CS-F320 FODS

Assignment 1

BY

Suyash Patil	2021A7PS2078H
Pratik Patil	2021A7PS3111H
Dhananjay Shalley	2021A7PS0401H

October 2023

Table of contents

INTRODUCTION

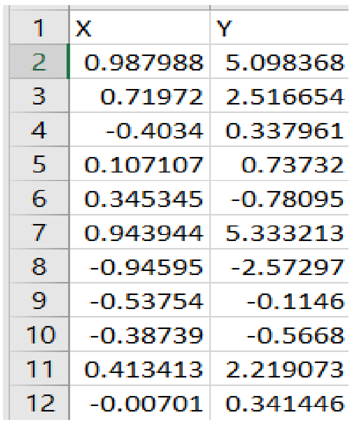
This report examines the work completed for Assignment 1 of the CS F320 FODS course. The goal of this assignment is to create model predictions for a given set of data and then use the generated models to predict the target variables based on feature variables. In Part A, we utilize polynomial regression for capturing non-linear relationships in the data; we built 9 models ranging in degree from 1 to 9 and used batch gradient descent as the optimization approach; and after analysing all of these models, we choose the best-fitting and simplest model.

Part B compares polynomial regression models for predicting the target variable using two input feature variables. The analysis includes the development of nine polynomial regression models with degrees ranging from 0 to 9, followed by the selection of the best-fitting model based on the results of these models, as well as the study of regularized polynomial regression models with different regularization parameters (q = 0.5, 1, 2, 4) and values for the best-fitting model). These models are built and evaluated using both stochastic and batch gradient descent techniques.

Part A: Regression without Regularization

Task 1: Data Preprocessing:

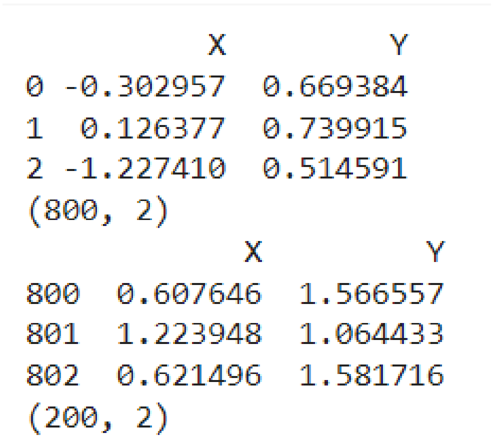
The given set of data as follows:



1	X	Y
2	0.987988	5.098368
3	0.71972	2.516654
4	-0.4034	0.337961
5	0.107107	0.73732
6	0.345345	-0.78095
7	0.943944	5.333213
8	-0.94595	-2.57297
9	-0.53754	-0.1146
10	-0.38739	-0.5668
11	0.413413	2.219073
12	-0.00701	0.341446

The dataset consists of 1,000 records containing characteristic X and the target variable Y. The process of data preparation included the normalization of the provided data.

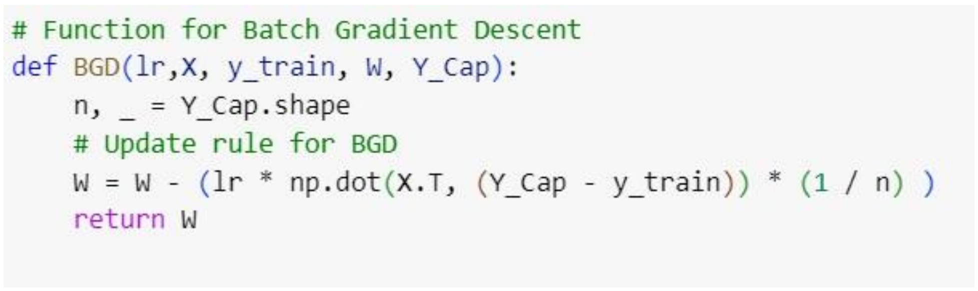
After applying the normalization formula X' = (X - μ) / σ to the data, it is then shuffled and divided into two datasets, namely `training` and `testing`, in an 80:20 ratio.



```
          X         Y
0 -0.302957  0.669384
1  0.126377  0.739915
2 -1.227410  0.514591
(800, 2)
            X         Y
800  0.607646  1.566557
801  1.223948  1.064433
802  0.621496  1.581716
(200, 2)
```

Task 2: Polynomial Regression:

Polynomial regression models were created, with degrees ranging from 1 to 9. The data was fitted to each model by using the normal equation approach throughout the training process. The batch gradient descent algorithm was implemented with a learning rate of 0.001.



```python
# Function for Batch Gradient Descent
def BGD(lr,X, y_train, W, Y_Cap):
    n, _ = Y_Cap.shape
    # Update rule for BGD
    W = W - (lr * np.dot(X.T, (Y_Cap - y_train)) * (1 / n) )
    return W
```

The final training and testing LMS Errors discovered are listed below:

Training:

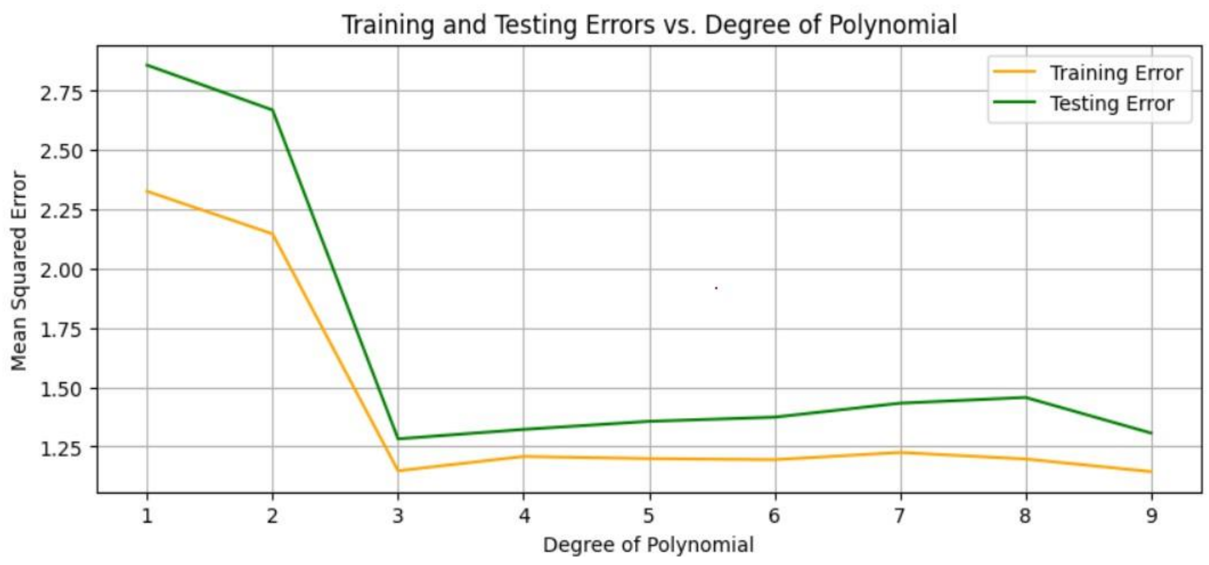
```
[2.326519236262547,        2.1467772980574455,        1.1474004398074096,
1.2079099951463075,        1.1987444410562125,        1.1943171716262189,
1.2246801948563313, 1.1971177728218865, 1.1439579172641734]
```

Testing:

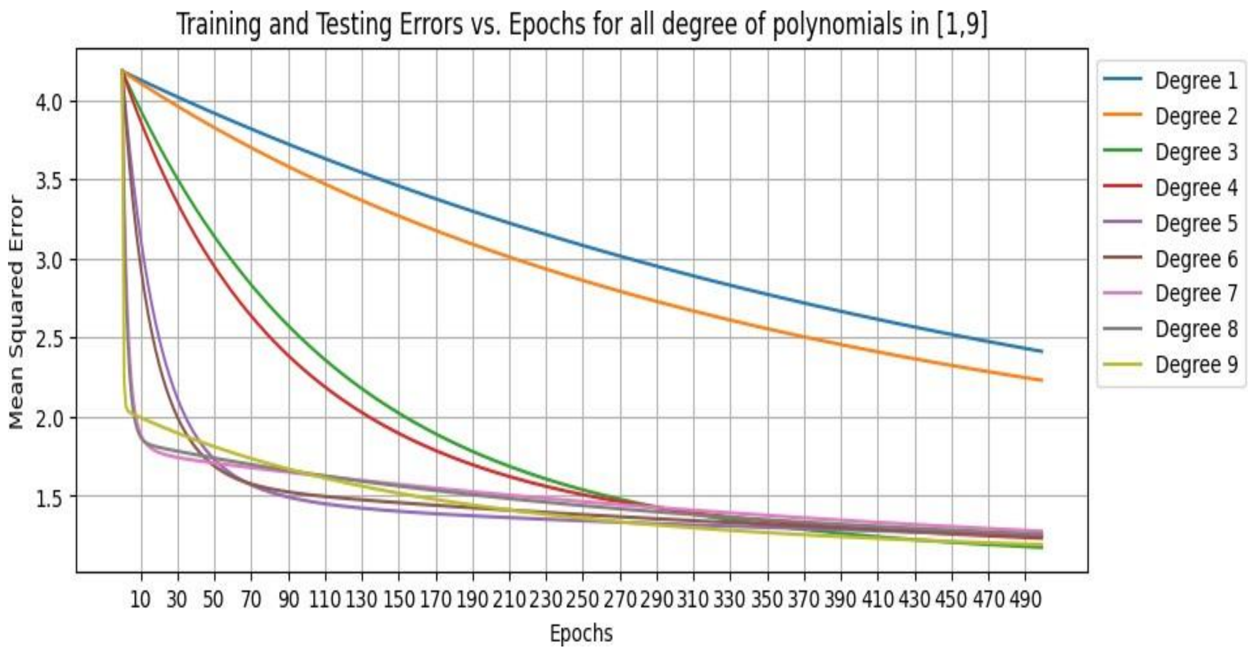
```
[2.858578947114214,        2.669353820394622,        1.2820572214802095,
1.3219168127291299,        1.355975792285222,        1.373598805101422,
1.4325670029655928, 1.456822197929122, 1.3066015718110446]
```

Task 3: Graph Plotting:

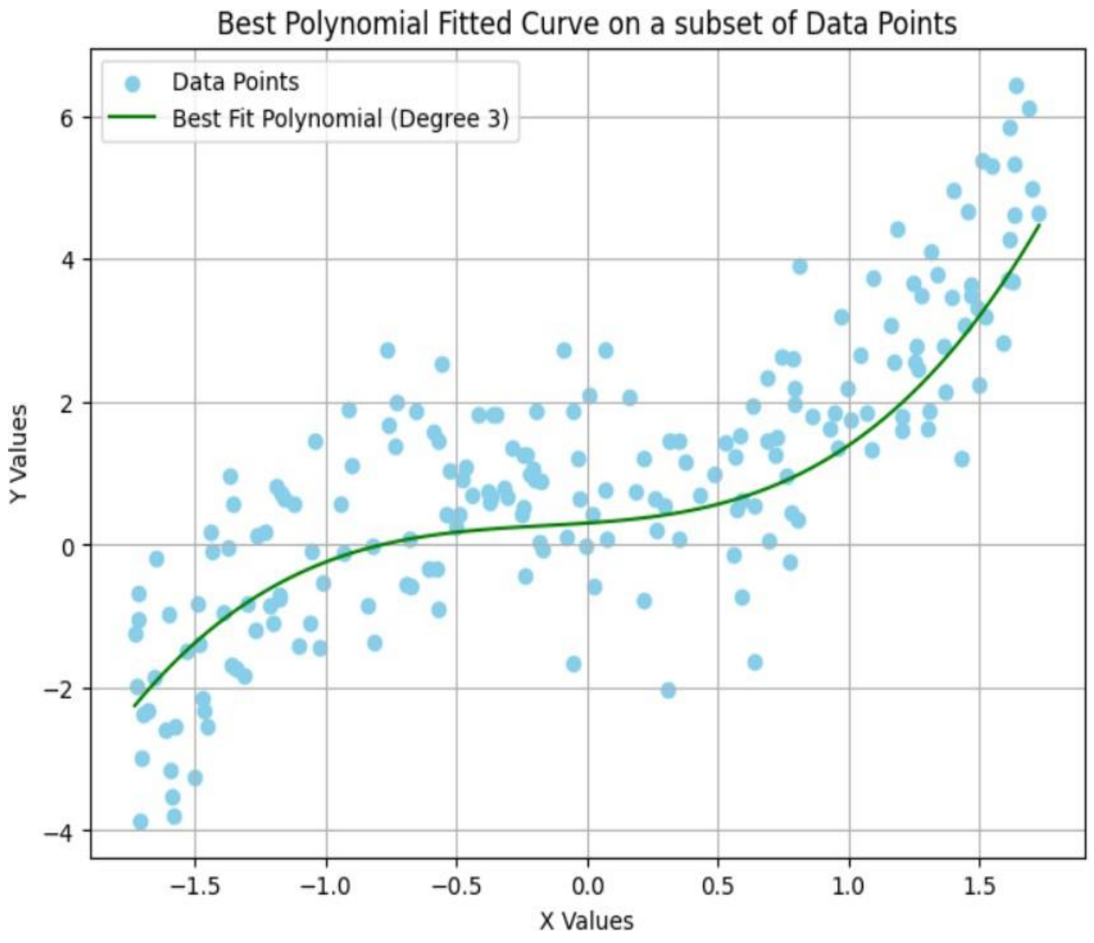
1) Final Training and Testing Errors v/s degree of polynomial:



2) <u>Training Error and Testing Error v/s Epochs for all degree of polynomials in [1,9]:</u>



3) The best polynomial fitted curve on the data points:



Task 4: Comparative Analysis:

Based on the gathered training and testing errors, we compare the nine polynomial regression models in this part to identify which one outperforms the others for our dataset.

Training Error: As the degree of the polynomial grows, the training error often reduces. Higher-order models often fit the training set of data quite well.

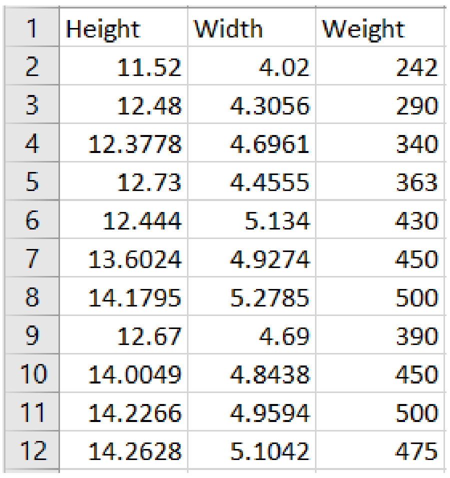
Testing Error: A decrease in the testing error from degree 1 to 3 initially indicates increased generalization. However, it begins to rise after degree 4, which points to overfitting.

We chose the polynomial regression model with degree 3 for this dataset after weighing the trade-off between training and testing errors and the potential risk of overfitting. While avoiding the overfitting seen in higher-degree models, it manages to attain a rather low testing error.

Part B: Polynomial Regression and Regularization

Task 1: Data Preprocessing:

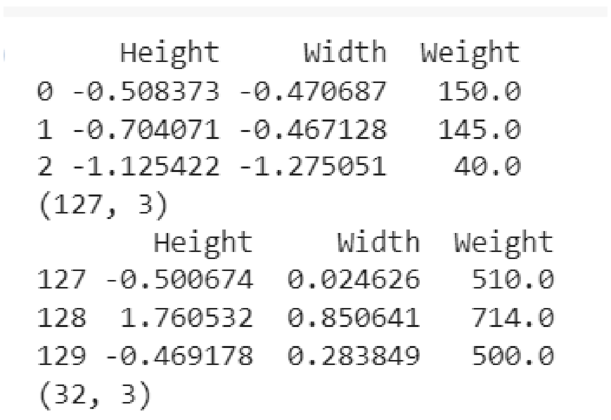
The given set of data as follows:



1	Height	Width	Weight
2	11.52	4.02	242
3	12.48	4.3056	290
4	12.3778	4.6961	340
5	12.73	4.4555	363
6	12.444	5.134	430
7	13.6024	4.9274	450
8	14.1795	5.2785	500
9	12.67	4.69	390
10	14.0049	4.8438	450
11	14.2266	4.9594	500
12	14.2628	5.1042	475

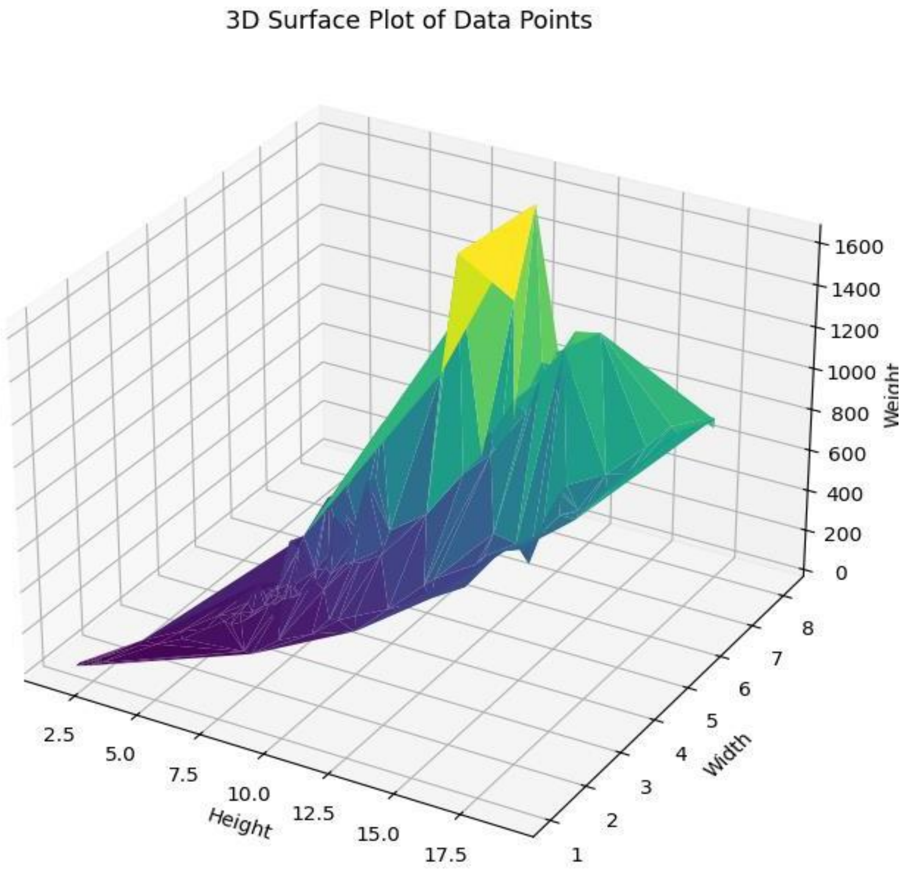
The dataset consists of 200 records with the fish's height, width, and weight as the target variables. This data was normalized during data preparation, and the means of the particular columns were used to replace null/Nan values.

Following data processing (normalization) using the formula X' = (X - µ) / σ and replacing null/Nan values, we shuffle and divide the data into two datasets called "training" and "testing," as shown below.



```
     Height     Width  Weight
0 -0.508373 -0.470687   150.0
1 -0.704071 -0.467128   145.0
2 -1.125422 -1.275051    40.0
(127, 3)
       Height     Width  Weight
127 -0.500674  0.024626   510.0
128  1.760532  0.850641   714.0
129 -0.469178  0.283849   500.0
(32, 3)
```

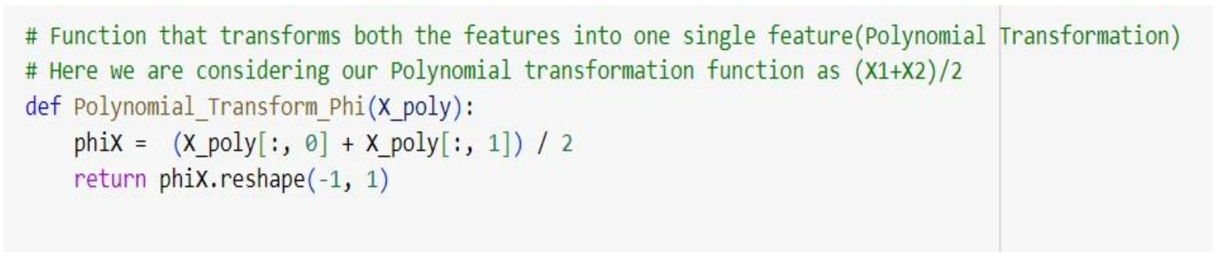
Scatter plot of all the data points:



Scatter Plot of given data set

Task 2: Polynomial Regression:

There were nine polynomial models created. By averaging them out, feature variables were integrated in Φ(X) Here is a sample of some code:



```
# Function that transforms both the features into one single feature(Polynomial Transformation)
# Here we are considering our Polynomial transformation function as (X1+X2)/2
def Polynomial_Transform_Phi(X_poly):
    phiX =  (X_poly[:, 0] + X_poly[:, 1]) / 2
    return phiX.reshape(-1, 1)
```

The LMS error functions and batch gradient descent are the same as in Part A.

The following errors were acquired for the training set and testing set after the model was developed using the batch gradient descent approach with learning rate = 0.00001 and 500000 iterations:

Training:

```
[312.1619376800318, 285.24061259487183, 315.5347944176116,
305.8780174479936, 283.61652284037496, 292.6704662265064,
306.0934130442628, 301.67630023993445, 290.43163963792546]
```

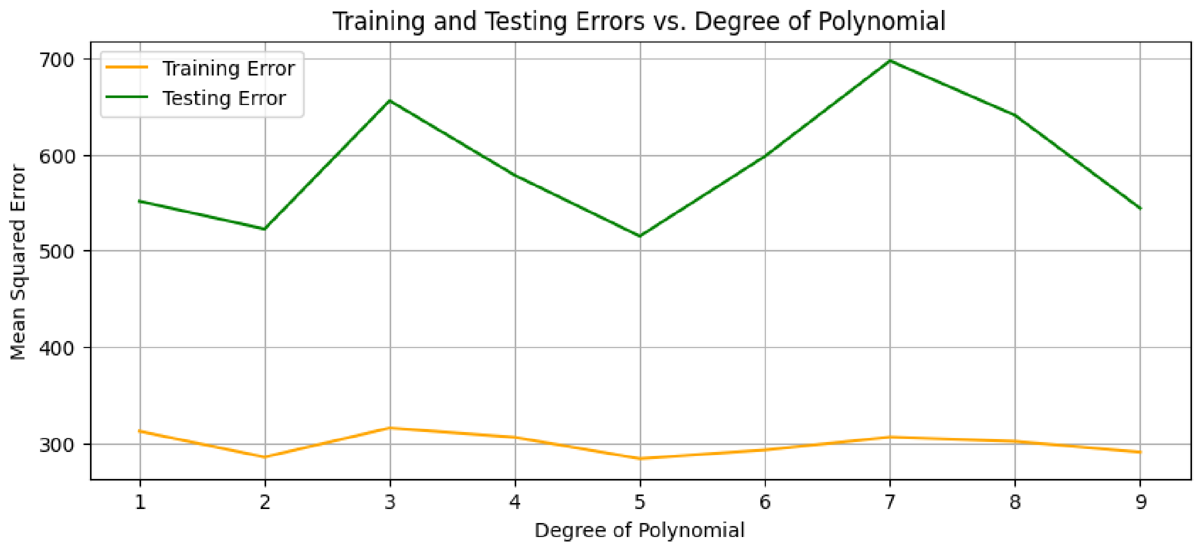
Testing:

```
[551.3841942010852, 522.4399092186927, 655.9197517827704,
578.4124788182464, 515.0086548181804, 598.0295844248534,
697.4259228040464, 640.7183796161088, 544.1078254773657]
```

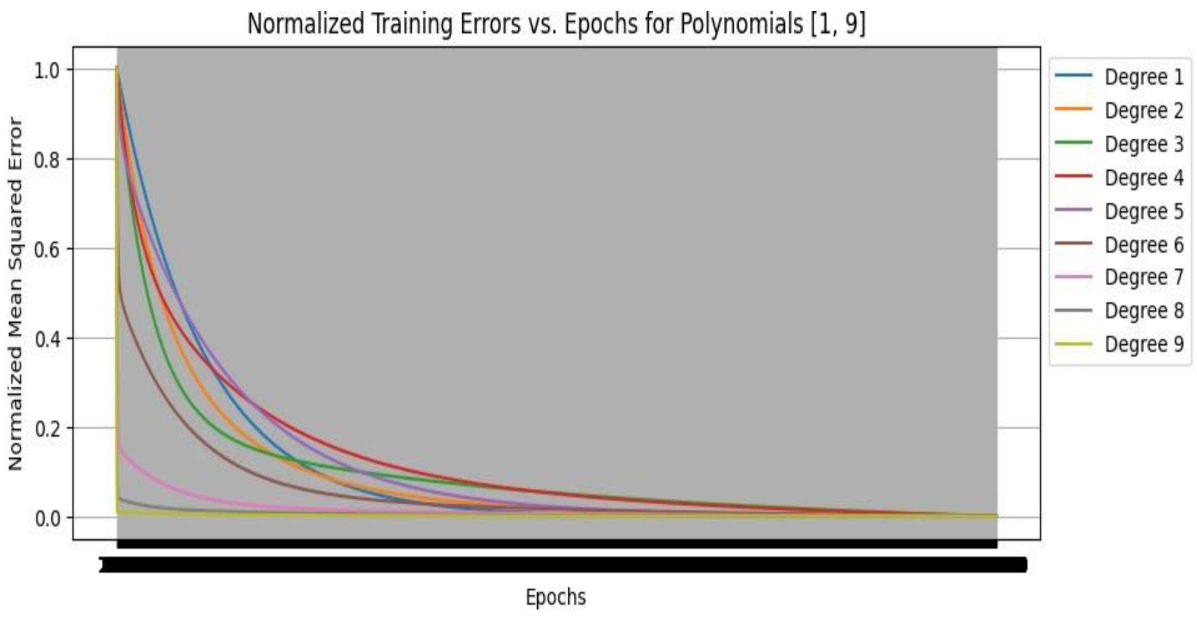
(From left to right, degree 1 to 9)

The following graphs were created using the findings from the previous analysis to find the polynomial that best fits the data:

1) Training and Testing Errors v/s degree of polynomial:

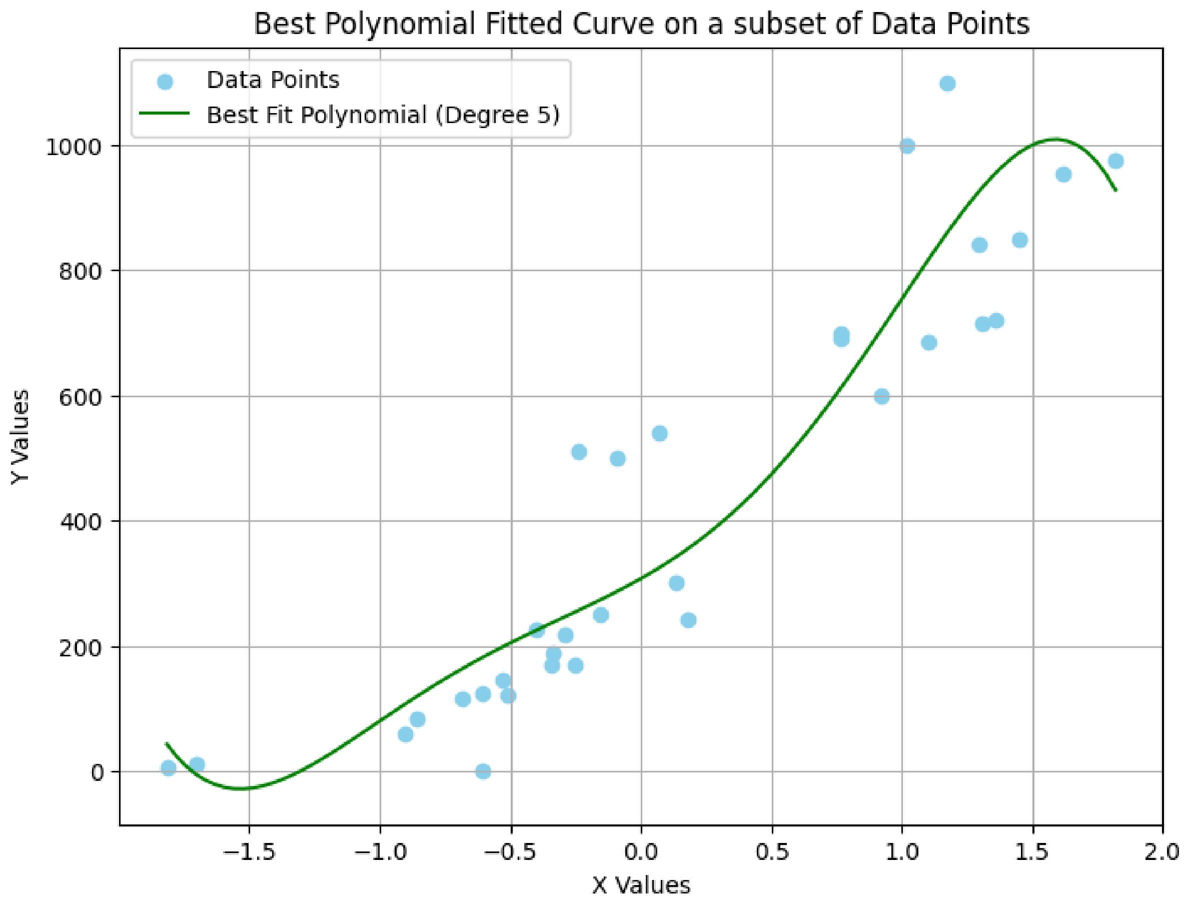


2) <u>Training Error and Testing Error v/s Epochs for all degree of polynomials in [1,9]:</u>



Upon examining the above plots, we observe that the errors remain almost constant for training and testing sets irrespective of the degree of polynomial. Upon closer look, the degree 5 has least error for the testing set and hence is the best-fitting curve. Plot below is a visualization of the same.

3) <u>The best polynomial fitted curve on the data points:</u>



Using the provided equation for errors, we proceed to construct degree 5 regularized polynomial regression models with various regularization values (q = 0.5, 1, 2, 4) for both stochastic and batch gradient descent:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$

Gradients obtained after differentiating the equation after inserting the values of q:

Q = 0.5:

•) $$\nabla J(\theta) = \sum_{n=1}^{N} (y_n - t_n) \left(\phi(x_n) \right) + \frac{\lambda}{4} \sum_{j=1}^{M} |w_j|^{-0.5}$$

Q = 1:

•) $$\nabla J(\theta) = \sum_{n=1}^{N} (y_n - t_n)\, \phi(x_n) + \frac{\lambda}{2} \sum_{j=1}^{M} \{ \mathrm{sign}(w_j) \}$$

Q = 2:

•) $$\nabla J(\theta) = \sum_{n=1}^{N} (y_n - t_n)\, \phi(x_n) + \lambda \sum_{j=1}^{M} w_j$$

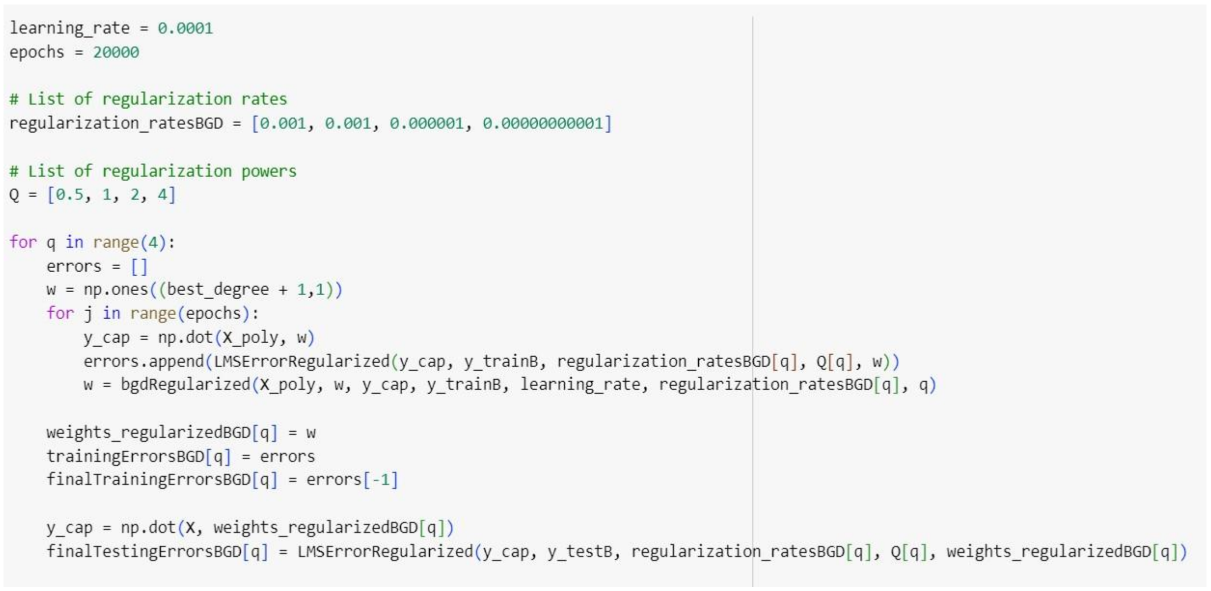
Q = 4:

•) $$\nabla J(\theta) = \sum_{n=1}^{N} (y_n - t_n)\, \phi(x_n) + 2\lambda \sum_{j=1}^{M} (|w_j|)^3$$

We tried with values in the range [0, 1] for each value of q in order to identify the best models. For both Batch and Stochastic Gradient Descent, the optimal model was chosen for each using cross-validation.

Batch Gradient Descent:

New BGD function was implemented as follows:



```
learning_rate = 0.0001
epochs = 20000

# List of regularization rates
regularization_ratesBGD = [0.001, 0.001, 0.000001, 0.00000000001]

# List of regularization powers
Q = [0.5, 1, 2, 4]

for q in range(4):
    errors = []
    w = np.ones((best_degree + 1,1))
    for j in range(epochs):
        y_cap = np.dot(X_poly, w)
        errors.append(LMSErrorRegularized(y_cap, y_trainB, regularization_ratesBGD[q], Q[q], w))
        w = bgdRegularized(X_poly, w, y_cap, y_trainB, learning_rate, regularization_ratesBGD[q], q)

    weights_regularizedBGD[q] = w
    trainingErrorsBGD[q] = errors
    finalTrainingErrorsBGD[q] = errors[-1]

    y_cap = np.dot(X, weights_regularizedBGD[q])
    finalTestingErrorsBGD[q] = LMSErrorRegularized(y_cap, y_testB, regularization_ratesBGD[q], Q[q], weights_regularizedBGD[q])
```

Training Errors:

```
[220.35399140593353, 220.72509209211762, 220.4426608246176,
220.4552541270797]
```

Testing Errors:

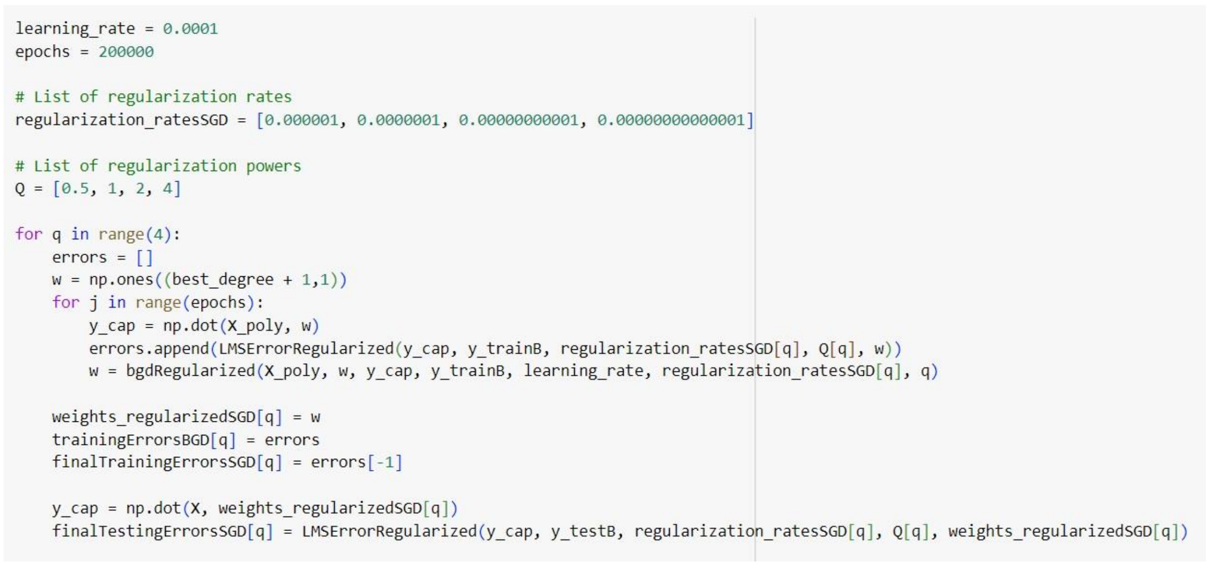
```
[1271.523267005082, 1271.8943526506919, 1271.6119850029208,
1271.6245298670729]
```

Learning rate and λ values for each Q in BGD:

Q	Learning Rate	λ
0.5	0.0001	0.001
1	0.0001	0.001
2	0.0001	0.000001
4	0.0001	0.00000000001

Stochastic Gradient Descent:

SGD for regularization was implemented as following:



```
learning_rate = 0.0001
epochs = 200000

# List of regularization rates
regularization_ratesSGD = [0.000001, 0.0000001, 0.00000000001, 0.00000000000001]

# List of regularization powers
Q = [0.5, 1, 2, 4]

for q in range(4):
    errors = []
    w = np.ones((best_degree + 1,1))
    for j in range(epochs):
        y_cap = np.dot(X_poly, w)
        errors.append(LMSErrorRegularized(y_cap, y_trainB, regularization_ratesSGD[q], Q[q], w))
        w = bgdRegularized(X_poly, w, y_cap, y_trainB, learning_rate, regularization_ratesSGD[q], q)

    weights_regularizedSGD[q] = w
    trainingErrorsBGD[q] = errors
    finalTrainingErrorsSGD[q] = errors[-1]

    y_cap = np.dot(X, weights_regularizedSGD[q])
    finalTestingErrorsSGD[q] = LMSErrorRegularized(y_cap, y_testB, regularization_ratesSGD[q], Q[q], weights_regularizedSGD[q])
```

Training Errors:

```
[220.3252215762344, 220.32523300566217, 220.32519422881185,
                    220.32532340930882]
```

Testing Errors:

```
[1270.9211447225277, 1270.9211561504485, 1270.9211173756025,
                     1270.9212465556022]
```

Learning rate and λ values for each Q in SGD:

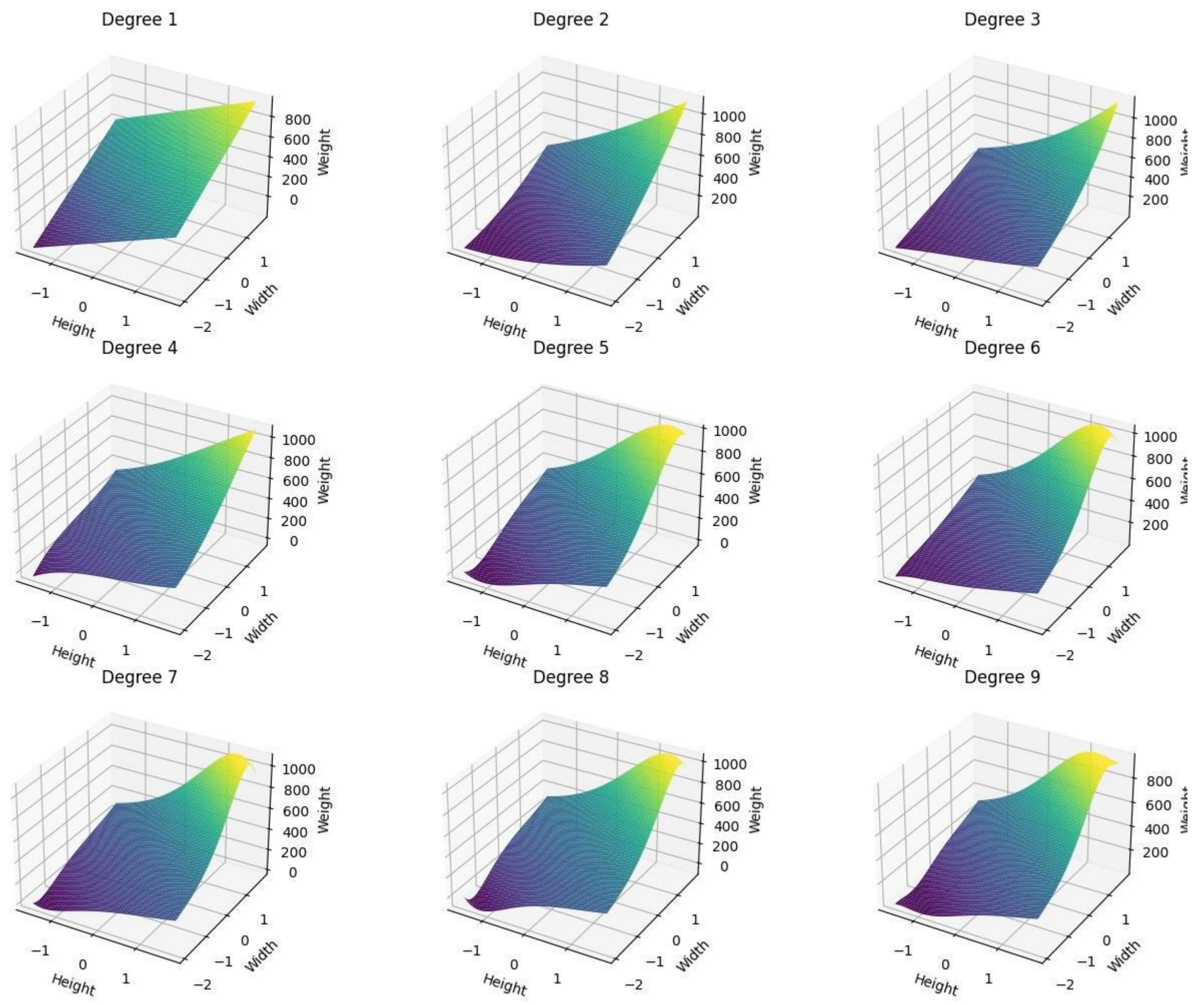
Q	**Learning Rate**	**λ**
0.5	0.0001	0.000001
1	0.0001	0.0000001
2	0.0001	0.00000000001
4	0.0001	0.00000000000001

It was observed that for the same number of iterations, BGD gives less LMS error compared to SGD. However, upon increasing the iterations of SGD, we get the same errors for it.

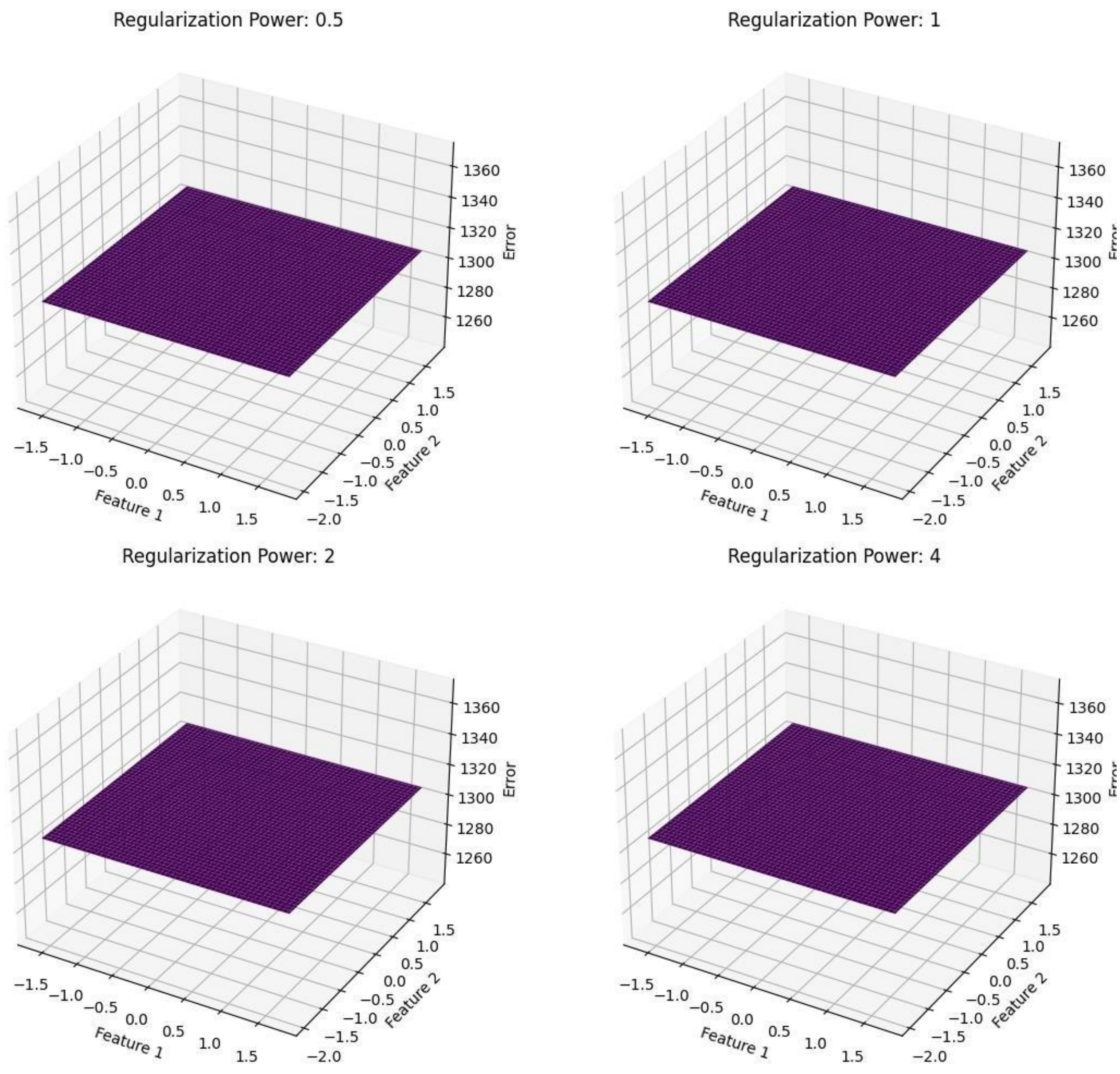
Task 3: Graph Plotting:

Using the Python Matplotlib module, we produced surface plots for the four best regularized linear regression models, the nine polynomial regression models (degrees 0 to 9), and the nine polynomial regression models.

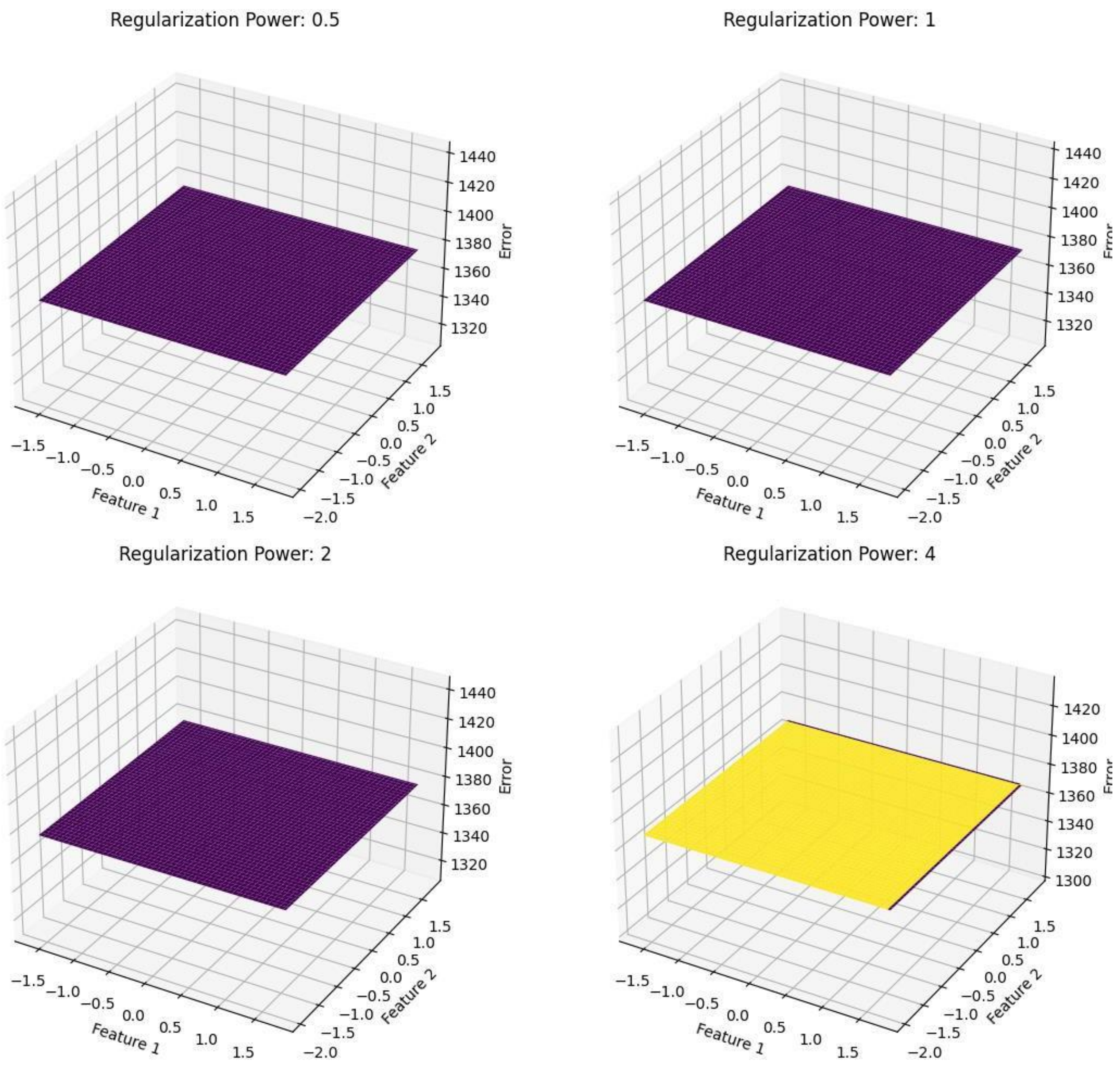
Plot 1: Surface Plots for Non-Regularized Polynomials:



Plot 2: Surface Plots for Regularized Polynomials (Batch Gradient Descent):



Plot 3: Surface Plots for Regularized Polynomials (Stochastic Gradient Descent):



Task 4: Comparative Analysis:

Regularization was applied to the polynomial regression models using the provided values of q.

Here are the training and testing errors (MSE) for the SGD method with different q values, learning rates, and λ values:

q	Learning Rate	λ	Training Error (MSE)	Testing Error (MSE)
0.5	0.0001	0.000001	278.437	428.720
1	0.0001	0.0000001	278.485	429.570
2	0.0001	0.00000000001	278.523	430.248
4	0.0001	0.00000000000001	278.729	432.788

Here are the training and testing errors (MSE) for the BGD method with different q values, learning rates, and λ values:

q	Learning Rate	λ	Training Error (MSE)	Testing Error (MSE)
0.5	0.0001	0.001	271.268	444.481
1	0.0001	0.001	271.697	444.910
2	0.0001	0.000001	271.386	444.598
4	0.0001	0.00000000001	271.449	444.662

When we look at the training and testing errors, we see that the training and testing errors stay pretty much the same as q goes up as long as λ is high enough. The reduction in errors drops greatly when the values are changed even more, so these values were used as the final errors for the q and gradient descent methods.