# Project 1 - fysstk4155

Tor-Andreas Bjone, Jens Junker Pedersen

October 11, 2022

## Contents

**Abstract**

In this report we looked at the differences between the ordinary least square-, lasso- and ridge-regression methods. We also used the resampling methods of bootstrap and cross-validation. This was all used on a dataset made using the Franke function with noise and terrain data from the Oslo area taken in the SRTM mission in 2000. We did not find big differences between the regression methods for the Franke function, but Lasso and Ridge did better on the terrain data.

# 1 Introduction

In supervised machine learning, linear regression is an indispensable tool. It is used to find linear relationships between variables. We call the predicting variables independent at the variables being predicted for dependent. An example of where this is used is in blood pressure medication, where we can give a patient various dosages of certain drugs and watch how their blood pressure responds. Then we can find what drugs has the the highest correlation with lowering the blood pressure.

We will look at three linear regression methods, called ordinary least squares (OLS), Least absolute shrinkage and selection operator (Lasso) and Ridge regression. The OLS method is fast and easy to implement but it can be inaccurate when we have highly correlated independent variables. The Ridge and Lasso method solves this by introducing a hyperparameter $\lambda$, which is used to regularize the model. Therefore depending on the data, all three models can be useful. The linear fitting will be done using the Vandermonde design matrix, which is a design matrix for a polynomial fit. Though there are many other ways to set up a design matrix.

The data used will be that of the Franke function, that is a function with two gaussian peaks and a dip, which is used for testing purposes. Then we will look at terrain data from the Oslo area. This data will be taken from the Space shuttle radar topography mission (SRTM) from year 2000 and will be downloaded from https://earthexplorer.usgs.gov/.

The data will be split into a training set, which we train the model on, and a test set, on which we will do the model fitting. Then we evaluate the model with the mean squared error, which we want to be as low as possible. And the score function, which we want to be as high as possible. We can then use these values to compare the different regression models.

Then we look at two resampling methods called the bootstrap and cross-validation. These are methods to draw repeated samples of the data and refit a model on each sample. This gives additional information that would not be attained by only fitting the model once. This will reduce the variance of the model, but likely also increase the bias which can make us miss relevant relations between the dependent and independent variables. This leads to what is called the Bias-Variance tradeoff, which is the problem of minimizing both the bias and the variance.

Also resampling methods can be quite computationally expensive as we will have to refit the model over and over again.

This report is heavily based on the lecture notes by Morten Hjorth-Jensen, found on his github https://github.com/CompPhysics/MachineLearning and the code for the project can be found at https://github.com/jensjpedersen/Projects_FYS-STK4155/tree/main/Project1.

## 2 Theory

### 2.1 Linear regression

In machine learning we call the independent variable $\mathbf{x}$ a feature and the dependent variable $\mathbf{y}$ a response. A regression model aims at finding a likelihood function $p(\mathbf{y}|\mathbf{x})$, that is the conditional distribution for $\mathbf{y}$ with a given $\mathbf{x}$. The estimation of $p(\mathbf{y}|\mathbf{x})$ is made using a data set with $n$ cases $i = 0, 1, 2, ..., n-1$ of a response variable $y_i$ and a set of predictor variables $\mathbf{x}_i = [x_{i0}, x_{i1}, ..., x_{ip-1}]$. The set of these vectors $\mathbf{X} = [\mathbf{x}_0\ \mathbf{x}_1\ ...\ \mathbf{x}_{n-1}]$ is called the design matrix of the model.

The aim of regression analysis is to explain $\mathbf{y}$ in terms of $\mathbf{X}$ through a functional relationship like $y_i = f(\mathbf{X}_{i*})$. When no prior knowledge on the form of $f(\cdot)$ is available, it is common to assume a linear relationship between $\mathbf{X}$ and $\mathbf{y}$. This assumption gives rise to the linear regression model where $\boldsymbol{\beta} = [\beta_0, \beta_1, ..., \beta_{p-1}]^T$ are the regression parameters and the error variable $\boldsymbol{\epsilon}$ is an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors, so that our model becomes

$$\tilde{y}(x_i) = \sum_{j=0}^{p-1} \beta_j x_{ij} = \mathbf{X}_{i*}\boldsymbol{\beta}.$$

Giving that the dependent variable is

$$y(x_i) = \tilde{y}_i + \epsilon_i = \mathbf{X}_{i*}\boldsymbol{\beta} + \epsilon_i.$$

These $n$ equations can be written on matrix form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{1}$$

When assuming that the noise $\epsilon$ is normally distributed around zero with a standard deviation of $\sigma$, we can calculate the expectation value

$$\mathbb{E}[\mathbf{y}] = \mathbf{X}\boldsymbol{\beta} \tag{2}$$

and the variance

$$\mathrm{Var}[y_i] = \sigma^2. \tag{3}$$

The calculations of these values can be found in appendix A

### 2.1.1 Ordinary least squares

The method of ordinary least squares computes the unique line that minimises the sum of squared differences between the true data and that line. In other words we have an optimisation problem on the from

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i-\tilde{y}_i\right)^2 .$$

We define the cost function for ordinary least squares as

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \frac{1}{n}\left[(\mathbf{y}-\tilde{\mathbf{y}})^T(\mathbf{y}-\tilde{\mathbf{y}})\right].$$

Inserting the predicted values $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$ we get

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \frac{1}{n}\left[(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right].$$

We call optimal parameter $\hat{\boldsymbol{\beta}}$ the one that minimises the cost function. This will be a zero of the derivative

$$\frac{\partial\boldsymbol{\beta}}{\partial\beta_j} = 0.$$

Doing this derivative gives

$$\mathbf{X}^T(\mathbf{y}-\mathbf{X}\hat{\boldsymbol{\beta}}) = 0.$$

Rewriting this we get that

$$\hat{\boldsymbol{\beta}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{4}$$

is the optimal parameter if the Hessian matrix $\mathbf{H} = (\mathbf{X}^T\mathbf{X})$ is invertible. This has expectation value

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta} \tag{5}$$

and variance

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

The calculations for this can be found in appendix A.

### 2.1.2 Ridge

The ordinary least squares method can be inaccurate when the model have highly correlated independent variables. Ridge regression tries to solve this by adding a regularization parameter $\lambda$, called a hyperparameter, to the optimization problem. We start by re-writing the optimization problem as

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i-\tilde{y}_i\right)^2 = \frac{1}{n}||\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}||_2^2,$$

where we have used the definition of a norm-2 vector, that is

$$||\boldsymbol{x}||_2 = \sqrt{\sum_i x_i^2}.$$

By adding the regularization parameter $\lambda$ we get that

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_2^2,$$

where we require that $||\boldsymbol{\beta}||_2^2 \leq t$, where $t$ is a finite number larger than zero. We define the cost function to be optimized, that is

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

Minimizing the above equation in the same way as we did OLS gives the optimal parameter

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{7}$$

This means that the Ridge estimator scales the OLS estimator by the inverse of a factor $(1+\lambda)$.

#### 2.1.3 Lasso

We re-write the optimization problem as

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1,$$

with the norm-1 as

$$||\boldsymbol{x}||_1 = \sum_i |x_i|.$$

And this gives us the cost function

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \left\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda||\boldsymbol{\beta}||_1.$$

Doing the derivative with respect to $\boldsymbol{\beta}$ and recalling that the derivative of the absolute value is

$$\frac{d|\beta|}{d\boldsymbol{\beta}} = \text{sgn}(\boldsymbol{\beta}) = \begin{cases} 1 & \beta > 0 \\ -1 & \beta < 0, \end{cases}$$

we have that the derivative of the cost function is

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

and reordering we have

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda sgn(\boldsymbol{\beta}) = 2\boldsymbol{X}^T\boldsymbol{y}.$$

This equation does not have a nice analytical expression, but can be solved using standard convex equation can however be solved by using standard convex optimization algorithms.

## 2.2 Resampling methods and Bias-Variance Tradeoff

When working with a dataset of independent and identically distributed events, called IIDs, we can draw repeated samples from the training set and refit the model on each sample. For each model refitting we can obtain additional information that would not be attainable by only fitting the model once. This will be more computationally expensive as we need to fit the same model over and over again. Let's call the number of fits for $m$. Then we use the central limit theorem to get the approximation of the standard deviation

$$\sigma_m \approx \frac{\sigma}{\sqrt{m}}. \tag{8}$$

This equality is true when $m$ goes to infinity and a good approximation for large $m$ values. What this equation says is that the more we resample a dataset, the lower the variance gets. But this also means that if we have erroneous assumptions in our model, these assumptions will be strengthened. In other words we get higher bias, and this can make us miss relevant relations between features and responses. This is called underfitting. On the other hand, high variance can be an indication of overfitting. The bias-variance tradeoff is the problem of trying to minimize the error in both variance and bias at the same time.

To see problem clearer we will re-write the cost function for the ordinary least squares method and show that this is dependent on the variance and bias of the model. Since this cost function is the mean squared error, we can write it as

$$C(\boldsymbol{\beta}) = \mathbb{E}\left[(\mathbf{y}-\tilde{\mathbf{y}})^2\right] = \mathbb{E}\left[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}})^2\right],$$

where $f(x)$ is assumed to be deterministic, so that $\mathbb{E}[\mathbf{f}] = \mathbf{f} = \mathbb{E}[\mathbf{y}]$, or in other words $f$ is unbiased. We now take the equation of the cost function

$$\mathbb{E}\left[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}})^2\right] = \mathbb{E}\left[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}})^2\right],$$

and by adding and subtracting $\mathbb{E}\left[\tilde{\mathbf{y}}\right]$ in the parenthesis on the right side, we get

$$\begin{aligned}
\mathbb{E}\left[(\mathbf{y}-\tilde{\mathbf{y}})^2\right] &= \mathbb{E}\left[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}}+\mathbb{E}\left[\tilde{\mathbf{y}}\right]-\mathbb{E}\left[\tilde{\mathbf{y}}\right])^2\right] && (9)\\
&= \mathbb{E}\left[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}\left[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}[\epsilon^2] + 2(f-\mathbb{E}[\tilde{\mathbf{y}}])\mathbb{E}[\epsilon] && (10)\\
&+ 2\mathbb{E}[\epsilon]\mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}}\right] + 2\mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}}\right](f-\mathbb{E}[\tilde{\mathbf{y}}]) && (11)\\
&= \mathbb{E}\left[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}\left[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}[\epsilon^2] + 2\mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}}\right](f-\mathbb{E}[\tilde{\mathbf{y}}]) && (12)\\
&= \mathbb{E}\left[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}\left[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}[\epsilon^2] && (13)\\
&= (\mathrm{Bias}\left[\tilde{y}\right])^2 + \mathrm{Var}\left[\tilde{f}\right] + \sigma^2, && (14)
\end{aligned}$$

where we used that $\mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]\right] = \mathbb{E}[\tilde{\mathbf{y}}]$,

$$(\mathrm{Bias}[\tilde{y}])^2 = (\boldsymbol{y}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 \tag{15}$$

and

$$\mathrm{var}[\tilde{f}] = \mathbb{E}\left[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\right]. \tag{16}$$

Since $\sigma^2$ is the variance of the dataset there is not much we can do about this. So what plays a role in the mean squared error is the bias and variance of the model. This means that to reduce the mean squared error we need to minimize these two parameters.

