# Assignment Report

## 1A:

Initially in the code, we input the values of the parameters of Beta Distribution and the number of surveys being conducted. Then we collect details of each survey. Then we plot the posterior probabilities along with the foremost prior probability.

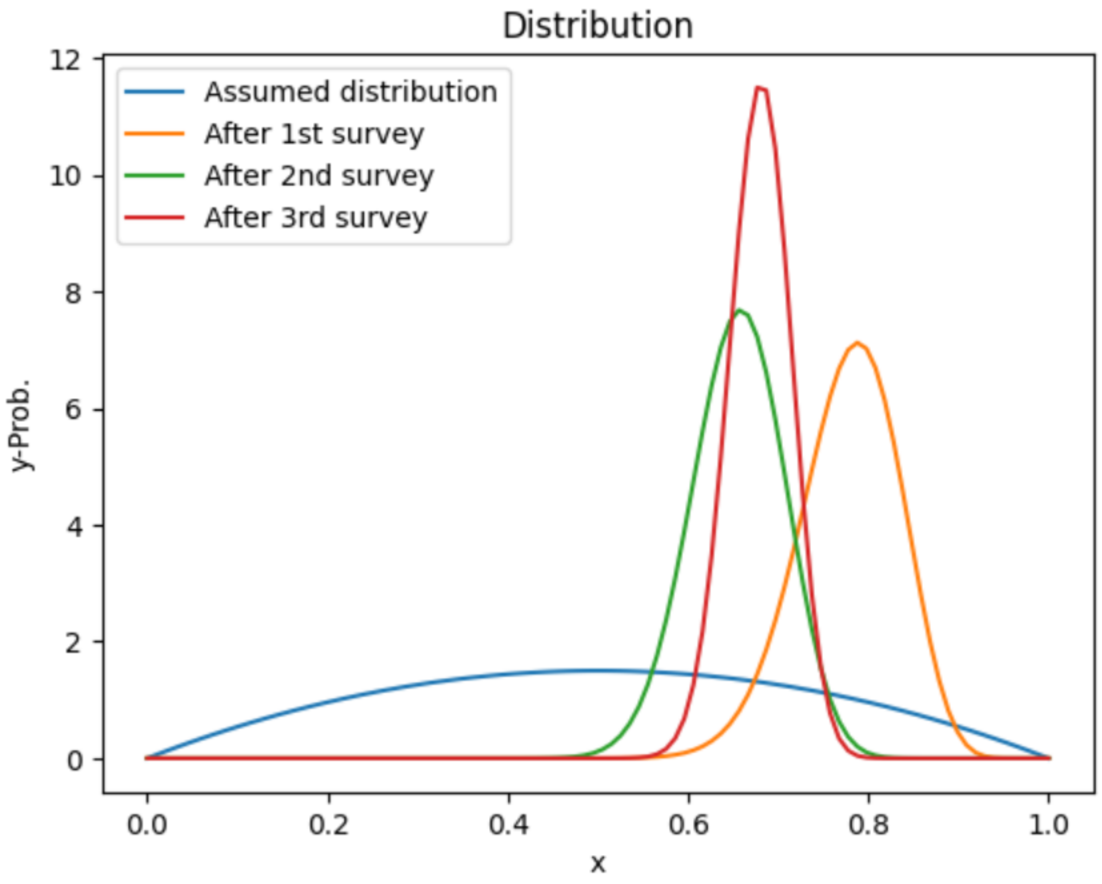
Based on the parameters of the Beta distribution (2,2), we plot the prior probability of *s*. After collecting the results of the 1st survey (40 of 50 in favour), we evaluate the likelihood of *s* and plot the 1st posterior probability of *s*.

The customers' responses in the survey conducted satisfies the properties of Binomial distribution. They are:

1. There are only two possible outcomes: like or dislike the product.
2. The total number of customers surveyed in a survey is fixed.
3. For each response of a customer, the probability of liking the product changes.

We neglect the constant while calculating the posterior probability *s* as at the end, as we restrict the area under the product of the likelihood of *s* and the prior probability of *s* to be equal to 1. Henceforth, we arrive at the likelihood of *s*.

After the 2nd survey (13 of 30 in favour), we plot the posterior probability of *s* in the same fashion considering the posterior probability of the 1st survey as the prior probability of the 2nd survey. We consider so because, in the end, the 2nd survey just gives more data which can be combined with the previous data. This 'new' combined data is used to calculate the likelihood and plot the posterior probability of *s* after the 2nd survey with the original prior probability and new likelihood. After the 3rd survey (70 of 100 in favour), we repeat the process and arrive at the plot shown below.


Distribution
y-Prob.
x
Assumed distribution
After 1st survey
After 2nd survey
After 3rd survey
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

## 1B:

The submitted code can be used to develop 10 different types of  models with varying in algorithms. They are:

1. Normal Gradient Descent without regression.
2. Normal Gradient Descent with q = 0.5 regression.
3. Normal Gradient Descent with q = 1(Lasso) regression.
4. Normal Gradient Descent with q = 2(Ridge) regression.
5. Normal Gradient Descent with q = 4 regression.

The learning rate and no of iterations at which the model adapts can also be varied(depending on how many iterations the system can handle).
The Algorithm is implemented in the following way:

- At the beginning , we read the data from the provided csv file using the pandas package in python.
- Later , we Normalise and then shuffle the data using appropriate inbuilt python functions.
- Then we perform the requested 80-20 split for training and testing data models.
- Next we define a generateValuesMatrix function which returns data points based on the input test/train dataset
- The main Algorithm of this assignment is "gradientDescent" in which we create a data_train matrix containing the coefficients of the respective degree of equations we use. The matrix for
  1st Degree is

$$\begin{pmatrix} 1 & w11 & w21 & -p1 \\ 1 & w12 & w22 & -p2 \\ . & . & . & . \\ . & . & . & . \end{pmatrix}$$

2nd Degree is

$$\begin{pmatrix} 1 & w11 & w21 & w11^2 & w11w12 & w21^2 & -p1 \\ 1 & w12 & w22 & w12^2 & w12w22 & w22^2 & -p2 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \end{pmatrix}$$

- We next create a column matrix with all the coefficients
  For 1st Degree: For 2nd degree:

$$\begin{pmatrix} W0 \\ W1 \\ W2 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} W0 \\ W1 \\ W2 \\ W3 \\ W4 \\ W5 \\ 1 \end{pmatrix}$$

- We now multiply the above two matrices to get error matrix
  Error matrix for 1st Degree:

$$\begin{pmatrix} W0 \\ W1 \\ W2 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & w11 & w21 & -p1 \\ 1 & w12 & w22 & -p2 \\ . & . & . & . \\ . & . & . & . \end{pmatrix} = \begin{pmatrix} W0+W1w11+W2w21-p1 \\ W0+W1w12+W2w22-p2 \\ . \\ . \end{pmatrix}$$

