

# FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

October 7, 2019

## Abstract

dtcfvgbjhkgvfcdxszeasxrtdgyubhnijhugfvcdfguhij ftufykubhk nj jkdsi jkjk dsjknslj fd,nøj ka n,sdf jknm bnjdsajk nmnj mbfdjksa ,nfdsa mn n,asdf

## 1 Introduction

The purpose of this report is to analyze the performance of several regression methods – Ordinary Least Squares (OLS), Ridge Regression, and LASSO<sup>1</sup> Regression. In addition to implementing these regression methods, we will be validating (and cross-validating) our results by calculating the *mean squared error*, *bias*, and *variance*, giving us an understanding of how these models' predictive capabilities vary as functions of complexity.

Once we have a clear picture of how these regression models behave, we hope to determine which model best fits our input data. This will require analyzing our validation data and determining what complexity and hyperparameter minimizes our total error.

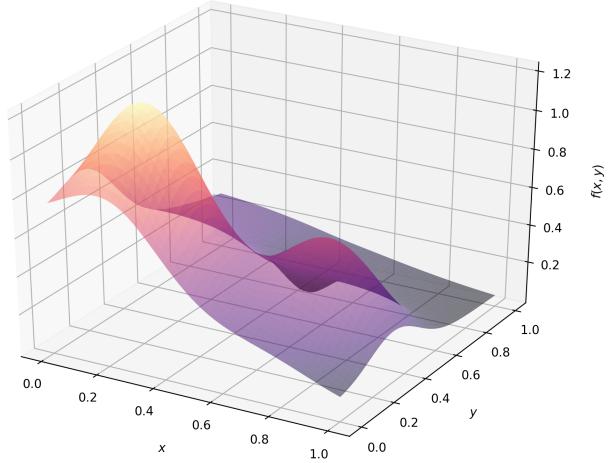


Figure 1: The *Franke function* for  $x$  and  $y$  values ranging from zero to one.

## 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:

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

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.

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.

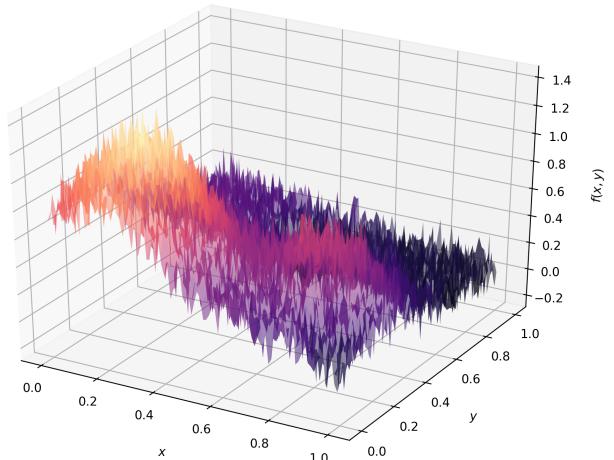


Figure 2: The *Franke function* for  $x$  and  $y$  values ranging from zero to one, with a Gaussian noise  $N(0, 0.01)$

<sup>1</sup>Short for *least absolute shrinkage and selection operator*.

## 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.

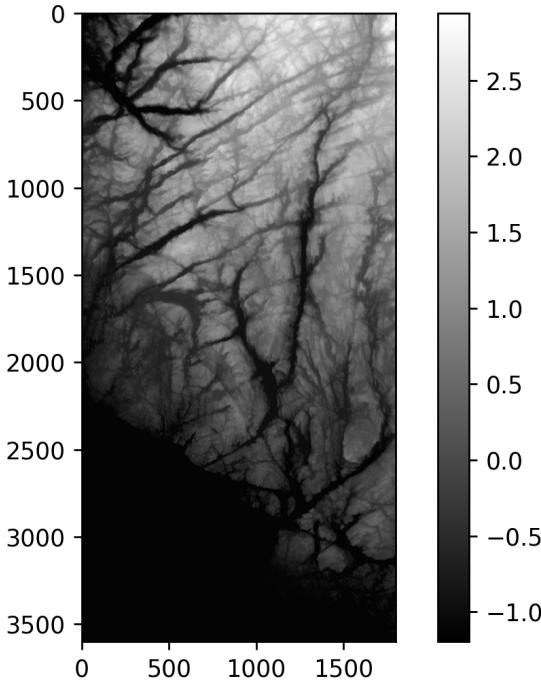


Figure 3: Altitude data from Møsvatn Austfjell, from the USGS website [?].

