ELL409 Assignment 1

Ritvik Gupta
2019MT10512

1 Polynomial Curve Fitting

In this part of assignment, we had to implement the concepts of Linear Regression to solve the problem of polynomial curve fitting. We have done this for two data sets, one with Gaussian Noise and another with some Non-Gaussian Noise.

1.1 Gaussian Noise

Here, the noise in the data set is Gaussian. Hence, we have to minimise the least-squares error. This part has been solved using two methods: computing **the Moore-Penrose Pseudoinverse** and by **gradient descent algorithm** by implementing them in `numpy`.
The data given for this task is shown below in the scatter plot:

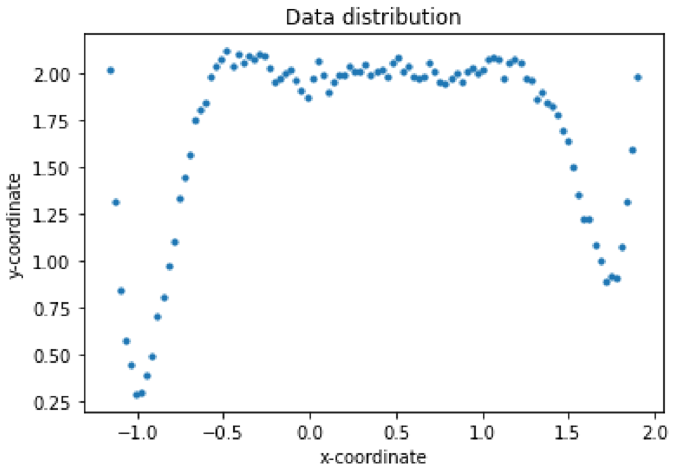

Figure 1: Data distribution for data set with Gaussian noise

For both the parts, we assume the data to be from a polynomial of degree M and then generate a design matrix containing feature vectors $(1, x^1, x^2, ..., x^M)$ for each x in the data set. In the data distribution, the range of x is from around -1 to 2, hence in case of higher degree of polynomials, say 10, the feature vector would be containing values of orders ranging from less than 10^{-10} to around 10^3. This would prevent the proper convergence of gradient descent. Hence, the data has been normalized to have zero mean and unit variance.
We then minimise the least-squares error, i.e., $\sum_{i=0}^{n}(h_\theta(x_i) - t_i)^2$ using the above mentioned two methods. Using the implementation of the above methods done using `numpy`, I have obtained the following results and observations.

Experiments[1]

Firstly, to get an idea about the degree of the polynomial in the data set, we plot the polynomial curves of various degrees obtained by fitting it on the data set.

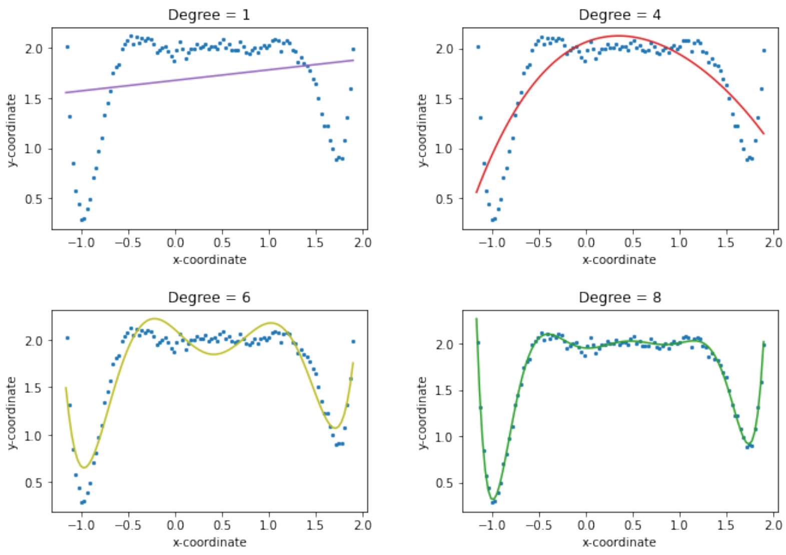

Figure 2: Polynomials of different degrees fitted on the data

The above graphic shows how the polynomial curve changes on increasing the degree of the polynomial. We see that curves of degree 1 and 4 barely manage to predict the data with accuracy, while the curves of higher degrees fit the data in a much better way. This is because increasing the polynomial degree, increases the complexity of the model and now it can model a much more complex curves due to the increased number of parameters.

