COL774 Assignment 1

Dhairya Gupta, 2019CS50428

Q1) (a)

In this part, implemented batch linear regression with least squared error. We tried various different learning rates. Plots of which are shown below. We run gradient descent for a maximum of 100000 iterations.

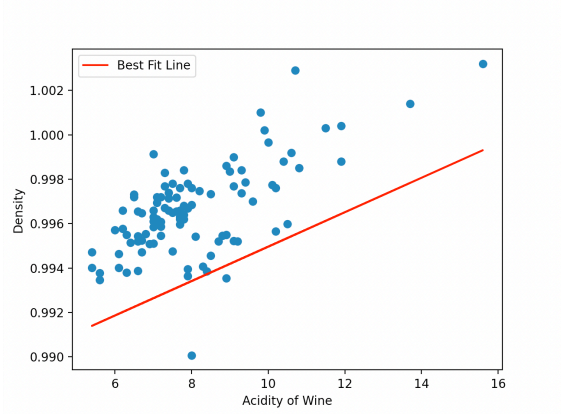
Convergence: We reach convergence if $|J(\theta^{(t)}) - J(\theta^{(t-1)})| < 10^{-8}$ where $\theta^{(t)}$ is the value of the parameter after iteration 't'. Value was chosen as line of best fit. From below observations and graphs, we see that for lower learning_rates, error term should be higher to account for the effect of η in convergence. (Also mentioned in part(e))

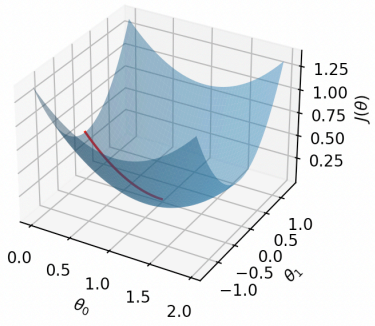
Learning Rate: We choose various different learning rates

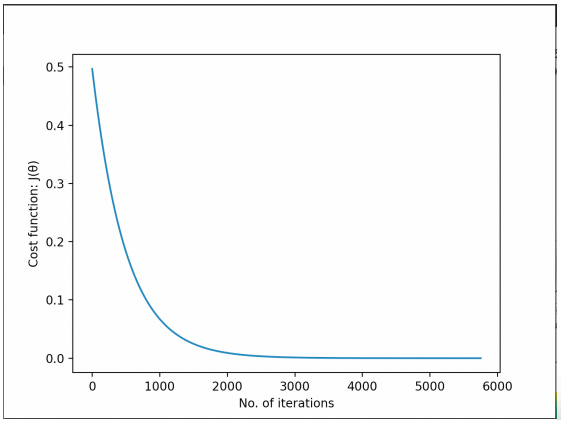
$\eta = 0.001$: converges in 5752 iterations. $(\theta_0, \theta_1) = (0.99346035, 0.00133595)$
$\eta = 0.025$: converges in 293 iterations. $(\theta_0, \theta_1) = (0.99600651, 0.00133937)$
$\eta = 0.1$: converges in 79 iterations. $(\theta_0, \theta_1) = (0.99635129, 0.00133983)$
$\eta = 0.5$: converges in 15 iterations. $(\theta_0, \theta_1) = (0.99655927, 0.00134011)$
$\eta = 1.7$: converges in 26 iterations. $(\theta_0, \theta_1) = (0.99675375, 0.00134038)$

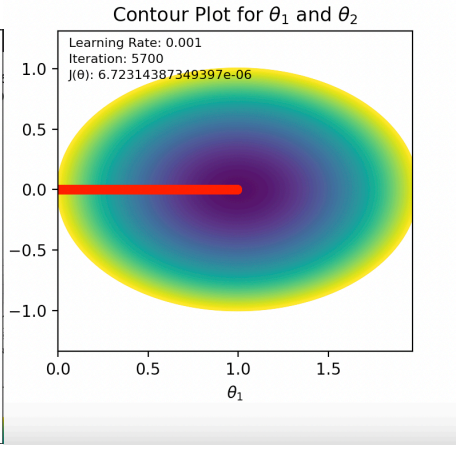
(b, c, d)

