Project 1 FYS-STK4155

Ingebrigt Kjæreng

October 10, 2023

Abstract

I explored the use of different regression methods to determine which model gave the predicted data values that as closely as possible mimicked the real data. I used the OLS method, Ridge and Lasso methods for linear regression using a Franke function containing x and y with 4 terms. I compared the predicted data compared to the mean by generating data plots, with the Lasso model performing best, then I compared the MSE and R2-values as functions of polynomial degree and value of regularization parameter lambda. The OLS method gave the best MSE for polynomial degrees 1,2 and 3, while Ridge regression gave the best results for degrees 4 and 5. The OLS method gave a perfect R2 score, but the Ridge regression performed best for R2 as it gave more realistic result. The Beta coefficients in the 3 different models were reviewed for their performance across different polynomial degrees and values of lambda. Here, the Lasso method gave the smallest values and provided the most flexibility. The residuals of the error terms were reviewed, with the OLS and Ridge regression giving distributions closest to 0. I did bootstrapping and cross-validation resampling techniques in order to study the bias-variance trade-off. I explored the same things using real topographic data. The project demonstrated the difference in ability, strength and flexibility across the 3 models and the importance of choosing the right model for solving specific problems.

1 Introduction

In this report I will be studying in details various regression methods, the Ordinary Least Squares, Ridge and Lasso methods. My aim is to show the difference in these regression methods' ability to give accurate predictions using polynomial fits for one of the methods for a multi-variable function in different degrees, and use centering to remove the mean from each feature and obtaining the intercept for other regression methods. I use different regression methods, and our purpose is to obtain polynomial fits with predicted values in multiple variables [x,y,z] that is as close as possible to the real data. This has important applications in the real world. In practical applications, these predictions are crucial for decision-making, risk assessment and resource allocation. Different datasets and problems may require different modeling techniques, as is the case in the real world, and in many Real-world scenarios, time, energy and money is limited. Choosing the most efficient regression method can lead to significant cost savings, such as reduced material waste and increased product quality. Regression analysis is often used to understand the relationships between variables. By selecting the most suitable regression method, researches can gain insights into natural phenomena, discover patterns and formulate scientific hypotheses. It can be used for medical diagnosis, predicting patient outcomes, understanding the impact of factors on environmental variables, such as climate change, pollution levels and habitat concentration. It is used in stock price prediction, risk assessment and portfolio optimization, demand forecasting, pricing strategy optimization and customer segmentation. Accurate modeling can result in increased revenue and market competitiveness.

In order to achieve the method that yield the most accurate results, one of the things I seek is to find the regression model that yield a Mean Squared Error across the regression line that is as close as possible to 0. If it is 0, it means that my predicted model is a perfect fit. In the implementation, I have plotted a given so-called ≪Franke Function≫ using OLS, Ridge regression and Lasso method with the purpose of seeing what models give data points for the scaled, split data that are as close as possible to the real data points using a specific polynomial fit, in the range from 0 to 5. I seek to minimize the MSE and the R-squared coefficient, getting these values as close as possible to zero. Then, I also look at the beta coefficients from the regression line and evaluate their performance across

the models. I looked at the performance of these parameters for these three regression methods. I evaluated these parameters for each regression. Then I have applied cross-validation and bias-variance tradeoff analysis, before applying all these techniques to real, topographic data.

Two of the most basic and useful statistical measurements I look at are the mean and variance. β often varies a lot because we have a lot of normally distributed ≪noise≫ in the error term, skewing the results, which is assumed to be the case in the real, topographic data. My job is to determine to what degree the results are influenced by this ≪noise≫. Since it is not given whether Lasso, Ridge or OLS will give the MSE that is closest to 0, I want to use our results to explain how our function is better approximated, and to identify all the variables that are taking place in determining how to arrive at the best fit with the overarching goal being to see patterns across models and developing a framework to know better what model to use given a certain type of problem. To achieve this I use statistical interpretation, I examine the model used, characteristics of model, degree of polynomial and variables of the polynomial. I look at the performance of the MSE and R-squared for these three regression methods, and how the MSE develops across different model complexities (meaning polynomial degrees). Because I want to know what types of models are best suited to different kinds of problems, I also want to see how the regression methods deal with different kinds of problems, in the case of Ridge regression the problem of data overfitting. Another thing that happens is looking at the variance of the coefficient Beta(β) across models aswell as the covariance of the error terms are, where β is the vector of the coefficients optimal parameter for a given model and then analyze the residuals of the error terms of each model to find the model that has the distribution that is the most fitted around 0.

Another important objective is to review the β-coefficients in the different models, which are coefficients along the regression line,as well as reviewing the statistical interpretation of our output data, including the normally distributed noise and to what degree they clutter or skew our results.

2 Methods

2.1 Mathematical Model Assumptions:

The first method I use in regression is the OLS method (Ordinary Least Squares). This is a linear regression model, a statistical technique used for modeling the relationship between a dependent variable and independent variables by fitting a linear equation to the observed data. The model consists of Y, a vector of observed values for a dependent variable, X, a design matrix with columns where each column represent an independent variable, £, a vector of error terms (residuals), meaning the difference between the observed and predicted values, and β, a vector of coefficients to estimate, where the length of β will be equal to the number of terms in the model. The model is expressed as:

$$Y = X\beta$$

The objective is to minimized the residual sum of squares:

$$^T() = (Y - X\beta)^T(Y - X\beta)$$

and, estimate the coefficients β that minimize the cost function.

$$Cost(\beta) = \sum_{i=1}^{n}(y_i - x_i\beta)^2$$

. The optimization process aims to find the coefficients that results in the best linear fit to the data. This is achieved by minimizing the mean squared error (MSE).

The optimal estimator for β is:

$$\beta = (X^T(X))^-1(X^T(Y))$$

The OLS uses input vectors, consisting of a vector of observed independent variable values, and X, our design matrix of independent variables. These variables are used to predict variations in another variable, the dependent variable. They are also called features, and can be continuous such as in age

or temperature, or categorical like in gender or region. The output vector, also called the target or response variable is what I am trying to explain or predict using the input vector.

Each element of β is the estimated effect of the corresponding independent variable on the dependent variable, holding all other variables constant. OLS assumes that the errors £ are normally distributed with a mean of 0 and independent. It also assumes the independent and dependent variables are linear. Because the independent variables have different scales, it affects the convergence of the optimization algorithm and makes it difficult to interpret the magnitude of coefficients. In this project, therefore the data has been scaled so all variables have the same scale. This doesn't change the relationship between variables but improves the stability, convergence and numerical stability of the OLS estimator. Also, the data has been split into "training" and "test" data, where the training data represents 80This is because I want to assess the performance of OLS on unseen data, avoid overfitting, where the model becomes too specialized on the training data, and doesn't generalize well to new, unseen data. The splitting of the data is what allows for calculation of the MSE and R2 on the testing set, which is done because the test data simulates the scenario where the model encounters new unseen data. If the model performs very good on the training data but poorly on the test data, it's a sign the model has overfit the training data, and it provides an estimate of real-world performance. The test data then ensures the model's performance isn't inflated due to memorizing the training data. Splitting the data also allows for cross-validation, which I will describe later.

The OLS Method in this Project was implemented using a Franke function consisting of 4 terms and the code for generating the function and then splitting and scaling the data is as follows:

```
#Two-polynomial model with the given Franke function, OLS Method:

#Comment: Here, it is used the brute force Matrix Inversion theta = np.linalg.inv(X.T
                                        @ X) @ X.T @ y,
#rather than the SVD. The matrix inversion is applied to the design matrix X, where (X
                                        .T @ X)
#calculates the matrix product of the transpose and X. The product is called the inner
                                         product matrix.
# Then, it calculates the inverse of the inner product matrix, and then calculates the
                                         matrix product
# of the transpose of X. The overall point is to find the coefficients
#that best matches the data, in this case using the OLS method.
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define the Franke function
def FrankeFunction(x, y):
    term1 = 0.75 * np.exp(-(0.25 * (9 * x - 2)**2) - 0.25 * ((9 * y - 2)**2))
    term2 = 0.75 * np.exp(-((9 * x + 1)**2) / 49.0 - 0.1 * (9 * y + 1))
    term3 = 0.5 * np.exp(-(9 * x - 7)**2 / 4.0 - 0.25 * ((9 * y - 3)**2))
    term4 = -0.2 * np.exp(-(9 * x - 4)**2 - (9 * y - 7)**2)
    return term1 + term2 + term3 + term4

# Generate data using the Franke function
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
x, y = np.meshgrid(x, y)
z = FrankeFunction(x, y)

# Flatten the 2D arrays for data splitting
x_flat = x.ravel()
y_flat = y.ravel()
z_flat = z.ravel()

# Split the data into training and test sets (80% training, 20% testing)
x_train, x_test, y_train, y_test, z_train, z_test = train_test_split(x_flat, y_flat,
                                        z_flat, test_size=0.2, random_state=42)

# Scale the data using StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train.reshape(-1, 1))
x_test_scaled = scaler.transform(x_test.reshape(-1, 1))
y_train_scaled = scaler.fit_transform(y_train.reshape(-1, 1))
```

```
y_test_scaled = scaler.transform(y_test.reshape(-1, 1))
z_train_scaled = scaler.fit_transform(z_train.reshape(-1, 1))
z_test_scaled = scaler.transform(z_test.reshape(-1, 1))
```

Here, the data is split to ensure that the model generalizes well to new, unseen data. The data is scaled using "StandardScaler". Scaling is done to ensure that all input features are on a similar scale. This can help improve the convergence and performance and ensures that no single feature dominates the learning process due to its larger magnitude, which can lead to slower convergence or biased results. The choice to split the data into training and test sets with an 80-20 ratio and to scale the data using StandardScaler in this particular case was made to ensure that the polynomial regression model is trained effectively, generalizes well to unseen data, and is numerically stable during optimization. These choices are common best practices in machine learning when working with regression models and can lead to more robust and interpretable results.

In using the OLS method, I have made an assumption that there exists a continuous function f(x) and a normal distributed error £ N(0,σ^2), which describes our data y=f(x)+£. We then approximate this function f(x) with our model from the solution of the linear regression equations, this is what we call the OLS. That is our function f is approximated by y where we minimized (y-y) squared, with y = X/β. I now want to show that the expectation value of y_i is E(y_i) = $X_i \cdot |\mathbf{B}$ and that its variance is Var(y_i) = σ^2, I start with the linear regression model:

$$y = X\mathbf{B} + \epsilon$$

where:

- y is the vector of observed values.
- X is the design matrix.
- $\mathbf{B}$ is the vector of regression coefficients we want to estimate.
- ϵ is the error term with mean zero and variance σ^2.

In ordinary least squares (OLS) regression, I minimize the sum of squared errors, which is equivalent to minimizing the following expression:

$$\min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where y_i are the observed values, $\hat{y}_i$ are the predicted values, and n is the number of observations. The predicted values $\hat{y}_i$ are given by $\hat{y}_i = X_i \cdot \mathbf{B}$, where X_i is the i-th row of the design matrix X. The expectation E(y_i):

$$\mathrm{E}(y_i) = \mathrm{E}(X_i \cdot \mathbf{B} + \epsilon_i) = X_i \cdot \mathbf{B} + \mathrm{E}(\epsilon_i) \quad \text{since } \mathrm{E}(\epsilon_i) = 0 \text{ (mean of the error)} = X_i \cdot \mathbf{B}.$$

So E(y_i) = $X_i \cdot \mathbf{B}$.
The variance is given as: Var(y_i):

$$\mathrm{Var}(y_i) = \mathrm{Var}(X_i \cdot \mathbf{B} + \epsilon_i) = \mathrm{Var}(\epsilon_i) \quad \text{since } \mathrm{Var}(X_i \cdot \mathbf{B}) = 0 \text{ (constant)} = \sigma^2.$$

The objective function to estimate β is given by:

$$\hat{\beta} = \operatorname{argmin} \sum_{i=1}^{n} (y_i - X_i^T \beta)^2,$$

where y_i is the observed response variable, X_i is the predictor variable for the ith observation, and β is the parameter vector to be estimated.

The OLS estimator $\hat{\beta}$ can be expressed as:

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

The expected value of $\hat{\beta}$:

$$E(\hat{\beta}) = E\left((X^T X)^{-1}(X^T y)\right) = (X^T X)^{-1}(X^T E(y)) = (X^T X)^{-1}(X^T (X^T \beta))$$

$$= (X^T X)^{-1}((X^T X)^T \beta)).$$

Since $X^T X$ is a constant matrix, $(X^T X)^{-1}(X^T X)$ is the pseudoinverse of $X^T X$. However, $(X^T X)^{-1}(X^T X)$ is an identity matrix, which means that the mathematical expression simplifies to:

$$E(\hat{\beta}) = \beta.$$

The above equations are very important in the context of this project because these are what allow us to estimate the model parameters and minimize the residual sum of squares. This is the fundamental principle of linear regression, namely to find the values of beta that minimizes the residual sum of squared differences between the observed responses yi and the predictions Xi^T/β. This is what is used to fit the linear regression model to the data. Additionally, the equations provide an expectation of the estimator, which helps assess the properties of the estimator with the goal of it being unbiased. The final result above shows that under certain assumptions, the OLS estimator is a linear and unbiased estimator of the true parameter vector. They also indirectly highlight some of the assumptions underlying linear regression models, such as the assumption of normally distributed errors and no multicollinearity.

In Ordinary Least Squares (OLS), the estimated parameter vector $\hat{\beta}$ is given by:

$$\hat{\beta} = (X^T X)^{-1}(X^T y),$$

The variance-covariance matrix of $\hat{\beta}$, denoted as $\text{Var}(\hat{\beta})$, is defined as:

$$\text{Var}(\hat{\beta}) = \sigma^2 (X^T X)^{-1},$$

where σ^2 is the variance of y.

Now, let's derive $\text{Var}(\hat{\beta})$ step by step:

$$\text{Var}(\hat{\beta}) = (X^T X)^{-1}(X^T \text{Var}(y))((X^T X)^{-1})^T$$

$$= (X^T X)^{-1}(X^T \text{Var}(y))(X^T X)^{-1} = \sigma^2 (X^T X)^{-1}(X^T X)(X^T X)^{-1}.$$

Since $(X^T X)$ is symmetric, we have $(X^T X)^T = X^T X$. Therefore, we can simplify the expression further:

$$\text{Var}(\hat{\beta}) = \sigma^2 (X^T X)^{-1}(X^T X)(X^T X)^{-1} = \sigma^2 (X^T X)^{-1}.$$

So, the variance of the estimated optimal parameters $\hat{\beta}$ in OLS is $\sigma^2 (X^T X)^{-1}$.

The primary goal is finding the optimal parameters β by optimizing the mean squared error of the cost function. By testing the different Linear Regression methods and determining which one is the best fit for our particular problem, I aim for as my final result to find the optimal Bias-Variance tradeoff. By doing this I will find a model that responds as well as possible to the new, unseen data.

The Bias is the error due to simplistic assumptions in the model. High bias typically lead to overpredicting or underpredicting the data. The Variance is the error due to excessive complexity in the model. High variance can mean the model fitting the noise in the data. Below I will explain in detail what the mathematical expression for the Bias-Variance tradeoff consists of and how it is interpreted.