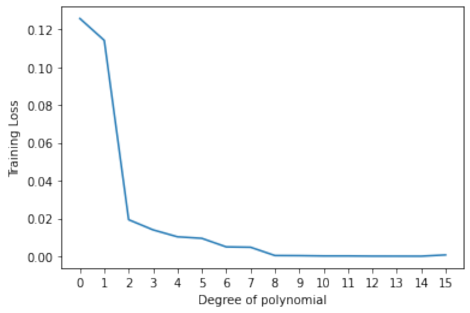

Figure 3: Loss v/s Degree of Polynomial

The above plot shows how the losses reduce on increasing the model complexity, i.e., increasing the degree of the polynomial. We can clearly see that the loss decreases by a considerable amount at $M = 8$ and remains the same for the subsequent degrees. Following the principle of Occam's Razor, we can say that the degree of the polynomial must be **8** itself.

[1]Code for the experiments can be found in the folder itself

Following this, to explore the convergence of gradient descent, we plot some graphs by running gradient descent on the data and logging the intermediate losses and weights.

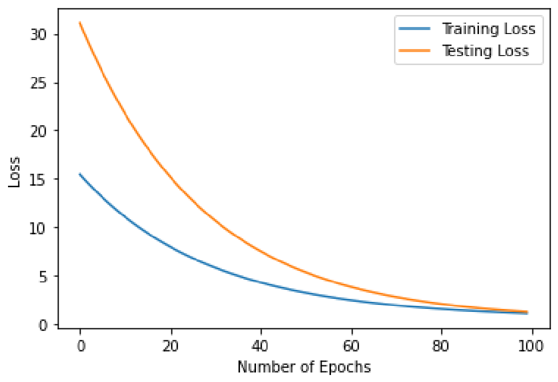

Figure 4: Training and Testing loss v/s Number of epochs passed

The above plot shows how the losses decrease on increasing number of passing epochs. For this plot, the data was split into 20 samples for Training and 80 samples for Testing. We can see that initially the losses for the training set are lower as the model has not run for many epochs to generalise well on the testing data. After sufficiently number of epochs, the losses converge and thereafter decrease at a very slow rate. This is because, now we are near the minimum and the updates result in much smaller changes in the losses.

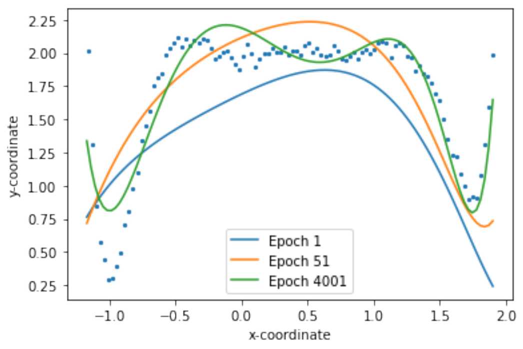

Figure 5: Curves obtained at the end of 1, 51, and 4001 epochs of gradient descent

The above plot shows how the curve adjusts its parameters to fit the data on each iteration of the gradient descent algorithm. Initially, the curve does not fit the data at all but, after 51 epochs, the curve becomes much better in terms of covering the data. Still it does not model the data properly. After 4001 epochs, the curve fits the data almost properly and in a much better way. It also fits the maxima and minima properly, which the curve after 51 epochs did not do at all.

Another observation here can be the fact that the model learnt after 4001 epochs fits the right portion of the data points in a much better way as compared to the left portion of the data points. This is majorly due to the fact that the right portion of the data points have a much higher density whereas the data points on the left are much more sparse. Hence, the driving factor in the loss minimization becomes the right portion of data points as they contribute more towards the MSE loss and the curve tends to fit the points on the right more accurately. One way to solve this problem can be to re-weight the losses from the individual data points. We can give higher weights to the points on the left and lower weights to the weights on the right.

