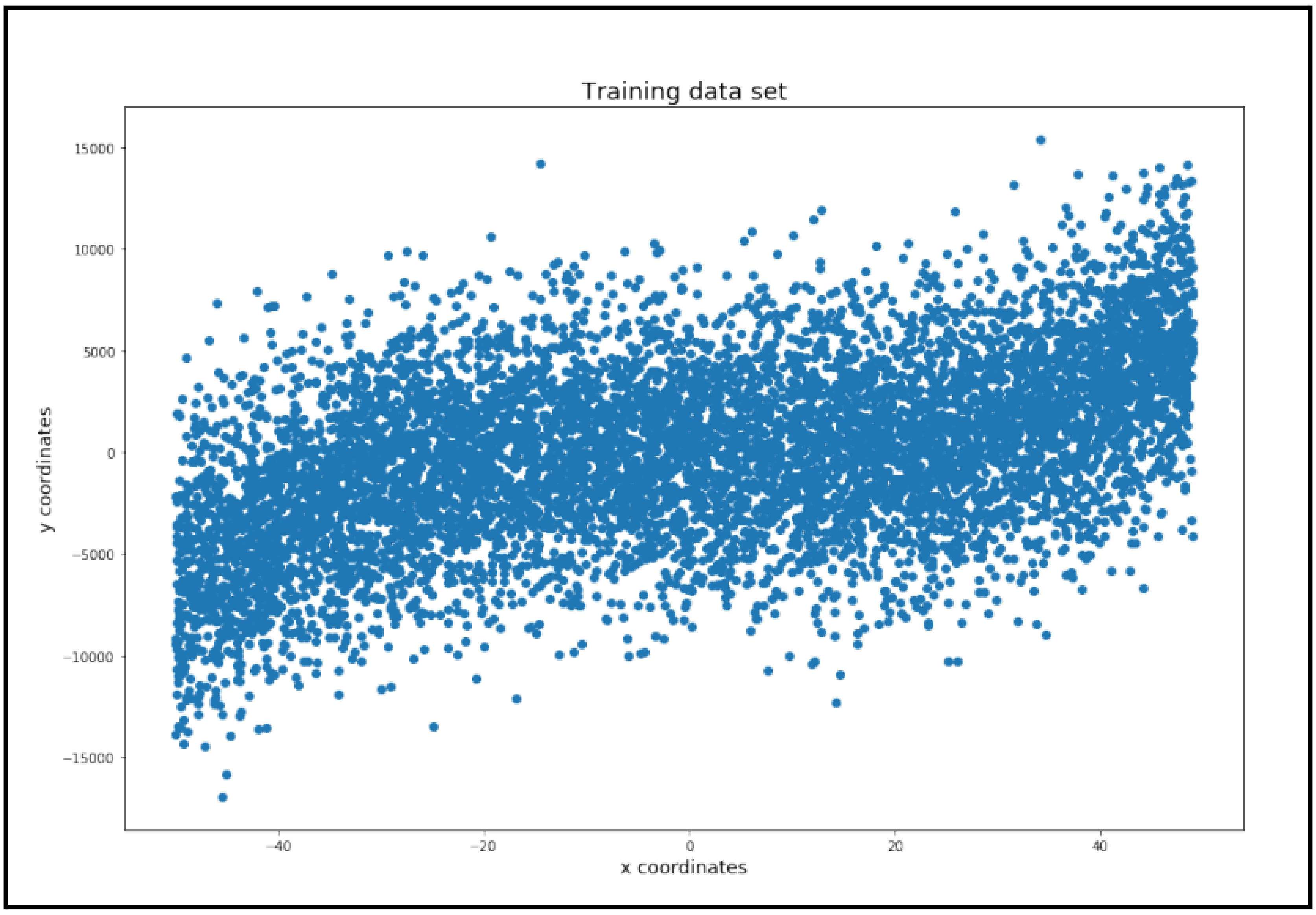
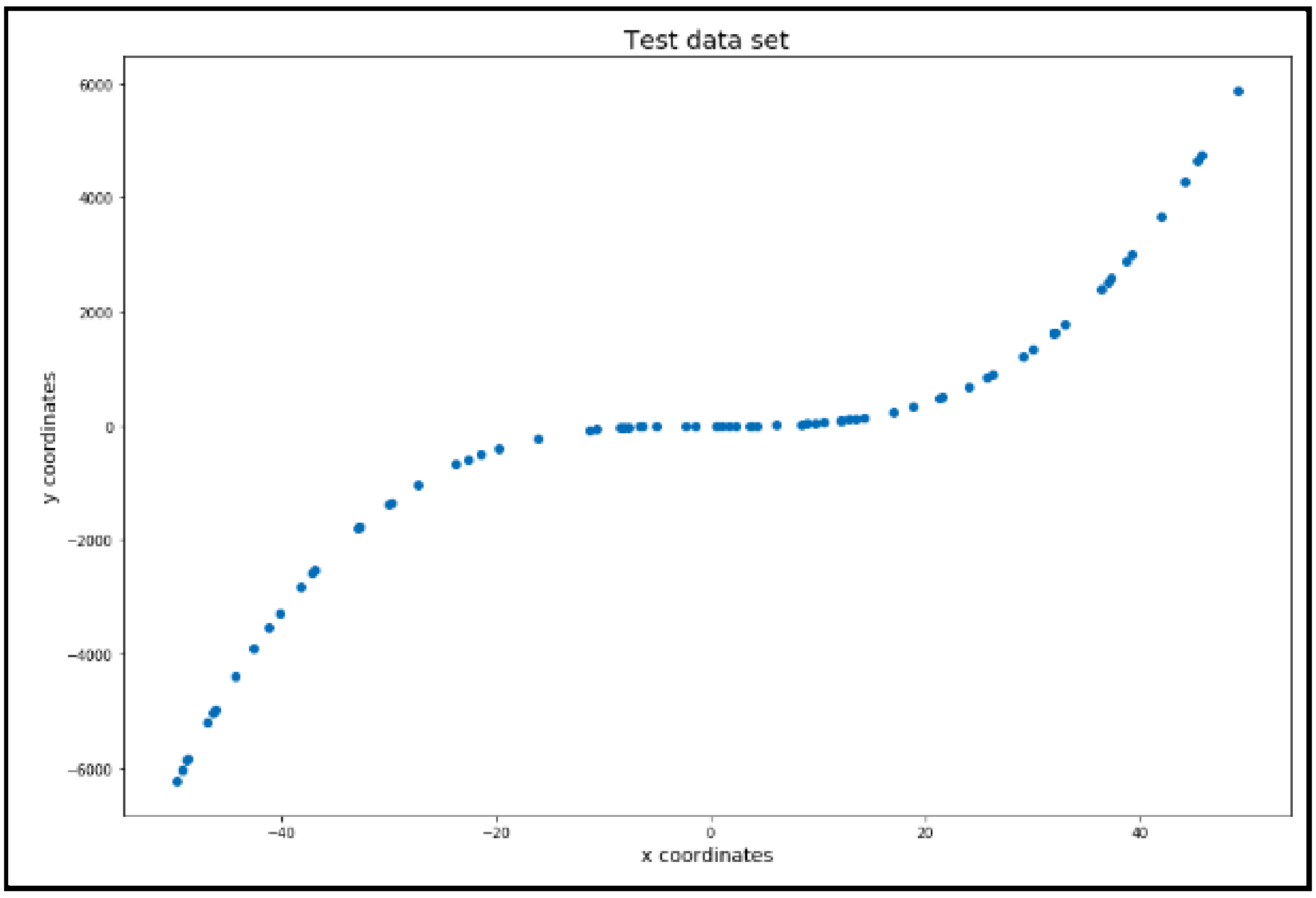
A detailed study of BIAS-VARIANCE TRADEOFF

