Analysis of Regression and Resampling Methods

Håkon Olav Torvik, Vetle Vikenes and Sigurd Sørlie Rustad

University of Oslo
Norway
October 10, 2021

CONTENTS

INTRODUCTION

Regression analysis is a statistical method for fitting a function to data. It is useful for building mathematical models to explain noisy observations. There are several regression methods to achieve this, all with their strengths and weaknesses. We will in this paper study three different methods; ordinary least squares, Ridge and Lasso regression. All the code, results and instructions on running the code can be found in our GitHub repository[1].

The mathematical model obtained from regression analysis can be used to predict an outcome, given some previously untested input. To build an accurate model, one has to train it on a lot of data. Real-world datasets usually have a fixed size, and getting more data can be practically impossible. By training the model on one particular data set only, we might risk that our model is only efficient for that specific configuration, while predictions made about different data sets will be poor. It is therefore useful to have tools to avoid this for small datasets. Resampling methods are such tools. In addition to the regression methods, we will also study the effect of bootstrapping and cross-validating the data.

In order to study this, we need data to analyze. We will in this paper test the methods on two datasets. We will generate one dataset ourselves using a noisy analytical bivariate function, the Franke function (1), while the other data set is real-world terrain data. The fitting we will perform for both data sets is a polynomial fit with x and y dependence of the form $[1,\ x,\ y,\ x^2,\ xy,\ y^2,\ \dots]$.

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

We will begin by using the simplest regression method, ordinary least squares (OLS), to fit polynomials up to fifth order on the Franke function. At first we will not resample the data, only test our implementation and do some basic analysis. We will then add the resampling methods, Bootstrapping and Cross-Validation, and analyze various quantities from the resulting fits. After that we will do the same analyses as before, but using the more complex regression methods Ridge and Lasso. Having tested different methods on the Franke function, we will repeat these analyses in the end when studying terrain data. This paper will not study the effects of different scaling methods, train-test split ratios and amount of noise.

We have also written code to test our implementation of the methods, where we compare our results with those obtained using the open-source machine learning package Scikit-learn. These results are presented in Appendix B.

EXERCISE 1: ORDINARY LEAST SQUARE (OLS) ON THE FRANKE FUNCTION

We generate a dataset drawing $n = 30$ uniformly distributed points in $x, y \in [0,1]$. Equation (1) is then used to create n^2 z-values. In this domain, the Franke function takes values

[1] https://github.com/sigurdru/FYS-STK4155/tree/main/project1

$z \in [-0.45,\ 1.52]$ before scaling. To this we add normally distributed stochastic noise. The noise has a mean of $\mu = 0$, and variance $\sigma^2 = 1$. It is scaled by a factor ϵ, which we normally set to $\epsilon = 0.2$. This gives a substantial amount of noise, without completely drowning out the data. The data with and without noise is visualized in figure 1.

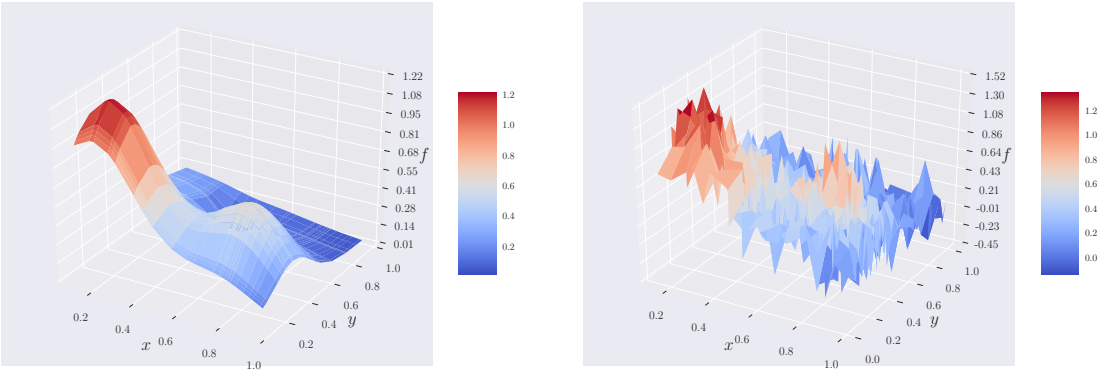

Figure 1. Here we have displayed our noisy data along with the analytical function. The left figure shows the analytical Franke function, while the right figure is the Franke function with noise.

It is good practice in regression analysis to scale the data, such that different features take values in the same order of magnitude. There are several different methods for scaling, one of which is StandardScaler. This is done by subtracting the mean from the data, before dividing this by the standard deviation. This centres the data around 0 and sets the variance to 1. Because our x and y is already in the domain $[0, 1]$ with variance 1, this only shifts it down, and has no outcome on the regression results. For z generated using (1) the effect is small, but we choose to do this for consistency. Our code allows for the following different scaling methods: no scaling, Normalizer and MinMax, though these will not be used. One main advantage of scaling the data becomes evident when we consider terrain data later on. The altitude of the terrain could vary hundreds of meters between the lowest and highest point, and if we were to study the errors of our model predictions, we would not necessarily be able to determine whether an error of order 10 is good or bad. By scaling the data we get the relative error of our prediction, which gives a simpler assessment of the resulting fit.

We are fitting a bivariate polynomial of degree P. With P as the highest polynomial degree, the polynomial will have $p = (P+1)(P+2)/2$ terms. It can, without noise, be written

$$z(x, y) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \dots \tag{2}$$

where the β-parameters are coefficients for each term. The x and y's are collected in a design matrix X. The columns of X are the terms, while the rows are all terms for a single datapoint. X thus have size $(n^2 \times p)$. The βs are collected in a vector $\boldsymbol{\beta}$ of length p. These are unknown parameters to be determined.

Thus, out model with stochastic noise takes the form

$$\mathbf{z} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{3}$$

One challenge in our case is that our data is two-dimensional, i.e. $\mathbf{z}$ has dimensions (n, n). We create a workaround by mapping our two-dimensional data, to a one dimensional $(n^2, 1)$ vector. This is an important note, and we will do this throughout the report.

We want to create a model to predict $\mathbf{z}$ given an input. $\tilde{\boldsymbol{\beta}}$ are the optimal parameters obtained from regression. Our prediction $\tilde{\mathbf{z}}$ is then given by

$$\tilde{\boldsymbol{z}} = X\tilde{\boldsymbol{\beta}}.$$

To asses the quality of the model, we need a way to quantify the error. We will use the mean square error (MSE), and R^2-score, given by

$$MSE(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = \frac{1}{N}\sum_{i=0}^{N-1}(z_i - \tilde{z}_i)^2 = \frac{1}{N}\left\{(\mathbf{z} - X\tilde{\boldsymbol{\beta}})^T(\mathbf{z} - X\tilde{\boldsymbol{\beta}})\right\}.$$

$$R^2(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z})^2}, \quad \bar{z} = \frac{1}{n}\sum_{i=0}^{n-1} z_i$$

MSE will always be larger than 0, and an MSE of 0 means our model fits the data perfectly. The R^2-score is a number between 0 and 1, with 1 being a perfect fit. In order to find the optimal parameters $\tilde{\boldsymbol{\beta}}$ we need a cost function to minimize. Choosing MSE as the cost-function gives the expresion for OLS-regression.

$$C(\boldsymbol{\beta}) = MSE(\boldsymbol{z}, \tilde{\boldsymbol{z}})$$

Minimizing this is done by doing the derivative with respect to all the parameters, and finding the zero.

