FYS-STK4155
Project 1

Trude Halvorsen & Tiril K. Trondsen

University of Oslo, Department of Physics

October 4, 2024

In this project, we studied different regression techniques, including Ordinary Least Squares (OLS), Ridge, and Lasso regression, and applied them to both synthetic and real world data. Our results demonstrate that for the Franke function, Ridge regression with polynomial degree 4 and λ= 0.0001 yields the optimal model, which balances the performance of the model with its complexity. In contrast, for the real terrain data, OLS performed the best at capturing the complexity of our data, with the optimal degree found to be at around 16.

1 Introduction

Topographic data represent the terrain of our planet's surface, including mighty mountains and deep valleys. Such data are commonly used in different fields, such as navigation and engineering. Our primary goal in this project was to understand how various regression techniques, particularly Ordinary Least Squares (OLS), Ridge, and Lasso regression can be applied to both synthetic and real world data such as topographic data, revealing the models' strengths and limitations. We also aimed to explore how resampling techniques, such as bootstrap and cross-validation, can effectively evaluate the accuracy of models using different training and testing datasets.

We started our project by applying the different regression methods to the two-dimensional Franke function, a common test function used for evaluating fitting algorithms. After ensuring that our codes were functional and properly tested, we applied our models to real topographic data. Beginning with OLS, we analyzed the polynomial approximations of the Franke function and studied the model performance and effectiveness through Mean Squared Error (MSE) and the R^2 score. Furthermore, we wanted to improve the robustness of our models and integrated Ridge and Lasso regression. Then we compared these with

OLS and studied the bias-variance trade-off using the resampling methods. Finally, we expanded our analysis to real-world topographic data, comparing and discussing the performance of each model. We studied polynomial functions with increasing complexity, up to 20th degree, and used various values for the regularisation parameter λ, which helps to adjust the model to focus on the important coefficients.

The structure of this report is as follows: Firstly, we introduce the theoretical background of the regression techniques and models used. Next, we present the results from applying these methods to both the Franke function and real world data. Finally, we discuss the model performance and efficiency, before concluding with a summary of our key findings.

2 Method

Regression analysis is an important concept in machine learning. Its main goal is to find the best-fit line that represents the relationship between features and a target. This enables us to make predictions based on new input data, meaning we can input values for the features and use the regression model to estimate the corresponding target value. For instance, if we have a model that predicts house prices based on features like size, location, and number of bedrooms, we can enter the details of a new house to estimate its price.

There are different types of regression techniques, such as OLS, Ridge, and Lasso regression, and each method has its unique way of fitting the data. While some methods focus on minimizing the overall error in predictions, other models use regularization techniques to prevent overfitting, ensuring that the model generalizes well to unseen data. By understanding these techniques, we can choose the best method for our specific dataset and goal.

2.1 Linear Regression

We have made the assumptions that there exists a continuous function $\boldsymbol{f}(\boldsymbol{x})$ and a normally distributed error $\boldsymbol{\epsilon} \sim N(0, \sigma^2)$ due to noise or other uncertain factors, which describes our data, as shown in equation 1:

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{1}$$

We approximate this function $\boldsymbol{f}(\boldsymbol{x})$ with our model $\tilde{\boldsymbol{y}}$ from the solution of the linear regression equation, that is our function $\boldsymbol{f}$ is approximated by $\tilde{\boldsymbol{y}}$, where we minimized $(\mathbf{y} - \tilde{\boldsymbol{y}})^{\mathbf{2}}$ with:

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

Where $\mathbf{X}$ is the design matrix and the approximation $\tilde{\boldsymbol{y}}$:

$$\tilde{\boldsymbol{y}} = \sum_{j=0}^{n-1} \beta_j x_i^j$$

To analyze our models we use the MSE and R^2 values. The mean square error (MSE) is a risk metric corresponding to the expected value of the squared error or loss [1], defined as:

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

The closer the fit, the smaller the MSE value becomes, with the optimal case being an MSE of zero.

The R^2 score is defined as:

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

Where $\tilde{\boldsymbol{y}}_{\boldsymbol{i}}$ is the predicted value of the i-th sample and y_i is the corresponding true value, and we define the mean value of $\boldsymbol{y}$ as:

$$\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$$

The R^2 score function computes the coefficient of determination, which measures how well the model is likely to predict future samples, and the best possible score is 1.0 [1].

2.1.1 The OLS Regression

The Ordinary Least Squares method (OLS) is a technique used in regression analysis to find the best-fitting linear relationship between a dependent variable and one or more independent variables. OLS achieves this by estimating the parameters that minimize the sum of the squared differences between the actual data points and the predicted values of the model.

To find the optimal parameters $\boldsymbol{\beta}$, we define a cost function, which provides a measure of the error between the actual values $\mathbf{y}_i$ we want to predict and the parameterized values $\tilde{\mathbf{y}}_i$. The cost function for OLS is defined as follows:

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

In this equation, $C(\boldsymbol{\beta})$ represents the mean squared error, y_i is the actual value, and $\tilde{y}_i$ is the predicted value. The goal is to minimize this cost function to estimate the parameters $\boldsymbol{\beta}$. The optimal parameters that minimize the cost function can be derived using the formula:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Here, $\mathbf{X}$ is the design matrix containing the independent variables, and $\mathbf{y}$ is the vector of observed dependent variable values.

The expectation value of $\boldsymbol{y}$, from equation 1, for a given element i is given by equation 3.

$$\begin{aligned}
\mathbb{E}(y_i) &= \sum_j x_{ij}\boldsymbol{\beta}_j = \mathbf{X}_{i,*}\boldsymbol{\beta} \\
\hat{y} &= \mathbf{X}\boldsymbol{\beta} \approx f(x) \\
y &= \mathbf{X}\boldsymbol{\beta} + \epsilon \\
y_i &= \sum_{j=0}^{p-1} \mathbf{X}_{ij}\boldsymbol{\beta}_j + \epsilon_i \\
\mathbb{E}[y_i] &= \mathbb{E}\left[\sum_{j=0}^{p-1} \mathbf{X}_{ij}\boldsymbol{\beta}_j + \epsilon_i\right] \\
&= \mathbb{E}\left(\mathbf{X}_{i,*}\boldsymbol{\beta}\right) + \mathbb{E}(\epsilon_i) \\
&= \mathbf{X}_{i,*}\boldsymbol{\beta}
\end{aligned} \tag{3}$$

Where the expectation of the error is $\mathbb{E}[\epsilon_i] = 0$.

