An Attempt of Applying Linear Regression on Parametrizing Topography Map Data

Martin Moen Carstensen, George Stanley Cowie, Jan Egil Ødegård

(Dated: October 5th, 2020)

The source code files can be found in our GitHub Repository

We've used several linear regression methods to attempt to find a best-fit polynomial to two different data sets. First a randomly drawn out data set using the Franke function, then SRTM topography map data from NASA. We've used the Ordinary Least Squares (OLS)-method, as well as the Ridge and Lasso shrinkage methods, to attempt to find an optimal fit. We've also made use of Bootstrap and Cross Validation resampling techniques to further optimize our results. The OLS scheme was particularly prone to overfitting, while both the Lasso and Ridge methods seems to keep overfitting at bay. In general, it seems like the the Lasso method yields slightly better results than Ridge, both in terms of stability and accuracy. For resampling techniques, we're led to believe that Cross-Validation would yield better or more stable results than the bootstrap counterpart. In the end we also attempt at applying the linear regression to recreate the maps we are using. This is however attempted without any success.

I. INTRODUCTION

Linear regression is a widely used method in data analysis when attempting to connect data which has been provided in a number of different ways. This data might stem from everything from a science laboratory where the data could correspond to material properties, or from a psychology study where one is attempting to link sleep to mental well-being. Trying to figure out in what way the data correlate to each other, and to what degree this correlation holds, is an important aspect to all statistical analysis in science.

In this report, we will look closer at three different linear regression models, namely the ordinary least squares (OLS), the ridge shrinkage method, and the lasso shrinkage method. We will look closer at accuracy scores of these methods, and in addition, we will implement some resampling techniques in our data analysis to further improve and stabilize our results. We will also discuss the importance of bias and variance in the total error. We will first be using an analytic, weighted exponential function with added noise to do the statistical analysis. This will then be followed by applying these methods to a topography mapping data, in an attempt at applying our newfound knowledge to real data.

II. THEORY

The theoretical groundwork will mainly revolve around the different regression methods we will use to approximate the data, as well as the methods and parameters we will use to optimize and determine the eligibility of the different methods respectively.

A. Regression Methods

We will delve deeply into three different regression methods. First, we will be looking at ordinary least squares regression, followed by ridge regression, and finishing with lasso regression. To start things off, we need to define the concept of a design matrix.

1. *Ordinary Least Squares (OLS)*

Ordinary least squares, abbreviated OLS, is a regression method revolving around the minimization of the mean squared error (MSE) of the problem at hand. Starting off, we are approximating the data set $\mathcal{L}$ that we want to perform our linear regression on to be a polynomial of degree p. The data in the data set are given as $\mathcal{D}_\mathcal{L} = \left\{(y_i, \mathbf{x}_i)\ ,\ i \in \{0, 1, \cdots, n-1\}\right\}$, with y_i being the value corresponding to a set of parameter values $\mathbf{x}_i$. Doing this, we can define the vector $\beta \in \mathbb{R}^p$ to be the coefficients of the different degrees of the polynomial:

$$\beta = [\beta_0\ ,\ \beta_1\ ,\ \cdots\ ,\ \beta_p]^T. \tag{1}$$

Defining the data set values contained in $\mathcal{L}$ as $\mathbf{y}$ and the corresponding error term as ϵ in vector form, we have:

$$\mathbf{y} = [y_0\ ,\ y_1\ ,\ \cdots\ ,\ y_{n-1}]^T \tag{2}$$

$$\epsilon = [\epsilon_0\ ,\ \epsilon_1\ ,\ \cdots\ ,\ \epsilon_{n-1}]^T \tag{3}$$

where $\mathbf{y}, \epsilon \in \mathbb{R}^n$, where n are the number of data points contained in the data set $\mathcal{L}$. Finally, we introduce the design matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ to be the matrix of

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & \cdots & x_0^p \\ 1 & x_1 & x_1^2 & \cdots & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & \cdots & x_{n-1}^p \end{bmatrix} \tag{4}$$

We can then write our polynomial approximation of the points $\mathbf{y}$ using the following matrix-vector equation:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon \tag{5}$$

Considering that the error ϵ is unknown in both magnitude and shape, we approximate the data set values to be on the form:

$$\tilde{\mathbf{y}} = \mathbf{X}\beta \tag{6}$$

We can now define the cost function C to be the mean squared error in the following way:

$$C(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. = \frac{1}{n}\Big[(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})\Big] \tag{7}$$

Here, $\mathbf{y}$ are the actual data points and $\tilde{\mathbf{y}}$ are our approximated values to those data points. Implementing equation 6 for $\tilde{\mathbf{y}}$ in equation 7, we get our cost function as a function of the parameter β:

$$C(\mathbf{y}, \beta) = \frac{1}{n}\Big[(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)\Big] \tag{8}$$

To get the optimal parameters β, we wish to minimize the cost function. To do this, we take the derivative of the cost function with regards to the j-th β-parameter, and set this value to 0: (M. Hjorth-Jensen, slide 16 [1])

$$0 = \frac{\partial C(\beta)}{\partial \beta_j} \implies 0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) \tag{9}$$

Solving equation 9 for β will thus give us the optimal β-parameter for the OLS-scheme (we denote the optimal with a hat $\hat{\beta}$:

$$\hat{\beta}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{10}$$

As such, given a vector $\mathbf{x'}$ containing data we want to fit, our optimally predicted values using the OLS-scheme will be given as:

$$\tilde{\mathbf{y}}_{OLS} = \mathbf{x'} \cdot \hat{\beta}_{OLS}. \tag{11}$$

2. *Ridge Regression*

In ridge regression, we add another term to the mean squared error analysis with a free regularization parameter λ, such that we are looking to optimize the following cost function:

$$C(\beta) = \frac{1}{n}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_2^2. \tag{12}$$

Here $||\mathbf{x}||_2$ is the norm-2 of the vector $\mathbf{x}$, and we define λ to be strictly positive ($\lambda \geq 0$). By doing the derivative of the cost function in terms of the parameters β_j in the same way as we did for the OLS-method, and putting this to 0, we get:

$$\frac{\partial C(\beta)}{\partial \beta} = 0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) - \lambda\beta \tag{13}$$

Solving this for our parameter β, we get the optimal for the ridge method as being:

$$\hat{\beta}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbb{1})^{-1}\mathbf{X}^T\mathbf{y} \tag{14}$$

Where $\mathbb{1}$ is the identity matrix. As is apparent in equation 14, we still have a free parameter λ which we are free to adjust. There is typically a value for λ in which the mean squared error is minimized. We will therefore, in addition to doing a linear regression, also optimize the MSE for the parameter λ. To do this, we split the data set we are analysing into a validation set and a training set (more on splitting of data sets in section II B 1). We do the ridge regression on the training set for a given λ, and calculate the MSE of this model using the validation set. We repeat this for a series of values λ, and the value which returns the best accuracy score (MSE), we use as ideal the ideal parameter λ for that given complexity. When this value for λ has been determined, and given a vector $\mathbf{x'}$ containing data we want to fit, our optimally predicted values using ridge regression will be given as:

$$\tilde{\mathbf{y}}_{Ridge} = \mathbf{x'} \cdot \hat{\beta}_{Ridge} \tag{15}$$

This method is certainly more brute force and computationally heavy compared to the OLS method, but it also tends to yield slightly more accurate results. One must therefore do an assessment on whether or not one would want a slightly more accurate result for a relatively large amount of extra computational effort.

3. *Lasso Regression*

The final regression method we will be using is the "Least Absolute Shrinkage and Selection Operator", abbreviated Lasso. In Lasso regression, we replace the term $\lambda||\beta||_2^2$ which was used for ridge regression in equation 12, with a term $\lambda||\beta||_1$. As such, our cost function for Lasso regression reads as:

$$C(\beta) = \frac{1}{n}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_1. \tag{16}$$

Here $||\mathbf{x}||_1$ is the norm-1 of the vector $\mathbf{x}$. This subtle difference from the case in Ridge makes it so that there are no closed-form solutions for β that can be estimated of the cost-function (Hastie et Al. p.219-223 [2]). For this reason, we will be using standardized functionalities in SKLearn to find the β_j's which will optimize this given cost function. This method will numerically find the gradient of the cost function stated in equation 16, and predict the model β's based on this gradient (F. Pedregosa et Al. [3]).

B. Resampling & Optimization Techniques

When doing linear regression on a given data set $\mathcal{L}$, the given data points we have for regression is potentially low compared to the complexity of an optimal fit. When this is the case, there are certain resampling and optimization techniques one can use to make use of what we have got. Here specifically, we will be looking into the bootstrap and cross-validation methods as resampling techniques, as well as the scaling of the data for further optimization. We will first discuss the splitting of the data into a training set and a testing set.

1. *Training, Test & Validation Data*

When doing regression analysis on a data set, we typically split our data set into a test set and a training set. Typically, we take everything between 20%-35% and set this aside purely for testing, while we apply the linear regression analysis on the remaining 65%-80% of the data. The test data is then used to measure the accuracy of our model. Sometimes, we also split the training set further into a training set and a validation set. This is for example used in ridge regression, as discussed in section II A 2, where we have a hyperparameter λ which we have to optimize for before applying our result on the test data.

The primary reason as to why we split the data we do regression analysis on away from the data we use to determine the validity of our approximations is to determine whether our model is overfitted, underfitted, or just right. Looking at figure 1, we see an illustration of how the test data error will at some point start to grow. When this occurs, we have hit a point where the data is being overfitted. When overfitting becomes an issue, we typically start to model the error of the training data more than the actual data it is representing, and thus the error grows. See figure 2 for a illustration.