$E[(y - \hat{y})^2] = \text{Bias}[\hat{y}] + \text{Var}[\hat{y}] + \sigma^2$, where $\text{Bias}[\hat{y}] = E[(y - E[\hat{y}])^2]$ and $\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$

The true data generation process: $y = f(x) + \epsilon$, where ϵ is normally distributed with mean 0 and variance σ^2.

$\hat{y} = X\hat{B}$, where $\hat{B}$ are the parameters to estimate.

The cost function: $C(X, \hat{B}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 = E[(y - \hat{y})^2]$.

The expected value of the squared difference between y and $\hat{y}$ can be broken down to bias, variance, and noise terms:

$$E[(y - \hat{y})^2] = E[(y - E[\hat{y}] + E[\hat{y}] - \hat{y})^2] = E[(y - E[\hat{y}])^2 + (E[\hat{y}] - \hat{y})^2 + 2(y - E[\hat{y}])(E[\hat{y}] - \hat{y})]$$

$$= E[(y - E[\hat{y}])^2] + E[(E[\hat{y}] - \hat{y})^2] + 2E[(y - E[\hat{y}])(E[\hat{y}] - \hat{y})].$$

1. **Bias Term**:

$$\text{Bias}[\hat{y}] = E[(y - E[\hat{y}])^2]$$

Meaning:

$$E[\hat{y}]$$

is the expected value of the predicted values

$$\hat{y}$$

from the linear regression model.

$$y$$

is the response variable in the dataset. These are the actual observed values one is trying to predict using the linear regression model.

$$y - (E[\hat{y}]$$

is the difference between the true values y and the expected predicted values for each data point, and shows how much the actual data deviates from the average prediction made by the model for each data point. Now

$$y - (E[\hat{y})^2]$$

squaring this difference ensures all deviations are treated as positive values, penalizes larger deviations more heavily than smaller ones, and emphasizes the impact of outliers or extreme data points on the overall bias.

$$E[(E[\hat{y}] - \hat{y})^2]$$

is the expected value of the squared differences between the true values y and the expected predicted values for all data points.

This term measures the squared difference between the expected value of the predictions

$$([\hat{y}])$$

and the true data (y). It show how much the model underpredicts or overpredicts the true data across different samples. A high bias indicates that the model is consistently inaccurate.

2. **Variance Term**:

$$\text{Var}[\hat{y}] = E[(E[\hat{y}] - \hat{y})^2]$$

$$\hat{y}$$

is, unlike y mentioned in the Bias term, the actual predicted values from the linear regression model. The values that the model predicts for each data point.

$$(E[\hat{y}] - \hat{y}$$

is the difference between the average predicted values values and the expected predicted values for each data point, and shows how much each individual prediction deviates from the average prediction made by the model.

Squaring this difference ensures all deviations are treated as positive values, penalizes larger deviations more heavily than smaller ones, and emphasizes the impact of outliers or extreme data points on the overall bias.

This term is the expected value of the squared differences between the expected predicted values and the actual predicted values for all data points. measures the variability or spread of the model's predictions around their expected value. It quantifies how sensitive the model is to variations in the training data. A high variance means the data is highly sensitive to changes in the training data, possible due to overfitting, and the model fits the noise in the data instead of the patterns.

3. **Variance of the Noise Term**:

$2E[(y - E[\hat{y}])(E[\hat{y}] - \hat{y})] = 2E[(y - E[\hat{y}])E[\hat{y}] - (y - E[\hat{y}])\hat{y}] = 2(E[(y - E[\hat{y}])E[\hat{y}]] - E[(y - E[\hat{y}])\hat{y}])$.

The term

$$\sigma^2$$

is the irreducible error, and is the minimum error that any model will have due to inherent randomness in the data. It is the variance of the noise (£) in the data generation process.

Var[y] measures the average squared deviation between individual predictions and the expected absolute prediction. This quantifies the variability or "variance" of the error or noise in the models predictions. The expression as a whole measures how much the noise in the predictions deviates from the average absolute prediction.

In the expression again:

$$\begin{aligned} E[(y - \hat{y})^2] &= \text{Bias}[\hat{y}] + \text{Var}[\hat{y}] + 2(E[(y - E[\hat{y}])E[\hat{y}]] - E[(y - E[\hat{y}])\hat{y}]) \\ &= \text{Bias}[\hat{y}] + \text{Var}[\hat{y}] + 2(E[(y - E[\hat{y}])E[\hat{y}]] - E[(y - E[\hat{y}])\hat{y}]). \end{aligned}$$

Examining the term $2(E[(y - E[\hat{y}])E[\hat{y}]] - E[(y - E[\hat{y}])\hat{y}])$:

$$\begin{aligned} 2(E[(y - E[\hat{y}])E[\hat{y}]] - E[(y - E[\hat{y}])\hat{y}]) &= 2(E[E[yE[\hat{y}] - E[\hat{y}]^2]] - E[E[y\hat{y}] - E[\hat{y}]y]) \\ &= 2(E[E[y]E[\hat{y}] - E[\hat{y}]^2] - E[E[y\hat{y}] \\ -E[\hat{y}]E[y]]) &= 2(E[E[y]E[\hat{y}] - E[\hat{y}]^2 - E[y\hat{y}] + E[\hat{y}]E[y]]) = 2(E[E[y]E[\hat{y}] - E[y\hat{y}]]). \end{aligned}$$

Looking at $E[E[y]E[\hat{y}] - E[y\hat{y}]]$:

$$\begin{aligned} E[E[y]E[\hat{y}] - E[y\hat{y}]] &= E[E[y]E[\hat{y}]] - E[E[y\hat{y}]] = E[E[y]]E[E[\hat{y}]] - E[E[y\hat{y}]] \\ &= E[y]E[\hat{y}] - E[y\hat{y}] = E[\hat{y}]E[y] - E[y\hat{y}] = E[\hat{y}]E[y - E[y]] = E[\hat{y}] \cdot 0 = 0. \end{aligned}$$

$$E[(y - \hat{y})^2] = \text{Bias}[\hat{y}] + \text{Var}[\hat{y}] + 0 = \text{Bias}[\hat{y}] + \text{Var}[\hat{y}] = \text{Bias}[\hat{y}] + Var[\hat{y}] + \sigma^2.$$

The other measurements used in the code are the MSE, R2, Beta-coefficients comparison and Error term residuals. I will explain these terms and how they are implemented in the code:

I aim to get an MSE score as close as possible to 0. The equation for MSE is $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, with n the number of data points, yi the actual values,

$$\hat{y}_i$$

the predicted values. It means that when the Mean Squared Error is close to 0, the mean difference between the observed and predicted values is close to 0, and indicates a good fit of the model to the data.

I aim to get the R2 coefficient as close as possible to 1. The equation for R2 is $1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, which measures the proportion of the variance in the dependent variable that is explained by the independent variables, and it is a value between 0 and 1. The expression in the numerator is the sum of squared residuals, the SSE, so it is different from the MSE in that it is not a mean, but a sum. The variables have the same meaning as in the MSE explanation. The SSE accounts for the unexplained variance in the dependent variable. The expression in the denominator is the SST, the total sum of squares, the total variance in the dependent variable. I want the R2 coefficient as close as possible to 1, which indicates that a larger proportion of the variance in the dependent variable is explained by the model. The code for MSE and R2 is provided and expansively commented in the .ipynb file. The Beta-coefficients are another measurement. If they are small and close to 0, they indicate a small change in the corresponding independent variable(X) has a minimal impact on the dependent variable. It indicates that a particular independent variable doesn't contribute much to explaining the variance in the dependent variable. If this is the case, the model will not respond well to new unseen data, so if I want to, I can remove it. However, I don't necessarily want them to be small and close to 0 in other types of mdoels, so that when the method is reproduced, my aim is to provide something that makes it easy to modify a parameter to change the size of the β-coefficients. The Ridge and Lasso methods contain the regularization parameter λ, serving this purpose.

For instance, the Ridge method, showing how the regularization parameter is easily modifiable in the code I have used looks like this(this can also be done in the Lasso code):

```
# Initialize lists to store MSE and R-squared values for different degrees and lambda
                                        values
degree_range = range(1, 6)
lambda_values = [0.0001, 0.001, 0.01, 0.1, 1.0]

beta_coefficients_list = []
```

after this the code has a specific way of defining the R2-code used in the Ridge method, that looks different from the Lasso and OLS. This is part of the code, but it is commented in such a way that it is easy to follow what has been done where in the .ipynb file. The point is that I can modify the lambda values from an easy vector configuration.

The formula for β-coefficients: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$

Where Y is the dependent variable, X1,X2,...,Xp are the independent variables, $\beta 0$ is the intercept meaning the value of Y when all X-variables are 0. $\beta 1$, $\beta 2$ etc. are the beta-coefficients associated w. the independent variables. These coefficients represent the change in Y for a 1-unit change in the corresponding X, when all other X stay constant. £ is the error of the residual, the difference between the predicted and actual value of Y.

In the Franke function study I aim to get Error term residuals that are closely aligned (not too much variance) and close to 0. I assume it will be difficult to achieve this with the real topographic data. I assume the error are normally distributed with mean around 0 and constant variance. So when when the residuals are around 0, it suggests that the assumptions are met, important for the validity of statistical inference and hypothesis testing. When the error term is mean-centered, it ensures the model's parameter estimates are unbiased and consistent, so they converge to the true population values. Each coefficient means the average change in the dependent variable for a 1-unit change in the corresponding independent variable, and this interpretation is less meaningful if the residuals are biased away from 0. It also indicates that the relationship between the predictors and the response variable is linear.

2.2 Ridge and Lasso Regression

I have then used the Ridge regression method which in mathematical terms has an expression equal to the OLS method, but it adds a regularization term to the cost function, meaning its a new cost function to be optimized. This is in place to shrink the regression coefficients to prevent overfitting, and they minimize a penalized residual sum of squares. The cost function is:

$$Cost(\beta) = \sum (y_i - x_i\beta)^2 + \lambda \sum (\beta_j)^2$$

Ridge regression computes the coordinates of y with respect to the orthonormal basis U. It then shrinks these coordinates by the factors $((dj)^2)/((dj)^2 + \lambda)$. A greater amount of shrinkage is applied to the coordinates of basis vectors with smaller $(dj)^2$.

To estimate the coefficients β, the cost function is minimized by differentiating with respect to β and setting its derivative = 0. The Ridge regression estimator is given by

$$\beta^(Ridge) = ((X^T(X) + \lambda I)^{-1})(X^T(Y))$$

, where I is the identity matrix, λ is the regularization parameter, and the addition of these two prevents overfitting by shrinking the coefficients towards 0. It sets a penalty on the size of the regression coefficients. When there are many correlated variables in a linear regression model and their coefficients can become poorly determined and show high variance. A very large positive coefficient on one variable can be canceled by a similarly large negative coefficient on its correlated cousin. The problem is solved by imposing the size constraint on the coefficients. The solution adds a positive constant to the diagonal of XT X before inversion. This makes the problem nonsingular, even if XT^X is not of full rank. In the case of orthonormal inputs, the ridge estimates are a scaled version of the OLS estimates, that is, $\beta(\hat{ridge}) = \beta/(1 + \lambda)$, and it can also be derived as the mean or mode of a posterior distribution. However, we don't have orthonormal inputs in this case. The Ridge can be analysed by the SVD of the N × p matrix X, that has the form $X = UDV^sT$, but in this case I used the brute force matrix inversion method like in the OLS case, described in the OLS code earlier in this subsection. I have used the same python code as in OLS to generate data and scale and split, taking place before Ridge-specific code creating all the plots for measurements.

The Ridge regression sacrifices some bias to reduce variance.

The third method I used is Lasso regression. It uses a different cost function than Ridge and OLS:

$$\text{Cost}(\beta) = \sum (Y_i - X_i\beta)^2 + \lambda \sum |\beta_j|$$

The second term is referred to as the L1 regularization term. This is equal to the sum of absolute values of the coefficients multiplied by a regularization parameter λ, which in this case takes on different values. The L2 regularization term used in Ridge, in constrast does not use the absolute value, and also it is squared, meaning this term in Ridge can only take on positive values. So here, it can take on negative values also. L1 regularization has the effect of sparsity-inducing feature selection. It tends to force some coefficients to be exactly zero, effectively eliminating certain independent variables from the model. Lasso is particularly useful when suspecting that only a subset of independent variables is truly important for predicting the dependent variable. It automatically selects relevant features. The choice of the regularization parameter in Lasso controls the degree of sparsity in the model. Larger values lead to sparser models with fewer non-zero coefficients.

The optimal parameter for β in Lasso is written as: $\hat{\beta}^{\text{Lasso}} = \arg\min_{\beta} \left\{ \sum (Y_i - X_i\beta)^2 + \lambda \sum |\beta_j| \right\}$

It represents the estimated coefficients that minimize the cost function with L1 regularization. It strikes a balance between fitting the data and preventing overfitting. It reduces the model complexity by shrinking some parameters to 0. The optimal parameter includes λ, the regularization term, such that by adjusting the value of this, I can control the amount of regularization applied.

A bit more on the mathematical difference and utility value between Ridge and Lasso is that in the general case, I can view them as Bayes estimates, and I can look at the optimal parameter for the Lasso:

$$\tilde{\beta} = \text{argmin}_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\} \tag{3.53}$$

for $q \geq 0$.

Thinking of $|\beta_j|^q$ as the log-prior density for β_j, these are also the equi-contours of the prior distribution of the parameters. The previously mentioned Ridge regression with L2-regularization corresponds to the value $q = 2$, the penalty term in the sum of squared coefficients, while $q = 1$ corresponds to the Lasso regression with L1-regularization, the absolute sum of the coefficients. For $q \leq 1$, the prior is not uniform in direction, but concentrates more mass in the coordinate directions. The prior corresponding to the $q = 1$ case is an independent double exponential distribution for each input, with density $(1/2\tau)\exp(-|\beta|/\tau)$ and $\tau = 1/\lambda$. So in general, when reproducing the code, changing the values of λ in the Ridge and Lasso code with the simple configuration mentioned earlier, we can make the relationship and results of $q = 1$ and $q = 2$ different, which can be applied to any suitable problem or analysis.

The case $q = 1$ (lasso) is the smallest q such that the constraint region is convex; non-convex constraint regions make the optimization problem more difficult.

So the lasso, ridge regression are Bayes estimates with different priors. They are derived as posterior modes, maximizers of the posterior. It is more common to use the mean of the posterior as the Bayes estimate. Ridge regression is also the posterior mean, but the lasso is not. By varying the values of $q = 1$, which would be the same as varying the values of λ, the regularization parameter in the code, I can encourage more aggressive feature selection by decreasing the value, potentially getting fewer non-0 coefficients. It can simplify the model and identify important features, and obviously has great utility value. By increasing the values of $q = 2$, I can control the magnitude of the coefficients more effectively but wouldn't perform feature selection as aggressively as Lasso. This is beneficial for controlling overfitting but doesn't provide the same utility value as the Lasso.

2.3 Resampling Methods

:

Before applying any resampling methods, I did an analysis of the bias-variance tradeoff in the OLS model. This had the purpose of assessing the performance of the model on the original dataset to understand how well the model captures the underlying relationships in the data. Also, it was done to identify underfitting, where the data is too simple to capture the model's complexity(high bias), or

overfitting, where the data is too complex and fits the noise in the data (high variance) in the OLS. It could also give me some pointers to select the model complexity, and assess how robust the model is to variations in the data. What I am looking for is a model with low bias and moderate variance, because it will generalize well to new data.