Furthermore, the variance of $\boldsymbol{y}$ for a given element i is given by equation 4:

$$\mathrm{Var}(y_i) = \mathbb{E}[y_i^2] - (\mathbb{E}[y_i])^2 = \sigma^2 \tag{4}$$

The steps for the derivation are as follows:

$$\begin{aligned}
\mathrm{Var}(y_i) &= \mathbb{E}[y_i^2] - (\mathbb{E}[y_i])^2 \\
&= \mathbb{E}\left[(\mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i)^2\right] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\
&= \mathbb{E}\left[(\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2(\mathbf{X}_{i,*}\boldsymbol{\beta})\epsilon_i + \epsilon_i^2\right] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\
&= \mathbb{E}\left[(\mathbf{X}_{i,*}\boldsymbol{\beta})^2\right] + 2\mathbb{E}\left[(\mathbf{X}_{i,*}\boldsymbol{\beta})\epsilon_i\right] + \mathbb{E}\left[\epsilon_i^2\right] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\
&= \mathbb{E}\left[\epsilon_i^2\right] \\
&= \mathrm{Var}(\epsilon_i) \\
&= \sigma^2
\end{aligned}$$

Therefore, $y_i \sim N(\mathbf{X}_{i,*}\boldsymbol{\beta}, \sigma^2)$, and y follows a normal distribution with mean value $\mathbf{X}\boldsymbol{\beta}$ and variance σ^2.

With the OLS expressions for the optimal parameters $\hat{\boldsymbol{\beta}}$, we can show that $\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$ as demonstrated in equation 5:

$$
\begin{aligned}
\mathbb{E}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\right] \\
&= \mathbb{E}\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta}+\epsilon)\right] \\
&= (\mathbf{X}^T\mathbf{X}^{-1})\mathbf{X}^T\mathbf{X}\mathbb{E}(\boldsymbol{\beta}) + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}(\epsilon) \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\
&= \boldsymbol{\beta}
\end{aligned} \tag{5}
$$

Moreover, the variance of the optimal $\boldsymbol{\beta}$ is given by

$$
\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} \tag{6}
$$

We can show this with the following steps:

$$
\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}\left[\left(\hat{\boldsymbol{\beta}} - \mathbb{E}(\hat{\boldsymbol{\beta}})\right)\left(\hat{\boldsymbol{\beta}} - \mathbb{E}(\hat{\boldsymbol{\beta}})\right)^T\right] \\
&= \mathbb{E}\left[\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} - \boldsymbol{\beta}\right)\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} - \boldsymbol{\beta}\right)^T\right] \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}\mathbf{y}^T]\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T
\end{aligned}
$$

We use that $\mathbb{E}[\mathbf{y}\mathbf{y}^T] = \mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T + \sigma^2\mathbf{I}$:

$$
\begin{aligned}
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\left(\mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T + \sigma^2\right)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \boldsymbol{\beta}\boldsymbol{\beta}^T + \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}
$$

We see that the variance of the optimal parameters is influenced by the design matrix $\mathbf{X}$ and the noise variance σ^2.

2.1.2 The Ridge Regression

OLS is a straightforward way to do regression, but it can overfit the training data, especially when the model has a high complexity. Ridge regression gives us a more general model that performs better on new data by adding a regularization term. The regularization helps prevent overfitting by constraining the coefficients of the regression model, leading to more robust predictions on unseen data.

Ridge regression will shrink the regression coefficients by penalizing their size. The ridge coefficients, $\hat{\boldsymbol{\beta}}_{Ridge}$, minimizes a penalized residual sum of squares. The parameter λ controls the amount of shrinkage, where a larger λ will result in a larger amount of shrinking [6]. The main idea behind Ridge

regression is therefore that it reduces the values β_i as a function of λ [1].

To find the optimal parameters $\hat{\boldsymbol{\beta}}_{Ridge}$, we define a new cost function to be optimized (with the regularization parameter λ). This leads to the Ridge regression minimization problem, where we require that $\|\boldsymbol{\beta}\|_2^2 \leq t$, with t being a finite number larger than zero:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

Using the matrix vector expression for the cost function for Ridge regression, and leaving out the parameter $1/n$ in front of the standard mean squared error equation, we have:

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

Further, we are taking the derivatives with respect to β, and obtain the optimal parameters for Ridge regression:

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

2.1.3 The Lasso Regression

Lasso and ridge are two very similar regression techniques, but there are also some key differences [3]. Lasso stands for Least Absolute Shrinkage and Selection Operator. It is a regression technique that is especially useful when dealing with datasets that have many features, as it helps to prevent overfitting.

Lasso regression modifies the ordinary least squares (OLS) cost function by adding a regularization term that penalizes the absolute size of the coefficients. Minimizing the new cost function 7 leads to Lasso regression:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \tag{7}$$

The first term represents the least squares loss, which measures how well the model fits the data, while the second term is the Lasso penalty, with λ controlling the strength of the penalty.

Lasso can force some coefficients to reduce their values until they eventually become zero while others remain unaffected or shrink less dramatically. It is useful for selecting important features because it reduces the complexity of models by removing irrelevant variables that do not contribute to the overall prediction [4]. This helps reduce the risk of overfitting. By finding the best value for λ we can balance between fitting the training data and maintaining the model simplicity.

2.2 Bias-variance trade-off

The bias-variance tradeoff seeks to balance underfitting and overfitting by finding the optimal model complexity, where we have low bias and low variance, minimizing the overall test error.

We consider a dataset $\mathcal{L}$ consisting of data $\boldsymbol{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{x}_i), j = 0, \ldots, n-1\}$. We assume that the true data is generated from a noisy model:

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

Where ϵ is normally distributed with mean zero and standard deviation σ^2.

From the derivation of the OLS method, we defined an approximation to $\boldsymbol{f}$ in terms of parameters $\boldsymbol{\beta}$ and design matrix $\boldsymbol{X}$ as:

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$$

The $\boldsymbol{\beta}$-parameters are then found by optimizing the cost function:

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

Where the expected value $\mathbb{E}$ is the sample value.

We want to rewrite this expression to contain the variance of the model (deviation from the true data), the mean value of the model (bias), as well as the variance of the noise.

We start off by writing out the cost function fully:

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[\boldsymbol{y}^2\right] - 2\mathbb{E}\left[\boldsymbol{y}\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}^2\right] \tag{8}$$

Initially, we will look at the first term, $\mathbb{E}[\boldsymbol{y}^2]$, making the substitution of $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. For simplification we will write $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}$:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}^2] &= \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon})^2] \\
&= \mathbb{E}[\boldsymbol{f}^2] + 2\mathbb{E}[\boldsymbol{f}\boldsymbol{\epsilon}] + \mathbb{E}[\boldsymbol{\epsilon}^2] \\
&= \mathbb{E}[\boldsymbol{f}^2] + 2\mathbb{E}[\boldsymbol{f}]\mathbb{E}[\boldsymbol{\epsilon}] + \mathbb{E}[\boldsymbol{\epsilon}^2]
\end{aligned}$$

