Project 1, Regression analysis and resampling methods

Sakarias Frette, Mikkel Metzsch-Jensen, William Hirst

October 5, 2021

The code for this project is at this Github adress. Please read the ReadME file before running the code.

Contents

1 Introduciton and general method

In this project we will study the Franke's Function f(x, y) in the domain $x, y \in [0, 1]$. Franke's function, which is a weighted sum of four exponentials, reads as follows

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

We generate data from this function by adding stochastic noise. In practice the noise is computed by drawing points from a normal distribution with mean $\mu = 0$ and

variance σ^2. We discretize the x,y data points in a $N \times N$ mesh-grid such that we get N^2 uniform distributed points in the space $x, y \in [0, 1]$. The target value z is then generated as follows

$$z(x, y) = f(x, y) + N(0, \sigma^2).$$

1.1 Scaling

For every data set one should consider whether it is nessecary to scale the data. Hence we investigate the behaviour of the Franke's function in our domain $x, y \in [0, 1]$. For a quick judgment we take a look at the 3D plot as showed in figure 1.

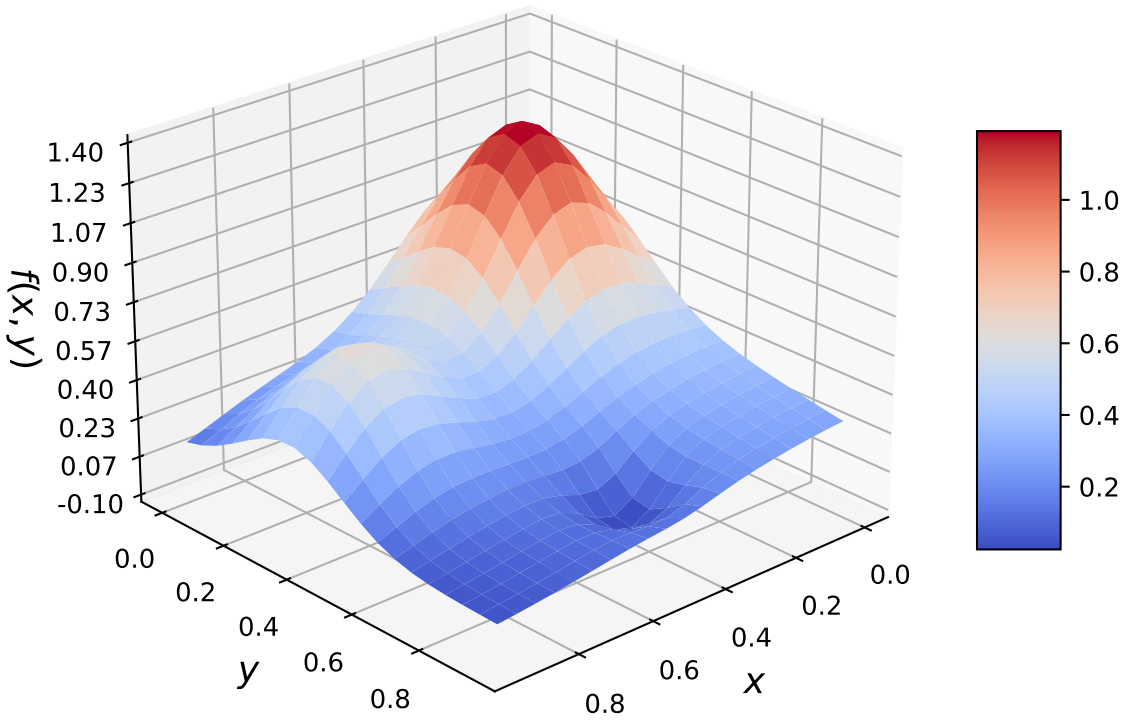


Figure 1: A 3D plot of Franke's function $f(x, y)$ in the relevant domain of $x, y \in [0, 1]$. For this plot we used a grid spacing of 0.05.

From figure 1 we see that our data is already quite "well-behaved". By making an even finer discretization than used in figure 1, with $dx, dy = 10^{-4}$, we compute the minimum and maximum value for z within our domain:

$$\min z = 0.011,$$
$$\max z = 1.220.$$

We observe that the target values are already very close to the [0,1]-range which is a common range to scale for. Since the x,y-values already lies within the [0,1]-range it should not be necessary to scale the data by any scaling-factors. In fact if we divide by the standard deviation, we will increase the variance in the dataset since the standard deviation is less than one in this case. However, since there might be some benefit of scaling by the mean (especially for Lasso regression), we shall do this consistently throughout the exercises involving Franke's function in order to compare the results. That is, we scale the design matrix and the target values by subtracting the mean. When generating predictions one should add back the mean value subtracted in the beginning. Note that we scale train and test data individually to avoid adding any further correlations.

1.1.1 Fitting of the function

In this project we will fit polynomials of different degree to Franke's function. Since we have two input variables (x, y) a polynomial of degree n will refer to every combination of $x^i y^j$ for which $i + j \leq n$. For a polynomial of $n = 2$ our model $\tilde{z}$ will read:

$$\tilde{z}(x, y)_{n=2} = \beta_0 + \beta_1 x + \beta_2 y + +\beta_3 x^2 + \beta_4 xy + \beta_5 y^2.$$

We implement our polynomial model in the form of a design matrix X. We assign each term from the polynomial to the columns X, such that we for the $n = 2$ case get

$$\mathbf{X}_{n=2} = \begin{bmatrix} 1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 \\ 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N-1} & y_{N-1} & x_{N-1}^2 & x_{N-1} y_{N-1} & y_{N-1}^2 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{bmatrix}.$$

Notice that the design matrix in general have dimensions $(N^2 \times (n+1)(n+2)/2$, as we have N^2 points in the meshgrid and $(n+1)(n+2)/2$ terms in our polynomial model. This means that we have to flatten the meshgrid of the x_i, y_j index-combinations into a vector suitable for the columns of the design matrix. Notice that the first column in the design matrix will be scaled to zero when subtracting the mean of every column, and thus this is effectively left out of the regression. By implementing the design matrix our polynomial model can be represented as

$$\tilde{z}(x, y) = \mathbf{X}\beta.$$

1.1.2 Splitting of data

When working with fitting of data it is essential and customary to distribute the data into training and test data. We choose to use a 80-20 split, meaning that we use 80%

of the data for our regression model (training) and 20% of the data, which is unseen for the model, to evaluate the model predictions (test).

1.1.3 Notation

We will not distinct matrices and vectors in the notation as we write both with bold letters throughout the paper. However, in many situations the bold letters just make the notations more cluttered and when left out we expect the presence of vectors and matrices to be clear from the context.

2 Exercise 1

First we perform Ordinary Least Squares (OLS) regression using polynomials up to fifth order ($n = 5$). In short, we define the OLS regression as a minimization of the cost function

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

By taking the derivative and setting it equal to zero, one can show that the optimal β-parameters is found as

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

In addition one will get that the variance of the β-parameters β_i is given as the diagonal elements

$$\text{diag}\Big(\text{var}(\hat{\beta}_0),\ \text{var}(\hat{\beta}_1),\ \ldots,\ \text{var}(\hat{\beta}_{n-1})\Big) = (\mathbf{X}^T\mathbf{X})^{-1},$$

for which we can calculate the standard error as $SE(\hat{\beta}_i) = \sqrt{\text{var}(\hat{\beta}_i)}$.

By the use of the the above relations we perform OLS with polynomials with $n = 5$ for a $N \times N$ grid with $N = 25$ (625 points) and added noise with $\sigma = 0.2$. Using the standard error (SE) for $\hat{\boldsymbol{\beta}}$, we can display the results with a 95% confidence interval (assuming $\hat{\boldsymbol{\beta}}$ is standard distributed). The values of $\hat{\boldsymbol{\beta}}$ and the confidence interval is displayed in both table 1 and figure 2.

Table 1: $\hat{\beta}$-values for OLS regression with a polynomial of degree $n = 5$ using $N = 25$ and $\sigma = 0.2$. The uncertainty is the result of a 95%-confidence interval.

Beta	Train uncertainty	Test uncertainty
0.00	$\pm$0.00	$\pm$0.00
5.65	$\pm$13.13	$\pm$29.47
1.81	$\pm$14.54	$\pm$22.35
-24.09	$\pm$66.03	$\pm$143.70
-4.26	$\pm$54.08	$\pm$104.16
0.34	$\pm$70.05	$\pm$128.41
29.92	$\pm$151.83	$\pm$325.09
18.02	$\pm$111.78	$\pm$246.34
0.98	$\pm$121.01	$\pm$198.55
-24.75	$\pm$156.09	$\pm$315.61
-8.95	$\pm$159.61	$\pm$347.41
-28.55	$\pm$120.98	$\pm$256.76
15.05	$\pm$115.41	$\pm$237.71
-12.25	$\pm$128.46	$\pm$234.38
42.30	$\pm$162.12	$\pm$341.77
-2.93	$\pm$62.38	$\pm$140.52
10.57	$\pm$54.03	$\pm$117.66
2.80	$\pm$52.20	$\pm$123.58
-13.29	$\pm$53.70	$\pm$118.11
10.94	$\pm$54.80	$\pm$126.30
-20.04	$\pm$63.13	$\pm$137.01

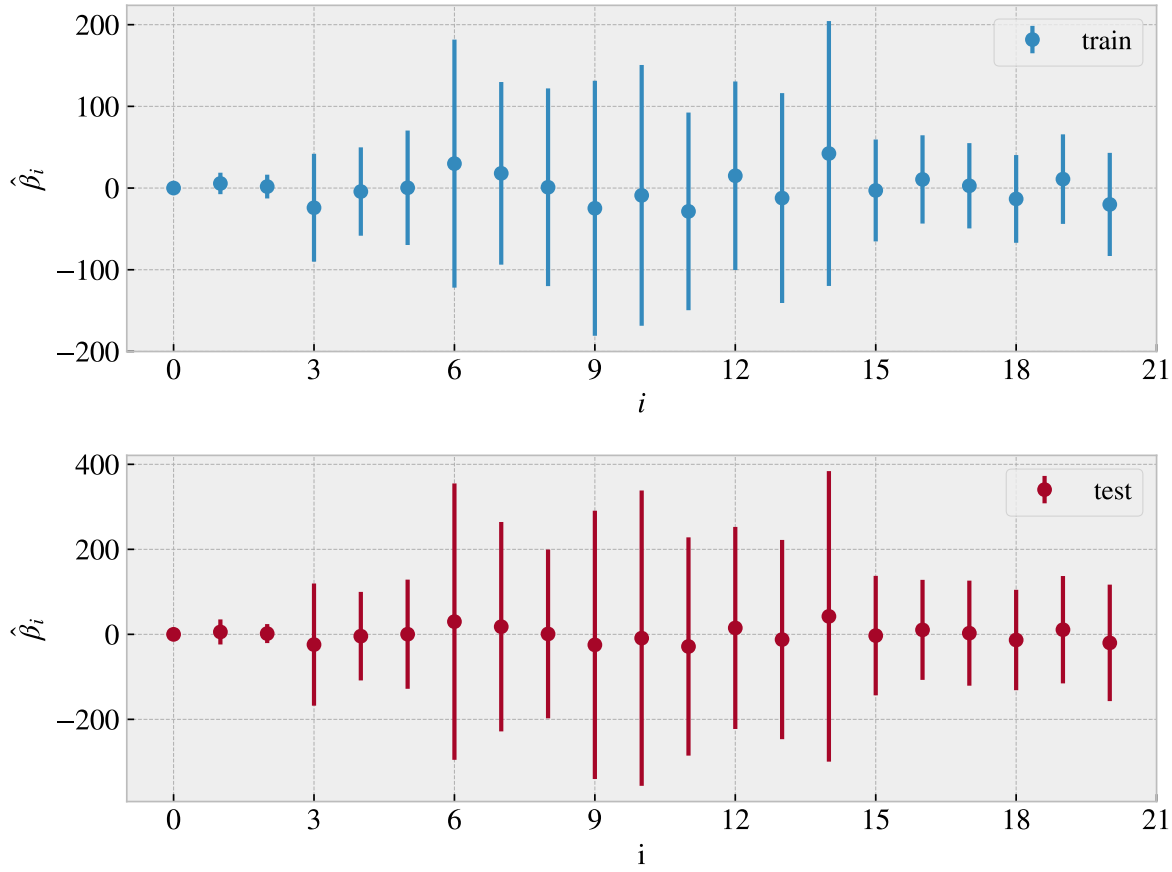


Figure 2: $\hat{\beta}$-values for OLS regression with a polynomial of degree $n = 5$ using $N = 25$ and $\sigma = 0.2$. The uncertainty is the result of a 95%-confidence interval.

