Assignment 1 Semester 1 COL774 - 2025-2026

Aadi Govil - 2023CS10490

1. Linear Regression

For this part and the next, I have created functions to perform gradient descent, which work on both of them. Thus I have separated out loss, gradient and gradient descent functions, and have mostly kept plotting logic in the file for this question. Plotting was mostly done using meshgrids from numpy and contour functionality inbuilt in matplotlib.

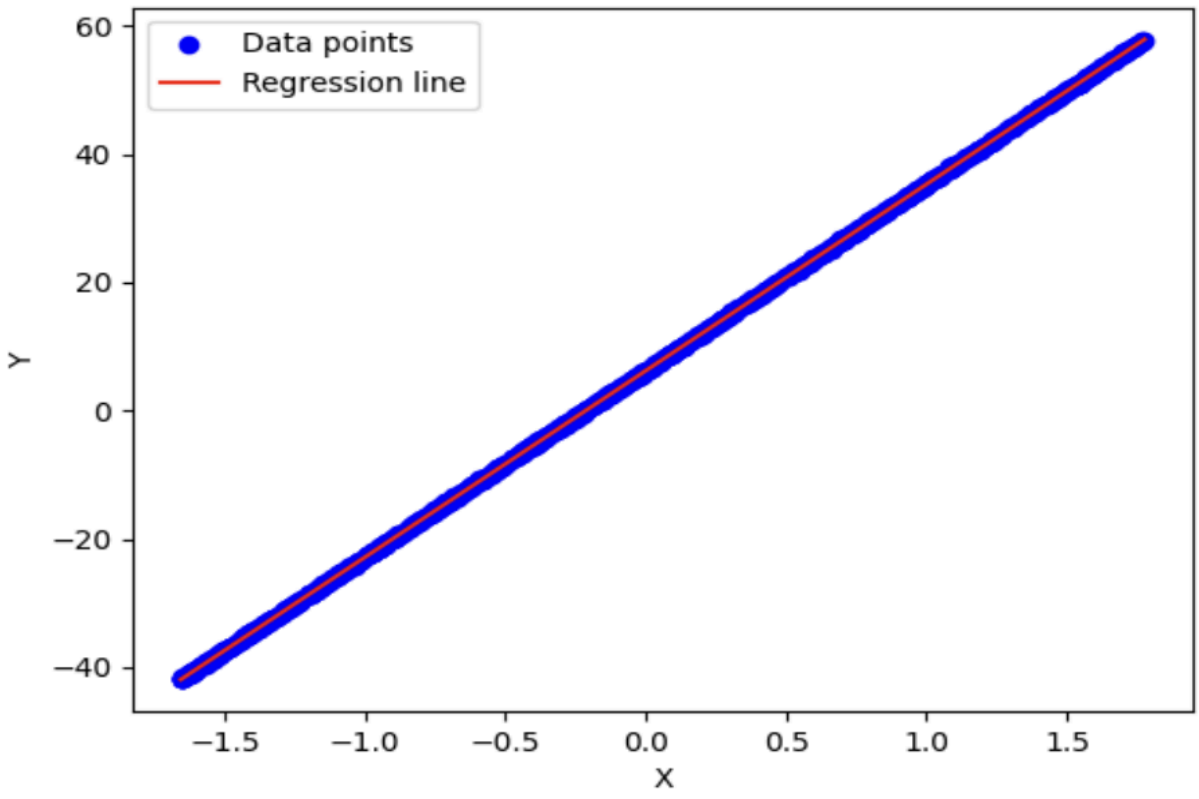
1. C. Learning Rate = 0.01

   Stopping criteria: Difference between consecutive J(θ) <= 10^(-9)

