# Approximation of Topographical Datasets Using Linear Regression Project 1 – FYS-STK4155

Anders Thorstad Bø

*Department of Mathematics (Fluid Mechanics), University of Oslo, Oslo, Norway**

This project uses three different linear regression analysis methods to investigate their performance against 1D- and 2D-datasets, using $n^{th}$-order polynomial functions for prediction models. The methods are Ordinary Least Squares (OLS), Ridge and Lasso. It also displays how to make better predictions and get better estimates of hyperparameters in the model by using resampling techniques such as bootstrapping and k-fold cross-validation. For testing the different methods, I use the first Franke-function as a topographical analog. Here, I got the best performance with a $5^{\text{th}}$-order polynomial for all methods, giving a MSE $\approx 0.003$ and $R^2 \approx 0.96$ with hyperparameters $\lambda = 10^{-7}$ for Ridge and $\lambda = 10^{-10}$ for Lasso. As a final showcase, I use of the implementation to try to approximate the surfaces from topographical data from two mountainous areas in Norway to see how the models handles real-world data.

## I. INTRODUCTION

When using machine learning techniques to predict the outcome of datasets, a common issue that arises is overfitting and underfitting, often related to the complexity of the prediction model(Hastie et al., 2009). These effects will decide whether or not the resulting model are able to accurately make predictions on test data, benchmark data, and new data after training.

One important factor is to strike a balance between the performance of the model on the training data sets and how generally applicable it is to similar dataset. That way the finished model will be able to give a satisfactory representation of the new datasets of the same type, as it will be general enough to capture different effects in the new dataset that necessarily were not present in the training data (Goodfellow et al., 2016).

The aim of this project is to show how to use the linear regression methods **Ordinary Least Squares** (OLS), **Ridge**, and **Lasso** to make prediction models for 2D – datasets. Together with these methods, the project uses resampling techniques like **bootstrapping** and **K-fold-cross-validation** in order to tuning the hyperparameters in Ridge- and Lasso-regression, as well as investigating the **Bias-Variance trade-off** to indicate a suitable polynomial degree for the model (Hastie et al., 2009; Raschka et al., 2022).

For testing the methods and models, I am using a 1D exponential-function, a similar 2D exponential-function, and the first Franke-function (Franke, 1979) as data functions. This allows for seeing how the regression measures like mean square error and $R^2$-score evolve with more elaborate data functions. Only results for the Franke-function test case will be presented.

As final test of the methods and algorithms, I am using real topographical dataset from two different places in Norway. The first dataset is from a small mountain range in Etnedal, Oppland, and the second dataset is from the mountains in the western part of Jotunheimen, near Årdal. Both these datasets has fluctuations between areas in the dataset, and the expectation is that they will challenge the methods.

The report will first go through relevant theoretical concepts like the regression methods, resampling techniques, and give a general overview on how to interpret the measures and model performance.

Next, it gives a breakdown of the most important algorithms and method implementations, and a description on how I use them to generate the results.

Finally, the article presents the results from the benchmarking, and important discussion points that comes up in these results. The results and discussion on the model performance on the topographical datasets follows after that.

The very last parts of the article gives some final remarks and conclusions from the work, as well as some future perspectives, followed by a brief appendix explaining where and how to access the full program files, and how to use the different program files.

## II. THEORY AND METHODS

This section presents the theory and methods used in this project, first giving a quick overview of the three

* andetb@uio.no; https://github.com/andersthorstadboe/project-1-fys-stk4155-lin-regression

regression methods and their cost functions. After that, it gives a presentation of the usage and advantages of resampling and cross-validation, and finishing up with a discussion on the theory behind measuring the performance of the analysis.

But first, lets define some basic assumptions regarding the datasets the project uses. The datasets consists of two parts, the dependent data $\boldsymbol{y} = [y_0, y_1, \dots, y_i, \dots, y_{n-1}]$, and the independent variables $\boldsymbol{x} = [x_0, x_1, \dots, x_i, \dots, x_{n-1}]$, for $i \in [0, n-1]$. The vectors, $\boldsymbol{y}$, $\boldsymbol{x}$, is here the outcome or output, and the input, respectively.

They are related such that $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) = y_i(x_i)$. The function $\boldsymbol{y}(\boldsymbol{x})$ is assumed to be a smooth function. Using regression, the analysis will then try to find some specific relationship that describes the behavior of $\boldsymbol{y}$ given $\boldsymbol{x}$, such that this relationship can be used to create a model capable of approximating $\boldsymbol{y}$ to a satisfactory degree (Deisenroth et al., 2020).

### A. Linear regression methods

Linear regression models seek to represent the relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$ using a vector of parameters, $\boldsymbol{\beta}$, and a design matrix, $\boldsymbol{X}$, and an error vector, $\boldsymbol{\varepsilon}$. The design matrix is the collection of all training inputs, and is, in this context, of dimension $\mathbb{R}^{n\times p}$, where $n$ is the number of samples, and $p$ is the number of so-called features. The other vectors then need to have dimensions $\boldsymbol{\beta} \in \mathbb{R}^p$, $\boldsymbol{\varepsilon} \in \mathbb{R}^n$. The resulting model function used to approximate $\boldsymbol{y}$ can then be written as $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ describes the unknowns in the system (Hjorth-Jensen, 2023).

The basic idea is now to define a function describing how well $\tilde{\boldsymbol{y}}$ fits to $\boldsymbol{y}$, usually called a loss or cost function, $C$. Commonly, this function gives measure of the difference between the dataset and the model function. In this context, $C$ is dependent on the unknown $\boldsymbol{\beta}$'s, i.e. $C = C(\boldsymbol{\beta})$. The task of the regression analysis is then to minimize this difference. Since the unknown is $\boldsymbol{\beta}$, the optimization requires that $\partial_\beta[C(\boldsymbol{\beta})] = 0$. By solving for $\boldsymbol{\beta}$, the method finds the optimal values, $\hat{\boldsymbol{\beta}}$, that minimizes the difference between $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$.

The three linear regression methods the project uses take the same type of cost function as their basis, namely the **mean squared error** (MSE), which can be defined as

$$\begin{aligned}\mathrm{MSE}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right)^T\left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right) = \\ = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \tilde{y}_i\right)^2\end{aligned} \tag{A.1}$$

with $n$ as the total number of samples (Goodfellow et al., 2016).

If the regression only considers the MSE as its cost function, the method can be denoted **Ordinary Least Squares**, (OLS) (Raschka et al., 2022). Substituting for $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$, and taking the derivative of Eq.(A.1) with respect to $\boldsymbol{\beta}$ yields

$$\begin{aligned}\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \frac{\partial}{\partial \boldsymbol{\beta}}\left(\frac{1}{n}\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)\right) = 0 \;\Rightarrow \\ \hat{\boldsymbol{\beta}}_{\mathrm{OLS}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}\end{aligned} \tag{A.2}$$

as the optimal parameters for the OLS-cost function.

The two other methods uses similar cost functions to that of the OLS-method, expect that they make use of a so-call **regularization**-term that is added to the MSE. This regularization term seeks to put a penalty on the instances of $\{\beta_j\}_{j=0}^{p-1}$ that are large compared the other instances. The idea is that the regularization should make the $\beta_j$ more comparable, limiting the dominance of one or more very large $\beta_j$'s.

The models in this project uses $\mathrm{L}^1$- and $\mathrm{L}^2$-regularization for this purpose. This is related to the $\mathrm{L}^p$-norms for $p = 1, 2$, at giving an additional term as

$$\lambda||\boldsymbol{\beta}||_r = \lambda\left(\sum_{j=0}^{p-1}(\beta_j)^r\right)^{1/r} \tag{A.3}$$

where $r$ is the order of the norm, to avoid confusion with the numbers of features/parameters, $p$, and $\lambda$ as the **regularization parameter** or **hyperparameter** (Raschka et al., 2022).

The **Ridge**-regression method uses the squared of the $\mathrm{L}^2$-norm in addition to the MSE to define the cost function, reading as

$$\begin{aligned}C(\boldsymbol{\beta})_{\mathrm{Ridge}} = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \tilde{y}_i\right)^2 + \lambda\sum_{j=0}^{p-1}(\beta_j)^2 \\ = ||\boldsymbol{y} - \tilde{\boldsymbol{y}}||_2^2 + \lambda||\boldsymbol{\beta}||_2^2\end{aligned} \tag{A.4}$$

Performing the same differentiation as for the OLS-method, the optimal parameters for the Ridge case can be found as

$$\begin{aligned}\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 \\ \frac{\partial}{\partial \boldsymbol{\beta}}\left(\frac{1}{n}\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}\right) = 0 \\ \Rightarrow\; \hat{\boldsymbol{\beta}}_{\mathrm{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}\end{aligned} \tag{A.5}$$

where $\boldsymbol{I}$ is the identity matrix (Hastie et al., 2009; Hjorth-Jensen, 2023)

Both the preceding methods have analytical expressions for the optimal parameters, $\hat{\boldsymbol{\beta}}$, making finding a solution a relatively straight-forward linear algebra problem. One issue that may arise for the OLS-case is that the matrix product $\boldsymbol{X}^T\boldsymbol{X}$ needs to be invertible. If $X$ has linearly dependent column vectors, inverting that vector product is not possible. A method for aiding in will be presented in section II.A.1 (Hjorth-Jensen, 2023).

The last method, named **Lasso**-regression, uses the $L^1$-norm in addition to the MSE-expression, which making the cost function

$$C(\boldsymbol{\beta})_{\text{Lasso}} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 + \lambda\sum_{j=0}^{p-1}|\beta_j| = ||\boldsymbol{y}-\tilde{\boldsymbol{y}}||_2^2 + \lambda||\boldsymbol{\beta}||_1 \tag{A.6}$$

Again, differentiating Eq.(A.6) with respect to $\boldsymbol{\beta}$ yields the expression

$$\frac{\partial C(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = \frac{\partial}{\partial\boldsymbol{\beta}}\left(\frac{1}{n}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|\right) = 0$$
$$-\frac{2}{n}\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}) + \lambda\frac{\partial}{\partial\boldsymbol{\beta}}(|\boldsymbol{\beta}|) = 0 \tag{A.7}$$

Reordering to isolate the $\boldsymbol{\beta}$-terms on the LHS, with

$$\frac{\partial}{\partial\boldsymbol{\beta}}(|\boldsymbol{\beta}|) = \text{sgn}(\boldsymbol{\beta})$$

the expression to be optimized for becomes

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda\text{sgn}(\boldsymbol{\beta}) = \boldsymbol{X}^T\boldsymbol{y} \tag{A.8}$$

where $\lambda$ absorbs the $(n/2)$-factor. This expression does not have an analytical solution, and will need to be solved numerically to obtain the optimal parameters, $\hat{\boldsymbol{\beta}}_{\text{Lasso}}$ (Hastie et al., 2009).