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \equiv 0 = X^T(\boldsymbol{z} - X\tilde{\boldsymbol{\beta}}). \tag{4}$$

We can rewrite this as

$$X^T\mathbf{z} = X^TX\tilde{\boldsymbol{\beta}} \implies \tilde{\boldsymbol{\beta}} = (X^TX)^{-1}X^T\mathbf{z} \equiv \hat{\boldsymbol{\beta}} \tag{5}$$

This is OLS-regression. Thus $\hat{\boldsymbol{\beta}}$ are the optimal parameters that minimizes the cost function.

Now that we have an expression for the optimal parameters, we also want to evaluate how good our results are. We do this by splitting our data into two sets, one set for training (80%) and one set for testing (20%). We can then find the optimal parameters ($\hat{\boldsymbol{\beta}}$) using the train data, and then compare our test data to the actual data.

We can now perform OLS on our data and plot the MSE and R^2-score as a function of polynomial degree. It is important to note that when we fit using a polynomial of degree p, all terms from lower polynomial degrees are included in the fit. We expect the MSE to strictly decrease and R^2 to strictly increase for the train data, as our model becomes more complex. We expect the same for the test data, however at a certain polynomial degree we might se overfitting. This would make our error increase and R^2 score decrease. The results are plotted in figure 2. Overfitting is when the model nearly perfectly fits the training data, since it tries to fit specific data points. This results in a poor performance on the test data, as these data points are different from the train data. Splitting the data is thus a way to prevent overfitting, as we choose the model complexity which minimizes the test error.

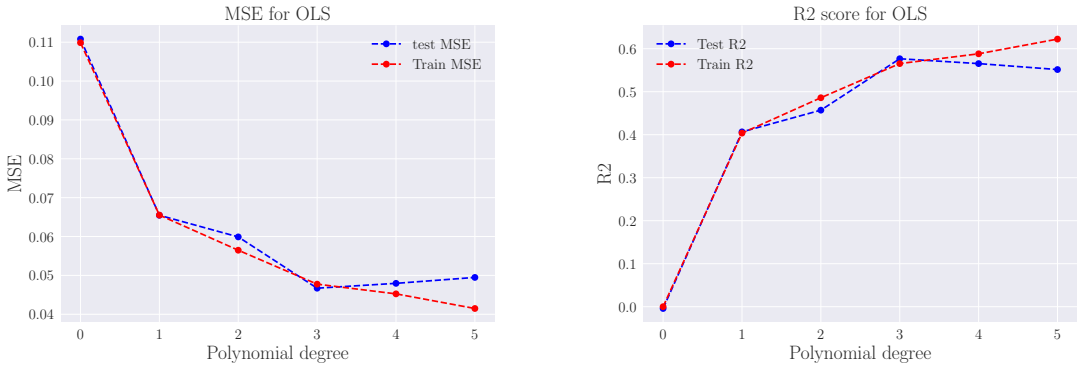

Figure 2. Here we have plotted the MSE and R^2-score for OLS. The left figure shows the MSE and right shows the R^2 score. The red dotted line is from the train data, and the blue dotted line is from the test data.

From figure 2 we see that the MSE strictly decrease and R^2-score increase for the test data. This indicates that we have not reached a point of overfitting yet.

NOTE: REWRITE ABOVE WHEN PLOTS ARE UPDATED

It is also interesting to look at the variance in the parameters, because this would give an indication of overfitting. We know that for a set of optimal parameters $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \mathbf{z}$, We have the expectation value

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \mathbb{E}[(X^T X)^{-1} X^T \mathbf{z}] = (X^T X)^{-1} X^T \mathbb{E}[\mathbf{z}] = (X^T X)^{-1} X^T X \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}.$$

With this we can calculate the variance in $\hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}\left[(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^T\right] \\
&= \mathbb{E}\left[((X^T X)^{-1} X^T \mathbf{z} - \hat{\boldsymbol{\beta}})((X^T X)^{-1} X^T \mathbf{z} - \hat{\boldsymbol{\beta}})^T\right] \\
&= (X^T X)^{-1} X^T \mathbb{E}\left[\mathbf{z}\mathbf{z}^T\right] X (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T \\
&= (X^T X)^{-1} X^T \left[X\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T X^T + \sigma^2\right] X (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T \\
&= \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T + \sigma^2 (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T = \sigma^2 (X^T X)^{-1}
\end{aligned}$$

Here we used $\mathbb{E}(\mathbf{z}\mathbf{z}^T) = X\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T X^T + \sigma^2 \mathbb{1}$, where $\sigma^2 = \epsilon = 0.2$ is the variance of the noise and $\mathbb{1}$ is the identity. Now we have an estimate for the variance in parameter $\hat{\boldsymbol{\beta}}_j$ given by

$$\boldsymbol{\sigma}^2(\hat{\boldsymbol{\beta}}_j) = \sigma^2 [(X^T X)^{-1}]_{jj}. \tag{6}$$

Using equation (6) we can plot the confidence interval for the different parameters, for several polynomial degrees (see figure 3).

Figure 3. Here we have plotted the ideal parameters, for different polynomial degrees and with the confidence intervals. Top left figure has polynomial degree $p = 1$, top right $p = 3$ and bottom left $p = 5$. All the plots are of $\boldsymbol{\beta}$ as a function of the index in the vector. The confidence interval is $\pm 2\boldsymbol{\sigma}(\hat{\boldsymbol{\beta}}_j)$, where $\boldsymbol{\sigma}(\hat{\boldsymbol{\beta}}_j)$ is given by equation (6).

We see in figure 3 that the variance increases as a function of p. This is expected because we should get overfitting for higher polynomial degrees. For larger p we also have more parameters to tune. This means that a change in one, could be compensated for by tuning the other parameters in the vicinity. We can see the result of this in the bottom left plot in figure 3. We observe a large variance in the middle, where we have a high density of parameters, and a lower variance on the edges. One also notices an comparatively low variance for the first parameter β_0, because the fixed intercept.

EXERCISE 2: BIAS-VARIANCE TRADE-OFF AND BOOTSTRAPPING

Here we want to evaluate OLS using bootstrapping as a resampling technique. Bootstrap is a way of simulating having more data than you actually have, by reusing the same data. Lets say you have N datapoints, then for every bootstrap-iteration, you would draw N points, with replacement. Then perform the regression analysis, and repeat this process B times. By placing back the datapoints, you would in many cases use the same datapoint several times, even in the same regression. This random sampling will simulating having more data, and still reduce the amount of overfitting compared with training on the same data over and over.

Now, with bootstrapping, we want to take a look at when we get overfitting. It is common to see overfitting when we have a high polynomial degree (p_{max}) compared to the number of datapoints. Thus in figure 4 we have plotted the test and train MSE for $p_{\text{max}} = 20$, using $B = 90$ resampling iterations. The left plot shows an indication of overfitting at polynomial degree $p = 9$. Thus on

the right hand plot we have displayed the MSE for $p_{\text{max}} = 10$, and as expected we see overfitting at $p = 9$. One unexpected result is the sharp increase and decrease in MSE. This indicates that our model is much better for some polynomial degrees than others. The sharp fluctuations can be explained by the surge in complexity from one polynomial degree to the next. This is displayed in figure 3, when we increase the polynomial degree by two, the number of parameters more than double.