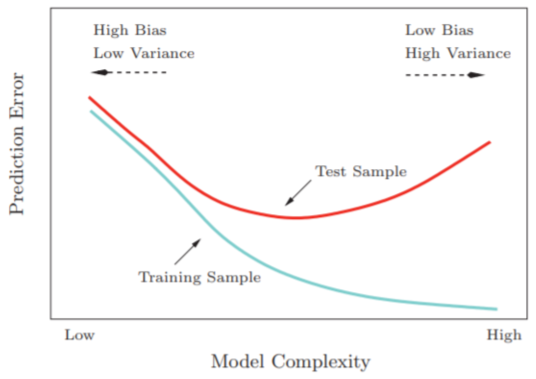


Figure 1. Figure showing how the error of the training and test data evolves as a function of complexity of the model. The training data typically gets a continuously better fit, while the test data error will at some point start to grow.[a]

[a] Source: Figure 2.11 of Hastie et Al. [2]

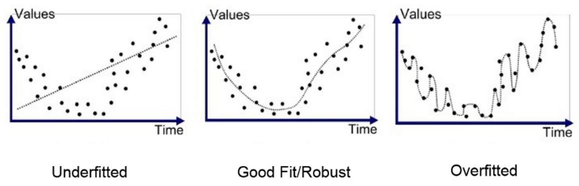


Figure 2. An illustration of the cases for underfitting, overfitting, and just perfectly fitting a function could look like.[a]

[a] Source: Anup Bhande, Medium.com [4]

2. *Bootstrap Resampling*

Given a (typically small) data set $\mathcal{L}$ where values $\mathbf{y} = [y_0, \cdots, y_{n-1}]$ is contained, we could choose to reuse some of the data in an arbitrary way in an attempt to optimize the results. The bootstrap method revolves around the idea of creating K new data sets $\mathbf{y}_i^*$, $i \in \{0, 1, \cdots, K-1\}$ from the already existing vector $\mathbf{y}$. These new vectors $\mathbf{y}_i^*$ are all of the same size as $\mathbf{y}$ ($\mathbf{y}_i^* \in \mathbb{R}^n$), and consist of a randomly selected values from $\mathbf{y}$. These values can be drawn repeatedly, and some elements might not be drawn at all, such that certain vectors $\mathbf{y}_i^*$ likely would contain several copies of the same element from $\mathbf{y}$. This is illustrated in figure 3. We then perform the linear regression analysis on our new vectors $\mathbf{y}_i^*$ individually, and determine the accuracy score (MSE, R2, etc..) for each of these bootstraps separately. The total accuracy score is then simply given by the mean value of each of these individual accuracy scores, which then is taken to be the accuracy of the entire model.

The reason bootstrap resampling is to be preferred

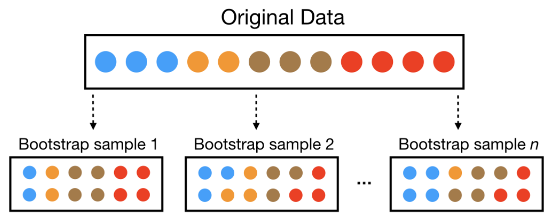


Figure 3. Illustration of bootstrap resampling of a given data set. Here, n is used as the amount of bootstraps in place of K. The original data represent our $\mathbf{y}$, while the bootstrap samples represent our $\mathbf{y}_i^*$.[a]

[a] Source: Figure 2.6 of Boehmke & Bradwell [5]

over not resampling at all, is due to the stability of the results that it generates, as well as an overall better model. The latter is especially true for smaller data sets, where resampling could help better the accuracy score. The stability of the accuracy score metric will also significantly improve, given as a mean value of many accuracy scores will tend to be close to the "true value" of said accuracy score for the model compared to any individual value in said mean.

3. *Cross-validation Resampling*

Given the same (typically small) data set $\mathcal{L}$, another way to resample the data set values is by something called cross-validation. Specifically, we will be dealing with what is known as "K-fold cross-validation". The general idea is that you take your training data and split it into K different sub data sets, which we nickname "folds". We then take the first of these folds, and turn this into a validation data set. We perform the linear regression method of choice on the the remaining $K - 1$ folds, and use the validation data to determine the accuracy of the model using parameters of choice (e.g. MSE, R2-score, etc..). We then take the second fold, turn this into a validation set, performs regression on the remaining $K - 1$ folds, and evaluate the accuracy of the validation set using the same parameters. This is repeated K times, and we then take the average (mean value) of all of the accuracy parameters to determine the overall accuracy of the model. This specific method in which one fold is left out for validation is known as the "leave-one-out cross-validation". (Abbreviated LOOCV) An illustration of this process can be seen in figure 4.

The reason this resampling technique may prove useful is in many ways the same as the ones for the bootstrap method. The stability increases, and the accuracy also is slightly improved.

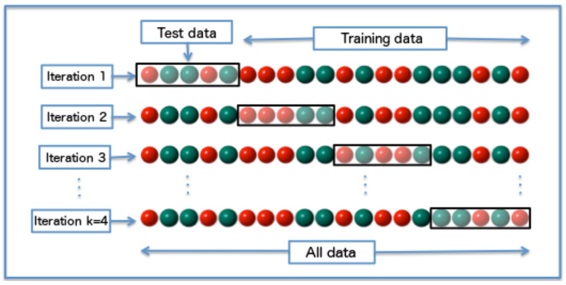


Figure 4. Illustration of an exhaustive Leave-one-out cross-validation (LOOCV) for $K = 4$ folds.[a]

[a] Source: Fabian Flöck / CC BY-SA [6]

4. *Scaling data*

In data sets, we could have large outliers in the parameters, and these outliers could then again potentially distort the results significantly. The way we work around this is by scaling the data in such a way that the significance brought by the outliers will be minimized.

For this project, we will specifically only be scaling in terms of the subtraction of the mean values of the columns of the design matrix. Doing this, any outliers that are significantly deviated from the mean that could distort will have their contributions minimized, while the more relevant datapoints will get a corresponding significance boost.

C. Optimization parameters

When doing approximations to a data set, one would ideally want to define parameters one could maximize or minimize to indicate how well our functional approximations match the data set. One of the most important of these is the mean squared error, which we will look closer into. Among other noteworthy parameters, one also finds the "R2-score", the mean absolute error, etc.. which will briefly be mentioned.

1. Mean Squared Error (MSE)

The Mean Squared Error, often abbreviated as MSE, is defined as:

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

where $\tilde{\mathbf{y}}$ denotes the set of approximated values and $\mathbf{y}$ denotes the set of analytically correct values. Using shorthand notation, equation 17 is usually written as:

$$\text{MSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{y} - \tilde{\mathbf{y}})^2\big] \tag{18}$$

The MSE can be decomposed and written in terms of three terms (See Appendix A for derivation):

$$\text{MSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2. \tag{19}$$

$$\text{MSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \Big(\text{Bias}\big[\tilde{\mathbf{y}}\big]\Big)^2 + \text{Var}\big[\tilde{\mathbf{y}}\big] + \sigma^2. \tag{20}$$

The first term in this expression is known as the bias, or rather the square of the bias, while the second term of this expression is the variance of our modelled values. The final term is the noise term, which is typically an unknown quantity.

We can interpret the bias term as being the error stemming from trying to overly simplify a complex problem. The contribution from the bias term is typically large for small values of complexity, while it shrinks as the complexity of any given model increases. The variance on the other hand is interpreted as the error stemming from overly fitting the data. If we overly fit the data, we will have a tendency to fit the noise more than the actual data itself. As such, we will expect low variance for a low-complexity model, while it tends to grow the higher the complexity becomes.

2. Other Parameters

In addition the the mean squared error, there are other parameters we will use to determine the validity of our models:

The **R2-score** is defined as:

$$R^2(\mathbf{y}, \tilde{\mathbf{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}, \tag{21}$$

where $\mathbf{y}$ are the actual data set values, $\tilde{\mathbf{y}}$ are the approximated data set values using our regression methods, and $\bar{y}$ is the mean value of the data set values $\mathbf{y}$. Given the definition of the R2-score, it can attain any value lower than 1, and it can also attain negative values. An R2-score of 1 represents a perfect fit, and the closer the acquired R2-score is to 1, the better.

The **Bias** has already been briefly discussed, and is defined as:

$$\text{Bias}[\tilde{\mathbf{y}}] = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2}. \tag{22}$$

Here, f_i is the true function value at a given point and $\tilde{\mathbf{y}}$ is a vector of the modelled values at corresponding points. We typically don't know the true value f_i for a given point, so we replace this with the data points y_i when this is the case.

The **Variance** has also been briefly discussed already, and is defined as:

$$\text{Var}[\tilde{\mathbf{y}}] = \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2. \tag{23}$$

Here, $\tilde{\mathbf{y}}$ is the vector containing all of the modelled values using regression methods.

Confidence Intervals We do not know the true variance of β, we will therefore use the t distribution to approximate the normal distribution. The upper and lower intervals are calculated as follows

$$l = \mu - t_{[(1-\alpha/2)\ ,\ n-1]}\sigma \tag{24}$$

$$u = \mu + t_{[(1-\alpha/2)\ ,\ n-1]}\sigma \tag{25}$$

Where σ is the standard deviation approximated by $\sqrt{\sigma^2}$. σ^2 is then again approximated by (Hastie et Al. p.47 [2]):

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{26}$$

The variance of the β parameters are estimated by

$$Var(\hat{\beta}) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2. \tag{27}$$

III. ALGORITHM

We will now be looking closer at how we specifically will be implementing the already mentioned linear regression and resampling methods. First we will however introduce and briefly discuss the data sets we will be using to perform the linear regression analysis on.

A. Data Sets to be Used

We will be using two different data sets when working with the linear regression. One with an analytic solution, and one that does not have an analytic solution.