### 1. Singular Value Decomposition

In order to tackle the possible issues when inverting the matrix product $\boldsymbol{X}^T\boldsymbol{X}$, a method called **Singular Value Decomposition** (SVD) can be used. This splits the matrix $\boldsymbol{X}\in\mathbb{R}^{n\times p}$ into three different matrices, usually denoted $\boldsymbol{U},\boldsymbol{V},\boldsymbol{\Sigma}$ such that

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{A.9}$$

Here, $\boldsymbol{U}$and$\boldsymbol{V}$ are orthogonal matrices of dimension $\mathbb{R}^{n\times n}$, $\mathbb{R}^{p\times p}$. $\boldsymbol{\Sigma}$ has the singular values of $\boldsymbol{X}$ on the diagonal, zeros elsewhere, and has dimension $\mathbb{R}^{p\times p}$.

Decomposing $\boldsymbol{X}$ like this ensures that the matrix product $\boldsymbol{X}^T\boldsymbol{X}$, which now writes as

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{A.10}$$

Since $\boldsymbol{U}$ is orthogonal, $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$, so

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{A.11}$$

The implementation can use this relation to calculate $\hat{\boldsymbol{\beta}}$ from Eq.(A.2), in most cases where $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible (Hastie et al., 2009).

## B. Statistical concepts ("Part d")

In the assumptions from the beginning of this section, the datasets are described by a continuous function, $f(\boldsymbol{x})$. It is common to consider some added noise to this to get the dataset, $\boldsymbol{y}$, such that the full dataset is $\boldsymbol{y} = f(\boldsymbol{x})+\boldsymbol{\varepsilon}$, with $\boldsymbol{\varepsilon}$ as normally distributed noise with mean value zero, and variance, $\text{Var}(\boldsymbol{\varepsilon}) = \sigma_\varepsilon^2$.

It is useful to say something about the mean, or **expected**, value and variance of the dataset and regression methods, in order to gain some insight into the concepts. First, lets define the expected value and variance of a variable $\omega$ as

$$\mathbb{E}[\boldsymbol{\omega}] = \frac{1}{n}\sum_{i=0}^{n-1}\omega_i = \mu_\omega \tag{B.1}$$

$$\text{Var}(\boldsymbol{\omega}) = \mathbb{E}[(\boldsymbol{\omega}-\mu_\omega)^2] = \mathbb{E}[\boldsymbol{\omega}^2] - (\mathbb{E}[\boldsymbol{\omega}])^2 \tag{B.2}$$

such that, as an example

$$\mathbb{E}[\boldsymbol{\varepsilon}] = \mu_\varepsilon = 0$$

and

$$\text{Var}(\boldsymbol{\varepsilon}) = \mathbb{E}[\boldsymbol{\varepsilon}^2] - (\mathbb{E}[\boldsymbol{\varepsilon}])^2 = \mathbb{E}[\boldsymbol{\varepsilon}^2] = \sigma_\varepsilon^2$$

Since the approximation $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$ is an attempt on replicating the behavior of $f(\boldsymbol{x})$, the following assumes that $\boldsymbol{X}\boldsymbol{\beta}\approx f(\boldsymbol{x})$ holds to a certain degree. This makes makes it possible to state that $\boldsymbol{y}\approx\boldsymbol{X}\boldsymbol{\beta}+\boldsymbol{\varepsilon}$. To calculate the expected value of $\boldsymbol{y}$, from the definition, this gives

$$\mathbb{E}[y_i] = \mathbb{E}\left[\sum_{j=0}^{p-1}x_{ij}\beta_j + \varepsilon_i\right] = \mathbb{E}\left[\sum_{j=0}^{p-1}x_{ij}\beta_j\right] + \mathbb{E}[\varepsilon_i] \tag{B.3}$$

Since $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$, this leaves only the first expectation value. Denoting the different row vectors of the design matrix, $\boldsymbol{X}$, as $\boldsymbol{X}_{i,*}$, this simplifies to

$$\mathbb{E}[y_i] = \mathbb{E}\left[\sum_{j=0}^{p-1}x_{ij}\beta_j\right] = \boldsymbol{X}_{i,*}\boldsymbol{\beta} \tag{B.4}$$

Here, the fact that the expectation value of the set of outcomes, $y_i$, is the original approximation, $\boldsymbol{X}_{i,*}\boldsymbol{\beta}$ confirms the assumption that $\boldsymbol{X}\boldsymbol{\beta} \approx f(\boldsymbol{x})$ for sufficiently large sample sizes (Deisenroth et al., 2020).

To calculate the variance of $\boldsymbol{y}$, using the definition in Eq.(B.2) to write

$$\begin{aligned} \mathrm{Var}(y_i) = \mathbb{E}\left[(y_i - \mu_{y_i})^2\right] = \mathbb{E}[y_i^2] - (\mathbb{E}[y_i])^2 = \\ = \mathbb{E}[(\boldsymbol{X}_{i,*}\boldsymbol{\beta} + \varepsilon_i)^2] - (\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2 \end{aligned} \tag{B.5}$$

The first expression, written out, gives

$$\begin{aligned} \mathbb{E}[(\boldsymbol{X}_{i,*}\boldsymbol{\beta} + \varepsilon_i)^2] = \\ \mathbb{E}[(\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2 + 2\boldsymbol{X}_{i,*}\boldsymbol{\beta}\ \varepsilon_i + \varepsilon_i)^2] = \\ = \mathbb{E}[(\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2] + 2\mathbb{E}[(\boldsymbol{X}_{i,*}\boldsymbol{\beta})]\ \mathbb{E}[\varepsilon_i] + \mathbb{E}[\varepsilon_i^2] = \\ = (\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2 + \sigma_\varepsilon^2 \end{aligned} \tag{B.6}$$

Substituting this into the expression for the variance of $\boldsymbol{y}$ gives

$$\mathrm{Var}(y_i) = (\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2 + \sigma_\varepsilon^2 - (\boldsymbol{X}_{i,*}\boldsymbol{\beta})^2 = \sigma_\varepsilon^2 \tag{B.7}$$

These two relations gives the information that, given the assumptions, the dataset $\boldsymbol{y}$ follows a normal distribution with expected value $\boldsymbol{X}\boldsymbol{\beta}$ and variance $\sigma_\varepsilon$. That is, the expected outcome from approximating $\boldsymbol{y}$ by $\boldsymbol{X}\boldsymbol{\beta}$ is $\boldsymbol{X}\boldsymbol{\beta}$ itself, with the variance of the outcome decided by the original noise of the data. This is a great outcome, as the approximation has not added any additional variance.

Now, lets do the same with the expression for the optimal values for the OLS-method (ref. Eq.(A.2)). First, the expected value is

$$\begin{aligned} \mathbb{E}[\hat{\boldsymbol{\beta}}_{\mathrm{OLS}}] = \mathbb{E}\left[\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}\right] = \\ = \mathbb{E}\left[\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\right]\ \mathbb{E}\left[\boldsymbol{X}^T\right]\ \mathbb{E}\left[\boldsymbol{y}\right] \end{aligned} \tag{B.8}$$

The second step is allowed if all elements inside the expectation value are statistically independent, meaning $\boldsymbol{X}$ does not influence $\boldsymbol{y}$ and vice verse. Since both are known in this case, this is a reasonable assumption (Deisenroth et al., 2020).

Using again that $\mathbb{E}[\boldsymbol{y}] = \boldsymbol{X}\boldsymbol{\beta}$, similarly for the other elements with $\boldsymbol{X}$, the final result is

$$\mathbb{E}[\hat{\boldsymbol{\beta}}_{\mathrm{OLS}}] = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\ \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{\beta} \tag{B.9}$$

as the expected value of $\hat{\boldsymbol{\beta}}_{\mathrm{OLS}}$.

Moving on to the variance, dropping the subscript for simplicity, this gives

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathbb{E}\left[(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^T\right] \tag{B.10}$$

Using the result from the expectation value calculation, substituting for $\hat{\boldsymbol{\beta}}$, and performing the matrix multiplication yields

$$\begin{aligned} \mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathbb{E}\left[\left((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{\beta}\right)^2\right] = \\ = \mathbb{E}\left[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\right]\ \mathbb{E}\left[\boldsymbol{X}^T\right]\ \mathbb{E}\left[\boldsymbol{y}\right]\dots \\ \dots\mathbb{E}\left[\boldsymbol{y}\right]^T\mathbb{E}\left[\boldsymbol{X}\right]\ \mathbb{E}\left[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\right] - \boldsymbol{\beta}\boldsymbol{\beta}^T = \\ = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\ \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T]\ \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \end{aligned} \tag{B.11}$$

This can be simplified by writing out $\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T]$ as

$$\begin{aligned} \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \mathbb{E}\left[(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon})(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon})^T\right] = \\ = \mathbb{E}\left[\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T\right] + \mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T] = \\ = \boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T + \sigma_\varepsilon^2\boldsymbol{I} \end{aligned} \tag{B.12}$$

and substituting this into the expression for the variance

$$\begin{aligned} \mathrm{Var}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\left(\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T + \sigma_\varepsilon^2\boldsymbol{I}\right)\dots \\ \dots\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T = \\ = \boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} + \dots \\ \dots(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\sigma_\varepsilon^2\boldsymbol{I}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T = \\ = \boldsymbol{I}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{I} + \boldsymbol{I}\sigma_\varepsilon^2\boldsymbol{I}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T = \\ = \sigma_\varepsilon^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} \end{aligned} \tag{B.13}$$

as the variance of $\hat{\boldsymbol{\beta}}_{\mathrm{OLS}}$.

Similarly to the analysis of $\boldsymbol{y}$, the expectation value for the optimal parameters is the original values. The variance, the noise from the original dataset still contributes, as a scale for the matrix of $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$.

The diagonal elements of the resulting matrix gives an estimate of the variance of the regression parameters. This estimate can give an indication on how a given $\beta_j$ may change between different datasets in a population(Hjorth-Jensen, 2023).

### C. Resampling and cross-validation

Performing a regression analysis on a dataset, optimizing for the regression parameters, $\boldsymbol{\beta}$, can be done once per dataset. However, for a given dataset, this approach may result in a bad prediction model for various reasons. These can be that the amount of samples in the dataset are too few to capture the real behavior, or the training data may be a dataset only capturing a certain subset of the behaviors. Being able to repeat the regression analysis multiple times may result in better predictions.

