Problem 1. Linear Regression

Answer to the problem goes here.

1. J(θ) and ∇_θJ(θ) in the matrix form can be written as follows:
   $J(\theta) = \frac{1}{2m}$ (Xθ - Y)^T (Xθ - Y)

   $\nabla_\theta J(\theta) = \frac{1}{m}$(X$\theta$ - Y)X

   (i) learning rate $\eta = 0.40$
   (ii) stopping criteria : $\forall$j $\nabla_{\theta j}$J(θ) <= 10^{-11}
   (iii) Final set of parameters (θ) = $\begin{bmatrix} 0.9966163 \\ 0.00134019 \end{bmatrix}$

2. The hypothesis function is a straight line and equation of the line can be written in the form of :
   y = mx + c
   here m = 0.0013402, c = 0.9966201

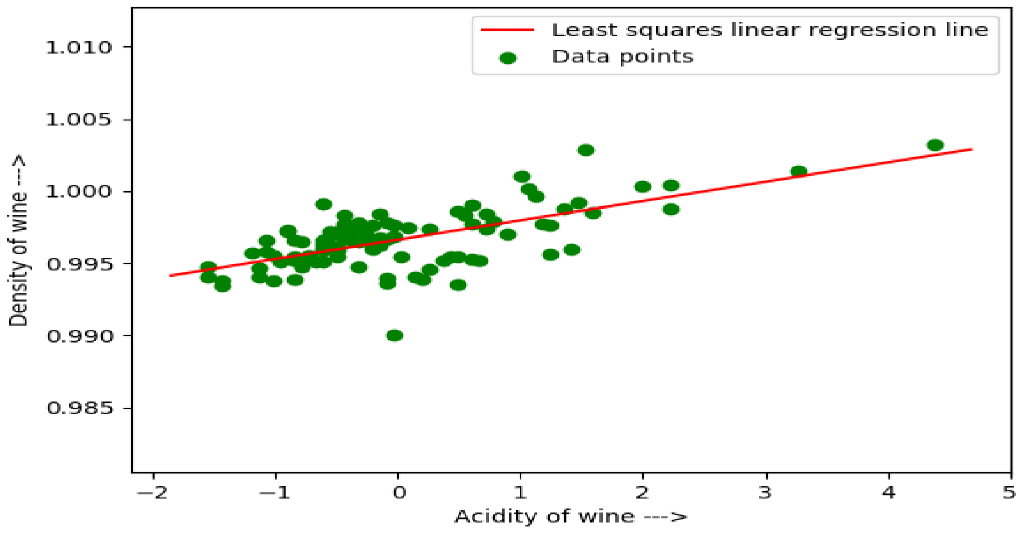

Figure 1: Linear Regression using gradient descent with $\eta = 0.40$

3. 3 dimensional mesh showing the error function (J()) on z-axis and the parameters in the xy plane. The red line shows how gradient descent converges.