By:
Anmol Agarwal 2019101068
Raj Maheshwari 2019101039

Plot of training data set



Plot of test data set



A brief explanation of what function the method, LinearRegression().fit() performs.

Let us assume that our current training dataset D consists of ‘n’ instances.
Let the degree of the polynomial be ‘deg’. Now, there will be (deg+1) coefficients (after including the constant term as well).
The given function takes 2 inputs:
* Matrix containing the input features for each instance (n X (deg+1)) matrix
* Corresponding actual y (the ‘y’ value which was observed in the real world) which is a (n X 1 array)

As far as the theoretical part of linear regression is concerned, we are aware of the **gradient descent method** which is used to obtain the coefficients which reduce the average value of the squared error to be the minimum. Also, in gradient descent we usually make small steps. Thus, for the jth coefficient, we do the following:

```
Repeat until convergence {
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}
```

Here, alpha is predetermined and can be manipulated.

- If the alpha is very small, chances are that the number of iterations required for convergence will be very large.
- If alpha is very large, chances are that we escape the minima(jump over the local minima).

A thing we noticed

We noticed that the LinearRegression.fit() does not always obtain optimal coefficients.
We **tailormade** a test case for function f(x)=1+2x+3x^2+3x^3+2x^4+x^5 and kept the training dataset size as 2 for the 1st case and 9 for the second case. In the 1st case, the coefficients produced were not optimal and overall MSE was not zero which proves that the **internal**

implementation does not go all the way but stops after some fixed precision/error threshold.(refer to manipulation of alpha). But in the second case, since the number of training examples is more, the error with non-optimal coefficients would have been large and hence, the LinearRegression.fit() would have given some more iterations and reached the optimal value.

Assignment specific use of LinearRegression.fit():

Each training data set size is 8000/10=800 and so, the first argument is a matrix of size (800 X (deg+1)) and the second argument is an array of (800 X 1).

Details regarding training of the models

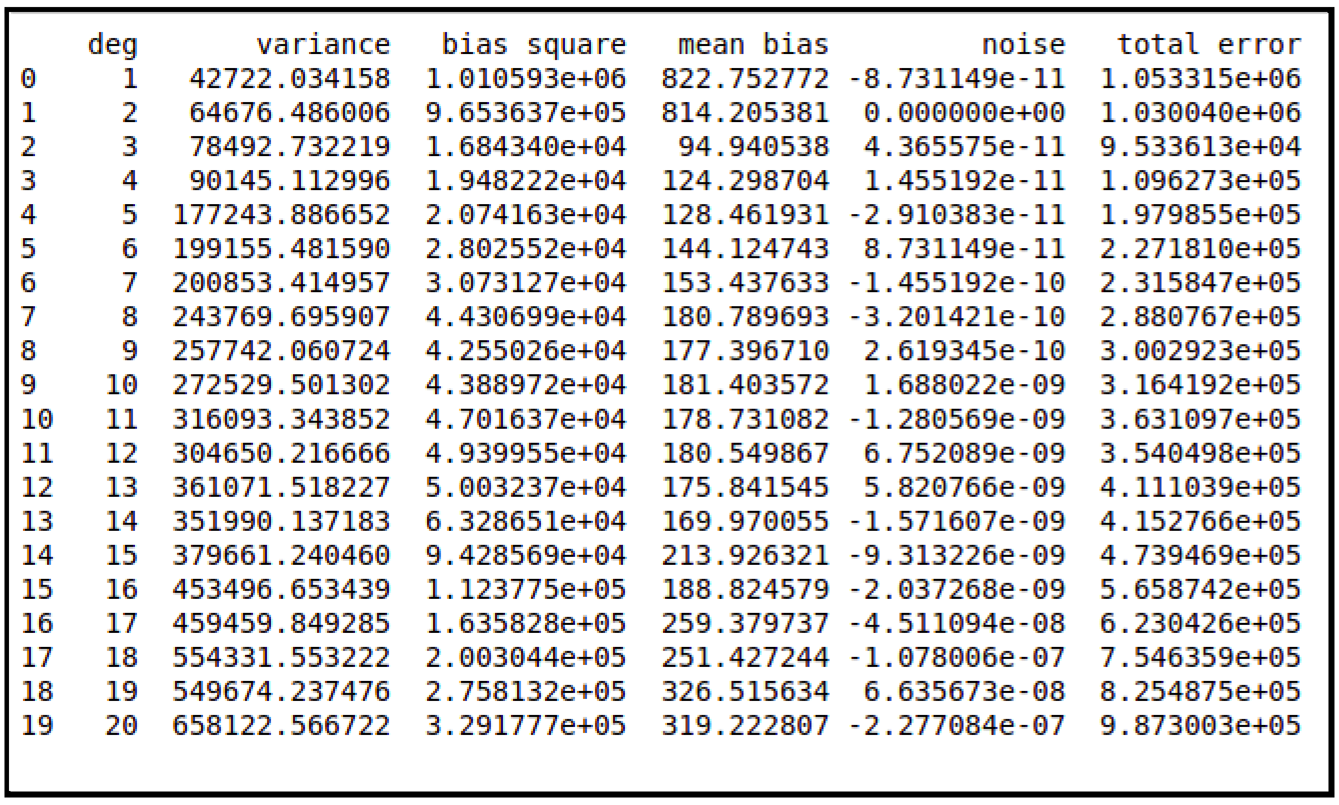
For the given data sets, we are trying to fit a linear regression model on the training dataset for polynomials ranging from degree 1 to 20.
Let's say that our regression function is a polynomial of degree 'n' and hence, the polynomial would be determined by the (n+1) coefficients (after including the constant term as well).
Also, let the size of the entire training set be 'sz'. We are **partitioning the training dataset into 10 equal parts** (after randomly shuffling the entries in the training dataset) each of size 'sz/10' and then training a polynomial of degree 'n' on each of these partitions. In other words, we are finding the (n+1) coefficients by linear regression on each of the partitions as a training dataset.
So, for a fixed polynomial degree, we will have **10 classifiers**. **It is worthwhile to note that each of these classifiers can be represented by a (n+1) length vector which corresponds to the (n+1) coefficients of the 'n' degree polynomial.**