To combat these effects, it is possible to utilize so-called resampling techniques. The overarching approach of these is to quantify the uncertainty by gaining more information about the prediction model. This is done by performing the regression analysis repeatedly on different samples of the training data, and assessing how the resulting models differ form each other (Hjorth-Jensen, 2023).

The two techniques of interest for this project is **bootstrapping** and **k-fold cross-validation**. But first, lets present a related concept.

#### 1. Bias-Variance trade-off "Part e)"

The concept of a Bias-Variance trade-off in the context of regression analysis tries to quantify certain relationships in the dataset assumptions. This, in turn, will give additional information on how to best choose the different parameters in the model, such as the regression parameters, $\boldsymbol{\beta}$, different hyperparameters, like $\boldsymbol{\lambda}$, and model complexity, such as polynomial degree, $p$, as in this study (Hjorth-Jensen, 2023).

A common way to do this is to express the prediction error in terms of a **Bias**- and **Variance**-function, then investigating where, with respect to a value of a parameter, the functions tends towards the same value. That is, where the trade-off between the contribution to the overall error from the two functions are at an optimal point. This point then gives an indication to which value the different parameters should be set in the final prediction model (Hastie et al., 2009)

With the same assumptions to the dataset and model as made in section II.B, and with the cost function for the OLS-method, it is possible to write

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

by the definition of the expected value from Eq.(B.1). Expanding the square inside the expected value yields

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2] = \mathbb{E}[(\boldsymbol{y}^2 - 2\boldsymbol{y}\tilde{\boldsymbol{y}} + \tilde{\boldsymbol{y}}^2] = \\ = \mathbb{E}[\boldsymbol{y}^2] - 2\,\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}^2]\end{aligned} \tag{C.2}$$

assuming $\boldsymbol{y}$, $\tilde{\boldsymbol{y}}$ are statistically independent. Taking this term by term, this expression simplifies, so starting with $\mathbb{E}[\boldsymbol{y}^2]$

$$\begin{aligned}\mathbb{E}[\boldsymbol{y}^2] = \mathbb{E}[(f(\boldsymbol{x}) + \boldsymbol{\varepsilon})^2] = \\ = \mathbb{E}[f(\boldsymbol{x})^2 + 2f(\boldsymbol{x})\boldsymbol{\varepsilon} + \boldsymbol{\varepsilon}^2] = \\ = f^2 + 2\,\mathbb{E}[f\boldsymbol{\varepsilon}] + \sigma_\varepsilon^2\end{aligned} \tag{C.3}$$

with $\mathbb{E}[\boldsymbol{\varepsilon}^2] = \sigma_\varepsilon^2$, and $\mathbb{E}[f(\boldsymbol{x})^2] = f^2$ since $f(\boldsymbol{x})$ is a non-stochastic function. Another way to define the expectation value of a function, $g(\boldsymbol{x})$ is by the integral definition (Deisenroth et al., 2020),

$$\mathbb{E}[g(\boldsymbol{x})] = \int_{\eta\in\mathcal{D}} g(\eta)p(\eta)d\eta \tag{C.4}$$

and for the term $\mathbb{E}[f\boldsymbol{\varepsilon}]$, this gives

$$\begin{aligned}\mathbb{E}[f\boldsymbol{\varepsilon}] = \int_\eta f(\boldsymbol{x})\boldsymbol{\varepsilon}(\eta)p(\eta)d\eta = \\ = f(\boldsymbol{x})\int_\eta \boldsymbol{\varepsilon}(\eta)p(\eta)d\eta = f(\boldsymbol{x})\mathbb{E}[\boldsymbol{\varepsilon}] = 0\end{aligned} \tag{C.5}$$

This gives Eq.(C.3) as

$$\mathbb{E}[\boldsymbol{y}^2] = f^2 + \sigma_\varepsilon^2 \tag{C.6}$$

The second term in Eq.(C.2) will simplify as

$$\begin{aligned}\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] = \mathbb{E}[(f+\boldsymbol{\varepsilon}]\tilde{\boldsymbol{y}}] = \\ = \mathbb{E}[f\tilde{\boldsymbol{y}}] + \mathbb{E}[\boldsymbol{\varepsilon}\tilde{\boldsymbol{y}}] = \\ = \mathbb{E}[f\tilde{\boldsymbol{y}}] + \mathbb{E}[\boldsymbol{\varepsilon}]\;\mathbb{E}[\tilde{\boldsymbol{y}}] = \mathbb{E}[f\tilde{\boldsymbol{y}}]\end{aligned} \tag{C.7}$$

Finally, using the definition of the variance from Eq.(B.2),

$$\begin{aligned}\mathrm{Var}(\tilde{\boldsymbol{y}}) = \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 = \\ = \mathbb{E}[\tilde{\boldsymbol{y}}^2] - (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 \;\Rightarrow \\ \Rightarrow\; \mathbb{E}[\tilde{\boldsymbol{y}}^2] = \mathrm{Var}(\tilde{\boldsymbol{y}}) + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2\end{aligned} \tag{C.8}$$

and substituting Eq.(C.6), (C.7), (C.8) into Eq.(C.2)

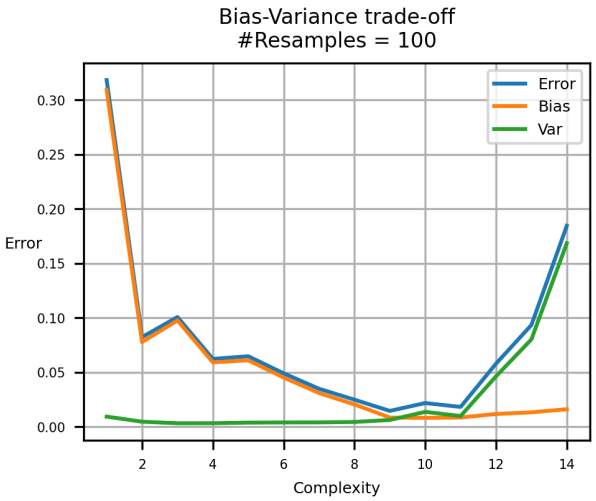


FIG. II.1: Showing the error, bias and variance plotted against model complexity with results from a regression analysis using bootstrapping. There is a clear point here showing where the variance takes over as the dominant contributor to the overall error.

gives

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2] &= f^2 + \sigma_\varepsilon^2 - 2\,\mathbb{E}[f\tilde{\boldsymbol{y}}]\ldots \\ &\cdots + \mathrm{Var}(\tilde{\boldsymbol{y}}) + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 = \\ &= \mathbb{E}[f^2] - 2\,\mathbb{E}[f\tilde{\boldsymbol{y}}] + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \mathrm{Var}(\tilde{\boldsymbol{y}}) + \sigma_\varepsilon^2 = \\ &= \mathbb{E}\left[(f-\mathbb{E}[\tilde{\boldsymbol{y}}])^2\right] + \mathrm{Var}(\tilde{\boldsymbol{y}}) + \sigma_\varepsilon^2 = \\ &= \mathrm{Bias}[\tilde{\boldsymbol{y}}] + \mathrm{Var}(\tilde{\boldsymbol{y}}) + \sigma_\varepsilon^2\end{aligned} \tag{C.9}$$

Using that $f(\boldsymbol{x}) = \boldsymbol{y}$, the $\mathrm{Bias}[\tilde{\boldsymbol{y}}]$ becomes

$$\mathrm{Bias}[\tilde{\boldsymbol{y}}] = \mathbb{E}[(\boldsymbol{y}-\mathbb{E}[\tilde{\boldsymbol{y}}])^2] \tag{C.10}$$

This analysis illustrates how the expected value of the error, based on the MSE cost-function, splits into two different terms that depends on the dataset, $\boldsymbol{y}$, and model, $\tilde{\boldsymbol{y}}$.

The Bias-function is a measure of the mean difference between the data points, $y_i$, and the mean value of model points. A low value here indicates that the model values tend to be close to the data points. The Var-function measure the spread of the model points around the mean value of the model. A high value in the Var-function indicates that distances of the model values vary a lot, indicating that different prediction outcomes may differ significantly between different datasets.

The ideal case is a model resulting both low bias and low variance, since this minimizes both the tendency of the model to "miss" the true values, and the spread of the predictions on different datasets of the same type. Fig.II.1 depicts the typical tendency of these two measures with respect to model complexity. It shows a clear point where the contribution to the overall error shifts from being from the bias to the variance, i.e. where the trade-off complexity point is for that specific model.

### 2. Bootstrap method

One issue that may arise when using real-world data, is that the samples sizes from experiments are too small to capture all of the relevant trends in the system. One of the effects that this has is that the variance of the dataset becomes large, since the spread in data points between different experiments may be large. This can lead to a model that takes this apparent inconsistency into account, resulting in a bad fit.

To tackle this inconsistency, the bootstrap method leverages the result from **central limit theorem** by "creating" more data from the original data by resampling. In short, the central limit theorem says that any population of random variables of an arbitrary distribution, will be normally distributed in the limit as the number of variables tends to infinity (Bishop, 2006).

The bootstrap method uses this by resampling, with or without replacement the original dataset $n$ times, in effect creating "new" datasets. For each new dataset, a regression fit is made, and using a Bias-Variance trade-off approach, the mean values of the bias and variance can be computed for different parameters, such as complexity or regularization parameters. The resampling has the effect that the variance should tend towards a variance the dataset would have if the number of samples where large (Bishop, 2006).

The replacement statement indicates how the "drawing" of samples happens. Allowing for replacement means returning the drawn sample back into the set, having the effect that the same sample may appear multiple times in the resulting resampled data set (Raschka et al., 2022).

An implementation of this for this project can be found in Sec.III.B.1.

### 3. K-fold cross-validation

Another method useful for model selection is K-fold cross-validation. The basic idea is to take the training data, and split it into $k$ folds, where $k-1$ folds are training folds, and one is a test fold. The training happens on the training folds, and and the evaluation of the fit uses the test fold. The process repeats $k$-times,

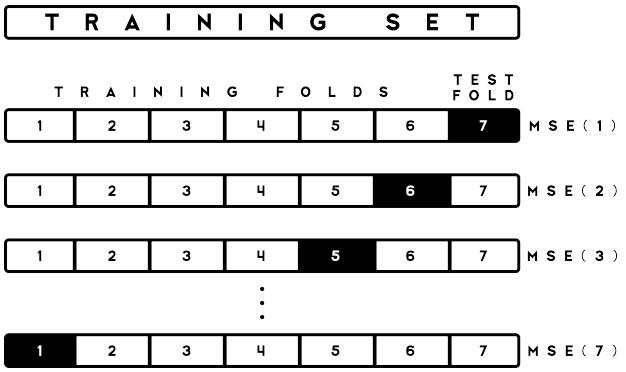


FIG. II.2: A conceptual illustration of how the k-fold algorithm works, and how the training data is divided into seven different folds (based on figure on page 117 from (Raschka et al., 2022)).

as shown in Fig.II.2, each iteration calculating an error for that fold. In the end, the individual error estimates is combine to give a average performance of the model. The splitting is done without replacement, ensure that the training and evaluation sets does not have overlapping values (Raschka et al., 2022).