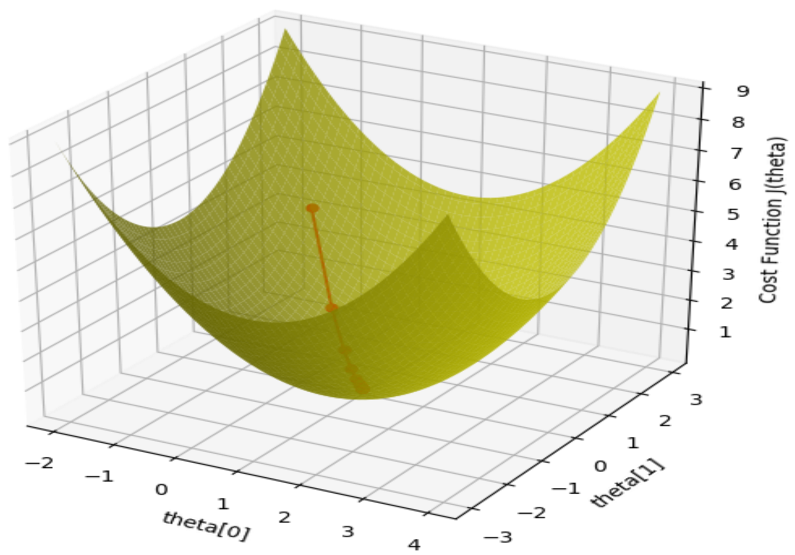

Figure 2: 3D Mesh grid and convergence of batch gradient descent

4. Drawing the contours of the error function J(θ) and the convergence of gradient descent at each iteration.

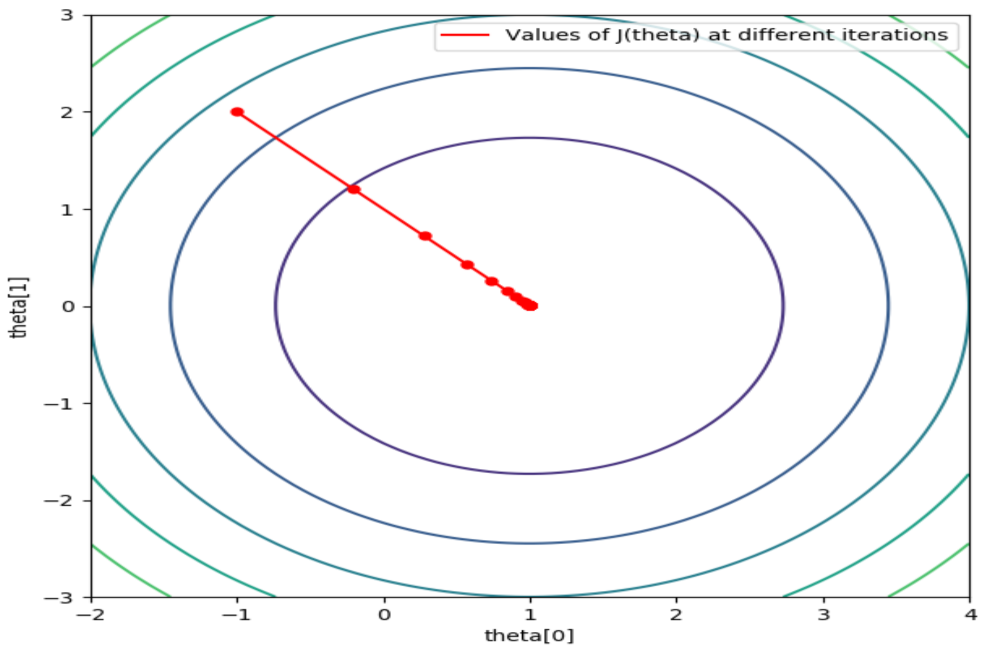

Figure 3: **Contour representation of gradient descent at η = 0.40**

5. when we increase the value of learning rate η, the oscillations increases i.e. it overshoots. if we increase the value of η to 2 or beyond 2, then it never converges (programatically, after some iterations, python data type is unable to store such a large value, resulting in "nan".)
   when η is 1.99, the oscillations are too high and it takes a lot of iterations to converge. The contours for η at 0.1,0.5,0.9,1.3,1.7 are shown below.