$$\eta = 0.001$$

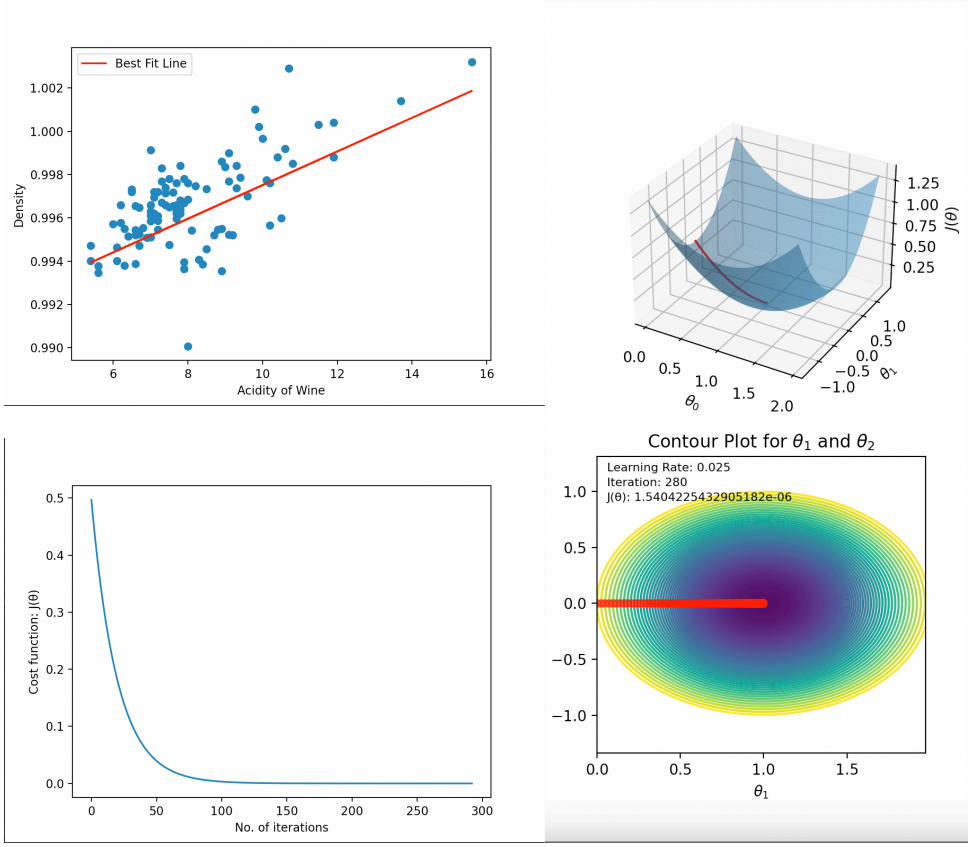




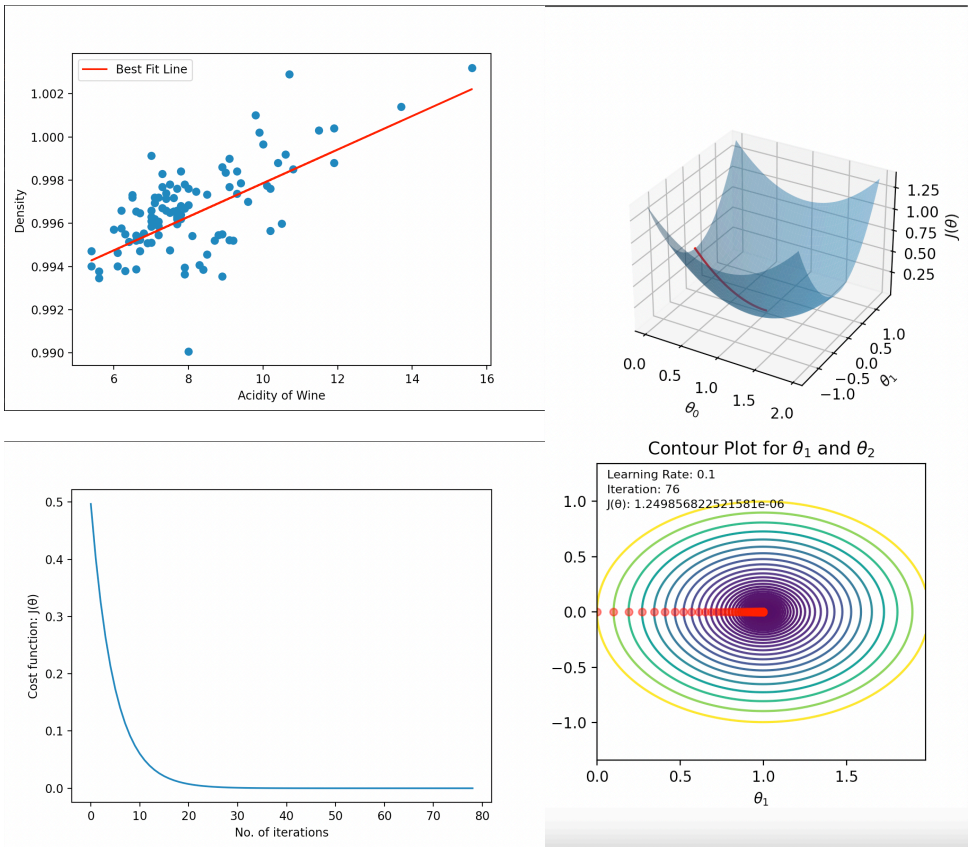




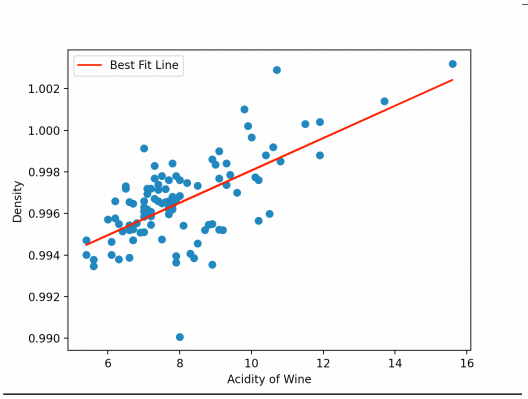
$$\eta = 0.025$$

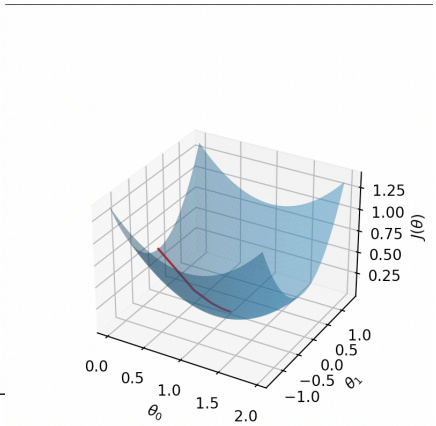


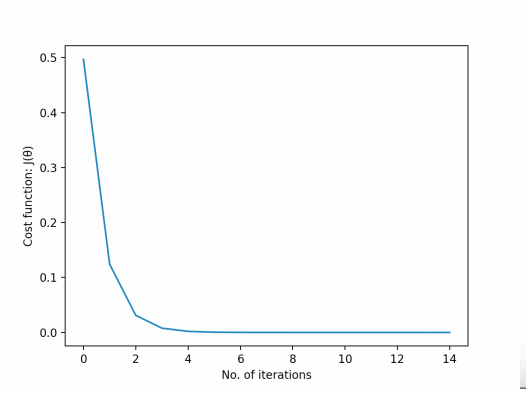
$$\eta = 0.1$$

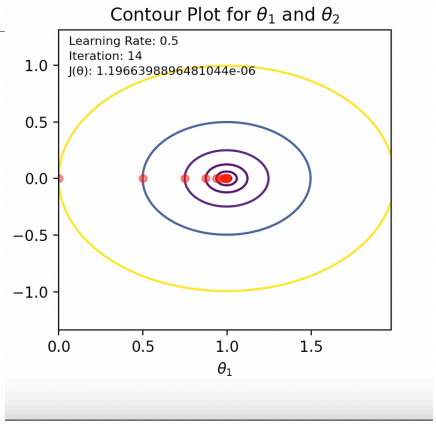


$\eta = 0.5$

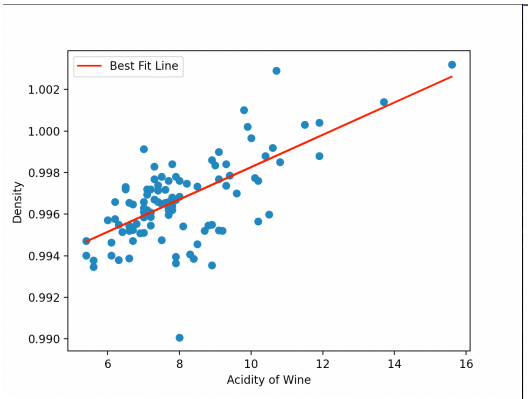

Best Fit Line
Density
Acidity of Wine

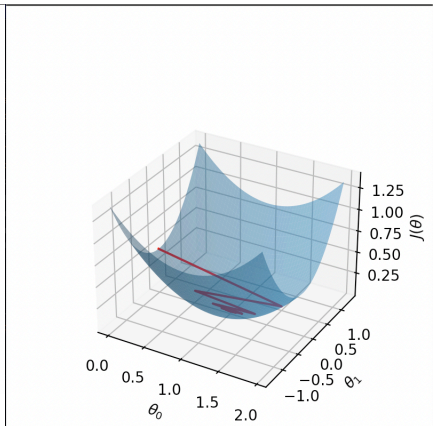

J(θ)
θ0
θ1

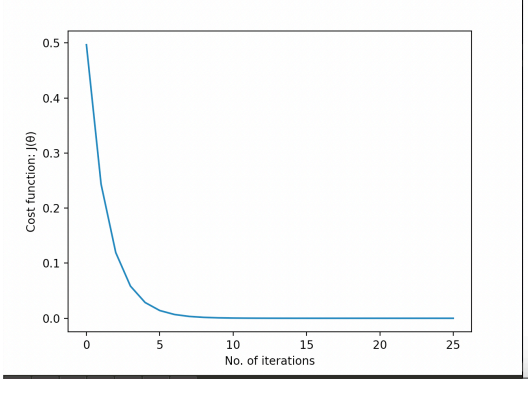

Cost function: J(θ)
No. of iterations

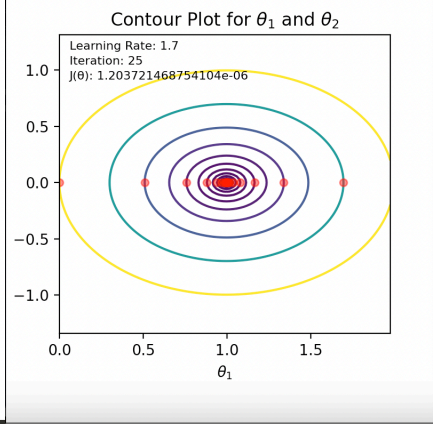

Contour Plot for θ1 and θ2
Learning Rate: 0.5
Iteration: 14
J(θ): 1.1966398896481044e-06
θ1

$\eta = 1.7$


Best Fit Line
Density
Acidity of Wine


J(θ)
θ0
θ1


Cost function: J(θ)
No. of iterations


Contour Plot for θ1 and θ2
Learning Rate: 1.7
Iteration: 25
J(θ): 1.203721468754104e-06
θ1

(e) A few observations about the above parts are the following:

- For a lower learning rate, the number of iterations taken to converge is more. This happens because the magnitude of change in θ at each step is smaller for small learning_rate. Hence, convergence is slower.