### 2.2.1 Bootstrap

The steps of the independent bootstrap is:

```
1. Draw with replacement $n$ numbers for the observed variables $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$.
2. Define a vector $\boldsymbol{x}^*$ containing the values which were drawn from $\boldsymbol{x}$.
3. Using the vector $\boldsymbol{x}^*$ compute $\hat{\boldsymbol{\beta}}^*$ by evaluating $\hat{\boldsymbol{\beta}}$ under the observations $\boldsymbol{x}^*$.
4. Repeat this process $k$ times.
```

The advantage of bootstrapping is that i's very general, as it does not require distributional assumptions. This makes bootstrap more accurate when data is not well behaved or the sample size is small. We use that when $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}(\boldsymbol{X})$ is a function of random variables, $\hat{\boldsymbol{\beta}}$ itself must be a random variable. Thus it has a pdf $p(\boldsymbol{x})$. The aim of the bootstrap is to estimate $p(\boldsymbol{x})$ by the relative frequency of $\hat{\boldsymbol{\beta}}$. If the relative frequency closely resembles $p(\boldsymbol{x})$, then using numerics, it is straight forward to estimate all the interesting parameters of $p(\boldsymbol{x})$ using point estimators. $p(x)$. If we draw observations in accordance with the relative frequency of the observations, we can replace $p(x)$ by the relative frequency of the observation $X_i$. And in the case that $\hat{\boldsymbol{\beta}}$ has more than one component, and the components are independent, we use the same estimator on each component separately. The histogram of the relative frequency of $\hat{\boldsymbol{\beta}}^*$ is then the estimate of the probability distribution $p(x)$. This is however not of big interest as we want to look at the mean squared error, bias and variance which we find by using those estimators on $\hat{\boldsymbol{\beta}}^*$.

### 2.2.2 Cross-validation

The steps of cross-validation for the various values of $k$ is:

```
1. Shuffle the dataset randomly.
2. Split the dataset into $k$ groups.
3. For each unique group:
        a. Decide which group to use as a set for test data
        b. Take the remaining groups as a set for training data
        c. Fit a model on the training set and evaluate it on the test set
        d. Retain the evaluation score and discard the model
5. Summarize the model using the sample of model evaluation scores
```

The advantage of cross-validation is that the data splitting is not done randomly so we don't get any unwanted influnece on the model building or prediction evaluation. This is called a $k$-fold cross-validation structures the data splitting involves dividing the samples $k$ 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.

# 3 Method

## 3.1 Source code

All our code developed in the project can be found in the following Github repository, `https://github.com/jensjpedersen/Projects_FYS-STK4155/tree/main/Project1`. Read the README.md for information on the content of the different python files and how to use and reproduce the results presented in this report.

## 3.2 Datasets

Our data was generated from a random uniform distribution on the interval [0, 1] using the numpy's random module. Two arrays was generated at random, with each containing n=20 data points. These arrays, x and y was combined in two mesh grids, X and Y each of size n x n. Thus, our total dataset consisted of 400 data points. Before any fitting was done on the data the mesh grids was reshaped into one dimensional arrays.

## 3.3 Scaling of Data

If our predictors represent different scales, then it is important to standardize the design matrix. And as the x and y data is on the same interval $[0, 1]$ both for the Franke and Terrain data, we do not need to scale the data. That is, the magnitude of our best fit polynomial coefficients lives on the same scale independent of features corresponding x and y data.

### 3.3.1 Franke Function

The Franke Function was used to test and validate our models. The Franke function is a waited sum of four exponentials and takes two arguments as input:

$$f(x,y) = \frac{3}{4}exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \tag{17}$$

$$+\frac{1}{2}exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}exp(-(9x-4)^2 - (9y-7)^2) \tag{18}$$

This function is widely used for testing of different fitting algorithms. Before we applied our fitting algorithms on the terrain data, we used this function to test and validate that our own developed algorithms worked properly.

Random normal distributed noise with an amplitude of 0.2, zero mean and a standard deviation of 1 was added to the data generated from the Franke function. Our data was split in train and test data, with 20% preserved for testing purposes, which is a common train/test split. Skliearms train_test_split function was used for splitting of the data. In order to keep the dataset consistent between different runs, a random seed generator was used. Our training data was fitted to polynomials up to degree 12 in x and y.

The following step by step guide can be used to generate a similar dataset:

1. Generate two arrays x and y, each with uniform distributed values in the interval [0, 1] with N = 20 data points.
2. Create a mesh grid of x values (X) and y values (Y), both with shapes $N \times N$
3. Reshape both mesh grids to one-dimensional arrays.
4. Generate your franke data: $y_{\text{data}} = f(X, Y) + 0.2 \cdot \mathcal{N}(0, 1)$, where f is equation 17
5. Generate your design matrix ($X_{\text{data}}$) with shape $N^2 \times l(p_{\text{max}})$, where $l$ is given by equation 19 and $p_{\text{max}} = 12$ is the maximum polynomial degree. For further details on how the generate the design matrix see section 3.5.
6. $X_{\text{data}}$ and $y_{\text{data}}$ is then split into training:

   $$X_{\text{train}}^{220 \times xl(p_{max})}, y_{\text{train}}{}^{220\times 1}$$

   and test data:

   $$X_{\text{test}}^{80 \times l(p_{max})}, y_{\text{test}}{}^{80\times 1}.$$

   , where a test size of 20% of our data was used.

### 3.3.2 Terrain data

The data is from the Oslo area. The specific radar map we are looking at can be downloaded from https://earthexplorer.usgs.gov/scene/metadata/full/5e83a3ee1af480c5/SRTM1N59E010V3/. If the link is outdated, the information of the image location can be found in table 1. It's from the SRTM mission in February of 2000 with entity ID SRTM1N59E010V3. The data has a resolution of one arc-sec, which gives each pixel a resolution of 30m. This is given as a .tif file from the website and we use imageio's v2.imread to read the image. But as this dataset is too big, 1801x3601 pixels, for what computation power we have available there is two ways we could go forward. Either we start by rescaling the dataset and therefore lose information or we create a model on a smaller part of the data. We have chose the latter and will look at a slice in the top left corner of 30x30 datapoints. Meaning we will look at an area of 900 square meters. We will do a polynomial fit on this using all the methods described in the theory section. The selected terrain slice is shown in figure 1.

Two meshgrids X and Y of latitudinal and longitudinal coordinates relative to the upper left corner was used to fit the data. X and Y has equally spaced values in range [0,1], with 30 data-points in each grid direction. This means that a longitudinal/latitudinal length of 0.1 in our model corresponds to 3m in real coordinates. The data was split in train and test data with a test size 0.2%.

Table 1: Corner positions of Oslo radar topography data taken in the SRTM mission in the year 2000.

| Corner | Position |
|---|---|
| NW | Lat 60°00'00"N, Long 10°00'00"E |
| NE | Lat 60°00'00"N, Long 11°00'00"E |
| SE | Lat 59°00'00"N, Long 11°00'00"E |
| SW | Lat 59°00'00"N, Long 10°00'00"E |

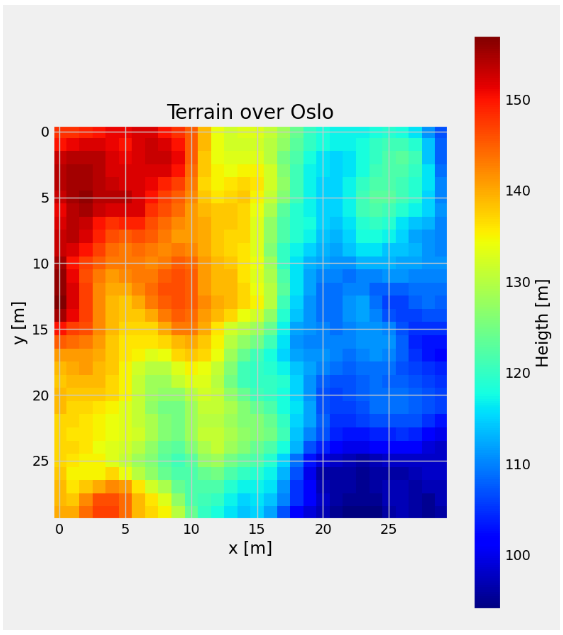


Figure 1: Color map of terrain data used for model fitting. The x and y coordinates are latitudinal and longitudinal in meters relative to the upper left corner.

### 3.4 Error estimation

To assess the accuracy of our models we will use the mean squared error

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

where n is the number of datapoints. This measures the difference between the model value and the actual value and will be taken from sklearn.metrics. And we will use the score function

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

which is used to score the model and we will take this from sklearn's LinearRegression().fit.score.

## 3.5 Design matrix

Our training data was fitted to polynomials up to degree 12 in x and y for the Franke function and data, and up to degree 30 for the terrain data. Thus, a design matrix of the form:

$$X = \begin{bmatrix} 1 & x_0 & y_0 & x_0^2 & x_0y_0 & y^2 & \dots & y_0^p \\ 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y^2 & \dots & y_1^p \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y^2 & \dots & y_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1}y_{n-1} & y_{n-1}^2 & \dots & y_{n-1}^p, \end{bmatrix}$$

was used for OLS-, Ridge- and Lasso regression. To generate our design matrix we used the function

```
FUNCTION create_X(x, y, n)
    X = array(size p, size n)
    FOR i = 1 TO i = p
        q = int((i)*(i+1)/2)
        FOR k = 0 TO k = i
            X[:,q+k] = (x**(i-k))*(y**k)
        ENDFOR
    ENDFOR
    return X
ENDFUNCTION
```

This function is based on the lecture notes of Morten Hjorth-Jensen [1]. Here p is the polynomial degree and X is the Deign matrix. In order to reduce the number of computations, we generated one Design matrix with number of features corresponding to the maximum polynomial degree $p_{max} = 12$. This matrix has l features including the intercept given by the equation:

$$l(p) = int(((p+1)*(p+2)/2)) \tag{19}$$

In order to fit the lower order polynomials this matrix was sliced in the following way

$$X_{\text{train}}[:, l(p)].$$

## 3.6 Regression methods

Since sklearn does not implement the use of psudo-inverse, which is a SVD based inverse we will write our own regression methods and use numpy's psudo-inverse. This is because otherwise we get a limitation on the polynomial degree as the determinant of the Hessian matrix of the training set will go to zero, meaning that the matrix will be singular and not invertible. We show this in table 2, where we calculated the determinant of the Hessian matrix with respect to the training data, with $N_{\text{train}} = 160$ rows, for different polynomial degrees (p).