Next, we plot some graphs to explore the variations between full-batch gradient descent and stochastic gradient descent.

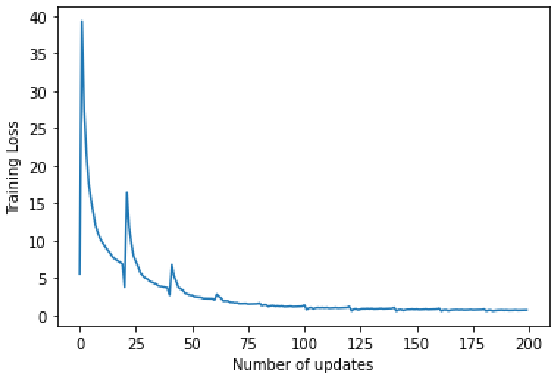

Figure 6: Loss v/s Number of updates performed

In the above plot, the losses after each update of stochastic gradient descent have been plotted.[2] We can see that the losses, unlike in full-batch gradient descent, do not decrease monotonically and there are some peaks in between. This is because we are training the model on a single data point at once and using that to update the weights. This may not be the best choice for update in weights as it does not consider the other points, but it does converge after sufficient number of epochs. As we can see, the losses are still decreasing, just not monotonically. Hence, stochastic or mini batch gradient descent is a good choice of algorithm to use in case of memory constraints due to which only a few data points may be processed at a time.

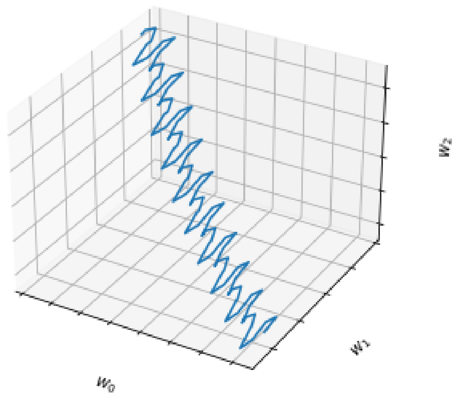

Figure 7: Weight updates in parameter space for stochastic gradient descent

The above figure is a plot of the weight updates done during a run of stochastic gradient descent.[3] The updates here are quite arbitrary in the sense that they do not lead towards the minima always. But over multiple update iterations in the algorithm, the weights do get updated to values nearer to the minima and converge to the point of minima.

[2]For the sake of experimentation, the model has been trained on only 10 data points.

[3]Only 10 data points have to been used to fit a quadratic polynomial and only the updates for first 10 epochs have been shown here.

Next, to observe changes caused by adding regularization, we plot the polynomial curves obtained on different values of the regularization constant λ.

