COL774

Report of Assignment 1

DEVANSH DALAL (2012CS10224)

Note: In all the questions all the columns of input matrix X are normalized first before doing any processing.

Q1. Linear Regression

a. **Batch Gradient Descent**:

The Batch gradient descent is implemented in function *p1(η,ε)* which takes the learning rate η and stopping criteria ε as arguments for optimizing $J(\vartheta)$. The variation of number of iterations for convergence as the function of η for $\varepsilon=0.000001$ is shown in the table below.

$$J(\theta) = \frac{1}{2m}\sum_{i=0}^{m}(y^{(i)} - \theta^T x^{(i)})$$

η	Θ_0	Θ_1	no. of iterations
0.001	5.8391	4.6169	13538
0.01	5.8391	4.6169	1349
0.1	5.8391	4.6169	130
0.5	5.8391	4.6169	21
1	5.8391	4.6169	4
1.6	5.8391	4.6169	28
1.9	5.8391	4.6169	127
1.99	5.8391	4.6169	1320
2.1	_	_	doesn't converge

VARIATION OF CONVERGENCE ON LEARNING RATE

Observations:

- The optimal values of η, θ, ε were observed as follow.

$$\Theta_0 = 5.8391$$

$$\Theta_1 = 4.6169$$

$$\eta = 1$$

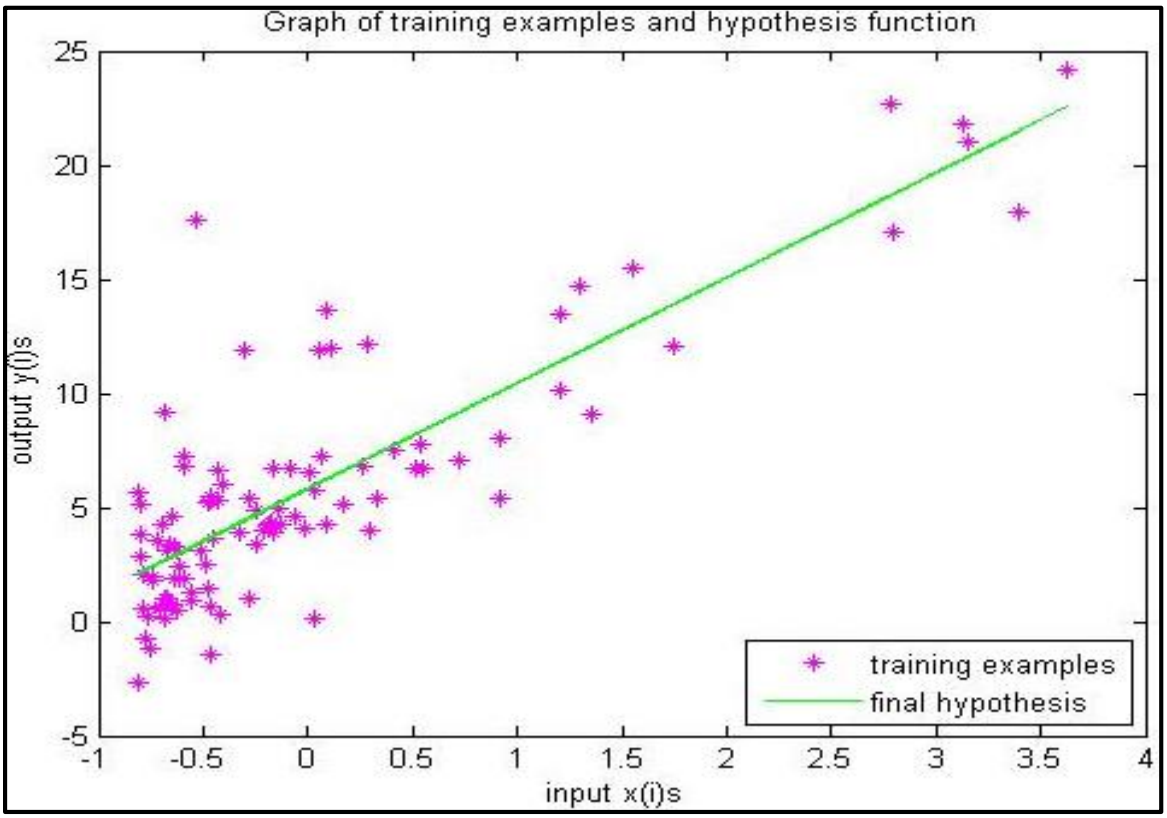
$$\varepsilon = 0.000001$$

iterations=4

- At higher values of learning rate the values of jumps over the optimal value in successive iterations and at about η >= 2.0, the batch gradient doesn't converge.
- One method to solve these overshooting of theta is that to change η dynamically, i.e. decrease η with the increasing number of iterations.
- On increasing stopping criteria the no. of iterations reduced but the accuracy of solution was also decreased.

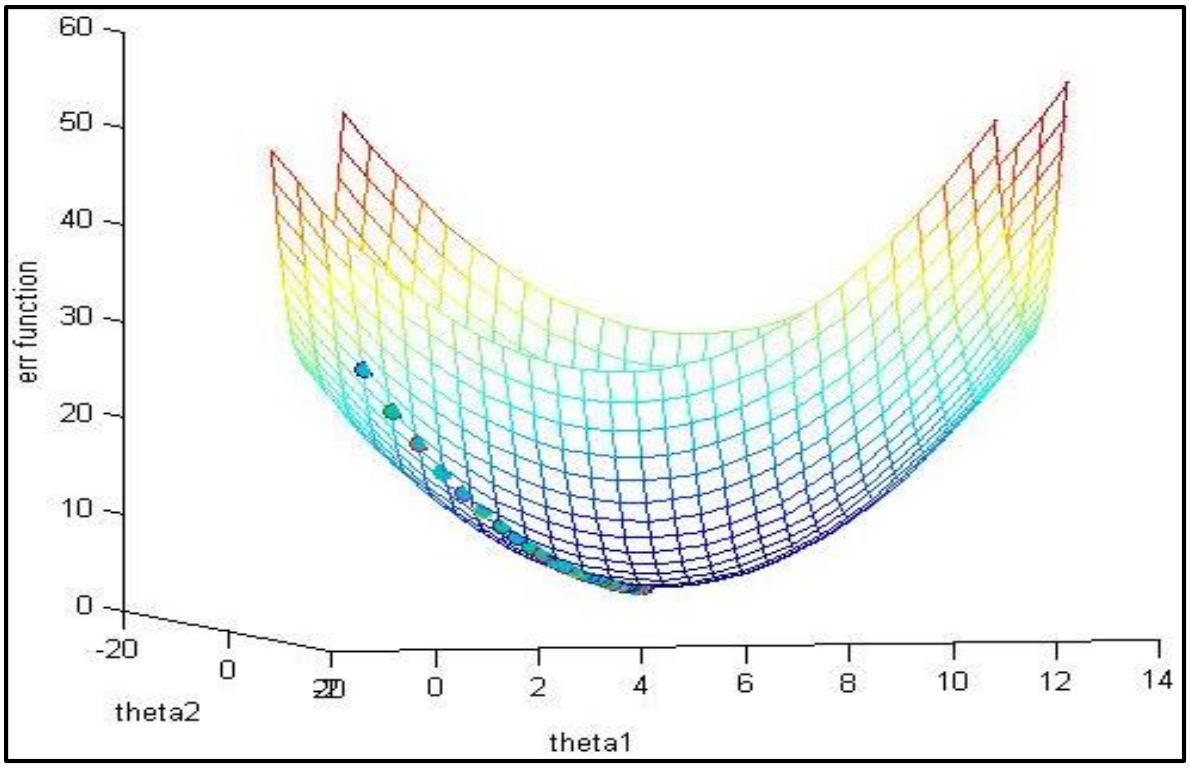
b. **Batch Gradient Descent**:

The hypothesis function learned by my algorithm for η=1 and ε=*0.000001* is shown in the figure below.