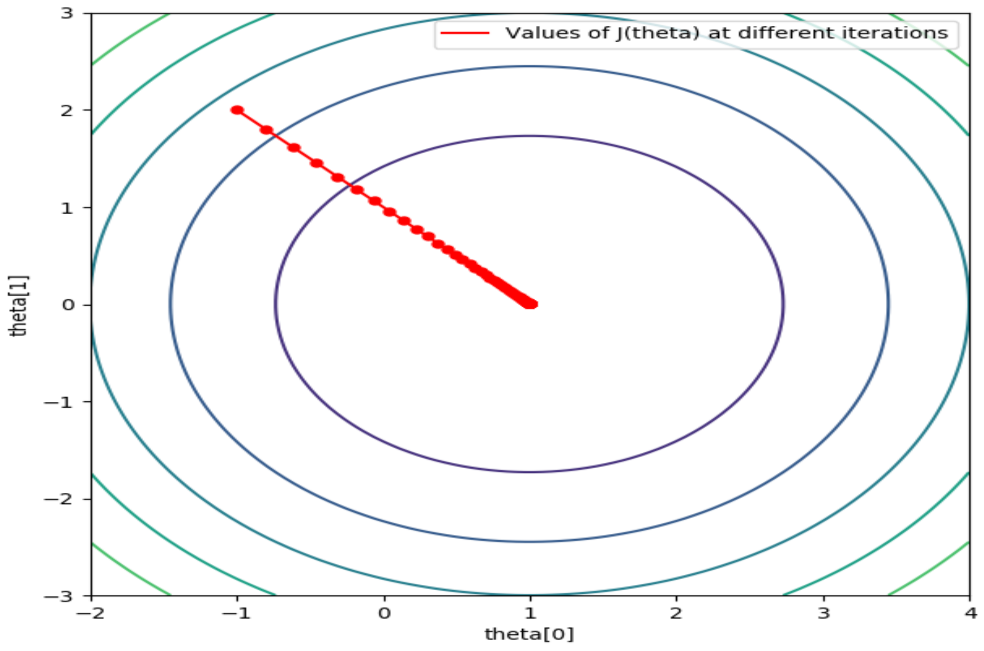

Figure 4: Contour Plotting at $\eta = 0.1$

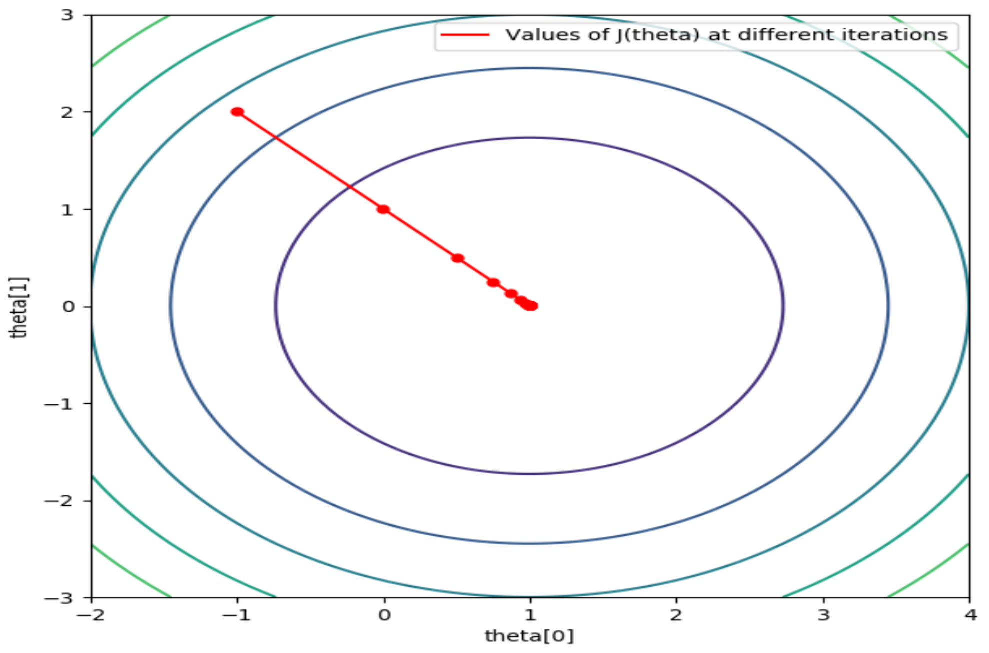

Figure 5: Contour Plotting at $\eta = 0.5$

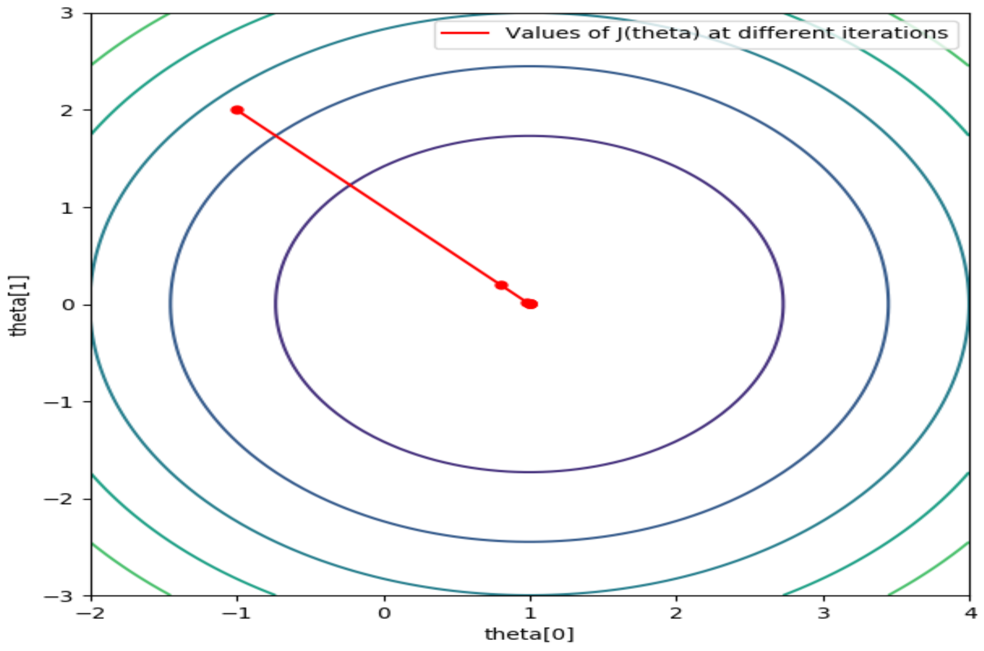

Figure 6: Contour Plotting at $\eta = 0.9$

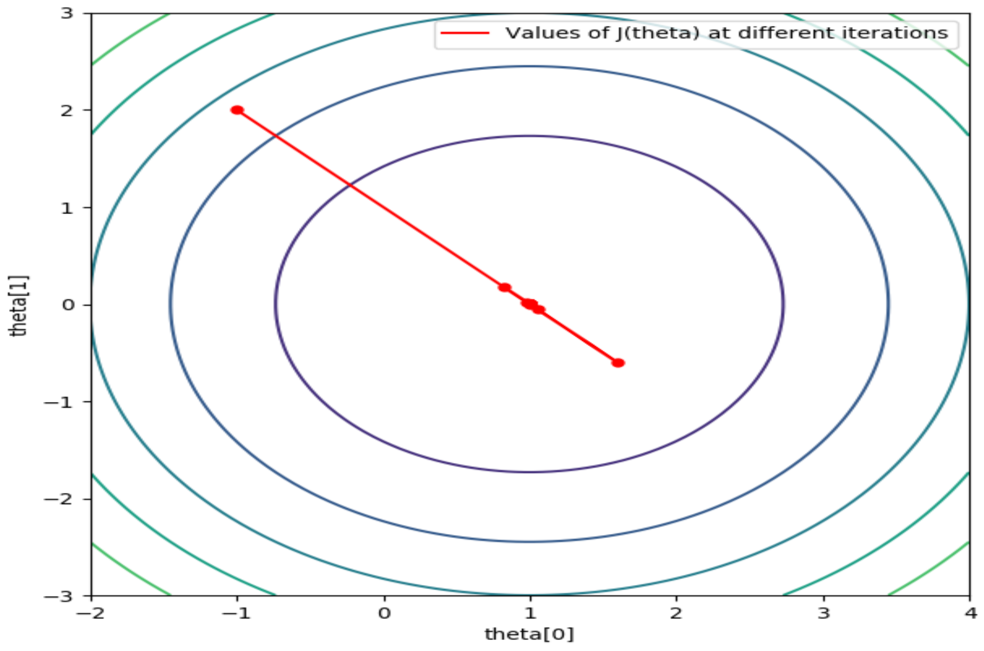

Figure 7: Contour Plotting at $\eta = 1.3$

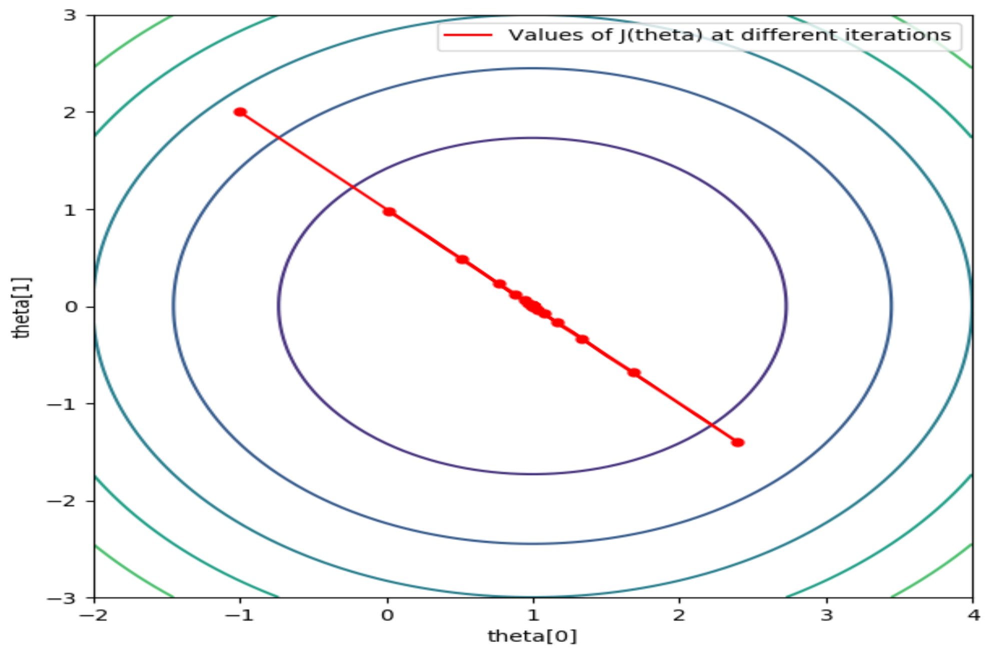

Figure 8: Contour Plotting at $\eta = 1.7$

From these observations, we can say that, when η is 0.1,0.3 and 0.5, Gradient descent converges a bit faster, while at 1.3 and 1.7 gradient descent converges a bit slower due to more oscillations around optimum value. when η is 2.1,2.5 Gradient descent never converges.

Problem 2. Locally Weighted Linear Regression

Answer to the problem goes here.

1. In an unweighted linear regression, the normal equation is
   $X^T X\theta = X^T Y$
   and theta can be written as
   $\theta = (X^T X)^{-1} X^T Y$

   After solving above equation , the value of (θ) is:
