Comparison of Methods of Linear Regression (FINAL VERSION WILL BE ADDED BEFORE WEDNESDAY 3PM)

M.S Ingstad & F.L Nilsen
(Dated: October 8, 2019)

We studied polynomial fitting as a use of Linear Regression to compare the Ordinary Least Squares, Ridge, and Lasso methods. With k-fold cross validation and resampling, we used the mean square error and R2 score to evaluate their performance. We found that the OLS method gave the best model of the Franke function if we used enough data points and not too much noise. Otherwise the Ridge and LASSO methods performed slightly better. Using terrain data from Stavanger, Norway, we found that OLS best approximated the area, giving an average root mean square error of 10m for 50^{th} degree polynomials.

I. INTRODUCTION

Linear algebra allows the use of matrix inversion to solve sets of equations. In fact, with a broader perspective, any relation between some independent variables x_i and dependent variables y_i can be viewed as such a set of equations. The equations may not have exact solutions, and random noise in the data increases the difficulty, but an approximate solution can still be found using methods from linear algebra. Thus, by collecting real life data and assigning it to different input parameters and connecting it to some output parameters as a set of equations, one can quantify their relation. Using this result, new input data can be used to predict how the corresponding output data would be. As computing power has risen in the last decades, various methods have been developed to best make these predictions, where applications range from targeted ads to self-driving cars.

In this report, we will consider the method of linear regression, which is closely related to the interpretation of solving a set of equations. In particular we will compare the performance of the ordinary least squares, Ridge, and Lasso methods. By doing this, we aim to gain understanding of which of the different methods work best in different cases, varying noise and data points.

First, we fit a real multivariate function $f : \mathbb{R}^2 \to \mathbb{R}$ to a polynomial containing the two independent variables. We use the k-fold cross validation method to split the data into multiple sets of training and test data. After fitting on the training data, we use the mean squared error and the R2 function as to compare the performance of the methods using multiple values of the hyperparameter λ for the last two. To further explain the differences of the methods, we also use the bootstrap method as an alternative method to resample the data, and consider the biases and variances of the predictions.

To explore a more real life application, we also use a set of terrain data from Stavanger, Norway. Here, the independent variable is the height, and the dependent variables become the x- and y-coordinates. We again analyze the data using polynomials of different degrees.

II. THEORY

To best fit a polynomial of degree p to a set of data points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ of size n one must solve a set of equations given by

$$\begin{aligned}
\beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \cdots + \beta_p x_1^p &= y_1 \\
\beta_0 + \beta_1 x_2 + \beta_2 x_2^2 + \cdots + \beta_p x_2^p &= y_2 \\
&\vdots \\
\beta_0 + \beta_1 x_n + \beta_2 x_n^2 + \cdots + \beta_p x_n^p &= y_n.
\end{aligned} \tag{1}$$

This can then be rewritten to the form

$$\mathbf{X}\hat{\beta} = \hat{y}, \tag{2}$$

where $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\hat{\beta} \in \mathbb{R}^p$ and $\hat{y} \in \mathbb{R}^n$. It is not given that the equations in (1) are consistent, meaning that an analytical solution does not exist. Instead one must define a cost function, $C(\hat{\beta})$. Then, the $\hat{\beta}$ for which $C(\hat{\beta})$ is minimized is used as the best fit for the data set. How this cost function is designed will vary, depending on the method of regression that is used.

Ordinary least squares (OLS)

In OLS we have the cost function defined as the Mean Squared Error (MSE), given by

$$C(\mathbf{X}, \hat{\beta}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \frac{1}{n} \left\| \hat{y} - \mathbf{X}\hat{\beta} \right\|_2^2, \tag{3}$$

Where $\tilde{y}_i$ are elements of $\hat{\tilde{y}}$ given by $\mathbf{X}\hat{\beta}$. To find the optimal values of $\hat{\beta}$ one need to differentiate this function with regards to β, and set it to 0. The solution then becomes [2, p.45]

$$\hat{\beta}^{\text{OLS}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \hat{y}. \tag{4}$$

Another way to solve this is by using the pseudo inverse of the matrix $\mathbf{X}$, denoted $\mathbf{X}^+$. This matrix has the property that it is the best approximate solution to the

equation $\mathbf{AX} = \mathbf{B}$ [3]. "Best approximate solution" is then defined as the solution that minimizes $\|\mathbf{AX} - \mathbf{B}\|_2$. This is the same as minimizing the cost function for OLS (3), as $\left\|\hat{\beta}\right\|_2^2$ is minimized when $\left\|\hat{\beta}\right\|_2$ is minimized. We then get the solution for the OLS minimization problem as

$$\hat{\beta}^{\text{OLS}} = \mathbf{X}^{+}\hat{y} \tag{5}$$

The variance of β-value i given by OLS is [2, p. 47]

$$\text{var}(\beta_i) = \left(\mathbf{X}^T\mathbf{X}\right)_{ii}\sigma^2. \tag{6}$$

Ridge

For ridge regression, the cost function is defined as

$$C(\mathbf{X}, \hat{\beta}) = \left\|\hat{y} - \mathbf{X}\hat{\beta}\right\|_2^2 + \lambda\left\|\hat{\beta}\right\|_2^2. \tag{7}$$

$\tilde{y}_i$ is the same as in equation (3), but here a hyperparameter $\lambda \geq 0$ is introduced. We see that when λ is large, the last term dominates, so the cost function is minimized by making the coefficients β_i small. So the parameter λ shrinks the coefficients in $\hat{\beta}$. This means that the expectation value for the prediction is not the same as OLS, where the expectation value is that with the lowest MSE. We say that λ gives some bias to our estimation. For ridge we have an analytical solution for $\hat{\beta}$ given by [2, p. 64]

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

For each coefficient the variance is given by [4, p. 12]

$$\text{var}\left(\beta_i\right) = \sigma^2\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{X}\left[\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\right]^T \tag{9}$$

LASSO

The last method for linear regression we will look at is the Least Absolute Shrinkage and Selection Operator (LASSO)-method. We now define the cost function as

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

This minimization problem has no closed form solution, so in order to optimize $\hat{\beta}$ one need to use some numerical approach, like gradient descent. Like ridge, LASSO also adds some bias to the estimation.

Bias-Variance tradeoff

When modeling, we aim to approximate the test data with a minimal MSE. Although increasing the complexity of the model makes the model more accurate in terms of lowering the error when approximating the training data, higher complexity may also cause overfitting which increases the error with respect to the test data. This two-sided effect of increasing a models complexity can be illustrated through rewriting the expression for the MSE (3)

$$\text{MSE} = \frac{1}{n}\sum_i\left(f_i - \mathbb{E}[\hat{\tilde{y}}]\right)^2 + \frac{1}{n}\sum_i\left(\tilde{y}_i - \mathbb{E}[\hat{\tilde{y}}]\right)^2 + \sigma^2. \tag{11}$$

This relation is derived in Appendix A. Here the first term is called the bias. We see this is the squared distance between the expected value of the model, and the the true function. As we increase our model complexity this term should get smaller as we allow our model more freedom in approximating the actual function. The second term in equation (11) is known as the variance. This is here the squared distance between the estimated values, and the expectation values for the estimates. This term should become larger as complexity increases. This comes from that we have a finite number of data points. The fit will then depend on exactly which data set we have. This matters less for low complexity, since the model has fewer "degrees of freedom" to model the data. As we increase the complexity however, the model will be heavily dependent on the data points as the extra freedom is used to model noise, so the variance increases. We see that a too high model complexity makes the model too dependant on the specific data one have so the true MSE gets large. This is called over-fitting. On the other hand, if one makes the model too simple the model will not be able to represent the actual function, which again gives a large MSE. This is called under-fitting. One then gets a trade-off, where one have to find the "correct" model complexity somewhere in between, this trade-off is called the bias-variance tradeoff.

III. METHOD

We aim to model 3 sets of corresponding data as if each data point represented one dependent variable as a function of two independent variables $f : \mathbb{R}^2 \to \mathbb{R}$. In particular, we use the real multivariate Franke function (13) and height data from Stavanger, Norway. Our model combines the values along the x- and y-axis into an n'th order 2d polynomial with all its cross terms. For each term, values evaluated at the different data points make up the columns of a design matrix. This will be used to solve (2), where y is the corresponding data from dependent variable. The complexity of the model now depends on the degree of the polynomial used for fitting, and we

use this to compare how the different methods of linear regression compare on models of varying complexity. We also investigate the dependence on noise and amount of data points used, and for the Ridge and Lasso methods that optimizes the cost functions (7) and (10), we also consider the dependence on the hyperparameter λ. For Ridge in particular, we make a search in 2 dimensions of λ and complexity to find the optimal model, and compare its dependence on noise and the number of data points.

Numerical solutions

To solve the OLS minimization problem we will mainly use the method given by equation (5). This is because calculating the inverse of the matrix $\mathbf{X}^T\mathbf{X}$ can be time consuming, and can lead to numerical errors. To calculate the pseudo-inverse we will use the numpy function `numpy.linalg.pinv`, which uses a singular value decomposition to find the pseudo inverse. For ridge regression we will use equation (8) to find the β-coefficients. To minimize problems with numerical stability we will use singular value decomposition (SVD) to invert the matrix $\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I}$. We then use that the inverse of a matrix $\mathbf{A}$ will be given by

$$\mathbf{A}^{-1} = \left(\mathbf{U}\mathbf{D}\mathbf{V}^T\right)^{-1} = (\mathbf{V}^T)^{-1}\mathbf{D}^{-1}\mathbf{U}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T, \tag{12}$$

where $\mathbf{U}$, $\mathbf{D}$ and $\mathbf{V}$ are given by the singular value decomposition. To then find the inverse of matrix $\mathbf{D}$, one only needs to take the reciprocal of all non-zero diagonal elements, since $\mathbf{D}$ is a diagonal matrix. To find the SVD we will use the numpy function `numpy.linalg.svd`.

For the LASSO method, there is no closed form solution, so we have not written our own code to find $\hat{\beta}^{\text{LASSO}}$. Instead we have used the `sklearn.linear_model.Lasso` class in scikit-learn. To solve this problem scikit-learn uses an iterative solution, where we can set a tolerance and a maximum number of iterations. This method uses a coordinate descent method, so it updates one coefficient at a time, by calculating the partial derivative of the cost function with respect to the coefficient, and then changes the coefficient with the derivative times some step size, or learning rate. This learning rate is chosen automatically by scikit-learn.

To test our regression methods, we have designed a small test case with 5 data points, where we know the analytical solution for OLS for a 1st order polynomial. We then have unit test for all regression methods, where we check if they find the solution within a tolerance of 10^{-14}. For ridge and LASSO we have used the same test case, but set λ to be very small. To test our matrix inversion using the singular value decomposition we generate a 10×10 matrix where the elements are randomly distributed by a normal distribution with variance 1 and mean 0. We then calculate the inverse with the SVD-inversion function, and test weather or not the elements of the inverse matrix times the original matrix correspond to that of the unit matrix within a tolerance of 10^{-10}.

Datasets

To test the different methods for linear regression, we will use two different datasets. First we look at the Franke function, defined as

$$\begin{aligned} f(x,y) &= \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ &+ \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ &- \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{13}$$

We will use equation (13) evaluated with, and without random noise added. The random noise will be uniformly distributed, with $\mu = 0$. The standard deviation of the noise will be set as a fraction of the difference between the maximum and the minimum value of the function. This gives

$$\sigma = \hat{\sigma}(f_{max} - f_{min}), \tag{14}$$

where $\hat{\sigma}$ is the fracion we use to set the noise. To evaluate the Franke function function we will pick random points with $x, y \in [0, 1]$, where x and y have a uniform probability distribution. We also vary the amount of points to further evaluate the different methods.

We will also look at a dataset consisting of height data from a region close to Stavanger in Norway. This dataset consists of 3601×1801 data points, so 6485401 data points in total. This is a quite large dataset, so we will also look at a smaller portion of the set, consisting of x indices between 0 and 300, with y indices between 300 and 500, giving a total of 60 000 data points.

Since we now are looking at a two dimensional surface, we cannot construct the $\mathbf{X}$-matrix in the way we got it from equation (1). To construct the martix $\mathbf{X}$ we then use the numpy function `numpy.polynomial.polynomial.polyvander2d`. This function constructs the matrix for a polynomial of degree (i, j), so one can have different polynomial degree for the two dimensions. We will only be using the same degree for each dimension, and will then refer to the degree of the fit with a single number.

Normalization

When applying the different methods of linear regression, we need to find a vector $\hat{\beta}$ such that equation (2) is

best satisfied. The matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ can have elements that are of completely different order of magnitude, which can give numerical errors. To avoid this, we will normalize the matrix by multiplying with a diagonal matrix, which we will call $\mathbf{D} \in \mathbb{R}^{p\times p}$. Then we will also need to multiply with $\mathbf{D}^{-1}$, to avoid changing the expression. Equation (2) then becomes

$$(\mathbf{XD})\left(\mathbf{D}^{-1}\hat{\beta}\right) = \hat{y}. \tag{15}$$

Since $\mathbf{D}$ is a diagonal matrix, a diagonal element d'_{ii} of the inverse matrix $\mathbf{D}^{-1}$ is found by simply taking $1/d_{ii}$, where d_{ii} is an element in $\mathbf{D}$. All off-diagonal elements in $\mathbf{D}^{-1}$ are 0 ($\mathbf{D}^{-1}$ is also diagonal). To find element d_{ii} we use the maximal element of column i in $\mathbf{X}$, and set $d_{ii} = \left[\max_k (\mathbf{X}_{ki})\right]^{-1}$. Equation (15) then becomes the "new" linear regression problem, where we use $\mathbf{XD}$ as the $\mathbf{X}$-matrix, in order to find $\mathbf{D}^{-1}\hat{\beta}$. Applying these normalized β values on a set of test data X normalized in the same way as the design matrix used for training returns an un-normalized prediction $\hat{\tilde{y}}$.

Result analysis

To analyse the results given by the different methods, we will use the mean squared error and the R^2 function. The MSE (3) is the cost function optimized by the ordinary least squares method, while the R^2 score is given by:

$$R^2 = 1 - \frac{\sum_i (f_i - \tilde{y_i}])^2}{\sum_i (f_i - \mathbb{E}[f_i])^2} \tag{16}$$

To best evaluate the data we will split the datasets into two parts, training data and testing data. We use the training data to fit our model, and then the test data to calculate the MSE and the R^2 values. This is called the train-test split, and is used to avoid overfitting the model. To obtain an average value of MSE and R^2, we also resample the data, doing the train-test split iteratively. One way of doing this is to split the dataset into k parts for different values of k. For each value of k, one can then pick out one part to act as the test data and use the rest as training data. With this split one calculates the value for R^2 and/or MSE. Then repeat the calculation, using another part as the test data, until all parts have been used as test data. One then uses the average of the R^2 and/or MSE for all parts and all desired values of k as a measure for how good the model fits. This is known as k-fold cross-validation, and we used a value of $k = 5$ in our evaluations.

Another way to resample the data and compute the average of the errors is bootstrapping. If the dataset has n data points, one chooses n data points randomly **with** replacement, and use this dataset to calculate the value you are interested in. This process can be repeated as many times as needed. The variance of the estimates will then approach the real variance of the value one are analysing. We use this combined with the train-test split, to calculate the bias and variance of our models. We do this by first splitting the data into 80% training data and 20% testing data, and then use the bootstrap method on the training data to find several predictions for the test data. We then use all predictions compared with the actual testing data to calculate the MSE, bias and variance of the model. In particular, we calculate these values with fits on the Franke function by OLS, Ridge and Lasso, with varying values of the hyperparameter λ and the noise of the data.

To test if we actually get overfitting, we will also look at the MSE and R^2 values for the training data. Here we expect to get significantly lower errors when we increase the complexity, as this should increase the variance when comparing to the test data, but comparing to the training data there should not be any variance.

IV. RESULTS

Franke Function

Generating data for the Franke-Function, we first made a contour of the data in Figure 1, illustrating its shape and thereby potential challenges of fitting with a polynomial. It appears to have two maxima, one minima, and two sadle points in the area we generate the data.

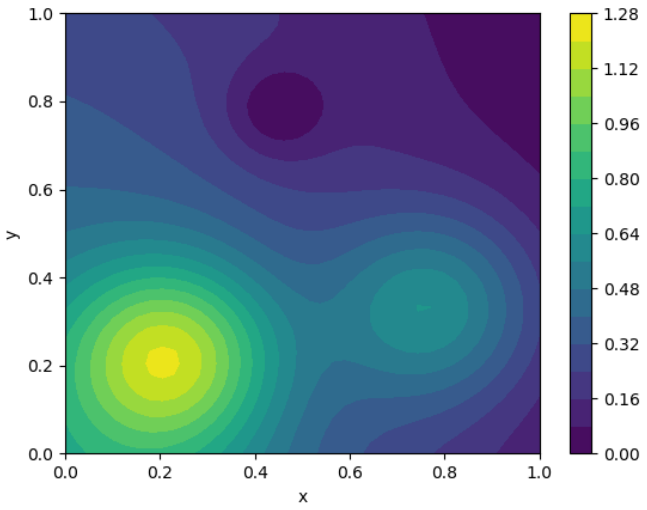


Figure 1. Contour of the Franke function

We first tried to fit the Franke function with a 2nd order polynomial using 500 data points, and $\hat{\sigma} = 0.01$. The resulting β coefficients are written in Table I. There we also calculate the standard deviation from equation (6). Doing the same for 60 000 data points, the results

Table I. Results for OLS on the Franke function using 500 data points and $\hat{\sigma} = 0.01$ with a 2nd order polynomial. $\sigma(\beta)_{\mathrm{a}}$ is the standard deviation from the analytical expression, while $\sigma(\beta)_{\mathrm{b}}$ is the standard deviation from the Bootstrap method. We also calculate the relative difference (Rel diff), between the two values. The 95% confidence interval for each β is given by $\beta \pm 1.96\bar{\sigma}$

term	β	$\sigma(\beta)_{\mathrm{a}}$	$\sigma(\beta)_{\mathrm{b}}$	Rel diff	$1.96\bar{\sigma}$
1	1.55	0.05	0.09	0.62	0.14
y	-2.56	0.23	0.29	0.27	0.51
y^2	1.30	0.22	0.23	0.08	0.44
x	-2.16	0.23	0.36	0.54	0.58
xy	6.24	1.00	1.29	0.29	2.24
xy^2	-4.91	0.91	1.04	0.14	1.92
x^2	0.76	0.23	0.32	0.41	0.53
x^2y	-3.10	0.94	1.15	0.22	2.05
x^2y^2	2.81	0.86	0.95	0.10	1.77

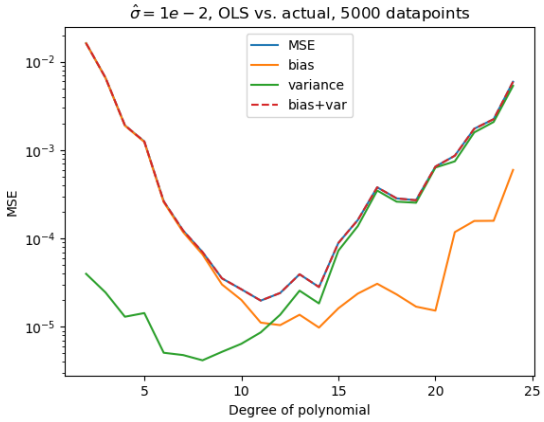
Table II. Results for OLS on the Franke function using 60 000 data points and $\hat{\sigma} = 0.01$ with a 2nd order polynomial

term	β	$\sigma(\beta)_{\mathrm{a}}$	$\sigma(\beta)_{\mathrm{b}}$	Rel diff	$1.96\bar{\sigma}$
1	1.27	0.00	0.01	1.13	0.01
y	-1.44	0.02	0.03	0.80	0.05
y^2	0.38	0.02	0.03	0.57	0.05
x	-1.10	0.02	0.04	0.88	0.05
xy	2.05	0.09	0.14	0.62	0.22
xy^2	-1.55	0.08	0.12	0.43	0.20
x^2	-0.09	0.02	0.03	0.67	0.05
x^2y	0.31	0.08	0.12	0.46	0.20
x^2y^2	0.09	0.08	0.10	0.29	0.18

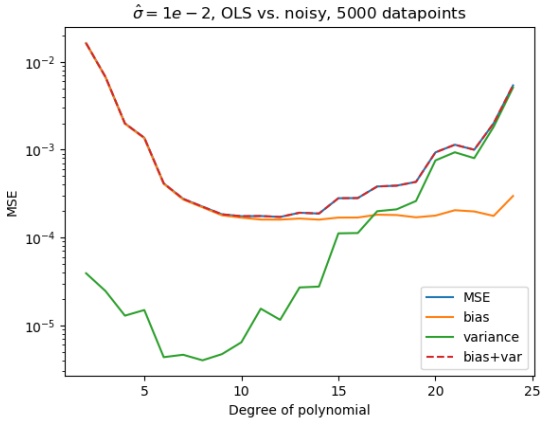
are in Table II. We see that the 95% confidence intervals for β are smaller when using more data points. We note that the values of β and thus the standard deviations are normalized by (15). They could be converted back by multiplying with their respective diagonal elements of $\mathbf{D}$.

Bias-Variance Tradeoff

Further applying the Ordinary Least Squares method to the data, Figure 2 shows how the MSE along with the bias and variance varies with complexity of the model. We evaluated these values both compared to the actual data before added noise in 2(a), as well as the noisy data in 2(b). We see from Figure 2 that for OLS, the MSE has a minimum for a polynomial degree of 11, which also appears to be when the bias and variance cross. This minimum appears to be lower in 2(a) than in 2(b) as the bias flattens out in 2(b) but goes further down in 2(a). This results in a lower minimal MSE for OLS when comparing to the actual data, as opposed to comparing to the



(a)Bias variance compare to actual data

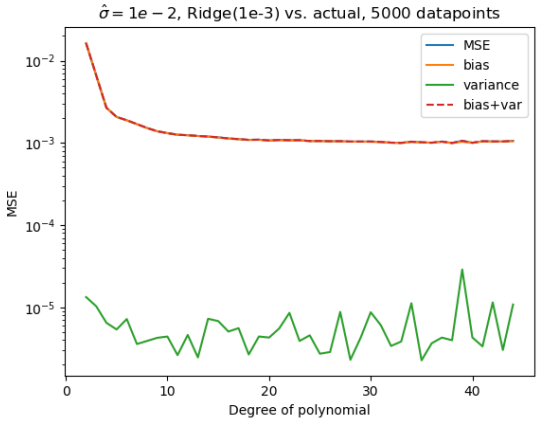


(b)Bias variance compared to noisy data

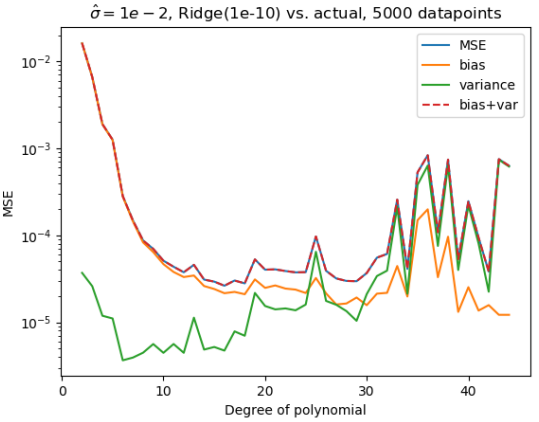
Figure 2. Illustrates the bias variance trade-off, when comparing to actual data and noisy data. This data was created using OLS3, with 5000 data points of the Franke Function with a noise of 1 percent

noisy data the model is based of. For lower polynomial degrees, the variance is generally lower, and for higher degrees, it becomes higher and almost the source of all MSE relative to the bias. The bias starts out high on the other hand, making up most of the MSE at low polynomial degrees. It then becomes lower up to a degree of about 15, before increasing again, but still making up a much smaller portion of the MSE than the variance.

We also the found the MSE along with the bias and variance of when applying the Ridge method with values of $\lambda = 10^{-3}$ and $\lambda = 10^{-10}$ in Figure 3. Both are modeled on training data with added noise of $\hat{\sigma} = 1e-2$, and compared to test data with no added noise. In particular, the k-fold error with k=2,3,4,5 is used to evaluate MSE, bias and variance. From Figure 3(a) wee see that the variance remains constantly at a value between about 10^{-6} and 10^{-5}, while the MSE follows the bias tightly decreasing at a slower rate for higher polynomial degrees, but generally never increasing. Although there is a lot of

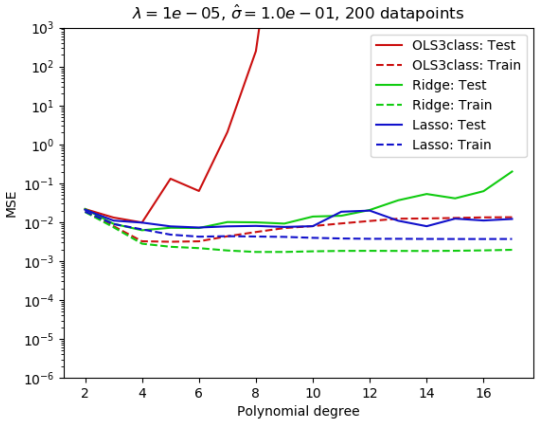


(a)Bias variance for $\lambda = 10^{-3}$

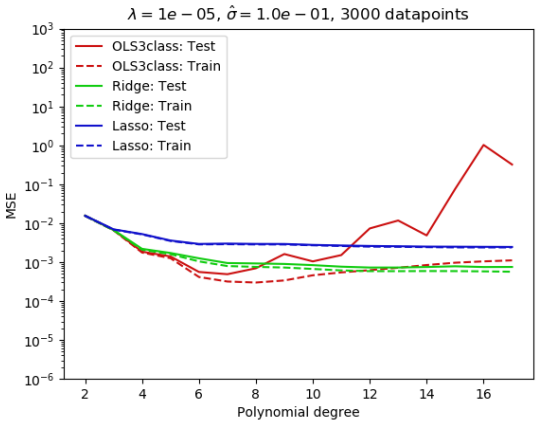


(b)Bias variance for $\lambda = 10^{-10}$

Figure 3. Bias variance with 5000 datapoints, with values of $\lambda = 1e-3, 1e-10$ on Ridge. The values were evaluated compared to actual data instead of the noisy data. The first plot was generated with 2 resamples.



(a)Evaluated on 200 data points of the Franke function



(b)Evaluated on 3000 data points of the Franke function

Figure 4. OLS, Ridge, and Lasso are colored in red, green and blue respectively. The solid lines show the MSE evaluated on the test data using the k-fold error, while the dashed lines show the MSE evaluated on the training data. Ridge and Lasso use $\lambda = 10^{-5}$, and the data has a noise of $\hat{\sigma} = 10^{-1}$

noise in the data from Figure 3(b), we see that the variance starts to increase slightly at a high enough degree of the polynomial, making the MSE have a minimum point before increasing again similar to OLS. This minimum point between 10^{-5} and 10^{-5}, is more than one order of 10 lower than 3(a) with a minimum of about 10^{-3}

Also including the Lasso method, we compared the general relation between the MSE when evaluated on the testing data, compared to when it was evaluated on the training data in Figure 4. In 4(a) we can see that the MSE quickly becomes very large, while the Ridge and Lasso methods with a hyperparameter of $\lambda = 10^{-5}$ are more stable. The Lasso methods appears to remain stable for all 17 polynomial degrees tested for, while the MSE of Ridge starts to increase around a degree of 10.

In Figure 4(b), there are more datapoints, and we see that the MSE gives lower MSE for a larger range of polynomials, this time having a minimum noticeably lower than the Lasso method, and about the same as the Ridge method. The test MSE of the Ridge method is nearly the same as its MSE evaluated on the training data, while for Lasso, the two are on top of each other.

Further investigating the Ridge method, Figure 5 shows two seaborn heatmaps showing the R^2 score as a function of the hyperparameter λ and the polynomial degree. In the first plot, the fit on noisy training data has an error evaluated on test data with no noise, while in the second plot, the fit is evaluated on the used training data. We see that in the first case, the optimal error is with a middle value of λ and polynomial degree, while in the second case, the best fit is for the lowest λ and highest polynomial degree.

Generating similar data with varying noise and amount of data points, III collects the found optimal value of λ and polynomial degree, with a corresponding R^2 score and MSE.

Lastly for the Franke Function, Figure 6 shows again

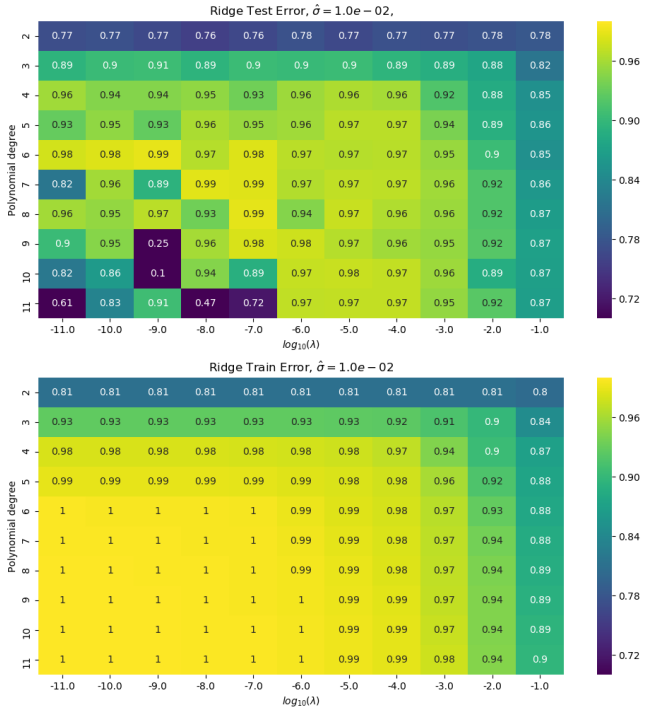


Figure 5. Generated using 200 data points of the Franke Function. Shows R2 score for varying polynomial degrees and values of λ for Ridge evaluated on the Franke function with $\hat{\sigma} = 0.01$ noise. $\hat{\sigma}$ refers to the standard deviation of the Gaussian noise as a fraction of the distance between maximum and minimum values. Optimal R^2 and MSE can be found in Table III

Table III. Results for ridge regression on the Franke function,where λ optis the value for λ that gives the smallest MSE, and p is the corresponding optimal polynomial degree

$\log_{10}\hat{\sigma}$	# data points	R^2	MSE	p_{opt}	$\log_{10}\lambda_{\text{opt}}$
-3.0	100	0.9156	9.061e-02	5	-4.0
-3.0	200	0.9890	3.161e-02	6	-9.0
-3.0	400	0.9973	1.954e-02	10	-10.0
-3.0	1000	0.9990	8.742e-03	11	-11.0
-2.0	100	0.9184	8.995e-02	5	-4.0
-2.0	200	0.9884	3.344e-02	6	-9.0
-2.0	400	0.9966	1.884e-02	7	-11.0
-2.0	1000	0.9990	1.321e-02	10	-11.0
-1.0	100	0.8869	1.432e-01	4	-5.0
-1.0	200	0.9487	6.866e-02	5	-4.0
-1.0	400	0.9683	5.838e-02	6	-6.0
-1.0	1000	0.9832	3.760e-02	6	-7.0

the R^2 score as a function of λ and polynomial degree, but with more values tested for.

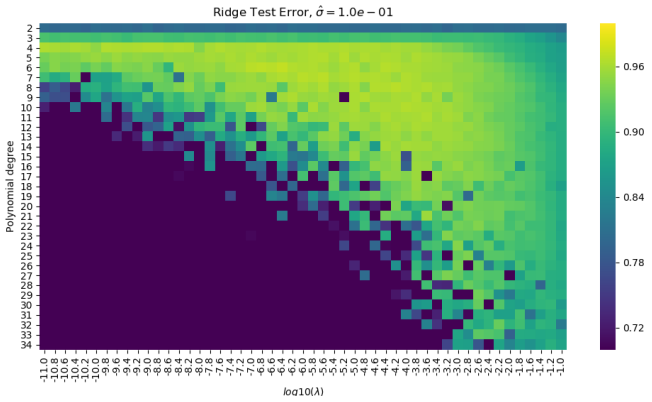


Figure 6. R2 score for varying polynomial degrees and values of lambda for Ridge evaluted on the Franke function with $\hat{\sigma} = 10\%$ and 400 data points used. The optimal polynomial degree and value of λ found was degree 6 and $\log_{10}(\lambda) = -6.4$ with $R^2 = 0.9729$

Terrain data

The slice of terrain data we attempt to model is shown in Figure 10(a). The values range from a minimum of 1083m and a maximum of 1483m. Figure 7 shows the MSE of the fit with different values of the polynomial degree. We see that with increased complexity, the MSE decreases, but with a decreasing rate. It reaches a minimum MSE of about 200, giving an average deviation of $\sqrt{200\,\text{m}^2} \approx 14\,\text{m}$ from the actual terrain data.

Figure 8 shows the R^2 score of the Ridge method as a function of the hyperparameter λ and the polyomial degree. Table IV and V shows the best results for different amount of data points, both with our own Ridge function, and the Ridge function from scikit-learn [1]. The RMSE is the root of the MSE, which corresponds to an interpretation of an average error in metres. From the tables, we see that the error generally becomes less for more data points, and that polynomial degrees increases, and λ value decreases as this happens.

Figure 9 shows the R^2 score of the LASSO method as a function of the hyperparameter λ and the polynomial degree. The results gotten may be slightly inaccurate as the Ridge method never reached convergence with our requested tolerance. Due to the high processing time, we were only able to set a maximum of 20 000 iterations.

With the optimal value of p and λ suggested in Table IV, Figure 10 shows a comparison between the actual data, and the approximated data applied by training Ridge on 12 000 data points. Figure 11 further shows similar comparison using a fit by a 50^{th} order polynomial using the OLS method.

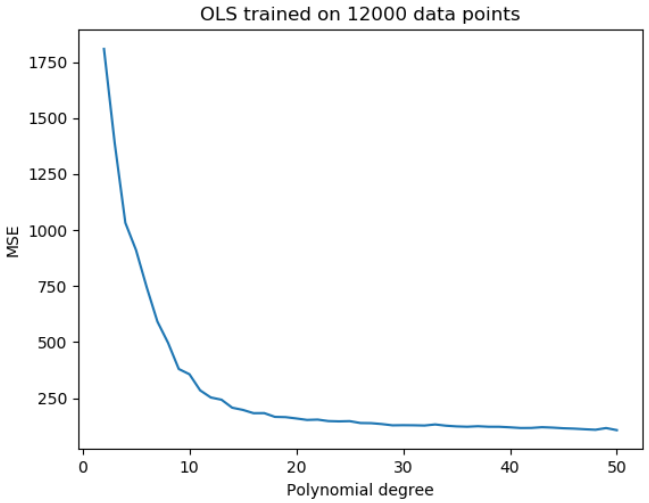


Figure 7. OLS fitted on 20% of the indiced terrain, with MSE evaluated on all of the terrain as a function of polynomial degree.

Table IV. Optimal Ridge parameters for indiced terrain data

# data points	R^2	RMSE	p_{opt}	$\log_{10}\lambda_{\text{opt}}$
120	0.8049	4.320e+01	4	-9.0
180	0.8508	3.881e+01	11	-7.0
300	0.8738	5.584e+01	18	-5.0
600	0.9090	3.706e+01	15	-10.0
1800	0.9263	2.369e+01	24	-10.0
6000	0.9272	2.351e+01	24	-10.0
12000	0.9293	2.279e+01	24	-9.0

Table V. Same but with Ridgeskl

# data points	R^2	RMSE	p_{opt}	$\log_{10}\lambda_{\text{opt}}$
120	0.8049	4.320e+01	4	-9.0
180	0.8508	3.881e+01	11	-7.0
300	0.8738	5.584e+01	18	-5.0
600	0.9087	3.702e+01	15	-10.0
1800	0.9300	2.261e+01	21	-11.0
6000	0.9465	1.968e+01	24	-11.0
12000	0.9472	1.973e+01	22	-11.0

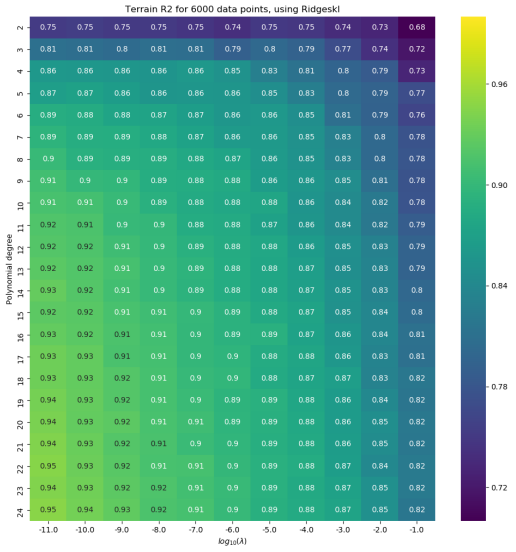


Figure 8. R2 score for varying polynomial degrees and values of lambda for Ridge evaluated on the terrain data with 20% of data points used. The optimal polynomial degree found was 24 which corresponds to the highest degree tested for. The corresponding optimal value of λ found was $\log_{10}(\lambda) = -11.0$ with $R^2 = 0.9472$ and a root mean

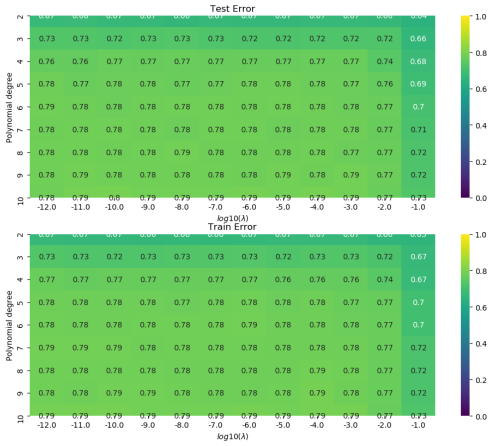
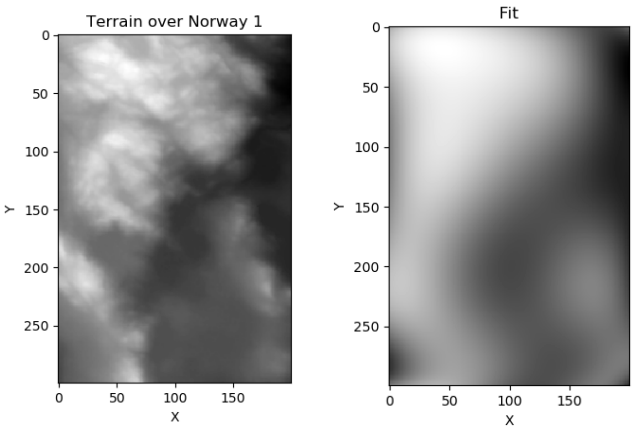


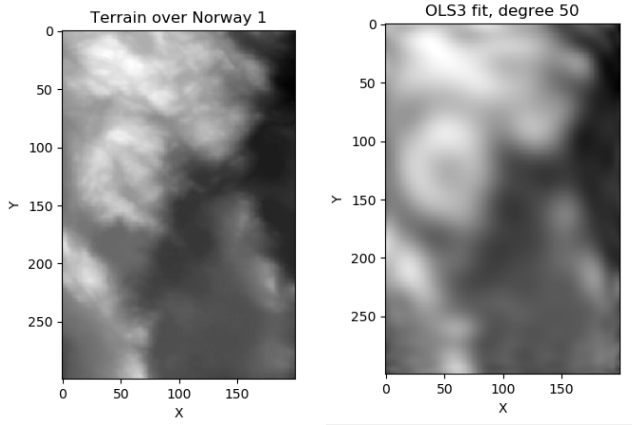
Figure 9. R2 score for varying polynomial degrees and values of lambda for Ridge evaluated on 60000 points of the terrain data.



(a)Actual terrain data

(b)Approximation made with ridge by the optimal value from Table IV with 12000 data points.

Figure 10. Actual terrain data, vs approximated terrain data



(a)Actual terrain data

(b)Approximation made by OLS with a 50^{th} degree polynomial

Figure 11. Actual terrain data, vs approximated terrain data

V. DISCUSSION

Franke function

Looking at the contour of the franke function in Figure 1, it appears to be relatively well suited to be modeled by a polynomial, but particularly the minimum at around $x = 0.5$ and $y = 0.8$ might be hard to model if the polynomial is not of high enough degree. The tables I and II show the standard deviation of the estimated beta parameters of such a low degree polynomial of the second order. There seems to be some difference between the theoretical values and the numerically estimated values we found, which may in part be because one would need an infinite amount of points to reach the theoretical values. Using their average, however, we can see a clear difference between the 500 data points used in table I and the 60 000 datapaoints used in table II. This seems logical, as for a low number of total data points, an even lower number might be necessary to describe a specific feature of the output. For instance, the earlier mentioned minimum, might contain about 10 points in total, and with only 10 points the randomness of the noise might be skewed in a certain direction. The expectation value gaussian noise only approaches the average μ as the amount of data points approaches ∞. When various features of the output is masked, the values of β best fitting the model will change based on the noise, in turn increasing their variance and standard deviation. This is essentially overfitting to noisy data, and the tables seem to confirm that this effect decreases with a higher number of data points.

Still using OLS, but now considering the variance of the prediction instead of the coefficients of β, we see from figure 2 that even with a relatively high amount of data points in 5000, the variance increases when the polynomial degree becomes too high. This seems to also illustrate overfitting, as higher degree polynomials allows for the model to accomodate for more points, thus increasing the chances of fitting to a masked feature as discussed above. Oppositely, some small collection of points with randomly similar noise, might be big enough for a high order polynomial design matrix to accommodate for, therefore increasing the variance when compared to other datasets with other random fluctuations. The plots also show the bias which seems to decrease with initial increase of polynomial degree, illustrating how a higher complexity allows for accommodating more features of the dependent variable data. We also note that the bias + variance is on top of the MSE as it should. Thus, for degrees¡10 when the bias is nearly on top of the MSE curve, this indicates that this is what contributes most to the MSE, and the variance has next to no impact. Similarly, for degrees > 15, the opposite happens, and now the variance is what contributes most, while the bias has next to no impact. There is an effect that we didn't foresee however, as the bias does not keep decreasing indefinitely with higher polynomial degree, instead it starts increasing at about a 20^{th} order polynomial. The reasons for this may be in part due to numerical instability, but also possibly that this order of polynomial is enough to fit all the major aspects of the Franke function in our given area, so adding higher order polynomials may just cause instability. After all, when fitting with polynomials, one must be aware that outside the fitting area, they will generally explode in value.

With regards to the difference between Figure 2(a) and Figure 2(b), where the first has an MSE evaluated when compared to actual test data, while the second has an MSE evaluated compared to noisy test data, we clearly see that the first case has a much lower MSE around the area where the bias stops decreasing, and the variance

starts increasing. We think that this is because when comparing to noisy data, there will always be a static MSE as a result of the noise present. Say we had the optimal fit in the form of the formula for the Franke function, it would still not hit all points perfectly, as they would have random noise added to them. When comparing to the actual data however, the prediction would then be exact. In essence, this means that OLS fit in a way outperforms the data itself, as the found coefficients more closely describes the actual function, than the noisy data it was modeled on. This way of mitigating noise is a strong advantage of linear regression. Further confirming this view of the two plots, is the fact thtat the bias flats out at around a 10^{th} order polynomial, and doesn't change until it starts increasing again at around an order of 25. As shown in equation (11), the bias represents the deviation of the data, and as discussed, this has an optimal value when comparing to noisy data. The variance on the other hand, remains mostly the same for the two plots, as this represents the predicted data compared to itself, which is modeled on noisy data in both cases.

We also considered the ridge method, investigating its dependence on both complexity as we did for OLS, as well as the hyperparameter λ. When we looked at the R^2 score in Figure 5 we found that while looking at the training data the best fit seemed to be a low λ and a high complexity, this changed when we compared with the test data. Here we see that increasing the complexity stops being beneficial after roughly a sixth order polynomial. This again demonstrates the bias-variance tradeoff, as increasing the complexity decreases the bias, which is visible in both datasets. When comparing to the training data however, the variance contribution to the error does not appear, as there is no variance of the prediction when compared to itself. So at high complexities, where we expect variance to cause most of the error, this error would only be visible on the test data. Figure 5 shows this, so the results here are as we expected. In particular, we note that low values of λ with the ridge method corresponds to the OLS method, as $\lambda = 0$ in the ridge cost function (7), gives the OLS cost function (3). From Figure 5 we see that for such low values of λ the test error becomes large at a lower complexity, corresponding to the increase in variance showed in Figure 2. Now considering the columns farther to the right with higher values of λ, we see that the result becomes better for larger complexities, which implies that increasing the value of λ in ridge, decreases the variance. This is what we would expect from theory.

Going to far to the right in the columns of Figure 5 seems to result in a worse R^2 score again. Since this is visible both on the training data, and on the test data, this seems to indicate that this error is not because of the variance, but instead mostly due to the bias increasing for the lowest values of λ in ridge. This is also what we would expect from theory, as adding elements that are too big to the diagonal of the $\mathbf{X}^T\mathbf{X}$ matrix, would lose some resolution from the actual data in the columns. Comparing Figure 3(a) to Figure 3(b) also seems to confirm the effect of the ridge method on the bias and variance. Here, higher λ clearly leads to a higher bias, but a stable variance for polynomials of up to the 40^{th} order. Lower λ decreases the bias greatly, but through the noise in Figure 3(b), we can still see that the variance eventually starts increasing at around a 35^{th} order, which causes the MSE to increase again.

Thus, we can conclude that the λ of the ridge method can be used to balance the bias-variance tradeoff, where in the case of Figure 5, the optimal value lies somewhere in the middle of the plot with an R^2 score of 0.99. When we look at the optimal values in Table III, we can see that in general, increasing the noise results in the optimal value of λ being shifted to higher values, while increasing the amount of data points does the opposite. In explicit, we see that at 1000 data points when $\log_{10}\hat{\sigma} = -3.0$ the best prediction is a 11^{th} order polynomial, when $\log_{10}\hat{\sigma} = -2.0$ it is a 10^{th} order polynomial and when $\log_{10}\hat{\sigma} = -1.0$ the best fit is a 6^{th} order polynomial. This trend fits well with the bias-variance tradeoff, as a higher value for $\hat{\sigma}$ necessarily gives a higher variance of the model, and to counter this one can choose a lower order polynomial with a higher bias. Also choosing a higher value of λ increases this effect. Oppositely, when increasing the amount of data points, the bias will decrease, and a higher order polynomial and a lower value of λ making ridge closer to OLS is possible.

We chose not to make as many plots with the Lasso method, as it uses an iterative method of finding the optimal solutions, and takes considerably longer time to compute, but in Figure 4 we plotted its test error vs. train error compared to both ridge and OLS. When comparing training error to test error, we would expect overfitting to appear as a divergence between the error on the test data, and the error on the training data. Figure 4(a) uses a small data set of 200 points, and here we see that this divergence is much bigger for the OLS method than ridge or LASSO. The divergence is visible already at a 3^{rd} order polynomial for all methods, but OLS has the clearest divergence, and starts overfitting considerably at around a 4^{th} order polynomial, while ridge and LASSO both keep a relatively stable difference between training and test data up to a 10^{th} order polynomial. After this, the test MSE of ridge starts increasing, while LASSO remains stable, where the fluctuations seem to be mostly due to the random nature of the data. The optimal MSE of the test data of the three methods seem to be either LASSO or ridge, showing that biased methods like ridge and LASSO can give a better estimate than OLS.

However, when we look at a larger dataset in Figure 4(b), we see a different result. Here, at the 8^{th} order polynomial, the result from OLS reaches a minimum value that is slightly better than the one from ridge. This is because the variance will decrease as we add more data points. This means the error in OLS for predic-

tions with high complexity will decrease, as the number of data points go up, meaning that methods like ridge and lasso will be relatively less effective when comparing with OLS. This could be because they introduce a bias, with the aim of reducing variance. But if the variance already is low, the main effect of the hyperparameter becomes to introduce some bias, which increases the error of the estimate. So when one has many data points compared to the complexity one wishes to model, OLS seems to work best as it minimizes the bias. However if one have few data points, and have a model with high complexity OLS seems to be more affected by a high variance than methods like ridge and LASSO.

Regarding the difference between ridge and LASSO, we see from Figure 4(a) that ridge gives a lower MSE than LASSO for lower complexity, like for 4^{th} order polynomials. However for much higher complexity, like 16^{th} order polynomials LASSO outperforms ridge. This seems to indicate that if one has a very complex model, but few data points, LASSO might be the better choice. This makes some sense, as the cost function for LASSO uses the l_1 norm, while ridge uses the l_2 norm. Since the l_1 norm is typically larger than the l_2 norm, LASSO adds a larger bias. This also means that the decrease in variance is larger, which is good for high complexity, few data points and large noise.

Terrain data

When we applied the methods on the terrain data, we see much of the same as when we used them on the Franke function. In Table IV we find that when looking at few data points, we get a worse estimate based on both the R^2 score and the MSE. This is as expected, and makes intuitive sense as more points means more detail of the terrain we try to fit. What is more interesting is that with about 100-300 data points, the optimal values for λ are higher than those for about 1000-10000 data points. We also see that with many data points, the optimal polynomial degree goes up. This is similar to the results we saw on the Franke function. Unlike the results from the Franke function, it is important to note that since there is next to no noise in the terrain data, this cannot be a contributing factor in increasing the variance. Instead it stems solely from the difference in selected data. As we use the k-fold error, many different training and test sets is selected. Thus, if there aren't enough data points, not all training sets will include the same main features of the data, resulting in increased variance. To counter this, the optimal "strategy" with ridge regression seems to be to select a low complexity and a relatively high λ if we read of from Table IV.

This gives a higher bias, but then also a lower variance, which turn out to be more important. As we saw with the Franke function, more data points results in the variance becoming less important, so one can pick a higher complexity and a smaller λ. This is clearly shown in Figure 8, where we see that generally with 6000 data points, with the values we tested, the best results are for high complexity and small λs. It seems that the closer the method is to OLS for a given complexity, the better the result is.

Regarding the differences between Table IV and V, the first is based on data from our ridge method using the singular value decomposition, while the second is based on the ridge function in scikit-learn [1]. We see that for $\#datapoints \leq 600$, the results are mostly similar, but for more data points, our function gives worse results, and higher optimal values of λ. Unlike in Figure 8 which is based on the scikit-learn function, a corresponding figure ("Ridge_6000.png" in B) based on our ridge function gives a decreasing R^2 score in the leftmost column. We think this happens due to problems with the singular value decomposition method, where scikit-learn uses something else. Figure 7 seems to confirm that our ridge method is numerically unstable for too many data points, as this is trained on 12 000 data points, up to polynomial degrees of 50, showing no signs of increasing error. As argued earlier, OLS corresponds to Ridge with low values of λ, so if the leftmost column of Figure 8 were to be worse than columns to the right on an analytical basis, this would not be consistent with Figure 7. The reason OLS works better than ridge at low values of λ is that instead of the singular value decomposition, we used the pseudo inverse matrix (5) [3]

Assuming that our OLS method is indeed numerically stable, we can see from Figure 7 that when the amount of data points is large enough, it is hard to get to a high enough complexity where the variance increases enough for overfitting to occur, as there are no signs of this for 50^{th} degree polynomials. This can also be seen for the 24^{th} order polynomials in Figure 8, albeit we note that in V 12000 data points yields a 22^{nd} order polynomial as the optimal one. Since OLS works well improves until a degree of 50 for the same amount of data points, this is not as expected. Random chance could of course be an explanation, but another could be that also the ridge method in scikit-learn shows signs of numerical instability.

As for the LASSO method, it seems to give more stable, but less optimal results than our ridge method as shown in Figure 9. From Figure we see that the R^2 score remains relatively homogeneous, with the best fit at around 0.79. This is lower than that of ridge, which gives an R^2 score of about 0.9 at the same polynomial degree of 10. This result agrees well with our previous discussions, where we argued that LASSO should be used when there is either a lot of noise or few data points, neither of which is the case here. The low scores are however also in part due to method taking too long to converge on the best solution.

With this in mind, we used the optimal value of λ for the Ridge method from table IV, along with the OLS method with a polynomial degree of 50, to plot the cor-

responding terrain fits in Figure 10 and Figure 11. The recreation of the height map based on the ridge fit in 10 does not contain the same resolution as 10(a), but regarding the high points in white, and low points in dark, it seems to fit the largest patterns. With 50^{th} order OLS however, the results look much closer to the actual terrain. As discussed, this is due to the high complexity, and in this case no detrimental adding of bias as done by ridge.

If the terrain were to be modeled by the same methods, but perhaps using something other than n'th degree 2d polynomials, the results may be a bit different than we got here. In general, we would say that modeling terrain with polynomials is not very physical, as polynomials explode in value outside the range of the model, which obviously does not happen with the terrain. These models could therefore in no way be used to predict how the terrain we did not model on would appear.

VI. CONCLUSION

From applying the OLS, ridge and LASSO methods to fit 2d-polynomials to the Franke function, we found that their performance when compared to each other depended on the amount of data points and amount of noise inversely. For the various methods, this determined which order of polynomial gave the optimal MSE of the model applied on the test data, where the polynomial order in general terms corresponds to the complexity of the model. We were able to explain this relation studying how the bias was generally more important for lower complexities, and the variance was generally more important for higher complexities. Comparing these effects on the different methods, we were able to conclude that OLS generally works best if there is little noise and exact data to approximate. If there is a lack of data points, or too much noise however, OLS can overfit from too much variance. Here, Ridge and LASSO become more useful, as introducing the hyperparameter λ to the cost function increases the bias to decrease the variance, in effect manipulating the bias-variance tradeoff and potentially bringing the MSE down. We also found in cases where the variance is very high LASSO can give slightly better results than ridge for the same value of λ. Ridge however, preformed noticeably better between the cases where LASSO or OLS gave the best results. LASSO was also very time consuming compared to the other methods, as it does not have a closed form solution.

On the terrain data, the available number of data points and the lack of significant noise led to the OLS method being our best approximation. We did however encounter problems with numerical instability of the Ridge method, which may have affected its results. Further investigation on this could yield a more certain conclusion on its performance. Although the case of fitting with polynomials illustrated many of the aspects and differences between OLS, ridge and LASSO, we lastly note note that alternative models might yield somewhat different results, and may be more logical to use especially with regards to the terrain data.

[1] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

[2] Hastie, T., Tibshirani, R., and Friedman, J. (2017). *The Elements of Statistical Learning Data Mining, Inference, and Prediction (12th printing)*. Springer Series in Statistics. Springer New York, New York, NY, second edition.

[3] Penrose, R. (1956). On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52(1):17–19.

[4] van Wieringen, W. N. (2015). Lecture notes on ridge regression.

Appendix A: Rewriting MSE

The cost function (3) can first be rewritten as the expectation value for the squared error

$$\mathrm{MSE}(\mathbf{X},\hat{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}\left[\left(\hat{y}-\hat{\tilde{y}}\right)^2\right], \quad \text{(A1)}$$

where $\hat{\tilde{y}}$ is the y-values predicted by the matrix $\mathbf{X}$, and $\tilde{y}_i$ are the elements of $\hat{\tilde{y}}$. If we now assume the data points in $\hat{y}$ are generated by a function f plus noise ε with a variance σ^2 and zero mean such that $\hat{y} = f(\hat{x}) + \hat{\varepsilon}$, we get

$$\mathbb{E}\left[\left(\hat{y}-\hat{\tilde{y}}\right)^2\right] = \mathbb{E}\left[\left(f(\hat{x})+\hat{\varepsilon}-\hat{\tilde{y}}\right)^2\right]. \quad \text{(A2)}$$

We now add and subtract $\mathbb{E}\left[\hat{\tilde{y}}\right]$ to get

$$\mathbb{E}\left[\left(\hat{y}-\hat{\tilde{y}}\right)^2\right] = \mathbb{E}\left[\left(f(\hat{x})+\hat{\varepsilon}-\hat{\tilde{y}}+\mathbb{E}\left[\hat{\tilde{y}}\right]-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)^2\right]. \quad \text{(A3)}$$

We calculate the expression inside the parentheses to

$$\begin{aligned}
&\left(f(\hat{x})+\hat{\varepsilon}-\hat{\tilde{y}}+\mathbb{E}\left[\hat{\tilde{y}}\right]-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)^2 \\
&= \left(\left(f(\hat{x})-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)+\hat{\varepsilon}+\left(\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right)\right)^2 \\
&= \left(f(\hat{x})-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)^2+\hat{\varepsilon}^2+\left(\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right)^2+2\hat{\varepsilon}\left(f(\hat{x})-\mathbb{E}\left[\hat{\tilde{y}}\right]\right) \\
&+2\varepsilon\left(\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right)+2\left(\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right)\left(f(\hat{x})-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)
\end{aligned} \quad \text{(A4)}$$

From equation (A3) we see that we should take the expectation value for the expression in (A4) to get the MSE. since we have that the noise $\hat{\epsilon}$ are independent and identically distributed, so we can take the expectation value of these independently. Since $\mu = 0$ the expectation value is 0, so the fourth and fifth term disappear. For the sixth term we have the difference between the prediction and the expectation value of the prediction times the difference between the expectation value of the model and the actual function. These quantities are independent, as the difference between the actual function and the expectation value of the prediction is not stochastic. We can then take the expectation value separately. We then see that the expectation value for $\mathbb{E}\left[\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right]$ is 0 since $\mathbb{E}\left[\mathbb{E}\left[\hat{\tilde{y}}\right]\right] = \mathbb{E}\left[\hat{\tilde{y}}\right]$. So the last term is also 0. We then get the final expression

$$\begin{aligned}
\mathrm{MSE}(\mathbf{X},\hat{\beta}) &= \mathbb{E}\left[\left(f(\hat{x})-\mathbb{E}\left[\hat{\tilde{y}}\right]\right)^2+\hat{\varepsilon}^2+\left(\mathbb{E}\left[\hat{\tilde{y}}\right]-\hat{\tilde{y}}\right)^2\right] \\
\mathrm{MSE}(\mathbf{X},\hat{\beta}) &= \frac{1}{n}\sum_i\left(f_i-\mathbb{E}[\hat{\tilde{y}}]\right)^2+\frac{1}{n}\sum_i\left(\tilde{y}_i-\mathbb{E}[\hat{\tilde{y}}]\right)^2+\sigma^2,
\end{aligned} \quad \text{(A5)}$$

by setting element i in $f(\hat{x})$ to f_i and element i in $\hat{\tilde{y}}$ to $\tilde{y}_i$ and using that $\mathbb{E}\left[\hat{\varepsilon}^2\right] = \sigma^2$.

Appendix B: Additional results

Can be found in the github repository https://github.com/Magnus-SI/FYS-STK3155