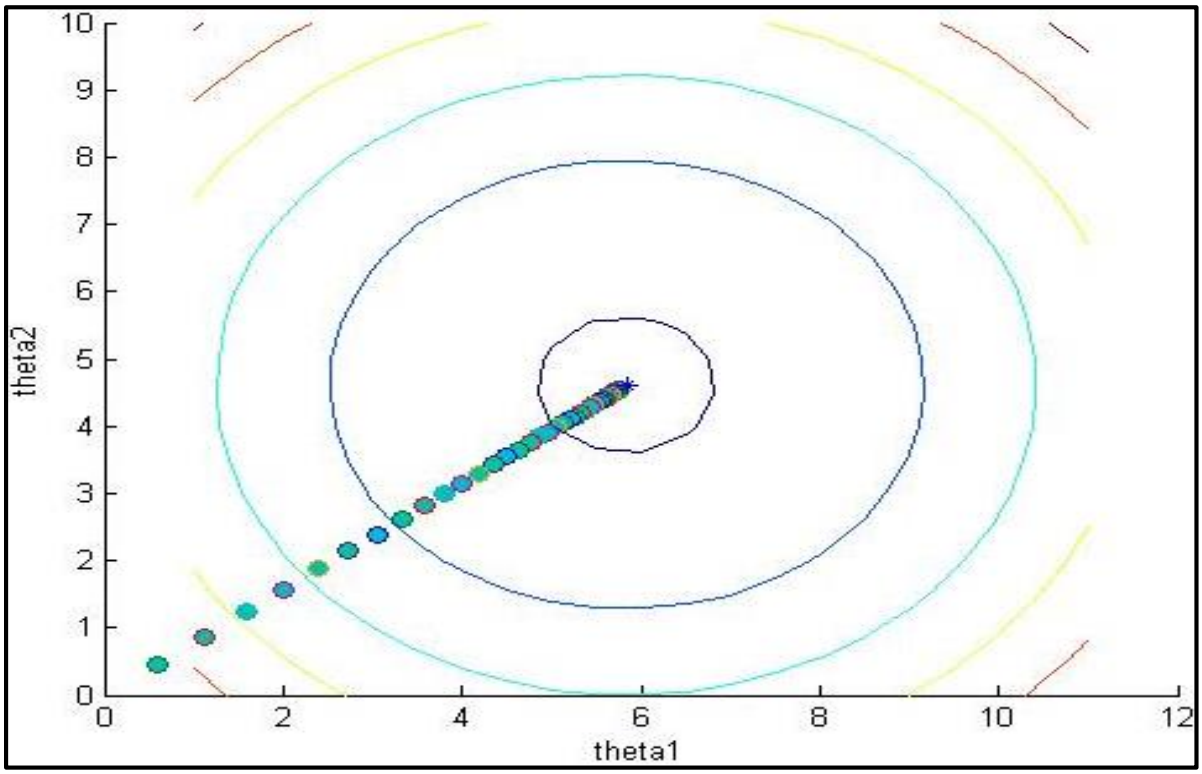
PLOT OF HYPOTHESIS VS TRAINING EXAMPLES

c. I have plotted the J(θ) for different values of θ using the meshgrid function in a 3D plot and the animation shows the actual convergence of gradient descent by displaying the error value using the current set of parameters at each iteration at a time gap of 1 second.



CONVERGENCE OF GRADIENT DESCENT

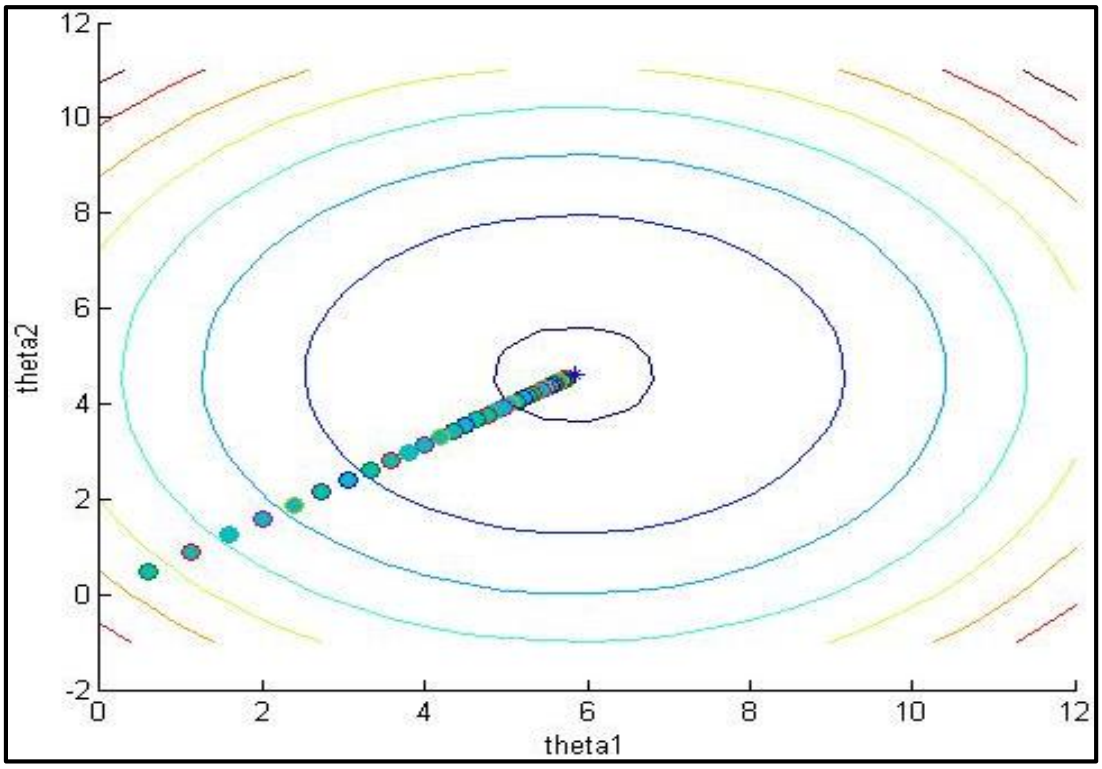
d. **Contour of the error function**



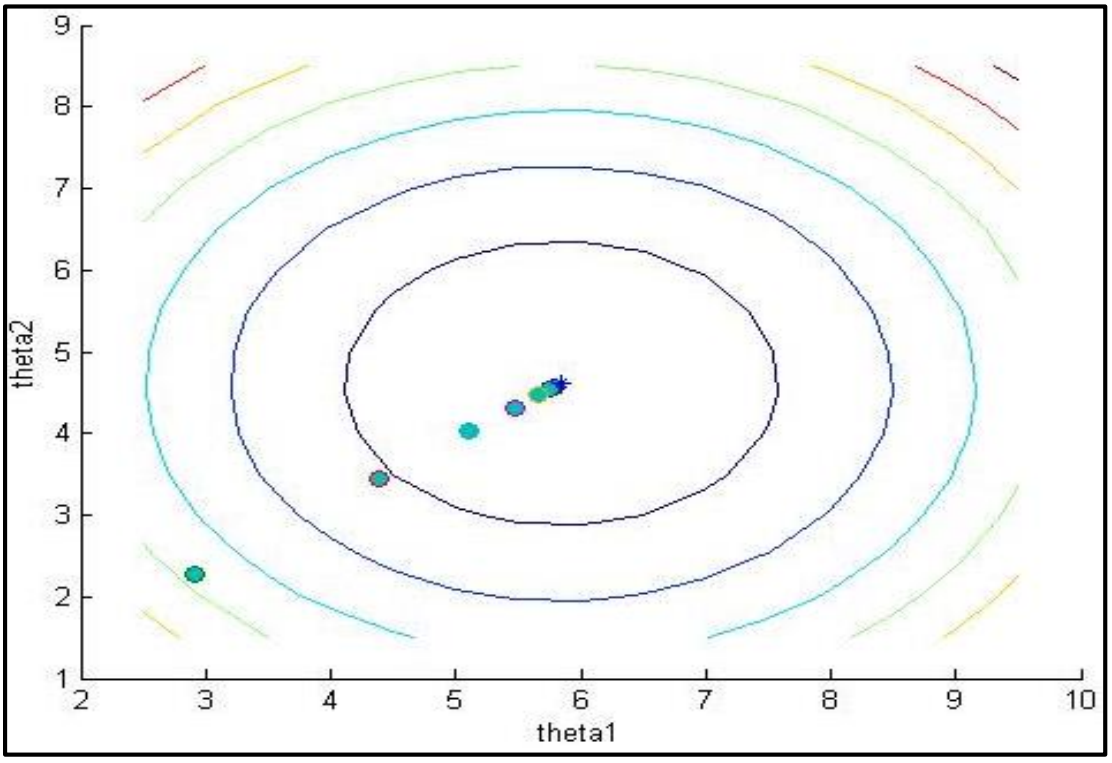
CONVERGENCE ON CONTOUR PLOT

e. Contour plot at different learning rates.

