MACHINE, DATA AND LEARNING

ASSIGNMENT 1: QUESTION 1

TEAM NUMBER: 86

TEAM MEMBERS:
Tathagata Raha (2018114017)
Arathy Rose Tony(2018101042)

TASK

Calculate the bias and variance of a dataset that is not sampled yet, and then predict the degree of the best fit curve

Re-Sample Data

You have been provided a dataset consisting of pairs (xi,yi). It can be loaded into your python program using pickle.load() function. Split the dataset into training and testing(90:10 split). Now divide the train set into 10 equal parts randomly, so that you will get 10 different dataset to train your model.

Train Data Models

After re-sampling data, you have 11 different datasets (10 train sets and 1 test set). Train a linear classifier on each of the 10 train set separately, so that you have 10 different classifiers or models. You have 10 different models or classifiers trained separately on 10 different training set, so now you can calculate the bias and variance of the model. You need to repeat the above process for the following class of functions:
$y = mx + c$
$y = ax^2 + bx + c$
$y = ax^3 + bx^2 + cx + d$ ...
And so on up till polynomial of degree 9. Tabulate the values of bias and variance and also write a detailed report explaining how bias and variance changes as you vary your function classes.

Note: Whenever we are talking about the bias and variance of model, it refers to the average bias and variance of the model over all the test points.

SOME BASIC DEFINITIONS

Bias

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. A model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to a high error on training and test data.

$$Bias^2 = (E[\hat{f}(x)] - f(x))^2$$

where $f(x)$ represents the true value, $\hat{f}(x)$ represents the predicted value

Variance

Variance is the variability of a model prediction for a given data point. The variance is how much the predictions for a given point vary between different realizations of the model.

$$Variance = E[\hat{f}(x) - E[\hat{f}(x)]]^2$$

where $f(x)$ represents the true value, $\hat{f}(x)$ represents the predicted value

Error

Noise is a unwanted distortion in data. Noise is anything that is spurious and extraneous to the original data, that is not intended to be present in the first place, but was introduced due to faulty capturing process.

Bias-Variance TradeOff

If our model is too simple and has very few parameters then it has high bias and low variance. On the other hand, if our model has a large number of parameters then it’s going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

EXPLANATION OF THE CODE/APPROACH

Header files included

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
import numpy
import random
import pickle
import pandas
import matplotlib.pyplot as plt
```

Some global variables that can be used to check specific outputs

```
debug = 1  # 1 if you want to see the variable values during the program execution
graphing = 1  # 1 to see the graphs
```

STEP 1: LOADING THE DATASET AND VISUALISING IT

Load the dataset

Here we load the data_set from data.pkl file which is stored in the same directory as the current notebook.

```
f = open('data.pkl', 'rb')
data_set = pickle.load(f)
data_set_size = len(data_set)
f.close()
```

Now the data_set contains the entire data set points in the format <x,y>. The total number of elements in the data set is stored in data_set_size.

Split The Dataset Into X and Y

Here we split the <x,y> pairs into x and y separate arrays

```
x = data_set[:, 0]
y = data_set[:, 1]
```

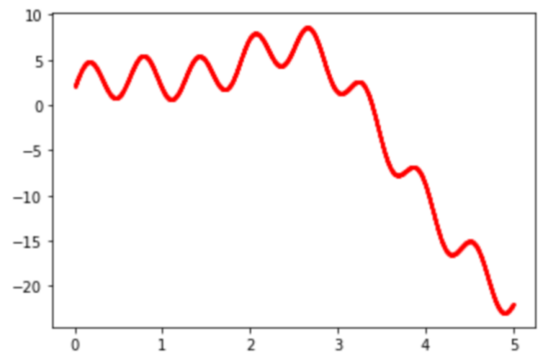
Graphing the given dataset

Here we plot the given dataset, just to get the feel of the dataset provided.

```
fig = plt.figure()
```

```
plt.plot(x, y, 'r.', markersize=2)
plt.show()
```

GRAPH OUTPUT



STEP 2: RESAMPLING DATA SETS

Split The Dataset Into Testing And Training Datasets

This line splits the dataset into the testing and training datasets in the ratio 1:9

```
xTrain, xTest, yTrain, yTest = train_test_split(
   x, y, test_size=0.1, random_state=3, shuffle=True)
test_data_size = len(xTest)
train_data_size = len(xTrain)
```

Split The Training Dataset Into 10 Different Training Datasets

Here, we run a loop 10 times and store the train dataset values from start_in to the end_in as a numpy array in a list (X_train_data_sets and Y_train_data_sets), and update the values of start_in and end_in in each iteration of the loop.