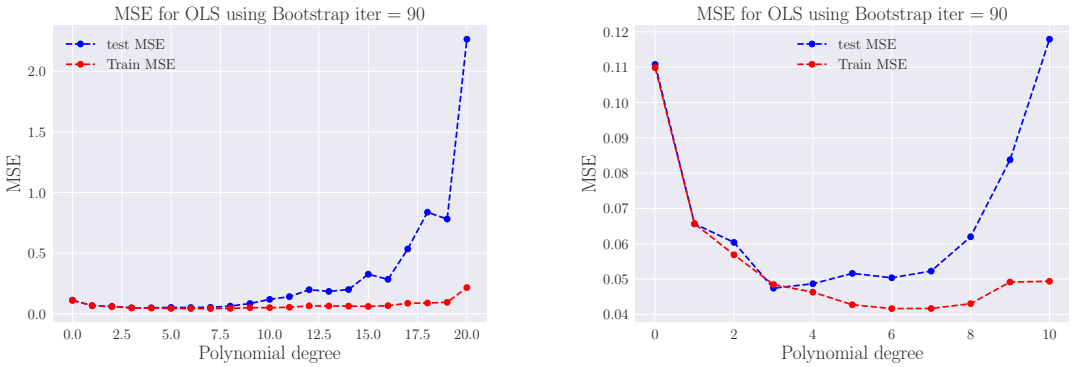

Figure 4. Overfitting, høyre med støy, venstre uten støy.

Before we take a look at bias variance trade-off, we will derive the expressions. With our data model given in equation (3) we find our model by considering the cost function, given as

$$C(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=0}^{N-1}(z_i - \tilde{z}_i)^2 = \mathbb{E}[(\mathbf{z} - \tilde{\mathbf{z}})^2]$$

where $\mathbb{E}$ is the expected value.

We can show that this can be written as [2]

$$\mathbb{E}[(\mathbf{z} - \tilde{\mathbf{z}})^2] = \frac{1}{N}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \frac{1}{N}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \sigma^2 \tag{7}$$

where f_i is the true data value at point i, and σ^2 the standard deviation of the noise.

In equation (7) the first term represents the square of the bias, the second term represents the variance while the last term represents the variance of the irreducable error ϵ. When performing linear regression the variance is a measurement on how much our model changes with different training sets. High variance will therefore occur if a different training set resulted in very different values of the individual estimators, $\boldsymbol{\beta}$. This will be the case for overfitting when our model is essentially trying to reproduce variations from the noise. The bias provides information about the difference between our model and the true data values. If our model is missing out on underlying

[2] The derivation is given in Appendix A

structures in our data we would get a high bias. High bias will thus be the case for an underfitted model. Our goal is therefore to minimize the bias and variance in our model.

Using equation (7) we can plot the bias, variance and MSE for the test and train data, using the bootstrap method. For both datasets we expect the bias and MSE to start high and variance to start low. For the train data we should get a strictly better fit for higher polynomial degree, making both the bias and MSE to decrease. For the testing data however, when we start to see overfitting (around $p = 9$) variance and error should increase, while bias stays low. In figure 5 we have plotted bias, variance and MSE for the test and train data, using $B = 90$ resampling iterations. Because we expected to see overfitting at around $p = 9$ we plotted for $p_{\max} = 10$.

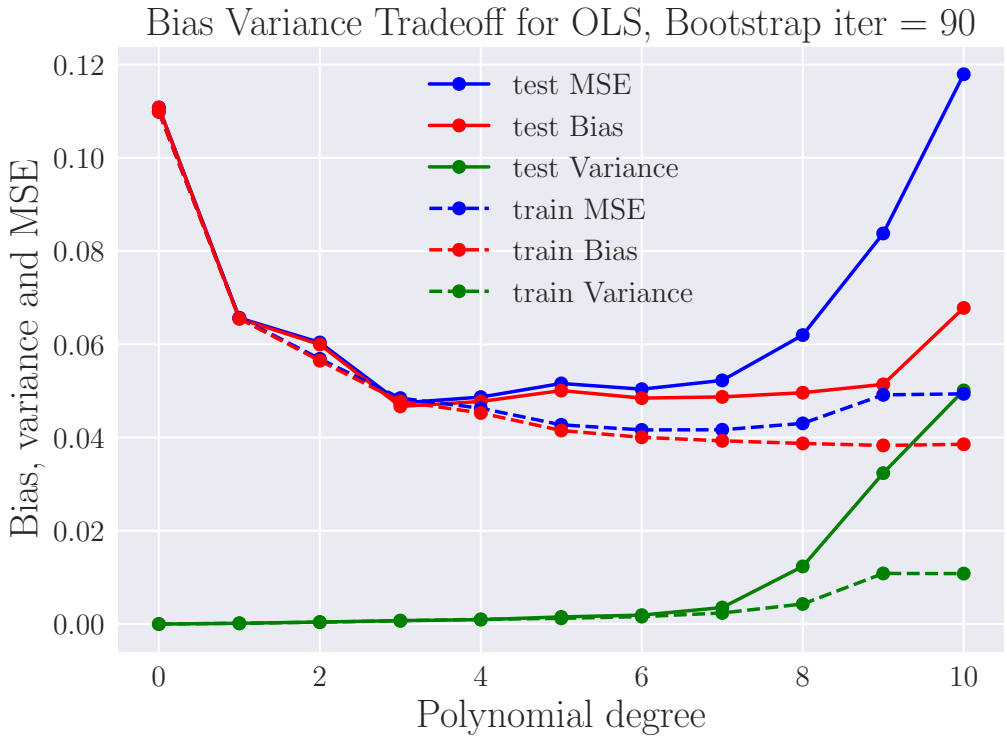

Figure 5. Here we have plotted the bias, variance and MSE for the test and train data. The x-axis shows the polynomial degree, and the y-axis indicates the bias, variance and MSE. The dotted lines are for the train data and solid lines are the test data.

As expected we se an increase in MSE and variance for the test data, when we start overfitting. The bias does not act completely as expected. It does decrease in the beginning, however it starts increasing with the MSE again at around $p = 8$. This however could be due to the sensitive nature of bootstrapping. As we saw in C, bias variance analysis with bootstrapping is very sensitive to the number of resampling iterations B, and gives unexpected results because of it. One thing we also saw in the overfitting experiment (figure 4) is the sharp increase and decrease for specific polynomial degrees. Taking all of this into account, the optimal fit that minimizes bias, variance

and MSE for OLS is $p = 6$.

The last term of (7) is a constant, and related to the noise in the data. Using data with more noise will give worse fit, and less noise gives a better fit. However, the relative bias and error will stay the same. In this paper all results have been produced with the same amount of noise. Had we used a different value, we would except the MSE-plots to be similar, with the only difference being the value of the error. As a sanity check, we remake figure 5 with lower and higher noise. See figure 6. Comparing with 5, we see that the graphs are relatively the same, just taking lower and higher values.

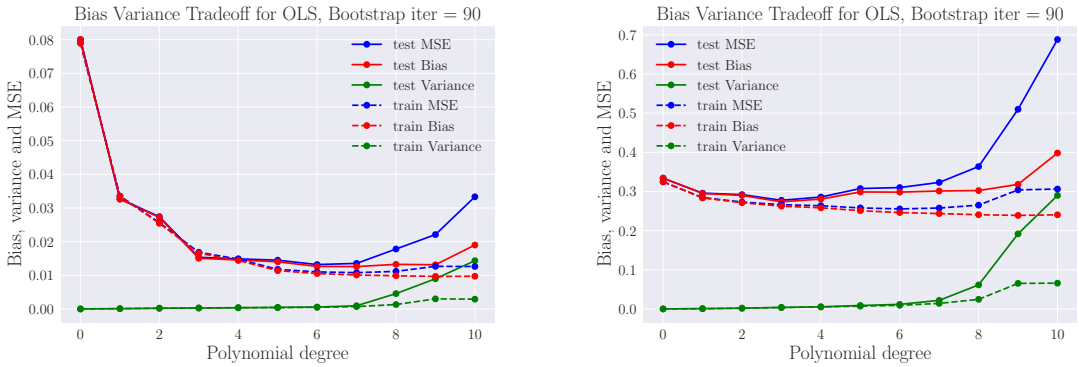

Figure 6. Here we have plotted the bias, variance and MSE for the test and train data, exactly as figure 5, but using lower (left) and higher (right) levels of noise. This is to show we get similar results

EXERCISE 3: CROSS-VALIDATION AS RESAMPLING TECHNIQUE, ADDING MORE COMPLEXITY

Here we have implemented cross-validation as the resampling technique. This works by first shuffling the data randomly, then split the data into k different groups. We can then decide on one group to be the test-data and perform linear regression on the rest. After that we can calculate the MSE, and repeat this process k-times. The final MSE would then be the mean MSE, after all the iterations. In figure 7 we have done this, trying for $k = [5, 7, 10]$.

