ELL409
Assignment 1 Report

-Rajdeep Das
2019MT10718

PART 1 A

L2 error or sum of squares error is the considered error function, unless stated otherwise.

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2.$$

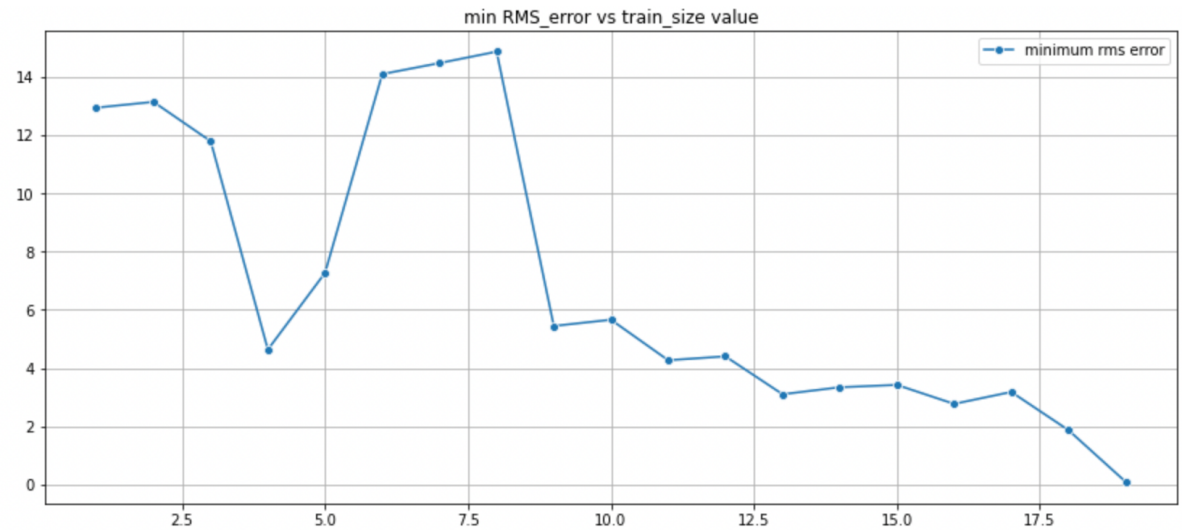
m refers to the number of terms in the polynomial, for degree add 1

-->Moore Penrose Pseudoinverse (pinv)-

20 points dataset

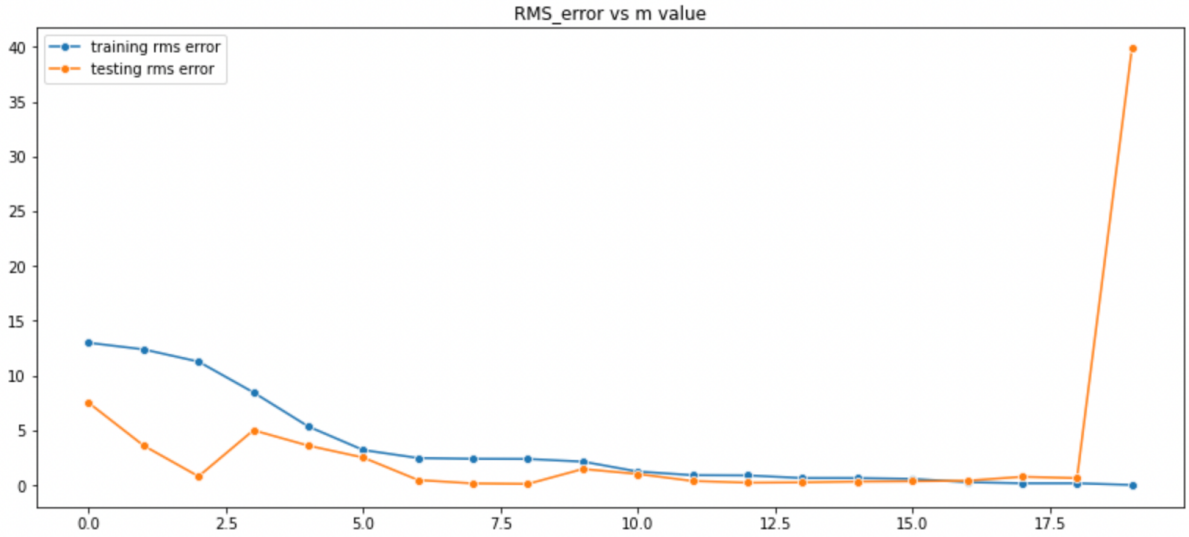
1)Train-validation split:

The training data was split into training and validation data that reduced the RMS error for the validation part. The optimal split was found to be 19 : 1, as can be seen from the graph below.



2) Relation between error and number of terms in polynomial

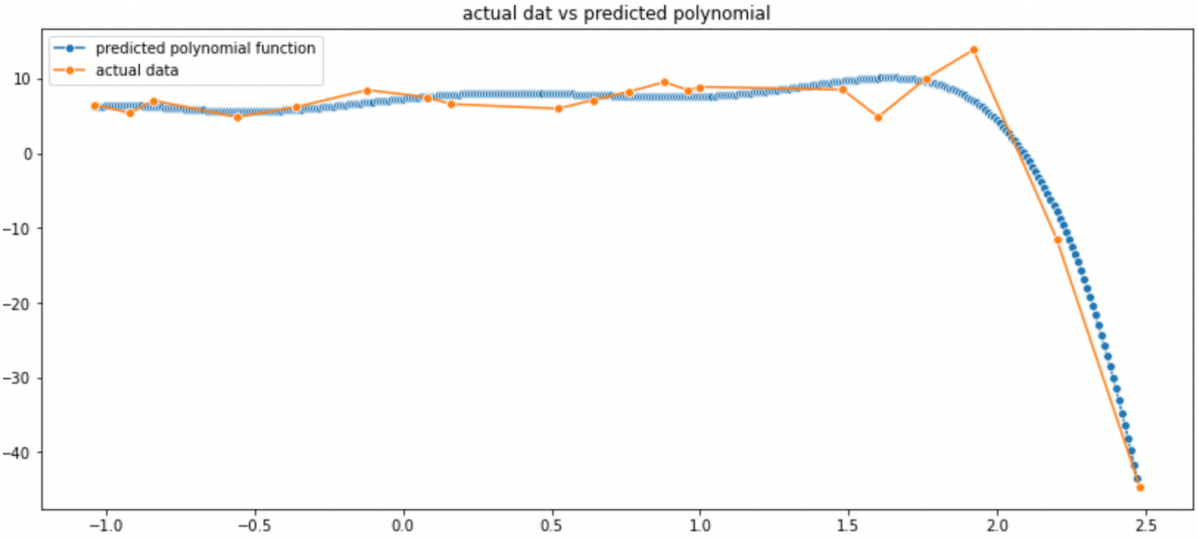
Using the optimal training-validation split the training errors and testing errors were obtained for the respective parts by considering 20 polynomials from degree 0 to 19 by applying least square regression by np.pinv function. The following relationship between training and testing RMS error vs value of m was obtained. It can be seen as m increases predicted polynomial overfits the data causing decrease in training RMS error but a decrease and subsequent increase for the testing RMS error as is expected.



3) Plotting predicted function/Hyperparameter tuning

From the above graph it can be seen that the testing RMS error is the least for m=8 , which is also supplied by the output of the code below.

```
m optimal:  8
err optimal:  0.09644204611377383
wml optimal:  [ 7.30088814  3.95135829 -3.39618027 -6.98691054  6.22455751  3.21412324
 -3.15646528  0.45484283]
```

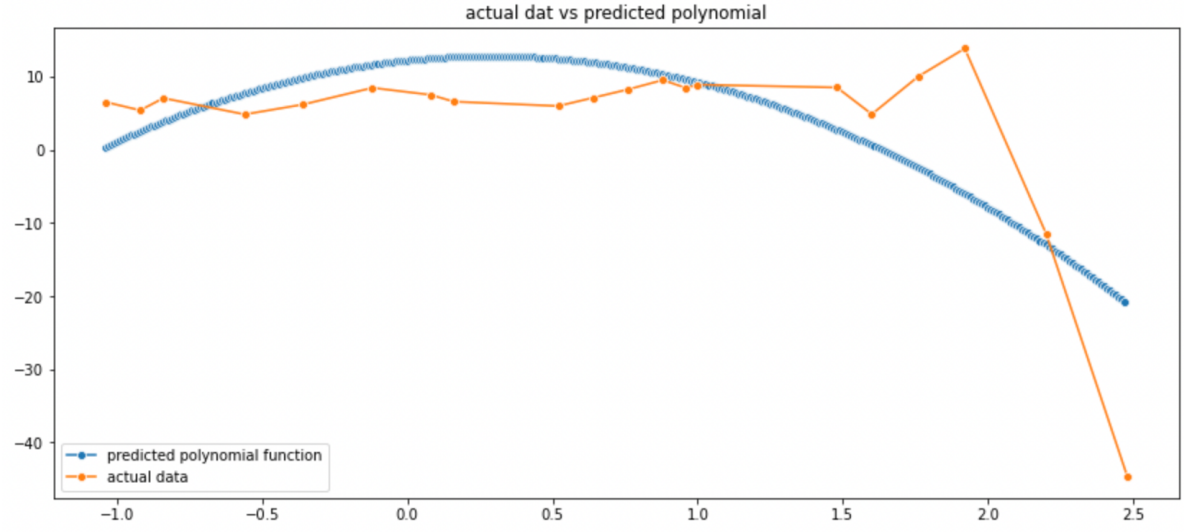


A function was plotted using this data for m=8 which resulted in this:

4)<u>Underfitting and Overfitting cases</u>

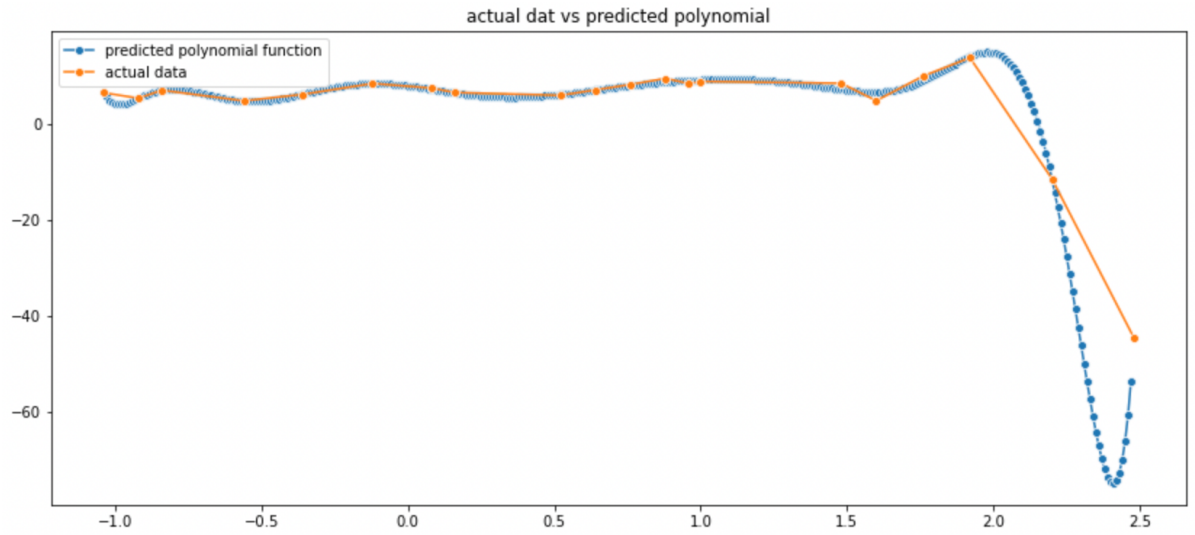
We can change the value of m to determine cases where there is overfitting and underfitting.

For m=3 the function plotted was something like this,



We can see that it doesn't follow the trends of the data intricately at all, so it is a case of underfitting.

For m=13, we get the following map,



We can see that the function follows the data too closely which can lead to sharp inflections as the functions tries to overfit the data completely.

5)Changing error functions:

->L1 loss/MAE:

Mean absolute error or L1 loss

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

On applying L1 loss instead of L2 error in all the steps up till here we find that optimal m we get is m=8,

```
m optimal:  8
err optimal:  0.048221023056886914
wml optimal:  [ 7.30088814  3.95135829 -3.39618027 -6.98691054  6.22455751  3.21412324
 -3.15646528  0.45484283]
```

The optimal value of L1 error being 0.04 which is less than the one given by L2 loss! But this function won't be pursued as we have seen

that it's not differentiable and L2 will give appropriately similar results (wrt. error).

Custom modified L1 loss function

Error Function= $\sqrt{\sum_{i=1}^{n}}$ exp (| y_true – y_predicted |)

Finding the optimal value of m for RMS testing error minimization: The optimal value that we get is for m=8 and modified L1 error being 0.55.

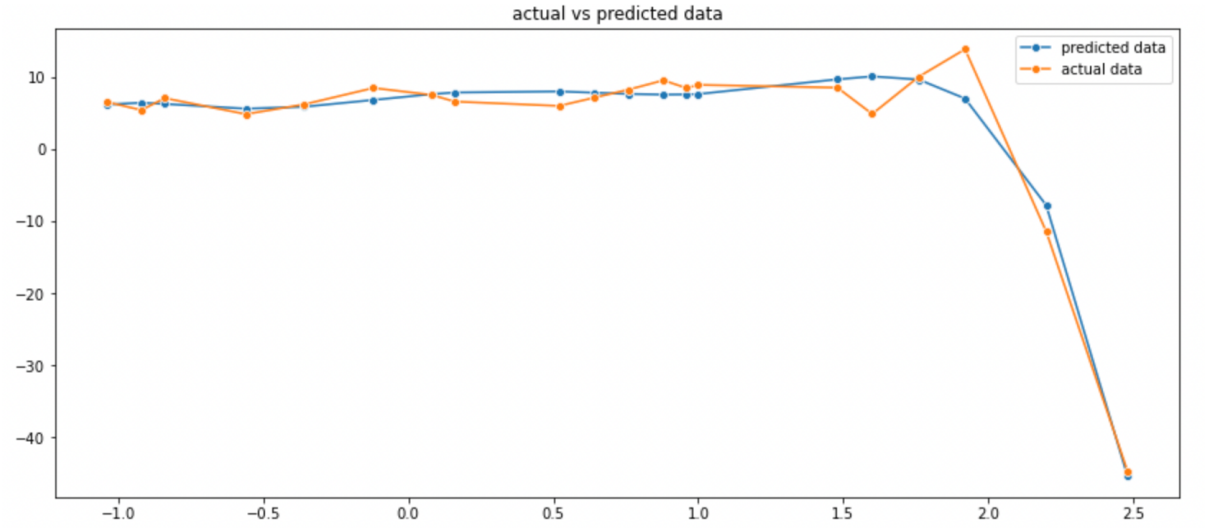
```
m optimal:  8
err optimal:  0.5506228789115257
wml optimal:  [ 7.30088814  3.95135829 -3.39618027 -6.98691054  6.22455751  3.21412324
 -3.15646528  0.45484283]
```

This error comes out to be more than that of L2's RMS error which was 0.09.