### 3.6.1 OLS

We first tried a simple Ordinary Least Square model for fitting the train data. The optimal values for beta (our polynomial coefficient's) with OLS regression $\hat{\boldsymbol{\beta}}_{OLS}$ was calculated from equation (4). We compared our own OLS with Sklearn's method (sklearn.linear_model.LinearRegression). For the same training data, without any re-sampling. Both methods agreed up to polynomial fits of degree nine. For polynomials of degree larger than 9, the two methods was no longer in

Table 2: Determinant of $(X_{train}^T X_{train})$ with respect to polynomial degree, using N=???????

| Polynomial degree | $det(X_{train}^T X_{train})$ |
|---|---|
| 1 | 208035.65 |
| 2 | 891473.68 |
| 3 | 99.34 |
| 4 | $7.91 \cdot 10^{-10}$ |
| 5 | $8.58 \cdot 10^{-31}$ |
| 6 | $1.92 \cdot 10^{-64}$ |
| 7 | $1.13 \cdot 10^{-113}$ |
| 8 | $5.28 \cdot 10^{-181}$ |
| 9 | $2.21 \cdot 10^{-269}$ |
| 10 | 0.0 |
| 11 | 0.0 |
| 12 | 0.0 |

agreement on the Mean Squared Error (see figure 2). This probably due to the fact that sklearn's OLS method not uses the pseudo inverse as discussed in section 3.6

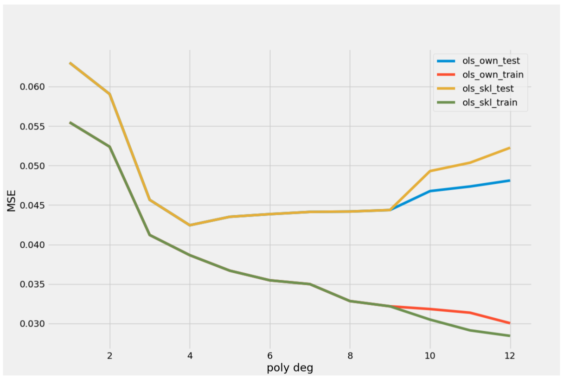


Figure 2: Comparison MSE produced from our own OLS method and sklearn's, predicted on the test and train data obtained from the Franke function. The training data was used fro finding the optimal model with OLS

### 3.6.2 Ridge

In the case of Ridge regression we introduced an array of regularization parameters $\lambda = [10^{-6}, 10^{-5}, \ldots, 10^{1}]$. The optimal values of beta with ridge regression $\hat{\boldsymbol{\beta}}_{Ridge}$ was calculated from equation (7). For implementing equation we used bumpy.eye as the identity matrix. We compared our own implementation of Ridge to Sklearn's (`sklearn.linear_model.Ridge`). Both methods predicted produced the exact same best fit models.

### 3.6.3 Lasso

The same regularization parameters as used for Ridge regression was used to find the optimal values for beta with Lasso regression, $\hat{\boldsymbol{\beta}}_{Lasso}$. We did not develop our own algorithm for Lasso

Table 3: Mean MSE scores with standard deviation calculated from 50 independent bootstrap runs. The minimum MSE scores obtained in the Polynomial degree range was used to estimate the statistics.

| Regression method | Data | Polynomial degree range | $\lambda$ | mean MSE | std MS |
|---|---|---|---|---|---|
| OLS | Franke Data | 1-12 | | 0.0443 | 0.0001 |
| OLS | Terrin Data | 1-30 | | 3.0 | 0.2 |

regression. Insted we used the function provided by the sklearn's python module.

## 3.7 Resampling

### 3.7.1 Bootstrap

Bootstrap re-sampling was done on the training data with $N_{train} = 320$ data points for the Franke data and $N_{train} = 720$ data points for the terrain data.

Our bootstrap algorithm used the re-sample dunction from learn, `sklearn.utils.resample`. This function was used to draw $N_{train}$ re-samples from our training (data with replacement), $N_{boots}$ times. For each re-sample our best fit model for the given regression method, polynomial degree and regularization parameters was found. Our predicted values from our test data ($\boldsymbol{X}_{train}$) was stored in an matrix of size $N_{test} \times N_{boots}$. Different scores (MSE, Bias, Variacne, R2) was calculated in this matrix.

Bootstrap was used to estimate sample statistics from our dataset. $N_{boots} = 100$ bootstrap iterations was used to predict sample statistics on the Franke data. Only $N_{boots} = 20$ iterations was done on the terrain data due to limited computing capacity. On order to get an estimate in precision on our sample statistics on our dataset the bootstrap algorithm was run 50 times. The standard deviation of the 50 best MSE values was calculated. The results are summarized in table 3. The train and test datasets was kept the same under all bootstrap runs.

### 3.7.2 K-fold Cross Validation

K-fold re-sampling was used to get an idea of how representative out train/test split was to the whole dataset. K-fold re-sampling was done on the whole dataset ($\boldsymbol{X}_{data}$ and $\boldsymbol{y}_{data}$). Before our was split , all the data was randomly shuffled. The dataset was the split into $N_{splits} = 10$. MSE was calculated on the predicted value on each test split. We tested our own implementation of the K-fold algorithm against python's sklearn function, `sklearn.model_selection.KFold`. On the Franke data skelarn's method produced similar results as for bootstrap with OLS regression (see Figure 9 and 8). This was not the case for our own implementation. Our own implementation produced at least four times as large values compared to sklearn's implementation. We decided to used sklearn's implementation for our K-fold analysis.

The python code for our own implementation of the K-fold algorithm is listed below:

```python
# The size of the test fold (n_test_data) is calculated
n_data = len(y_data)
n_test_data = int(np.ceil(len(y_data)/n_splits))

# Data is shuffled by creating a shuffeld array of indices.
idx_shuffled = np.arange(n_data)
np.random.shuffle(idx_shuffled)
```

```
        mse_kfold_deg = []
        for i in range(n_splits):
            test_idx = idx_shuffled[n_test_data*i:n_test_data*(i+1)]
            train_idx = np.delete(idx_shuffled, np.s_[n_test_data*i:n_test_data*(i
    +1)])

            # Get train and test split
            X_kfold_train = X_data[train_idx,:]
            y_kfold_train = y_data[train_idx]
            X_kfold_test = X_data[test_idx,:]
            y_kfold_test = y_data[test_idx]

            # Use regression_method to predict on test split
            y_kfold_pred = self.regression_method(regression_method, X_kfold_train
    , y_kfold_train, X_kfold_test, lamb = self.lamb)

            # Claculate MSE score for predicted data.
            mse = np.mean((y_kfold_test[:,np.newaxis] - y_kfold_pred[:,np.newaxis
    ])**2)
            mse_kfold_deg.append(mse)

        # Return mean of MSE predicted on all test splits.
        return np.mean(mse_kfold_deg)
```

We did not mange to find out why our own implementations and sklearn's differed. One possible problem with our own method is that the case where the size of the dataset is not divisible by number of splits is not handled properly. However, this should not be a problem since we used 10 splits in all our analysis.

## 4 Results and Discussion

### 4.1 Franke Data

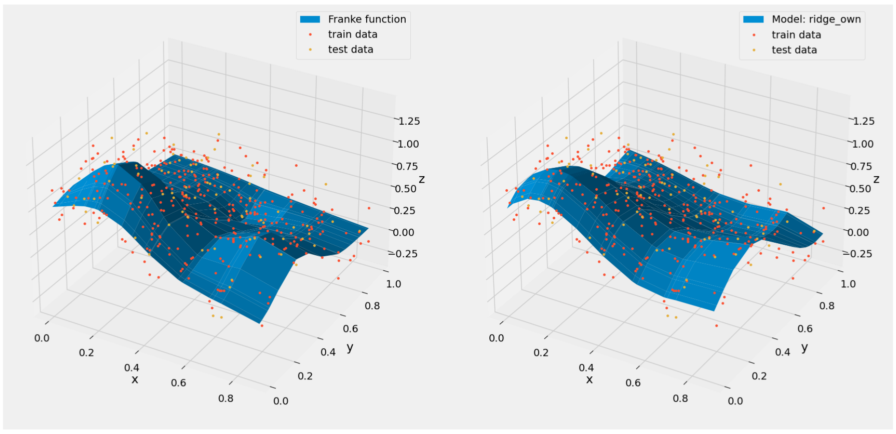
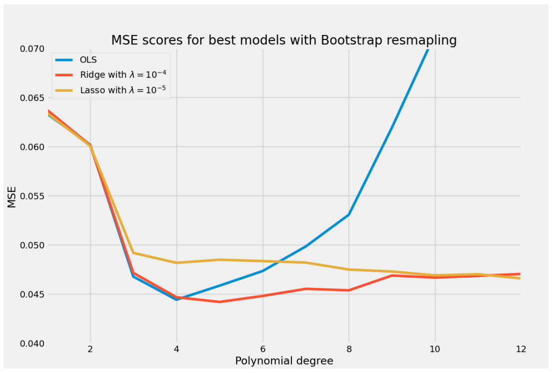
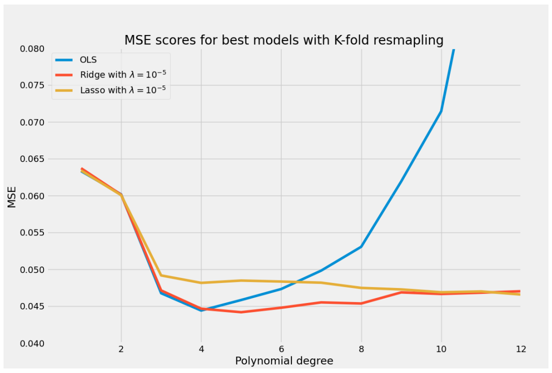


Figure 3: Left figures shows the frank function plotted as a function of x and in y coordinates the intervall [0,1]. The red and yellow dots shows or train- and test-data with added normal distributed noise, $0.2 \cdot \sigma(0, 1)$. The lowest MSE, calculated with bootstrap re-sampling was obtained with Ridge regression with a regularization parameter $\lambda = 10^{-5}$ and a polynomial degree of 5. Right figure shows the best fit model. Train- and test-data is plotted as for the left figure.



