Preprint typeset using LaTeX style emulateapj v. 12/16/11

REGRESSION ANALYSIS AND RESAMPLING METHODS PROJECT 1

Per-Dimitri B. Sønderland and Vala M. Valsdóttir

Final version October 9, 2019

ABSTRACT

In this project two datasets where analysed with Linear regression more precisely Ordinary Least Squares, Ridge and Lasso regression. One dataset is generated from the Franke function and the other is terrain data of the Oslo fjord. With the Franke function there was added different values for an error term to get a better understanding of how these regression method work. Resampling with K-fold Cross-Validation and Bootstrap was also performed and the methods evaluated after using these. For high noises it seems that Ridge and Lasso are better at predicting a function than the Ordinary Least Squares when it comes to the Franke function. While for the terrain data the best fit is made by the Ridge and Ordinary Least Square.

Subject headings: Machine Learning — Linear Regression — Ordinary Least Square — Bias-Variance trade off — Cross Validation — Ridge Regression — Lasso Regression

1. Introduction

In recent years there has been an increase in the use of Machine Learning (ML) algorithms, these algorithms originate from statistics and have been used long before ML became a field of study. Today big datasets have become a regular component in most sciences and therefore scientists require a set of tools to deal with all the information at hand. This is why ML has become the word on everybody's mouth the past years. However, the problem most scientist encounter is the uncertainty in knowing if the computer is actually predicting correctly. This is why one needs to perform a statistical analysis of the errors obtained by using these algorithms. If one decides to use ML it is important to have enough data to feed to the program, if the dataset is big enough the program can than try to predict what should come next. This is very useful in many different fields like medicine, cyber security, finance, mathematics and of course also physics. When learning about ML for the first time one usually starts with Linear Regression, Ridge Regression and Lasso Regression which are the most basic supervised learning techniques. The Regression family include the Ordinary Least Squares (OLS) and Singular Value Decomposition (SVD) which are methods used for predicting data points. The original data is then split in test and training data and the statistical errors often obtained by using resampling methods like cross-validation or bootstrap. By doing so a scientist can then evaluate if these algorithms are giving the right prediction needed for the problem at hand. In this paper all the above methods are used on the well known Franke function and then also on real digital terrain data, together with statistical analysis of the errors obtained. This should give a good indication of how these algorithms and methods work for both known functions and real data and thus help to the understanding of how ML works in practice.

2. Theoretical background

Let $X \in \mathbb{R}^p$ be a real valued random input vector, and let $Y \in \mathbb{R}$ be a real valued random output variable, also called the response variable. The basis of our discussion is that we seek a function $f(X)$ that helps us to predict the variable Y, given the values x we feed in as the input to X. To formulate the theory we need a way to evaluate how close our input dependent function $f(X)$ is to our output, that is, to what extent does it predict Y. An intuitive first step is to simply subtract the one from the other $Y - f(X)$. We call this our *error*. It's worth noting that the error is the deviation between observed output variable and the true valued unobservable function. While a *residual*, which we will encounter later, is the difference between the observed output and the *fitted* function. We take into account our response data points $y_i \in Y$, by taking the mean $\mathrm{E}(Y - f(X)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))$. Note that uppercase X denotes the generic aspects of the variable, and lowercase x means observed value. Our expression is almost done, but requires a last edit. By minimizing the sum we might obtain differences that are equal but opposite in signs such that the terms cancel, e.g. $(y_1 - f(x_1)) = -(y_2 - f(x_2))$. In that case we might minimize our total error, but not necessarily be working with the function that is optimal in terms of prediction.

p.d.b.sonderland@fys.uio.no, vala.m.valsdottir@fys.uio.no
[1] Department of Physics, University of Oslo, P.O. Box 1048 Blindern, N-0316 Oslo, Norway

To avoid this problem we can either take the absolute value or the square value of our error. While both methods are perfectly possible and come with different advantages and disadvantages, we shall focus on the latter, namely taking the squared value

$$C\left(Y, f\left(X\right)\right) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f\left(x_i\right)\right)^2. \tag{1}$$

We call this equation the cost(or loss) function. We wish to find the $f\left(X\right)$ that minimizes this expression. The solution to the optimization problem is $f(x) = \mathrm{E}(Y|X = x)$, which is known as the *regression function*. Thus the best point of Y at any point $X = x$ is the conditional expectation value. Which is the mean of Y given $X = x$. (5, p. 18)

2.1. Linear Regression and Ordinary Least Squares

A linear regression continues the aforementioned discussion, but assumes that the regression function $f(x) = \mathrm{E}(Y|X)$ is linear with respect to its inputs. It is common to assume a linear relationship between X and Y when there is no prior knowledge on the functional relationship $f(X)$, or if one can infer that it's a reasonable approximation. Linear models are also easier to fit than non-linear ones and are therefore more pedagogical to start with. The linear regression model takes the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j\beta_j \tag{2}$$

where the input vector $X_j^T = (X_1, X_2, \ldots, X_p)$ is called *the design matrix*. The specific method we describe in this section is the *Ordinary Least Squares(OLS)* method, which is the most widely used method of linear regression. We are now assuming we don't know the underlying real function, and we want to create an estimation model in its place. Given a set of empirical values of data we want to pick the coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$ that minimize residual sum of squares, which is expressed by our cost function. We begin by inserting equation 2 into our cost function 4, and obtain (we ignore the denominator henceforth as it cancel in the derivation of the expression)

$$C\left(\beta\right) = \sum_{i=1}^{n}\left(y_i - f\left(x_i\right)\right)^2 \tag{3a}$$

$$= \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 \tag{3b}$$

$$= \underbrace{\sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p} x_{ij}\beta_j\right)^2}_{(y_i-\tilde{y}_i)^2=(\varepsilon_i)^2} \tag{3c}$$

$$= \sum_{i=1}^{n}\left(\varepsilon_i\right)^2 \tag{3d}$$

where we have moved the intercept into the sum. The whole term inside the bracket, ε_i, is called the residual. Hence the whole expression is the residual sum of squares. Before we minimize we rewrite the equation in a more convenient matrix form

$$C(\boldsymbol{\beta}, \boldsymbol{X}) = \left\{\left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right)\right\} \tag{4}$$

Next we minimize the cost function,

$$\min_{\beta\in\mathbb{R}^p}\left\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right\}. \tag{5}$$

This is done by taking the derivative

$$\frac{\partial C}{\partial\beta} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta), \tag{6}$$

and setting it equal to zero

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0. \tag{7}$$

We then obtain the unique solution by solving for β

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{8}$$

This last step requires an inversion of the moment matrix $\mathbf{X}^T\mathbf{X}$ (1) (5) (4)

2.1.1. Variance

From the results obtained in this section a covariance matrix for $\boldsymbol{\beta}$ can be derived.

$$\mathrm{Var}(\boldsymbol{\beta}) = \sigma^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1} \tag{9}$$

The diagonal of such a matrix is the variances of which shall be useful in this article. σ^2 depends on which distribution our underlying data has. For a sample this can be approximated by

$$\sigma^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}\left(y_i - \tilde{y}_i\right)^2 \tag{10}$$

2.2. Perfect Multicollinearity

In some cases the moment matrix $\mathbf{X}^T\mathbf{X}$ is singular, meaning that the inversion of the matrix is impossible. This corresponds to two or more rows and columns being linearly dependent. In statistics, this phenomenon is called *perfect multicollinearity*(or *collinearity)*. Essentially the moment matrix is in this case equivalent to a reduced form which is not a $n\times n$ matrix, where $n\times n$ is a requirement for inversion. An equivalent statement is that the *determinant*$(\mathbf{X}^T\mathbf{X}) = 0$, thus checking the determinant is a way to test for this.

2.3. Singular Value Decomposition

To work around the problem of a singular matrix and perfect multicollinearity one can use a *Singular Value*

Decomposition (SVD) algorithm. The SVD of a $m{\times}n$ matrix $\mathbf{X}$ has the form

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{11}$$

We have that $\mathbf{U}$ has the form $m{\times}m$, and $\mathbf{V}$ the form $n{\times}n$. Both $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. D has the form $m{\times}n$ and is a diagonal matrix with non-negative numbers. To find $\mathbf{U}$ take the matrix product $\mathbf{X}\mathbf{X}^T = \mathbf{W}_1^{m\times m}$. Use the eigenvalue-equation $\mathbf{W}_1\mathbf{v} = \lambda\mathbf{v}$, and find the unique eigenvalues by $\det(\mathbf{W}_1 - \lambda\mathbf{I}) = 0$. Insert the eigenvalues back into the eigenvalue-equation and pick the most convenient set of eigenvectors, which then makes an eigenspace. Next orthonormalize the matrix corresponding to this eigenspace(this might require a method such as the Gram–Schmidt process) resulting in $\mathbf{U}$. $\mathbf{V}$ can be found by doing the same procedure, but with $\mathbf{X}^T\mathbf{X} = \mathbf{W}_2$. Both $\mathbf{W}_1$ and $\mathbf{W}_2$ will always have the same eigenvalues. The square roots of these eigenvalues are then inserted into the diagonal of the $\mathbf{D}$-matrix in descending order from $D_{1\times 1}$ to $D_{m\times n}$. In numerical calculations with packages such as those available in python, the procedure above is enhanced by various techniques throughout many of the steps.

(7)(8)

2.4. Ridge Regression

2.4.1. Multicollinearity

A related problem to perfect multicollinearity is *multicollinearity* where, although the moment matrix is invertible, there exists two or more variables that are approximately linearly dependent. When there are many correlated variables in a linear regression model, their coefficients(β_i) can become poorly determined, and exhibit high variance. Such a problem can materialize in the form of a wildly large positive coefficient one one variable being canceled by a comparably wildly large negative coefficient on a correlated variable. It is thus hard to asses the relative importance of the independent variables in terms of the dependent variables, which is essentially our goal with linear regression.

2.4.2. Overfitting

Another similar problem is when our coefficients become too tightly trained on the training data or simply noise in the data such that the amount of coefficients we have is also in some sense unruly. Simply put, in the case of overfitting we have more coefficients than what should have been deduced and justified by the data at hand.

To alleviate the problem of wild and unruly coefficients we now introduce a size constraint on the coefficients

$$\boldsymbol{X}^T\boldsymbol{X} \to \boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I} \tag{12}$$

Such a constraint is called a *regularizer*. We first look at the *Ridge regression* technique, where we add to the least squares loss function a regularizer defined as the L2 norm of the parameter vector we wish to optimize over. The L2 norm is defined as $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$. We start with the OLS cost function 4, but with the added constraint

$$C(\boldsymbol{\beta}, \boldsymbol{X}, \lambda) = \left\{\left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right)\right\} + \lambda\beta^T\beta \tag{13}$$

This is essentially the *Lagrangian multiplier method*, where the Ridge cost function(13) is the *Lagrangian function*. The optimization problem that leads to the formulation of the Lagrangian function is the OLS cost function(3b) subject to $\sum_{j=1}^{p}\beta_j^2 \leq t$. Next we minimize equation 13 and formulate it in terms of the L2 norm to capture the fact that we're working with the Ridge method

$$\min_{\beta\in\mathbb{R}^p} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2. \tag{14}$$

By the same procedure as with OLS(6) we obtain an expression for the coefficients

$$\beta^{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{15}$$

(1)(5)(4)

2.5. Lasso Regression

The Lasso regression introduces a regularizer like the Ridge, but with a subtle difference. This time around we have the OLS cost function subjected to the constraint $\sum_{j=1}^{p}|\beta_j| \leq t$. The L2 Ridge penalty $\sum_1^p \beta_j^2$, is now replace by the L1 Lasso penalty $\sum_1^p|\beta_j|$. The minimization problem for Lasso, which is analogous to 14, now looks like the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \tag{16}$$

where L1 norm is defined as $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$. A lot can be said on the difference between Ridge and Lasso regression, but to be short the most important take away is that the nature of the shrinkage of the coefficients are different. Ridge regression will shrink them asymptotically towards zero, while Lasso regression allows coefficients to become exactly $= 0$. (1)(5)(4)

2.6. Evaluation of the model

To evaluate the model we need a way to check its performance. We introduce the biased cost function *Mean Squared Error*(*MSE*), and the *Coefficient of Determination*(*R2*).

$$MSE(\hat{y}, \tilde{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{17}$$

$$R^2(\hat{y}, \tilde{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \mathbb{E}(y))^2} \tag{18}$$

and where

$$\mathbb{E}(y) = \frac{1}{n}\sum_{i=0}^{n-1} y_i \tag{19}$$

Both the MSE and the R2 describe how well our model estimate $\hat{\tilde{y}}$, fit the observed data $\hat{y}$. The original cost function we minimized(3a), and thus made unbiased, is also an MSE. But unlike the minimized cost function, the MSE we're dealing with here is biased. The MSE is always non-negative and the closer the value is to zero the better our model performs in estimating the actual data. The R2 measures how well our model performs relative to the average which in two dimensions would simply be a flat line through the data points. The closer R2 is to 1 the better it performs relative to the average. A score equal to zero means that it is as good as the average, and a negative score means that the average would be a better estimate than the model at hand.