After iterating over all the folds, the output from the validation process should then give information on what value of e.g. hyperparameter will give the best model performance. Using this as input, a final fit can be made on the test dataset, generating the final prediction model (Raschka et al., 2022).

An advantage of using the technique is that the evaluation, or testing, happens on independent subsets of data. The individual iteration's test scores (here the MSE-values) then does not depend on any data points present in any of the other test folds. This may give a better generalization to the fit than f.ex. bootstrapping, which uses the same dataset with the values shuffled, so the different datasets are not independent.

Another advantage of k-fold cross-validation is that it ensures that all data points makes it into the training of the model. If the bootstrap method is implemented using replacement, there is no guarantee that the entire training set will be used, although with a high number of resamples this is likely. If the sample size is small, utilizing the entire dataset set aside for training is advantageous (Raschka et al., 2022).

There are some guideline on choosing the number of folds to split the training data into. If the size of the training set is small, a larger value of $k$ may give better performance by lowering the bias in each fit. But it may also increase the variance as the fold size decrease, the training folds will be more equal. (Raschka et al., 2022) states that $k \in [5, 10]$ seems to generally give good results.

### D. Model selection and assessment

The main reason for performing a regression analysis, together with the resampling and cross-validation, is to gain knowledge of what method and steps produces the best prediction model for a given dataset. This also includes what value of the hyperparameter, $\lambda$, will provide the "right" regularization to the $\hat{\boldsymbol{\beta}}$-terms in order to get rid of possible overfitting to training data.

The notion of **model selection** refers to using different methods to figure out the best possible parameters for a given dataset. Performing the regression analysis on allocated training data, utilizing validation methods such as k-fold cross-validation and bootstrapping, should then give a set of optimal parameters as output.

After concluding, these parameters are input into a final prediction model. The **model assessment** then refers to how well the resulting model performs against test datasets. This is a quantitative assessment, looking at different error metrics to check how much the final prediction deviates from the "real" test case (Hastie et al., 2009).

The two error metrics relevant for this project is the MSE, as presented in Eq.(A.1), and the $R^2$-score, which is given as

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

where $\bar{y}$ is the mean value of the dataset $\boldsymbol{y}$.

## III. CODES AND CODE DESCRIPTION

This section presents the algorithms and programs performing the analysis. It will give a conceptual overview of the different algorithms for the regression analysis, and the resampling and cross-validation approaches.

It will also present some of the post-processing programs that produces the figures and visualizations presented in this report. Information on how to access the different programs written for this project, as well as additional figures and results can be found in App.A.

### A. Python packages

This project uses the Python programming language together with three well-known packages, **NumPy, matplotlib** and **scikit-learn**. The NumPy-package is used for array-manipulation and -computation such as all matrix/array calculations, and for computing statistical metrics such as averages, variances, etc.(Harris et al.,

2020). matplotlib is used as the visualization tool for the project, and all figures presented in this report and on the GitHub-page is generated using this package (Hunter, 2007). The scikit-learn-package contains the machine learning classes, methods, and functions used in this project. Examples are for resampling, splitting data, and the Lasso-method (Pedregosa et al., 2011).

### B. General analysis algorithms

The approaches for the three different regression methods are fairly similar, but they have their difference. A general algorithm outline is given in Alg.1.

**Algorithm 1** General algorithm for the regression methods. Some steps only applicable for certain methods, stated in the step using *cursive*.

```
Initialize datasets (test function, topographical data)
Set model complexity list, p_deg, by fixing p_max
Initialize list of λ's (Ridge and Lasso-method)
Initialize storage containers for outputs
Split dataset into training and test data
for each degree in p_deg do
    Create design matrices from x_train, x_test
    if Scaling data = True then
        Subtract mean from training data
    end if
    for λ in list (Ridge and Lasso-method) do
        Calculate β̂ from method equation.
        if If matrix inversion fails (OLS-method) then
            Use SVD to calculate β̂
        end if
        Train model using β_0 from OLS-method.
        Calculate MSE and R^2-score
    end for
    Store output in containers
end for
Store output data from p-loop in common container.
```

Something to note here is that the splitting of the datasets into training and test data happens before entering the loop over polynomial degrees, $p_{\text{deg}}$, such that the input to the creation of the design matrices, $\boldsymbol{X}_{\text{train}}, \boldsymbol{X}_{\text{test}}$, is equal for each polynomial degree.

Another detail is that, when performing the OLS-analysis, the approach tries a straight-forward calculation of $\boldsymbol{\beta}$ using Eq.(A.2).

If this ends in issues with the matrix inversion of $\boldsymbol{X}^T\boldsymbol{X}$, the program uses the SVD-approach. This may be beneficial, as the SVD-approach needs more calculation steps, the runtime will be somewhat reduced.

Also, the design matrices does not include the intercept column by default, decided by the two small programs creating the matrices based on $p_{\text{deg}}$ (see App. A). This requires that this intercept is added to the prediction model before calculating the regression metrics. This intercept is the $\beta_0$-value from the OLS-analysis, and is used by all three methods.

Finally, a comment on the scaling of the data. There is an option to scale simply by subtracting the mean from the training data. The default option scales the training data, and the calculation of the $\hat{\boldsymbol{\beta}}$ uses the scaled version. The fitting of the model is done on the unscaled versions, such that the calculations of the metrics happens with the original dataset.

#### 1. Resampling and validation

For performing the Bias-Variance trade-off analysis using a bootstrap-method on the OLS-regression case, requires a small adjustment to the general algorithm. Alg.2 shows the basic approach.

**Algorithm 2** Addition to the general algorithm for using a bootstrapping-method for a Bias-Variance trade-off analysis with the OLS-method.

```
Same initialization steps as Alg.1
Choose number of bootstrapping steps, n_boots
for each degree in p_deg do
    Same design matrix and scaling as Alg.1
    Initialize container for storing outputs
    for 0 to n_boots do
        Resample training data
        Calculate β using SVD-method
        Calculate intercept, β_0
        Calculate ỹ = X_test β + β_0
        Store ỹ, β_0 and β for step
    end for
    Output ỹ's, β_0's and β's
    Calculate error, Bias and Var using Eq.(C.9)
end for
Plot error, Bias and Variance wrt. p_deg
```

The program uses the *scikit-learn*-module **utils** for the bootstrap-loop resampling using the function *resample*, which follows the method described in Sec.II.C.2 (Pedregosa et al., 2011).

The same basic approach was chosen for doing the k-fold cross-validation on all three regression methods, but with different resampling approach based around the *scikit-learn*-module **model_selection**-module's class *KFold*. Alg.3 show the general implementation outline.

**Algorithm 3** Addition to the general algorithm for k-fold cross-validation performed on all three regression methods.

```
Same initialization steps as Alg.1
Choose number of folds, initialize k-fold object
for each degree in p_deg do
    Create design matrix
    Initialize container for storing cross-validation scores
    for λ's in λ-list do
        for k number of folds do
            Choose training and test folds
            Same design matrix and scaling as Alg.1
            Initialize container for storing cross-validation scores
            Regression analysis (all three methods)
            Calculate MSE for each λ_i and store
        end for
        Calculate MSE for each p_deg and store
    end for
end for
Calculate mean value of MSE for all methods
Plot scores wrt. λ
```

Some care has been made to create general purpose visualization methods to use in this project, but the nature of the data requires some special attention in many different cases. As a general note on the visualization approach is to keep the figures as clean and uncluttered as possible. Some work has also been put into choosing suitable color-maps for the different figure-types.

## IV. RESULTS

This section will present the results from the three different regression methods on the two different datasets. It will also go through some notable results from the statistical investigation, like the Bias-Variance trade-off, and the cross-validation results. Some additional results are shown in App.B.

### A. Datasets

#### 1. Franke-function

The Franke-function is a much-used test function for assessing interpolation and regression methods, as its shape has some challenging features. It consists of two tall Gaussian peaks, and one fairly sharp Gaussian dip. Eq.(A.1) shows the function, and Fig.IV.1 shows a rendering of the function on the domain $[0,1]\times[0,1]$ (Franke, 1979).

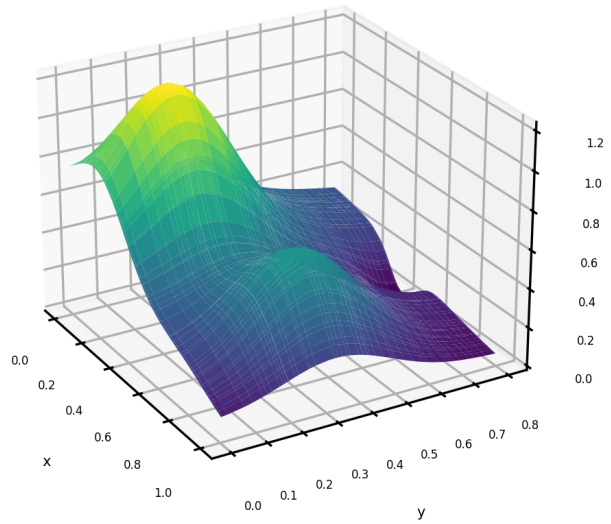


FIG. IV.1: Visualization of the Franke function without added noise.

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

#### 2. Topographical data

The real-world test data for this project is topographical data was downloaded from `https://hoydedata.no/LaserInnsyn2/`. Two areas where chosen, one around Etnedal in central Norway, and the second of the western parts of Jotunheimen near Årdal. Fig.IV.2 shows a rendering of the Jotunheimen dataset. A visualization of the other dataset can be found in App.B. The data was downloaded with a 10 m resolution, resulting in fairly large data sets. The figures show the datasets significantly downsampled, and these are the ones the the regression analysis was performed on.