- For this error matrix, all we have to do is find the square, which can be dine using the numpy package in python.
- Then we create slope(gradient matrix) as shown below

$$\begin{pmatrix} \partial E/\partial W0 \\ \partial E/\partial W1 \\ \partial E/\partial W2 \\ . \end{pmatrix}$$

slope_matrix =

On expansion for degree 1 it becomes :

$$\begin{pmatrix} \Sigma^{N}{}_{n=1}\ (W0+W1w1n+W2w2n-p1)(1) \\ \Sigma^{N}{}_{n=1}\ (W0+W1w1n+W2w2n-p2)(w1n) \\ \Sigma^{N}{}_{n=1}\ (W0+W1w1n+W2w2n-p2)(w2n) \end{pmatrix}$$

- We now find new coefficients by subtracting the product of the slope matrix and learning rate from the old coefficient matrix in case of no regularisation.
- For the other cases, we modify the slope as shown for respective q values
    1. For q = 0.5:
       We multiply the lambda matrix with coefficient matrix power -0.5 and add it to the original slope.
    2. For q = 1:
       We multiply the lambda matrix with coefficient matrix and add it to the original slope.
    3. For q=2:
       We multiply the lambda matrix with coefficient matrix square and add it to the original slope.
    4. For q=4:
       We multiply the lambda matrix with coefficient matrix power 3 and add it to the original slope.

Description of algorithms:
The regression model for degree 1 with 2 variables is
Y = W0+W1x1+W2x2

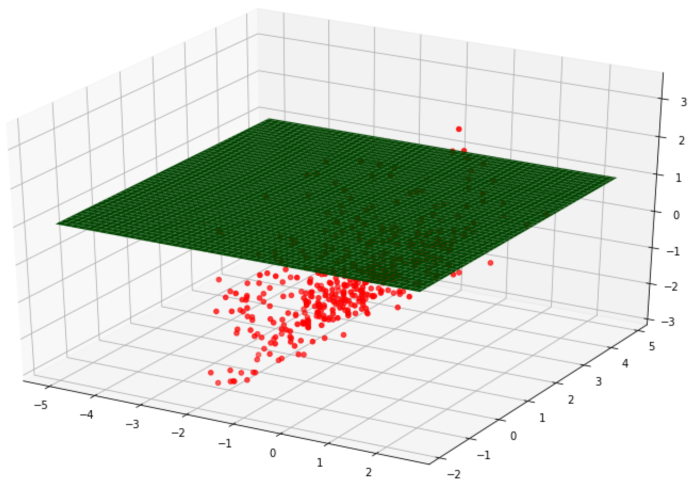
The Error function for it is:
$E(w) = \Sigma ( Y'_n - Y_n)^2 /2$

The Gradient Descent algorithm is an optimization algorithm that is used to get estimates of the coefficients and to minimise the error function.

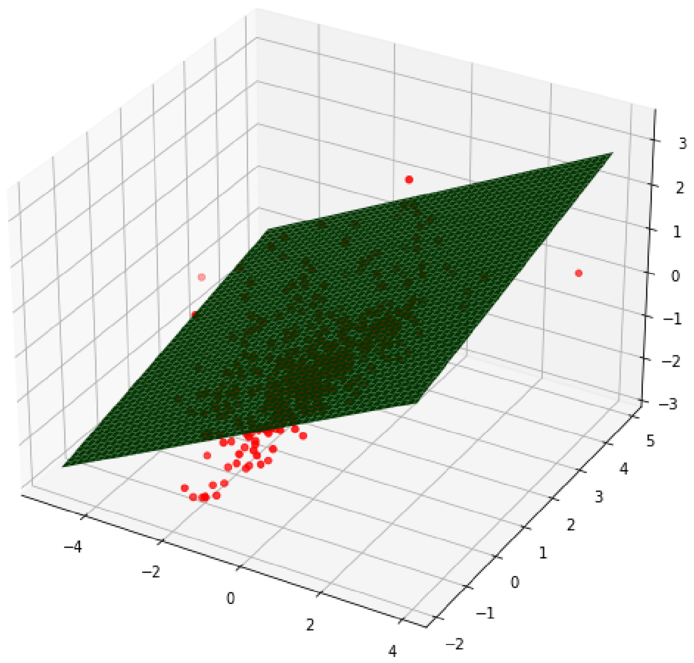
| **Gradient descent** | | | |
|---|---|---|---|
| Degree | Minimum Training Error | Minimum Testing Error | Learning Rate |
| 0 | 862.2154 | 228.7845 | 0 |
| 1 | 277.6781 | 83.1028 | 1.5x10^-5 |
| 2 | 268.5147 | 79.1105 | 1.5x10^-5 |
| 3 | 264.1103 | 80.5043 | 1.5x10^-5 |
| 4 | 271.7286 | 70.3256 | 1.5x10^-6 |
| 5 | 265.7356 | 75.5667 | 0.75x10^-7 |
| 6 | 272.0534 | 175.3611 | 1x10^-9 |
| 7 | 1925.6227 | 4747.9600 | 0.6x10^-10 |
| 8 | 13764.0473 | 37983.9803 | 0.8x10^-11 |
| 9 | 1000000 | 10000000 | 1x10^-15 |