I have used two resampling methods after running the initial analysis, which repeatedly draw samples from the training set refit a model on each sample to get additional information about the fitted model. This is a good approach since the simulations can be treated as computer experiments, the results can be analyzed with the same statistical tools used when analyzing experimental data,

The first one is bootstrapping: The bootstrap method does not rely on any specific mathematical expression like the aforementioned, the bootstrap method is used to estimate the sampling distribution of a statistic or parameter. The statistic of interest is typically denoted as a function of the observed data, such as: Statistic of Interest: $\theta(X)$, where X represents the observed data. $\theta(X)$ in this case corresponds to the performance metrics (mean squared error, R-squared) for polynomial regression models of different degrees. Some basic mathematical assumptions about what I do are: The probability of obtaining an average value z is the product of the probabilities of obtaining arbitrary individual mean values xi, with the constraint that the average is z. I can express this through the following expression: $p\ (z) = \int [dx_1 \cdot p(x_1)] \int [dx_2 \cdot p(x_2)] \ldots \int [dx_n \cdot p(x_n)] \cdot \delta\left(z - \frac{x_1+x_2+\ldots+x_n}{n}\right)$, also I have used the integral expression for the Dirac-function:

$$\delta(z - \frac{x_1 + x_2 + \ldots + x_m}{m}) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dq\, e^{iq(z - \frac{x_1+x_2+\ldots+x_m}{m}) + i\mu q - i\mu q} \tag{1}$$

this can be rewritten as:

$$\delta(z - \frac{x_1 + x_2 + \ldots + x_m}{m}) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dq\, e^{iq(z - \frac{x_1+x_2+\ldots+x_m}{m})} \cdot e^{i\mu q} \cdot e^{-i\mu q} \tag{2}$$

The integral representation of the Dirac delta function:

$$p^(z) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dq\, e^{iq(z - \frac{x_1+x_2+\ldots+x_m}{m})} \cdot \left[\int_{-\infty}^{\infty} dx\, p(x) \cdot e^{iq(\mu - x)/m}\right]^m \tag{3}$$

The above is important because it provides an interpretation of what the bootstrapping does. It is a mathematical representation of the relationship between the resampled dataset and the original dataset. It quantifies how well a statistic calculated from the resampled data matches a specific value observed in the original data. The bootstrapping draws multiple random samples from the original dataset. These resampled datasets are used to estimate the sampling distribution of a statistic of interest, in this case the bias and variance. The Dirac delta-function quantifies how likely it is to obtain a specific value from the resampled data. The integral over all possible values of z represents the aggregation of all possible values that the statistic of interest could take in resampled datasets. The result of this integral operation represents the pdf or density of obtaining the statistic from the resampled datasets. It shows then how likely different values are when calculated from resampled data.

Since $\hat{\beta} = \hat{\beta}$(X) is a function of random variables, $\hat{\beta}$ itself is a random variable. It has a pdf, and the aim of bootstrapping is to estimate $p(t)$ by the relative frequency of $\hat{\beta}$. If the relative frequency closely resembles p(t), I can estimate all the interesting parameters of p(t) using point estimators. If $\hat{\beta}$ has more than 1 component which are independent, I use the same estimator on each component separately. If I knew the pdf of Xi, p(x) I could do this by drawing lots of numbers from p(x), using these numbers compute a replica of $\hat{\beta}$ called $\hat{\beta} *$. If I draw observations in accordance with the relative frequency of the observations, I obtain the same result in some asymptotic sense. The steps for bootstrap in general and how it has been used in this code are: 1. Draw a replacement n numbers for the observed values x=(x1,x2,...,xn). I have done this step in the code by generating synthetic data points x and y using the Franke function, which serves as the "observed values" in this context. 2. Define a vector x* containing the values drawn from x. The code uses the generated x and y data points directly. 3. Using the vector x* to compute $\hat{\beta}*$ by evaluating $\hat{\beta}$ under the observations $x*$. The code fits a polynomial regression model (OLS) to the data, where $x*$ corresponds to the x_train_scaled values. It creates polynomial features based on the chosen degree and then fits a linear regression model (model) to predict z (the Franke function values) using the scaled x_train data. 4. Repeat this process k times. The code fits a polynomial regression model (OLS) to the data, where x* corresponds

to the x_train_scaled values. It creates polynomial features based on the chosen degree and then fits a linear regression model (model) to predict z (the Franke function values) using the scaled x_train data. It is entirely possible that different reruns of the Bootstrap method yield different results. I have done several reruns of the Bootstrap method on the OLS code, and the results are described in the appendix "Re-runs of bootstrap plus more lambda-configurations and polynomial fit results.ipynb".

Cross-validation: The second resampling method I applied is the Cross-validation method. It requires partitioning the data into distinct training and test sets, ensuring that each data point is used exactly once for testing (in K-fold cross-validation). Cross-validation doesn't involve random sampling with replacement, as bootstrapping does.

Cross-validation uses the Leave-One-Out-Cross-Validation(LOOCV), a practical implementation of the Dirac-delta from Bootstrapping. I am evaluating the model's performance at each individual data point. LOOCV evaluates the model's performance in this way, similar to how the Dirac delta represents a spike at a single point. The LOOCV focuses the evaluation on a single data point. In LOOCV, I consider all iterations like the Dirac uses cumulative effects. The LOOCV is a limited case of k-fold cross-validation as the number of folds approaches the no. of data points. I can look at the first part of the Dirac delta-function $\frac{1}{2\pi}\int_{-\infty}^{\infty} dq\, e^{iq(z-\frac{x_1+x_2+\ldots+x_m}{m})}$ as an integral over the whole dataset, which would be divided into k subsets in the cross-validation.

When the repetitive splitting of the data set is done randomly, samples may accidently end up in a fast majority of the splits in either training or test set. Such samples may have an unbalanced influence on either model building or prediction evaluation. To avoid this k-fold cross-validation structures the data splitting. The samples are divided into k more or less equally sized exhaustive and mutually exclusive subsets. In turn (at each split) one of these subsets plays the role of the test set while the union of the remaining subsets constitutes the training set. Such a splitting warrants a balanced representation of each sample in both training and test set over the splits. Still the division into the k subsets involves a degree of randomness. This may be fully excluded when choosing.

The cross-validation happens as the datasets of k (the no. of partitions the dataset is split into) are shuffled randomly, the data is split into k groups, the model is fit on a training set and evaluated on the test set, where I have decided how much data belongs to test and training, then I retain the evaluation score and discard the model.

The K-fold cross-validation:

$$\text{CV} = \frac{1}{K}\sum_{i=1}^{K}\text{Metric}_i$$

Where:

$$\text{CV} : \text{Cross-validation metric (e.g., mean squared error, accuracy).}$$

$$K : \text{Number of folds (user-defined parameter).}$$

Metric_i : The performance metric computed for the i-th fold, based on model predictions on test for that fold.

It partitions the data into K equal-sized and non-overlapping subsets, for each of the K iterations it selects one of the K folds as the test set, and the remaining K-1 as training set, it trains the predictive model on the training set, evaluates the model's performance on the test set, and computes a performance metric for this iteration. Then it calculates a summary metric for model performance by averaging the K individual performance metrics, in the case of linear regression, the MSE.

When applying the Cross-validation to Ridge and Lasso, I fit the linear regression model for each λ in the grid using the training set, and the corresponding estimate of the error variance:

$$\sigma - i\lambda as\beta(-i)\lambda = (X^T(-i)*, X(-i)* + \lambda * Ipp)^(- 1)((X^T(-i), *Y - i)$$

2.4 Modification Parameters and Reproducibility

To test the influence of the different models, If I want the difference between the *specific models*(OLS, Ridge and Lasso) to be smaller or larger, I can change the values of the parameter λ in the Ridge and Lasso regression, where a larger λ will make the Ridge and Lasso models more different from the OLS. In the code, this is easily modifiable by a parameter vector called $lambda_values$ in the Ridge and Lasso parts of the code. The model is most reproducible when these values are fixed across runs w. different data sets.

Another way to ensure reproduciblity is to make sure the split between training and test is fixed across runs w. different inputs. For instance, I can always go with 80 percent training data and 20 percent test data. This is modifiable in the code.

If I want to increase or decrease model complexity ***Change the degree range:*** Changing the values for the variable "$degree_range$" used in the code allows me to fit regression models of varying degrees to the same data, so I can investigate how model complexity affects the results without modifying the underlying data generation process.

If I want to change my data, to check a different relationship between variables than what I am currently checking: In the code I have used for all three methods, the input data is not given directly but it is preprocessed and generated from the ***Franke function***, which is modifiable and configured in the beginning of the code. The concrete input data is fed to the model in the following way:

x_train_scaled and x_test_scaled, y_train_scaled and y_test_scaled, z_train_scaled and z_test_scaled: These are the scaled values of the x-coordinates of the data points.

The design matrix $X_train_x, X_test_x, X_train_y$, and so on, are created by stacking polynomial features of the scaled x and y coordinates from the Franke Function. These design matrices are used as input to the linear regression models.

To change the input I can change the Franke Function: I can modify the underlying data distribution (x,y,z), by modifying the Franke function. If I don't want to use a Franke function, I can change the way the input is given to the model by removing the Franke function and adding the data another way, for instance by adding a line to read a data file, create definitions for the flattened and reshaped data of that file, then add a definition of the polynomial, and add a model fit in the code to interpret the data file as input. An example of an exact way the Franke function is replaced by file data is given in the code.

The design matrix is constructed from the (x,y,z) coordinates with polynomial features, e.g., x, x^2, x^3, y, y^2, y^3, etc. When I generate (x,y,z) data points based on this Franke function, the values of $x, x^2, x^3, and x^4$ etc. are calculated from this relationship using the original coefficients. X will be the first column in the design matrix, x^2 the second, etc. When constructing the design matrix, the columns corresponding to $x, x^2, x^3, and x^4$ are created based on the calculated values using the original coefficients. Increasing or decreasing a coefficient will affect the magnitude of the corresponding term's influence on the relationship. Setting a coefficient to zero effectively removes the contribution of that term from the relationship. Let's say in a medical context x, x^2, x^3, x^4 represents no. of illnesses, symptoms, vaccines and spreads and I have a prior distribution I use to determine the coefficients in the Franke function, and I then recieve results (a posterior) telling me symptoms are fewer but the spread is higher requiring me to alter the assumptions about the symptoms and spreads ($x^2 and x^4$), I can change the coefficients of $x^2 and x^4$ in the Franke function to alter how much importance each term has. This will then change the weight of some of the columns in the design matrix, and the results will be altered accordingly. In summary, I have applied to the Franke function the:

1. OLS Method
2. Ridge Regression
3. Lasso Regression
4. Bootstrapping
5. Cross-Validation

in 5 different code sections. 6. I have applied all the above methods to a parametrized dataset, topographical terrain data from Tel Aviv and Haifa, Israel, downloaded from https://earthexplorer.usgs.gov/.

3 Results

3.1 Franke Function Results

3.1.1 MSE Before Resampling

A comparison of the MSE-score for the Franke function using OLS, Ridge and Lasso with regularization parameter λ = [0.0001, 0.001, 0.01, 0.1, 1]:

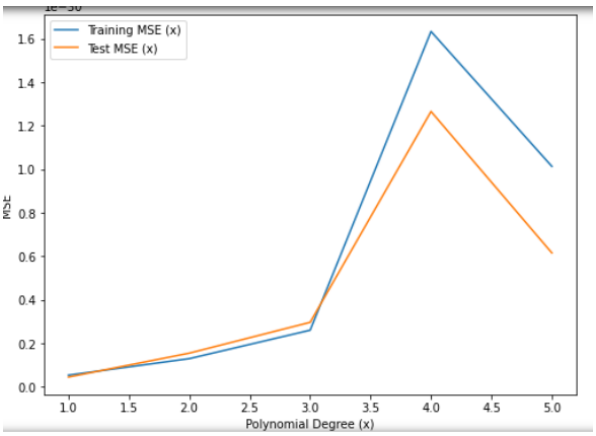


Figure 1: **MSE OLS**

1

OLS MSE for Training and Test data is given in Figure 1: What am I looking for? The code creates a loop that iterates over different polynomial degrees and fits the polynomial model for each degree. Additionally, I am storing the MSE and R-squared values for both the training and test sets for each polynomial degree. The Franke Function is a non-linear, multi-modal function with peaks, valleys, and varying degree of curvature. The Franke Function is expected to perform worse especially for higher-degree polynomials than the single-variable polynomial, and overfitting might occur as the model tries to capture the noise in the data. This is also what happens. OLS performs very well for degree 1, 2 and 3, but the MSE scores for degree 4 and 5 are very bad. For Ridge regression I expect training MSE to increase as lambda increases. Larger lambda creates stronger regularization, preventing the model from fitting the data too closely. The model becomes less likely to overfit, resulting in a higher training MSE. Training R2 to decrease as lambda increases. A decrease indicates that the model is becoming less able to explain the variance in the training data, expected with stronger regularization. Expect test MSE to initially decrease as lambda increases. This is because mild regularization can improve generalization by reducing overfitting. However, if λ increases too much, test MSE may start to increase again because the model becomes too biased. Results: Test and training MSE differ at degree 5 for all values of lambda, it is close to 0 for degrees 1 and 2 across all values of lambda. Test and training MSE is close to 0.5 for lambda = 0.001 and lambda = 0.01 for the third and fourth degree polynomial. It is between 0.5 and 1.0 for the lambda = 0.0001 and 0.1, so somewhat higher. Test MSE is close to 2 for the 5th degree polynomial for all values of lambda except lambda=1, so the Ridge regression gives a worse MSE than the OLS method here. Training MSE is between 1.5 and 2 for the

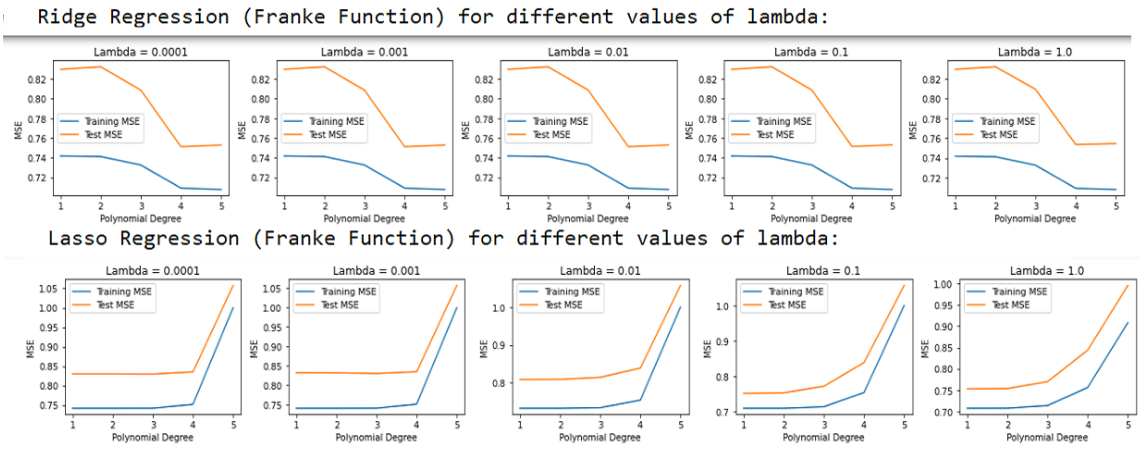


Figure 2: **MSE Ridge(top) and MSE Lasso(bottom) for different values of λ**

2

5th degree polynomial for all values of lambda except lambda=1, also here the Ridge regression gives a worse MSE than the OLS method. The test and training MSE is extremely small and close to 0 for all polynomial degrees when lambda=1, so this specific value of the parameter lambda gives the best MSE score we have gotten thus far, across polynomial degrees. This shows that the MSE doesn't strictly increase or decrease as the lambda increases, but it actually diverges. However, the MSE in all cases increase as the polynomial degree increases. At the low degree polynomials, the MSE was low with the OLS method, and performed better than Ridge regression(given in the top half of Figure 2). (The MSE for different values of λ in Lasso is given in the bottom half of figure 2).