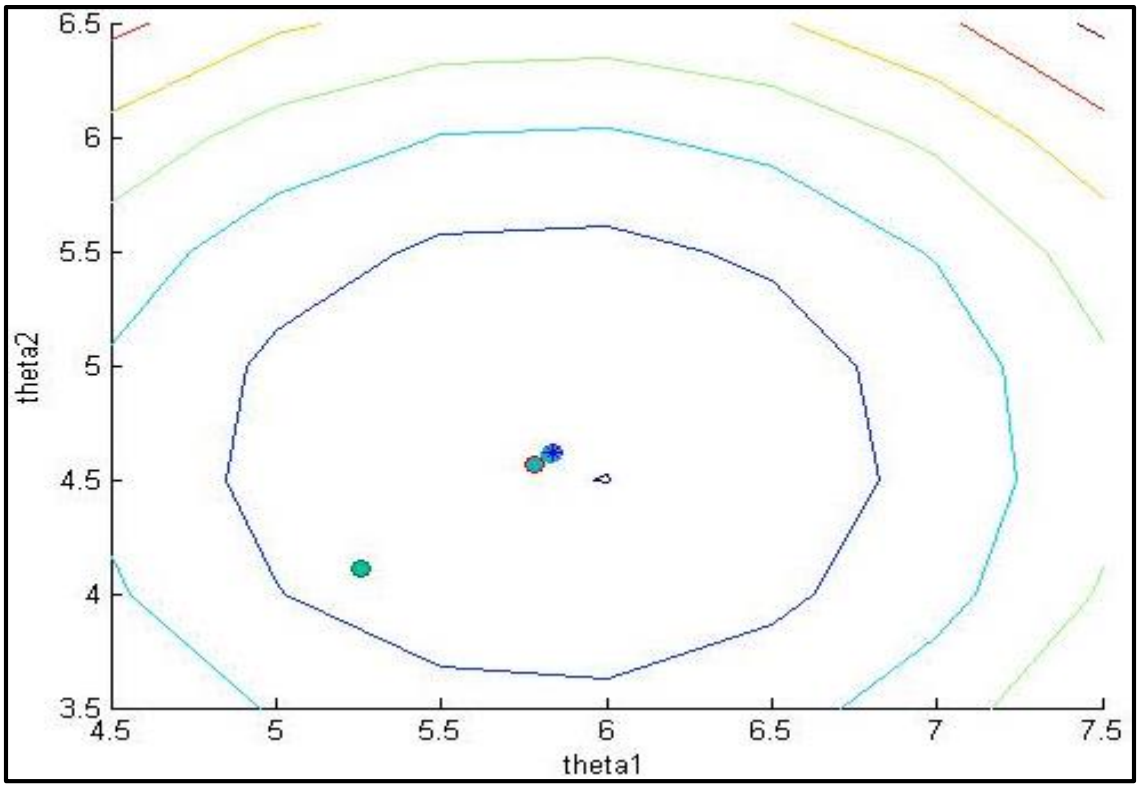
Below are the contour plot for different learning rates as asked in the question.



