ELL409 Assignment 1 Report

Ayush Singh, Entry Number- 2019MT60748

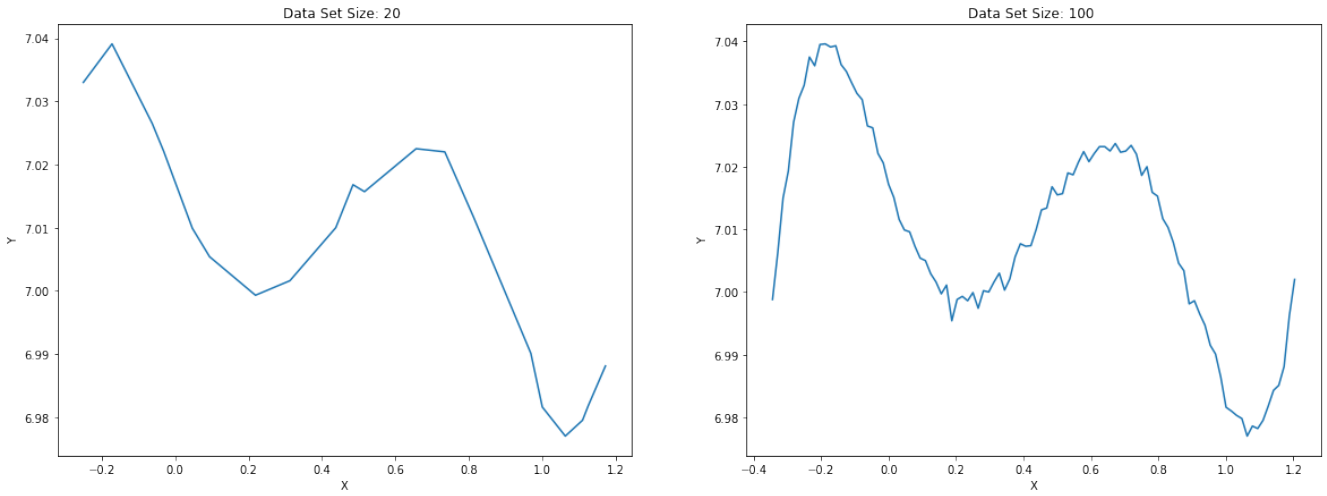
Abstract

This report consists of 3 Parts - 1A (Section- 1.1),1B (Section- 1.2), and 2(Section - 2) for the respective problems.

1 Problem 1

1.1 Problem 1A

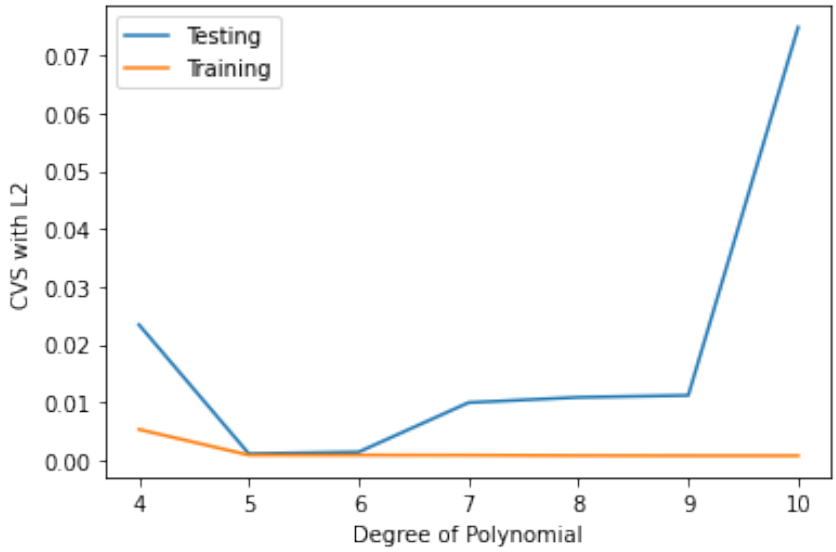
First we look at the data Plots, to get a feel of the graph:



Note that this data has 3 turns (the curve bends around 3 times). So the optimal degree might be 3. In most of the graphs used to find under-fitting and over-fitting, I have used the Cross Validation Score. This is basically, the sum of the individual test scores using an appropriate error function(usually Sum of Squares or L_2 error).

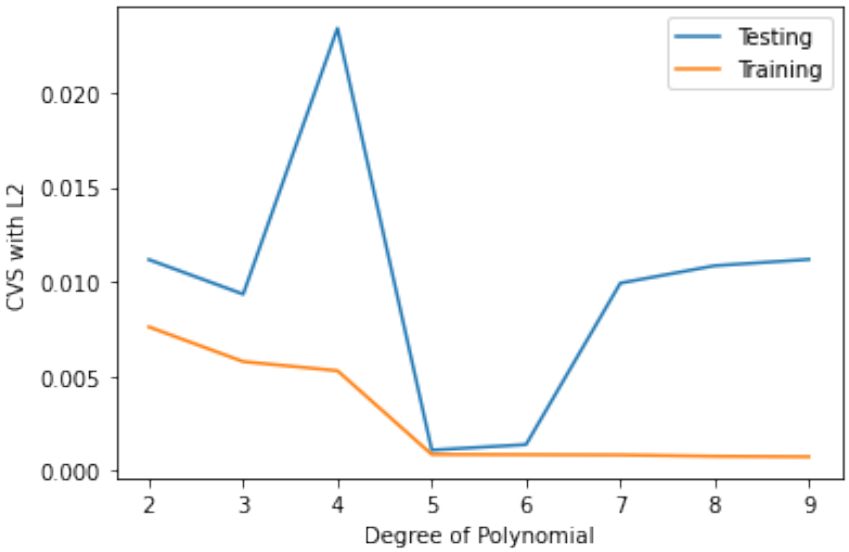
1.1.1 Curve-Fitting using Pseudo Inverse

We take batch size = 20, and use pseudo inverse to find the optimal degree.



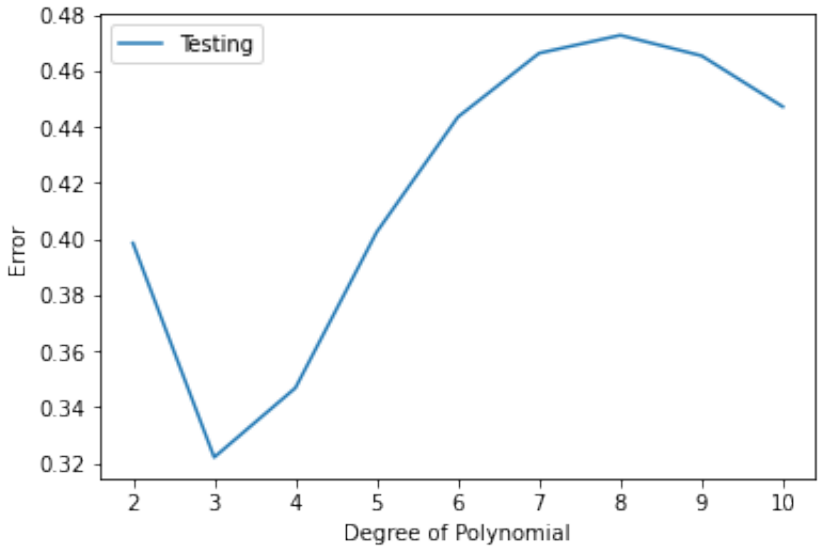
We can see that the optimal, or the so-called sweet spot for the testing error, of the degree occurs at degree 5.
Also, we can see that the training error keeps on decreasing as M as increases. Thus, according to this the degree of the polynomial should be 5.