   Final set of parameters (θ) = $\begin{bmatrix} 0.32767322 \\ 0.17531247 \end{bmatrix}$
   The hypothesis function is a straight line and equation of the line can be written in the form of :
   y = mx + c
   here m = 0.17531247, c = 0.32767322

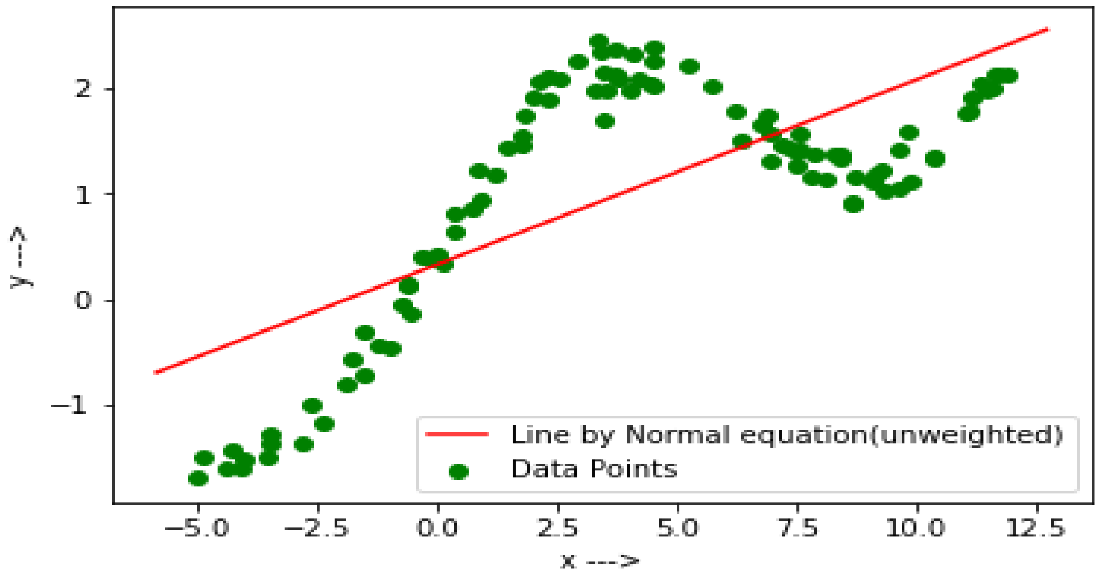

Figure 9: Linear Regression using normal equation

2. Locally weighted linear regression assigns a weight to each data point in training set, depending on the distance of the data point from the query point.
Weight is associated based on the formula given in the question, which depends on a bandwidth parameter τ.
Our goal is to minimizes the cost function defined by:
J(θ) = $\frac{1}{2m}(X\theta$ - Y$)^T$W(Xθ - Y)

In a weighted linear regression, the normal equation is:
X^{T}WXθ = X^{T}WY

therefore, theta can be written as:
θ = (X^{T}WX)$^{-1}$X^{T}WY

After solving above equation at certain number of query points with τ = 0.8, the curve can be plotted as:

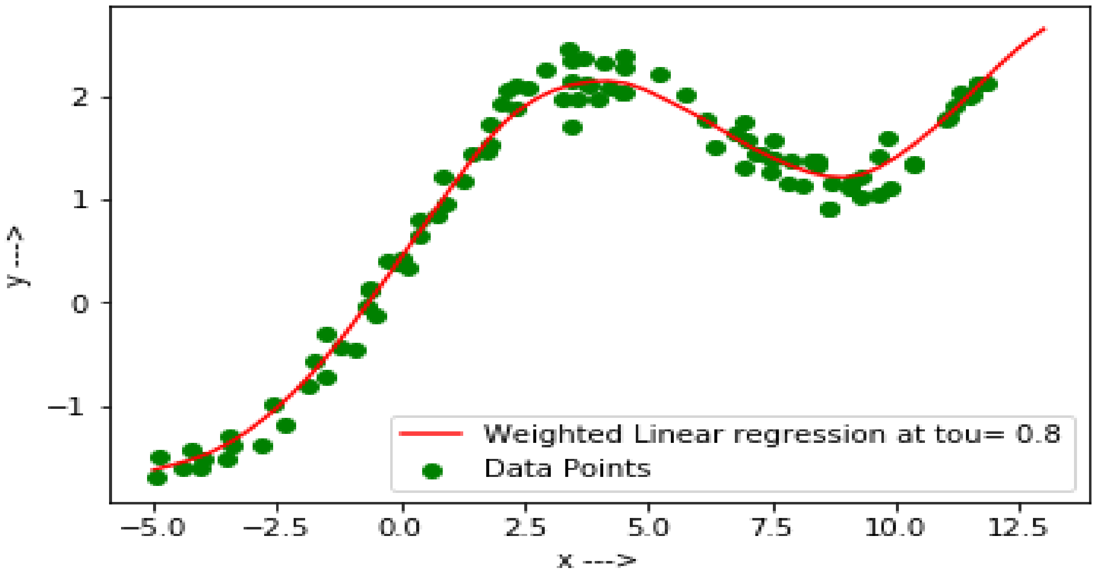

Figure 10: Locally weighted Linear Regression at τ = 0.8

3. **Plotting Locally weighted linear regression at different values of τ i.e. τ = 0.1,0.3,2 and 10**

