Name: Bharat Kashyap Karri
ID: 2020AAPS0319H

FODS Assignment 1

1A

1A) Given Prior $\cdot\ \beta(\mu|a,b) \rightarrow$ Beta distribution

NAME BHARAT KASHYAP KAKKI ID. 2021AAPS0219H

Prior $\beta(\mu|2,2) = \frac{\Gamma(a+2)}{\Gamma(2)\Gamma(2)} \cdot \mu^{2-1} \cdot (1-\mu)^{2-1}$ $\quad \Gamma(x) = \int_0^\infty u^{x-1} e^{-u}\, du$

In general $\beta(\mu|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \cdot \mu^{a-1} \cdot (1-\mu)^{b-1} = P(\mu)$

$\mu \rightarrow$ true mean $\quad D \rightarrow$ dataset $\quad \mu = P(\text{people liking the update})$

$P(\mu|D) = \frac{P(D|\mu) \cdot P(\mu)}{P(D)}$ $\quad P(D) \rightarrow$ constant

$P(\mu|D) \propto P(D|\mu) \cdot P(\mu)$

$\propto P(\{\text{liked}\}|\mu) \cdot P(\{\text{liked}\}|\mu) \cdots P(\{\text{liked}\}|\mu) \cdot P(\{\text{disliked}\}|\mu) \cdots P(\{\text{disliked}\}|\mu) \cdot P(\mu)$

$\propto \mu^m \cdot (1-\mu)^n P(\mu)$ where m people in the sample liked the update and n people did not like the update.

$\propto \mu^m (1-\mu)^n \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \cdot \mu^{a-1} \cdot (1-\mu)^{b-1}$

$\propto \mu^{m+a-1} \cdot (1-\mu)^{n+b-1}$

$P(\mu|D) = \frac{\mu^{m+a-1} \cdot (1-\mu)^{n+b-1}}{\int_{all\,\mu} \mu^{m+a-1} \cdot (1-\mu)^{n+b-1}} = \frac{\Gamma(a+b+m+n-2)}{\Gamma(m+a-1)\Gamma(b+n-1)} \cdot \mu^{m+a-1} \cdot (1-\mu)^{n+b-1}$

$P(\mu|D) = \beta(\mu|a+m, b+n)$

Prior : $\beta(\mu|2,2)$ $\quad$ Posterior 1 : $\beta(\mu|2+40, 2+10) = \beta(\mu|42,12) \rightarrow$ Prior to next posterior

Posterior 2: $\beta(\mu|42+13, 12+17) = \beta(\mu|55,29)$

Posterior 3: $\beta(\mu|125,59) \rightarrow$ [redacted] Posterior of 0.

Name : BİRHAT KARAYAĞ KAPEL

ID : 2010AAPS0119A

If likelihood [redacted] is a binomial distribution then the posterior is given as beta distribution

Here the beta distribution is given by $\beta(a|\alpha,\gamma)$

Taking the mean probability as a : $a = 0.181$

Likelihood of a = P(liked the update) = 0.181

Probability for m people in a sample size of N to have liked the update

$$= \frac{N!}{m!\,(N-m)!} \cdot (0.181)^{m} \cdot (0.819)^{N-m}$$

1B

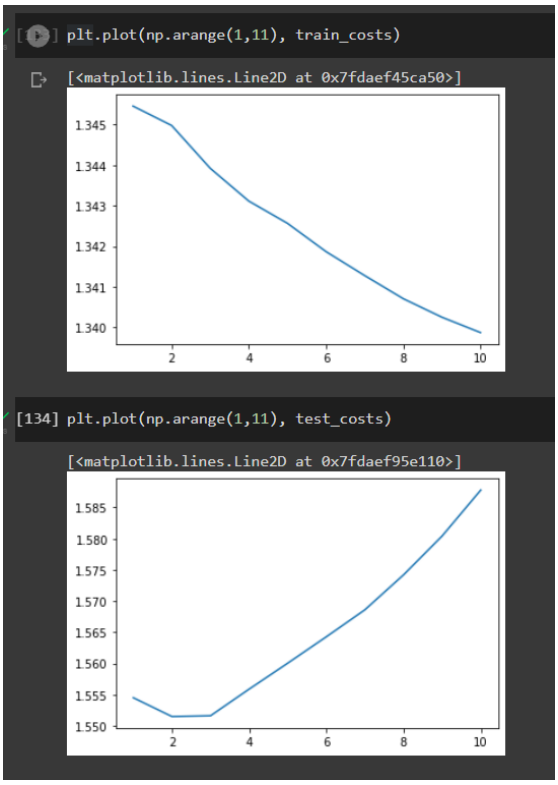
i) The model is a regression model. In the first few implementations it is a polynomial regression model where each column is gaussian normalized to prevent overflow. Each column corresponds to the transformed input data.