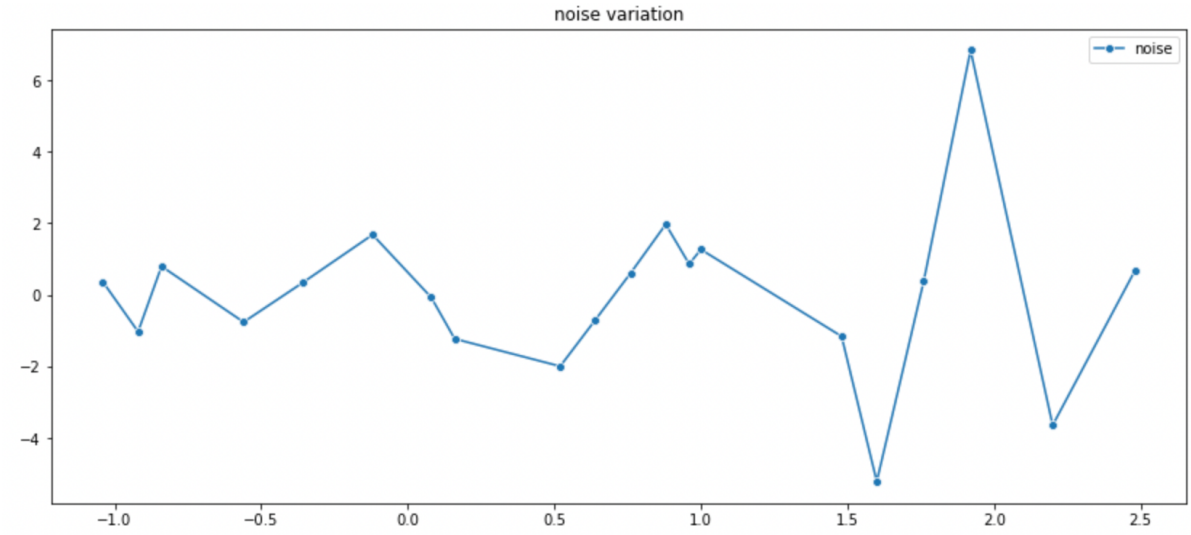
6) Noise variance:

We calculate the noise by first finding out the predictions for the entire training data and then taking the value of its difference with the true data.

The predicted data for 20 data points comes out to be something like this:



The corresponding noise data comes out to be :



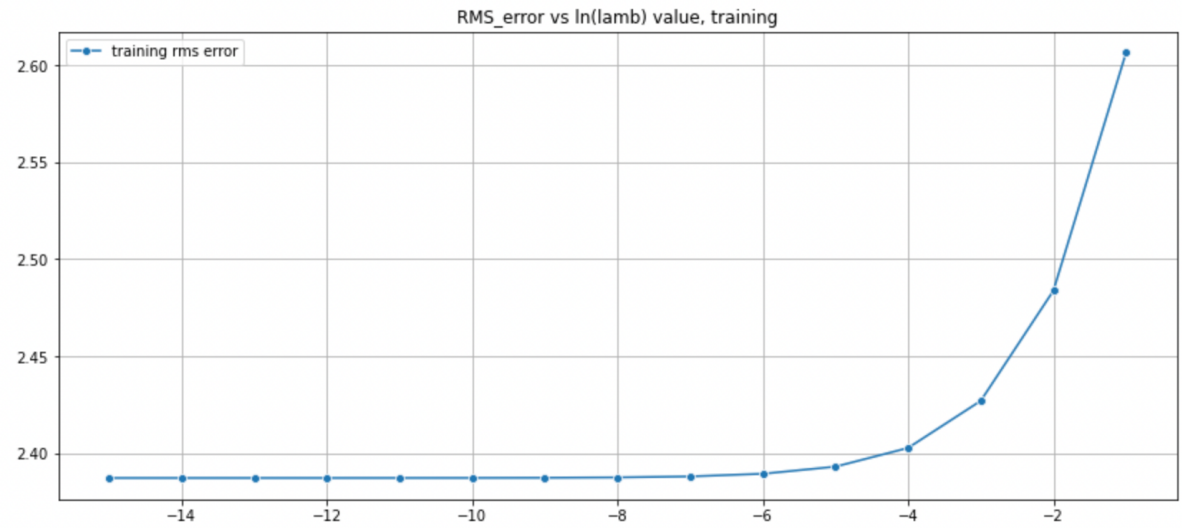

```
mean of noise 1.7821299991283013e-13
variance of noise 5.349538023050263
```

Zero mean noise is received as expected because the given noise was a zero gaussian noise. The variance is 5.34. So the noise can be modelled by N(0,5.34)

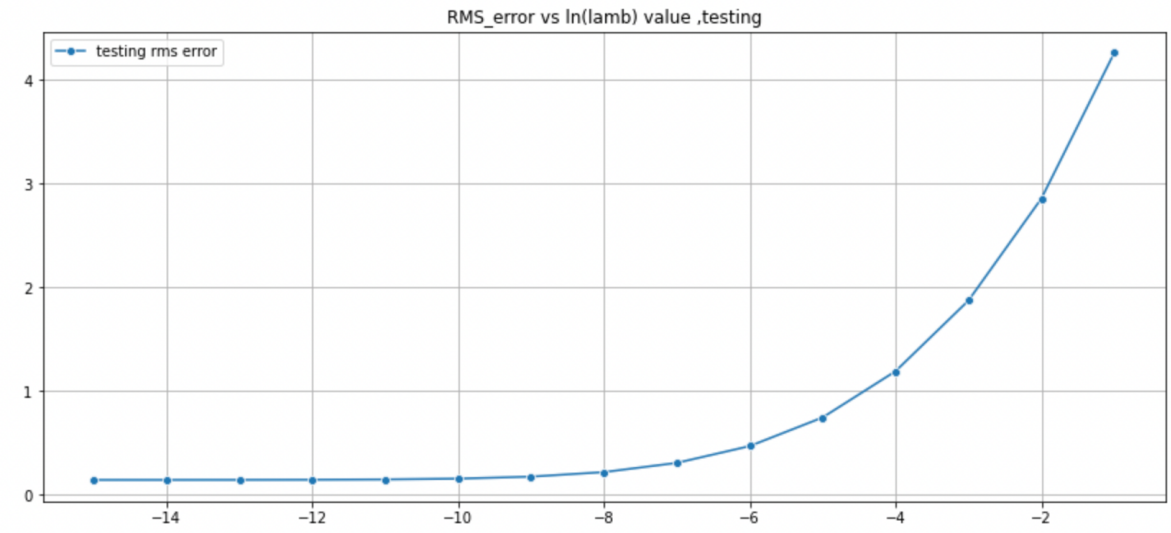
(Higher noise variance with respect to the 100 point data set scenario is probably due to the fact that the 20 point dataset is siphoned off from the original dataset of more than 100 points)

7)Regularization:

Regularization was done by a variable regularization parameter (lambda) and the results of the testing RMS error and training RMS error vs the value of lambda was obtained as shown,



The training RMS error vs lambda gives us the expected trend that and increase in lambda causes underfitting so training error keeps on increasing.



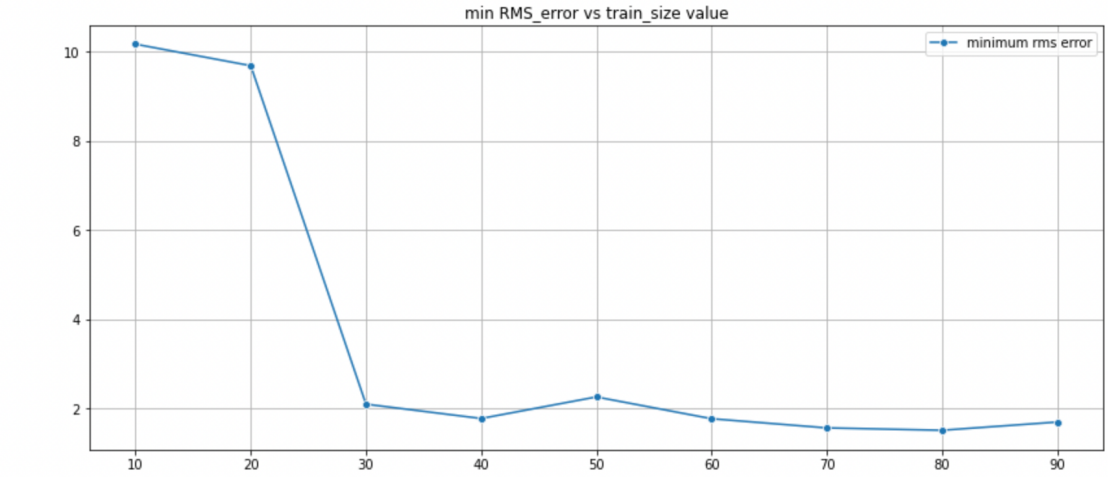
However the trend for testing RMS error vs an unexpected one, this is maybe due to less data or maybe the trend is being followed on a

minute scale, which isn't being captured by the graph. The expected trend is for testing error to first decrease and then simultaneously increase with the value of lambda.

100 points dataset:

1)Train-validation split

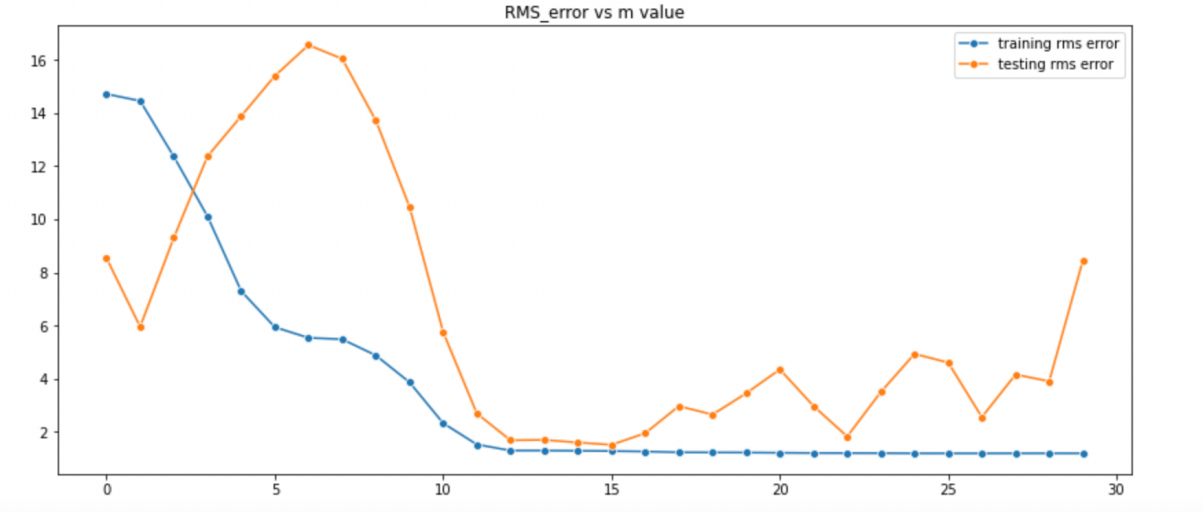
The Moore Penrose pseudoinverse approach was performed for a variable training data size to find the optimal training size to the value that reduces the RMS error for the validation data. It was found that a 80 train- 20 validation split works the best as can be seen from the graph below.



2) Relation between error and number of terms in polynomial/ Hyperparameter tuning

Using the 80 training and 20 validation split the training errors and testing errors were obtained for the respective parts by considering

30 polynomials from degree 0 to 29 by applying least square regression by np.pinv function. The following relationship between training and testing RMS error vs value of m was obtained. It can be seen as m increases predicted polynomial overfits the data causing decrease in training RMS error but a decrease and subsequent increase for the testing RMS error.

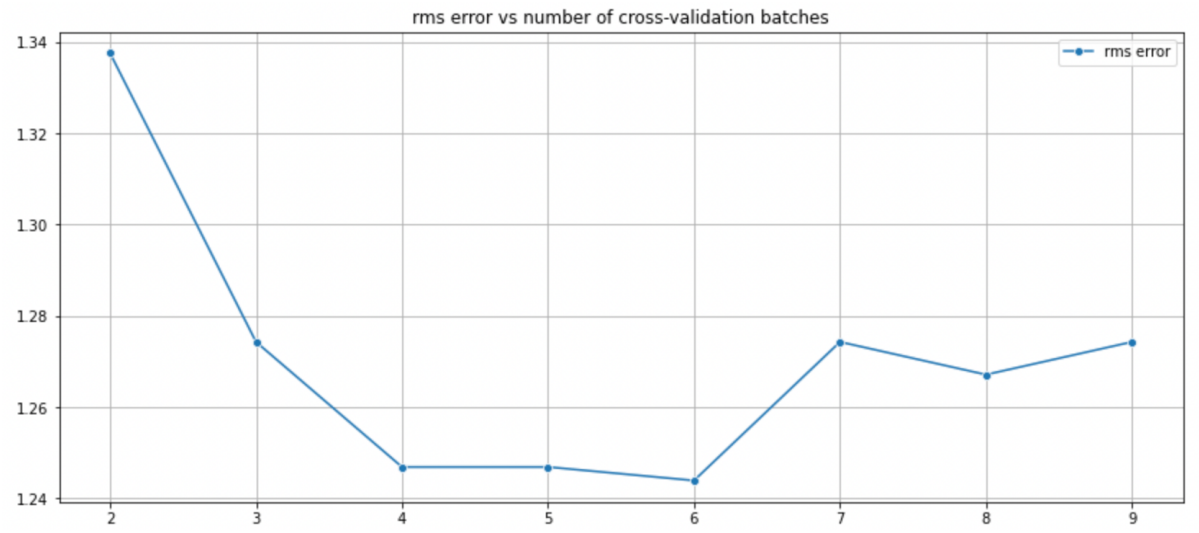


It could be seen that the following inputs were deemed optimal value of m by code and also reinforced by the graph. (testing error minimizer)

```
m optimal:  15
err optimal:  1.508459058172129
wml optimal:  [  6.65450311   0.93815835  -0.84628501 -17.83549222  13.03827056
  59.54215087 -40.39691887 -65.51182515  51.74567057  23.02126479
 -27.32721366   1.82533708   4.64937409  -1.69412197   0.1862886 ]
```

3)Cross-Validation

k-batch cross validation approach was taken to further optimize the regression model and thereby reducing the error. Value of k=6 was found to be best for the dataset as it minimized the testing error