Since ϵ is normally distributed with mean zero and standard deviation σ^2, the term simplifies to:

$$\mathbb{E}[\boldsymbol{y}^2] = \boldsymbol{f}^2 + \sigma^2 \tag{9}$$

Next, we look at the term $2\mathbb{E}[\boldsymbol{y}\hat{\boldsymbol{y}}]$:

$$2\mathbb{E}[\boldsymbol{y}\hat{\boldsymbol{y}}] = 2\mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon})\tilde{\boldsymbol{y}}]$$

$$= 2\mathbb{E}[\boldsymbol{f}\tilde{\boldsymbol{y}}] + 2\mathbb{E}[\boldsymbol{\epsilon}\tilde{\boldsymbol{y}}]$$
$$= 2\boldsymbol{f}\mathbb{E}[\tilde{\boldsymbol{y}}] + 2\mathbb{E}[\boldsymbol{\epsilon}]\mathbb{E}[\tilde{\boldsymbol{y}}]$$

Again, since we know that ϵ has mean zero, we can make the substitution of $\mathbb{E}[\boldsymbol{\epsilon}] = 0$, resulting in the term:

$$2\mathbb{E}[\boldsymbol{y}\hat{\boldsymbol{y}}] = 2\boldsymbol{f}\mathbb{E}[\tilde{\boldsymbol{y}}] \tag{10}$$

The last term to consider is $\mathbb{E}[\tilde{\boldsymbol{y}}^2]$. From the definition of variance, we know that:

$$\text{var}[\tilde{y}] = \mathbb{E}[\tilde{\boldsymbol{y}}^2] - (\mathbb{E}[\tilde{\boldsymbol{y}}])^2$$

meaning that we can rewrite the last term $\mathbb{E}[\tilde{\boldsymbol{y}}^2]$ as:

$$\mathbb{E}[\tilde{\boldsymbol{y}}^2] = \text{var}[\tilde{y}] + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 \tag{11}$$

We can now rewrite Equation 8, using the terms we found in Equation 9, Equation 10 and Equation 11:

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] &= \boldsymbol{f}^2 + \sigma^2 - 2\boldsymbol{f}\mathbb{E}[\tilde{\boldsymbol{y}}] + \text{var}[\tilde{y}] + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 \\ &= \boldsymbol{f}^2 - 2\boldsymbol{f}\mathbb{E}[\tilde{\boldsymbol{y}}] + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \text{var}[\tilde{y}] + \sigma^2 \\ &= \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \text{var}[\tilde{\boldsymbol{y}}] + \sigma^2 \\ &= \text{Bias}[\tilde{y}] + \text{var}[\tilde{y}] + \sigma^2\end{aligned}$$

The bias occurs due to the model making wrong assumptions and not understanding the complexity in the data. This results in a difference between the average prediction of model and the true value.

The variance tells us how much our model will change if we estimate it using different training data. If a method has high variance then small changes in training data leads to large changes in the model. So we have a higher variance when we have more flexible statistical methods. Ideally the estimate in a model should not vary too much between training sets.

σ^2 represents the variance of the noise in the data. This error can not be reduced because it is inherent in the noisy model.

2.3 Resampling Methods

To further investigate how well our models are working, we introduce resampling methods. This involves repeatedly drawing samples from a training set, and fitting our desired model on these samples. This method can give us more information about our model, compared to only fitting our model once using our original training sample [1]. We will discuss and use the two most commonly used resampling methods, namely bootstrap and cross-validation.

2.3.1 Bootstrap

By dividing out data into different subsets and training or testing on these multiple subsets, bootstrap helps us estimate the uncertainty in our models predictions, and help us understand how the model performs.

The idea behind the bootstrap method is to randomly draw n numbers from the training data, with replacement. Each sample is the same size as the original dataset, and we evaluate $\hat{\beta}$ from the samples we have drawn. This process is repeated k times, resulting in k different estimations of $\hat{\beta}$. By calculating the average of these estimates, we get a more robust total estimate of the parameter.

2.3.2 K-Fold Cross-Validation

Since we often work with limited amounts of data, we are not able to set aside a validation set to assess how well our model is performing. [6]. Another issue with repetitive, random splitting of data, is that samples may be overly reused in either the training or test data. The K-fold cross-validation avoids this by dividing the samples into K subsets. At each split, one of the K subsets is being used as the test set, while the other $K-1$ sets are being used for training. This ensures that we will get a balanced representation of each sample, so no sample is having an unbalanced influence on the model. After all of the subsets have been used as test data, we retain the evaluation score and discard the model [1].

2.4 Implementations

Before applying our models to real-world data, we first tested our implementation using the Franke function, defined by 12 and visualized in Figure 1 below.

2.4.1 Franke function

$$\begin{aligned} f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) &+ \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ + \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) &- \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned} \tag{12}$$

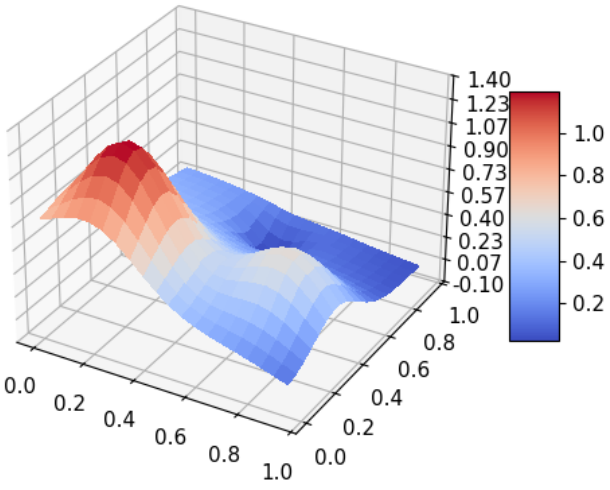

Figure 1: The Franke Function visualized.

The Franke function provides a useful benchmark for testing our models with synthetic data that resembles real-world surface complexities. It allows us to compare OLS, Ridge, and Lasso regression techniques, ensuring our implementation is robust and helping us assess model accuracy, complexity, and generalizability before applying the methods to real data.

For all of our implementations of OLS, Ridge and Lasso regression on the Franke function, we used 10 000 data points, and added stochastic noise. However, for the bootstrap and K-fold cross-validation, we reduced the number of data points to 100 due to computational complexity. In addition, we chose to evaluate our Ridge and Lasso regression models using regularization parameters $\lambda = [0.0001, 0.001, 0.01, 0.1, 1.0]$.

We randomly split the data into test and training sets. We used 20% of the data for testing and 80% for training, using Scikit-learn. By splitting our data, we can evaluate how well our model performs on unseen data. If we train and test our model on the same dataset, the model might learn the noise and other patters in the data, and we risk overfitting. This can give a worse performance when the model gets new data.

To scale our data, we used the StandardScaler from Scikit-learn. This function standardize features by removing the mean and, scaling them to unit variance. This will ensure that all the features are on the same scale, which will reduce the risk of any single features dominating due to a larger magnitude. The process of scaling data will enhance the performance of our model, and make our inputs more appropriate for the algorithms we want to use.