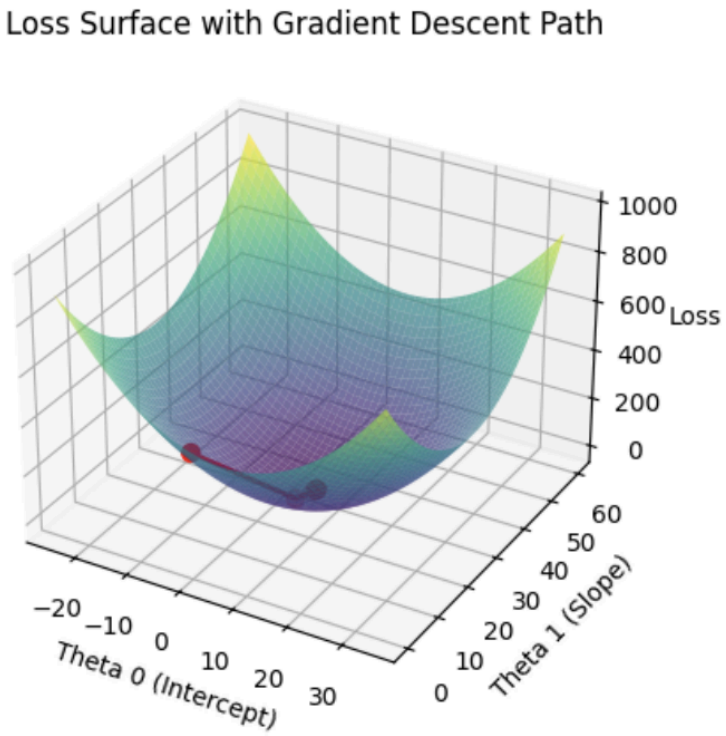
   Parameters : Intercept: 6.218612931112601

   Slope : 29.064447450305227

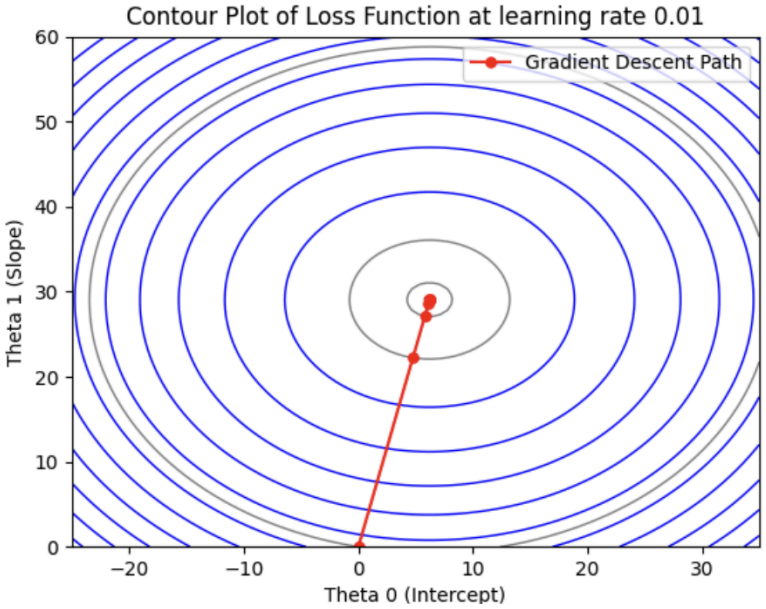
2.



3.

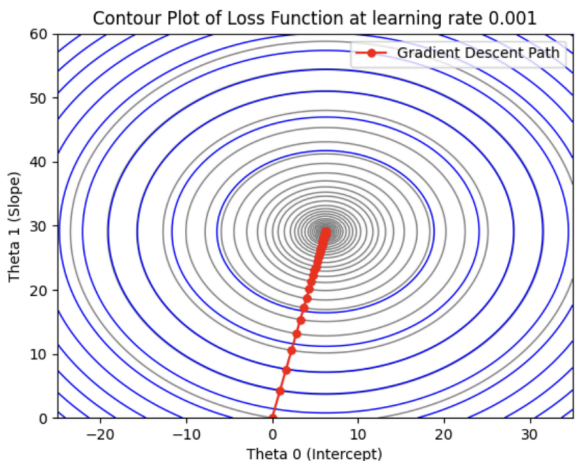


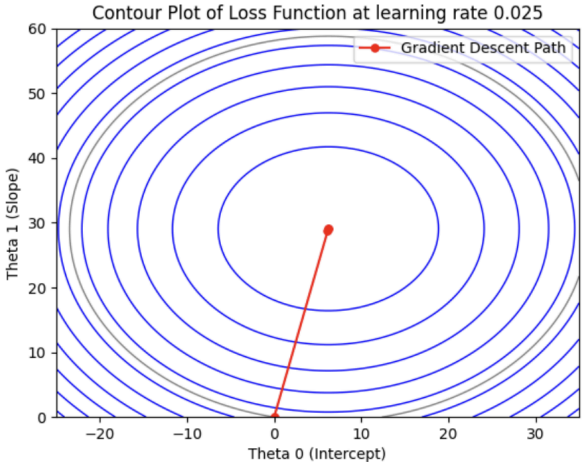
4.