```
X_train_data_sets = list()
Y_train_data_sets = list()
start_in = 0
train_data_sets_size = int(train_data_size/10)
end_in = train_data_sets_size
for i in range(10):
   X_train_data_sets.append(xTrain[start_in:end_in])
```

```
    Y_train_data_sets.append(yTrain[start_in:end_in])
    start_in += train_data_sets_size
    end_in += train_data_sets_size
```

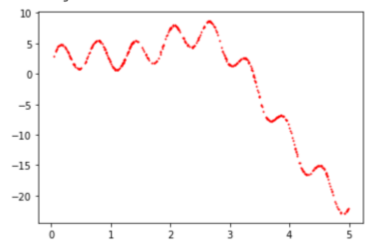
Graphing Each Of The Training Datasets

Here we graph each of the training datasets separately. (to check if the datasets are sampled properly)

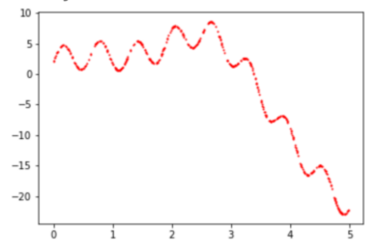
```
for i in range(10):
    print("Training set ",i)
    fig = plt.figure()
    plt.plot(X_train_data_sets[i], Y_train_data_sets[i], 'r.', markersize=2)
    plt.show()
```

GRAPH OUTPUT

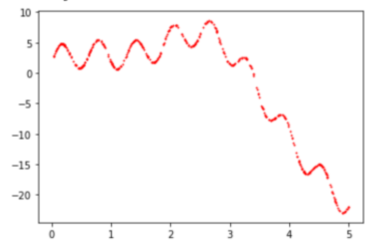
```
Training data  0
```



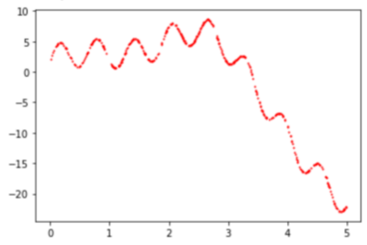
```
Training data  1
```



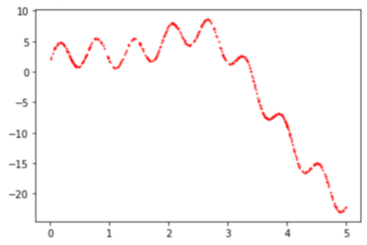
```
Training data  2
```



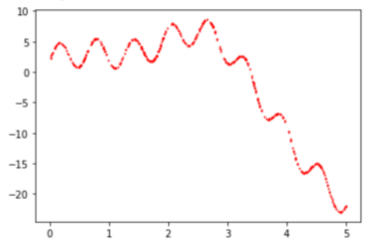
Training data 3



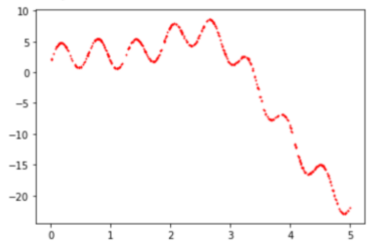
Training data 4



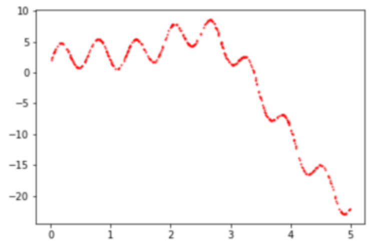
Training data 5



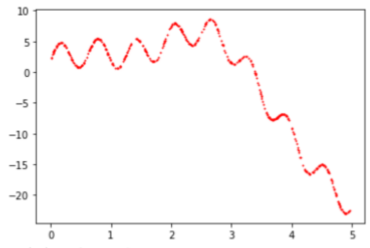
Training data 6



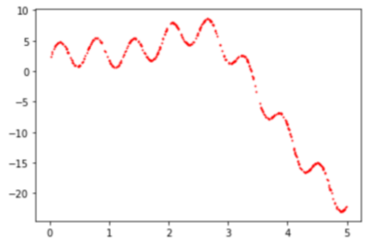
Training data 7



Training data 8



Training data 9



STEP 3: TRAINING A MODEL

Function just to store what exactly to be done to train a polynomial regression model of a given degree and given training dataset number

The following function returns a polynomial regression model for the given degree. I use this to find the lines required to make a model, of a given degree