(For polynomial degrees 3,4 and 5). As the polynomial degree increased from 3 to 4 and 5, MSE increased, meaning the model responded worse to an increase in model complexity, as expected. (The difference/bias between predicted values and mean value grew). With the Ridge regression, the MSE decreased for degrees 4 and 5, across all values of the regularization term (The difference in MSE for smallest and largest lambda is given in Figure 3), meaning; in this case, the regularization term had no effect on the MSE. (The difference/bias between predicted values and mean value). With the

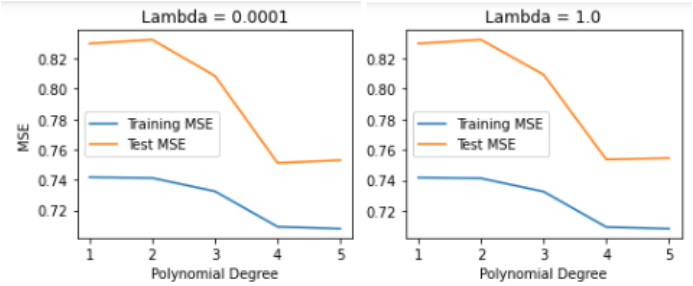


Figure 3: **MSE Ridge for λ=0.0001 and λ=1**

3

Ridge regression, the drop in MSE for larger polynomial degrees was more significant for the test than the training data. Because the test data contains fewer data points(predicted values), fewer data points responded less well to an increase in model complexity. For Degrees 4 and 5, the Ridge regression performed better than OLS. Another difference from the OLS method was that the test data performed better than the training data with OLS, but here training performed better than test, which is expected because of not enough data points for the OLS. For Lasso, as lambda values increase, MSE is expected to increase for both training and test sets because higher lambda-values encourage the model to simplify and reduce the magnitude of its coefficients. The model may become less flexible and less capable of fitting the training data well. As polynomial degrees grow larger, I expect MSE to decrease on the training set, as a higher-degree polynomial can fit the training data more closely. This may lead to overfitting, causing the test MSE to increase as the model fails to generalize to

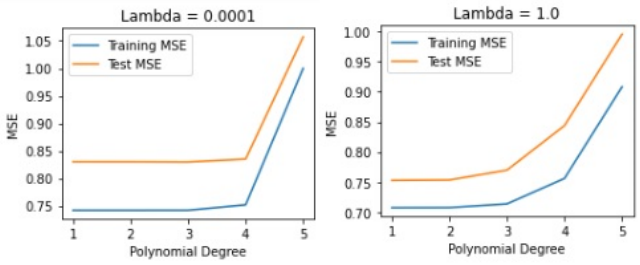


Figure 4: **MSE Lasso for λ=0.0001 and λ=1**

4

unseen data. The interaction between lambda and polynomial degrees: Larger polynomial degrees can make the model more complex, potentially requiring higher lambda-values to prevent overfitting. Unlike the previous 2 models, the Test and Training MSE differ significantly for all values of lambda and all polynomial degrees. The Test and Training MSE both decrease, like in the Ridge regression at polynomial degrees 4 and 5. The exception is for lambda = 0.1. This particular value of the

regularization term has a training and test MSE that doesn't decrease for the polynomial degrees 4 and 5. Instead it increases in the same way as for degrees 1, 2 and 3(See Figure 4). In Lasso, the degree 4 polynomial is the only one that has an increasing MSE across all degrees. OLS gave the best MSE for degrees 1,2, and 3. For degree 4, the Ridge regression gave a marginally better MSE than Lasso, and for degree 5 Ridge regression was clearly the best performer overall.

3.1.2 R2 Before Resampling

For the OLS Method the R-squared stays stable at 1, indicating overfitting, as this is the perfect score(Figure 5) For Ridge, the R2 score is very consistent across all lambdas(Top Half of figure 6). For

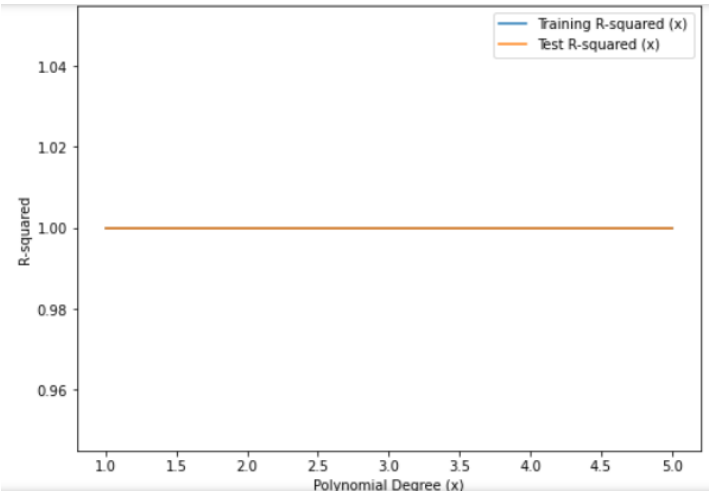


Figure 5: **R2 OLS**

5

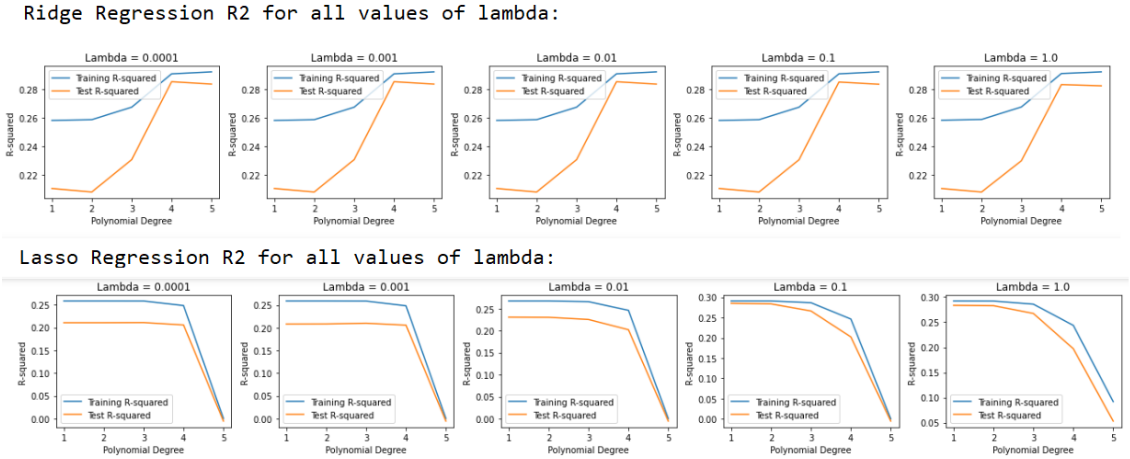


Figure 6: **R2 for Ridge and Lasso for all values of λ**

6

lambda=0.0001, the R2 is slightly decreasing from degree 4 to degree 5, and the test R2 is a tiny bit closer to the training R2 than in the lambda=1 case. For lambda=1 the test R2 decreases a tiny bit less than for lambda=0.0001 suggesting increased regularization strength stabilizes the R2 somewhat. The R2 score doesnt strictly increase or decrease as lambda gets larger, but it actually diverges. However, interestingly, it diverges for the same polynomial degrees as the Ridge MSE, and here the R2 slightly decreases from degree 1-2 and from degree 4-5 whereas it strictly increases from degree 2-4. Divergent R2 scores for different lambda values emphasizes the need to carefully select an appropriate lambda so that the correct amount of regularization is applied. Divergent R2 scores indicate that the model's

flexibility varies significantly with different lambda values. Some lambda values may produce models (polynomial fits, using Ridge regression line) that are too rigid and unable to capture the underlying patterns in the data (high bias), while others may produce models (polynomial fits) that fit the training data too closely and fail to generalize well to unseen data (high variance). The R2 scores gives a better result than the OLS, which just gave a score of 1. Thus the Ridge regression has handled the problem of overfitting.

With Lasso(Bottom Half of Figure 6), The Test and Training R2 are different from each other for all cases, where lambda=1.0 is the one containing the least difference. (See Figure 7 for a comparison of λ=0.0001 and λ=1) The difference between training and test is bigger for lambda=0.0001 than for

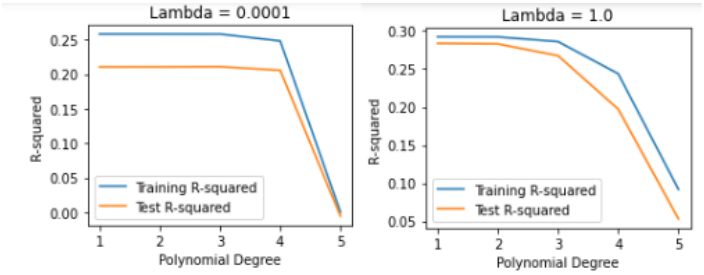


Figure 7: **R2 Lasso for λ=0.0001 and λ=1**

7

lambda=1, and the R2 for both test and training decreased as the polynomial degree increased. For degrees 1-3, the difference in R2-score was very small for all values of lambda, but for degrees 4 and 5, the R2 score rapidly decreased towards 0 and ended up at 0.05 for lambda=1. For lambda=0.0001 and lambda=0.001, the R2 score was consistent from degree 1-4, and the cutoff took place from degree 4 to 5.(Again compare the plots in Figure 6) With the other lambdas however the cutoff started from degree 3 to 4 and continued more steeply from degree 4 to 5. It is primarily the Lasso regression, not the strength of the regularization that drives the R2 towards 0 for higher-degree polynomials, but the regularization strength influences how early the cutoff takes place. For lambda=1, the R2 score was also higher at the lower degree polynomials than for lambda=0.0001, and as lambda increased, the R2 score for the lower degree polynomials (1-3) increased from 0.25 to 0.30(17 percent). From the cutoff starting point (Degree 3 or 4 depending on the value of lambda), the cutoff was between (80-100 percent). The Lasso regression for lambda=0.1 and lambda=1 gave marginally better R2 scores at the degrees 1-3 than all cases of Ridge, but overall the Ridge regression produced significantly better R2 scores. They were also more consistent. Both methods also produced better results than the OLS Method, which was highly likely to be overfitted.

3.1.3 Beta-coefficients

The β coefficients are given in the linear model for the dependent variable y, the variable I want to predict as:

yi = ¡yi¿ = β0xi0+β1xi,1+β2x1,2+...+βn-1xi,n-1+£i, where $< yi >$ is the mean value. To find the parameters βi the spread of the cost function w.r.t. β, $C(\beta)$ is minimized, so $min(\beta|R^h)1/n(y - x\beta)^T(y - x\beta)$ is solved. It means $(dC(\beta))/(d(\beta)^T) = 0 = (X^T(y - x\beta))$ is solved. It represents the derivative of the cost function of the linear regression problem, with respect to the coefficients β, and finding the values of β that minimizes this cost function. The second equation says that the difference between the observed target variables and the predicted values are orthogonal/perpendicular to the columns of the design matrix. It finds the values of β that minimize the sum of squared differences between the actual and predicted data points. My goal is to test OLS, Ridge and Lasso's ability to minimize the β-coefficients for this specific problem.

The Beta-coefficients in mathematical terms are:

$$\beta_p = \frac{(x - \bar{x}) \cdot (y - \bar{y})}{x - \bar{x}) \cdot (x - \bar{x})}$$

Where β is the beta coefficient for the predictor variable p, x is the predictor variable, $\bar{x}$ is the mean of the predictor variable x, y is the target variable and $\bar{y}$ is the mean of the target variable

y. This is called the covariance between the predictor variable and the target variable divided by the variance of the predictor variable. The aim is to find the values of β that minimize the sum of squared differences between the predicted and actual values of y. The estimation of the coefficients is the result of 1. regressing b on a to produce the residual z = x - $\bar{x}$1; 2. regressing y on the residual z to give the coefficient βa. Step 2 is a univariate regression, using the orthogonal predictors a and z. The jth multiple regression coefficient is the residual after regressing xj on x0,x1,...,xj-1,xj+1,...,xp. It represents the additional contribution of xj on y, after xj has been adjusted for x0,x1,...,xj-1,xj+1,...xp. If xp is highly correlated with some of the other xk's, the residual vector zp will be close to 0, and the corresponding β-coefficient will be very unstable and it'll be true for all the corresponding variables. In such situations, I may have all the Z-scores bec close to 0; I can delete one of them, but I can't delete them all. Z-scores for Ridge and Lasso further down in this subsection shows this is the case.

For OLS only β1 has a value of 1, and this happens across all the polynomial degrees.(Figure 8). For Ridge, the Beta-coefficients show the beta coefficient for the scaled z-values. β1 is extremely

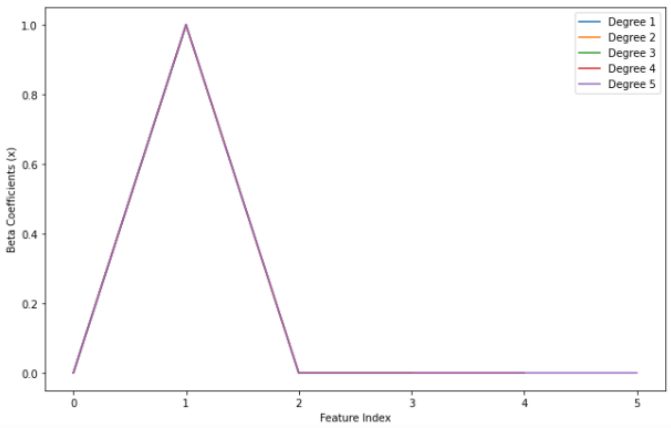


Figure 8: **Beta-Coefficients OLS**

8

close to 1 across all lambda-values. β0, β2 etc. are extremely close to 0 for all lambda-values. The polynomial perform largely within the same range of values but exhibit different behaviour. There is a difference in the Beta-coefficients as lambda increases, the higher the degree of the polynomial, the slightly closer to 0 the coefficients get as lambda increases. (Left Half of Figure 9) The higher the regularization term, the bigger is the difference between the polynomial degrees, but that difference remains very small. What does happen however is that the coefficient β4 is very close to 0 for the higher values of lambda, so the regularization in the Ridge regression renders this coefficient irrelevant. β5 is also close to irrelevant here, and it is the *closest* to 0 for λ=1. Also, the intercept stays consistent at -0.19 for all values of λ.

For Lasso, the Beta-coefficients explain something more complex: The Beta-coefficient graphs tell us how they change as the regularization strength is varied. There are 2 separate effects taking place here: 1. The regularization drives the coefficients away from 0. 2. The Lasso regression itself drives them back to 0, and the higher the degree of the polynomial, the stronger it drives them towards 0. In the case of the 5th degree polynomial, the regularization is completely negated by the Lasso regression' driving the coefficients to 0.