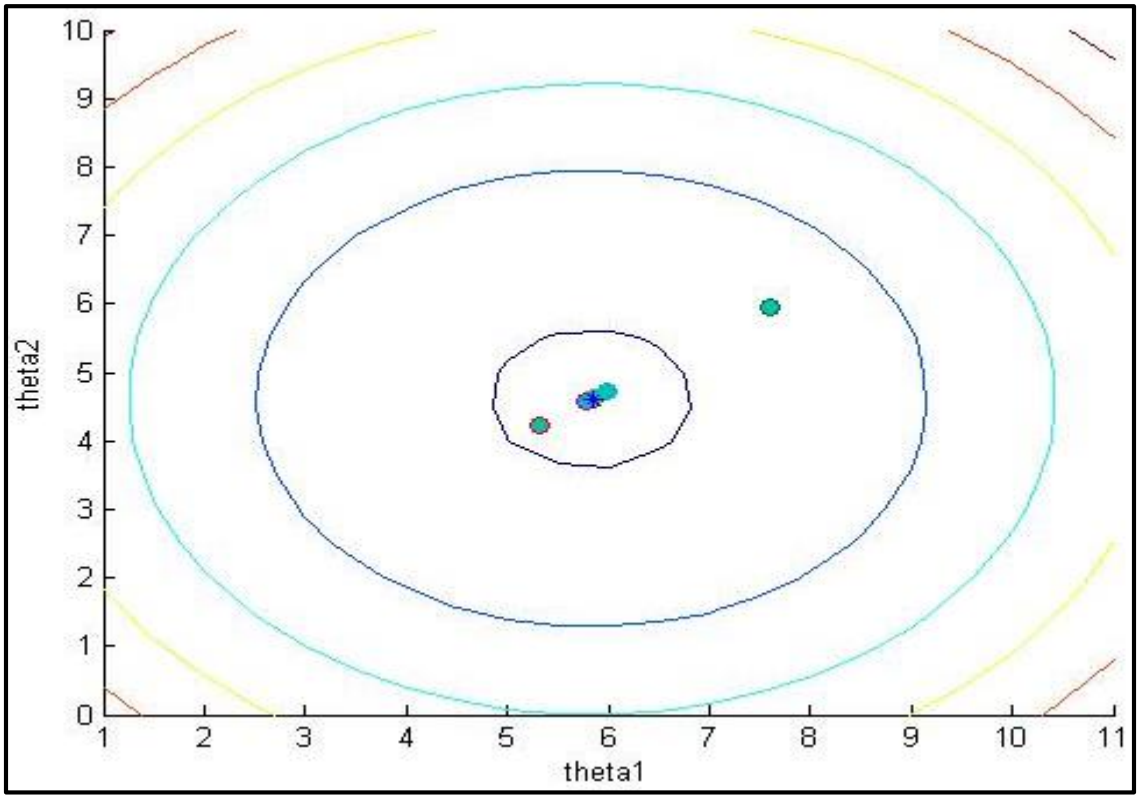
FOR LEARNING RATE = 0.1



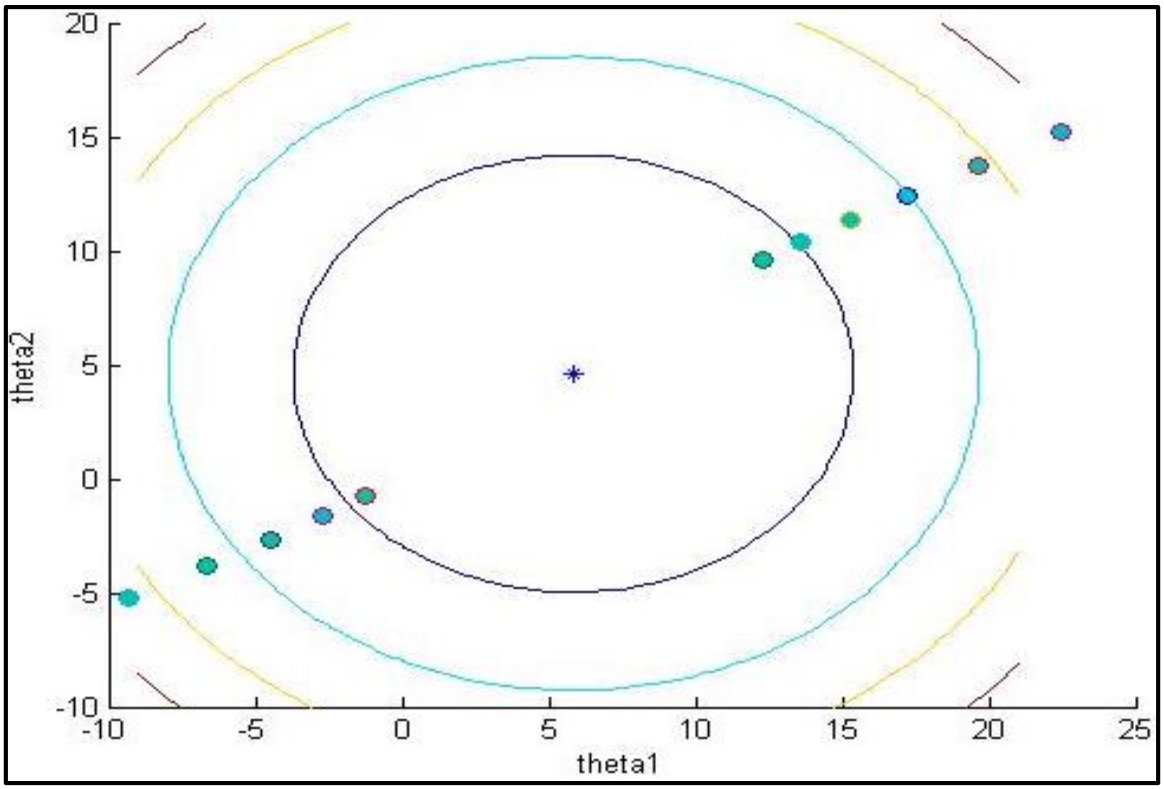
FOR LEARNING RATE = 0.5



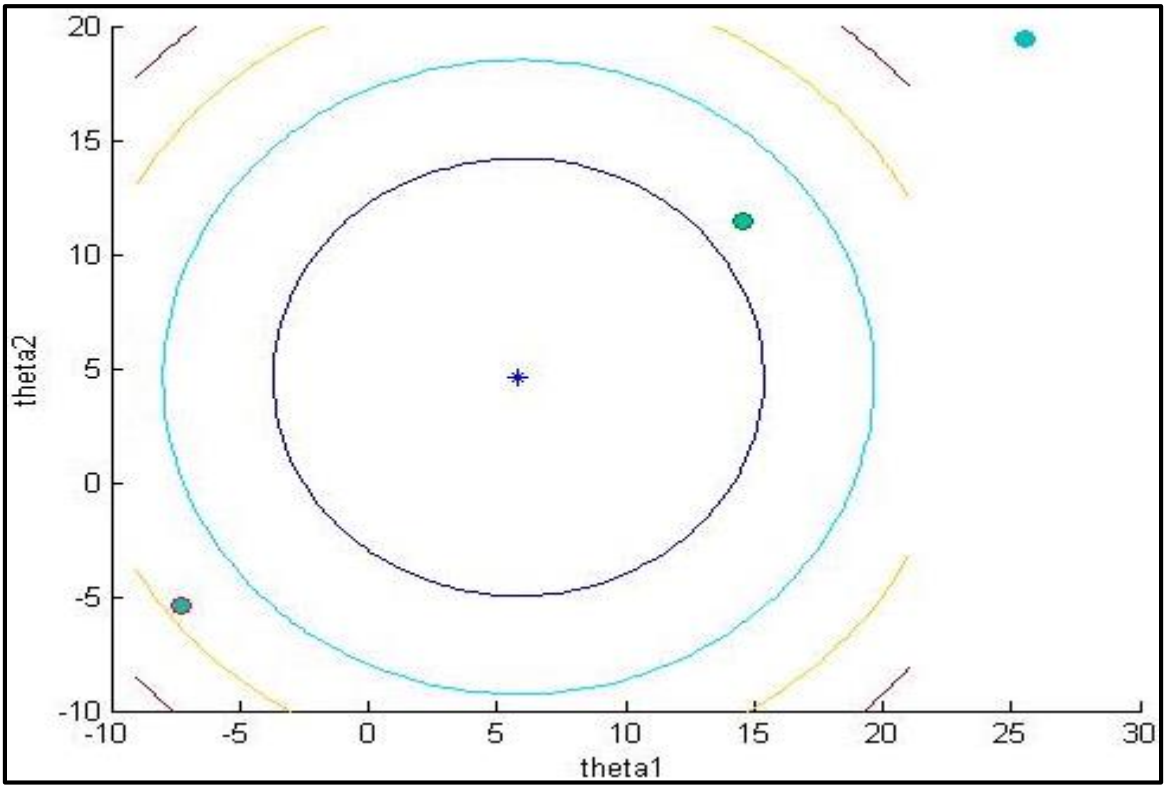
FOR LEARNING RATE = 0.9



FOR LEARNING RATE = 1.3



FOR LEARNING RATE = 2.1



FOR LEARNING RATE = 2.5

In the last 2 graphs the gradient descent is actually not converging, it is diverging away from the optimal value of the theta and also in the case of learning rate = 0.1 or less the convergence is very slow but it converges. So for an efficient leaning algorithm must choose learning rate appropriately for better performance.

Q2. Locally weighted Linear Regression

a. **Unweighted linear regression**:

The Unweighted linerar regression using normal equations gives.

$$X^T * X * \theta = X^T * Y$$

Which gives

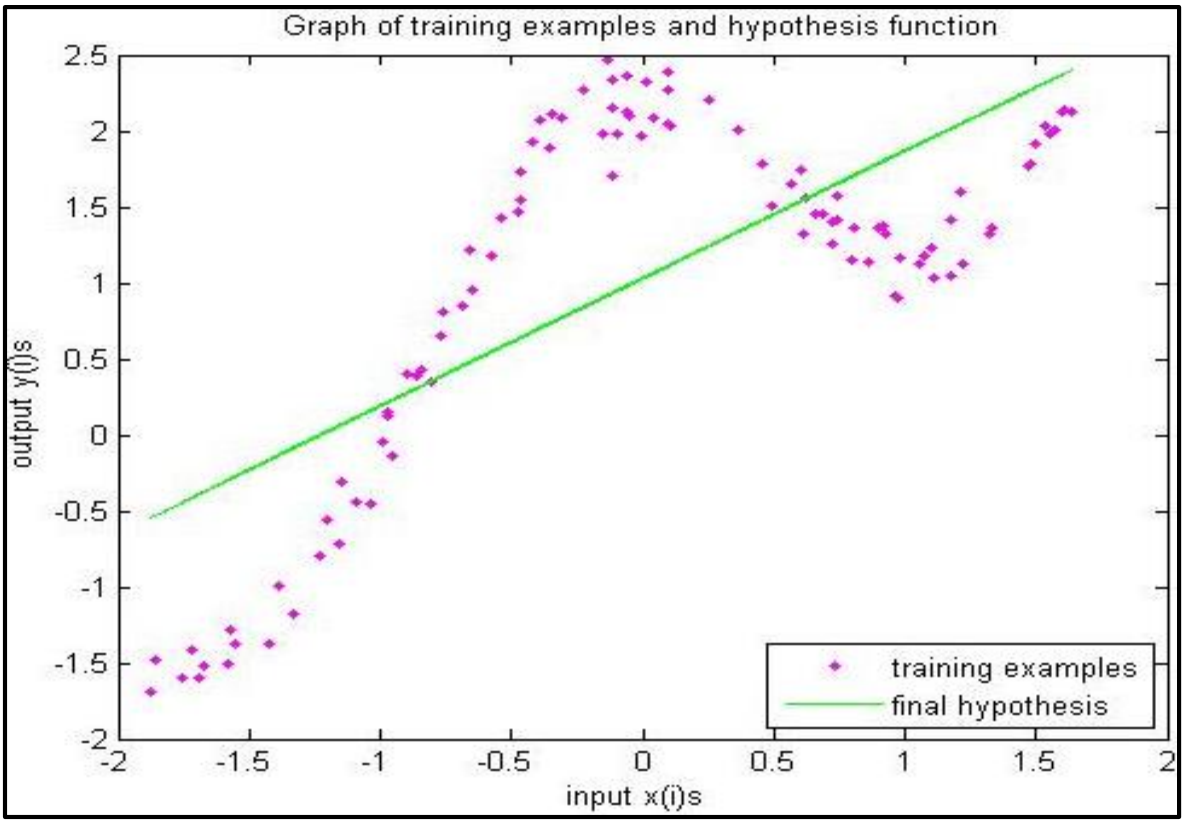
$$\theta = (X^T * X)^{-1} * X^T Y$$

$$\theta = pinv(X) * Y$$

Where pinv() returns the pseudo inverse of a matrix and is builtin matlab.

Using this equation, the optimal θ was observed as:

$$\boldsymbol{\theta}^* = \begin{bmatrix} \mathbf{1.0313} \\ \mathbf{0.8394} \end{bmatrix}$$



UNWEIGHTED REGRESSION USING NORMAL EQN

b. **Weighted linear regression**:

For weighted linear regression, the matrix W to be determined first. So by derivation W can be found as:

$$W = diag(w)$$

$$where\ w^{(i)} = \mathrm{e}^{-\frac{(x-x^{(i)})^2}{2\tau^2}}\ for\ a\ particular\ point\ x$$

Derivation of normal equation in weighted linear regression

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2\right)$$

In the matrix notation we can write it as :

$$J(\theta) = \frac{1}{2m}(X\theta - Y)^T W(X\theta - Y),$$

$$\nabla_\theta J(\theta) = \frac{1}{2m}\nabla_\theta((X\theta - Y)^T W(X\theta - Y))$$

$$= \frac{1}{2m}\nabla_\theta((\theta^T X^T - Y^T)W(X\theta - Y))$$

$$= \frac{1}{2m}\nabla_\theta(\theta^T X^T WX\theta - Y^T WX\theta - \theta^T X^T WY + Y^T WY)$$

$$= \frac{1}{2m}(\nabla_\theta(\theta^T X^T WX\theta) - \nabla_\theta(Y^T WX\theta) - \nabla_\theta(\theta^T X^T WY) + \nabla_\theta(Y^T WY))$$

$$= \frac{1}{2m}(2X^T WX\theta - 2X^T WY)$$

Putting $\nabla_\theta J(\theta) = 0$, we get :