To summarize,

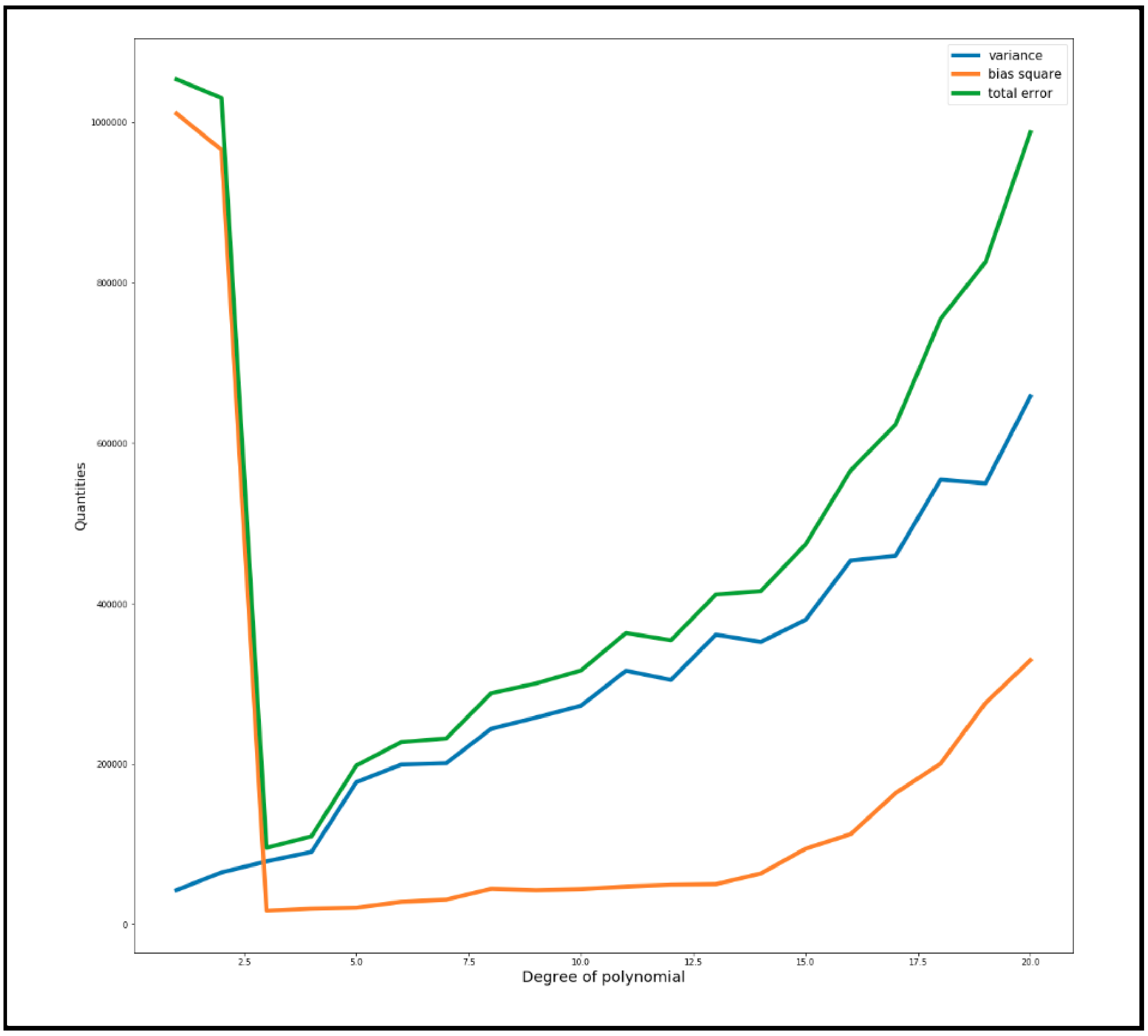
- Number of models we are using: 20
- Number of **different versions(realizations) of a model** using a particular degree: 10