```
def create_polynomial_regression_model(degree,i):
    i = 0
    # prepare the matrix of the powers of x
    poly_features = PolynomialFeatures(degree=degree)
    # transpose the x row matrix into a column matrix
    x = X_train_data_sets[i][:, numpy.newaxis]
    # get a matrix containing the higher powers of X in the format: [1 X X^2 X^3 ...]
    X_train_poly = poly_features.fit_transform(x)
    poly_model = LinearRegression()
    # fit the transformed features to Linear Regression
    poly_model.fit(X_train_poly, Y_train_data_sets[0])
    y_test_predict = poly_model.predict(
        poly_features.fit_transform(xTest))    # predicting on test data-set
    return poly_model
```

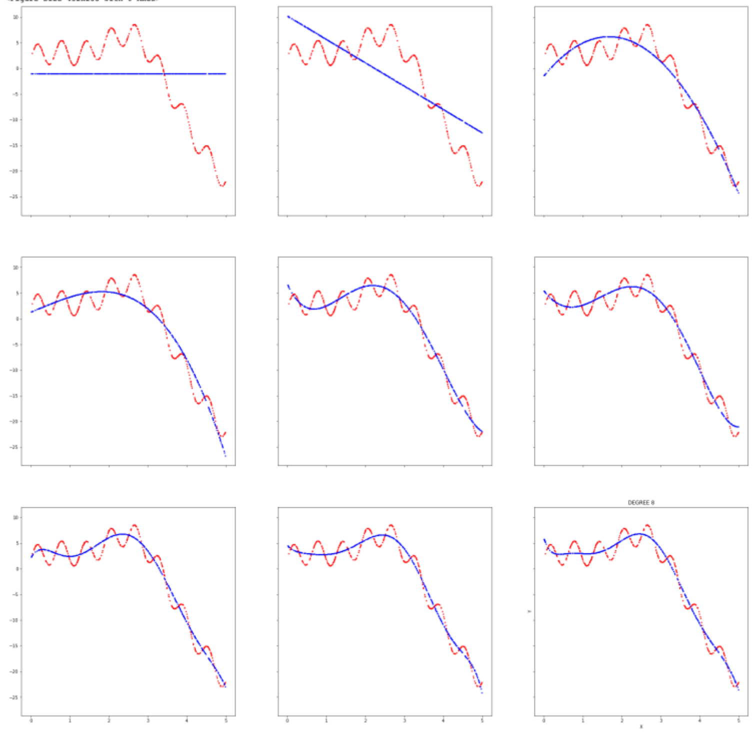
Plotting A Graph Of The Trained Polynomial Regression Model

Here, we take each of the training datasets, and plot the training dataset points and the values predicted by the model on the test dataset points, to visualise the provided data.