For batch size = 100, also we can see below that the degree should be 5 according to the U shape at 5.

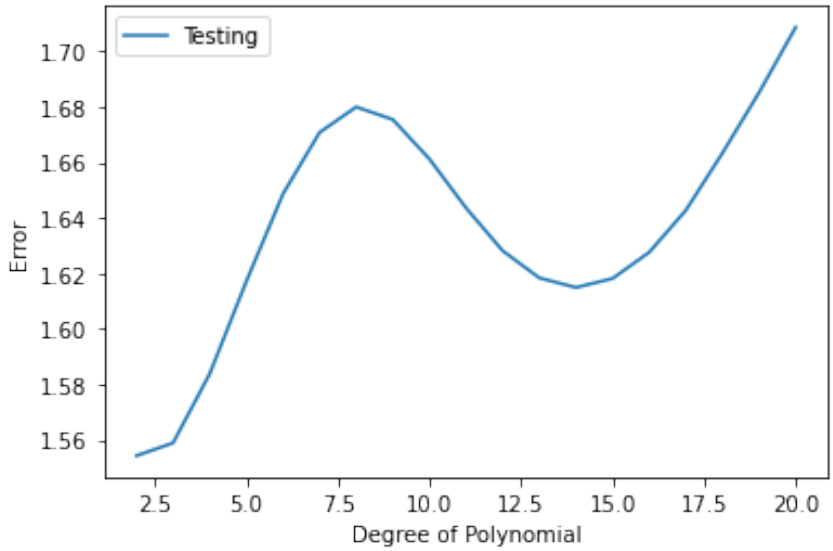


1.1.2 Curve-Fitting using Gradient Descent with Data Size 20

Using Stochastic Gradient Descent on Batch Size = 1 and a 10-Fold Cross Validation, we get the following:



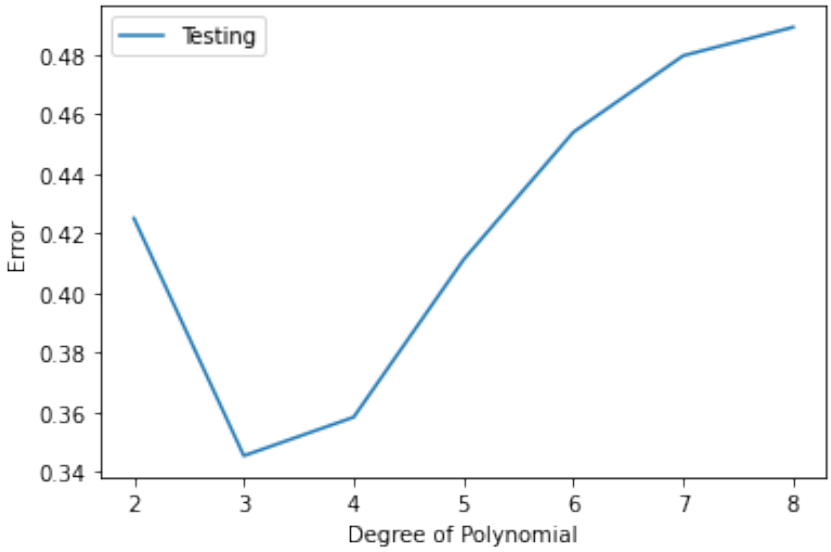
This also agrees with the previous plots.
Now, we use Full Batch Gradient Descent and a 10-Fold Cross Validation.



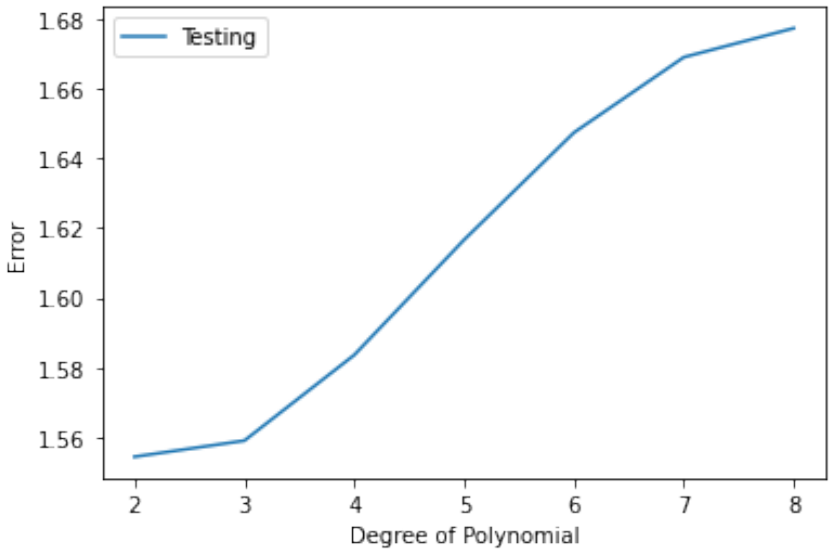
These curves tell us that for this small data size, the GD has the sweet spot near 3. I think the reason this is better for the degree 3, is because we can roughly see that there are 3 turns that the data takes.

1.1.3 Curve-Fitting using Gradient Descent with Data Size 100

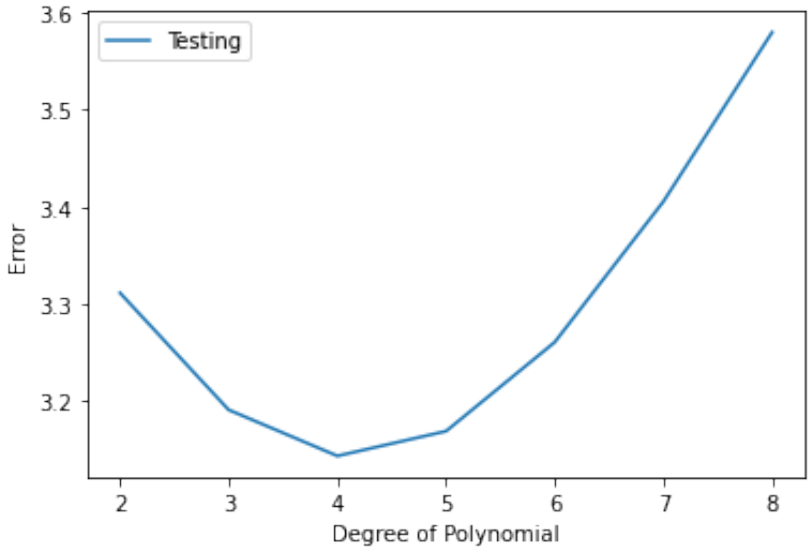
Using Stochastic Gradient Descent on Batch Size = 1 and a 10-Fold Cross Validation, we get the following:



This also agrees with the previous plots.

