# FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

October 4, 2019

**Abstract**

## 1 Introduction

Things we need to write about:

Background on regression analysis and resampling methods. Mention: OLS Ridge Lasso k-fold cross-validation Bias-Variance trade of?

We will first study how they preform for the two dimensional Franke-function. (A bit about the Franke-function, maybe a tldr for the method).

We will then implement them for some real terrain data for Møsvatn Austfjell in Norway. mm. (biggest lake in Norway )

## 2 Data

In this report, we will be working with two separate datasets.

### 2.1 The Franke Function

The first dataset will be given by the *Franke function*, which is defined as follows:

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

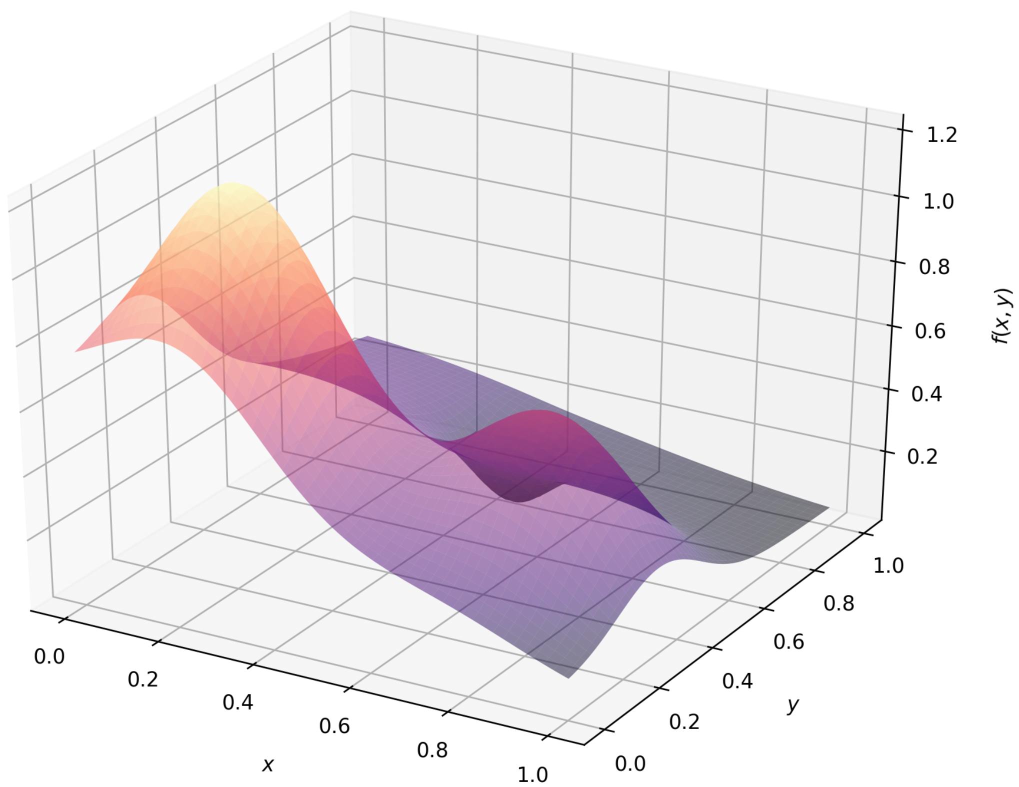


Figure 1: The *Franke function* for $x$ and $y$ values ranging from zero to one.

We will be solving the Franke function for 100 $x$-values and 100 $y$-values in the range $[0, 1]$, leaving us with a grid containing a total of 10000 $xy$ coordinate pairs. This leaves us with the values plotted in Figure 1.

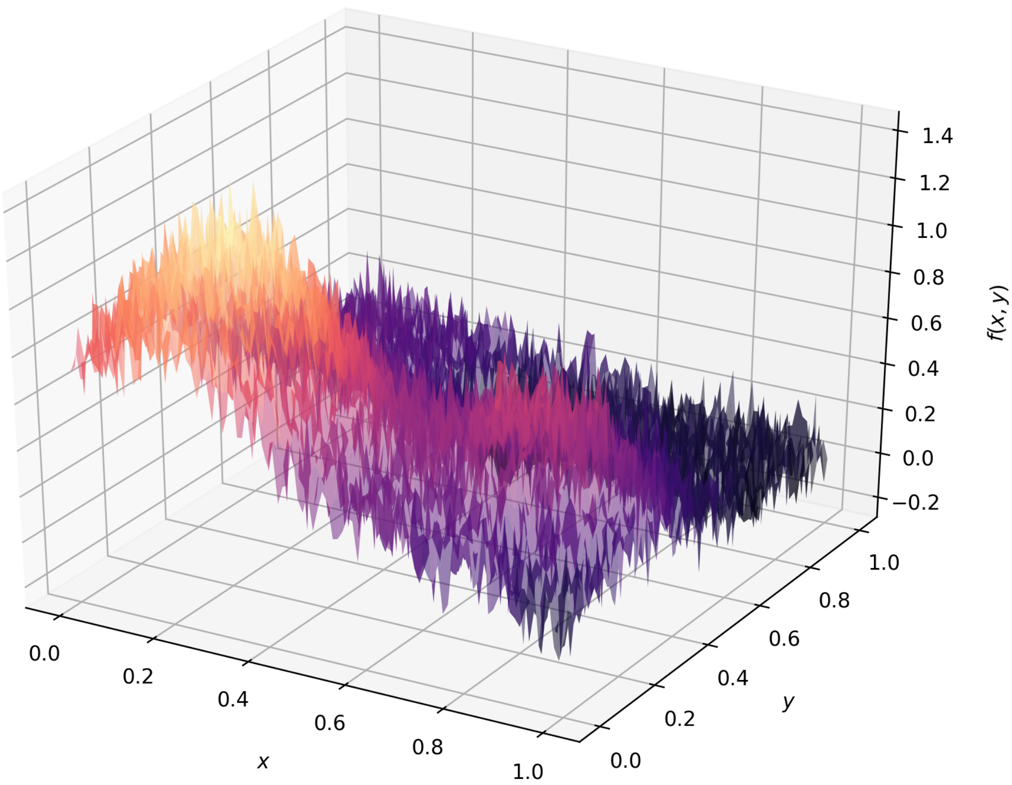


Figure 2: The *Franke function* for $x$ and $y$ values ranging from zero to one, with a Gaussian noise $N(0, 0.01)$

In addition, we will also be adding *Gaussian noise* to each value $f(x, y)$, such that we are left with values as seen in Figure 2.

## 2.2 Møsvatn Austfjell

For our second dataset, we will be using real data taken from the *U.S. Geological Survey* [?] official website `https://earthexplorer.usgs.gov/`. More specifically, we will be using a `.tif` file containing altitude data for a rectangular region of Møsvatn Austfjell shown in Figure 3.