2.7. Bias-Variance Tradeoff

We can include two more values in evaluating some important aspects of a model - *The Bias and the variance.*

- The Bias is the difference between the actual value of what we're trying to predict and the expected value of the prediction. A model with high bias will have little concern for the training data and will have tendency to oversimplify the model. It will both gives us a high error on the training and test data. This phenomenon go by the name of underfitting.

$$\mathrm{Bias}[\tilde{y}] = \mathbb{E}[(y - \mathbb{E}[\tilde{y}]) \tag{20}$$

- The Variance is a measure of the variability of the prediction. A model with high variance will pay too much attention to the training data. It's thus obviously well performing on the training set and will give low errors, but it does not generalize well and will perform poorly on the test data(unseen data). In this case the model suffers from the phenomenon of overfitting

$$\mathrm{Var}[\tilde{y}] = \mathbb{E}\left[\tilde{y}^2\right] - \mathbb{E}[\tilde{y}]^2 \tag{21}$$

When the MSE is used on data that was not used to estimate the model it is called *Mean Squared Prediction Error(MSPE)*, or simply *expected test MSE.* When the expectation is taken across many test sets the MSPE can be decomposed into three terms

$$MSPE = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] \tag{22}$$

In the following this is shown mathematically: First assume that our model is generated from a noise model

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon} \tag{23}$$

with the noise normally distributed, with $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\boldsymbol{\varepsilon}) = \sigma^2$. Our predictor $\tilde{\boldsymbol{y}} = \hat{f}_S(\boldsymbol{x})$ has been trained on the data S. The variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are from our training set. Further assumptions is that ε is independent of S and that all variables $\boldsymbol{\varepsilon}, S, \boldsymbol{x}, \boldsymbol{y}$ are random variables. For simplicity $\boldsymbol{\varepsilon}$ denotes both $\varepsilon_{training}$ and ε_{test}. In the following section we will ignore boldface typing of the vectors for readability.

$$\begin{aligned} MSPE &= \mathbb{E}\left[\left(y - \hat{f}_S(x)\right)^2\right] \\ &= \mathbb{E}\left[y^2\right] + \mathbb{E}\left[\hat{f}_S^2(x)\right] - 2\mathbb{E}\left[y\hat{f}_S(x)\right] \end{aligned} \tag{24}$$

Next we add $\mathbb{E}^2[y] - \mathbb{E}^2[y]$ and $\mathbb{E}^2\left[\hat{f}_S(x)\right] - \mathbb{E}^2\left[\hat{f}_S(x)\right]$, and use the formula for the variance 21, but with different variables.

$$\begin{aligned} MSPE = {} & \mathrm{Var}(y) + \mathbb{E}^2[y] + \mathrm{Var}\left(\hat{f}_S(x)\right) + \mathbb{E}^2\left[\hat{f}_S(x)\right] \\ & - 2\mathbb{E}\left[f(x)\hat{f}_S(x)\right] \end{aligned} \tag{25}$$

where the last term comes from inserting equation 23 into the last term in 24: $\mathbb{E}\left[y\hat{f}_S(x)\right] = \mathbb{E}\left[\left(f(x) + \varepsilon\right)\hat{f}_S(x)\right] = \mathbb{E}[\epsilon]\mathbb{E}\left[\hat{f}_S(x)\right] + \mathbb{E}\left[f(x)\hat{f}_S(x)\right] = 0 + \mathbb{E}\left[f(x)\hat{f}_S(x)\right]$. Terms with the expected value of ε will henceforth be ignored. Continuing we'll need the relations

$$\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y] + \mathrm{Cov}(x, y) \tag{26a}$$
$$\mathrm{Var}(x + y) = \mathrm{Var}(x) + \mathrm{Var}(y) + 2\,\mathrm{Cov}(x, y) \tag{26b}$$
$$\mathrm{Var}(x - y) = \mathrm{Var}(x) + \mathrm{Var}(y) - 2\,\mathrm{Cov}(x, y). \tag{26c}$$

On the right side in equation 25 we insert the noise model 23 into the first and second term. We use relation 26b on the first term and $\mathbb{E}^2[y] = \mathbb{E}^2[(f(x) + \varepsilon)] = \mathbb{E}^2[f(x)]$ on the second. Then relation 26a is used on the fifth term to unpack the expected value, resulting in

$$\begin{aligned} &= \mathrm{Var}(f(x)) + \mathrm{Var}(\epsilon) + \mathbb{E}^2[f(x)] + \mathrm{Var}\left(\hat{f}_S(x)\right) + \mathbb{E}^2\left[\hat{f}_S(x)\right] \\ &-2\mathbb{E}[f(x)]\mathbb{E}\left[\hat{f}_S(x)\right] - 2\,\mathrm{Cov}\left(f(x), \hat{f}_S(x)\right). \end{aligned}$$

The variances and covariance of the f and $\hat{f}_S$ terms are collected by applying relation 26c, giving

$$\begin{aligned} &= \mathrm{Var}\left(f(x) - \hat{f}_S(x)\right) + \mathrm{Var}(\epsilon) + \mathbb{E}^2[f(\boldsymbol{x})] + \mathbb{E}^2\left[\hat{f}_S(x)\right] \\ &-2\mathbb{E}[f(x)]\mathbb{E}\left[\hat{f}_S(x)\right] \end{aligned}$$

Finally we end up with

$$\begin{aligned} &MSPE = \\ &\left(\mathbb{E}[f(x)] - \mathbb{E}\left[\hat{f}_S(x)\right]\right)^2 + \mathrm{Var}\left(f(x) - \hat{f}_S(x)\right) + \mathrm{Var}(\epsilon). \end{aligned} \tag{27}$$

If for simplicity sake we assume that the sample is fixed in advanced, that is, deterministic, then we can rewrite $\mathbb{E}[f(x)] = f(x)$. This means that the variance simplifies

$$\begin{aligned} &\mathrm{Var}(f - \hat{f}_S) = \mathrm{Var}(f) + \mathrm{Var}(\hat{f}_S) - 2\,\mathrm{Cov}(f, \hat{f}_S) \\ &\quad = \mathrm{Var}(\hat{f}_S) + \mathrm{E}\left[(f - \mathbb{E}(f))^2\right] - \mathbb{E}[(f - \mathbb{E}[f])(\hat{f}_S - \mathbb{E}[\hat{f}_S])] \\ &\quad = \mathrm{Var}(\hat{f}_S) \end{aligned}$$

as the last two terms have parentheses where $\mathbb{E}[f(x)] - f(x) = 0$. Our simplified MSPE then becomes

$$MSPE = \mathbb{E}(f - \mathbb{E}[\hat{f}_S])^2 + \mathrm{Var}(\hat{f}_S) + \mathrm{Var}(\epsilon). \tag{28}$$

The the added $\mathbb{E}$ expectation on the first term gives the same expression as in 27 because of $\mathbb{E}[f(x)] = f(x)$ and $\mathbb{E}(\mathbb{E}(\hat{f}_S)^2) = \mathbb{E}(\hat{f}_S)^2$. The latter follows from relation 26a. The terms in 28 correspond to

$$MSPE = \text{Bias}^2 + \text{Variance} \ + \ \text{Irreducible Error} \tag{29}$$

The bias and variance here is explained by the bullet-points at equation 20 and 21. The *Irreducible Error* is basically the noise. It gives us the effect of the observed noise. It is irreducible in the sense that it does not hinge on anything but the underlying noise-distribution.

(6)(1)(5)

3. Method

In this paper we take a closer look at the Franke function and do a thorough regression analysis on the data obtained by it. Then we look at real terrain data and use our knowledge, gained from the regression analysis on the Franke function, to fit the regression models to the terrain data.

The Franke function looks like

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ &+\frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ &-\frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned} \tag{30}$$

and in our project we call $f(x,y)$ for z. This is a function in 3D and therefore a good start to analyse this before the terrain data which is also in 3D. In addition to the Franke function there is an error term giving us $z = \tilde{z} + \epsilon$, this term plays a big role in the analysing of the data points gained from the Franke function. As discussed to great extent in the theory section 2 we need to create a design matrix so the β-coefficients can be obtained. In this project we use polynomials of different degree to fit both the Franke function and the terrain data. This is done for the two variables x and y, as an example we show the second degree polynomial for two variables

$$\hat{\mathbf{X}} = \begin{bmatrix} 1 & x_0 & y_0 & x_0^2 & x_0y_0 & x_0^2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1}y_{n-1} & x_{n-1}^2 \end{bmatrix}. \tag{31}$$

After making a design matrix the β-coefficients can be obtained by the three different regression models that we are looking at in this paper

$$\hat{\beta}^{OLS} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{32}$$

$$\beta^{\text{Ridge}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{33}$$

In the case of Lasso we used the library Scikit-Learn from Python to find the β coefficient. The equation for the β for Lasso and Ridge looks the same which can be confusing but as discussed before there is subtle difference between them. For choosing the β coefficients resampling methods are also taken into use, in this project K-fold cross-validation and Bootstrap.

3.1. K-Fold Cross-Validation

Whilst doing a linear fit on a dataset there is a problem with knowing how good the fit actually is. To solve this problem one can use resampling methods to obtain additional information about the fitted model. This is done by dividing the dataset into a training and a test set, further it is drawn samples from the training set and refitted to the model at hand. If this is done for a linear regression fit one can repeatedly draw different samples from the training data and do a fit to each one and then examine the result of each fit. Using a method in this manner will give more information about the fitting model which would not be obtained by only doing the fit once. There are many resampling methods the bootstrap, jackknife, the blocking method and k-fold cross-validation. In this paper the k-fold cross-validation was used for the purpose of testing the linear fit of the data. Cross-Validation is used to estimate the test error associated with the learning method so one can evaluate its performance and the level of flexibility, or in other words, model assessment and model selection (1). The reason for the importance of resampling is the need for estimating the expectation values and their possible sources of errors. When not knowing if the expectation value is correct resampling helps to indicate how accurate they actually are. Cross-Validation with k-fold is thus used for this particular experiment because it provides a reasonable way of splitting the data and therefore prohibits the multiple usage of samples. This avoids the unbalanced influence on the model building and the prediction evaluation. The k-fold splits the data into k number of subsets, in our case $k = 5$ more or less equally sized. One of the sets is the test set and the rest is then used as a training set. Thereafter, one loops over the set for $k = 5$ and for each loop the test set changes from one to another while the rest of the sets are used as training set on that particular test set. For each fold the model is fitted and the mean square errors are computed. One then get to evaluate these errors for five different test sets on different training set within the original dataset. To illustrate how the k-fold cross validation works in practise we borrowed a figure from (2).

It is also important to shuffle the dataset randomly before it is split into k groups. This is important for the learning of the model, without the shuffle the training

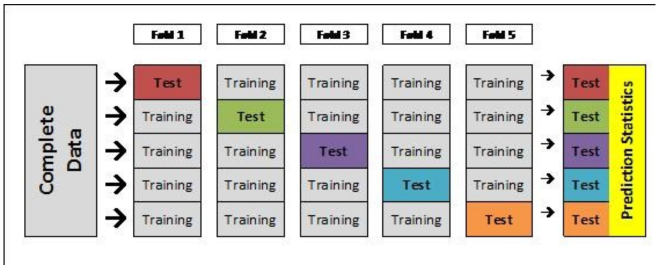

FIG. 1.— Illustration of how the K-fold Cross-Validation works in practise borrowed from Golden Helix Blog (2)

data and test data are two separate sets of data, after being split into k number of folds. Therefore the learning of the model becomes difficult and the errors will get higher. If the data is shuffled randomly this will make the model better and the k-fold will give more accurate errors. The implementation of k-fold in the code of this experiment will be thoroughly examined in the section Code and Implementation 4.

3.2. Bootstrap

Bootstrap was used whilst obtaining the results for the Bias and Variance Trade-off. The main difference between this and the k-fold is that the bootstrap does indeed open for the multiple usage of the samples. Bootstrap picks n variables out of the dataset z, then we use these variables to compute z^* by evaluating z under the observation of the variables picked out(1). This was a better method to use in this particular problem since the k-fold cross-validation method calculates the MSE for each fold then the mean of the MSE is taken and we look at these when comparing our models. With the Bias and Variance terms we wish to look at these for each fold and this proved complicated to achieve in a meaningful way. Therefore the k-fold method was replaced by bootstrap in this part of the analysis.

4. Code and Implementation

In this experiment exploring the possibilities of ML with regression, two datasets where evaluated, the data generated from the Franke function and digital terrain data downloaded from //earthexplorer.usgs.gov/. To be able to do an evaluation of both dataset there where made different files with codes *franke.py*, *Bootstrap_Franke.py* and *regression_SRTM_data.py*. *Bootstrap_Franke.py* only includes the code for the bootstrap method used to obtain the values for the Bias Variance Trade-off. The other two files contain mostly the same sets of codes but the evaluation of the terrain data and the Franke function are quite different and therefore these codes where in seperated files. The code in *franke.py* is build out of many small functions which are fed into bigger functions that plots th figues needed to do a sufficient analysis.

In the function *Errorbars* computes β coefficient for the model chosen with confidence intervals for each coefficient, and then $plot_Errorbar$ plots these for different polynomials. The data is then split into train and test data in the function *Test_Train_Reg*, to split the data a function called *train_test_split* from Sckikit-Learn, which is a free software machine learning library for Python, is applied. The function splits 80% into training data and 20% into test data. In the *Test_Train_Reg* function one can chose which model to use, OLS, Ridge or Lasso. This function is manly used when computing the plots without resampling method being used which are done in the codes *Plot_nthPoly_regular* and *Plot_nthLambda_nthPol_Regular*. To achieve more correct results for the errors (MSE and R^2-score) cross-validation is applied in $Kfold_hm$, where hm stands for home-made. There is a Kfold code in Scikit-Learn which splits the data, but for the purpose of understanding the machinery behind, a home made code was considered better in this experiment. The home made K-fold function shuffles the data randomly for then to spit it in k number of splits using the function *np.split*() which is from the Python library numpy. Then one loops over number of splits, in our case splits = 5, where the train design matrix $X_{train} = X_{ksplits}$ is made. Furthermore *np.delete*() is applied to delete the split from the training set that will be used as a test set. The same is also done with the output function z. One can choose the model one wants to use, for example if one chooses OLS the $beta_{train}$ is calculated using matrix inversion, if one chooses Ridge it is calculated with $(1+\lambda)^{-1}\beta_{OLS}$ and for Lasso the Scikit-Learn function which we will address more later. The β is then used to compute the predicting function $z_{predict}$ and z_{test}, in the end the mean square error is calculated giving a list of these for each fold. Below there is a excerpt of the home-made k-fold code. This code was used for the terrain data as well, and gave good results that are further discussed in the Result and Discussion sections 5 ??.

```
def Kfold_hm(X,z, lamb):
    #Model tells us if we are using OLS or SVD-Ridge or Lasso

    #shuffling the data
    shuffle_ind = np.arange(X.shape[0])
    np.random.seed(seed)
    print(np.random.randint(10))
    np.random.shuffle(shuffle_ind)

    Xshuffled = np.zeros(X.shape)
    zshuffled = np.zeros(X.shape[0])
    for ind in range(X.shape[0]):

        Xshuffled[ind] = X[shuffle_ind[ind]]
        zshuffled[ind] = z[shuffle_ind[ind]]


    X_k = np.split(Xshuffled, splits)
    z_k = np.split(zshuffled, splits)


    MSE_train = []
    MSE_test = []
    for i in range(splits):

        X_train = X_k
        X_train = np.delete(X_train, i, 0)
        X_train = np.concatenate(X_train)

        z_train = z_k
        z_train = np.delete(z_train, i, 0)
        z_train = np.ravel(z_train)

        X_test = X_k[i]
        z_test = z_k[i]

        if (lamb == 0):
            beta_train = np.linalg.pinv(X_train.
    T.dot(X_train)).dot(X_train.T.dot(z_train))

        else:
            beta_train = Ridge_hm(X_train,
    z_train,lamb)

        z_tilde = X_train.dot(beta_train)
        z_predict = X_test.dot(beta_train)

        MSE_train_i = MSE(z_tilde, z_train)
        MSE_test_i = MSE(z_predict, z_test)

        MSE_train = np.append(MSE_train,
    MSE_train_i)
        MSE_test = np.append(MSE_test,
    MSE_test_i)

    return MSE_test, MSE_train, z_test,
    z_predict
```

Furthermore there are functions to give the plots needed. $Plot\_nthPoly\_error_{M}ean$ plots the MSE or the R^2-score computed from the home-made Kfold function for different polynomials. This gives a picture of how well the regression models predict, and also an indication of when they start to overfit or underfit. Overfitting is when the algorithm fits the data to well and underfitting is when the algorithm fits the data to poorly. It is important to know how much complexity is needed for achieving the best fit. In the function *Plot_nthLamdas_error_Mean* the plots for the MSE or the R^2-score for Ridge and Lasso regression are obtained for different λ. Last part of the code is the function *Plot_nthPoly_nthLamdas*, it shows plots for different λ against the complexity of the model. In *regression_SRTM_data.py* the terrain data is further explored to detail. Here one finds all the same elements that are present in *franke.py*, the main difference is that we are not interested in the same plots as when doing the regression analysis. Here we mainly want to see how well OLS, Ridge and Lasso fit the actual terrain data. To be able to make the best fit one has to find the optimal polynomial for OLS and for Ridge and Lasso the best polynomial combined with the optimal λ-value. This we have learned how to do in our the analiysis of the Franke function, so we simply use these codes to find the optimal parameters.