### B. Bias-Variance trade-off

The concept of a Bias-Variance trade-off presented in Sec.II.C.1 should show point where the complexity of the model leads to an increasing Var-value. As the variance increases, this points to an increase in the spread between the different models created by the bootstrap-iterations. Increasing spread between fits, in turn, may indicate that the model complexity gives an overfitted model, meaning

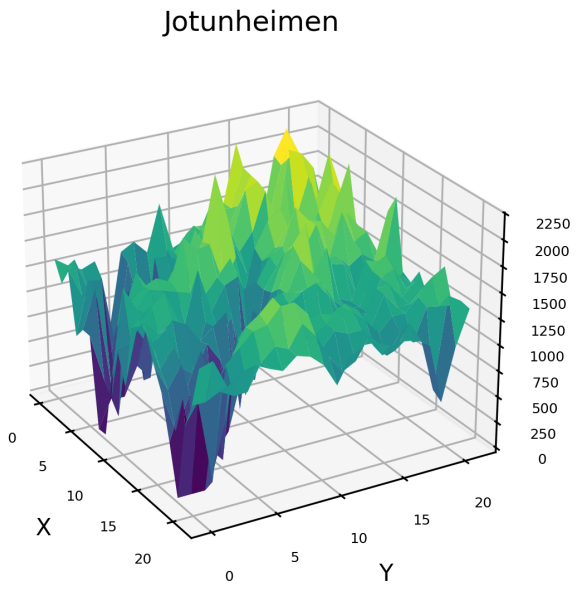


FIG. IV.2: Rendering of topographical data downloaded from `https://hoydedata.no/LaserInnsyn2/` of the part of Jotunheimen near Årdal, Norway.

that the model will be too specialized on the training data, and not general enough. Looking at Fig.IV.3, this model complexity seems to be at around $p = 8$ for the OLS cost-function when applied to the Franke test function on a $(Nx, Ny) = (15, 15)$-grid size. The smaller sample size was chosen to try to exaggerate the effect of the bootstrapping.

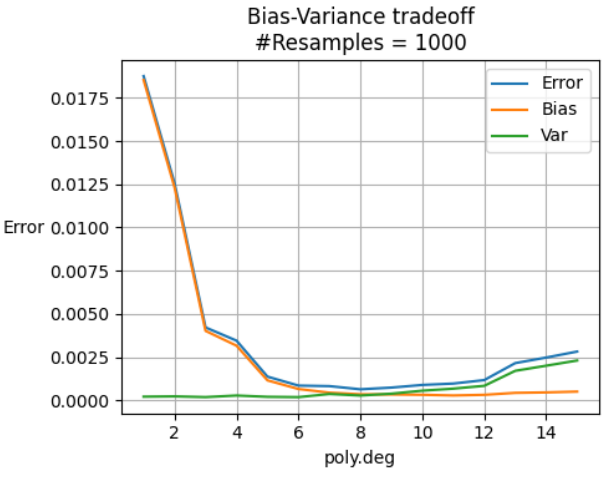


FIG. IV.3: This showing the Bias-Variance trade-off after 1000 bootstrap-iterations using the OLS cost-function up to $p = 15$ on the Franke-function.

TABLE IV.1: Input variable values for the regression analysis on the Franke function.

| **Variable** | **Value** | **Comment** |
|---|---|---|
| Domain $(x, y)$ | $[0, 1] \times [0, 1]$ | |
| Grid $(Nx, Ny)$ | $(40,\ 40)$ | Chosen for runtime |
| Mod.complexity | $p \in [1, 5]$ | |
| Reg.parameter | $\lambda \in [10^{-10}, 10^{10}]$ | Regularization |
| Test size | 0.2 | |
| Iteration # | $n = 1000$ | For bootstrap |
| folds | $k = 8$ | For k-fold-method |

### C. Results for Franke-function analysis

Tab.IV.1 shows inputs for the regression analysis with the Franke-function as test function. Two plots generated from running the program with these values as input are shown in Fig.IV.4, IV.5. These show the MSE for the Ridge and Lasso cost-functions, respectively. The same general trend can be seen in both, indicating that the optimal parameter range for $\lambda$ and $p$ lies in the upper right corner. The same basic trend can also be found when computing the $R^2$-score.

Fig.IV.6 shows the MSE-values for the OLS-method. Since this does not depend on a regularization parameter, the MSE for both the training and test data are shown in the same figure. The same trend as for the two other methods can be seen here as well, a decreasing error as the model complexity increases. But, as shown in Sec.IV.B, this trend will not continue as $p$ increases. One last result that may be interesting to comment on is

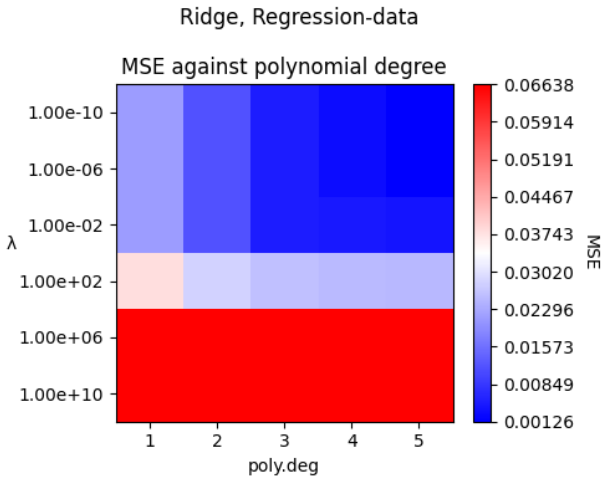


FIG. IV.4: Showing the MSE-values for the different parameters $\lambda$ against model complexity, $p$ from a Ridge-analysis of the Franke function. Blue parts indicate the optimal range for the two parameters.

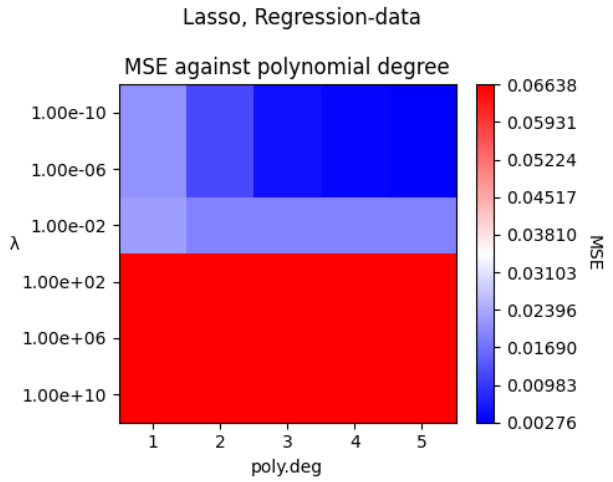


FIG. IV.5: This shows the MSE-values for the different parameters $\lambda$ against model complexity, $p$ from a Lasso-analysis of the Franke function. Blue parts indicate optimal range for the two parameters.

how the regression parameters $\hat{\boldsymbol{\beta}}$ behave with increasing complexity. Fig.IV.7 shows these for the OLS-analysis. As $p$ increases, this plot shows more and more fluctuating $\beta$-values, and for $p = 5$ the ones with the greatest magnitude correspond to powers of $x, y$ up to $2 - 4$. Looking at the shape of the test function, it seems reasonable to weight these powers more than others.

Tab.IV.1 shows the input parameters chosen for the final prediction model for approximating the Franke-function after reviewing the results. The cap for the model complexity was set at $p = 5$. Looking at Fig.IV.3, this complexity is lower than what the Bias-Variance trade-off analysis shows, so increasing $p$ may give a better

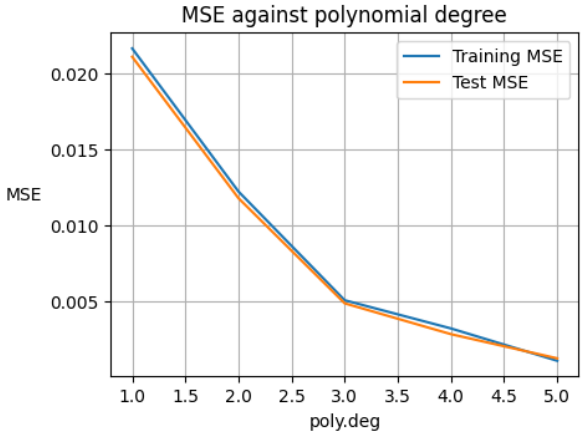


FIG. IV.6: Showing the MSE for the training and test data calculated for the prediction model from the OLS-method.

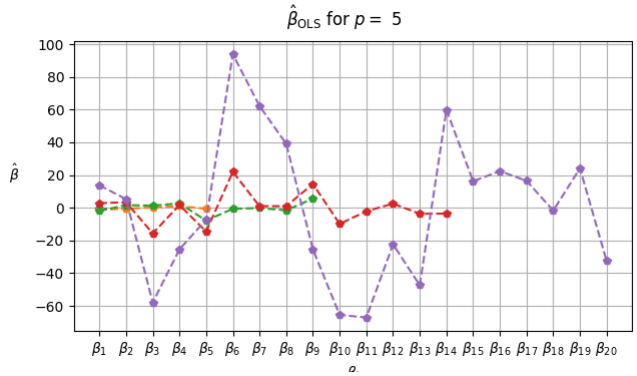


FIG. IV.7: This shows the value of the $\beta_i$'s as model complexity increases. Notice that the magnitude increases as $p$ increases.

fit.

TABLE IV.2: Final input-parameters after the regression analysis for the Franke-test function.

| **Variable** | **OLS** | **Ridge** | **Lasso** |
|---|---|---|---|
| Mod.complexity, $p$ | 5 | 5 | 5 |
| Reg.parameter, $\lambda$ | – | $10^{-7}$ | $10^{-10}$ |

The Ridge- and Lasso-predictions for the Franke-function using the optimal input parameter is rendered in Fig.IV.8, IV.9, and the corresponding test metrics for all three methods can be found in Tab.IV.3. For additional figures showing the final prediction surfaces, see App.B.

The table below shows that a good overall performance of all three regression methods. The Lasso-method does

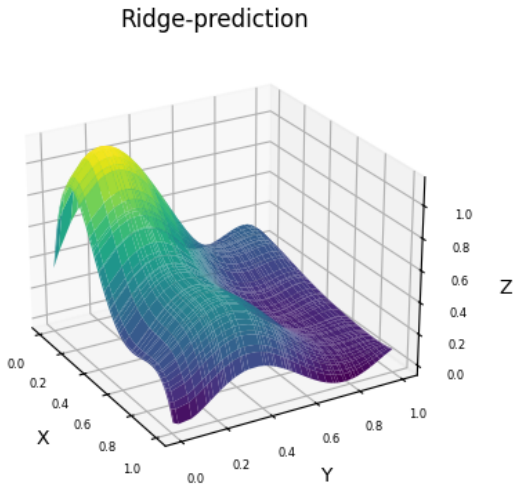


FIG. IV.8: Final prediction model for the Franke-function using the optimal parameters with the Ridge cost-function.