From table 1 and figure 2 we see a noticeable difference in the uncertainty amplitude for different $\hat{\beta}_i$. Generally we have a bigger uncertainty for $\hat{\beta}_i$ with $i \approx [6, 14]$, meaning that these $\hat{\beta}$-values have a higher variance. Also, since the test data is four times smaller than the training the data, we see as expected that the uncertainty is greater for the test data prediction.

Lastly we can calculate the mean squared error (MSE) and R^2 score for our training and test data respectively as

$$\text{MSE}(\mathbf{z}, \tilde{\mathbf{z}}) = \frac{1}{N^2} \sum_{i=0}^{N^2-1} (z_i - \tilde{z}_i)^2,$$

$$R^2(\mathbf{z}, \tilde{\mathbf{z}}) = 1 - \frac{\sum_{i=0}^{N^2-1} (z_i - \tilde{z}_i)^2}{\sum_{i=0}^{N^2-1} (z_i - \mathbb{E}(z))^2}.$$

where $\mathbf{z}$ is the input data and $\tilde{\mathbf{z}}$ is the model prediction. The error results from the $n = 5$ OLS-regression is shown in table 2

Table 2: MSE and R^2 score for the test and train data on the OLS model. Using $n = 5$, $N = 25$ and $\sigma = 0.2$.

	Train	Test
MSE	0.045	0.050
R^2	0.64	0.50

As expected we see that the MSE for the training data is slightly better than MSE for the test data. This is because the model was trained to fit the train data and hence the fit is expected to be equal to or better than the fit on the test data.

3 Exercise 2

In this exercise we aim to study the bias-variance tradeoff by implementing the bootstrap resampling technique.

3.1 Error as a function of model complexity

Before diving into the analysis we compute the training and test MSE for OLS regressions as a function of the complexity n. This let us identify possible regions of low and high bias and variance respectively. The result are shown in figure 3.

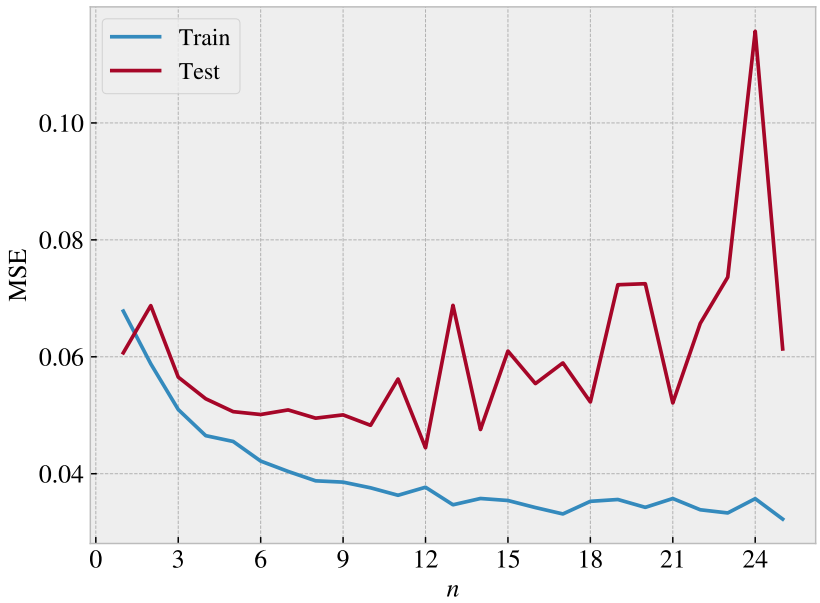


Figure 3: By performing OLS on the training data, we calculated the MSE between model predictions and input data for training and test data respectively. This was done as a function of the complexity n. We used $N = 25$ and $\sigma = 0.2$.

On figure 3 we see that the training MSE decreases steadily as the complexity increases, while the test MSE appears to be decreasing towards a minimum and then increasing again for higher complexities. This looks approximately similar to Fig. 2.11 of Hastie, Tibshirani, and Friedman, except for the fact that our results is a bit more rough. As expected the training MSE decreases as a function of the complexity since we make the model more and more adaptable to fit the training data. When the complexity gets to high, we will not only capture the general trend in the training data but also start fitting the model to the random noise applied to the training data. This does not translate well when using the model on the test data, and hence the predictions will start to deviate for higher complexities as we see. This is called overfitting. Judging from figure 3 the minimum test MSE sits around $n = 12$, which

indicate that a 12th-order polynomial is an appropriate model complexity to describe the general trend of Franke's function in our domain.

In order to create a less noisy results that match Fig. 2.11 of Hastie, Tibshirani, and Friedman even better, we can repeat the complexity analysis multiple times by generating new data points each time (apply new random noise). Notice that this is normally not possible with limited data. But it serves a great point of showing the general trend of training and test MSE respectively regarding overfitting. The result is shown in figure 4.

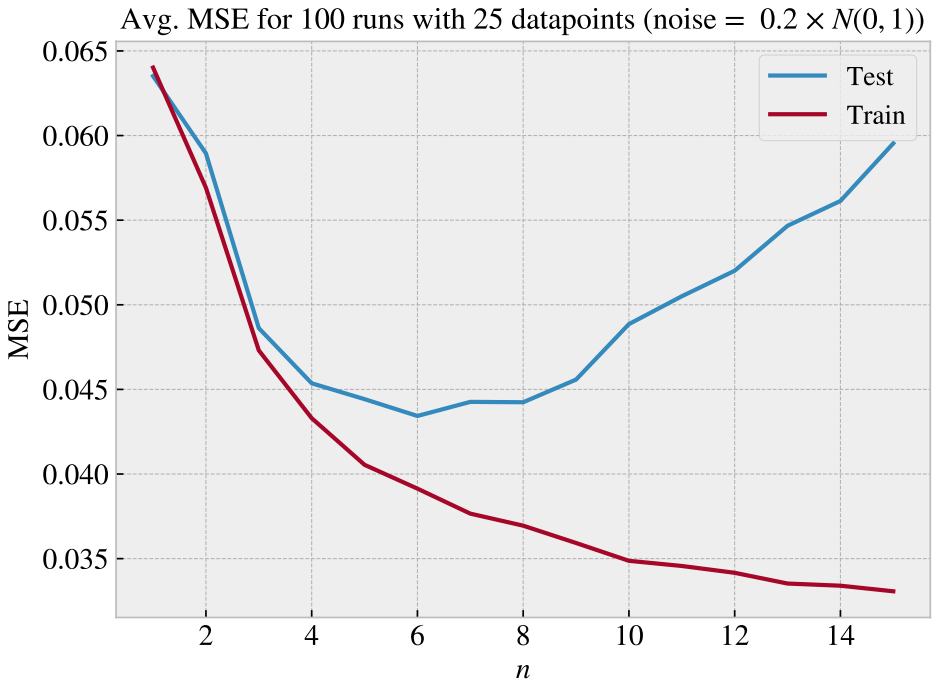


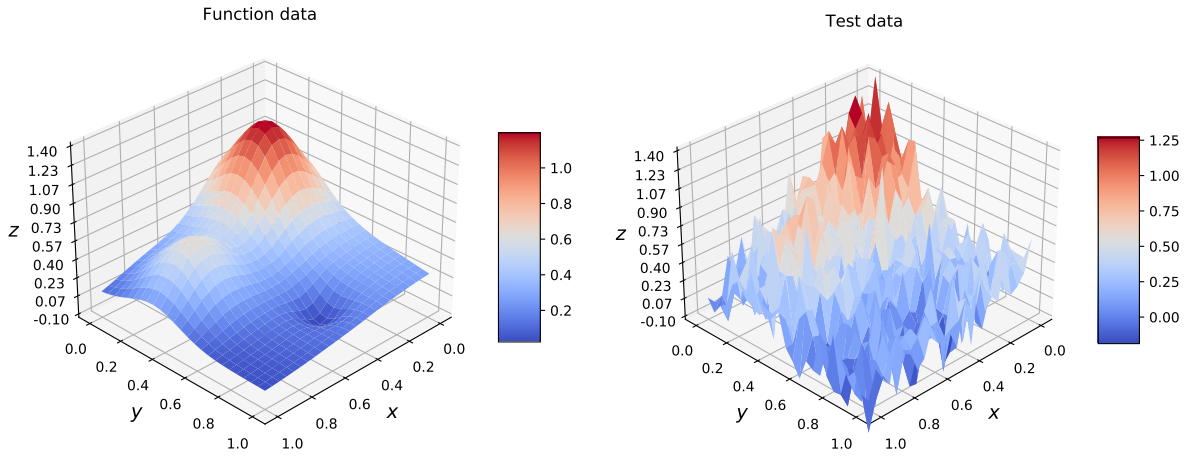
Figure 4: Average MSE for 100 repeatedly runs of the setup used for generating figure 3. This averaged version highlights the general trend of the training and test MSE as the complexity increases.

We see that the averaged result in figure 4 resembles Fig. 2.11 of Hastie, Tibshirani, and Friedman even better.

3.2 Visualization of the predictions

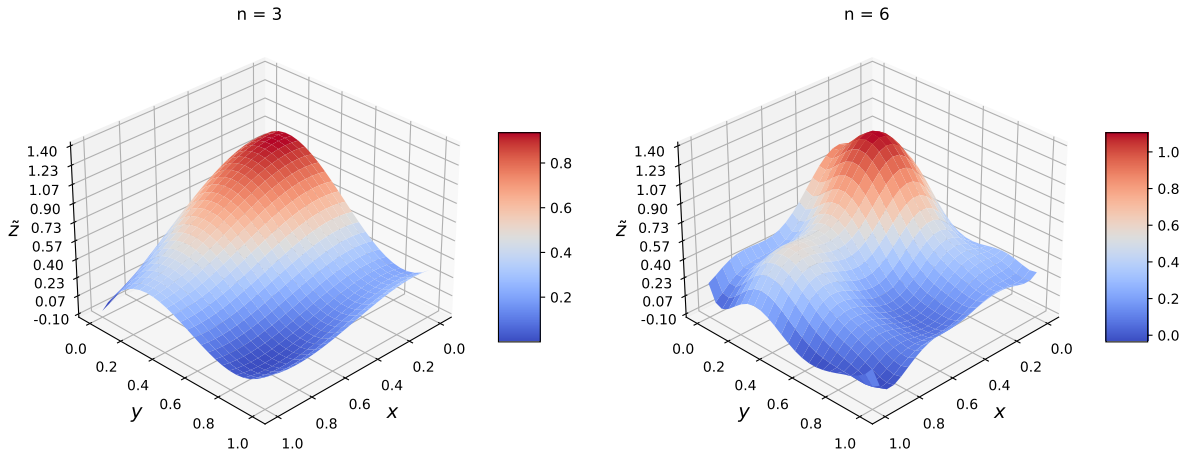
In order to verify visually that our predictions is reasonable we plot the predictions on selected order of complexities. Hence we generate two complete data sets: training and test, both on size $N = 25$ and with applied noise of $\sigma = 0.2$. We then train the model on the training data using OLS and plot the model predictions on the test data. The results are shown in figure 5. We see that the predictions converge fairly good towards the expected trends of Franke's function. In addition we see visually

that it agrees with the results on figure 4, which indicated that $n = 6$ might be the best complexity level. For $n = 20$ the prediction is clearly adding to many complex variations as a result to overfitting.



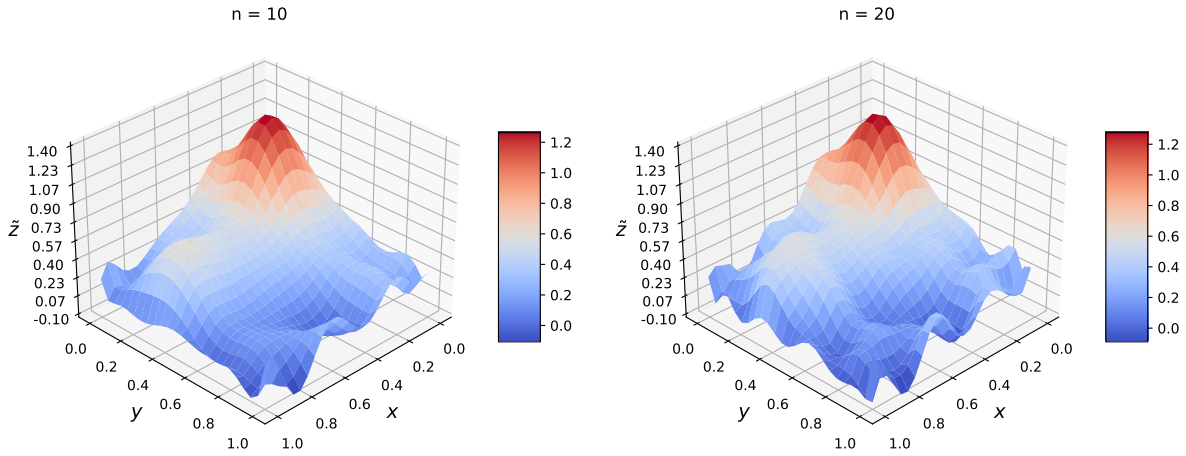
(a) Franke's Function $f(x, y)$

(b) Test data $z = f(x, y) + \epsilon$



(c) Prediction $\tilde{z}$ for $n = 3$

(d) Prediction $\tilde{z}$ for $n = 6$



(e) Prediction $\tilde{z}$ for $n = 10$

(f) Prediction $\tilde{z}$ for $n = 20$

Figure 5: Subfigure a and b show the Franke function with and without noise. Subfigures c through f show different OLS predictions for the Franke function with a given complexity.

3.3 Derivation of the Bias-Variance decomposition

We now move on to the Bias-Variance tradeoff analysis. We consider a general dataset $\mathcal{L}$ which consists of the datapoints $\boldsymbol{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{X}_j), j = 0, \ldots, n-1\}$. In this task we will assume that the true data is made from a model with normally distributed noise, with variance σ^2 and zero mean. Our model is thus

$$\mathbf{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}. \tag{1}$$

As previously stated, our model is given as $\tilde{\mathbf{y}} = \boldsymbol{X}\boldsymbol{\beta}$, where the optimal $\boldsymbol{\beta}$ is found as the minimization of the MSE cost function, which can also be written

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

We can decompose the MSE error into bias, variance and the noise variance. This is known as the Bias-Variance tradeoff. We have

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

By taking the expectation value of the components we get

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

Next we use that that the the noise ϵ is normally distributed with a mean equal to 0, and variance σ^2. Thus we have $\mathbb{E}[\epsilon] = 0$ and $\mathbb{E}[\epsilon^2] = \sigma^2$ which yields

$$\begin{aligned}
\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] &= \mathbb{E}[(f - \mathbb{E}[\tilde{y}])^2] + \sigma^2 + \mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y})^2] \\
&\quad + 2(f - \mathbb{E}[\tilde{y}])\,\mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y})].
\end{aligned}$$

Finally we have that the expectation value of an expectation value is the expectation value itself ($\mathbb{E}[\mathbb{E}(x)] = \mathbb{E}(x)$), which makes the last term in the above cancel out. Thus we get

$$\begin{aligned}\mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}})^2] &= \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\tilde{y}-\mathbb{E}[\tilde{y}])^2] + \sigma^2 \\ &= \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{y}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{y}])^2 + \sigma^2 \\ &= \quad \text{Bias}^2 \quad + \quad \text{Variance} \quad + \text{Noise variance.}\end{aligned} \tag{3}$$

Hence we arrived at the bias-variance relation. The decomposition shows that the MSE error can be explained in terms of

- Bias: A systematic error of the prediction.
- Variance: How much the predictions vary/fluctuates.
- Noise variance: The contribution from the noise.

However, in practice we don't know f_i (the unnoised data points) which is used to compute the bias. Thus we have to substitute f_i with our noised data points y_i in the calculation of bias. We can calculate the effect of this substitution as

$$\begin{aligned}\mathbb{E}[(y-\mathbb{E}[\tilde{y}])^2] &= \mathbb{E}[(f+\epsilon-\mathbb{E}[\tilde{y}])^2] \\ &= \mathbb{E}\left[\left(\epsilon + (f-\mathbb{E}[\tilde{y}]\right)^2\right] \\ &= \mathbb{E}[\epsilon^2] + \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[-2\epsilon(f-\mathbb{E}[\tilde{y}])] \\ &= \sigma^2 + \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[-2\epsilon]\,\mathbb{E}[f-\mathbb{E}[\tilde{y}] + \text{Cov}(-2\epsilon, f-\mathbb{E}[\tilde{y}]) \\ &= \sigma^2 + \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] = \sigma^2 + \text{bias}^2.\end{aligned}$$

From the above calculation it turns out that we can achieve the correct bias calculation using using y as

$$\text{bias}^2 = \mathbb{E}[(y-\mathbb{E}[\tilde{y}])^2] - \sigma^2.$$

3.4 Bias-Variance analysis

Now we will perform the Bias-Variance analysis on our Franke's function dataset. For this we are going to use the bootstrap resampling technique to generate the necessary ensembles to perform the analysis on.

The bootstrap method is based on random sampling with replacement of the data points. This can be used to calculate the accuracy to sample estimates.

Lets consider a dataset consisting of m points. We generate a new dataset by drawing m sampling units with replacement from the original dataset. This means that the new dataset can consists of several identical points from the original set. By doing this B times we get B approximated data sets for which we can calculate the estimator of

interest. We can then compute the bias and the variance using the approximated ensembles. That is, for each point we compute the systematic error of the prediction (bias) and how the predictions fluctuates about the mean value (variance).

According to the central limit theorem, under the assumption that the our original data points is independent and identically distributed (iid), the sample estimates become normal distributed with mean value corresponding to the sample mean and a standard deviation σ for which the sample deviation is $\sigma/\sqrt{m}$. Thus when calculating the mean of the estimator over all the bootstrap resamples, we obtain an estimator value with an error of $\sigma/\sqrt{m}$.

We begin by calculating bias and variance for $N = 22$ and $\sigma = 0.2$. As a rule of thumb we use the same number of bootstraps as the number of training points. The result is shown in figure 6.

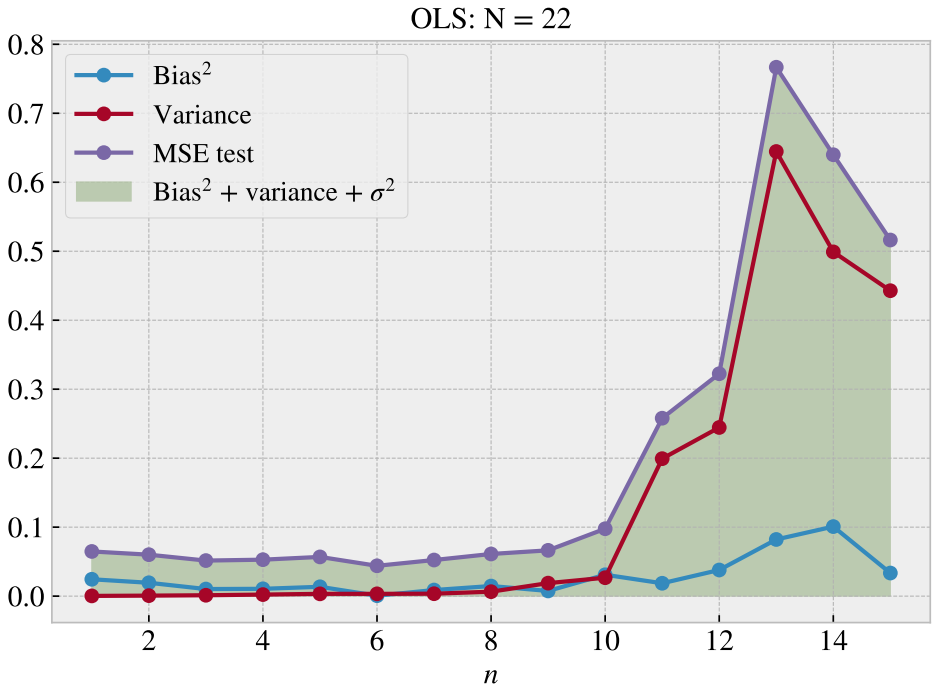


Figure 6: Bias-variance analysis for $N = 22$ and $\sigma = 0.2$ using the same number of bootstraps as traning points. We see that the main contribution to the test MSE comes from bias at low complexity and for variance at higher complexity.

In figure 6 we have investigated the tradeoff between high bias and low variance at low complexity vs low bias and high variance at high complexity. In our case the bias doesn't drop considerably for higher complexities when the variance increases (at $n \geq 10$). However, we see that the bias is the dominant error contributor at low complexities and variance is the dominant error contributor at higher complexities.

Notice also that we get a perfect match with the theoretical decomposition done in section 3.3 as the green coloring of figure 6 confirms the relation

$$\text{MSE} = \text{Bias}^2 + \text{variance} + \sigma^2.$$

From figure 6 we also observe that the MSE is at it lowest point for $n = 6$.

We now investigate the effect of varying the number of points, which turns out to affect the specific characteristics of the tradeoff quite a lot. The result is seen on figure 7.

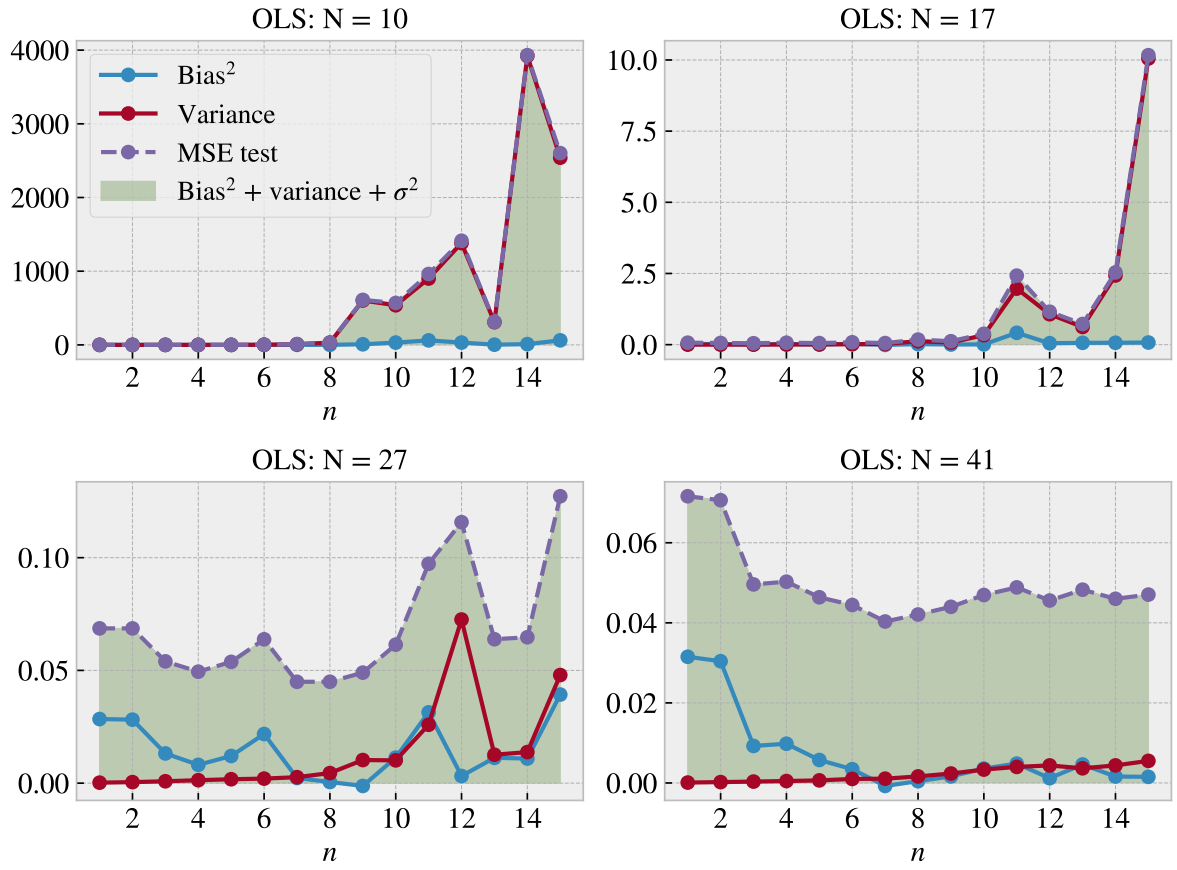


Figure 7: Bias-variance analysis similar to figure 6 but now with different choices of grid size: N = [10, 17, 27, 40].These was explicitly chosen to show some interesting features.

