An Exploratory Research Into Linear Regression and Resampling Techniques for Studying Topographical Data

Ida Monsen, Vetle Henrik Hvoslef, and Leah Hansen
Department of Physics, University of Oslo
(Dated: October 15, 2023)

In our exploratory research, we have successfully implemented the linear regression methods Ordinary Least Squares, Ridge, and Lasso methods. Additionally, we implemented the bootstrap resampling technique and the k-fold cross-validation resampling method. We have used these methods to make models of both an artificially constructed dataset based on the Franke function and on real topographical data. In our final result, we were able to make a model for the Franke function with an R^2 score of 0.999 and a model for the topographical data with an R^2 score of 0.657, both using OLS regression. *Keywords: machine learning, linear regression, ridge, lasso, ordinary least squares, cross-validation*

I. INTRODUCTION

Regression analysis and resampling methods are foundational components in the field of machine learning and statistical analysis. They provide the fundamental framework for understanding the underlying relationships within data sets and predicting future trends with a degree of reliability.

A common form of linear regression analysis is the Ordinary Least Squares (OLS) method, which in principle minimizes the sum of the distances squared between the data points and our regression line. Where our data points are the observed values and our regression line is a representation of our predicted values. [1]

The Ridge and Lasso regressions are additions to the OLS method. They both add a penalty to the OLS, which is an addition to the sum of the distances squared in the traditional OLS method and punishes a model with high variance in its predictions. The Lasso method is similar to Ridge regression, however, its penalty is slightly different.

The practice of resampling is a crucial practice that allows us to effectively utilize the training data available in order to improve our model. In our research we have implemented two common resampling methods, the bootstrap method and the k-fold cross-validation method. However, we focused on the cross-validation method as our main resampling method.

In short, all resampling involves re-selecting data points from our original data set in order to create multiple sub-samples which we use to perform various analyses or model fits.[2]

The bootstrap method involves generating multiple samples by sampling with replacement from the original data set. [2]

The cross-validation method entails partitioning the data set into subsets, and using one subset for testing while training on the others. This is very useful when the original data set is small.[2]

In this report, we detail how we have studied the following regression methods, the Ordinary Least Squares (OLS), Ridge, and Lasso methods. Additionally, we detail how we performed both the bootstrap method and the k-fold cross-validation method in order to resample our data.

Our approach involves fitting a polynomial model with x and y dependencies. The $Franke$ function serves as our initial model, as it enables comprehensive investigations with randomly generated data sets. For a more detailed explanation of the theory behind our study see Section II.

In the end, we progressed from synthetic data to real-world topographical data, showcasing the performance of our models in practical applications. The final aim of our exploratory research is to not only comprehend the methodologies presented to the reader but also discern their usefulness in various contexts.

II. FORMALISM

A. Data

In the first part of our study, we will use the Franke function Eq.(1) to create data resembling topographical data. The Franke function is given as

$$\begin{aligned} f(x,y) = {} & \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ & + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ & + \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ & - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right), \end{aligned} \tag{1}$$

[1] https://github.com/leahelha/FYS-STK4155/tree/main/Project1

where $x, y \in [0, 1)$ in a uniform distribution. The Franke function will imitate a simple topographical landscape on which we will test our regression and resampling methods. Note, x and y in the uniform distribution are variables, and are *not* the same as the entries y_i of $\boldsymbol{y}$ or $\tilde{y}_i$ of $\tilde{\boldsymbol{y}}$ which represent a model of our real data and predicted data respectively. Explanation and discussion of $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$ can be found in Section II B.

As we progress past the use of the Franke function we will instead use real topographical data. This data covers an area in Norway and was downloaded from [3], which again was generated using the website [4].

B. Linear regression methods and our model

In the field of statistical analysis and machine learning, regression analysis is the process of finding the relationship between dependent and independent variables, in order to make future predictions. In our study we will be using linear regression models. These are models which assume that the regression function is linear. [2]

We will be using [2] as our reference source for all of Section II B unless otherwise stated.

1. *Ordinary Least Squares*

The most common form of linear regression is the Ordinary Least Squares method (OLS). This method aims to find the best-fitting linear equation that represents the relationship between a dependent variable and one or more independent variables. In our model, we will use OLS regression in order to estimate the optimal parameters $\boldsymbol{\beta}$. We will evaluate our model using the Mean Squared Error (MSE), Eq.(5) as explained in Section II B 4. We also use what is called a design matrix, $\boldsymbol{X}$.

The design matrix, $\boldsymbol{X}$, is a matrix of input variables that aims to describe our true data. It is an essential basis for our predictive model, $\tilde{\boldsymbol{y}} = \boldsymbol{X\beta}$, which we aim to make similar to the true model, $\boldsymbol{y}$. The $\boldsymbol{\beta}$ parameters are unknown parameters we will attempt to estimate through our regression methods. In our case, the design matrix is best described as

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0 & y_0 & x_0 y_0 & \dots & x_0^d & y_0^d \\ 1 & x_1 & y_1 & x_1 y_1 & \dots & x_1^d & y_1^d \\ 1 & x_2 & y_2 & x_2 y_2 & \dots & x_2^d & y_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}y_{n-1} & \dots & \dots & x_d & y_d \end{bmatrix}, \tag{2}$$

where we are assuming a polynomial fit, $[x, y, x^2, y^2, xy, \dots]$, of degree d to our model.

Note, x and y in the design matrix are variables and are *not* the same as the entries y_i of $\boldsymbol{y}$ or $\tilde{y}_i$ of $\tilde{\boldsymbol{y}}$ which represent a model of our real data and predicted data respectively. This is relevant to our numerical and analytical study.

We assume that our true data $\boldsymbol{y}$ can be represented by a continuous function $f(\boldsymbol{x})$,

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon}$ is some noise that we assume is normally distributed with zero mean and a variance σ^2. We approximate the unknown function with another continuous function $\tilde{\boldsymbol{y}}(\boldsymbol{x})$ which is given by

$$\tilde{\boldsymbol{y}}(\boldsymbol{x}) = \boldsymbol{X\beta}.$$

Here $\boldsymbol{\beta}^T = [\beta_0, \beta_1, \beta_2, ..., \beta_{p-1}]$ are the unknown parameters we want to estimate and $\boldsymbol{X}$, of size $n \times p$, is the design matrix which describes our dataset.