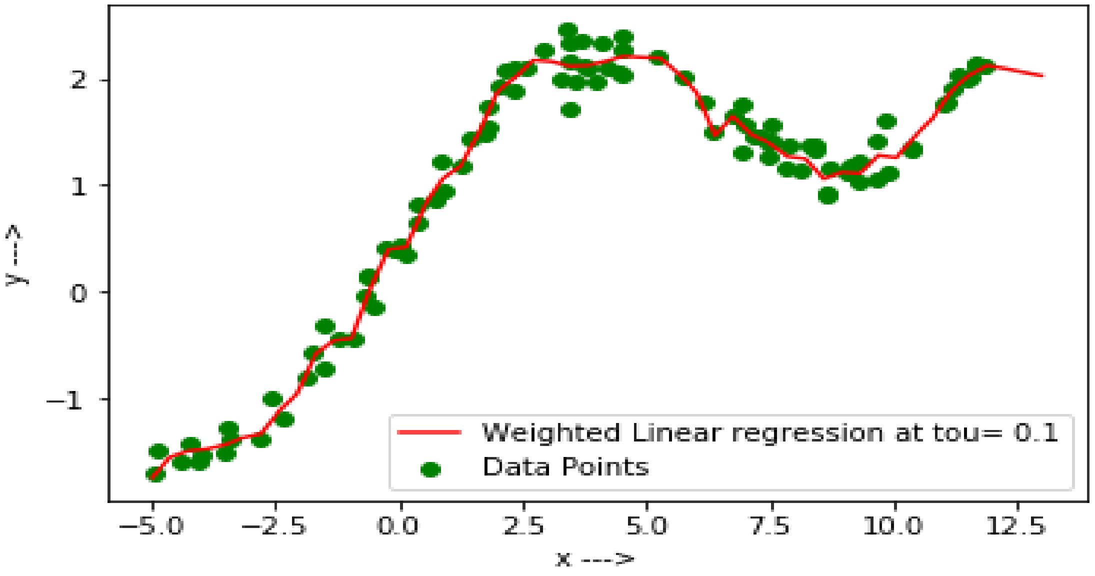

Figure 11: **Locally weighted Linear Regression at τ = 0.1**

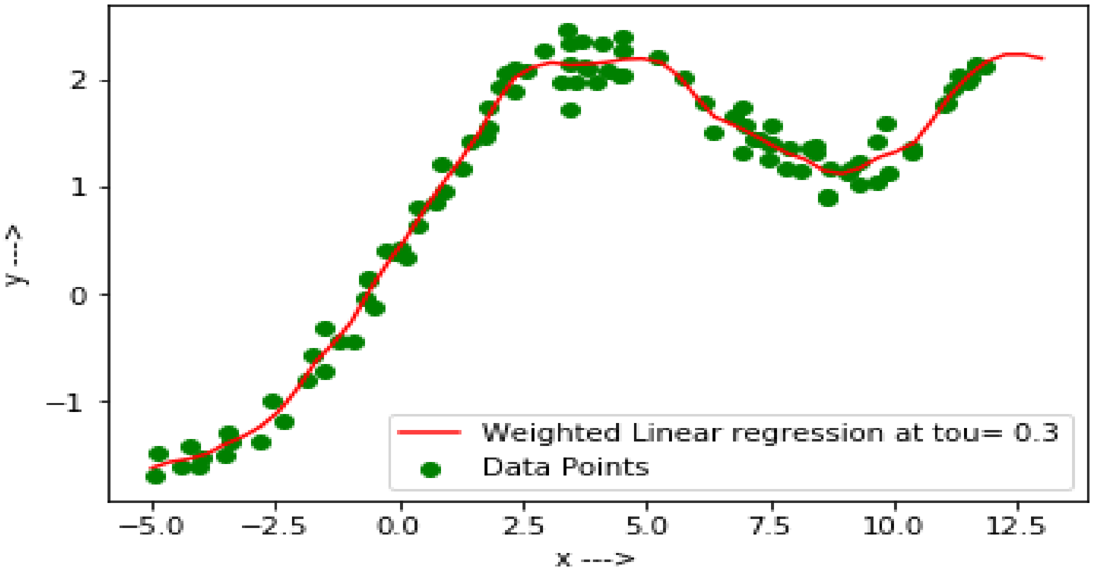

Figure 12: **Locally weighted Linear Regression at τ = 0.3**

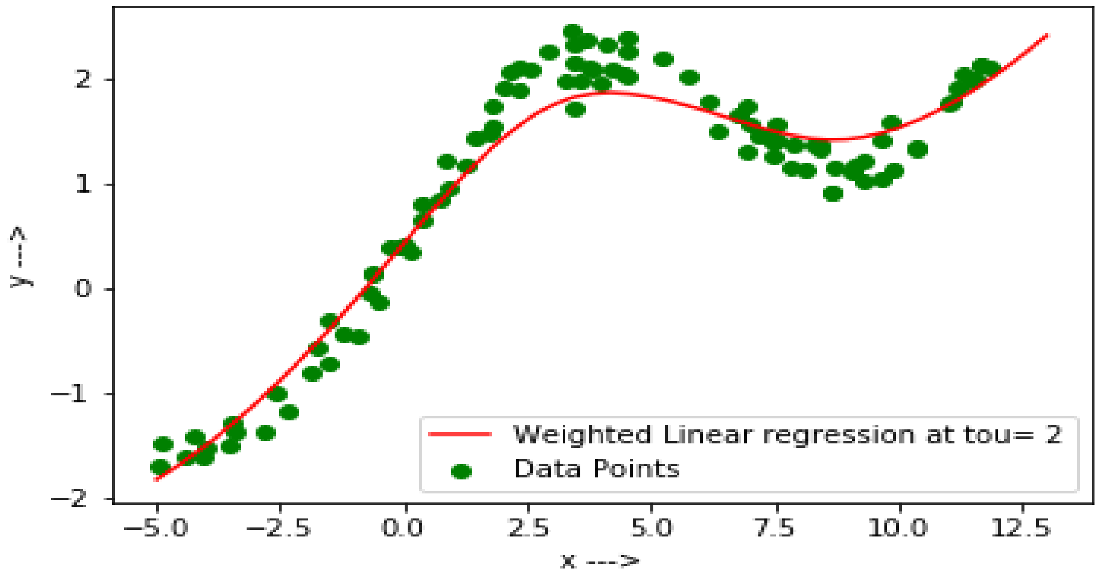

Figure 13: **Locally weighted Linear Regression at τ = 2**

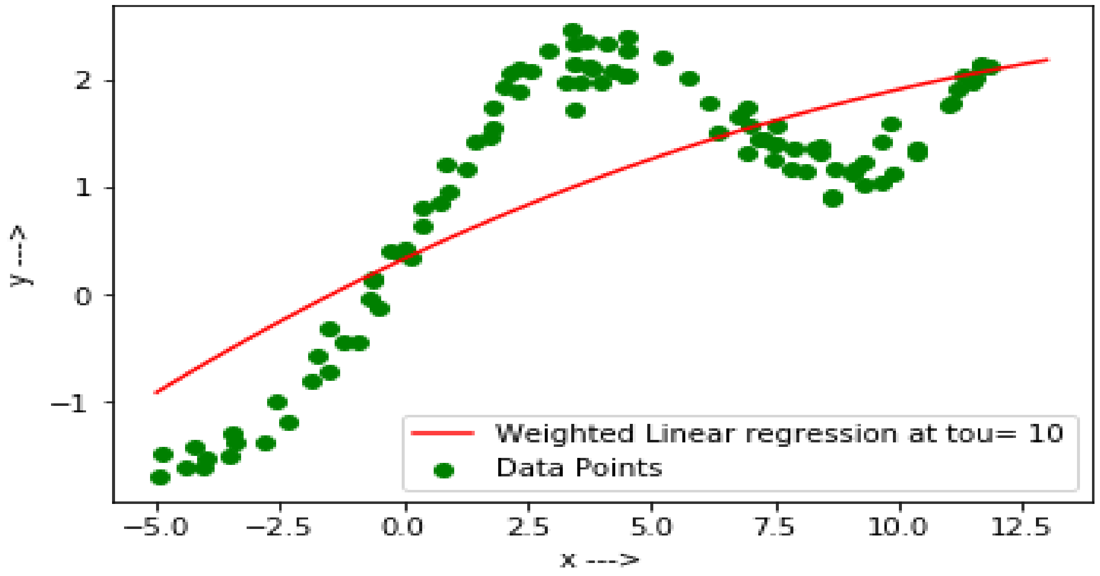

Figure 14: **Locally weighted Linear Regression at τ = 10**

From above observations, we can conclude that, $\tau = 0.3$ works best out of these four values.
As we increase the value of τ , the weight decreases slowly with distance(the range of points that plays significant role in deciding the optimum theta increases), consequently,

the prediction becomes less accurate.
If the prediction is more based on neighbourhood points, it gives better results as in case when $\tau = 0.3$
When τ becomes too large(in magnitude i.e either ∞ or -∞ , the weight for each training example will become 1, which is equivalent to linear regression.
Thus it is "underfitting" in this example for $\tau = \infty$ or -∞
when τ becomes too small (close to 0), then weight associated with every training example will become almost 0. This will be,as if we don't have any training data.
In this case, the predictions can be completely inaccurate.

Problem 3. Logistic Regression

Answer to the problem goes here.

1. Log-likelihood function for logistic regression is:

$$L(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

   The Newton's rule can be written as following:-

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1} \nabla_\theta L(\theta)$$

   The Newton's method converges faster than the batch gradient descent method. But computing hessian at each iteration takes a large amount of time.