From 7 we see that the number of data points greatly affects the bias and variance relationship. Generally we see that that variance doesn't explode quite as drastically for higher N. This makes sense since the higher number of points makes the fitting less prone to overfitting.

4 Exercise 3

4.1 Cross validation

An alternative method to Bootstrap is Cross validation. Consider a dataset consisting of m points, that is shuffled, such that either the columns switch place or the rows switch place. For a given value of k, cross validation splits the dataset into k parts, where one part is the test data and the rest is training data. Now, the first iteration takes the first part as the test data. The second iteration sets the test data to the second part of the dataset, putting the first part and the third and outwards as training data, and this goes on until the kth iteration where the last part is train data. this is shown in figure 8 below. The error is calculated for each iteration and averaged to find the total error of the method. ¨

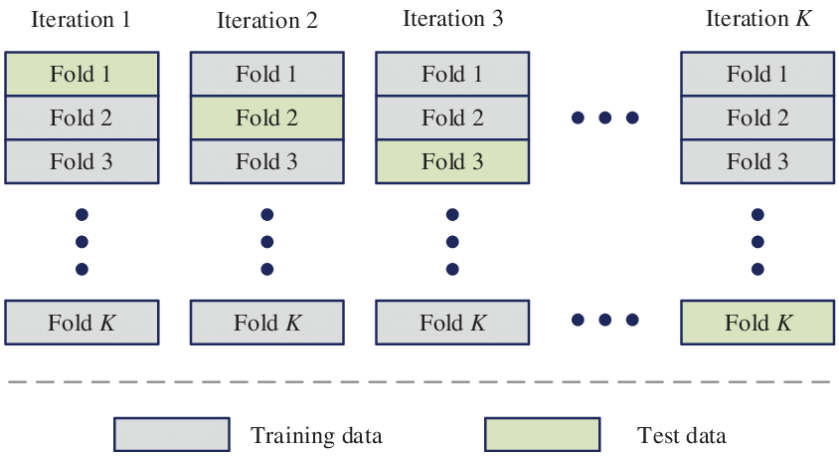
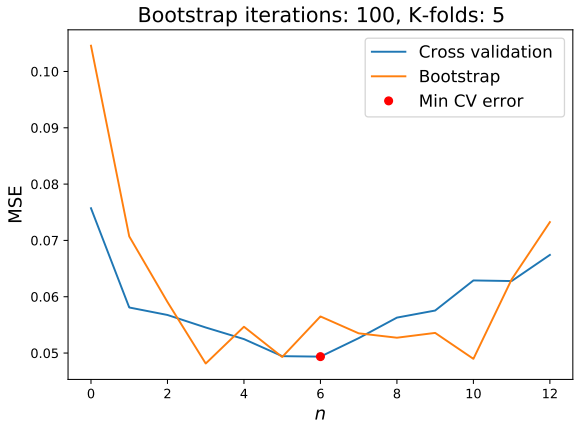
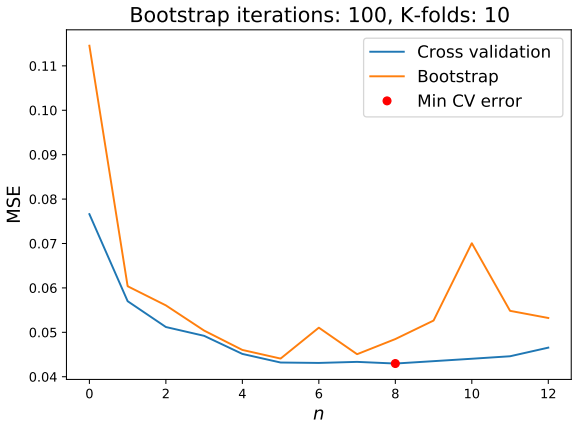


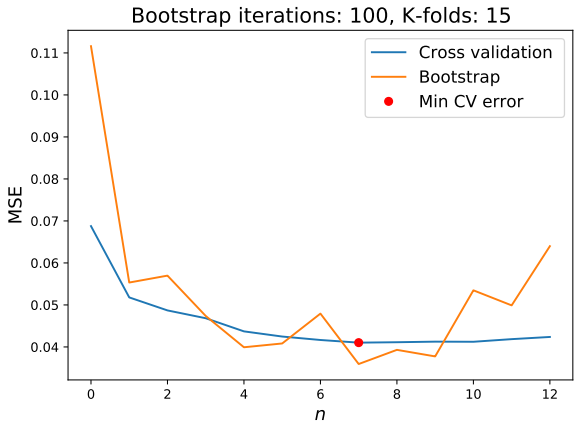
Figure 8: K-fold cross validation example chart.



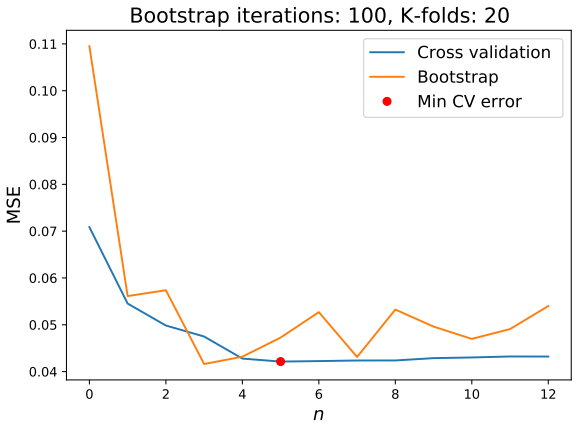
(a) Cross validation and bootstrap comparison with k-fold number = 5.



(b) Cross validation and bootstrap comparison with k-fold number = 10.



(c) Cross validation and bootstrap comparison with k-fold number = 15.



(d) Cross validation and bootstrap comparison with k-fold number = 20.

Figure 9: MSE plots for different k-fold values

The ideal number of folds for a given situation is not easy to determine, as shown in figure 9 below. Here, with ordinary least squares as linear regression method we see that by increasing the number of k-folds, we get a more smooth change in MSE for cross validation. For higher order of complexity bootstrap will tend to overfit, but Cross validation remains with relatively low error.

5 Exercise 4

5.1 Ridge Regression

From section 2 we have an expression for the optiomal β values. The inverting of $\mathbf{X}^T\mathbf{X}$ yields no problem as long as the matrix is not singular. The Ridge method addresses this issue by introducing a parameter λ. Our cost function is thus

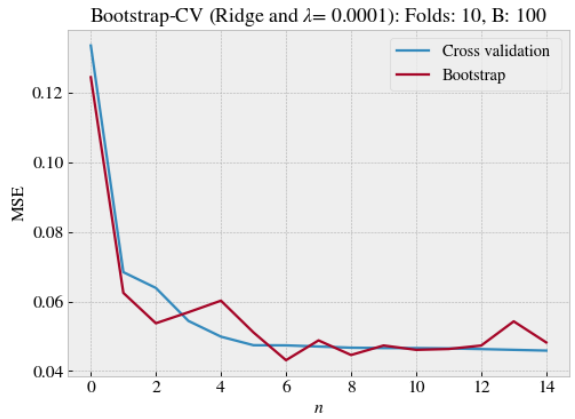
$$C(\mathbf{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 + \frac{\lambda}{n}\sum_{i=0}^{n-1}\tilde{z}_i^2 = \frac{1}{n}\left[(\boldsymbol{z}-\tilde{\boldsymbol{z}})^T(\boldsymbol{z}-\tilde{\boldsymbol{z}}) + \lambda\tilde{\boldsymbol{z}}^2\right], \qquad \tilde{\mathbf{z}} = \mathbf{X}\boldsymbol{\beta},$$

which gives us the optimal β as

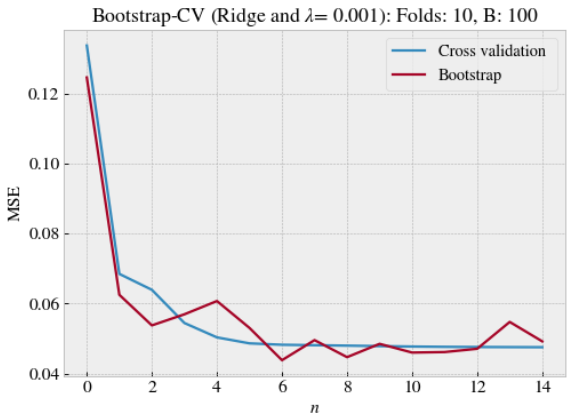
$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\boldsymbol{I})^{-1}\mathbf{X}^T\mathbf{z}.$$

5.2 Ridge analysis

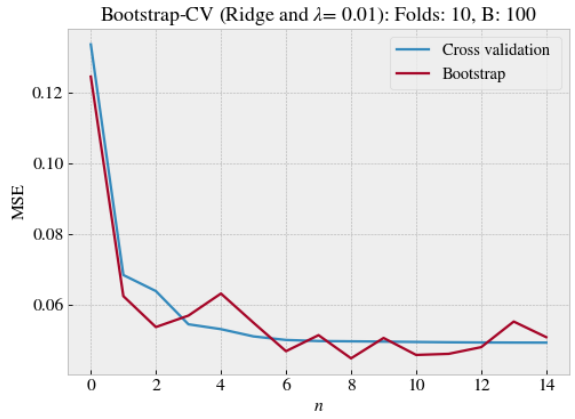
Firstly we want to repreat the bootstrap, cross-validation comparison but for Ridge. This time we will do the comparison for varying values of λ. The result is shown bellow:



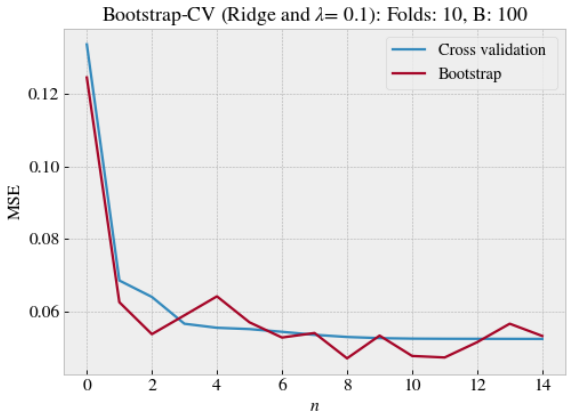
(a) Test data from Franke's function for N = 30 and $\sigma = 0.2$



(b) Predictions of the test data for OLS method with n = 6.



(c) Predictions of the test data for Ridge method with n = 6 and $\lambda 1.3 \times 10^{-11}$.



(d) Predictions of the test data for Lasso method with n = 13 and $\lambda 2.8 \times 10^{-6}$.

