Project 1 FYS-STK3155 (FINAL VERSION WILL BE ADED 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. On (some percent) noisy data generated based on the Franke Function with two independent variables, we found that the Ridge method performed best with an optimal choice of $\lambda = \textit{something}$ and polynomial degree (some degree). Using terrain data from (some area), Norway, we found that (what works best). Finally, we showed how the results could be explained in part by the bias variance trade-off.

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 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. This gives a similar situation as for the Franke-Function, and 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 need to 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. In that case one can 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

In 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 tradeoff, where one have to find the "correct" model complexity somewhere in between, this trade-off is called the bias-variance trade-off.

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 1^{st} 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 percentage of the difference between the maximum and the minimum value of the function. We will call the fraction of the difference between the maximum and minimum value we choose as noise $\hat{\sigma}$. To evaluate this 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

$$(\mathbf{XD})\left(\mathbf{D}^{-1}\hat{\beta}\right) = \hat{y}. \tag{14}$$

Since $\mathbf{B}$ is a diagonal matrix, a diagonal element d'_{ii} of the inverse matrix $\mathbf{B}^{-1}$ 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 $d_{ii} = \left[\max_k (\mathbf{X}_{ik})\right]^{-1}$. Equation (14) then becomes the "new" linear regression problem, where we use $\mathbf{XD}$ as the $\mathbf{X}$-matrix, in order to find $\mathbf{D}^{-1}\hat{\beta}$.

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:

$$R^2 = 1 - \frac{\sum_i (f_i - \tilde{y}_i])^2}{\sum_i (f_i - \mathbb{E}[f_i])^2} \tag{15}$$

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.

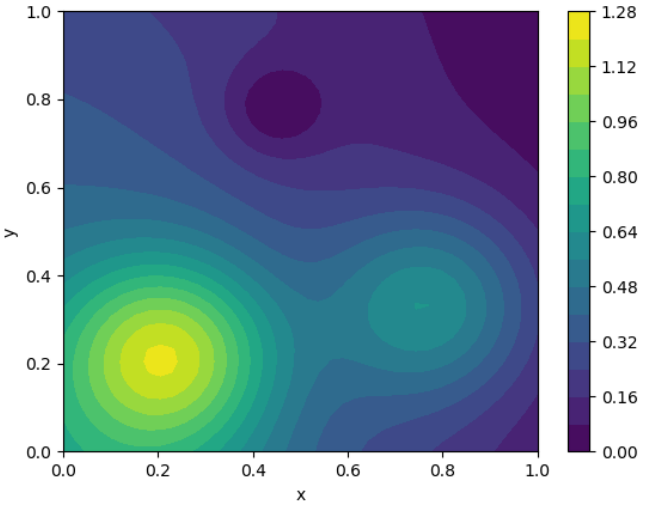
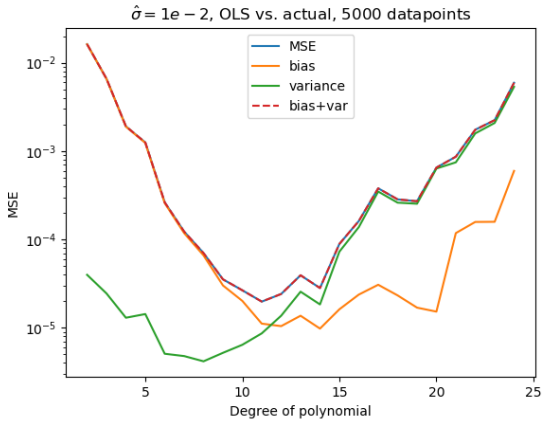


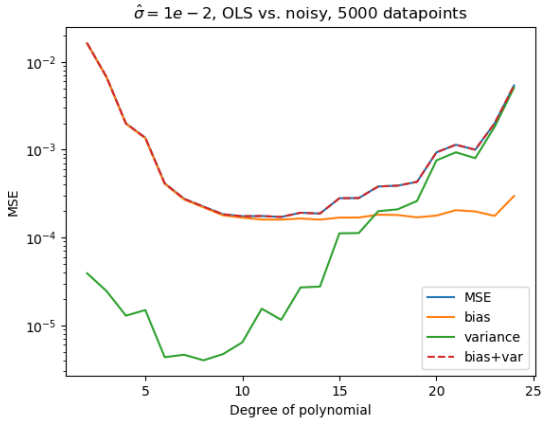
Figure 1. Contour of the Franke function