1. *Approximations to Non-Polynomial Functions*

To affirm that the methods we will be using when analyzing map data, we first want to analyze a function that we can express analytically. For this, we will be using the Franke function $f(x, y)$, which is defined as the following weighted sum of exponentials:

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left\{-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right\}\\ &+\frac{3}{4}\exp\left\{-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}\right\}\\ &+\frac{1}{2}\exp\left\{-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right\}\\ &-\frac{1}{5}\exp\left\{-(9x-4)^2-(9y-7)^2\right\}. \end{aligned} \tag{28}$$

This equation is defined for $x, y \in [0, 1]$, and takes the form as seen in figure 5.

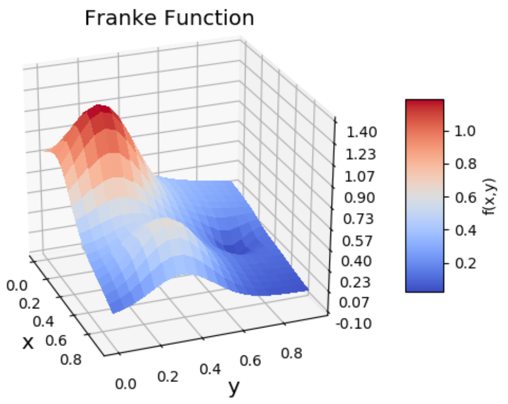


Figure 5. The Franke function visualized in a 3-dimensional plot. Somewhat resembling a mountainous topological map.

We will be adding a normally distributed noise term ϵ to this expression which runs as $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with σ being the standard deviation of said normal distribution. The reason for use of the Franke-function is to simulate a data set which we can't approximate perfectly with a finite-degree polynomial, yet we have an analytic solution of which we can use to compare the results acquired. We also add the noise term in to model any random artifacts which might interfere with the true value of a measurement for example.

We will be drawing a set of N random points (x_i, y_i) , $i \in \{0, 1, \cdots, N-1\}$ and evaluate the corresponding value (with added noise) on each of these given points z_i. We will then perform the linear regression method of choice on roughly 80% of this data set, and evaluate the accuracy on the remaining 20% using one (or several) of the optimization parameters as discussed in section II C.

2. *Approximation to Real Data*

For the 'real' data set, we will be working on topographical SRTM data, as provided by NASA[7]. The data set $\mathcal{L}$ which is provided gives us a 2-dimensional array of parameter values (x_i, y_j), and a function value corresponding to that set of function values $z(x_i, y_j)$. The parameter values correspond to a latitude and a longitude on the Earth, while the function values correspond to the height above the ocean at said position.

We will be drawing N random points on the map (x_i, y_i) , $i \in \{0, 1, \cdots, N-1\}$ and pick out the data points corresponding to these coordinates z_i. We will perform the linear regression method of choice then on this entire data set. We will then draw a completely separate data set of M points out (x_j, y_j) , $j \in \{0, 1, \cdots, M-1\}$ and evaluate the accuracy of our model on these data points using the same optimization parameters as with the Franke function. Finally, we will also do a qualitative check on the map by attempting to recreate the map from the data points using linear regression methods.

We assume that the surface topography of the Earth to be flat.[8] Doing this, and also assuming no topographic discontinuity on the surface, it would be analytic and possible to model with a sufficiently complicated polynomial. As such, linear regression on topography maps is a meaningful act of data analysis.

In figure 6, we see the specific map we will be using to create a model on. This is a topographical map of the Oslo fjord and the surrounding areas, which includes cities like Drammen, Moss, Fredrikstad, Sandefjord, and Oslo.

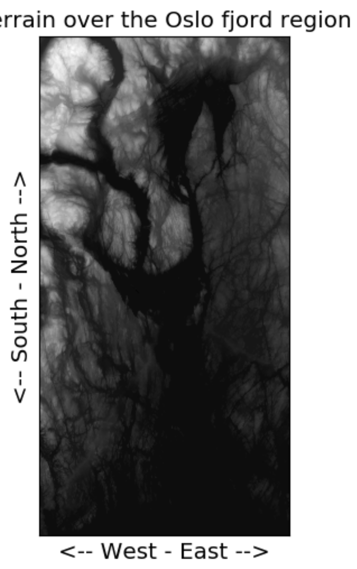


Figure 6. A topographical map of the Oslo fjord region and its surroundings. This is the map data we will be attempting to model using the linear regression methods.

B. Algorithms of Regression Analysis & Resampling

We have already discussed the theory behind the methods themselves in sections II A and II B, and we will now discuss how we will implement these in a program.

1. *Ordinary Least Squares (OLS)*

Having gotten the data set $\mathcal{L}$ and split it into a training set and a test set, we will follow the following list of actions for the OLS scheme.

- Scale the design matrix by subtracting the mean of the training design matrix columns away. We subtract the mean of the training design matrix columns away from both the test and training design matrices.

- We find the optimal β-parameter values using equation 10, using the training design matrix $\mathbf{X}_{train}$ and the corresponding "true value" training data values $\mathbf{y}_{train}$.

- We perform the dot product in equation 11, with the vector of optimal β parameters we've found in the previous step, together with the design matrices for the training and test sets. We then find the approximated true values our model gives us for the training and test sets respectively.

- We can then use the approximated true values $\tilde{\mathbf{y}}_{train}$ and $\tilde{\mathbf{y}}_{test}$ to test the accuracy of our model using MSE, R2-score, etc..

- Eventually repeat this for a set of different model complexity levels.

We can then attempt to optimize the given MSE values using resampling techniques, or analyze how they differ as a function of complexity of our model.

2. *Ridge Regression*

Given that we have got a data set and split it into a training set and a test set, we perform the following operations for the Ridge-scheme:

- We scale by subtracting the mean of the training design matrix columns away from both the test and training design matrices.

- We split our training set further down into a training set and a validation set.

- We find the optimal β-parameter values using equation 14, using the training design matrix $\mathbf{X}_{train}$. We repeat this process for a series of different hyperparameter values λ.

- We perform the dot product in equation 15 using the β-values calculated above on the validation design matrix to get an array of approximated values $\tilde{\mathbf{y}}_{validate}$

- We calculate the Mean Squared Error of the results for every single hyperparameter λ with the validation set and the corresponding approximated values. The λ which gives the lowest error score will then be used to model the data. The corresponding β-values will be taken as the optimal.

- We can then perform the dot product in equation 15 on the test design matrix using the above β-values. We then use these calculated $\tilde{\mathbf{y}}_{test}$-values to calculate the final accuracy of our model using MSE.

- Eventually repeat these steps for a set of different model complexity levels.

We can also further attempt to optimize the given MSE values using resampling techniques, or analyze how they differ as a function of complexity of the model.

3. *Lasso Regression*

Given a data set that is split into a training set and a test set, as well as the implemented lasso method of SKLearn, we perform the following general algorithm for the Lasso regression method:

- We initialize a loop over a set of hyperparameter values $\lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_n\}$
- Using the lasso method of the linear model package of SKLearn, we fit our model to the training design matrix $\mathbf{X}_{train}$ and corresponding data points $\mathbf{y}_{train}$.
- We then predict new values for the test values $\tilde{\mathbf{y}}_{test}$ corresponding to the design matrix $\mathbf{X}_{test}$.
- We calculate the accuracy score for the specific value of λ_i, and store this value together with the model. We then repeat these steps for all values of λ
- We then pick the value of λ that gives us the best accuracy score, and use the corresponding model as the one of our choice. We use the corresponding accuracy score and model
- Eventually repeat this for a set of different model complexity levels.

4. Resampling Techniques - Bootstrap

Given a data set properly split up into training and test sets, and already scaled, the following process is to be applied for the Bootstrap resampling technique:

- Create a bootstrap sample of same length as your data set, using randomly sampled data from said data set.
- Perform a linear regression of your choice on this set, and extract an accuracy score corresponding to your results. Store this accuracy score.
- Repeat the above steps for n different bootstraps.
- Take the mean value of the accuracy scores from the n bootstrap samples

The mean value from this will then be considered your total accuracy score.

5. Resampling Techniques - Cross Validation

Given a data set properly split up into training and test sets, and already scaled, the following process is to be applied for Cross-Validation (Specifically LOOCV):

- If you have an ordered data set, shuffle the data set such that the order of your data set is randomized
- Split your data set into K different folds of approximately the same size.
- Perform a linear regression on $K-1$ of your folds. Use the modelled result on your test data set to extract a set of predicted values $\tilde{\mathbf{y}}_{pred}$ corresponding to this fold.
- Use an accuracy score of your liking (preferably MSE) to calculate how precise the model is. Store this accuracy score.
- Repeat this process such that every single fold is used as a validation set exactly once.
- Take the mean value of the accuracy scores from the K different runs.

The mean value you calculate in the end will then be considered your final accuracy score.

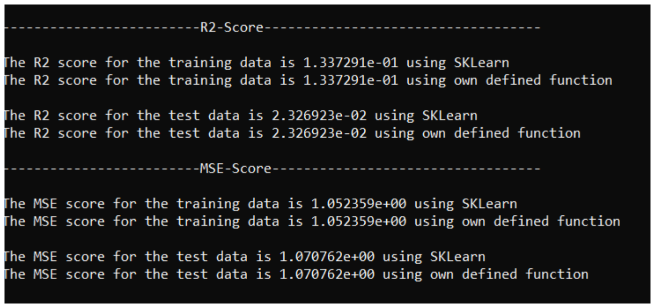
IV. RESULTS

The results will be split into a section on the functional analysis of the Franke function and an analysis of the map data regression. The Franke function analysis will be a more thorough one, with more emphasis on the actual methods and how they differ. The map data analysis will emphasize the qualitative results that the regression yields.

A. Franke Function Regression Results

We will be looking closer at the MSE and R2 scores for a 5th order polynomial, and compare this result to the SKLearn package results. We will also check the MSE accuracy score for the different methods as a function of complexity of our model. We will be looking at how the Bias-Variance trade-off looks like using bootstrap. We will also look closer at the results from doing Cross-Validation. Using the Ridge and Lasso-methods, we will also look at the ideal hyperparameter value λ for a given complexity.