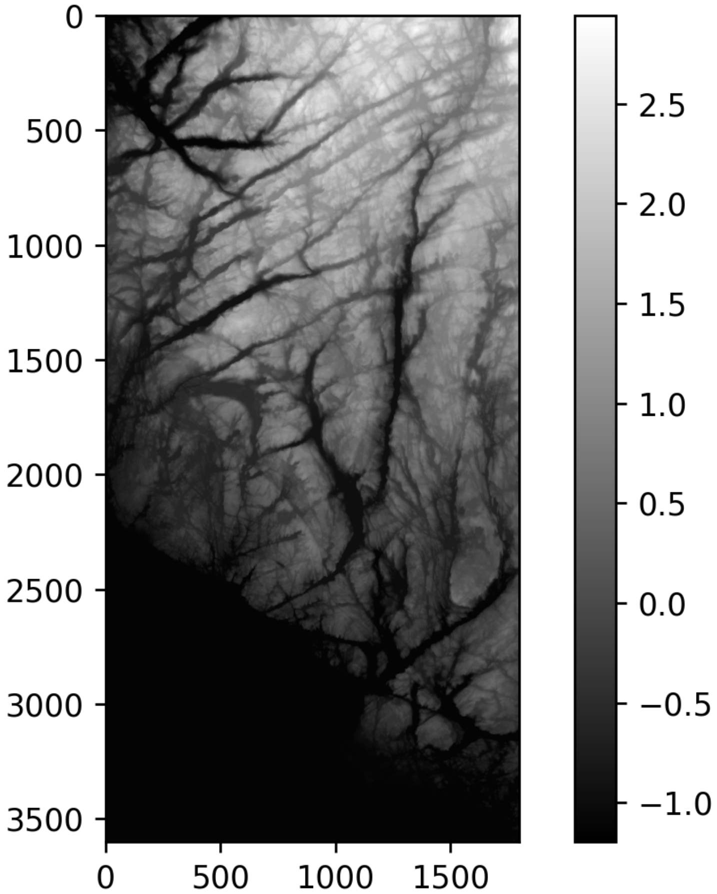


Figure 3: Altitude data for Møsvatn Austfjell, from the USGS website [?].

# 3 Method

## 3.1 Generalization of Multidimensional Polynomials

If we wish to construct a $p$-dimensional polynomial of degree $d$, we need to know what terms need to be included to give us a completely generalized polynomial. For a 1-D polynomial of second degree, we would have three terms:

$$f(x) = \beta_1 + \beta_2 x + \beta_3 x^2$$

Where $\beta$ is a vector containing each coefficient. For a 2-D polynomial of second degree, we would have 6 terms:

$$f(x, y) = \beta_1 + \beta_2 x + \beta_3 y + \beta_4 xy + \beta_5 x^2 + \beta_6 y^2$$

And for a 3-D polynomial of second degree, we would have 10 terms:

$$f(x, y, z) = \beta_1 + \beta_2 x + \beta_3 y + \beta_4 z + \beta_5 xy + \beta_6 xz + \beta_7 yz + \beta_8 x^2 + \beta_9 y^2 + \beta_{10} z^2$$

There are many possible combinations of $p$ and $d$, and the number of terms blows up significantly as these values increase. We can, however, create a general expression [?] for any $p$ and $d$ using summation notation:

$$f(\mathbf{x}) = \sum_{\sum_{j=1}^{d} i_j \leq p} \left( \beta_{i_1, i_2, \cdots, i_d} \prod_{k=1}^{d} x_k^{i_k} \right) \tag{1}$$

Alternatively, a simple **python** script can be used to find all the terms' exponents by calculating all permutations of the natural numbers from zero to $d$ in sets of length $p$, then removing all results whose sum is greater than $d$.

```
powers = np.arange(0, degree + 1, 1)
powers = np.repeat(powers, p)
exponents = list(permutations(powers, p))
exponents = np.unique(exponents, axis = 0)

if p != 1:
expo_sum = np.sum(exponents, axis = 1)
valid_idx = np.where(np.less_equal(expo_sum, degree))[0]
exponents = np.array(exponents, dtype = np.int64)
exponents = exponents[valid_idx]
```

```
else:
exponents = np.array(exponents, dtype = np.int64)
```

### 3.2 Ordinary Least-Squares (OLS) Regression

We are given a $p+1$-dimensional dataset[1] consisting of $N$ datapoints per feature such that:

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,p} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,p} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{2}$$

Where the $N \times p$ matrix $\mathbf{X}$ contains the dataset's *input data*, and the $N$-vector $\mathbf{y}$ contains its *output data*, such that each row in $\mathbf{X}$ corresponds to a single output in $\mathbf{y}$.

Now, we wish to find a $p$-dimensional polynomial of degree $d$ which most closely matches our dataset. We will need a design matrix $\mathbf{A}$; this will require using the knowledge presented in (1), since a design matrix should contain each polynomial term as an individual column.

Next, we will be using the method of *least squares* [?], whereby we attempt to minimize the *residual sum of squares*:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \mathbf{A}_i^{\mathrm{T}} \beta)^2$$

Where $\mathbf{A}_i$ represents the $i^{\text{th}}$ row in $\mathbf{A}$, and *beta* is the set of coefficents in the aforementioned polynomial; in matrix form, this can be written more concisely:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^{\mathrm{T}} (\mathbf{y} - \mathbf{A}\beta)$$

To minimize the $RSS$, we can differentiate it with respect to $\beta$ and set the right-hand side equal to zero – this allows us to solve for $\beta$, which will give us the coefficients to the polynomial that best matches our dataset:

$$\mathbf{A}^{\mathrm{T}}(\mathbf{y} - \mathbf{A}\beta) = 0 \iff \mathbf{A}^{\mathrm{T}} y = \mathbf{A}^{\mathrm{T}} \mathbf{A}\beta$$

Solving for $\beta$ then gives us our desired result:

$$\beta = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} \tag{3}$$

Using the set of coefficients given by (3), we can then match the dataset from (2) as effectively as possible.

### 3.3 Ridge Regression

The solution for $\beta$ given in (3) can be used without issue in many cases, but if the matrix $\mathbf{A}$ is singular[2], we run into an issue – namely, we cannot take the inverse of a singular matrix! As a result, we must look to more robust methods; one such method is called *ridge regression*.

The process of obtaining our vector of coefficients $\beta$ via ridge regression is functionally very similar to that of OLS. The main difference is that we include an extra term in the residual sum of squares:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \mathbf{A}_i^{\mathrm{T}} \beta)^2 + \lambda \sum_{i=1}^{N} \beta^2$$

In matrix form, this can be rewritten:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^{\mathrm{T}} (\mathbf{y} - \mathbf{A}\beta) + \lambda \beta^{\mathrm{T}} \beta$$

Where $\lambda$, known as the *hyperparameter*, is a scalar value. Performing the same process as in the previous subsection, we are left with a solution similar to that in (3):

$$\beta = (\mathbf{A}^{\mathrm{T}}\mathbf{A} + \lambda \mathbf{I})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} \tag{4}$$

In cases where $\mathbf{A}$ is singular, it is therefore possible to make very few changes to the OLS algorithm and still get a good result, one must simply optimize the hyperparameter and find a $\lambda$ that minimizes the *mean squared error* (yet to be introduced) of our polynomial approximation.

[1] Meaning a set of $p$ input features and 1 output.
[2] Meaning that $\det(\mathbf{A}) = 0$

### 3.4 LASSO Regression

The *least absolute shrinkage and selection operator*, commonly abbreviated as *LASSO*, is a method that implements the **L1** norm in place of the **L2** (or Euclidian) norm used in ridge regression. The residual sum of squares is therefore given by:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \mathbf{A}_i^{\mathrm{T}}\beta)^2 + \lambda \sum_{i=1}^{N} |\beta| \tag{5}$$

Unfortunately, differentiating the above with respect to $\beta$ will not work as intended, since we cannot take the matrix-form derivative of $\lambda \sum_{i=1}^{N} |\beta|$. As a result, we must use an iterative *gradient descent* method to minimize the right-hand side of (5).