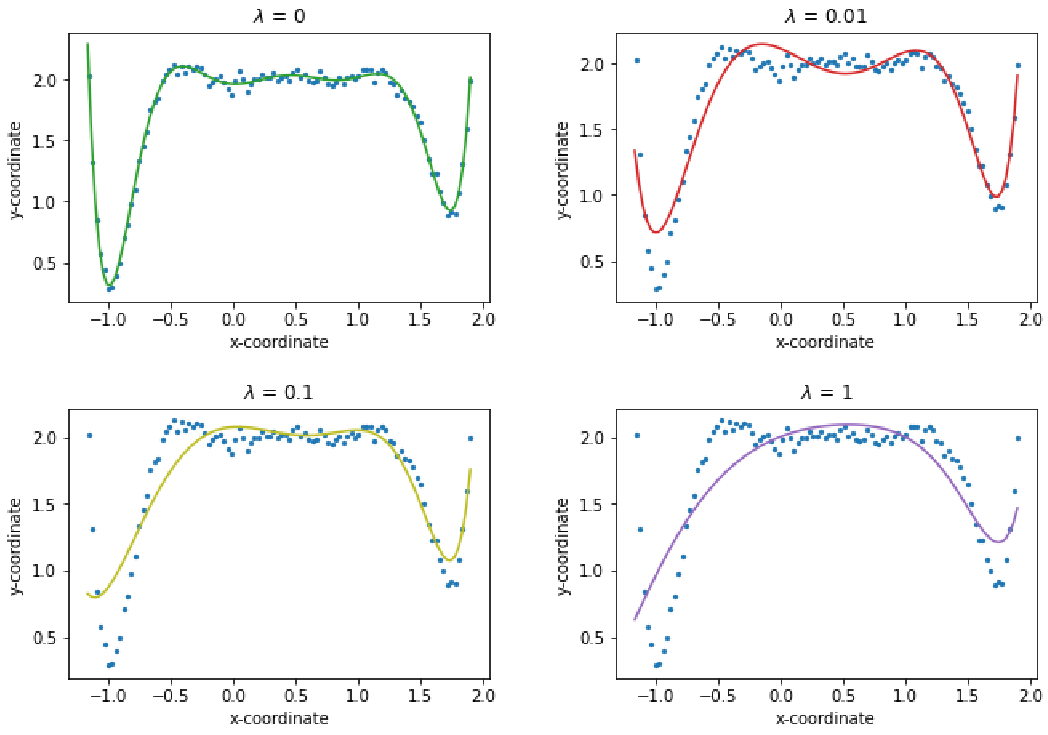

Figure 8: Polynomials obtained on different values of λ

In the above figure, we can see the effect of regularization on the complexity of the model. The regularization term in the regularized loss ($\lambda\frac{||w||_2}{2}$), penalises the model for choosing weights with larger L_2 norm and hence the model chooses weights with smaller values. Regularization comes from estimating the MAP estimate using a prior and an increased regularization strength implies a greater influence of the prior. The larger the value of λ is, the stronger the effect of regularization and the influence of the prior becomes.
In this case, if the weights are restricted to be smaller, then the model is not able to fit the various sharp peaks and inflections present in the data. We can clearly see in the figure how the model becomes less complex as the regularization strength is increased.
The optimal value of λ for the problem of polynomial curve fitting is 0 (for $M = 8$). This is because we have the prior information that the underlying function that we have to fit is a polynomial and we cannot assume the prior distribution of the weights to be gaussian with zero mean and λ variance. In fact, the weights of the polynomial need not be restricted to be around 0. Regularization is useful when we model some random function as a polynomial and also when the data points given to us are very less and we fit a polynomial of higher degree on it.

Here, we have considered the effects of adding more data to the training data set.

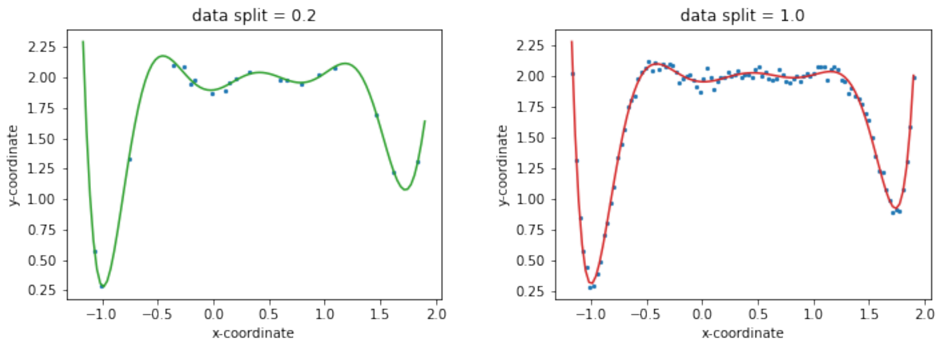

Figure 9: Plot for polynomials obtained using the first 20 data points and the entire data set

In the above figure, we can see that the curves obtained are very similar. This is due to the fact that the data points on the left are quite uniformly distributed across the data set and thus there is not much difference in the two curves. Hence, all the results after training the model on the first 20 points and the entire data set are similar.