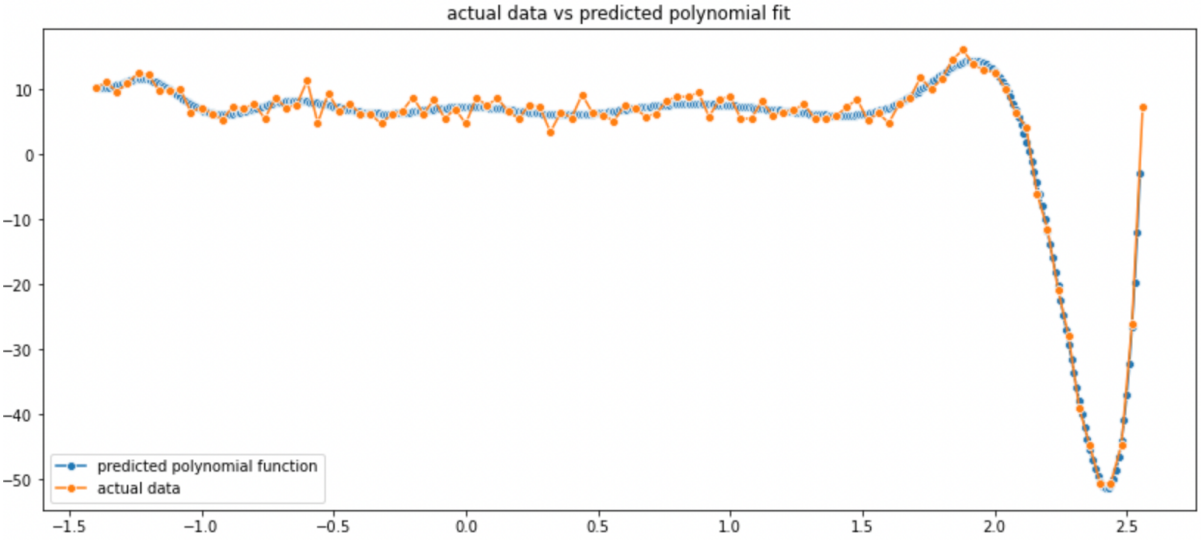


The following data was obtained when the cross-validation was run for k=6

```
m of rounds:  [18, 12, 18, 29, 27, 11]
testing errors of rounds:  [1.6750330559687072, 1.6195072872707088, 1.296224353403048, 1.0016513114509924, 1.18914266
19328618, 1.6749345495433454]
m final 19
rms error for m_final 1.243937337574132
```

It was found that the optimal value of m is 19 and the corresponding error by applying regression fit on entire 100 points of the dataset gives the rms training error as 1.24

4) The W_ml was also obtained from the above calculation and was used to map out the predicted polynomial function and was compared to the original datapoints. X-axis ->ids , y->axis ->values

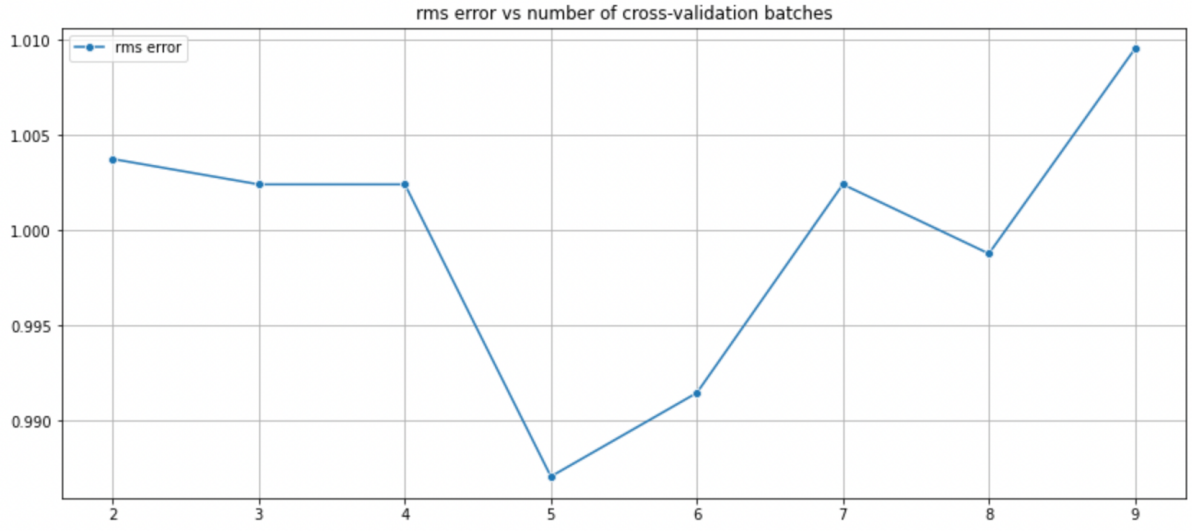


5) Testing out other loss functions:

Mean absolute error or L1 loss

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

On applying L1 loss instead of L2 error in all the steps up till here we find that optimal k=5 for cross validation,



The optimal value that we get is for m=21 and L1 error being 49.15. (The RMS error won't make sense in this case as its only used for L2 error to make the dimensions of the data and error similar.)

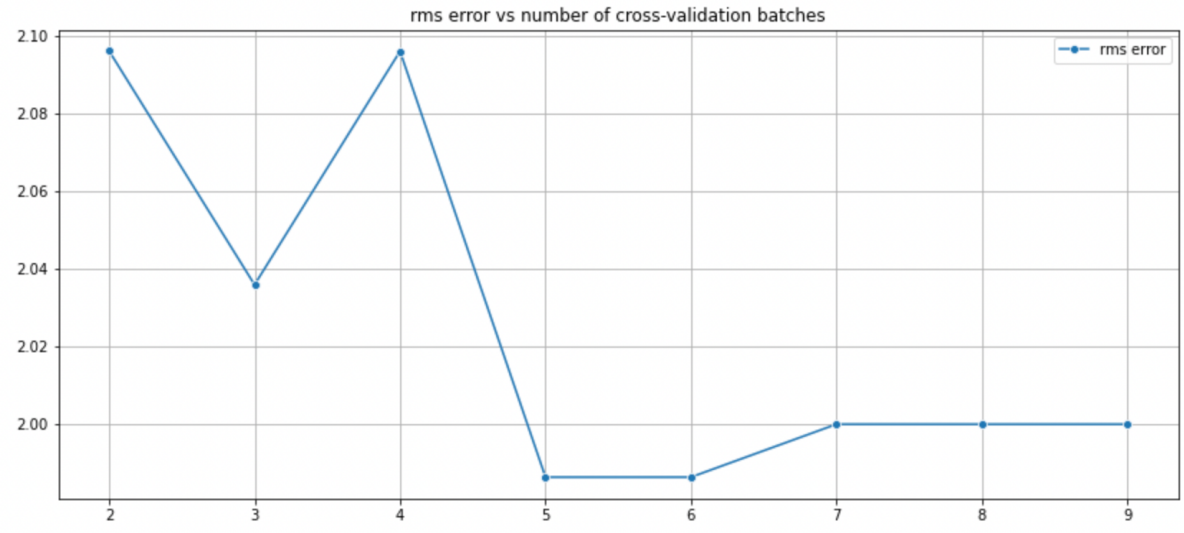
```
m of rounds:  [18, 26, 18, 29, 27, 11]
testing errors of rounds:  [11.200513739563366, 10.706783952265308, 7.251268117257474, 6.414856634098303, 7.015234909
482189, 10.598058858129903]
m final 21
error for m_final 49.15018561814749
```

This is way higher than RMS error for L2 loss.

Custom modified L1 loss function

Error Function= $\sum_{i=1}^{n}$ exp (| y_true – y_predicted |)

Finding the optimal value of k for cross validation:



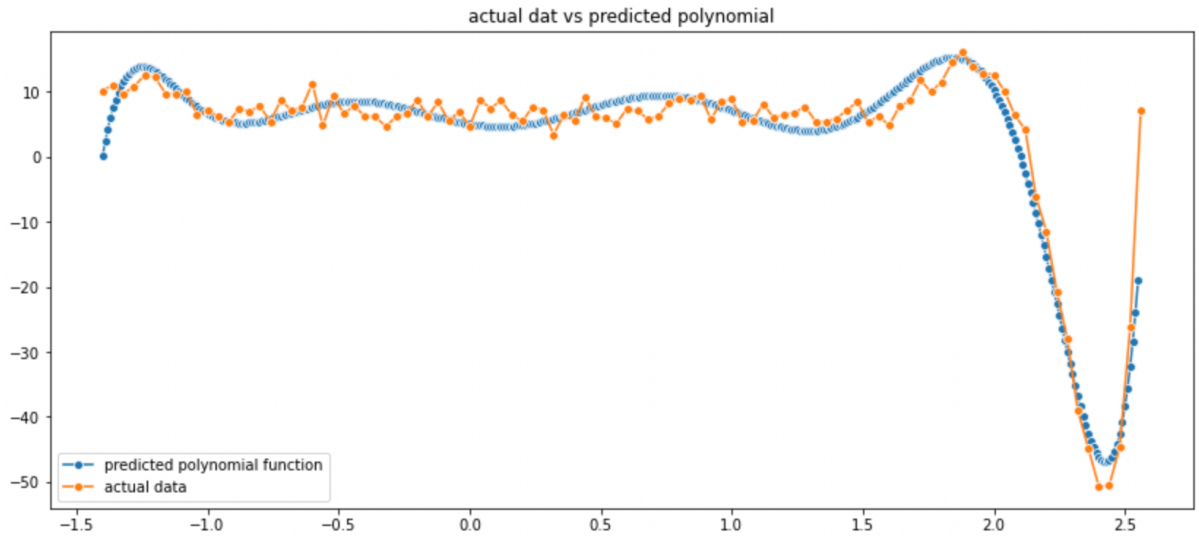
The optimal value that we get is for m=17 and modified L1 error being 5.28. (Log was taken of the actual error to match the dimensions , as RMS is taken for L2 error)

```
m of rounds:  [12, 12, 12, 29, 27, 11]
testing errors of rounds:  [2.4906291575105435, 2.3832426823947417, 2.1067520166937492, 1.6320560174951806, 1.8566448
788459955, 2.49930289172734]
m final 17
error for m_final 5.284489156701655
```

Even this error is more than that of L2's RMS error.

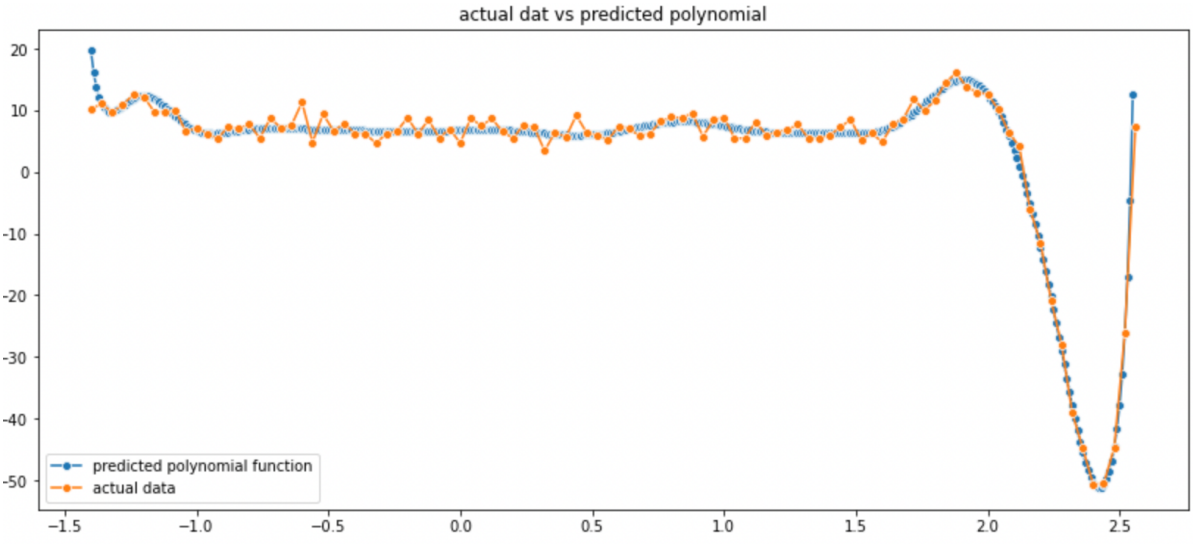
6) Cases of underfitting and overfitting:

If we had taken m=11 instead then the predicted polynomial would have been:

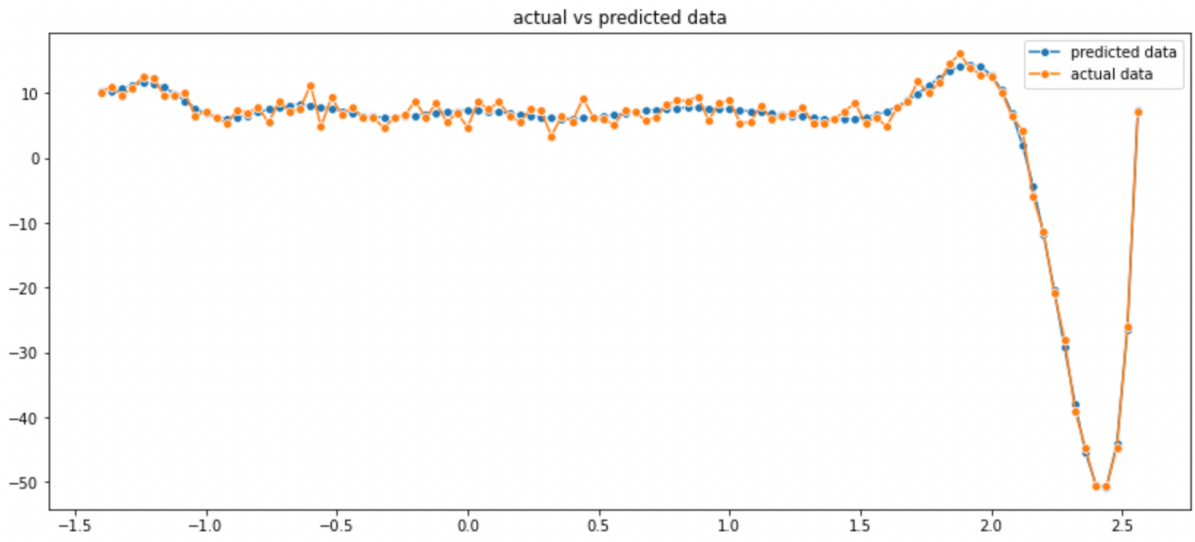


It can be seen that it weakly captures the data trend but still bypasses many fluctuations-> underfitted data

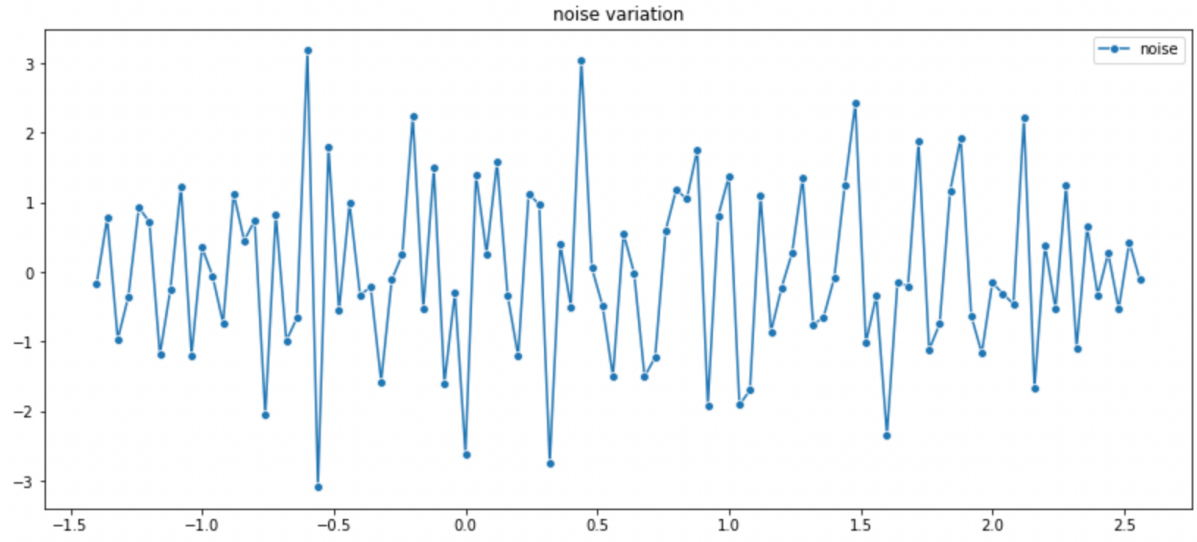
If m was 30 instead then we see that the predicted polynomial is overfitting the training data causing heavy fluctuations at the ends, due to the high degree of the predicted polynomial.