Here, the coefficient index 0 means β0, coefficient index 1 means β1, coefficient index 5 means β5, etc. So, with λ=0.0001, we are missing β3, β4 and β5 entirely (These are rendered irrelevant by the Lasso Regression). What's important to notice here is that the higher-degree polynomials have beta-coefficients further away from 0 when the regularization is very low. The higher the degree of the polynomial, the faster Lasso regression drives it to 0 as regularization strength increases. I also see that the regularization term strictly influences how fast the Lasso regression drives them towards 0. The larger the regularization term, the faster it drives them towards 0, so there is a positive relationship. As with Ridge, the regularization itself drives the coefficients towards from 0, but the Lasso drives them more rapidly back to zero than Ridge.

In a similar way as with Ridge, I have removed the mean from each feature and plotted the intercepts for each value of lambda along with the beta coefficients. The intercepts are all close to 0,

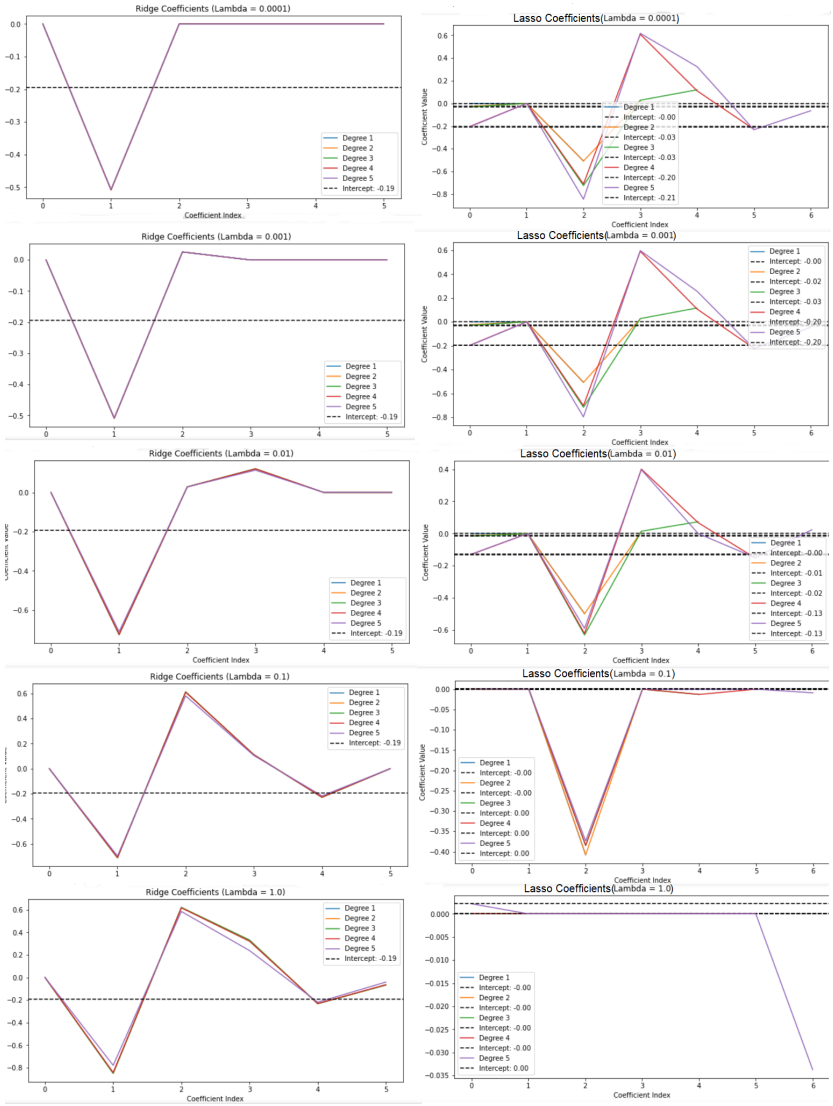
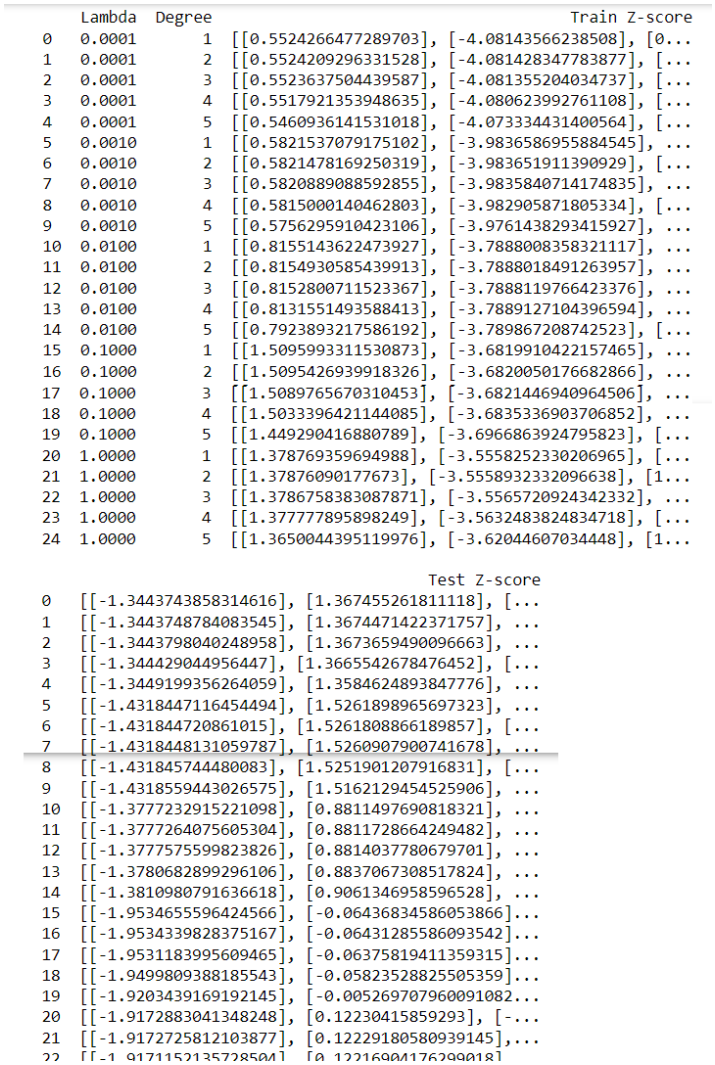


Figure 9: **Beta-Coefficients for Ridge and Lasso**

9

suggesting that the regularization term is effectively setting the intercept close to 0 during the model's training process because it believes that including the intercept doesn't significantly improve model's performance. However the intercepts are higher for the lower degrees of λ. So the Lasso also drives the intercept towards 0 as regularization increases. The choice between Lasso and Ridge (and the value of the intercept) might depend on whether I want to prioritize model interpretability or prediction accuracy. Lasso tends to produce sparse models with some coefficients set to exactly zero, which can make the model more interpretable. Ridge tends to shrink coefficients towards zero but doesn't set them exactly to zero, which can lead to better predictive performance. Evaluating the Franke Function near the origin can help answer whether Lasso or Ridge is the winner in this case: The first 2 terms of the Franke Function contain Gaussian terms with positive coefficients far away from the origin, whereas term 3 has a positive coefficient centered far away from the origin so they all have small influence close to the origin. The 4th term contains a negative Gaussian term with coefficient -0.2, far from the origin but with a negative impact on the function's value near the origin. This is the same intercept we get consistently in Ridge regression, but it only occurs for polynomial degree 4 and 5 for λ=0.0001 and =0.001, so the Ridge intercept is the better fit. However the model's ability to influence the size of the coefficients was weaker than Lasso. The Beta-Coefficients for Ridge are pretty unstable, and we see that the Z-scores are all small.(10) With Lasso, the coefficients became closer to 0 for the 4th and 5th degree polynomial with lambda=1, such that the method and regularization



	Lambda	Degree	Train Z-score	Test Z-score
0	0.0001	1	[[0.5524266477289703], [-4.08143566238508], [0...	[[-1.3443743858314616], [1.367455261811118], [...
1	0.0001	2	[[0.5524209296331528], [-4.081428347783877], [...	[[-1.3443748784083545], [1.3674471422371757], ...
2	0.0001	3	[[0.5523637504439587], [-4.081355204034737], [...	[[-1.3443798040248958], [1.3673659490096663], ...
3	0.0001	4	[[0.5517921353948635], [-4.080623992761108], [...	[[-1.344429044956447], [1.3665542678476452], [...
4	0.0001	5	[[0.5460936141531018], [-4.073334431400564], [...	[[-1.3449199356264059], [1.3584624893847776], ...
5	0.0010	1	[[0.5821537079175102], [-3.9836586955884545], ...	[[-1.4318447116454494], [1.5261898965697323], ...
6	0.0010	2	[[0.5821478169250319], [-3.983651911390929], [...	[[-1.431844720861015], [1.5261808866189857], [...
7	0.0010	3	[[0.5820889088592855], [-3.9835840714174835], ...	[[-1.4318448131059787], [1.526090790074167 8], ...
8	0.0010	4	[[0.5815000140462803], [-3.982905871805334], [...	[[-1.431845744480083], [1.5251901207916831], [...
9	0.0010	5	[[0.5756295910423106], [-3.9761438293415927], ...	[[-1.4318559443026575], [1.5162129454525906], ...
10	0.0100	1	[[0.8155143622473927], [-3.7888008358321117], ...	[[-1.3777232915221098], [0.8811497690818321], ...
11	0.0100	2	[[0.8154930585439913], [-3.7888018491263957], ...	[[-1.3777264075605304], [0.8811728664249482], ...
12	0.0100	3	[[0.8152800711523367], [-3.7888119766423376], ...	[[-1.3777575599823826], [0.8814037780679701], ...
13	0.0100	4	[[0.8131551493588413], [-3.7889127104396594], ...	[[-1.3780682899296106], [0.8837067308517824], ...
14	0.0100	5	[[0.7923893217586192], [-3.78986720874252 3], [...	[[-1.3810980791636618], [0.9061346958596528], ...
15	0.1000	1	[[1.5095993311530873], [-3.6819910422157465], ...	[[-1.9534655596424566], [-0.06436834586053866]...
16	0.1000	2	[[1.5095426939918326], [-3.6820050176682866], ...	[[-1.9534339828375167], [-0.06431285586093542]...
17	0.1000	3	[[1.5089765670310453], [-3.6821446940964506], ...	[[-1.9531183995609465], [-0.06375819411359315]...
18	0.1000	4	[[1.5033396421144085], [-3.6835336903706852], ...	[[-1.9499809388185543], [-0.05823528825505359]...
19	0.1000	5	[[1.449290416880789], [-3.6966863924795823], [...	[[-1.9203439169192145], [-0.005269707960091082...
20	1.0000	1	[[1.378769359694988], [-3.5558252330206965], [...	[[-1.9172883041348248], [0.12230415859293], [-...
21	1.0000	2	[[1.37876090177673], [-3.5558932332096638], [1...	[[-1.9172725812103877], [0.12229180580939145],...
22	1.0000	3	[[1.3786758383087871], [-3.5565720924342332], ...	[[-1.9171152135728504], [0.12216904176299018]...
23	1.0000	4	[[1.377777895898249], [-3.5632483824834718], [...	
24	1.0000	5	[[1.3650044395119976], [-3.62044607034448], [1...	

Figure 10: **Z-Scores for Ridge**

10

gave a better performance overall. The OLS model' gave a uniform modelling across all polynomial degrees, with —B1 being = 1, and all the others were equal to 0. While a β-coefficient equal to 0 can be useful, the OLS method does not provide the same flexibility in terms of deciding the parameters when I want the Beta coefficients to be small and close to 0. The Beta-coefficients for Lasso were quite unstable, as shown by the small Z-scores(11). The Lasso regression drove the coefficients more effectively towards 0 than the Ridge regression, so as long as we can balance the strength of the factor driving the coefficients away from 0 (The Polynomial Degree) with the factor driving them back to 0 (The Regularization) it provides more flexibility than the OLS and the Ridge regression. When managed carefully, the Lasso regression is more flexible, but the intercept is more aligned with the true data near the origin with the Ridge regression. The difference in the way the regulariaztion terms drive the coefficients aswell as R2 towards zero in Lasso regression but not in Ridge is consistent with the the differences between the two methods identified in the Ridge and Lasso part of the Methods section.

3.1.4 Residuals before Resampling

For OLS, the error term residuals are very evenly distributed around 0. See Figure 12. (They take on extremely small values). For the Ridge regression the residuals have the highest frequencies at -1.25 and -0.75 approximately, but they are evenly spread out from -1.5 to 1.0. The model is here

```
Test Z-scores
0                    -1.3444, 1.3669, -2.5494, -1.6457, -2.5494, 0.1619, -1.3444, -2.8507, -0.4406, -0.4406, -1.3444, 1.
0656, -3.4532, -2.2482, -2.2482, 0.7644, -4.0557, -3.4532, -1.9469, -0.1394, -3.7545, -3.1519, -1.9469, -1.9469, 1.3669, -3.4
532, -2.8507, -0.1394, 0.7644, 0.1619, -3.1519, -2.5494, 1.0656, -3.7545, -2.2482, -3.4532, -2.5494, 0.7644, -4.0557, 1.0656,
1.0656, -3.1519, -3.4532, -0.1394, 0.7644, 0.7644, -1.0432, 1.0656, 0.7644, -2.5494, -0.4406, 1.0656, -4.0557, -1.6457, -2.85
07, -4.357, 0.4631, -2.2482, -1.6457, -2.2482, 0.1619, -2.8507, -3.1519, -4.0557, 1.0656, -1.9469, -2.2482, -1.0432, -2.8507,
-2.5494, 1.3669, 1.3669, -0.1394, -0.1394, -1.0432, -0.4406, -0.7419, -1.3444, -3.4532, -1.3444
1                         -1.3447, 1.3618, -2.5476, -1.6454, -2.5476, 0.1589, -1.3447, -2.8483, -0.4426, -0.4426, -1.34
47, 1.0611, -3.4498, -2.2469, -2.2469, 0.7603, -4.0512, -3.4498, -1.9462, -0.1418, -3.7505, -3.149, -1.9462, -1.9462, 1.3618,
-3.4498, -2.8483, -0.1418, 0.7603, 0.1589, -3.149, -2.5476, 1.0611, -3.7505, -2.2469, -3.4498, -2.5476, 0.7603, -4.0512, 1.06
11, 1.0611, -3.149, -3.4498, -0.1418, 0.7603, 0.7603, -1.044, 1.0611, 0.7603, -2.5476, -0.4426, 1.0611, -4.0512, -1.6454, -2.
8483, -4.3519, 0.4596, -2.2469, -1.6454, -2.2469, 0.1589, -2.8483, -3.149, -4.0512, 1.0611, -1.9462, -2.2469, -1.044, -2.848
3, -2.5476, 1.3618, 1.3618, -0.1418, -0.1418, -1.044, -0.4426, -0.7433, -1.3447, -3.4498, -1.3447
```

Figure 11: **Z-Scores for Lasso**

11

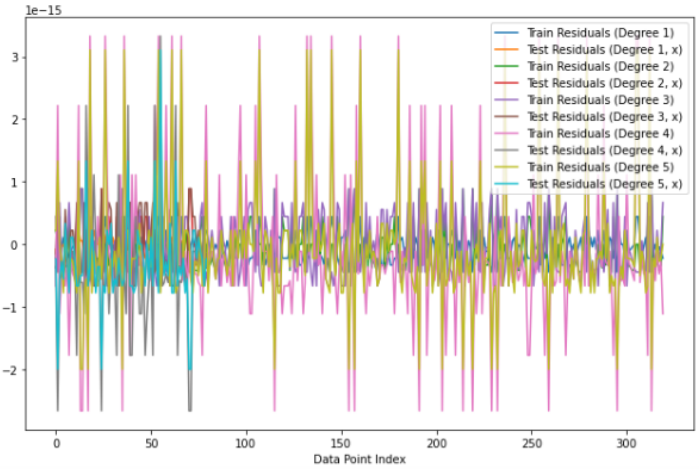


Figure 12: **OLS Residuals**

12

predicting values that are from -1.5 to 1.0 more than or less than the actual values. The distribution of residuals are quite evenly spread out around 0. Finally, I analyzed the residuals of the error term of the regression equation for both test and training data. The residuals most frequently had values of between -1.5 and 0. Comparing with Ridge, where the distribution of residuals took on values from -1.5 to 1.0, the Lasso error term residuals are on average overestimating the actual values, and they are further away from 0 than the Ridge, so the Ridge regression is more well-fitted. The OLS method produced residuals that performed better than both Ridge and Lasso.(Figure 13)(14).