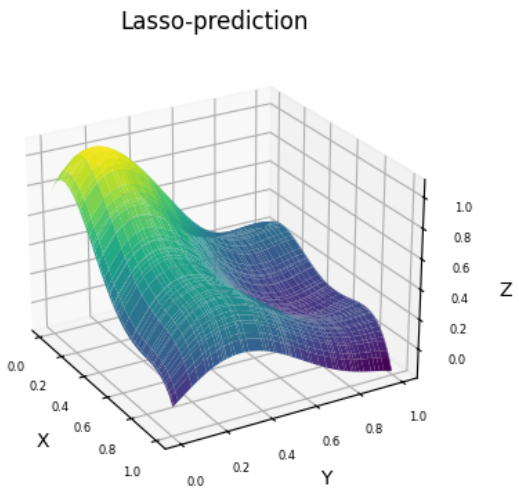


FIG. IV.9: Final prediction model for the Franke-function using the optimal parameters with the Lasso cost-function.

TABLE IV.3: Final error values after the regression analysis for the Franke-test function. Here, it is clear to see that the Lasso-method does worse than the two others, but all three methods performs very well.

| **Metric** | **OLS** | **Ridge** | **Lasso** |
|---|---|---|---|
| MSE | 0.0022 | 0.0022 | 0.0035 |
| $R^2$ | 0.9718 | 0.9718 | 0.9551 |

worse than the two others, but not significantly. The OLS- and Ridge-method performs equally well up to this number of decimal points. Visually comparing Fig.IV.8 to the Franke-surface in Fig.IV.1, there is an overall similarity, but there are some small differences. As an example, the prediction struggles at the edges of the domain, where the dataset does not show such steep slopes, but flatten out. The same was seen with the OLS-prediction, but for the model from the Lasso-method, the edges correspond better to the target. See App.B for details.

### D. Results from Topographical data-analysis

With the results from the Franke-function showing a good overall performance for the implemented methods, lets look into how the methods perform on the topographical data. The table below show the input variables used in the model selection for these two datasets. Many of the variables where kept equal to those used against the Franke-function. The model complexity was increased, with $p_{max} = 10$, as the target data had a much more complicated shape. The hope was to be able to follow the peaks and valleys more closely with a higher

TABLE IV.4: Input variable values for the regression analysis on the topographical data.

| **Variable** | **Value** | **Comment** |
|---|---|---|
| Domain $(x, y)$ | $[0, Nx] \times [0, Nx]$ | |
| Grid $(Nx, Ny)$ | $(21,\ 21)$ | Variable by choice of $N$ |
| Mod.complexity | $p \in [1, 10]$ | |
| Reg.parameter, $\lambda$ | $\lambda \in [10^{-10}, 10^{10}]$ | Regularization |
| Test size | 0.2 | |
| Iteration # | $n = 1000$ | For bootstrap |
| # of folds | $k = 8$ | For k-fold-method |

complexity. The following figures show the MSE and $R^2$-score for the Etnedal-dataset. From these figures, it can be seen that the MSE-values are very large. The reason for this might be that, since the MSE returns the sum over all squared differences, and the data in the dataset spans from zero to more than 1200, this metric will amplify these difference by the square. The $R^2$-score shown in Fig.IV.11 may there be a better way to show how well the regression-method did, as it subtracts a relative number from one. The two metrics do, at least, give the same basic indication on the optimal parameter range, which is shown to be around $p = 9$, and a $\lambda$-value around $1e^2$. A similar behavior was seen with the Lasso-analysis as well.

The table below shows the model selection parameters chosen to run the final prediction with. As the Jotunheimen dataset is more complicated than the Etnedal dataset, it is reasonable that the model complexity ended up somewhat higher for this dataset.

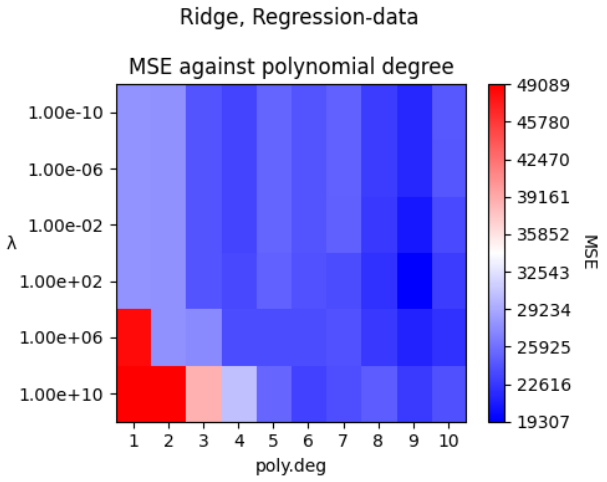


FIG. IV.10: This shows the MSE-values for the different parameters $\lambda$ against model complexity, $p$ from a Ridge-analysis of the Franke function. Blue parts indicate optimal range for the two parameters.

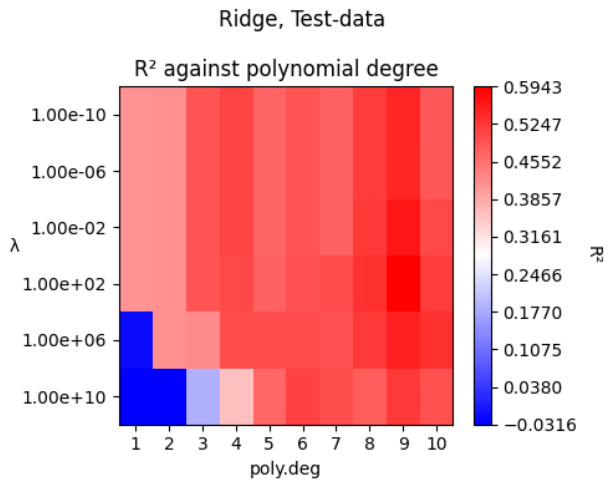


FIG. IV.11: This shows the $R^2$-score for the different parameters $\lambda$ against model complexity, $p$ from a Ridge-analysis of the Franke function. Red parts indicate optimal range for the two parameters.

TABLE IV.5: Final input-parameters after the regression analysis for the topographical datasets.

| **Variable** | **OLS** | **Ridge** | **Lasso** |
|---|---|---|---|
| **Etnedal-dataset** | | | |
| Mod.complexity, $p$ | 6 | 9 | 9 |
| Reg.parameter, $\lambda$ | – | $7 \times 10^3$ | $6 \times 10^1$ |
| **Jotunheimen-dataset** | | | |
| Mod.complexity, $p$ | 6 | 10 | 10 |
| Reg.parameter, $\lambda$ | – | $2 \times 10^3$ | $10^{-10}$ |

The analysis was performed with a higher model complexity as well, but $p > 12$ seemed to lead to overfitting. This is the reason for $p = 10$ as the limit. The regularization parameter for the Lasso-method coming out of the analysis is much lower than the Ridge-parameter. This has the effect that the Lasso cost-function tends towards the OLS cost-function. The final surface plot of the Lasso prediction model confirmed this.

Fig.IV.12, IV.13 shows the resulting prediction model by the Ridge-method on both datasets. Visually comparing these shows a slight resemblance, but the methods clearly struggle with the steep gradients, and are not able to replicate the sharp peaks and valleys. As with the Franke-predictions, they also also struggle at the edges of the domain.

The final table shows the resulting regression metrics. The Ridge-method clearly performs best out of the three, for both datasets. Here, the root-MSE (RMSE) is also shown, which is simply the root of Eq.(A.1), in an effort to scale the number down to be more relatable to the

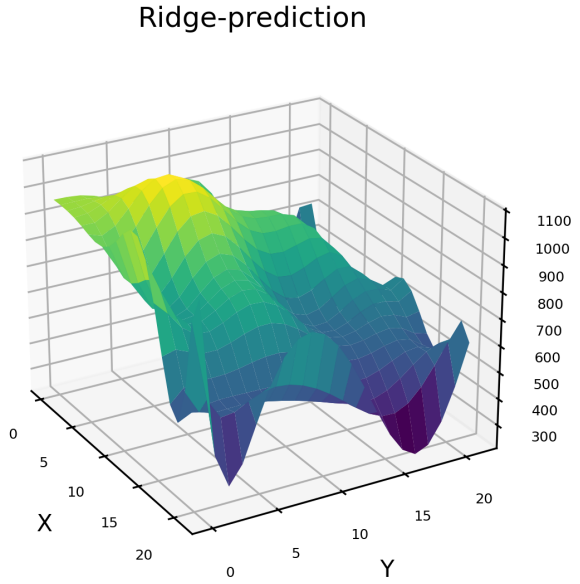


FIG. IV.12: Final prediction model for the Etnedal dataset using the optimal parameters with the Ridge cost-function.

scales of the dataset. The model misses by the same order of magnitude for both datasets, and looking at the RMSE, it is easier to see that the "only" corresponds to a maximum of 300 meters for the worst performing method. As this dataset spans from zero to 2100, it is certainly not insignificant, but also not too bad.

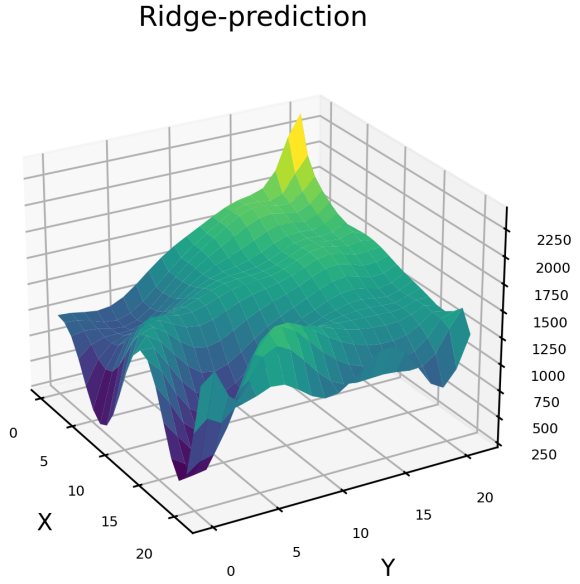


FIG. IV.13: Final prediction model for the Jotunheimen dataset using the optimal parameters with the Ridge cost-function.

