CS F320 Foundation of Data Science

Assignment-1 Report

VENKAT ROHITH PAMARTI - 2020A7PS0100H

RAHUL JAUHARI – 2020A7PS0106H

SALADI GURU HARSHAVARDHAN – 2020A7PS0170H

Part-A

There's a new software in the market developed by a company and we're eager to know whether the product can attract the customers.

Let X denote a random variable indicating whether the product is liked or not.

It takes X: 1, if the product is liked.

0, if it's not liked.

S – denotes the probability of a customer liking the product.

X	1	0
P(X)	s	1-s

The probability mass function is given by: $s^x \times (1-s)^{1-x}$

(i) Likelihood of s:

The term Likelihood refers to the process of determining the best data distribution given a specific situation in the data.

Suppose there's a survey in which a out of n people liked the product and b of them didn't like: $b = n - a$.
Likelihood of the event is given the probability of liking the product is: $P(D/s)$.

We know that the data contains 'a' people linking and 'b' people disliking.
Also, each person's choice is independent i.e the outcomes are independent.
Outcomes are: $D = \{L, NL, L, L, NL, L \ldots\ldots, L\}$.

$$L(s) = P\left(\frac{\{L, NL, L, L, NL, L \ldots\ldots, L\}}{s}\right)$$

$L(s) = P(L/s)\ x\ P(NL/s)\ x\ P(L/s)\ x\ P(L/s)\ x$ .....
(as they are independent).

We know that $P(L/s)$ is s. and $P(NL/s)$ is 1-s.

$$L(s) = s^a \times (1-s)^b$$

Having got the evidence, what makes the event most likely to happen that s need to be chosen. This is called likelihood function of this event happening given s parameter.

Hence to maximise L(s) we take log on both sides and differentiate wrt to s such that L'(s)=0,
Upon solving,
We get: $a/s = b/1-s$
Solving, $s = a/(a+b)$

Hence the Maximum likelihood estimator of s: $a/(a+b)$

Out of the 50 customers surveyed, 40 of them liked the update:

Here a = 40 and b = 10.

Hence maximum likelihood estimate of s is: 0.8.

Out of the 30 customers surveyed, 13 of them liked the update:

Here a = 13, b = 17.

Hence maximum likelihood estimate of s is: 0.43333.

Out of the 100 customers surveyed, 70 of them liked the update:

Here a = 13, b = 17.

Hence maximum likelihood estimate of s is: 0.7.

(ii) Posterior Distribution of s:

Given that s follows beta distribution with the parameters α, β = (2,2).

According to Bayes' theorem we know that

$$P(s/D) = \frac{(P(D/s)\ x\ P(s))}{P(D)}$$

S is following beta distribution and it is a prior distribution because we haven't seen any data and upon intuition or domain knowledge we came up with a distribution.

In our case it's a beta distribution with the parameters: $\alpha, \beta = (2,2)$.

Hence,

$$P(s) = \frac{\Gamma(\alpha + b)}{\Gamma(a)\ x\ \Gamma(b))} \times\ s^{\alpha-1} \times (1-s)^{\beta-1}$$

$P(D) = \int P(D,s)ds$

$P(D) = \int P(D/s)\ P(s)ds$, this will be a constant as it accounts for all values of s.

$$P(s/D) \propto P(D/s)\ P(s).$$

We earlier found out, $P(D/s) = s^a \times (1-s)^b$

Hence,

$$P(s/D) \propto s^a \times (1-s)^b$$

$$P(s/D) \propto s^a \times (1-s)^b \times \frac{\Gamma(\alpha + b)}{\Gamma(a)\ x\ \Gamma(b))} \times\ s^{\alpha-1} \times (1-s)^{\beta-1}$$

Ignoring the constants, $P(s/D) \propto s^a \times (1-s)^b \times\ s^{\alpha-1} \times (1-s)^{\beta-1}$

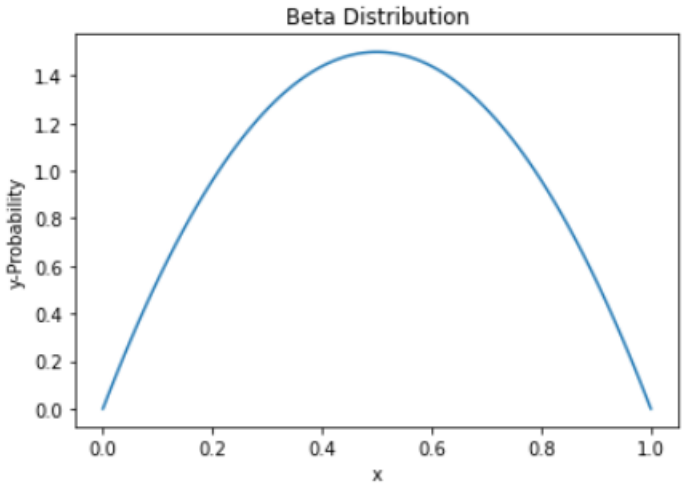
We know that $P(D) = \int P(D/s)\, P(s)$, substituting and solving, we find out that,

$$P(s/D) = \frac{\Gamma(a+\alpha+b+\beta)}{\Gamma(a+\alpha)\, x\, \Gamma(b+\beta)} \times s^{a+\alpha-1} \times (1-s)^{b+\beta-1}$$

Now comparing it with the standard of normal distribution we can say that, it's in beta distribution with the parameters: $(a+\alpha, b+\beta)$.

Hence the posterior distribution of s: $beta(s, a+\alpha, b+\beta)$.

Prior:



Survey 1:

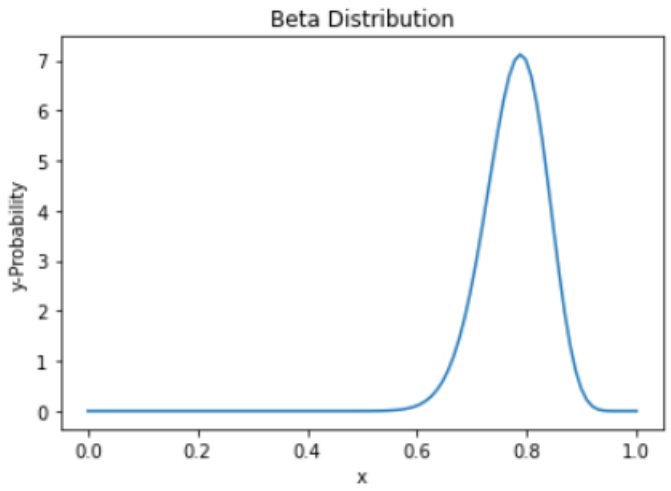
Out of the 50 customers surveyed, 40 of them liked the update:

Here a = 40 and b = 10.

We know that initial parameters (prior distribution) are: 2,2

Hence the posterior distribution of s: $beta\ (s, 42{,}12)$.

Posterior:



Survey 2:

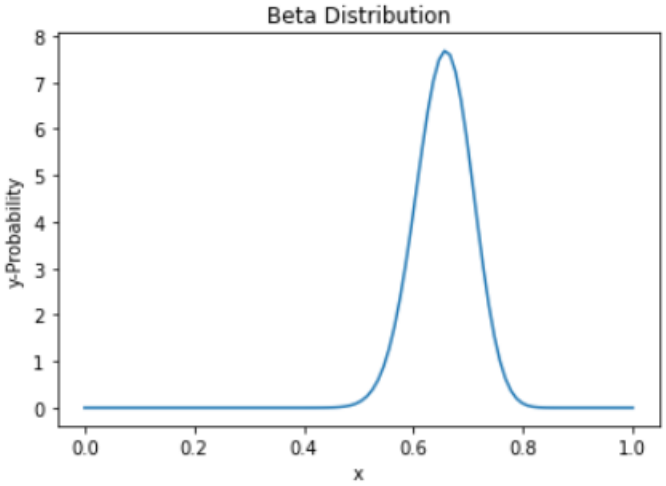
Out of the 30 customers surveyed, 13 of them liked the update:

Here a = 13, b = 17.

And the prior distribution parameters are: 42,12. (The posterior distribution for the previous case acts as prior here).

Hence the posterior distribution of s: $beta\ (s, 55{,}29)$.

Posterior:



Survey 3:

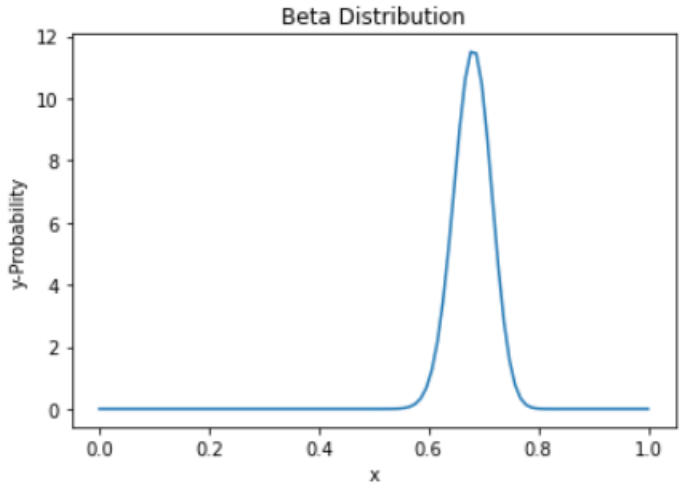
Out of the 100 customers surveyed, 70 of them liked the update:

Here a = 70, b = 30.