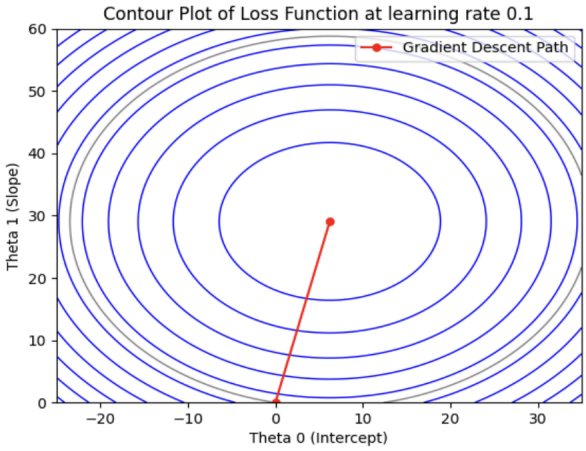


Grey circles represent contours at the points recorded every 0.2s
Blue lines represent other contours

5.







We observe in all these plots that as the learning rate increases, time taken to converge lessens. The number of iterations hence decreases with increase in the learning rate. This happens as the jump is always taken in the right direction, due to us using the whole data to calculate loss/gradient, as well as it being a quadratic function.
Iterations to converge for learning rate 0.001 = 10305
Iterations to converge for learning rate 0.01 = 1142
Iterations to converge for learning rate 0.025 = 472
Iterations to converge for learning rate 0.1 = 121
As we can see the number of epochs for convergence decreases as the learning rate is increasing.

2. Sampling, closed form and stochastic gradient descent

I have done generation of data, solving using closed form, analysis and plotting of data in q2.py. Implementation of gradient descent is in other files, as that is commonly shared between q1 and q2. I have used certain parameters decided on my own using experimental data for the window sizes and the max number of epochs that the gradient descent would run for, for each of the batch sizes. I have kept window sizes almost inversely proportional to the size of the batch.

2. C. For convergence I have kept the criterion taught to us in class, but I have modified it a little, to not divide by the window size (as the condition collapsed to $|J(\theta t) - J(\theta t - k)|/k < \varepsilon$) where k is the window size. For different batch sizes, I have taken different window sizes and as we are doing only 1 subtraction, it made sense to me to not divide by k while doing the comparison check. I ran the program with both and got better results in the case I did not divide by k. I have also kept a max epoch condition so that the program does not run indefinitely. This condition is only triggered for the 80 batch size case.

Results for different runs of the program with different batch sizes:

```
Batch size: 1, Parameters: [2.97050601 1.03287898 1.95842668], Training Loss: 1.0133515207994448, Test Loss: 1.008863564208852, Iterations to Converge: 221108
Batch size: 80, Parameters: [3.00016155 1.00540529 1.99827281], Training Loss: 1.0018766959399115, Test Loss: 0.9979344435393009, Iterations to Converge: 1000000
Batch size: 8000, Parameters: [2.96562999 1.00655577 1.99584048], Training Loss: 1.0017751509909836, Test Loss: 0.9979645044444326, Iterations to Converge: 16226
Batch size: 800000, Parameters: [2.88600714 1.02398453 1.99006669], Training Loss: 1.0034509920852388, Test Loss: 0.9996041214173605, Iterations to Converge: 11763
Closed form solution parameters: [2.9993048080130027, 0.9992328931829996, 1.9982642544527554], Training Loss: 1.0016128307447993, Test Loss: 0.9978160055104192
```

3. A. The different batch sizes converge to approximately the same value, except for the 800000 batch one. If we had used a lower epsilon for it, then it would have converged to a slightly better value, but in the case of the 800000 batch size, the moves are so small that it takes a very long time to converge.

Relative speed of convergence:
The relative speed of convergence decreases with increasing batch size, as is to be expected as 1 single move takes a higher amount of time to compute the gradient/loss for. This is a general trend, but we also observe that the rates of 1 and 80 batch size are similar. This might be due to the 1 batch size taking turns in the wrong direction more often due to its short batch size.