- However, the above point is not strictly correct. Consider $\eta = 0.5$ and $\eta = 1.7$. In this case, each iteration overshoots the minima, but does not overshoot enough for $J(\theta)$ to diverge. In this case, even with higher η, the overall reduction in $J(\theta)$ is less because overshooting the minima dampens the rate of decrease in cost.
- The best fit line is more accurate for higher value of learning rate. This is because for the given convergence criterion $|J(\theta^{(t)}) - J(\theta^{(t-1)})| < 10^{-8}$ is equivalent to $\frac{|J(\theta^{(t)}) - J(\theta^{(t-1)})|}{|\theta^{(t)} - \theta^{(t-1)}|} . |\eta * \nabla_\theta J(\theta)|_{\theta=\theta^{(t)}}| < 10^{-8}$ by the update rule for gradient descent. If η is less, then by this equation(assuming left fractional term to be bounded in a neighbourhood of the minima), convergence criterion would hold true for a larger neighbourhood of minima. Hence, the line obtained for a larger learning_rate for the given convergence criterion is a better fit.

For values higher than $\eta = 1.7$, the gradient descent algorithm starts diverging(as the overshoot increases).

2(a).

Resulting y from sampling had an approximate mean 4.00 and variance 22.00.

(b, c).

For SGD, we have chosen the following criteria. $\eta = 0.001$ for all values of batch_size = {1, 100, 10000, 1000000}. Convergence criterion is the following: after every *n_check_conv* batch iterations, we compute the average cost over these batches and compare this cost to the average of previous *n_check_conv* batches. If difference in average values is less than a *min_error,* we have converged.

Other hyper-parameters to tune are

- *n_check_conv:* Number of consecutive batches for which average cost is calculated
- *min_error:* Minimum absolute difference between average costs, below which we converge.
- *batch_size*: size of a single batch
- *max_iter: maximum iterations for which loop runs*
- *plot2D_iter_skip:* plot only iterations which are a multiple of plot2D_iter_skip

Consider the cases given below:

1. ***Batch_size = 1000000***

This is the simple case of batch linear regression. In this case, we set n_check_conv to be 1(i.e. check convergence after every batch iteration). Threshold is chosen as 0.0005. Learning rate = 0.001