**Algorithm 1** The LASSO algorithm, over the course of 500 iterations.

$z = \sum_i A_i^2$
$i = 0$
**while** $i \leq 500$ **do**
    $i = i + 1$
    $j = 0$
    **while** j ¡ p **do**
        $\hat{y} = \sum_{k \neq j} \beta A_{*,k}$
        $\rho = \sum_k A_{*,k}(\mathbf{y} - \hat{\mathbf{y}})$
        **if** $\rho < -\lambda/2$ **then**
            $\beta_j = (\rho + \lambda/2)/z_j$
        **else if** $\rho > \lambda/2$ **then**
            $\beta_j = (\rho - \lambda/2)/z_j$
        **else**
            $\beta_j = 0$
        **end if**
    **end while**
**end while**

### 3.5 Mean Squared Error

To get a measure of success with respect to the implemented method and parameters, we can calculate the mean difference in the squares of each measured output $y_i$ and their respective predicted outputs $\hat{y}_i$:

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] \tag{6}$$

The lower the $MSE$, the closer the polynomial approximation is to the original dataset. If it is too low, however, we run the risk of overfitting our dataset, which is not desireable either – fortunately, this not an issue within the scope of this report.

### 3.6 $\mathbf{R^2}$ Score

Another measure of success is the *coefficient of determination*, colloquially known as the $R^2$ score, is given by the following expression:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \tag{7}$$

The closer $R^2$ is to one, the closer the polynomial approximation is to the input/output dataset, although a perfect score can once again arise due to overfitting just as in the case of the $MSE$.

### 3.7 Bias-Variance Tradeoff

Before we continue, we can decompose the range of outputs $\mathbf{y}$ as follows:

$$\mathbf{y}(\mathbf{X}) = f(\mathbf{X}) + N(0, \sigma) \tag{8}$$

Where $f(\mathbf{X})$ represents the *actual* function used to generate the dataset, and $N(0, \sigma)$ is a Gaussian noise with a standard deviation of $\sigma$.

As a regression model increases in complexity[3], it so happens that the *variance* of a prediction increases. Variance is defined as follows:

$$\mathrm{Var}(\mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbb{E}[\mathbf{y}]) \tag{9}$$

On the other hand, we have that the *bias* of the prediction decreases as the complexity increases. We define the bias as:

[3]For a polynomial regression, this would refer to its *degree*.

$$\mathrm{Bias}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\mathbf{y}]) \tag{10}$$

Note that in order to calculate the bias, we need to know the original function $f$ used to generate $\mathbf{y}$.

Interestingly enough, taking the sum of (9) and (10) as well as $\sigma^2$ will yield the *mean squared error*. We will show this to be the case in the following subsection.

### 3.7.1 Derivation

We wish to show that:

$$\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 \tag{11}$$

We begin by rewriting the $MSE$ into summation notation, decomposing the terms as defined in (8), and adding/subtracting a term $\mathbb{E}[\hat{\mathbf{y}}]$:

$$\begin{aligned}\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(f_i + \varepsilon - \hat{y}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(f_i + \varepsilon - \hat{y}_i + \mathbb{E}[\hat{\mathbf{y}}] - \mathbb{E}[\hat{\mathbf{y}}])^2\end{aligned}$$

Next, we set $a \equiv f_i - \mathbb{E}[\hat{\mathbf{y}}]$ and $b \equiv \hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]$ and expand:

$$\frac{1}{n}\sum_{i=1}^{n}(a - b + \varepsilon)^2 = \frac{1}{n}\sum_{i=1}^{n}(a^2 - 2ab + b^2 - 2b\varepsilon + \varepsilon^2 + 2a\varepsilon)$$

The next few steps are messy, and require lots of algebraic manipulation:

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\varepsilon^2) + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 - \frac{2}{n}\sum_{i=1}^{n}\varepsilon(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) + \\ &\frac{2}{n}\sum_{i=1}^{n}\varepsilon(f_i - \mathbb{E}[\hat{\mathbf{y}}]) - \frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) = \\ &\frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 - \mathbb{E}[\varepsilon]\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) + \\ &\mathbb{E}[\varepsilon]\frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}]) - \frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])\end{aligned}$$

Finally, we see that our original assumption given by (11) is correct.

$$= \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 \qquad \square$$

Where $\frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])$ is clearly the *bias* and $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2$ is the *variance*.

## 3.8 Cross-Validation

So far, all our methods involving validation[4] may have involved the separation of our data into a training and testing set, whereby the vector of coefficients is calculated using the *training* set, and the validation is performed on the *testing* set that remains. There is however one more way to obtain a clearer picture of how well a model works: *k-fold* cross validation.

In short, once the training and testing set have been separated, we can choose a value for $k$. Next, we divide the training set into $k$ equally sized parts[5]. The next step is to be performed $k$ times; here we take $k-1$ of the parts and combine them into a temporary training set, and leave the last part as our testing set, and we perform validation on the testing set, and save the values. During each iteration, we must shuffle our parts such that each step has a unique training-testing split.

Finally, we can take all the calculated $MSE$ values, among others, and take their average. This leaves us with a well-rounded result *without* the need for more input data!

[4]This refers to calculating the $MSE$, variance, bias, and so on.

[5]We can have slightly unequal-sized parts without it being an issue, if the size of the dataset doesn't divide perfectly into $k$.

# 4 Results

## 4.1 Franke-Function

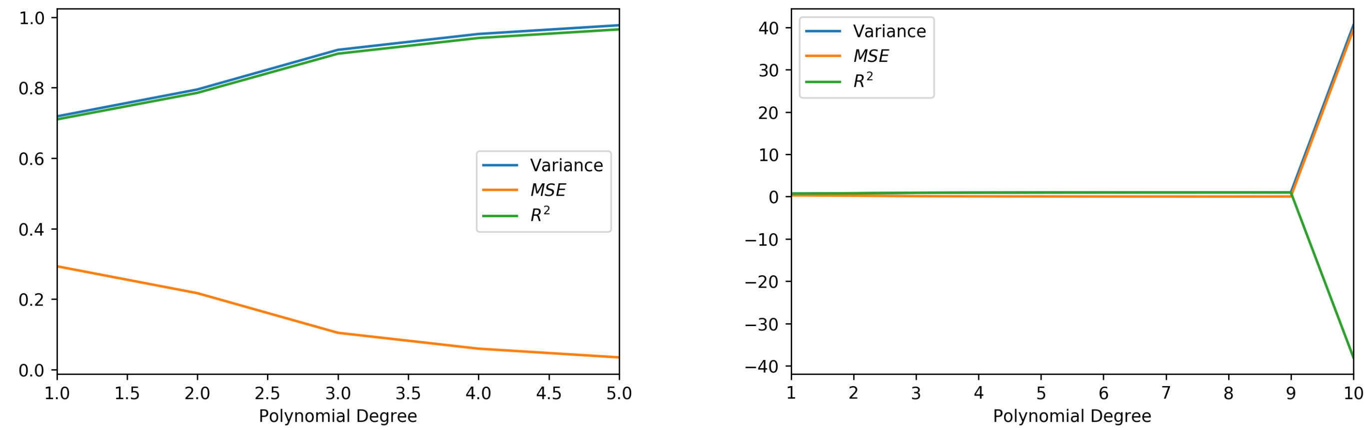


Figure 4: Plots of the mean square error, the $R^2$-score and the $\sigma$ variance of the $\beta$-values against the polynomial degree after performing a standard least square regression analysis on the Franke-function