(a) Mean MSE score for 100 bootstrap re-samples. MSE scores is plotted as a function of polynomial degree, for the best fit models obtained with OLS, Ridge ($\lambda = 10^{-j}$) and Lasso ($\lambda = 10^{-5}$) regression.



(b) Mean of MSE scores calculated on all ten test folds with K-fold re-sampling. Our best fit models from OLS, Ridge ($\lambda = 10^{5}$) and Lasso ($10^{-5}$) regression is plotted as a function of polynomial degree. $\lambda$ is the regularization parameter.

In figure 5 we see the mean squared error and R2 score on the Franke function for a polynomial up to degree five. The Predicted MSE scores with respect to polynomial degree was predicted on both the training aand test data. As expected out model gets better as the polynomial degree increases. That is, our MSE score decreases and R2 score increases. For polynomial degree

Table 4: Table of Best mean squared error (MSE) scores obtained with different re-sampling and regression methods. The training- and testing-data splits from the Franke function was used for model fitting and prediction respectively. $\lambda$ is the best choice of regularization parameters for Ridge and Lasso regression. n refers to number of re-samples/splits used in Bootstrap and K-fold.

| Polynomial degree | $\lambda$ | Regression method | Re-sampling method | MSE |
|---|---|---|---|---|
| 4 | | OLS | Bootstrap (n=100) | 0.04440 |
| 5 | $10^{-4}$ | Ridge | Bootstrap (n=100) | 0.04425 |
| 11 | $10^{-5}$ | Lasso | Bootstrap (n=100) | 0.04673 |
| 5 | | OLS | K-fold (n=10) | 0.04181 |
| 6 | $10^{-5}$ | Ridge | K-fold (n=10) | 0.04155 |
| 12 | $10^{-5}$ | Lasso | K-fold (n=10) | 0.04548 |

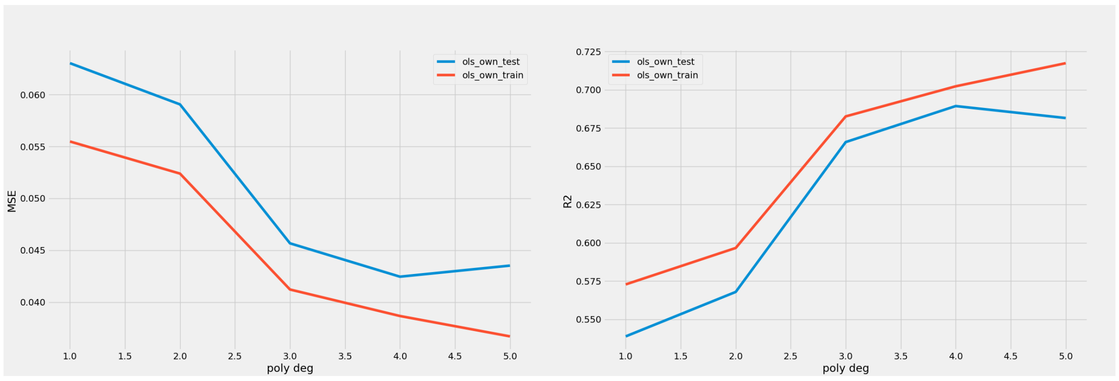


Figure 5: Mean squared error and score function for the Ordinary Least Squares (OLS) method on the Franke function for training and test data with a test/train split of 0.2

5, the MES score increase for the test data and decreases for the training data. The opposite behavior is seen for the R2 score. This is an expected behavior. For increases polynomial degree we should expect that predictions on the training data gets better and betters, since our model is trained on the exact same data. Thus, for the prediction on training data we expect the MES to approach zero as the polynomial degree increases. The same behaviour can not be expected when we predict MES and R2 score on the test data. Our model has never seen the test data. We expect that the MES score predicted on the test data decreases to a certain polynomial degree. Afterward it will increase, when we have reached over fitting. That is, our model is fitted to well to the training data, and is no longer generalizable to other similar distributed datasets. This is the exact behavior we observe in figure 5. However, the effect if over fit is not presented very well cause of the low polynomial degree.

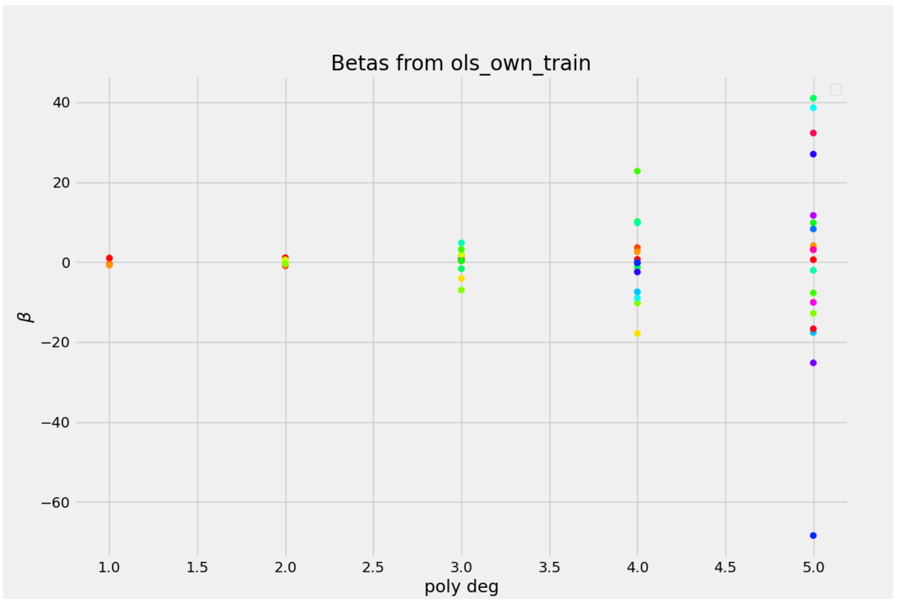
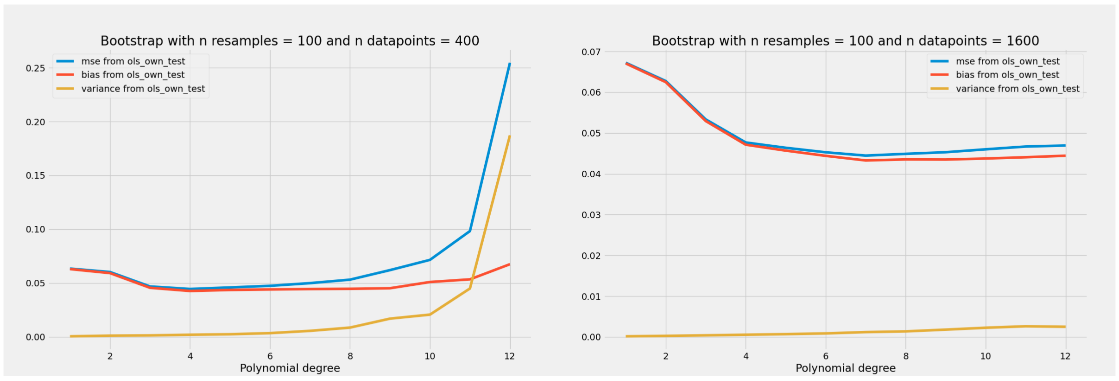


Figure 6: $\beta_i$ values plotted with respect polynomial degree for our best fit model with OLS on the training data, $\boldsymbol{X}_{train}$ The colors represent specific $\beta_i$, in order to make it possible to track specific $\beta_i$ for different polynomial degrees.

In figure 6 we see for each polynomial degree the beta parameters $\beta_i$ as a dot in the plot. This is for the OLS model on the training data from the Franke function. We see that as the polynomial degree increases, the $\beta$ values deviate more and more from zero. This can be a result of over fitting since we do not have any regularization parameter on the OLS model.

We expect the different $\beta_i$ does not vary significantly with increasing polynomial degree. It's not easy to say if this is the case from inspection of the plot, due to a limited number of colors and large number of $\beta_i$. It seems to be the case for the lower $\beta_i$ terms. It does not make intuitively sense if a certain $\beta_i$ varies a lot for different polynomial degree's. The less complex model (lower poly deg) should to a certain degree reflect the values in a more complex model. By introducing higher polynomials and hence more $\beta_i$'s, our model can represent more and more complex patterns (more variations in shorter length scale). However we expect the lower oerder $\beta_i$ to be more or less the same. Because they represent the less complex parts of our model.

In figure 7 we can see the bootstrap used on the OLS method on the Franke function for 100



(a) 400 datapoints (b) 1600 datapoints

Figure 7: Bootstrap with 100 resamples for the OLS method on the Franke function on the test set.

resamples and two different number of datapoints. We see that for the lower number of points we get signs of overfitting, while we do not see this behaviour at these polynomial degrees for 1600 datapoints. We see that for 400 datapoints the optimal bias-variance tradeoff happens at a polynomial degree of around 4 and for 1600 datapoints at around polynomial degree 7.

This means that our best fit model is depend the size of the dataset. For larger dataset sizes the mean value of our data points will resemble more and more the real distribution. This is also the case for our training data and test data. To achieve over fit the complexity of the model needs to be very large.

In figure 8 the MSE predicted on the training data with OLS and bootstrap resampling is plotted as function of polynomial degree. Here we see a similar trend as for the predictions with bootstrap on the test data. If we had trained the model on the real training data, we would expect that the predictions of MSE on the training data approaches zero. Our predicted model should be able to 100% replicate our data, since the model is trained the exact same noise. In this scenario it does not make much sense to talk about over fitting. Cause we need two independent datasets to predict the generalizability of our model.

However in our case we did re sample the training dataset (bootstrap with 100 resamples) before we trained our model. Thus, our re sampled dataset should follow the same distribution, but each individual data points would vary from or original training data. As long as the number of data points in our original training data is sufficiently large. That is, the expected value of our noisy model for the Franke function approaches the Franke funciton without noise. We expect that prediction on re sampled data VS. a separate test split would yield similar results. By comparison of figure 7 (left) and 8, this is what we observe.