Our MSE for the OLS method is

$$MSE(\boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X\beta})^T(\boldsymbol{y} - \boldsymbol{X\beta}),$$

where n is the number of data points. Note that this is what we will use as our loss function throughout our analysis for the OLS method. For a further explanation of the MSE see Section II B 4. [2]

We optimize our parameters $\boldsymbol{\beta}$ using what is called the *loss function* (also referred to as the *cost function*), $C(\boldsymbol{X}, \boldsymbol{\beta})$. The loss function is defined by taking the MSE between the true model, $\boldsymbol{y}$, and the expected output $\tilde{\boldsymbol{y}} = \boldsymbol{X\beta}$,

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X\beta})^T(\boldsymbol{y} - \boldsymbol{X\beta}).$$

We find the optimal parameter $\boldsymbol{\beta}$ for the OLS method by differentiating the loss function with respect to the transposed $\boldsymbol{\beta}$ vector,

$$\frac{\partial C(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}^T} = \boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X\beta}) = 0.$$

This gives the relation

$$\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T\boldsymbol{X\beta}.$$

With an invertible matrix $\boldsymbol{X}^T\boldsymbol{X\beta}$, we get the solution to the optimal $\boldsymbol{\beta}$ parameter as

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X\beta})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{3}$$

See Appendix V A for more.

2. *Ridge regression*

Ridge regression is an addition to the OLS method which shrinks the vector of optimal β parameters, $\hat{\boldsymbol{\beta}}$, by penalizing the loss function if the $\boldsymbol{\beta}$ values get too large. The Ridge regression loss function is similar to the OLS loss function, it simply adds a penalty as seen here.

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X\beta})^T(\boldsymbol{y} - \boldsymbol{X\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}.$$

Here $\lambda \geq 0$ is the regularization parameter and controls the shrinkage of the optimal parameter, $\hat{\boldsymbol{\beta}}$. The larger λ is, the smaller our $\hat{\boldsymbol{\beta}}$ gets.

By taking the derivative of the loss function with respect to $\boldsymbol{\beta}$ and equating it to zero, we derive that the optimal parameters for the Ridge regression are

$$\hat{\boldsymbol{\beta}}^{Ridge} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{4}$$

[2]

3. Lasso regression

Similar to the Ridge method, we have the Lasso regression method. The key difference between the two is what they add to the loss function in comparison to the OLS method. If we look at the loss function of the Lasso method below, we see that the term $\boldsymbol{\beta}^T\boldsymbol{\beta}$ in the Ridge method has been replaced with $\|\boldsymbol{\beta}\|_1$. Which gives us a loss function like:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_1,$$

where $\|\boldsymbol{\beta}\|_1$ is the sum of the absolute values of the individual elements of $\boldsymbol{\beta}$, and the rest of the parameters are the same as before.

We minimize the loss function by deriving it with respect to the $\boldsymbol{\beta}$ parameter and equating it to zero, this gives us:

$$\frac{\partial C(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\frac{2}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T\boldsymbol{X} + \lambda sgn(\boldsymbol{\beta}) = 0,$$

where $sgn(\beta)$ is defined as

$$\text{sgn}(\beta) = \begin{cases} 1 & \beta > 0 \\ -1 & \beta < 0. \end{cases}$$

The derivative of the loss function is discontinuous, and as a result, we do not have an analytical solution to the optimal $\boldsymbol{\beta}$ parameter for the Lasso method. However, we may find an optimal parameter for $\boldsymbol{\beta}$ numerically, as we will focus on in this study.

4. R^2 score and MSE

In our study, there are two crucial metrics we will use to evaluate the performance of our model at its various stages. These metrics are the Mean Squared Error (MSE) and the R^2 score.

The general equation for the MSE is given by

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2, \tag{5}$$

where n is the number of observations, y_i is the true value of the ith observation and $\tilde{y}_i$ is our predicted value for the ith observation.

The R^2 score, or the coefficient of determination, measures how well the predicted model is a fit to the dataset. It is given by

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}, \tag{6}$$

where again n is the number of observed values, y_i is the ith datapoint in the training data and $\tilde{y}_i$ is the ith datapoint in the predicted value.

C. Resampling methods

Resampling is the process of repeatedly drawing samples from a training set and refitting our model on a selection of samples in order to obtain more information about the model. It is a crucially important method in machine learning and allows us to create more data to better evalutate our predictive model. By resampling the training data we can compute the test error, Err_{test}, with varying training data. The two methods we will be focusing on in this study are the *bootstrap* and the *k-fold Cross-validation* methods. We will be using [2] as our reference source for all of Section II C.

1. Bootstrap method

As is common practice, we split our total data set into training and testing. The bootstrap method involves randomly drawing samples with replacements from our training data set. This resampling is done n times, each time creating a "new" training set the same size as the original. We fit our model to each training set and observe the behavior of our regression model for each n.

The recipe for what we do in this study looks something like this:

1. Given a dataset $\boldsymbol{X}$ we randomly split it into training and test data. Let the training data be defined as $\boldsymbol{x} = (x_1, x_2, ..., x_n)$.

2. We draw n random samples with replacement from $\boldsymbol{x}$, creating a new training dataset $\boldsymbol{x_T}$ of length n.

3. Using $\boldsymbol{x}_T$ we estimate the optimal parameter $\hat{\boldsymbol{\beta}}_{\boldsymbol{T}}$

4. Repeat steps 1-3 k times and in the end, we can evaluate the k different models.

We compute the average MSE, bias, and variance for the k bootstrap samples.

From here we can perform a bias-variance analysis, see Section II D, using $\boldsymbol{z}_{pred} = \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\boldsymbol{T}}$ for the bias calculation in Eq.(8).