We implemented OLS regression for the Franke function using matrix inversion as shown in the code snippet below. To estimate the coefficients β, we use the standard OLS matrix coefficient formula, previously introduced in equation 2.1.1. We used the same approach for all regression methods, adding λ and identity matrix I for Rigde and Lasso regression.

```
for i in range(1, degree + 1):
    c = int((i + 2) * (i + 1) / 2) # number of columns
    X_tilde_train = X_train_scaled[:, :c] # choosing columns for the degree
    X_tilde_test = X_test_scaled[:, :c]

    beta = np.linalg.pinv(X_tilde_train.T @ X_tilde_train) @ X_tilde_train.T @ z_train_scaled
    beta_values.append(beta)

    y_train_pred = X_tilde_train @ beta + z_train_mean
    y_test_pred = X_tilde_test @ beta + z_train_mean

    MSE_train_values.append(mean_squared_error(z_train, y_train_pred))
    MSE_test_values.append(mean_squared_error(z_test, y_test_pred))
    R2_train_values.append(r2_score(z_train, y_train_pred))
    R2_test_values.append(r2_score(z_test, y_test_pred))

degrees = np.arange(1, degree + 1)
```

2.4.2 Real World Data

We used real terrain data from the Project 1 Assignment Text [1], which provides digital terrain data over a region of Norway. To visualize this data in three dimensions, we used the imageio library [5] and read about three-dimensional plotting in Matplotlib from [2]. we plotted the terrain data in both 2D and 3D to better understand its structure. Due to the large size of the dataset, we chose to sample every 10th point from the original data, to reduce the size, while still keeping the necessary information. Then we began applying the regression techniques to the terrain data. We split the data into training and test sets with a 80/20 split. For the terrain data, we constructed a polynomial design matrix using a function create X that we found in the lecture notes from week 35 [1].

We also scaled and centered out real-world data usind Scikit-learns StandardScaler, so that the features contribute equally to the model. We removed the intercept and then centered the data by subtracting the mean, which helps improve the stability of the model. For all of our implementation of models on the real terrain data, we used 5000 data points, except for on the K-fold cross-validation where we reduced it to 3000 points, due to computational efficiency.

3 Results

In the following results section, we present the most relevant plots to demonstrate the performance of the different regression techniques on both synthetic

and real-world data. We focus on the comparison between OLS, Ridge, and Lasso regression, as well as the impact of cross-validation and bias-variance trade-off.

3.1 Franke Function

3.1.1 OLS

We generate our own dataset for the Franke Function, with $x, y \in [0, 1]$, uniformly distributed. In addition, we added stochastic noise to the function using the normal distribution. We performed a standard ordinary least square regression analysis with polynomials in x and y up to the 5th order, using 10 000 data points.

Plots below show MSE, R^2 and parameters β as functions of the polynomial degree.

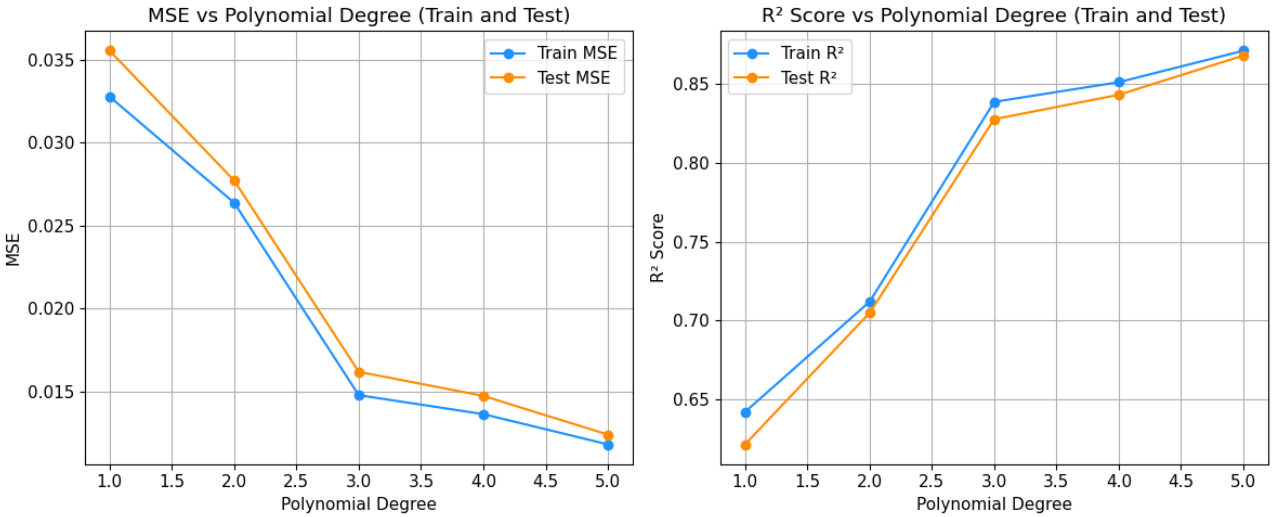

Figure 2: MSE and R^2 values plotted against polynomial degree with n=10 000 data points.

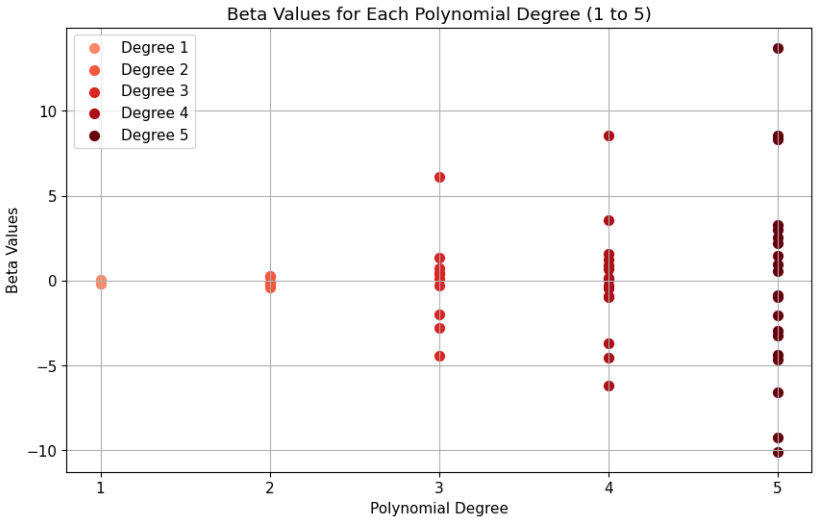

Figure 3: Beta values for each polynomial degree, with n=10 000 data points.

From Figure 2 we observe that the MSE is decreasing towards zero, while the R^2 score is increasing towards 1, which is the expected behaviour. The MSE for lower polynomial degrees are high, because the model is not complex enough to capture the patterns in the data correctly. As the model grows more complex, it is apparent that also the MSE will decrease. Our test and train functions are following each other closely, due to the large amount of data points we have chosen, which gives the model more data to train on, and thus makes it react well to unseen data.