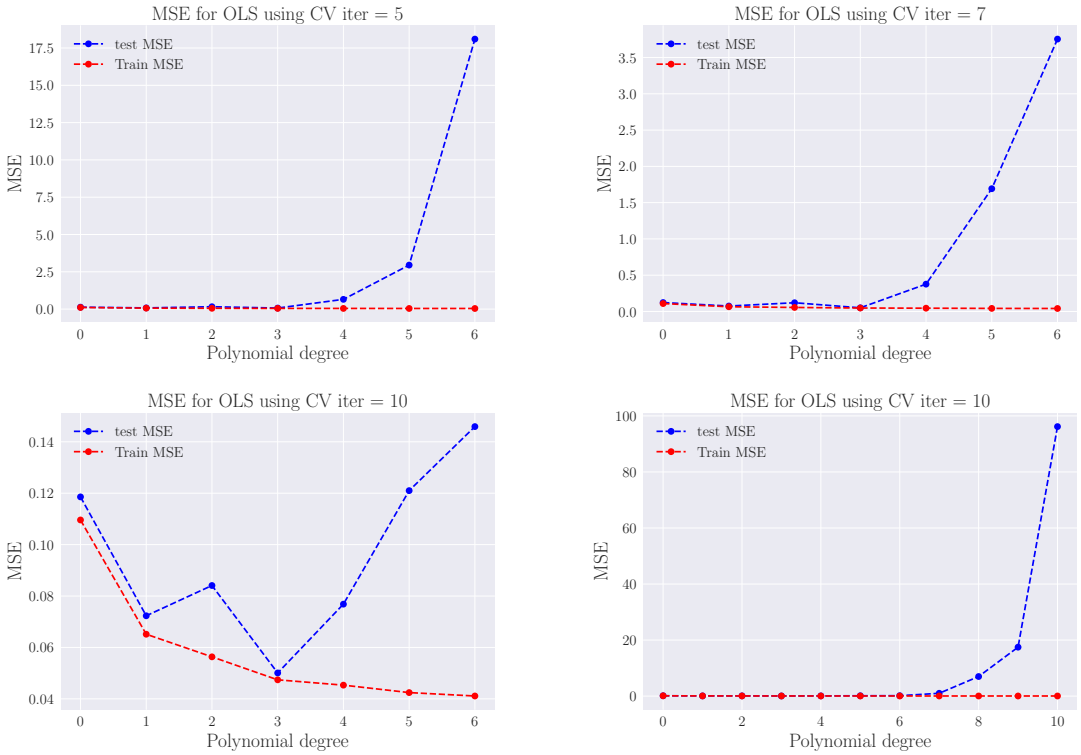

Figure 7. Here we have plotted the MSE when performing cross-validation as the resampling technique. On the x-axis we have polynomial degree, and MSE on the y-axis. The top left, right and bottom left figures are with $k = [5, 7, 10]$ iterations respectively. The bottom left plot is to illustrate the diverging MSE for higher polynomial degree, which we see irregardless of iteration-number.

When bootstrapping we saw overfitting at around $p = 6$. Looking at the plots in figure 7 there seems to be consistent overfitting at around $p = 3$. In the bottom right plot we have plotted for higher polynomials to illustrate that this error does not decrease for higher p. This does not depend on the number of resampling iterations. We see no obvious reason that the overfitting should start sooner compared to bootstrapping, and thus ground it in random fluctuations. We have consistently seen large variation in results based on number of resampling, and the random seed used when producing the results. Another interesting result is that a higher number of resampling iterations seems to decrease the overfitting effects. This coincide with the analysis on early overfitting. Having a larger number of resampling iterations should dampen the odd effects due to randomness.

EXERCISE 4: RIDGE REGRESSION ON THE FRANKE FUNCTION WITH RESAMPLING

In some cases, it might happen that the matrix $\mathbf{X}^T\mathbf{X}$ is singular, such that it is non-invertible. This is the case if some of the independent variables are highly correlated. Then OLS will not work, see (5). However, by adding a small number λ from the diagonal, the matrix is no longer

singular. This is called Ridge regression. The optimal parameters $\hat{\boldsymbol{\beta}}$ are given by

$$\hat{\boldsymbol{\beta}} = (X^T X + \lambda I) X^T \mathbf{z}, \tag{8}$$

where I is the identity matrix. The parameter λ can take any value, so finding the optimal is paramount. This is done by running the regression for many different values of λ and choosing the one giving the best results. Note that when $\lambda = 0$, the expression becomes that of OLS.

Now we want to perform the same bootstrap analysis as we did for OLS, using Ridge. Here we also take a look at when we have overfitting, for different values of λ and with bootstrap. Figure 8 shows MSE as function of polynomial degree for 4 different values of λ, starting at $10^{-}4$, increasing with a factor of 100 for each plot. We see that test error varies a lot, making it non-trivial to determine what is overfitting and underfitting. For the 3 first λ-values it seems we have overfitting for $p = 9$, while for $\lambda = 100$, it can not be conclusively decided. $p = 3$ and $p = 8$ seem to be the best two values before overfitting for the 3 smallest λ.

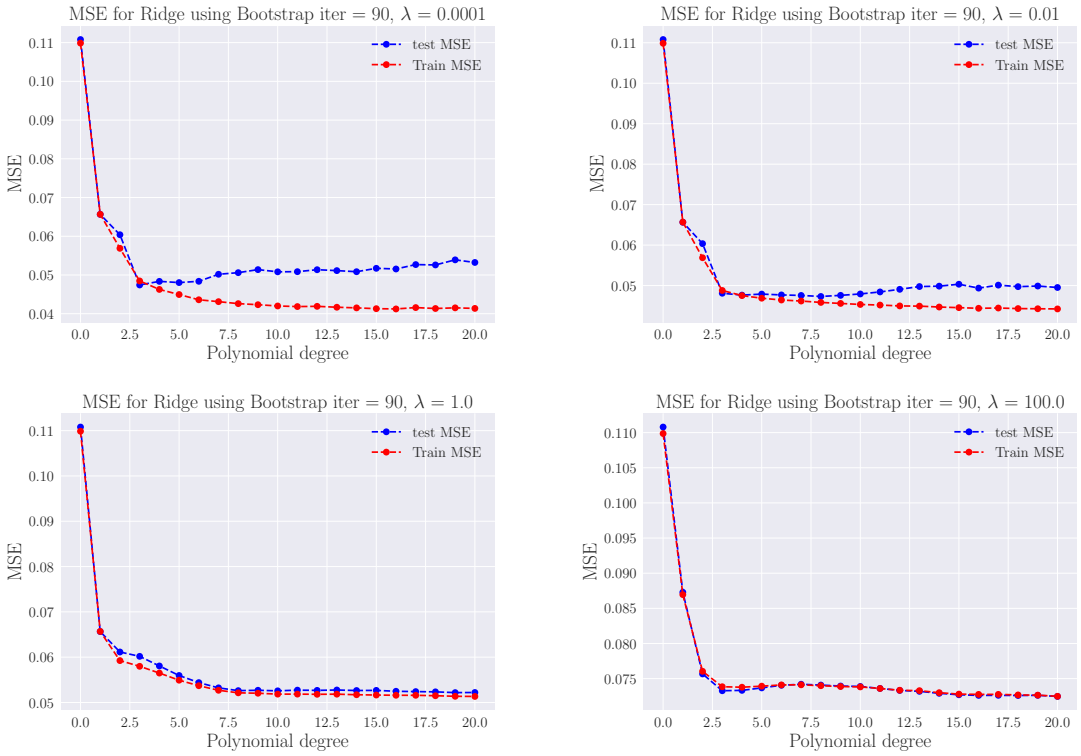

Figure 8. MSE using Ridgeregression and bootstrapping, as function of complexity, for different values of λ.