2. *Cross-validation method*

In our study, we have implemented the k-fold cross-validation method, which is a common and simple method for resampling data. The method can be reduced to the following steps:

1. We randomly shuffle the dataset and divide it into k groups.
2. For the k groups we chose one group to be the test data and the rest to be one training data set.
3. We fit a model on the training data and evaluate using the test data. Our method of evaluation is the MSE.
4. We repeat steps 1-3 k times. In the end, we summarize and learn from the model evaluations. We have done this by taking the average of the MSE values calculated for each model.

In our case, we will be using the results of the k-fold cross-validation method to optimize our final model so that we find a balance between underfitting and overfitting, see Section II D for more.

D. The bias-variance trade-off

The variance of a model tells us how much the model may vary from the mean. We define the variance of an arbitrary vector $\boldsymbol{a}$ by,

$$Var(\boldsymbol{a}) = \mathbb{E}(\boldsymbol{a}^2) - \mathbb{E}(\boldsymbol{a})^2, \tag{7}$$

where $\mathbb{E}(\boldsymbol{a})$ is the expectation value of the vector $\boldsymbol{a}$.

A model with high variance and a low bias means that the model is very flexible and could indicate over-fitting. Over-fitting means the model is too well fitted to the specific test data, which can include fitting to noise, but does not generalize well to predict new data.

The bias of a model is an indicator of how much the predictive model deviates from the behavior of the true data. For our polynomial dataset we will be calculating the bias by

$$Bias_{\text{degree}}(\boldsymbol{z_{test}}) = \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{z}_{\text{test},i} - \frac{1}{m}\sum_{j=1}^{m}\boldsymbol{z}_{\text{pred},ij}\right)^2, \tag{8}$$

where n is the number of data points in our test data and m is the number of predictions for each data point. $z_{test,i}$ is the ith value of the test data and $z_{pred,ij}$ is the jth predicted value for the ith data point.

A model with high bias and low variance has the tendency to be overly simplistic as it operates under assumptions that can cause our model to miss relevant patterns in the behavior of our training data. When this happens it is called under-fitting.

Ideally, we would like to keep both the bias and the variance of our model low. However, having both low bias and low variance is difficult. Bias and variance of a model are sources of error where if you have a high value of one, you're likely to have a low value of the other. This is called the bias-variance trade-off.

It is possible to study the bias-variance trade-off analytically as we have done in the appendix, Section V B.

E. Procedure

For doing the analyses, we implemented our own class *regression_class* in Python. This class, which can be found in the GitHub link [5], is designed to do Ordinary Least Squares, Ridge and Lasso regression, with polynomial models up to a degree specified by the user.

The arguments passed to the class are the predictor variables $\boldsymbol{x}$, the response variables $\boldsymbol{y}$, the maximum polynomial degree *n_deg_max* and the λ values one wants to try for Ridge and Lasso. The class then constructs a design matrix of the predictor variables $\boldsymbol{x}$ and divides it into test and training using Scikit-learn's function *train_test_split*.

Before performing linear regression on our data, we choose to normalize the design matrix. This ensures that we do not end up working with numbers that differ by several orders of magnitude, which the original topographical data and hence design matrix do. It gives us the benefit that the data generated from the Franke function and the real topographical data are more comparable after normalizing both of them. This is the reason that we also choose to normalize the data from the Franke function, even though that would not be necessary in an isolated case. Note that the normalization is done using the mean and standard deviation of the test data, and not the whole data set.

We scaled the values by,

$$x_j \rightarrow \frac{x_j - \overline{x}_j}{\sigma(x_j)},$$

where $\overline{x}_j$ and $\sigma(x_j)$ are the mean and standard deviation, respectively, of the feature $x_j^{(i)}$.

Explicitly, the normalization is done as

$$x_{ij,norm} = \frac{x_{ij} - \overline{\boldsymbol{x}}_j}{\sigma(\boldsymbol{x}_j)}, \tag{9}$$

where x_{ij} is a specific element in the design matrix, $\overline{\boldsymbol{x}}_j$ is the mean of the corresponding column and $\sigma(\boldsymbol{x}_j)$ is the standard deviation of that column.

The functions of the class do the regression for each polynomial degree, as well as for each λ in the case for Ridge and Lasso. We store the $\boldsymbol{\beta}$ values, the mean squared error and the R^2 score for both the training and test data.

When implementing the OLS and the Ridge regression method numerically, we used Eq.(3) and Eq.(4) respectively. However, as the Lasso regression method does not have an analytical solution, we utilized a pre-made class by Scikit-learn named Lasso. Following this, we wrote out own code for calculating the MSE and R^2 score, using Eq.(5) and Eq.(6), respectively.

We studied two resampling methods, one being the bootstrap method and the other being the cross-validation method. The approach for the bootstrap method is given in Section II C 1, while the approach for k-fold cross-validation is given in Section II C 2. When implementing the bootstrap method for 20 × 20 data point, we performed a bias-variance analysis. We plotted the MSE, $bias$, and $variance$ of our artificial data.

1. Testing

We developed and tested our class and its functions using data generated by the two-dimensional Franke function, Eq.(1), at points $x, y \in [0, 1)$ chosen randomly from a uniform distribution. Since our final goal is to use linear regression on real topographical data, the Franke function is a good test case, as it's surface somewhat resembles a terrain with hills and valleys.

In testing our code, we performed OLS and Ridge regression on the Franke function, using both our own implementation and the Scikit-learn library for comparison. Lasso regression was not included in our tests as we already use Scikit-learn for this.

We fitted a polynomial model of degrees 1 to 5 do a data set of size 101 × 101 points, and used the λ values $[0.0001, 0.001, 0.01, 0.1, 1.0]$ for Ridge regression.

For each model, we then compared the individual $\boldsymbol{\beta}$ values and the MSE of the training data obtained using our own code and Scikit-learn. The two approaches agree to a precision of 10^{-7} for all individual values. When we concluded our testing and found the results to indicate that we had implemented our regression methods correctly, we advanced to proper analysis of the Franke function and also real topographical data.