As one can see in Figure 2, we are only visualising the MSE and R^2 score for polynomials up to degree 5. For even higher polynomial degrees, we would expect that the MSE for the test data would increase, as the model would start overfitting. When only looking at polynomial degrees up to five though, we observe that our lowest test MSE occurs at degree 5.

The β-values for each polynomial degree is visualised in Figure 3. The number of β-values will increase with the increasing complexity of the model, because the betas for a given polynomial of degree n is given as $\beta = [\beta_0, \beta_1, \beta_2, ..., \beta_n]$. We are also observing that as we increase the polynomial degree, the variation and magnitude of the β-values will also increase. These larger β-values may suggest that the higher-degree models are more complex, which means that the model can fit data precisely, but may be prone to overfitting.

3.1.2 Ridge

Moving further, we are adding Ridge regression for the Franke function. In Figure 4 we have visualised the MSE with respect to the polynomial degree.

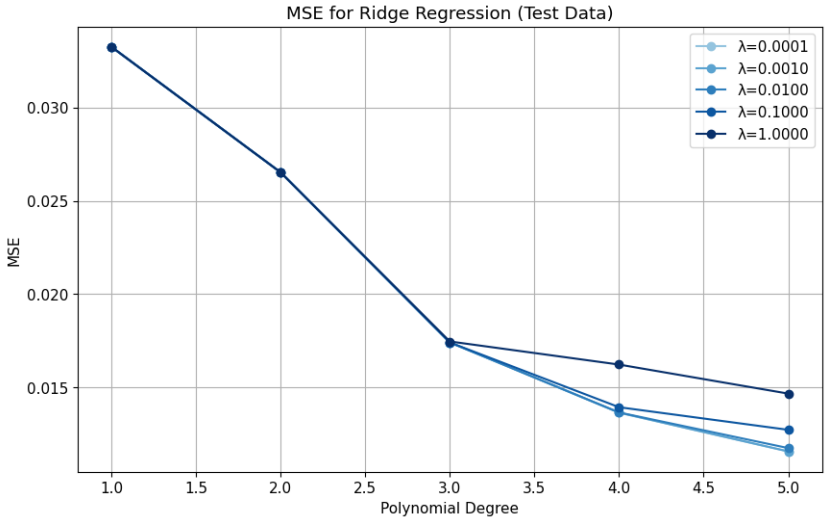

Figure 4: MSE for Ridge regression on test data with n=10 000 data points.

From the graph, it is apparent that the λ-value that yields the lowest MSE is $\lambda = 0.0001$. We also see the same tendencies as we did for OLS on the Franke function, where the MSE is decreasing with increasing model complexity.

Comparing the results for the MSE using Ridge regression with those obtained from using OLS, we can see that for low λ-values, the two methods are behaving similarly. That is because for $\lambda = 0$, we will simply have Ordinary Least Squares regression. For the larger λ, we observe that the model is giving a worse MSE for polynomial degrees higher than 3, than the smaller λ-values. This could be because larger λ-values makes the model overly simple, and therefore fail to capture the complexity in the data correctly, resulting in underfitting.

3.1.3 Lasso

In Figure 5 we have visualised the MSE for Lasso regression on the test data for the Franke function, for varying λ-values using n=10 000 data points.

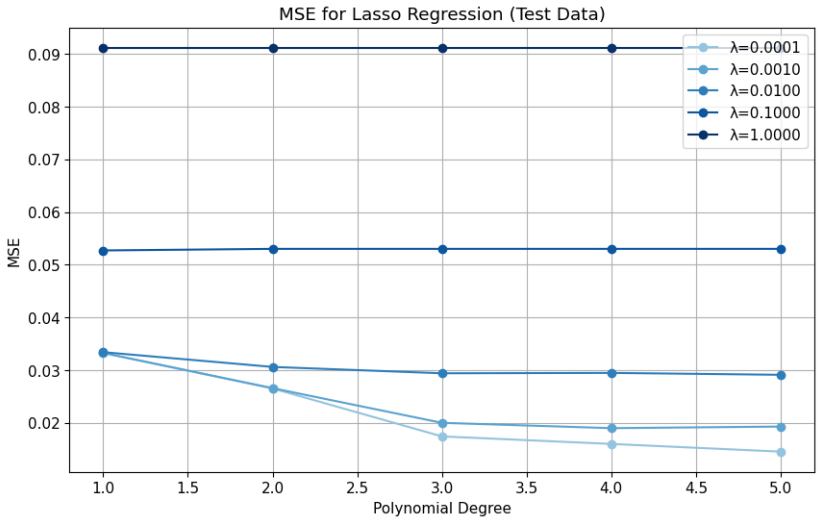

Figure 5: MSE for Lasso regression on test data with n=10 000 data points.

We observe from the plot that for the larger values of λ, the MSE does not converge. For the smaller λ-values, it is apparent that the MSE is decreasing with the increasing model complexity.

Lasso regression not only regularizes the coefficients but also forces some of them to zero. This is useful when some variables are irrelevant if we have correlated terms. As we are performing regression on the Franke function, which has correlation between terms, this means that Lasso regression will set some of the correlated terms to zero. This means that we are losing information that may not actually be irrelevant, giving us a worse MSE than we achieved using both OLS and Ridge.

3.1.4 Comparing OLS, Ridge and Lasso

By performing a grid search to find the optimal λ-value for Ridge and Lasso regression, we found the optimal value to be $\lambda = 0.0001$ for both Lasso and Ridge. The MSE for all three regression models have been plotted together in Figure 6.

From the plot presented, it looks like Ridge performs better in our case. We know from the method section that Ridge regression modifies the OLS by correcting for high-value coefficients by adding the penalty term. OLS can be unstable if there is a high correlation between features. When dealing with the Franke function, we know that there is some correlation between the terms, which Ridge then helps with by adding the regularization. Lasso can reduce

coefficients to zero, unlike Ridge, which only shrinks them. By setting them to zero, we remove information that might be relevant and we lose valuable information, which is why Ridge is the better option in this case.

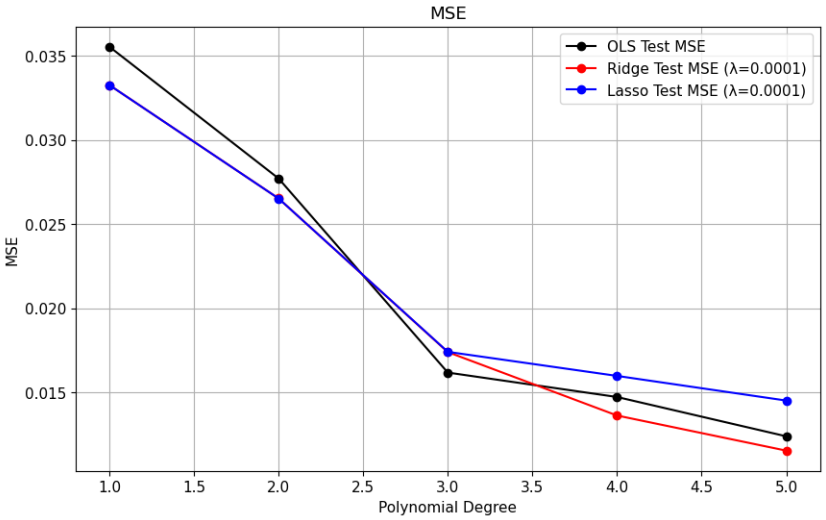

Figure 6: Comparison of the MSE with OLS, Ridge and Lasso on test data, with $\lambda = 0.0001$ and n=10 000 data points.