## Plots for Normal Gradient Descent:

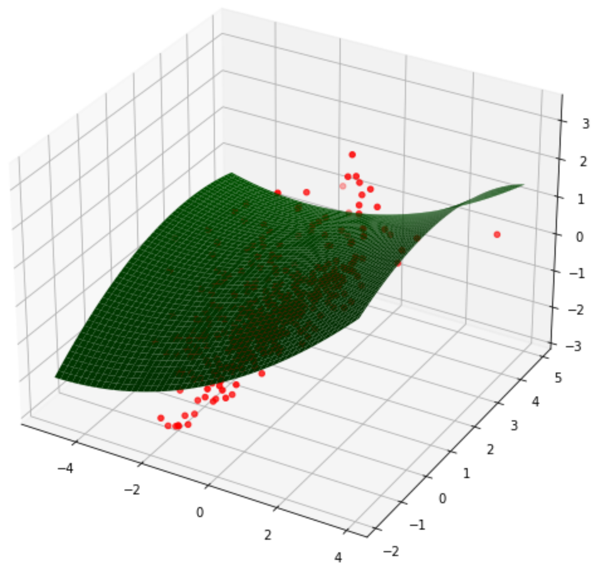
Degree = 0



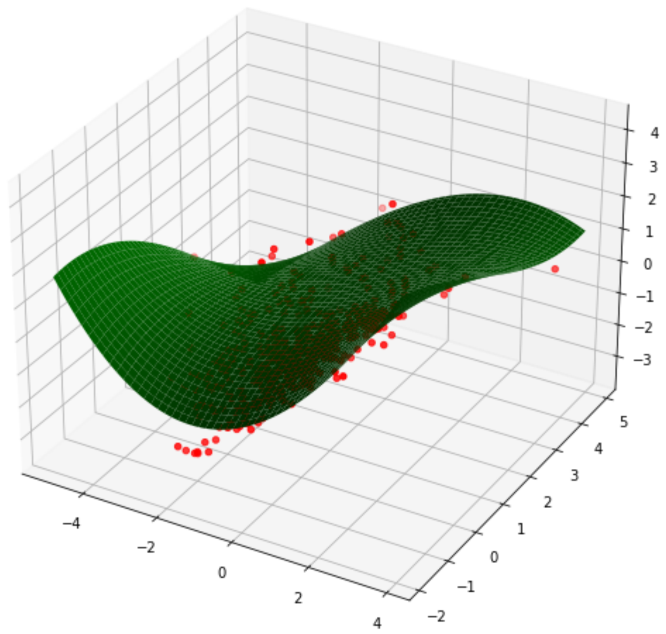
Degree = 1



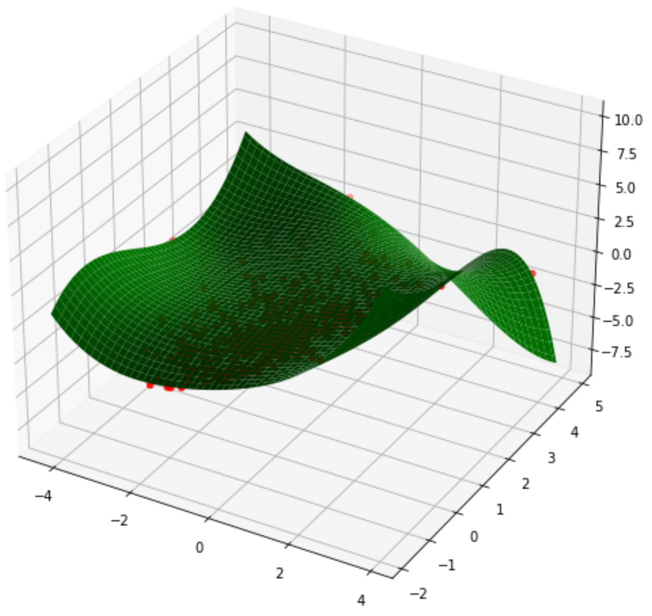
## Degree = 2



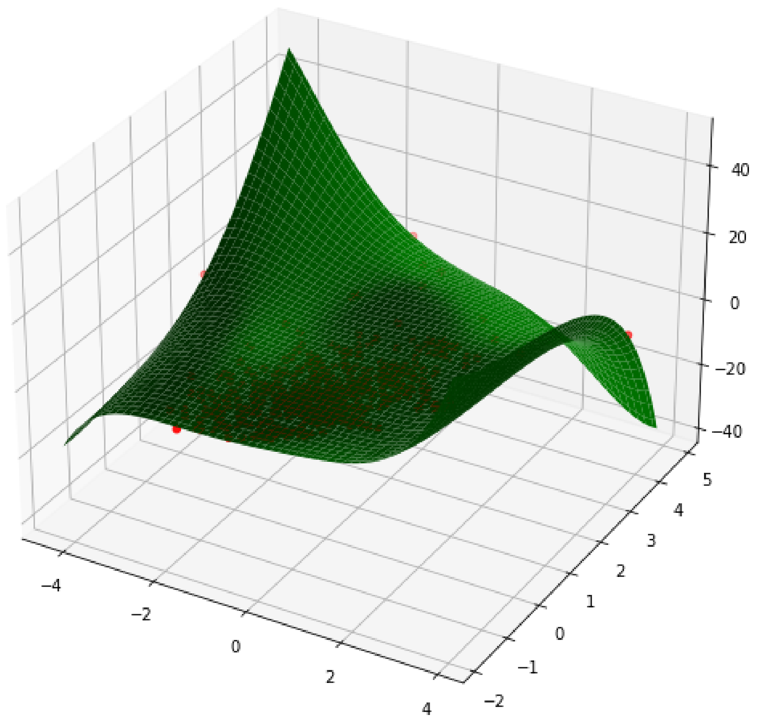
## Degree = 3



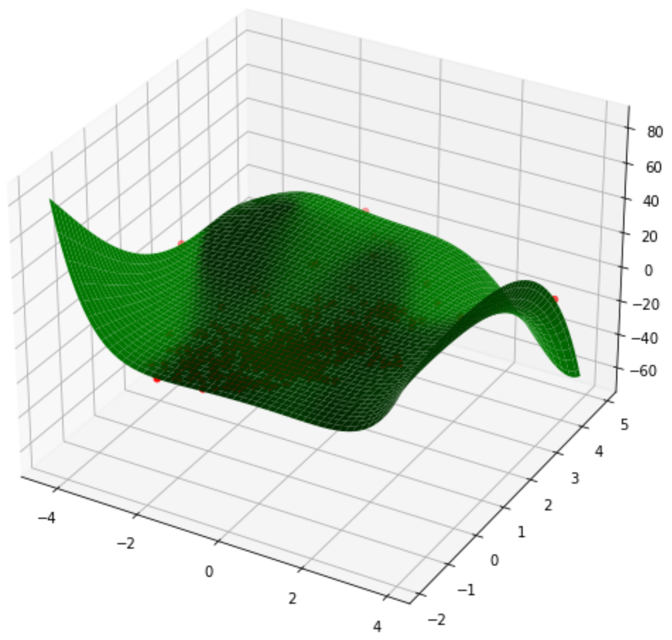
## Degree = 4



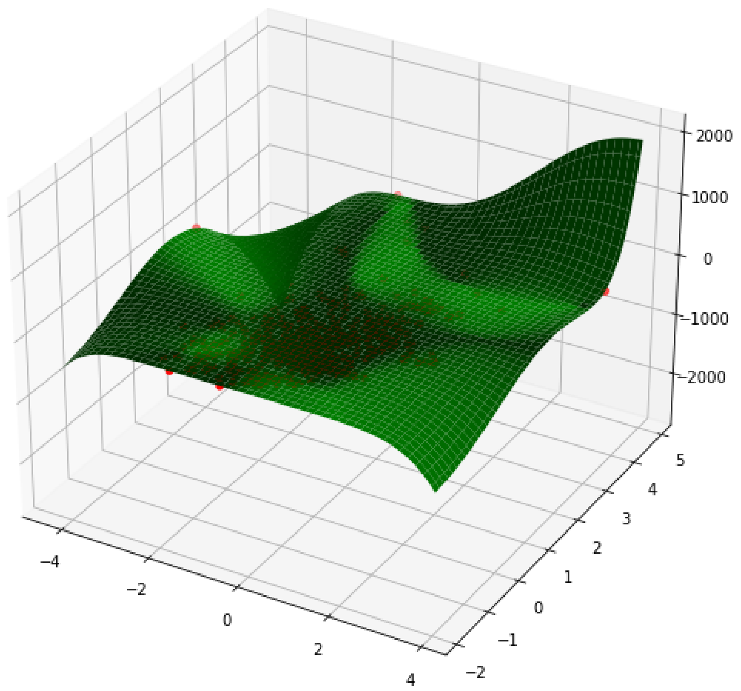
## Degree = 5



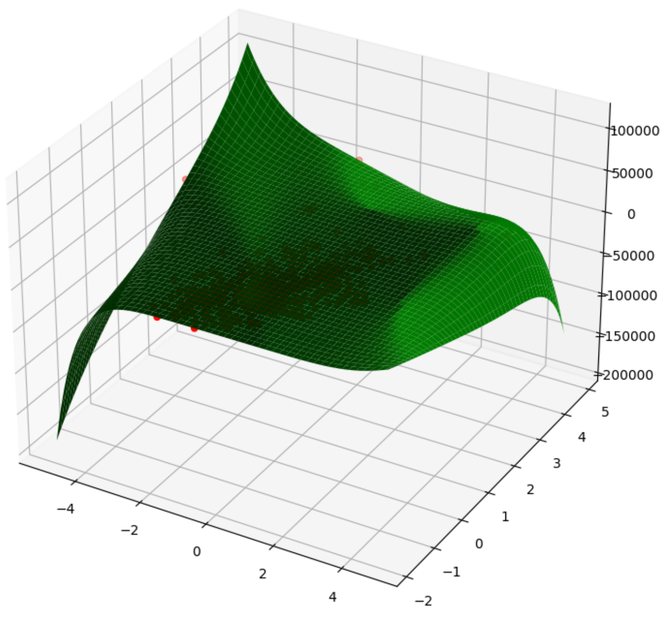
## Degree = 6



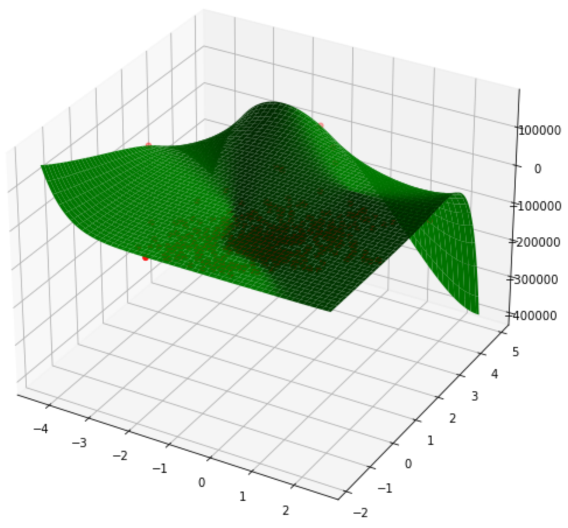
## Degree = 7



Degree = 8



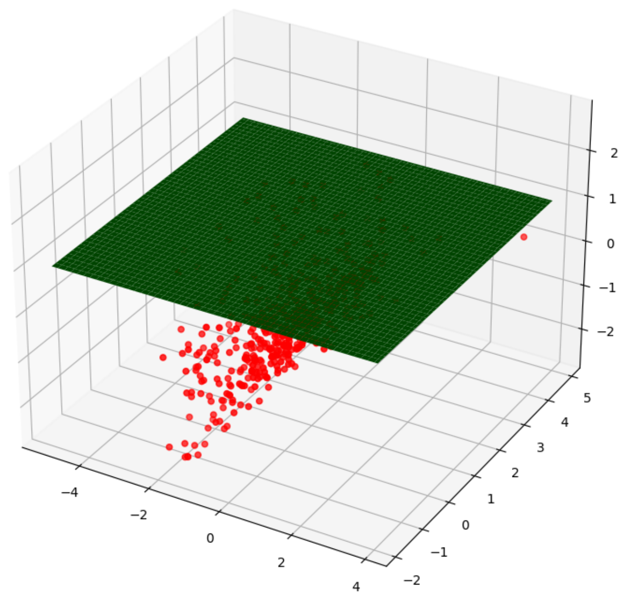
Degree = 9