1. *R2-score, Mean Squared Error & Confidence Intervals for Ordinary Least Squares*



```
-----------------------R2-Score---------------------------------

The R2 score for the training data is 1.337291e-01 using SKLearn
The R2 score for the training data is 1.337291e-01 using own defined function

The R2 score for the test data is 2.326923e-02 using SKLearn
The R2 score for the test data is 2.326923e-02 using own defined function

-----------------------MSE-Score--------------------------------

The MSE score for the training data is 1.052359e+00 using SKLearn
The MSE score for the training data is 1.052359e+00 using own defined function

The MSE score for the test data is 1.070762e+00 using SKLearn
The MSE score for the test data is 1.070762e+00 using own defined function
```

Figure 7. Command line output of the resulting accuracy scores for an OLS regression on test and training data respectively for a 5th degree polynomial. The run used 1000 sample points, in which 20% was set aside as test data. The noise term ran as $\epsilon \sim N(0, \sigma^2 = 1)$.

In figure 7, we see the accuracy score results of an ordinary least squares run for a 5th degree polynomial. We've specifically looked at the R2 score and the MSE values as metric parameters. To affirm that our functions are working correctly, we've compared the metric results with that of the SKLearn package, which brings us similar results.

In figure 8, we see the different values for β_j (as given with their corresponding factor), together with its associated 95% confidence interval. We observe that the confidence intervals for the purely x and y terms are large,

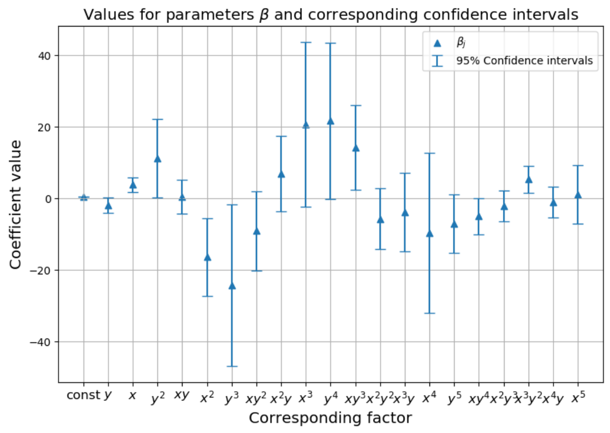


Figure 8. Confidence intervals for $N = 200$ data points, using a test size of 20% and noise running as $\epsilon \sim 0.2 \cdot N(0, 1)$.

while the terms containing some mix between them (e.g. x^2y^2) have smaller confidence intervals.

2. *Complexity and Accuracy Score of Test & Training Data*

In figure 9, we see the MSE scores for the training and test data using the OLS scheme. We see that the test data MSE starts to grow a lot after a while, which is a clear cut sign of overfitting.

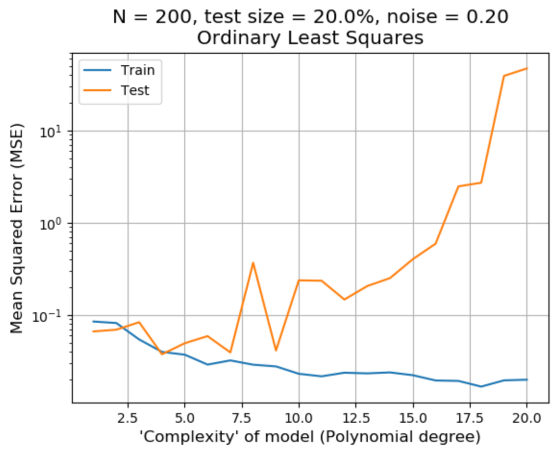


Figure 9. Complexity of model plotted against the MSE value for both training and test data. We used a total of $N = 200$ data points, with a test size of 20% and a noise term running as $\epsilon \sim N(0, 1)$.

In figure 10, we see the MSE scores for the training and test data using the ridge method. In orders of magnitude, the test data manages to closely resemble that of the training data for this case, especially compared to the case with OLS in figure 9

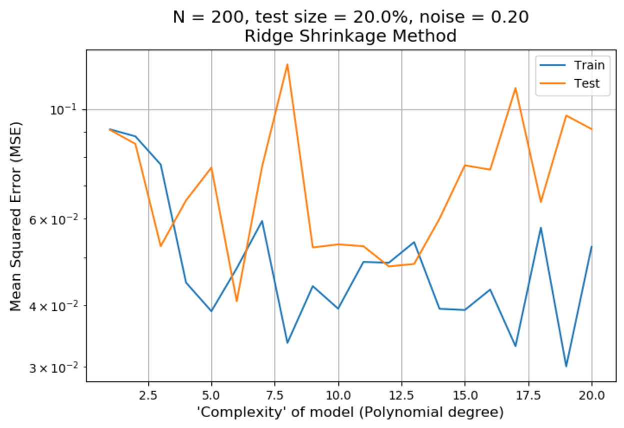


Figure 10. Train vs Test MSE for Ridge. This is for a test size of 20%, a total of $N = 200$ data points and noise running as $\epsilon \sim \frac{1}{5}N(0,1)$.

In figure 11, we see the MSE scores for the training and test data using the lasso method. The results presented here are comparable to those when using ridge.

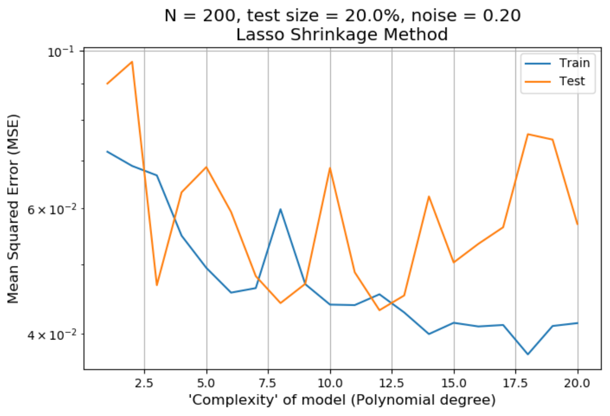


Figure 11. Train vs Test MSE for Lasso. This is for a test size of 20%, a total of $N = 200$ data points and noise running as $\epsilon \sim \frac{1}{5}N(0,1)$.

In figure 12, we see the MSE scores on the test data for all three methods compared on top of each other. We observe that the shrinkage methods both yield significantly better results than OLS does. Increasing the noise tenfold and using half the points(as seen in figure 13) further confirms this trend. Ridge and Lasso between the two seem to produce equally good models. Increasing the complexity doesn't seem to separate the two methods either (as seen in figure 14).

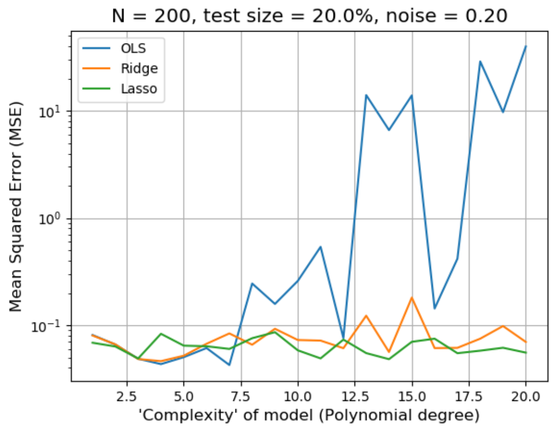


Figure 12. Comparison between the three methods (OLS, Ridge and Lasso), using the MSE-score on the modelled test data. Here, we've used $N = 200$ data points, and noise running as $\epsilon \sim \frac{1}{5}N(0,1)$.

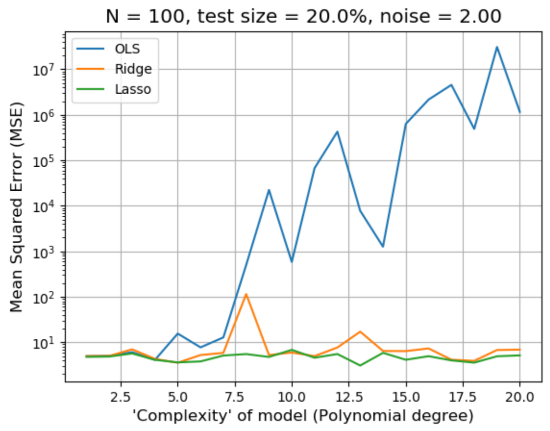


Figure 13. Comparison between the three methods (OLS, Ridge and Lasso), using the MSE-score on the modelled test data. Here, we've used $N = 100$ data points, and noise running as $\epsilon \sim 2N(0,1)$.

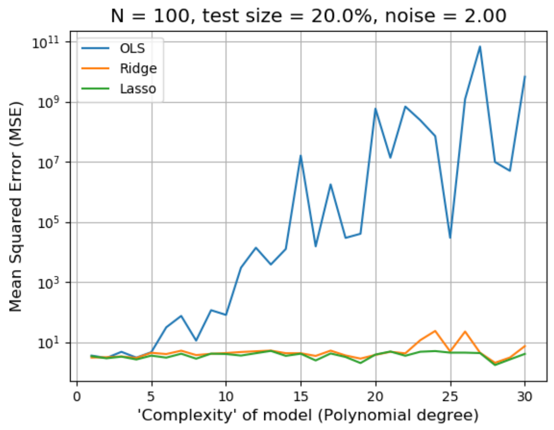


Figure 14. Comparison between the three methods (OLS, Ridge and Lasso), using the MSE-score on the modelled test data. Here, we've used $N = 100$ data points, and noise running as $\epsilon \sim 2N(0, 1)$. The polynomial degree is here run up to a degree of 30.