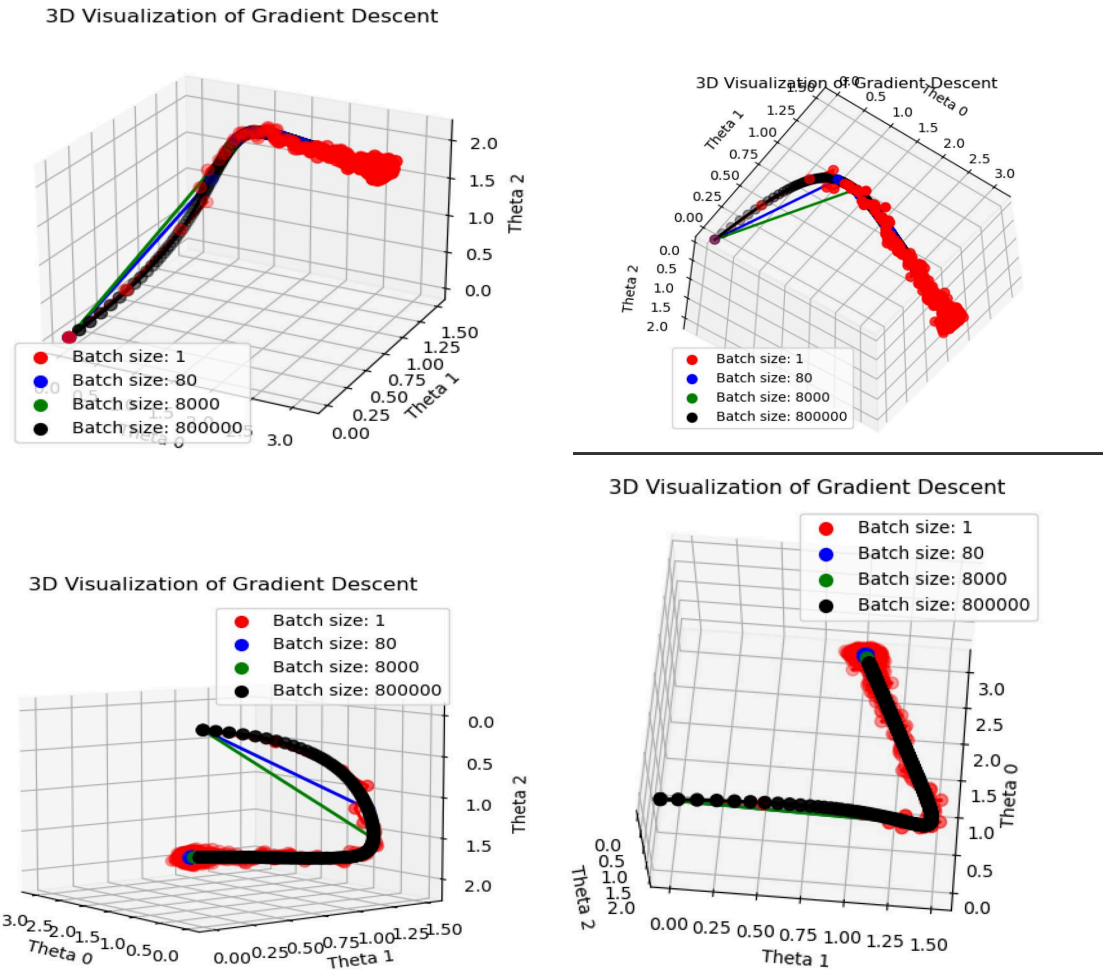
Number of iterations:
The number of iterations to converge in the above figure are the number of times that the parameters are changed. And we can clearly see the order for them.

C. The true parameters, ones observed in the close form, and ones observed by SGD are very nearby, showcasing that the modelling of the problem leads to accurate solutions via SGD/close form parameter finding.
The calculated parameters are closer to the true parameters in case of closed form calculations, than in the case of SGD, which is to be expected, given we objectively minimize the loss function in the case of closed form solutions. But one thing to note

over here is that the 80 batch size SGD has a closer θ0 and θ2 value to the true parameters, for which I do not have an explanation.

4. The test error and the training error are both written in the figure above. We can see that both of them do not differ significantly for any of the models, showcasing the fact that these models are able to train on given data without overfitting, and are able to perform well on the test data as well.