Estimate of underlying Polynomial

As discussed earlier, from Figure 3, the underlying polynomial must be of degree **8**. For fitting degree 8 polynomial on the entire data set, we can take $\lambda = 0$ as we have already picked the least complex model. After running the least-squares regression on the entire data, we get,

```
Loss: 0.0006959032691692958
weights=[ 1.95519072 -0.06712881 1.33394401 -0.92882294 -4.18577591 5.05074741
1.0734225 -3.20754543 0.98597196]
```

Therefore, the polynomial is the polynomial given by $f(x) = \sum_{i=0}^{8} w[i]x^i$, where `w` has been given above. We can estimate the noise variance using the loss obtained. We have defined the loss as,

$$E(w) = \frac{1}{N}\sum_{i=0}^{N}(t_i - w^T\phi(x_i))^2$$

Also, the noise variance is given by, $\sigma_{noise}^2 = \frac{1}{N}\sum_{i=0}^{N}(t_i - w^T\phi(x_i))^2 = E(w) = 0.0006959 = 6.959 \times 10^{-4}$

1.2 Non-Gaussian Noise

Here, the noise in the data set provided is of some non-Gaussian distribution. The data given for this task is shown below in the scatter plot:

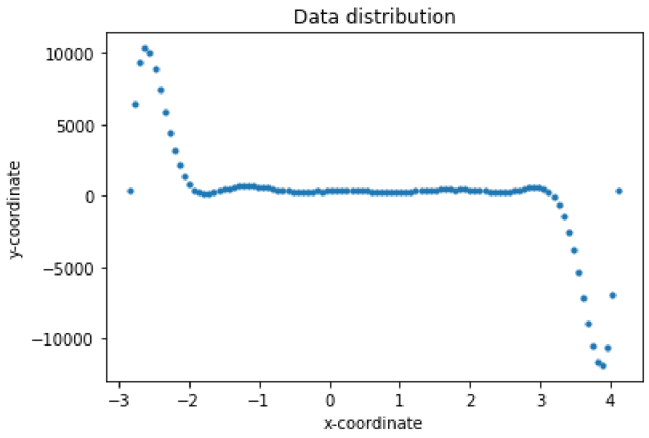

Figure 10: Data distribution for data set with non-Gaussian noise

We have to try and characterise the noise and find out what kind of noise it is. We know, that values $t_i - y(x_i)$ comprise the noise in the data. Hence, we can to identify the noise by analysing these values. For this, first we have to fit a polynomial of suitable degree on this data. To find the suitable degree of polynomial, we plot the various polynomials found at various values of M.

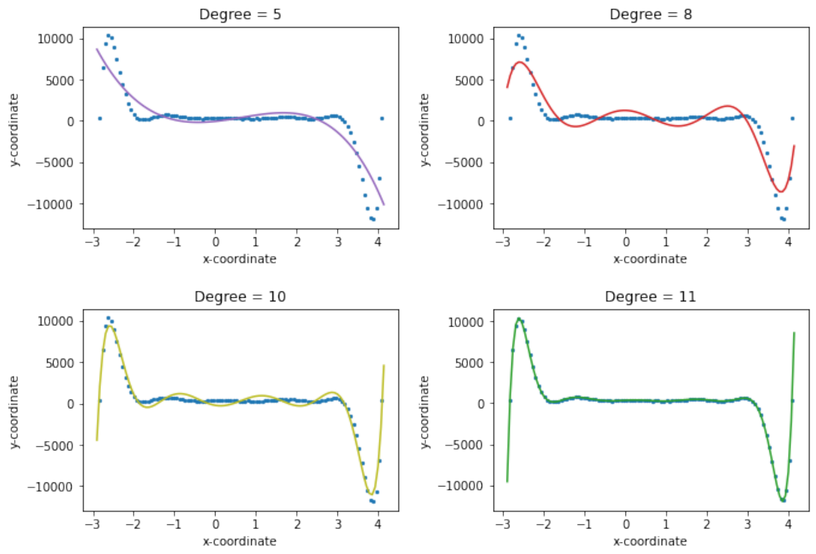

Figure 11: Polynomials of different degrees fitted on the data

From the above figure, it can be concluded that the degree of the underlying polynomial must be **11**. This is simply because the 11^{th} degree polynomial is the polynomial with lowest degree which fits the data well. Further, the graph of loss v/s degree of polynomial has also been plotted.