In the outer loop, we are iterating over integers from 1 to 20 (the degrees of the polynomials). In the inner loop, we are training a polynomial of degree 'deg' 10 times (each time on a different partition of the dataset) and using the coefficients obtained to calculate the bias and variance quantities for a model using polynomials of a particular degree 'deg'. **Our net model for a given degree of polynomial will be the average of the coefficients of these classifiers.**

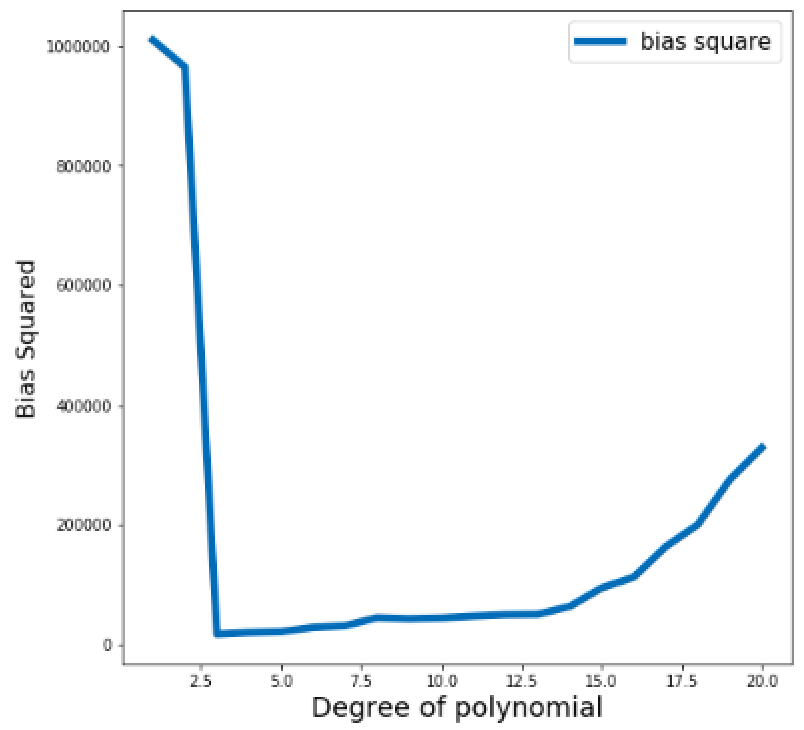
Table of bias variance quantities:



```
    deg       variance  bias square   mean bias         noise   total error
0     1   42722.034158  1.010593e+06  822.752772 -8.731149e-11  1.053315e+06
1     2   64676.486006  9.653637e+05  814.205381  0.000000e+00  1.030040e+06
2     3   78492.732219  1.684340e+04   94.940538  4.365575e-11  9.533613e+04
3     4   90145.112996  1.948222e+04  124.298704  1.455192e-11  1.096273e+05
4     5  177243.886652  2.074163e+04  128.461931 -2.910383e-11  1.979855e+05
5     6  199155.481590  2.802552e+04  144.124743  8.731149e-11  2.271810e+05
6     7  200853.414957  3.073127e+04  153.437633 -1.455192e-10  2.315847e+05
7     8  243769.695907  4.430699e+04  180.789693 -3.201421e-10  2.880767e+05
8     9  257742.060724  4.255026e+04  177.396710  2.619345e-10  3.002923e+05
9    10  272529.501302  4.388972e+04  181.403572  1.688022e-09  3.164192e+05