The table 1 shows the metrics for both Ridge and Lasso regression models, focusing on the best estimated lambda values, MSE, and R^2 scores.

Model	Best Estimated Lambda	MSE Score	R^2 Score
Ridge	0.0001	0.01366	0.86480
Lasso	0.0001	0.02015	0.80053

Table 1: Comparison of Ridge and Lasso Regression Metrics

Both Ridge and Lasso have the same optimal lambda value of 0.0001. Ridge has a lower MSE and higher R^2 score, which may suggest that Ridge handles feature correlation more effectively, while the penalty term in Lasso may remove important variables by setting coefficients to zero.

3.1.5 Bias variance trade off

Even if we previously have found the model with the lowest MSE, this does not mean that it necessarily is the best model overall. To find the best fit, we need to balance the performance of the model with the complexity. To do this, we will be performing a bias-variance trade-off, using the bootstrap method, which is visualised in Figure 7.

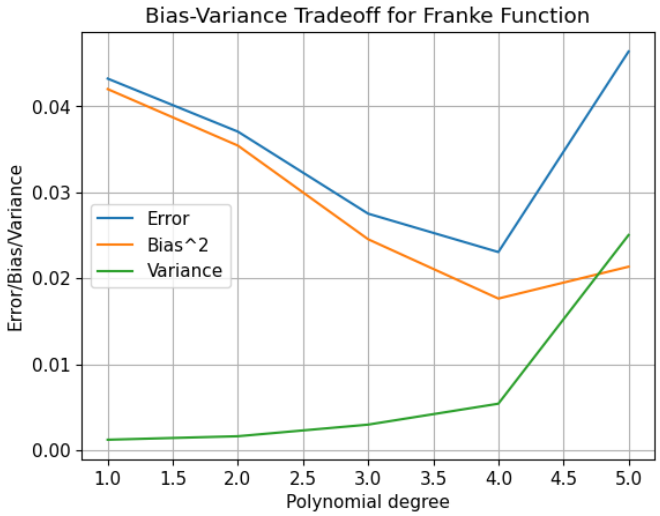

Figure 7: Bias-variance trade-off plotted against polynomial degree for the Franke function with n=100 data points.

The idea behind the trade-off is that we are looking to find the balance between underfitting and overfitting our model. Observing this figure, we will see that the test error is minimized at degree 4. This is where the bias and variance is low at the same time, meaning that the optimal model complexity is a 4 degree polynomial. For higher and lower degrees, the model will be prone to respectively over and underfitting.

3.1.6 Cross validation

Lastly, we want to evaluate the model performance using cross validation. In Figure 8, we have visualised the MSE values for various K-folds for increasing model complexity.

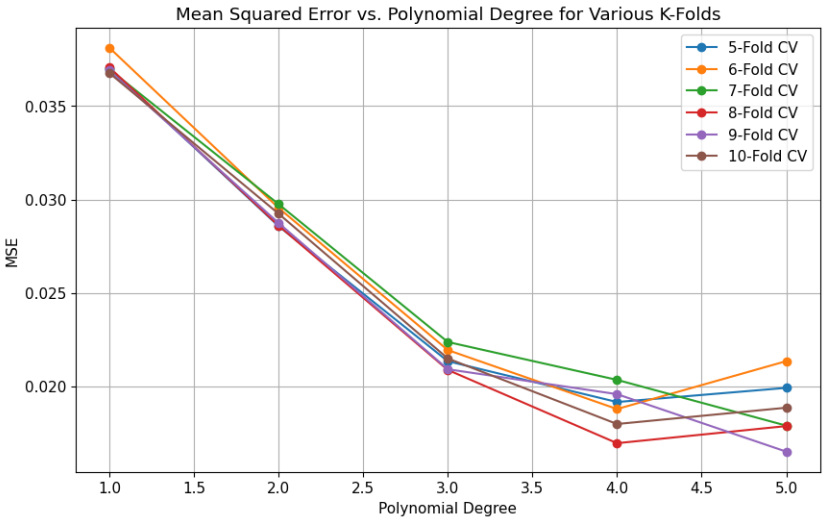

Figure 8: K-fold plotted against polynomial degree for the Franke function with n=100 data points.

We can see that the MSE is decreasing with the increasing polynomial degrees up to around degree 4. This is understandable, as more complex models capture the underlying patterns in the Franke function. Around degree 4, the MSE seems to stabilize, and then slightly increase for some values of K. The MSE values are very similar for the different folds, which indicate that the performance is not too sensitive to the choice of K. The optimal polynomial degree is therefore around degree 4, as we can see that there is some overfitting for the higher degrees. 8 folds yields the lowest MSE score for polynomial degree 4.

3.2 Real Terrain Data

Moving on to the real terrain data, we first downscaled the data by picking out every 10th data point. The visualisation of the downscaled terrain can be seen in Figure 9.

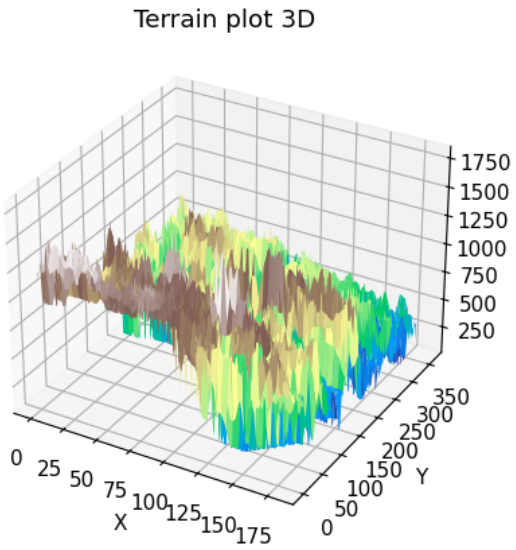

Figure 9: Our 3D terrain visualized.

3.2.1 OLS

In figure 10 we see how the MSE and R^2 values for the train and test datasets for real-world data varies with polynomials up to 20th degree.