TABLE IV.6: Final error values after the regression analysis for the topographical datasets.

| Metric | OLS | Ridge | Lasso |
|---|---|---|---|
| **Etnedal-dataset** | | | |
| MSE | 18557 | 14999 | 19250 |
| RMSE | 136.22 | 122.47 | 138.74 |
| $R^2$ | 0.6270 | 0.6985 | 0.6131 |
| **Jotunheimen-dataset** | | | |
| MSE | 85915 | 62253 | 90331 |
| RMSE | 293.11 | 249.51 | 300.55 |
| $R^2$ | 0.4563 | 0.6060 | 0.4284 |

## V. DISCUSSION

This section will go through some of the major points from the results section. In short, those are a comparison of method performance, how the different model parameters influence the results, and the effect that resampling had on the analysis.

### A. Method comparison

In the previous section, it was shown that the Ridge cost-function performed best on the more complicated datasets. But it was not necessarily better on the Franke-function. Looking at the two plots in Fig.IV.8, IV.9, the Ridge-prediction might replicate some of the gradients better, while the Lasso-model does a better job on the edges of the domain. This might be attributed to the how the regularization term in their respective cost-functions are defined.

The $L^1$-norm for Lasso has the effect that it might create a sharp cutoff for some of the larger regression parameters, $\boldsymbol{\beta}$ that forces them to zero. This penalty might smooth out some of the sharper gradients.

The $L^2$-norm for Ridge gives a continuous regularization on the larger regression parameters, but never forces them to zero. So the higher order terms in the polynomial model will always contribute some to the overall behavior of the model. This smoothing effect on the Lasso-prediction may be the reason why it struggled to replicate the topographical data, where the need to handle steep slopes is crucial to perform well.

### B. Model parameters

Another aspect that might model performance, is the different parameters. A common parameter to all the method is the choice of modelling function. This project has used a polynomial to fit data, and the performance is closed linked to this parameter, here denoted $p$. Lowering $p$ too much makes the model too general, as it makes a shape that can "easily" fit between a set of data points. The opposite happens for a too large $p$, making the model too specialized, such that it struggles on other, similar data.

As shown with Fig.IV.3, this effect came into play as $p \geqslant 8$. The greater the variance of the model points to a model that has been too specialized to a specific dataset, and the expected error will tend to increase together with $p$ across this level of complexity.

The other parameter used in the project is the regularization parameter, $\lambda$, in the Ridge- and Lasso-method. As seen in, f.ex., Fig.IV.4, IV.5 a smaller $\lambda$-value tended to give a smaller error and a better $R^2$-score. Combined with what Fig.IV.7 showed, which was that the $\beta$-values tended to increase in magnitude for increasing $p$, this indicates that penalizing the $\boldsymbol{\beta}$-values might help the fit.

A similar plot was done for the final $\hat{\boldsymbol{\beta}}$'s for the Franke-prediction. This can be found in App.B, and shows the regression parameters for the Lasso-method as a bit smaller than the two others. It may explain, again, why the prediction model look flatter than the other two, since the polynomial coefficients are smaller for Lasso.

### C. Effect from resampling

The presentation of the Bias-Variance trade-off results showed that, even for a fairly low sample size of a $15\times15$-grid, taking advantage of the bootstrap method to resample the training data gave good results. It showed that it was possible to obtain a low-bias-low-variance prediction for the OLS-method for the Franke-function. It also made it comparable to the other two methods.The other resampling method used in this project was the k-fold cross-validation. The scores obtained from this analysis on the Franke-function was shown in Fig.B.1, B.2.

Here, it is possible to see the effect mentioned above, where the Lasso cost-function will have a sharp cutoff for a certain regularization, while the Ridge cost-function has a smoother transition. The two plots show the same trend, and shows that the $\lambda$-values has a limit where the regression parameters are regularized too much, resulting in an increasing error. Choosing the parameter value close to this limit had a negative impact on the model performance.

### D. Choice of assessment metrics

During the assessment of the models created from the topographical data, it was noticed that the error values

returned by the MSE was very large, while the $R^2$-score was in the expected range. The reason for these large values was attributed to the definition of the MSE, which returns squared differences. As the datasets contains values in each $(x, y)$-point in the range $[0, 2100]$, as small deviation for the original dataset might be on the order of 100, or more. Squaring these values makes them even bigger. Therefore, choosing suitable metrics to assess the model might give less drastic values. Here, the choice fell on the RMSE, which brought the error values back to the original range, making the interpretation simpler.

## VI. CONCLUSIONS

This section will try to summarize the main parts of this report, and give some closing remarks with a critical view on the work.

This report has made an attempt on using implementations of three different linear regression methods to replicate two types of datasets of similar nature. The Franke-function was used as a verification to check the implementation against a fairly complicated, but well-known test function. The real-world data, on the other hand, can be seen as a validation of the methods and implementation, i.e. seeing how well the same methods would perform on a more real dataset.

The main trend from the analysis on the Franke-function is that all three methods, especially when done together with resampling techniques, were able to replicate the test dataset to a good approximation. All three had MSE's below 0.004 and got $R^2$-scores above 0.95 on the final model assessment with optimal parameters.

The limit on the maximum polynomial order of $p = 5$ seems a bit too low, considering the result for the Bias-Variance trade-off analysis, but keeping the complexity as low as possible is preferable, since this creates a more general purpose model for other types of functions of similar sort.

Regarding the validation of the methods, they did struggle a lot against the real data. It should be mentioned that the chosen datasets had very complicated shapes, and a lot of sharp gradients. It was therefore expected that the models, based on polynomials, would not perform that well since polynomials tend to create smoother shapes.

The Ridge-method performed best on these real datasets, and the final models do resemble the originals to a certain degree. One thing common to all three was that they seem to over- or undershoot a lot near the edges of the domain.

### A. Perspectives and Improvements

Looking ahead, one extension to this analysis can be to look into what hyperparameters that comes out as optimal by using the bootstrap method together with the Ridge- and Lasso-method. It is a fairly straightforward task, and it would be interesting to see if it agrees with the cross-validation.

Another idea would be to make use of other techniques for tuning the hyperparameter, $\lambda$. Now, the only tuning is with k-fold cross-validation, which seems to work well, but comparing other methods to what came out of that method would be interesting.

Lastly, since the topographical datasets had such steep slopes, trying to find other modelling approaches to use for the linear regression models might be worthwhile, as there might exist functions that handle these gradients better then the polynomial approach.

## REFERENCES


Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 1st edition, 2006. ISBN 978-0-387-31073-2.

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. URL `https://mml-book.com`.

Richard Franke. A critical comparison of some methods for interpolation of scattered data. *Calhoun: The NPS Institutional Archive*, 1979.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi:10.1038/s41586-020-2649-2.

T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN 9780387848846. URL `https://books.google.no/books?id=eBSgoAEACAAJ`.

Morten Hjorth-Jensen. Introduction to machine learning. `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html`, 2023. Accessed: 2024-09-27.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing, 2022.

### Appendix A: Accessing the programs

The Python-scripts and functions, as well as additional save figures generated during this project can be found in a GitHub-repository by following `https://github.com/andersthorstadboe/project-1-fys-stk4155-lin-regression`, which is a public repository. The programs and function-files are commented in a way such that a user with prior knowledge on linear regression analysis and their implementation, should be able to run them and understand the output. Using the programs requires that the user downloads and keeps the folder and file structure at is in the repository.

Some comments on the different files at the end here. The *project-01.02.py*-file runs a regression analysis on three different cases. It outputs figures and a suggestion on the optimal values for a final model. The *project-real-data-01.02.py* runs the same regression analysis on datasets stored in a separate folder. The output is also the same. The two supporting files are *reg_functions.py* and *support_funcs.py*. They contain the regression methods and other supporting methods for running the two main files.

### Appendix B: Additional results

The following figures shows some additional results from the analysis of the Franke-function and the topographical data. The figure caption should give relevant information on the contents and purpose of the figure.

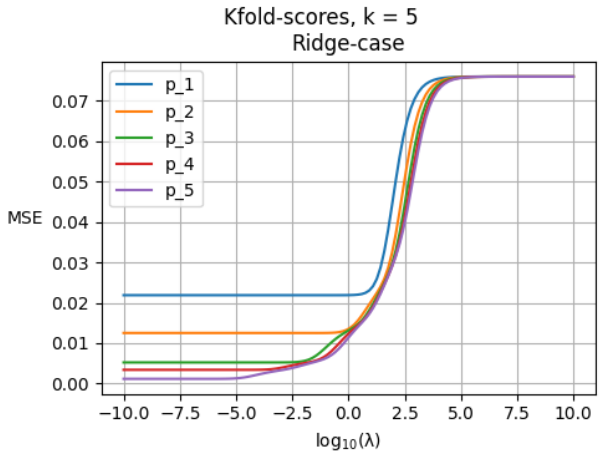


FIG. B.1: Shows the k-fold-scores for $k = 5$ the Ridge-method on the Franke-function. Notice the slope at around 2.5.

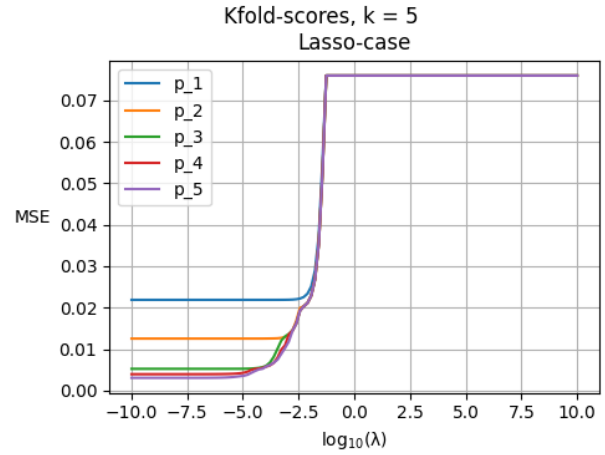


FIG. B.2: Shows the k-fold-scores for $k = 5$ for the Lasso-method on the Franke-function. Notice the sharp slope at around $-1$.

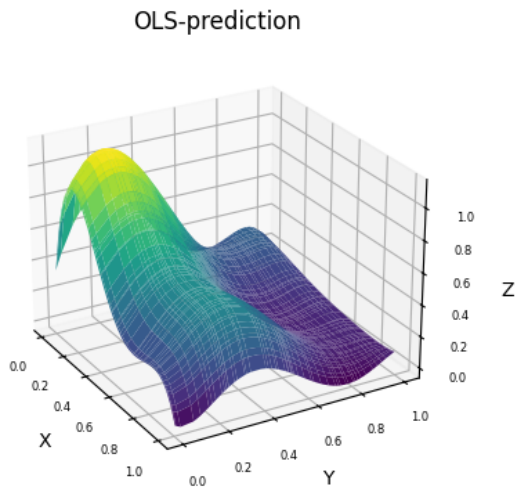


FIG. B.3: The final model of the Franke-function made with the OLS-method. It should be noticed that this is almost identical to the Ridge-model. This can also be seen by looking at the $\beta$-values for the final models. See Fig.B.9.