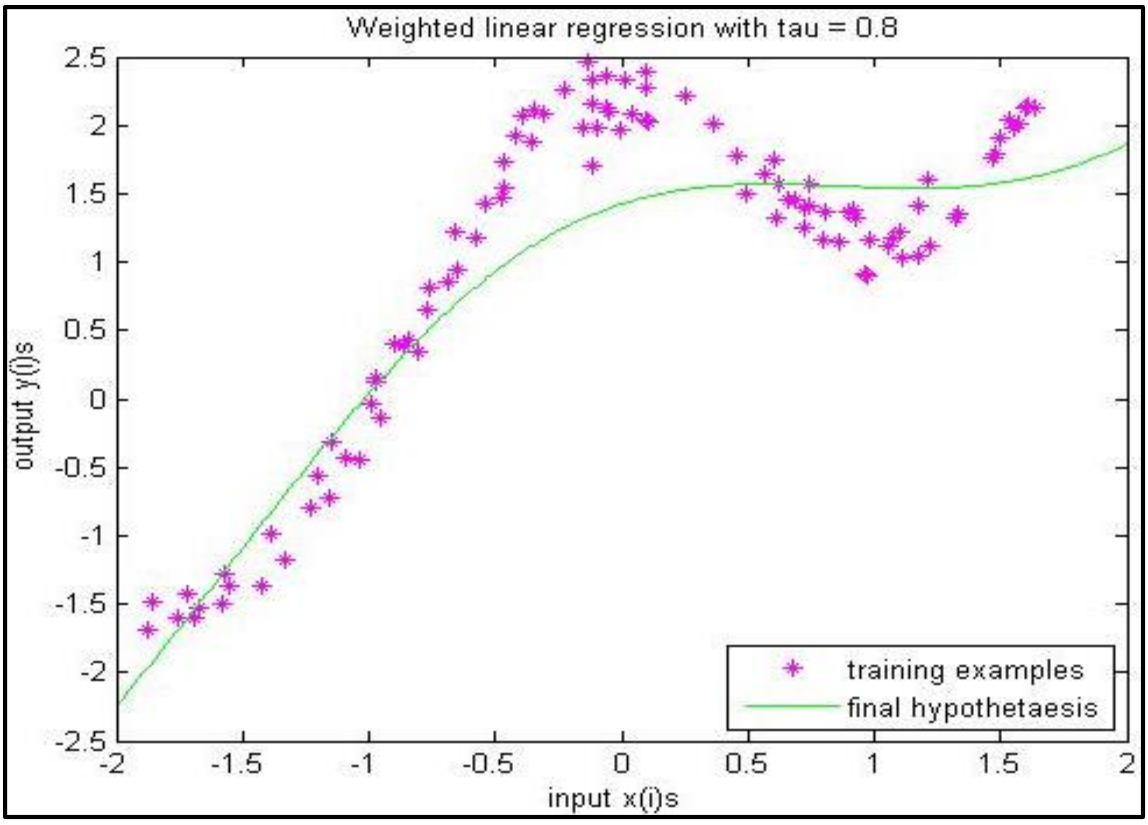
$$X^T WX\theta - X^T WY = 0$$

$$\theta = (X^T WX)^{-1} X^T WY$$

So the expression of θ is given by

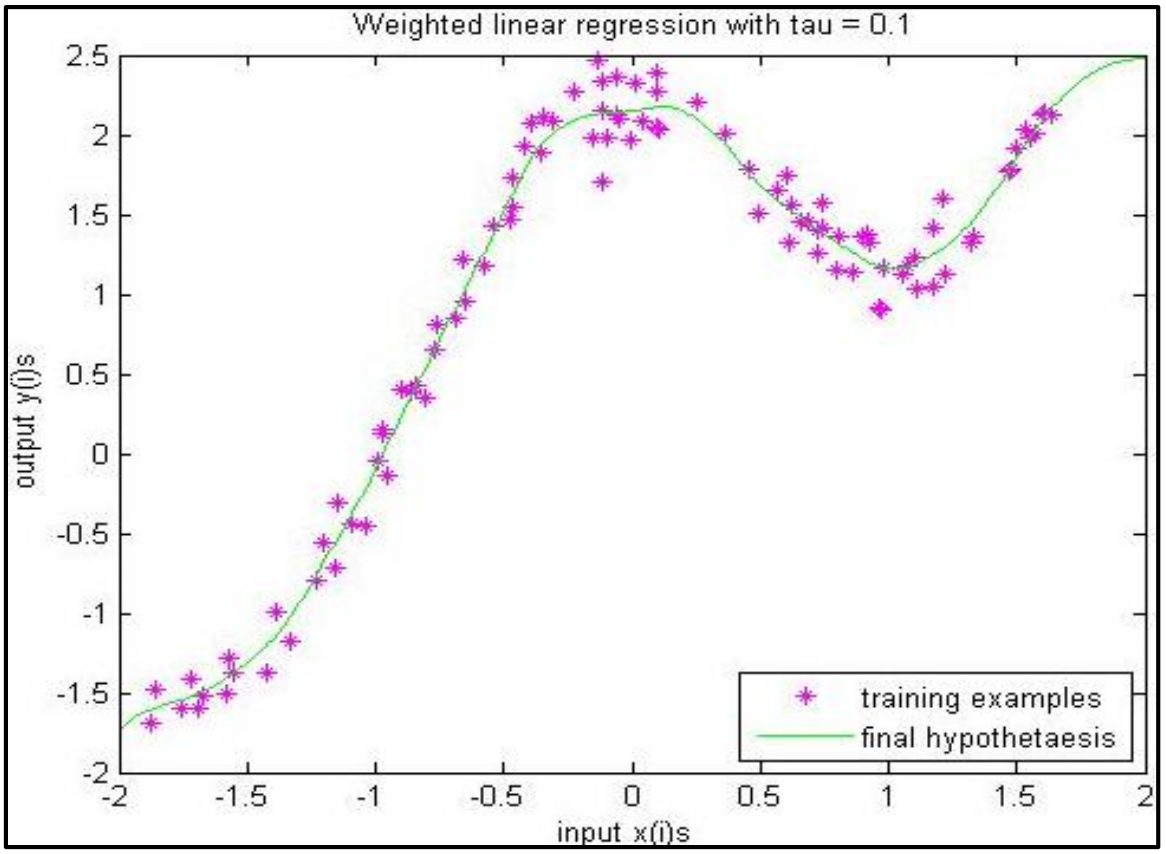
$$\theta = (X^T * W * X)^{-1} * X^T * W * Y$$

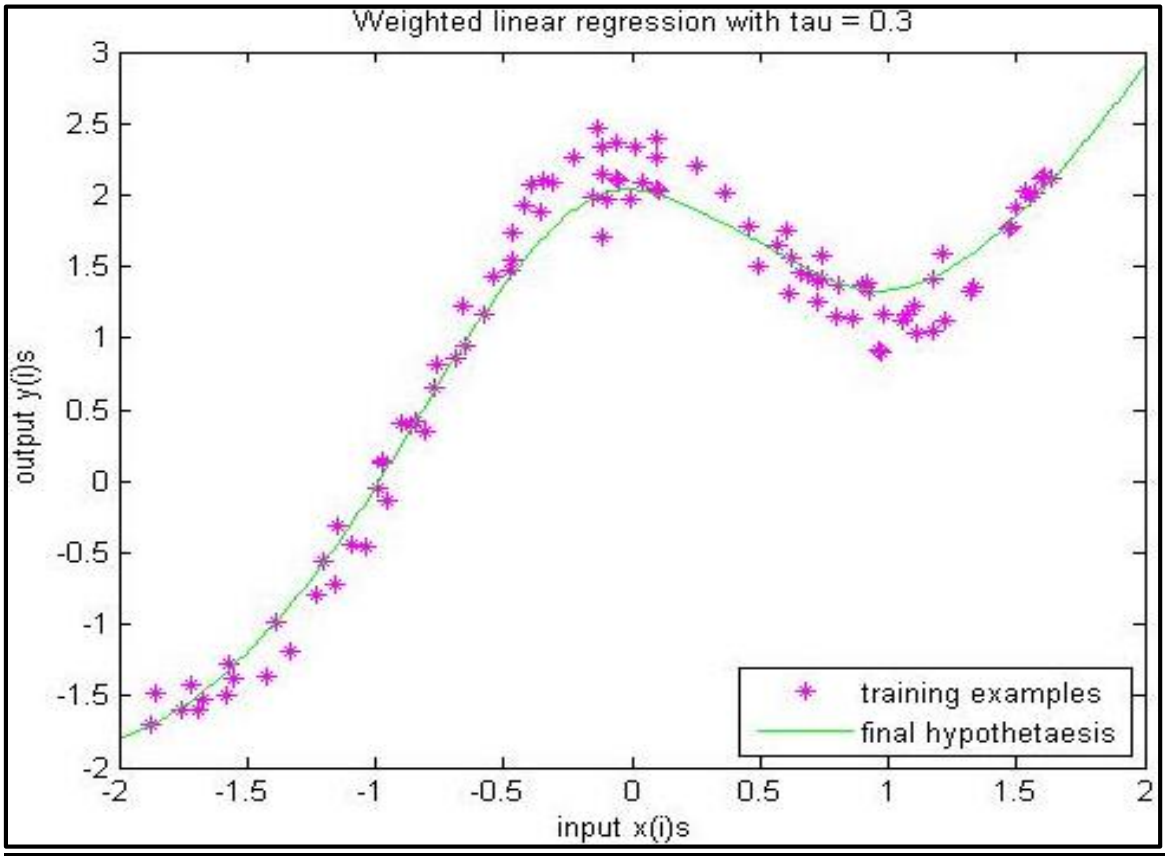
Following is the figure showing the learnt θ learned for bandwidth, $\tau = 0.8$.