3. *Bias-Variance trade-off w/ Bootstrap*

In figure 15, we see the Bias-Variance trade-off from running the Ordinary Least Squares scheme. We note that, for lower complexity, bias is the dominating contribution to the MSE value. With increasing complexity, we see that the variance starts to dominate. (Note the logarithmic y-axis. At higher complexities, Bias is several orders of magnitude lower than the variance).

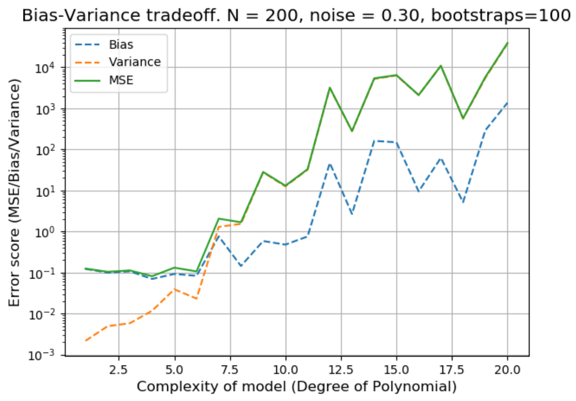


Figure 15. Bias-Variance trade-off using the OLS scheme for 200 data points, 100 bootstrap samples and a noise term running as $\epsilon \sim \frac{3}{10}N(0, 1)$. We note that bias and variance are similar at roughly polynomial degree 7 for this run.

In figure 16, we see the Bias-Variance trade-off from running the Ridge shrinkage scheme. We see the same tendencies here with the Bias dominating for lower complexities, while the variance starts to take hold as complexity increases. We don't observe that the Bias and Variances cross each other at any point however.

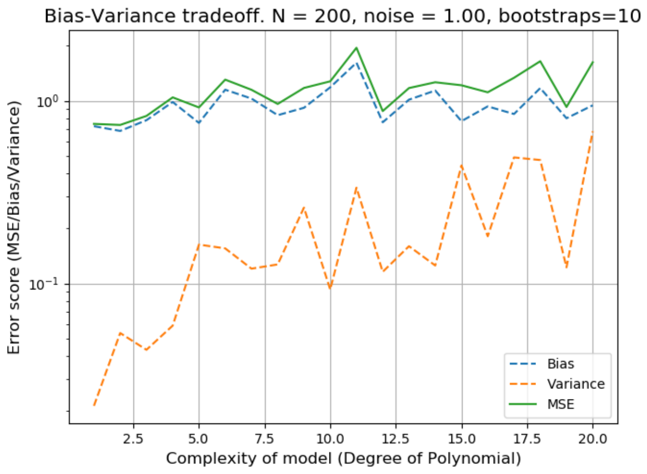


Figure 16. Bias Variance trade-off for the Ridge method up to degree 20 for $n = 10$ bootstraps, $N = 200$ data points and noise running as $\epsilon \sim N(0, 1)$. Run times for this specific program scaled terribly with higher degree of polynomials, and as such we decided to stop the runs for a max polynomial degree of 20.

In figure 17, we see the Bias-Variance trade-off from running the Lasso shrinkage scheme. We see that these results are quite similar to those of running the ridge method.

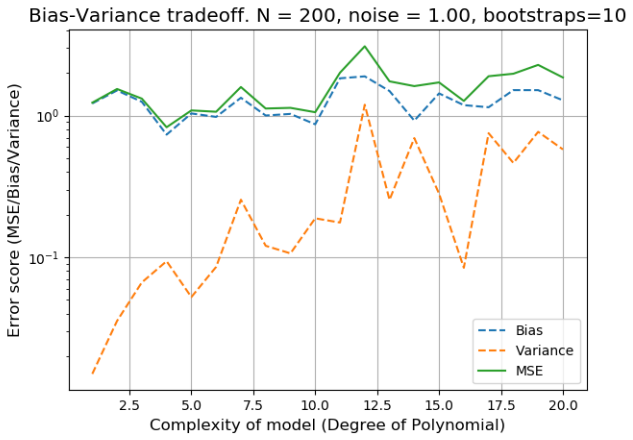


Figure 17. Bias Variance trade-off for the Lasso method up to degree 20 for $n = 10$ bootstraps, $N = 200$ data points and noise running as $\epsilon \sim N(0, 1)$. Run times for this specific program scaled terribly with higher degree of polynomials, and as such we decided to stop the runs for a max polynomial degree of 20.

4. *Cross Validations*

In figure 18, we see the comparison of the MSE accuracy score results between using cross-validation and bootstrap as resampling methods when using the OLS scheme. We see that the cross-validation method for this specific case and run seems to yield better results than the bootstrap case.

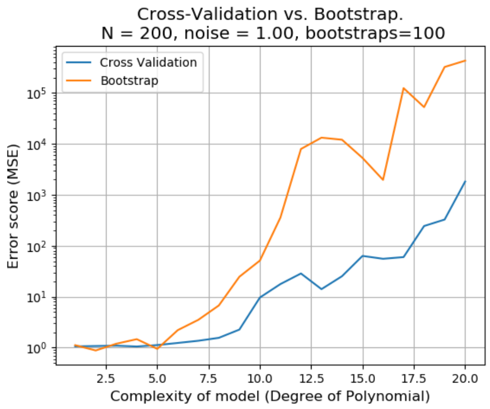


Figure 18. Cross Validation and bootstrap plotted vs corresponding MSE for $K = 5$ folds and $n = 100$ bootstrap samples, using $N = 200$ data points and a noise term running as $\epsilon \sim N(0,1)$. This is for the Ordinary Least Squares (OLS) scheme.

In figure 19, we see the comparison of the MSE accuracy score results between using cross-validation and bootstrap as resampling methods when using the Ridge scheme. We see that the cross-validation method for this specific case seems to yield more stable results than the bootstrap counterpart. This might also be due to our reduction in the number of bootstraps ($n = 10$ instead of $n = 100$), which is a result of computational run times.

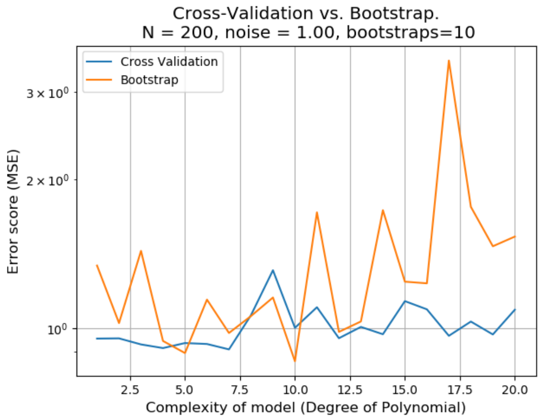


Figure 19. Cross Validation and bootstrap plotted vs corresponding MSE for $k = 5$ folds and $n = 10$ bootstrap samples, using $N = 200$ data points and a noise term running as $\epsilon \sim N(0,1)$. This is for the Ridge scheme.

In figures 20 & 21, we see the comparison of the MSE accuracy score results between using cross-validation and bootstrap as resampling methods when using the Lasso scheme. We see that the cross-validation method is extremely stable, compared to the bootstrap method. Two separate runs with different number of data points N and noise terms confirms this.

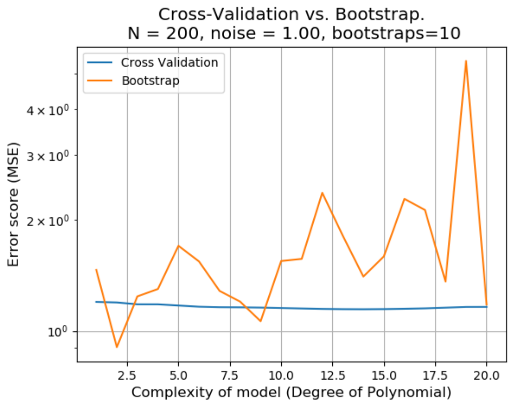


Figure 20. Cross Validation and bootstrap plotted vs corresponding MSE for $k = 5$ folds and $n = 10$ bootstrap samples, using $N = 200$ data points and a noise term running as $\epsilon \sim N(0,1)$. This is using the Lasso scheme.

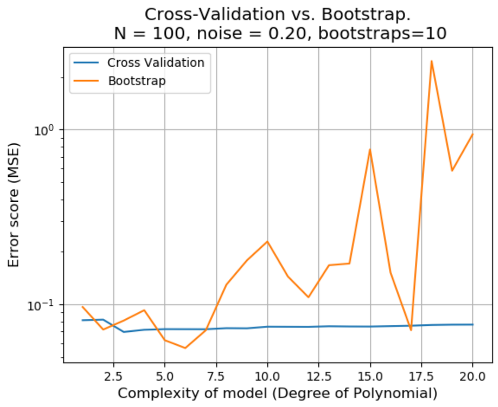


Figure 21. Cross Validation and bootstrap plotted vs corresponding MSE for $k = 5$ folds and $n = 10$ bootstrap samples, using $N = 100$ data points and a noise term running as $\epsilon \sim \frac{1}{5}N(0,1)$. This is using the Lasso scheme.

5. *Hyperparameter-fitting*

In figure 22, we see a plot of the hyperparameter fit for λ using ridge, as a function of the resulting MSE. We can here see how the MSE varies as a function of every given λ at this complexity (polynomial degree). For this complexity, we then pick out the given $\lambda \approx 0.31$ and use this when modelling our optimal fit. In figure 22 we also compare our predicted Ridge hyperparameter result to that provided by (F. Pedregosa et Al [3]). We observe that these methods separately, purely qualitatively speaking, yields comparable results.