3.1.5 Bootstrapping performed on the OLS

Before running bootstrapping, I performed a bias-variance tradeoff analysis on the OLS data. The bias-variance graph before bootstrapping shows that the Test data is getting a lower bias and higher variance as the polynomial degree increases, and stablizes after the 4th degree polynomial, so the bias and variance doesn't change for the 5th degree.(15) (This was compared with Fig 2.11 in Springer et. Al(Reference 3)).

3.1.6 K-fold Cross-Validation on OLS, Ridge and lasso

After bootstrapping, I got better scores for MSE and R-squared. Test MSE is now exhibiting divergent behaviour as polynomial degree increases, whereas Training MSE decreases. In this first run, I got these results: Test MSE as a function of Polynomial Degree after Bootstrapping: Begins at 0.8 for the 1st degree polynomial, decreases to 0.785 for the 3rd degree polynomial, increases to 0.86 for the 4th degree, and decreases to 0.84 for the 5th-degree polynomial.(16) The bootstrap method produced different MSE scores and patterns for different runs. In each iteration the bootstrap randomly samples data points with replacement. Because of the randomness each resampled dataset will be slightly different. The dataset is also relatively small, leading to increased variability in the MSE. In the 2nd run of the bootstrap the Test MSE as a function of Polynomial Degree after Bootstrapping began at 0.8 for the 1st degree polynomial, decreased to 0.785 for the 3rd degree polynomial, increased to 0.86 for the 4th degree, and decreases to 0.84 for the 5th-degree polynomial.

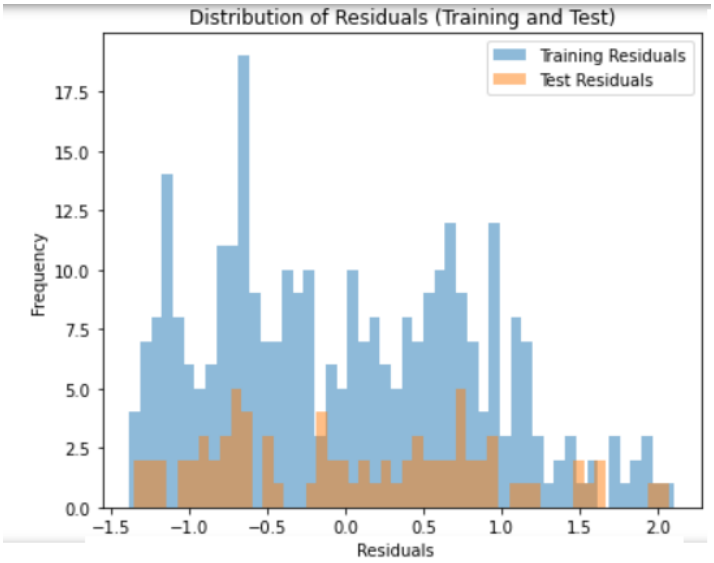


Figure 13: **Ridge Residuals**

13

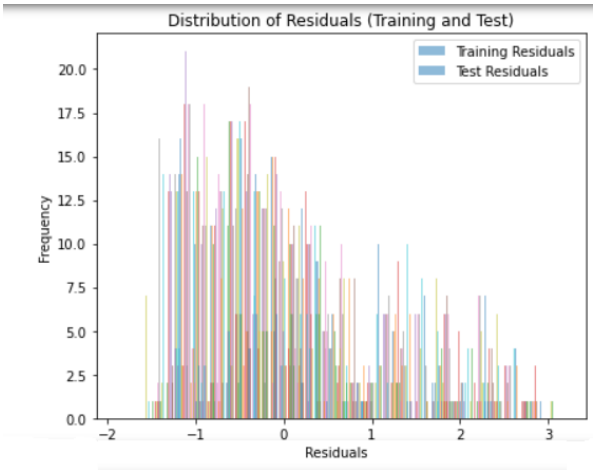


Figure 14: **Lasso Residuals**

14

I also performed a Bias/Variance-tradeoff analysis after bootstrapping, where the first run gave me better scores than before bootstrapping for MSE and R-squared. $Bias^2$ was now increasing, though not strictly, as polynomial degree increases, whereas variance decreased. On the 2nd run, The Bias-Variance Tradeoff After Bootstrapping on the 2nd run gave a Variance from 0 to 0.1, and a $Bias^2$ around 0.6. These values were in general 0.1-0.15 higher than in the first run.

The curvature was different on different runs of the bootstrap, and the MSE results were quite different. After bootstrapping, low variance means that the model's predictions are not sensitive to variations in the training data. In other words, the model is not overreacting to noise or random fluctuations in the data. But the high bias indicates that this is a case of underpredicting. The model is too simplistic to capture the complexity of the data. It doesn't adapt well to the training data, and it results in systematic errors. I ran cross-validation with the OLS method using k=5 and k=10, with k being the number of partitions of the dataset. In short, with smaller k, each dataset contains more data. See Methods(2.3) for more information. Test MSE after Cross-Validation with 5-folds: Gave an MSE at 0.8 for the 1st degree polynomial, strictly increasing to 0.825 for the 4th degree polynomial, and 0.845 for the 5th degree. The polynomial degree that results in the lowest cross-validated MSE is considered the optimal parameter for the OLS model for the given Franke function and dataset. In this

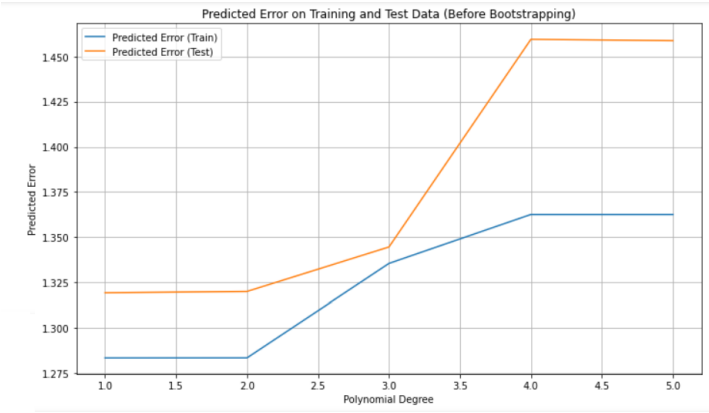


Figure 15: **Bias-Variance Tradeoff before Bootstrapping**
15

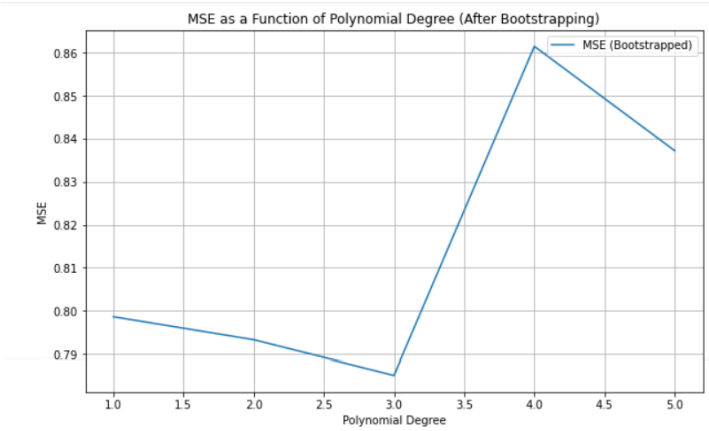


Figure 16: **Bootstrap with the OLS Method**
16

case it is the 1st degree polynomial (Figure 18). Test MSE after Cross-Validation with 10-folds: Gave an MSE at 0.705 for the 1st degree polynomial, but not strictly increasing like the 5-folds, topping at 0.742 for the 3rd-degree polynomial, decreasing to 0.728 for the 4th-degree and up to 0.746 for the 5th-degree(Figure 19). It gave a lower MSE than the bootstrap, whereas the 5-fold gave a higher MSE than the bootstrap. With more examples demonstrating this in the appendix, Cross-validation or bootstrapping does not always give the best results. It depends on the specific run, sometimes bootstrapping comes out on top, sometimes CV comes out on top. With Ridge regression I used one of the k folds as a validation set to find the optimal parameters for this particular set of parameters/data. There is a looping over the lambda-parameters, and then it finds where the training block begins and where the test block begins. There was a small correlation between polynomial degree and decrease in MSE for 3 of the lambda values. The 5th degree polynomial for the lambda=0.1 returns the best MSE, and is the optimal parameter for Ridge(17) I've run across numbers of k-folds to understand whether the results are independent of k. The results are not completely independent of the no. of folds(See 22 to 20 for a more in-depth graphing over k-folds with different values of λ). As the no. of folds increase, the MSE becomes closer to 0. So the optimal parameter is actually the 5th-degree polynomial for lambda=0.1 with 10-folds cross-validation. With Lasso, as the polynomial degree increased, the MSE increased across all lambda-values. For the first 3 polynomial degrees, its very consistent around 0.75. After the 3rd degree-polynomial it gradually increased to around 0.78 for the 4th-degree and takes off rapidly to around 1 for the 5th-degree. For lambda=1, the 5th-degree polynomial gives an MSE of 0.92, showing that higher regularization strength drives the coefficient down to 0 (Consistent with earlier findins that it mitigates the prevalent effect). I've run across numbers of k-folds to understand whether

the results are independent of number of folds, k. The results are not completely independent of the no. of folds. I see that as the no. of folds increase, the MSE became closer to 0 for the lambda-values 0.0001, 0.001, and 0.01. As for lambda=0.1, the opposite was the case, and for lambda=1, the no. of folds produced random results, and didnt have any relation to the MSE. So, the higher regularization strength canceled out the effect of the increased no. of folds. These results occured when using both training and test data(26).

When using only the test data, the relationship was quite different(shown in Figures:

3.2 Topographical Data Results

I expected that the linear regression models for the topographical data would yield similar results with essentially no difference between the 3 models for several of the measurements. This is what occured for some of the measurements, but not all. It could be attributed to:

Linearity: If the relationship between the predictor variables and the target variable in your topographical data is relatively linear, OLS can work quite well. OLS assumes a linear relationship between predictors and the target variable, and if that assumption holds, other regularization techniques like Ridge and Lasso may not provide significant advantages.

Low Multicollinearity: When predictor variables in the dataset are not highly correlated with each other, the differences between OLS, Ridge, and Lasso can be minimal. Ridge and Lasso are often more beneficial when multicollinearity is present, as they can help mitigate its effects: The correlation between two variables X_i and X_j is measured using the Pearson correlation coefficient, which is defined mathematically as:

$$\rho(X_i, X_j) = \frac{\text{Cov}(X_i, X_j)}{\sqrt{\text{Var}(X_i) \cdot \text{Var}(X_j)}}$$

In this equation:

- $\rho(X_i, X_j)$ represents the Pearson correlation coefficient between variables X_i and X_j. - $\text{Cov}(X_i, X_j)$ represents the covariance between X_i and X_j. - $\text{Var}(X_i)$ and $\text{Var}(X_j)$ represent the variances of X_i and X_j, respectively.

Small Feature Space: If the dataset has a relatively small number of features or predictors compared to the number of observations, overfitting may not be a significant concern. Regularization techniques like Ridge and Lasso are more valuable when dealing with high-dimensional data where there is a risk of overfitting.

Noise Level: If the topographical data is relatively clean with low levels of noise, the added complexity of Ridge and Lasso may not be necessary. These regularization techniques are more helpful when dealing with noisy data, as they can help reduce model variance.

Inherent Simplicity: Topographical data may have a relatively straightforward relationship between predictor variables (e.g., elevation, slope, aspect, etc.) and the target variable (e.g., terrain charac-

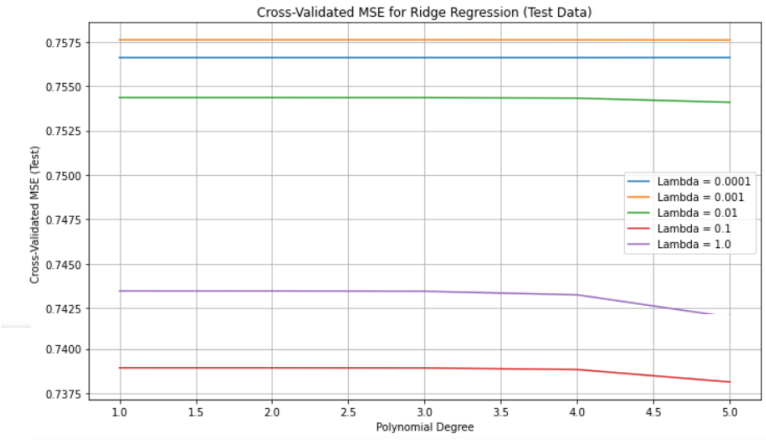


Figure 17: **Cross-Validation using Ridge Regression**

17

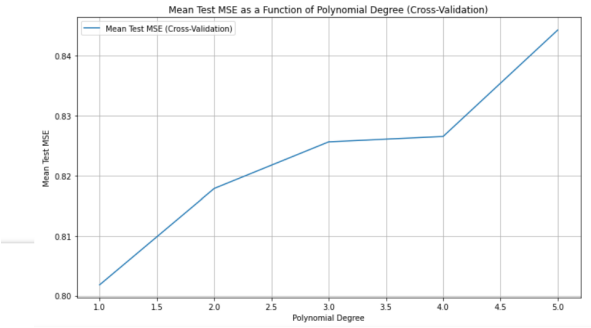


Figure 18: **5-fold Cross-Validation with the OLS Method**
18

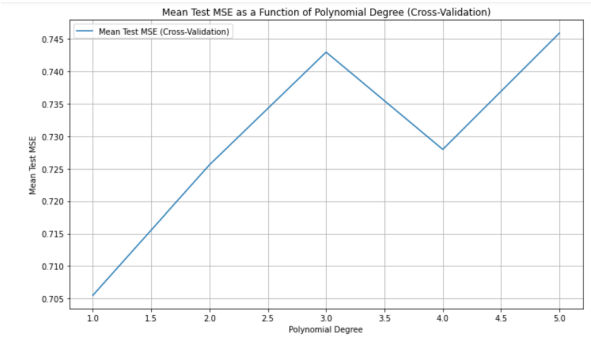


Figure 19: **10-fold Cross-Validation with the OLS Method**
19

teristics). In such situations, simple linear regression (OLS) can capture the underlying patterns effectively.