5. Due to the difference in the relative speeds of convergence, I have used different timescales to capture the data for different batch sizes.
For batch size 1 : 0.001 s
For batch size 80 : 0.005 s
For batch size 8000 : 0.1 s
For batch size 800000 : 0.2 s
The graphs I obtained for them are :



How the shape of the movement changes: There is a lot more erraticness to the movement in case of smaller batch sizes, while the movement smoothens out to a much

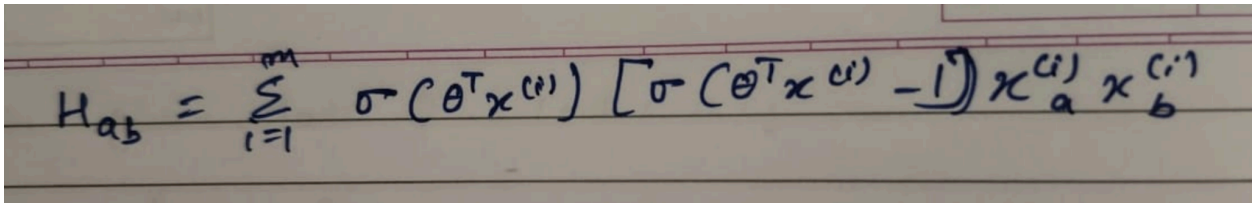
greater extent in the case of larger batch sizes, as can be seen in the figure. Moreover this diagram is showing us the speed of movement itself as well, we can clearly observe that the relative speed of convergence is greater for the smaller batch sizes.

Both of these make intuitive sense, as the relative speed of convergence depends on the time it takes to compute one change in the parameters, which is higher for higher batch sizes as computing gradient, loss for bigger batches takes more time.

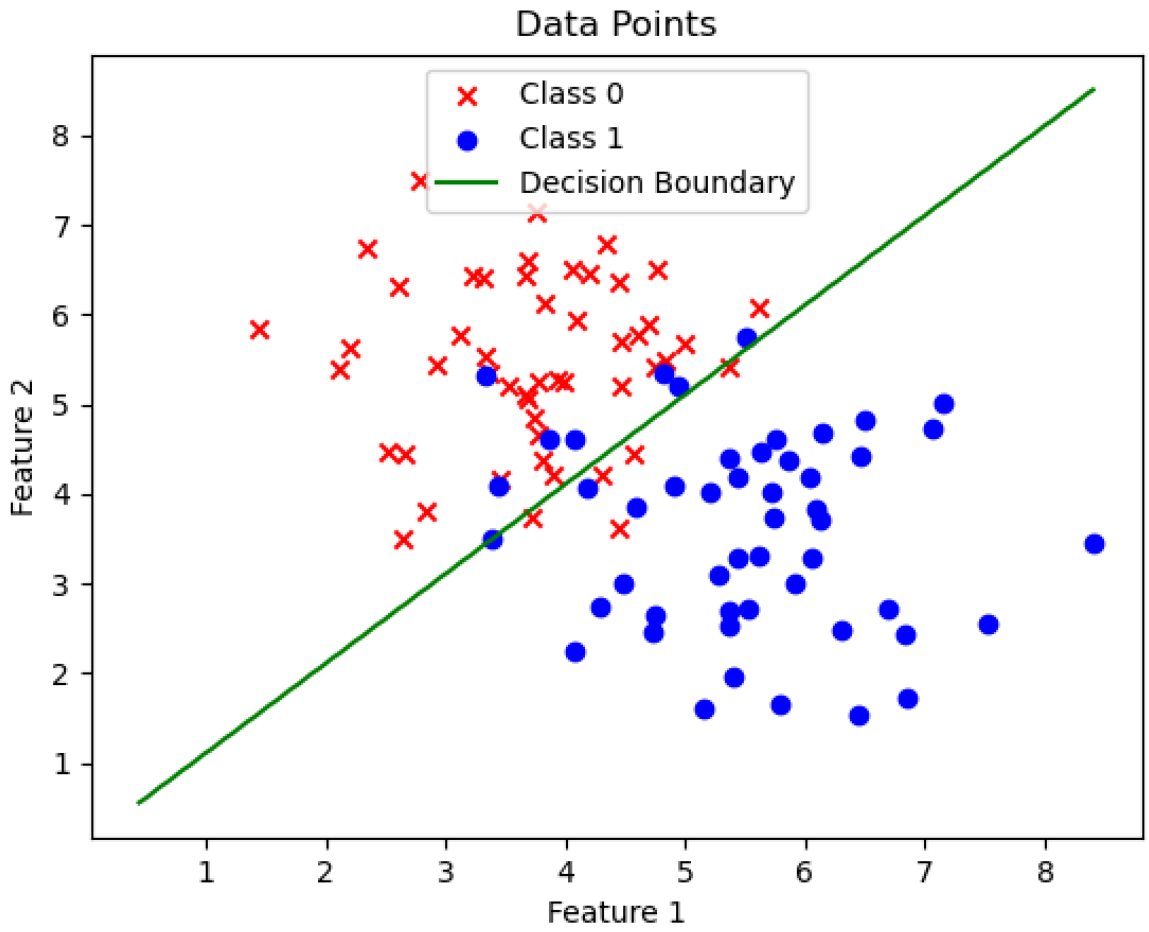
Erraticness for smaller batch sizes, especially batch size of 1 showcases that the movement in that case is not well guided as it is taking too small a portion of the whole, thus resulting in misguided moves from time to time momentarily, but in the end taking them to the correct answer.