Bias Variance Trade-Off

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

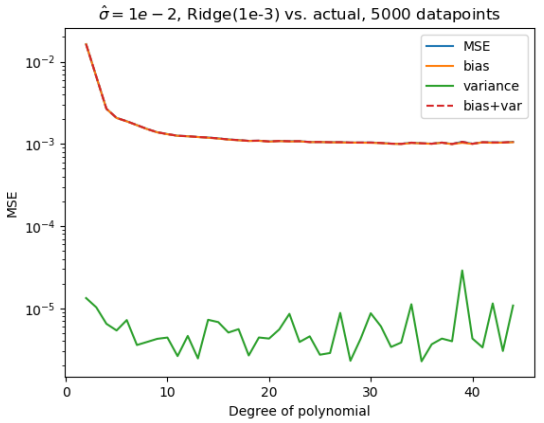


(a)Bias variance compare to actual data

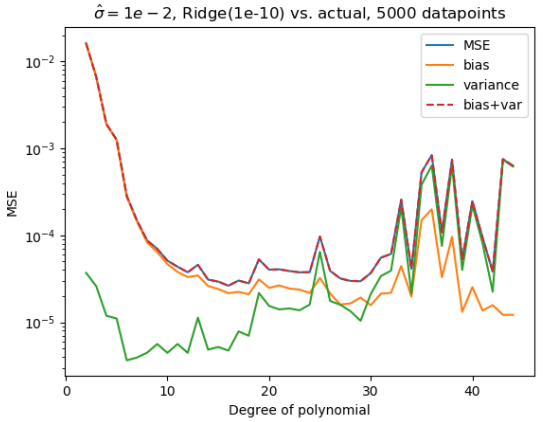


(b)Bias variance compared to noisy data

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



(a)Bias variance for $\lambda = 1e-3$



(b)Bias variance for $\lambda = 1e-10$

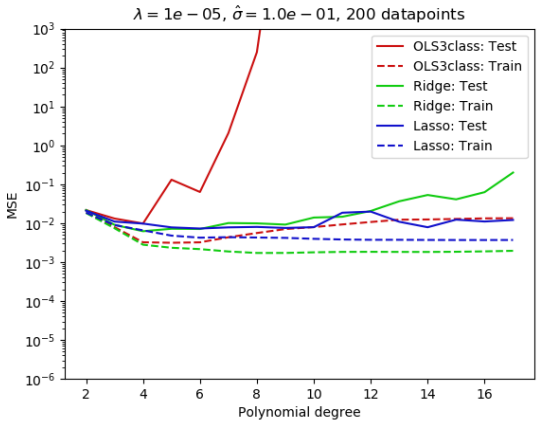
Figure 3. Bias variance with 5000 datapoints, with values of $\lambda = 1e-3, 1e-10$ on Ridge. The values were evaluated compared to actual data instead of the noisy data. The first plot was generated with 2 resamples.

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.

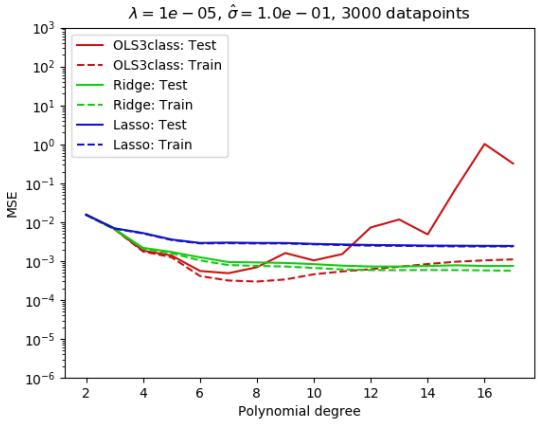
We also the found the MSE along with the bias and variance of when applying the Ridge method with values of $\lambda = 10^{-3}$ and $\lambda = 10^{-10}$ in Figure 3. Both are modeled on training data with added noise of $\hat{\sigma} = 1e-2$, and compared to test data with no added noise. In particular, 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 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



(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 $\hat{\sigma} = 10^{-1}$

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, I 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