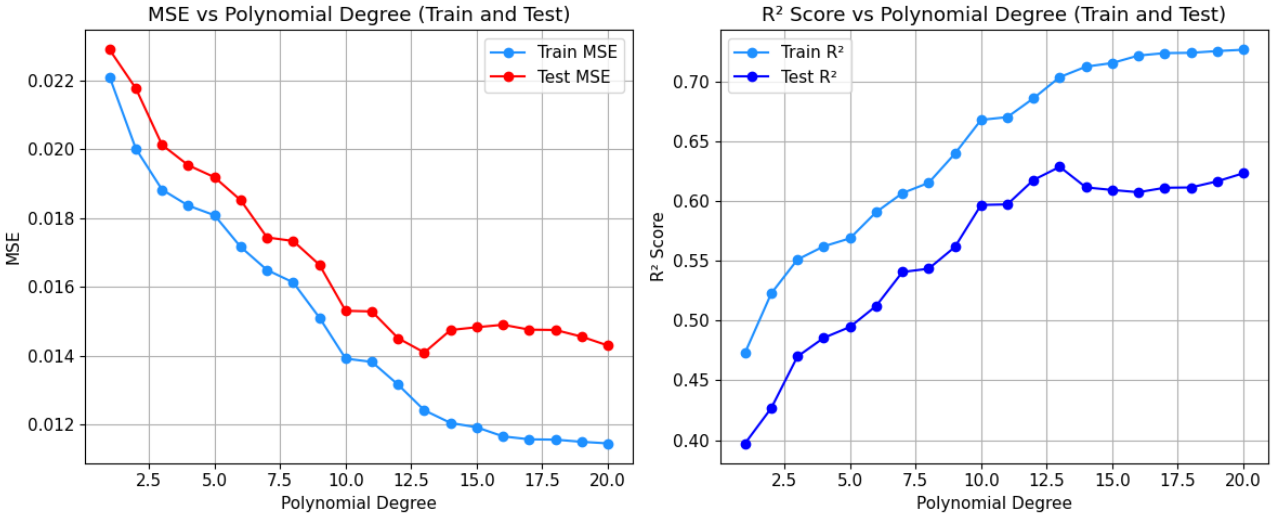

Figure 10: The MSE and R^2 for the train and test datasets for real terrain data with n=5000 data points.

As the polynomial degree increases, the training MSE continues to fall. However, for the test data, we observe higher MSE around the 14th- and 15th-degree polynomials compared to the 12th- and 13th-degree. The higher polynomial degrees provide a higher complexity, where the model performs well on training data, but not as good on new test data. The model may start picking up noise in the data rather than actual patterns, so we get overfitting. Therefore, it is more important to focus on the test MSE when evaluating the model rather than the training MSE. We observe a similar trend in the R^2 score, but we only included the plot for MSE in the report.

3.2.2 Comparing OLS, Ridge and Lasso

In graph 11 we see how the MSE varies with polynomial degree for OLS, Ridge and Lasso. We used the best λ value for Ridge and Lasso, which we found to be 0.0001 from the previous plots with different lambdas.

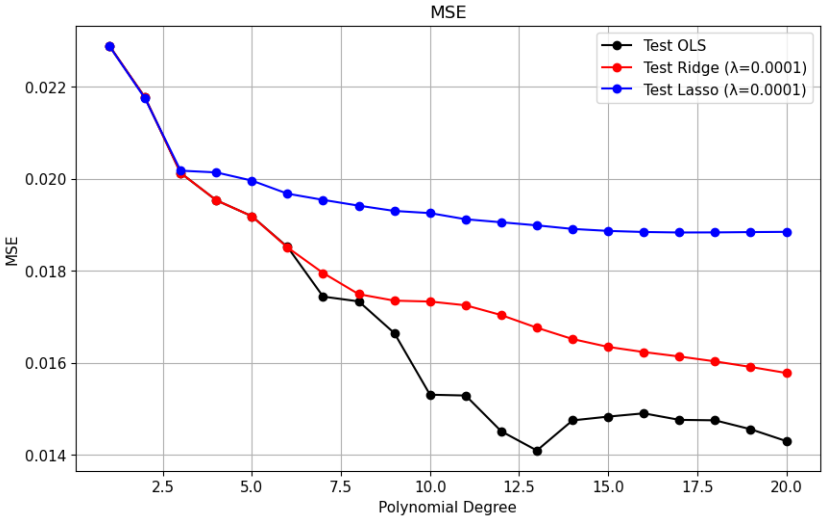

Figure 11: Shows the MSE versus polynomial degree for OLS, Ridge and Lasso.

In graph 11 we observe that the MSE for OLS decreases significantly as the degree increases, with the lowest point occuring around the 13th degree. After this degree, the MSE increases again and fluctuates a little up to degree 20. Overall the OLS has the lowest MSE at higher degrees. The OLS does not have any regularization, and then the model can capture the complexity at higher degrees.

For Rigde regression, the MSE is higher for degrees over 6th, compared to OLS. The penalty term in Ridge regression can prevent the model from fitting the more complex polynomials at the higher degrees. Moving on to Lasso

regression, we observe that it performs worse compared to OLS and Ridge, with the MSE values high and relative stable from around 4th degree. The reason for this might be that Lasso can prevent the model for capturing the more complex models, as it can set coefficients to zero and therefore discard important polynomial terms. So all three models perform similarly until 3th degree, where the OLS and Ridge continue to decrease, while Lasso stabilize. At 7th degree, the OLS outperform Ridge.

3.2.3 Bias-variance trade-off

Figure 12 illustrates the tradeoff between the bias and variance.

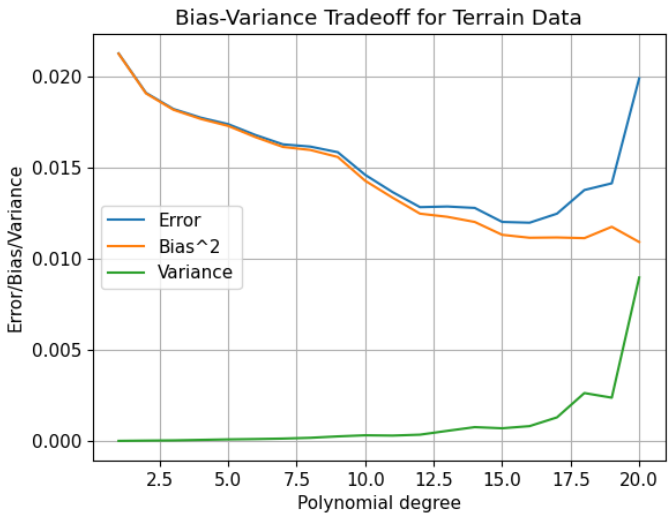

Figure 12: Bias-Variance Trade-off for Terrain data.

The bias starts high at lower degrees, because the model is not flexible enough to represent the true data, leading to underfitting. As degrees increase up to 16 degree, the bias decreases and flattens out more, suggesting that higher polynomial degrees can capture the underlying structure of the data and reduce the bias.