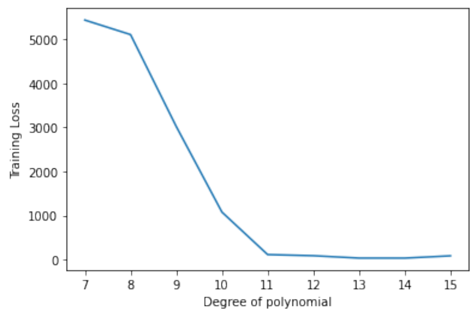

Figure 12: Loss v/s Degree of Polynomial

It can be clearly seen that the loss decreases by a considerable amount for $M = 11$ and remains mostly constant afterwards. This is a reconfirmation to our claim that the polynomial is of degree 11.
Using this information, we then find an estimate for the underlying polynomial by fitting an 11 degree curve to the data. This polynomial found can be used as a good estimate for $y(x_i)$, and can be used to find the noise on each data point.
(Note that we have used least-squares fitting to estimate the polynomial which is used when the noise is Gaussian. But even in this case, the estimated polynomial can be used as a good approximation for $y(x_i)$)
Next, we plot the histograms for $t_i - y(x_i)$.

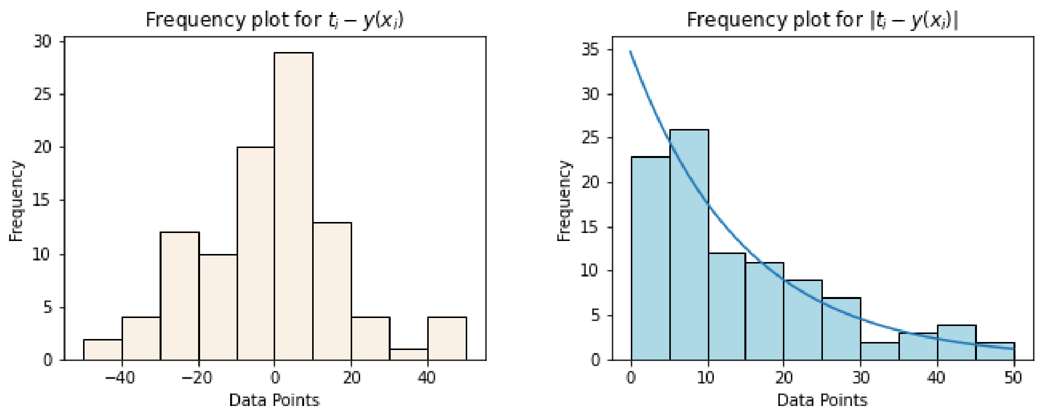

Figure 13: Histograms for $t_i - y(x_i)$

As the first histogram resembles the Poisson probability distribution function, so the guess for the noise type would be Poisson distribution with a shifted mean so as to incorporate negative values as well.
The second histogram resembles the exponential probability distribution function and hence, $|t_i - y(x_i)|$ may be distributed exponentially. Thus, the absolute value of noise may be generated from an exponential distribution.

2 Linear Regression on Time Series Data

For this part, we have to use linear regression models to model the given time-series data. The data given is shown below.

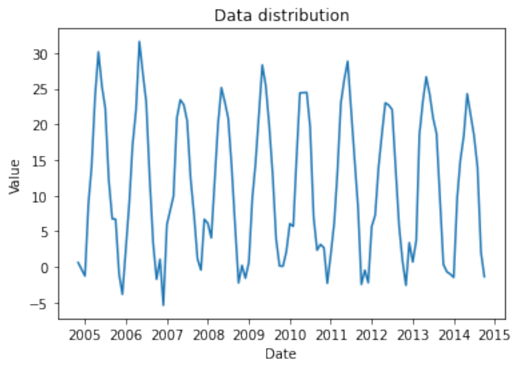

Figure 14: Data distribution for time-series data

Attempt 1

From the above data, we can see that the has somewhat similar patterns over months of every year. The data patterns thus, repeat over the year. This motivates us to plot the data for the various months and years.

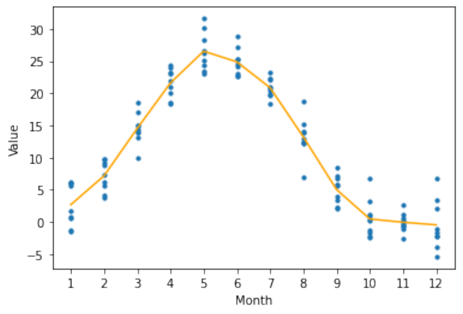

Figure 15: Month wise data distribution

From the data plots, we can see a clear trend in the values based on their months. We can try to fit a curve and model a polynomial which is a function of the month by fitting it on the given data. One trivial way is that we just return the average values of the month for which we have to predict the value but that would not work if we are required to give prediction on a continuous set. So we try and fit a polynomial on this month wise data.

The next plot is the curve of degree 7 fitted on the month wise data. Using the this curve, a loss of 5.42238 was obtained on the test set. While using just the average values provides a loss of 5.15135 which is even lower than the loss achieved using this model, returning average value is only valid if the test set is discrete and would not work very well on continuous data. Whereas, this model interpolates the values for the intermediate values and can even predict values on continuous data.

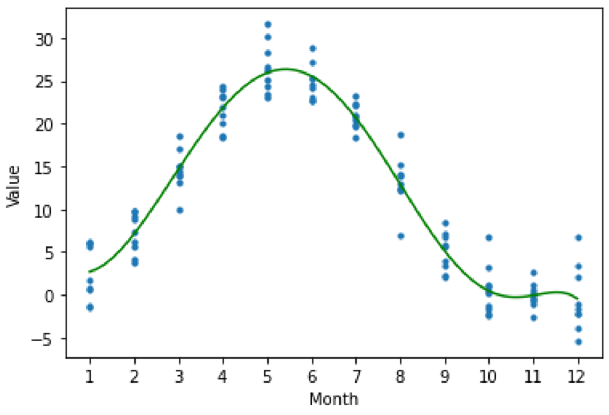

Figure 16: Plot for polynomial fitted on month wise data

Attempt 2

In the earlier attempt, we only modelled the data to be dependent only on the month value and discarded the year data. This, however is not correct. There might be some dependency of the values on the year data as well and completely ignoring it is not correct.
In this attempt, we try to model the data using a multi variable polynomial fit. Here, we consider both the year and month data. For some inputs x_1, x_2, we try to model the data by fitting the following polynomial.

$$f(x_1, x_2) = \sum_{\substack{i+j \leq M \\ 0 \leq i,j \leq M}} \sum w_{ij} x_1^i x_2^j$$

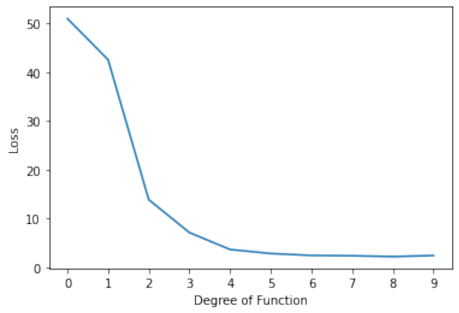

Figure 17: Loss v/s Degree of Polynomial

For identifying the ideal degree of the polynomial, we plot the graph for loss v/s degree of polynomial as shown below. From this plot, we can clearly see that the loss decreases considerably at degree 5 and oscillates in nearby values thereafter. Thus, the ideal degree of polynomial is 5.
As such a multi variable polynomial has a lot of weights for higher degrees, it is essential to ensure that the model is not overfitting. Hence, L_2 regularization must be applied. After a lot of tweaking of λ, the value of 0.2 was found to be optimal.
Using this model, a loss of 4.25185 was obtained on the test set, which is considerably lower in comparison to our loss in attempt 1. This suggests that indeed, discarding the year data is not right and there is some correlation between the year data and the values.