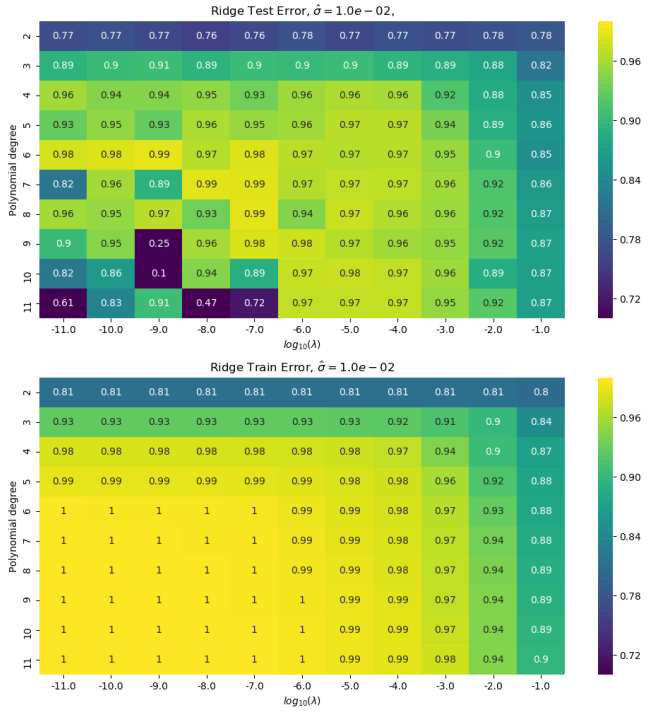


Figure 5. Generated using 200 data points of the Franke Function. Shows R2 score for varying polynomial degrees and values of λ for Ridge evaluated on the Franke function with $\hat{\sigma} = 0.01$ noise. $\hat{\sigma}$ refers to the standard deviation of the Gaussian noise as a fraction of the distance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

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

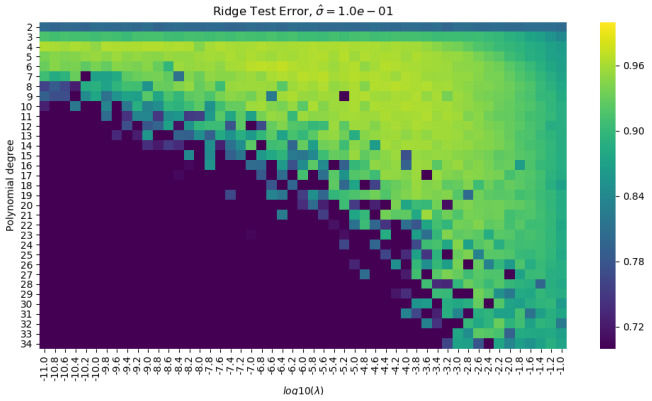


Figure 6. R2 score for varying polynomial degrees and values of lambda for Ridge evaluted on the Franke functio with $\hat{\sigma} = 10\%$ and 400 data points used. The optimal polynomial degree and value of λ found was degree 6 and $\log_{10}(\lambda) = -6.4$ with $R^2 = 0.9729$

Table I. Results for ridge regression on the Franke function, where λ_{opt} is 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	$\log_{10}\lambda_{\text{opt}}$
-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

Table II. Optimal Ridge parameters for indiced terrain data

# data points	R^2	RMSE	p	$\log_{10}\lambda$
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

Terrain data

The slice of terrain data we attempt to model is shown in Figure 9(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. It is clear that with increased complexity, the MSE decreases, but with a decreasing rate.

Figure 8 shows the R^2 score of the Ridge method as a function of the hyperparameter λ and the polyomial degree. Table II and III 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.

With the optimal value of p and λ suggested in Table II, Figure 9 shows a comparison between the actual data, and the approximated data applied by training Ridge on 12 000 data points. Figure 10 further shows similar comparison using a fit by a 50'th degree polynomial using the OLS method.