   After applying Newton's method with initial $\theta = 0$, the θ converged at:

$$\theta = \begin{bmatrix} 0.22329537 \\ 1.96261552 \\ -1.9648612 \end{bmatrix}$$

2. The decision boundary separating the region where h(x)>0.5 ,from where h(x)<=0.5.) can be plotted as:

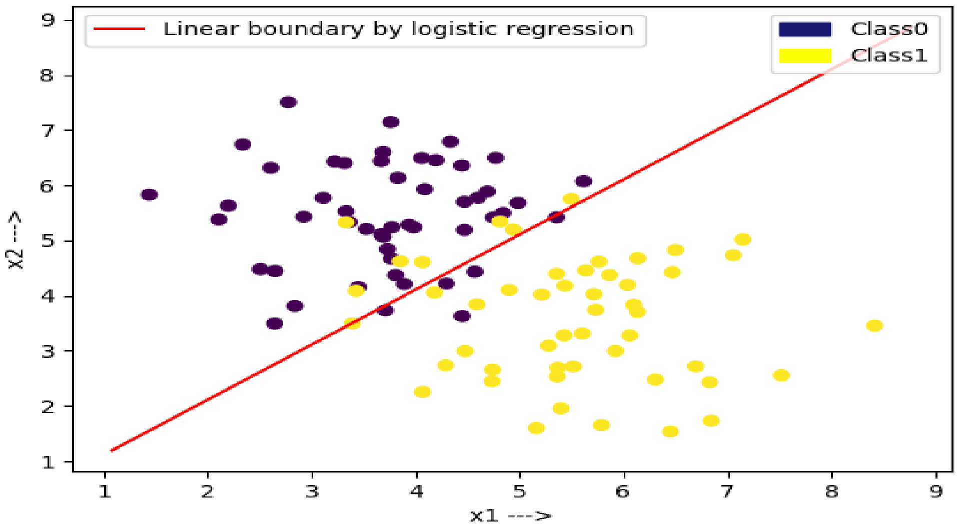

Figure 15: Logistic Regression using Newton's method

Problem 4. Guassian Discriminant Analysis

1. There is a strong assumption in GDA which is distribution of $x^{(i)}$ is Guassian distribution.
   $y^{(i)}$ belongs to bernoulli's distribution.

   Now if we assume that $\sum_0 = \sum_1 = \sum$, we will get following value of $\mu_0, \mu_1 and \sum$:

$$\mu_0 = \begin{bmatrix} 98.38 \\ 429.66 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 137.46 \\ 366.62 \end{bmatrix}$$

$$\sum_1 = \begin{bmatrix} 287.482 & -26.748 \\ -26.748 & 1123.25 \end{bmatrix}$$

2. Plotting the data given based on the class it belongs to:

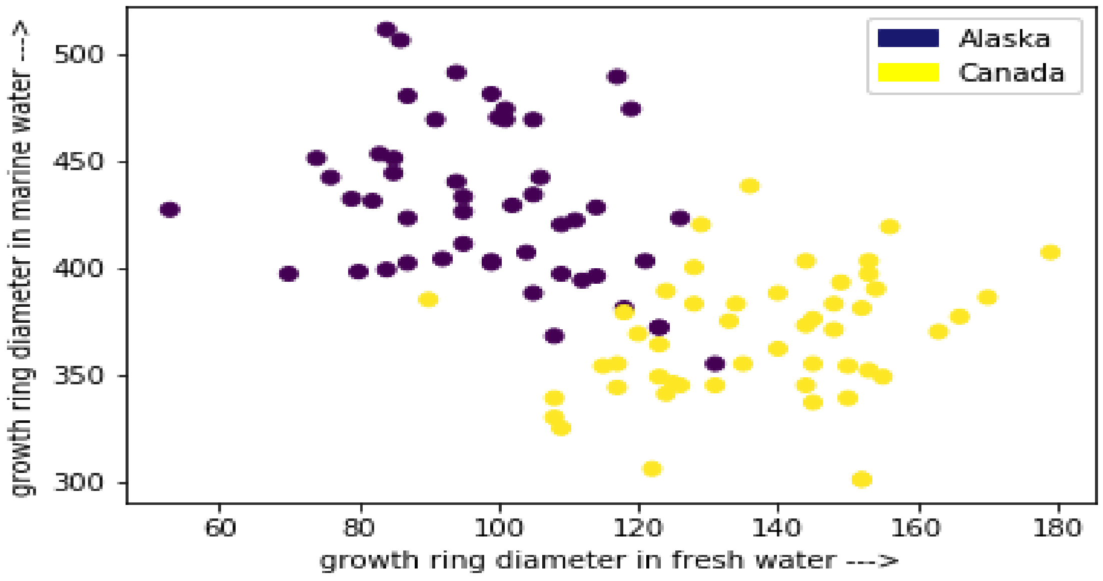

Figure 16: Training Data with label

3. Equation of the boundary when $\sum_0 = \sum_1 = \sum$ can be written as:-
   $(\mu_0^T - \mu_1^T)\sum^{-1} X - (\log(\frac{\phi}{1-\phi}) + \frac{1}{2}(\mu_1^T \sum^{-1} \mu_1 - \mu_0^T \sum^{-1})\mu_0) = 0$

   The above equation is linear in terms of X and the curve can be a straight line or plane or hyper plane depending on the dimensions of X.