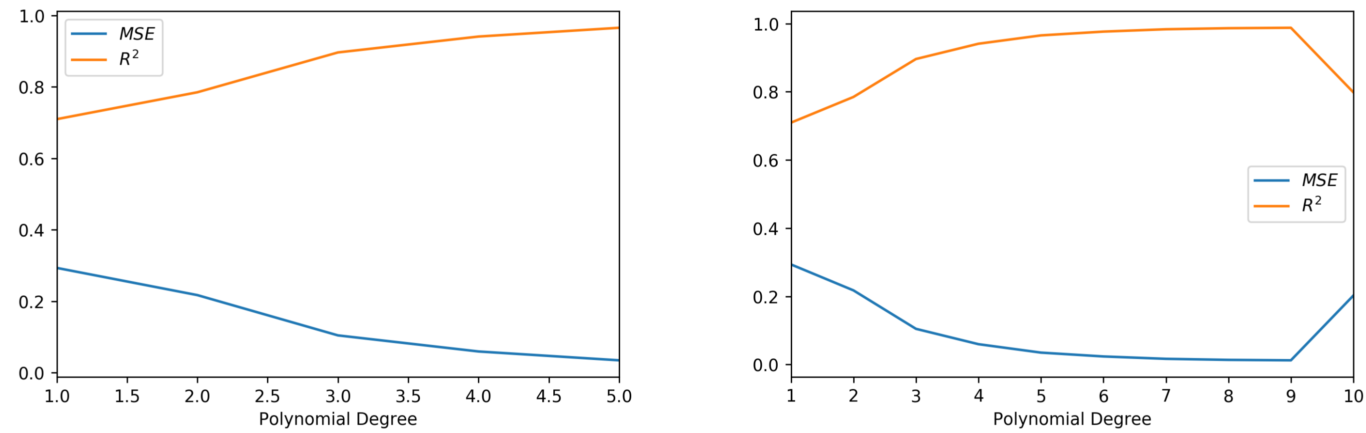


Figure 5: Plots of the mean square error and the $R^2$-score against the polynomial degree after performing a standard least square regression analysis on the Franke-function and performing a $k$-fold cross-validation

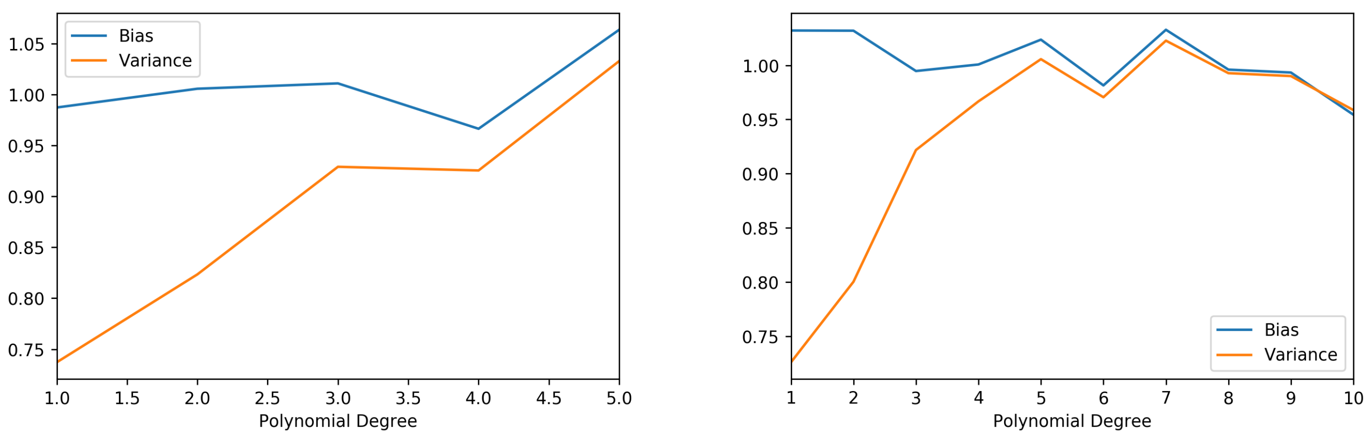


Figure 6: Plots of the bias and the varince against the polynomial degree after performing a standard least square regression analysis on the Franke-function

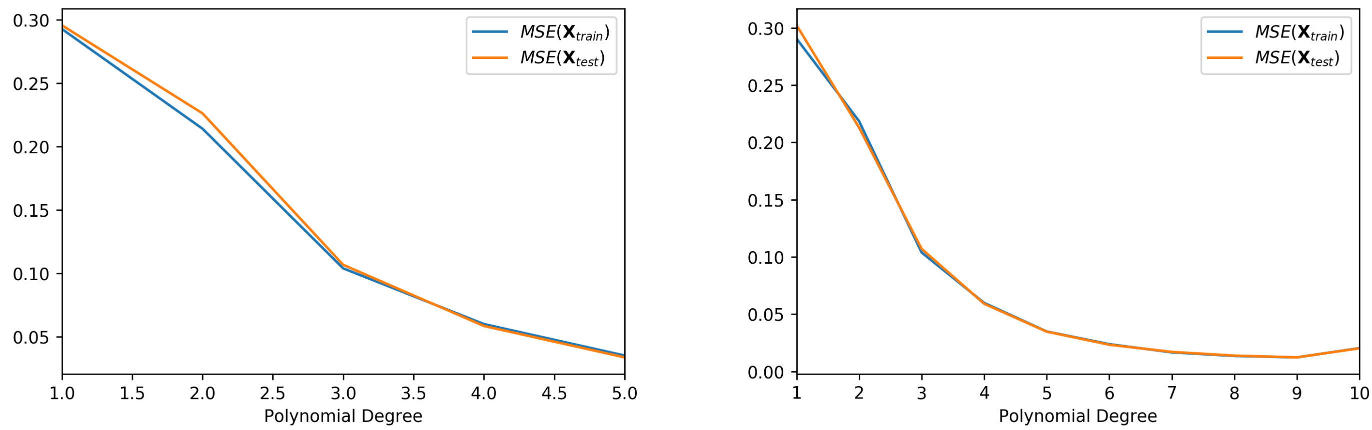


Figure 7: Plots of the mean square error for the training data and the testing data, against the polynomial degree after performing a standard least square regression analysis on the Franke-function

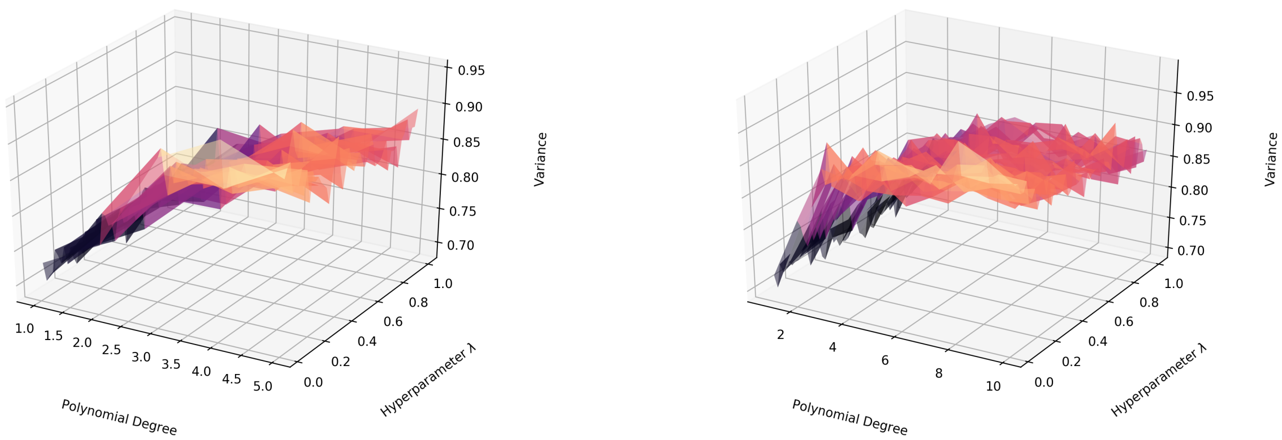


Figure 8: Plots of the varinace against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the Franke-function