To explore the λ-dependence more in-depth, we now choose a single polynomial degree and plot the error as function of λ. In figure 9 we have done this for the two values noted above,$p = 3$ and $p = 8$.

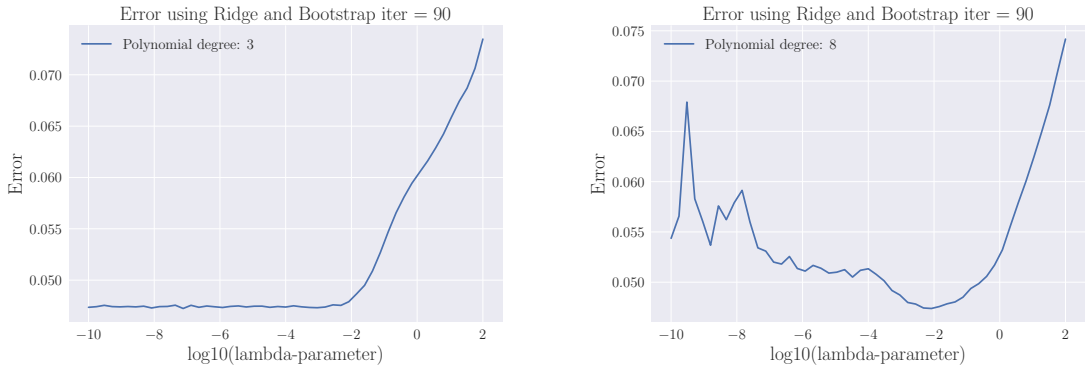

Figure 9. MSE using Ridgeregression and bootstrapping, as function of λ, for different polynomial degrees.

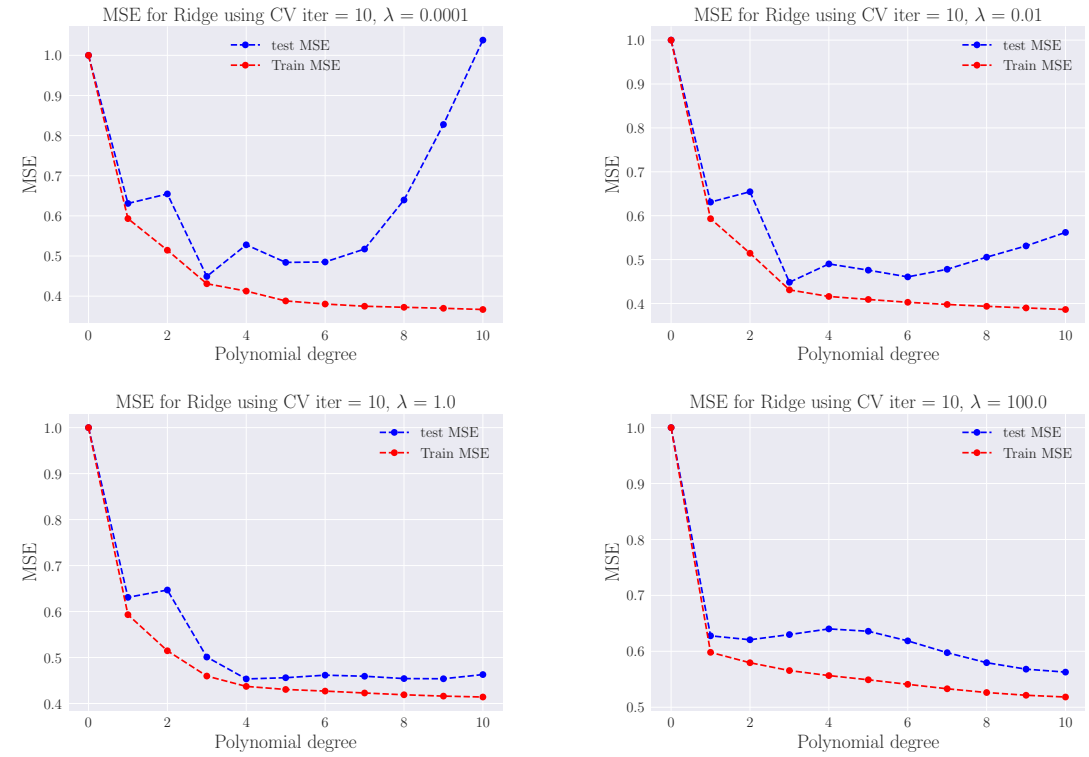

Figure 10. MSE using Ridgeregression and cross-validation, as function of complexity, for different values of λ.

EXERCISE 5: LASSO REGRESSION ON THE FRANKE FUNCTION WITH RESAMPLING

EXERCISE 6: ANALYSIS OF REAL DATA

Having used our data analysis methods to study the Franke function we will now move on to study data from the real world. We will consider Norwegian terrain data from here[3]. Because the datasets are large, we will only consider one small part of `SRTM_data_Norway_1.tif`, namely $f_{i,j}$ where $i, j \in [50, 100]$, i.e. $N = 50$ datapoints in each direction. It is useful to consider dimensionless data, thus we divide by the highest point $z_{i,j} = f_{i,j} / \max |f_{i,j}|$, and let $i, j \in [50, 100] \to x, y \in [0, 1]$ when plotting. Then we get the data visualized in figure 11.

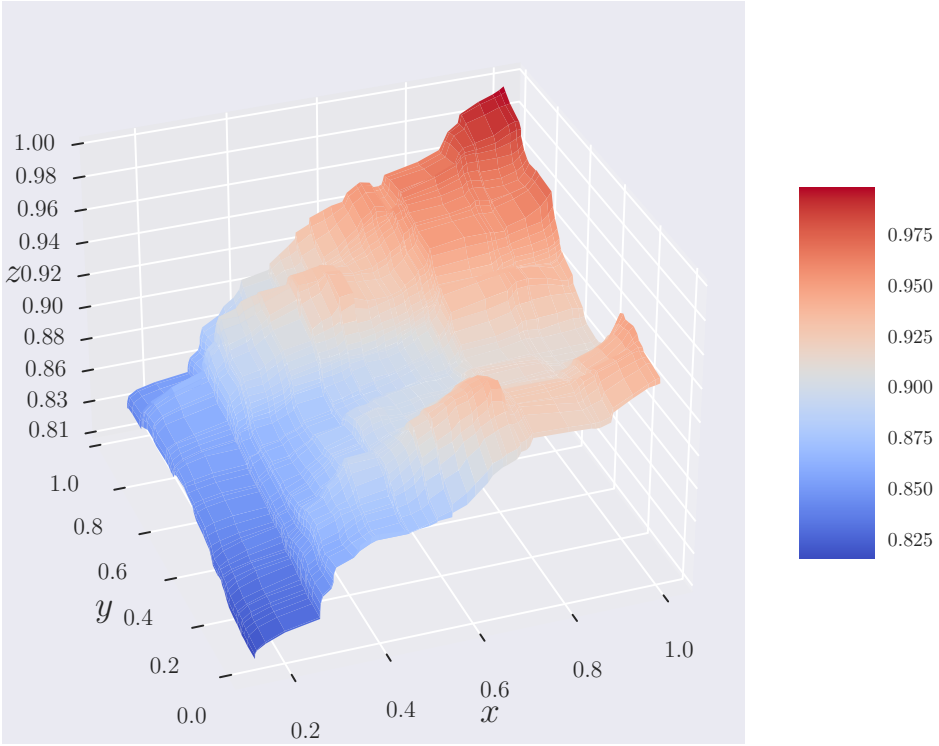

Figure 11. Here we have plotted the terrain data which we will use in this exercise.

We begin by studying the MSE using OLS-regression and bootstrapping as the resampling technique. Our results should mirror those we got in exercise 2, where the test MSE decreases in the beginning, and increases when we start overfitting. Choosing $B = 50$ iterations and max polynomial degree $p_{\text{max}} = 30$ we get the MSE illustrated in figure 12.

---

[3] https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects/2021/Project1/DataFiles

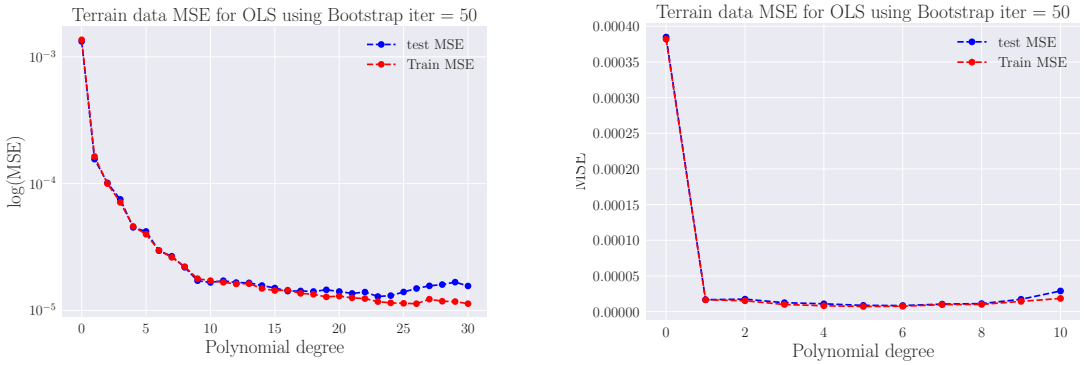

Figure 12. n50 log til venstre. n15 til høyre

(with a logarithmic y-axis. Using a data set with \$50\{}cross50\$ data points yields a little noticeable change in the mean squared error, but we see a potential overfitting occuring for \$P>15\$. To see that we get overfitting for the terrain data eventually, we plot the mean squared error for the same resampling method, but with \$n=10\$ points in each direction up to a polynomial degree of \$P=10\$, shown on the right panel in figure \{}ref{fig:terrain_OLS_MSE_bootstrap}. Here, we can see an evident overfitting starting to occur for polynomial degrees greater than \$6\$.)<– GAMMEL ANALYSE

As a simple sanity check, we plot the result of our fits for different polynomial degrees, which we can compare to the original data from figure 11. To capture the complexity of our data set, we consider four polynomial degrees, using $P \in [10, 20, 30, 50]$, where the first two are shown in the upper left and right panel of figure 13, respectively, while the latter two in the bottom left and right panel of the same figure, respectively.

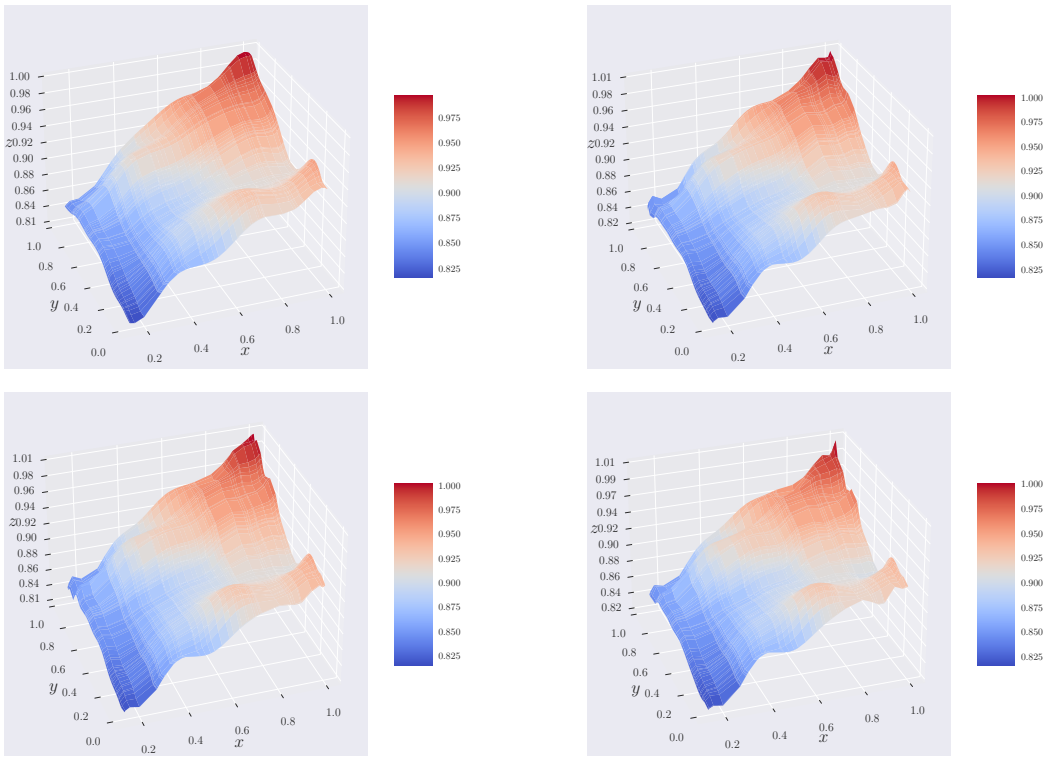

Figure 13. Data til venstre. Fit til høyre

In figure 13 we see that $p = 10$ gives a decent fit, as it captures the essential structure of the terrain. It does, however, miss out on many of the rough details in the landscape. For example the small elevation on the corner $(x, y) = (1, 1)$. With $p = 20$ we are able to reproduce more nuanced structures like the peak mentioned above. Increasing up to $p = 30$ does not change the fit much, other than a few more details at certain points. Finally, we study the resulting fit with $p = 50$. Although it provides some more rough futures of the terrain, it illustrates some difficulties of the fitting. The shape of the mountain top at $x = 1$ and $y = 1$ is no longer consistent with the original terrain data, and we git a tiny valley at $x = 1$ and $y = 0.8$, which is not an actual feature of the terrain data. Still, there are details missing along the x-axis at $y = 1$, and the structure of the two peaks at $y = 0$ is still not being represented accurately. We emphasize that the misrepresentations obtained with $P = 50$ might be a consequence of the splitting of train and test data, as well as that prticalur ploynomial degree. However, we are still not able to accurately predict important structures of the terrain.

We also include a plot of the mean squared error with bootstrap resampling for each factor 5 polynomial degree from $p = 5$ up to $P = 50$, shown in figure 14.

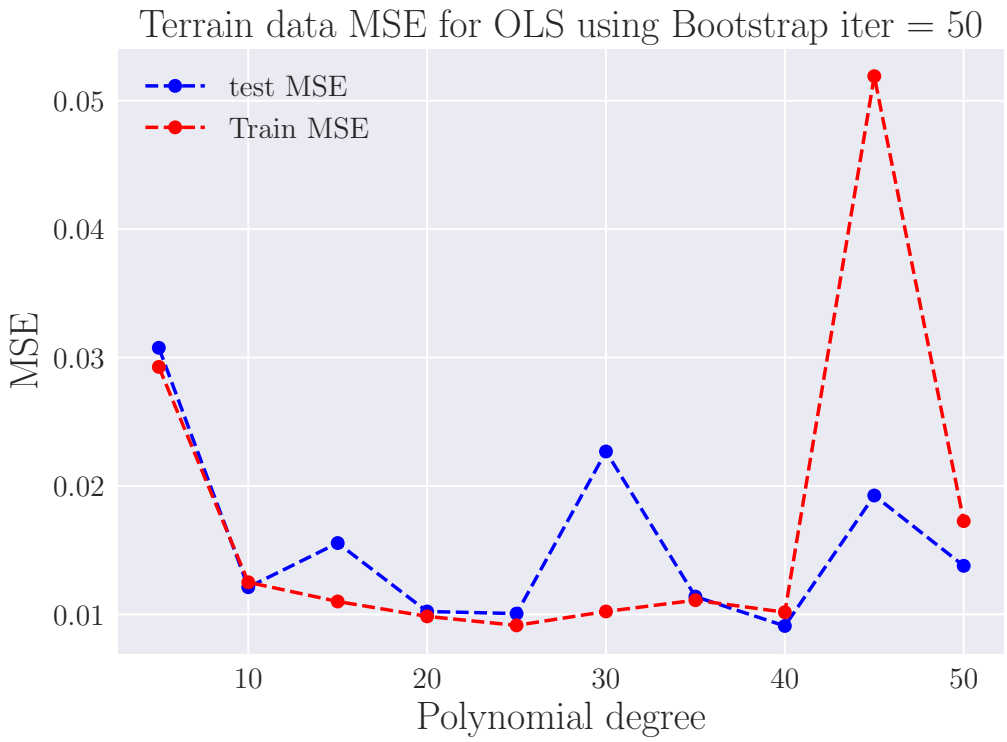

Figure 14. MSE

This illustrates the difficulties of fitting this terrain data, as subtle structures will still be missing in our fitted model, despite using polynomial degrees where overfitting is taking place. Despite large fluctuations in the MSE for higher order polynomials, it's worth mentioning that the MSE values themselves are not particularly large, which is expected since we don't include noise in our data. !SJEKK!

Next we move on to studying the MSE with cross validation. As with the bootstrap analysis, we expect the MSE to decrease in the beginning, and increase when overfitting. We plot use $p_{\max} = 10$ and $k = [5, 10]$ folds, and get the results shown in figure 15.

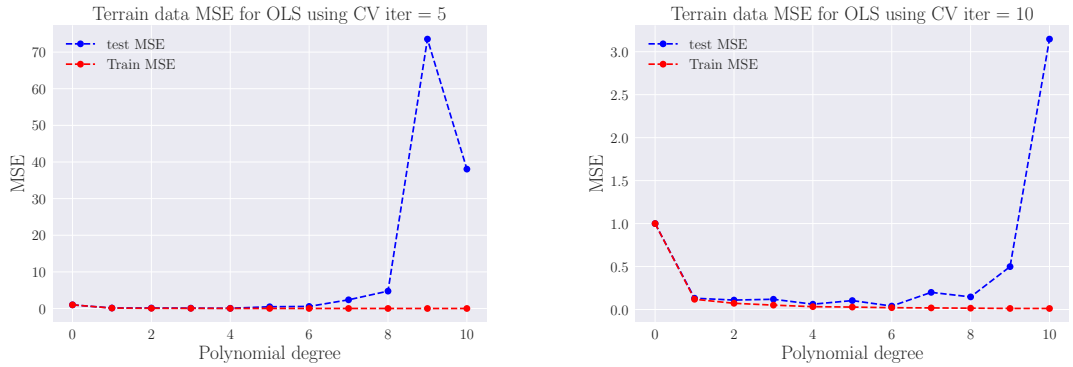

Figure 15. k-fold 5 til venstre. k-fold 10 til høyre

(Looking at figure 15 we see that the test MSE becomes incredibly large when $P > 8$. Doubling the number of k-folds, the MSE drops significantly, but there is still a clear increase for $p > 8$.) <– GAMMEL TOLKNING

Now we move on to bias-variance trade-off analysis, still expecting the same behavior as exercise 2. I.e. For both test and train, the bias and MSE should start high and variance low. The train data will get a strictly better fit for higher polynomial degree, making both the bias and MSE to decrease. For the testing data however, when we start to see overfitting (around $p = x$) variance and error should increase, while bias stays low. Using $p_{\text{max}} = 25$,

we are unable to see the relative behaviour of the quantities we plot for 50 data points, so we use a logarithmic scale on the y-axis. The result is shown on the left panel in figure 16. Once again, we include a result obtained with fewer data points to see any noticeable effect. The bias-variance trade-off result for 15 data points is shown on the right panel of figure 16, up to $P = 10$. This clearly shows an increase in the test MSE and test variance as P increases, while the bias declines.

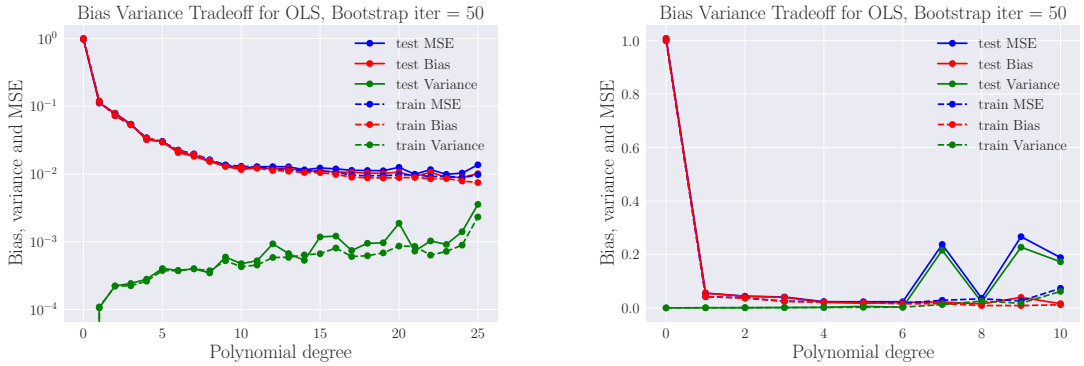

Figure 16. n50 log til venstre. n15 til høyre

Appendix A: Bias-variance Decomposition

We assume that our true data is generated from a noisy model with normally distributed noise ϵ with a mean of zero and standard deviation σ^2, i.e.

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

We have approximated this function with our design matrix $\mathbf{X}$ and our parameters $\boldsymbol{\beta}$ such that our model becomes $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$, where the values of $\boldsymbol{\beta}$ were obtained by optimizing the mean squared error via the cost function, given by

$$C(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right]$$

where $\mathbb{E}$ is the expected value.