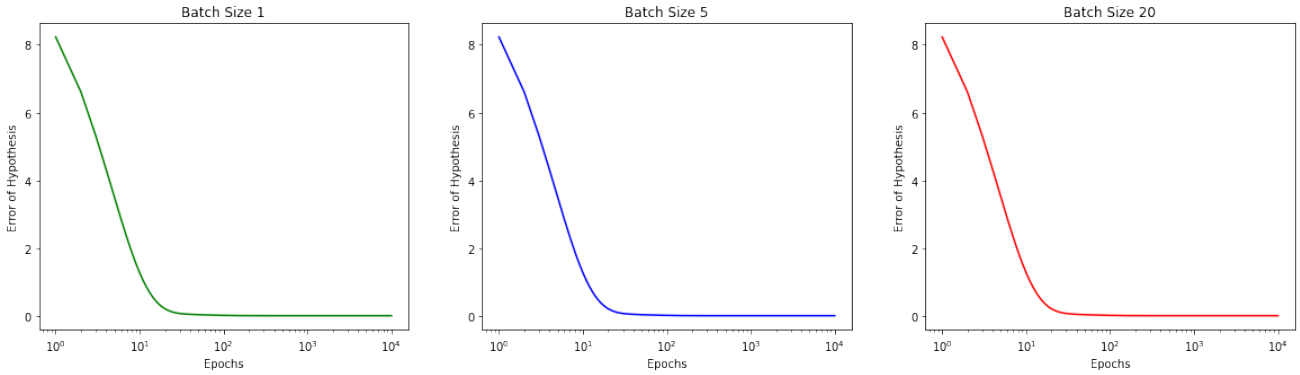


Now, we use Full Batch Gradient Descent and a 10-Fold Cross Validation.



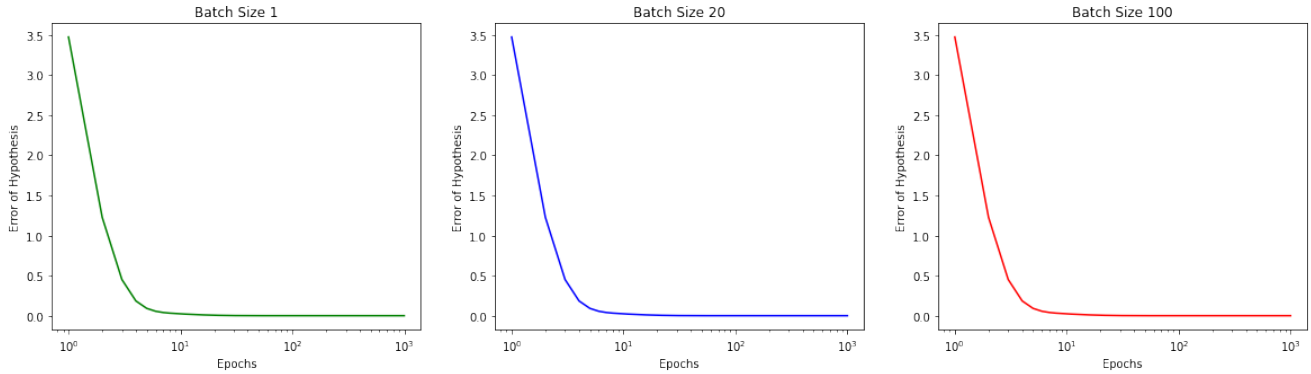
These curves tell us that for this small data size, the GD has the sweet spot near 3. I think the reason this is better for the degree 3, is because we can roughly see that there are 3 turns that the data takes.

1.2 Plotting the loss functions with number of iterations data size = 20



Please note that scale on Epochs is Logarithmic. The convergence here is pretty fast, the error dips to its smallest values at Epoch 100.

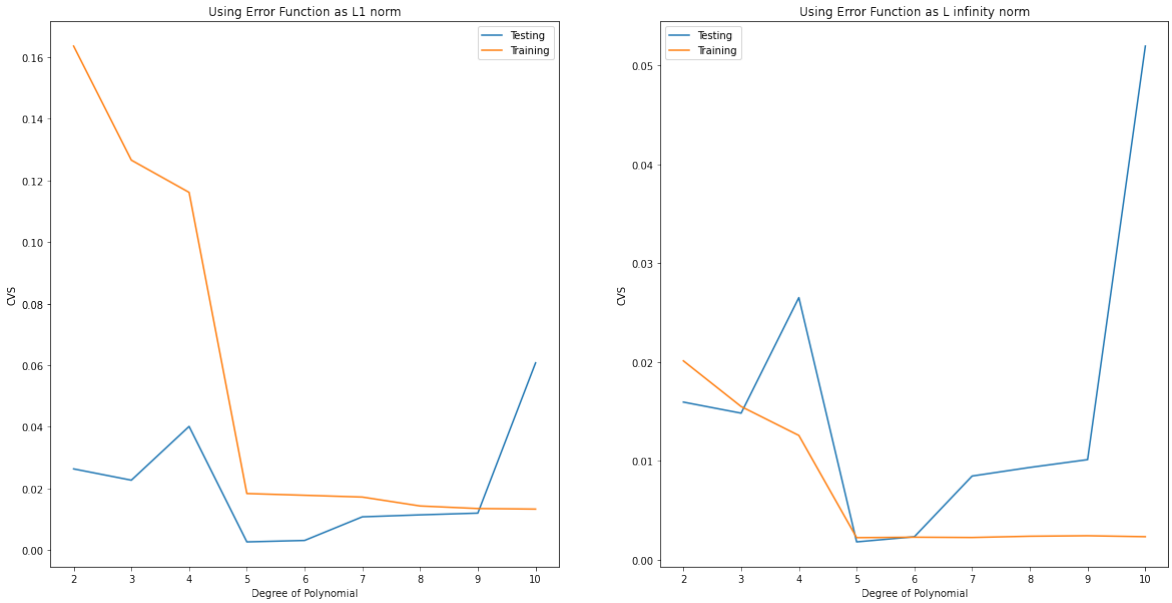
1.3 Plotting the loss functions with number of iterations data size = 100



Please note that scale on Epochs is Logarithmic. The convergence here is even faster than before. The error dips to its smallest values at Epoch 10.