III. RESULTS AND DISCUSSION

A. The Franke function

We first carried out regression analysis on the Franke function Eq.(1). Specifically, we generated a grid of 101×101 points $x, y \in [0, 1)$ and evaluated the Franke function at these points. These points comprise our dataset for the Franke function. The methods we used were the OLS, Ridge, and Lasso regression for polynomial models up to degree 15 and with λ values $[0.0001, 0.001, 0.01, 0.1, 1.0]$. Figures Fig.(1) and Fig.(2) show a graph of the mean squared error and R^2-score, respectively, that we get when using the Ordinary Least Squares method.

As can be seen, the test and training data give almost equal mean squared error and R^2-score in this case. The test error does not blow up when the polynomial degree increases but tapers off just like the training error does. This is probably because we have data without noise, and therefore have no noise on which to overfit our model. We also observe that the R^2-score approaches 1, which would be a perfect fit.

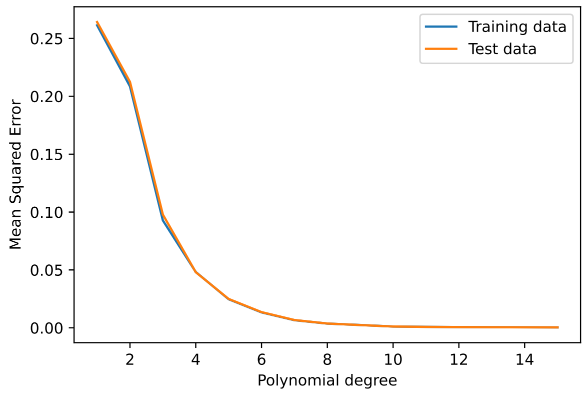

FIG. 1: Mean squared error of the Franke function using Ordinary Least Squares regression. Polynomial degree of the model used for regression on the x-axis and mean squared error calculated on normalized data on the y-axis.

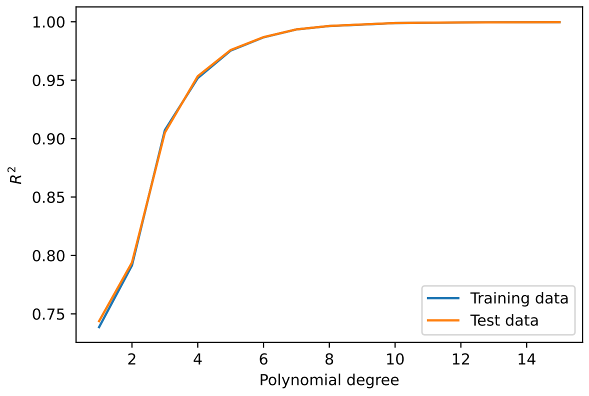

FIG. 2: R^2-score of the Franke function using Ordinary Least Squares regression. Polynomial degree of the model used for regression on the x-axis and R^2 calculated on normalized data on the y-axis.

To determine the best polynomial degree for the Ordinary Least Squares regression, we use k-fold with $k = 5$ folds. The results are presented in table I. As we can see, the mean squared error decreases all the way up to polynomial degree 14, and there is a slight increase again for polynomial degree 15. We choose to stop at polynomial degree 10, since the gain in mean squared error beyond that is small. At this degree, we gain almost nothing by further increasing the complexity, as we already have an excellent fit. This also agrees with Fig.(1) and Fig.(2).

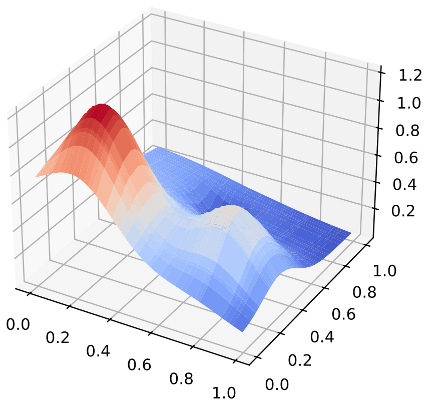

FIG. 3: The Franke function.

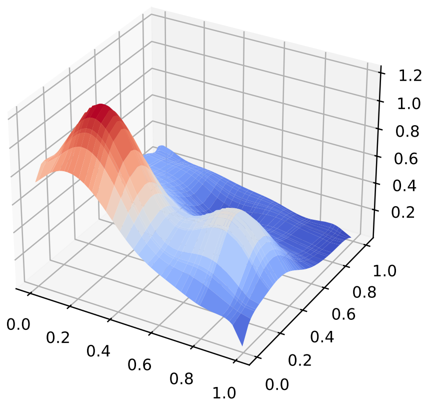

FIG. 4: Prediction of Franke function using an Ordinary Least Squares model with polynomial degree 10.

Figure 4 shows a prediction of the Franke function using polynomial degree 10. By comparing the prediction with the true Franke function in Fig.(3), we see that we have an excellent fit. The R^2 score of this model is 0.999, and the mean squared error of the test data is 0.00112.

Degree	MSE k-fold
1	0.26043
2	0.20829
3	0.093419
4	0.048079
5	0.024691
6	0.013437
7	0.0066359
8	0.0037301
9	0.0024824
10	0.0011447
11	0.00077706
12	0.00057659
13	0.00026373
14	0.0001496
15	0.00018267

TABLE I: Average mean squared error for the Franke function using k-fold with $k = 5$ folds. The regression method used is Ordinary Least Squares, and the mean squared error is presented for each polynomial degree.

When it comes to Ridge and Lasso regression, none of them gave better results than the OLS. The Lasso regression did not converge after 1000 iterations, which resulted in worse performance than both OLS and Ridge regression. Ridge and Lasso still showcase some similar patterns. Plotting the MSE for one particular polynomial degree as a function of λ shows how the performance varies depending on which λ value we use. A representative plot is shown in Fig.(5).

The MSE decreases when we decrease the λ value we use, and this turns out to be the case for almost all polynomial degrees across both Ridge and Lasso regression. Table II shows this result for Ridge regression. Only for polynomial degree 5 is the optimal λ value different from 0.0001, which is the smallest one we tested. And upon closer inspection, this other λ still gives an almost identical mean squared error as the lowest λ.