7) Now in order to find noise, the predicted values were calculated based on the predicted polynomial and the value of the difference was taken with the given data points. X-axis ->ids , y->axis ->values.



Corresponding noise distribution came out to be:

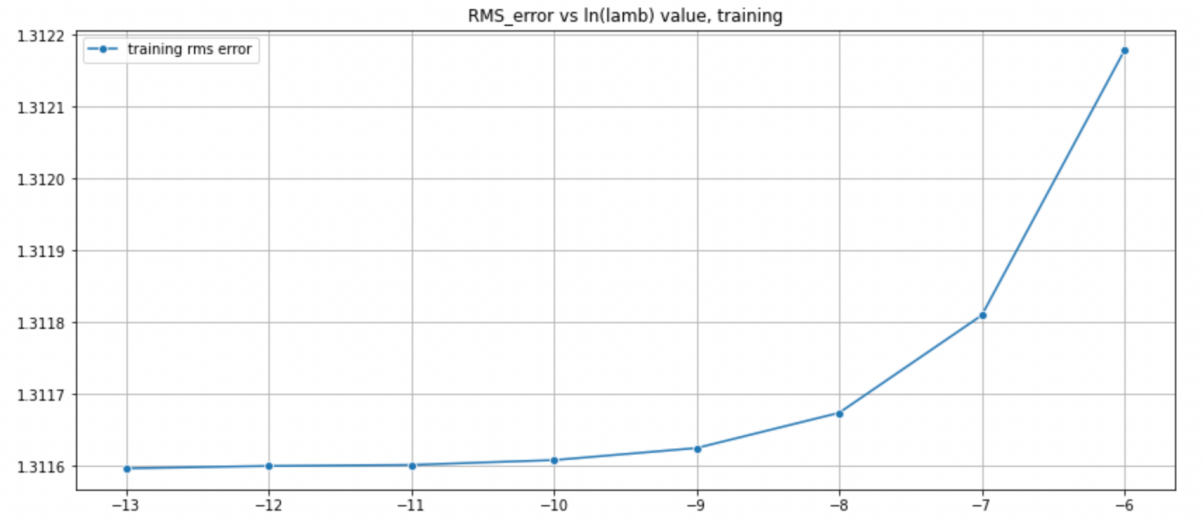



```
mean of noise 4.687205765740998e-09
variance of noise 1.5473800998110199
```

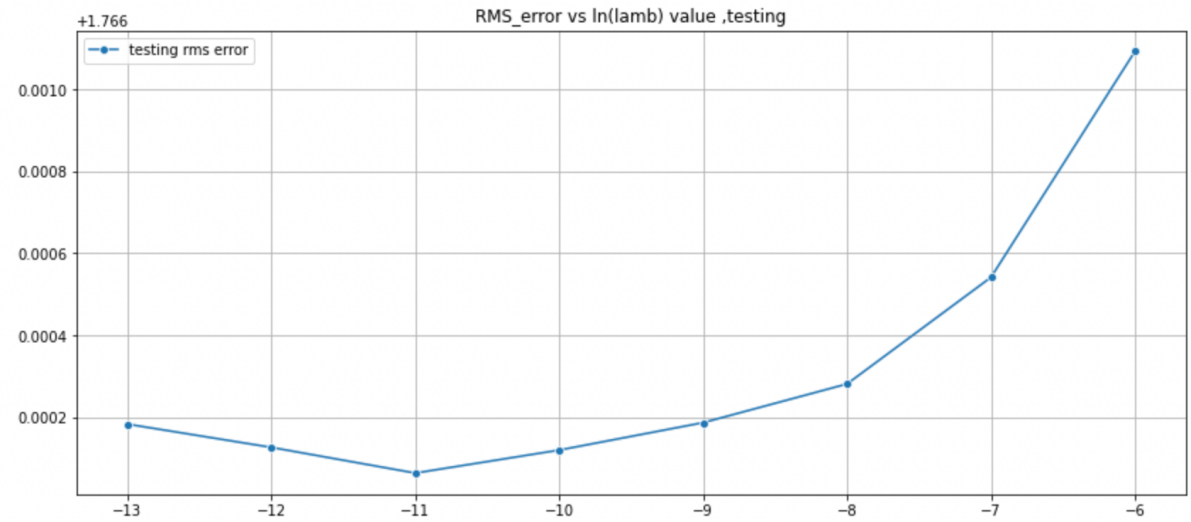
We see that the mean we get is quite close to 0 so the noise can me approximated to zero mean gaussian noise as was given.
The variance we get is 1.54 so the noise can be approximated by N(0,1.54).

8) Regularization performed:

To find the trend of error vs value of regularization parameter(lambda) the map of RMS error was made vs lambda on the x-axis. It was done separately for training and testing data because of very minute differences which would be indistinguishable otherwise.



We can see that as lambda increases, training error increase which is true as the model underfits the data, due to high regularization.



We can see that as lambda increases the testing error value drops to a minimum and bounces back as per the expected trend. At

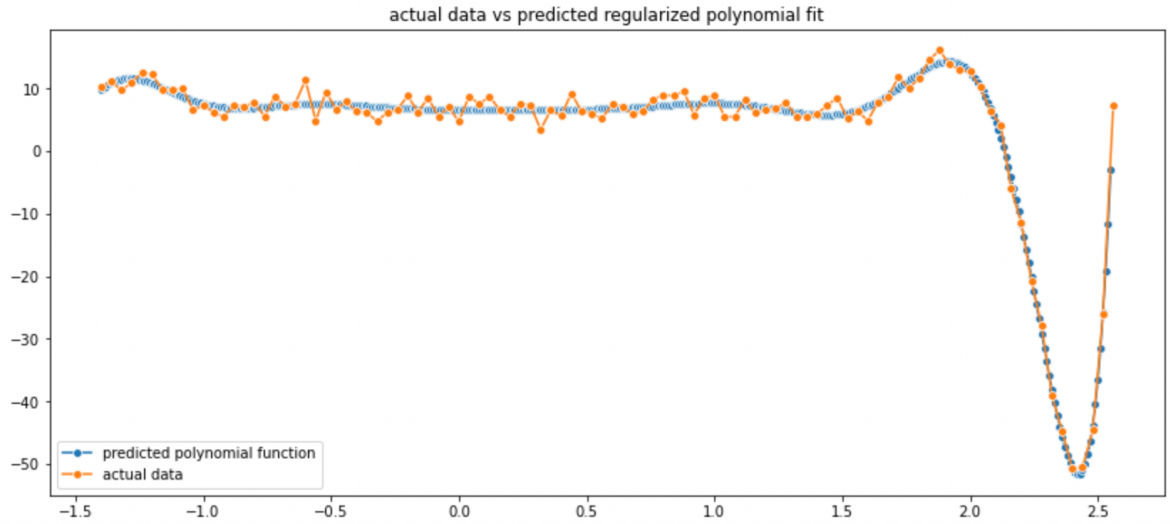
lambda=-11 we find the sweet spot between overfitting and underfitting.

9) On finding the optimal lambda=-11, we use this regularization parameter and apply cross-validation (k=6) on this. The following optimal value of m is obtained.

```
m of rounds:  [17, 17, 12, 14, 17, 11]
testing errors of rounds:  [1.676879007995895, 1.6125639246259023, 1.3560034801408, 1.133891113487614, 1.249265060741
2907, 1.6773732982842087]
m final 14
rms error for m_final 1.3155627710363553
```

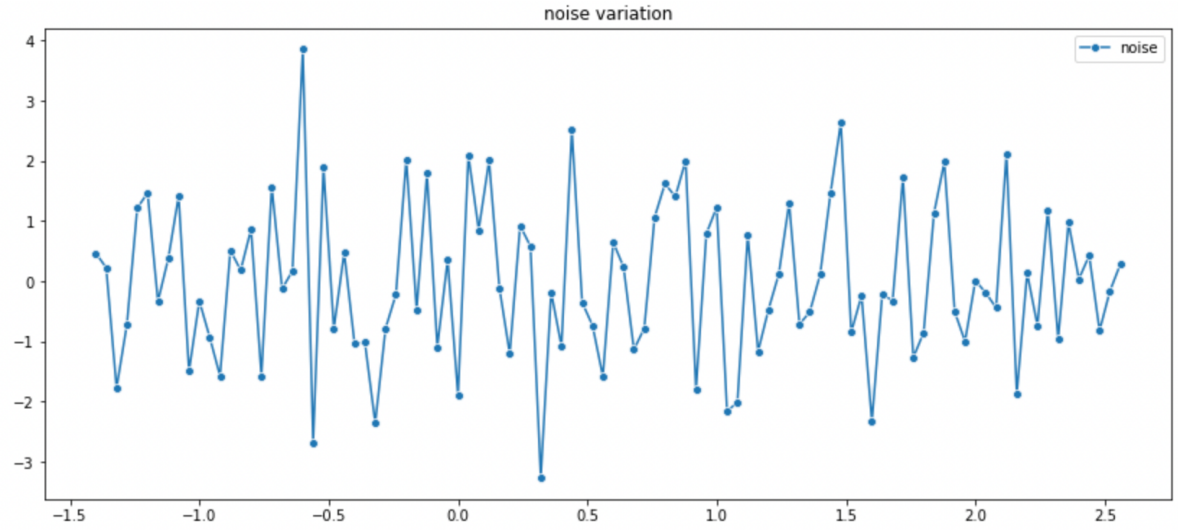
Optimal m that minimizes rms testing error is 14 and the corresponding value of training error on the entire data set is 1.31.

10) We find the polynomial fit for m=14 and we get the following result.(Looks quite similar to non-regularized but has intricate differences)



11)Noise Estimation

On applying the same steps as before we find that the noise mean is again very close to 0 .Variance has risen because of reducing some of the overfitting that was happening before. Again this noise can be modelled by N(0,1.73)



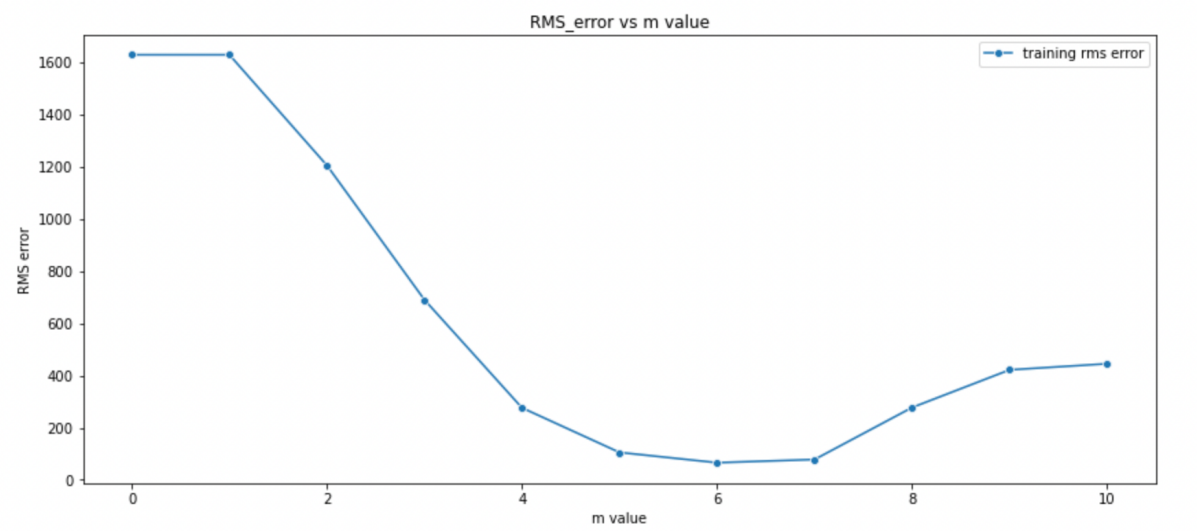

```
mean of noise -2.201964287973368e-06
variance of noise 1.730049687848573
```

→Gradient Descent:

20 points dataset

1)polynomial order vs learning rate trade-off:

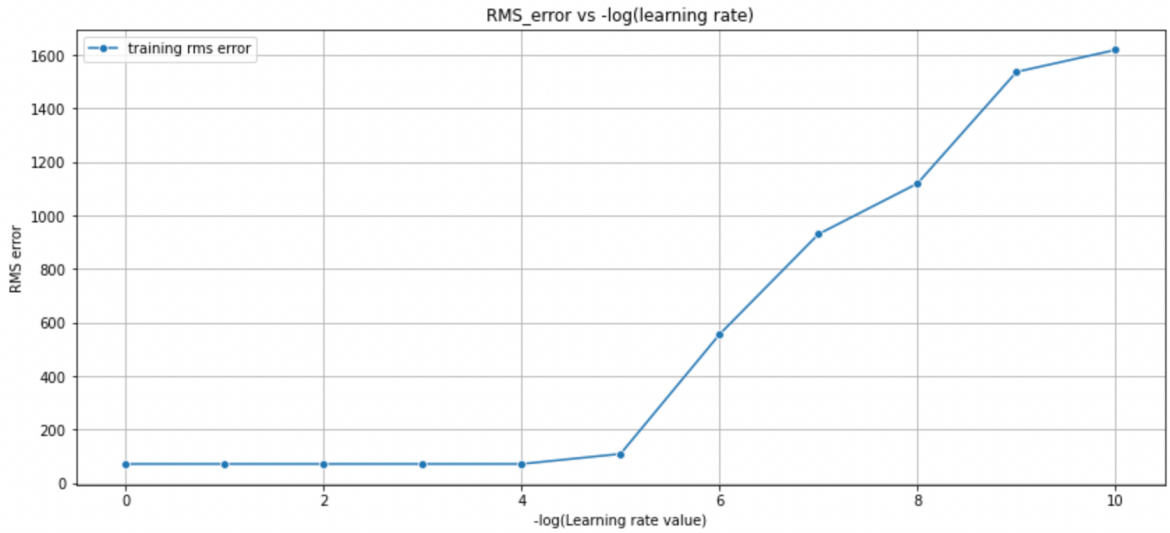
Increasing the order of the polynomial to be fitted should theoretically decrease the training error, but for a finite number of iterations, the learning rate must be decreased for a higher degree so that the calculations do not diverge to inf. Due to this most of the results have been taken for training data as it gets difficult to fit the data so in the case of underfitting both training and testing data will have same trends.
A plot for training error was made against m to find the optimal value for 10k iterations.



It was found that m=6 had the highest efficiency and was used for further optimizations.

2)Now the learning rate was optimized for m=6 by considering 10^-x for first 10 natural numbers as values of x.

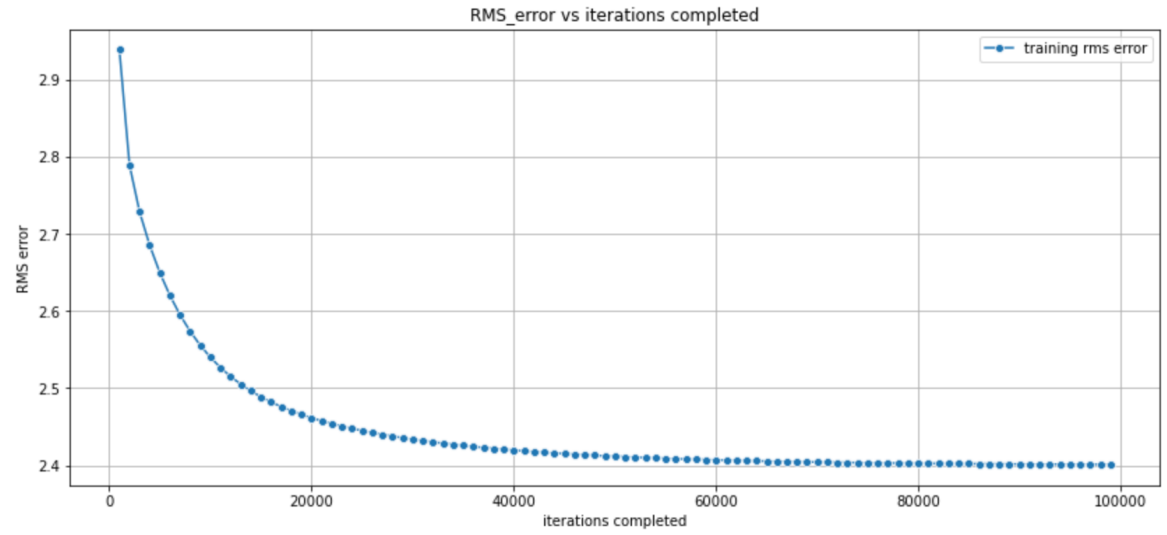
```
3
lr 0.0001
71.74996692923972
--- 1.8919708728790283 seconds ---
4
71.74829288605358
--- 1.9175097942352295 seconds ---
5
109.34719854008259
--- 1.923264980316162 seconds ---
6
555.883817701473
--- 1.9017610549926758 seconds ---
```



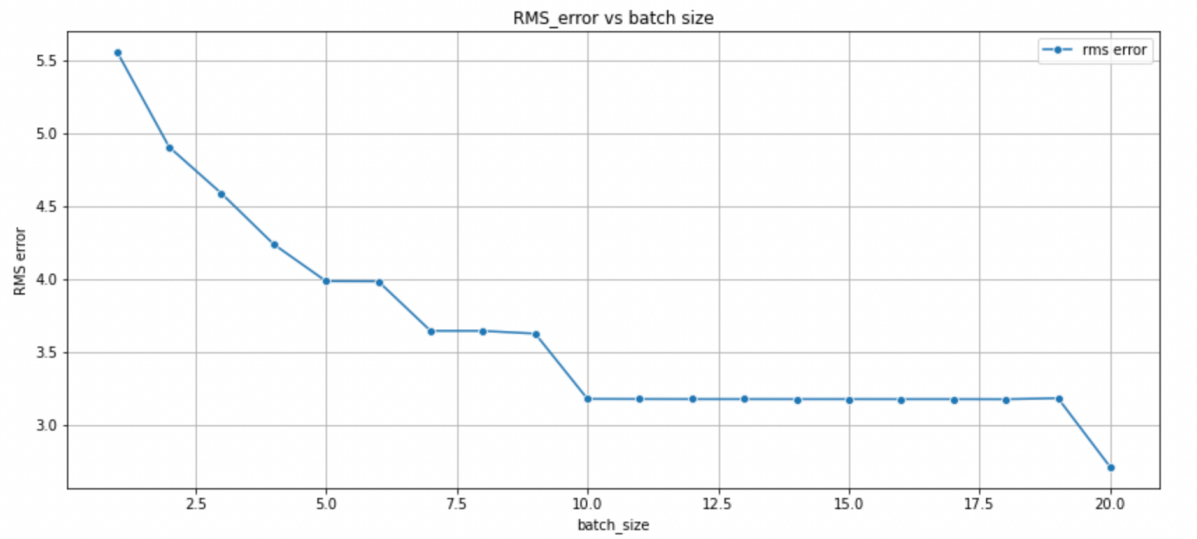
It was found that learning rate of 1e-4 was the best . (The learning rate was dynamically set i.e. if the calculations diverged then the leaning rate would drop itself. This caused the initial part of the graph to look flat as all the learning rates dropped to 1e-4 to avoid infinite blowups)

A manual check revealed that learning rate of 1.2E-4 worked too without depreciating to 1E-5.

3)Error vs number of iterations
The expected trend for this graph is for error to decrease as iterations increase as the model tries to accommodate for more data. That is what was seen.



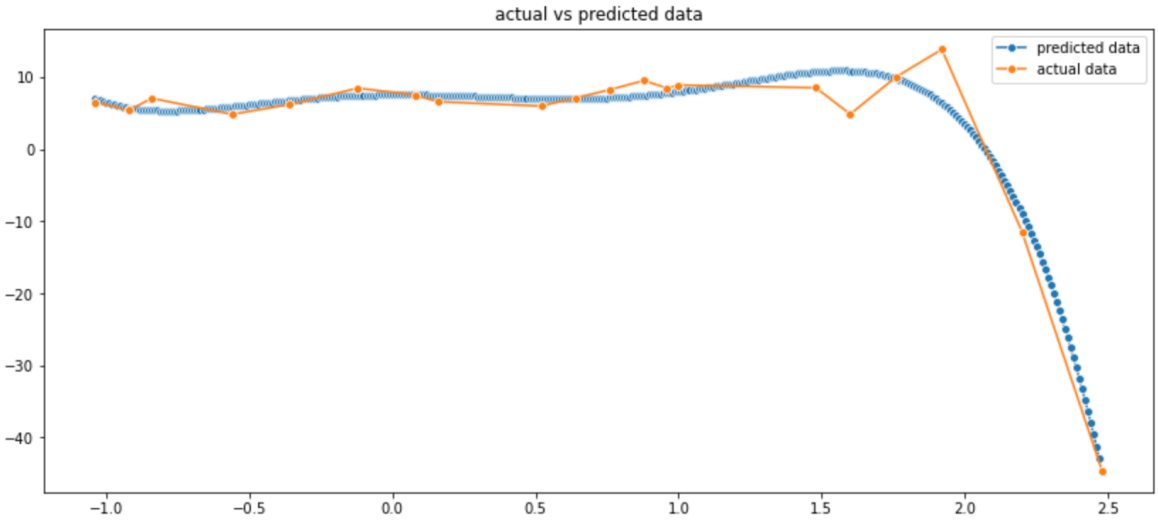
4)Batch Implement
Various batch sizes were tried out to find the optimal size.



A batch size of 20(full batch) was found to be optimal to reduce error.

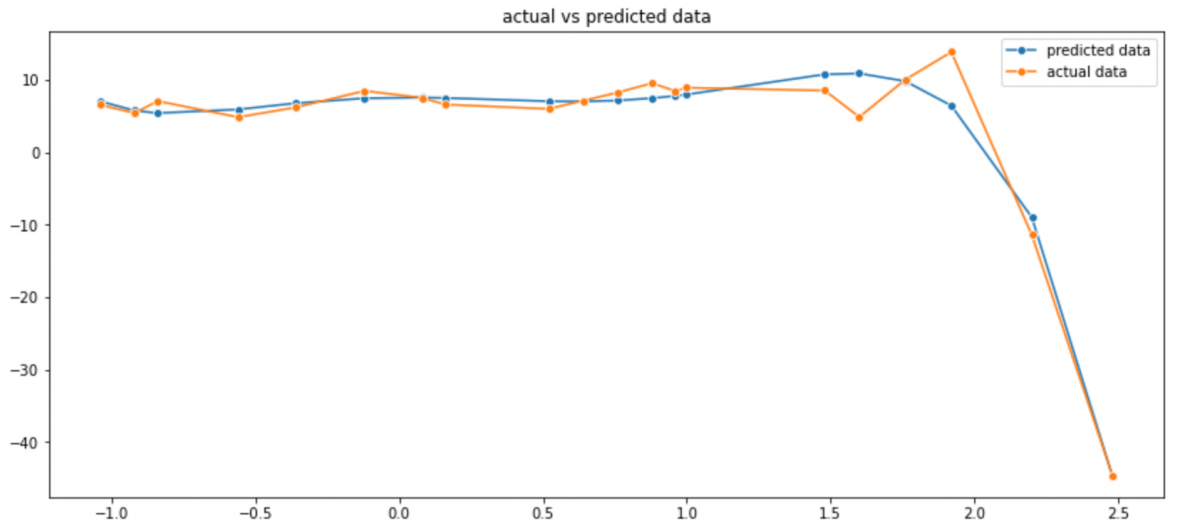
```
min error at batch size 20 error 2.7107640902793366
```

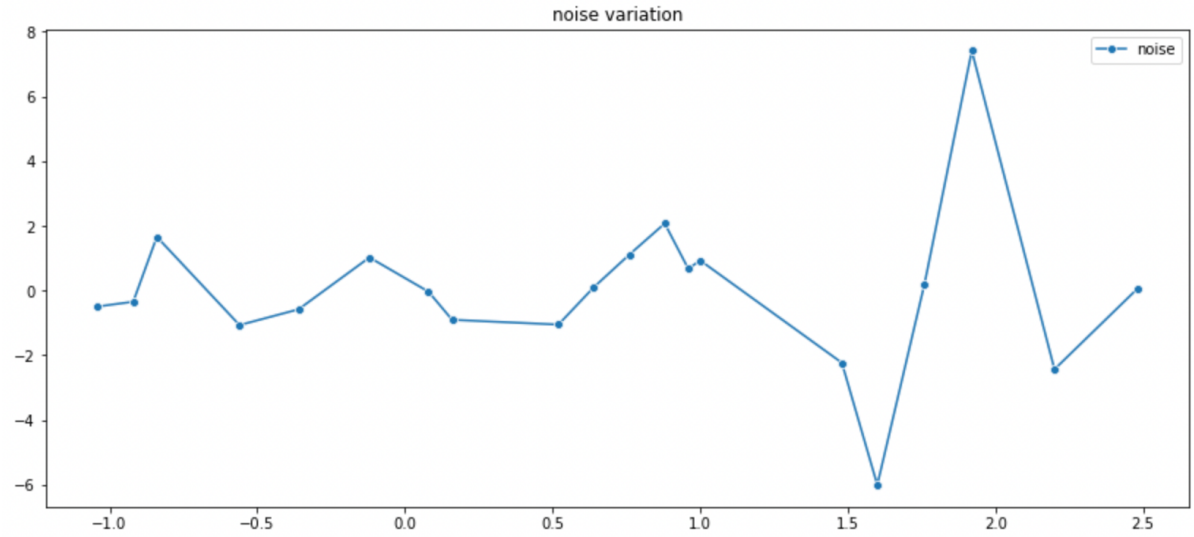
5)Polynomial prediction
Using all the optimization data a polynomial was fit using 50k iterations. (A bit underfit due to time and learning rate restraints)



6)Noise prediction

The w_ml that we get from the batch gradient descent was used to predict data and calculate noise.



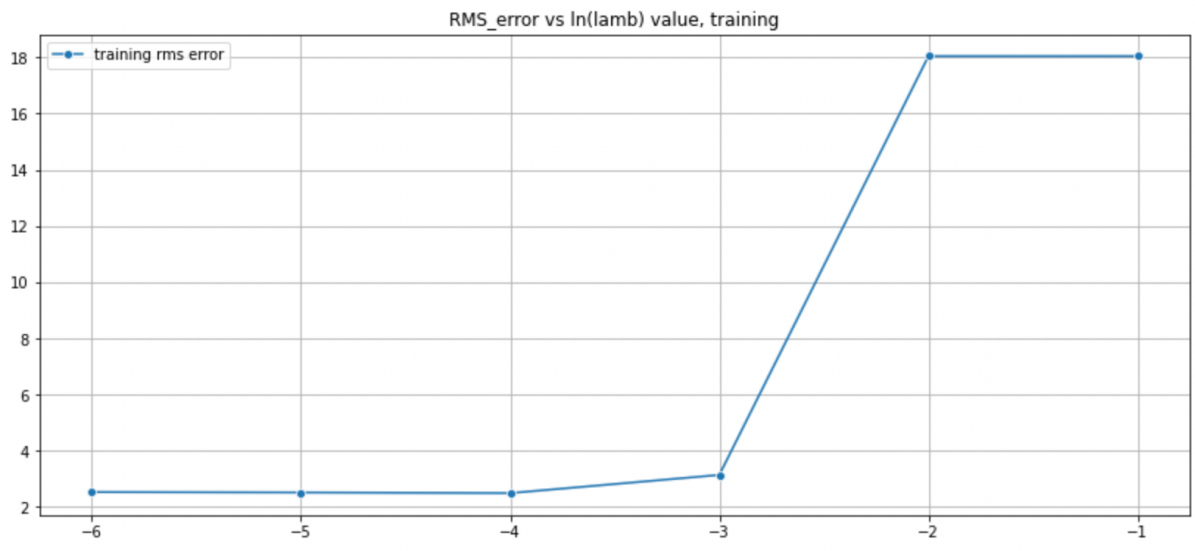



```
mean of noise 0.0031377949076924596
variance of noise 5.815658571954047
```

The noise comes to be about 0, which should be the case given that the noise is a zero mean gaussian.

7)Error vs lambda

The RMS training error was plotted against lambda(regularization parameter) to check the trend between them.



This shows error increasing as lambda increases due to underfitting.