1.3.1 Using Different Error Functions for data size 20

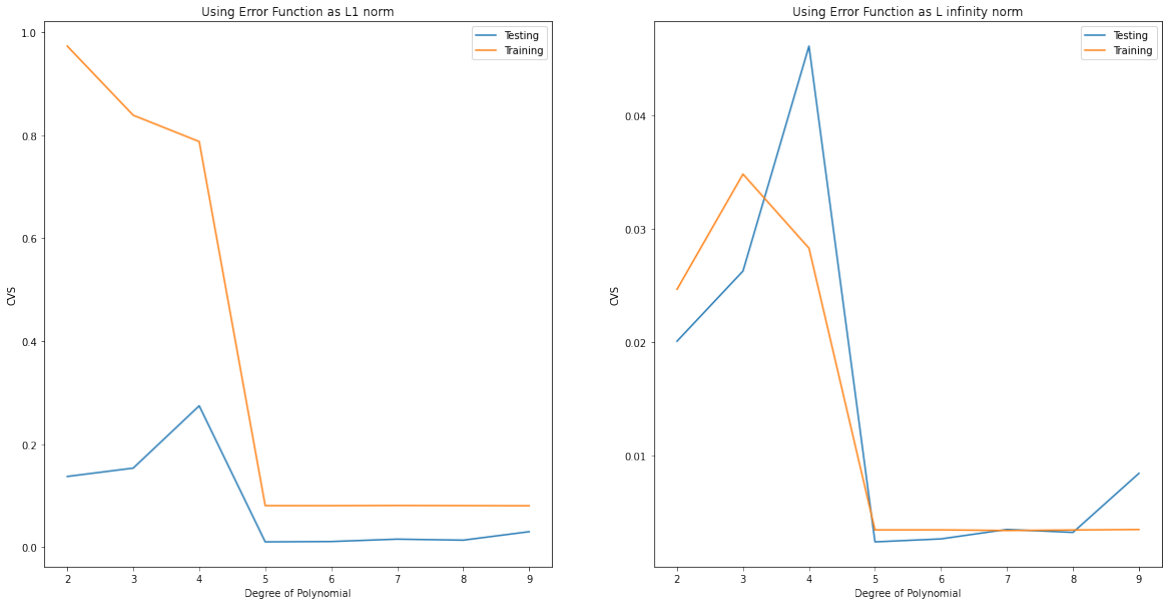
I have already use Sum of Squares or L2(which is the root of Sum of Squares) error. Now, I will use, L_1 and L_∞ norms.



From this, we can clearly see that the so- called sweet spot is hit at degree 5, which was the same as the sum of squares error. Also, at degrees lower than 5, we have higher training and testing error, which imply under-fitting. Also, at degrees higher than 5, we have decreasing training error which implies over-fitting, since the testing error decreasing.

1.3.2 Using Different Error Functions for data size 100

I have already use Sum of Squares or L2(which is the root of Sum of Squares) error. Now, I will use, L_1 and L_∞ norms.

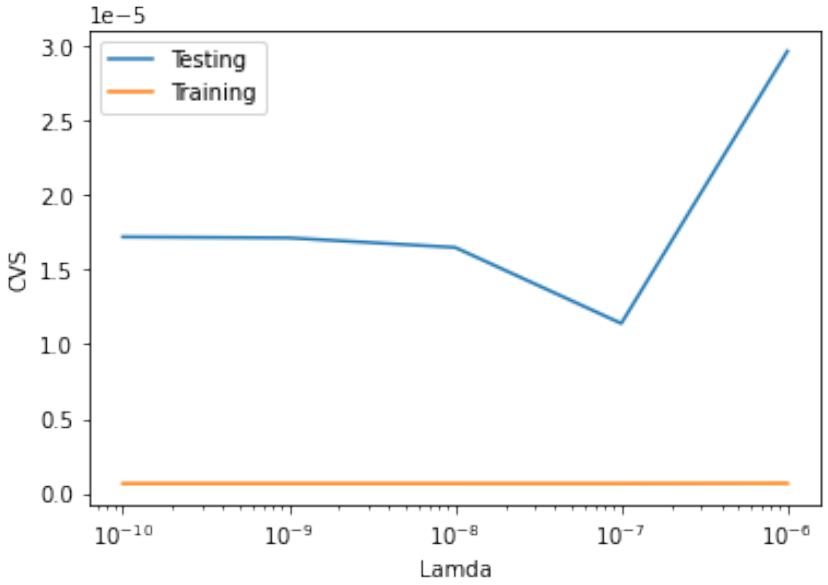


From this, we can clearly see that the so- called sweet spot is hit at degree 5, which was the same as the sum of squares error. Also, at degrees lower than 5, we have higher training and testing error, which imply under-fitting. Also, at degrees higher than 5, we have decreasing training error which implies over-fitting, since the testing error decreasing.

Finally, we note that the error values for data set 100 using these error functions are, in fact, quite higher than the for data size 20. This can be attributed to the fact that there is more under-fitting in the data size 100, when the value of the degree is same. This is why we get higher error for size 100.

1.3.3 Regularisation: Finding the optimal regularisation λ for data size 20

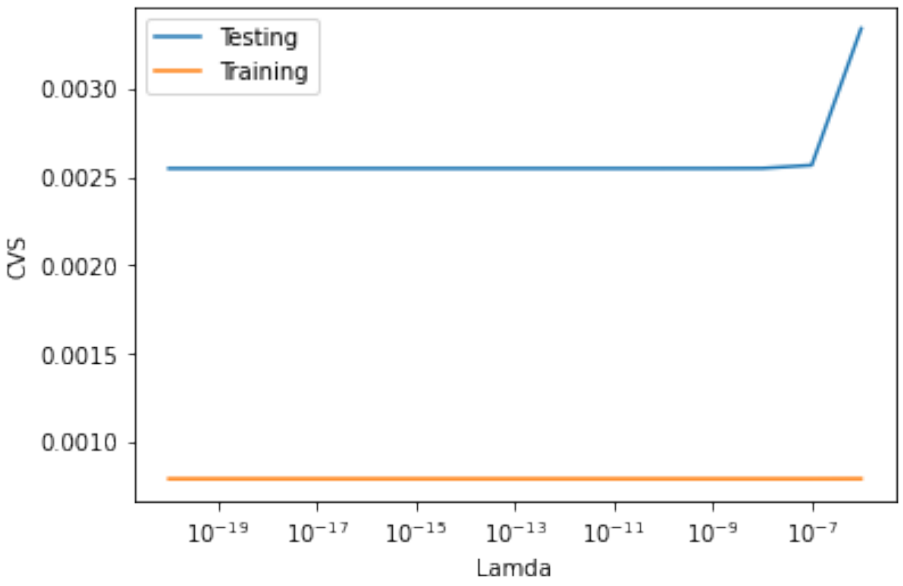
Plotting the cross validation values:



We get the optimal $\lambda = 10^{-7}$

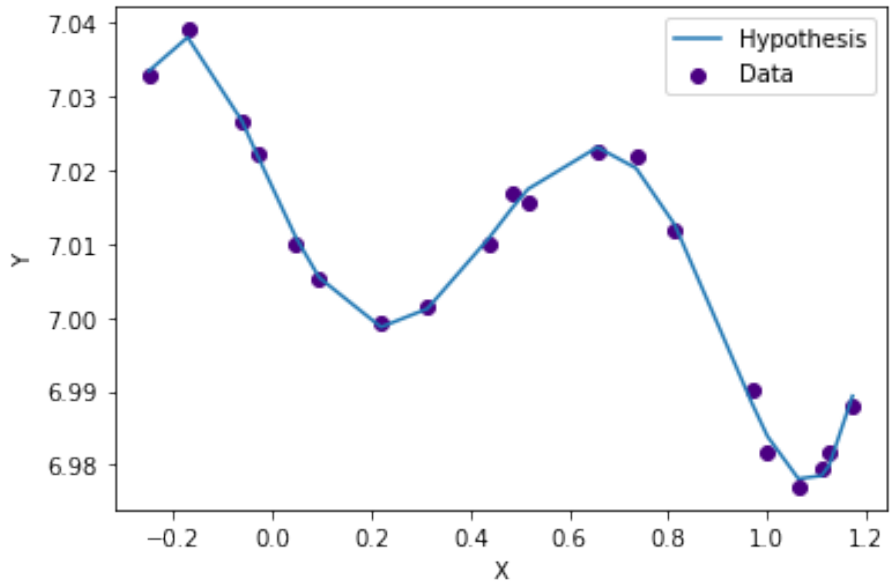
1.3.4 Regularisation: Finding the optimal regularisation λ for data size 100

Plotting the cross validation values:



Here, we do not have an optimal λ since all $\lambda \leq 10^{-7}$ give same testing error, and after that testing error increases. Hence, we can assume $\lambda = 10^{-7}$ here as well.

1.3.5 Guess for underlying polynomial with data size = 20

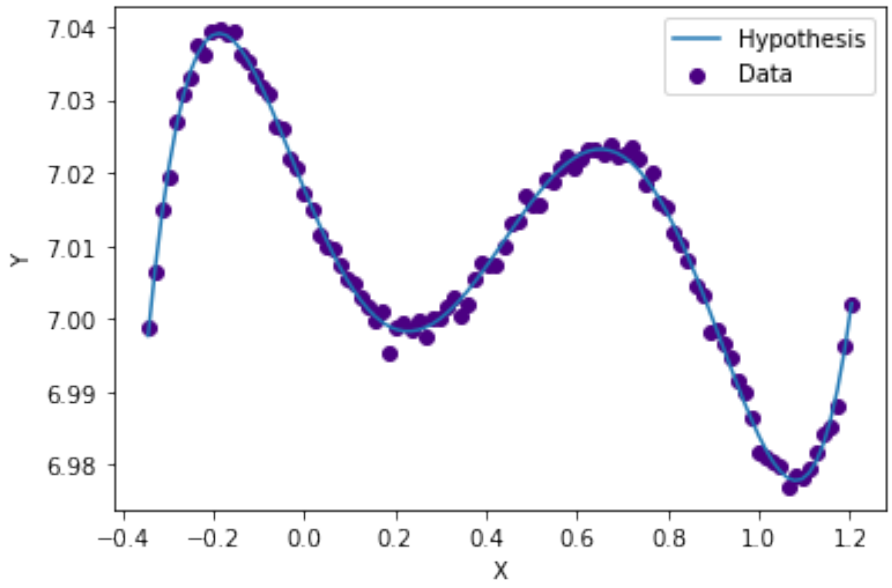


We get the mean of the noise as $\mu = 3.8 \cdot 10^{-8} \approx 0$. Also, we get the standard deviation $\sigma = 0.001214 \implies$ Variance $\sigma^2 = 1.474 \cdot 10^{-6}$

The weights of this polynomial are

$\mathbf{W} = [7.017 \quad -0.147 \quad 0.116 \quad 1.185 \quad -2.165 \quad 0.978]$

1.3.6 Guess for underlying polynomial with data size = 100



We get the mean of the noise as $\mu = -7.01 \cdot 10^{-9} \approx 0$. Also, we get the standard deviation $\sigma = 0.001121 \implies$ Variance $\sigma^2 = 1.257 \cdot 10^{-6}$

The weights of this polynomial are

$\mathbf{W} = [7.0177 \quad -0.151 \quad 0.114 \quad 1.224 \quad -2.224 \quad 1.003]$

1.3.7 Final guess of the underlying polynomial

My final guess for the underlying polynomial is in accordance with the ones in the previous section, that is The weights of this polynomial are

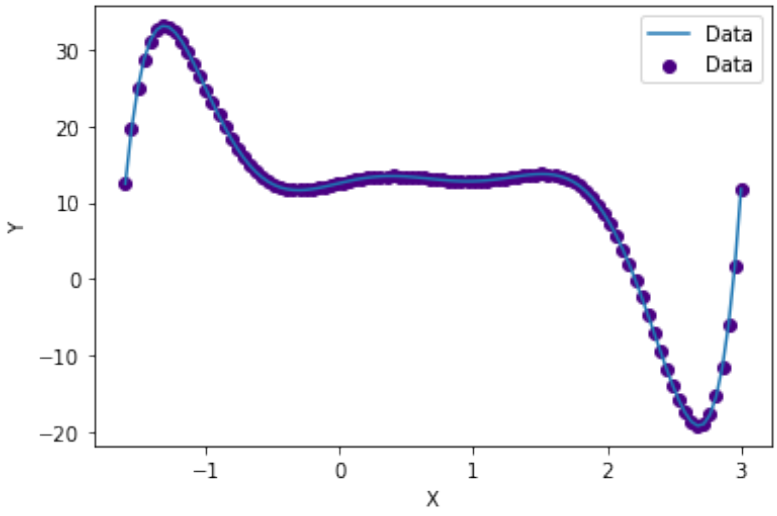
$\mathbf{W} = [7.018 \quad -0.151 \quad 0.114 \quad 1.224 \quad -2.224 \quad 1.003]$

I am using this because using the estimate from the polynomial for data set 20 uses less data, and therefore, is less accurate. Also, we must note that the Gradient Descent method, which gave the optimal degree as 3, was not chosen.

I did this because the error value of the GD at degree 3 was ≈ 0.32 was considerably higher than the error I obtained for the Pseudo Inverse Method at degree 5 which was ≈ 0.01. These values can be seen from the appropriate plots in sections 1.1.1, 1.1.2, and 1.1.3.

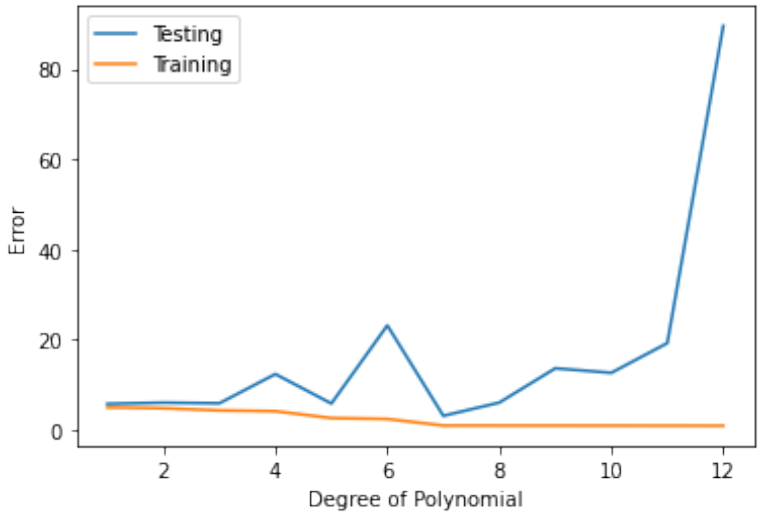
1.4 Problem 1B

We look at the data, this time for all 100 points:



1.4.1 Determining the optimal Degree of the Polynomial:

We plot the Cross Validation Test and Train scores for this Data against the degree of polynomial.

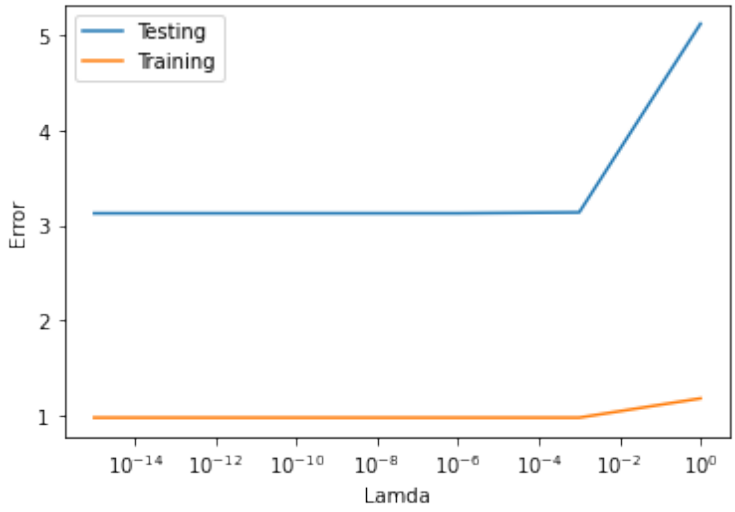


From this, we realize that the testing error is minimised at degree = 7, since that is also the sweet spot or the U- shape of the curve. Also, there is a minima at degree 5, which can also be seen as a pretty good estimate. But since the minima occurs for the last time at 7 and since error at 7 is lesser than the one at 5, we assume the degree to be 7.

Also, we can see that the training error decreases with higher degree which is the expected trend. Also the testing error keeps on increasing after degree 7.

1.4.2 Determining the optimal value of the Regularisation Coefficient λ:

We plot the Cross Validation Test and Train scores for this Data against the Regularisation Coefficient λ with constant degree as 7.



Here, we notice that the test error doesn't vary with λ until $lamda = 10^{-2}$ after which it increases. Hence, I have taken the regularisation constant to be 10^{-6}, so as to still not over-fit much.

1.4.3 The best fit polynomial

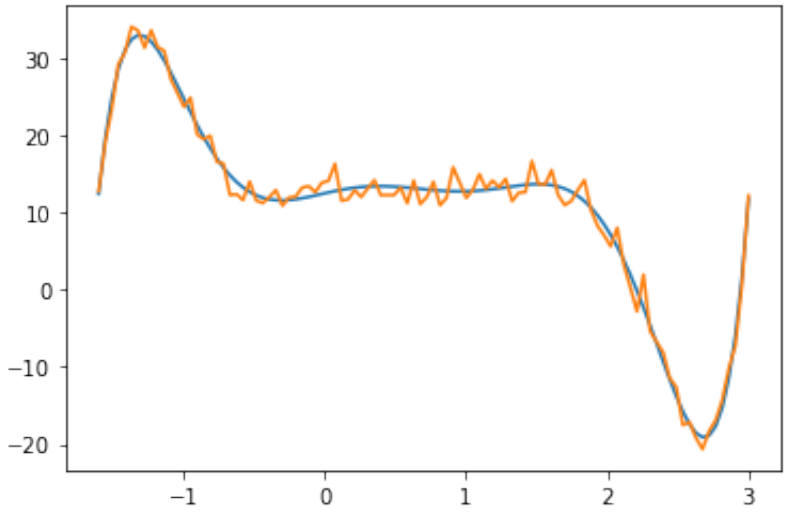
When, we calculate the polynomial with these specifications,
$w_0 = 12.5467622 \quad w_1 = 4.17425227$
$w_2 = -1.290399 \quad w_3 = -13.30614298$
$w_4 = 12.0880470 \quad w_5 = 2.10226078$
$w_6 - 4.48944318 \quad w_7 = 0.97857554$

So the polynomial $p(x)$ is given by:

$$p(x) = \mathbf{W}^T \phi(x) \quad ; \quad \mathbf{W} = [w_0\ w_1\ w_2\ ....\ w_7] \quad , \quad \phi(x) = [1\ x^1\ x^2\ ...\ x^7] \tag{1}$$

where both $\mathbf{W}$ and $\phi(x)$ are column vectors.

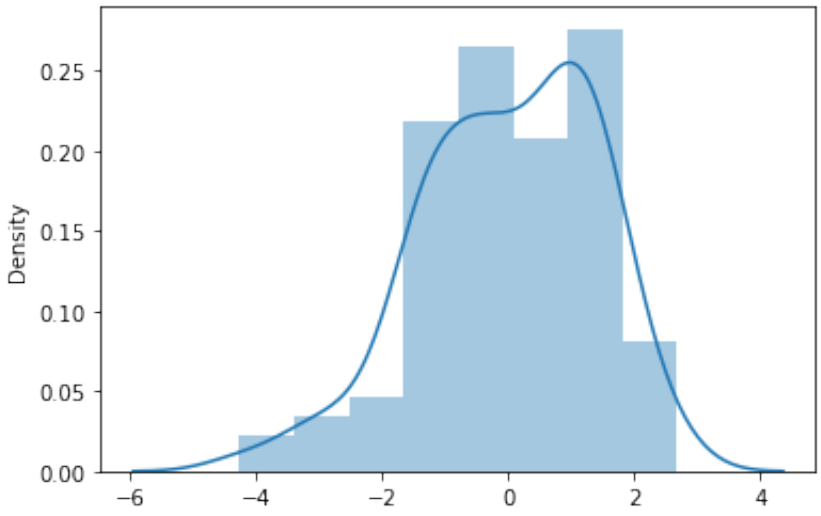
Now, we try to see how the curve fits the data:



We can see that this curve approximates the data pretty nicely.

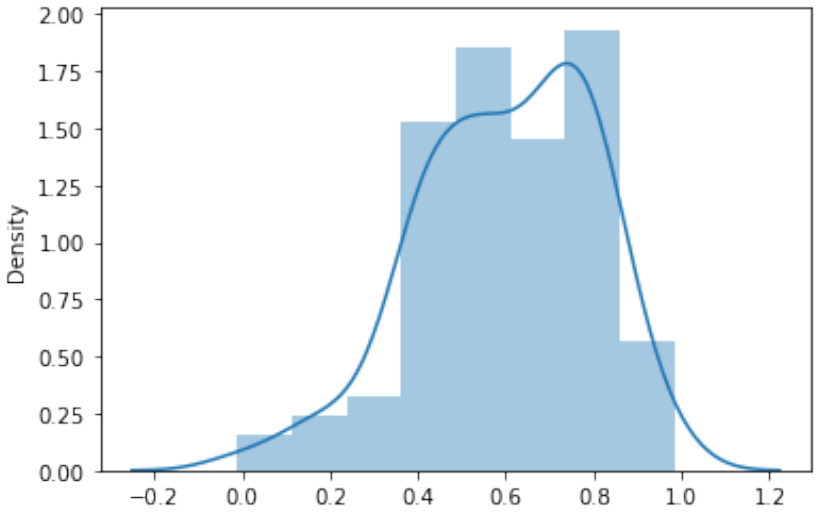
1.4.4 The Distribution of Noise δ

The noise comes out to be in the range (-4.28,2.68). Here, I also found that $\mu_\delta = 10^{-7} \approx 0$, also $\sigma_\delta^2 = 1.96$, We plot the histogram of noise δ, as:



We will attempt to fit this noise distribution into a Beta Distribution. Therefore, we transform the noises δ by scaling by (1/7) and then shifting by (+0.6). Thus, a noise δ gets transformed to θ, where $\theta = \frac{\delta}{7} + 0.6$

Here is, the distribution over θ.



From statistical data here, we can see that if there were to be a Beta distribution over θ, with parameters α and β such as:

$$p(\theta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} \cdot (1-\theta)^{\beta-1}; \tag{2}$$

where B corresponds to Beta Function.
I identified the mean of this distribution over θ to be $\mu = 0.6$. We know from the above link, for a Beta Distribution:

$$\mu = \frac{\alpha}{\alpha+\beta} \implies \frac{3}{5} = \frac{\alpha}{\alpha+\beta} \implies 3 \cdot (alpha + \beta) = 5 \cdot \alpha \implies 2 \cdot \alpha = 2 \cdot \beta \implies \alpha = \frac{3}{2} \cdot \beta \tag{3}$$

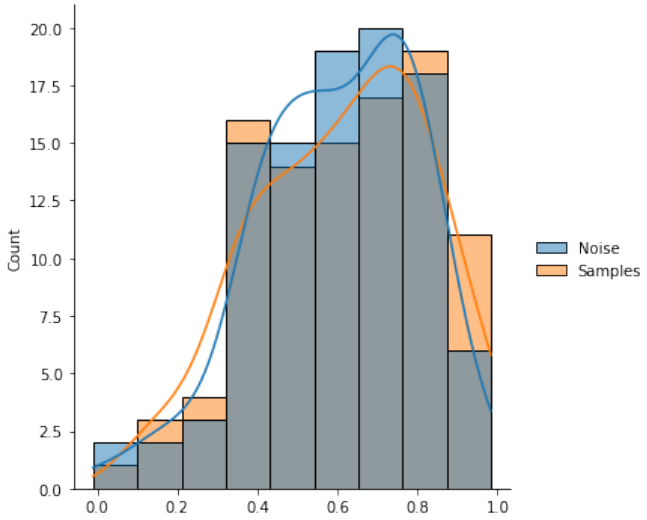
Also, the mode κ is around 0.7, so we use

$$\kappa = \frac{\alpha - 1}{\alpha + \beta - 2} \implies \alpha = 2.4, \quad \beta = 1.6 \tag{4}$$

Finally, transforming θ back to δ, we get

$$p(\delta) = 0.016 \cdot (\delta + 4.2)^{1.4} \cdot (2.1 - \delta)^{0.6} \tag{5}$$

I generated a random distribution over θ from the Beta distribution with parameters α and *beta* as obtained above. Plotting the two distributions on top of one another:



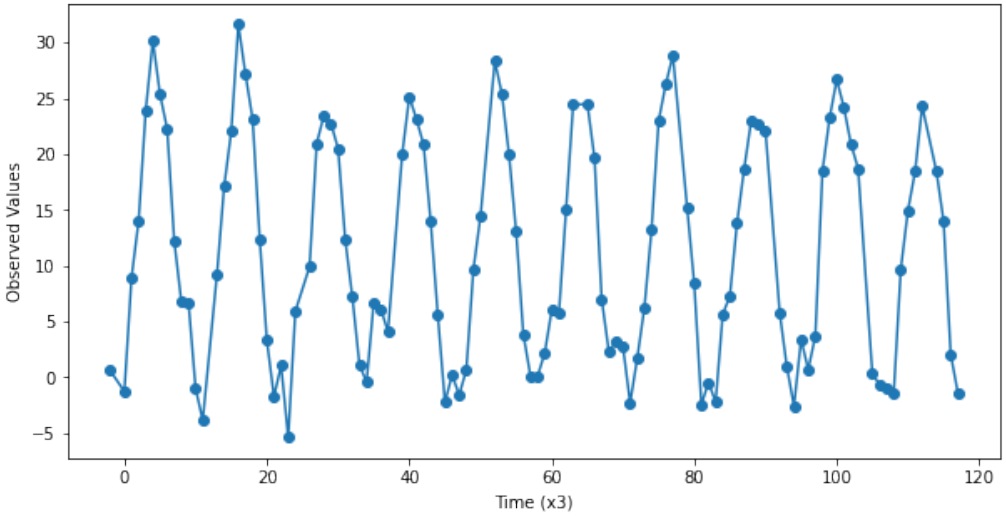
We can see the similarity between the two distributions. Hence, this distribution is a fine approximation of the actual noise distribution.
Note that I am not saying this is the noise distribution.
But since there are not many common continuous distributions known to me except for Beta and Gaussian, I have assumed here that the distribution is Beta, since the noise is told to be non-gaussian.

2 Problem 2

2.1 Time Series Data Interpretation

First, we notice that the days on each data point all have entries 1, so the only variables here are the years(x_1) and months(x_2). So, we create another parameter x_3 which is defined as $x_3 = x_1 \cdot 12 + x_2$. We will henceforth refer to x_3 as time.

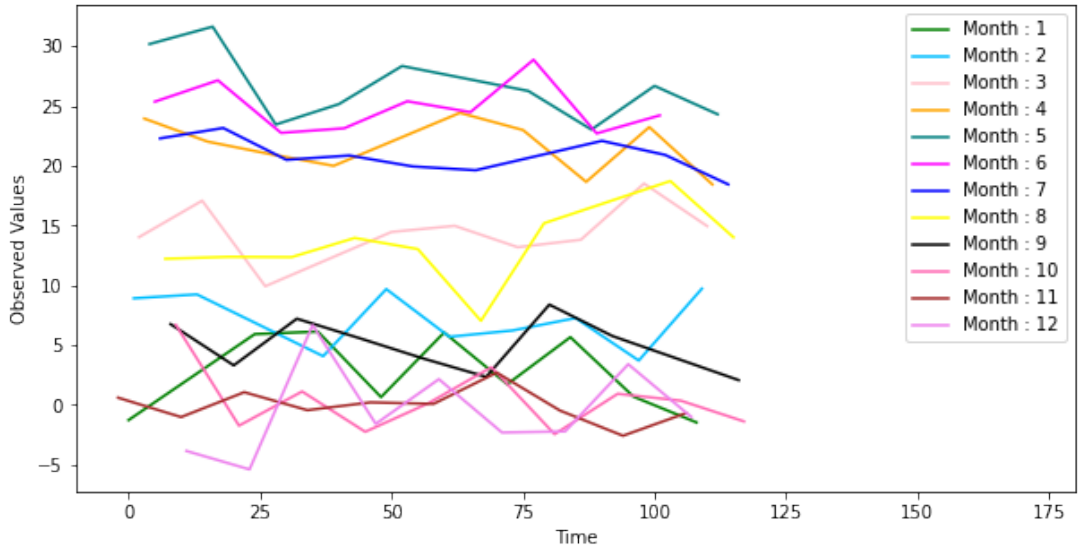
First, we plot the data in terms of time:



We notice that this data is approximately periodic, with the period being 12 counts of x3(that is one year), which makes sense because this is a real world data and trends in the real world are almost always yearly.