For each training set, 9 graphs are plotted, each corresponding to the model of each degree (from 0 to 9)

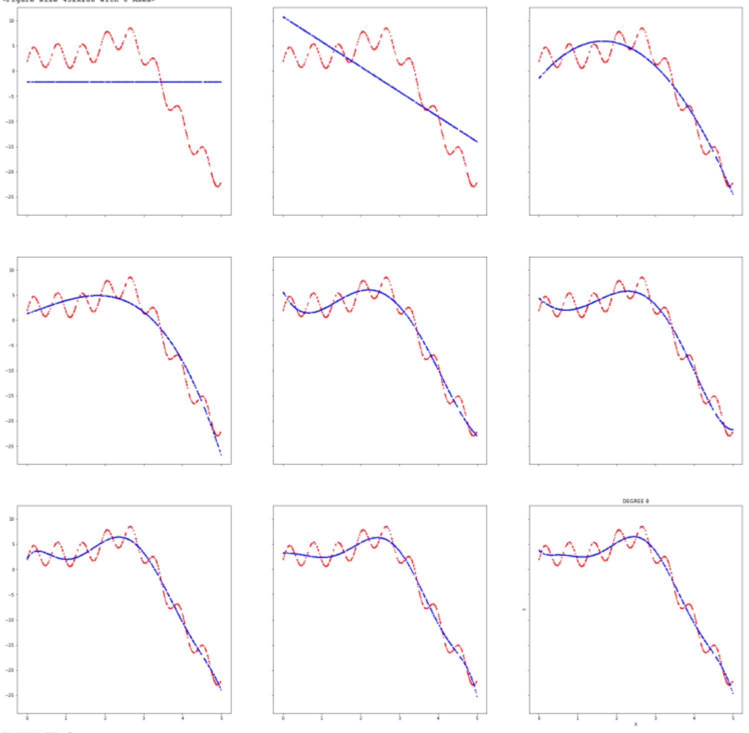
```
for i in range(10):
    print("TRAINING SET ", i)
    f = plt.figure()
    f, axes = plt.subplots(nrows=3, ncols=3, sharex=True, sharey=True, figsize=(30, 30))
    x = X_train_data_sets[i][:, numpy.newaxis]  # transposing it
    y = Y_train_data_sets[i]
    for degree in range(0, 9):
        axes[int(degree/3)][int(degree % 3)].plot(x, y, 'r.', markersize=4)
        poly_features = PolynomialFeatures(degree=degree+1)
        X_train_poly = poly_features.fit_transform(x)
        poly_model = LinearRegression()
        poly_model.fit(X_train_poly, Y_train_data_sets[i])
        y_test_predict = poly_model.predict(
            poly_features.fit_transform(xTest[:, numpy.newaxis]))
        axes[int(degree/3)][int(degree % 3)].plot(xTest[:, numpy.newaxis], y_test_predict, 'b.', markersize=4)