The interpretability of topographical data is usually very high, and has contributed to the ease of modeling, as the physical and geographical characteristics often have clear and understandable relationships.

When performing Boostrapping on the topographical data, running 1000 bootstrap iterations over 5 polynomial degrees was not possible due to the huge amount of data points. So I decided to reduce the number of bootstraps to 100 for the OLS, and 20 for the Ridge and Lasso and reduced the degree range to 1-4 rather than 1-5. I didn't edit the number of data points because there were too many in the original data, so setting a low number would run a very high risk of yielding a set of points not being representative.

These linear regression models in and of themselves were not capable of handling this very large

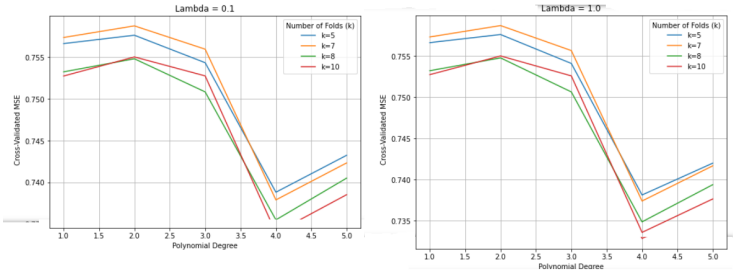


Figure 20: **Cross-Validation, Ridge, k folds for λ=0.1, λ=1**
20

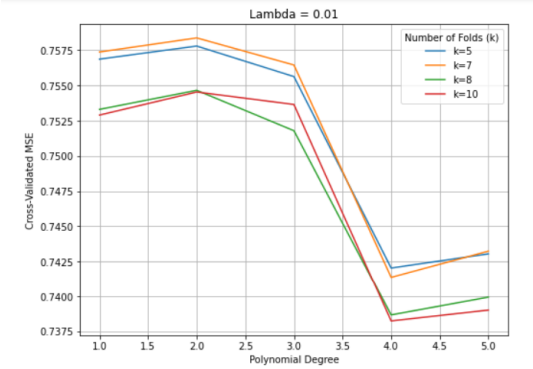


Figure 21: **Cross-Validation using Lasso Regression with k folds for λ=0.01**
21

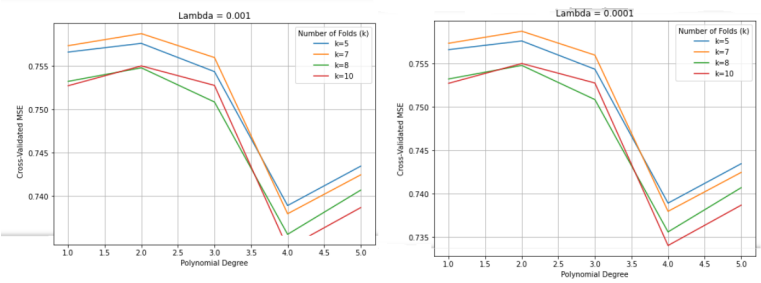


Figure 22: **Cross-Validation, Ridge, k folds for λ=0.0001, λ=0.001**
22

dataset.

3.2.1 MSE Before Resampling

The MSE didn't exhibit any difference between polynomial degrees, and it was consistently at 1 for the training data at at 0.975 for the test data with the OLS method. The same occured to the Ridge and Lasso, where the MSE was constant around 1 and 0.975 respectively. The lack of variation for the different regression methods indicate that the regularization term has no effect on the data in this case. By increasing the regularization term to [10, 100, 1000, 10000] I didn't get any different results, and it seems that the topographical data has perfectly linear relationships and/or high collinearity where Ridge and Lasso does not have any effect.(The Ridge MSE was exactly the same for λ=[0.0001, 0.001, 0.01, 0.1, 1] aswell. (27) and (28

3.2.2 R2, Beta-Coefficients and Residuals Before Resampling

The Training R2-score was 1.92 for the first-degree polynomial, and from 1.94-1.95 for the 2-5th degree polynomial both for OLS, Ridge and Lasso, and there was no difference in the R2 Score across the models. The Test R2 score was 0.53 for the 1st degree-polynomial, and decreased to 0.4 for the 2nd-degree polynomial before increasing to 0.42-0.43 for the 3-5th degree polynomial for both OLS, Ridge and Lasso. The score increased and decreased by very small sizes across all degree polynomial and it hardly exhibits any specific increasing or decreasing behavior. The R2 scores for the test data is a lot lower than the training data, they are both equally far from the perfect score of 1 but on opposite sides. The test R2 decreased from the 1st to 2nd degree polynomial, while the training R2 increased in the same span.(29),(30. The large difference in R2 scores for test and training indicates overfitting, likely due to the large number of data points. It may also because the model is too complex, or because of a high-dimensional feature space, and some of the features may be noisy or irrelevant due to a lack of good feature engineering. As the dimensionality (p) increases, the amount of data

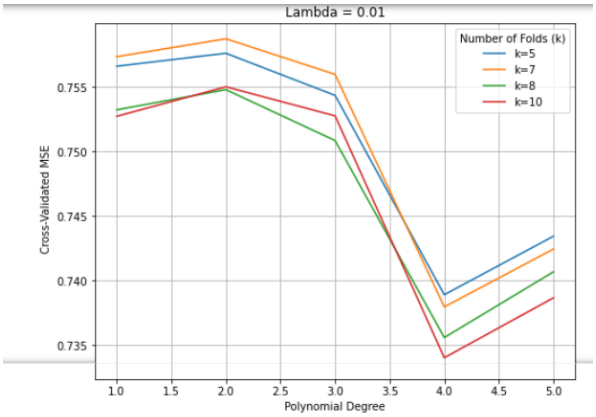


Figure 23: **Cross-Validation, Ridge, k folds for λ=0.01**
23

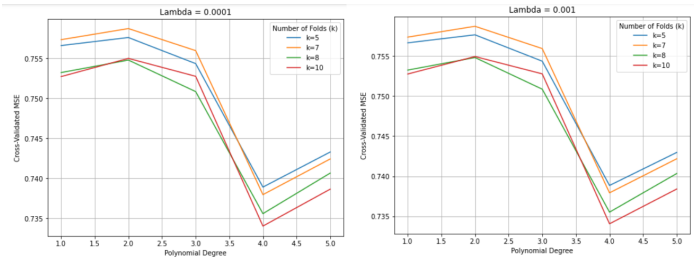


Figure 24: **Cross-Validation using Lasso Regression with k folds for λ=0.0001 and λ=0.001**
24

needed to adequately cover the feature space grows exponentially. The high-dimensional feature space mathematically simply means that in the linear regression model, where n is the no. of data points

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon, \quad p \gg n$$

The same results applied to Ridge regression when it used the same lambda values as the Lasso regression, confirming what I saw with the MSE that the regularization didn't have any effect for λ small and very large. The Beta-coefficients changed quite a bit from OLS to Ridge and Lasso. Here, the Ridge and Lasso was successful in shrinking the coefficients towards 0. The Ridge and Lasso effectively imposed a penalty on the data using a large range of λ-values. The OLS Beta-coefficient was just one significant coefficient(31), β1, having a value of 1.45. With Ridge and Lasso, the coefficients didn't change with the change in regularization term L1 to L2, but stayed exactly the same(32). They were different from the OLS Coefficients(Here Beta-coefficients for Ridge and Lasso are shown in the same plot as they are completely identical, and this was the case for λ=[0.0001,0.001,0.01,0.1,1] with Ridge.) All the coefficients are exactly 0, meaning both Ridge and Lasso completely eliminated the β-coefficients. So they performed an effective "feature selection". It also indicates sparsity and simplification, meaning only a subset of the available predictor variables is included in the final model. The Residuals for the OLS method showed that the residuals were within $e^{-}6$ to 0(they were 0 with an accuracy equal to 6 decimal points). There were no differences between polynomial degrees, which indicates that the complexity of the model (the polynomial degrees) are appropriate, and training residuals had a frequency of 8 at 0, with very small frequencies from 0 to 2.75. There were no differences in the residuals for OLS, Ridge and Lasso(34) and (33).

3.2.3 Bootstrapping applied to OLS

After applying bootstrapping to the OLS data, the test MSE divergend compared to the test MSE without bootstrapping for the OLS(35). Because of computational cost, only the first 4 degrees were

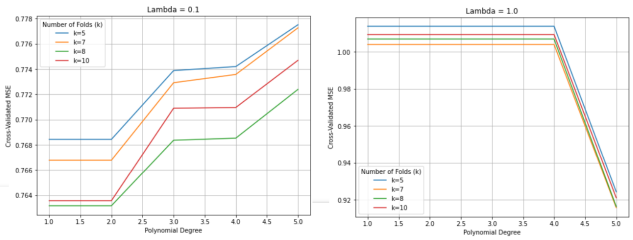


Figure 25: **Cross-Validation using Lasso Regression with k folds for λ=0.1 and λ=1**
25

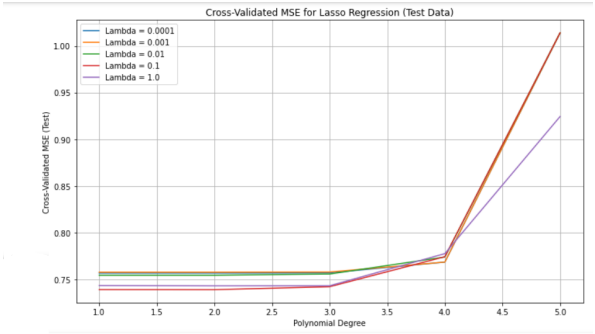


Figure 26: **Cross-Validation using Lasso Regression on Test data**
26

analyzed. The MSE took turns between increasing and decreasing, and stayed within 0.97-0.98. The MSE was diverging, but by very small values and was likely caused by random variablity in the data.

3.2.4 K-fold Cross-Validation for OLS, Ridge and Lasso

First, I analyzed the OLS method using 5-folds Cross-Validation(36). The test and training data followed the exact same pattern until after the 3rd degree, where the test MSE and R2 began to decrease slightly faster. The Cross-Validation performed significantly worse than the bootstrap and the original OLS analysis with respect to MSE. Using Ridge(37), the MSE decreased slightly as the number of k folds increased from 5 to 8, but unexpectedly it fell when k increased from 8 to 10. The MSE increased as polynomial degree rose from degree 1 to 2 in all instances, but it didn't increase from degree 2 to 4(Only 4 degrees were analysed due to computational restrictions described in the introduction of the section). With Lasso, the results were exactly identical to the results with Ridge, and like the beta-coefficient, the difference between L1 and L2 regularization strength did not affect the outcome(38)

4 Appendix

The following results and the corresponding code used to implement was considered not relevant enough to include in the main analysis, but especially the first point about more bootstrapping runs, is an interesting point and should be viewed in accordance with the main analysis. The appendix file on https://github.com/ingebrigtkj/Project-1-FYS-STK4155 itself contains more comments.

4.1 More Bootstrapping Runs on the Franke Function

The Appdenix file on https://github.com/ingebrigtkj/Project-1-FYS-STK4155 contains more runs of the bootstrap on the Franke function with the OLS Method. This was done to show that the bootstrap method will give different results on different occasions because it partitions the data differently. The correct method is to do many bootstraps with the same dataset and model, and pick the one giving

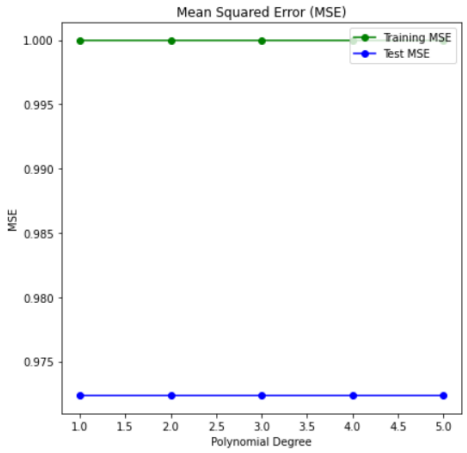


Figure 27: **Topographical data, MSE OLS**
27

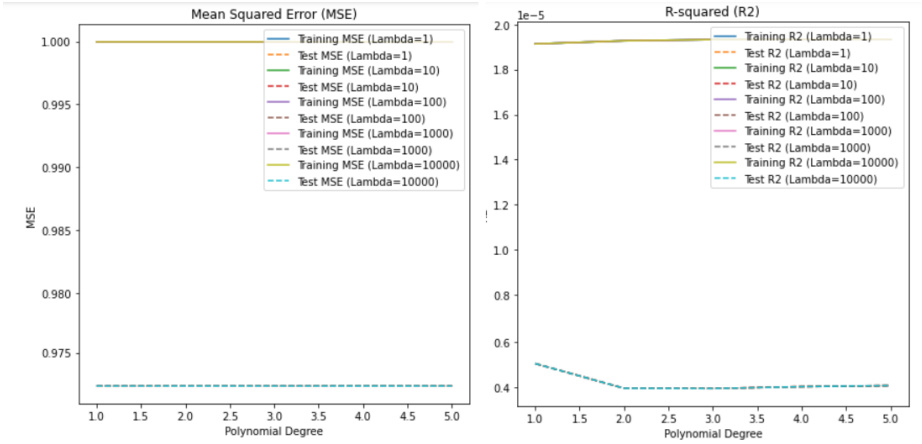


Figure 28: **Topographical data, MSE Ridge to the left, and MSE Lasso to the right**
28

the best MSE. The appendix contains the other runs. Here is an example displaying that the results were quite different for different reruns: The test and training MSE were both decreasing in this case, with the 4th and 5th degree showing equal MSEs. The training MSE decreased from 0.7 to 0.6, and the test MSE (checked in Bootstrapping and crossvalidation) from 0.56 to 0.36. The test MSE had a big dropoff as degree increased(39).

The MSE after bootstrapping gave a strictly decreasing graph beginning at 0.550 and decreasing to 0.375. The MSE after 10-folds cross validation gave an MSE beginning at 0.69, increasing to 0.7 for the 2nd degree polynomial before dropping down to 0.61 for the 4th degree and 0.63 for the 5th degree. Also, the bootstrapping outperformed the crossvalidation in this case.

4.2 Ridge and Lasso with more regularization strength on the Franke Function

Using the initial Franke Function model, Ridge and Lasso was also implemented with 9 different regularization strengths. The regularization strengths were both smaller and larger than the original ones, and the results were identical or very close to identical. The code used for this is included in the appendix on https://github.com/ingebrigtkj/Project-1-FYS-STK4155.