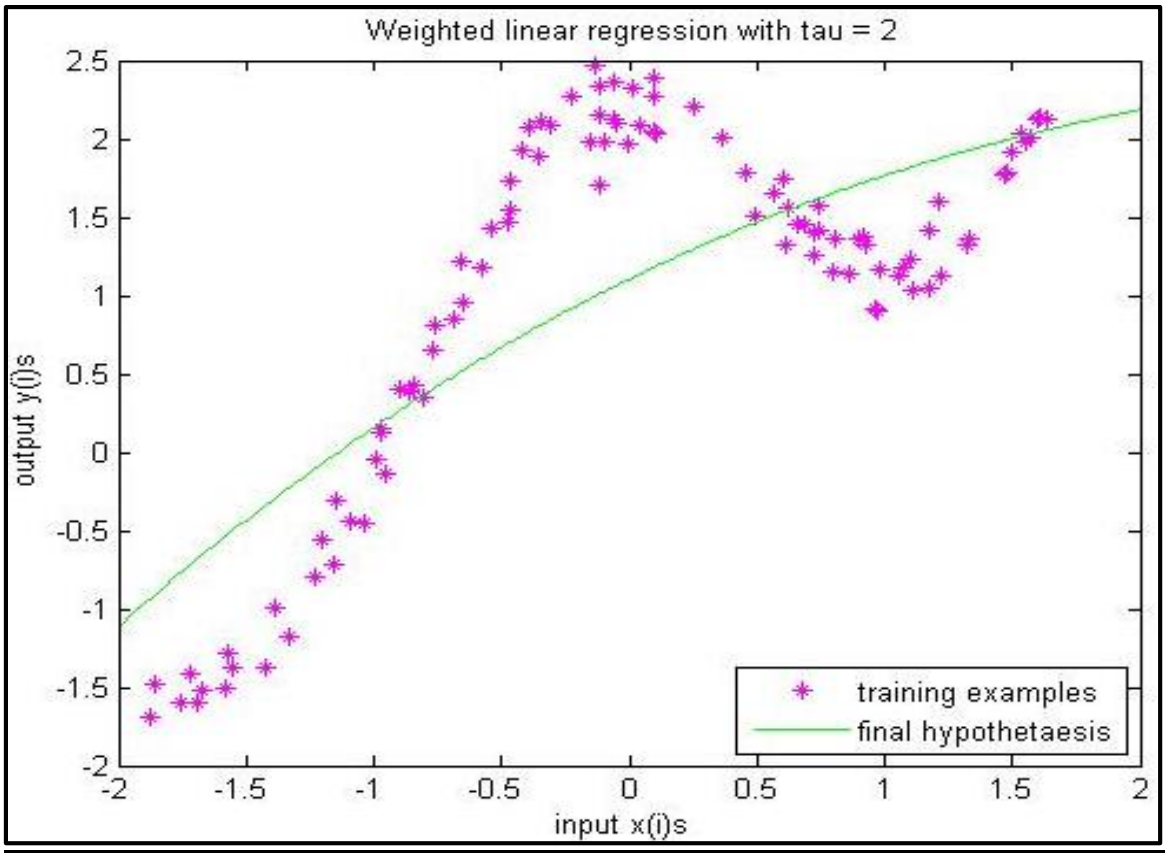


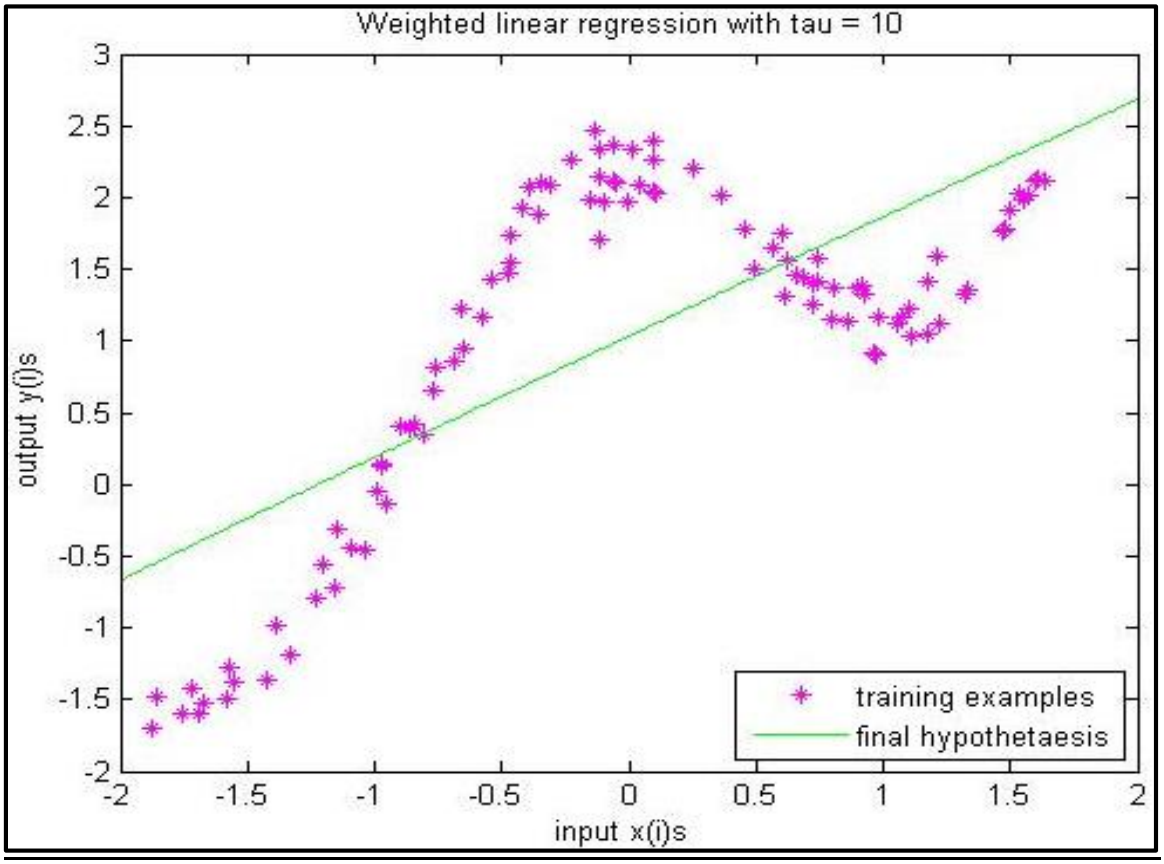
WEIGHTED REGRESSION USING NORMAL EQUATIONS FOR BANDWIDTH=**0.8**

c. **Plot for various values of τ:**


Weighted linear regression with tau = 0.1
output y(i)s
input x(i)s
training examples
final hypothetaesis


Weighted linear regression with tau = 0.3
output y(i)s
input x(i)s
training examples
final hypothetaesis


Weighted linear regression with tau = 2
output y(i)s
input x(i)s
training examples
final hypothetaesis


Weighted linear regression with tau = 10
output y(i)s
input x(i)s
training examples
final hypothetaesis

From the plots for various values of tau it is observed that for larger values like 10 or 2 of bandwidth we may underfit the data while for very small values like 0.1 or 0.3 may lead to overfitting and such kind learning essentially fails to predict the behavior of new test points. So a moderate value of tau is better in such situations, for e.g tau =0.3. Also our choice depends on the application and what distribution of data is coming , for example if the data is noisy then we would prefer larger value of tau. On the other hand if the data is not noisy and is well behaved and predictable then we can be more aggressive and choose smaller value of tau.