        plt.title("DEGREE "+str(degree+1))
        plt.xlabel("X")
        plt.ylabel("Y")
    plt.show()
```

GRAPH OUTPUT

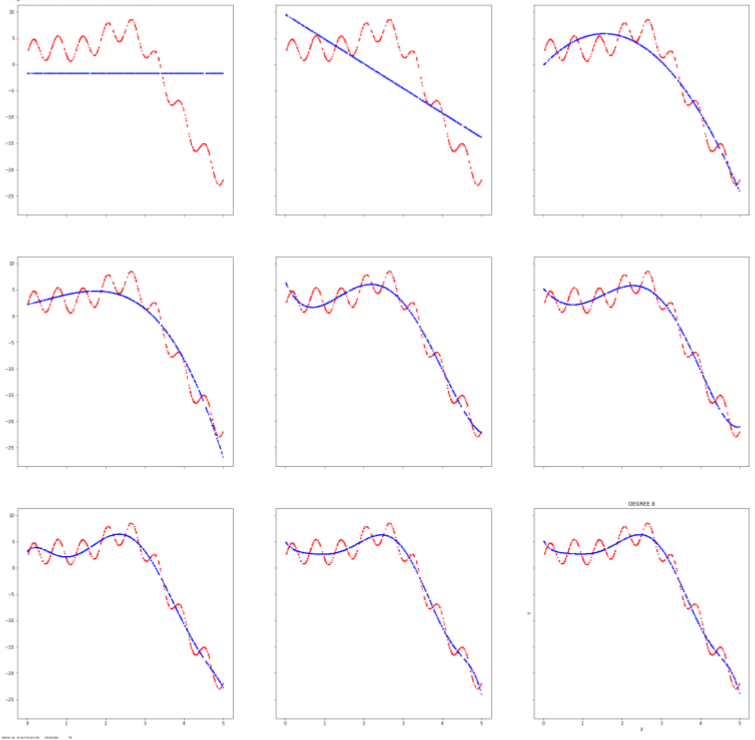
```
TRAINING SET  0
<Figure size 432x288 with 0 Axes>
```



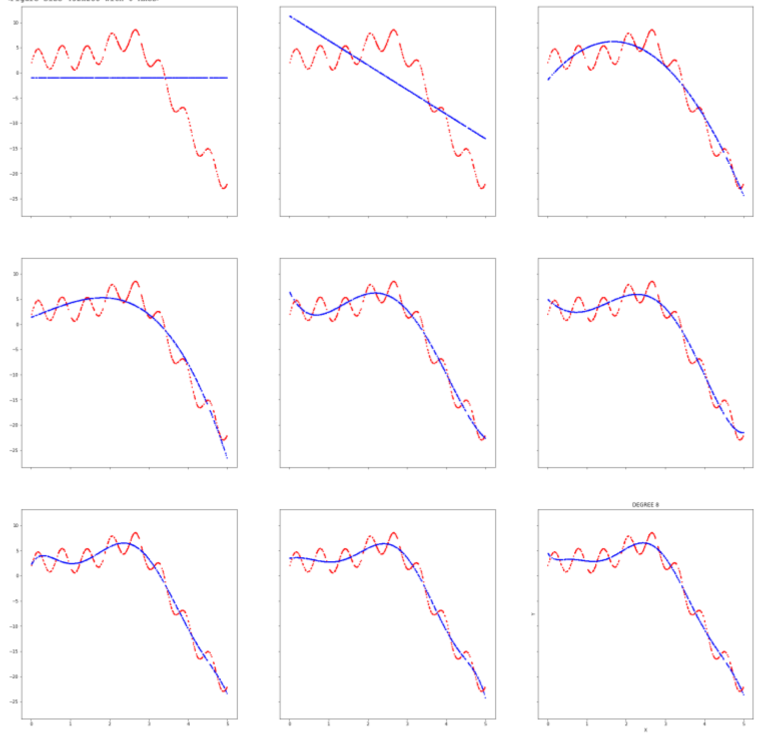
```
TRAINING SET  1
<Figure size 432x288 with 0 Axes>
```



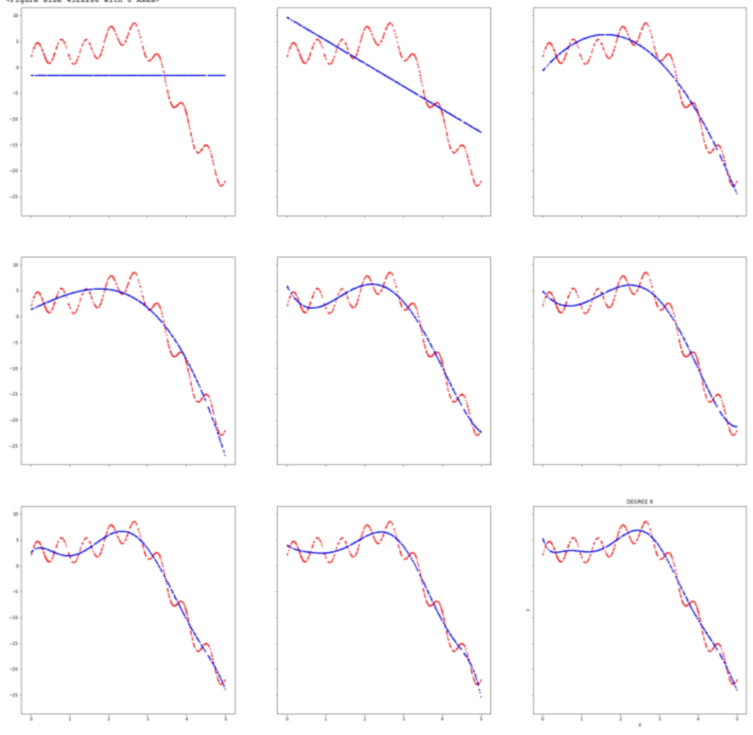
```
TRAINING SET  2
<Figure size 432x288 with 0 Axes>
```



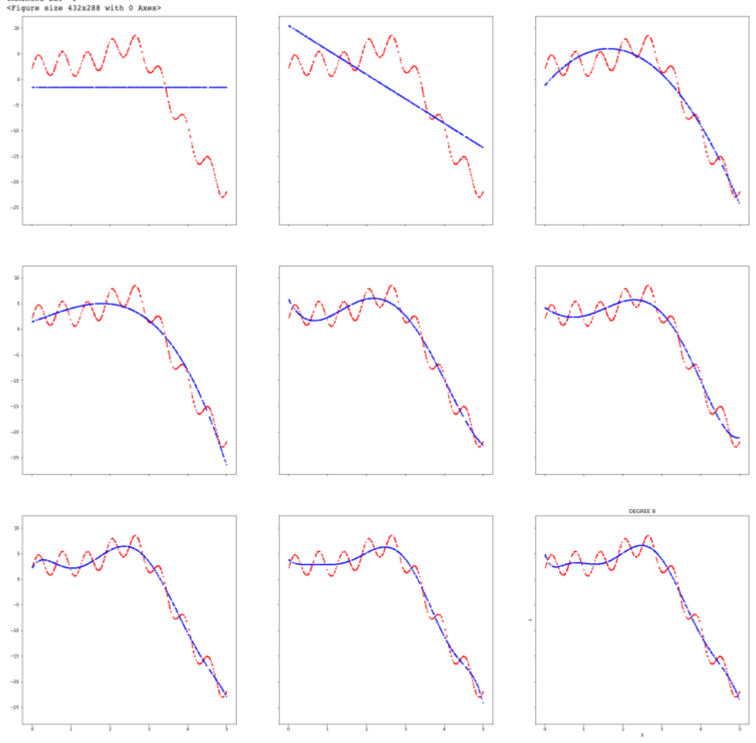
```
TRAINING SET  3
<Figure size 432x288 with 0 Axes>
```



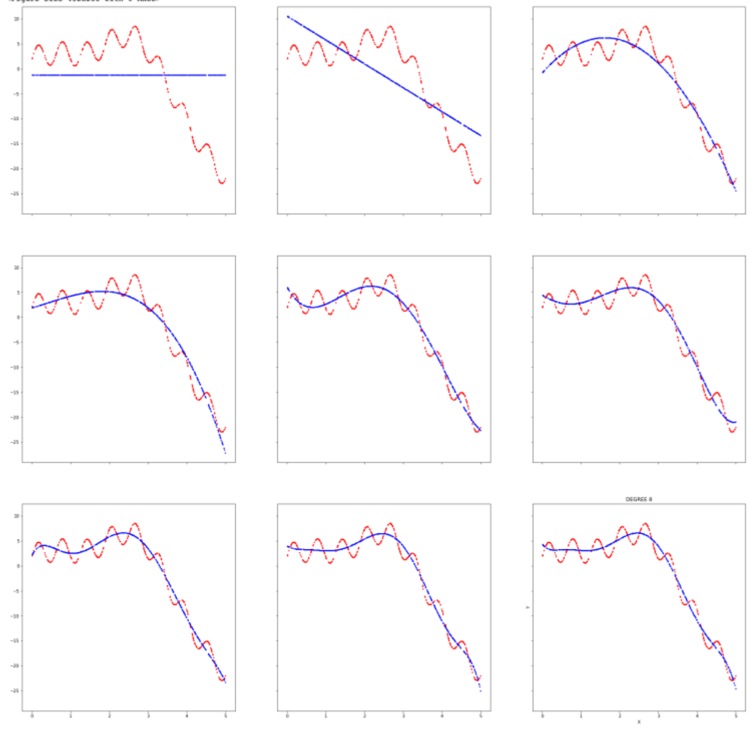
```
TRAINING SET  4
<Figure size 432x288 with 0 Axes>
```



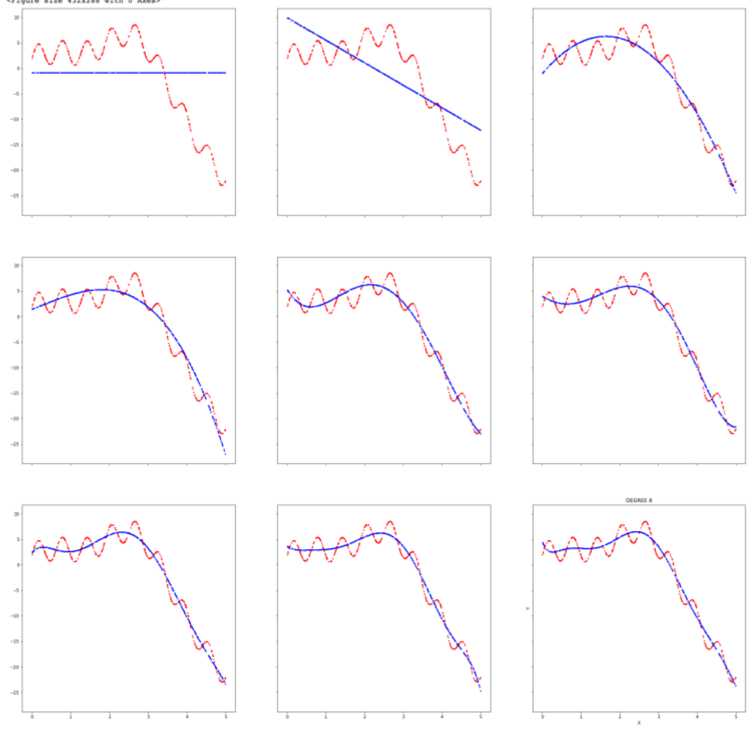
```
TRAINING SET  5
<Figure size 432x288 with 0 Axes>
```



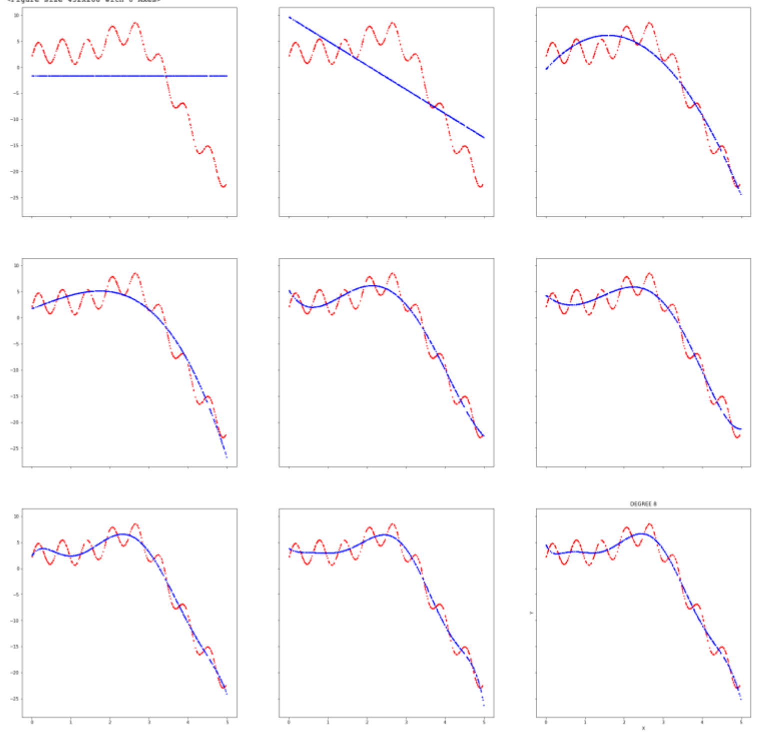
```
TRAINING SET  6
<Figure size 432x288 with 0 Axes>
```



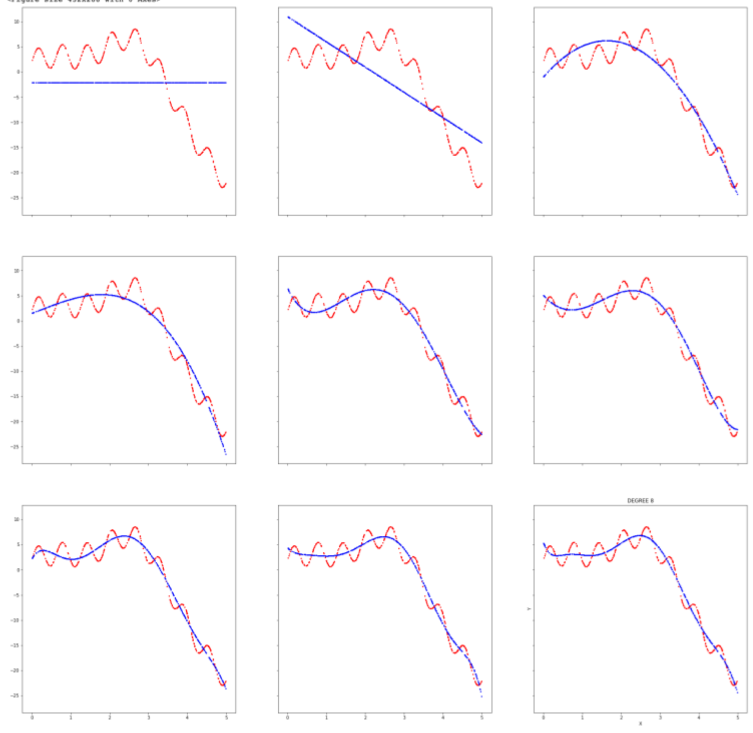
```
TRAINING SET  7
<Figure size 432x288 with 0 Axes>
```