When inverting the matrix $\mathbf{X^TX}$ the function np.linalg.pinv() was used instead of np.linalg.inv(), taken from the Python numpy library. The function np.linalg.inv() does indeed invert a matrix which has $det(\mathbf{X^TX}) = \mathbf{0}$, but from reading the theory 2 one knows that it is not possible to invert a matrix with determinant zero. Therefore np.linalg.pinv() was chosen instead which solves this by using SVD according to our understanding of the information obtained from the official website for the numpy library: https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.linalg.pinv.html. Pinv stands for pseudo-inverse of a matrix and applying this function instead should give more reasonable results for when the matrix is not invertible, which happens for polynomials higher than 10 for the *franke.py* and also occurs in the terrain data. Below is a table which shows the polynomial degree and the determinant of $\mathbf{X^TX}$ for the two cases *franke.py* and the terrain data.

TABLE 1
This shows the $det(\mathbf{X^TX})$ for Franke and the Terrain data for Linear Regression for different polynomials $p = 0, 1, 2, 3....10$. For polynomial higher than $p = 9$ the matrix becomes singular.

p	$det(\mathbf{X^TX})$ Franke	$det(\mathbf{X^TX})$ Terrain
	Linear Regression	
0	399.99	399.99
1	442225	542936
2	5886237	13373960
3	3724.825	28984.23
4	4.1632e-07	2.5208e-05
5	2.8790e-26	3.7839e-23
6	4.0405e-57	3.9477e-52
7	3.4971e-102	1.068e-94
8	5.185e-164	2.555e-153
9	3.2912e-245	1.6595e-230
10	0	0

This shows that it is indeed necessary to apply np.linalg.pinv(), the data also shows better results using this function, this is discussed further in the results and discussion sections 5 **??**. All the home made codes talked about here have been checked against the functions from the Scikit-Learn library. In the code one can find these and check our results if need be.

5. Results: Franke Function

The following section is devoted to the results obtained from the regression analysis of the Franke function. First of all the Franke function is analysed through applying three different regressions; Linear, Ridge and Lasso. The coefficients β obtained by these methods and their errors are further investigated with confidence intervals. MSE and the R^2-score are important factors in evaluating the fit of the models and therefore are plotted for all the three models and represented here. For the best fit the MSE has to be low and the R^2-score as close to 1 as possible. Lasso and Ridge has an extra parameter to consider λ so in the subsections for these two one has to consider the models for different values of this parameter as well. In the plots there is also added a different noise-terms to the Franke function to see how well the models fit with for different values these.

5.1. Results for Linear Regression with OLS

The coefficients β is an important part of the puzzle to be able to fit the OLS to the Franke function like previously explained in the theory section 2. In OLS $\beta = (\mathbf{X^TX})^{-1}\mathbf{Xz}$ and the goal is to find the coefficient β that minimizes the cost function, which results in the

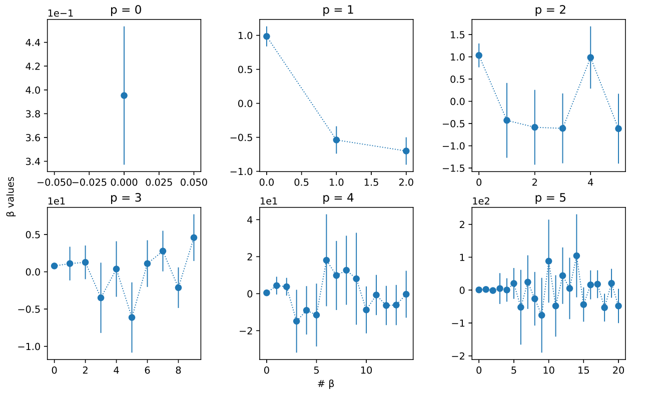

FIG. 2.— Figures of the coefficient β with confidence intervals for polynomials $p = 0, 1, 2, 3, 4, 5$ with noise $\sigma^2 = 0.5$.

OLS fitting best to the the data. So in the evaluation of the coefficient β one also has to take into account the degree of the polynomials. In figure 5.1 we look at the values of the β coefficients obtained by the OLS for $p = 0, 1, 2, 3, 4, 5$ with noise $= 0.5$. The aim is that the coefficient fluctuate as little as possible, so that the mean value of each point is more or like the same. This means that the OLS is fitting these parameters as close to the data points as possible. Another thing to consider is the errors in each coefficient β that comes from the noise factor, or in other words the error in our approximation. Since the Franke function is a known function we add an error which is normally distributed with a variance $\sigma^2 = noise = 0.5$ in this case. In the case of a unknown function this is not necessary since there will be an error occurring natural in the fitting process, but in the case of a known function the OLS will fit the data perfectly if an error term is not added. So the noise, in this study of the Franke function, is simply to get experience with handling Regression and its possible flaws by adding this term. Back to figure 5.1, if the confidence intervals of the coefficients are to great it might indicate that it is not the best fit even though the parameter do not fluctuate much. This is because we want to find the best line that describe our data points but if the model is fitting to well to the noise term it will result in greater errors in the coefficients β. As one can see from the figure, the greater the polynomial the less fluctuation, and taking into account the error bars, the figure indicates that the best choice is polynomial $p = 3$ or $p = 4$. The noise obviously plays a big part in these results and a greater noise added to the original function can confuse the OLS and make it overfit the data, this will be demonstrated further whilst presenting the results for this experiment.

In figure 3 one sees how big the role of the error term is, here the variance of the noise is $\sigma^2 = 0.1$ and one can observe from the figure that the error bars are therefore smaller than for $\sigma^2 = 0.5$. Again the fluctuations are quite similar to figure 5.1, so the best choice is still polynomials $p = 3$ or $p = 4$.

Further, in the illustration of the point for the error term, in figure 4 the variance of the noise is set to $\sigma^2 = 0.01$. Here the error bars are almost vanishingly small for polynomials up to $p = 3$, and it is therefore clear that for this low noise $p = 3$ is the best fit of the data for the OLS. All these figures are obtained without any resampling methods or splitting of data, this is just to evaluate how the regression model produces coefficients and how well they initially fit to the data points.

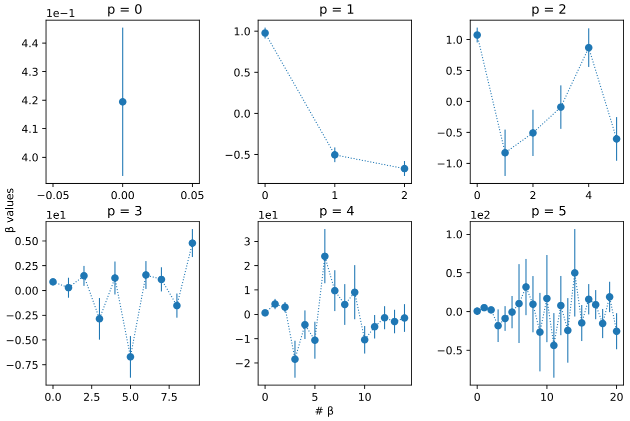

FIG. 3.— Figures of β with confidence intervals for polynomials $p = 0, 1, 2, 3, 4, 5$ with noise $\sigma^2 = 0.1$.

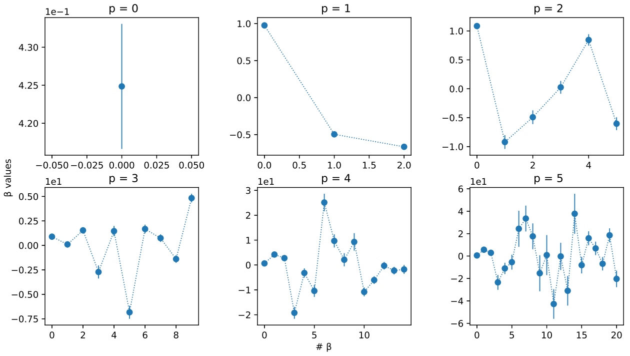

FIG. 4.— Figures of β with confidence intervals for polynomials $p = 0, 1, 2, 3, 4, 5$ with noise $\sigma^2 = 0.01$.

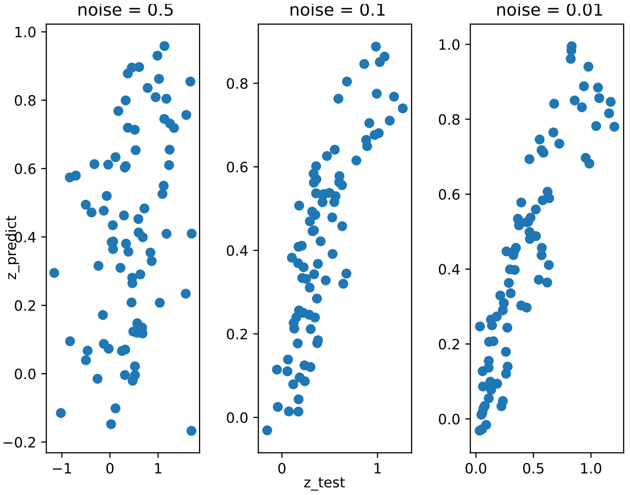

FIG. 5.— Figures of the relationship between $z_{predict}$ and z_{test} for different noise $\sigma^2 = 0.5, 01, 0.01$.

Next in the analysis of the Franke function we split our data into train and test and evaluate how the model predicts points with the use of this splitting method. The design matrix and the data points obtained by the Franke function are split into X_{train}, X_{test}, z_{train} and z_{test}.

Then the coefficients β are produced by calculating the minimum of $(\mathbf{X}_{\mathbf{train}}^{\mathbf{T}}\mathbf{X}_{\mathbf{train}})^{-1}\mathbf{X}_{\mathbf{train}}^{\mathbf{T}}\mathbf{z}_{\mathbf{train}}$. These coefficients are used in $z_{predict}$, which is supposed to predict new data points for the Franke function. Figure 5 is made to illustrate how well the $z_{predict}$ function predicts the data points in relation to the z_{test} function. Here, again we see how the OLS relies on our choosing of the noise term, for low noise we have more aligned points and for higher noise the points are more scattered. That means that when the noise is low the $z_{predict}$ function is able to predict the points of the test data well, but for higher noise the $z_{predict}$ function gets confused and is not able to predict as well as we wish to.

5.2. Results for Linear Regression with OLS plotted for MSE and R^2-score

The Mean Square Error (MSE) and the R^2-score are important parts in the analysis of the results, it tells us more about how the regression models fit to the data points when we introduce splitting and resampling to our project. Splitting the data into train and test and studying the MSE from the two sets tells us if the model is overfitting or underfitting to the data points as talked about before in previous sections. In the underlying subsection we will investigate these theories and try them out on our own data and see if they behave accordingly.

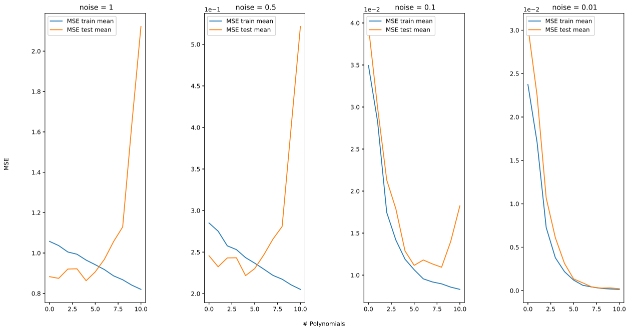

Fig. 6.— Figure of the MSE plotted against different degrees of polynomials up to $p = 10$. This is the MSE train mean, MSE test mean without resampling for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

In figure 6 we look at the MSE plotted against different degree of polynomials, and how it behaves after being split into train and test data for different noises without resampling. As one can see from the figure the train data (blue line) and the test data (orange line) can be said to evolve with the noise going from high to low. With high noise, $\sigma = 1$, the train data quickly starts to overfit the data points and rises to high MSE even for low polynomials. With smaller noise $\sigma = 0.1$ the model behaves more like the theory discussed previously: The two sets underfit the data for lower polynomials but at $p = 5$ the test data starts overfitting the dataset and shoots up, whilst the training data reaches it saturation point around $p = 9$ since it has no more points to train on. The aim when looking at this figure is to find the polynomial for which the test and train data are neither underfitting or overfitting the data, this is often called the optimum area where many precise data are being fitted. For low noise this area is obtainable and when the noise is as low as 0.01 we see that train and test data reach low MSE quite quickly for low polynomials and stabilizes as the complexity rises.

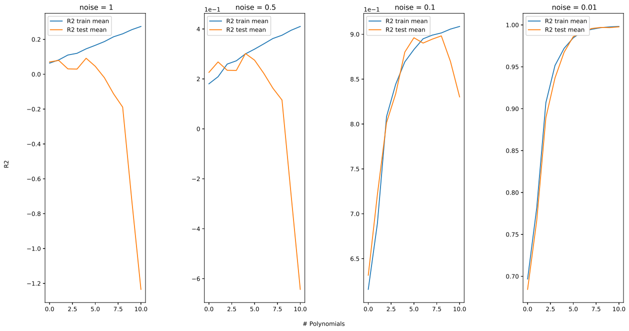

Fig. 7.— Figure of the R^2 plotted against different degrees of polynomials up to $p = 10$. This is the R^2 train mean, R^2 test mean without resampling for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

To get a better understanding of what is actually happening we can look at the R^2-score for rising complexity and different noises. The R^2-score = 1 describes a perfect fit to the model. For the noise = 1 the R^2-score for the train set becomes negative, this simply means that the it is fitting badly to the data. For noise = 0.1 the R^2-score for the training set rises to 0.9 and the test set follows until it dives for higher polynomials. When the noise is as low as 0.01 both the training and test sets almost reach R^2-score = 1 indicating that the lower noise makes the OLS fit best to the data. Both figure 6 and 7 are without resampling with k-fold, this means that what we are seeing here is just one validation and therefore insufficient to say if the OLS is giving the accurate and precise estimates.