In batch gradient descent all of the sample data is used to train the model in each epoch. This results in a slower training process with more number of iterations. With stochastic gradient descent training is done with only one sample for each epoch. Thus training is faster with approximately the same accuracy.

Regularization is taken as the sum of absolute values of each weight to the power of the required norm. It is also multiplied by a hyper-parameter called the regularization constant.

ii) Batch Gradient Descent errors: Degree, Training error, Testing error

```
[[ 1.          1.3454587   1.5544906 ]
 [ 2.          1.34498056  1.55148404]
 [ 3.          1.34392362  1.55160594]
 [ 4.          1.34311396  1.55587694]
 [ 5.          1.3425665   1.56003595]
 [ 6.          1.34186746  1.56427138]
 [ 7.          1.34127743  1.56860186]
 [ 8.          1.34070649  1.57420134]
 [ 9.          1.34024991  1.58043302]
 [10.          1.33987043  1.58779804]]
```

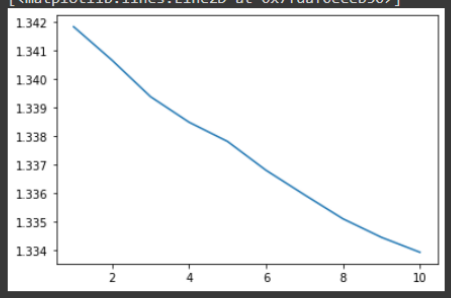


The first graph shows the training error and shows that the higher the degree of the polynomial lesser is the training error. However it can be seen that the testing error rises after degree 2 thus showing that the model starts to overfit after degree 2.

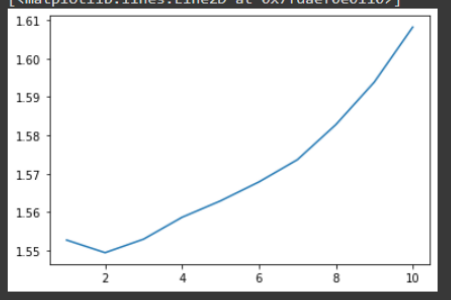
Stochastic Gradient Descent errors: Degree, Training error, Testing error

```
[[ 1.          1.34181892  1.55266317]
 [ 2.          1.34064605  1.5493665 ]
 [ 3.          1.33937258  1.55289108]
 [ 4.          1.33848216  1.55859585]
 [ 5.          1.33781033  1.56287786]
 [ 6.          1.33680346  1.56782025]
 [ 7.          1.33594635  1.57361976]
 [ 8.          1.33510799  1.58276294]
 [ 9.          1.33446126  1.59386312]
 [10.          1.33393898  1.60820828]]
```

```
[137] plt.plot(np.arange(1,11), train_costs)
[<matplotlib.lines.Line2D at 0x7fdaf0eeeb50>]
```



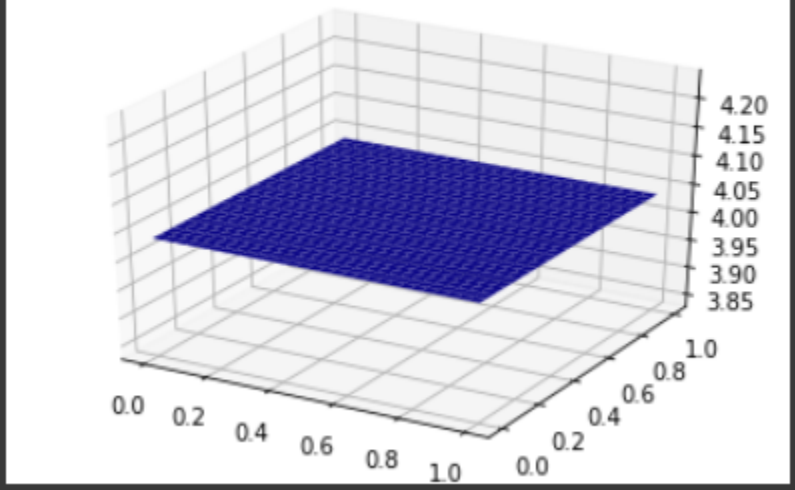

```
plt.plot(np.arange(1,11), test_costs)
[<matplotlib.lines.Line2D at 0x7fdaef0e0110>]
```