And the prior distribution parameters are: 55,29. (The posterior distribution for the previous case acts as prior here).

Hence the posterior distribution of s: $beta\ (s, 125{,}59)$.

Posterior:



Part B

(i) Brief Description of our model, algorithms, and how we implemented regularization:

Algorithm:

First we split the data in the ratio of 80:20 as training data and testing data respectively.

⇒ Without regularization, the error function is as below:

$$E(w_0, w_1, w_2) = (1/2N) * \sum_{n=1}^{N} (y_n - (w_0 + w_1x_{1n} + w_2x_{2n}))^2$$

where N denotes the no. of training data examples and w0,w1.. Are the weights or theta in our code.

Error must be minimum for our model to be the best possible model, that is, the error function must be convex. So for that we need to find our derivative of Error function.
There are 2 method to reach the minimum of the convex function

- Gradient descent
- Stochastic Gradient Descent method

In the **Gradient Descent method,** we will reach the values of parameters w0, w1, and w2 which minimizes the Error using E(w(k+1)) < E(w(k))
where

$$\text{Here } \partial E/\partial w_0 = (1/2N) * \sum_{n=1}^{N} 2 * (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-1)$$

$$\partial E/\partial w_1 = (1/2N) * \sum_{n=1}^{N} 2 * (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-x_{1n})$$

$$\partial E/\partial w_2 = (1/2N) * \sum_{n=1}^{N} 2 * (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-x_{2n})$$

This thing is delta in out code

```
def gradient_descent(X, y, theta, learning_rate, iterations):
    cost_history = np.zeros(iterations)
    m=len(X)
    for i in range(iterations):

        delta = np.dot(X.T, (np.dot(X, theta) - y))
        theta = theta - (learning_rate * delta) / m
        cost_history[i] = cost1(X, y, theta)

    return theta, cost_history
```

In **Stochastic Gradient Descent method**, we will reach the values of parameters w0, w1, and w2 which minimizes the Error using E(w(k+1)) < E(w(k)).
This method is exactly the same as the Gradient descent method except that we will choose one random training example (x1t, x2t, yt) among the 'N' given training examples to build the model. The equations used in this method slightly differ from the Gradient descent method as depicted below:

$$E(w_0, w_1, w_2) = (1/2) * (y_n - (w_0 + w_1x_{1n} + w_2x_{2n}))^2$$

$$w^{(k+1)} = w^{(k)} - \eta * (\nabla E)_{w = w^{(k)}}$$

$$\partial E/\partial w_0 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-1)$$
$$\partial E/\partial w_1 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-x_{1t})$$
$$\partial E/\partial w_2 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-x_{2t})$$

The code for it is as follows:

```
def sgd(data, theta, alpha, num_iters):
    m = len(data)
    X = data[:,0:-1]
    y = data[:,-1].reshape(-1, 1)
    error_his = np.zeros(num_iters)

    for iter in range(num_iters):
        idx = np.random.randint(0, m, 1)[0]
        delta = (X[idx].reshape(-1, 1)) * (np.dot(X[idx].reshape(1, -1), theta)[0,0] - y[idx][0])
        theta = theta - (alpha/m)*delta
        error_his[iter] = cost(data, theta)

    return theta,error_his
```

Regularization: In general, when the degree of the polynomial increases the weights the coefficients become more and more flexible and increase in order to fit the data.
Hence we need to limit the coefficients and subject them to condition.

Now our problem boils down to minimizing the below expression.

With regularization, the error function is as below:

$$E(w_0, w_1, w_2) = (1/2N) * \sum_{n=1}^{N} (y_n - (w_0 + w_1 x_{1n} + w_2 x_{2n}))^2 + (\lambda/2) * (|w_1|^q + |w_2|^q)$$

where λ denotes the balancing factor, which controls the growth of weigths.

We can apply both gradient descent and stochastic gradient descent method to find weight values.
In our code we used gradient descent method along with regularization.

Code is as follows:

```
# regularisation
def gradient_descent_with_regularisation(data, theta, learning_rate, iterations, lambda_, q):
    X = data[:,0:-1]
    y = data[:,-1].reshape(-1, 1)
    cost_history = np.zeros(iterations)

    for i in range(iterations):

        delta = np.dot(X.T, (np.dot(X, theta) - y))
        if q>1:
            theta = theta - ((learning_rate/len(X))*delta+(1/2)*lambda_*q*np.power(theta,q-1))
        elif q==1:
            a=0
            for j in range(len(theta)):
                if theta[j]>0:
                    a+=1
                elif theta[j]<0:
                    a-=1
                else:
                    a+=0
            theta = theta - ((learning_rate/len(X))*delta+(1/2)*lambda_*q*a)
        else :
            a=0
            for j in range(len(theta)):
                if theta[j]>0 :
                    a+=lambda_*np.power(np.absolute(theta),-0.5)/4
                elif theta[j]<0:
                    a-=lambda_*np.power(np.absolute(theta),-0.5)/4
                else:
                    a+=0
            theta = theta - ((learning_rate/len(X))*delta+(1/2)*lambda_*q*a)


        cost_history[i] = cost(X, y, theta, lambda_, q)
    return theta, cost_history
```

How to implement regularization:
Step:-

1. Normalized the x-data using the min-max normalization technique

```
def norm(data):
    normalize = (data - data.min()) /(data.max()-data.min())
    return normalize
```

2. Split the data into training and testing set ratio 80:20
3. Then we create a feature matrix which stores the value of all the x terms possible from degree 0 to degree 9 (like x1, x2, x1*x2, x12, x22 ....., x19, x29)with respect to each data point in training data and testing data, respectively.
4. Then for a particular number of iteration (we used 10^5) and storing the derivative of error ∂E/∂w0 this is cost in our code.

5. Our error function will be

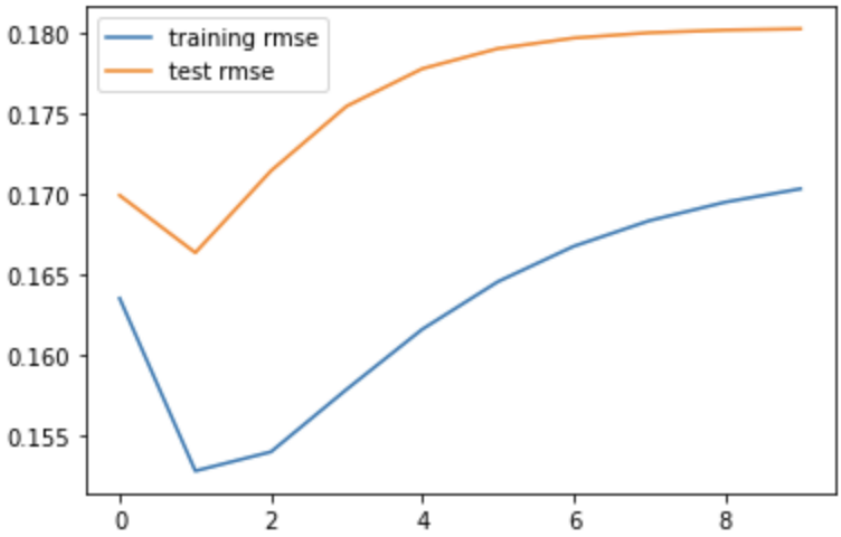
$$E(w_0, w_1, w_2) = (1/2N) * \sum_{n=1}^{N} (y_n - (w_0 + w_1x_{1n} + w_2x_{2n}))^2 + (\lambda/2) * (|w_1|^q + |w_2|^q)$$

6. In error function lambda signifies how much weigthage we have to give to growth of weights(theta)
7. One more thing is crucial to update weights that is learning rate. We set learning rate to 0.001
8. Then after this we will loop throught the number of iterations and update the weight using the formula w_new = w_old - (learningRate) * ∂E/∂w0 +lambda*q*sig(w)^(q-1). sig() is a sigmoid function which is 0 if x is 0, 1 if x>0 and -1 if x<0.

ii) Tabulate the training and testing errors obtained using polynomial regression models of various degrees and your observations on overfitting.