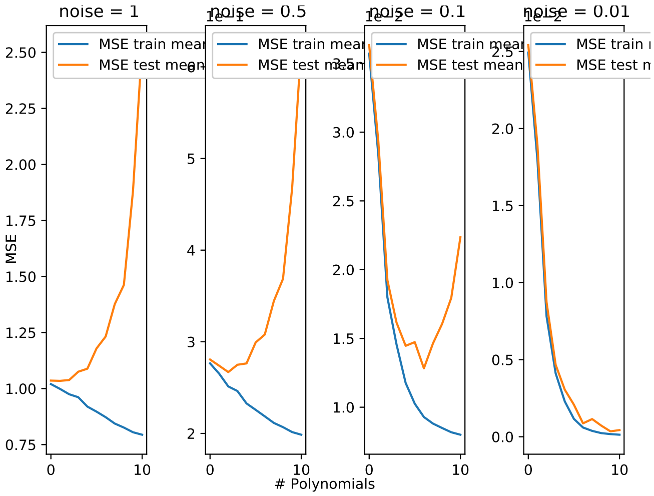

Fig. 8.— Figure of the MSE plotted against different degrees of polynomials up to $p = 10$. This is the MSE train mean, MSE test mean with resampling for OLS with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

Resampling methods are the backbone for validating our results, without it the bias and variance trade-off, which is the ultimate way to see how accurate and precise our model is, gives little to no answers that can be used in a scientific setting. A scientific experiment needs to be set up so others can reproduce the results multiple times, and resampling is a way to actually reproduce results multiple times in one run of a code. Figure 8 shows the MSE plotted for rising polynomials up to $p = 10$ with the resampling method k-fold cross-validation with five folds, and for different noises. The difference from previously is that after splitting the data into train and test, we run it through the k-fold cross-validation code and get a new MSE for each run, then the mean for each run is calculated and plotted for rising complexity. The result is that we are able to evaluate our model more accurately. The goal is to find the optimum area and from figure 6 for noise $= 1$ it might look, for the untrained eye, that it actually has a optimum area where the test data dives down to MSE $= 0.9$ before it rises and overfits, but when evaluated with k-fold cross-validation we quickly see that the MSE starts in 1 and never goes under this value. This means that the bias reduction, by increasing complexity, never sufficiently compensates for the variance increase. This is because of the variance of the noise $\sigma = 1$ dominates. Therefore the test data always overfits the points. Evaluating the model multiple times shows that for the bias and variance trade-off theory to give a clear optimum area is for the error variance term to be $\sigma = 0.1$. For $\sigma = 0.01$ the test data settles with the train data up to $p = 10$, from theory it is expected to rise and start to overfit but this might occur for higher polynomials than checked in this experiment.

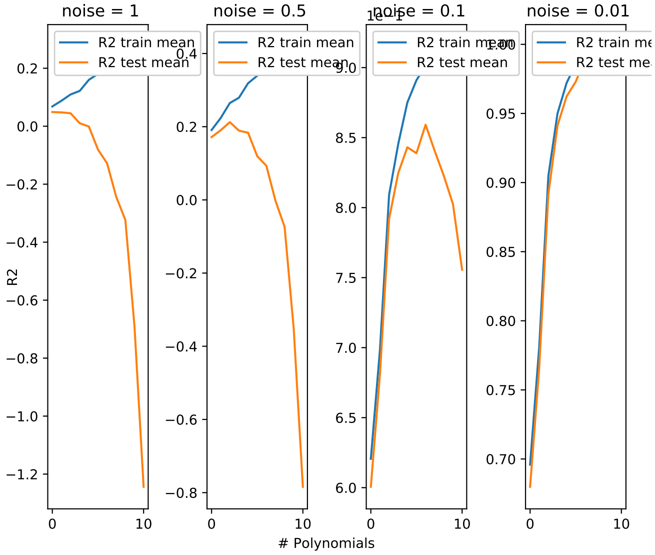

Fig. 9.— Figure of the R^2 plotted against different degrees of polynomials up to $p = 10$. This is the R^2 train mean, R^2 test mean from the home-made K-fold code with OLS, for noise $\sigma^2 = 0.5, 0.25, 0.1, 0.01$.

The R^2-score for the same case, OLS with resampling k-fold cross-validation, is evaluated in figure 9 for different noise. Here we see the same picture as with the MSE in figure 8 but from a different angle, the R^2-score indeed also indicates that $\sigma = 1$ confuses the OLS and gives a poor fit to the real data points in the Franke function.

5.3. Results for Ridge Regression

In the evaluation of Ridge Regression model there is not only the coefficient β one needs to take into account, it is also the parameter λ, this makes Ridge different from OLS which makes the indication that the $(\mathbf{X^TX})^{-1}$-term is not singular. With Ridge one is able to obtain reasonable results for singular matrices. The results obtained for this model is hereby presented.

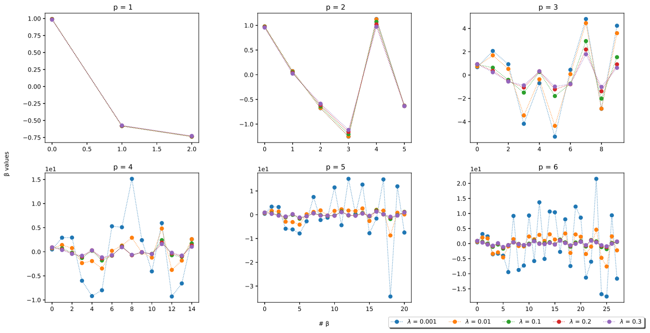

Fig. 10.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of λ for the Ridge regression with noise $\sigma^2 = 1$.

As for OLS the coefficent β for different polynomials are looked at but now also for different λ parameters this can be seen in figure 5.3. Again little fluctuation is preferable but the difference is how the β coefficients behave for different values of the parameter λ. In this figure the variance of the noise is $\sigma = 1$. The error bars where not included in these figures this is because the figure shows different values of λ which gave many error bars and made the figure very hard to read. The main goal of these figures is to show how Ridge regression with different values of λ affects the fluctuations of the β coefficients. for $\lambda = 0.3$ (the purple line) one can see that for higher polynomials it forces the β coefficients to zero in this case. Even if little fluctuation is preferable the goal is not that the coefficients are zero, this will give $\tilde{z} = \mathbf{X}\beta = \mathbf{0}$. The goal for Ridge is to show that there is an actual solution by adding the λ parameter. By looking at this figure the main hint is that small values of λ might be preferable and this will show it self to be true later on in the evaluation of these results. Furthermore, one sees that all the values of λ preform the same for lower polynomials, the difference is for higher polynomials where one sees that there is a difference for the values, $\lambda = 0.001$ looks like the best choice for $p = 4$ and $p = 5$ at first glance.

Figure 5.3 and 5.3 shows the same plots for the variance of the noise being $\sigma^2 = 0.5$ and $\sigma = 0.1$. These show mostly the same behaviour, if there was error bars this would show that the errors where smaller since the only error in the β coefficients are generated from the noise term. For lower noise one also sees that there is a difference between the values of λ for lower polynomials. For noise with variance $\sigma^2 = 0.1$ one can see a small difference in the fluctuation in the coefficients β compared to higher noise, the mean value of the points are smaller here.

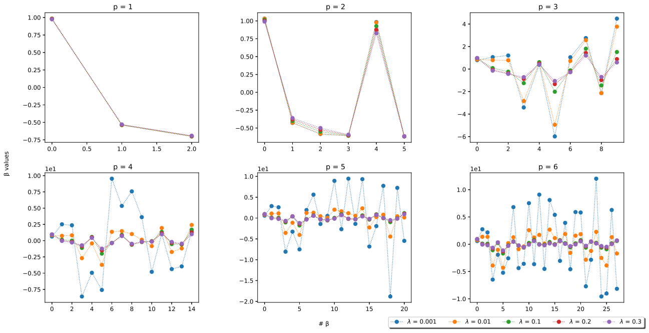

FIG. 11.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of λ for the Ridge regression with noise $\sigma^2 = 0.5$.

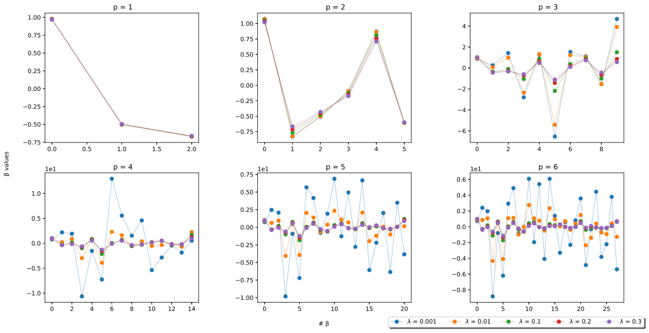

FIG. 12.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of λ for the Ridge regression with noise $\sigma^2 = 0.1$.

5.4. Results for Ridge regression plotted for MSE and the R^2-score

In the following subsection the MSE and R^2-score is evaluated for Ridge regression with and without resampling k-fold cross-validation. Firstly the MSE and R^2-score are plotted against different values λ for a fixed polynomial for then to be plotted for rising polynomials and fixed values of λ.

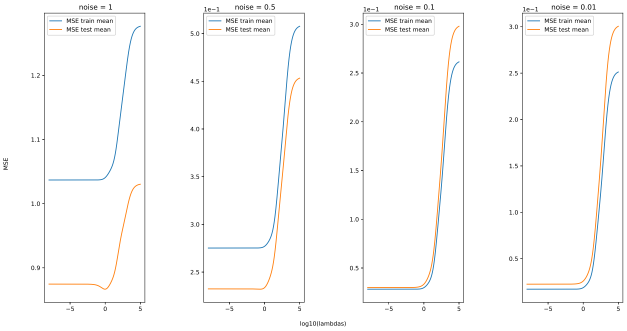

FIG. 13.— Figure of the MSE plotted against different $\log(\lambda)$ for polynomial = 2. This is the MSE train mean, R^2 test mean for Ridge without resampling for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

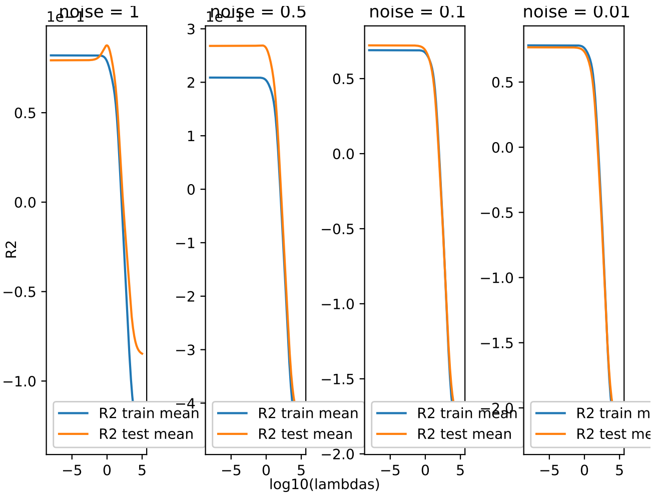

FIG. 14.— Figure of the R^2-score plotted against different $\log(\lambda)$ for polynomial = 2. This is the R^2 train mean, R^2 test mean for Ridge without resampling for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

For better understanding the figures in this section the tables 2 and 3 are included in the discussion. These tables show the values of the λ_{min} and λmax for the MSE and the R^2-score. In figure 13 one can see the MSE plotted for different $log(\lambda)$ for polynomial $p = 2$ without resampling and the same for figure 14 but in that case for the R^2-score. If one compare these to the figures with resampling 15 and 16 one sees that the train set is shifted. Without resampling the train set starts at higher MSE values than the test set, this is shown foremost for the highest noise term. The figures without resampling illustrate well why we need to evaluate our model by the use of resampling. What these figures are suppose to show is the lowest MSE for a λ-value for the polynomial at hand, so we know which value to use for the λ parameter for that exact polynomial to get the best fit.

TABLE 2
THE VALUES OF λ_{min} FOR MSE WITH RIDGE REGRESSION WITH AND WITHOUT RESAMPLING

σ^2	λ_{min} for MSE	$\log(\lambda)$ for MSE
Ridge without resampling for $p = 2$		
1	0.937	-0.0281
0.5	0.2222	-0.6533
0.1	1e-08	-8.0
0.01	1e-08	-8.0
Ridge with resampling $p = 2$		
1	1.265	0.1022
0.5	0.430	-0.367
0.1	0.108	-0.966
0.01	0.115	-0.94
Ridge with resampling $p = 4$		
1	0.282	-0.549
0.5	0.0711	-1.148
0.1	0.00015	-3.81
0.01	5e-05	-4.30

The tables 2 and 3 show these values for λ for the MSE and R^2-score and as one sees without resampling for $p = 2$ these give the exact same values, this is not the case with resampling but the values are still close. When evaluating the model with k-fold for five splits this means that the MSE and the R^2-score are evaluated five times by shuffling with a random seed and this may effect their values and therefore give slightly different values for λ_{min} and λ_{max} for MSE and R^2. Even so, by evaluating the

TABLE 3
THE VALUES OF λ_{max} FOR R^2-SCORE WITH RIDGE REGRESSION WITH AND WITHOUT RESAMPLING

σ^2	λ_{min} for R^2	$\log(\lambda)$ for R^2
Ridge without resampling for $p = 2$		
1	0.937	-0.0281
0.5	0.2222	-0.653
0.1	1e-08	-8.0
0.01	1e-08	-8.0
Ridge with resampling $p = 2$		
1	1.427	0.154
0.5	0.456	-0.341
0.1	0.0801	-1.096
0.01	0.0903	-1.044
Ridge with resampling $p = 4$		
1	0.300	-0.523
0.5	0.063	-1.200
0.1	0.00016	-3.780
0.01	5e-05	-4.30

model only once like we do without resampling we can not be certain the values are precise or accurate, and as seen from the table with resampling give slightly higher values for the λ-values than without resampling, and by looking at the figures one also sees that the test data follows the train data more accurately with resampling. Without resampling the points in the train data and the test data are split once and can therefore be perceived as two different data sets, while with resampling the data is shuffled and all of it used as training set and all of it used as test sets at some point in the process.