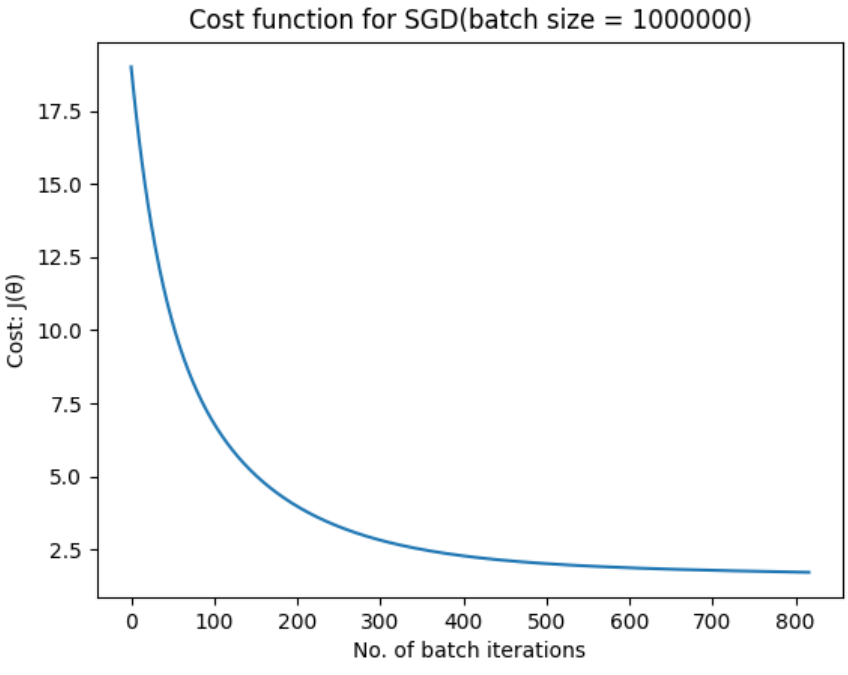
Values obtained are:

$$\theta_0 = 0.77309957,\ \theta_1 = 1.46012212,\ \theta_2 = 1.75684784$$

Error obtained with test case is 15.633.

Time to attain convergence is 29.044s

No. of iterations = 817



2. Batch_size = 10000

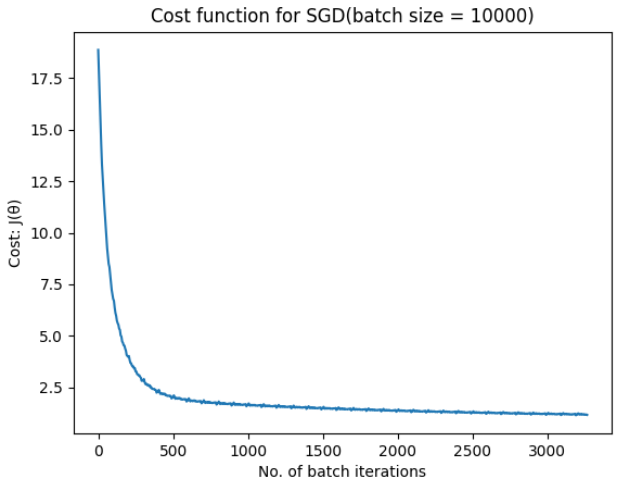
Here, we set n_check_conv to some prime number(e.g., n_check_conv = 43). Min_error remains 0.0005. Plot and animation is shown for every 5th iteration. Values obtained are:

$$\theta_0 = 1.85602672,\ \theta_2 = 1.25001187,\ \theta_3 = 1.91690347$$

Error obtained with test case is 4.75879216

Time to attain convergence: 1.35 s

No. of iterations = 3268



3. Batch_size =100

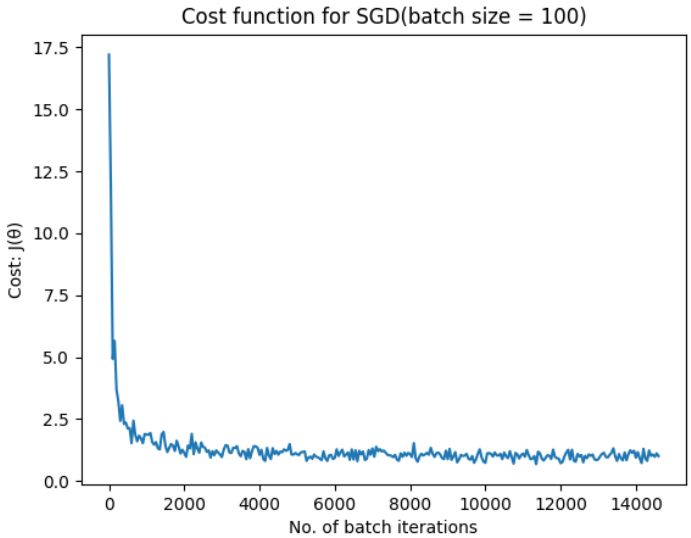
Here, we set n_check_conv to some prime number(e.g. n_check_conv = 1229) so that no 2 batches are completely same. In the plot given below, plot2D_iter_skip = 50. (Cost after every 50^{th} iteration is plotted). For previous 2 cases, every iteration was shown. Values obtained are:

$$\theta_0 = 2.94102564,\ \theta_1 = 1.00984332,\ \theta_2 = 1.99680484$$

Error obtained with test_case: 0.98947843

Time to attain convergence: 0.77s

No. Of iterations:14637



4. Batch_size = 1

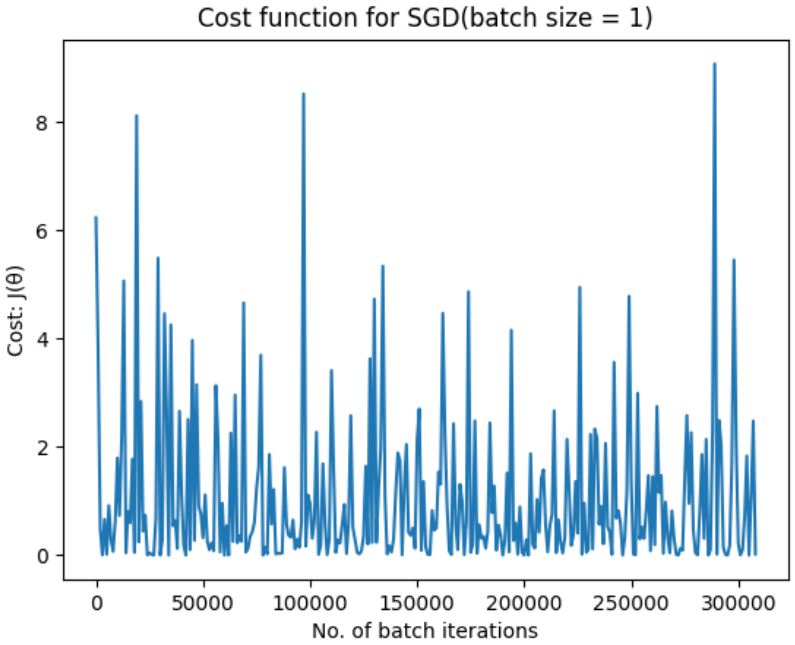
Here, we set n_check_conv to some prime number(e.g. n_check_conv =17157). Error threshold is set to 0.001. Learning rate is 0.001. In the plot and animation, every 1000th iteration is shown
Values obtained are:

$$\theta_0 = 3.02750947,\ \theta_1 = 0.97502927,\ \theta_2 = 1.97644705$$

Error obtained with test case: 1.04671625

Time to attain convergence: 15.65s

No. Of iterations: 308826



5. Error of sample theta $\theta^T = [3\ 1\ 2]$ with test data = 0.98294692

We see that the the error on the test set of sample theta is (unsurprisingly) the lowest. Here, we can observe that the cost function curves become less smooth as batch size decreases. We can also see that for the same error threshold and learning rate, the convergence is quickest when batch size is between 1 and 1000000. This is true because for batch size 1000000, every iteration takes a long amount of time, and for random data, multiple gradient steps for a smaller batch size more than compensates for the single gradient step of larger batch size in the same time. Hence, convergence is quicker. However, as batch size approaches 1, the oscillations in cost become larger and so, it becomes difficult to converge. Therefore, there is an increase in convergence time. The error also decreases as we move towards lower batch size as number of iterations(i.e. gradient steps are more). Another interesting observation is that the error for the test set on the sample_theta is very close to the test_error obtained with size 100, again showing that lower batch sizes are more optimal and converge quicker.

(d). From the animation, it is evident that for batch size = 1000000, the gradient steps are towards the minima. As we move towards lower batch sizes, the oscillations around the gradient path increase in amplitude. For larger batches, the point does not get as close to the minima as desired given our convergence criteria (to get better results, we should reduce error threshold for large batches). Also, for smaller batch sizes, θ does not directly converge to the minima but rather oscillates around it. This explains the observation that error of batch size 100 is less than 1 as both of reached the point of expected minima but the oscillations in case of batch size 1 are larger, leading to more error on the test set.

3.

For Logistic Regression using Newton's Method, update rule is

$\theta^{(t+1)} \leftarrow \theta^{(t)} - H^{-1} \nabla_\theta J(\theta)|_{\theta=\theta^{(t)}}$, where H is the Hessian matrix. For logistic regression,

$H = \sum_{i=1}^{m} x^{(i)} x^{(i)^T} \sigma(z_i)\sigma(1 - z_i)$ where σ is the sigmoid function and $z_i = \theta^T x^{(i)}$. Also,