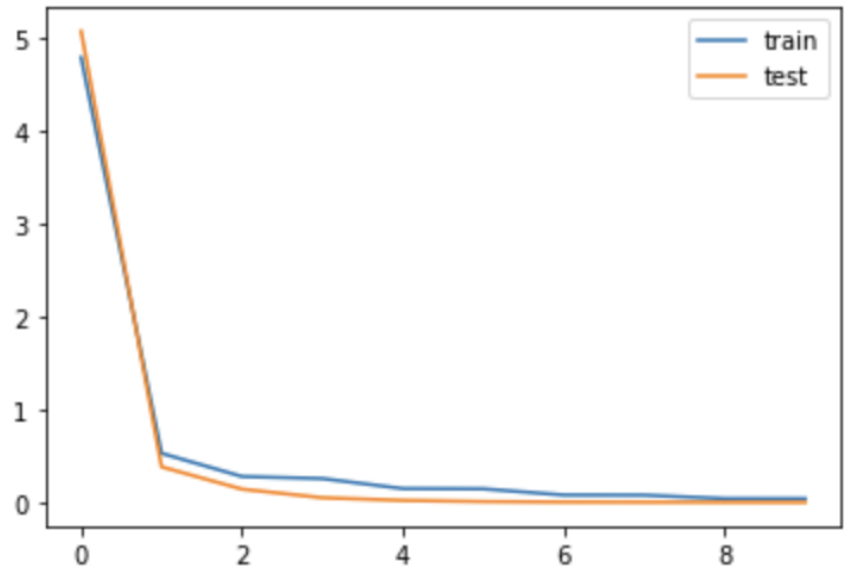
The below values are for gradient descent, learning rate =0.001

Degree	Minimum training error	Min Testing error
0	0.199158	0.1685573631874136
1	0.183995	0.15670312286272514
2	0.188281	0.15936748416002244
3	0.193688	0.16320133850947527
4	0.197828	0.16581466651092455
5	0.200729	0.16737130979016152
6	0.202721	0.16826582897293313
7	0.204088	0.16877732082957858
8	0.205035	0.16907208954558922
9	0.205699	0.1692443710536634



The below values are for stochastic gradient descent, learning rate =0.001

Degree	Minimum training error	Min Testing error
0	4.789564877980839	5.071106309623556
1	0.5289005099289475	0.3836026768021376
2	0.27950136491901933	0.14330941983510884
3	0.2569972621667106	0.05126755809353156
4	0.14930667323809665	0.022155104352153045
5	0.14643334345739725	0.010016617918160935
6	0.0803739264386061	0.004714146747560138
7	0.07898303994576088	0.0032373568999290907
8	0.041698332919626616	0.002384067774178835
9	0.040976321960352	0.0017425470702687733



Regularization

q= 0.5

ln(Lambda)	Minimum Training Error	Minimum testing error
0	4.839046724003086	5.147799098864665
-20	1.6438291960486355	1.74544984371007
-15	1.643829196052063	1.7454498483898873
-10	1.6438291997968117	1.7454546407500189
-4	1.6478855074977974	1.769001616937765
1	5.165918917067095	5.475261535594858

q= 1

ln(Lambda)	Minimum Training Error	Minimum testing error
0	4.870133069363808	4.836114898465209
-20	1.6594308444525911	1.7455321684842697

-15	1.6595027995833866	1.748146821488081
-10	1.729830389423765	1.8918380349756319
-4	4.9285355079609	5.167599327256884
1	5.447091347531007	4.472689646757103

q= 2

ln(Lambda)	Minimum Training Error	Minimum testing error
0	4.948546419365348	5.187328328403459
-20	1.659437230631723	1.7462105528294574
-15	1.6651878857188154	1.7740026002939495
-10	2.8407447710438456	3.0941854950945284
-4	4.883757290574205	5.123397592503173
1	5.8025263102819	6.758435683467456

q=4

ln(Lambda)	Minimum Training Error	Minimum testing error
0	4.8293536935181365	5.068506052113175
-20	1.6684384714436604	1.7815774543800225
-15	2.346920418188723	2.5755810188499937
-10	3.834683143537135	4.076591928610083
-4	4.64683705613598	4.886606562837265
1	4.854622369611236	5.093687611354622

Observations on Overfitting:

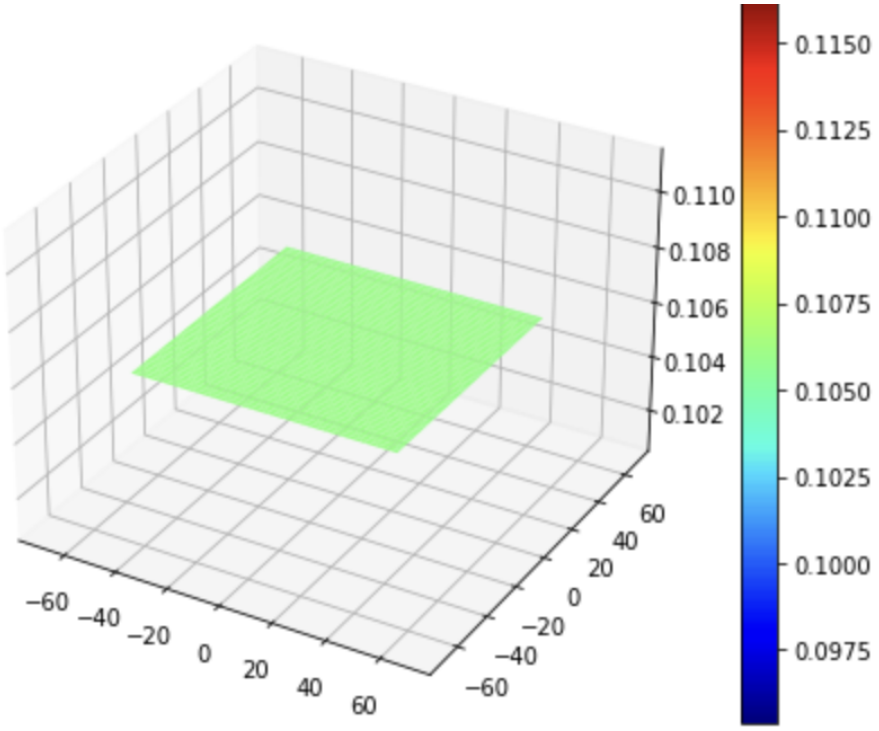
In general, a regression model is said to be overfitting the data if it performs well on the training data but does not perform well on testing data.

For the gradient descent approach we found out that both training and testing error initially decreased as we moved from degree 0 to 1 and for degree >1 both training and testing errors increased. There is no case of overfitting as there is no polynomial of degree>1 which fits the training set well and fails to fit testing data.

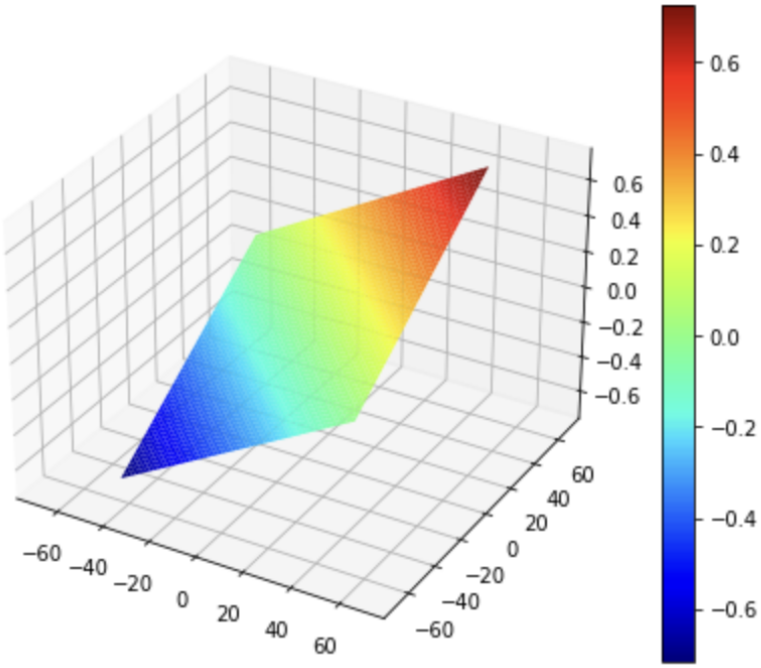
For the stochastic gradient descent method, we observed that the error is almost same from degree 3 to degree 9 which is the least error for the testing data. Among polynomial regression models of degree > 3, we will choose the polynomial regression model of degree 3 as the best model by using the principle of Ockham's Razor (We will choose the simpler model). Hence, this polynomial model of degree 3 is the best model. For a polynomial of degree>3 fits the training set better than for degree 3. So we can conclude that the polynomial with degree >3 overfits the given data.

iii) Surface plots of the predicted polynomials (Plot of $x1$, $x2$ vs y($x1$, $x2$) where y is the predicted polynomial.)