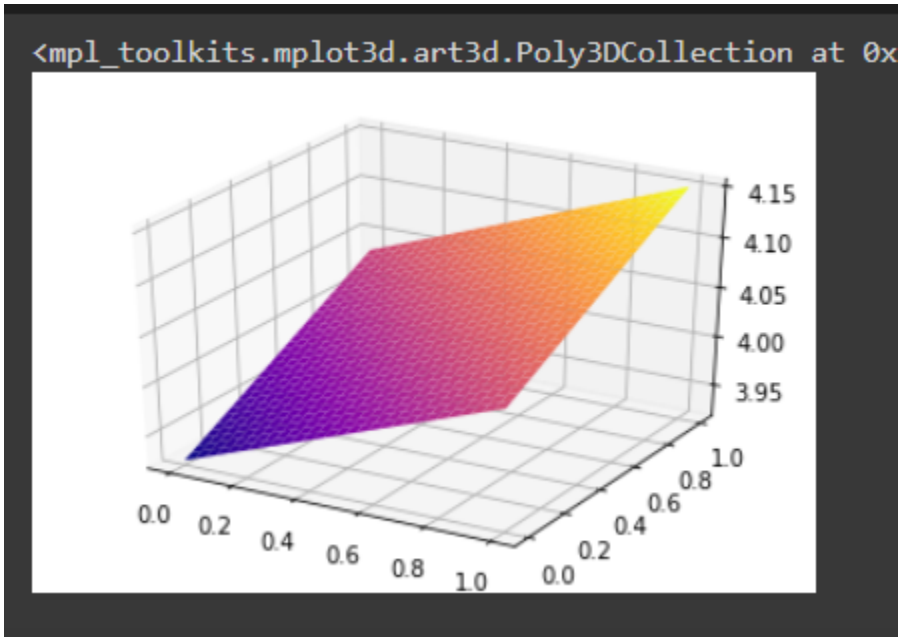
SImilar to batch gradient descent, stochastic gradient descent also shows that the model overfits after degree 2.

iii) Surface plots for different degrees
Degree 0:

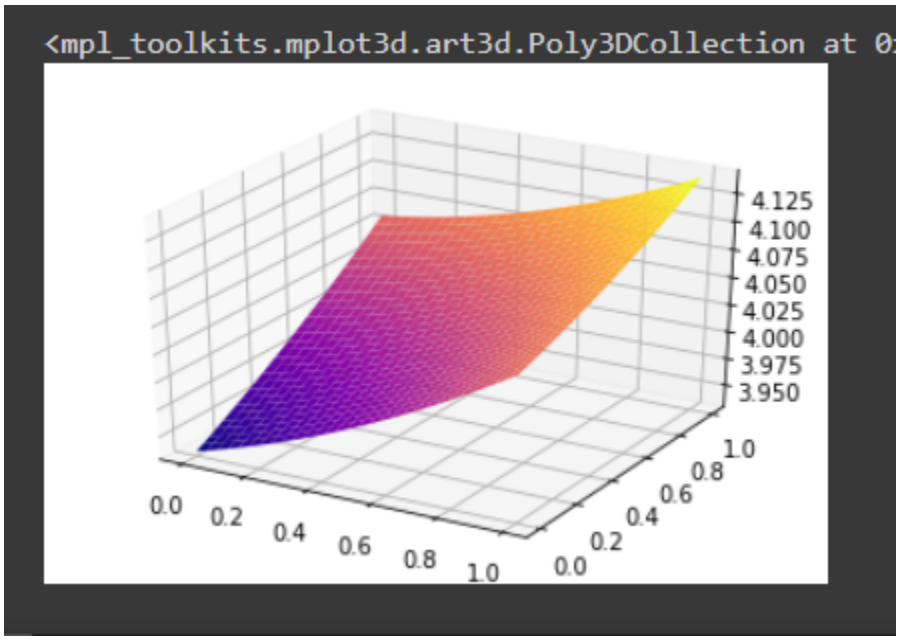
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7fda
```



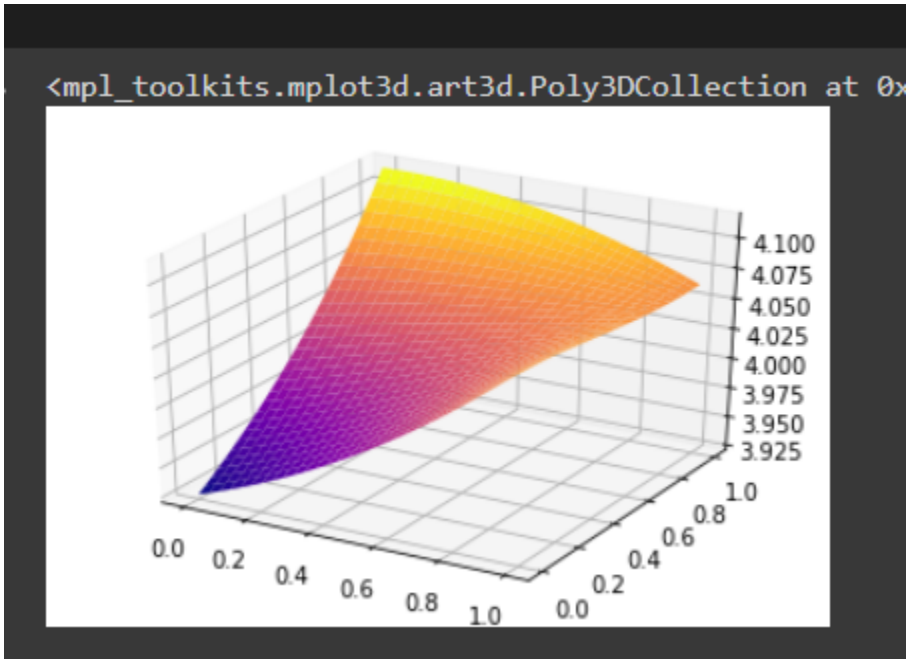
Degree 1:



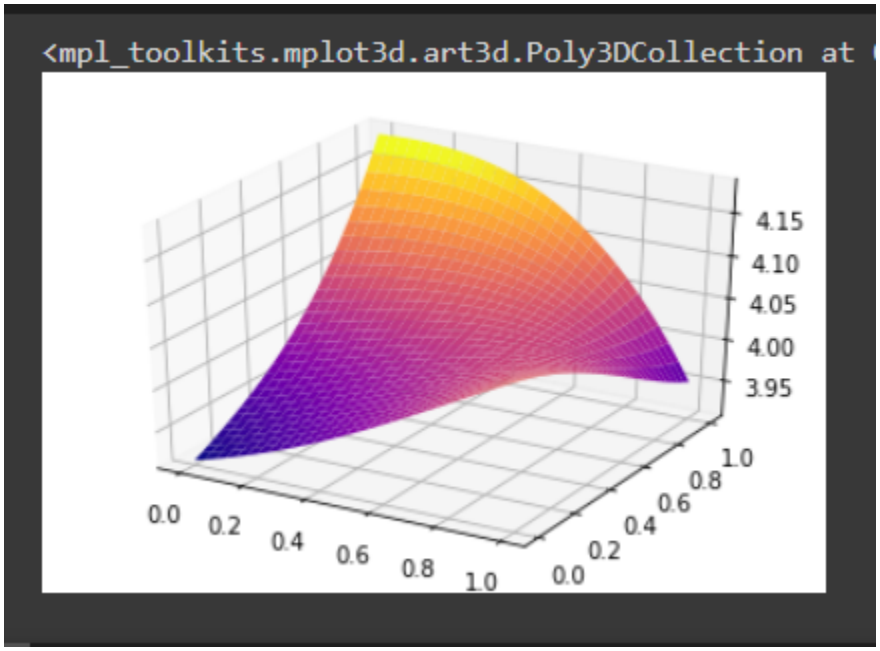
Degree 2:



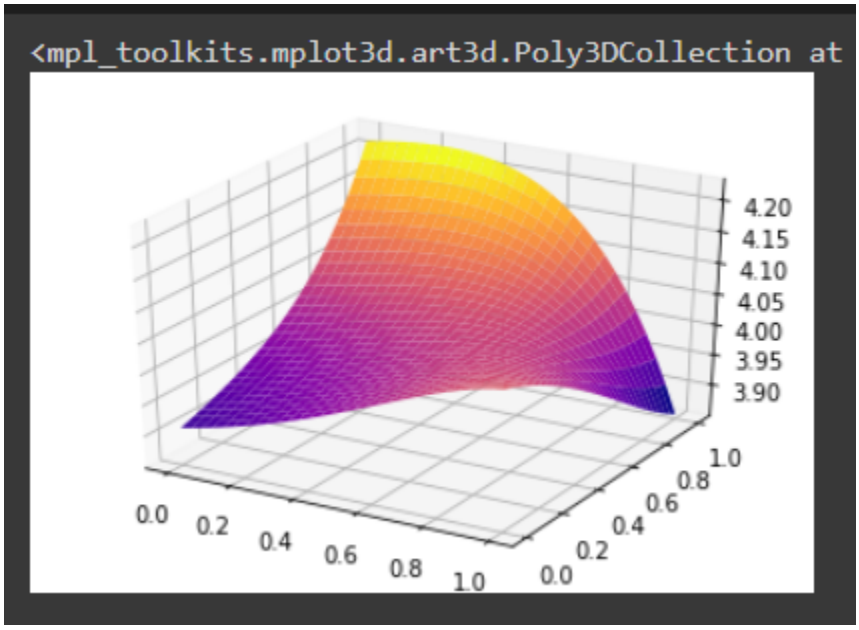
Degree 3:



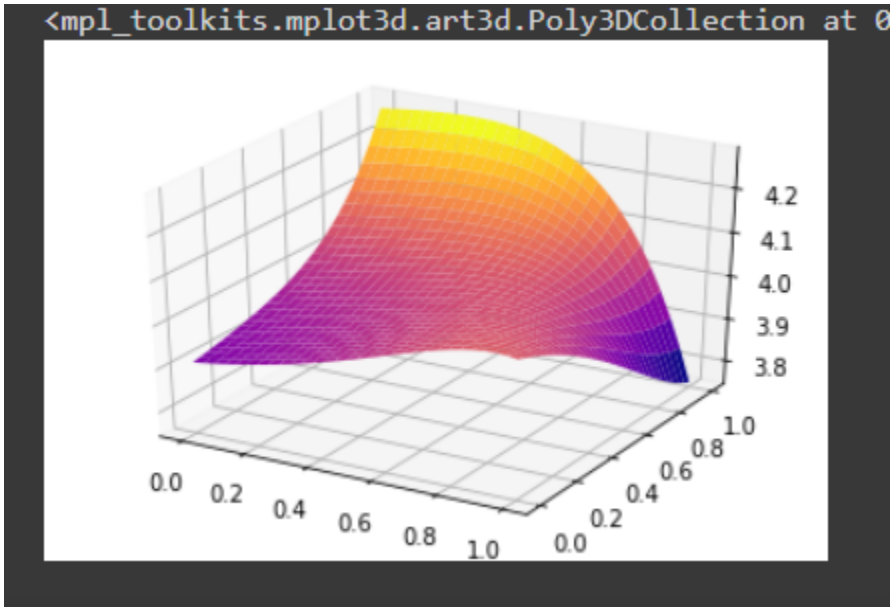