The variance remains very low for polynomial degrees up to around 12 degree, meaning that the model is stable and its predictions do not fluctuate much with changes in the data. Then the variance starts to increase slowly, but then we get a sharp increase around polynomial degree 18. This is a sign of overfitting where the model becomes too complex and starts to capture noise.

At high degrees the variance is high because small changes in the training data can lead to large differences in predictions, so we get a sensitive model. The

error decreases as bias decreases with increasing degree. Then the error starts to increase around degree 18 and higher, reflecting the impact of the variance.

The bias-variance trade-off gives us a picture of where the model is underfitting and overfitting, based on the model complexity, and identifies where the bias and variance are balanced. Around degree 16 the model has a balance in complexity where the bias decreases and represents the data more accurately, and the variance is still low. This means that the optimal degree for our model using OLS is at a complexity of around degree 16.

Moreover the number of data points impact the bias and variance. Larger datasets reduce variance as the model has more information about the function to learn from. While small datasets can give a high variance as the model does not have enough data to give stable estimates and can overfit to the little amount of data. From our figure, we can see that the bias and variance is low, and therefore minimizes the total error at around degree 15.

3.2.4 Cross-validation

The last figure 13 show how the MSE varies with different polynomial degrees when applying k-fold cross-validation on the terrain data, using folds from 5 to 10.

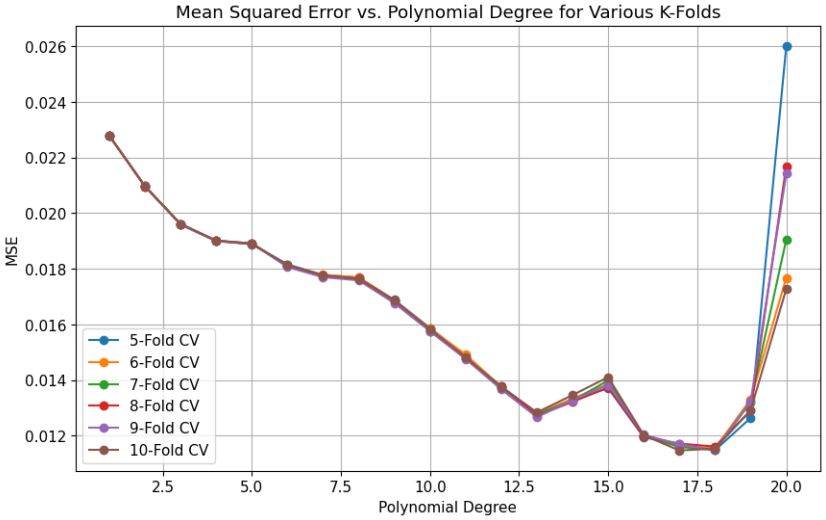

Figure 13: K-fold for Terrain data.

We see a decreasing MSE up to degree 13 for all k-folds, after this point the MSE begins to rise, then it drops around 15 degree and increases again at 19 degree, indicating overfitting, as higher degree polynomials tend to capture

noise in the data. We have a large spike at degree 20 for all k-folds, meaning a lot of overfitting for the highest degree polynomials. The results across the different k-folds are very consistent, indicating that the choice of number of folds does not have a huge impact. K-fold Cross Validation reduces the impact of data variability by using many training and testing sets, so it is not that reliant on one specific subset of data, which results in a robust way to evaluate the performance of our models.

4 Discussion

In this project, we used three regression models, OLS, Ridge, and Lasso, on both the Franke function and real-world terrain data. Our results highlighted both the strengths and weaknesses of these methods in different contexts.

Pros and Cons - OLS
OLS is the simplest model, it is easy to implement, and its formula directly estimates the relationship between the predictors and target variable, making it easy to interpret the coefficients. As seen in our results of the Franke function, OLS struggles with high correlation between features, which can lead to instability in the model, resulting in higher MSE values when compared to Ridge regression. In real terrain data, OLS captured the complexity with increasing polynomial degrees, but it also overfitted when the model became too complex for the data, which led to increased MSE at higher degrees.

Pros and Cons - Ridge
Ridge regression helps with high correlation between features by adding a penalty term, which makes the coefficient estimates more stable. When we studied the Franke function, Ridge performed a little better than OLS because it managed the high correlations. Ridge also showed a good balance between fitting the data and avoiding overfitting. This is shown by its lower MSE and higher R^2 score compared to lasso in Franke analysis. Although Ridge did well on Franke, it had some challenges with the higher polynomials in real terrain data. The penalty term in Ridge can limit its ability to capture more complex patters.

Pros and Cons - Lasso
Lasso regression is unique in its ability to shrink coefficients to zero. This can be advantageous in data where many features may be irrelevant. In our analysis, this tendency may have resulted in the removal of valuable features. This is evident in the analysis where it performed worse than both OLS and Ridge, indicating a potential loss of important information.

Moving forward, future improvements could involve more tuning of hyperparameters, using larger datasets, implementing other resampling techniques to make the model more robust and studying alternative modeling techniques.

5 Conclusions

Our analyses with OLS, Ridge, and Lasso have revealed important insights into model performance based on the characteristics of the dataset. In the Franke function context, Ridge proved to be the most effective method, with an optimal polynomial degree of 4, and $\lambda = 0.0001$, due to its ability to manage high correlation between features and prevent overfitting. In contrast, the real terrain data showed the strength of OLS in capturing complexity at higher polynomial degrees, where we found the optimal polynomial degree to be around 16.

6 Appendix

6.1 GitHub

All codes can be found at Github: https://github.com/tiriltk/FYSSTK4155/tree/main. This report does not include all generated plots. All plots for the Franke function and the real terrain data can be found in the respective Jupyter Notebooks with the corresponding names. Additional test runs can be found in the file projectcode1.ipynb.

7 Acknowledgement

Throughout this project, we got a lot of inspiration for our codes from Professor Morten Hjorth-Jensen's lecture notes [1].

The ChatGPT tool provided by UiO has been helpful in improving different parts of this project. It has been used to:

- Help adapt and modify code from our weekly exercises for the Franke function and real-world data analysis, particularly with generating plots.
- Check grammar and improve the language in the project.

References

[1] Morten Hjorth-Jensen, *Lecture Notes in Applied Data Analysis and Machine Learning*, August - September 2024. Available: https://github.com/CompPhysics/MachineLearning

[2] Jake VanderPlas, *Three-Dimensional Plotting*, 2016. Available: https://jakevdp.github.io/PythonDataScienceHandbook/04.12-three-dimensional-plotting.html

[3] Machine Learning Compass, *Ridge Regression*, 2024. Available: https://machinelearningcompass.com/machine_learning_models/ridge_regression/.

[4] Analytix Labs, *What are Lasso and Ridge Techniques?*, 2024. Available: https://medium.com/@byanalytixlabs/what-are-lasso-and-ridge-techniques-05c7f6630f6b.

[5] Imageio *Imageio: Python library for reading and writing image data.* Available: https://pypi.org/project/imageio/

[6] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer