1 Implementation

For the solution I have used Python version 3.8.10 . For operations related to linear algebra and to generate samples from Gaussian distribution with various parameters I am using *numpy*, to plot the graphs from results I am using *matplotlib*. I have implemented multiple functions to generate data and to perform Regression or Ridge Regression on the data and calculate the L2 distance between $\hat{\Theta}$ and Θ^*. The function to perform the regression is implemented in such a way that if $N < K$ then it will perform Ridge Regression otherwise it will perform Regression using OLS.

2 Graphs

I have noticed that when $N < K$, sometimes we are not able to find the inverse of the matrix $X^T X$ or even if we are able to find the inverse of the matrix, the values are too high which results in L2 distance being very high.

Below are some graphs for the L2 distance between $\hat{\Theta}$ and Θ^* for $N \in \{1 \dots 100\}$, $\lambda = 0.5$ and various values of K.

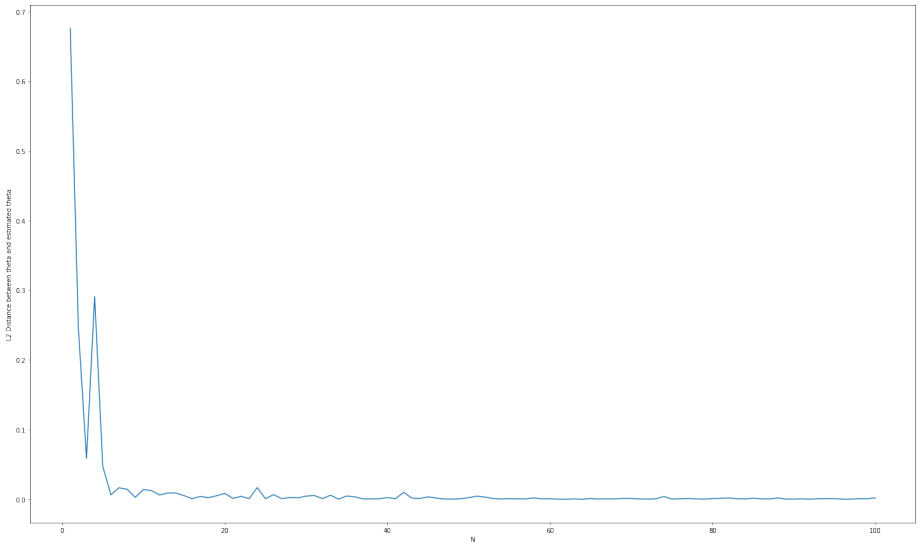

Figure 1: $K = 3$

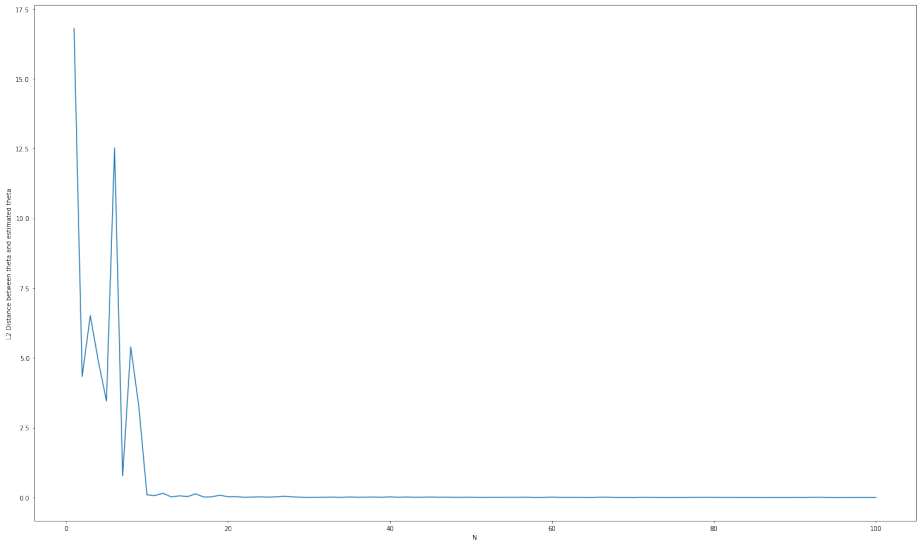

Figure 2: $K = 10$

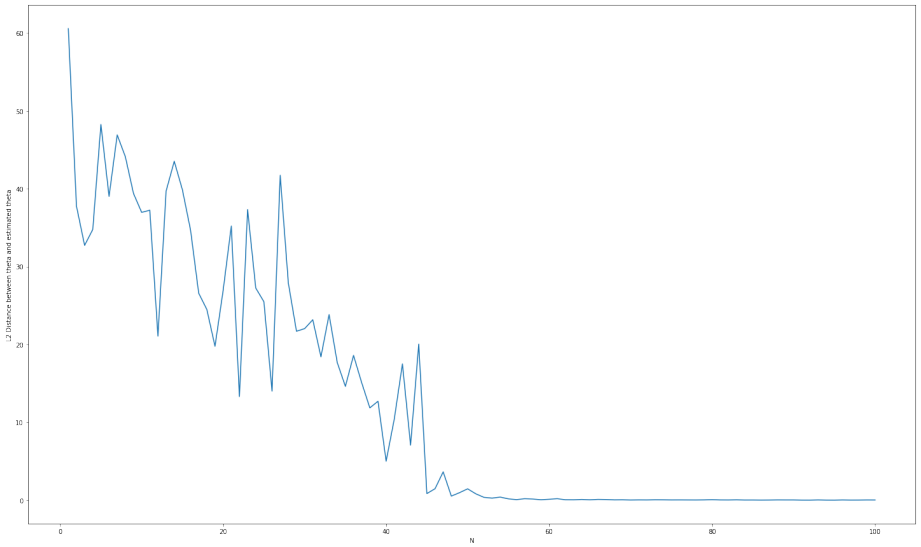

Figure 3: $K = 50$

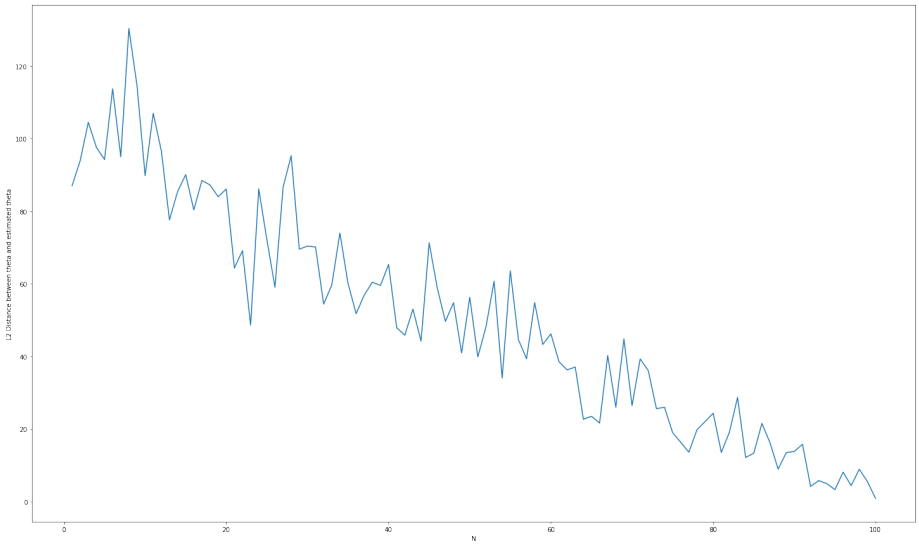

Figure 4: $K = 100$

As you can see, when $N < K$, Ridge Regression is used and L2 distance is high and as soon as we have $N \geq K$, the L2 distance decreases significantly (almost zero).

2.1 Ridge Regression

To find out how does the L2 distance between $\hat{\Theta}$ and Θ^* vary with the value of λ, I fixed values of N and K as 1000 and 50 respectively and calculated the L2 distance for values of λ in range $[0.001, 1.0]$ with 0.001 step increments. Below is the plot for same.

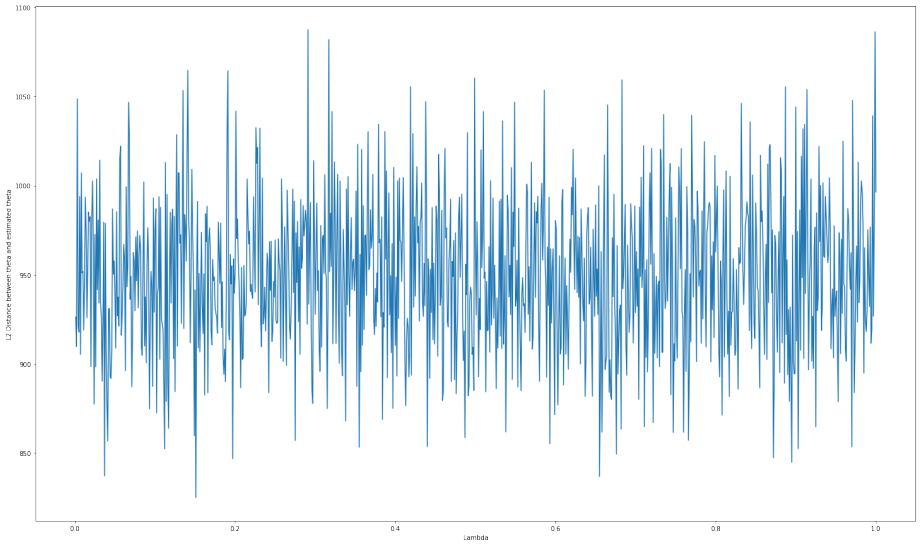

Figure 5: L2 distance between $\hat{\Theta}$ and Θ^* for various values of λ

It seems that there is no pattern in the L2 distance when we change the value of λ. I have tried large values of λ as well but the graph was similar.

A Python Code

```python
import numpy as np
import numpy.matlib
import math
import matplotlib.pyplot as plt

def generate_sample(K, Theta):
    X = np.random.default_rng().normal(0, 2, (1, K))
    Epsilon = np.random.default_rng().normal(0, math.sqrt(0.1))
    Y = np.dot(Theta, X.transpose()) + Epsilon
    return np.append(X, Y, 1)

def generate_data(N, K):
    Theta = np.random.default_rng().standard_normal((1, K))
    Samples = np.empty((0, K+1))
    for _ in range(N):
        Sample = generate_sample(K, Theta)
        Samples = np.append(Samples, Sample, axis=0)
    return Theta, Samples

def calculate_squared_error(Array):
    SquaredDiff = np.square(Array)
    Loss = np.sum(SquaredDiff)
    return Loss
```

```
24
25 def calculate_loss(X, Theta, Y):
26     return calculate_squared_error(np.matmul(X, Theta.transpose())
       - Y)
27
28 def run_simulation(N, K, Lambda):
29   Theta, Data = generate_data(N, K)
30   X, Y = Data[:, :-1], Data[:, -1].reshape(N, 1)
31
32   # if N is greater than or equal to K then solve OLS else use
       ridge regression
33   if N >= K:
34     ThetaStar = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.
       transpose(), X)), X.transpose()), Y).reshape(1, K)
35     # Loss = calculate_loss(X, ThetaStar, Y)
36     return calculate_squared_error(Theta - ThetaStar)
37   else:
38     RidgeTheta = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.
       transpose(), X) + Lambda ** 2 * np.matlib.identity(K)), X.
       transpose()), Y).reshape(1, K)
39     # RidgeLoss = calculate_loss(X, RidgeTheta, Y)
40     return calculate_squared_error(Theta - RidgeTheta)
41
42 KRange = [3, 10, 50, 100]
43 NRange = range(1, 101)
44 Lambda = 0.5
45
46 Results = []
47 for K in KRange:
48   Temp = []
49   for N in NRange:
50     Temp.append(run_simulation(N, K, Lambda))
51   Results.append(Temp)
52 Results = np.array(Results)
53
54 for i in range(len(KRange)):
55   fig, ax = plt.subplots()
56   fig.set_size_inches(25, 15)
57   ax.plot(NRange, Results[i])
58   ax.set_xlabel('N')
59   ax.set_ylabel('L2 Distance between theta and estimated theta')
60   plt.show()
61
62 Results = []
63 K = 1000
64 N = 50
65 LambdaRange = [x/1000 for x in range(1, 1001)]
66 for Lambda in LambdaRange:
67   Results.append(run_simulation(N, K, Lambda))
68 Results = np.array(Results)
69
70 fig, ax = plt.subplots()
71 fig.set_size_inches(25, 15)
72 ax.set_xlabel('Lambda')
73 ax.set_ylabel('L2 Distance between theta and estimated theta')
74 ax.plot(LambdaRange, Results)
75 plt.show()
```