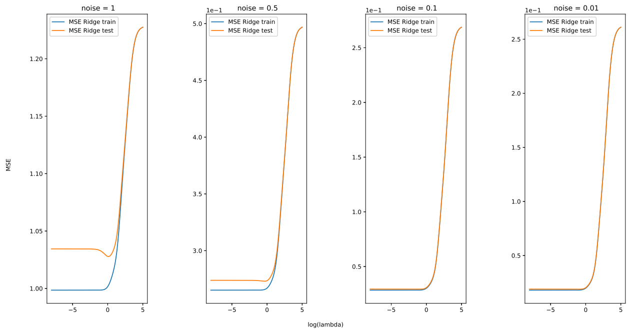

FIG. 15.— Figure of the MSE regression plotted against different $\log(\lambda)$ for polynomial = 2. This is the MSE train mean, MSE test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

For low noises one sees from figures 17, 18 that the λ-values are lower than for 15, 16. This can be explained by the polynomial degree being lower in the latter case and therefore requires higher λ-values to be able to make the best fit for the Ridge regression. Often in the case of real data and not a known function the λ-values are chosen to be small. λ purpose is merely to tweak the coefficients β, therefore it makes good sense that they are not supposed to be high values.

In figure 19 and 20 we see the MSE and R^2-score plotted against model complexity for set values of λ with noise $\sigma^2 = 0.1$. For the observant eye one sees that the best train set for given λ for $p = 2$ and $p = 4$ actually is the same as in tables 2 and 3.

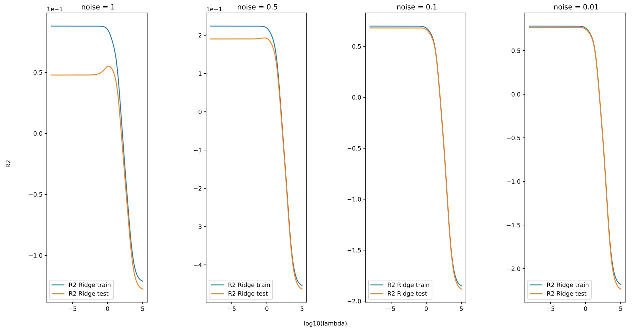

FIG. 16.— Figure of the R^2-score plotted against different $\log(\lambda)$ for polynomial = 2. This is the R^2 train mean, R^2 test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

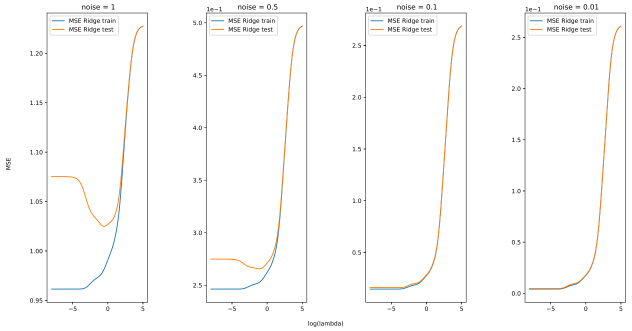

FIG. 17.— Figure of the MSE regression plotted against different $\log(\lambda)$ for polynomial = 4. This is the MSE train mean, MSE test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$ .For $\sigma^2 = 1$ the minimum $\log(\lambda) = -0.548$ and for $\sigma^2 = 0.5$ the minimum $\log(\lambda) = -1.146$

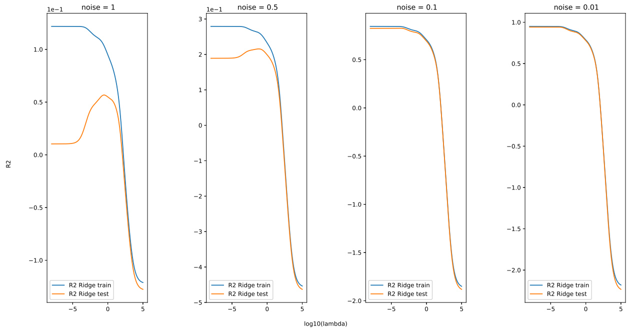

FIG. 18.— Figure of the R^2-score plotted against different $\log(\lambda)$ for polynomial = 4. This is the R^2 train mean, R^2 test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

For figure 21 and 22 one sees clearly how the variance of the noise comes into play. The test data predicts very poorly even with resampling. The test lines (dashed) should follow their respective train lines (full), but this is not the case when the variance of the noise is this great. Even though Ridge looks like it performs poorly for the variance of the noise being $\sigma^2 = 1$, it still gives small values for the MSE, but the R^2-score is also small almost zero for some values of λ, indicating that it is not performing to well. It might be a hint that the fitting

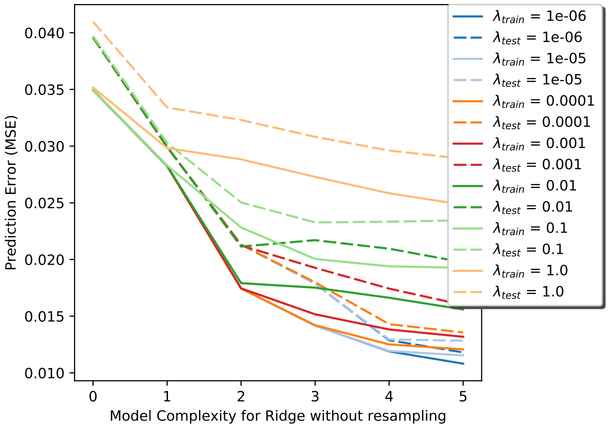

FIG. 19.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean without resampling for Ridge for noise $\sigma^2 = 0.1$. Number of $\lambda = 1$ then np.logspace was used for 10^{-6} to 1 with step 5.

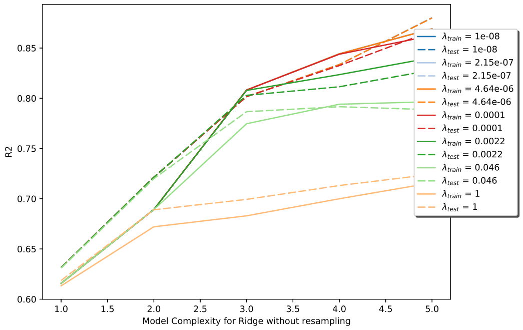

FIG. 20.— Figure of the R^2-score plotted against the complexity and for different values of λ. This is the R^2-score train mean, R^2-score test mean without resampling for Ridge for noise $\sigma^2 = 0.1$.

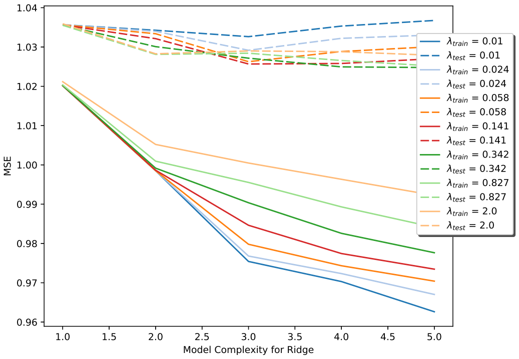

FIG. 21.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean with resampling for Ridge for noise $\sigma^2 = 1$. Number of $\lambda = 10$ then np.logspace was used for 10^{-2} to 10 with step 5.

with polynomials for the Franke function is not the best solution though this will not be explored further in this experiment. Again for the observant eye, the best λ for $p = 2$ and $p = 4$ do coincide with tables 2 and 3, which is a further indication that the best λ found for the rest of the polynomials also are more or less correct at least to the right order.

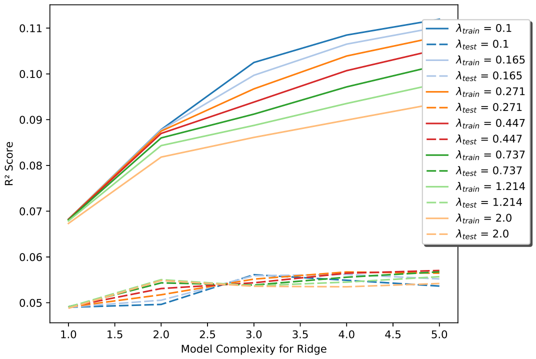

FIG. 22.— Figure of the R^2-score plotted against the complexity and for different values of λ. This is the R^2-score train mean, R^2-score test mean with resampling for Ridge for noise $\sigma^2 = 1$.

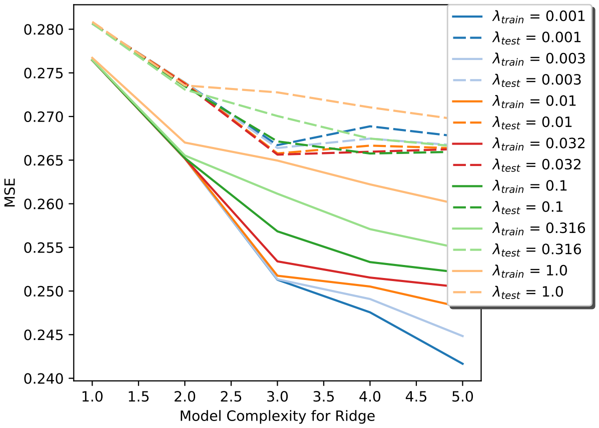

FIG. 23.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 0.5$. Number of $\lambda = 10$ then np.logspace was used for 10^{-2} to 10 with step 5.

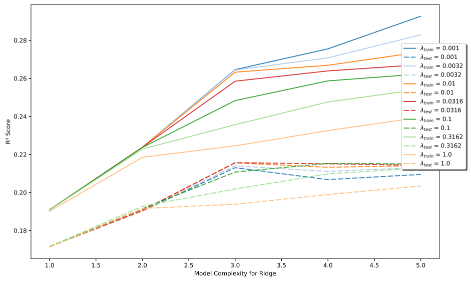

FIG. 24.— Figure of the R^2-score plotted against the complexity and for different values of λ. This is the R^2-score train mean, R^2-score test mean with resampling for Ridge for noise $\sigma^2 = 0.5$.

For figure 23 and 24 one can see the same plot as in 21 and 22 but for the noise with variance $\sigma^2 = 0.5$. One sees small improvements in how well the test set follows the train set, but again the noise term is to great. The real

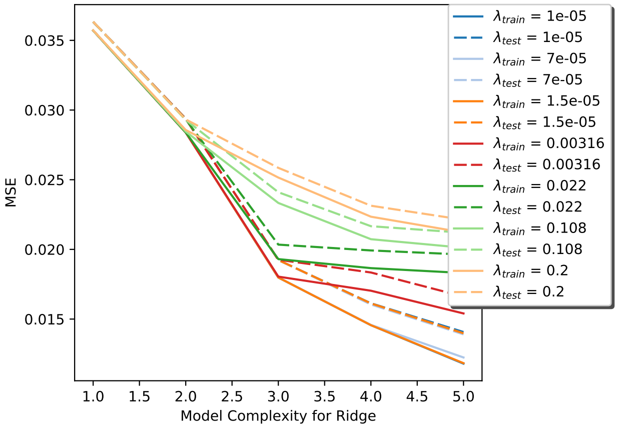

FIG. 25.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean from the home-made K-fold code with Ridge for noise $\sigma^2 = 0.1$. Number of $\lambda = 10$ then np.logspace was used for 10^{-2} to 10 with step 5.

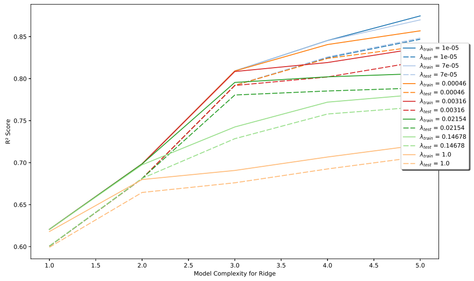

FIG. 26.— Figure of the R^2-score plotted against the complexity and for different values of λ. This is the R^2-score train mean, R^2-score test mean with resampling for Ridge for noise $\sigma^2 = 0.1$.

improvements are seen in figures 25 and 26 for $\sigma^2 = 0.1$. Without the variance of the noise dominating the train and test data behave more or less the same. That being said the MSE is rather low in all these figures for all the different noises, but the R^2-score is low, that being said it is even worse for OLS for higher noises, so even if the model is doing poorly for high noises it still looks like it is better than Linear regression.

5.5. Results for Lasso Regression

This subsection is devoted to the results produced by the use of Lasso regression. Since Lasso regression also include an extra parameter λ the setup of the figures are similar to how Ridge regression is setup.

Figures 5.5, 5.5 and 5.5 show the β coefficients for different polynomials and how they behave by using separate values of λ for variance of the noise $\sigma^2 = 1, 0.5, 0.1$. For high values $\lambda = 0.3$ illustrates well how the parameter works in the case of Lasso regression. It simply shrinks all the β coefficients to zero quite quickly. Similar to Ridge this is not what we wish to happen so these figures illustrate that a $\lambda = 0.1, 0.20.3$ is undesirable in this case. For $\lambda = 0.001$ and $\lambda = 0.01$ there are better

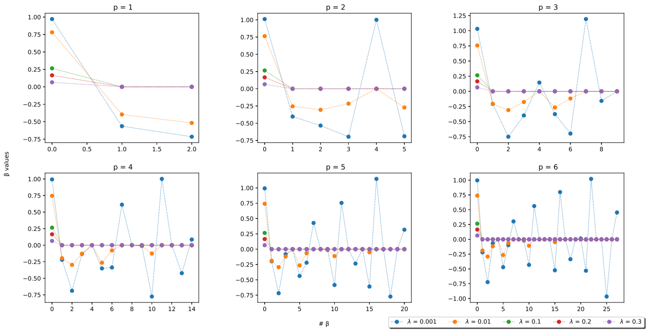

FIG. 27.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of $\lambda = 0.001, 0.01, 0.1, 0.2, 0.3$ for the Lasso regression with noise $\sigma^2 = 1$.

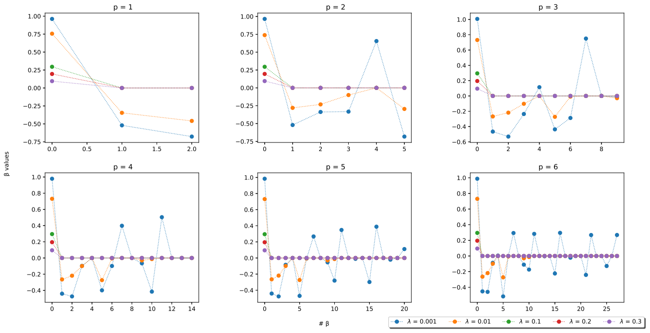

FIG. 28.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of $\lambda = 0.001, 0.01, 0.1, 0.2, 0.3$ for the Lasso regression with noise $\sigma^2 = 0.5$.