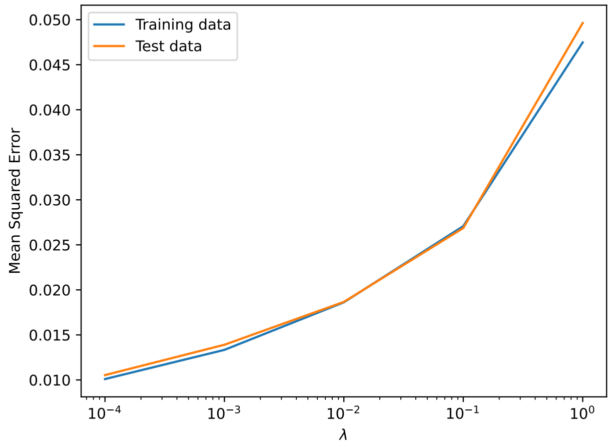

FIG. 5: Mean squared error of the Franke function using Ridge regression and a polynomial of degree 7. Different λ values on the x-axis and mean squared error calculated on normalised data on the y-axis.

As λ goes towards zero, both Ridge and Lasso get closer to OLS. A λ value of exactly zero is the same as just performing OLS regression. Our observation that Ridge and Lasso perform better as λ approaches zero, further solidifies our interpretation that the OLS method gives

Degree	λ	MSE test
1	0.0001	0.26417
2	0.0001	0.21245
3	0.0001	0.097816
4	0.0001	0.048055
5	0.001	0.024945
6	0.0001	0.013758
7	0.0001	0.010539
8	0.0001	0.007722
9	0.0001	0.006269
10	0.0001	0.0051204
11	0.0001	0.0041765
12	0.0001	0.0036928
13	0.0001	0.0034018
14	0.0001	0.0030729
15	0.0001	0.0026641

TABLE II: Shows the optimal choice of λ for each polynomial degree when doing Ridge regression on the Franke function. The optimal choice is the one which gives the lowest mean squared error of test data, and this lowest error is also included in the table.

the best results.

1. *Bias-variance analysis*

For our bias-variance analysis, we performed the bootstrap resampling method on the Franke function data, implementing OLS as our choice of linear regression. The result of this can be seen in Fig.(6). In the figure, we see that the optimal polynomial degree from our bias-variance analysis is at a degree of 5. We see this as the error is at its lowest for a polynomial of degree around 5 with bias and variance both remaining low. As expected from theory, the error rises as both bias and variance rise.

The bootstrap result of the optimal polynomial does not concur with the result of our k-fold cross-validation resampling, where we arrived at the best polynomial degree of 10.

This, we attribute to the difference in the number of data points selected for the two resampling methods, as the best polynomial degree is dependent on the data point size selection. For the bootstrap resampling method, we implemented 20×20 data points, while we implemented 101×101 data points for the k-fold cross-validation method.

Due to the differences in the size of the data points, the results of the bootstrap resampling method, Fig.(6), and the cross-validation method are not comparable. In our study, we have placed an emphasis on the k-fold cross-validation method as our main form of resampling. We emphasized the use of the cross-validation method due to its reputation of being an optimal regression method to implement when working on a small data set of observed values, [1]. That is what fits our case, as the topographical data we were able to work with is downsampled, due to the original dataset being too large. Additionally, we were unable to implement the bias-variance analysis successfully for a larger data set in the time we had to complete our study.

Thus, we prioritized our time on improving the resampling method most reputable for small data sets for the Franke function, as our aim was to progress to the real topographical data.

For future improvement we would like to implement the bias-variance analysis for a similar data point size selection at 101×101, and study at what polynomial degree we have an optimal model.

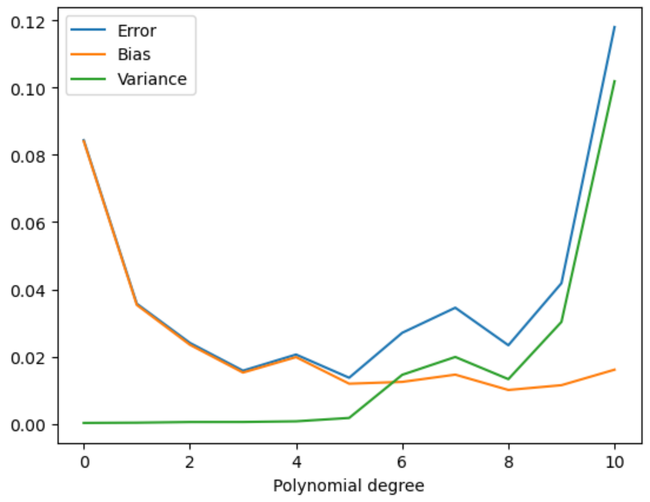

FIG. 6: Results of bias-variance analysis using 400 data points and 200 bootstrap samples. The error plotted is the MSE.

B. Topographical data

We then carried out a similar analysis of the topographical data. This data is shown in Fig.(7). As the data we used was too large to analyze in its entirety, we decided to downsample it to every 50th row and every 50th column and do regression on the downsampled data. This decreases the resolution of the terrain as seen in Fig.(8). However, our linear regression models did too poorly of a job of capturing the downsampled topographical data for the resolution to matter.

The OLS regression outperformed the other regression methods for the topographical data, just as it did for the Franke function. The mean squared error for the OLS regression done on the topographical data is presented in Fig.(9). Here we see that the test data has a larger MSE than the training data. This was to be expected, as the topographical data is more noisy than the data generated by the Franke function. In the figure, the gain in MSE tapers off somewhere around a polynomial of degree 20, but in order to determine the optimal polynomial degree we used k-fold with $k = 5$ folds. The results from this

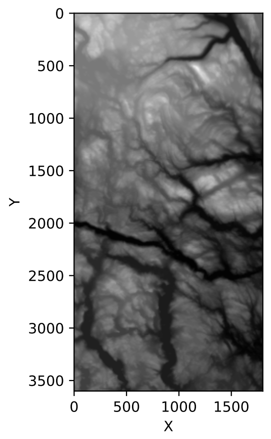

FIG. 7: Terrain data that we want to do regression on.

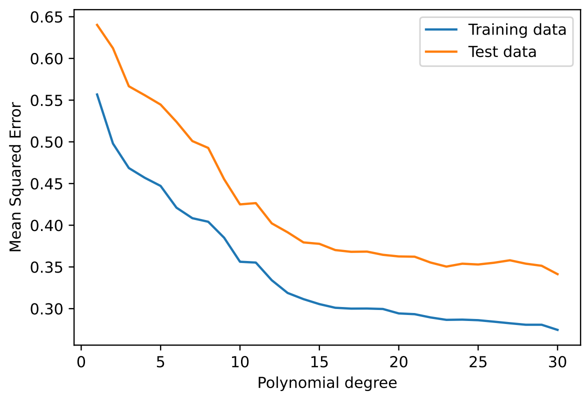

FIG. 9: Mean squared error of the topographical data using Ordinary Least Squares regression. Polynomial degree of the model used for regression on the x-axis and mean squared error calculated on normalised data on the y-axis.

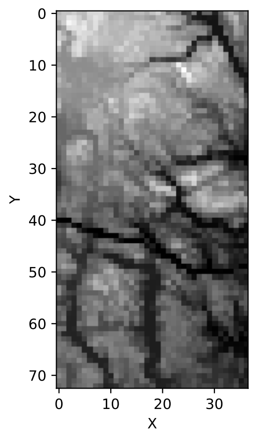

FIG. 8: Terrain data downsampled to every 50th row and every 50th columns, so that the resolution is decreased. This is the data we do regression on.

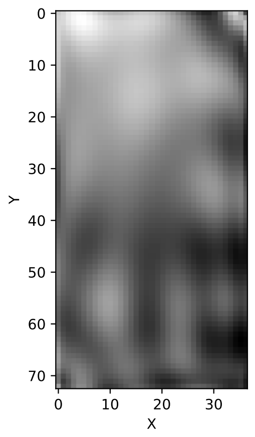

FIG. 10: Prediction of the terrain data using an Ordinary Least Squares model with polynomial degree 18.

k-fold analysis are presented in table III. We see that the average mean squared error decreases until polynomial degree 18, and then starts to increase again. We therefore choose 18 as our best polynomial degree.

This particular model has a best R^2 score of 0.657 for the test data, which is significantly worse than the best R^2 score 0.999 for the Franke function model. This is however as expected since the terrain data is much more complex. The mean squared error of this optimal model is 0.368 for the test data. Figure 10 shows a plot of this prediction. As we can see, it fails to capture the structure of the terrain data in detail. Only the most general trends can be seen in the prediction.

The mean squared error and R^2 score for Ridge and Lasso regression show the same pattern for topographical data as described for the Franke function. Figure 5 is still representative of the behaviour, where the mean squared error is better the closer λ is to 0. This again indicates that we gain nothing by introducing the parameter λ for Ridge and Lasso regression, and it is better to stick to the simpler ordinary least square model. The Lasso regression did not converge after 1000 iterations for the topographical data and therefore has the worst performance of the three regression types. The optimal mean squared error for test data is 0.468 and 0.513 for Ridge and Lasso regression, respectively, compared to a mean squared error of 0.368 for Ordinary Least Squares.

Degree	MSE k-fold
1	0.56677
2	0.51754
3	0.48339
4	0.47406
5	0.46713
6	0.44544
7	0.42945
8	0.42921
9	0.41026
10	0.3846
11	0.3658
12	0.33844
13	0.33123
14	0.3277
15	0.31206
16	0.33719
17	0.29613
18	0.29196
19	0.31856
20	0.31729
21	0.42451
22	0.45404
23	0.40194
24	0.5278
25	1.3734
26	0.60791
27	1.1278
28	1.8679
29	0.64619
30	0.50348

TABLE III: Average mean squared error for topographical data using k-fold with $k = 5$ folds. The regression method used is Ordinary Least Squares, and the mean squared error is presented for each polynomial degree.

C. Key points of discussion

Overall, we see that linear regression with Ordinary Least Squares and a polynomial model performs excellent on the Franke function, but not quite so well on the topographical data. Ridge and Lasso regression did not increase performance. Instead, the performance worsened with an increase in the λ value. We also needed to use a higher polynomial degree for the terrain data to achieve the best possible fit than what was needed for the Franke function. A summary of our key results and measurements can be found in table IV.

IV. CONCLUSION

In our study, we have used the linear regression methods Ordinary Least Squares, Ridge, and Lasso to fit data generated from the Franke function and real topographical data to a polynomial model of varying degrees. For both types of data, we found that the OLS method was the most suitable regression method. Neither Ridge nor Lasso increased the performance of either fit, with Lasso unable to even converge.

	Franke function	Topographical data
Regression type	OLS	OLS
Degree	10	18
MSE test	0.00112	0.368
R^2 test	0.999	0.657

TABLE IV: Summary of our results. The table shows the best regression method, the optimal polynomial degree, and corresponding measurements for both the Franke function and topographical data.

The OLS method was able to almost perfectly capture the Franke function, with an R^2 score of 0.999 and a test mean squared error of 0.00112 when using a polynomial model of degree 10, as seen in table IV. We found that the OLS model performed the best for a polynomial degree of 10, using k-fold with $k = 5$ folds.

The results for the topographical data were worse, which is just as expected, given that the real data is much more complicated than the Franke function. The OLS regression performed the best, and the Lasso regression did not converge for the real data. The optimal MSE produced for the test data was 0.368 for the OLS method, and 0.468 and 0.513 for Ridge and Lasso regression, respectively.

We found 18 to be the optimal polynomial degree for the OLS method when using k-fold with $k = 5$ folds. Given that the real topographical data was more complicated than the artificial landscape produced by the Franke function, a higher polynomial degree than for the Franke function was to be expected. The OLS model of the real data gave the best R^2 score of 0.657. This is far from the near-perfect fit we got from the Franke function data model.

For both the Franke function model and the topographical data model, we appear to have reached the limit of what can be done with linear regression. Our test model based on the Franke function data performed very well, however, the same can not be said for the model of the real data. The low MSE and R^2 score for our Franke function shows that we have arrived at a sufficient model for this data. However, based on the R^2 score of the real topographical data model, we believe that our polynomial model is too simple to accurately model the complex details of real terrain, regardless of our linear regression method.

In conclusion, our model was not well suited to fit real topographical data. However, we found that our model, specifically the version implementing the OLS method, works excellently for simple terrain such as the ones produced by the Franke function.

[1] B. Goodfellow and Courville, “Deep learning,” (2016).
[2] Morten Hjort-Jensen, “FYS-STK 4155 course material,” (2023).
[3] “Morten Hjort-Jensen Github page,” Last accessed: 15th October 2023.
[4] “The U.S. Geological Survey,” Last accessed: 15th October 2023.
[5] “The GitHub link to our code and supplementary materials,” .

V. APPENDIX

A. Various mean values and variances for the OLS method

The following section is a collection of derivations done by the authors of this paper relating to various mean values and variances for the OLS method. Reading this is not necessary to follow the paper, but is available to particularly interested readers.

Given the assumption that there exists a continuous function $f(\boldsymbol{x})$ and a normal distributed error $\varepsilon \sim N(0, \sigma^2)$ which describes our data,

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$$

We approximate the function $f(\boldsymbol{x})$ with our model $\tilde{\boldsymbol{y}}$. Where $\tilde{\boldsymbol{y}}$ is from the solution of the linear regression equations where we have minimized $(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$, and

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}.$$

Here, $\boldsymbol{X}$ is the design matrix, and $\boldsymbol{\beta}$ is the parameter vector. The design matrix is best described as follows,

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0 & y_0 & x_0y_0 & \dots & x_0^d & y_0^d \\ 1 & x_1 & y_1 & x_1y_1 & \dots & x_1^d & y_1^d \\ 1 & x_2 & y_2 & x_2y_2 & \dots & x_2^d & y_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}y_{n-1} & \dots & \dots & x_d & y_d \end{bmatrix}, \tag{10}$$

where we are assuming a polynomial fit, $[x, y, x^2, y^2, xy, \dots]$, of degree d to our model.

Note, x and y in the design matrix are variables and are *not* the same as the entries y_i of $\boldsymbol{y}$ or $\tilde{y}_i$ of $\tilde{\boldsymbol{y}}$ which represent a model of our real data and predicted data respectively.

We will now show that the expectation value of $\boldsymbol{y}$ for a given element i is

$$\mathbb{E}(y_i) = \sum_j x_{ij}\beta_j = \boldsymbol{X_{i,*}}\boldsymbol{\beta},$$

where $\boldsymbol{X_{i,*}}$ represents all the elements for the ith row in the design matrix summed.

Taking the expectation value of y_i we have

$$\mathbb{E}(y_i) = \mathbb{E}[\boldsymbol{X_{i,*}}\boldsymbol{\beta} + \boldsymbol{\varepsilon}_i] = \mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta}) + \mathbb{E}(\boldsymbol{\varepsilon}_i),$$

where entries in $\boldsymbol{X_{i,*}}\boldsymbol{\beta}$ are deterministic, so $\mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta}) = \boldsymbol{X_{i,*}}\boldsymbol{\beta}$.

Since ε_i is normally distributed with a mean at zero, its expectation value is zero, $\mathbb{E}(\varepsilon_i) = 0$.

Thus, $\mathbb{E}(y_i) = \boldsymbol{X_{i,*}}\boldsymbol{\beta}$.

Moving on, we also want to show that $Var(y_i) = \sigma^2$. We find the variance of y_i by

$$Var(y_i) = \mathbb{E}(y_i^2) - \mathbb{E}(y_i)^2.$$

This is pretty straightforward, we expand the y_i^2 and note that $y_i = \boldsymbol{X_{i,*}}\boldsymbol{\beta} + \varepsilon_i$.

$$\begin{aligned} Var(y_i) &= \mathbb{E}(y_i^2) - \mathbb{E}(y_i)^2 \\ &= \mathbb{E}[(\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 + 2\boldsymbol{X_{i,*}}\boldsymbol{\beta}\varepsilon_i + \varepsilon_i^2] - \mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta} + \varepsilon_i)^2 \\ &= \mathbb{E}((\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2) + \mathbb{E}(2\boldsymbol{X_{i,*}}\boldsymbol{\beta}\varepsilon_i) + \mathbb{E}(\varepsilon_i^2) - \mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 + \mathbb{E}(\varepsilon_i)^2 \end{aligned}$$

Since $\mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta}) = \boldsymbol{X_{i,*}}\boldsymbol{\beta}$ and $\mathbb{E}(A \cdot B) = \mathbb{E}(A) \cdot \mathbb{E}(B)$, all expectation values concerning ε_i not squared, become zero. When we also consider that the two terms concerning $\mathbb{E}(\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2$ cancel out, we are left with

$$Var(y_i) = \mathbb{E}(\varepsilon_i^2) = \sigma^2$$

Where we have that $\mathbb{E}(\varepsilon_i^2) = \sigma^2$ due to the definition of the distribution of ε, with a standard deviation of σ^2.
We now want to show that

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta},$$

where $\hat{\beta}$ is the optimal parameter for the Ordinary Least Squares regression method.
We have that,

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X^T X})^{-1}\mathbb{E}(\tilde{\boldsymbol{y}}).$$

Remember that $\tilde{\boldsymbol{y}} = \boldsymbol{X\beta}$, and the expectation value of this is just $\boldsymbol{\beta}$. Thus we have that,

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = \underbrace{(\boldsymbol{X^T X})^{-1}\boldsymbol{X^T X}}_{I}\boldsymbol{\beta} = \boldsymbol{\beta}$$

Finally, we want to show that $Var(\hat{\boldsymbol{\beta}}) = \sigma^2(\boldsymbol{X^T X})^{-1}$.