```
TRAINING SET  8
<Figure size 432x288 with 0 Axes>
```



```
TRAINING SET  9
<Figure size 432x288 with 0 Axes>
```



STEP 4: CALCULATE THE BIAS AND VARIANCE OF THE MODEL

Get the list of all the predicted values

First we get the list of all the y predicted values for all the models and for all the degrees separately in a 2-D array.
Here, for each model of each degree, we get the predicted y values for the given test datasets.
The values are stored as follows: y[train_data_set_no][degree]

```
y_predicted = []
for i in range(10):
    x = X_train_data_sets[i][:, numpy.newaxis]  # transposing it
    y = Y_train_data_sets[i]
    temp = []
    for degree in range(0, 9):
        poly_features = PolynomialFeatures(degree=degree)
        X_train_poly = poly_features.fit_transform(x)
        poly_model = LinearRegression()
        poly_model.fit(X_train_poly, Y_train_data_sets[i])
        y_test_predict = poly_model.predict(
            poly_features.fit_transform(xTest[:, numpy.newaxis]))
        temp.append(y_test_predict)
    y_predicted.append(temp)
print(len(y_predicted[0][0]))
```

Function for calculating the bias and the variance

Then we calculate the bias and variance as follows:

- For a given degree we append the values of the y_predicted for each model to a list
- Convert this list to a numpy array y_predicted_part
- Calculate the bias of this list by subtracting the mean of the model from the testing dataset
- Bias corresponding to the models of a given degree is the mean of this list
- Similarly calculate the variance of this list
- Variance corresponding to the models of a given degree is the mean of this list