10   11  316093.343852  4.701637e+04  178.731082 -1.280569e-09  3.631097e+05
11   12  304650.216666  4.939955e+04  180.549867  6.752089e-09  3.540498e+05
12   13  361071.518227  5.003237e+04  175.841545  5.820766e-09  4.111039e+05
13   14  351990.137183  6.328651e+04  169.970055 -1.571607e-09  4.152766e+05
14   15  379661.240460  9.428569e+04  213.926321 -9.313226e-09  4.739469e+05
15   16  453496.653439  1.123775e+05  188.824579 -2.037268e-09  5.658742e+05
16   17  459459.849285  1.635828e+05  259.379737 -4.511094e-08  6.230426e+05
17   18  554331.553222  2.003044e+05  251.427244 -1.078006e-07  7.546359e+05
18   19  549674.237476  2.758132e+05  326.515634  6.635673e-08  8.254875e+05
19   20  658122.566722  3.291777e+05  319.222807 -2.277084e-07  9.873003e+05
```



Our note regarding bias



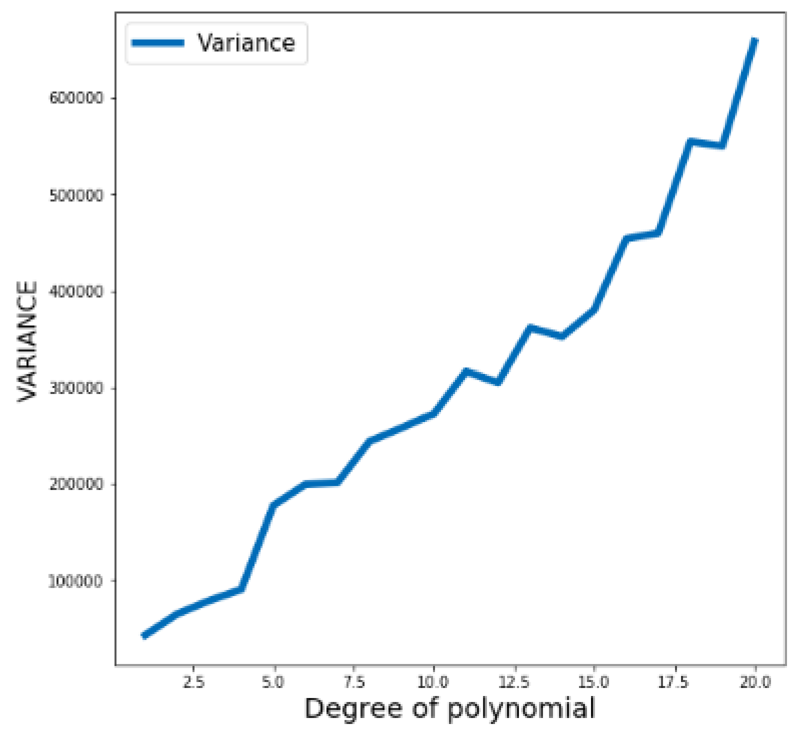
Bias is the difference between the average prediction of our model and the actual value which we are trying to predict. A model with high bias does not generalize the data well and oversimplifies the model. It always leads to a high error on training and test data.
High bias arises due to our classifier being "biased" to a particular kind of solution (e.g. linear classifier). **It means that the model being used is not robust enough to produce an accurate prediction. In other words, bias is inherent to our model .**

Model specific observations related to bias
In order to calculate bias squared for a polynomial of specific degree, we are doing the following:
Firstly, as has been mentioned above, we are training 10 versions of the same model.

Now, let a particular point in the test set be <x,y>. Now, let’s say that the 10 different realizations predict a y_value of $val_{1}, val_{2},.......val_{10}$ for this particular ‘x’. Then, the contribution of this test point to squared bias is [{ (val_{1}+val_{2}+...+val_{10})/10 } - y]^2. Obviously, we average this squared bias over all the test points.
[Here, (val_1+val_2+...+val_10)/10 : represents the average prediction we might make if we end up using a model of this particular polynomial degree].

In the assignment, **the trained model for polynomial degrees 1 and 2 is over simplified and does not fit the data well causing a high bias.**
At degree 3, the bias comes out to be minimum. This matches our expectations as we see that the plot of the dataset seems to be fitting a cubic function best.
Till degree 12, the bias remains almost constant or increases slightly.
From 12 onward, we notice that bias squared increases rapidly, even though theoretically it shouldn't because we are actually overfitting.
The reason for this, we believe, is that at high degrees of polynomial, LinearRegression().fit() is not able to produce optimal results because of insufficient number of data points. The algorithm (gradient descent) is unable to produce precise coefficients.
As demonstrated above, we showed that when LinearRegressin.fit() had to fit on a training data set of just 2 points, the optimal coefficients were not produced, rather, some other coefficients were produced. We feel that this is because 8000 (800 X 10) points aren’t good enough for manipulating more than 13 coefficients (for polynomials of degree greater than 11). However, the same size is sufficient for a model of polynomial degree 3 as the number of coefficients to manipulate is less.
Moreover, there is noise in the training data set which may produce inaccuracy.

A note about variance



Variance is the variability of a model prediction for a given data point. Again, imagine We can repeat the entire model building process multiple times. The variance is how the predictions for a given point vary between different realizations of the models.

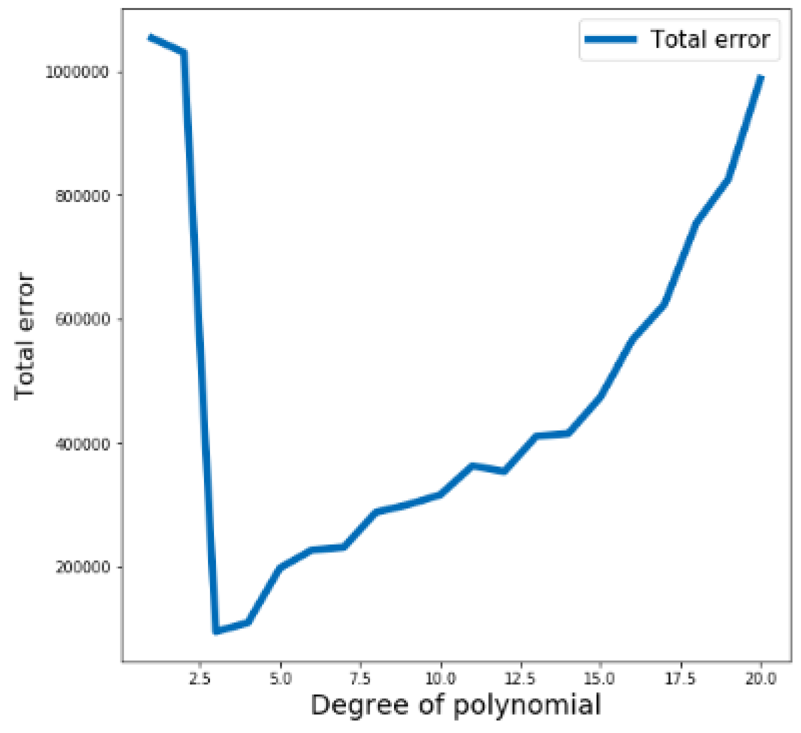
In other words, variance captures how much your classifier changes if we train on a different training set.
It represents how "over-specialized" the classifier is to a particular training set (overfitting).