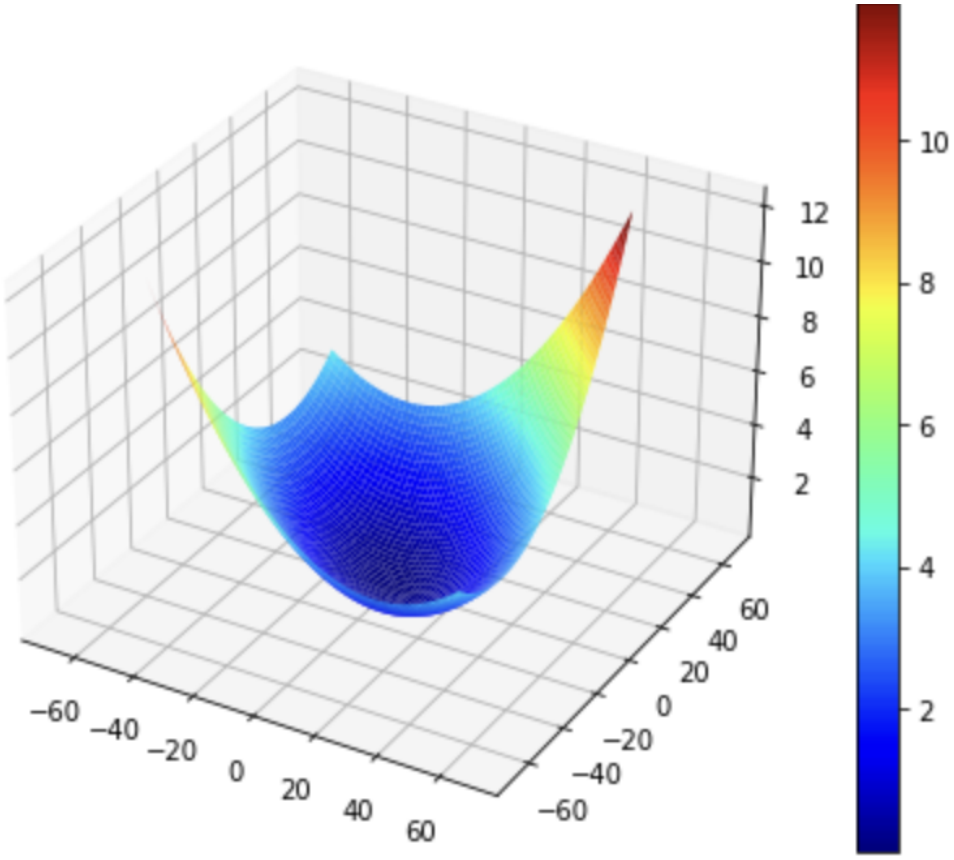
For polynomial of degree 0



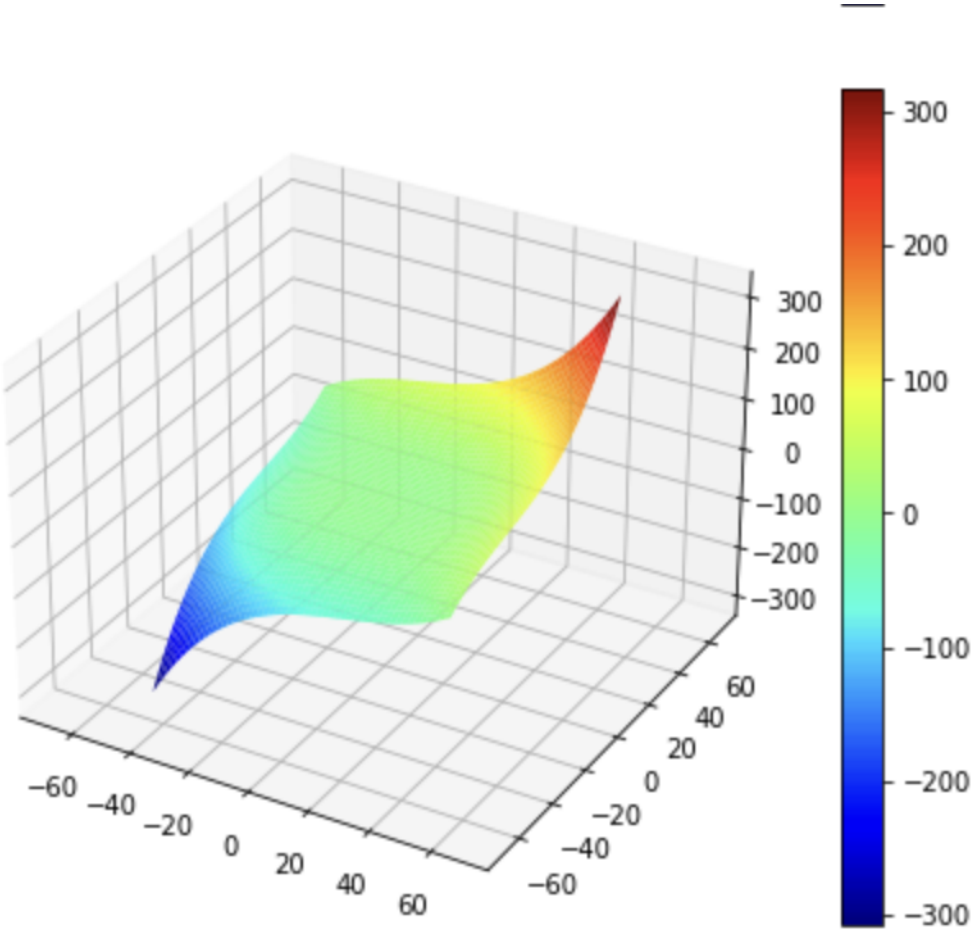
For polynomial of degree 1



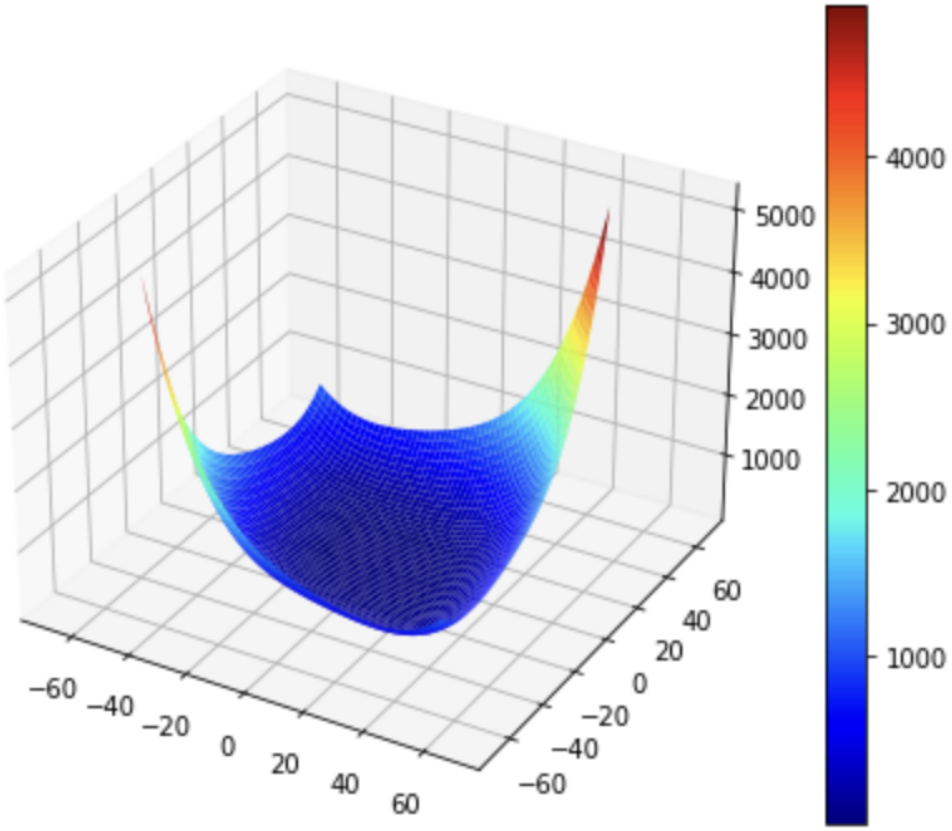
For polynomial of degree 2



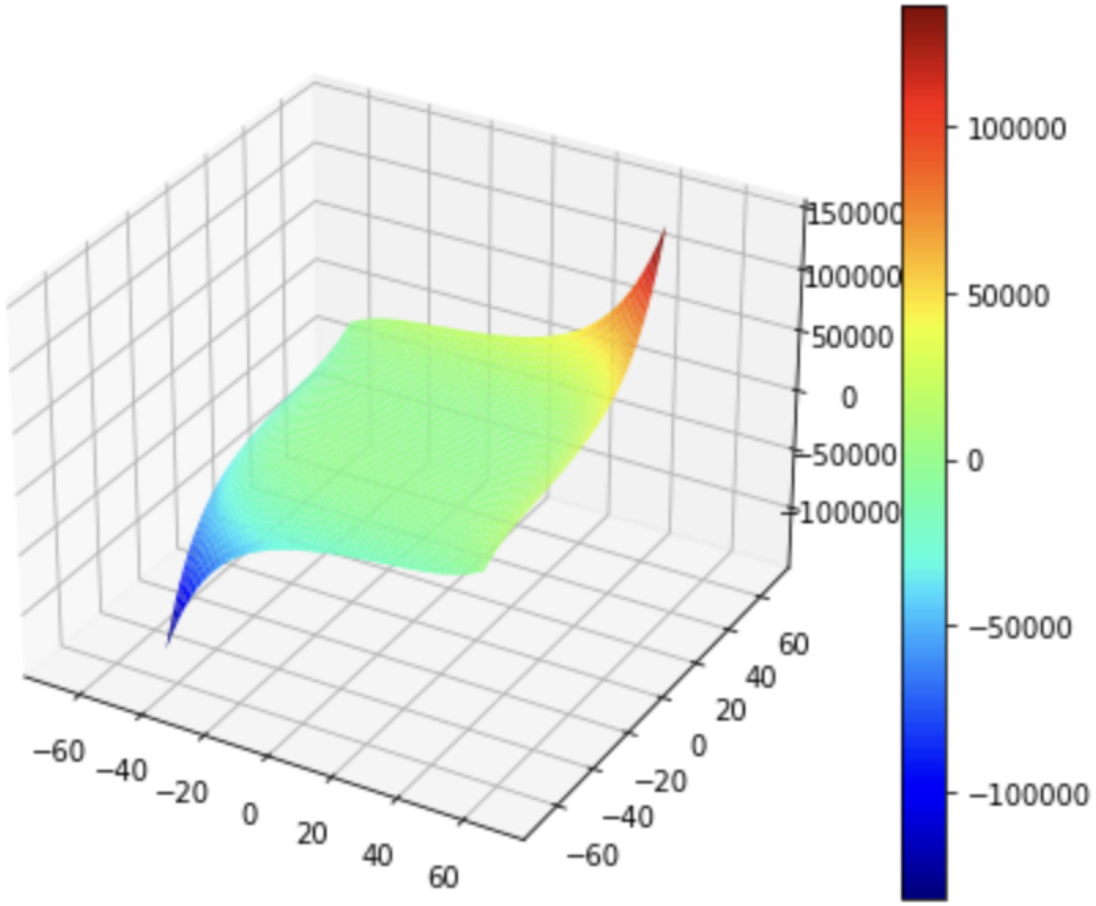
For polynomial of degree 3



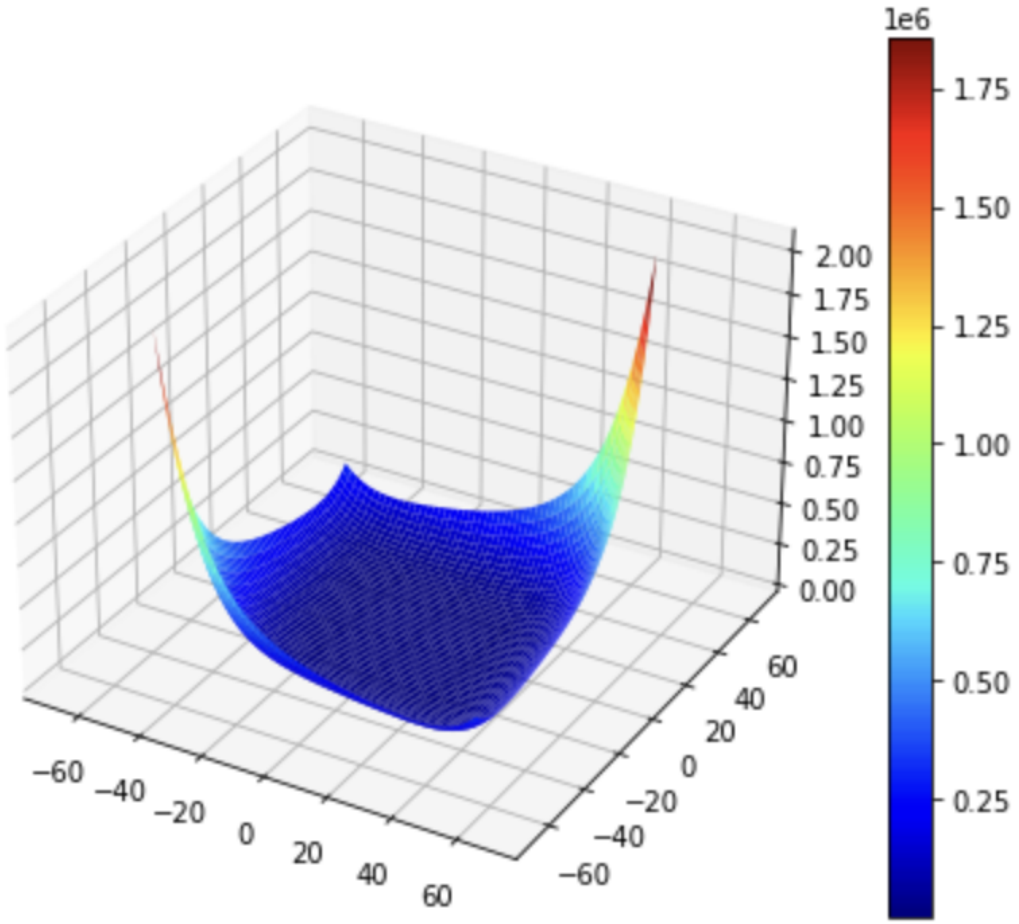
For polynomial of degree 4


5000
4000
3000
2000
1000
−60
−40
−20
0
20
40
60
4000
3000
2000
1000

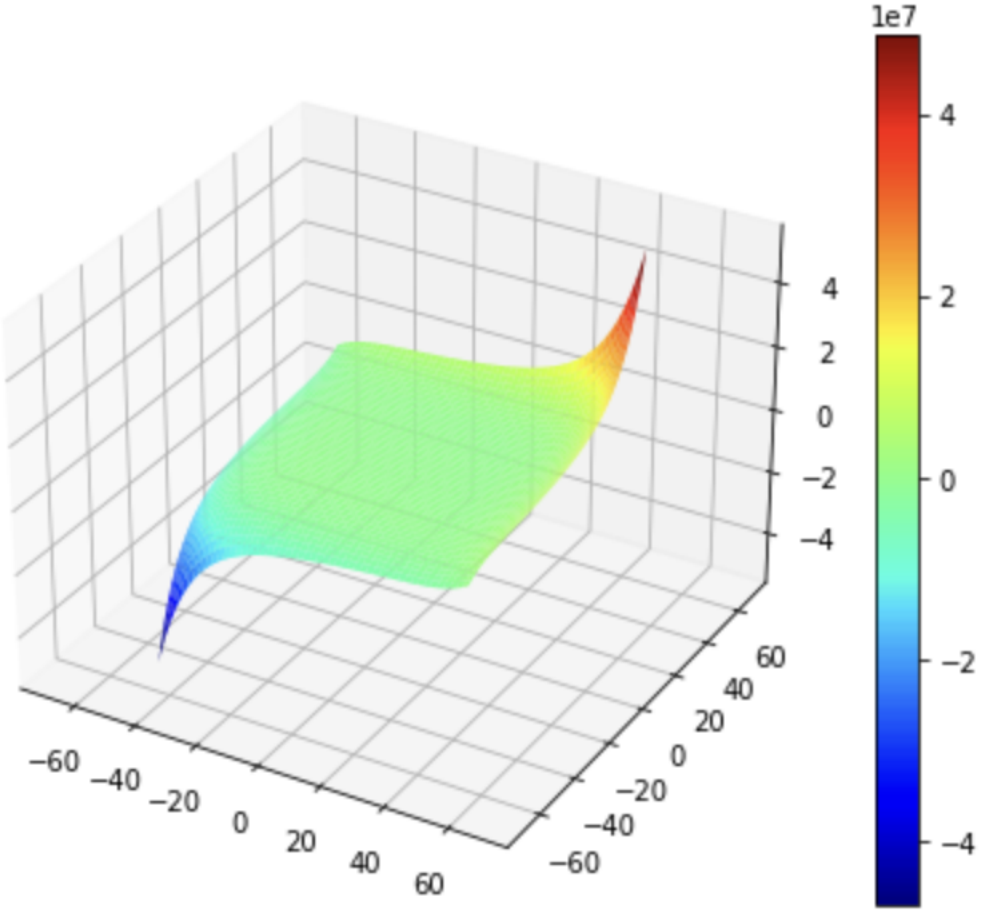
For polynomial of degree 5



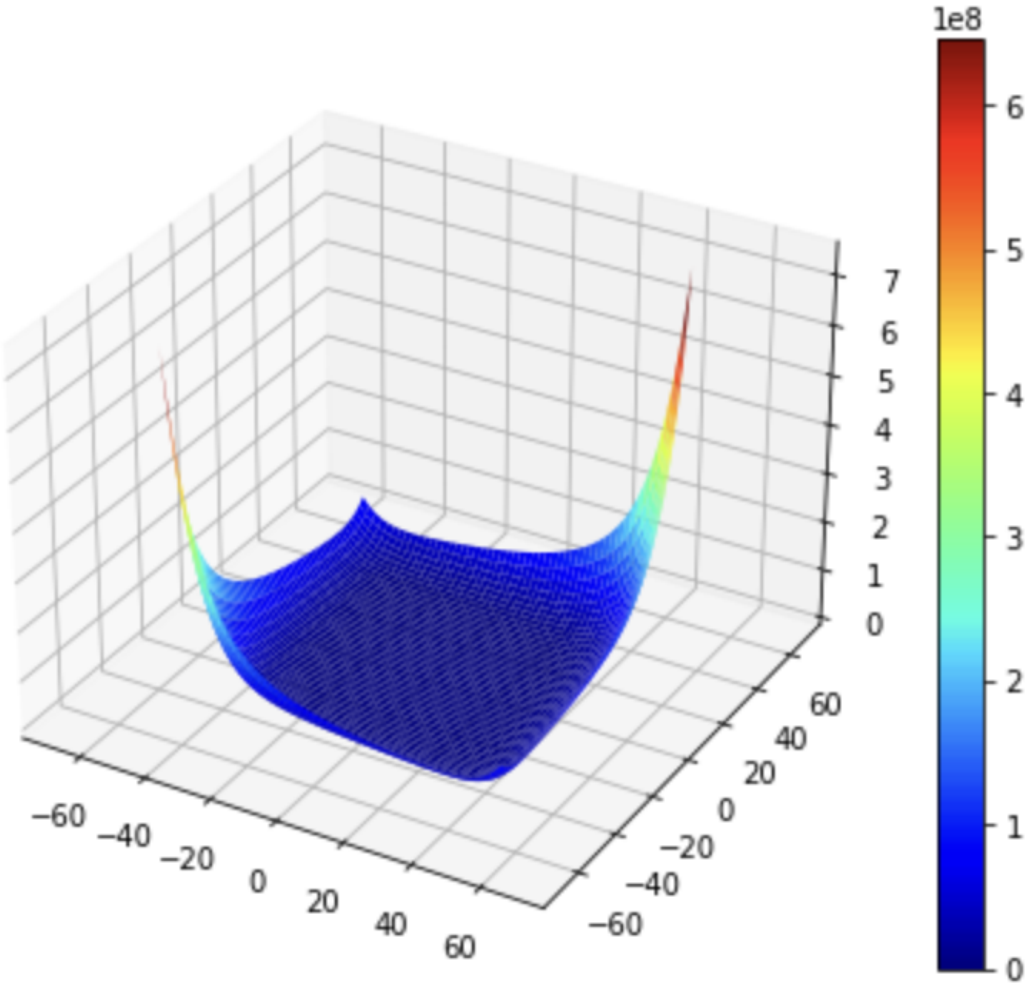
For polynomial of degree 6



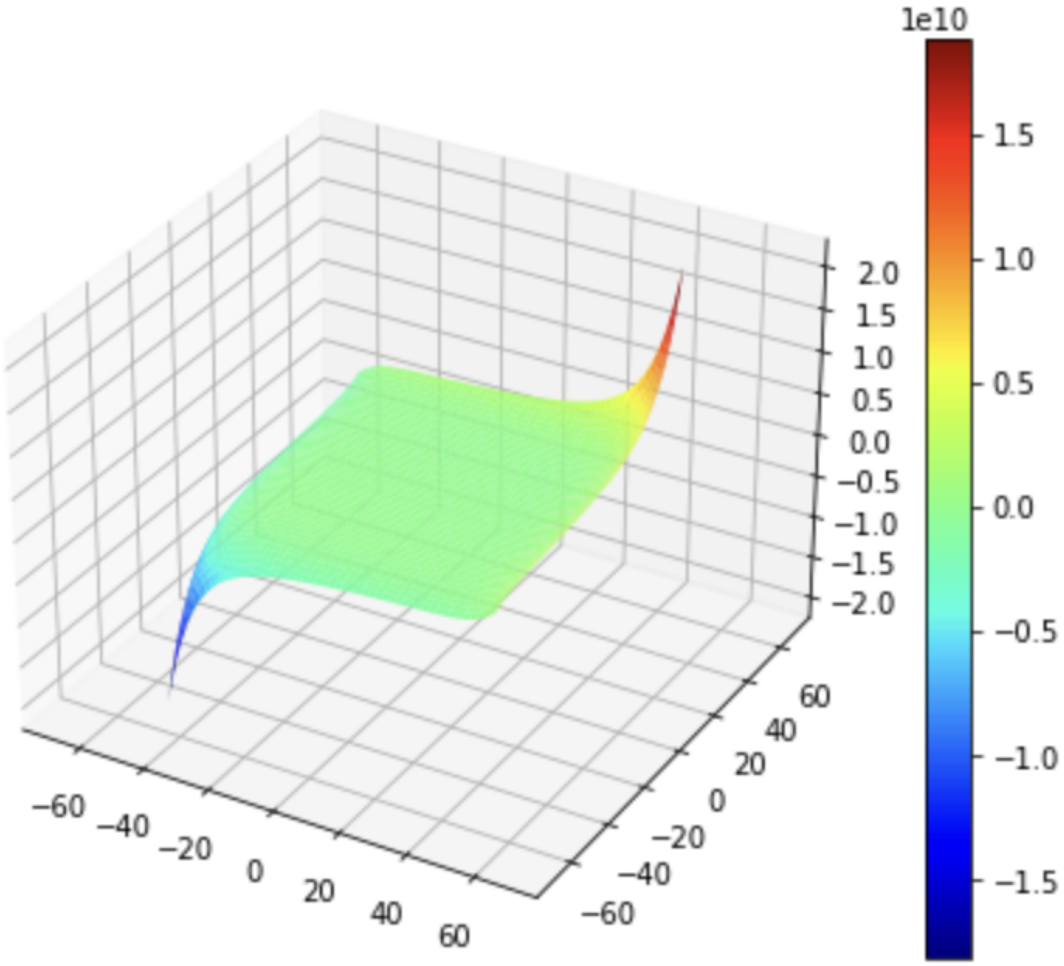
For polynomial of degree 7