```
def find_bias_variance(order):
    y_predicted_part = []
    for i in range(10):
        y_predicted_part.append(y_predicted[i][order])
    y_predicted_part = numpy.asarray(y_predicted_part)
    bias = numpy.abs(numpy.mean(y_predicted_part, axis=0) - yTest)
    variance = numpy.var(y_predicted_part, axis=0)
    return(numpy.mean(bias), numpy.mean(variance))
```

Then we call the function as follows, in order to populate the lists, bias and variance.

```
Bias = []
variance = []
for I in range(9):
    b, v = find_bias_variance(i)
    bias.append(b)
    variance.append(v)
print("Bias:", bias)
print("Variance:", variance)
```

The lists, bias and variance, now contain the bias and variance corresponding to a particular degree.

OUTPUT

```
Bias: [7.4497698199296085, 4.857085127491957, 2.0146837126148074,
1.8992438905143325, 1.514351950453967, 1.4989046988863748, 1.408943169493006,
1.4010330503465502, 1.4148533359268889]

Variance: [0.18820342581647526, 0.21159174599021216, 0.04072585310763539,
0.02740164535556403, 0.02842401971640762, 0.02777963018821782,
0.030423566408997745, 0.03850332626175187, 0.04271941216324869]
```

Tabulate the values

We use the pandas library in order to display the required items in a table format

```
final_table = dict()
```

```
final_table["DEGREE"] = range(1, 10)
final_table["BIAS"] = bias
final_table["BIAS^2"] = list(numpy.array(bias)**2)
final_table["VARIANCE"] = variance
final_table["MSE"] = list(numpy.array(final_table["BIAS^2"])+numpy.array(variance))
df = pandas.DataFrame(final_table)
print(df)
```

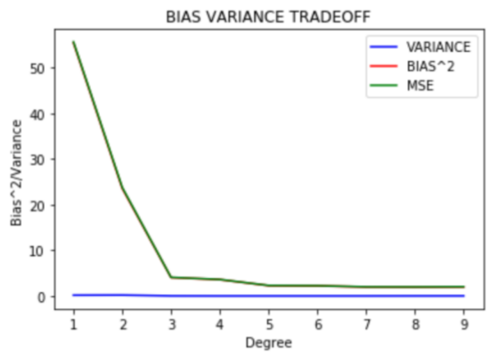
OUTPUT

```
   DEGREE      BIAS     BIAS^2  VARIANCE        MSE
0       1  7.449770  55.499070  0.188203  55.687274
1       2  4.857085  23.591276  0.211592  23.802868
2       3  2.014684   4.058950  0.040726   4.099676
3       4  1.899244   3.607127  0.027402   3.634529
4       5  1.514352   2.293262  0.028424   2.321686
5       6  1.498905   2.246715  0.027780   2.274495
6       7  1.408943   1.985121  0.030424   2.015544
7       8  1.401033   1.962894  0.038503   2.001397
8       9  1.414853   2.001810  0.042719   2.044529
```

Plot the bias-variance tradeoff