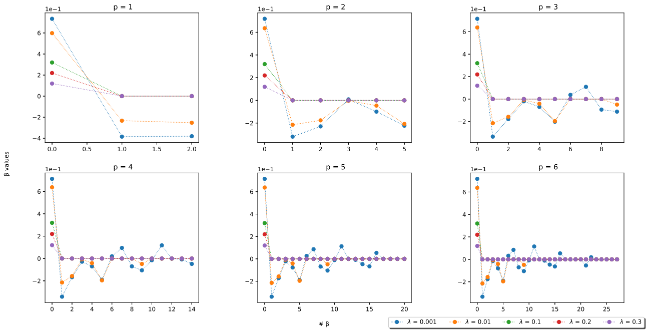

FIG. 29.— Figure of β plotted for different degrees of polynomials $p = 1, 2, 3, 4, 5, 6$ for different values of $\lambda = 0.001, 0.01, 0.1, 0.2, 0.3$ for the Lasso regression with noise $\sigma^2 = 0.1$.

results, giving small β values with low fluctuations for higher polynomials. The noise term again plays an important part, for low noise it looks like the Lasso method eventually shrinks all the β coefficients to zero, but as will be shown later it only means that for lower noise terms the λ-value needs to be lower than shown in these figures. For higher noise $\sigma^2 = 0.5$ for example we see that $\lambda = 0.001$ fluctuates nicely for higher polynomials $p = 4$ or $p = 5$.

5.6. Results for Lasso regression plotted for MSE and the R^2-score

For this section we look at how the MSE and R^2-score behave with the Lasso regression and what they can give us of information about Lasso regression as a model for the case of the Franke function.

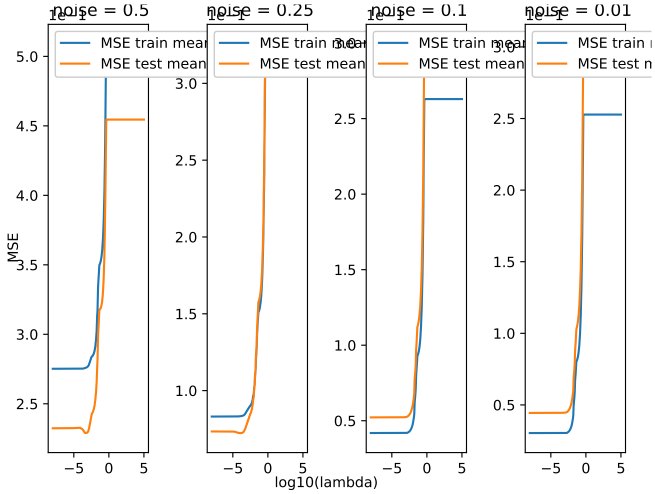

FIG. 30.— Figure of the MSE regression plotted against different $\log(\lambda)$ for polynomial $= 2$. This is the MSE train mean, MSE test mean without resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

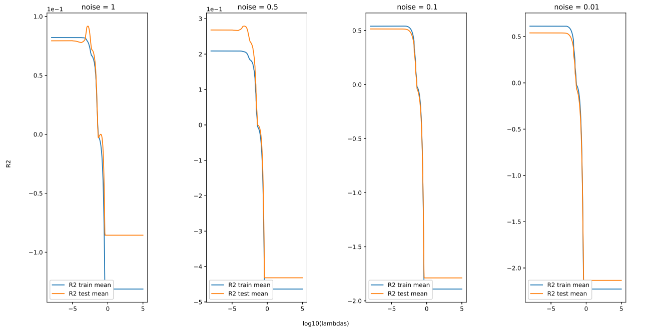

FIG. 31.— Figure of the R^2-score regression plotted against different $\log(\lambda)$ for polynomial $= 2$. This is the R^2-score train mean, R^2-score test mean without resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

Similar to the Ridge regression figures 31 and 5.6 shows the MSE and R^2-score plotted for different values of λ for $p = 2$ without resampling. The results are presented in table 4 and 5, which show the λ_{min} for the MSE and λ_{max} for the R^2-score. Again we see that the train and the test data are shifted compared to with resampling 32 and 35.

Like for the OLS figures above, which show MSE and R^2-score plotted against model complexity, figures 32, 35, 34 and 5.6 show how the model overfits and underfits and where our optimal values are, but here it shows our optimal values for the λ parameter. For low λ one can see that the MSE is low and this indicates that the bias and variance term are both low, this will be talked about in greater length in the subsection for the Bias and Variance trade-off 2.7. Then as λ rises so does the MSE, this means that the model starts overfitting to the noise term, before this happens the MSE has a little dip which is where we get the λ_{min}. We can not see this as clearly for lower noises, but the λ can become very small in these cases and as seen from 4 have values $\lambda = 1e - 08$ for noise variance $\sigma = 0.1$. For low noises it can seem like the OLS is the best model to use, this makes sense since the function for lower noises get closer to its original form

TABLE 4
THE VALUES OF λ_{min} FOR MSE WITH LASSO REGRESSION WITH AND WITHOUT RESAMPLING

σ^2	λ_{min} for MSE	$\log(\lambda)$ for MSE
Lasso without resampling for $p = 2$		
1	0.00144	-2.842
0.5	0.000551	-3.259
0.1	1e-08	-8.0
0.01	1e-08	-8.0
Lasso with resampling $p = 2$		
1	0.0018	-2.74
0.5	0.00052	-3.29
0.1	0.0011	-2.95
0.01	0.0014	-2.87
Lasso with resampling $p = 4$		
1	0.0006	-3.21
0.5	0.0002	-3.78
0.1	1e-08	-8.0
0.01	0.0001	-3.86

TABLE 5
THE VALUES OF λ_{max} FOR R^2-SCORE WITH LASSO REGRESSION WITH AND WITHOUT RESAMPLING

σ^2	λ_{min} for R^2	$\log(\lambda)$ for R^2
Lasso without resampling for $p = 2$		
1	0.0014	-2.84
0.5	0.00055	-3.26
0.1	1e-08	-8.0
0.01	1e-08	-8.0
Lasso with resampling $p = 2$		
1	0.0018	-2.74
0.5	0.00062	-3.21
0.1	0.0013	-2.89
0.01	0.0014	-2.84
Lasso with resampling $p = 4$		
1	0.002	-0.523
0.5	6 e-05	-1.200
0.1	1e-08	-8.0
0.01	0.00014	-3.86

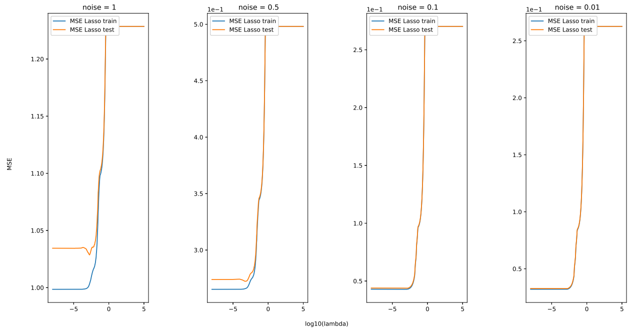

FIG. 32.— Figure of the MSE regression plotted against different $\log(\lambda)$ for polynomial $= 2$. This is the MSE train mean, MSE test mean with resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$. For $\sigma^2 = 1$ the minimum $\log(\lambda) = -2.74$ and for $\sigma^2 = 0.5$ the minimum $\log(\lambda) = -3.284$

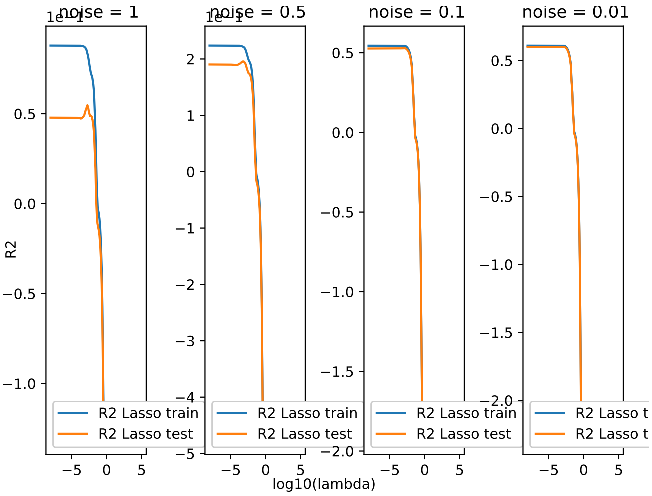

FIG. 33.— Figure of the R^2-score regression plotted against different $\log(\lambda)$ for polynomial = 2. This is the R^2-score train mean, R^2-score test mean with resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

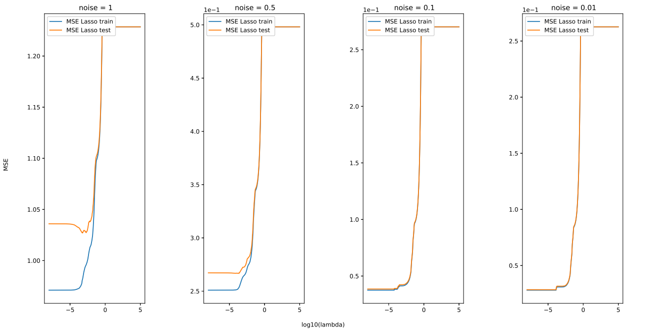

FIG. 34.— Figure of the MSE regression plotted against different $\log(\lambda)$ for polynomial = 4. This is the MSE train mean, MSE test mean with resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

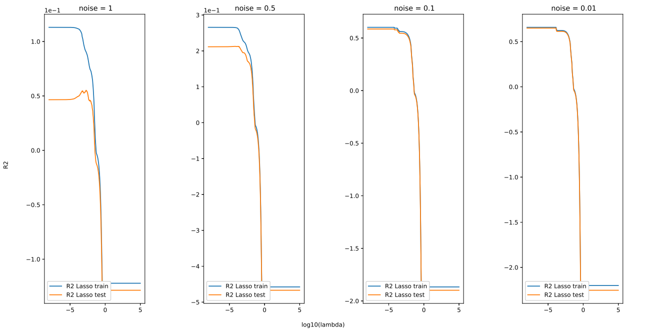

FIG. 35.— Figure of the R^2-score regression plotted against different $\log(\lambda)$ for polynomial = 4. This is the R^2-score train mean, R^2-score test mean with resampling for Lasso with noise $\sigma^2 = 1, 0.5, 0.1, 0.01$.

and then Ridge and Lasso may be excess in these cases. Here it can be mentioned that Ridge for lower noise terms behaves very similar to OLS for low λ. For higher noise terms Lasso and Ridge are better if one finds the right λ-value for the right polynomial and these values are shown in their respective tables. The R^2-score tells us more about which model to choose for this particular problem, it shows low values far from 1 but seems to perform better for higher noise terms than the OLS. To be sure we have to look at set λ values for model complexity.

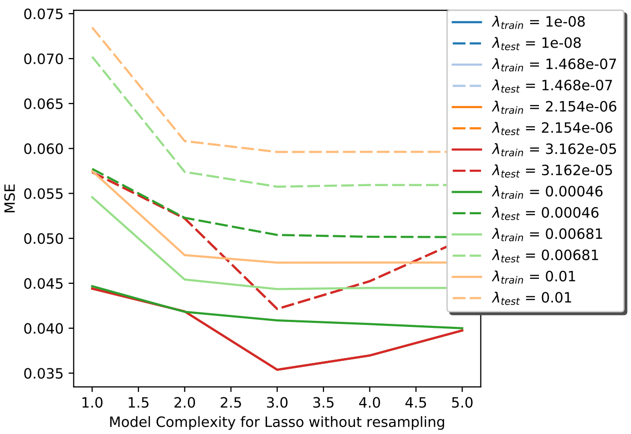

FIG. 36.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean without resampling for Lasso with noise $\sigma^2 = 0.1$. Number of $\lambda = 1$ then np.logspace was used for 10^{-6} with step 5.

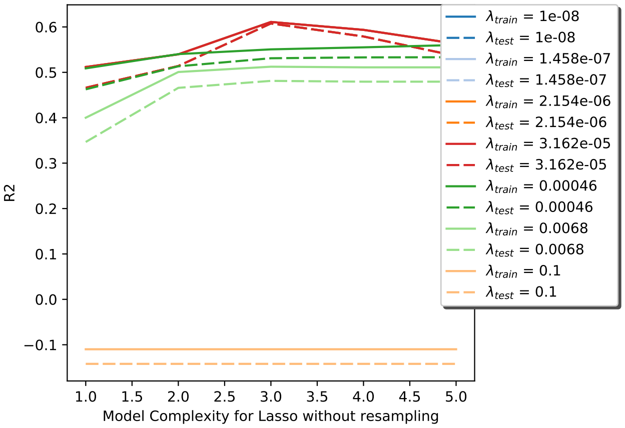

FIG. 37.— Figure of the R^2-score regression plotted against complexity and for different values $\log(\lambda)$. This is the R^2-score train mean, R^2-score test mean without resampling for Lasso with noise $\sigma^2 = 0.1$.

In figures 36 and 5.6 we see the MSE and the R^2-score plotted for the model complexity with set values of λ for noise variance $\sigma^2 = 0.1$. The model shows low MSE for some values of λ, particularly $\lambda = 1e-08$, but not lower than the OLS which has a MSE below 0.015. So without resampling and with low noise term the OLS performs better overall, this one also can see from looking at the R^2-score which gives around 0.5 for Lasso and 0.8 for the OLS for the same noise term.

With resampling we see how Lasso really works, in 38 and 5.6 we see the MSE and R^2-score plotted against model complexity for the noise variance of $\sigma^2 = 1$. Here it is clear what the Lasso method does, it stabilizes the MSE more or less as the complexity rises. For the observant eye one can also check that the λ-values do indeed coincide with the λ-values from 4 and 5 for $p = 2$ and

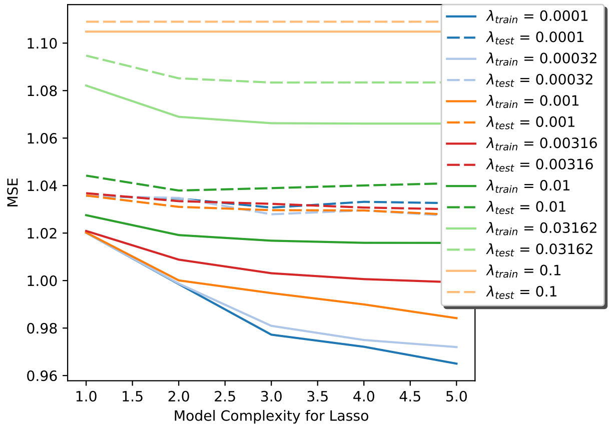

FIG. 38.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean from the home-made K-fold code with Lasso for noise $\sigma^2 = 1$. Number of $\lambda = 1$ then np.logspace was used for 10^{-6} to 1 with step 5.