For polynomial of degree 8


1e8
7
6
5
4
3
2
1
0
−60
−40
−20
0
20
40
60
6
5
4
3
2
1
0

For polynomial of degree 9



(iv) Comparative analysis study of the four optimal regularized regression models and best-fit classic polynomial regression model.

Model	q	Training Error	Testing Error
Best-fit model (unregularized degree-1 model is best from 1B (a))	-	0.183995	0.15670312286272514
Regularized	0.5	1.6438291960486355	1.7454498483898873
	1	1.6594308444525911	1.7455321684842697
	2	1.659437230631723	1.7462105528294574
	4	1.6684384714436604	1.7815774543800225

From the above table, we can see that the unregularized regression model of degree 1 fits the testing data very well as the value of testing error is the least

The degree 9 polynomial regression model underfits the data, as compared other degrees because both training and testing error are maximum.

Even after regularising the degree 1 polynomial model, it will still not be as effective as the degree 1 polynomial model that has not been regularised. Therefore, we may conclude that the best model among these five models is the best-fit unregularized regression model of degree 1.

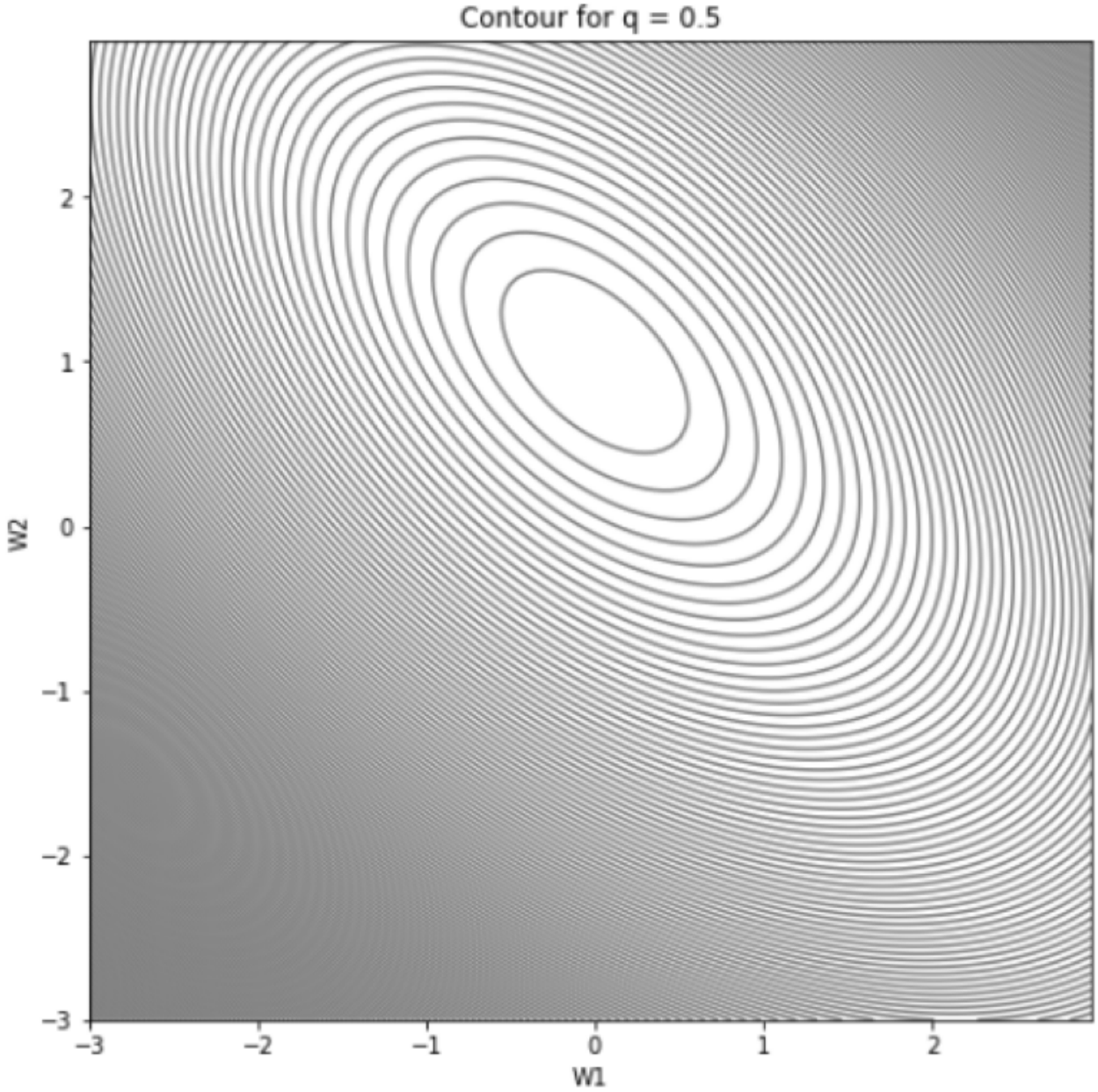
Part C

Minimize $E(w)$ **= 1/2** $\sum(yn - tn)^2$ $subject\ to$ $|w1|^q + |w2|^q \leq \eta$