Q 3. Logistic Regression

a. **Newton's method for optimizing** $\mathrm{L}(\theta)$

The update equation for Newton-Raphson method is given by:

$$\theta = \theta - H^{-1} * (\nabla_\theta \mathrm{L}(\theta))$$

$$where\ H\ is\ the\ hessian\ of\ L\ w.r.t\ \theta$$

On solving for $\nabla_\theta \mathrm{L}(\theta)$, we have

$$\frac{d\mathrm{L}(\theta)}{d\theta_j} = (y - h_\theta(x))x_j$$

$$where\ \ h_\theta(x) = \frac{1}{1+e^{-\theta' x}}.$$

And in matrix form, we have

$$\nabla_\theta \mathrm{L}(\theta) = X^T(Y - h_\theta(X))$$

Similarly, H_{ij} is obtained by differentiating $\frac{d\mathrm{L}(\theta)}{d\theta_j}$ w.r.t θ_i

$$H_{ij} = -\sum_{k=1}^{m} x_i^k x_j^k h_\theta(x^k)(1 - h_\theta(x^k))$$

In matrix representation, H can be finally written in closed form as

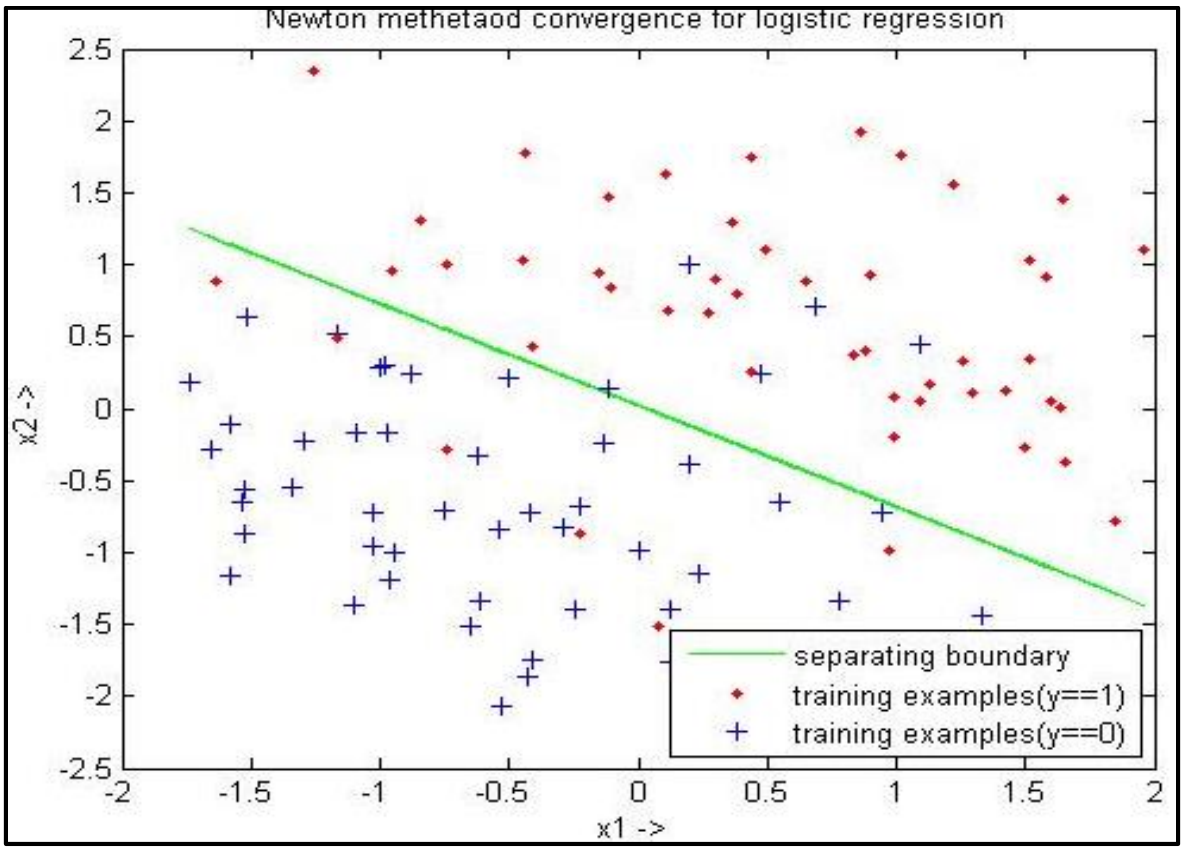
$$H = -X^T * diag\left(h_\theta(X).*\left(1 - h_\theta(X)\right)\right) * X$$

The Newton's method for optimizing $\mathrm{L}(\theta)$ is implemented in function *p1()*. The value of optimal θ observed with stopping criteria 0.000001 was:

$$\boldsymbol{\theta}^* = \begin{bmatrix} \mathbf{-0.0472} \\ \mathbf{1.4675} \\ \mathbf{2.0764} \end{bmatrix}$$

The number of iterations it took was **5** indicating that this method converges relatively faster than other methods like batch or stochastic gradient descent.

b. **Plot showing working of newton's method**



LEARNT HYPOTHESIS FUNCTION USING NEWTON'S METHOD

ϵ	no. of iterations
0.0000001	5
0.000001	5
0.00001	5
0.0001	5
0.001	4
0.01	4
0.1	3
1	2
2	1

NUMBER OF ITERATIONS WITH STOPPING CRITERIA

The above table suggests that newton's iteration converges very fast to the optimal values and the relative error decreases exponentially with the increasing number of iterations.

Q 4. Gaussian discriminant analysis:

a. **Parameters in GDA implementation**

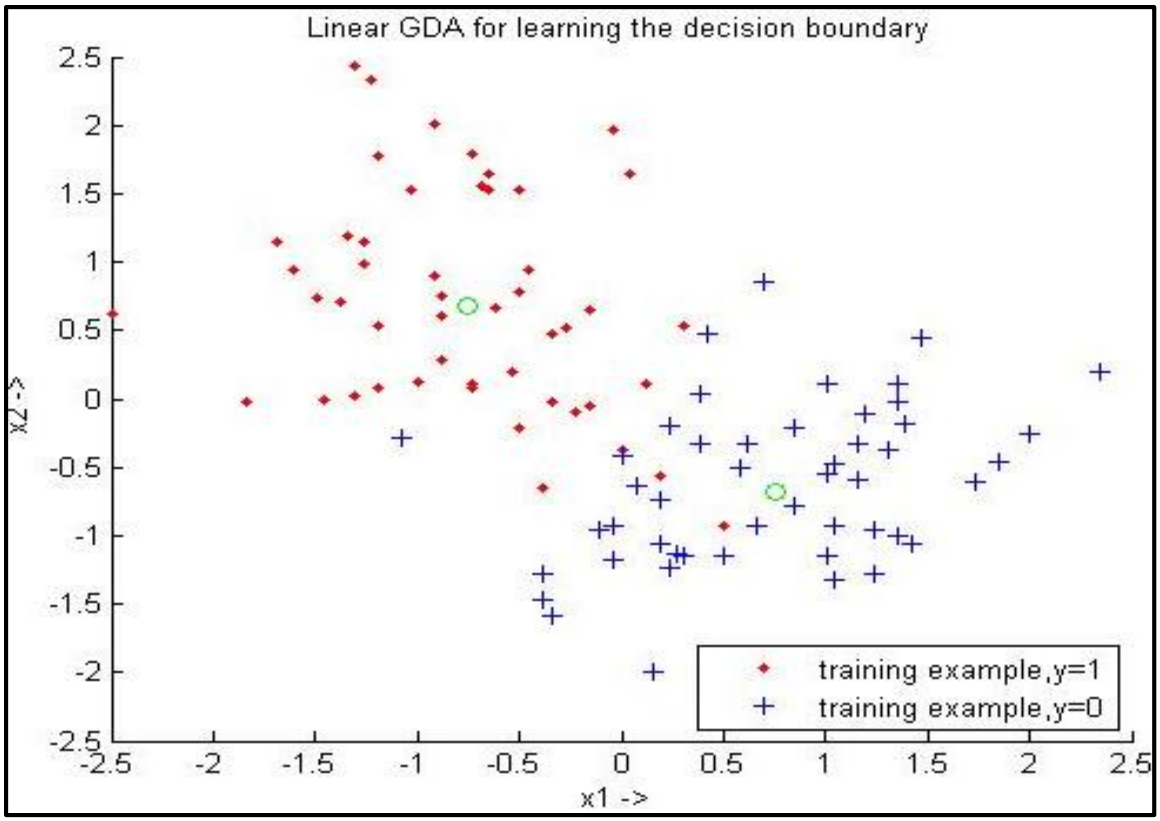
In this part a new column of zeros is not needed in the front of X matrix. The μ_0, μ_1 and Σ for the given dataset with all columns of normalized X were observed as:

$$\boldsymbol{\mu_1} = \begin{bmatrix} \mathbf{-0.7515} \\ \mathbf{0.6817} \end{bmatrix}$$

$$\boldsymbol{\mu_0} = \begin{bmatrix} \mathbf{0.7515} \\ \mathbf{-0.6817} \end{bmatrix}$$

$$\boldsymbol{\Sigma_0} = \boldsymbol{\Sigma_1} = \boldsymbol{\Sigma} = \begin{bmatrix} \mathbf{0.4252} & \mathbf{-0.0222} \\ \mathbf{-0.0222} & \mathbf{0.5253} \end{bmatrix}$$

b. **Plot of input data:**



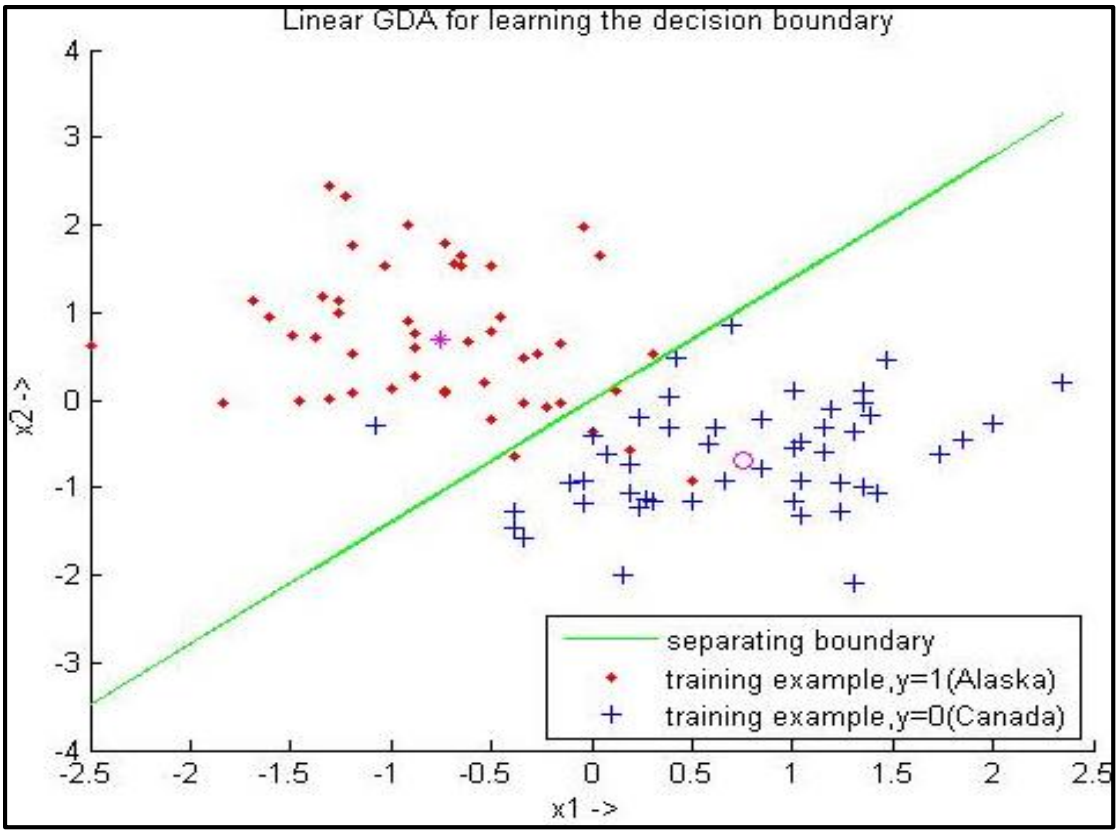
c. **Plot of input data:**

The equation of decision boundary derived is given by:

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

where $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and x_1 and x_2 are the variables representing the x and y axis respectively.

The function p123() plots the following graph which includes plotting the training points as well as the boundary separating the 2 classes.



d. **Generalized GDA:**

The values of μ_0, μ_1, Σ_0, Σ_1 for the given dataset with all columns of normalized X were observed as:

$$\boldsymbol{\mu_1} = \begin{bmatrix} \mathbf{-0.7515} \\ \mathbf{0.6817} \end{bmatrix}$$

$$\boldsymbol{\mu_0} = \begin{bmatrix} \mathbf{0.7515} \\ \mathbf{+0.6817} \end{bmatrix}$$

$$\boldsymbol{\Sigma_0} = \begin{bmatrix} \mathbf{0.4727} & \mathbf{0.1088} \\ \mathbf{0.1088} & \mathbf{0.4094} \end{bmatrix}$$

$$\boldsymbol{\Sigma_1} = \begin{bmatrix} \mathbf{0.3778} & \mathbf{-0.1533} \\ \mathbf{-0.1533} & \mathbf{0.6413} \end{bmatrix}$$

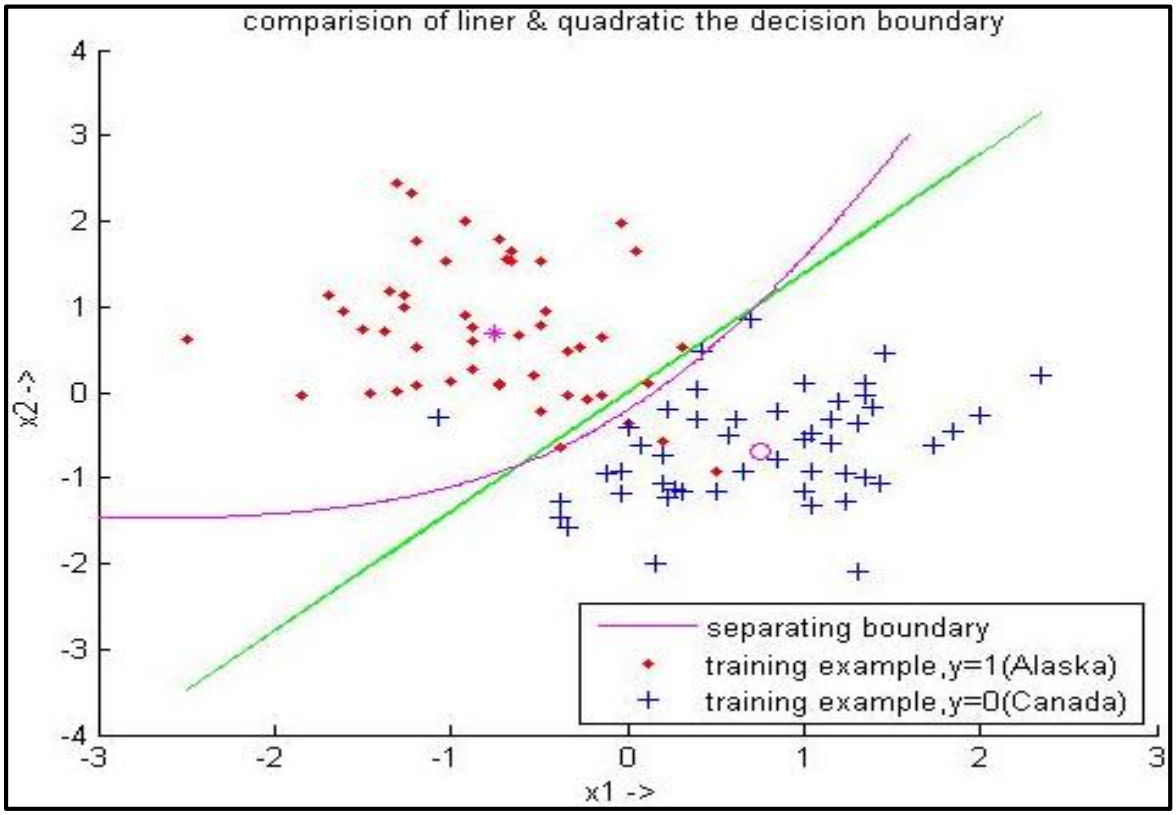
e. **Quadratic boundary in generalized GDA:**

The equation of decision boundary derived is given by:

$$\log(1-\phi) - \frac{(X-\mu_0)^T}{2} * \Sigma_0^{-1} * (X-\mu_0) - \frac{\log|\Sigma_0|}{2}$$
$$= \log\phi - \frac{(X-\mu_1)^T}{2} * \Sigma_1^{-1} * (X-\mu_1) - \frac{\log|\Sigma_1|}{2}$$

where $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and x_1 and x_2 are the variables representing the x and y axis respectively.

The equation of quadratic decision boundary along with previous plot is shown the figure below:



f. **<u>Analysis of linear and quadratic boundaries</u>:**

The quadratic decision boundary is actually hyperbolic due to the xy term if we simplify the equation given in previous part. The quadratic line is concave towards training examples with y=1 as these type of points are distributed more spherically when viewed from the linear line than other kind of points. It is clearly visible that the quadratic boundary is distinguishing one more point correctly from the linear line even in the given set of training examples. So if the distribution of new coming points doesn't change much, the quadratic boundary is expected to do marginally better as compared to the linear one. Actually linearity constraint on the straight line is restricting it from doing better job in classification as compared to the quadratic boundary. In actual practice, quadratic boundary should be preferred more as compared to the linear because standard deviation of the 2 classes the generally not completely identical as verified by the values of $\boldsymbol{\Sigma_0}$ and $\boldsymbol{\Sigma_1}$ from the previous part above.