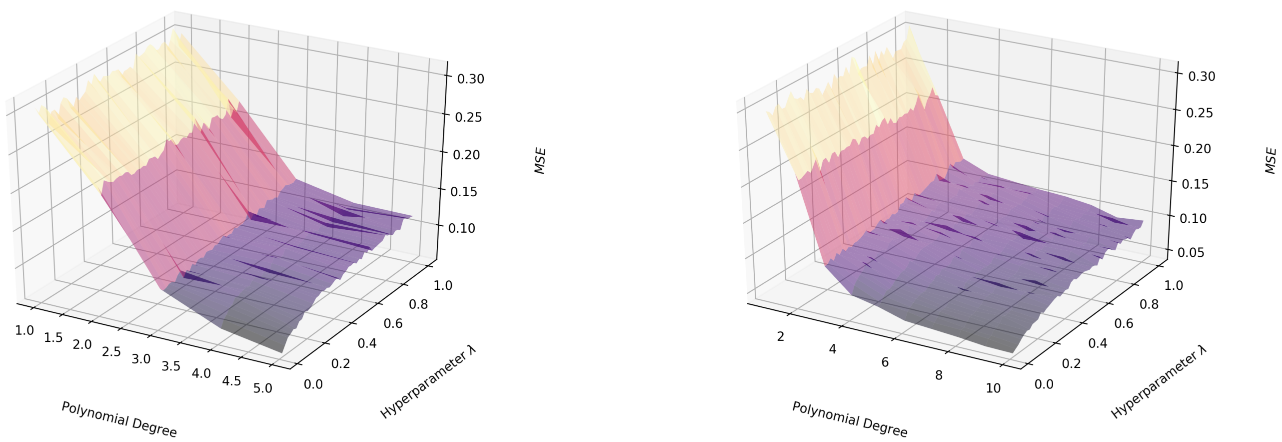


Figure 9: Plots of the mean square error against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the Franke-function

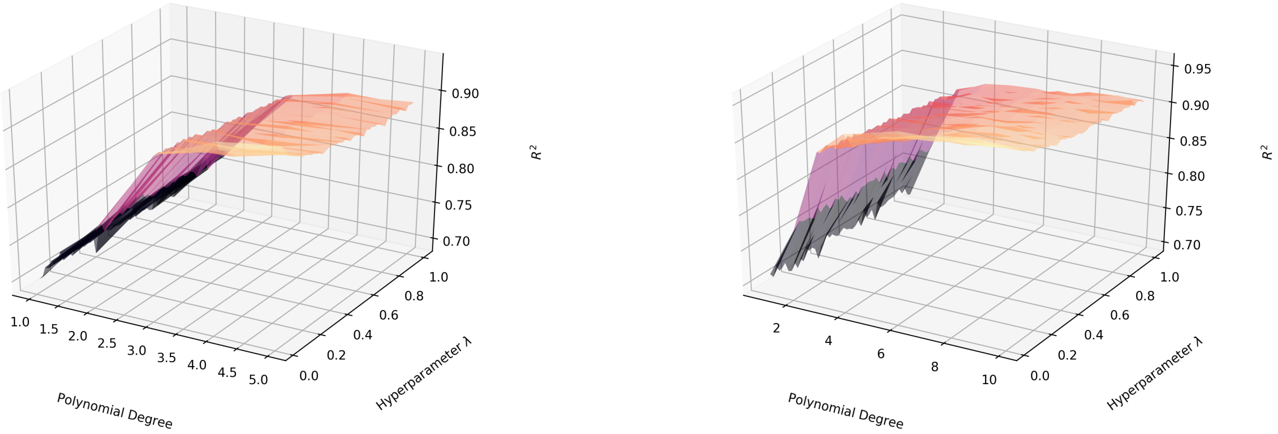


Figure 10: Plots of the $R^2$-score against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the Franke-function

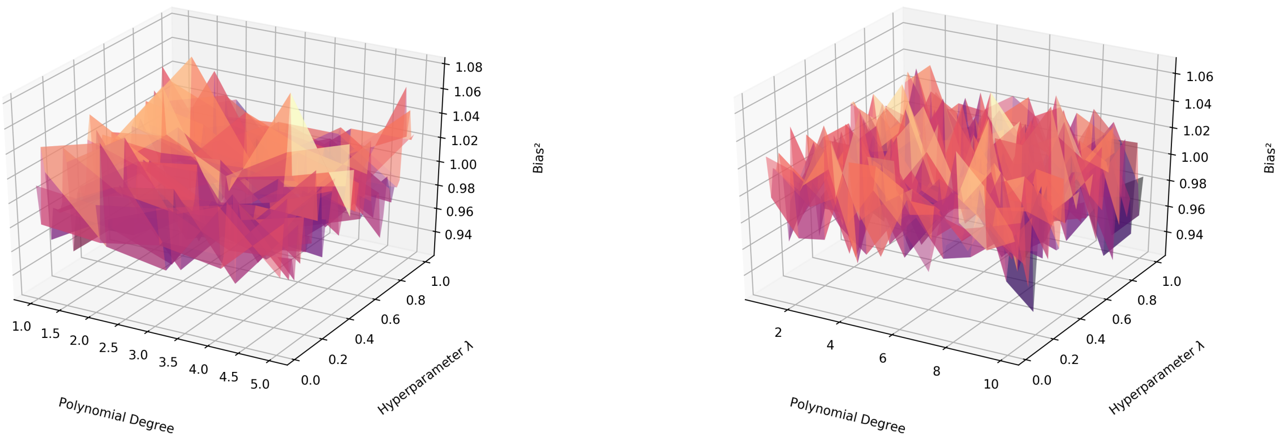


Figure 11: Plots of the bias squared against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the Franke-function

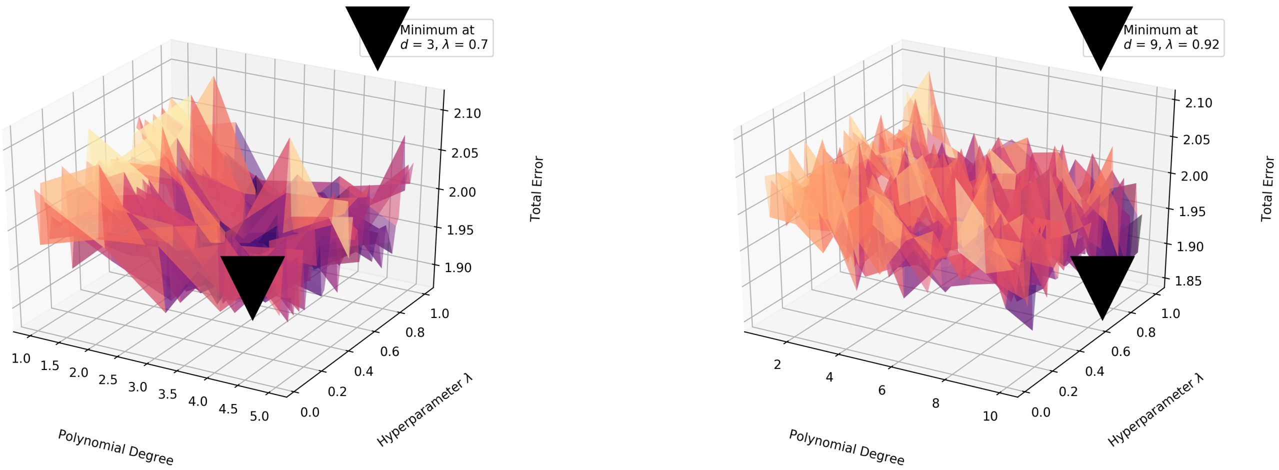


Figure 12: Plots of the total error against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the Franke-function

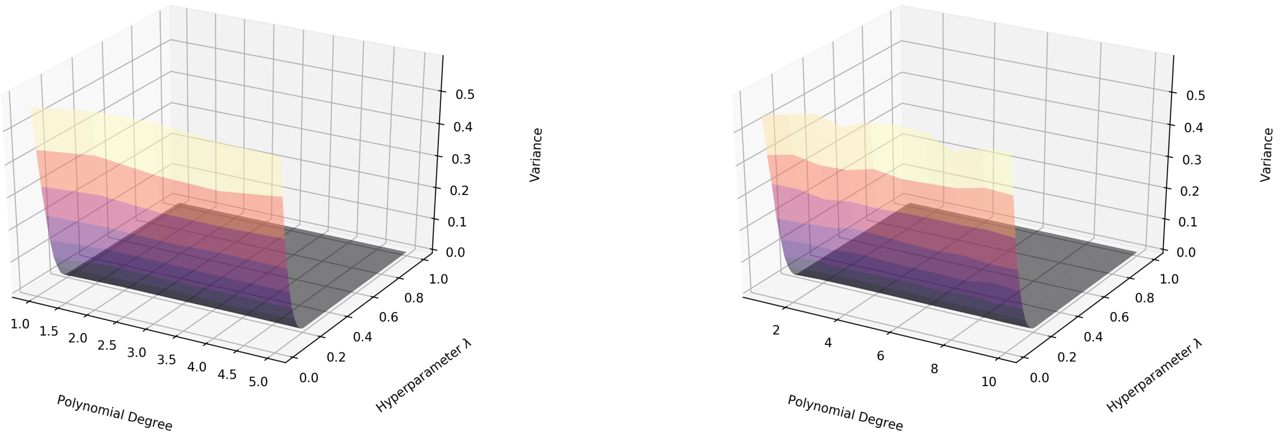


Figure 13: Plots of the varinace against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the Franke-function

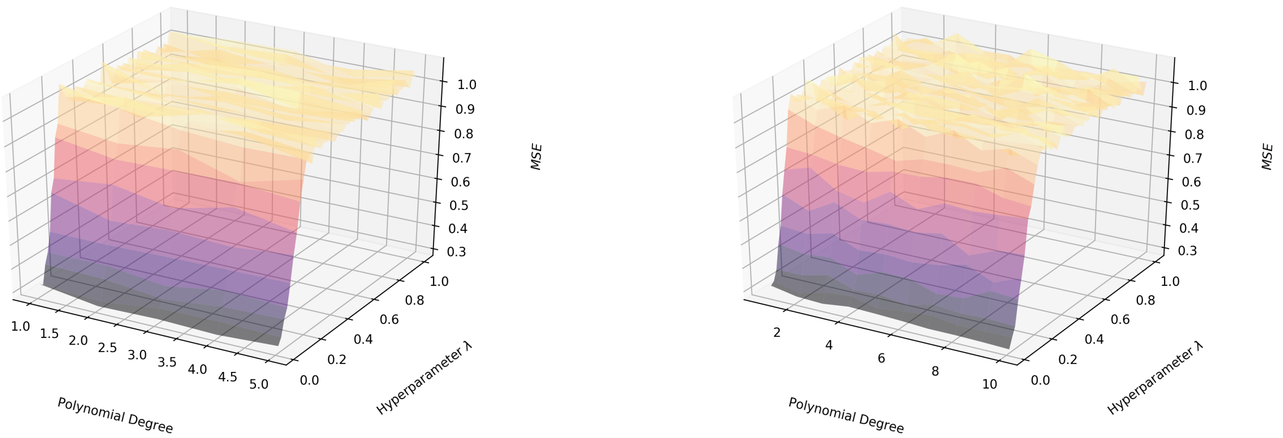


Figure 14: Plots of the mean square error against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the Franke-function

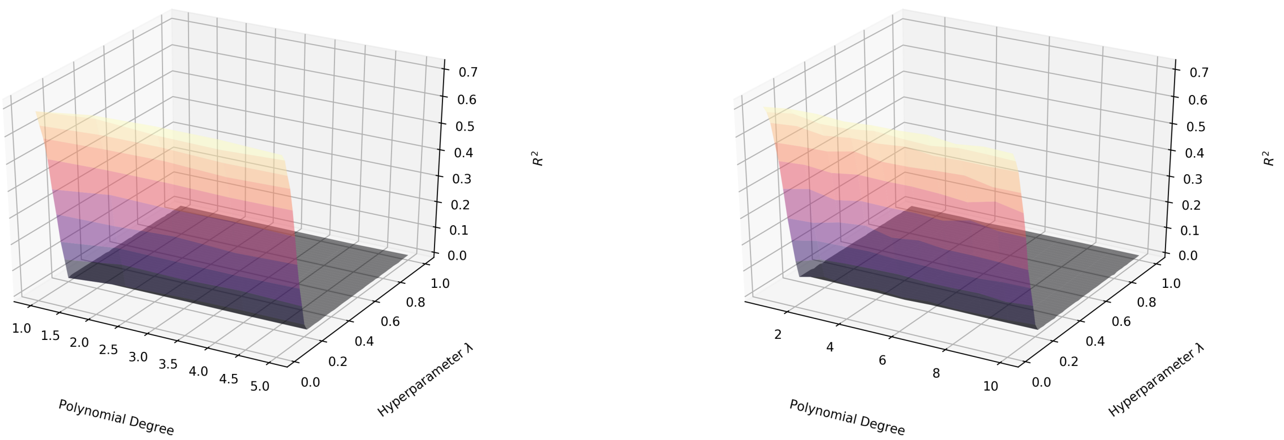


Figure 15: Plots of the $R^2$-score against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the Franke-function

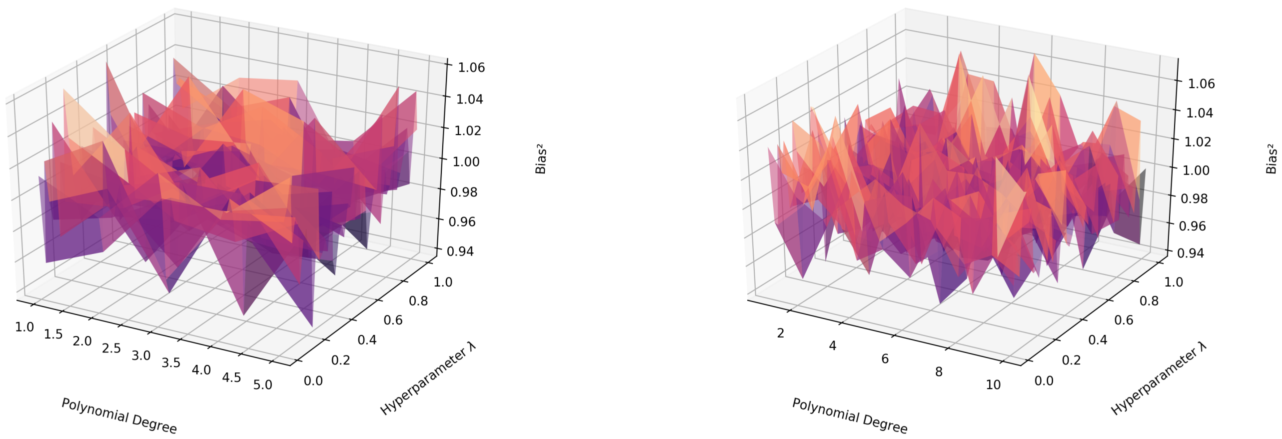


Figure 16: Plots of the bias squared against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the Franke-function