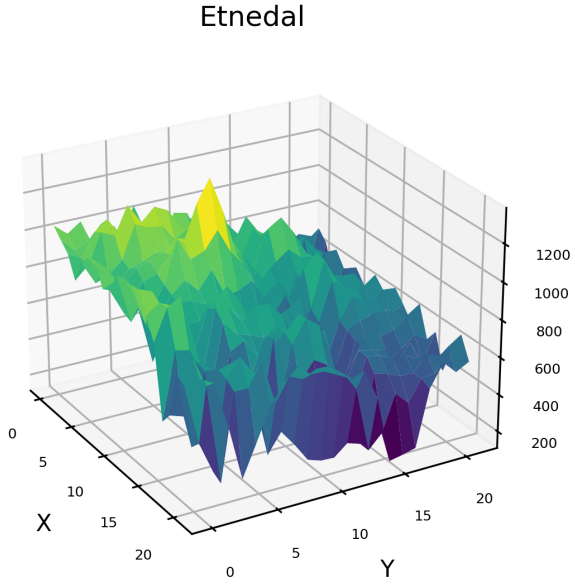


FIG. B.4: Rendering of topographical data downloaded from `https://hoydedata.no/LaserInnsyn2/` of the mountains in Etnedal, Norway.

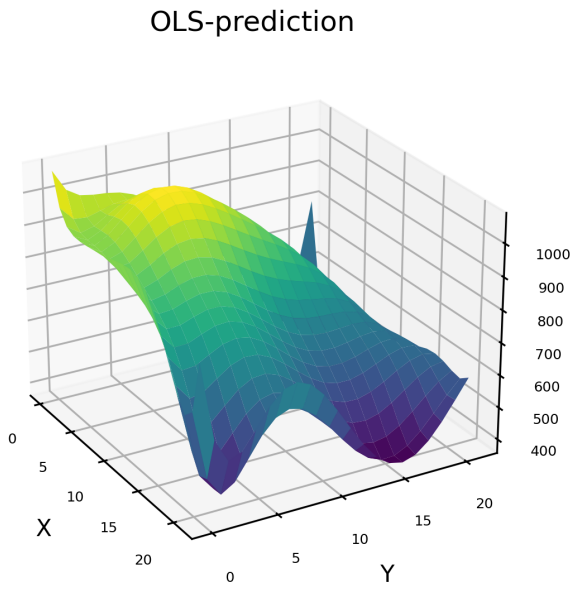


FIG. B.5: The final model from the OLS-method on the Etnedal dataset. As for the Lasso-method, it seems to struggle with the fluctuations in the data, and has created a smooth surface.

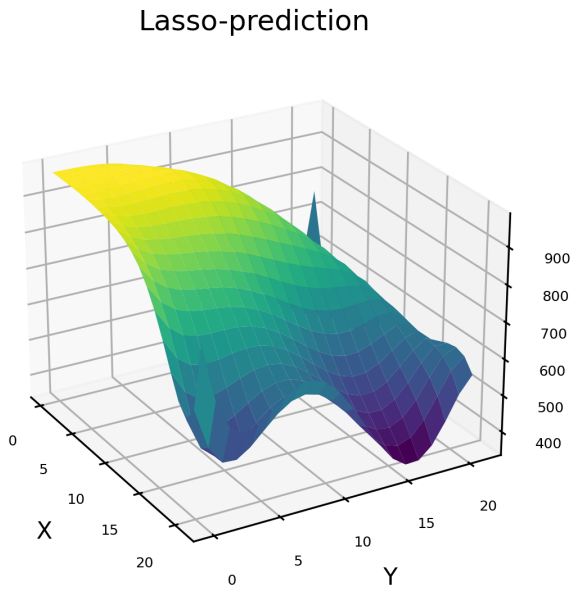


FIG. B.6: The final model from the OLS-method on the Etnedal dataset. As for the OLS-method, it seems to struggle with the fluctuations in the data, and has created a smooth surface.

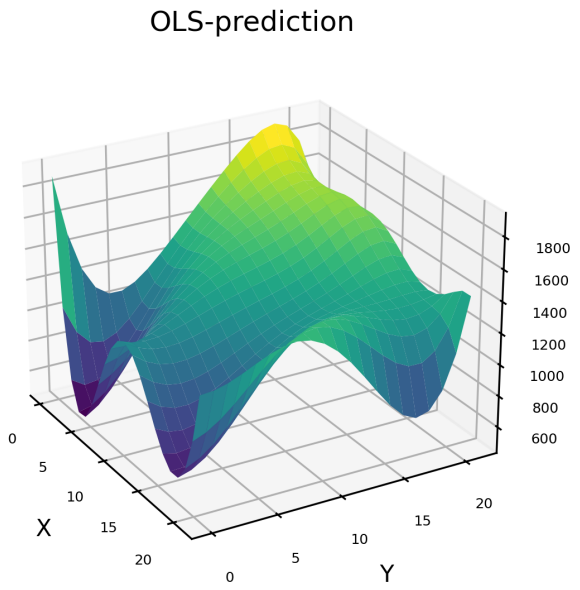


FIG. B.7: The final model from the OLS-method on the Etnedal dataset. As for the Lasso-method, it seems to struggle with the fluctuations in the data, and has created a smooth surface.

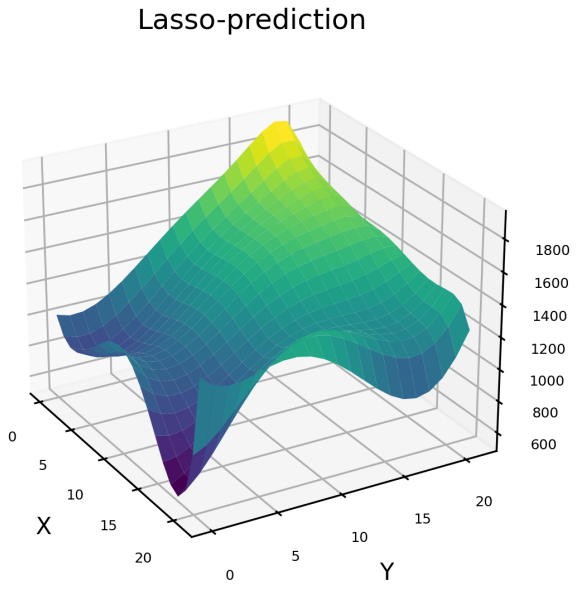


FIG. B.8: The final model from the OLS-method on the Etnedal dataset. As for the Lasso-method, it seems to struggle with the fluctuations in the data, and has created a smooth surface.

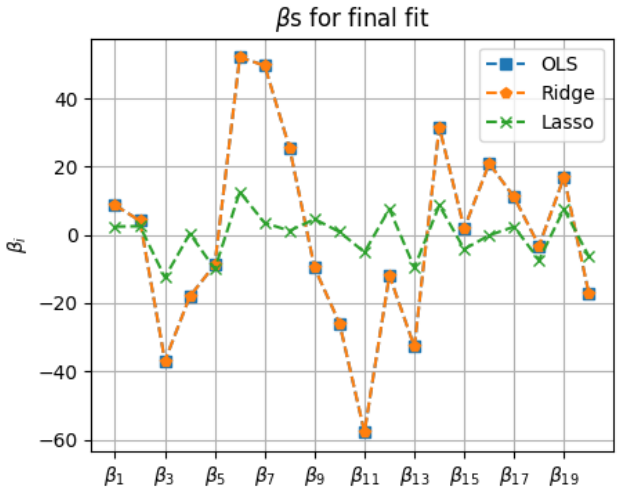


FIG. B.9: Showing the final $\beta$-values for the three model to the Franke-function. Notice that the OLS- and Ridge-values follow each other

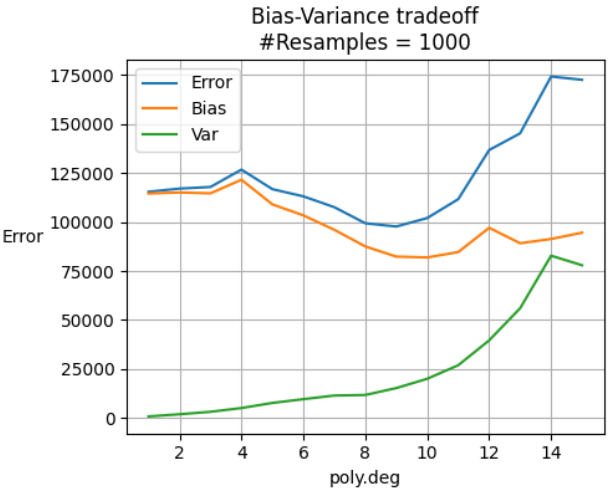


FIG. B.10: Showing the Bias-Variance trade-off after 1000 bootstrap-iterations using the OLS cost-function up to $p = 15$ on one of the topographical datasets. The bootstrap method has the expected effect, showing that there is a model complexity where the variance of the model starts to increase more rapidly.

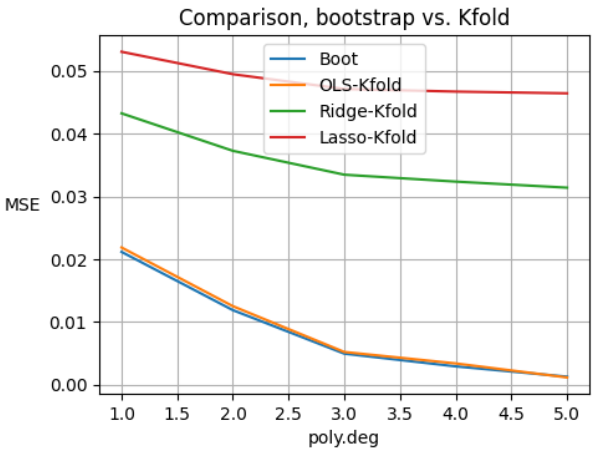


FIG. B.11: This figure shows a comparison between the bootstrap-result for the OLS-method, and the k-fold-scores obtained from the cross-validation for Ridge and Lasso on the Franke-function. It shows that the bootstrap method can help bringing the error down for a small dataset.