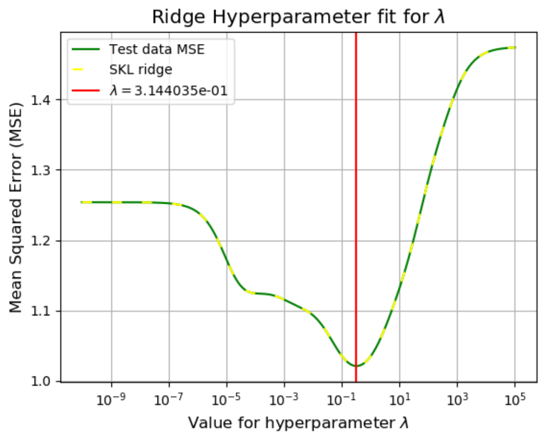


Figure 22. Hyperparameter fit for the Ridge shrinkage method. Here, we see that $\lambda \approx 0.31$ will give the optimal MSE for this given run. We're modelling this as a 5th degree polynomial, with noise running as $\epsilon \sim N(0,1)$ for $N = 200$ data points. We've overlayed the results using the SKLearn Ridge tool for confirmation.

In figure 23, we see a plot of the hyperparameter fit for λ using lasso, as a function of the resulting MSE. We can here see how the MSE varies as a function of every given λ at our given complexity.

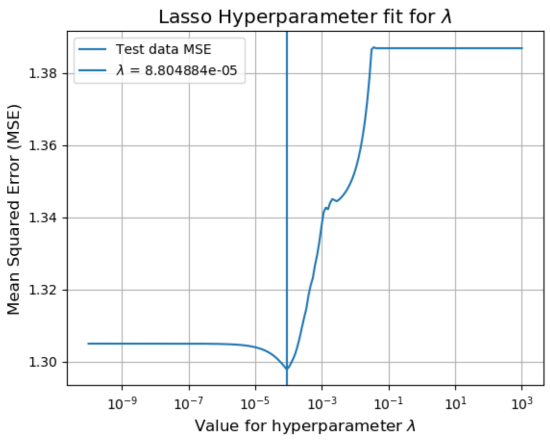


Figure 23. Hyperparameter fit for the Lasso shrinkage method. Here, we see that $\lambda \approx 8.8 \cdot 10^{-5}$ will give the optimal MSE for this given run. We're modelling this as a 5th degree polynomial, with noise running as $\epsilon \sim N(0,1)$ for $N = 200$ data points.

Here, it is important to note that we used the functionality of SKLearn to calculate. Doing this, we got a lot of error messages regarding the lack of convergence of our runs. As such, it is important to treat our results using the Lasso scheme with a pinch of salt.

B. Map Data Regression Results

Here follows the results from running the regression methods on the topography map data.

1. *Complexity and Accuracy scores*

In figure 24, we see the accuracy scores on the training and test from the topographical map for polynomial degrees between 1 and 45. We see a clear drop for the training data after roughly polynomial degree 38. We also note that the MSE values for $N = 1000$ are exceptionally high. (On the order of $\sim 10^4$).

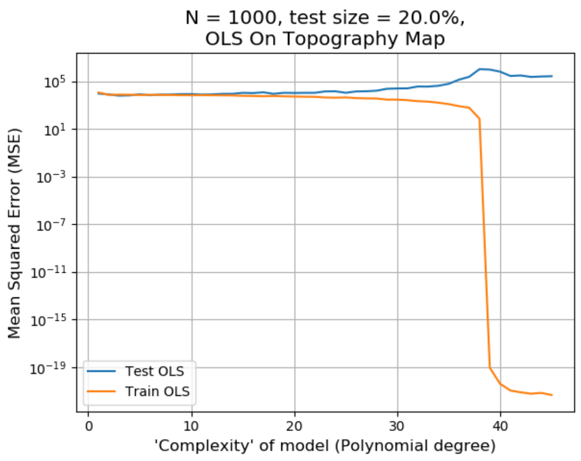


Figure 24. Complexity of model plotted against the MSE value for both training and test data on our given topography map using the OLS scheme. We used a total of $N = 1000$ data points and used a test size of 20%.

In figure 25, we see the accuracy scores on the test data for all of the different regression models when applied to the topographical map, for polynomial degrees between 1 and 45. We see that OLS score the worst among the three models, while Lasso seems to be slightly better than Ridge. The reason we limit our runs to $N = 1000$ is due to the sheer run time of the programs.

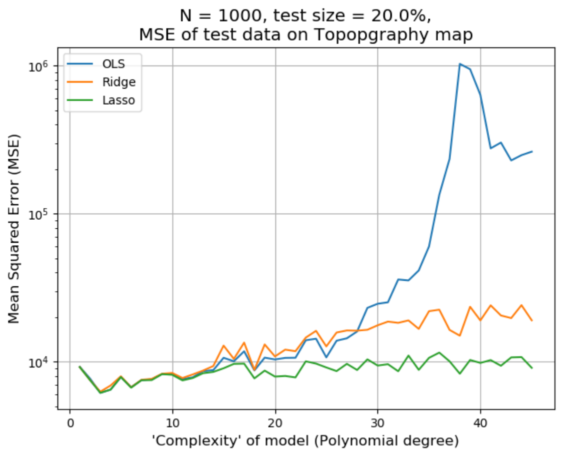


Figure 25. Complexity of model plotted against the MSE value for the test data on our given topography map using all three regression schemes. (OLS, Ridge & Lasso). We used a total of $N = 1000$ data points and a test size of 20%.

2. Recreating the topography

As a reminder, the map that we are attempting at recreating is seen in figure 6.

In figure 26, we see our attempt at recreating the map using the β coefficient values extracted from the OLS-method. This has been done for $N = 10.000$ data points at polynomial degree 30. This has not proven successful however.

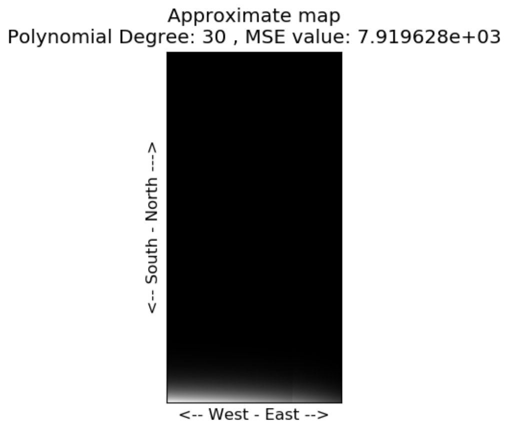


Figure 26. An attempt at recreating the map using $N = 10.000$ data points when using the OLS regression method. We have attempted this for the polynomial degree 30.

In figure 27, we see our attempt at recreating the map using the β coefficient values extracted by using the Ridge shrinkage method. This has been attempted for $N = 10.000$ data points, using polynomial degree 30. This has not proven successful either.

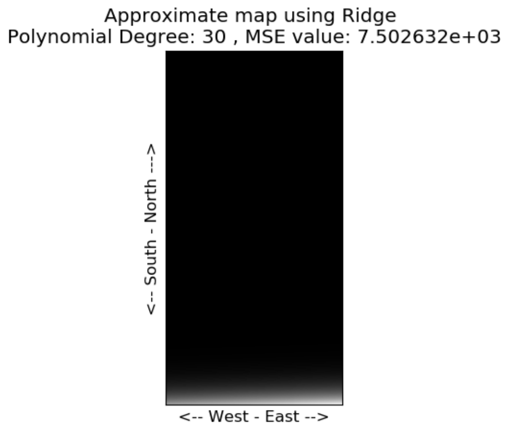

Approximate map using Ridge
Polynomial Degree: 30 , MSE value: 7.502632e+03
<-- South - North --->
<-- West - East -->

Figure 27. An attempt at recreating the map using $N = 10.000$ data points when using the Ridge shrinkage method. We have attempted this for the polynomial degree 30.

In figure 28, we see our attempt at recreating the map using the β coefficient values extracted by using the Lasso shrinkage method. This has been attempted for $N = 10.000$ data points, using a polynomial degree of 30. This, just like for the Ridge and OLS methods, has not yielded any sensible results.

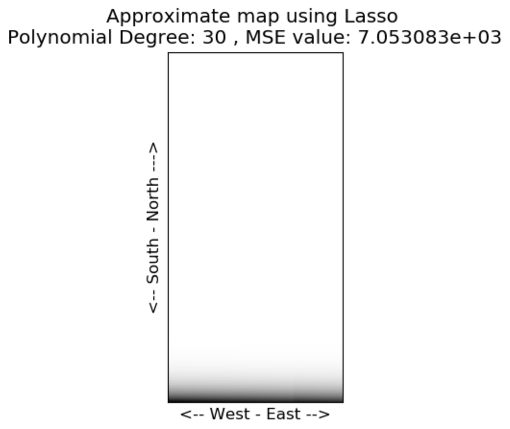

Approximate map using Lasso
Polynomial Degree: 30 , MSE value: 7.053083e+03
<-- South - North --->
<-- West - East -->

Figure 28. An attempt at recreating the map using $N = 10.000$ data points when using the Lasso shrinkage method. We have attempted this for the polynomial degree 30.

Looking at the MSE accuracy score values for all three runs, we see that they are comparable in size for this specific run. (same orders of magnitude). We also see relatively similar plots. This might indicate that the data we are trying to model shouldn't ideally use linear regression methods.

V. DISCUSSION

We will discuss three main components of the results and methods we have used to achieve these results. We will first discuss everything regarding the choice of design matrix. This is followed by a thorough discussion on our regression results when applied to the Franke function. We finish this section with discussing some of the applied data analysis on our SRTM map data.

A. The Design Matrix

The design matrix used in equation 4 is based on a polynomial expansion, much in the same way as in a Taylor expansion. As such, it would be interesting to consider other forms of functional expansions that potentially could be more suited for the data sets that we are analyzing. The form of the Design matrix is not static, and could be shaped to the problem at hand.