Degree 4:



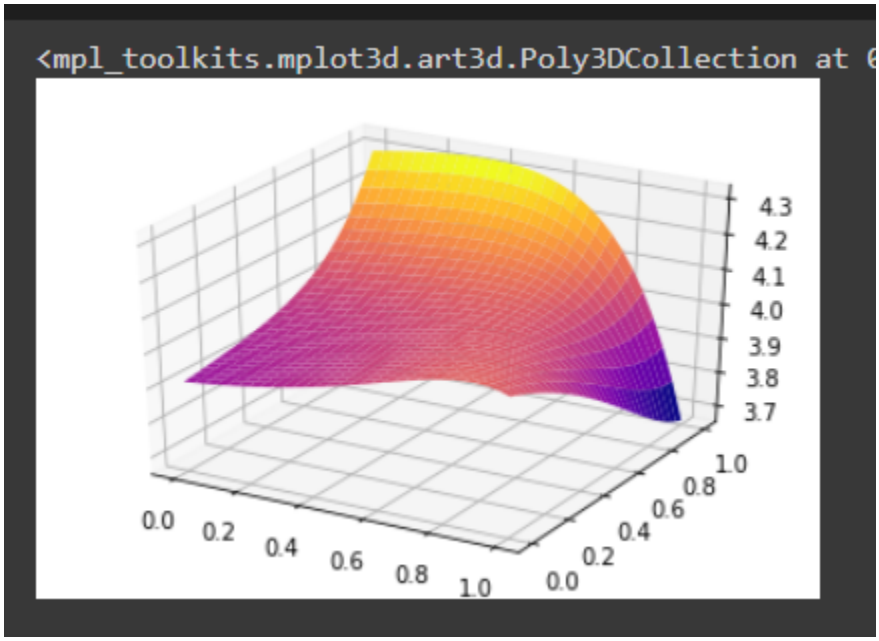
Degree 5:



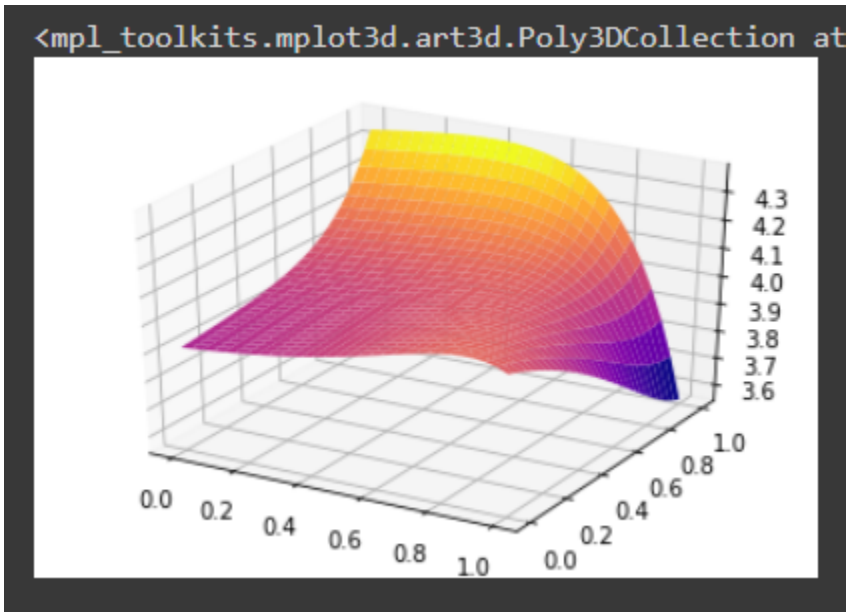
Degree 6:



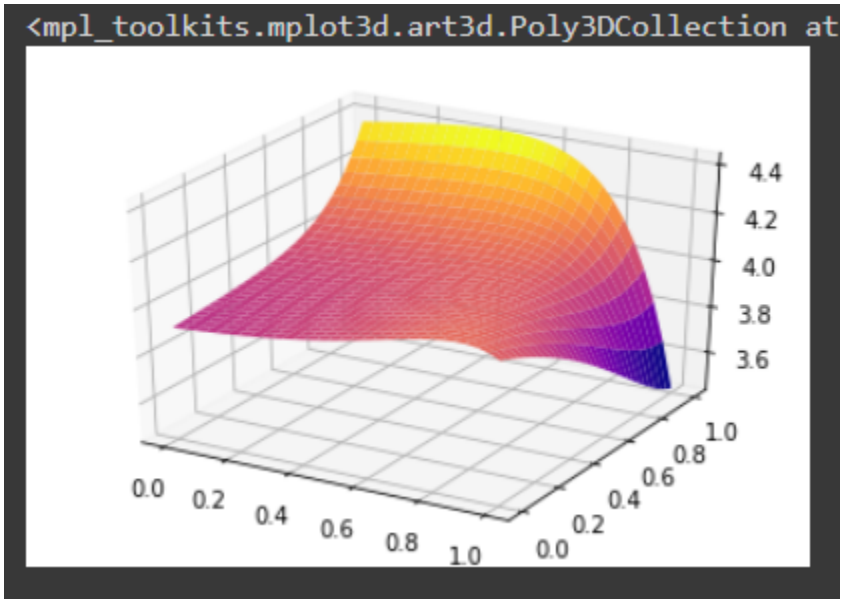
Degree 7:



Degree 8:



Degree 9:



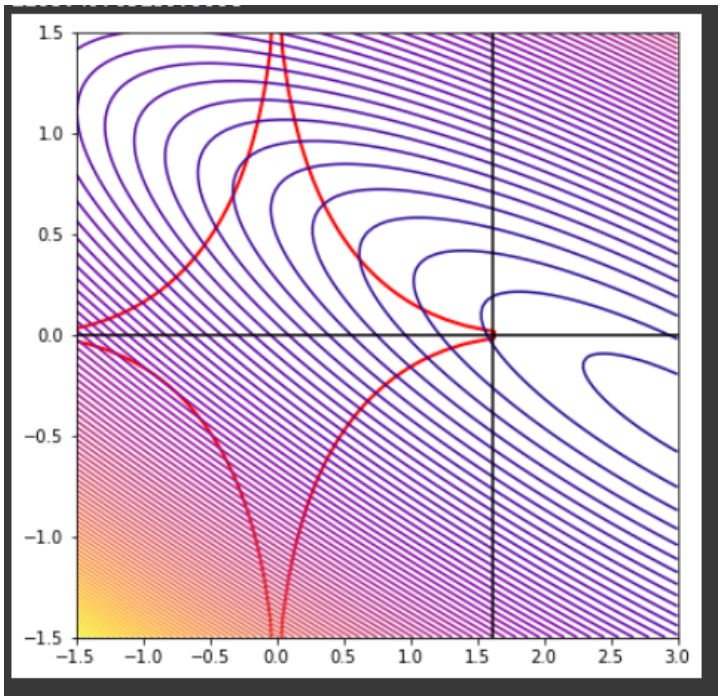
iv) After using grid search to find the optimum regularization