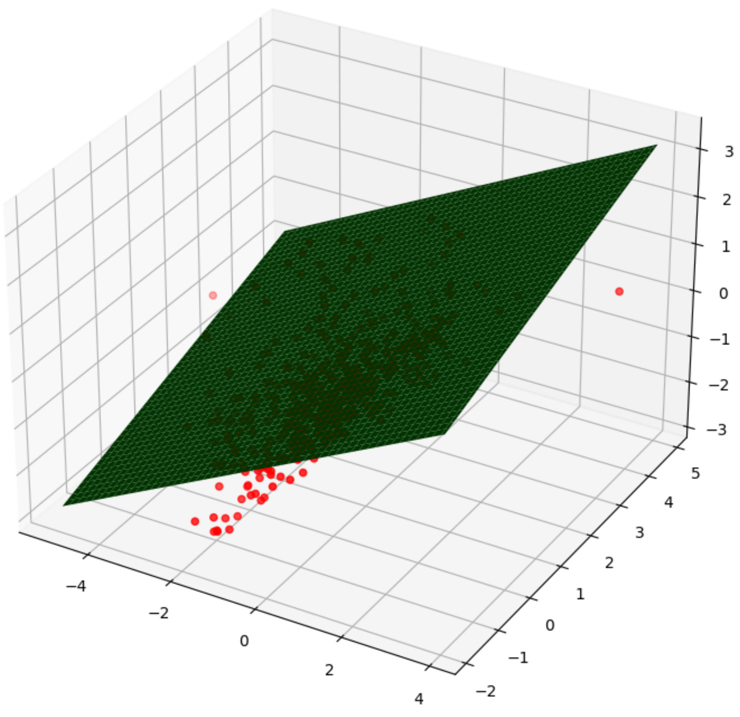
| Stochastic Gradient Descent | | | |
|---|---|---|---|
| Degree | Minimum Training Error | Minimum Testing Error | Learning rate |
| 0 | 860.6897 | 230.3102 | 0 |
| 1 | 282.6776 | 77.7972 | 1x10^-2 |
| 2 | 292.7737 | 53.9827 | 0.8x10^-3 |
| 3 | 287.9179 | 56.2604 | 1x10^-5 |
| 4 | 286.9969 | 54.8173 | 1.5x10^-6 |
| 5 | 486.5443 | 120.4379 | 0.75x10^-7 |
| 6 | 70111.6215 | 5231.9604 | 1x10^-9 |
| 7 | 1000000 | 243684.1954 | 0.6x10^-10 |
| 8 | 2695917.5698 | 100000000 | 0.8x10^-11 |
| 9 | 230803119.7118 | 97732716.1483 | 1x10^-14 |