For example, if there would have been some known periodicity to our data, one could be lead to assume that a Fourier expansion would be more suitable. If we know that we potentially have some rolling plains in our SRTM data, we could use Fourier expansions instead of polynomial expansions since this would fit the structure of our data better.

Another example of this is the nuclear binding energy from nuclear physics. The semi-empirical mass formula, which is derived from the liquid drop model, uses the nucleon number A and proton number Z to determine the binding energy of a nucleon (See Martin & Shaw, ch. 2.3 [9]). Using the derived relation values for the liquid drop model, coupled with symmetry and pairing terms, one can fit the design matrix with very specific polynomial values for A and Z and disregard any nonphysical combinations of the two.

B. Franke Function Regression Discussion

1. MSE, R2 score & Confidence Intervals for Ordinary Least Squares

In figure 7, we see the terminal output for the mean squared error and R2-score compared side by side with SKLearns own metric function. This is a simple test to confirm that the metric functions that we have created yield a correct result, and is therefore assumed to be working correctly. This result is merely used to affirm that the coming results, using our own metric functions, will give reasonable results.

Something noteworthy we can see in figure 8 regarding the confidence intervals is that the pure x and y terms have much larger confidence intervals than the corresponding cross terms (e.g. x^2y^3). The reason for this is unclear, but it seems like this is somewhat intuitively logical. When trying to model something where the slopes don't necessarily align purely in the x- or y-directions, we need to use the cross terms to properly model such gradients, and as such the cross terms will better fit the model than the purely ”linear” terms.

Another notable thing about the confidence intervals in figure 8 is that the confidence intervals for the factors in front of polynomial degrees 3 and 4 are significantly larger than for the other terms. The reason for this is unclear, and it is merely just an observation we are making on our data.

2. MSE as a Function of Complexity

In figure 9, we see the expected relationship between the testing data and training data with regards to overfitting. At a certain point, in this case around polynomial degree 7, the MSE value for the test data increases and diverges, whereas the MSE value for the training data keep on converging. This result is as expected from the theory (See figure [1]), and as such can be used as a confirmation that our results are reasonable.

In figures 10 and 11 using Ridge and Lasso respectively, we don't see a corresponding divergence between the training and test data. This could however be attributed to our low choice in polynomial degree. These methods are therefore notably more stable and less prone to errors stemming from overfitting as compared to their OLS counterpart. Increasing the runs up to a max polynomial degree of 40 was not enough to spark a significant change either. At this point, the computational run times become a very long, and it was not attempted to run for higher polynomial degrees.

In figures 12-14, we see comparison runs between all three methods on the same runs for varying noise factors, number of data points, and polynomial ranges. In all three cases, the OLS scheme is significantly worse off than both Ridge and Lasso, and is extremely prone to overfitting as compared with both the shrinkage methods. Even though OLS runs significantly quicker than both the shrinkage methods, it is not the ideal linear regression choice in creating a precise model.

Noticing that both the Ridge and Lasso methods return significantly better results than OLS by introducing the hyperparameter λ, one is led to conclude that a correct choice for λ potentially could keep the problem of overfitting at bay. One could then discuss the significance on how one will find the optimal λ parameter. We need to search through a range, often over several orders of magnitudes, for an ideal value of

λ. The amount of points one choose will significantly impact run time, and the larger the range, the less likely you are to hit a value close to the ideal hyperparameter. This is a trade-off between accuracy and computational speed which could significantly alter how precise our model truly is.

One thing to note for the Lasso method is that our cost function doesn't have an analytic solution when regarding the derivative. For this reason we use numerical methods to approach the derivative and in finding the zero of this (to be used in finding the minima of the cost function). Two problems that potentially could induce error are the following:

- We could be stuck at a saddle point.
- We could be stuck at a local minima

The implications of this is that we don't find the optimal model, but end up on a sub-optimal model. This in turn means that we could get better results using other models if we believe we are prone to this. Looking at roughly polynomial degree 4 on figure 12, we see the Lasso-method return slightly worse results than the other models. This could stem from a sub-optimal model as a result of the gradient method used.

3. Bias-Variance trade-off Using Bootstrap

Looking at the Bias-Variance trade-off for the OLS-scheme as seen in figure 15, we get expected results. For lower complexities of our model, the Bias is the dominating term between the two, but as complexity increases, they swap roles, and variance becomes the dominating term between the two. This is as expected from the theory as we have discussed, which further affirms that our script is working as intended.

Looking at the corresponding Bias-Variance trade-off plots for Ridge and Lasso in figures 16 and 17 respectively, we see some of the same tendencies, but not the full blown extremes between the bias and the variance. This is probably due to similar reasons as in the previous subsection regarding Ridge and Lasso: we don't run the model for a large enough polynomial degree. We can however see that the Bias is by far the dominating term in the beginning for both figures, but as complexity increases, variance starts to take hold. We could therefore expect that variance will increase above bias for large polynomial degrees, but this is something we have not had the time to address.

We also need to acknowledge that we only used 10 bootstraps for both the runs using Ridge and Lasso, while we used 100 bootstraps when running OLS. This is mainly due to Ridge and Lasso runs being time consuming, and using bootstrap you're in reality doing the same (time consuming) calculations n times. (Where n is the number of bootstraps). We therefore here return to the case on efficiency vs. precision, and the trade-off between the two.

The time consuming part of the calculations of Ridge and Lasso is running through the hyperparameters λ to find the ideal one. A potential solution to this is to find the ideal λ for a given bootstrap, and use this λ in all of the other bootstraps, effectively optimizing. This will however introduce the assumption that the ideal λ for any given polynomial degree is constant, and doesn't change depending on your data set. Whether or not this is true is unknown, and we have chosen not to spend time addressing this any further.

Another overall observation one can make in figures 15-17 is that the bias and variance never sum to be larger than the MSE. From equation 20, where MSE is represented by the Bias, Variance and noise term, this result is as expected. Doing bootstraps over several instances and averaging over them will lower the significance of the noise, and we will in reality be left with MSE being the sum of the Bias and Variance. This is exactly what we are seeing on our plots, and as such proves to be another confirmation of our programs eligibility.

4. Cross Validation

In figure 18, we can see the comparison of the Cross-Validation and Bootstrap resampling methods when using the OLS-scheme. We've specifically picked $K = 5$ number of folds for the Cross-Validation and $n = 100$ number of bootstraps. In this case, the LOOCV-method seems to hold an edge over the Bootstrap-method in regards to giving the best results, but we are only led to conclude this for this specific combination of K and n.

In figure 19, we can see the comparison of LOOCV and Bootstrap resampling methods using the Ridge-scheme. We've decreased the number of Bootstraps down to 10 due to computational run times, but in general it seems like the results for LOOCV are slightly more stable than that of the bootstrap method. One should also note that, qualitatively speaking, the runs using $K = 5$ folds and $n = 10$ Bootstraps approximately have the same run times, and as such, the runs should be comparable. In general these results seem to indicate that LOOCV is superior to Bootstrap in the modelling of the Franke function.

In figure 20, we can see the comparison of LOOCV and Bootstrap resampling methods using the Lasso scheme. Here, we immediately notice that the Cross-Validation results are extremely stable as compared to bootstrap. To further attempt to confirm if this is a valid result, we have in figure 21 lowered the number of data points and lowered the noise, and we still have the same stability present. Given that the Lasso-method yields error messages when run regarding non-convergence of objects, this could be a result of a systematic error in our implementation of the Lasso-model. It could also just be attributed to Lasso yielding extremely stable results when using LOOCV.

Overall, assuming that the results from running the Lasso-scheme in fact is correct, we are led to believe that the Leave-One-Out Cross Validation technique in fact will yield better, or at least more stable results, than the bootstrap counterpart. This however applies for our given choice of number of bootstraps and folds, and it is notable that we only used 10 bootstraps for both Ridge and Lasso. Increasing the number of bootstraps will likely better the score, but to what degree is unknown. Due to long computational time for higher number of bootstraps, we have chosen not to look any further into this.

5. *Hyperparameter Tweaking for Ridge & Lasso*

In figure 22, we can see the MSE value plotted against our choice of hyperparameters λ when using the Ridge shrinkage method. In an attempt to optimize for the best possible value λ, we pick out the global minimum within the range we are looking for. We've also plotted SKLearn's ridge method on top, and we observe that they overlap more or less perfectly. As such, we are led to believe that we have implemented the Ridge shrinkage method correctly, and we are correctly identifying the global minimum of the MSE accuracy score.

In figure 23, we have the MSE value plotted against our set of hyperparameters λ when using the Lasso shrinkage method. Our result somewhat resembles that of those when using Ridge in figure 22, but this has the added plateaus before and after the interesting bit roughly in the middle. Whether this is an expected anomaly of the Lasso method, or an indication that the method is implemented incorrectly, we are unsure of. This should also be treated together with the fact that using SKLearn's Lasso method yields convergence warnings, such that there are several alarm bells ringing to cast doubt on these results. However, assuming that these results are legitimate, our algorithm is able to pick up the global minimum MSE correctly, as indicated in the figure with the vertical line.

C. Map Data Regression Discussion

We now move over to discussing the results from doing some of the same analysis on the map data.

1. *Accuracy Scores for Map Data Analysis*

In figures 24, we have plotted training and test data together for the OLS scheme in the same way as we did for the Franke function in figure 9. We can here see that the training and test data are pretty close by all the way up to roughly polynomial degree 30. This means that we are not hitting the point of overfitting exceptionally early as we did for the Franke function case above. We should note that we have a really high MSE value ($\sim 10^4$), so we don't necessarily have any good reason to say that the model is a good fit at any point here. This is however relative, given that the function values of what we are modelling could be large, which also could attribute to a large MSE. The reasons could be several.