rate for each of the values of q. Given below are the regularization rates corresponding to each q and the minimum error rates.

q = 0.5, reg_const = 10e-5, loss = 1.05514
q = 1, reg_const = 10e-4, loss = 1.05407
q = 2, reg_const = 10e-5, loss = 1.04617
q = 4, reg_const = 10e-7, loss = 1.05677

Best model with respect to testing error without regularization is degree 2 with error rate 1.5585

1C

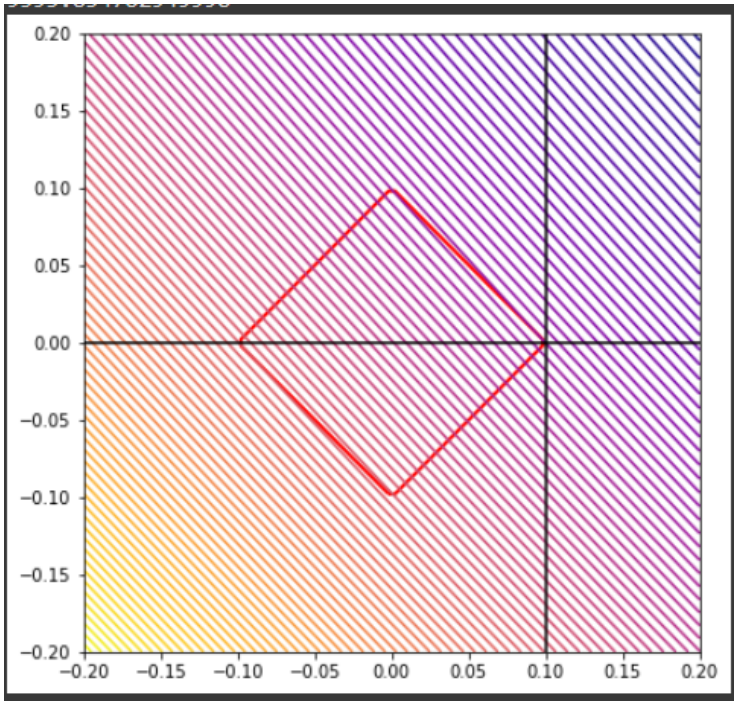
i) q = 0.5



Point of intersection - w1,w2 = 0, 1.62

Error at the point of intersection = 2108.5

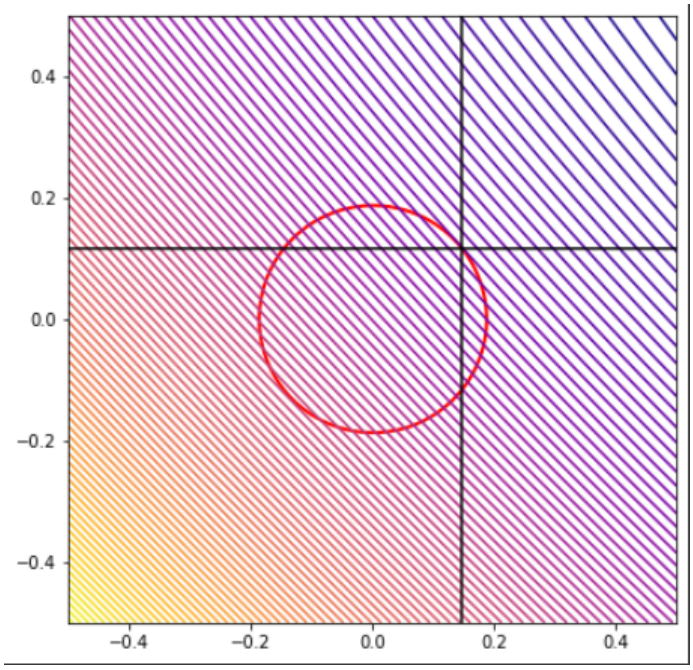
q = 1