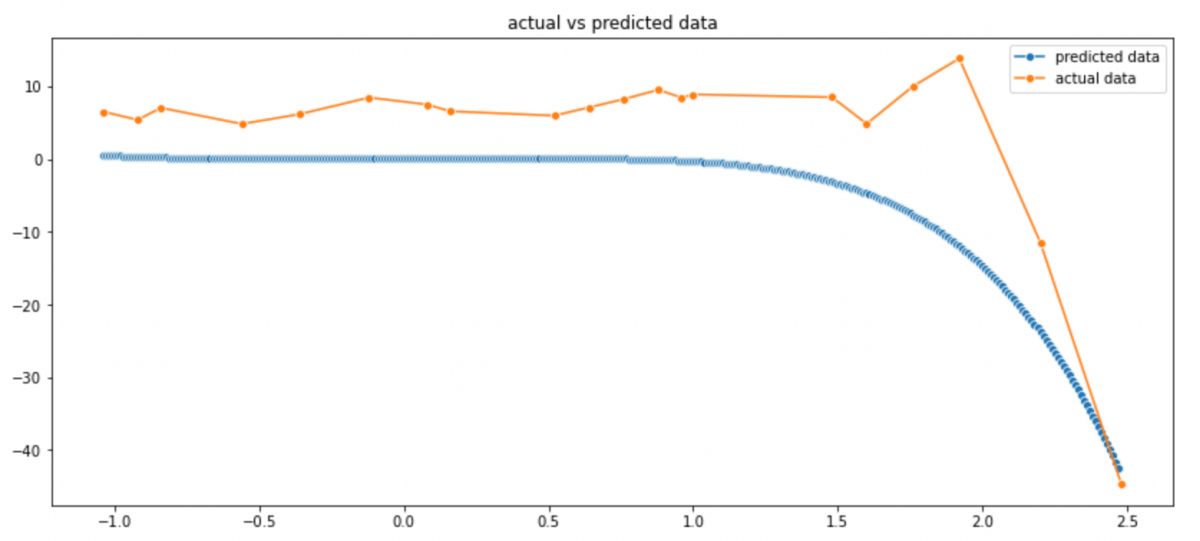
8)Changing loss function:

If we use L1 loss then we get a bigger loss than if we use the L2 loss, which is worse off.
Using the same steps as used above we get the following results for the loss.

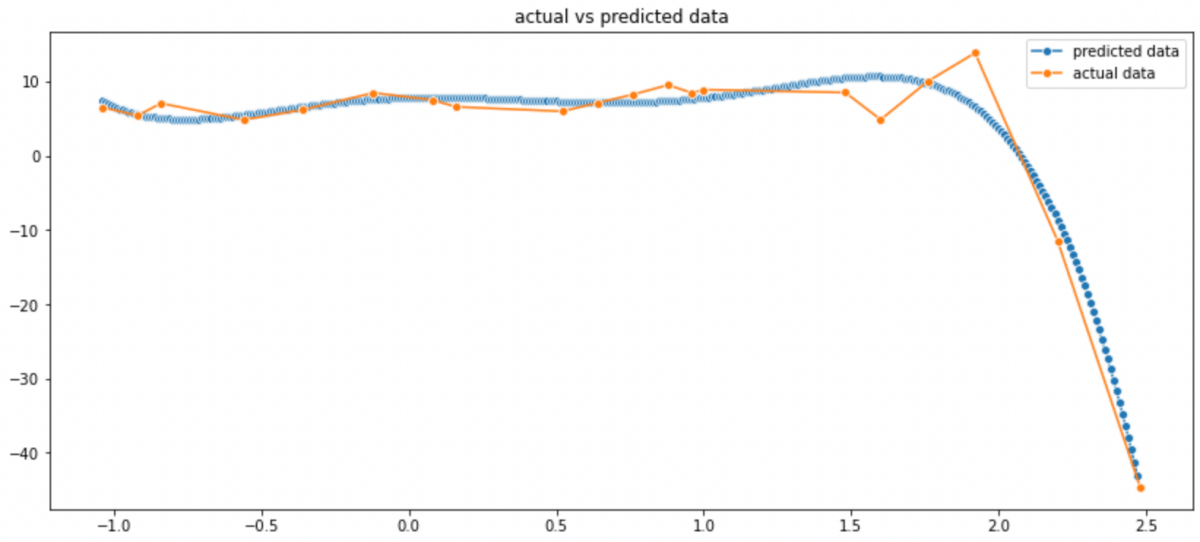
```
min error at batch size 20 error 19.46804035341976
```

9)overfitting and underfitting
Fitting in this case was controlled by iterations

Underfitted at 5 iterations:



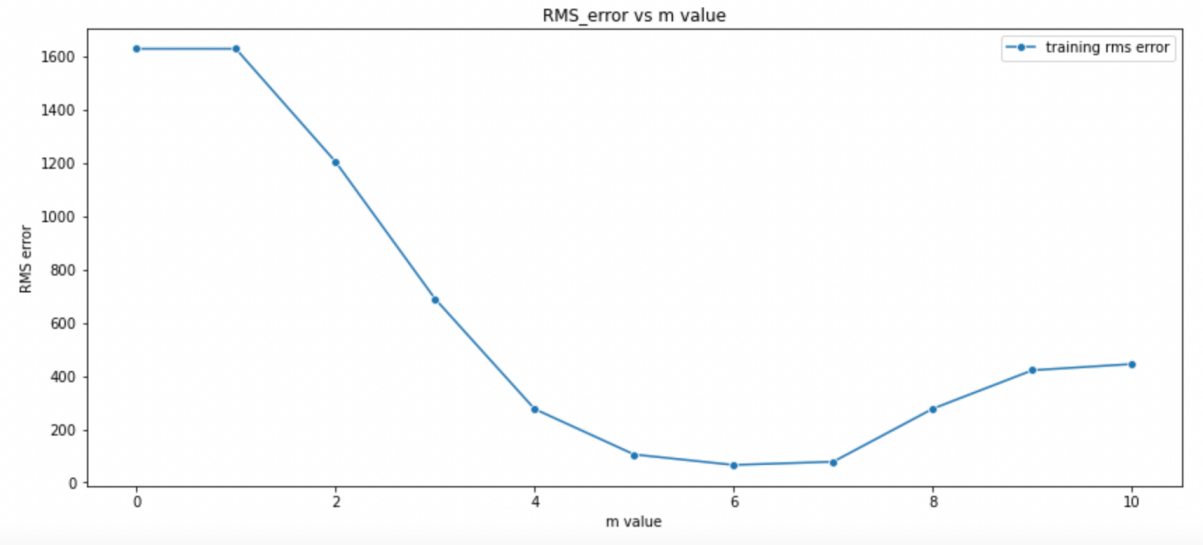
Overfitted at 500000 iterations(not yet completely overfitted due to lesser eta) :



<u>100 points dataset:</u>

1)Polynomial order vs learning rate tradeoff:

Similar as before we perform a graphical analysis for the order of the polynomial best suited for minimum error for 10k iterations , due to learning rate decreasing as order increases.

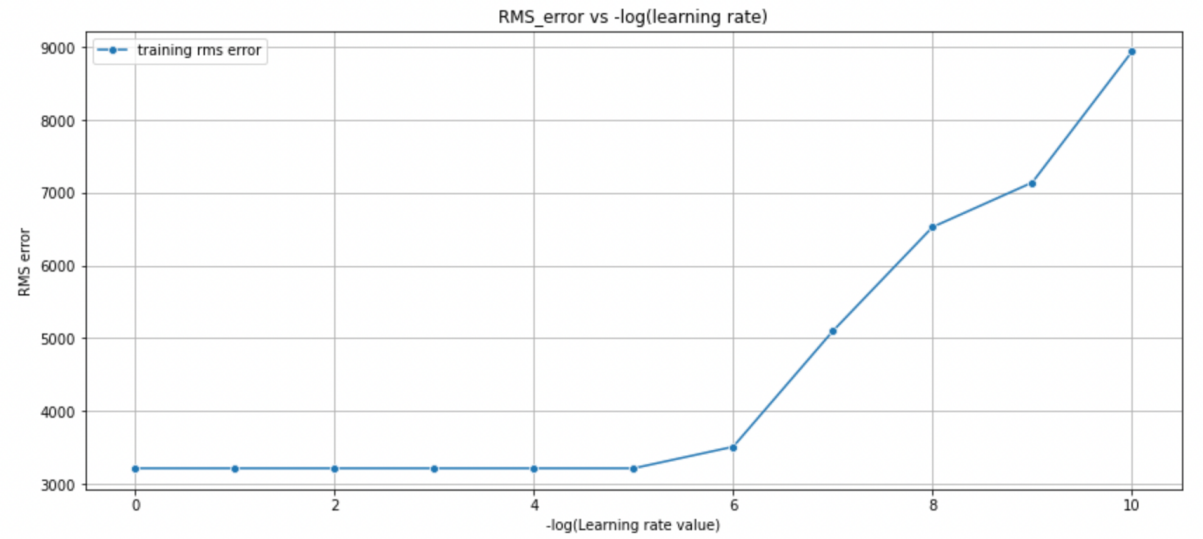


We get m=6 as the optimal fit.

2)Optimizing learning rate

We find that 1e-5 is the optimal learning rate.

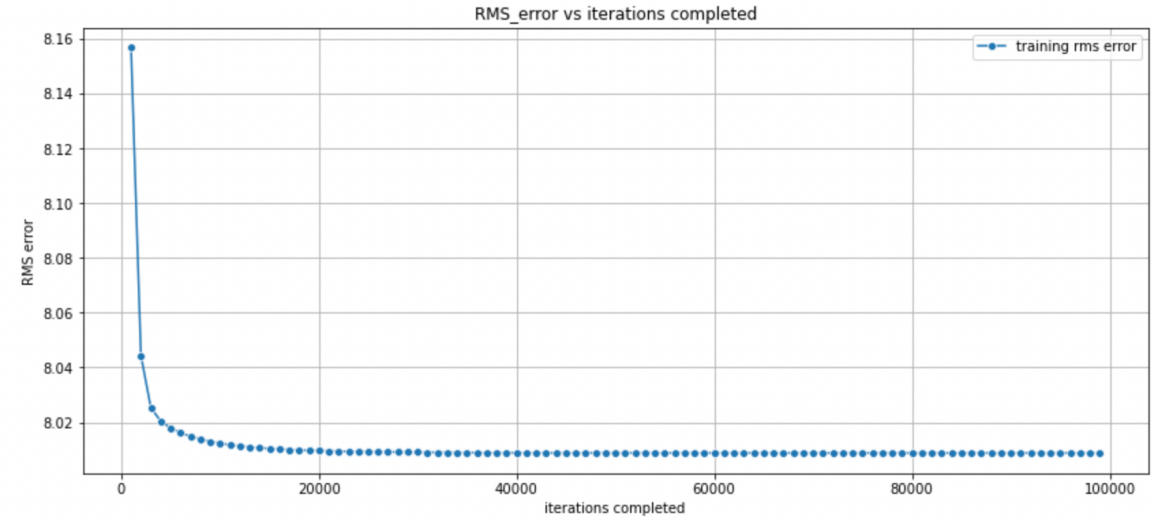
```
4
lr 1e-05
3214.778051004277
--- 19.324267148971558 seconds ---
5
3214.777221303952
--- 19.324858903884888 seconds ---
6
3510.6864146891985
--- 19.271053075790405 seconds ---
7
5099.8077775619695
--- 19.18195915222168 seconds ---
```



(The learning rate was dynamically set i.e. if the calculations diverged then the leaning rate would drop itself. This caused the initial part of the graph to look flat as all the learning rates dropped to 1e-5 to avoid infinite blowups).

It was manually seen that even a learning rate of 2.1e-5 worked well without dropping off to 1e-6 dynamically.

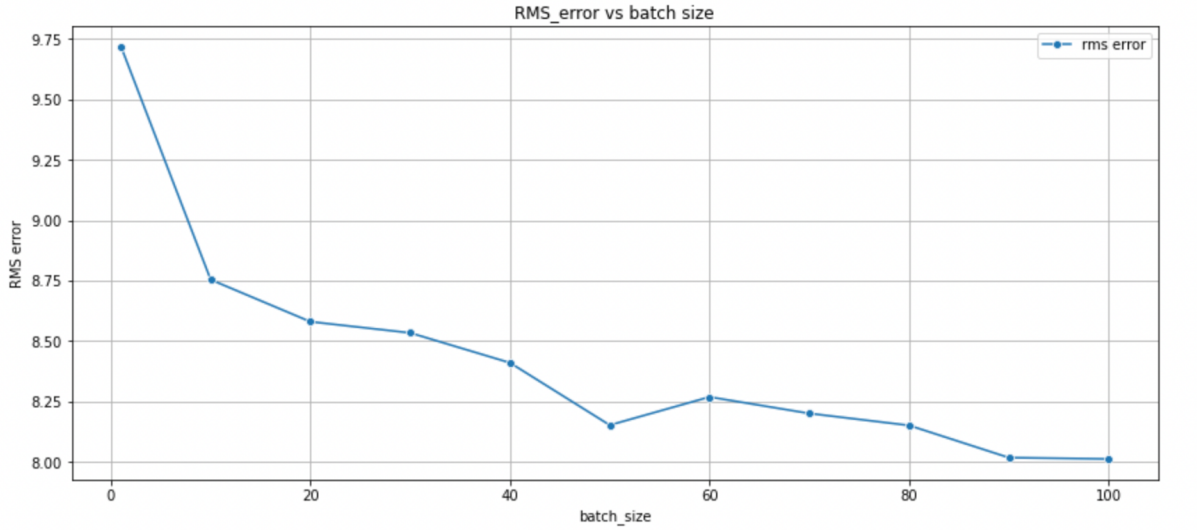
3)Error vs number of iterations



The data shows that error decreases on more iterations as expected as the weights get fine tuned.

