# FYS-STK4155 Project 1: Regression

Jan Ole Åkerholm
(Dated: October 7, 2019)

**Abstract**

Three variants of linear regression models, ordinary least squares (OLS), Ridge regression and Lasso regression, are studied. This is achieved by applying the models to a Franke function, which provides a 2D function with properties similar to real terrain, and real terrain data from Norway. The design matrices for the linear regression models are set up using 2d polynomials up to a degree of $m = 20$. There are unfortunately problems in the code that cause unexpected and clearly incorrect results, such as a completely incorrect bias-variance tradeoff situation. OLS shows some signs of overfitting even at lower complexities, and the use of linear regression models on terrain data in general is questioned.

Link to github repository: Click here

## I. INTRODUCTION

Regression is widely used in science as a means to build models based on observed data. It allows for fitting a continuous function to discrete data sets, which can be used to predict and explain various phenomena in a range of sciences. The methods also have well studied tools for validating their accuracy.

This project explores three various methods for linear regression: ordinary least squares, Ridge and Lasso regression. First these methods will be applied on the Franke function, a continuous function which has some similarities with terrain. The methods will then be applied on real terrain data from Møsvatn Austfjell in Norway. For each case, the error in the models will be studied in some detail.

## II. THEORY

### A. Franke function

The Franke function is a two-dimensional weighted sum of exponentials which visibly shows some similarities to terrain, it is defined as:

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

A plot of the function is shown in FIG. 1

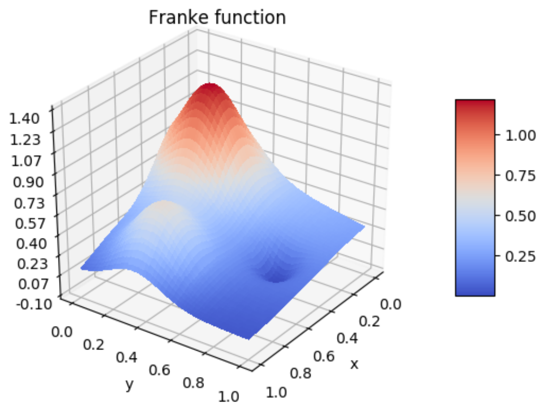

**FIG. 1:** Franke function on the domain $x, y \in [0, 1]$

### B. Ordinary least squares (OLS)

OLS and the other methods discussed here are also explained in detail by Hastie et. al. [1]

Suppose a data set of some observed phenomenon is given as:

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

where $y_i$ is the observed data for some predictor $\mathbf{x_i}$ and $\epsilon_i$ is random noise with mean value 0 and a variance of $\sigma^2$. The ordinary least squares model is then used to model the underlying function $\mathbf{f}(\mathbf{x})$ as:

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

where $\mathbf{X}$ is the so-called "design matrix", and $\beta$ are coefficients specific to the data. The design matrix can then be designed in some way that seems sensible for the problem at hand and depending on the predictors, and the model can then be created by finding the optimal parameters of $\beta$ for some data set.

Note that the predictor $\mathbf{x_i}$ may be multi-dimensional, for example in the case where the data point $y_i$ is some terrain data (e.g. height) dependant on two coordinates.

Finding the optimal parameters of $\beta$ is typically done by minimizing some cost function $C(\beta)$. In OLS, the *mean squared error* is typically used as the cost function, and is defined as:

$$C(\beta) = MSE(\beta) = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \tilde{y}_i)^2 \tag{2}$$

where $N$ is the number of data points. This can be written in matrix notation as

$$C(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) \tag{3}$$

The cost function is minimized by taking its derivative and setting it equal to 0:

$$\frac{\partial C(\beta)}{\partial \beta} = 0 \tag{4}$$

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \tag{5}$$

Solving for $\beta$ gives the normal equation for OLS:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{6}$$

The prediction can then be found by

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

#### *1. Confidence intervals of $\beta$*

Having calculated an optimal set of $\beta$-values, the confidence interval of $\beta_i$ can then be found from [1]:

$$(\beta_i - z^{1-\alpha}\sqrt{v_i}\hat{\sigma}, \beta_i + z^{1-\alpha}\sqrt{v_i}\hat{\sigma}) \tag{7}$$

where $v_i$ is the i-th diagnonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$ and $\hat{\sigma}^2$ is the variance of $\tilde{\mathbf{y}}$. $z^{1-\alpha}$ depends on how wide the confidence interval is, and for a 95% confidence interval $z^{1-\alpha} = 1.96$.

#### *2. Error measurement*

In addition to using the MSE (equation 2) to measure the quality of the model, the $R^2$ score is also used. It is defined as:

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

where $\bar{y}$ is the mean value of $\mathbf{y}$. For a perfect fit, the $R^2$ score function gives a value of 1.

## C. Resampling

When developing and testing the model the data is split into training and test data, so that the model can be tested on data which it has not been trained on. This is done to avoid training the model in such a way that it might fit the training data very well, but be completely useless as soon as it is used for data which is not part of the training (overfitting).