Figure 10: Comparing the predictions Subfigure

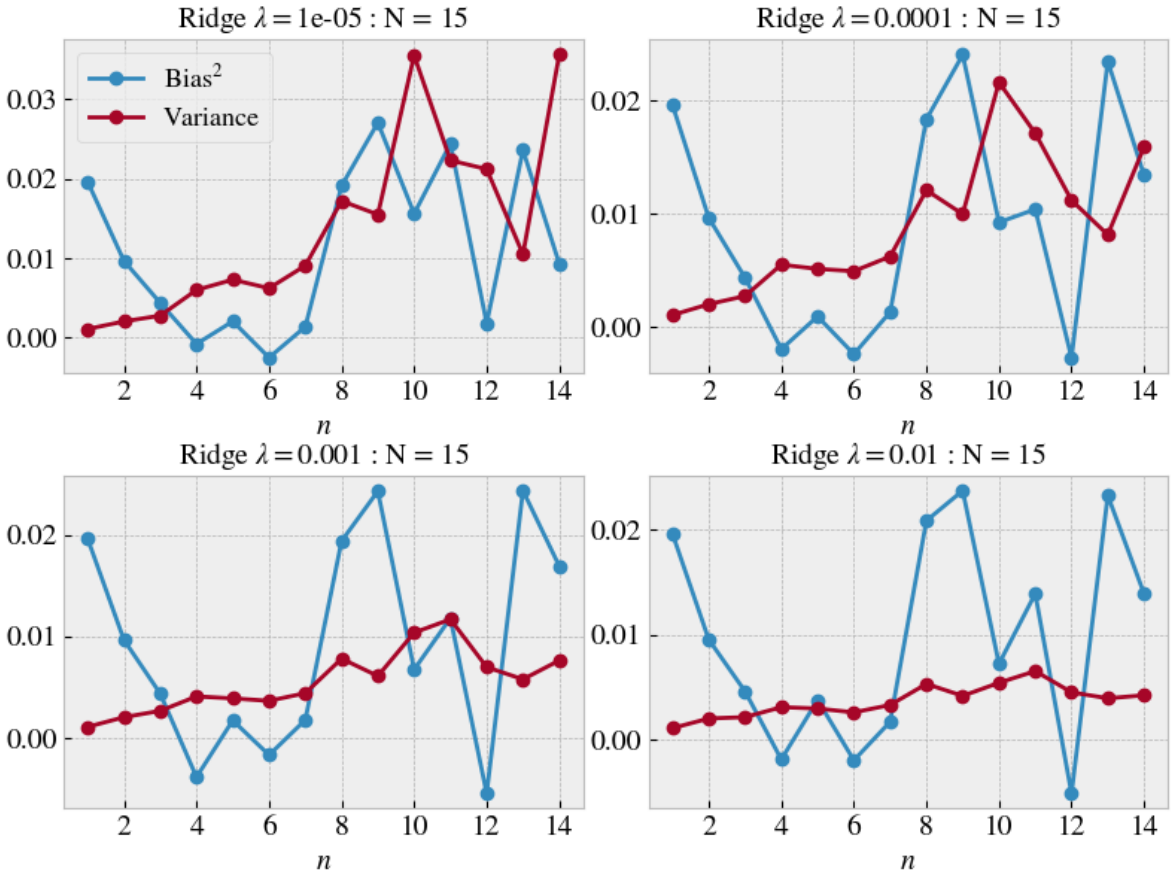


Figure 11: Bias variance trade-off for Ridge-method with λ-values [1e-5,1e-4,1e-3,1e-2] and N =15.

In figure (11) we have calculated the bias and variance for N = 15 using the Ridge method. The plot above shows that for small values of λ, the variance is large. This is expected as Ridge approximates OLS when λ is small. We also notice that for larger values of λ the variance decreases and the bias starts to dominate. It can therefore be interesting to see what λ does to β.

In figure (12) we see the values of β as a function of λ. Here we see that as λ increases, the variance of the betas decrease. Therefore one possible solution to the decrease of variance in our predictions, is that as λ increases, the variance in β decreases which leads to the decrease of variance in the prediction.

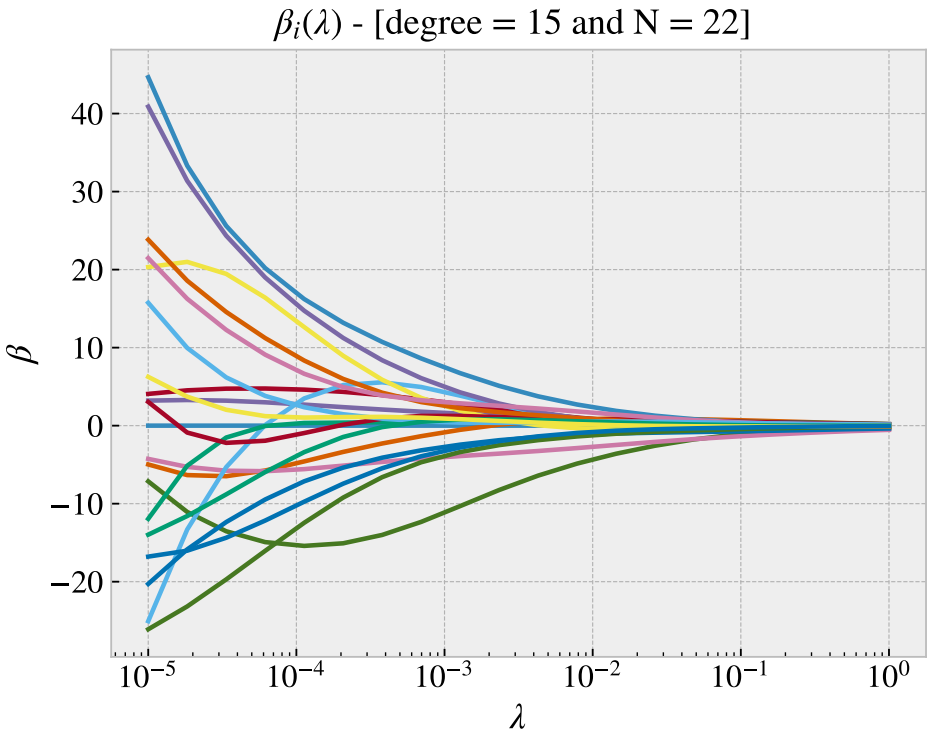


Figure 12: Betas as a function of λ.

Note also that some of the bias values in figure (11) are negative, which does not make sense given that the bias is plotted as bias squared. This can be explained by the number of points. In our derivation of our expression for bias, we assumed normal a distribution of noise. This assumption becomes increasingly bad as the number of data points decreases.

6 Exercise 5

6.1 Lasso Regression

Another regression method is Lasso regression. Here we take the absolute value of beta instead of the square, giving us

$$C(\mathbf{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i-\tilde{z}_i)^2 + \frac{\lambda}{n}\sum_{i=0}^{n-1} \mid \tilde{z}_i \mid = \frac{1}{n}\left[(\boldsymbol{z}-\tilde{\boldsymbol{z}})^T(\boldsymbol{z}-\tilde{\boldsymbol{z}}) + \lambda \mid \tilde{\boldsymbol{z}} \mid\right], \qquad \tilde{\mathbf{z}} = \mathbf{X}\boldsymbol{\beta}.$$

6.2 Lasso analysis

In figure 13 we have the Bias-variance trade-off with the Lasso method and N=12. With increased λ we get a dampening of variance, which is expected. We also see

that the dampening is faster for the Lasso method than for Ridge. Lastly we note that we here, as with 11, have negative bias^2 values. The same arguments holds here as well.

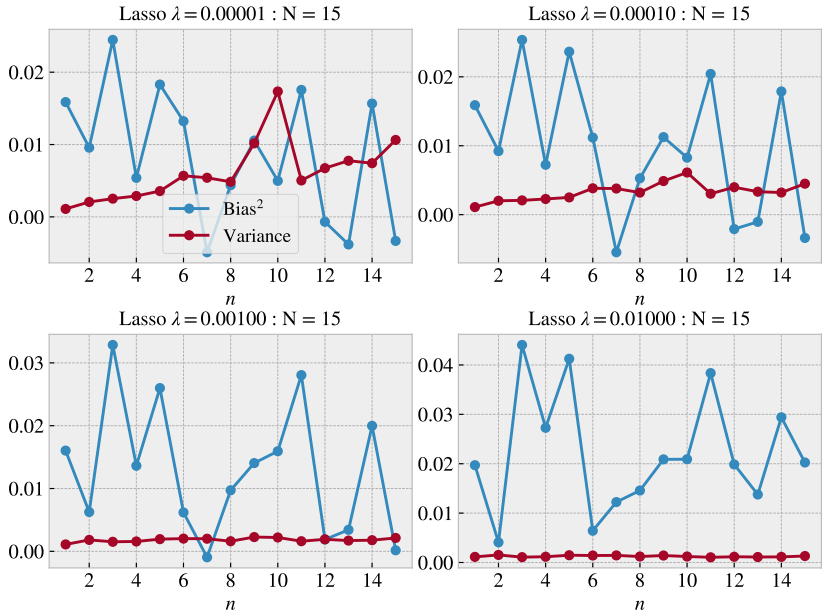


Figure 13: Bias Variance trade-off for Lasso with $N = 15$ and varying $\lambda \in [10^{-5}, 10^{-1}]$

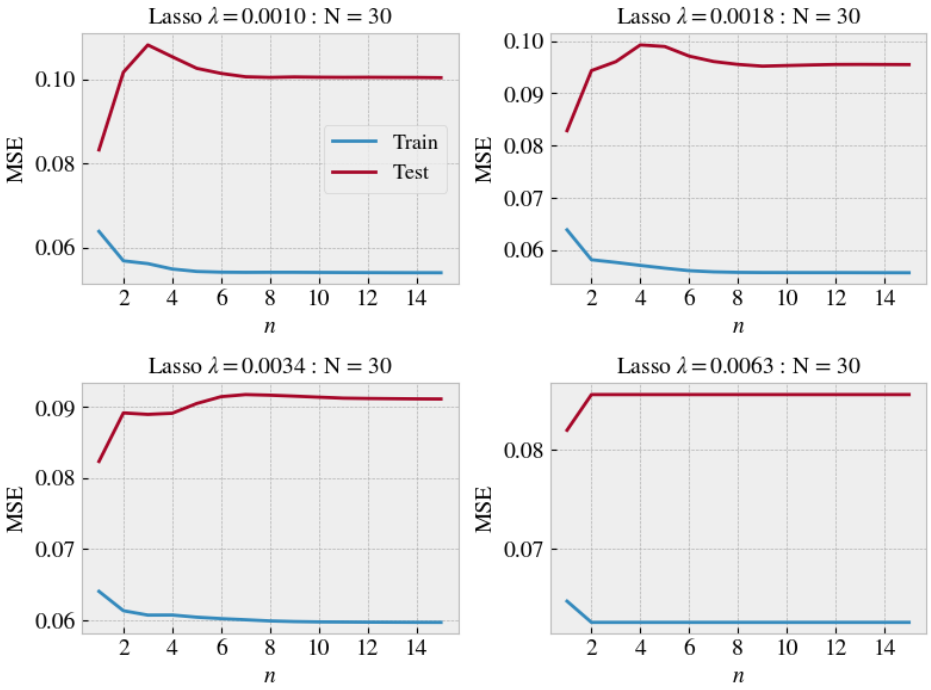


Figure 14: Lasso regression using Cross validation with k fold number $= 5$.

In figure (14) we see that the regression is somewhat sensitive to low values, but gets more stable for increasing values of λ. This is consistent with figure (12), where the variance of β declines as λ increases. In fact, lasso regression converges to a straight line, for $\lambda > 0.01$, thus we see that although the error is slightly higher for larger λ, it remains consistent around the error, with no extreme overfitting.

6.3 Finding the best model for Franke's function

We will now investigate which regression fits the general trend of Franke's function the best. We begin by estimating the best hyperparameters for each regression method: OLS, Ridge, and Lasso respectively. We generate a data set with $N = 30$ and noise $\sigma = 0.2$, and use the usual 80-20% split. We then perform regression on the training set and evaluate the MSE by cross-validation (with 5 k-folds) for different combinations of hyperparameters n and λ. The result is shown for OLS in figure 15, for Ridge in figure 16 and Lasso in figure 17.

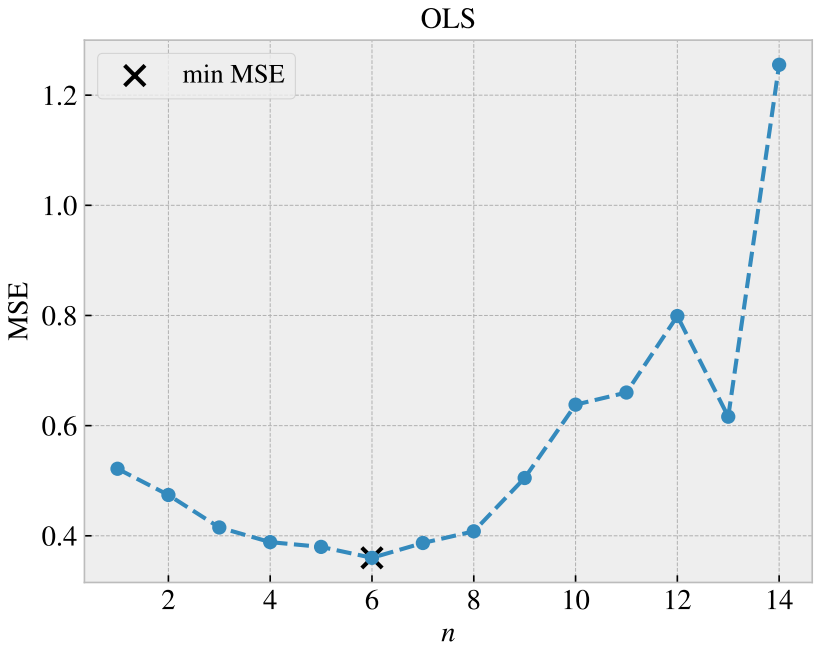


Figure 15: OLS regression on Franke's function with training MSE, computed by cross validation for 5 k-folds, as a function of n.