## <u>Plots for Stochastic Gradient Descent:</u>

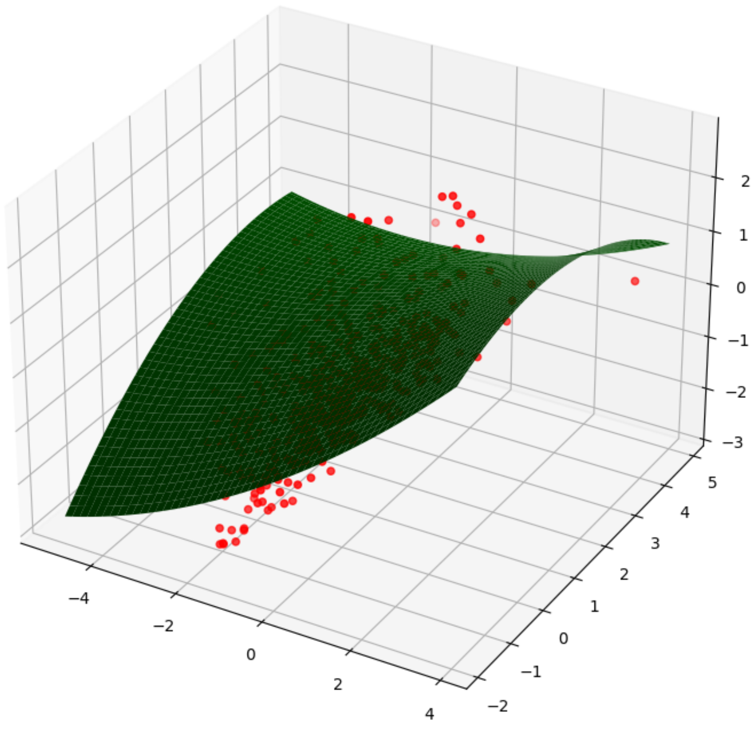
Degree = 0



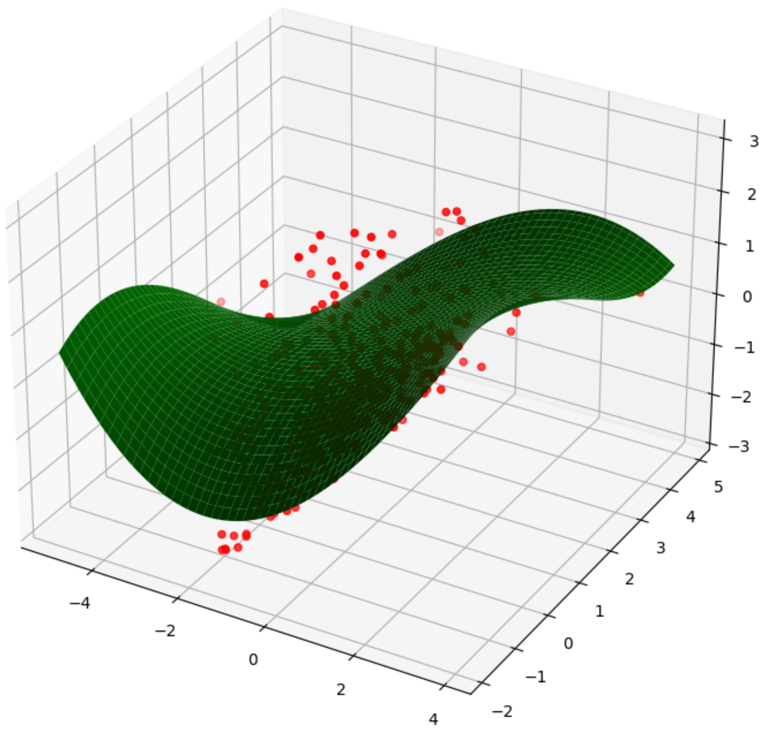
Degree = 1



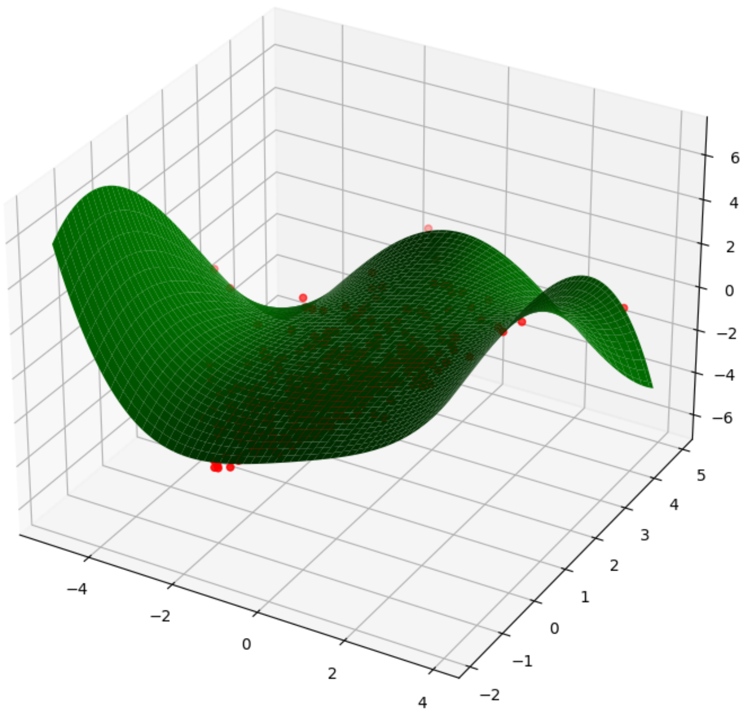
Degree = 2



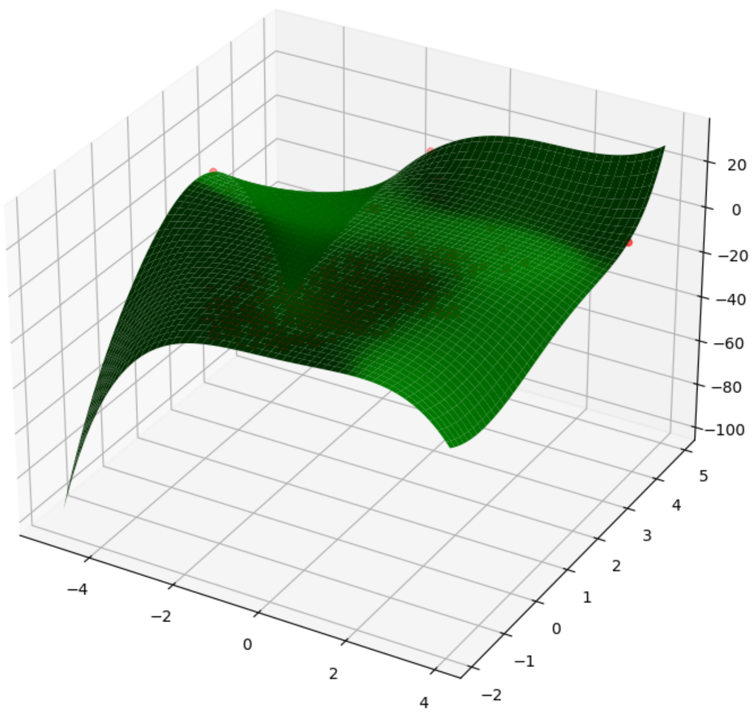
## Degree = 3



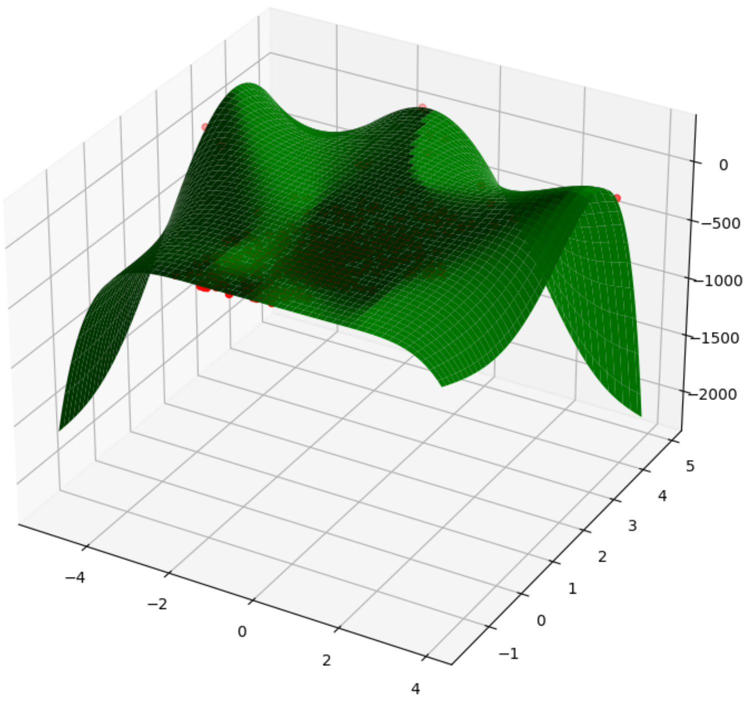
## Degree = 4



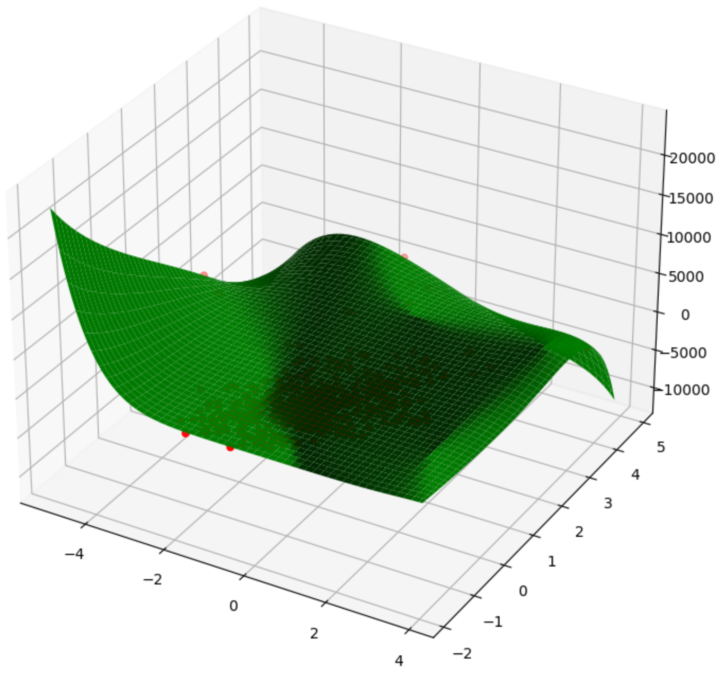
Degree = 5



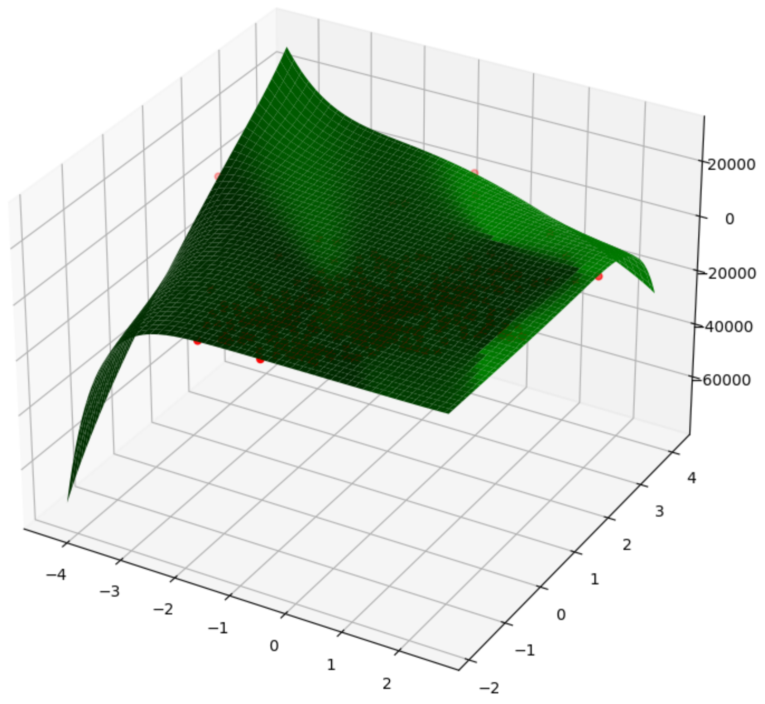
Degree = 6



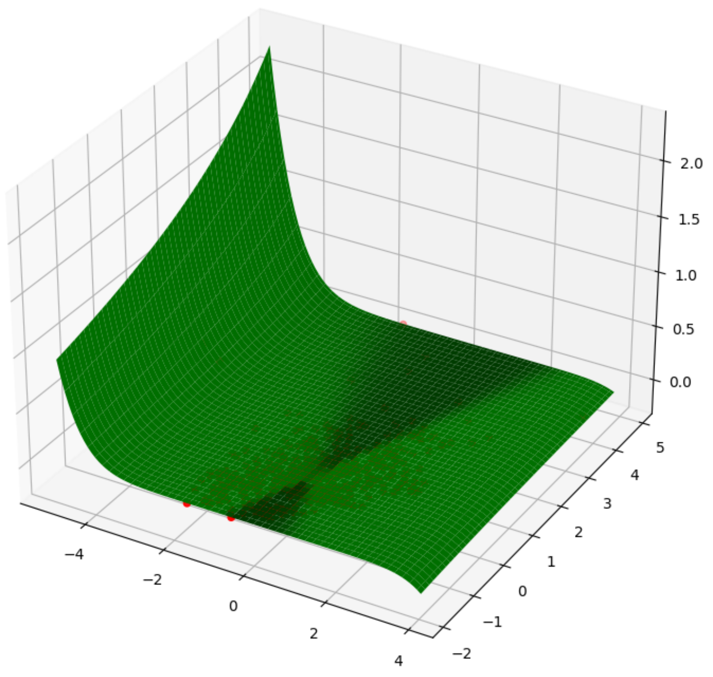
## Degree = 7



## Degree = 8



Degree = 9



## Conclusion(non regularised):

1. Keeping the number of iterations the same ,we have changed the learning rate for each of the degrees.
2. From the results tabulated above, we find the 4th degree model is the best model for Normal Gradient Descent.