$$\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}\left(\hat{\boldsymbol{\beta}}^2\right) - \mathbb{E}(\hat{\boldsymbol{\beta}})^2 \\
&= \mathbb{E}\left[\left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y}\right)^2\right] - \boldsymbol{\beta}^2 \\
&= \left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^\top\right)^2 \mathbb{E}\left(\boldsymbol{Y}^2\right) - \boldsymbol{\beta}^2 \\
&= \left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^\top\right)^2 \left(\underbrace{\mathbb{E}((\boldsymbol{X\beta})^2)}_{=(\boldsymbol{X\beta})^2} + 2\underbrace{\mathbb{E}\left(\varepsilon_i\right)\boldsymbol{X\beta}}_{=0} + \underbrace{\mathbb{E}\left({\varepsilon_i}^2\right)}_{=\sigma^2}\right) - \boldsymbol{\beta}^2 \\
&= \underbrace{\left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{X\beta}\right)^2}_{\boldsymbol{\beta}^2} + \left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\right)^2\sigma^2 - \boldsymbol{\beta}^2 \\
&= \left(\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\right)^2\sigma^2\left(\boldsymbol{X}^T\right)^2 \\
&= \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\underbrace{\left(\boldsymbol{X}^\top\boldsymbol{X}\right)^{-1}}_{I}\sigma^2 \\
&= \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\sigma^2
\end{aligned}$$

B. Bias-variance

Consider a data set $\boldsymbol{y}$, given by

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\varepsilon},$$

where we assume that it can be presented by a continuous function $\boldsymbol{f}(\boldsymbol{x})$ and a normally distributed error $\boldsymbol{\varepsilon}$ with a mean zero and a standard deviation of σ^2.
As we found in Section II B 1 when we applied the OLS method, we represented our model by $\tilde{\boldsymbol{y}}$ where this works as an approximation to $\boldsymbol{f}$. The loss function for the OLS method written in terms of $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$ is given by

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right], \tag{11}$$

where have noted that the loss function also equals the expression for the expectation value of the model error $\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]$.

Using the expectation term from Eq.(11) we can show the bias-variance trade-off analytically. First, we separate the term into its parts.

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right]=\mathbb{E}\left[(\boldsymbol{f}(\boldsymbol{x})+\boldsymbol{\varepsilon}-\tilde{\boldsymbol{y}})^2\right]$$

We then expand the quadratic term.

$$=\mathbb{E}\left[(\boldsymbol{f}(\boldsymbol{x})^2+\overbrace{2\boldsymbol{\varepsilon}\boldsymbol{f}(\boldsymbol{x})}^{0}-2\boldsymbol{f}(\boldsymbol{x})\tilde{\boldsymbol{y}}-\overbrace{2\boldsymbol{\varepsilon}\tilde{\boldsymbol{y}}}^{0}+\overbrace{\boldsymbol{\varepsilon}^2}^{\sigma^2}+\tilde{\boldsymbol{y}}^2)\right]$$

Because $\boldsymbol{\varepsilon}$ is normally distributed with mean zero, the expectation value of ε is zero. However, the expectation value of ε^2 is its variance σ^2.

With this in mind, we are left with

$$\begin{aligned}&=\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})^2)-2\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})\tilde{\boldsymbol{y}})+\sigma^2+\mathbb{E}\left(\tilde{\boldsymbol{y}}^2\right)\\&=\sigma^2+\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})^2)-2\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))\mathbb{E}(\tilde{\boldsymbol{y}})+\mathbb{E}\left(\tilde{\boldsymbol{y}}^2\right).\end{aligned}$$

We draw out the variance of $\tilde{\boldsymbol{y}}$ by inserting the terms $-\mathbb{E}(\tilde{\boldsymbol{y}})^2+\mathbb{E}(\tilde{\boldsymbol{y}})^2$. Using the Eq.(7), we get

$$\begin{aligned}=&\sigma^2+\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})^2)-2\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))\mathbb{E}(\tilde{\boldsymbol{y}})+\underbrace{\mathbb{E}\left(\tilde{\boldsymbol{y}}^2\right)-\mathbb{E}(\tilde{\boldsymbol{y}})^2}_{Var(\tilde{\boldsymbol{y}})}+\mathbb{E}(\tilde{\boldsymbol{y}})^2\\=&\sigma^2+Var(\tilde{\boldsymbol{y}})+\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})^2)-2\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))\mathbb{E}(\tilde{\boldsymbol{y}})+\mathbb{E}(\tilde{\boldsymbol{y}})^2.\end{aligned}$$

Then, we use the relation between the bias $Bias(\tilde{\boldsymbol{y}})$ of $\tilde{\boldsymbol{y}}$ and its expectation value $\mathbb{E}(\tilde{\boldsymbol{y}})$. This relation is given by

$$\begin{aligned}\mathbb{E}(\tilde{\boldsymbol{y}})^2=&(\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}}))^2\\=&\boldsymbol{f}(\boldsymbol{x})^2+2Bias(\tilde{\boldsymbol{y}})\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}})^2\end{aligned}\tag{12}$$

$$\begin{aligned}=&\sigma^2+Var(\tilde{\boldsymbol{y}})+\mathbb{E}(\boldsymbol{f}(\boldsymbol{x})^2)-2\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))\mathbb{E}(\tilde{\boldsymbol{y}})+\boldsymbol{f}(\boldsymbol{x})^2\\&+2Bias(\tilde{\boldsymbol{y}})\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}})^2\end{aligned}$$

We note that $\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))=\boldsymbol{f}(\boldsymbol{x})$ and $\mathbb{E}(\boldsymbol{f}(\boldsymbol{x}))^2=\boldsymbol{f}(\boldsymbol{x})^2$. Note also that using Eq.(12) we get that $\mathbb{E}(\tilde{\boldsymbol{y}})=\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}})$.

Combining these, we get:

$$\begin{aligned}=&\sigma^2+Var(\tilde{\boldsymbol{y}})+(\boldsymbol{f}(\boldsymbol{x})^2)-2\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}})]\\&+\boldsymbol{f}(\boldsymbol{x})^2+2Bias(\tilde{\boldsymbol{y}})\boldsymbol{f}(\boldsymbol{x})+Bias(\tilde{\boldsymbol{y}})^2.\end{aligned}$$

Several terms cancel out and we are finally left with

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right]=\sigma^2+Var(\tilde{\boldsymbol{y}})+Bias(\tilde{\boldsymbol{y}})^2.$$