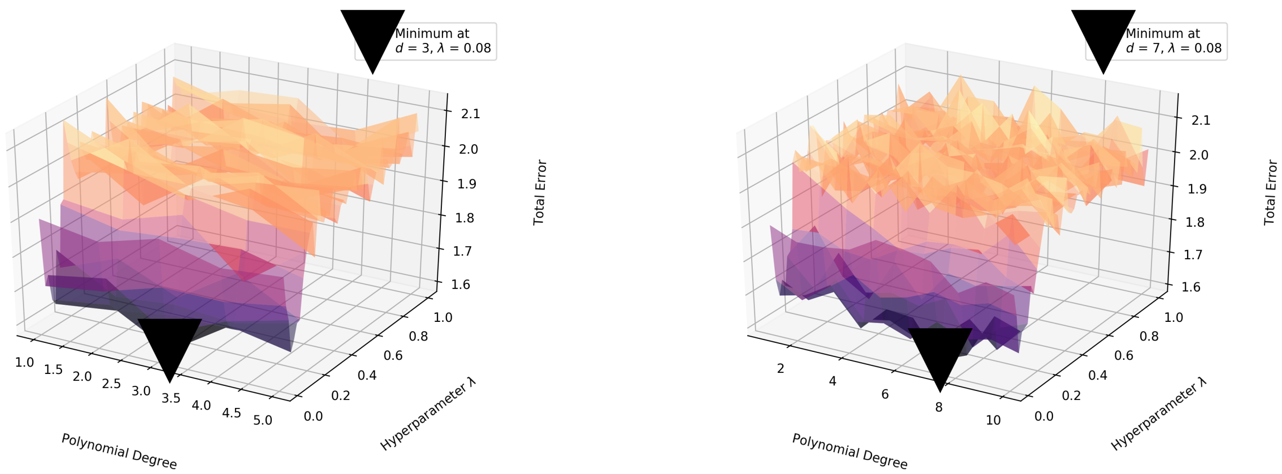


Figure 17: Plots of the total error against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the Franke-function

## 4.2 Real Data

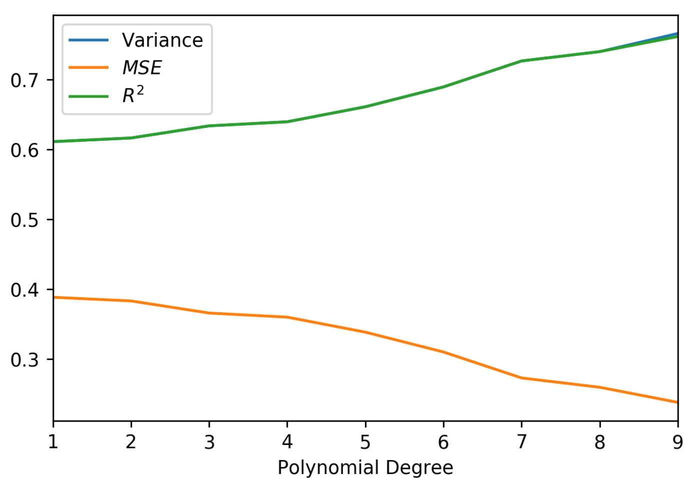


Figure 18: A plot of the mean square error, the $R^2$-score and the $\sigma$ variance of the $\beta$-values against the polynomial degree after performing a standard least square regression analysis on the real terrain data of Møsvatn Austfjell

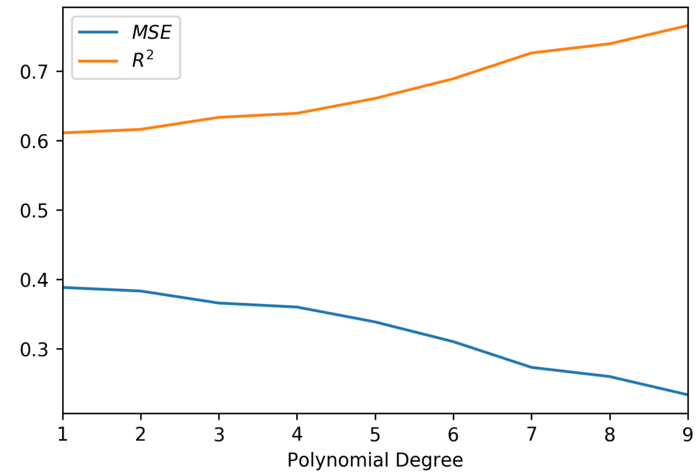


Figure 19: A plot of the mean square error and the $R^2$-score against the polynomial degree after performing a standard least square regression analysis on the real terrain data of Møsvatn Austfjell and performing a $k$-fold cross-validation

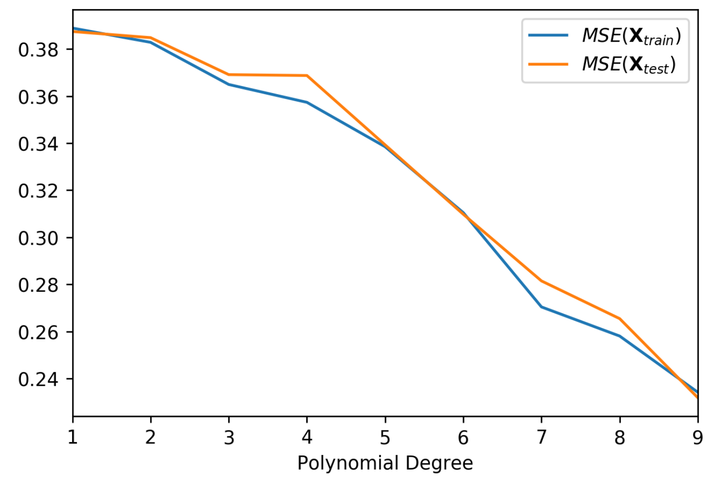


Figure 20: A plot of the mean square error for the training data and the testing data, against the polynomial degree after performing a standard least square regression analysis on the real terrain data of Møsvatn Austfjell

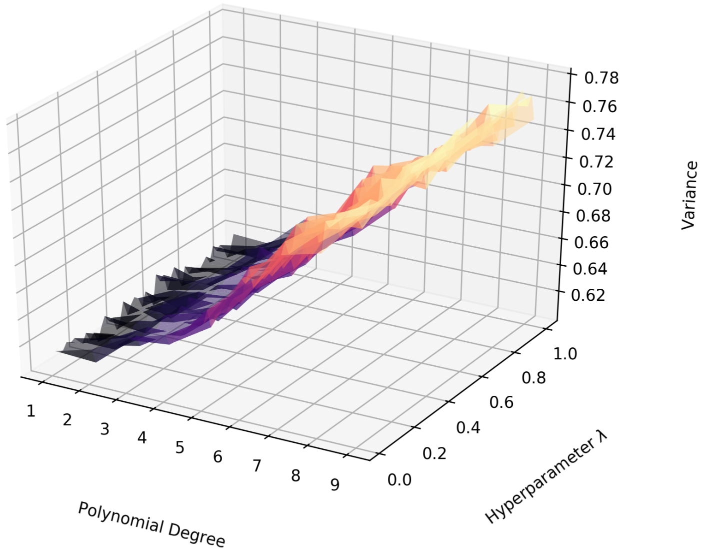


Figure 21: Plots of the varinace against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the real terrain data of Møsvatn Austfjell