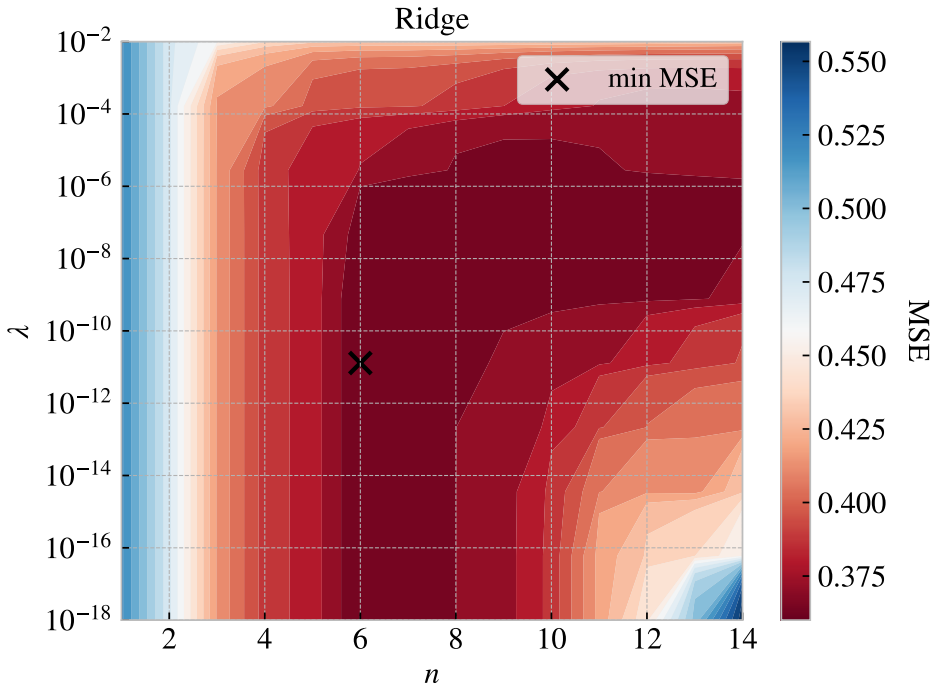


Figure 16: Ridge regression on Franke's function with training MSE, computed by cross validation for 5 k-folds, as a function of n and λ.

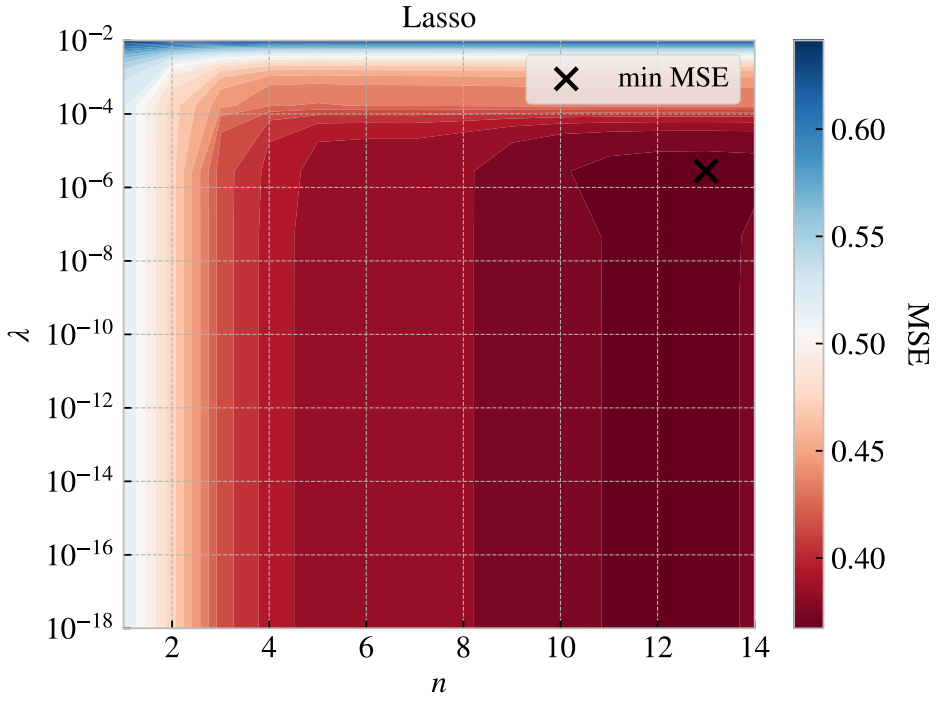


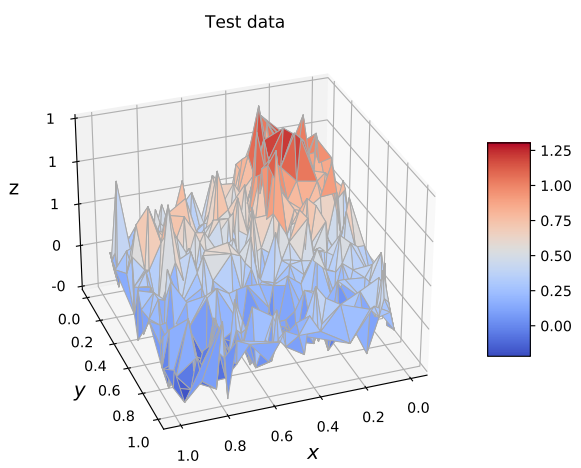
Figure 17: Lasso regression on Franke's function with training MSE, computed by cross validation for 5 k-folds, as a function of n and λ.

From figure 15, 16 17 we find the best hyperparameters to be as shown in table 3.

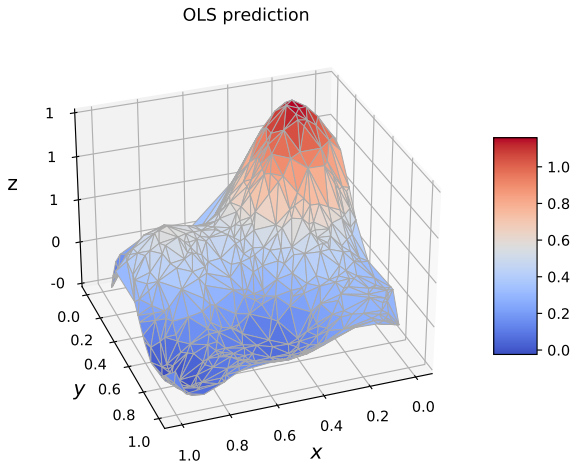
Table 3: The best hyperparameters for OLS, Ridge and Lasso regression on Franke's function found from figure 15, 16 17.

	OLS	Ridge	Lasso
n	6	6	13
λ		1.3×10^{-11}	2.8×10^{-6}

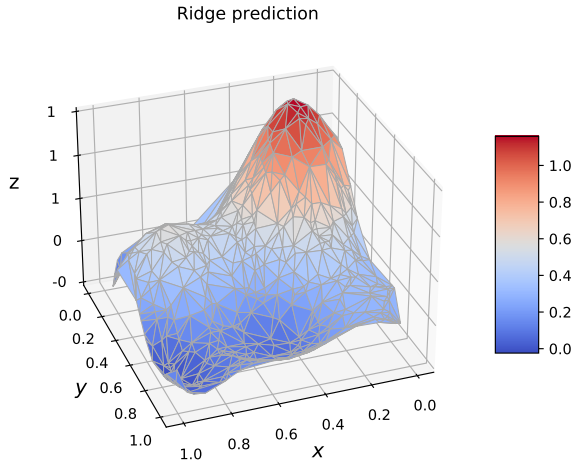
Using the optimal hyperparameters for each regression method we compute the predictions on the test data and evaluate the MSE between the predictions and the actual target points. On figure 18a we see a 3D plot of the test data which is expected to replicate with the predictions shown in figure 18



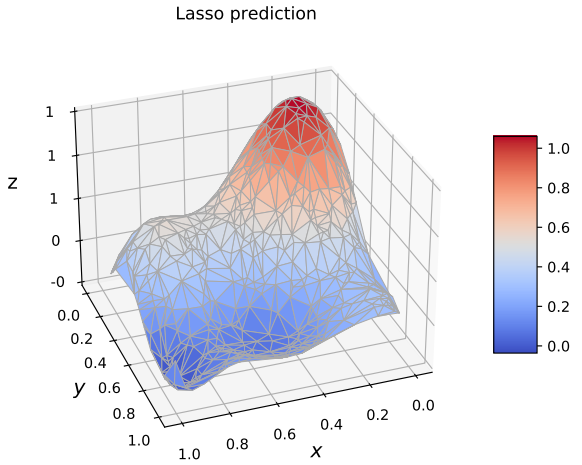
(a) Test data from Franke's function for N = 30 and $\sigma = 0.2$



(b) Predictions of the test data for OLS method with n = 6.



(c) Predictions of the test data for Ridge method with n = 6 and $\lambda 1.3 \times 10^{-11}$.



(d) Predictions of the test data for Lasso method with n = 13 and $\lambda 2.8 \times 10^{-6}$.

Figure 18: Comparing the predictions Subfigure

From figure 18 we see that all three methods create prediction that resemble the Franke function (see figure 5a). This means that all three methods were able to find the trend of the Franke's function even with the added noise.

Table 4: MSE score for the test data with methods; OLS, Ridge and Lasso. All methods using optimal values from training set.

	OLS	Ridge	Lasso
MSE	0.33894	0.33894	0.34050

From table 4 we can see that Ridge slightly produces a smaller MSE value, with

OLS coming second. Given the nature of our comparison, Ridge is therefore the best method for our data set. We notice however that the optimal λ-value for Ridge is very small (magnitude of 10^{-11}). As discussed previously, Ridge approaches OLS as λ decreases. Therefore we could speculate, given that the difference in MSE is so small, that the methods (OLS and Ridge) are approaching each other. This would make OLS the best method for our set.
Lastly it is worth saying that during the calculation for the optimal parameters, we came across problems while using sklearn's Lasso-method. For certain combinations of n and λ, sklearns functions gave warnings of convergence errors. In order to fix this problem we had to increase max iterations for the lasso-function. But, as the calculations were all ready computational heavy, an increase of max iterations to more than 1×10^4 would make the calculations unmanageable. Therefore we must treat or results as a comparison between the methods, limited by our numerical resources. This leads to the fact that our chosen parameters might not be the optimal parameters for each method.

7 Exercise 6

7.1 Test models on Terrain

In this exercise we will use OLS, Ridge, and Lasso regression respectively to analyze real terrain data. By using the website https://earthexplorer.usgs.gov/ we are able to retrieve real world terrain data. We choose a hilly area in Saudi Arabia which is shown in figure 19. The original resolution was quite high, so in order to process the data faster we reduced it to 37×37 (1369) data points. Due to lack of information regarding the units we treat them simply as unknown length units.