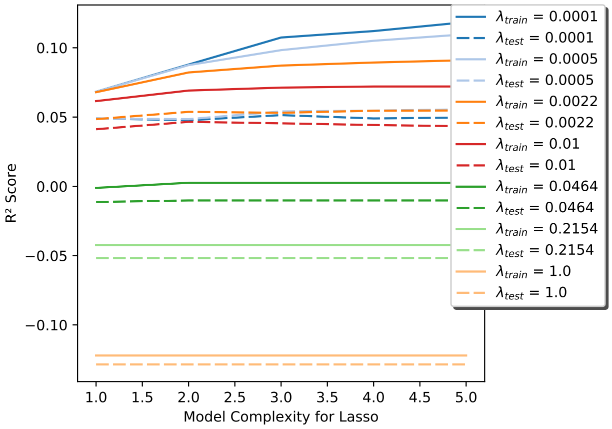

FIG. 39.— Figure of the R^2-score regression plotted against complexity and for different values $\log(\lambda)$. This is the R^2-score train mean, R^2-score test mean with resampling for Lasso with noise $\sigma^2 = 1$.

$p = 4$. This again shows that Lasso is a better choice when the noise is high ($\sigma^2 = 1$), whilst OLS starts out from MSE $= 1$ for $p = 1$ it rises quickly to much higher values, while for Lasso the MSE $= 1.04$ for $p = 1$ and then goes down from there as the complexity rises. The R^2-score is low, around 0.05 for the best performing λ-values and is also more or less stable, for OLS it drops down to negative values, so for high noise Lasso and Ridge are better choices for meaningful results. In figure 40 and 41 we see the same type of figure but with the noise variance being $\sigma^2 = 0.5$. The MSE is even lower for some of the λ-values, and the R^2-score is higher, indicating that also Lasso in this case is a better predicting model than OLS. For the last figures 42 and 43 the noise variance is $\sigma^2 = 0.1$. Here, as the complexity rises one can see from the figure and by looking at table 4 and 5 that the best λ-values are $1e-08$, and this gives the lowest MSE and highest R^2-score. For OLS on the other hand the MSE goes to zero for the same complexity, so for this noise

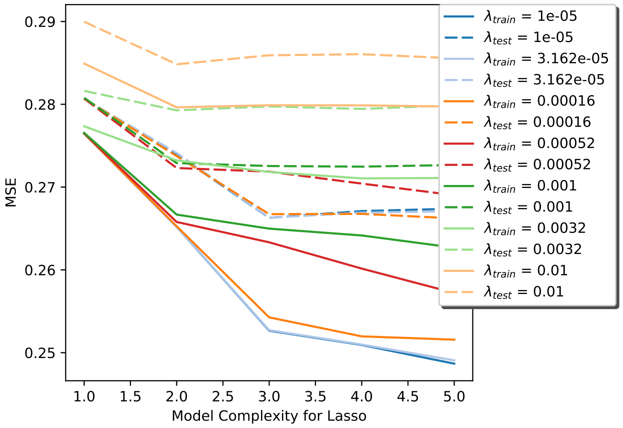

FIG. 40.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean from the home-made K-fold code with Lasso for noise $\sigma^2 = 0.5$. Number of $\lambda = 1$ then np.logspace was used for 10^{-6} to 1 with step 5.

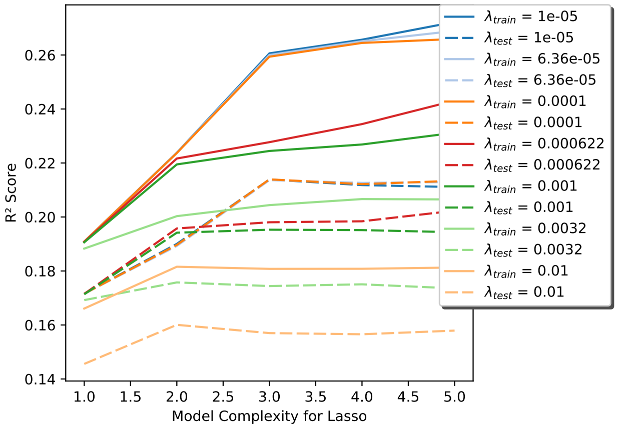

FIG. 41.— Figure of the R^2-score regression plotted against complexity and for different values $\log(\lambda)$. This is the R^2-score train mean, R^2-score test mean with resampling for Lasso with noise $\sigma^2 = 0.5$.

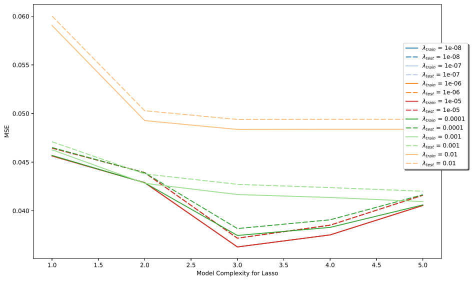

FIG. 42.— Figure of the MSE plotted against the complexity and for different values of λ. This is the MSE train mean, MSE test mean from the home-made K-fold code with Lasso for noise $\sigma^2 = 0.1$. Number of $\lambda = 1$ then np.logspace was used for 10^{-6} to 1 with step 5.

term the OLS is performing better. This is also the case for Ridge compared to the OLS. That said, the lowest

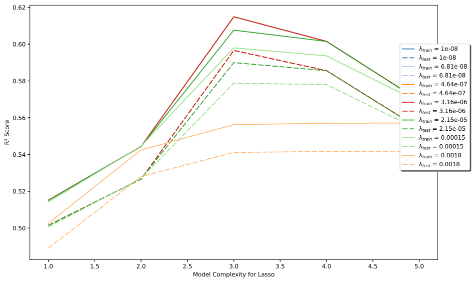

FIG. 43.— Figure of the R^2-score regression plotted against complexity and for different values $\log(\lambda)$. This is the R^2-score train mean, R^2-score test mean with resampling for Lasso with noise $\sigma^2 = 0.1$.

λ explored is $1e-08$ it can be that the best fit is made with a lower λ than found here, this one can read more about in the section for the results of the terrain data 6.

5.7. The Bias Variance Trade-off

In the end of our results for the Franke function and discussion of these we dive into the Bias Variance Trade-off. To obtain these results the resampling method Bootstrap was used.

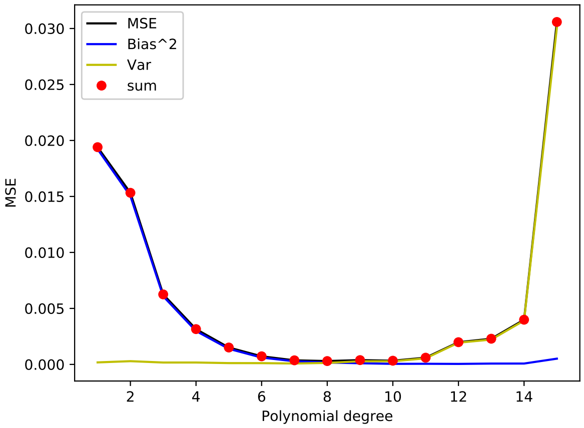

FIG. 44.— Figure showing the Bias Variance Trade-off obtained by using Bootstrap for OLS. The plot shows the Bias and the Variance plotted with the MSE with noise $\sigma^2 = 0.0001$.

In figure 44 one sees the Bias Variance Trade-off obtained with bootstrap whilst using noise variance $\sigma^2 = 0.0001$ for OLS. This low noise was used to get a good illustration of how the Bias Variance trade-off works according to theory. When using higher noise the variance term gets so great like in figure 46 that one can not really see how the bias decreases as the complexity rises. To supply on the figures the table 6 is also included with some of the values for the MSE, $Bias^2$ and the variance. All the values can be found in the file $ValuesforBiasVarMSE.txt$ in the Github repository. In the figures 44, 45 and 46 one sees how the model starts with underfitting the data, and here the $Bias^2$-term is the dominating one in the MSE, then as the complexity rises the variance becomes more and more dominating and the model overfits the data points. In the area where these two both are low and cross each other is called the optimum area as discussed before with other figures.

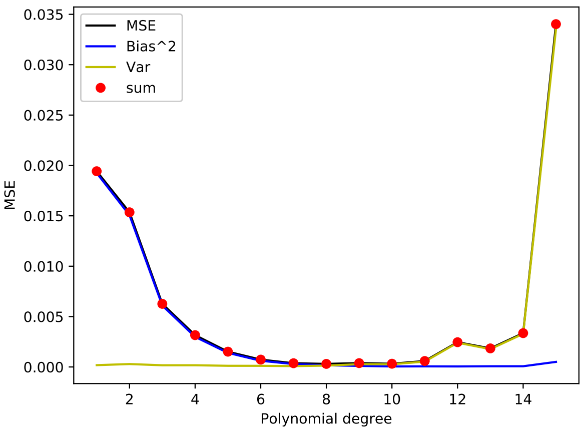

FIG. 45.— Figure showing the Bias Variance Trade-off obtained by using Bootstrap. The plot shows the Bias and the Variance plotted with the MSE with noise $\sigma^2 = 0.001$.

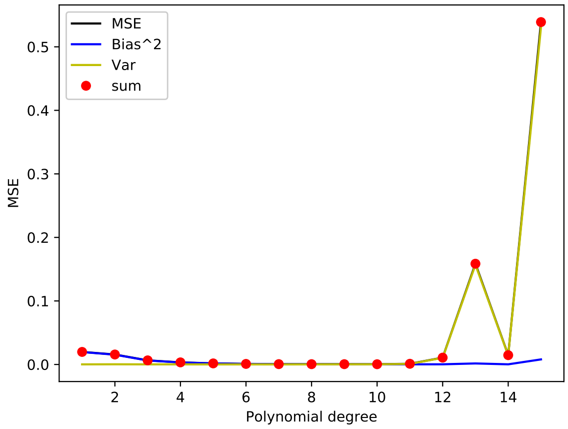

FIG. 46.— Figure showing the Bias Variance Trade-off obtained by using Bootstrap. The plot shows the Bias and the Variance plotted with the MSE with noise $\sigma^2 = 0.01$.

For Lasso one can see that the $Bias^2$ and Variance terms behave differently by looking at 6, here the variance is constant. This we believe is because of how Lasso works, using the shrinking parameter λ to stabilize the MSE as we have seen in previous subsections. Ridge also gives small values for the Variance whilst the $Bias^2$-term behaves like it should, decreasing with complexity. It might indicate that for Ridge the complexity needs to be even higher before the Variance term comes into play.

6. Results: Real data

The digital terrain data used for analysis was obtained from the *United States Geological Survey - Earth Explorer* website(9). The terrain chosen is of Oslo Fjorden, Norway.

TABLE 6
The Bias, Variance and MSE for Linear, Ridge and Lasso regression for different polynomials $p = 1, 2, 3....15$ obtained with the resampling method Bootstrap. For Ridge the parameter $\lambda = 1e-03$ and for Lasso $\lambda = 1e-04$ and noise for all $\sigma^2 = 0.001$ This is for the purpose of seeing how the Bias Variance Trade-off behaves for the three different methods so therefore it is picked out some of the polynomials with their values. All the values can be found in a txt file on Github.

p	Error (MSE)	$Bias^2$	Variance
	Linear Regression		
2	0.0154	0.0151	0.0003
5	0.0015	0.0014	0.0001
7	0.0004	0.0003	$8.364{\cdot}10^{-5}$
10	0.0003	$5.033{\cdot}10^{-5}$	0.0003
12	0.0025	$5.189{\cdot}10^{-5}$	0.0024
15	0.0018	0.0005	0.0335
	Ridge Regression		
2	0.0154	0.0151	0.0003
5	0.0029	0.0028	0.0001
7	0.0023	0.0022	$9.691{\cdot}10^{-5}$
10	0.0023	0.0018	$8.807{\cdot}10^{-5}$
12	0.0017	0.0016	$8.992{\cdot}10^{-5}$
15	0.0016	0.0015	0.0001
	Lasso Regression		
2	0.0153	0.0151	0.0002
5	0.0088	0.0086	0.0002
7	0.0091	0.0089	0.0002
10	0.0083	0.0081	0.0002
12	0.0079	0.0077	0.0002
15	0.0077	0.0075	0.0002

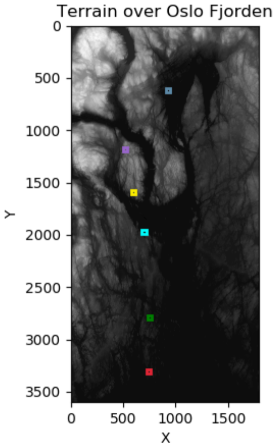

Fig. 47.— Figure shows the Oslo Fjord terrain data: The height is represented by the range from black to white. Where the darkest color contrast corresponds to zero height(water-level) and the lightest to the tallest point on the map($705m$). The map has a resolution of 1801 × 3601

With the terrain data we made use of the theory and methods applied during the analysis of the *Franke Function*. Unique for the terrain analysis is that we to a greater extent sought optimal parameters by algorithmic means. We did this by iterating over polynomial degrees, $p = [0, 30]$, and for each polynomial iteration we iterated over hyperparameters $\lambda = [1e-16, 1e+1]$ with 50 data points. The final λ and p for a given cut was chosen by which gave the best mean MSE on the training set using k-fold resampling method with five splits. For this high-polynomial-analysis to be computationally feasible we chose small cuts of data from the map(40×40), as can be seen by the colour boxes in 6. If we had chosen larger cuts of data we would have such a large signal-to-noise ration that every next polynomial degree would be better than the former up to such a high number that it would be too time-intensive. We wanted to look at a cut where it could be illustrated that higher polynomial degrees not necessarily led to a lower MSE. More concretely we wanted the training and test sets to be sufficiently small such that the phenomenon of overfitting would become apparent.