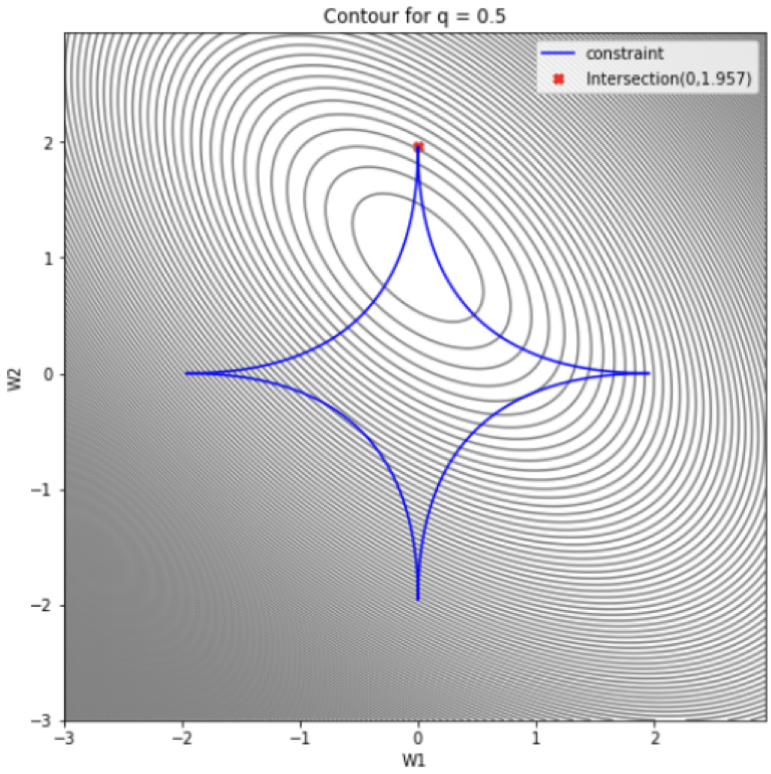
i) For For polynomial of degree 0.5 and η=1.4

$$|w1|^{0.5} + |w2|^{0.5} \leq 1.4$$

Error contour plot



Constraint regions and error function contours, showing the tangential contour and the point of intersection where the minima occurs



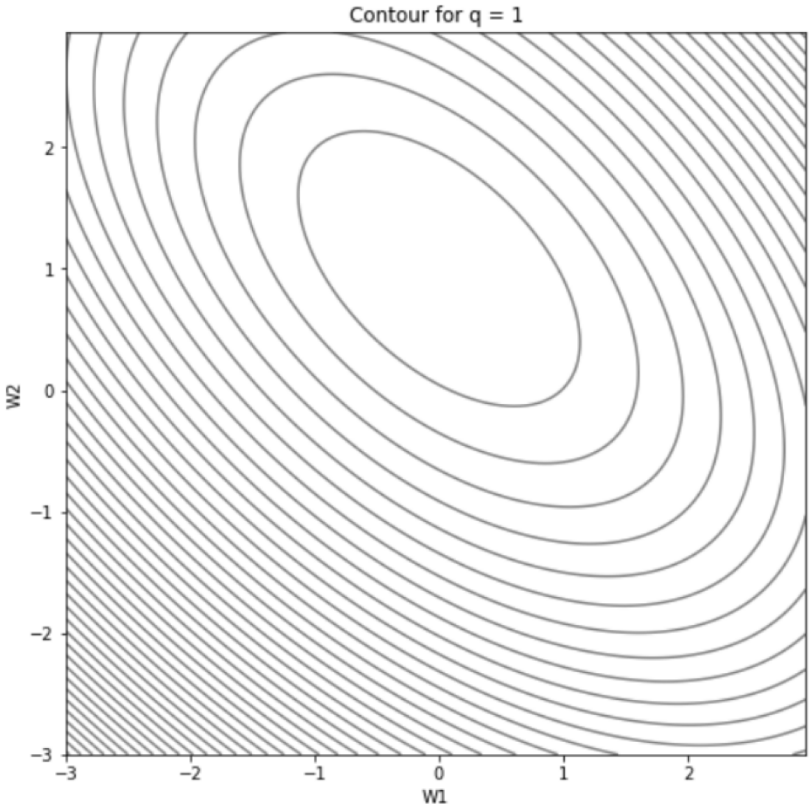
Both Constraint regions and error function contours intersects tangentially at (W1,W2) = (0,1.95)

W1 represents MLOGP and W2 represents GATSli

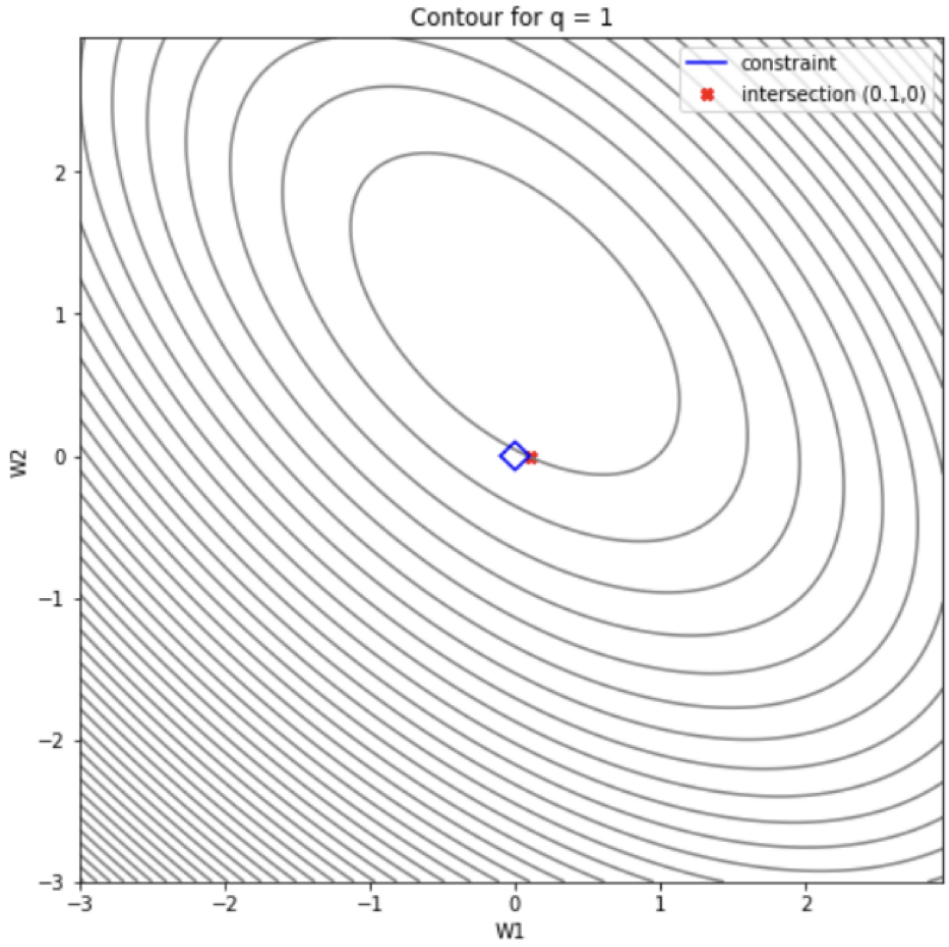
ii) For For polynomial of degree 1 and η=0.1

$$|w1| + |w2| \leq 0.1$$

Error contour plot



Constraint regions and error function contours, showing the tangential contour and the point of intersection where the minima occurs