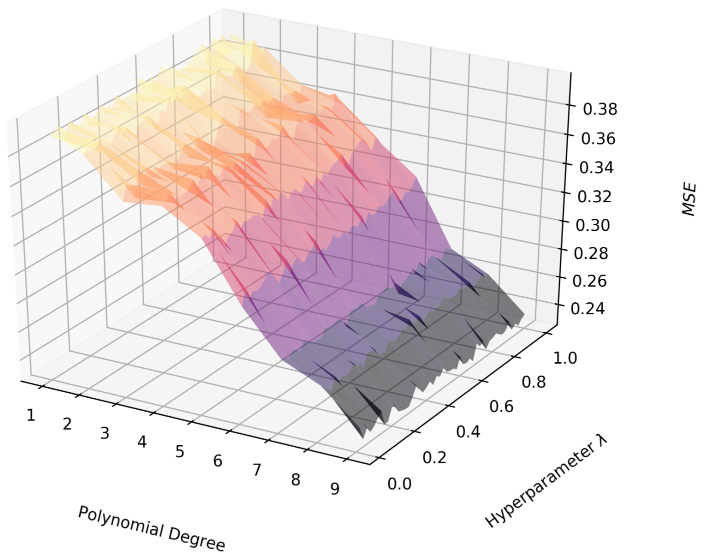


Figure 22: Plots of the mean square error against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the real terrain data of Møsvatn Austfjell

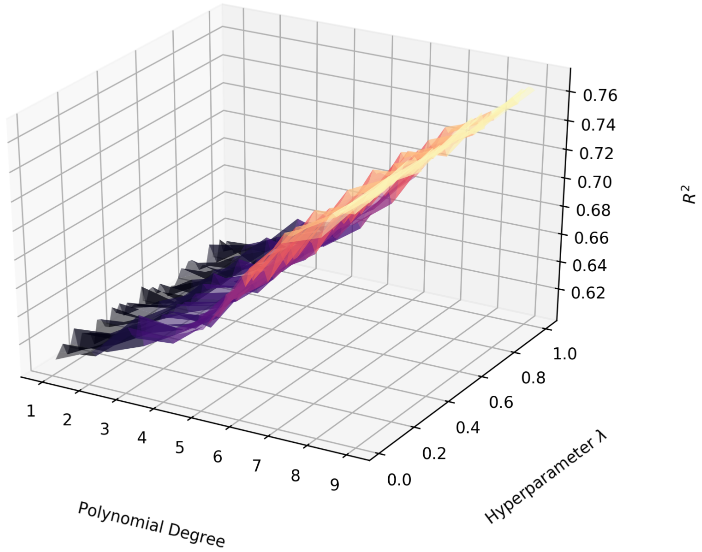


Figure 23: Plots of the $R^2$-score against the polyonomial degree and the hyperparameter $\lambda$ after performing Ridge regression on the real terrain data of Møsvatn Austfjell

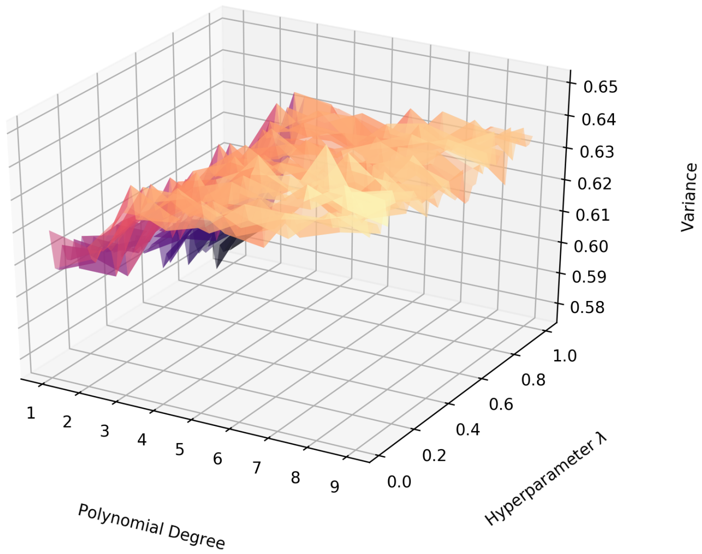


Figure 24: Plots of the varinace against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the real terrain data of Møsvatn Austfjell

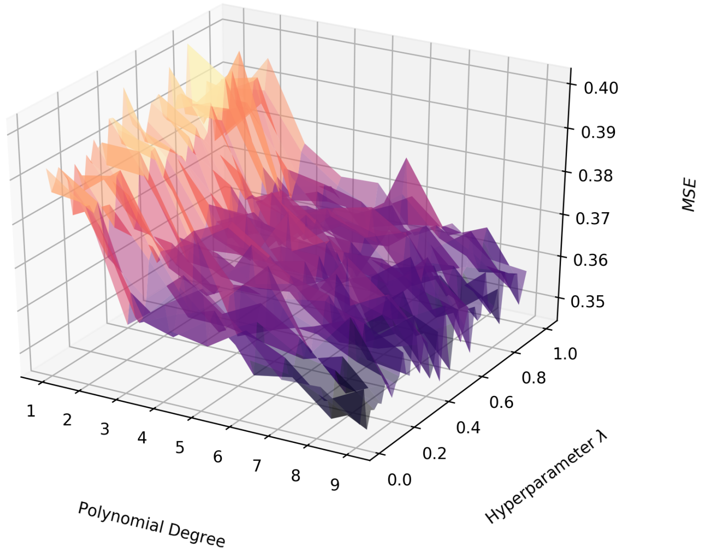


Figure 25: Plots of the mean square error against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the real terrain data of Møsvatn Austfjell

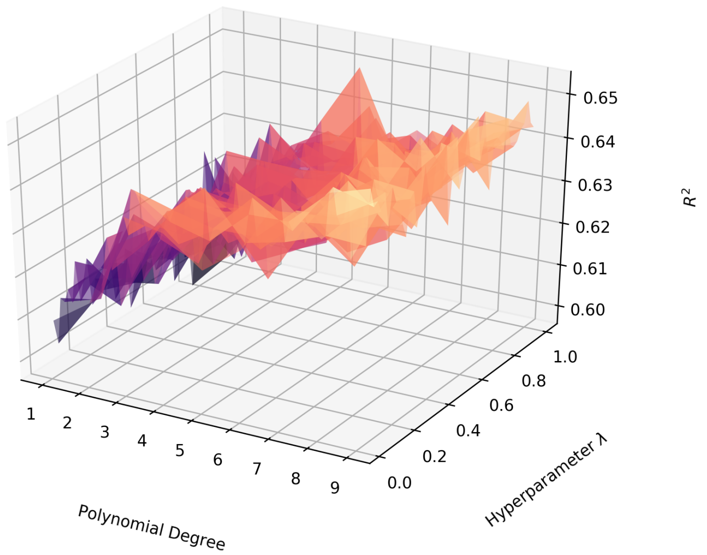


Figure 26: Plots of the $R^2$-score against the polyonomial degree and the hyperparameter $\lambda$ after performing Lasso regression on the real terrain data of Møsvatn Austfjell

# 5 Discussion

## 6 Appendix

## References