3. From the results tabulated above, we find the 2nd degree model is the best model for Stochastic gradient Descent.
4. We can also observe that the testing error increases with increase in degree of the polynomial, showing us the case of overfitting nature of the polynomial.

## Polynomial Regression of 1st degree

### Normal Gradient Descent

| q value | ln(lambda) | Minimum training error | Minimum Testing error |
|---|---|---|---|
| 0.5 | -15 | 288.1890 | 72.2655 |
|  | -900 | 288.1890 | 72.2655 |
|  | -30 | 288.1890 | 72.2655 |
|  | 5 | 294.9003 | 74.8209 |
| 1 | -15 | 286.9592 | 73.5909 |
|  | -900 | 288.1890 | 72.2655 |
|  | -30 | 288.1890 | 72.2655 |
|  | 5 | 296.8376 | 75.4231 |
| 2 | -15 | 288.1890 | 72.2655 |
|  | -900 | 288.1890 | 72.2655 |
|  | -30 | 288.1890 | 72.2655 |
|  | 5 | 293.5166 | 74.4282 |
| 4 | -15 | 288.1890 | 72.2655 |
|  | -900 | 288.1890 | 72.2655 |
|  | -30 | 288.1890 | 72.2655 |
|  | 5 | 289.1478 | 72.7048 |

**Stochastic Gradient Descent**

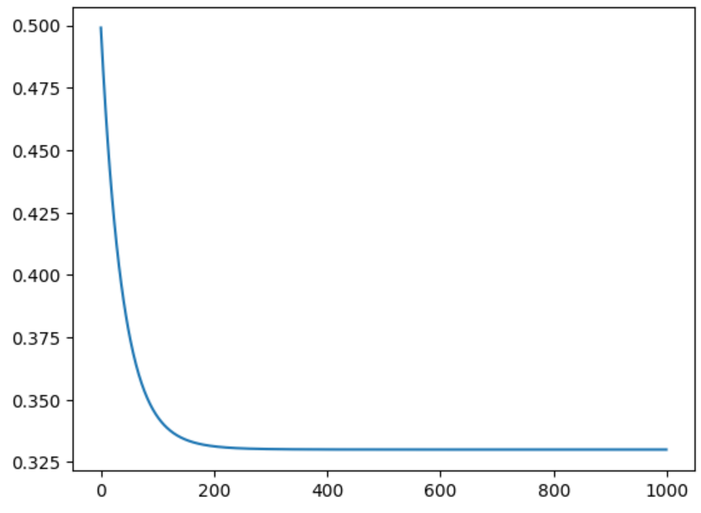
| Q value | ln(lambda) | Minimum training error | Minimum Testing error |
|---|---|---|---|
| 0.5 | -15 | 288.1912 | 72.2111 |
| | -900 | 288.1895 | 72.2330 |
| | -30 | 288.1901 | 72.2139 |
| | 5 | 332.0670 | 84.6340 |
| 1 | -15 | 288.1905 | 72.2419 |
| | -900 | 288.1935 | 72.253009 |
| | -30 | 288.19139 | 72.18585 |
| | 5 | 433.5364 | 110.6904 |
| 2 | -15 | 288.1918 | 72.2068 |
| | -900 | 288.1912 | 72.2069 |
| | -30 | 288.1915 | 72.2267 |
| | 5 | 447.2256 | 114.1601 |
| 4 | -15 | 288.1919 | 72.2654 |
| | -900 | 288.1907 | 72.2348 |
| | -30 | 288.1899 | 72.1839 |
| | 5 | 479.9969 | 122.5643 |