#### *1. K-fold cross validation*

In order to test the model on as much test data as possible, a method called K-fold cross validation is used. The method splits the data set randomly into $k$ sections (folds), and trains on $k-1$ of these sets while testing on the one which has not been part of the training. This is repeated $k$ times and each time provides one estimate for the accuracy of the model and the $\beta$ parameters. Usual values for $k$ is 5 or 10, and I have chosen to stick with 5 for this project.

## D. Ridge regression

Ridge regression is a method which adds *regularization* to the regression model. Regularization is a tool used to prevent the regression method failing if the data points are highly correlated, or to prevent overfitting of the data.

In ridge regression, the values of $\beta$ are constrained so that $||\beta||_2^2 \leq t$ for some finite number $t > 0$.

In practice the method is applied by changing the cost function so that the new problem to be minimized is:

$$C(\beta) = \frac{1}{N}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_2^2 \tag{8}$$

where $\lambda$ is some regularization parameter where multiple values should be tested in order to find the optimal result.

This leads to a new equation for the optimal values $\beta^{\text{Ridge}}$:

$$\beta^{\text{Ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y} \tag{9}$$

Adding a small value to the diagonal of $\mathbf{X}^T\mathbf{X}$ avoids problems if $\mathbf{X}^T\mathbf{X}$ is singular, and also decreases the condition number of the matrix. If $\lambda = 0$ the method is identical to OLS as expected.

## E. Lasso regression

Lasso regression is similar to Ridge regression, but instead of using the $L_2$-norm for the $\lambda$ term, the $L_1$-norm is used instead, so that the cost function now is:

$$C(\beta) = \frac{1}{N}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_1 \tag{10}$$

where the 1-norm is defined as:

$$||\beta||_1 = \sum_i |\beta_i|$$

There is no analytical solution for the optimal $\beta$ values in this case, so when using the Lasso method an iterative method is used instead.

## F. Bias-variance tradeoff

In order to gain more insight into the error of the model, the MSE can be decomposed into the bias, variance and irreducible error terms. Recall that $\mathbf{y} = \mathbf{f} + \epsilon$, and

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

Inserting $\mathbf{y} = \mathbf{f} + \epsilon$, and adding and subtracting $\mathbb{E}[\tilde{\mathbf{y}}]$:

$$\begin{aligned}\mathbb{E}\big[(\mathbf{y} - \tilde{\mathbf{y}})^2\big] &= \mathbb{E}\big[(\mathbf{f} + \epsilon - \tilde{\mathbf{y}} + \mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}])^2\big] \\ &= \mathbb{E}\big[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])^2\big] + \mathbb{E}\big[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2\big] + \mathbb{E}[\epsilon^{\mathbf{2}}] + \delta\end{aligned}$$

Where $\delta$ are terms on the form $\mathbb{E}[\epsilon]$ or $\mathbb{E}[\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{y}]$ which tend to 0 for large enough samples. Using the definitions of the expectation values then we get:

$$\mathbb{E}\big[(\mathbf{y} - \tilde{\mathbf{y}})^2\big] = \frac{1}{N}\sum_{i=0}^{N-1}(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{N}\sum_{i=0}^{N-1}(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2 \tag{11}$$

The first term is the bias squared, the second term is the variance of $\tilde{\mathbf{y}}$. $\sigma^2$ is the variance of $\epsilon$, i.e. the noise of the measurement. It is also known as the irreducible error.

In general, one expects the bias to be higher when the complexity of the model is low, and the variance to gradually increase along with the complexity of the model. This gives a tradeoff between the bias and variance, where the lowest MSE is given by some complexity where the sum of the bias and the variance is low.

## III. METHOD/IMPLEMENTATION

First the theory discussed will be applied to model a Franke function with noise. It will then be applied on real terrain data from Møsvatn Austfjell in Norway. For the rest of this report **x**, and **y** will be referring to the predictors (coordinates of the terrain), and **z** will refer to the height of the terrain.

For both the Franke function and the terrain data, the x and y axes are scaled to $[0, 1]$. Additionally, for the terrain data the height of the terrain is normalized so that the highest point $= 1$.

### A. Design matrix

For both the Franke function and the terrain data, the design matrix is set up with each row containing a polynomial on the form:

$$[x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, ...]$$

This means a polynomial of degree $m$ has

$$m\frac{m+1}{2}$$

terms in total.

### B. Code structure

The core of the code is contained in the folder "resources". The regression methods are contained in the file "regression.py" which contains a main class called "OLS()". This class contains methods for calculating the $\beta$ parameters for a given design matrix and set of data. When an instance is initialized the fit is immediately calculated, and a prediction can be made.

There are also subclasses of "OLS()" for Ridge and Lasso regression, where the fitting methods are modified to use the Ridge and Lasso methods.

The "resources.py" file contains functions for plotting a 2D surface, setting up the design matrix, finding the confidence intervals for the $\beta$ fit, finding the MSE, $R^2$ score. variance and bias. The "franke.py" file contains code for generating the Franke function.

### C. Franke function

Before starting on the real terrain data, I first started by using the methods on the Franke function. The code for this can be found in "franke/ridge_lasso_fitting.py". The Franke function is generated and gaussian noise with $\mu = 0, \sigma = 0.1$ is generated and added to the data. The K-fold split is then set up and the code runs for all three models (OLS, Ridge and Lasso), for a given set of complexities $m$ and lambda values $\lambda$ (for OLS the loop ends after one $\lambda$ value because the method does not depend on $\lambda$).

The file "testing/franke_fit.py" contains code for running the OLS regression on the Franke function and plotting the various fits. It also compares the results with Scikit-Learn's "LinearRegression" functionality, and calculates the confidence intervals of $\beta$.

### D. Terrain data

The terrain data is gathered from the Github project repository. I chose to use the data from Møsvatn Austfjell, as it seemed fairly interesting, but not too noisy and complex. The code in "terrain_fitting/terrain_testing.py" is structured in much the same way as the code for the Franke function fitting, fitting the data with various values for $m$ and $\lambda$ and plotting the results. A square section of the data set with $1000x1000$ points is selected for doing the experiments. The terrain used can be seen in FIG.

Additionally the file "terrain_fitting/final_fit.py" uses OLS only to fit the data with various complexities and plotting the resulting model. In this case I chose not to use K-fold because I suspect there might be something wrong with my cross validation code, which will be explained further in the results section.

## IV. RESULTS AND DISCUSSION

### A. Franke function

#### 1. *Ordinary Least Squares*

The result of the training of the OLS model on the Franke function with noise, up to a complexity of $m = 15$ can be seen in FIG. 2. The results are somewhat surprising, as the bias and variance are both above the MSE. As the MSE is expected to be a sum of the bias, variance and the irreducible error, this is not realistic, and I suspect it is due to some error in my code. I got the same result both with and without the cross validation, so the error must be somewhere else, but I have not had time to find it.

A clue might be that the MSE and variance appear to almost exactly balance eachother out, and the bias appears to be almost completely constant. Another clue is that the MSE drops to 0.01, or $0.1^2 = \sigma^2$, indicating that it is displaying the variance of the noise that was added to the data set.

The model showing an error that is exactly the noise of the given data set seems to be a strong indicator of overfitting, which is not surprising when the model complexity rises significantly, especially given that the Franke function is relatively simple.

**Scikit Learn comparison**
When comparing with the OLS method in Scikit Learn, the results for the fit and R2 score are the same. For $m = 2$ complexity:

| | $\beta_0$ | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ |
|---|---|---|---|---|---|---|
| My method | 1.17 | -1.031 | -0.794 | 0.100 | 0.857 | -0.301 |
| Scikit | 1.17 | -1.031 | -0.794 | 0.100 | 0.857 | -0.301 |

**$\beta$ with confidence intervals**
The $\beta$ values and their confidence intervals were calcu-

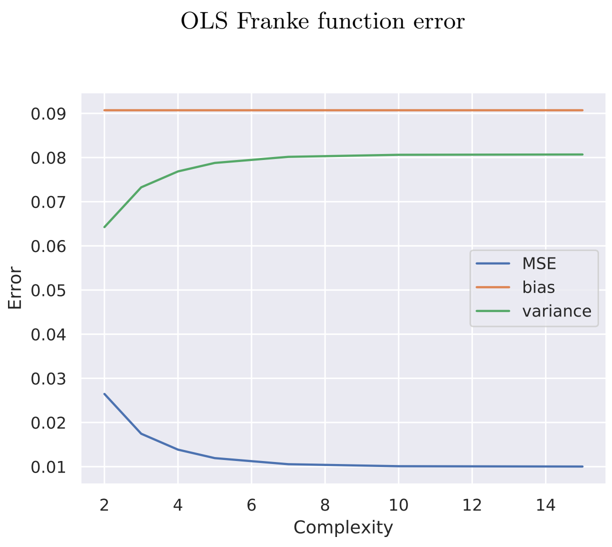

**FIG. 2:** MSE, bias and variance of the OLS model on the Franke function for complexities ranging from 2 to 15

lated as follows (for $m = 2$):

$$\beta_0 = 1.17141889 \pm 0.00255787$$
$$\beta_1 = -1.03123099 \pm 0.00744324$$
$$\beta_2 = -0.79432508 \pm 0.00752144$$
$$\beta_3 = 0.10020426 \pm 0.00660813$$
$$\beta_4 = 0.85686316 \pm 0.005875$$
$$\beta_5 = -0.30056054 \pm 0.00661286$$

### 2. *Ridge regression*

When making a models using Ridge regression, multiple values of $\lambda$ were used, the full result for the MSE can be seen in the heatmap in FIG. 3. A similar heatmap for the $R^2$ score can be found in the github repository under "franke_fitting/fig".

From the heatmap, it appears that the lower the $\lambda$-value, the better the MSE. The $R^2$ heatmap shows the same. Because the MSE and $R^2$ scores are calculated using cross validation, this appears to imply that the OLS is a better model than Ridge in this case, and that I have not experienced any overfitting yet up to a complexity of $m = 15$, but keeping in mind that there is some error in the code, making any conclusion on this is probably impossible with the current results. The Ridge regression shows the same problem with the bias-variance tradeoff as the OLS, as can be seen from FIG. 4

**Scikit Learn comparison**
When comparing with the Ridge method in Scikit Learn, the results for the fit and R2 score are the same, like in the case for OLS. For $m = 2$ complexity and $\lambda = 10$:

| | $\beta_0$ | $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ |
|---|---|---|---|---|---|---|
| My method | 1.17 | -1.027 | -0.791 | 0.098 | 0.854 | -0.302 |
| Scikit | 1.17 | -1.027 | -0.791 | 0.098 | 0.854 | -0.302 |

### 3. *Lasso regression*

For the Lasso regression, the heatmap in FIG. 5 shows the MSE as function of complexity and regularization

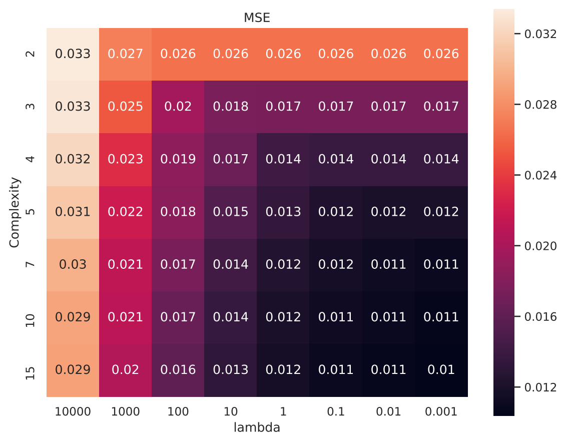

**FIG. 3:** MSE values for Ridge regression on the Franke function with noise. The lowest value is for $m = 15$ and $\lambda = 0.001$. Full size plots can be found in the Github repository.

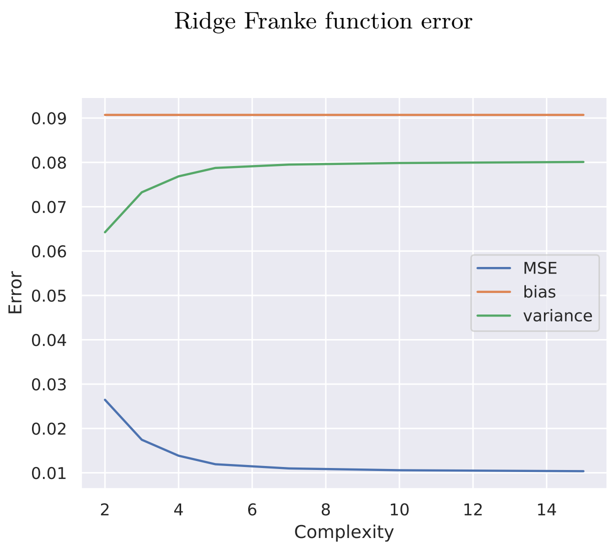

**FIG. 4:** MSE, bias and variance of the Ridge model on the Franke function for complexities ranging from 2 to 15

parameter $\lambda$. The $R^2$ score is shown in FIG. 6. Again it seems there are some problem in the code, as these do not look particularly realistic. The results appear to be *almost* independent of the complexity, although they do improve slightly for the smallest $\lambda$-values when the complexity increases. For the Lasso fitting the model from Scikit Learn was used, so it seems very unlikely that there is a problem with that model. More likely is that I was not able to properly figure out how to set up the design matrix for sending it to the Lasso method, or there is some bug in my code somewhere (seems likely). Since there are other things that also suggest there is something wrong with the calculation of the MSE, that might be a good place to start looking for bugs.

The bias-variance tradeoff also looks strange for the Lasso regression, though somewhat different than for the previous two methods, see FIG. 7.

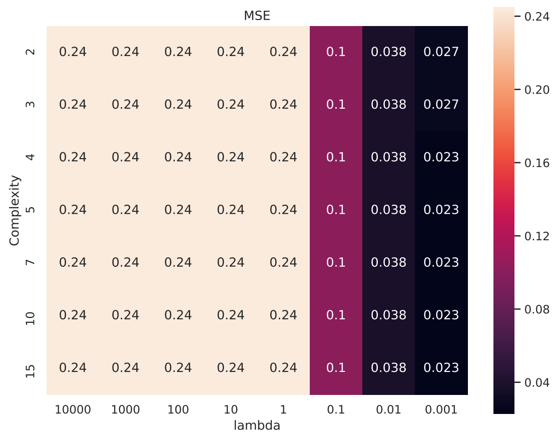

**FIG. 5:** MSE values for Lasso regression on the Franke function with noise. Full size plots can be found in the Github repository.

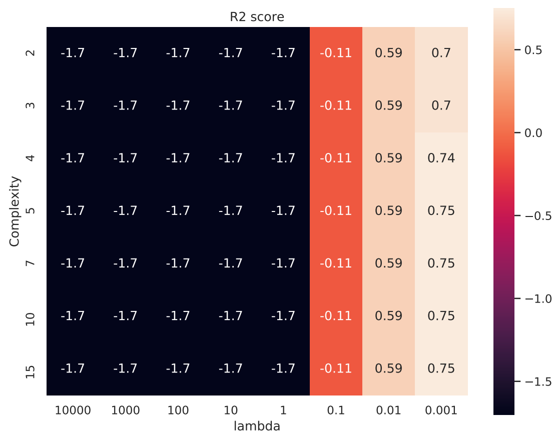

**FIG. 6:** $R^2$ values for Ridge regression on the Franke function with noise. The highest value is for $m = 15$ and $\lambda = 0.001$. Full size plots can be found in the Github repository.

### *4. Models compared*

Comparing the models with the current results seems somewhat futile given the obvious errors, that doesn't stop anyone from trying however. FIG. 8 shows the MSE for each model as a function of the regularization parameter $\lambda$. From the figure it seems that a smaller $\lambda$ is always better, and the OLS model seems to always provide the lowest MSE. Given the strange behaviour of the bias-variance tradeoff however, these results should not be trusted.

With the current result it seems that the OLS model always performs better, but at a complexity as high as 15 it seems unrealistic that there will be no overfitting, which the MSE does not show. I am not sure why this is the case. Fits of the Franke function for various complexities can be found in the appendix.

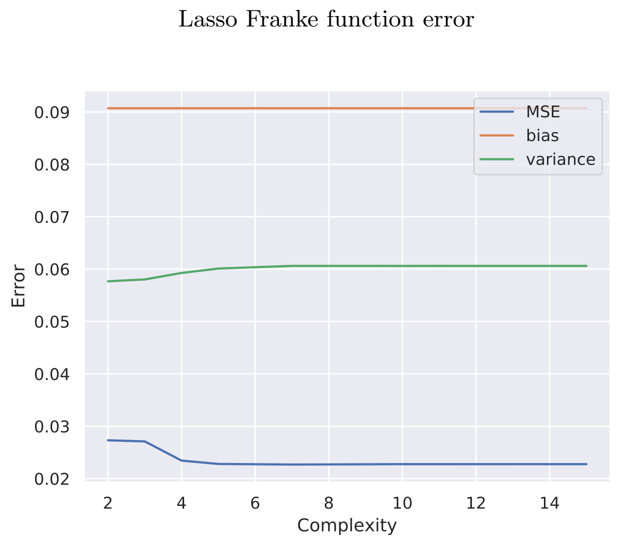

**FIG. 7:** MSE, bias and variance of the Lasso model on the Franke function for complexities ranging from 2 to 15

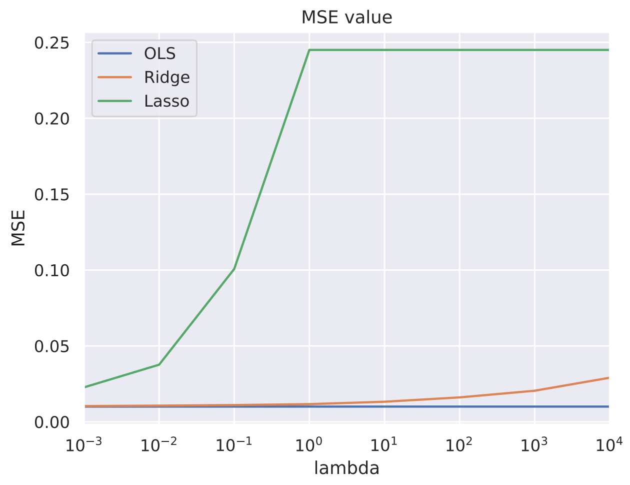

**FIG. 8:** MSE for the various models for different regularization parameters $\lambda$. OLS is not affected by changing the $\lambda$ value. Complexity is the one which provided the smallest MSE for each model ($m = 15$ for all)

## B. Terrain data

Since the terrain data was modeled in the same way as the Franke function, there are equally strange results for some of the plots. The heatmaps of the MSE values for Ridge and Lasso regression can be seen in FIG. 9 and 10.

Again it's clear that the MSE appears to improve as $\lambda$ approaches 0. In both cases the best model appears to be the model with the highest complexity $m = 20$ and lowest lambda $\lambda = 0.001$. FIG. 11 shows the MSE of the model for complexity $m = 20$. Again we see that OLS appears to perform best, even though we really should expect some overfitting at this point. This again is not realistic, but given that OLS seemed to perform best, I focused most on getting more results with OLS, rather than searching for a week for bugs in the code. With the results currently at hand it seems that the OLS performs best, and Lasso worst. The MSE of Lasso quickly rises as the values of $\lambda$ grow, and the MSE value for the Ridge model also grows with $\lambda$, especially after $\lambda \sim 10^2$.

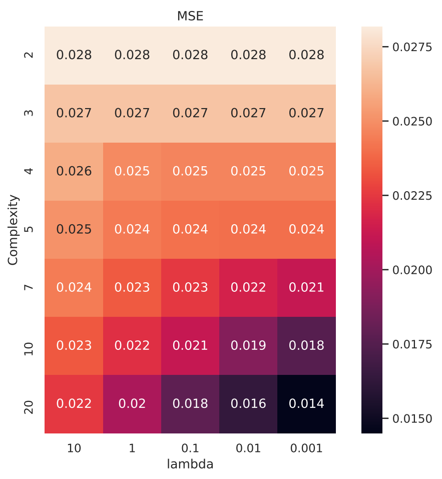

**FIG. 9:** MSE values for Ridge regression on the terrain data.

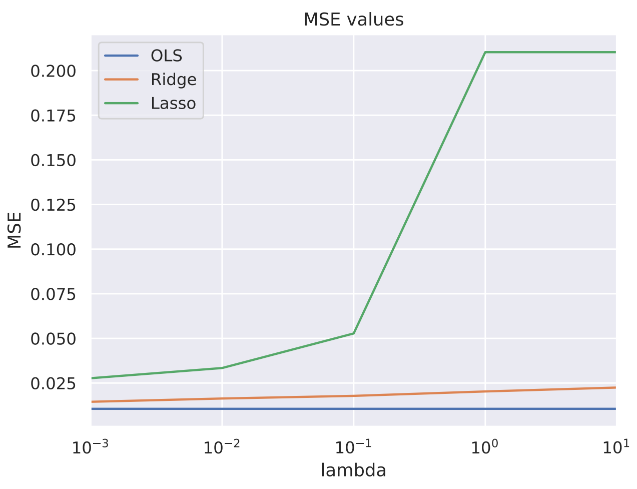

**FIG. 11:** MSE values for the different models as a function of $\lambda$ for terrain data. $m = 20$. Using K-fold cross validation with 5 folds.

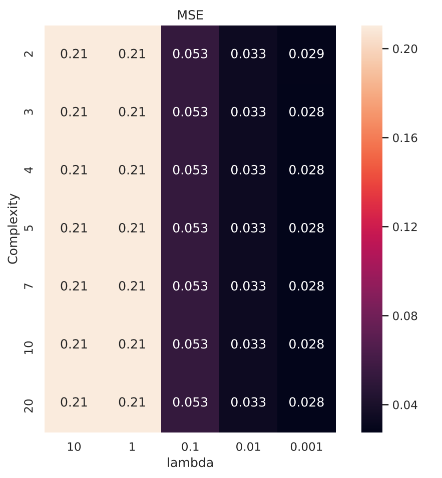

**FIG. 10:** MSE values for Lasso regression on the terrain data.

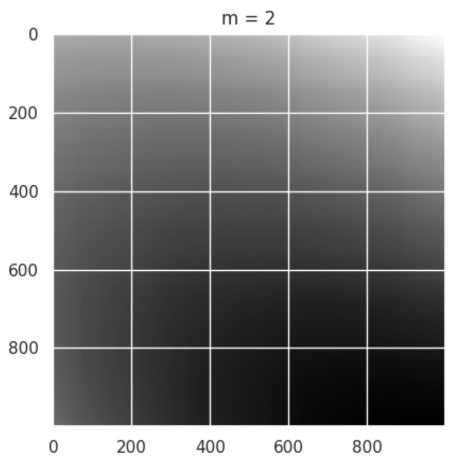

**FIG. 12:** Fit of the terrain data in FIG. 16 with a polynomial complexity of $m = 2$

### 1. *Terrain fit plots*

Using the OLS model I produced some fits of the terrain data which gave some interesting results, using complexities of $m = 2$, $m = 6$, $m = 10$ and $m = 20$. Heatmaps of these models can be seen in FIG. 12, FIG. 13, FIG. 14 and FIG. 15, respectively. From the heatmaps it's clear that as the model complexity increases, the model can represent more and more of the terrain features, but even at $m = 20$ the model is very far from representing the original data. This might be a good thing, since a perfect fit might very well also suggest that the model was overfitted to the data. Still, it is possible to recognize some features in the modeled data. For example, the dark valley moving from the southeast at about x = 800 to the west at just below y = 600 is visible in the model, and so are some other features larger features of the terrain. While some details are lost, the model appears to retain most of the major features for a complexity of $m = 20$.

## V. CONCLUSION

Three methods of regression were applied to two different kinds of data, ordinary least squares (OLS), Ridge regression and Lasso regression. The data sets were the Franke function, which is a smooth function, and terrain data acquired from the course Github page. Unfortunately there appears to be some sort of bug or mistake somewhere in the code which produces strange results for the mean squared error, bias and variance, making a proper analysis of the results very difficult.

Regardless, the regression methods all showed promise

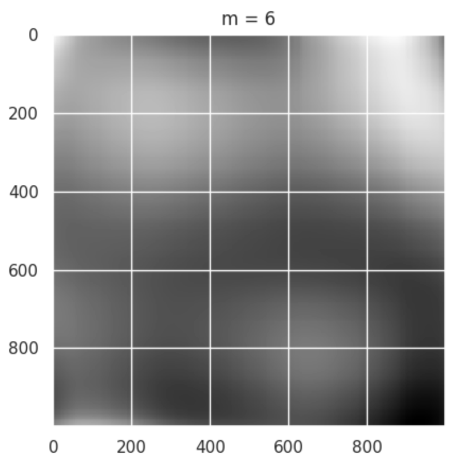

**FIG. 13:** Fit of the terrain data in FIG. 16 with a polynomial complexity of $m = 6$

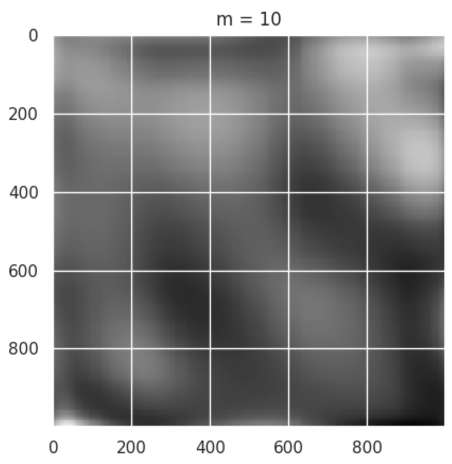

**FIG. 14:** Fit of the terrain data in FIG. 16 with a polynomial complexity of $m = 10$

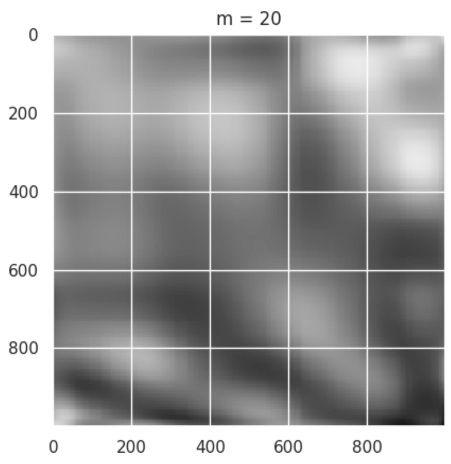

**FIG. 15:** Fit of the terrain data in FIG. 16 with a polynomial complexity of $m = 20$

in creating models that fit the given data, and the problems seem to mostly be related to the error analysis. With the given results, OLS appeared to perform better as far as a lower MSE and $R^2$ score was concerned, but there reasons to believe that the model quickly becomes overfitted when using OLS, something which should take longer with the Ridge and Lasso regression methods.

It is also worth considering that the OLS and Ridge methods are less computationally costly than Lasso, especially if the results are as good or better. In general the Lasso method took much longer to run, which is not unexpected given that it is an iterative method, as opposed to the analytical methods of OLS and Ridge. However OLS does have the advantage that it does not require a $\lambda$ value, which must first be found for the Ridge and Lasso methods, which can take a lot of time, especially if the data set is large.

### A. General remarks on linear regression models for terrain data

Using linear regression models on terrain data does not immediately appear to me as a very good use of linear regression. Extrapolating beyond the initial data set would not provide any reasonable results, as the height of terrain does not provide any information to the model about how the surrounding terrain would look. At best it seems it might work to extrapolate a very short distance outside the original data set.

Linear regression for terrain data could potentially be used for interpolating the altitude of an area if there are not enough data points to accurately cover a location, but I suspect there might be better ways to do provide such an interpolation, but that might also depend on the resolution of the data.

## REFERENCES

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2016. ISBN: 0387848576.

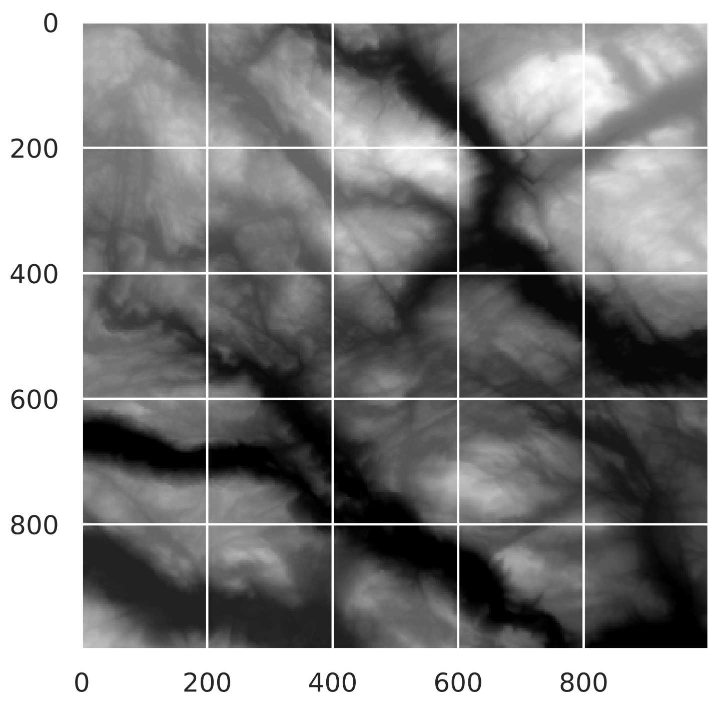

**FIG. 16:** Selected terrain data from Møsvatn Austfjell

## VI. APPENDIX

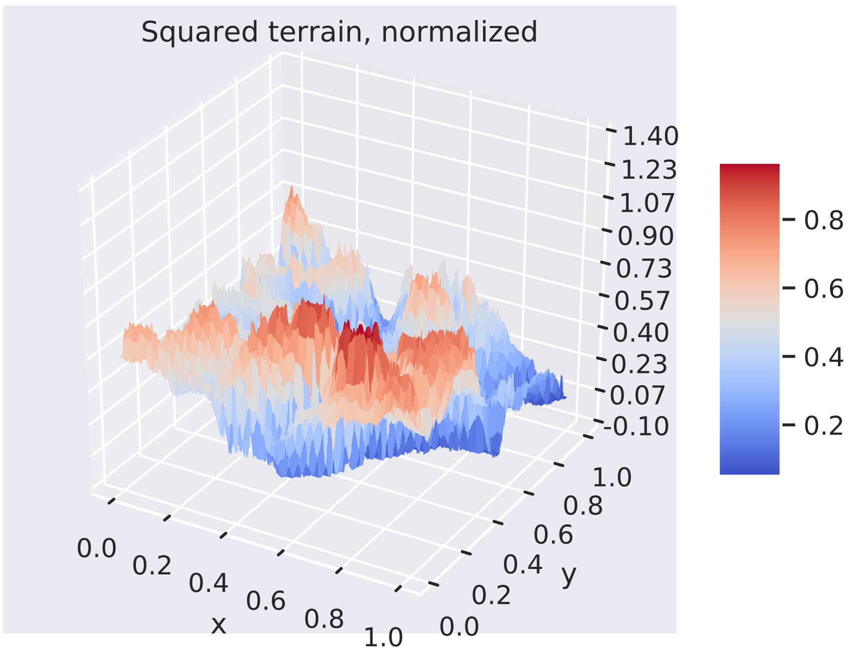

**FIG. 17:** Terrain data from Møsvatn Austfjell in 3D visualization. "Squared" implies that there are the same amount of data points on the x and y axes.

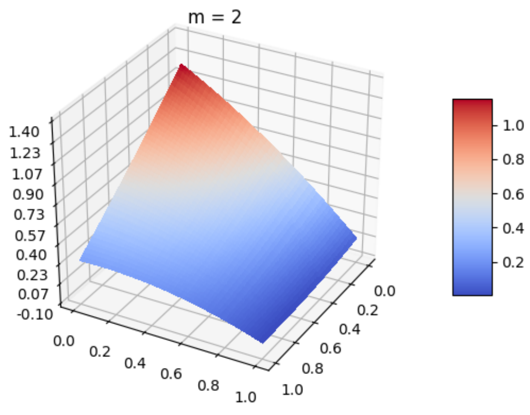

**FIG. 18:** Model fit using OLS with complexity $m = 2$

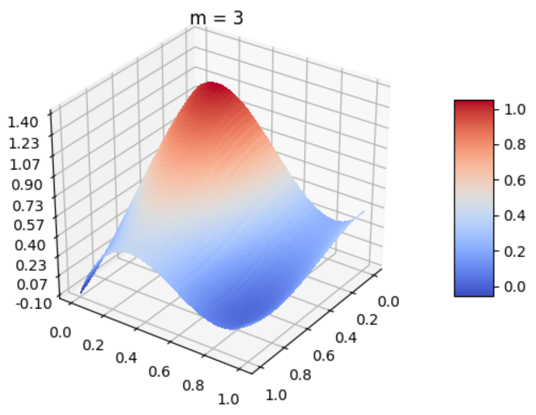

**FIG. 19:** Model fit using OLS with complexity $m = 3$

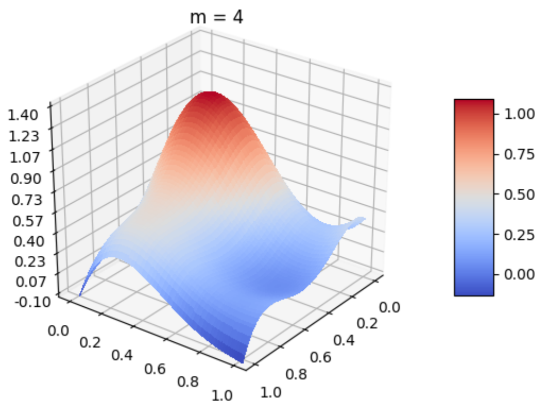

**FIG. 20:** Model fit using OLS with complexity $m = 4$

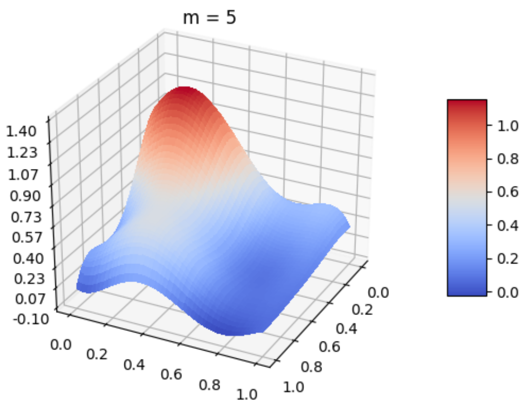

**FIG. 21:** Model fit using OLS with complexity $m = 5$