3. Logistic Regression:

The hessian matrix calculated by me was:



1. Parameters:
   - Intercept = 0.40125316
   - Coefficient of X1 = 2.5885477
   - Coefficient of X2 = -2.72558849
2.



4. Gaussian Discriminant Analysis

For this part I have calculated the GDA parameters for both cases using the formulae. Then I made a function to see whether a given point satisfies the boundary function defined by these GDA parameters. Using this function I plotted out all the values, using meshes in numpy.

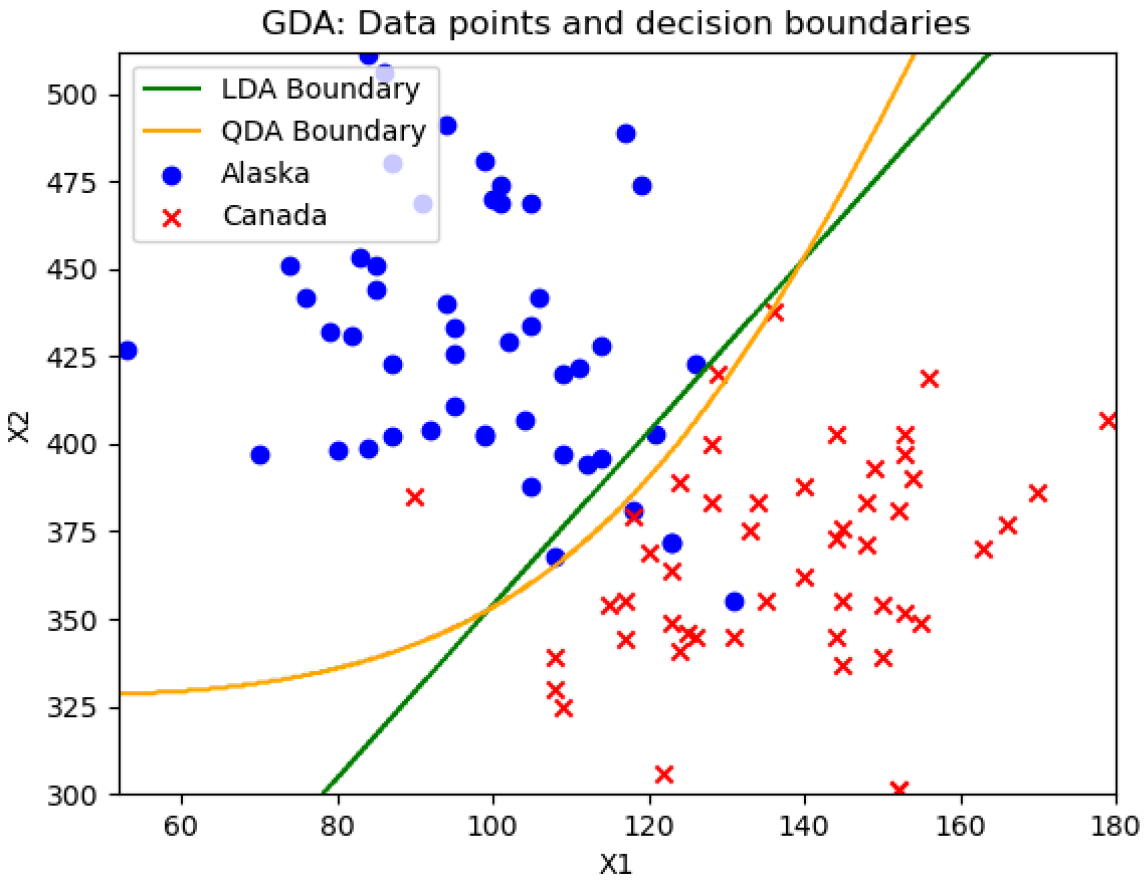
1.

```
Mean Alaska: [-0.75529433  0.68509431]
Mean Canada: [ 0.75529433 -0.68509431]
Covariance Matrix:
 [[ 0.42953048 -0.02247228]
 [-0.02247228  0.53064579]]
```