2.2 A way to solve

Since the data is periodic, we can realise that the data for any particular month across all the years should be almost similar. So, we plot the observed individually for each month across time:



Here, I got the idea to do a polynomial curve fit for every month separately. And then, for the output the test data points, I could just locate the months for each input and find the value of that data point specifically on the curve of the month to which it belongs.

I followed this method for the remainder of this part.

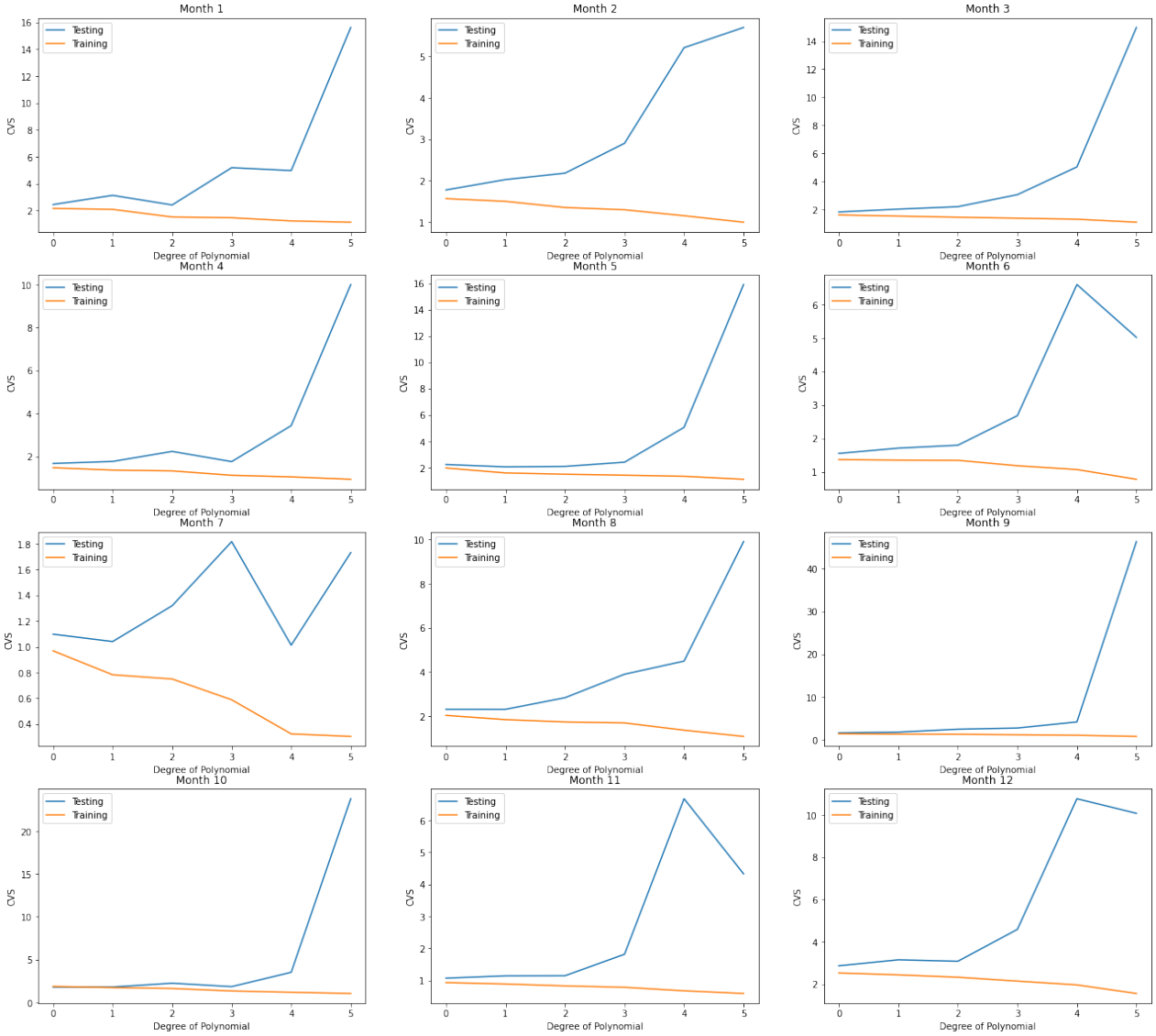
2.3 The first attempt

Initially I assumed the same degree for each month's polynomial to be 3 since that seemed to minimise the overall error.

I used this hypothesis to get curves for different months. But on the test data, I got a score of 12.40326, which indicated that my hypothesis was really far away from the actual data.

2.4 The second attempt

I realised that the different months need not have the same polynomial degree. I decided to use Cross Validation to find the best degrees for each month. This was quite an extensive task. We see these plots as:



Taking these into consideration, I took the individual degrees as degrees = [2,2,2,3,2,2,4,2,2,3,2,2], respectively for each month. With this, I acheived an error of 5.31167, which was considerably better than the first attempt.

2.5 Further attempts

After this, I also plotted the Cross-Validation curves for each Month with Lamda as regularisation parameters. This did not yield much in the way of better score. But these were the optimal regularisation values $[10^{-2}, 10^{-2}, 10^{-1}, 10^{-3}, 10^{-2}, 10^{-2}, 10^{-3}, 10{-}1, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}]$ for each month respectively.

Here are the weights of each months polynomials:

```
Month:  1
[-0.20712377  0.19835763 -0.0019271 ]
Month:  2
[ 9.49512167e+00 -1.03647739e-01  8.05693729e-04]
Month:  3
[ 1.45881249e+01 -4.97407313e-02  6.11450352e-04]
Month:  4
[ 2.45219035e+01 -2.50045645e-01  5.45296109e-03 -3.30796316e-05]
Month:  5
[ 3.06565954e+01 -1.31108024e-01  7.41205962e-04]
Month:  6
[ 2.52960699e+01 -1.12036543e-02  4.93082233e-05]
Month:  7
[ 2.15124129e+01  2.20971284e-01 -1.22662187e-02  1.86922254e-04
 -8.63832138e-07]
Month:  8
[ 1.29254959e+01 -4.77216132e-02  6.69030407e-04]
Month:  9
[ 5.24741404e+00  2.09949542e-02 -3.24692372e-04]
Month:  10
[ 9.45038704e+00 -5.52301559e-01  9.05154710e-03 -4.40010867e-05]
Month:  11
[-0.15720417  0.03391605 -0.00042752]
Month:  12
[-5.12197419e+00  1.70902379e-01 -1.22195660e-03]
```