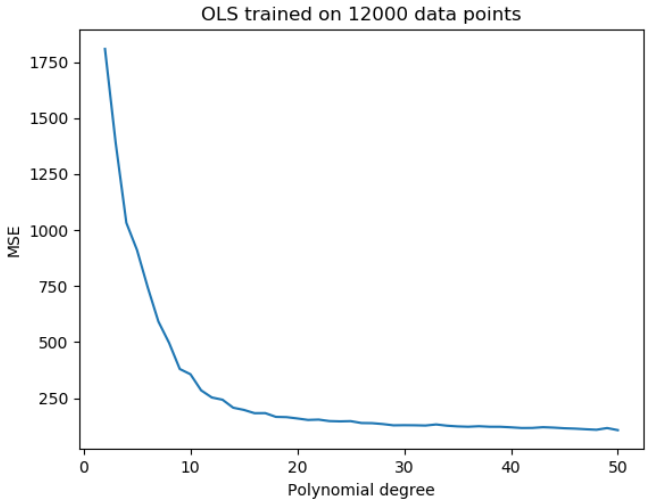
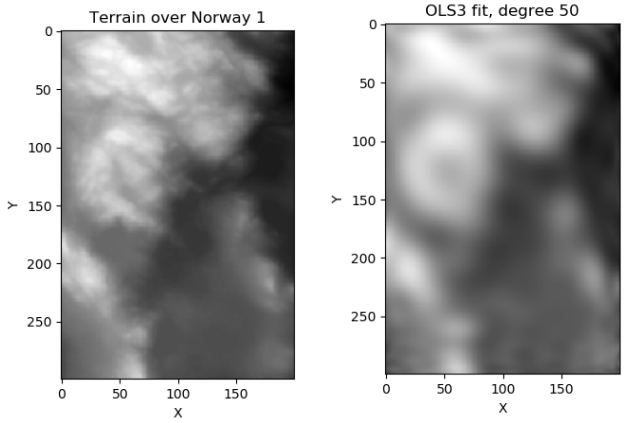


Figure 7. OLS fitted on 20% of the indiced terrain, with MSE evaluated on all of the terrain as a function of polynomial degree.

Table III. Same but with Ridgeskl

# data points	R^2	RMSE	p	$\log_{10}\lambda$
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



(a)Actual terrain data

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

Figure 10. Actual terrain data, vs approximated terrain data

NOTE:
When choosing and writing about subsections of terrain data, note that improved performance does not only come from higher density data per square area, but also

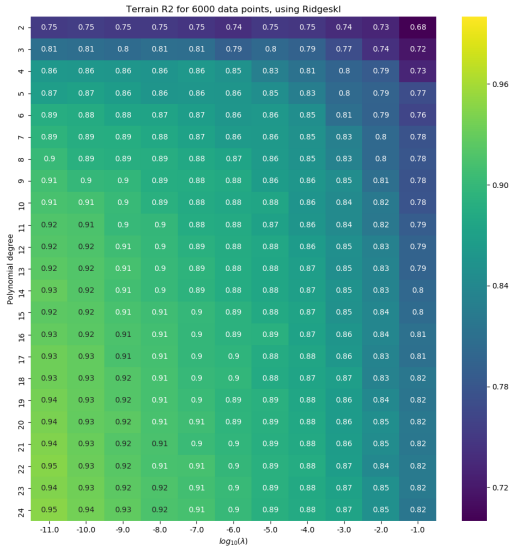
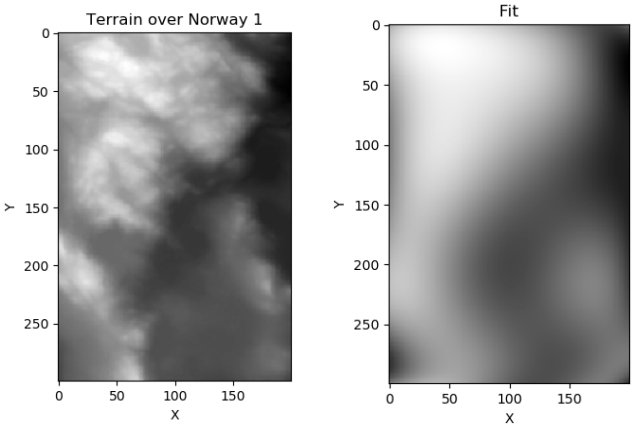


Figure 8. R2 score for varying polynomial degrees and values of lambda for Ridge evaluated on the terrain data with 20% of data points used. The optimal polynomial degree found was 24 which corresponds to the highest degree tested for. The corresponding optimal value of λ found was $\log_{10}(\lambda)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



(a)Actual terrain data

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

Figure 9. Actual terrain data, vs approximated terrain data

the fact that larger areas in general have more features that are harder to describe with polynomials. Polynomials in general would not be applicable on terrain which has a lot of sharp peaks and valleys as our data.