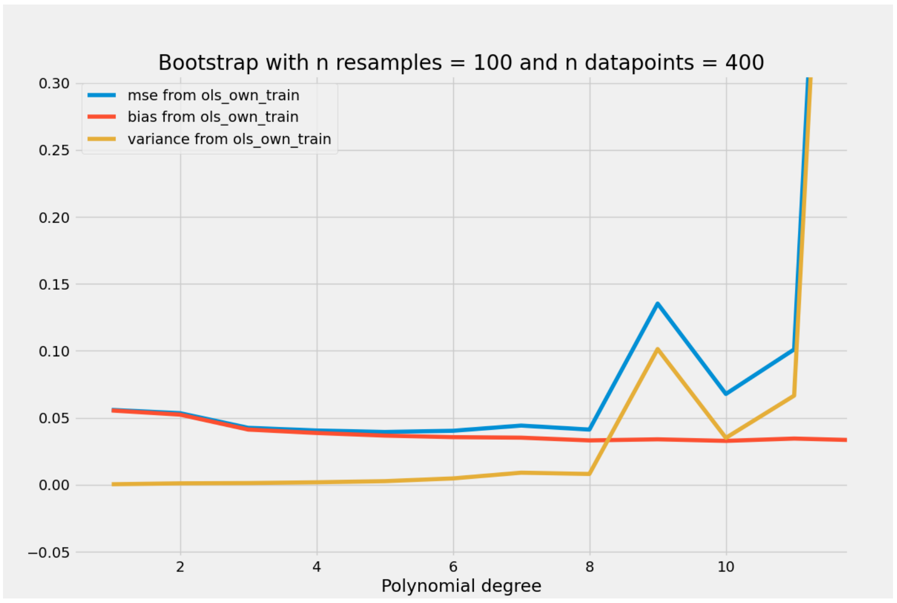


Figure 8: Bootstrap with 100 resamples for the OLS method on the Franke function on the training set with 400 datapoints.

Until now we have discussed our models in terms of MSE. However MSE is clolsy related to bias and variance (see eq. (9)). The variance describes the precision on our predicted models. As our model complexity increases, it will be able to pick up complex patterns in our training data. If we use another training dataset this pattern may not be contained in the data. Thus our complex model fit will give different results. If our models gets to complex it will describe patters that are unique to the training data and not the representative distribution. This is called over fitting and can be observed when the variance term rapidly increases with model complexity. We previous saw that the MSE rapidly increases after a certain polynomial. The reason is that the variance of the model increses, as seen in figure 7 b and 8.

The bias term is expected to go to zero as the model complexity increases. That is our expected value of our models is approaching the real function. In our results the bias term is decreasing (see fig 7 b and 8), as expected. But after a certain polynomial degree the bias term is slightly increseing. This is somewhat unexpected. However this may be due to the fact that the variance is increseing and it's harder to obtain a good statistic on the real expectation value of our model. This may have been resolved by increasing the number of bootstrap re samples. It's also worth metntioning that we did not used the exact function f(x) to esitamte the bias term. We used out test data, $\boldsymbol{y}_{test}$. That is our exact function (Franke function) plus added normal distributed noise.

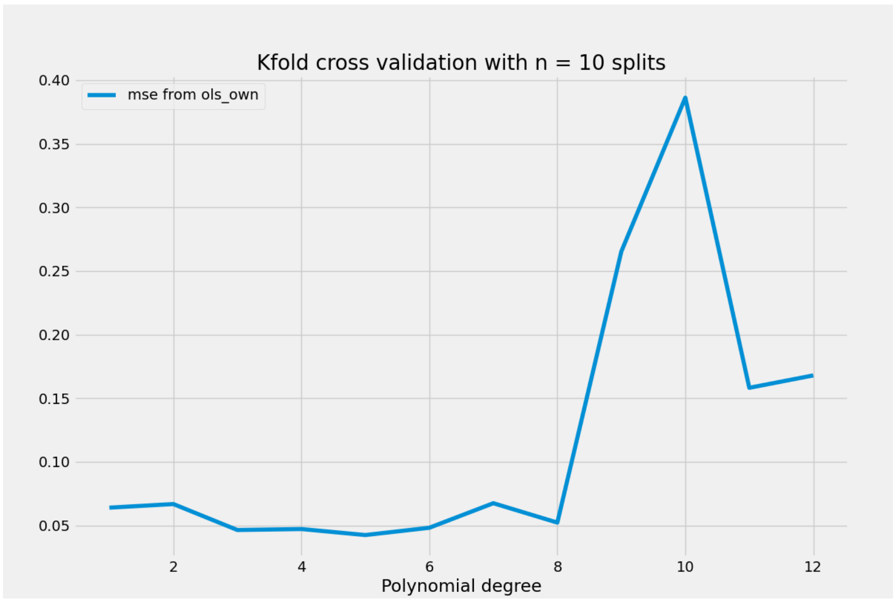


Figure 9: Mean squared error using cross-Validation with 10 splits on the Franke function.

In figure 9 we can see the MSE of Cross-validation with 10 splits on the Franke function using OLS. This is on the test data. We see that polynomial degree over 8 produces weird results. This can be because of over fitting. K-fold cross validation produces similar results as for bootstrap up to degree 8. Then the MES predicted for k-fold rapidly increases for polynomial degree 9. For bootstrap this increases is seen for polynomial 12. We used bootstrap to infer statistics on our data. And we re sampled the training data. Thus statistic inferred from our bootstrap re sampling is representative to how our training data is distributed. If our test and training data has different distributions, or the number of data points is to small to estimate the real expectation value of our data. Then, K-fold and Bootstrap is expected to disagree on statistics. K-fold divides the dataset in splits and tests/trains on all the splits. In this way we can be sure that our scores is not biased towards our particular train/test split. Our results from bootstrap and K-fold largely argreess with each other. This means that our original chosen train/test split is representative four our dataset. Thus, if we had done a new train test split, it's expected that the results would have been similar. Hence, the precision of our generalized model is good.

The minimum MSE from Ridge regression with bootstrap re sampling was found for polynomial degree of 6 with the hyper parameter, $\lambda = 0.001$. This is seen in figure 10. There is no sign of over fitting in this figure. Both the variance, bias and hence MSE are approximate constant for polynomial 3-12. This is because we have introduced a regularization parameter that's sets an upper bound on the second norm of the betas. For OLS when we increase the polynomial degree, the number of linearly depended columns in our design matrix increases. This causes high variance in betas's (see eq. 6), and hence a high variance in our model. This is not seen in case of Ridge regression.

In Figure 11 the predicted MSE with bootstrap re sampling on the test data as function of polynomial degree and different $\lambda$ is plotted in a heat map. We observe that the variation in MSE decreases with increasing $\lambda$. Thus, high $\lambda's$ tends to shrink the $\beta's$ to zero.

In Figure 12 the same heat map as for 11 is plotted, but instead with K-fold re sampling.

Again we observe similar values as for bootstrap. Thus, the statistics inferred with bootstrap re sampling is generalizable to the whole dataset.

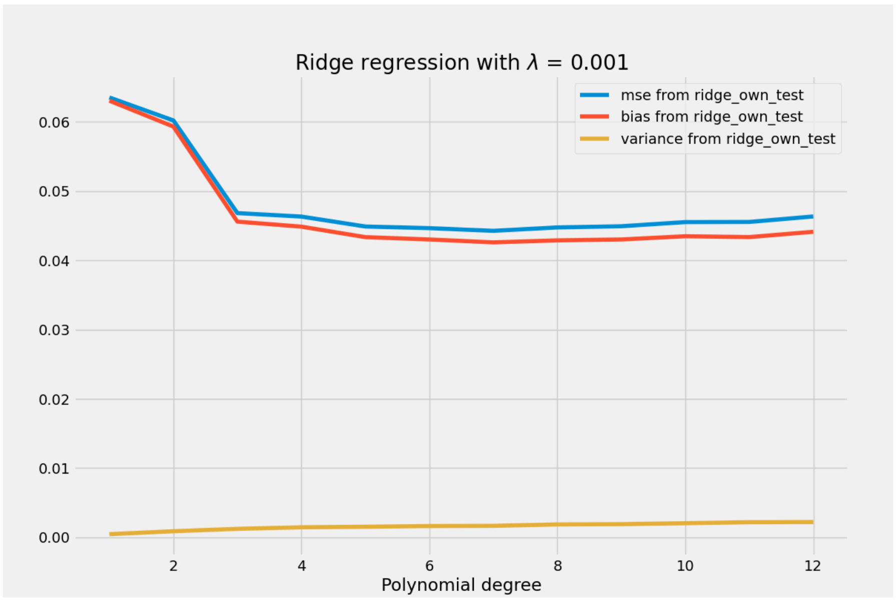


Figure 10: Ridge resgression on the Franke function.

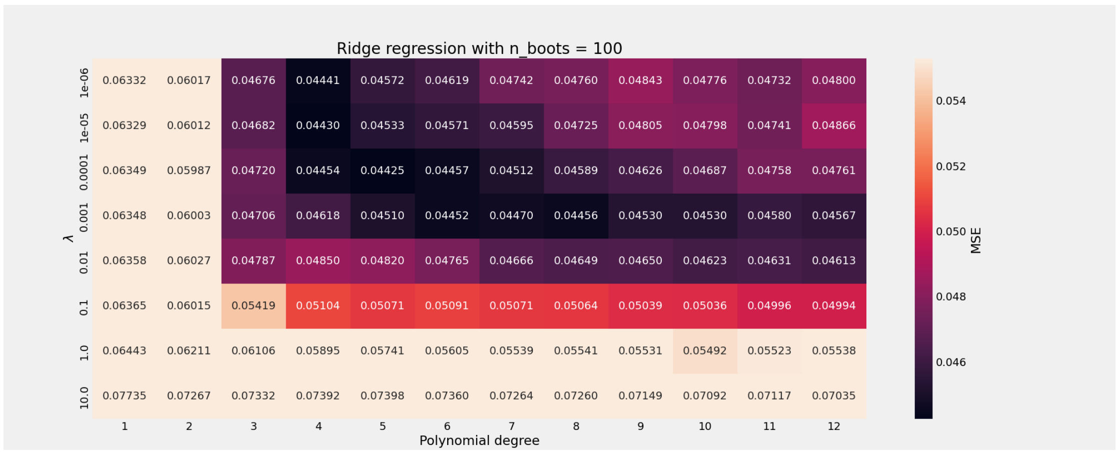


Figure 11: Correlation between MSE, the hyperparameter $\lambda$ and polyonmial degree for bootstrap on Ridge for on the Franke function.