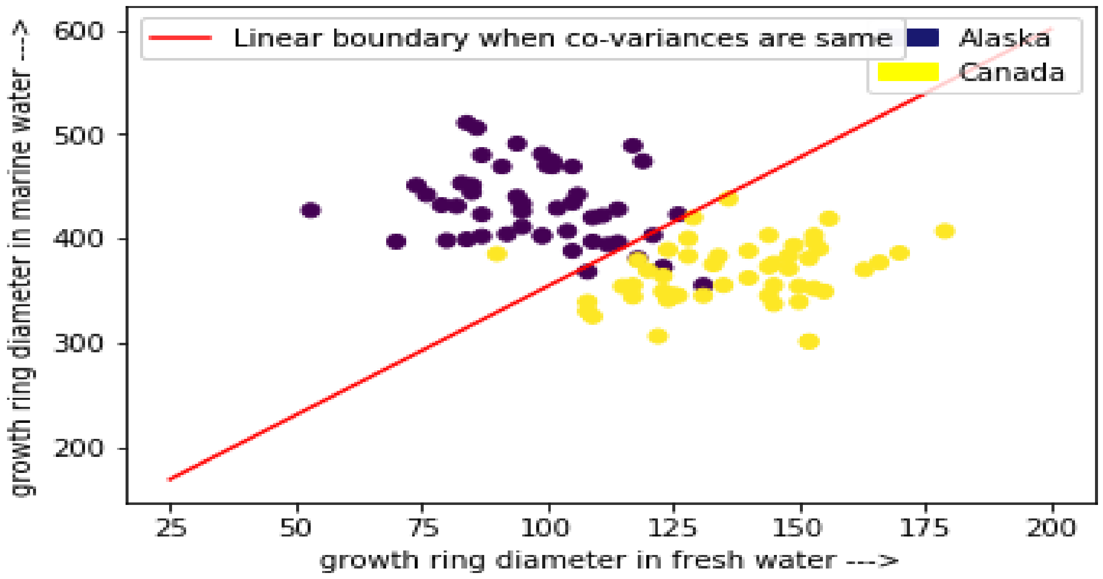

Figure 17: Decision boundary by GDA when covariance matrix is same

4. If the co-variance matrix are different , then we will get following parameters:

$$\mu_0 = \begin{bmatrix} 98.38 \\ 429.66 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 137.46 \\ 366.62 \end{bmatrix}$$

$$\sum_0 = \begin{bmatrix} 255.3956 & -184.3308 \\ -184.3308 & 1371.1044 \end{bmatrix}$$

$$\sum_1 = \begin{bmatrix} 319.5684 & 130.8348 \\ 130.8348 & 875.3956 \end{bmatrix}$$

5. Equation of the quadratic boundary obtained when $\sum_0 and \sum_1$are different:-
$\frac{1}{2}X^T(\sum_1^{-1} - \sum_0^{-1})X + (\mu_0^T \sum_0^{-1} - \mu_1^T \sum_1^{-1})X - (\log(\frac{\phi}{1-\phi}) + \frac{1}{2}\log(\frac{|\sum_0|}{|\sum_1|})) + \frac{1}{2}(\mu_1^T \sum_1^{-1} \mu_1 - \mu_0^T \sum_0^{-1} \mu_0) = 0$

The above equation is quadratic in X. the curve can be any conic. In this example, as per the values of coefficients the curve comes to be hyperbola.

The quadratic boundary obtained can be plotted as:

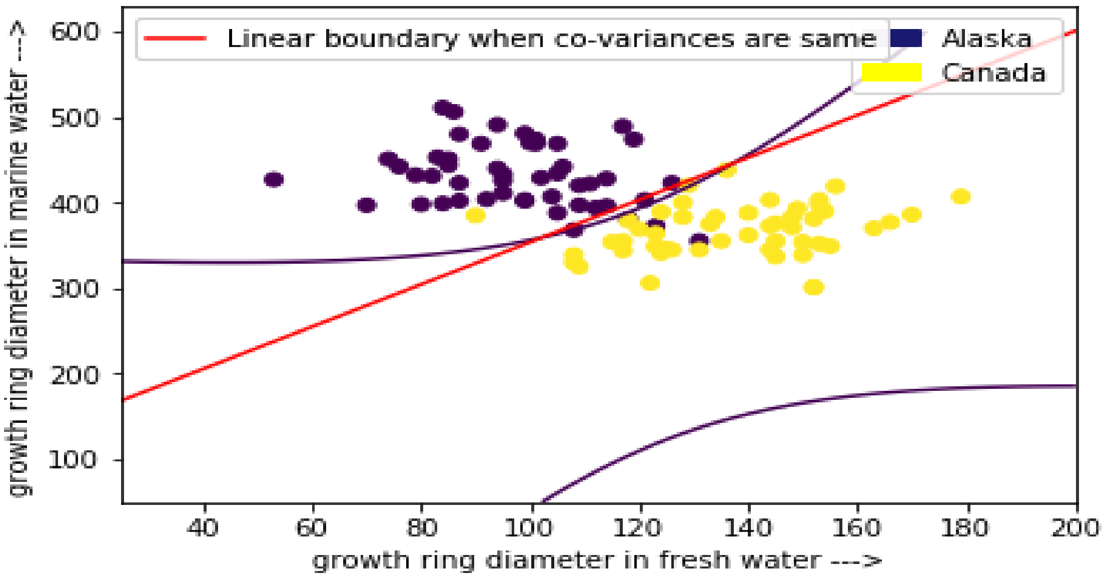

Figure 18: Quadratic boundary by GDA when covariance matrix is different

6. The quadratic boundary by GDA offers a non linear way of classifying data. Although it computes an additional sigma parameter, but it provides a better estimate of the decision boundaries.

   In case of linear boundary, there is a stronger assumption of equality of co-variance matrices. So linear boundary will perform better when underlying data will follow the assumption.