Project 1

Course: FYS-STK4155

Semester: Autumn 2020

Name: Sander Losnedahl

Preliminaries

This report will cover a few important, but necessary steps in the regression scheme before presenting the results form the sub-exercises. Each of these steps are implemented in all the exercises and will ultimately yield a more reliable result.

Scaling the data is necessary in order to make sense of the output values. If the data is unscaled, the resulting numbers will not make much sense as we have no reference to what are low and high numbers. When we scale the data we subtract the sample mean and divide by the sample standard deviation of the data, we essentially make the data centred around zero with standard deviation one. This way, we always have a reference to the outcome as it is always relative to zero. Additionally, most regression algorithms rely on the data having lower distance between the data points, making scaling an important pre-processing step.

The number of observations n has large impact on the regression algorithm and the results from it. Typically, having more data (larger n) results in a better estimation of the response. In fact, some algorithms like ordinary least squares rely on the number of observations to be greater than the polynomial degree p of the design matrix. Moreover, algorithms like ordinary least squares work best when $n >> p$. In this project, we will initially experiment with different sizes of n, but later exercises will use $n = 100$.

Noise level is a recurrent theme throughout this report. Noise is added to the design matrix itself (sub exercise a-e) in order to prepare for the real data. The noise that is added has a standard normal distribution with standard deviation 1 and maximum value of 1 at its mean of zero. This noise is then multiplied with a constant of 0.001 in order to adjust the noise level.

The train/test split will be 75/25 in this project. When performing regression, we need to both train the algorithm using the training set, and later validate the trained algorithm using the independent test set. There is no set ratio which is considered the best, but the training set should include the majority of the observations n.

Exercise 1.

a) The goal for this part of the exercise is trying to fit a linear model (linear in terms of regression coefficients) to the Franke function given as

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

where polynomial combinations of x and y in the span $x, y \in [0,1]$ will be the explanatory variables. The linear regression equation then takes the form

$$\widehat{f(x,y)} = \hat{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2}$$

where $\widehat{f(x,y)}$ or $\hat{y}$ is the least squares estimate of the Franke function, $\mathbf{X}$ is the $n \times p$ design matrix consisting of the aforementioned polynomial combinations of x and y, $\boldsymbol{\beta}$ are the $p \times 1$ regression coefficients and $\boldsymbol{\epsilon}$ is just random noise/unobserved random variables. In order to get the best estimate for the Franke function, we want to choose $\boldsymbol{\beta}$-values so that we minimize the residual sum of squares (thereby the name "least squares"). Solving equation 2 with the intent to minimize the residual sum of squares yields the estimates for the $\boldsymbol{\beta}$-values

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T f(x,y) \tag{3}$$

where T marks the transpose of the matrix. We can now get the estimate $\hat{\boldsymbol{\beta}}$ with equation 3 and then compute the estimate of the Franke function as seen in equation 2.
We now turn to implementing the regression algorithm (see github) where we start by selecting a low number of observations of $n = 10$, a polynomial degree of $p = 5$ and a noise level of 0.001. We start by creating the design matrix using polynomials of x and y (and their combinations) of degrees up to 5, scale it, split it into training and test set with ratio 75/25 and lastly estimate the regression coefficients and calculate the estimated Franke function. The regression coefficients are then found, but it is not always certain which values of beta coefficients minimize the residual sum of squares, so some uncertainty is always present. Therefore, we need to find which coefficients are uncertain, and which are certain. We quantify the uncertainty by using confidence intervals around the coefficients such that our uncertainty is within one standard deviation of the coefficient. We find said standard deviation by taking the square root of the diagonal elements of the covariance matrix

$$\sigma = \sqrt{diag((\mathbf{X}^T\mathbf{X})^{-1}S^2)} \tag{4}$$

where X is the design matrix and S is the sample variance of the response. By then taking $\beta_i \pm \sigma_i$ we can find the confidence interval of the regression coefficients which is plotted in figure 1

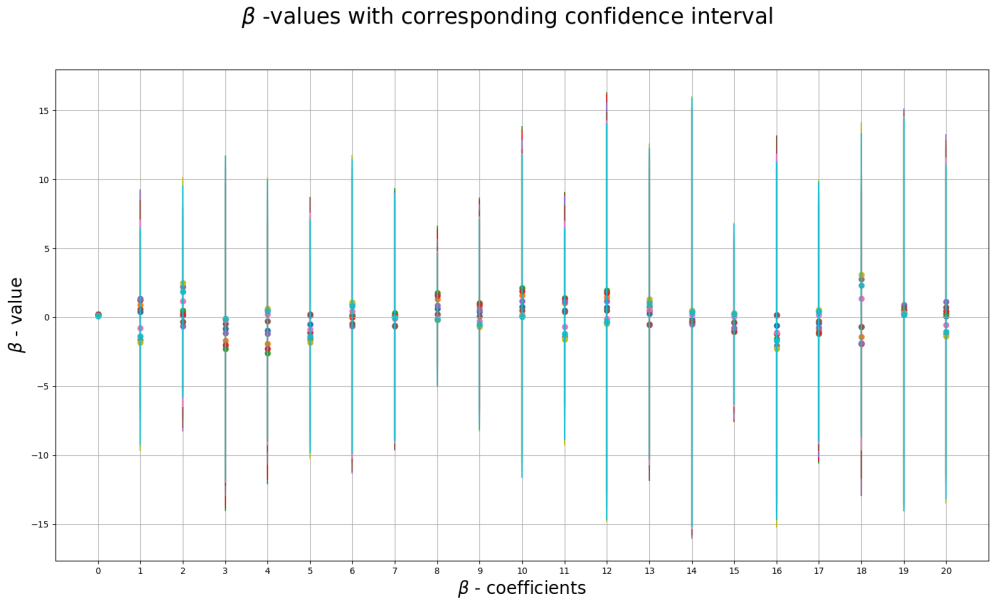

Figure 1: $\hat{\boldsymbol{\beta}}$-coefficients from performing ordinary least squares regression. Dots indicate the actual $\hat{\boldsymbol{\beta}}$-coefficients value while bars around indicate the confidence interval ($\pm\sigma$).

Since the response is the Franke function (a matrix), we get p times n regression coefficients instead of p, as seen in figure 1. However, one can still observe from figure 1 that some regression coefficients are more certain than others. We can also plot one slice of the p times n regression coefficient matrix to get a better view as seen in figure 2

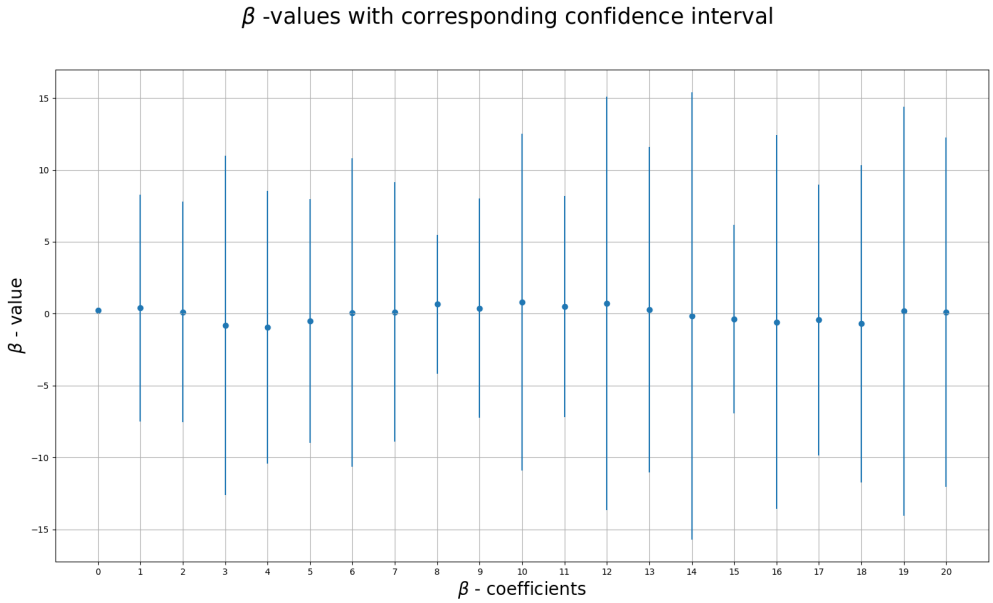

Figure 2: A slice of the $\hat{\boldsymbol{\beta}}$-coefficients from performing ordinary least squares regression. Dots indicate the actual $\hat{\boldsymbol{\beta}}$-coefficients value while bars around indicate the 95% confidence interval ($\pm\sigma$).

Now that we have calculated and gained some faith in our regression coefficient estimates,

we can utilize equation 2 to find $\hat{y}$ which is plotted together with the real Franke function in figures 3 and 4

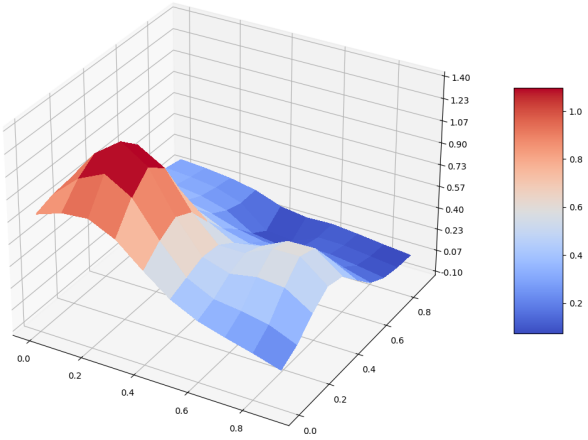

Figure 3: The real Franke function when we have 10 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

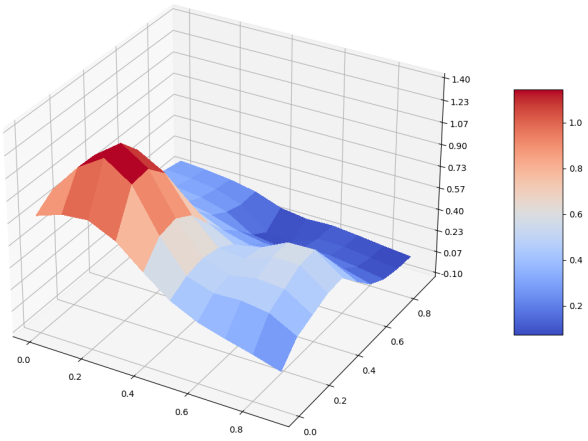

Figure 4: The estimated Franke function when we have 10 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

It can be observed from figures 3 and 4 that our estimate is pretty good, any we can plot the difference between the two to strengthen faith in the model

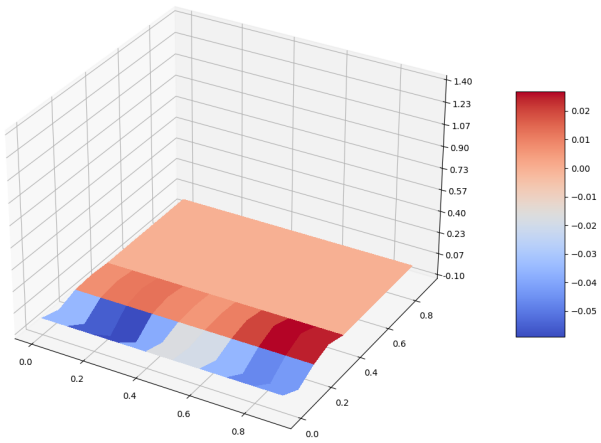

Figure 5: The difference between the real and estimated Franke functions when we have 10 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

As expected, the difference shown in figure 5 is very small. However, we have yet to quantify how small. This can be done using the mean square error (MSE) and R^2 which is calculated from equations 5 and 6

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \tag{5}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y}_i)^2} \tag{6}$$

where $\bar{y}_i$ is the mean value of the Franke function. The MSE gives us a value of how far our estimate falls from the real Franke function, while the R^2 gives us how strong the relationship is between the real Franke function and our estimate. When utilizing equations 5 and6 on the data shown in figures 3 and 4 we get the values in table 1

Table 1: MSE and R^2 between the real and estimated Franke function when we have 10 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

	MSE	R^2
Training	$2.4978127439196968 \times 10^{-24}$	1.0
Test	0.003919934844588911	0.9425989967371673

It should be obvious that the training set performs better than the test, as it is the training data that is used to fit the model, particularly when n is small. So what happens when we increase the number of observations to say $n = 100$? Figures QQQ, QQQ and

QQQ show again the real, estimated and differential Franke functions with the same exact parameters, except for n, which is now equal to 100

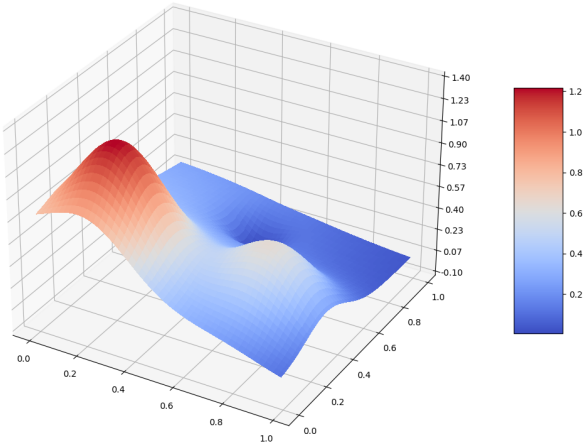

Figure 6: The real Franke function when we have 100 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

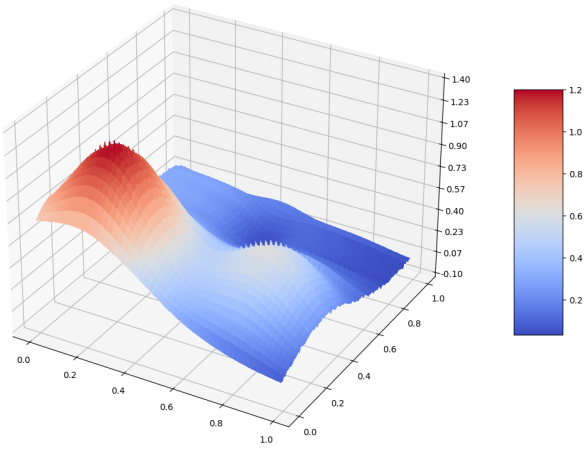

Figure 7: The estimated Franke function when we have 100 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

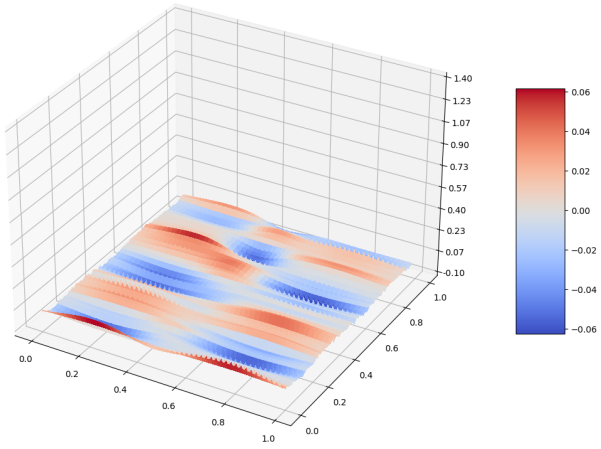

Figure 8: The difference between the real and estimated Franke functions when we have 100 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

We first observe that the figure is much smoother than the previous ones, as the function is discretized by n. We can again get a better understanding of these plots by finding the MSE and R^2 as shown in table 2

Table 2: MSE and R^2 between the real and estimated Franke function when we have 100 observations and polynomials of degree 5 using a noise-level of 0.001 and a 75/25 train/test split.

	MSE	R^2
Training	0.000565152580116653	0.9935076041084075
Test	0.0008407777639833455	0.9865450620944748

At first glance, one may panic as the training MSE and R^2 is lower than for $n = 10$. However, this value is not interesting as it is the test MSE that dictates how well the model performs. This is because we are only interested in how well our model is in predicting new data, which it has not trained on, and as we can see from table 2, both the test MSE and test R^2 is much higher with $n = 100$ than with $n = 10$. Therefore it is safe to say that increasing the number of observations drastically increases the predictive ability of the model.
Furthermore, we can compare figure 9 and figure 1 to see that the confidence of our regression coefficients increase as we increase n. This is actually the reason why the model improves.

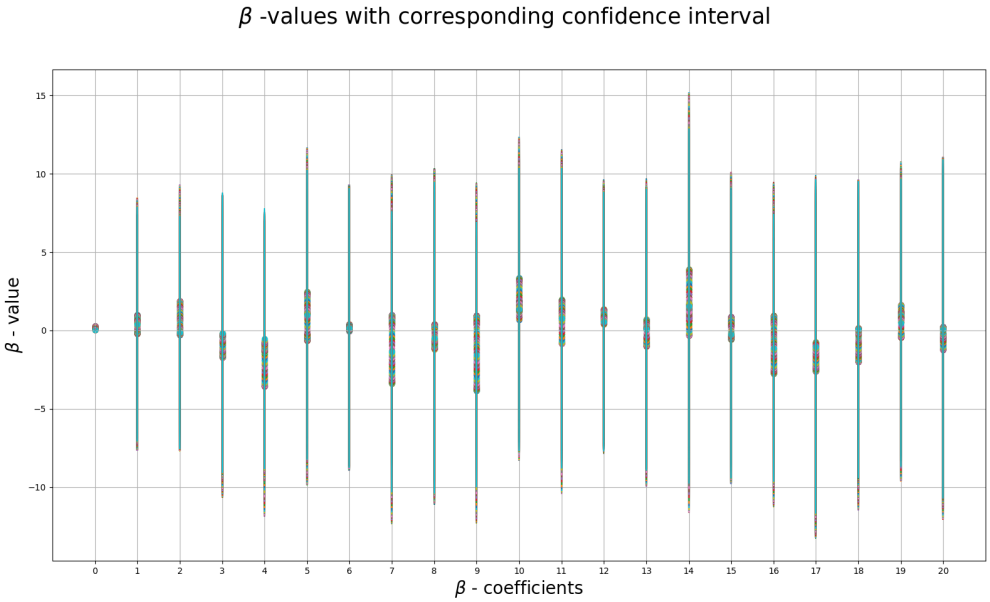

Figure 9: $\hat{\boldsymbol{\beta}}$-coefficients from performing ordinary least squares regression. Dots indicate the actual $\hat{\boldsymbol{\beta}}$-coefficients value while bars around indicate the confidence interval ($\pm\sigma$).

We can also see what happen if we were to increase the noise-level from 0.001 to 0.1 using $n = 100$ observations in figures 10

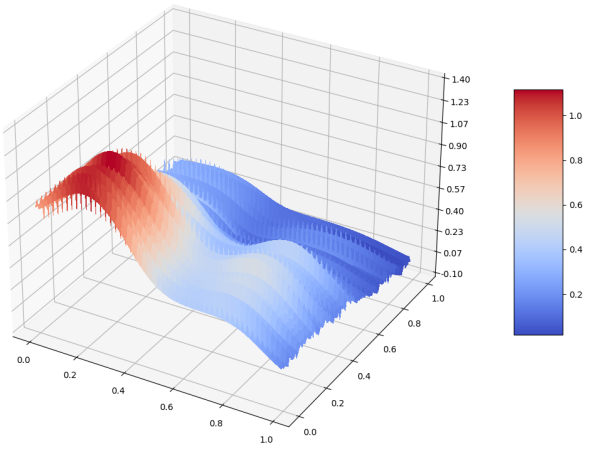

Figure 10: The estimated Franke function when we have 100 observations and polynomials of degree 5 using a noise-level of 0.1 and a 75/25 train/test split.

As expected, the noise makes the model worse at predicting $\hat{y}$. However, the overall traits of the Franke function can still be observed, but the errors shown in table 3 indicates that the performance is worse with more noise

Table 3: MSE and R^2 between the real and estimated Franke function when we have 100 observations and polynomials of degree 5 using a noise-level of 0.1 and a 75/25 train/test split.

	MSE	R^2
Training	0.0075270297407231375	0.9044112376567826
Test	0.009241122769969749	0.9012627398304937

We could easily remove all the noise from the mode, but we intend to keep a noise level of 0.001 as it will prepare us for the real data later in the project.

b) When we fit a linear model like in the last exercise, we always want to minimize the mean square error (MSE) of the test set. We can explain the MSE from equation 5 in terms of the expected value of its such that the MSE is simply the expected value of the difference between the actual response and the predicted response using regression. This mean we can write the MSE as $E[(y-\hat{y})^2]$ where y is the actual response while $\hat{y}$ is the predicted response (as we already know). By adding and subtracting the term $E[\hat{y}]$ to the inner bracket we can expand the MSE like in equation 7

$$\begin{aligned} MSE = E[(y-\hat{y})^2] = E[(y-\hat{y}+E[\hat{y}]-E[\hat{y}])^2] \\ E[(\hat{y}-E[\hat{y}])^2 + 2(\hat{y}-E[\hat{y}])(E[\hat{y}]-y) + (E[\hat{y}]-y)^2] \\ E[(\hat{y}-E[\hat{y}])^2] + E[2(\hat{y}-E[\hat{y}])(E[\hat{y}]-y)] + E[(E[\hat{y}]-y)^2] \end{aligned} \tag{7}$$

Since the expected value of $\hat{y}$ equals y when n is large, we can write $E[\hat{y}]-y = \text{constant}$ and thus

$$\begin{aligned} E[(\hat{y}-E[\hat{y}])^2] + 2(E[\hat{y}]-y)E[\hat{y}-E[\hat{y}]] + (E[\hat{y}]-y)^2 \\ E[(\hat{y}-E[\hat{y}])^2] + 2(E[\hat{y}]-y)(E[\hat{y}]-E[\hat{y}]) + (E[\hat{y}]-y)^2 \\ E[(\hat{y}-E[\hat{y}])^2] + (E[\hat{y}]-y)^2 \end{aligned} \tag{8}$$

where the constant $E[\hat{y}]-E[\hat{y}]$ equals zero, making the whole term equal zero.
We recognize the term $E[(\hat{y}-E[\hat{y}])^2]$ as the variance of estimator $\hat{y}$ and the term $(E[\hat{y}]-y)^2$ as the bias of the model, but squared. The bias quantifies how well the model fits the data points and variance is how well the model would translate to other data the model is not trained on. Increasing one tends to decrease the other, but not linearly since the bias is squared. Therefore, one can decrease the bias to a certain extent, but at some point, the loss of bias is not worth the gain in variance. This is called the bias variance trade-off.
Before investigating the bias-variance trade-off any further, we need to solve a problem related to the lack of data that is often the case with real data. When we generate data synthetically, the lack of data is never a problem, but in the real world, data is always limited and we now want to simulate that experience. Imagine now that we only have 100 observations and that these 100 observations are enough to represent the underlying distribution of the data, but not enough for our machine learning algorithm to function properly. We can then utilize the bootstrap resampling method to create data out of thin air. The bootstrap resampling method aims to take the original data of size n and create B new data sets each of size n. This is done by randomly assigning observations from the original data set to the B new data sets (with replacement). Then, a statistic is calculated for each of the B new data sets and the mean of all these statistics should represent the statistic of the original data set. The new statistic is often called the bootstrap statistic and will in this report be the MSE. Figure 11 shows a rough sketch of the process

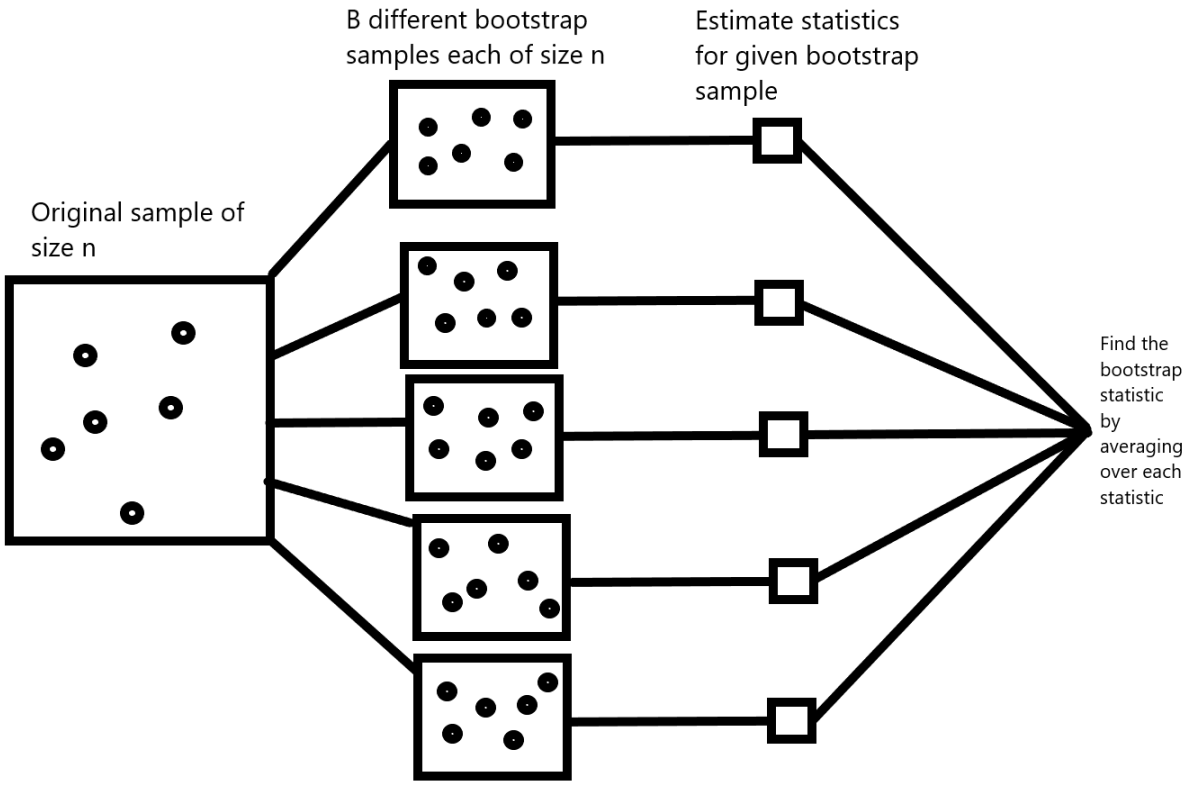

Figure 11: Illustration of the bootstrap process. The statistic in question here is the MSE.

The hope of the bootstrap method is that the bootstrap statistic represents the statistic of the underlying distribution of the original data set, regardless of the original datasets distribution. Be aware, the bootstrap method does not create new information, but simply exaggerate the already existing information, letting us perform operations like linear regression more easily.
Now that we have a method of creating more data, we can study the bias-variance trade-off more thoroughly. More spesifically, we want to study how the bias, variance and MSE changes as a function of polynomial degree. Now imagine we just have 100 observations. We can then utilize the bootstrap resampling method to create $B \times n$ more data. Let us generate $B = 100$ data sets each of size n and see how the our parameters of interest change with polynomial degree as seen in figures 12 and 13

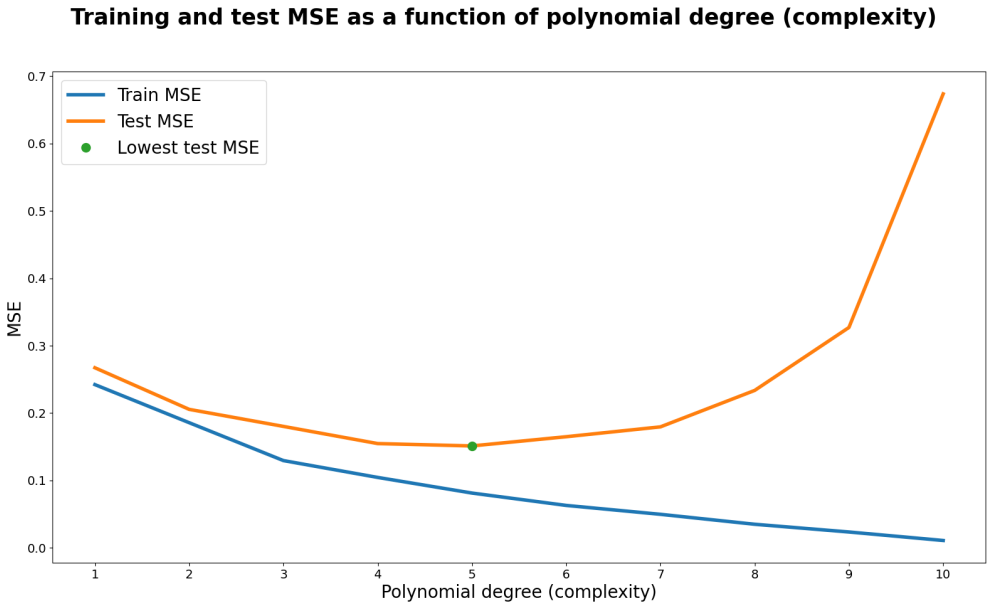

Figure 12: MSE as a function of polynomial degree using the bootstrap resampling method. Here we have 100 observations while we generate 100 bootstrap samples with a noise level of 0.001 and a 75/25 train test split.

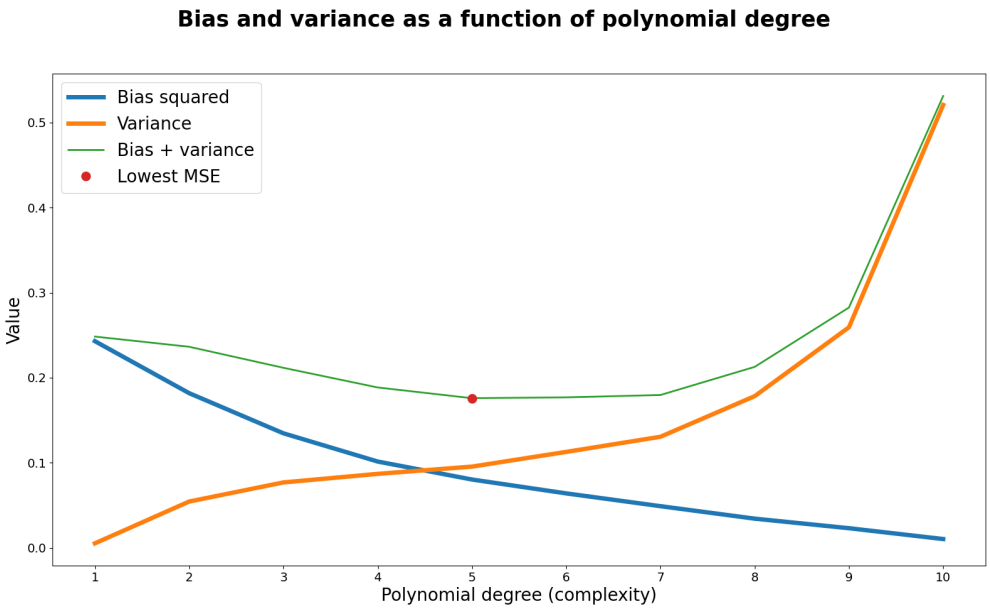

Figure 13: Bias and variance decomposition of figure 12. The bias and variance change as a function of polynomial degree using the bootstrap resampling method. Here we have 100 observations while we generate 100 bootstrap samples with a noise level of 0.001 and a 75/25 train test split.

We can observe that the bias starts high, but tends towards zero when we increase the complexity of our model. This is because a low order polynomial would be unable to fit the complex structure of the Franke function, thus making the difference between our predicted Franke function and the actual Franke function large. With an large increase

in polynomial degree, we can fit the model arbitrarily close to the actual Franke function. The variance behaves almost the opposite of the bias as it starts very low, but increases exponentially as we increase the complexity. This is because a low order polynomial would manage to somewhat predict the test data, even if the prediction is horrible. When we increase the complexity, the model is unable to accurately predict the test data because the model is over fitting the model to the data of the training set.
We can also see that the minimum MSE in figure 12 corresponds to the same polynomial as that of figure 13, strengthening our trust in the models predictive capabilities, but what would happen if we were to decrease the number of bootstrap samples? Let us now generate a new model where all the parameters are the same as of the previous one, but where we only generate 10 bootstrap samples. Then we get something akin to figure 14

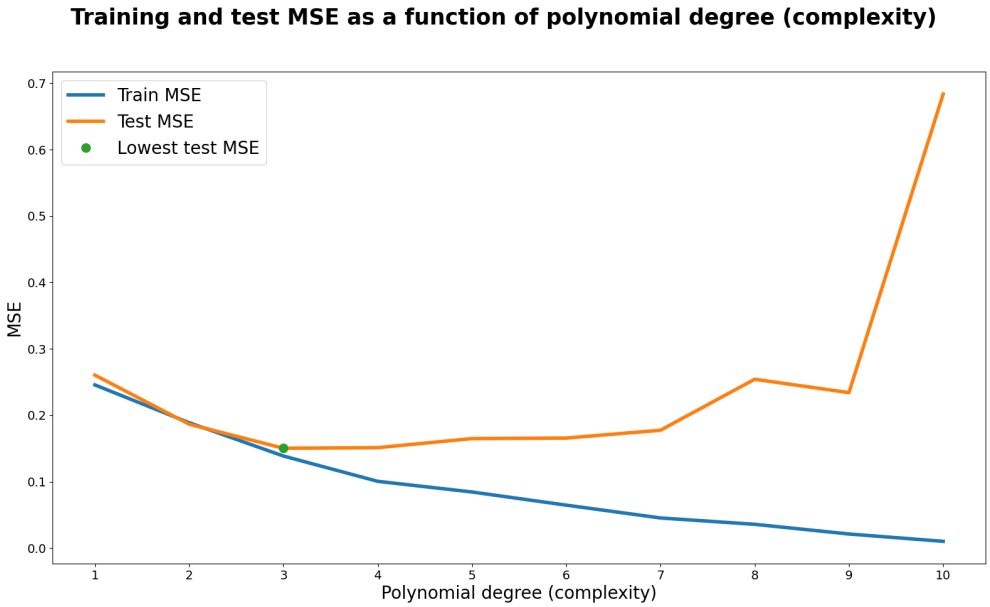

Figure 14: MSE as a function of polynomial degree using the bootstrap resampling method. Here we have 100 observations while we generate 10 bootstrap samples with a noise level of 0.001 and a 75/25 train test split.

We can observe from figure 14 that the point of lowest MSE is now at $p = 3$ which is lower than the one observed in figure 12. This is because more data means that the model can be trained more without overfitting the model. Thus, when we decrease the number of observations (or bootstrap samples in this case), we lower the threshold for when the model becomes overfit. This again means that a lower degree of polynomial is the ideal complexity for the model.
What is also observed from the code when using a low amount of bootstrap samples is that the model becomes increasingly unstable. This is because the bootstrap MSE relies on the mean of the B generated statistics. If we only generate a few bootstrap samples, the mean is more prone to randomness, which in turn means the algorithm is unstable. For this reason and the one described above (more data is good), we want to stick with 100 bootstrap samples from here on out.

c) We will now introduce another resampling method called cross-validation (CV) and repeat the analysis of the previous exercise. The cross-validation resampling method, like the bootstrap, also aims to create more data out of thin air. However, the process is quite different than that of the bootstrap. The main idea behind the CV is to divide the original data set into k different data sets, or folds as they are called, and then let the kth fold act as the test set while the other k-1 folds acts as the training sets. A model is then fit for each permutation and a statistic is calculated. Then, another fold will act as the test set while the others will act as the training set. This process repeats until every fold has been used as a test set and a statistic for each permutation is calculated. The average of the k different statistic is then calculated and is said to represent the actual statistic of the original data set.
It is typical to have 10, 5 or n different folds and the CV process is then referred to as 10-fold, 5-fold and leave one out (LOOCV) cross-validation methods. The LOOCV is a special case of the CV method where each observations acts as the test set once and this process is sketched in figure 15

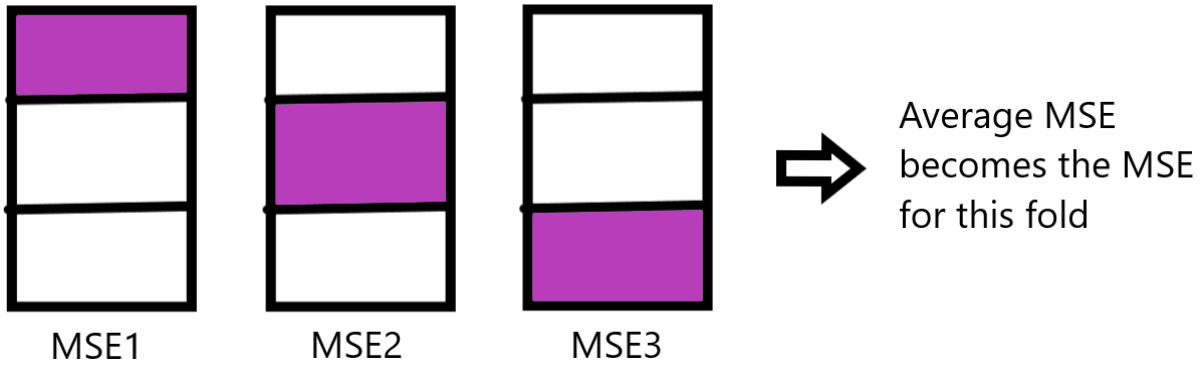

Figure 15: A rough sketch of the LOOCV process. The n observations acts as the test set once (purple squares) while the other $n-1$ observations acts as the training set (white squares). The model is then fit on each distinct permutation and a statistic is calculated (MSE in this project). Finally, an average of the k different statistics is calculated and will represent the statistic of the original data set.

We now want to repeat the process from the previous exercise where we see look at MSE as a function of polynomial degree for the 5-fold, 10-fold and leave one out cross-validations. Let us start with the 10-fold CV which is plotted in figure 16

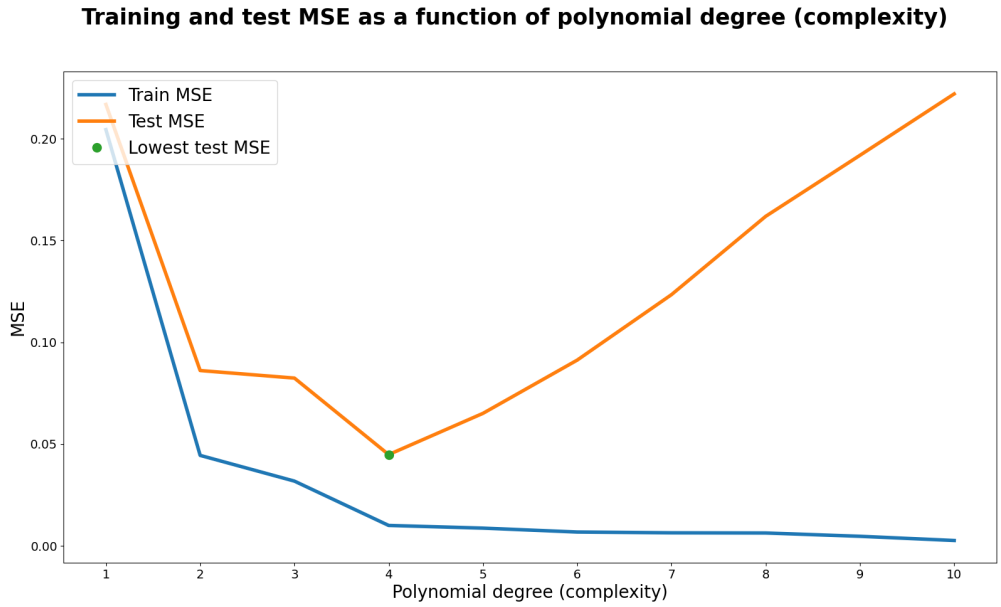

Figure 16: MSE as a function of polynomial degree up to 10 using the cross-validation resampling method. Here we have 100 observations while we split the data into 10 different folds with a noise level of 0.001.

We can observe from figure 16 that the MSE is at its lowest for $p = 4$, similar to the MSE for the bootstrap resampling method in figure 12. We can decompose the MSE into its bias and variance components as done in figure 17

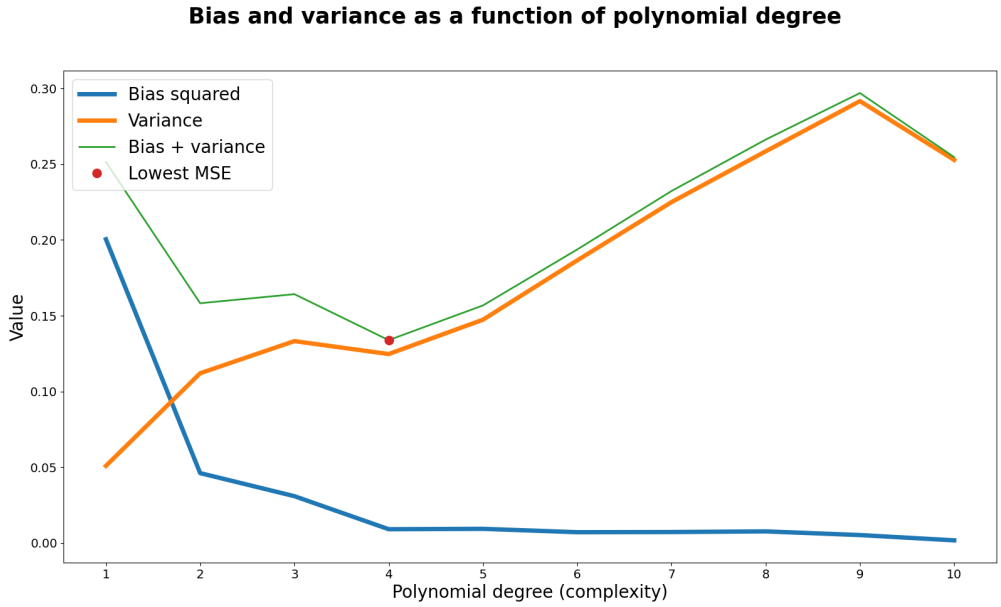

Figure 17: Bias and variance decomposition of figure 16. The bias and variance change as a function of polynomial degree using the CV resampling method. Here we have 100 observations while we use 10-fold CV with a noise level of 0.001.

It can be confirmed from figure 17 that the lowest MSE is found at $p = 4$ polynomial

degrees. The bias and variance behaves similar to that of bootstrap as seen in figure 13. We now want to see what happens when we set the number of folds k equal to 5 and the result is shown in figures 18 and 19

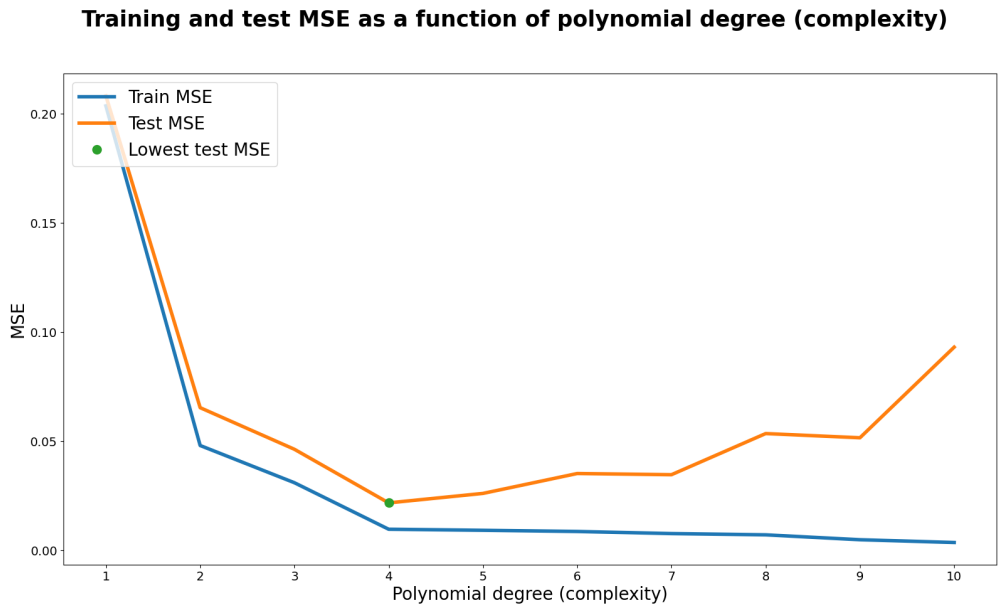

Figure 18: MSE as a function of polynomial degree up to 10 using the cross-validation resampling method. Here we have 100 observations while we split the data into 5 different folds with a noise level of 0.001.

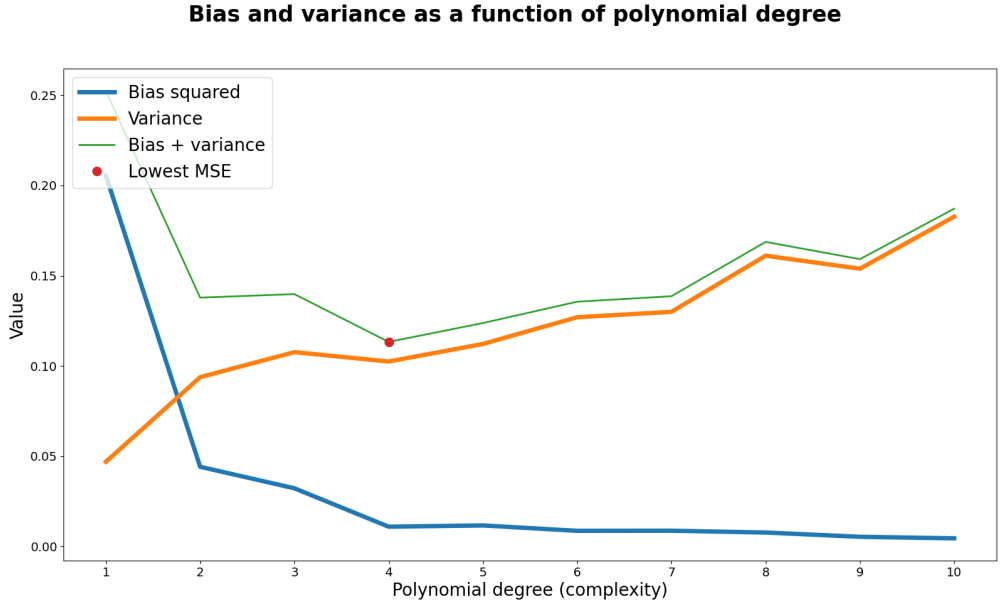

Figure 19: Bias and variance decomposition of figure 16. The bias and variance change as a function of polynomial degree using the CV resampling method. Here we have 100 observations while we use 5-fold CV with a noise level of 0.001.