Another thing to note in figure 24 is the large drop in MSE of the training accuracy at roughly polynomial degrees 38 to 39. One key contributor here is likely the limitation in our data set. Using $N = 1000$ points, our training set would consist of $N_{train} = 800$ points. At polynomial degree 39, the number of different parameters to fit is 820. As such, we would actually have a larger amount of degrees of freedom when fitting the model than amount of data points to fit to. This probably is the key contributor for the sudden increase in accuracy.

Looking at figure 25, we have tested the different methods on our set of real data to see how the different methods stack up. As expected, OLS starts overfitting at an early stage, and the corresponding MSE diverges. However, among Ridge and Lasso, we can see that Lasso consistently yields slightly better results than Ridge. As such, one is led to believe that Lasso would be the optimal choice of a linear regression model when trying to fit our topography data.

2. *Recreating the Terrain Data*

Looking at figures 26-28, we see our attempt at recreating the maps using the β-values. This has quite obviously proven unsuccessful, and looking at the MSE accuracy score, we can see that it is pretty high as compared with the runs for the Franke function. This seems to indicate that trying to find a polynomial approximation to advanced data sets is a sub-optimal approach.

We also need to note that these runs were done for $N = 10.000$ data points, for polynomial degree 30. This is a painstakingly slow process, and attempting to increase the number of data points or increase the order of the polynomial will further slow this process. This, coupled with the memory-limitation of most computers to store large matrices (for $N = 100.000$ and polynomial degree 30, we would roughly have 1 GB of memory used to store the matrix), means that this method of approaching an ideal model scales terribly.

D. Other noteable things

1. *Different Disregarded Metrics*

Another metric that we could have tested, and one that has been addressed several times already, is the aspect on computational time. Some of these methods, depending on how they are implemented, use orders of magnitude more time to compute compared to others. This depends on everything from the number of data points, number of folds for Cross-Validation, number of bootstraps, number of hyperparameters to check through, and the list goes on. This is something that has not been regarded in this project, and it should be definitely be addressed. Getting quantitative results on how much time the different methods use in comparison to one another is a significant factor when deciding which method is ideal for a given situation.

Some other metrics which we have not tested directly, and potentially could be addressed and be of interest, include:

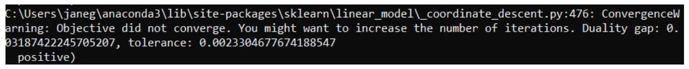
- The number of hyperparameters λ to run through, compared to the accuracy score and time spent running the script
- Checking how the accuracy score changes when the test size (for splitting training and test data) is varied.
- Test Cross-Validation and Bootstrap for a given polynomial degree, and vary the number of folds and bootstraps to see how the resampling methods stack up in that manner.

These are some of many others that we might not have thought about. The potentials for other metrics to test and analyze are endless.

2. *Doubts Regarding the Implementation of the Lasso Shrinkage Method*

In general, there are two main concerns that make us uncertain as to if Lasso has been correctly implemented. The biggest concern stems from the convergence warning which is flashed every time you run the program (see figure 29). This can be fixed by increasing the default tolerance, but this in turn also decreases the accuracy. The results SKLearn yields when it doesn't converge is however unclear, and is what primarily casts doubt on the Lasso results in this report.

Another concern we have is the results when plotting for the hyperparameter λ (see figure 23), where we have extremely flat plateaus for large and small values of λ. From a purely intuitive standpoint this seems like an anomaly, so it also casts doubt on our Lasso results.



```
C:\Users\janeg\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceW
arning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 0.
03187422245705207, tolerance: 0.0023304677674188547
  positive)
```

Figure 29. Screen capture of the convergence warning which is received when attempting to run the Lasso method from the SKLearn repository.

From figure 25, we have an indication that the Lasso method should be better at modelling the map data compared to both Ridge and OLS. If we could confirm this somehow, then this would confirm that our Lasso method is indeed working correctly. In figures 26-28, we have attempted to do recreate the maps using our β-values. This has however not worked, even with $N = 10.000$ data points, so we are not closer to affirming whether our implementation of Lasso is working or not.

VI. CONCLUSION

To conclude, we need to look at the two data sets we're analysing each on their own.

For the Franke function regression analysis, we have been getting pretty consistent results compared to the theory. For the OLS case we're getting a large degree of overfitting, while the shrinkage methods manage to keep the overfitting problem at bay. The Bias-Variance relationships also evolve as expected. Comparing Bootstrap to Cross-Validation yields us that Cross-Validation in general works better than Bootstrap, both in terms of accuracy and stability. Comparing the regression methods side by side, we see that Ridge and Lasso seem to function more or less identically, while the OLS-scheme has a large tendency to overfit the data.

For the map data case, we see that the OLS scheme, as with the Franke function case, gets overfitted quite badly as compared to the other methods. Here, we however note that Lasso actually consistently yield slightly better results than Ridge. Our attempt at confirming this by recreating the maps were however unsuccessful. We also note that we need to be careful to have more data points on our training set than degrees of freedom, such that our training data set MSE don't get set to machine precision.

There are several other metrics that can be used, and this, coupled with our uncertainties as to if Lasso has been correctly implemented, yields possibilities for further improvements on our work. This should further motivate on optimizing our algorithms for these, and other, data sets.

REFERENCES

[1] M. Hjorth-Jensen, "Lecture notes on machine learning, fys-stk3155/4155," (2020).
[2] T. Hastie, J. Friedman, and R. Tisbshirani, *The Elements of statistical learning* (Springer, 2017).
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).
[4] B. Anup, "What is underfitting and overfitting in machine learning and how to deal with it." (2018).
[5] B. Boehmke and B. Greenwell, "Hands-on machine learning with r," (2020).
[6] F. Flöck, "K-fold cross validation en," (2020).
[7] nasa jet propulsion labaratory, "Usgs eros archive - digital elevation - shuttle radar topography mission (srtm) 1 arc-second global," (2020).
[8] **NB:** We are painfully aware that some people have taken this assumption too far. This is merely meant to simplify our problem. Since we're dealing with data-sets sufficiently small, the curvature of the Earth plays little-to-no role. As such, transforming to spherical coordinates is an unnecessary hurdle. Don't worry, we are not flat-earthers.
[9] B. R. Martin and G. Shaw, *Nuclear and particle physics: an introduction* (Wiley, 2019).

Appendix A: Rewriting the Mean Squared Error in Terms of Bias and Variance Terms

We have the cost function, defined as:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big]. \tag{A1}$$

Where we assume that the true data $\mathbf{y}$ is on the following form:

$$\mathbf{y} = f(\mathbf{x}) + \epsilon = \mathbf{f} + \epsilon. \tag{A2}$$

Here we got that ϵ models random noise with a mean value $\mu = \mathbb{E}[\epsilon] = 0$ and a variance $\mathrm{var}[\epsilon] = \mathbb{E}[\epsilon^2] = \sigma^2$. We can thus write equation A1 as:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}})^2\big]. \tag{A3}$$

We can add and subtract a term of $\mathbb{E}[\tilde{\mathbf{y}}]$ inside the expectation value, giving us:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}+\epsilon-\tilde{\mathbf{y}}+\mathbb{E}[\tilde{\mathbf{y}}]-\mathbb{E}[\tilde{\mathbf{y}}])^2\big]. \tag{A4}$$

Now, we will do some clever grouping of the terms, such that we get an intended result when multiplying out the square:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\Big[\big((\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}]) + (\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}}) + \epsilon\big)^2\Big]. \tag{A5}$$

We can now multiply out the squared parentheses, giving us:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\Big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2 + (\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})^2 + \epsilon^2 + 2\epsilon(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}]) + 2\epsilon(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}}) + 2(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})\Big]. \tag{A6}$$

Using the linearity of expectation values: $\mathbb{E}[A+B] = \mathbb{E}[A]+\mathbb{E}[B]$, this can be rewritten as:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})^2\big] + \mathbb{E}\big[\epsilon^2\big] + 2\mathbb{E}\big[\epsilon(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])\big] + 2\mathbb{E}\big[\epsilon(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})\big] + 2\mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})\big]. \tag{A7}$$

Since ϵ is just noise, it is independent from any other variable. As such, we can write the product of expectation values as $\mathbb{E}[\epsilon\mathbf{A}] = \mathbb{E}[\epsilon]\mathbb{E}[\mathbf{A}]$. We also know that $\mathbb{E}[\epsilon] = 0$, so those terms vanish:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})^2\big] + \mathbb{E}\big[\epsilon^2\big] + 2\mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})\big]. \tag{A8}$$

For the 2nd term, we can swap the orders of the terms inside the square (since $(a-b)^2 = (b-a)^2$). We also note that $\mathbb{E}[\epsilon^2] = \sigma^2$. Also knowing that both $\mathbb{E}[\tilde{\mathbf{y}}]$ and $\mathbf{f}$ are both constants, we can pull this out of the expectation value in the last term. We thus have:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \sigma^2 + 2(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])\mathbb{E}\big[(\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})\big]. \tag{A9}$$

Using the property $\mathbb{E}[A+B] = \mathbb{E}[A]+\mathbb{E}[B]$ on the final term, and the fact that the expectation value of an expectation value is just the expectation value, the final term can be rewritten as:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{f}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \sigma^2 + 2(\mathbf{f}-\mathbb{E})(\mathbb{E}[\tilde{\mathbf{y}}]-\mathbb{E}[\tilde{\mathbf{y}}]). \tag{A10}$$

We note that the final term therefore must be equal to 0. Taking the remaining terms and writing out the definition of the expected value, we then get:

$$\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(f_i-\mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i-\mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2. \tag{A11}$$

Since the mean squared error $\mathrm{MSE}(\mathbf{y},\tilde{\mathbf{y}}) = \mathbb{E}\big[(\mathbf{y}-\tilde{\mathbf{y}})^2\big]$, this was what we were supposed to show. $\square$