5) Relation of error with batch size:

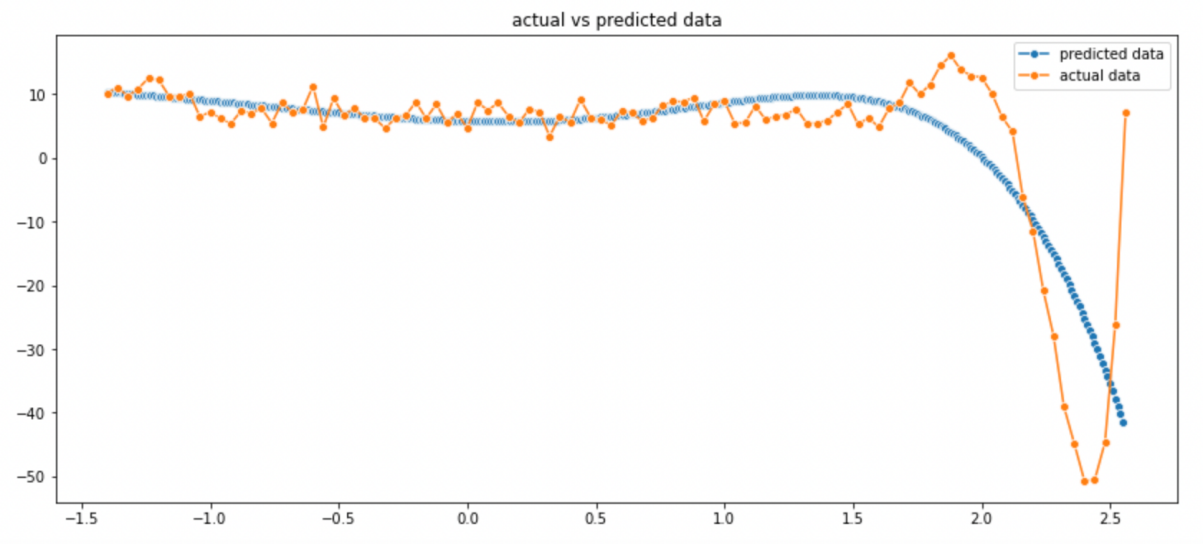
```
min error at batch size 100 error 8.012364533915399
```



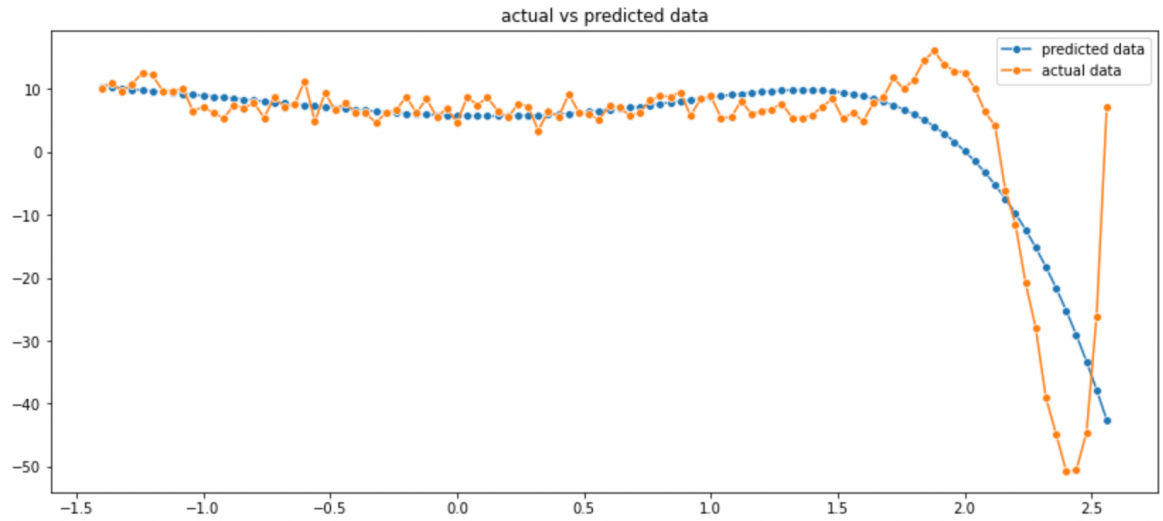
We see that the minimum training error is obtained when batch size is 100 (i.e. full batch gradient descent)

6)Polynomial approximation:

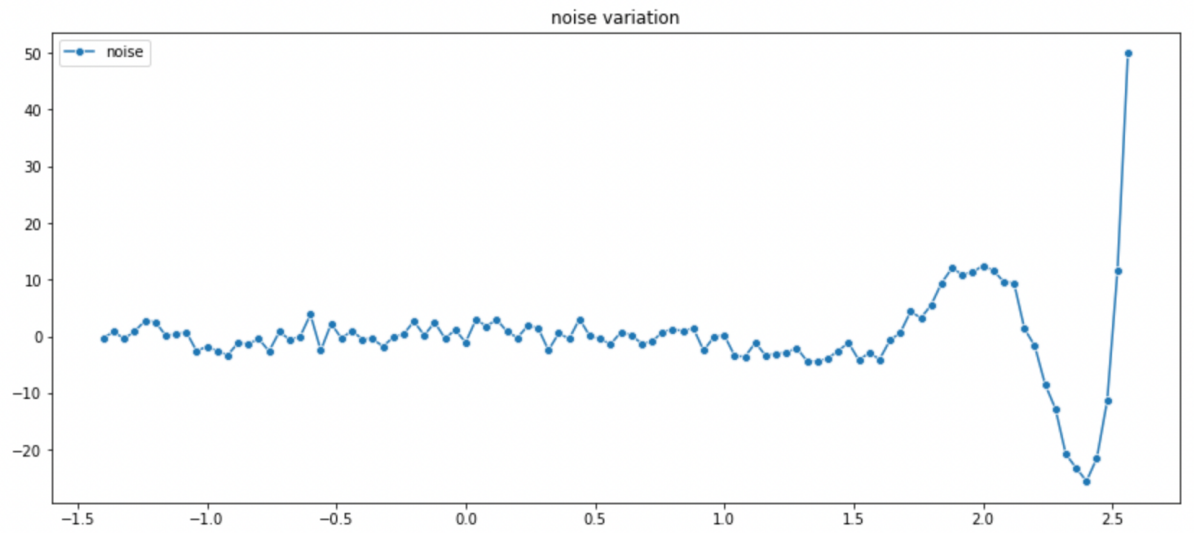
Using the w_ml obtained after 500k iterations a predicted polynomial was mapped and looked like the following. Bad approximation achieved due to large dataset that needs higher polynomial degree for a better mapping than 6, which is troublesome due to the learning rate vs degree trade-off.



7)Noise variation:



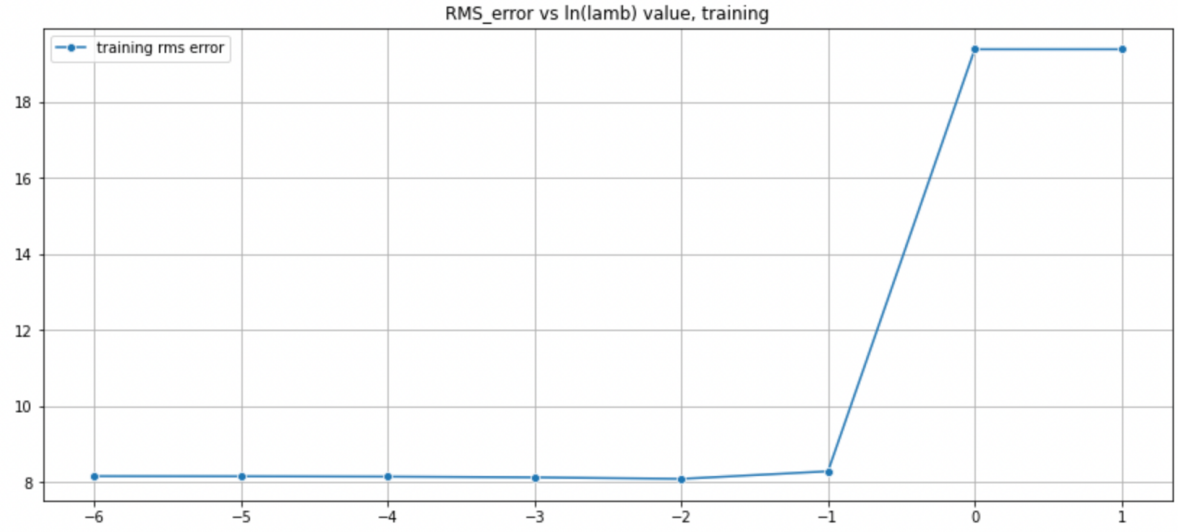
Predicted data was plotted with the actual data and was subtracted to give the noise distribution.




```
mean of noise -1.9709567311565479e-13
variance of noise 64.14192625508352
```

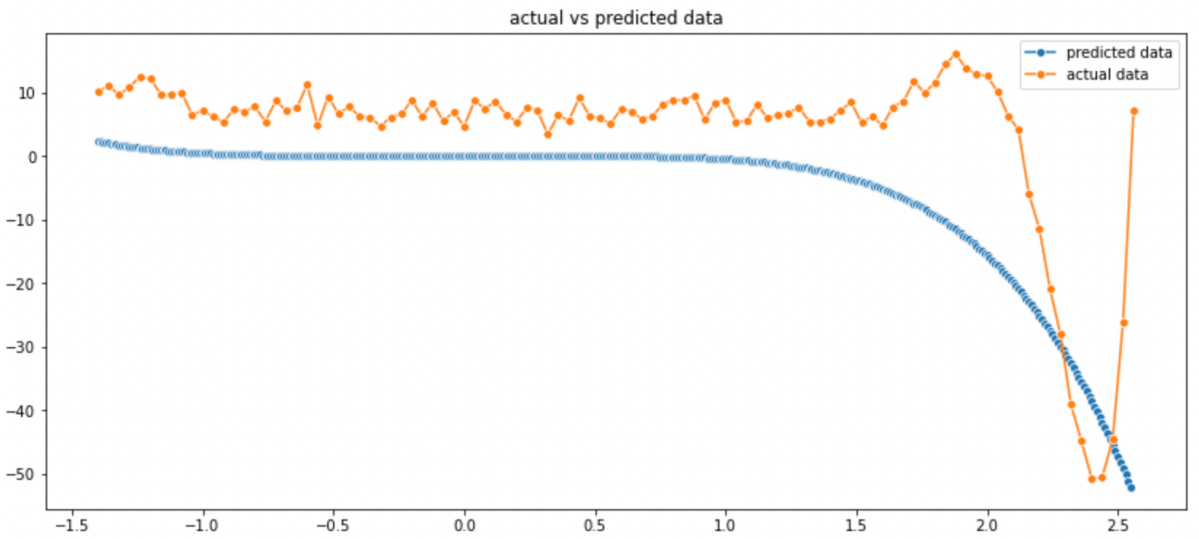
The mean is close to 0 which is expected for given 0 mean gaussian noise, but the variance is high due to underfitting by low degree polynomial.

8)Regularization:
Upon regularization we first find the relationship between error and ln(lambda), and error increases with lambda as predicted polynomial gets more and more underfitted.



9)Underfitting and overfitting:

We take just 5 iterations on a 5 degree polynomial through gradient descent to get a predicted function like this. This is clearly underfitted.



Again as before overfitting is very time taking to arrive at because even fitting properly requires more than 500k iterations and still looks underfitted, and for overtaking it would clearly take far more iterations and time to achieve.

10)Different loss function:

L1 loss function:

We perform the same steps as have been performed for L2 and compare the error results.

```
min error at batch size 100 error 205.22761424967848
```

For the same m value, batch_size, data_size, we get an error value of 205.22 which is way bigger than the L2 function, so it's worse loss function.

Few other observations (based on increasing size of dataset)

On going from the 20 point dataset to the 100 point dataset we could see that a higher order polynomial was needed to make a fitting polynomial prediction , and that's why the same order of polynomial that was giving a good fit for gradient descent for a reduced dataset didn't give a good approximation of the complete dataset, implying higher value of errors for the same conditions.

The mean of the noise went closer to 0 and variance decreased as it could account for the entire range of the data which might not have been possible for the reduced dataset as 20 points of data might not statistically represent 100 points of the same data.

Some of the trends that were not much visible or were unexpected for a reduced dataset were more visible and followed expected patterns for the 100 point dataset. Such as the case for Testing RMS error vs regularization parameter trend, which followed expected trends in the complete dataset but not the reduced dataset.

Also iterative procedures become much slower for larger datasets. For the same number of iteration we have to do more computations and thus increase the time taken for the algorithm. That's why gradient descent became a costly procedure for the complete dataset and couldn't be fitted properly.

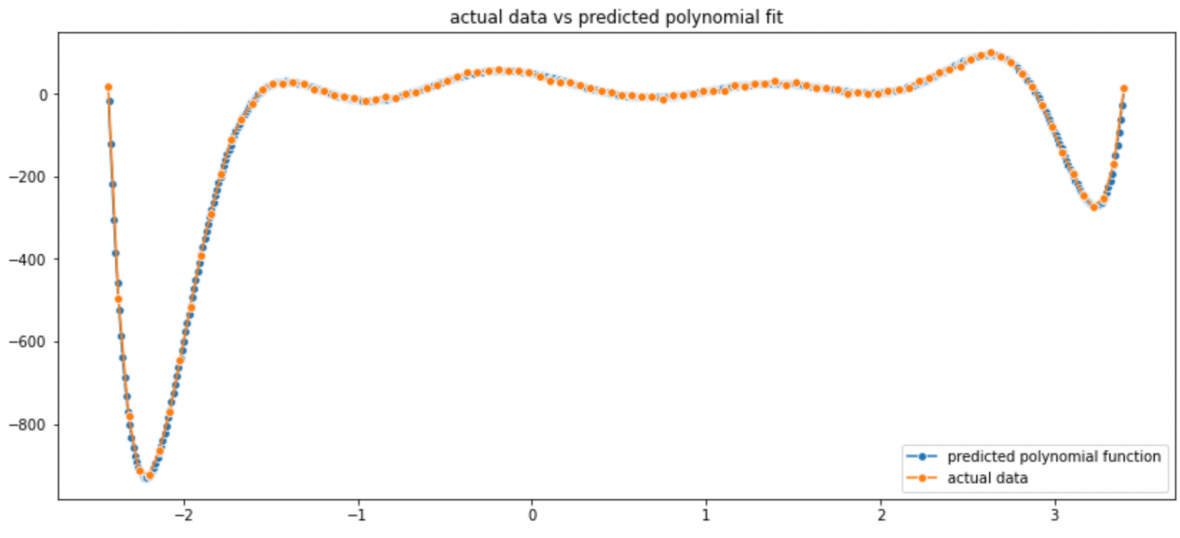
Final polynomial prediction

I would go with the moore penrose pseudoinverse result as my final polynomial prediction due to incomplete fitting done by gradient descent

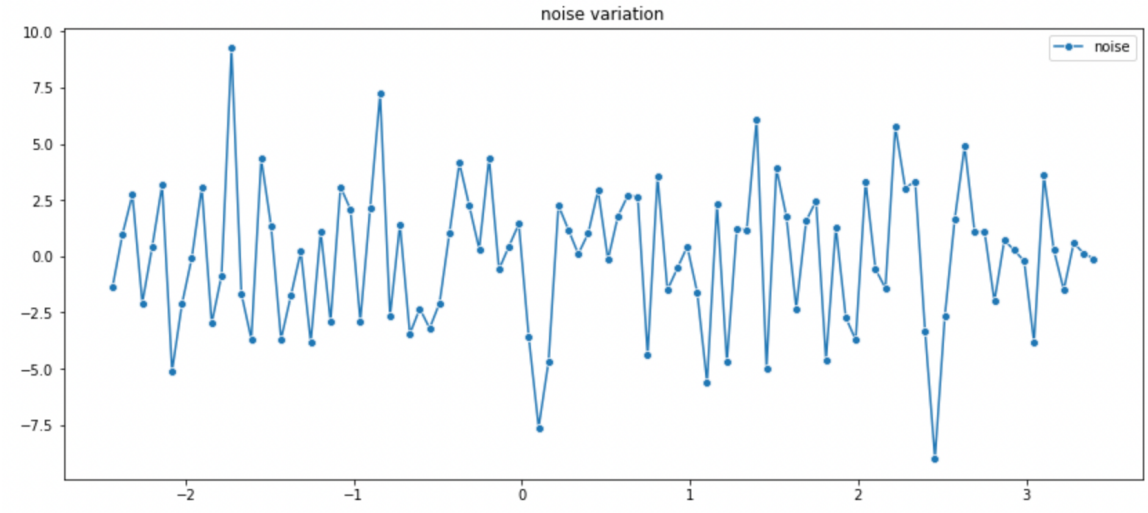
PART 1 B:

All the steps up till the noise fitting polynomial fit was done by linear regression by moore penrose approach.