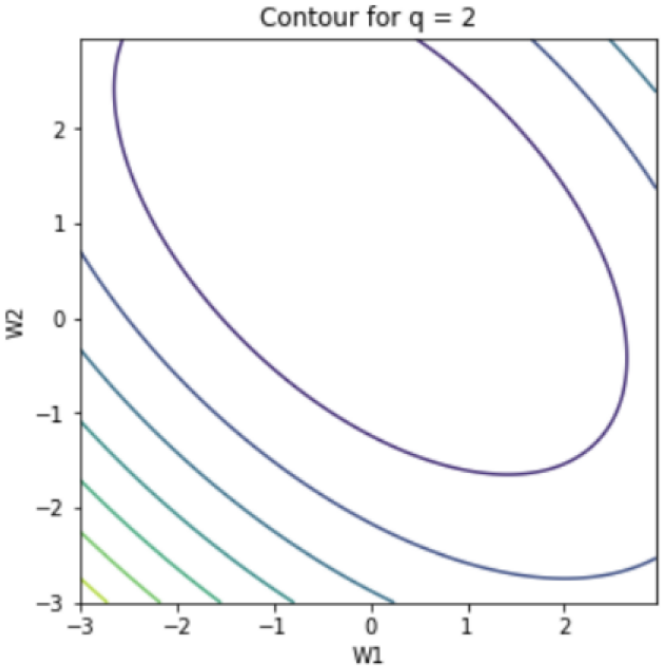
Both Constraint regions and error function contours intersects tangentially at (W1,W2) = (0.1,0)

W1 represents MLOGP and W2 represents GATSli

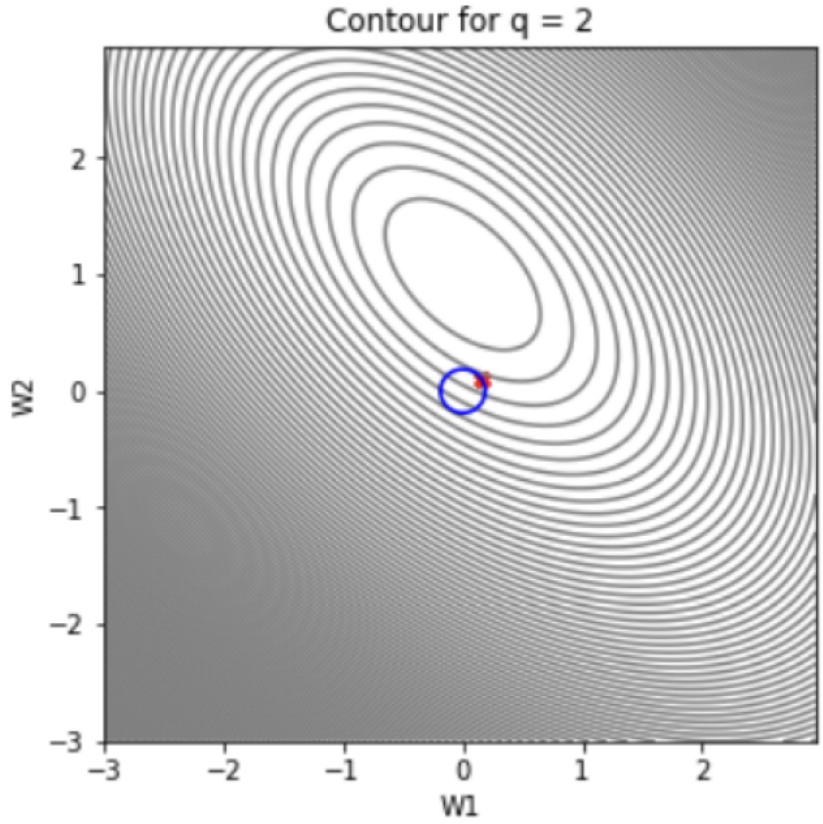
iii) For For polynomial of degree 2 and η=0.035

$$|w1|^2 + |w2|^2 \leq 0.035$$

Error contour plot



Constraint regions and error function contours, showing the tangential contour and the point of intersection where the minima occurs



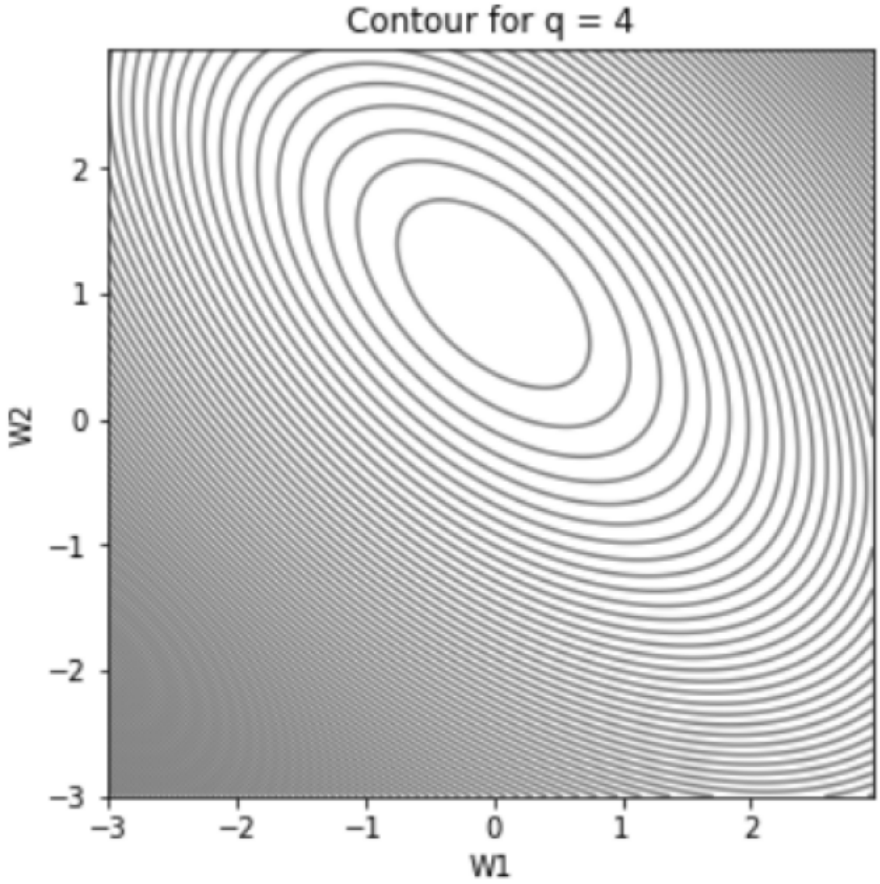
Both Constraint regions and error function contours intersects tangentially at (W1,W2) = (0.15,0.1)

W1 represents MLOGP and W2 represents GATSli

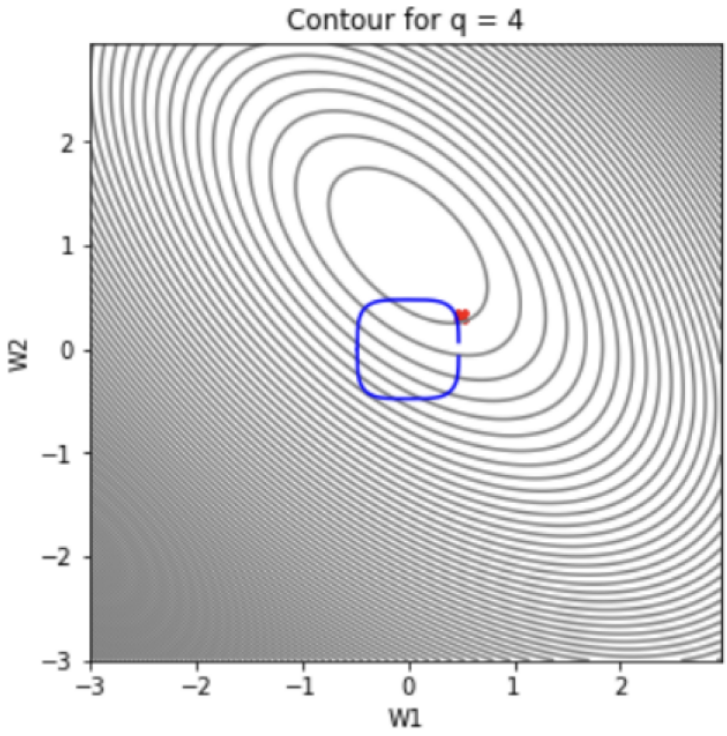
iv) For For polynomial of degree 4 and η=0.052

$$|w1|^4 + |w2|^4 \leq 0.052$$

Error contour plot



Constraint regions and error function contours, showing the tangential contour and the point of intersection where the minima occurs



Both Constraint regions and error function contours intersects tangentially at (W1,W2) = (0.5,0.3)

W1 represents MLOGP and W2 represents GATSli

Mean Squared Errors

```
def MSE(w1,w2):
    temp=np.average(((w1*temp_X[0,:]+w2*temp_X[1,:]-temp_Y))**2)
    return temp
```

i) For For polynomial of degree 0.5 and η=1.4

(w1,w2) = (0,1.95)

MSE = 3.2513107901236262

ii) For For polynomial of degree 1 and η=0.1

(w1,w2) = (0.1,0)

MSE =22.087034100970694

iii) For For polynomial of degree 2 and η=0.035

(w1,w2) = (0.15,0.1)

MSE= 18.78717186352106

iv) For For polynomial of degree 4 and η=0.052

(w1,w2) = (0.5,0.3)

MSE = 9.440455661391942

We get the least value for For polynomial of degree 0.5