6.1. Real Data Discussion

OLS

We were generally pleased with the OLS result. As can be seen from 7 they have R2 scores close to 1. It is also evident from the plots 6.2 and 6.2. Cut 5 in 6.2 distinguishes itself markedly from the others in that the optimal regression was modeled by a $15th$ order polynomial function, where the other cuts are close to the maximum degree of 30. The probable cause is that cut 5 is dominated by water, which gives a height value $z = 0$. This is equivalent to the set having substantially less data and thus being more prone to overfit as the polynomial degrees increase. An obvious rebuttal to this hypothesis is that cut 1 in 6.2 also has a lot of water. While this is true, cut 5 does have more water, and cut 1 has a large height gradient in the areas above water level($z \geq 1$) compared to cut 5. Probably as a consequence of its flatness cut 5 had the lowest MSE of all the cuts. On the other side of the spectrum, cut 4 and cut 6 had the largest MSEs and also had the most hilly terrain. We see visually in 6.2 that they have larger changes in height, pixel to pixel, compared to the plots with lower MSE. We note that even though cut 6 has the largest MSE of 15.350 it also has the best R2 of 0.991. The R2 score compares the given regression with the mean, which in this case would just be a plane through the mean height. This result captures a difference in the MSE and R2 and how to use them. We suggest that many peaks and troughs in the terrain contribute to the MSE, thus making it high, but since they are not very uniform through the terrain a flat plane would approximate them badly in relation to our regression, giving us a high R2 score.

Ridge

Much of the analysis in OLS apply to the ridge results and plots 6.2,6.2. Before we obtained our results we had an initial fear of shrunken intercepts as a problem in on our own model, as we do not remove the intercepts before the inversion with hyperparameters. We made an attempt at removing the intercepts, but this gave us very non-favourable results[1] in the form of flat surfaces. The attempt was done by removing the intercept column from

[1] The code for this is along with everything else available on the Github link in the appendix 9

the design matrix before calculating the coefficients(β) and then reintroduce its corresponding coefficient as the mean of the current data set z. What this approach possibly lacked was a form of standardization, unfortunately we did not get to improve upon this. The final results, with the intercept along for the ride, gave us far more pleasing results. A way of giving the final Ridge model a verdict is to look at how it compares with the OLS. The polynomial degrees are close to the OLS on all the cuts(7 and the plots look visually similar, but the Ridge regression fares consistently better than OLS shown by both MSE and R2 on all the cuts. It's not a big difference, but it's consistent. A peculiar thing was that on doing iterations of λ over 17 orders of magnitude, 5 of 6 cuts obtained the same optimal *lambda*, which can be seen in 9. They do differ at smaller decimals as can be seen on the text files on Github 9. Still it seems to indicate that the models for different cuts have something in common. We don't have any good explanation for this other than it might be a trait for the general data of the map.

Lasso

As mentioned in sections 4, we had issues with Scikit-learn's functions for the Lasso regression. This is very apparent just by looking at the final figures of Lasso regression 6.2 and table 7, compared to the figures created with our homemade Ridge regression function6.2. We suspected that the faultiness of the regression had its roots in our interpretation of how Scikit-learn handles the intercept. As can be seen in the code for the real data we tried to remove the intercept column from the design matrix and let Scikit-learn calculate the remaining coefficients$\beta_n, n \geq 1$. The intercept β_0 was then calculated by calling on Scikit-learn's own intercept variable. We then merged β_n and β_0, and reintroduced the first column of the design matrix for error analysis and plot. Our approach was similar to the one we tried with Ridge regression, but letting Scikit do regression. To our surprise this method gave us slightly better results overall in terms of total MSE, with the intercept left out giving us: $mean(MSE)_{\text{no intercept}} = 172.201$, and with intercept included: $mean(MSE)_{\text{intercept}} = 203.640$. This gives a ratio of $\frac{mean(MSE)_{\text{no intercept}}}{mean(MSE)_{\text{intercept}}} = 0.845$. While the MSE was better in the without intercept approach, it did not make any noticeable improvement to the final regression plots. It was also not consistently better, with terrain cut 1,2,3 and 6 giving a better MSE while terrain cut 4 and 5 slightly worse. For the sake of consistency we chose to include the *with intercept* results in this article, as it was also done this way with OLS and Ridge regression[2].

6.2. Confidence interval

The text files with the data included in the Github repository9 includes all the values of the coefficients with the limits of the confidence interval. We used one standard deviation, which corresponds to 68% confidence. The formula we used for the standard deviation $\sigma(\beta)$ was the diagonal elements and their square roots of equation 9, using the sample variance 10. Our uncertainties on the results were for some(not all) coefficients $2-3$ orders of magnitude larger than the coefficient itself. We were unsure whether it was an issue or not, as our regressions do run up to a relatively high polynomial degree. Another possible explanation is that the sample variance formula is not suited to calculate the variance, as the terrain may have a completely different probability distribution than what the sample variance assumes. The last explanation we considered was that it might be a simple error in the code. The Lasso regression we made that did not include the intercept in the calculations had standard deviations completely on the other side of the spectrum, with around 6 orders of magnitude less than the coefficients, which suggest that one of the approaches leads to erroneous standard deviations.

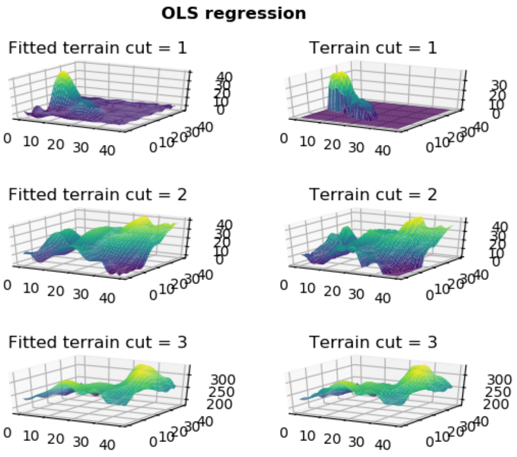

FIG. 48.— Figure shows OLS regression fit of cut 1,2,3 to the left, and terrain data of the same area to the right. Resolution is 40×40

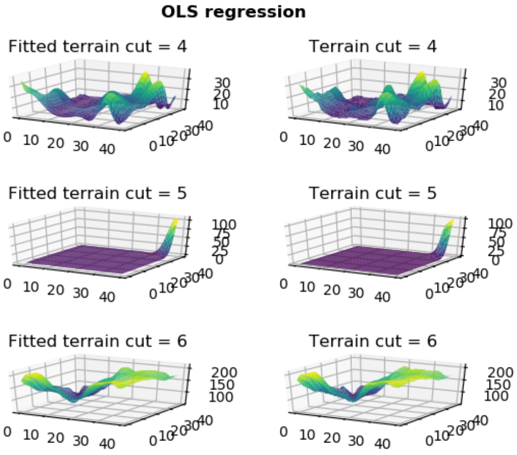

FIG. 49.— Figure shows OLS regression fit of cut 4,5,6 to the left, and terrain data of the same area to the right. Resolution is 40×40

[2] The data results and plots for both approaches are available on the Github link in the appendix 9. The code for both approaches are in the same code-file

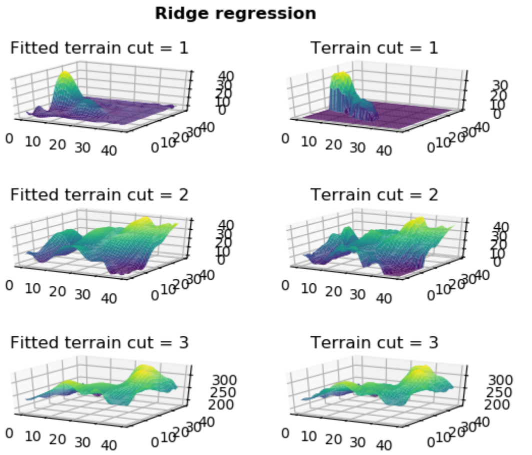

FIG. 50.— Figure shows Ridge regression fit of cut 1,2,3 to the left, and terrain data of the same area to the right. Resolution is 40 × 40

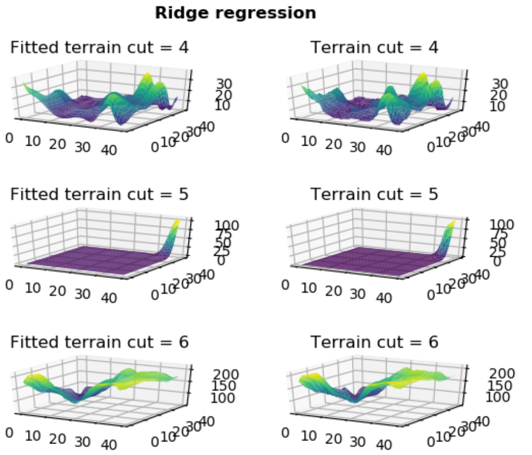

FIG. 51.— Figure shows Ridge regression fit of cut 4,5,6 to the left, and terrain data of the same area to the right. Resolution is 40 × 40

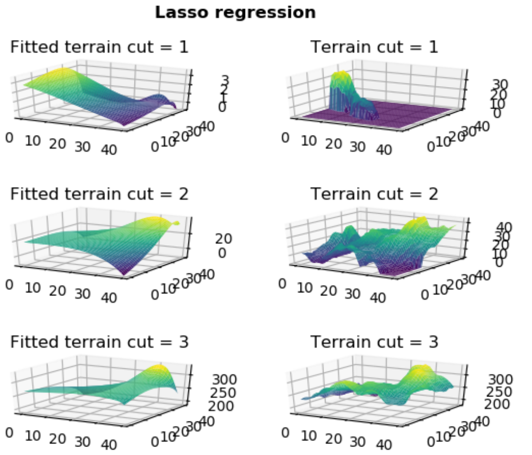

FIG. 52.— Figure shows Lasso regression fit of cut 1,2,3 to the left, and terrain data of the same area to the right. Resolution is 40 × 40

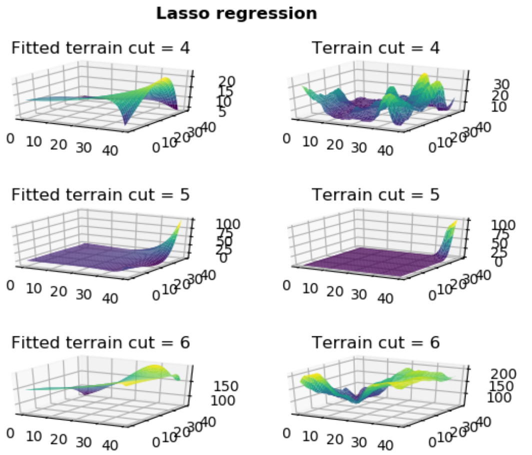

FIG. 53.— Figure shows Lasso regression fit of cut 4,5,6 to the left, and terrain data of the same area to the right. Resolution is 40 × 40

7. Perspective for Future Improvements

At the end of the day we found that the Scikit-Learn library was a little confusing, it was sometimes like sending variables into a black box and then getting out something you did not really know if was right or wrong. In our analysis of the Franke function the function used from Scikit-Learn most frequent was the Lasso regression function, which gave us the β coefficients. We did not see any problems by using this function before we used it with the terrain data, here it seemed to be problems with the fitting of the terrain. Though reading endlessly about this on their websites and other sources we where not able to solve this. For all we know this is to be expected from using Lasso on the terrain data. So for future improvements we would like to make our own home-made code for Lasso like we did with the k-fold and Ridge among others. This would have given us a deeper understanding of the machinery behind and we would be better equipped with knowledge whilst analysing our results for Lasso. Lastly, we would also like to improve upon our homemade Ridge function to include a functioning algorithm for doing regression with or without an intercept term to our pleasing.

8. Conclusion

Under the present study it was shown that the Linear regression approach can to some extent predict data from both known and unknown data. For the analysis of the Franke function OLS, Ridge and Lasso where thoroughly checked up to the theory behind these methods with and without resampling. Resampling was performed with K-fold Cross-Validation and Bootstrap and both proved to be high performing methods that helped us in our work to uncover the real MSE and R^2-scores of both our datasets. For the Franke function we discovered that the noise term, or in other words the approximation in our errors, played a big role and helped us see that Ridge and Lasso performs better than the OLS for high noises. For low noises the clear winner was the OLS. With the terrain data there was some doubts about whether our initial take on the Scikit-Learn function for Lasso was right or wrong. The Lasso method performed poorly on the terrain data and we where not able to find an answer for this problem. On the other hand Ridge and the OLS both behaved as expected and the functions for these two where made by the writers of this paper. This makes us suspect that we may have misunderstood how the Scikit-Learn function for Lasso actually works.

9. Appendix A: Link to the All Programs

Link to the project in Github

TABLE 7

Optimal choice of polynomial degree(p) and hyperparameter(λ) with respect to MSE test values. Total- MSE and R2 for optimal choice also listed. The test MSE, taken as the mean from k-folds of resampling, is thus the minimum value in the intervals $p = [0, 30]$ and $\lambda = [1e - 16, 1e + 1]$. All the values were rounded off at the third decimal digit, although at later decimals there are subtle differences. All the values can be found in a txt file on Github. Colours correspond to the cuts of the terrain figure

Terrain cut	Polynomial Degree	λ	Total MSE	Total R2	Test MSE
		OLS Regression			
1	28	0	4.978	0.910	5.521
2	27	0	3.701	0.965	4.471
3	30	0	10.090	0.989	13.677
4	28	0	1.613	0.969	1.970
5	15	0	0.868	0.984	1.264
6	27	0	15.350	0.991	18.603
		Ridge Regression			
1	29	7.906e-12	4.743	0.914	5.300
2	27	3.727e-12	3.605	0.966	4.215
3	30	3.727e-12	9.988	0.989	12.519
4	28	3.727e-12	1.524	0.970	1.791
5	16	3.727e-12	0.722	0.986	1.185
6	30	3.727e-12	14.049	0.991	17.271
		Lasso Regression			
1	30	5.689e-05	51.608	0.074	51.715
2	30	5.42e-04	41.586	0.610	42.041
3	30	2.44e-03	338.077	0.646	341.741
4	26	2.44e-03	36.489	0.304	36.862
5	30	5.17e-03	29.885	0.454	30.431
6	30	1.15e-03	724.197	0.580	727.753

REFERENCES

[1]H.J. Morten. Data Analysis and Machine Learning - lecture notes fall 2019,(2019).

[2]https://blog.goldenhelix.com/cross-validation-for-genomic-prediction-in-svs/- 2015,
(2015)

[3]van Wieringen Wessel N. Lecture notes on ridge regression, Version 0.30, July 22, 2019.

[4]Mehta et. al. A high-bias, low-variance introduction to Machine Learning for physicists, May 29, 2019

[5]Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer.

[6]Zavershynskyi, Maksym, 22 May, 2017
https://towardsdatascience.com/mse-and-bias-variance-decomposition-77449dd2ff55

[7]
Kirk Baker, Singular Value Decomposition Tutorial, March 2005,
https://davetang.org/file/Singular$_{V}alue_{D}ecomposition_{T}utorial.pdf$

[8]18.095 IAP - MIT Mathematics, lecture notes, 2016
http://math.mit.edu/classes/18.095/2016IAP/lec2/SVD$_{N}otes.pdf$

[9]SRTM terrain data https://earthexplorer.usgs.gov/