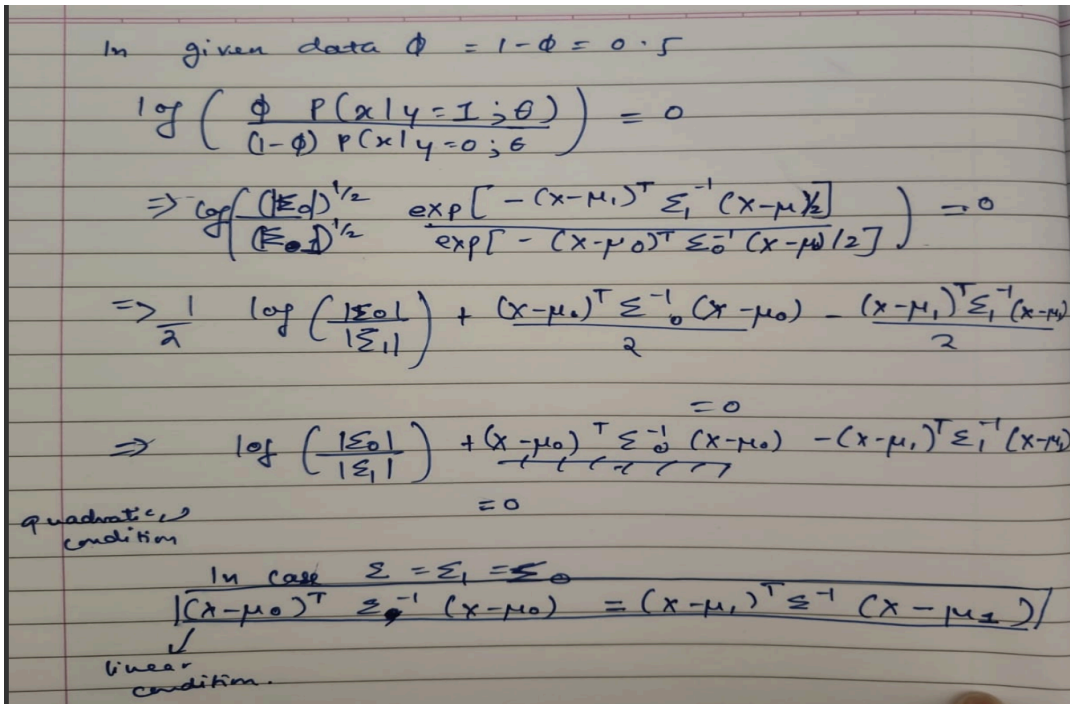
4.

```
QDA parameters:
Mean Alaska: [-0.75529433  0.68509431]
Mean Canada: [ 0.75529433 -0.68509431]
Covariance Matrix Alaska:
 [[ 0.38158978 -0.15486516]
 [-0.15486516  0.64773717]]
Covariance Matrix Canada:
 [[0.47747117 0.1099206 ]
 [0.1099206  0.41355441]]
```

2,3,5. Plots



3,5. Equations



6. Observations:
Both the boundaries misclassify 3 Alaskan points, and 1 Canadian point, all of which are deep inside the other territory, with points of the other kind surrounding them. These cannot be dealt with using such simple models, hence we do not consider them important.

We come to points on the boundary. We see that the linear line misclassifies 2 Alaskan points, while the quadratic boundary misclassifies 1 Canadian point, but this 1 Canadian point is very close to an Alaskan point. Thus we can see that the quadratic divide is doing much better as the data is mostly fitting to assumptions taken, and the data set is small.

The linear boundary works better for Canadian points, i.e if the Linear Boundary declares a point is Canadian it is more probable that the point is Canadian than if the Quadratic boundary declares a point as Canadian. Similarly the quadratic boundary works better for Alaskan points.