Point of intersection - w1,w2 = 0, 0.1

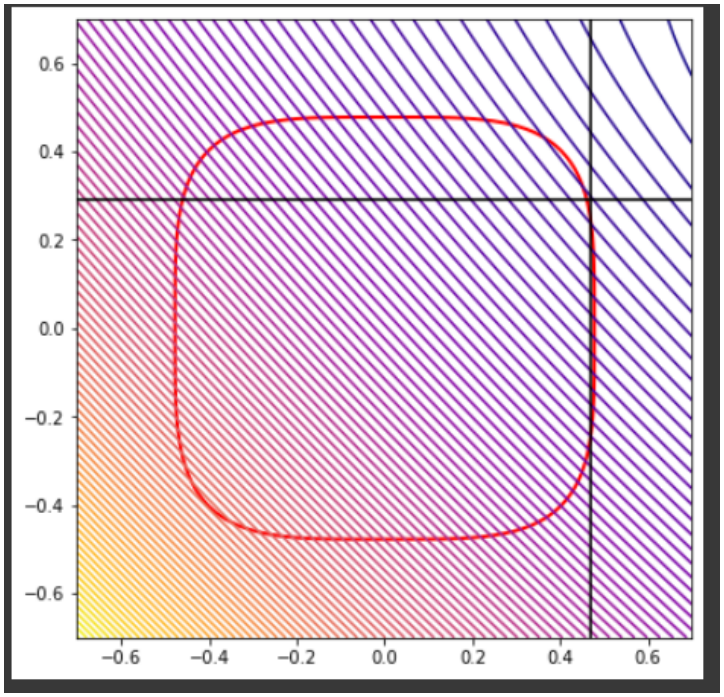
Error at the point of intersection = 9595.85

q = 2



Point of intersection - w1,w2 = 0.115, 0.145
Error at the point of intersection = 8236.35

q = 4



Point of intersection - w1,w2 = 0.29, 0.47
Error at the point of intersection = 5042.88