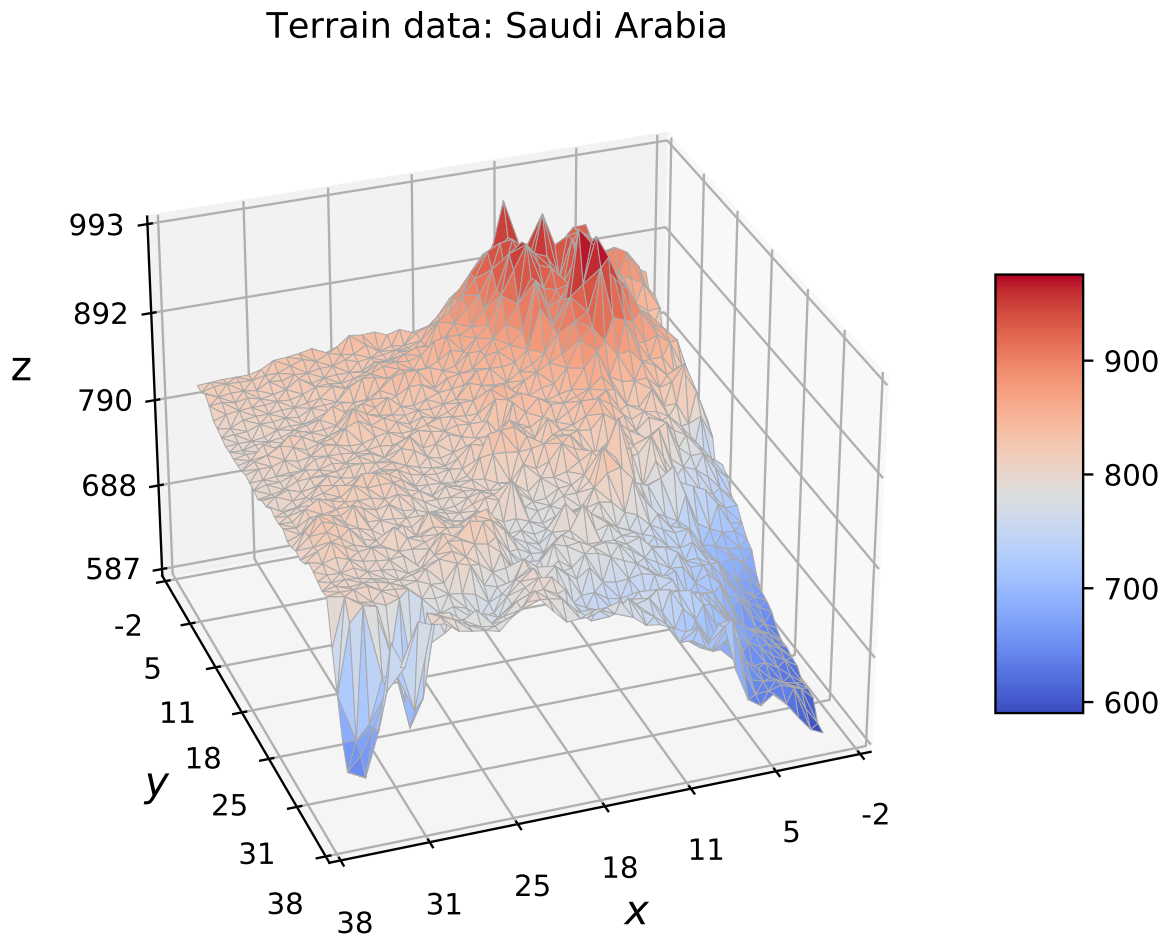


Figure 19: Terrain data from Saudi Arabia with $N \times N = 37 \times 37$ (1369 data points). We treat the units as unknown length units.

We perform a similar investigation on the terrain data as done in exercise 5 to determine the best hyperparameters for each regression method and then compare the MSE on the test data. The search for the best hyperparameters is shown in figure 20,

21 and 22

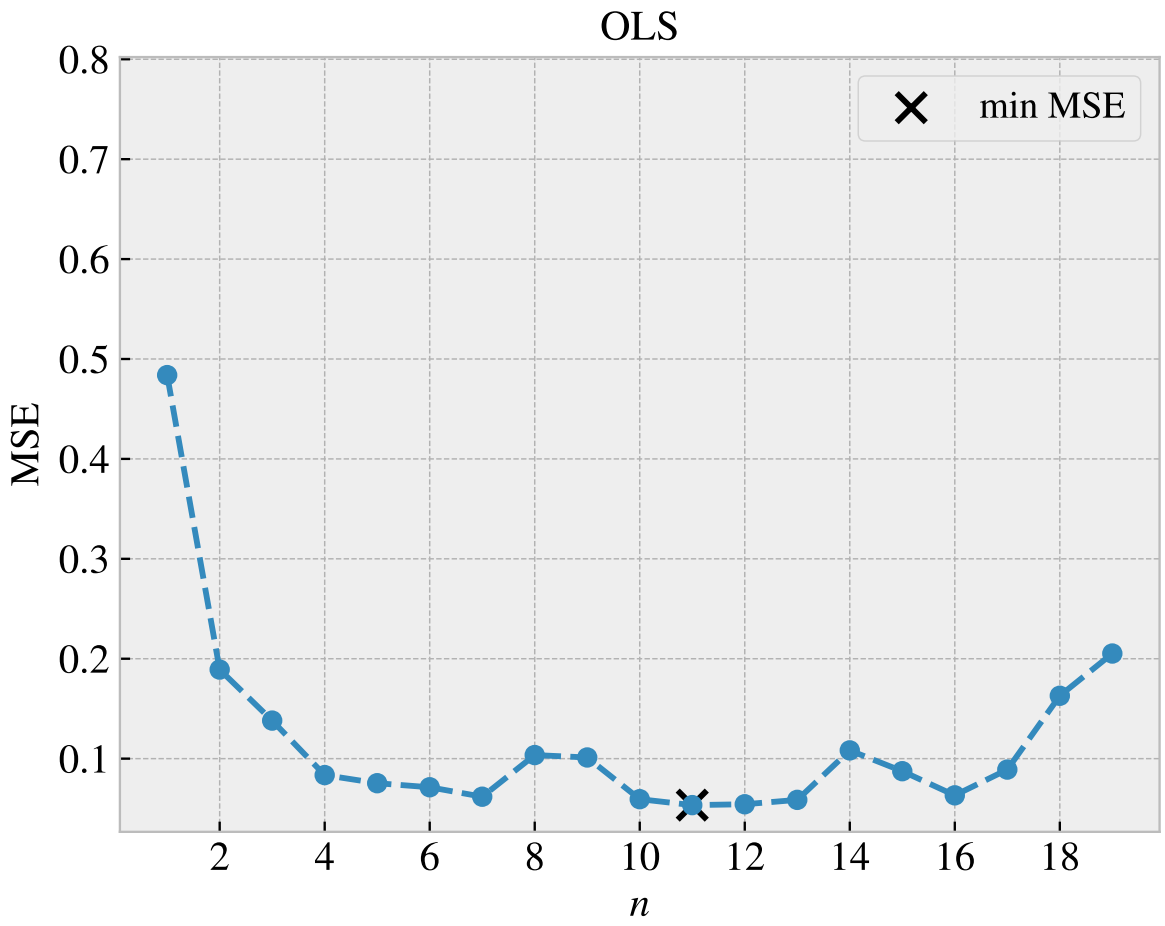


Figure 20: Best hyperparameter estimation for OLS on terrain data. The plot shows MSE (found by k-fold = 5 cross-validation) for the training data with varying$n$. The minimum MSE error is achieved for $n = 11$

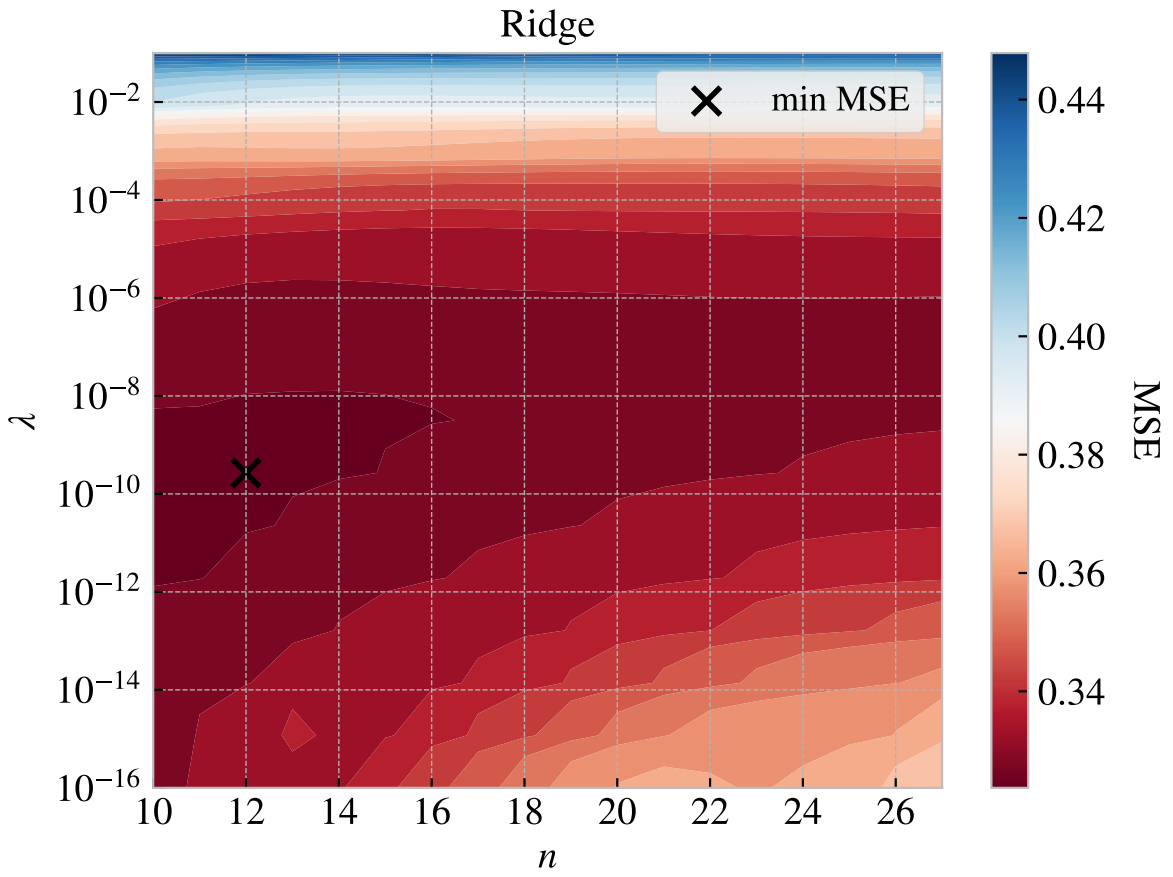


Figure 21: Best hyperparameter estimation for Ridge on terrain data. The plot shows MSE (found by k-fold = 5 cross-validation) for the training data with varying n and λ. The minimum MSE error is achieved for $n = 12$, $\lambda = 2.682\,695\,80 \times 10^{-10}$

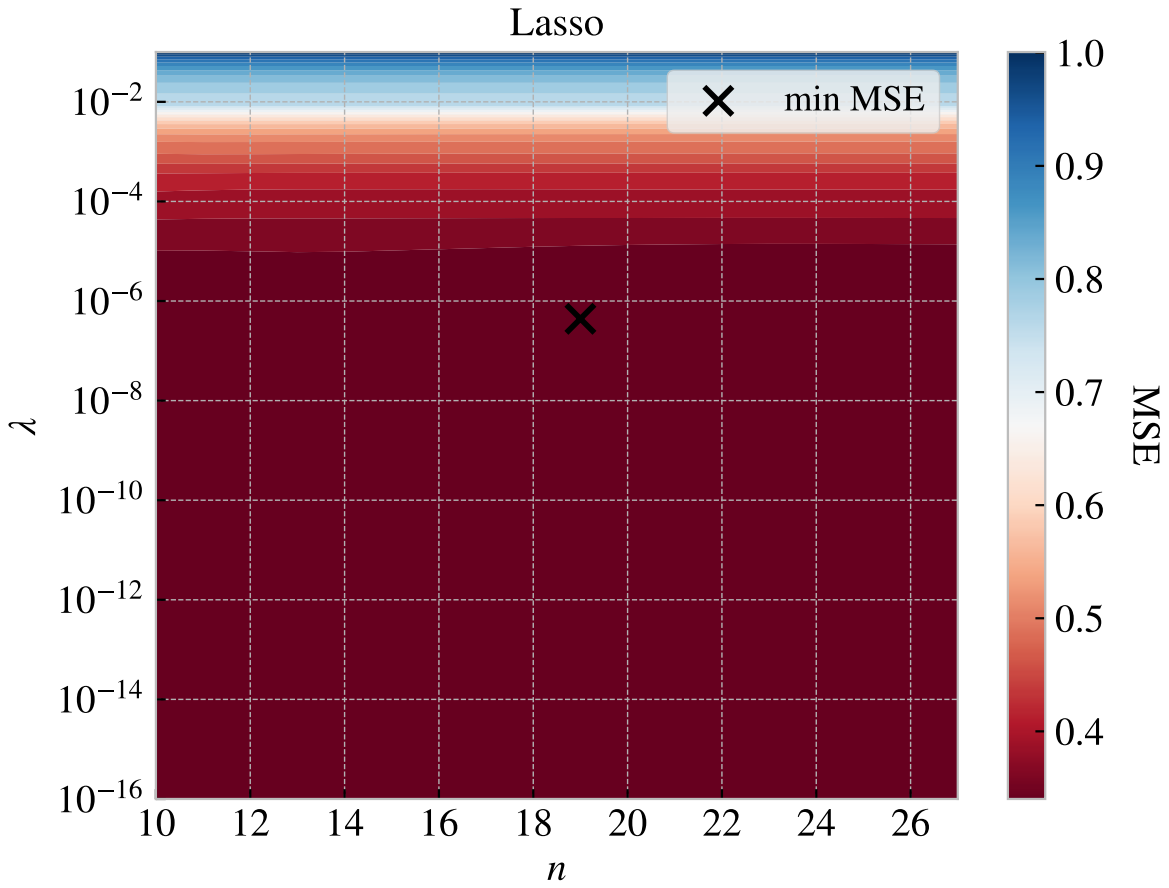


Figure 22: best parameters $n = 19$, $\lambda = 4.39397056 * 10^{-7}$.