What we observe when comparing figure 16 to figure 18 and figure 17 to figure 19 is

that the bias looks the same, but the variance increases at a lower rate when using 5-fold CV rather than 10-fold CV. As a result, the MSE is lower at higher polynomials, thus decreasing the risk of overfitting. We can continue this examination by seeing what happens when we use LOOCV as in figure 20 and 21

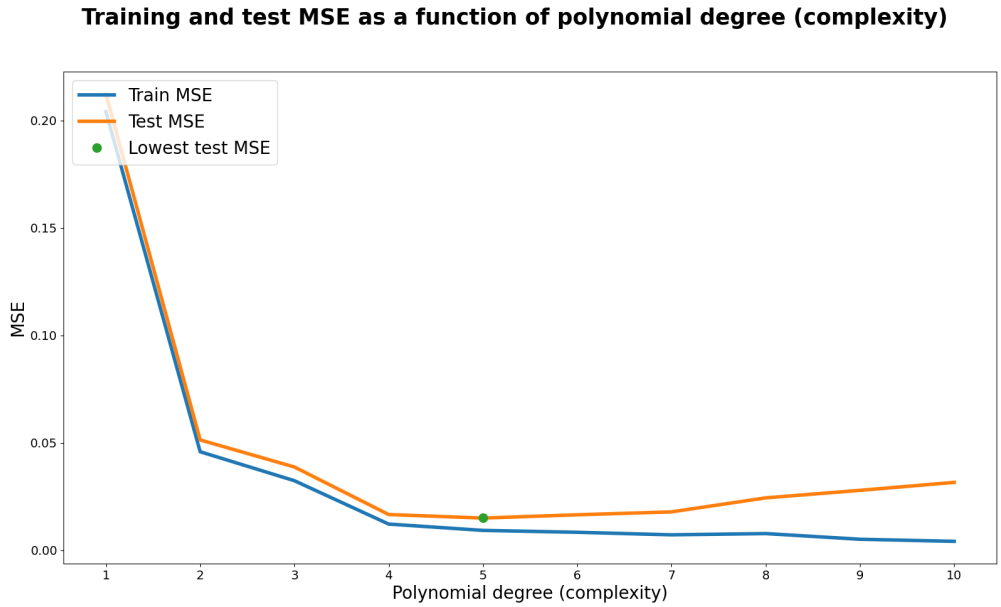

Figure 20: MSE as a function of polynomial degree up to 10 using the cross-validation resampling method. Here we have 100 observations while we split the data into n different folds with a noise level of 0.001.

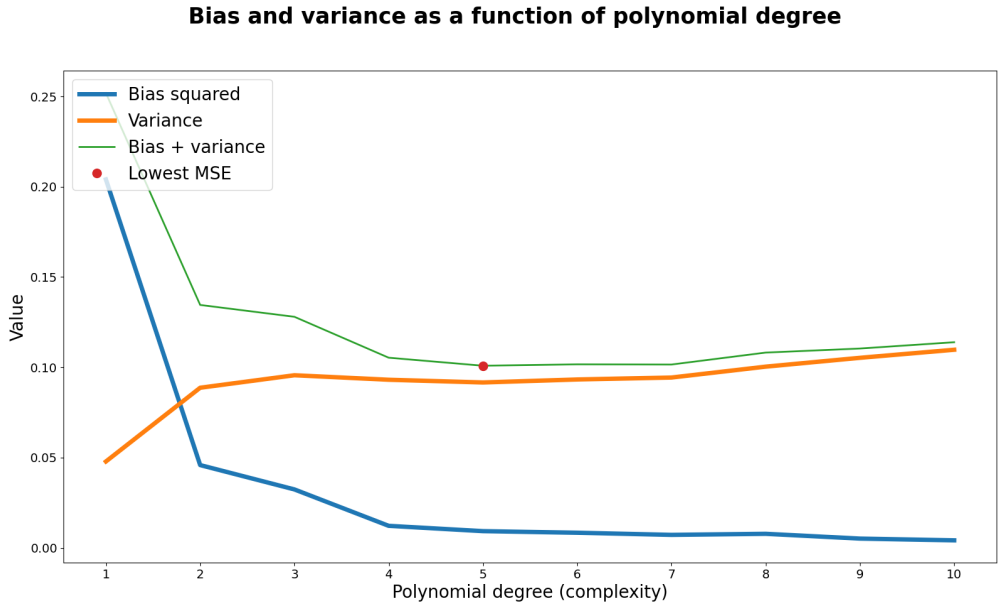

Figure 21: Bias and variance decomposition of figure 16. The bias and variance change as a function of polynomial degree using the CV resampling method. Here we have 100 observations while we use LOOCV with a noise level of 0.001.

Like previously discussed, the slope of the variance is less steep than that of the 10 and 5-fold CVs, and the MSE is therefore lower at higher polynomial degrees. The reason for this effect is that decreasing the fold size "creates" more data out of thin air, and having more data means that we could fit higher order polynomials without over-fitting, as previously discussed. This is observed from figures 20 and 21 as the lowest MSE is located at $p = 5$ instead of $p = 4$ as seen for higher folds. This effect could possibly also be achieved using bootstrap if we used more than 100 bootstrap samples. The two different methods seems to yield similar result which strengthen our believe in the resampling methods. This allows us to proceed to other linear regression schemes in the coming exercises.

d) We have only utilized the ordinary least squares (OLS) regression scheme so far, but now we want to utilize two different shrinkage methods, namely the Ridge and Lasso regression schemes. We will start by looking at the Ridge regression scheme and then proceed to Lasso in the next exercise.
Shrinkage methods like Ridge aim to find which variables are significant and which variables are not, and then removing the influence of the in-significant variables by either setting them to zero (Lasso) or close to zero (Ridge). This is done by adding the shrinkage factor λ to both sides of the OLS equation (equation 2) such that

$$\hat{\boldsymbol{y}}_{Ridge} + \lambda|\boldsymbol{\beta}|^2 = \mathbf{X}\boldsymbol{\beta} + \lambda|\boldsymbol{\beta}|^2 \tag{9}$$

Solving equation 9 for the regression coefficients $\boldsymbol{\beta}$ yields

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\hat{\boldsymbol{y}}_{Ridge} \tag{10}$$

Where $\mathbf{I}$ is the identity matrix. We can then repeat the analysis of exercises a through c, but this time using equation 10 to train the data.

References

- Reference