Assignment specific details about variance
In general, variance increases with increase in model complexity (here, the degree of the polynomial).
This is because as we increase polynomial complexity, **it becomes more flexible for the model to fit even the minor deviations in the current training dataset partition. Each time the model is trained, the increased flexibility of the higher degree polynomials causes the coefficients to turn out considerably different due to differences in the training set (the classifier fails to generalize on the data).**

In other words, the model tries to fit the data points for each of the 10 training sets. As the degree increases, the model better fits each of these data chunks (each of size 800) . This results in higher variation among the different training sets. At higher degrees of polynomial, the variance begins to saturate at a high value as there is no better fitting of data left to do. The difference in the placement of coordinates in these training chunks can be seen below.

A note on total error

We notice that the total error gets minimized at polynomial of degree 3. Details have been mentioned in the bias-variance tradeoff explanation below.



A detailed description as to why noise (irreducible error) does not change

even when we change polynomial

Irreducible error is the property of the dataset**.** The value of irreducible error must not change and is independent of the type of the algorithm being used**.** It is the error which a model cannot escape, no matter how perfect it is. On a glance at the coordinates in the test data set, we noticed that the test data set has a one-one mapping and hence, we are able to predict, even before we trained our model, that the theoretical value of irreducible error should be zero. This prediction of ours was validated by the values we obtained while training using polynomials of different degrees. The irreducible error is almost zero for all polynomial degrees (not exactly zero due to precision errors in python). As expected, the value of irreducible error does not change as ‘irreducible error’ is the aspect of data.

A note on the bias-variance tradeoff

We explain bias variance tradeoff with help of some cases:
For degrees 1 and 2: For polynomial degrees 1 and 2, the trained model is oversimplified (the polynomial degree is not sufficient enough to capture the insights from the dataset) and results in a high bias (underfitting). It performs equally badly for all 10 training sets, hence variance is low and hence, even on the test set, the predictions are almost equally bad for all the 10 realizations of the model.
For degree 3: At degree 3, total error (sum of bias and variance) is at its minimum, showing that the data is actually best fit on a cubic function.
From degrees 4 to 10: We see that bias keeps almost constant whereas variance increases. We already know that a cubic fits the test set best from the plot of the test data. The polynomials of degree greater than 3 are almost able to obtain the same level of bias as they manage to keep coefficients of higher 'x' powers (x^4 onwards) very small and hence, not much difference is able to arise. However, variance keeps on increasing and as even though the higher coeffs are low, the higher coeffs of the 10 realizations of a model are different to fit their corresponding train sets and hence, produce different values for the points in the test set.

Our comments on the dataset

Our perception about the datasets are as follows:

- The training data set follows a cubic function trend which is why we obtain a low total error when training our model to be a polynomial of degree 3.
- The test data set and training data set seem to display different properties as far as noise is concerned. Evidence of this is the fact that the training dataset has a very high amount of noise (even for the same input 'x', there are several 'y' values in the training set). (A brief example regarding this would be the selling price of 5 identical houses being sold at quite different prices (2Cr, 3Cr, 2.5 Cr, 2.25 Cr, 3.5 Cr). However, the test dataset theoretically has zero noise as whenever the same input 'x' is present, the output is the same as well.
- In the training set, even for a fixed value of 'x', the corresponding 'y' values seem to be highly dispersed.