The predicted polynomial and actual data mapped on the graph:



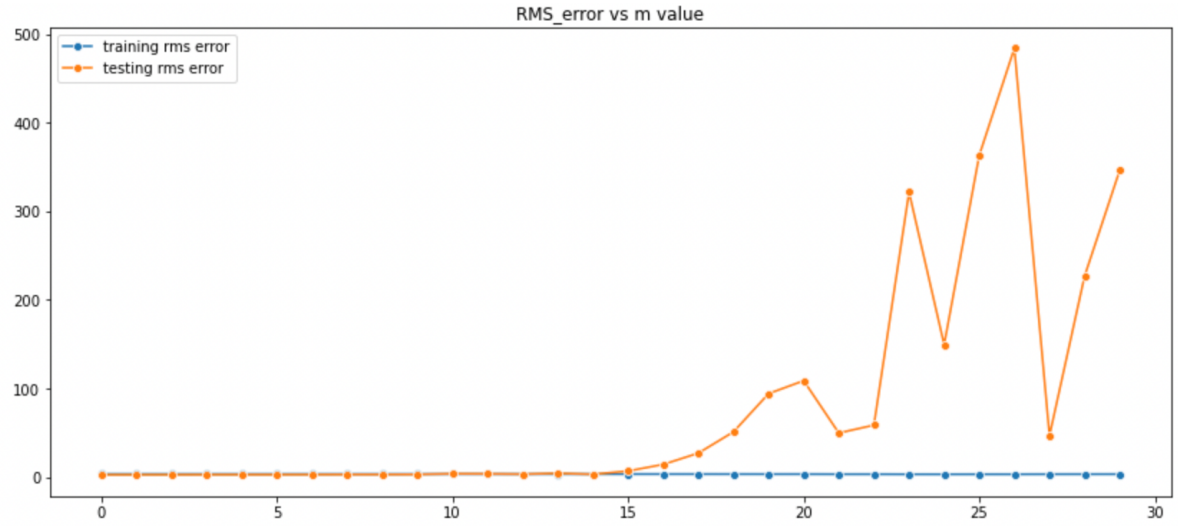
We get the following as the distribution of the noise.




```
mean of noise -5.361078692089905e-10
variance of noise 9.988516569221797
```

Next this distribution was put through another linear regressor starting from polynomial order 0 and increasing in magnitude so that a general curve can track the trend of the noise distribution and indicate the type of distribution, but that indicated that a zero order polynomial would best fit the data as the testing error kept increasing on increasing the order.

```
m optimal:  0
err optimal:  2.4973235889941896
wml optimal:  []
```



We can see here from all the above data that the mean is almost 0, and the variance is non-zero, and we also know that the noise distribution isn't gaussian. Among the common random value probability distributions , only uniform distribution and gaussian distribution has the property that the mean can be 0 without the variance being 0. So as we know that it is not gaussian , I suspect that the noise is uniformly distributed.

PART 2:

Polynomial basis function fitting approach:

First the data was loaded and investigated

	id	value	date	month	year
0	11/1/04	0.5992	1	11	4
1	10/1/12	0.9259	1	10	12
2	1/1/07	5.9182	1	1	7
3	2/1/06	9.2365	1	2	6
4	6/1/07	22.7446	1	6	7
...	...	...	...	...	...
105	9/1/06	3.2992	1	9	6
106	4/1/06	22.0183	1	4	6
107	7/1/05	22.2829	1	7	5
108	6/1/08	23.1280	1	6	8
109	10/1/10	3.1507	1	10	10

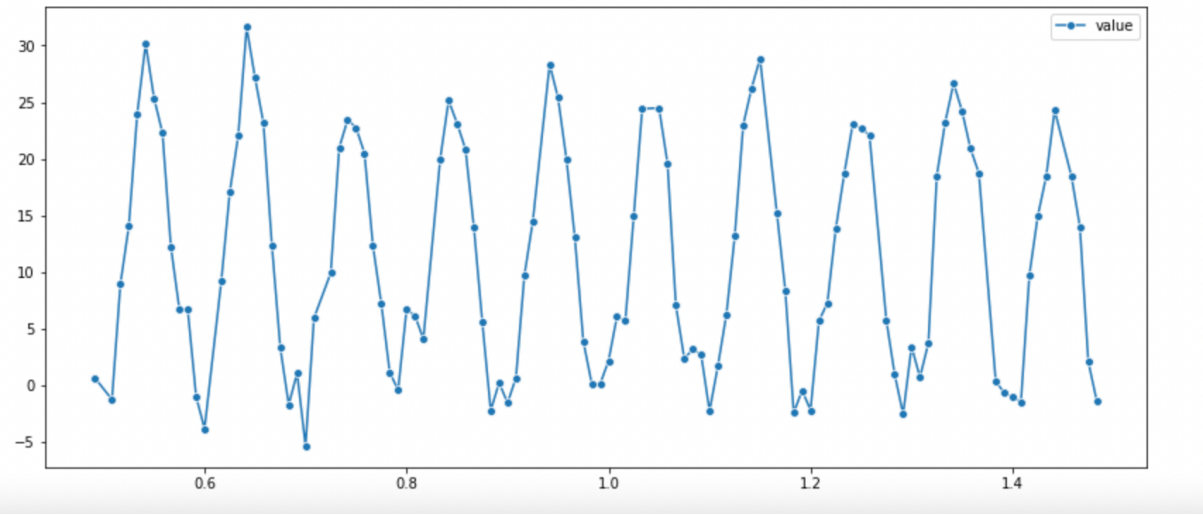
By a simple value count check it was easy to see that all the entries in the date column are 1, so it was simply ignored for the model.

Next the month and year was merged into a single function called xfunc: 12*year+month

	id	value	date	month	year	xfunc
0	11/1/04	0.5992	1	11	4	0.491667
1	10/1/12	0.9259	1	10	12	1.283333
2	1/1/07	5.9182	1	1	7	0.708333
3	2/1/06	9.2365	1	2	6	0.616667
4	6/1/07	22.7446	1	6	7	0.750000

The xfunc values were appropriately normalized down to prevent any subsequent blowups. (normalizing factor kept at 120)

The values were plotted as a function of xfunc to visualize the trends



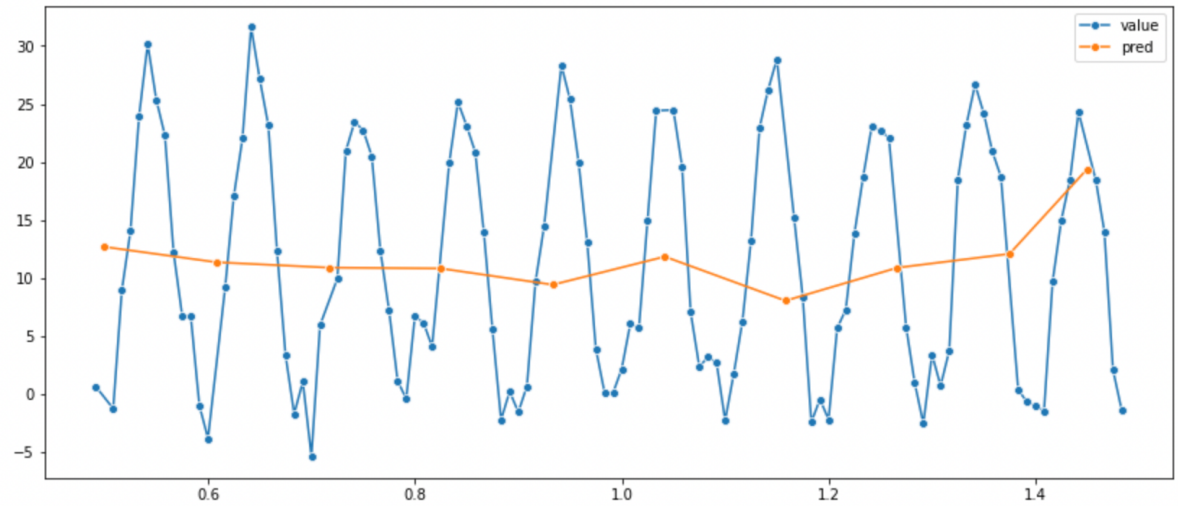
Standard polynomial basis function was applied and a linear regression model was fitted on training data using moore penrose pseudoinverse method.

```
phi[i]=x_i**i
```

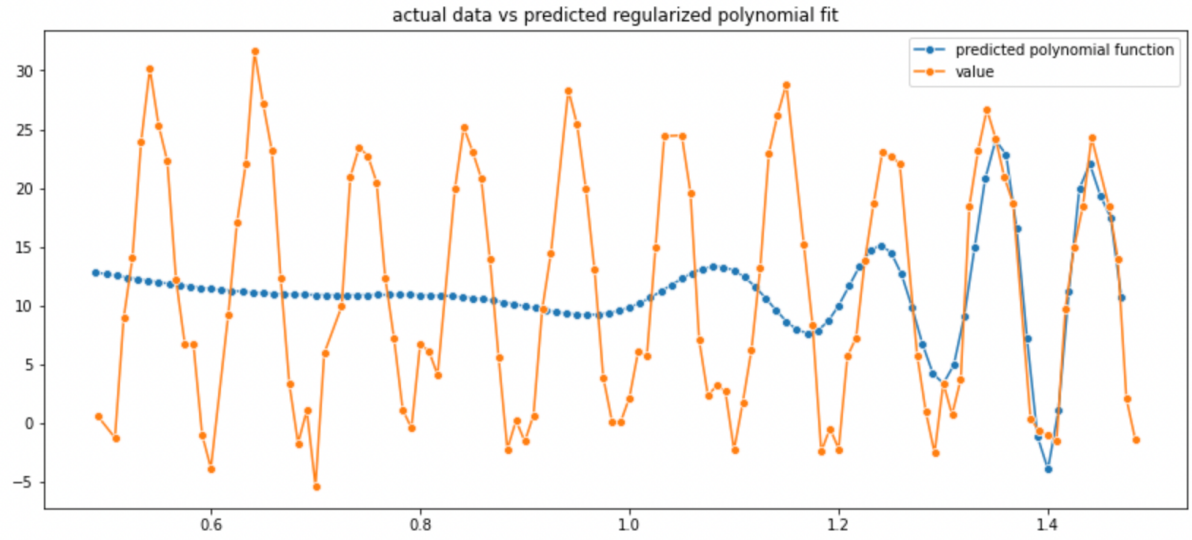
The optimal m value and rms error values obtained were as follows

```
80 9.322142540706999
```

The test set predictions were as follows



The predicted polynomial fitting function was as follows



It’s clear to see that this model fails to capture the pattern of the data and fails miserably.

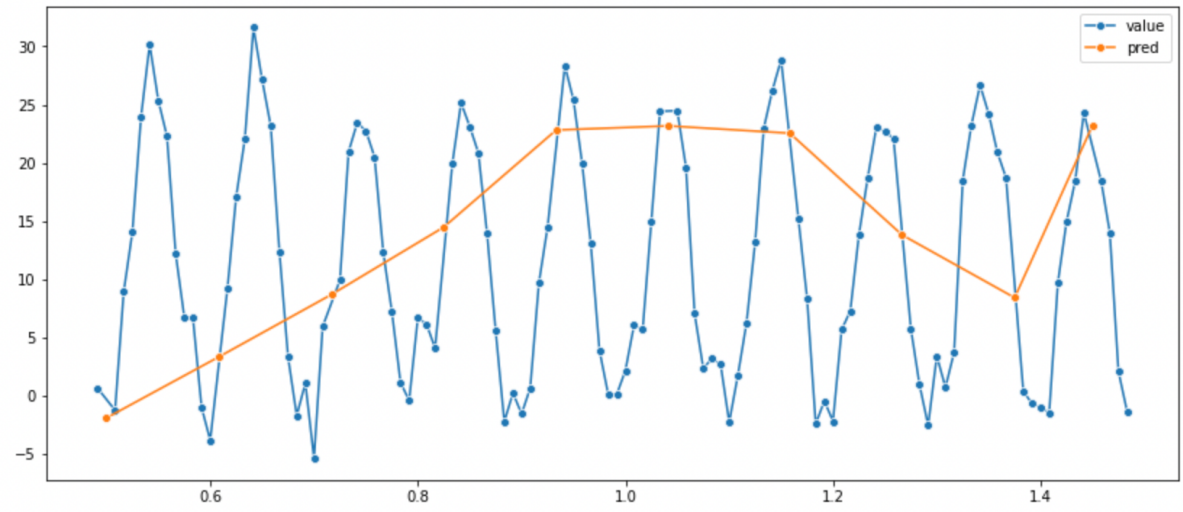
<u>Sinusoidal basis function fitting approach:</u>

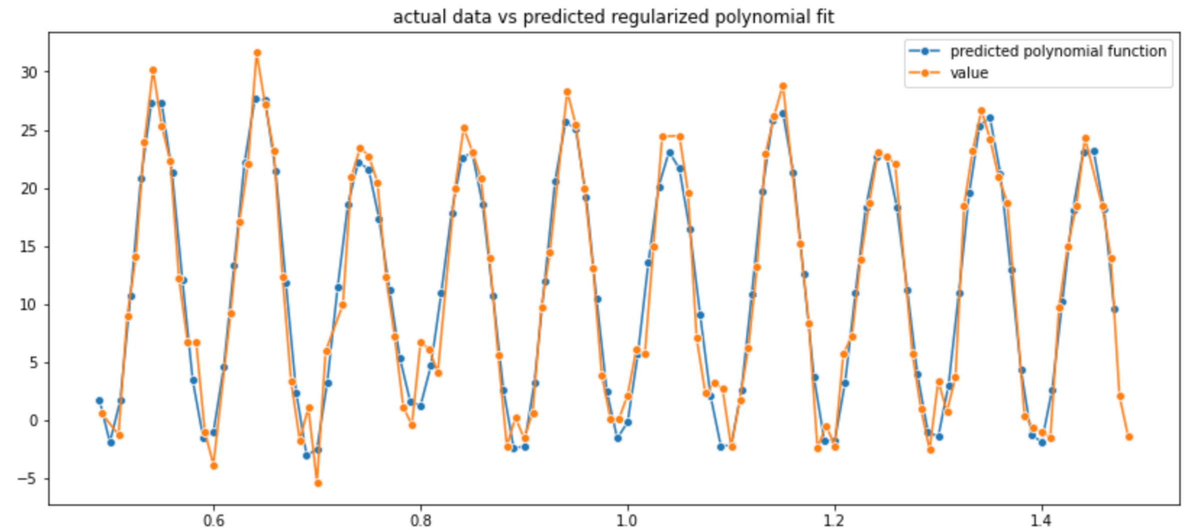
From the shape of the data it can be construed that perhaps a sinusoidal basis function would work better.

The following basis functions were tried out and there minimum rms testing error has been noted.

```
# sin(i*x_i)              2.3461
# sin(x_i**i)             9
# sin(x_i)**i+1           2.84
# sin(x_i)**i+x_i         2.86
# sin(np.pi*x_i)**i       1.917697855475552 current best
# sin(np.pi*x_i)**i+1     1.9174673875893822
# sin(np.pi*x_i)**i+x_i   1.9172379881187527
```

Considering the best function among these, the corresponding predictions and the predicted curve was plotted.





As we see that the function captures the periodic nature of the data and is a good fit for a regression model.

X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X