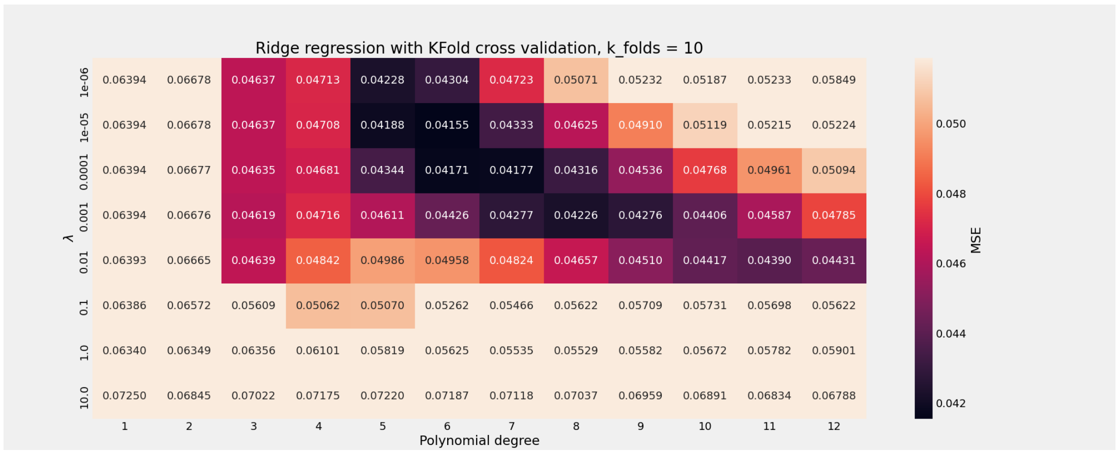


Figure 12: Correlation between MSE, the hyperparameter $\lambda$ and polyonmial degree for Cross-Valdidation for Ridge regression on the Franke function.

In Figure 13 the MSE predicted on the test data with Lasso regression and Bootstrap re sampling is plotted as function of polynomial degree with respect different choices of the regularization parameter $\lambda$. Form this heatmap we can observe that decreasing with increasing polynomial degree for lower order polynomials and small values for $\lambda$. That is, the bias of our model is decreasing. However for higher values of $\lambda$ the MSE score is more or less constant with respect to polynomial degree. Regularization parameters larger than 0.01 zeros out the beta values for our model. We observer that the best values for MSE is obtained for polynomial degree 12. Lasso regression gives similar results as for Ridge regression, with slightly higher MSE scores. The best fit model model with ridge regression is observed for lower polynomial degree's (range 4-8).

In Figure 14 the MSE predicted on the test data with Lasso regression and K-fold cross validation is plotted as function of polynomial degree with different values of $\lambda$. Again the magnitude of MSE scores largely agrees with those obtained with Bootstrap. Generally MSE scores is slightly larger than for Bootstrap. This is however not case for polynomial degree 11-12 where the K-fold methods gives us lower MSE scores compared Bootstrap.

In table 4 our best fit models for the different re sampling and regression methods are summarized. As observed from the heat maps, Lasso regression tend to decrease with increasing polynomial degree and gives us the lowest MSE score for polynomial degree 11 and 12 with Bootstrap and K-fold cross validation respectively. Ride and OLS regression gives us the best fit model in the polynomial degree range of 4-6. Lasso regression performs the worst it terms of MSE. OLS and Ridge gives similar MSE scores for their best fit models. However Ridge slightly outperforms OLS, for K-fold. However the opposite is true for Bootstrap re sampling. The disadvantage with OLS compared to Ridge is that it is very prone to over fitting.

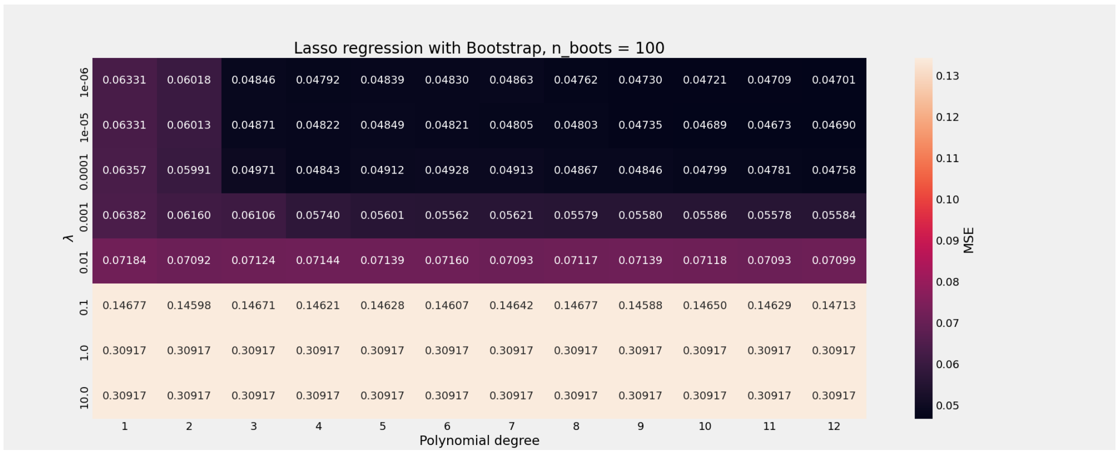


Figure 13: Correlation between MSE, the hyperparameter $\lambda$ and polyonmial degree for Cross-Valdidation for Lasso regression on the Franke function.

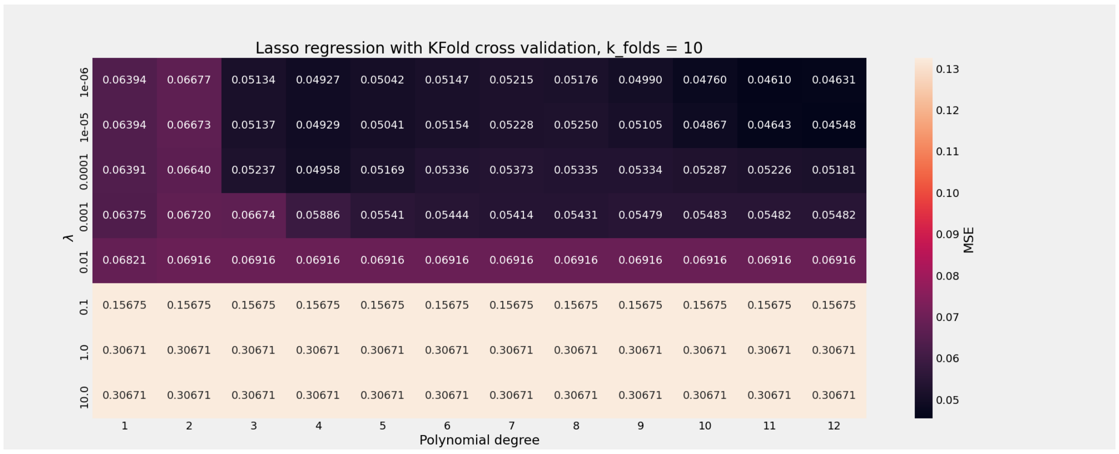


Figure 14: Correlation between MSE, the hyperparameter $\lambda$ and polyonmial degree for Cross-Valdidation for Lasso regression on the Franke function.

## 4.2 Terrain data

The best MSE values obtained with different regression and re sampling methods is listed in table 5. Figure 15 shows our dataset (left figures) compared to our best fit model with OLS (right figure).

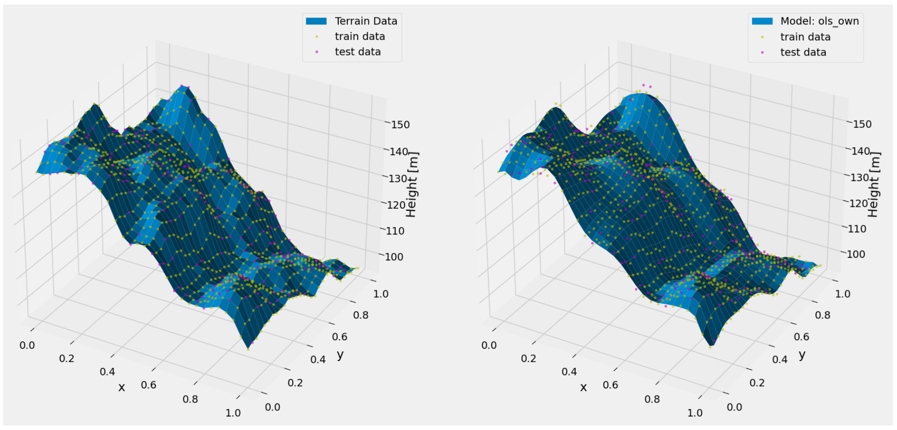
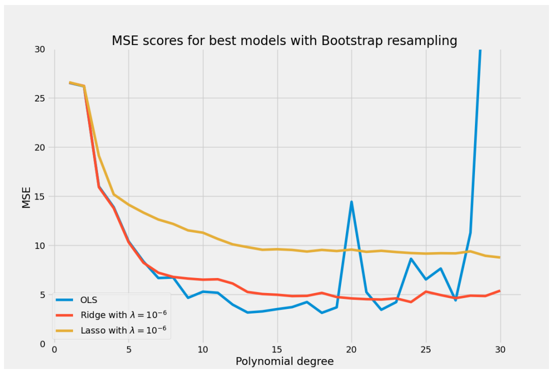
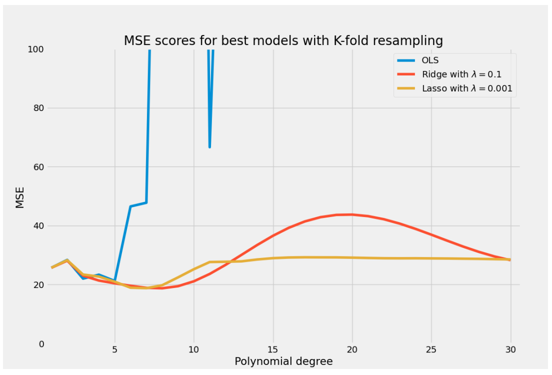


Figure 15: The left figure shows the surface of our sliced terrain data. Our train and test-data are represented by yellow and purple dots respectively. Hight above sea level in meters is plotted on the z axis. The x- and y-axis is takes values in intervall [0, 1]. This interval scales to 30m in real coordinates. The left figure shows our best fit model obtained with OLS for polynomial degree 18. Again the training- and test-data is plotted as in the left figure.



