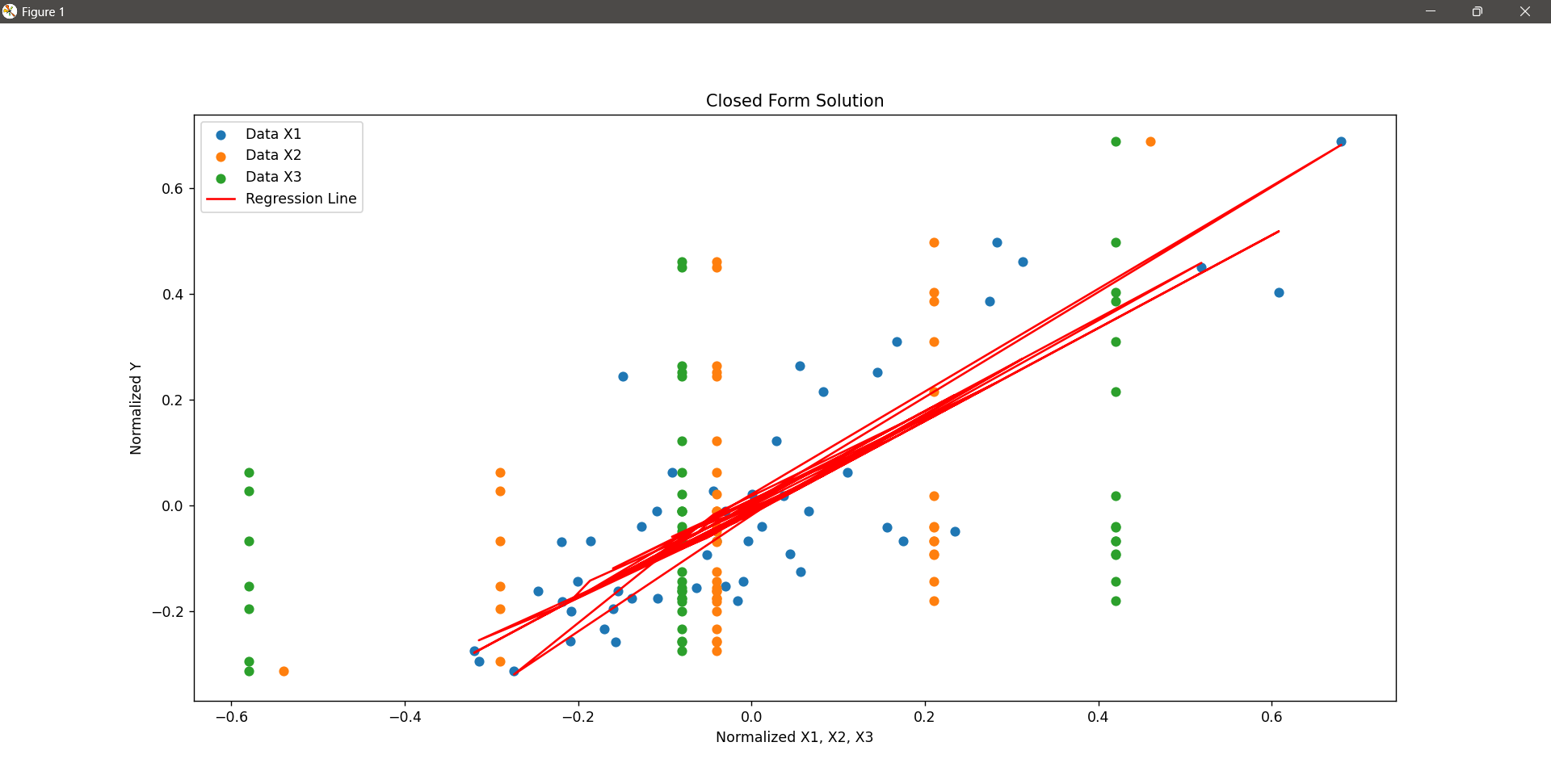


1. **Visualize the data and the results of the linear regression graphically**

**Linear regression graph:**

****

**Closed Form Solution:**

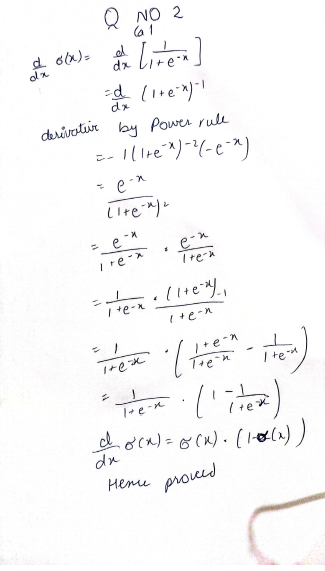
****

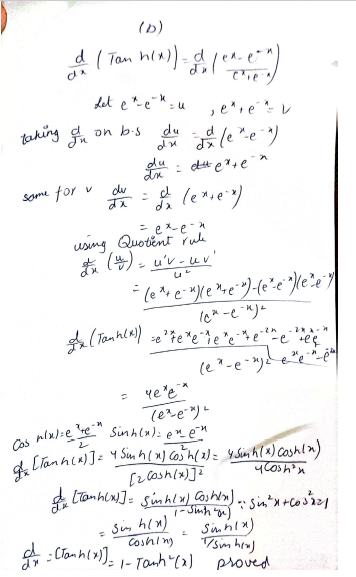
This graph shows the relationship between the normalized values of three variables (x1, x2, x3) and the residual error of a linear regression model. The model is obtained by using the closed form solution, which is a mathematical formula that directly computes the optimal values of the model parameters. The graph indicates that the model has a low error for most of the data points, except for some outliers that have a high error. The graph also suggests that the model is not very sensitive to the changes in the values of x1, x2, and x3.

1. **Results:**

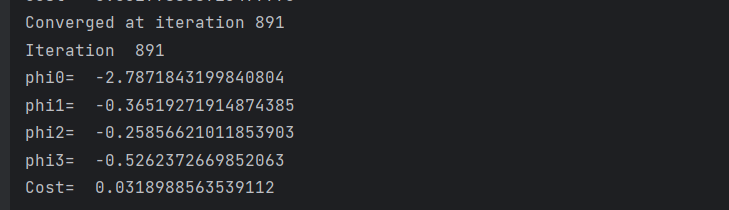
The result are that in simple linear regression using Gradient Descent we have to iterate the program for more than 100000 times to obtain minima values which take too much calculations and there is also a chance that we get only local minima which is not the correct solution. On the other hand the in closed form solution we get the global minima in just few steps which is closed to the minimum error. In this method the iterations are less and there are less calculations rather than linear regression using Gradient Descent.

**Task 2:**





**Task 3:**

****In this task we apply the logistic regression because the Data set Y have 2 outputs like 0 and 1. All the code is same as linear regression only the cost and hypothesis function gets changed.

**Code:**

import numpy as np  
import math  
file\_path = "DataX.dat"  
file\_path2 = "ClassY.dat"  
  
# Initialize empty lists to store the data for each column  
x1 = []  
x2 = []  
x3 = []  
y = []  
normalx1 = []  
normalx2 = []  
normalx3 = []  
normaly = []  
with open(file\_path2, 'r') as file:  
 for line in file:  
 # Assuming each line contains a single data point  
 data\_point = float(line.strip()) # Convert the line to a float  
 y.append(data\_point)  
  
with open(file\_path, 'r') as file:  
 for line in file:  
 values = line.strip().split()  
 x1.append(float(values[0]))  
 x2.append(float(values[1]))  
 x3.append(float(values[2]))  
  
maxvalx1 = max(x1)  
minvalx1 = min(x1)  
meanx1 = sum(x1)/50  
for x in x1:  
 normalx1.append(float((x - meanx1)/(maxvalx1 - minvalx1)))  
maxvalx2 = max(x2)  
minvalx2 = min(x2)  
meanx2 = sum(x2)/50  
for x in x2:  
 normalx2.append(float((x - meanx2)/(maxvalx2 - minvalx2)))  
maxvalx3 = max(x3)  
minvalx3 = min(x3)  
meanx3 = sum(x3)/50  
for x in x3:  
 normalx3.append(float((x - meanx3)/(maxvalx3 - minvalx3)))  
  
maxvaly = max(y)  
minvaly = min(y)  
meany = sum(y)/50  
for x in y:  
 normaly.append(float((x - meany)/(maxvaly - minvaly)))  
  
print("Column 1:", x1)  
print("Column 2:", x2)  
print("Column 3:", x3)  
print("Column y:", y)  
print("Column 1:", normalx1)  
print("Column 2:", normalx2)  
print("Column 3:", normalx3)  
  
phi0 = 0  
phi1 = 0  
phi2 = 0  
phi3 = 0  
learing\_rate = 0.02  
convergence\_threshold = 0.0001  
prev\_cost = float('inf')  
  
for i in range(10000):  
 sum1 = 0  
 sum2 = 0  
 sum3 = 0  
 sum0 = 0  
 cost = 0  
 for j in range(50):  
 hyp = phi0 + phi1 \* normalx1[j] + phi2 \* normalx2[j] + phi3 \* normalx3[j]   
 hypprob = 1 / (1 + math.exp(-hyp))  
 sum0 += (hypprob - normaly[j])  
 sum1 += (hypprob - normaly[j]) \* normalx1[j]  
 sum2 += (hypprob - normaly[j]) \* normalx2[j]  
 sum3 += (hypprob - normaly[j]) \* normalx3[j]  
 cost += normaly[j] \* math.log(hypprob) + (1 - normaly[j]) \* math.log(1 - hypprob)  
  
 cost = -cost / 50  
 phi0 -= learing\_rate \* (1/50) \* sum0  
 phi1 -= learing\_rate \* (1/50) \* sum1  
 phi2 -= learing\_rate \* (1/50) \* sum2  
 phi3 -= learing\_rate \* (1/50) \* sum3  
  
 if abs(prev\_cost - cost) < convergence\_threshold:  
 print("Converged at iteration", i)  
 break  
  
 prev\_cost = cost  
  
  
 print("Iteration " , i)  
 print("phi0= ", phi0)  
 print("phi1= ", phi1)  
 print("phi2= ", phi2)  
 print("phi3= ", phi3)  
 print("Cost= ", cost)  
  
  
print("Iteration " , i)  
print("phi0= ", phi0)  
print("phi1= ", phi1)  
print("phi2= ", phi2)  
print("phi3= ", phi3)  
print("Cost= ", cost)