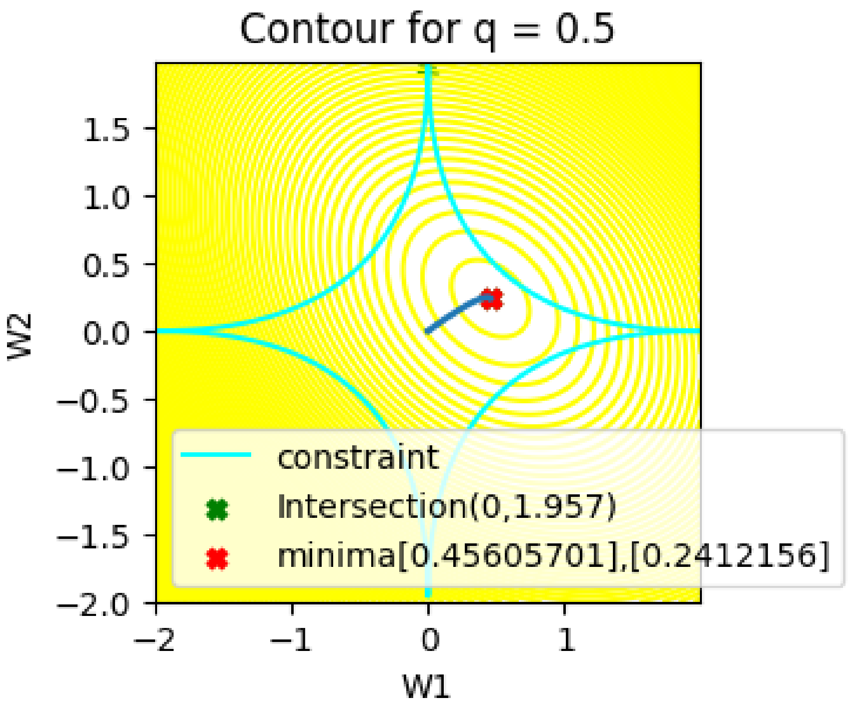
## **Conclusion:**

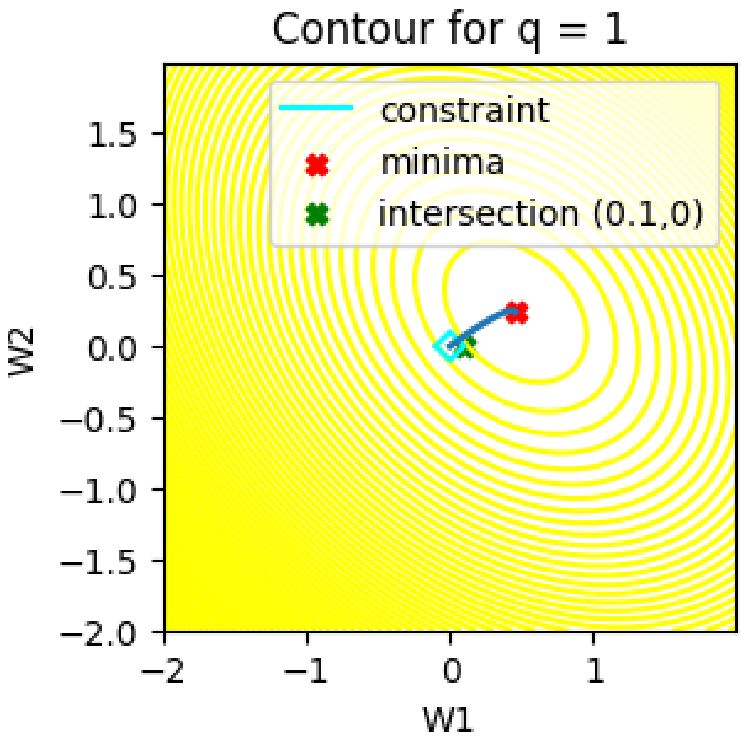
1. We have tabulated values of Minimum testing and training errors with respect to different types of regression(varying q values).
2. We've found that in case of normal gradient descent, the minimum testing error has the least value for lower values of ln(lambda).
3. Hence, we can conclude that, the models with lower values of ln(lambda) i.e, the models with lambda close to zero will be the best model.

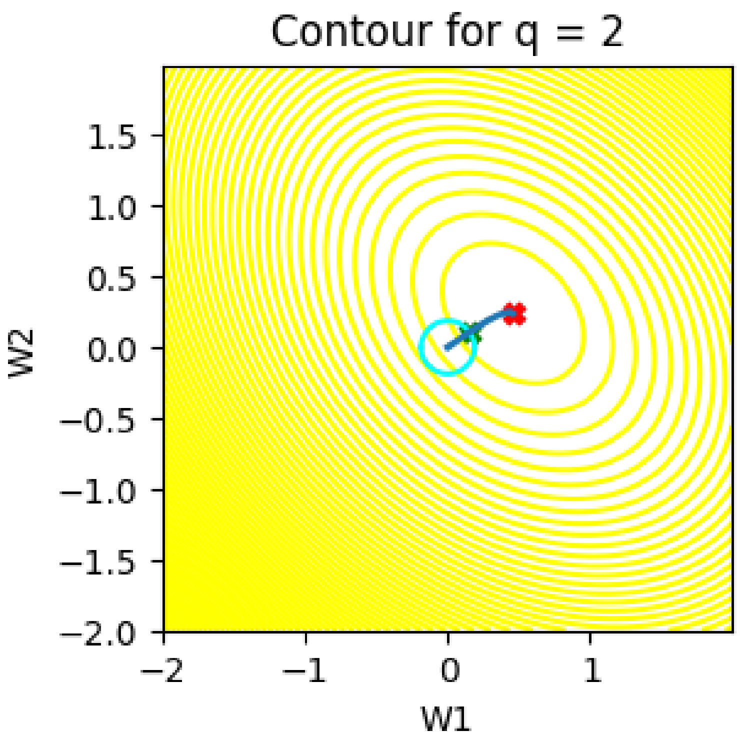
## 1C:

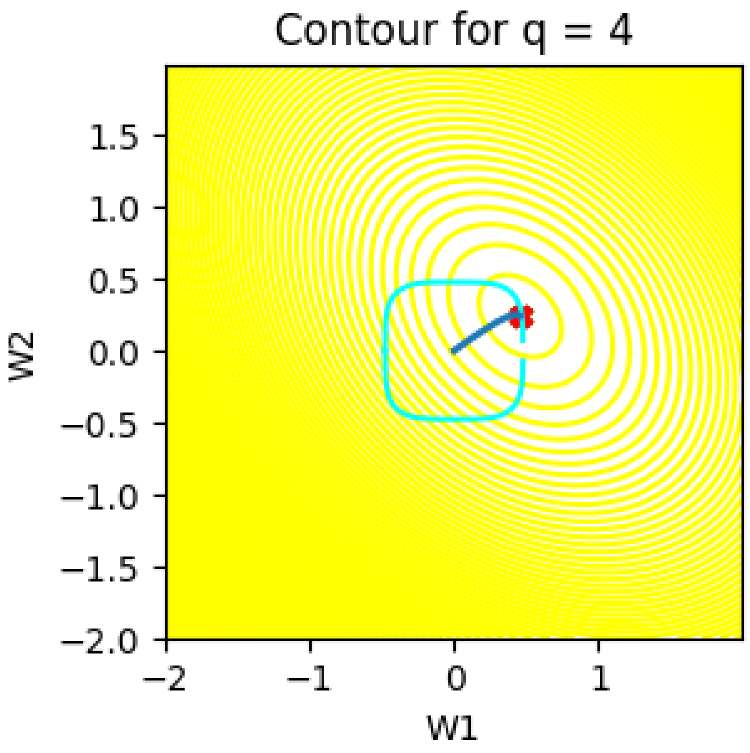
Plots for Constraint regions & error function contours




Contour for q = 0.5
constraint
Intersection(0,1.957)
minima[0.45605701],[0.2412156]
W1
W2
−2
−1
0
1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0


Contour for q = 1
constraint
minima
intersection (0.1,0)
W1
W2
−2
−1
0
1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0





| q | w1 | w2 | MSE |
|---|---|---|---|
| 0.5 | 0 | 1.957 | 25.7550 |

| | | | |
|---|---|---|---|
| 1 | 0.1 | 0 | 24.3005 |
| 2 | 0.15 | 0.108 | 24.1058 |
| 4 | 0.49 | 0.32 | 23.6247 |