(a) Mean of MSE scores calculated with bootstrap on the terrain data. $n_{\text{boots}} = 20$ re-samples was used. The mean MSE predicted on the test-data is plotted as function of polynomial degree for OLS-, Ridge- ($\lambda = 10^6$) and Lasso-regression ($\lambda = 10^{-6}$)



(b) Mean of MSE predicted on each test fold with use of the K-fold re-sampling technique. A total of ten folds was used. The MSE score is plotted as a function of polynomial degree for OLS-, Ridge- ($\lambda = 0.1$) and Lasso-regression ($\lambda = 10^{-4}$)

Figure 16: Comparison of OLS, Ridge and Lasso regression

In Figure 16 the MSE on the terrain test data is predicted with for the best fit regression methods with Bootstrap and K-fold cross validation. The MSE socre obtined with OLS and bootstrap reaches a minima for polynomial degree 18. This is not the case for K-fold cross validation, where the MSE explodes for polynomial degree 5 and higher. The bootstrap re sampling was done on the selected training data. If we had chosen a new training data split, we would expect a different result for our MSE score. This is what we show with the K-fold technique. The terrain is to complex, with to few data points to create a generalizable model of the terrain with OLS regression. For our optimal values of regularization parameters with ridge and lasso regression, we observe a trend of decreasing MSE scores with increasing polynomial degree. For ridge regression and lasso regression we got a minima for polynomial degree 18 and 30 respectively. Both gives scores that are higher than with OLS. But we dont see the increase in variance as for OLS regression. The MSE score for Lasso and Ridge is almost constant from polynomial degree 14-30. The reason is that, our penalty term kills some of the higer order beta terms (highly correlated features) that causes the unstable behaviour seen with OLS. We observe that Lasso produces a higer MSE vale than with Ridge regression. This is probably due to the fact that Lasso zeros out the highly correlated features compared to Ridge that shrinks the beta's.

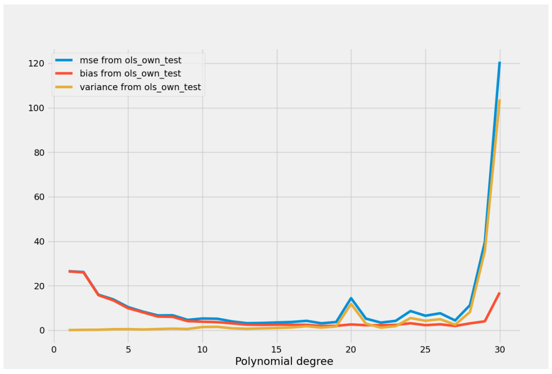
The best regularization parameters with Ridge regression and bootstrap was found from inspection of Figure 18. Here all the MSE scores predicted on our training data is plotted from polynomial degree 10-30 and regularization parameters. We observer that all MSE scores tend to increase for higher values of $\lambda$. That is, the complexity of our model is reduced for higher values of $\lambda$

In figure 19 the same heat map is plotted, but with K-fold cross validation instead of bootstrap. Here we observe almost the opposite trend than for bootstrap re sampling. The MSE score tends to explode for small values of $\lambda$ and with increasing polynomial degree. It indicates that complexity of data is not well represented by the higer order $\beta$'s. Thus, by intruding more dominant penalty term the complexity of our model is reduced and we don't get a problem with over fitting.

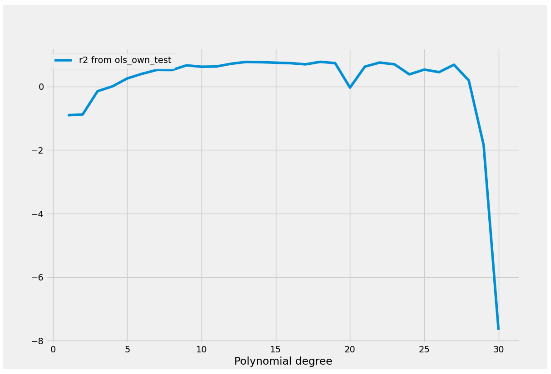
In Figure the heat map for Lasso regression with bootstrap re sampling is plotted. The heatmap is plotted from polynomial degree 10-30. However our best MSE score was found at polynomial degree 7 with $\lambda = 0.001$. The reason why this value is not shown in the heat map is to keep it consistent with the other heat maps for comparison reasons. Lasso regression yield higher values for the MSE than ridge regression. Generally the best scores for the different $\lambda$ is seen for polynomial 30. Also the standard deviation with respect to Polynomial degree is generally lower than for ridge. That is the penalty the correlated highly correlated features is higher than for ridge.

In figure 21 the heat map with Lasso regression and K-fold cross validation is plotted. Again, as for ridge regression we observe over fitting for the lower values of $\lambda$. The MSE scores increases from polynomial degree 17-30 for $\lambda = 10^{-6}$ and $\lambda = 10^{-5}$. For $\lambda = 0.1$ and $\lambda = 1.0$ the MES score is almost constant as function of polynomial degree. This is because the $\beta's$ is zeroed out. Lasso gives is less forgiving than Ridge which shrinks the parameters for $\beta$.

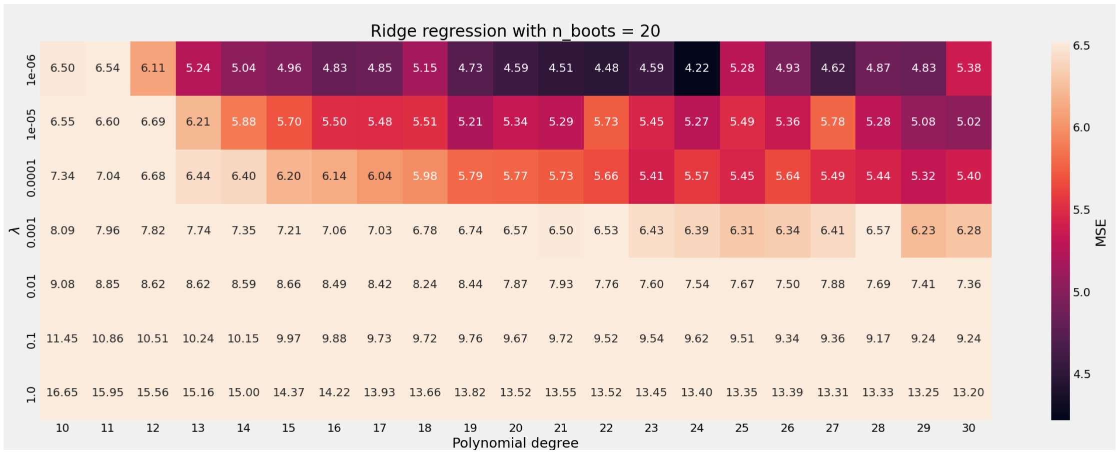
As explained in section 2 the MSE can be decomposed in a bias, variance and irreducible error term. This bias variance decompositions is shown in Figure 17b for OLS regression with bootstrap, predicted on our test data. Again as shown for the Franke data, we observe that Large increase on our MSE score for polynomial 30 is explained by the variance term. In figure **??** we observer the R2 score for the same model. The R2 score is highly related to the MSE and we observe the opposite trend. We chose not to include this bias variance plot for Ridge and Lasso regression. Because this does not provide us any more information than the MSE. We know that the reduction in MSE for low polynomial is due to the decrease in bias. And the increase for higer order polynomials is due to the increase in variance.



(a) Bias, Variancse and MSE obtained with OLS and bootstrap re-sampling. 20 re-samples was used to calculate the mean scores.



(b) R2 score obtained with OLS and bootstrap re-sampling. 20 re-samples was used to calculate the mean R2 score.



| λ \ Polynomial degree | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1e-06 | 6.50 | 6.54 | 6.11 | 5.24 | 5.04 | 4.96 | 4.83 | 4.85 | 5.15 | 4.73 | 4.59 | 4.51 | 4.48 | 4.59 | 4.22 | 5.28 | 4.93 | 4.62 | 4.87 | 4.83 | 5.38 |
| 1e-05 | 6.55 | 6.60 | 6.69 | 6.21 | 5.88 | 5.70 | 5.50 | 5.48 | 5.51 | 5.21 | 5.34 | 5.29 | 5.73 | 5.45 | 5.27 | 5.49 | 5.36 | 5.78 | 5.28 | 5.08 | 5.02 |
| 0.0001 | 7.34 | 7.04 | 6.68 | 6.44 | 6.40 | 6.20 | 6.14 | 6.04 | 5.98 | 5.79 | 5.77 | 5.73 | 5.66 | 5.41 | 5.57 | 5.45 | 5.64 | 5.49 | 5.44 | 5.32 | 5.40 |
| 0.001 | 8.09 | 7.96 | 7.82 | 7.74 | 7.35 | 7.21 | 7.06 | 7.03 | 6.78 | 6.74 | 6.57 | 6.50 | 6.53 | 6.43 | 6.39 | 6.31 | 6.34 | 6.41 | 6.57 | 6.23 | 6.28 |
| 0.01 | 9.08 | 8.85 | 8.62 | 8.62 | 8.59 | 8.66 | 8.49 | 8.42 | 8.24 | 8.44 | 7.87 | 7.93 | 7.76 | 7.60 | 7.54 | 7.67 | 7.50 | 7.88 | 7.69 | 7.41 | 7.36 |
| 0.1 | 11.45 | 10.86 | 10.51 | 10.24 | 10.15 | 9.97 | 9.88 | 9.73 | 9.72 | 9.76 | 9.67 | 9.72 | 9.52 | 9.54 | 9.62 | 9.51 | 9.34 | 9.36 | 9.17 | 9.24 | 9.24 |
| 1.0 | 16.65 | 15.95 | 15.56 | 15.16 | 15.00 | 14.37 | 14.22 | 13.93 | 13.66 | 13.82 | 13.52 | 13.55 | 13.52 | 13.45 | 13.40 | 13.35 | 13.39 | 13.31 | 13.33 | 13.25 | 13.20 |

Figure 18: Heatmap of MSE scores obtained with Ridge regression and Bootstrap re-sampling, with 20 bootstrap re-sampling iterations.