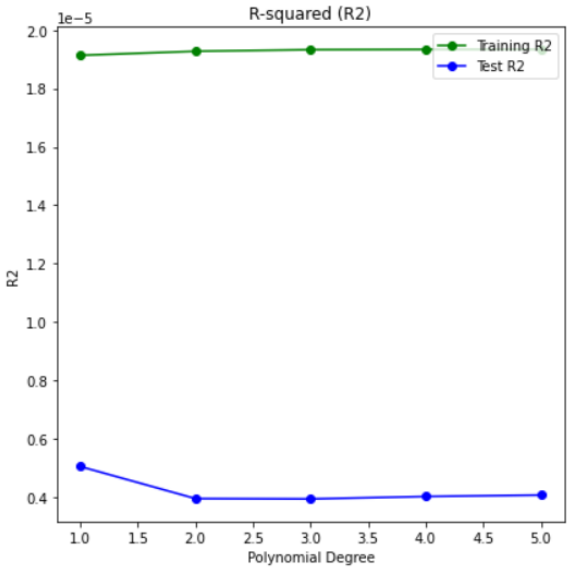


Figure 29: **Topographical data, R2 OLS**

29

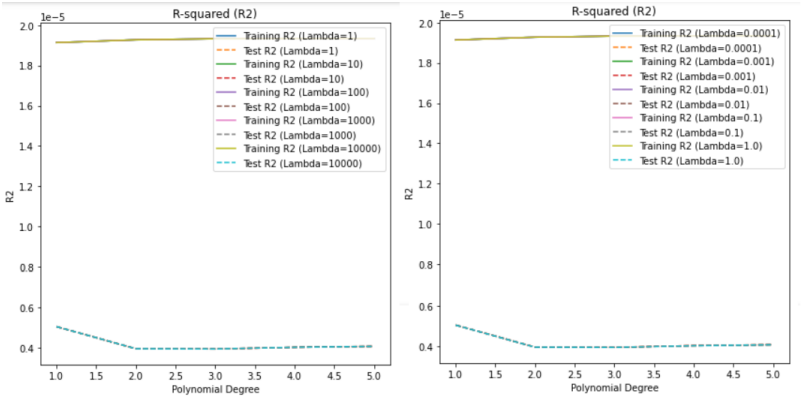


Figure 30: **Topographical data, R2 Ridge and Lasso**

30

4.3 Polynomial Fits on the Franke Function

The polynomial fits over the scaled data points were included as graphs, both the scaled data points were plotted in green, and the polynomial fits were plotted in green across the x- and y-axis for the OLS, Ridge and Lasso methods used on the Franke function. Some of them showed that the polynomial fits changed as the polynomial degrees changed, and for Ridge and Lasso as the λ-parameter changed. The results showed that the lower degree polynomials produced smoother fits, and the 5th degree polynomial fit the training data very closely, creating a more complex and flexible model with more spread. The higher regularization on Ridge and Lasso, the regularization made the model more constrained and there was a reduction of spread. These results are also interesting even though they were not included in the main analysis due to not being part of the selected measurements.

5 Conclusion

The aim is to strike a balance between what is a good model and what one is willing to spend when using that model. One should always try to strike a balance between model complexity and simplicity.

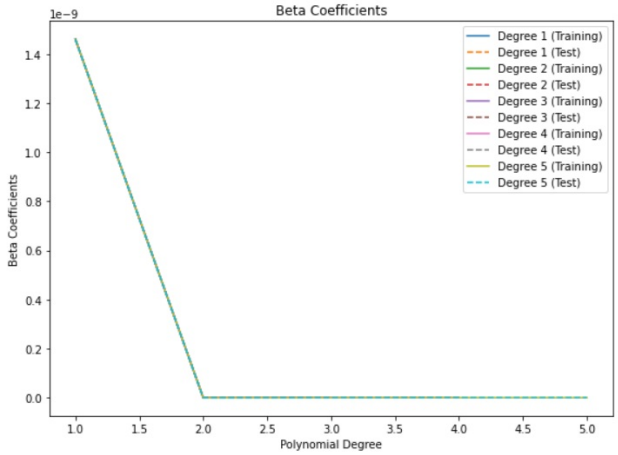


Figure 31: **Topographical data, Beta-coefficients OLS**
31

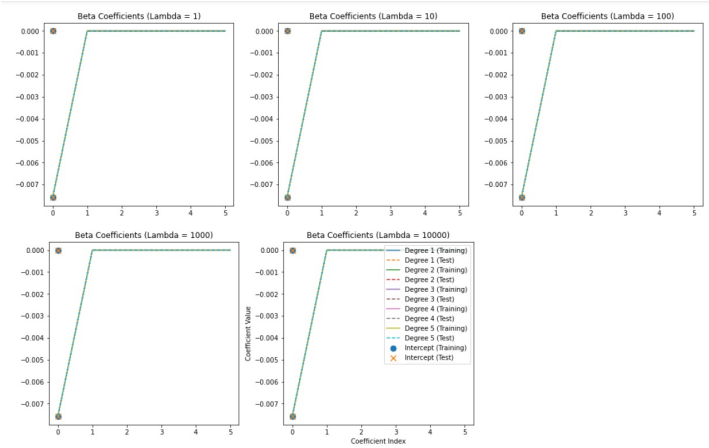


Figure 32: **Topographical data, Beta-coefficients Ridge and Lasso. The results are for large values of λ; the exact same results were the case for smaller values of λ**
32

The model needs to be ready to be applied to unseen data. With the Franke function, the Lasso model offered more flexibility than OLS and Ridge, but it didn't give better scores on all measurements. The MSE was better handled by OLS and Ridge, while the R2 score was best handled by Ridge. The β-coefficients were unstable for both Ridge and Lasso, but Lasso provided more flexibility in tuning the parameters. The Bootstrapping on Cross-Validation didn't necessarily give a better MSE score than the original OLS model, but running enough reruns was the key to achieving a marginally better MSE-score. For the Topographical Data, which had a lot more data points, many of the measurements gave no different results with the different models because of model ineffeciencies, while some had essentially inverse relationships at the lower polynomial degrees. The results showed the importance of picking the right polynomial degrees and tuning parameters for a certain type of problem, and how these parameters need to be individually evaluated in different types of problems with different data sets and mathematical assumptions. A good understanding of the data and the problem that needs to be solved is therefore imperative when deciding which methods to use. This means understanding the data complexity, homo/heteroscedasticity, regularization strength, flexibility, evaluation, data preprocessing like scaling, tuning and regularization techniques. This understanding will prevent the usage of more complexity and computation when less is needed, which will be important looking forward, for example

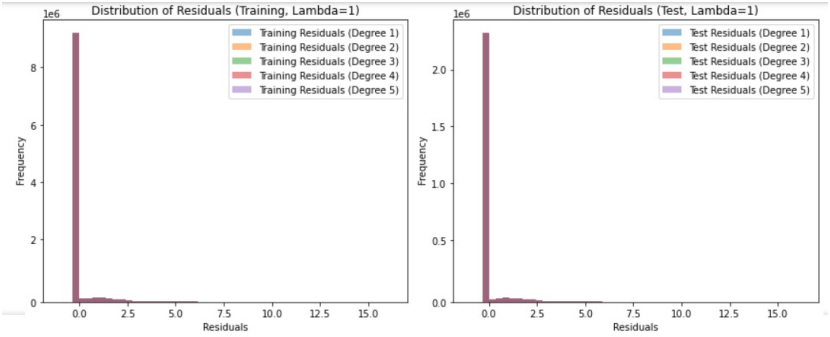


Figure 33: **Topographical data, Residuals Ridge and Lasso (Not all λ are included in the plot, but there were identical results for all values of λ)**
33

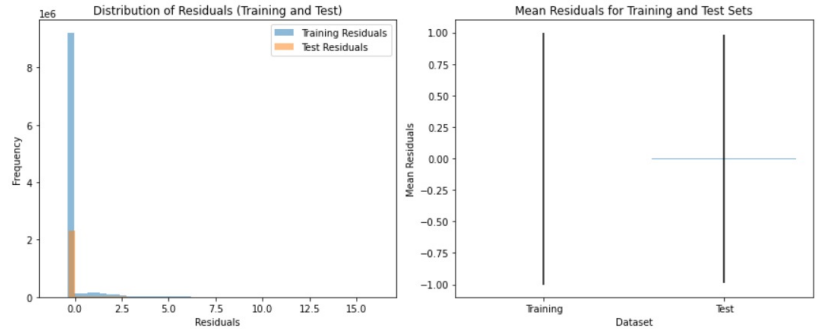


Figure 34: **Topographical data, Residuals OLS**
34

with respect to assessing when Stochastic Gradient Descent and more advanced techniques will be necessary given the extra computational resources required.

References

[1] https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week35.ipynb In addition, my own exercise notes for the derivation and expressions for the optimal parameters of the OLS method.

[2] https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week37.ipynb for notes on the Bootstrap and Cross-Validation in addition to the mathematical expressions for the integral Dirac-delta-function and its relationship with k-folds Cross-Validation

[3] https://link.springer.com/book/10.1007/978-0-387-84858-7 Chapter 3 for discussion of mathematical implementation of the OLS, Ridge and Lasso and discussion of relationship and differences between Ridge and Lasso with respect to the prior and posterior distribution.

[4] https://www.deeplearningbook.org/ Chapter 5 for an in-depth discussion about resampling.

[5] https://earthexplorer.usgs.gov/ for downloading topographical terrain data from Tel Aviv and Haifa region, Israel to be used for a rerun of the code without the Franke function.

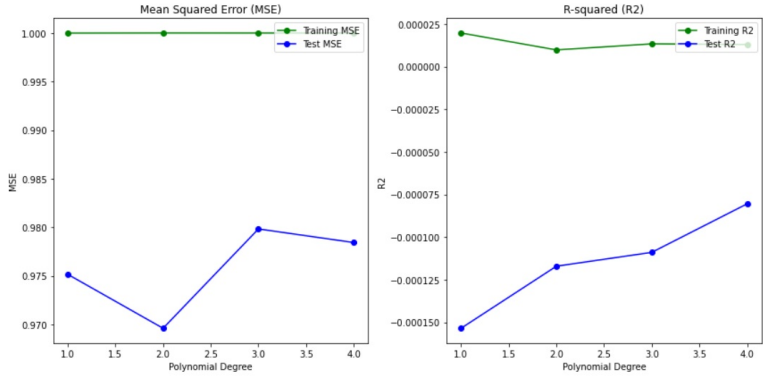


Figure 35: **Topographical data, Bootstrapped OLS. MSE on the left, R2 on the right**
35

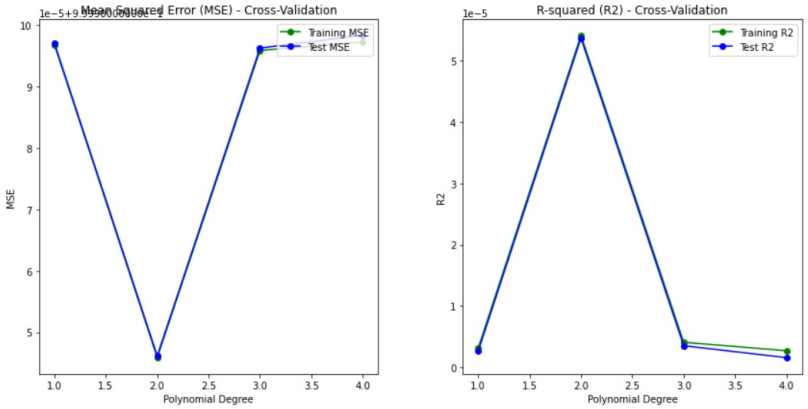


Figure 36: **Topographical data, Cross-Validation OLS with 5-folds)**
36

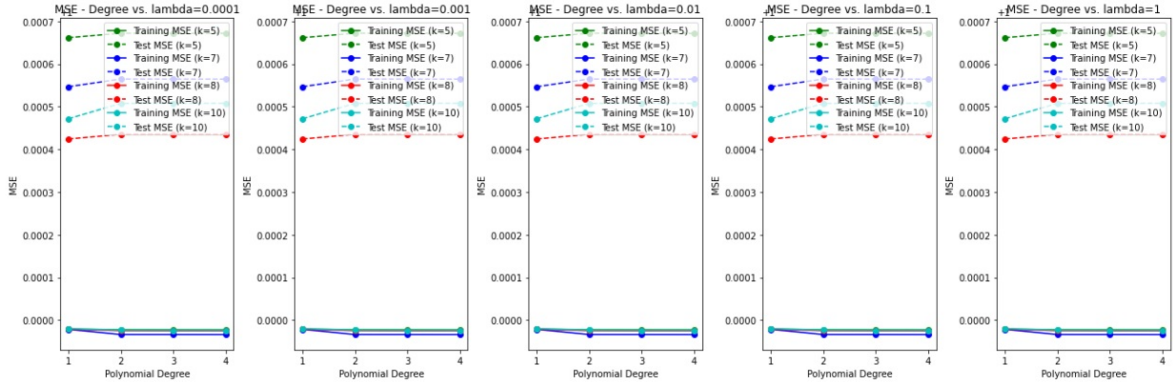


Figure 37: **Topographical data, Cross-Validation Ridge regression using 5,7,8 and 10-folds. In all cases, the graphs flat at 0.000 are training data. The graphs showing positive values are the test data.)**
37

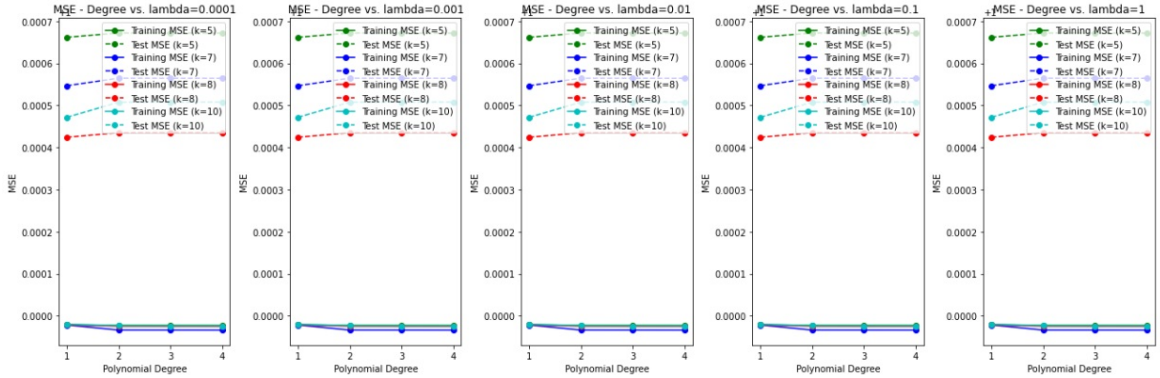


Figure 38: **Topographical data, Cross-Validation Lasso regression using 5,7,8 and 10-folds. In all cases, the graphs flat at 0.000 are training data. The graphs showing positive values are the test data.)**
38

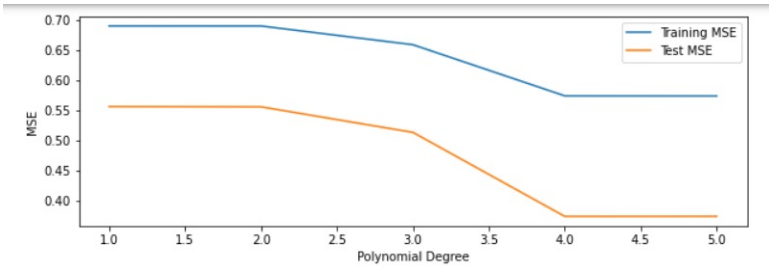


Figure 39: **4th run of the bootstrap method on the Franke function with the OLS Method. Results are very different from the MSE displayed in Section 3.1.5!)**
39