We want to show that the above expression can be written as NoteNOTE: husk å definer alt, skriv i teksten hva f, y, yhat og ytilde er!

$$\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2$$

We begin by inserting our model expression for $\mathbf{y}$ and adding and subtracting $\mathbb{E}[\tilde{\mathbf{y}}]$ inside the expected value, before we square the expression.

$$\begin{aligned}
\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] &= \mathbb{E}\left[(f(\mathbf{x}) + \boldsymbol{\epsilon} - \tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}])^2\right] = \mathbb{E}\left[((f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}]) + \boldsymbol{\epsilon} + (\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}))^2\right] \\
&= \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \boldsymbol{\epsilon}^2 + (\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2\right] \\
&\quad + \mathbb{E}\left[2\boldsymbol{\epsilon}(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}]) + 2\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}) + 2(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\right]
\end{aligned}$$

where the cross terms have been written on a separate line since the expected value is linear. Next we will focus on the cross-terms. Since $\boldsymbol{\epsilon}$ is normally distributed, it's expected value is simply the mean, which is zero in our case. The two cross terms involving $\boldsymbol{\epsilon}$ is therefore zero, so we only need to consider

$$\mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\right] = \mathbb{E}\left[f(\mathbf{x})\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[f(\mathbf{x})\tilde{\mathbf{y}}\right] - \mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}[\tilde{\mathbf{y}}]\right] + \mathbb{E}\left[\tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}]\right]$$

Since the expected value of an expected value is just the expected value itself the last two terms in the above equation both become $\mathbb{E}[\tilde{\mathbf{y}}]^2$, canceling each other out. Using that $f(\mathbf{x})$ is a deterministic function, we have $\mathbb{E}[f(\mathbf{x})] = f(\mathbf{x})$. Expressing $f(\mathbf{x})$ in terms of its expected value, we can write the first two terms in the above equation as

$$\begin{aligned}
\mathbb{E}\left[f(\mathbf{x})\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[f(\mathbf{x})\tilde{\mathbf{y}}\right] &= \mathbb{E}\left[\mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[\mathbb{E}\left[f(\mathbf{x})\right]\tilde{\mathbf{y}}\right] \\
&= \mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}] = 0
\end{aligned}$$

Hence, all the cross terms in the expected value cancel out, and we're left with

$$\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] = \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}\left[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2\right] + \mathbb{E}\left[\epsilon^2\right]$$

Using that $\mathbb{E}[\epsilon^2] = \sigma^2$ and writing the expected values as sums with the notation $f(\mathbf{x}_i) = f_i$, we get the desired expression. Since we have chosen $\mathbf{z}$ as our data variable we replace all the y variables with z, yielding

$$\mathbb{E}\left[(\mathbf{z} - \tilde{\mathbf{z}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \frac{1}{n}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \sigma^2 \tag{A1}$$

which is what we wanted to show.

Appendix B: Testing our implementation

In order to make sure our algorithms are running correctly, it is necessary to perform tests. We did this by comparing our results to those produced by scikit-learn. First of all we generated some simpler data for testing:

$$y = \exp\left\{-x^2\right\} + 1.5 \cdot \exp\left\{-(x-2)^2\right\} + \epsilon. \tag{B1}$$

Here ϵ denotes normally distributed noise with zero mean and variance $\sigma^2 = 0.1$. We look at when x runs from $x_{\text{min}} = 0$ to $x_{\text{max}} = 1$ in $N = 50$ randomly distributed steps. This generates the testing data visualized in figure 17.

Figure 17. Here you we have plotted the testing data along with the analytical function. The solid blue line denotes the analytical function.

We test the implementation of OLS and Ridge regression. For Lasso we already used the scikit-learn implementation, so there is no reason to compare it with itself.

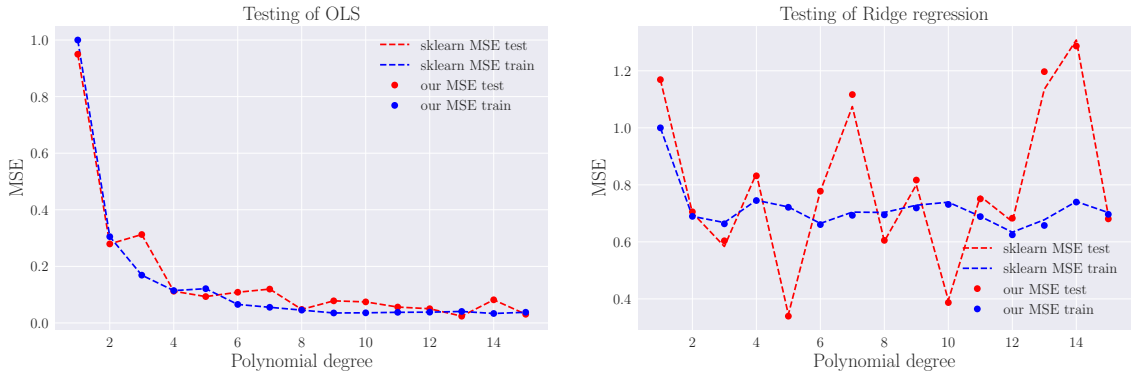

Figure 18. In this figure we have plotted the test and train MSE for both OLS (left plot) and Ridge regression (right plot). Along the x-axis we have polynomial degree, and MSE along the y-axis. The blue and red dashed line is the sklearn MSE respectively. We plotted our test and train MSE as red and blue dots respectively.

Looking at figure 18, our OLS implementation returns indistinguishable MSE's compared to sklearn. For ridge however we see some small deviations at polynomial degrees $p = 7$ and $p = 13$. The strong correspondence makes us confident that our implementations are correct, however, we one should be wary that our results deviate at some polynomial degrees. This is probably due to optimizations that the professional library sklearn has done.

Appendix C: Testing Bias Variance Tradeoff

To ensure a correct implementation of the Bias Variance tradeoff for the Bootstrap method, we want to compare our implemented results with results obtained by scikitlearn. This will also provide us insight of how these results behave. We consider the same data as when we tested our regression methods in Appendix B (equation (B1)). However with a slightly different domain, namely $x \in [-3, 3]$.

When we compare our own results with the ones obtained from scikitlearn, the result depends a lot on the particular setup we consider. We use $4/5$ of our data as the training data, 100 bootstrap iterations and polynomial degrees from $P = 0$ up to $P = 15$ and scale this data by subracting the mean before dividing by the standard deviation. The differences we see between when we compare are very sensitive to the configuration. To illustrate this, we first perform the bias variance tradeoff with $N = 170$ data points and then with $N = 169$ data points. The resulting plots for the two configuration are shown in figure 19, on the left and right panel, respectively.

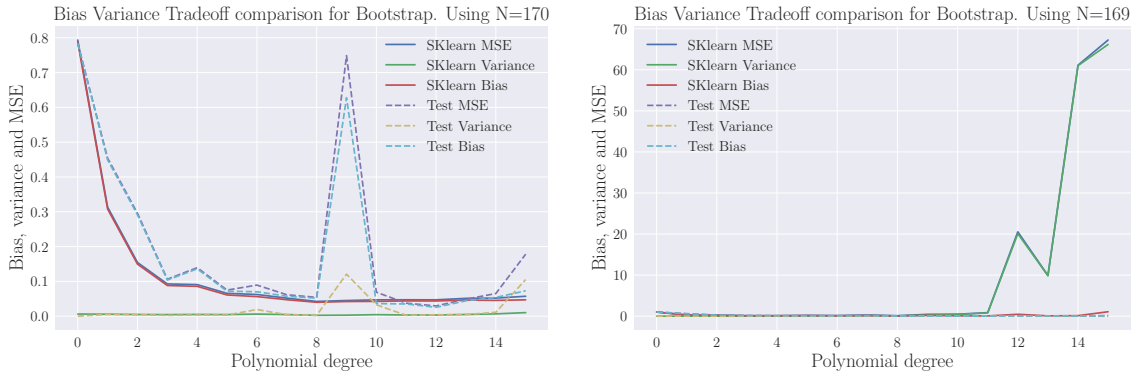

Figure 19. BV compare

For $N = 170$ data points, our model yields a low variance and a high bias and MSE at first, similar to scikit-learn, just as expected. For a polynomial degree of $P = 9$, we get an abrupt increase ouf our three metrics while scikit-learn remains stable. For $P = 15$ we see that our variance and MSE starts to increase, while the bias remains low, just as desired. This feature is less present for the scikit-learn. If we compare with $N = 169$, as shown on the right panel in figure 19, we see that the variance and MSE of the scikit-learn model exceeds a value of 60 for $P > 14$, which is not present in our model. This shows that the bias variance trade-off is generally sensitive to our particular configuration, so when performing a bias-variance trade-off analysis on the Franke function our result may be correct, despite odd behaiour for certain polynomial degrees.

---

[1] Ref.