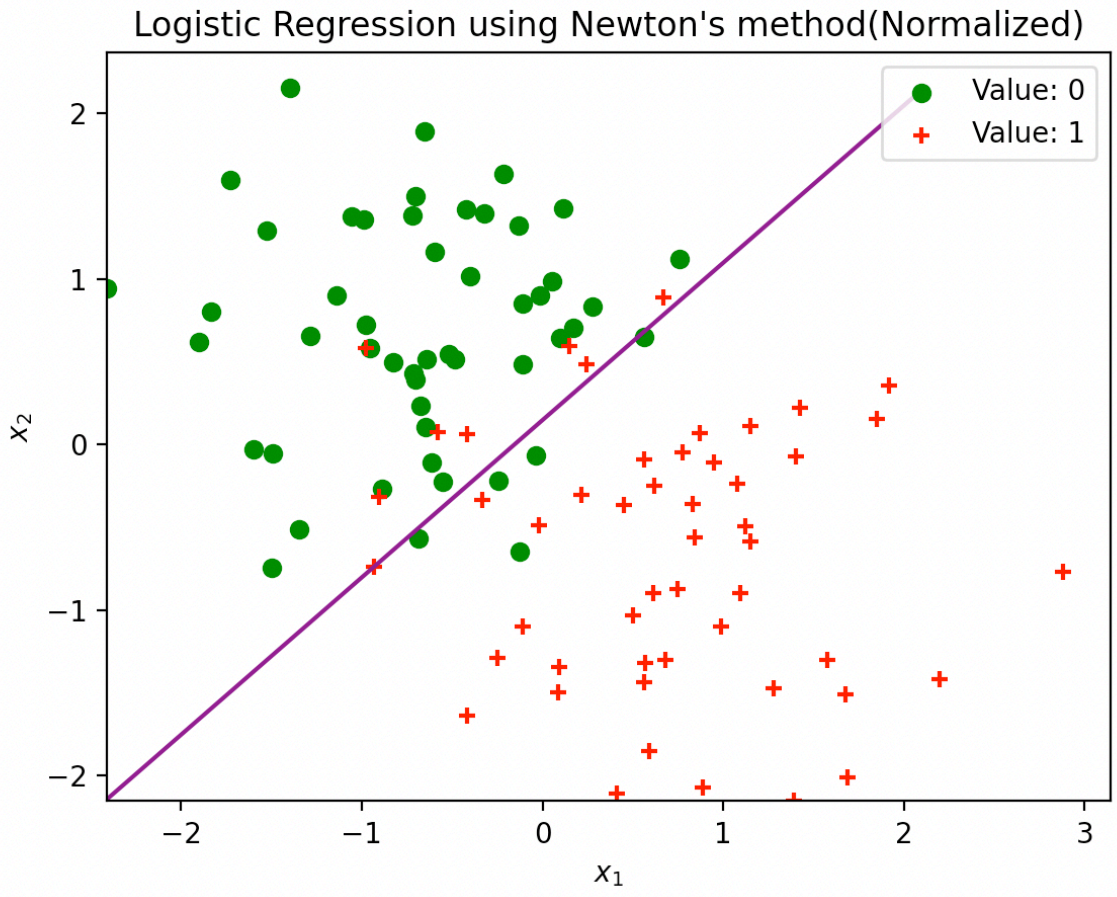
$\nabla_\theta J(\theta) = -\sum_{i=1}^{m} (y^{(i)} - \sigma(\theta^T x^{(i)}))x^{(i)}$. Using these 2 equations, we get values:

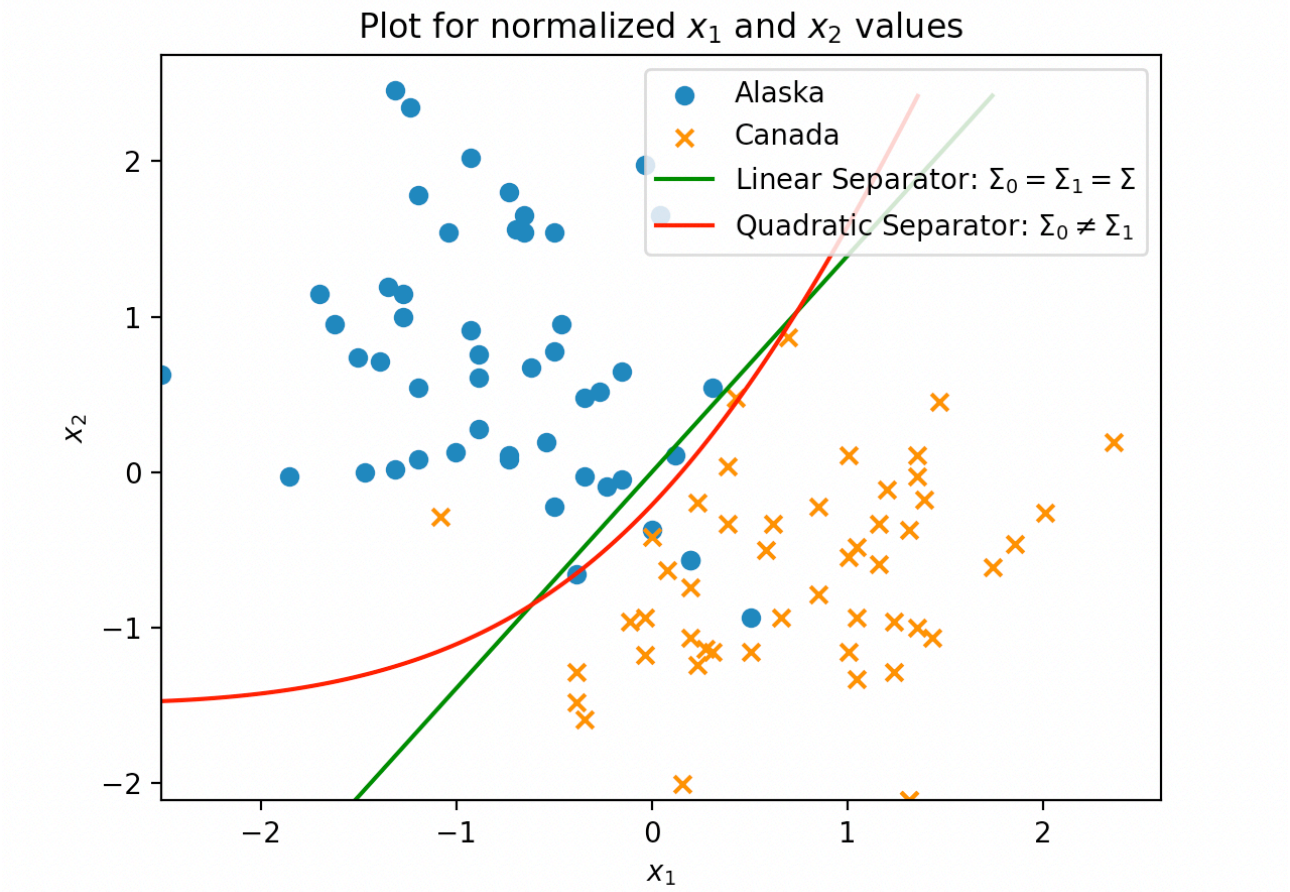
No. of iterations: 7

Min_Error:1e-5

$\theta = [\theta_0, \theta_1, \theta_2] = [0.40125316, 2.5885477, -2.72558849]$

Plot for data and decision boundary is given below:





4.

$$\phi = 0.5$$

$$\mu_0 = [[-0.75529433][0.68509431]]$$

$$\mu_1 = [[0.75529433][-0.68509431]]$$

If $\Sigma_0 = \Sigma_1 = \Sigma$ then:

$$\Sigma = [[0.42953048 \ -0.02247228][-0.02247228 \ 0.53064579]]$$

else:

$$\Sigma_0 = [[0.38158978 \ -0.15486516][-0.15486516 \ 0.64773717]]$$

$$\Sigma_1 = [[0.47747117 \ 0.1099206][0.1099206 \ 0.41355441]]$$

The values above are derived from the equations:

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

$$\Sigma_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}(x^{(i)} - \mu_0)(x^{(i)} - \mu_0)^T}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\Sigma_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}(x^{(i)} - \mu_1)(x^{(i)} - \mu_1)^T}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

The equation for quadratic boundary is given by:

$$\frac{1}{2}[(x_1 - \mu_1)^T\Sigma_1^{-1}(x_1 - \mu_1) - (x_0 - \mu_0)^T\Sigma_0^{-1}(x_0 - \mu_0)] + \log(\frac{1-\phi}{\phi}) + \frac{1}{2}\log(\frac{|\Sigma_1|}{|\Sigma_0|}) = 0$$

For the case when $\Sigma_1 = \Sigma_0 = \Sigma$, this boundary reduces to a linear expression:

$$(\mu_0^T\Sigma^{-1}x_0 - \mu_1^T\Sigma^{-1}x_1) + \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0) + \log(\frac{1-\phi}{\phi}) = 0$$

Below is shown the plot for quadratic and linear decision boundary based on given values

As we can see from the plot above, the quadratic decision boundary appears to be a better separator. It correctly classifies some points of Alaska which were giving output Canada according to the linear classifier. If we see the spread of Alaska points, they are spread more along the $x_1 = -x_2$ axis than the $x_1 = x_2$ axis. The opposite is true for points corresponding to Canada. Thus, we can see pictorially that the 2 distributions have different spreads(which lead to different covariance matrices). A linear boundary cannot account for this difference in spread, therefore it is less accurate. From our calculations of Σ_0, Σ_1 too we are able to tell the difference in covariance of both distributions. In particular, the more is the curvature of the quadratic boundary, the greater is the difference between Σ_0, Σ_1. In situations where data can be assumed to be from i.i.d normal distribution, such a quadratic classification model should be a good fit for the data.