V. DISCUSSION

Franke function

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 demonstrates the bias-variance tradeoff, as increasing the complexity decreases the bias, which is visible in both datasets, but at high complexity variance causes most of the error, which is only visible when comparing with the test data. So the results here are as we expected. We also see this result clearly in Figure 2. At a lower complexity, the error in OLS is primarily bias, but at high complexity variance is the dominant term.

From Figure 5 we also see that for low values of λ (close to OLS) the test error becomes large at a lower complexity. This shows that for OLS, variance becomes dominant much quicker than for ridge regression. We also see this by comparing Figure 2, Figure 3 and Figure 3(b). For OLS the variance becomes larger than the bias at around a 16^{th} order polynomial. For $\lambda = 10^{-10}$ in ridge, this first happens at around a 35^{th} order polynomial. With $\lambda = 10^{-3}$ we get that the variance and the bias stay at roughly constant values from around a 10^{th} order polynomial to a polynomial of degree 40. So while λ adds some bias to the approximation, it also decreases variance, and in some cases this gives a lower MSE than OLS, when one compares with the actual values. We can also see this in Figure 3. Here we see that for very low complexity, MSE will decrease when the complexity is increased. However after one would expect the variance to become dominant in MSE (at around 10^{th} order polynomials), the values for bias and variance stay relatively constant, but with some fluctuations. This Illustrates how ridge regression help limit the influence of variance, as complexity gets large.

From Figure 4, we also see that for a relatively small dataset (Figure 4(a)), a biased methods like ridge and LASSO can actually give a better estimate than OLS, even though the error when looking at the training data is always the lowest for OLS. Here OLS starts overfitting at around a 8^{th} order polynomial, while ridge and LASSO have a relatively constant MSE on the test data up to a 16^{th} order polynomial. 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 is slightly better than that from ridge. This is because the variance will decrease as we add more data points. This means the error in OLS for predictions with high complexity will decrease, as the number of data points go up. This will also mean 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 have 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.

We see another version of this when we look at the added noise. From Table I 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. We see a similar trend for the other numbers of data points. This 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. We also see in general that with higher noise

From Figure 4(a) we also see 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 have 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 try the different methods on the terrain data, we see much of the same as when we used them on the Franke function. In Table II we find that, when we look at few data points, we get a worse estimate, based on the R^2 and MSE, this is as expected, and makes intuitive sense. 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. With fewer data points, the variance of the model will be high, to counter this the optimal "strategy" with ridge regression seems to select a low complexity and a relatively high λ. This gives a higher bias, but then also a lower variance, which turn out to be more important. Again, with more data points, the variance becomes 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, so the closest one gets to OLS, for that complexity.

Since the terrain data has such a large number of data points, it is hard to get to a complexity where overfitting occurs. We can see this in Figure 8, where we have used 6000 points of the dataset, and plottet the R^2 score for various complexities and λs. In Figure 9 and Figure 10 we have plotted the fit given by an 8^{th} order ridge regression and a 50^{th} order OLS regression for the smaller terrain data. We see that with a 50^{th} order OLS the results look fairly close to the actual terrain. In Figure 7 we also see that even to a 50^{th} order polynomial, the MSE from the test-train split does not start to increase due to variance. So in this case OLS seems to work best.

VI. CONCLUSION

Some main points that should probably be here: (not final text, but just some ideas)

What we found out about fitting with polynomials using the different methods. Polynomials in general is a bad idea on a set of terrain data, but by constraining it to a smaller area, one can get a reasonable fit. They are still polynomials however, so outside the range of the fit, the solutions explode, and does not actually represent anything real.

Our results supports that in general OLS 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 a too high variance. Here, Ridge and LASSO becomes useful, as they increase the bias to decrease the variance which in total can bring the MSE down. We also found that for the same value of λ, LASSO can give better results at cases with very high variance than ridge, which preformed best between OLS and LASSO. LASSO was however very time consuming compared to the other methods, as it does not have a closed form solution.

Also explain in short how the overfitting relates to the increase in variance, and how values of lambda can compensate for this.

Finally some closing words on the weaknesses of what we did, and what might be done differently.

[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