With these parameters, our predictions now look like this:

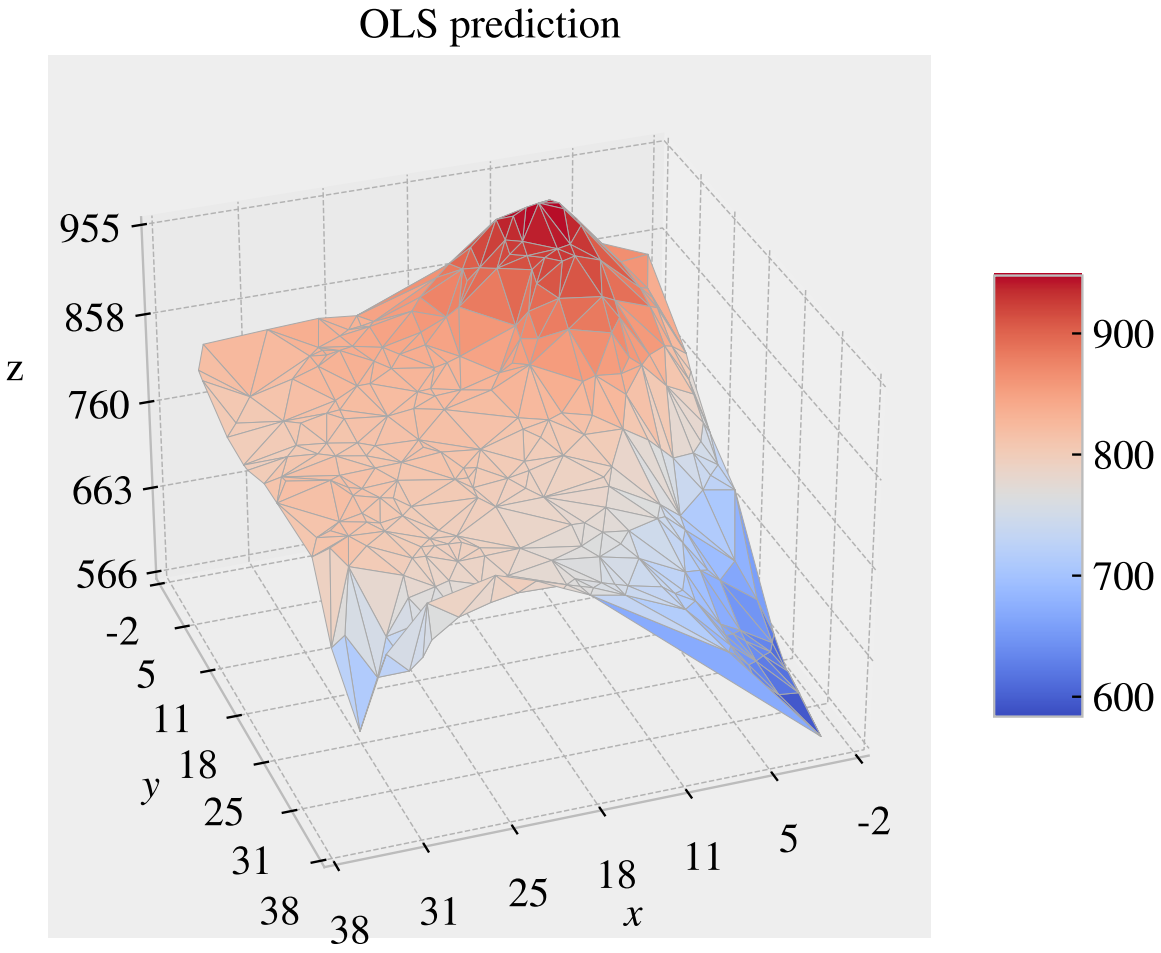

OLS prediction
z
955
858
760
663
566
-2
5
11
18
25
31
38
y
38
31
25
18
11
5
-2
x
900
800
700
600

Figure 23

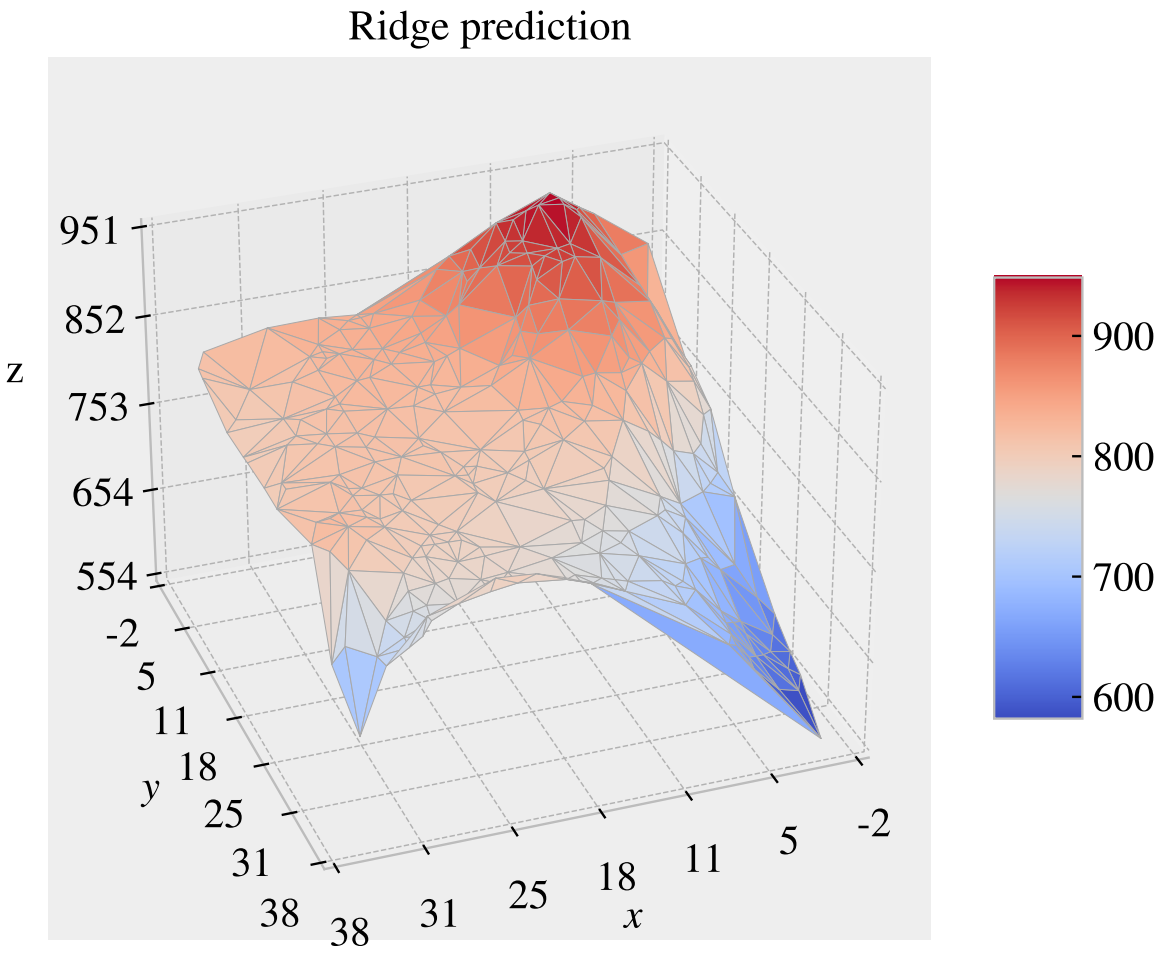

Ridge prediction
z
951
852
753
654
554
-2
5
11
18
25
31
38
y
38
31
25
18
11
5
-2
x
900
800
700
600

Figure 24

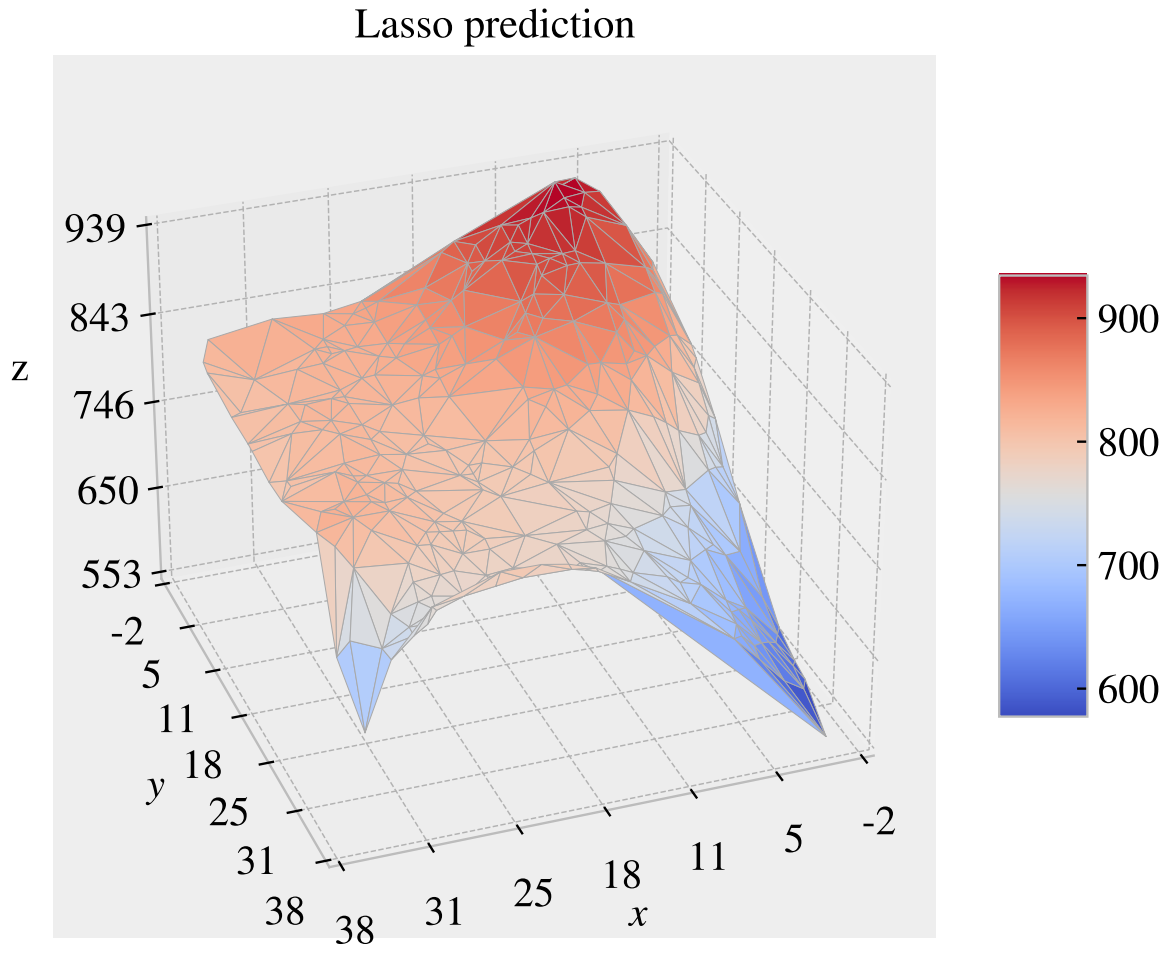


Figure 25

7.2 Max iterations, other stuff?

Max iterations for convergence rate will affect the result, but drastically affects the run time of the scripts as well. We tried only using results where the solver converged probably but for settings within the computational reach we were not able to find a minimum combinaiton of the hyperparameters n and λ