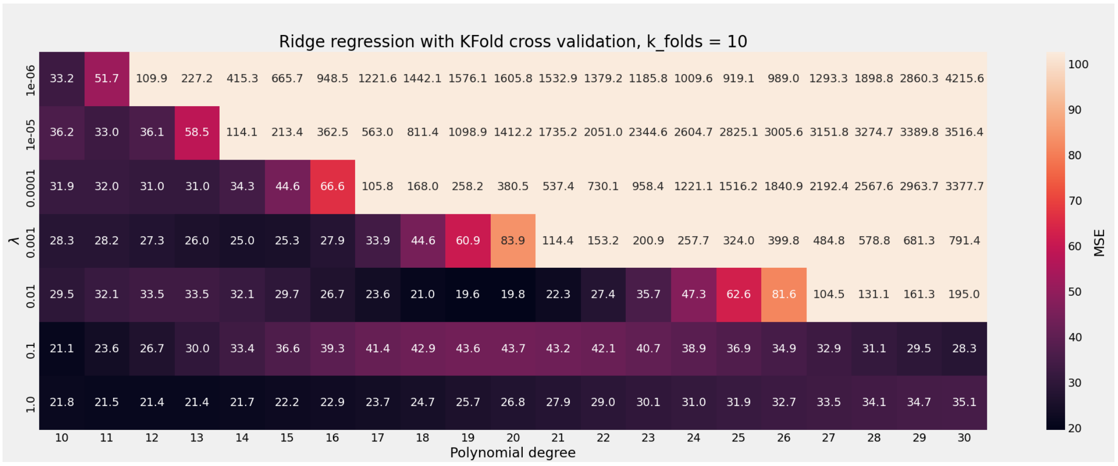


Figure 19: Heatmap of MSE scores obtained with Ridge regression and K-fold re-sampling, with 10 splits.

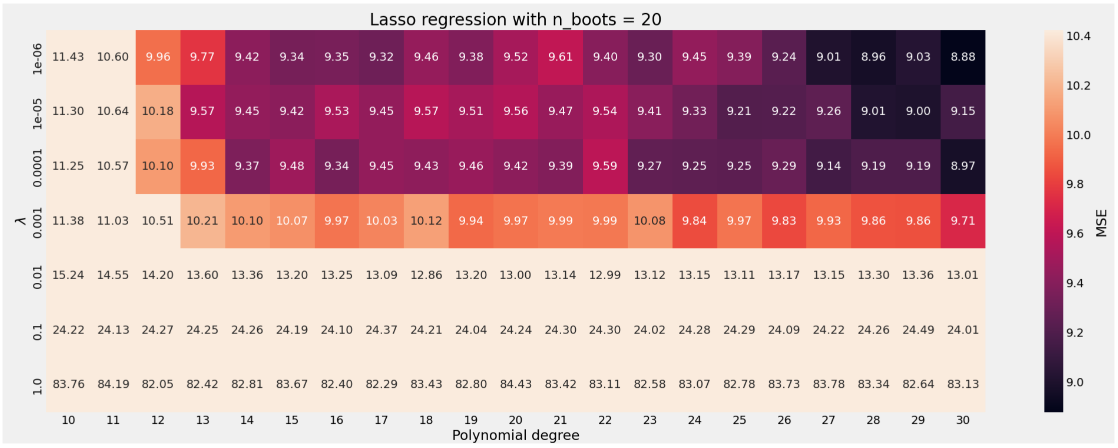


Figure 20: Heatmap of MSE scores obtained with lasso regression and Bootstrap re-sampling, with 20 re-samples.

Table 5: Table of Best mean squared error (MSE) scores obtained with different re-sampling and regression methods. $\lambda$ is the best choice of regularization parameters for Ridge and Lasso regression. n refers to number of re-samples/splits used in Bootstrap and K-fold.

| Polynomial degree | $\lambda$ | Regression method | Re-sampling method | MSE |
|---|---|---|---|---|
| 18 | | OLS | Bootstrap (n=20) | 3.12 |
| 24 | $10^{-6}$ | Ridge | Bootstrap (n=20) | 4.22 |
| 30 | $10^{-6}$ | Lasso | Bootstrap (n=20) | 8.88 |
| 5 | | OLS | K-fold (n=10) | 21.23 |
| 8 | 0.1 | Ridge | K-fold (n=10) | 18.74 |
| 7 | 0.001 | Lasso | K-fold (n=10) | 18.77 |

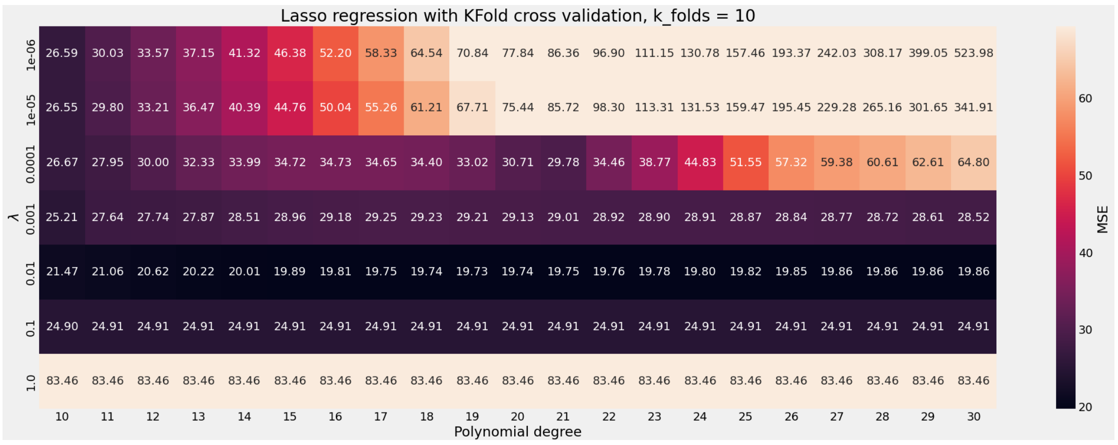


Figure 21: Heatmap of MSE scores obtained with lasso regression and K-fold re-sampling, with 10 splits.

In figure 18, 19, 21 and 21 we see the MSE for a number of polynomial degreees and $\lambda$'s on the terrain data function. We notice that for Lasso a bigger $\lambda$ tend to provide smaller MSE, while for Ridge it's the opposite. This means the OLS method and the Ridge method would produce similar results.

## 5 Conclusion

We found that using the Lasso and Ridge methods over the OLS on Franke data did not have much benefits on the MSE. But for the terrain data we got a much better with fit with the regularization parameter. This could be because the Franke function is a smooth and easy function to fit and the terrain data can have a much more complex shape. We also found that for the Franke function bootstrapping and k-fold resampling decreased the total error of the model up to certain polynomial degrees where the variance got exponentially higher because of overfitting. This is because when working with a smaller dataset, overfitting will happen sooner. For the Lasso and Ridge method we found that the $\beta$-parameters was smaller with Ridge than Lasso. This can be seen from the cost-function of each of the methods as the Lasso has the 1-norm and Ridge has the 2-norm. This makes Ridge regularize the $\beta$-parameters more.

## A Expectation values and variances for ordinary least squares

We use that the dependent variable

$$y_i = y(x_i) = \tilde{y}_i + \epsilon_i = \mathbf{X}_{i*}\boldsymbol{\beta} + \epsilon_i,$$

to get the expectation value

$$\mathbb{E}[y_i] = \mathbb{E}[\mathbf{X}_{i*}\boldsymbol{\beta} + \epsilon_i] = \mathbb{E}[\mathbf{X}_{i*}\boldsymbol{\beta}] + \mathbb{E}[\epsilon_i].$$

We now use that $\epsilon_i$ is a normal distributed error, meaning that $\mathbb{E}[\epsilon_i] = 0$. And we assume that $\mathbf{X}_{i*}\boldsymbol{\beta}$ are not stocastic variables, giving $\mathbb{E}[\mathbf{X}_{i*}\boldsymbol{\beta}] = \mathbf{X}_{i*}\boldsymbol{\beta}$. Then the expectaion value is

$$\mathbb{E}[y_i] = \mathbf{X}_{i*}\boldsymbol{\beta}. \tag{20}$$

We then calculate the variance

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

where we have taken the non stochastic variables out of the expectation value. We again use that the expectation value of a normal distributed variable is $\mathbb{E}[\epsilon_i] = 0$, and see that the variance of this variable is

$$\mathrm{Var}[\epsilon_i] = \mathbb{E}[\epsilon_i^2] - (\mathbb{E}[\epsilon_i])^2 = \mathbb{E}[\epsilon_i^2] = \sigma^2.$$

Using this we get

$$\mathrm{Var}[y_i] = \sigma^2. \tag{21}$$

The optimal parameter $\hat{\boldsymbol{\beta}}$ for ordinary least squares is given by

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

We then calculate the expectation value of the optimal parameter

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = \mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}],$$

where we assume that $\mathbf{X}$ is non stochastic. Using the expectation value from equation (20) on vector form we get

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}. \tag{22}$$

We then calculate the variance of the optimal parameter

$$
\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol\beta}) &= \mathbb{E}[(\hat{\boldsymbol\beta} - \mathbb{E}[\hat{\boldsymbol\beta}])^2] \\
&= \mathbb{E}[((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} - \boldsymbol\beta)^2] \\
&= \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{y} - \boldsymbol\beta]\,[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{y} - \boldsymbol\beta]^T\} \\
&= \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{y}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{y}]^T\} - \boldsymbol\beta\,\boldsymbol\beta^T \\
&= \mathbb{E}\{(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{y}\,\mathbf{y}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1}\} - \boldsymbol\beta\,\boldsymbol\beta^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbb{E}\{\mathbf{y}\,\mathbf{y}^T\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol\beta\,\boldsymbol\beta^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\{\mathbf{X}\,\boldsymbol\beta\,\boldsymbol\beta^T\,\mathbf{X}^T + \sigma^2\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol\beta\,\boldsymbol\beta^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{X}\,\boldsymbol\beta\,\boldsymbol\beta^T\,\mathbf{X}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} + \ \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol\beta\boldsymbol\beta^T \\
&= \boldsymbol\beta\,\boldsymbol\beta^T + \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol\beta\,\boldsymbol\beta^T = \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1},
\end{aligned}
$$

$$
\mathrm{Var}(\hat{\boldsymbol\beta}) = \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}. \tag{23}
$$

## References

[1] Morten H. J. *Statistical analysis and discussion of Ridge and Lasso regression.* "`https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html`". 2022.