```
plt.plot(final_table["DEGREE"], final_table["VARIANCE"], color="blue")
plt.plot(final_table["DEGREE"], final_table["BIAS^2"], color="red")
plt.plot(final_table["DEGREE"], final_table["MSE"], color="green")
plt.title("BIAS VARIANCE TRADEOFF")
plt.xlabel("Degree")
plt.ylabel("Bias^2/Variance")
plt.legend(["VARIANCE", "BIAS^2", "MSE"])
plt.show()
```

GRAPH OUTPUT



OBSERVATIONS AND INFERENCES

From the bias variance tradeoff graph, we can make the following observations:

- As the degree of the polynomial regression that we try to fit the data with increases, the bias value decreases. This is because of overfitting; that is the model now trained, has a lot of additional features and hence is able to fit in more dataset points, leading to less bias.
- As the degree of the polynomial regression that we try to fit the data with increases, the variance value increases. This is because of overfitting; now, the model tries to predict the noise component of the dataset too, hence leading to higher values of variance

From the bias variance tradeoff graph, we can make the following observations:

- As the degree of the polynomial regression that we try to fit the data with increases, the bias value decreases. This is because of overfitting; that is the model now trained, has a lot of additional features and hence is able to fit in more dataset points, leading to less bias.
- As the degree of the polynomial regression that we try to fit the data with increases, the variance value increases. This is because of overfitting; now, the model tries to predict the noise component of the dataset too, hence leading to higher values of variance

From the above plot, the given dataset is more likely to have a degree much larger than 10 because of the following reasons:

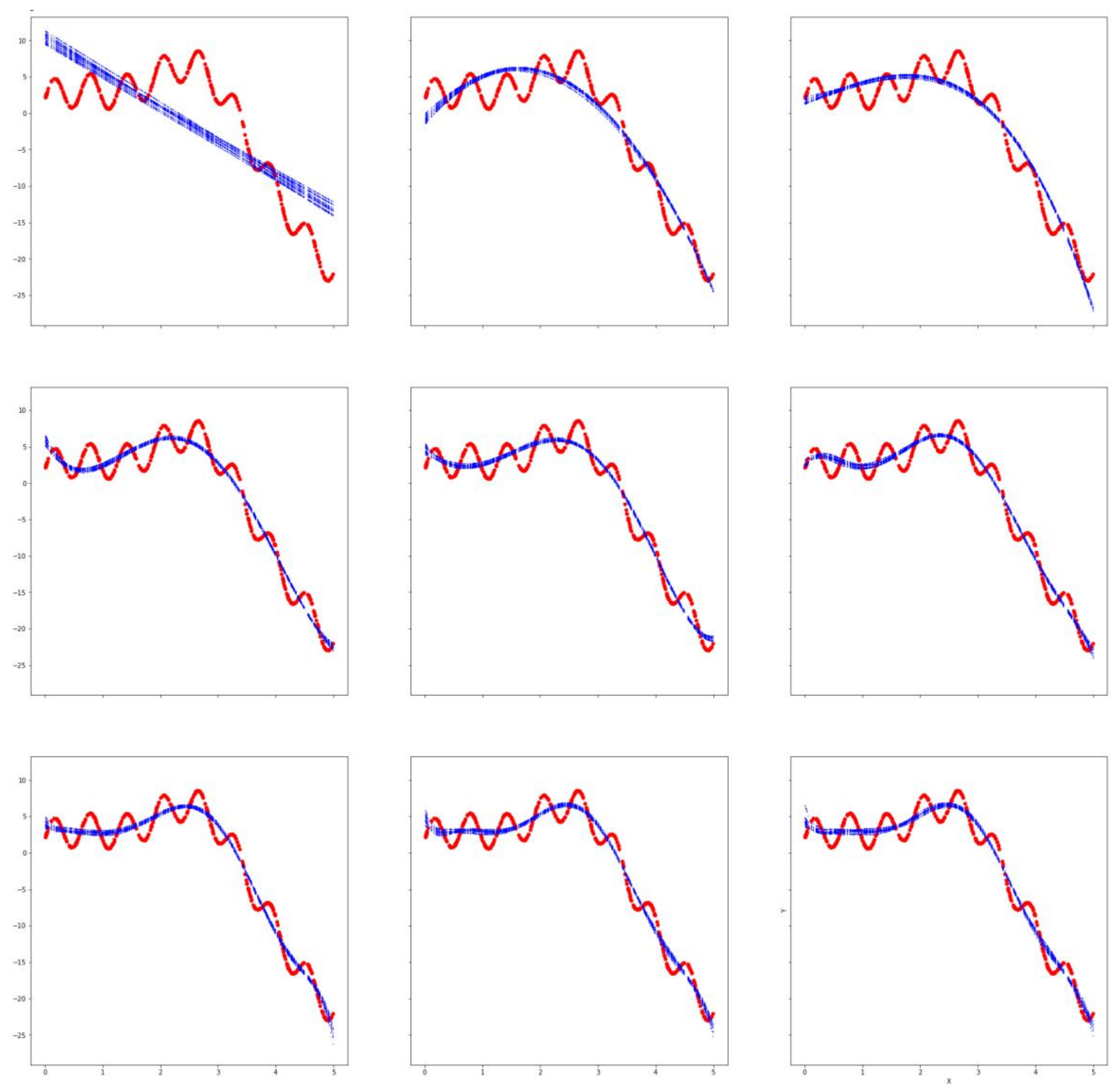
- Variance slightly decreases as the model complexity increases. This suggests that there is almost no noise in the system and the given dataset is well-structured.
- The linear model is not able to model it correctly (very high bias in the data that suggests underfitting).
- As the model complexity increases, we see that the model starts to better fit in the model (the bias decreases rapidly). This continues till degree=17, which suggests that the model is more likely to be of order 17.
- In the plot, the variance is much less compared to the bias because our model complexity (1 to 9) is much lower than the
- In the Bias vs. Variance plots, the variance is negligible compared to the bias, since our model is way too low order (all models where degree is much lower than 17 will be very high bias to variance).

FITTING THE TRAINED MODEL TO THE TESTING DATASET FOR DISPLAYING THE LINE OF BEST FIT

The code snippet

```
f = plt.figure()
f, axes = plt.subplots(nrows = 3, ncols = 3, sharex=True, sharey = True,figsize=(30,30))
for degree in range(0,9):
    xtemp=numpy.concatenate([xTest for i in range(20)])
    y_predicted_part=[]
    for i in range(20):
        y_predicted_part.append(y_predicted[i][degree])
    ytemp=numpy.array(y_predicted_part).reshape(-1)
    axes[int((degree)/3)][int((degree)%3)].plot(xTest, yTest, 'r.',markersize=10)
    axes[int((degree)/3)][int((degree)%3)].plot(xtemp, ytemp,'b.',markersize=1)
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Output



10
5
0
−5
−10
−15
−20
−25
0
1
2
3
4
5
X
Y