## 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:

$$f(x) = \beta_1 + \beta_2x + \beta_3x^2$$

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

$$f(x, y) = \beta_1 + \beta_2x + \beta_3y + \beta_4xy + \beta_5x^2 + \beta_6y^2$$

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

$$\begin{aligned} f(x, y, z) = & \beta_1 + \beta_2x + \beta_3y + \beta_4z + \beta_5xy \\ & + \beta_6xz + \beta_7yz + \beta_8x^2 + \beta_9y^2 + \beta_{10}z^2 \end{aligned}$$

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:

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

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<sup>2</sup> consisting of  $N$  datapoints per feature such that:

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

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*:

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

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

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

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:

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

<sup>2</sup>Meaning a set of  $p$  input features and 1 output.

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

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

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<sup>3</sup>, 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:

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

In matrix form, this can be rewritten:

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

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):

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

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.

### 3.4 LASSO Regression

---

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

```

1:  $z = \sum_i A_i^2$ 
2:  $i = 0$ 
3: while  $i \leq 500$  do
4:    $i = i + 1$ 
5:    $j = 0$ 
6:   while  $j \leq p$  do
7:      $\hat{y} = \sum_{k \neq j} \beta A_{*,k}$ 
8:      $\rho = \sum_k A_{*,k} (\mathbf{y} - \hat{y})$ 
9:     if  $\rho < -\lambda/2$  then
10:       $\beta_j = (\rho + \lambda/2)/z_j$ 
11:    else if  $\rho > \lambda/2$  then
12:       $\beta_j = (\rho - \lambda/2)/z_j$ 
13:    else
14:       $\beta_j = 0$ 
15:    end if
16:   end while
17: end while
```

---

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:

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

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).

### 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  $\hat{y}_i$ :

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

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 R<sup>2</sup> Score

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

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

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:

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

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  $\sigma$  and mean 0.

As a regression model increases in complexity<sup>4</sup>, it so happens that the *variance* of a prediction increases. Variance is defined as follows:

$$\text{Var}(\mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbb{E}[\mathbf{y}])^2 \quad (9)$$

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

---

<sup>4</sup>For a polynomial regression, this would refer to its *degree*.

---

<sup>3</sup>Meaning that  $\det(\mathbf{A}) = 0$

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

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 the square of (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:

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

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

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

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:

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

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

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

Where  $\frac{1}{n} \sum_{i=1}^n (f_i - \mathbb{E}[\hat{\mathbf{y}}])$  is 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<sup>5</sup> 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<sup>6</sup>. 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 Results

### 4.1 Franke-Function

Following below is several plots we made of our result after studying regression on the Franke-function. All the plots shows the results for both 1 to 5 polynomial degrees and 1 to 10 polynomial degrees.

Figure 4 to 7 details the ordinary least square method. Figure 4 shows the variance, mean square error and the  $R^2$ -score. Figure 5 shows the mean square error and the  $R^2$ -score after performing *k*-fold cross validation. Figure 6 shows the bias and the variance. Figure 7 compares the MSE of the training set and the testing set.

Figure 8 to 11 details the Ridge method. Here we also have plotted the results with  $\lambda$  differing between  $10^{-10}$  and 1. Figure 8 shows the varinace. Figure 9 shows the mean squared error, as well as where the minimum value was found. Figure 10 shows the  $R^2$ -score. Figure 11 shows the squared bias.

Figure 12 to 15 details the LASSO method. Here we also have plotted the results with  $\lambda$  differing between  $10^{-10}$  and 1. Figure 12 shows the varinace. Figure 13 shows the mean squared error, as well as where the minimum value was found. Figure 14 shows the  $R^2$ -score. Figure 15 shows the squared bias.

<sup>5</sup>This refers to calculating the *MSE*, variance, bias, and so on.

<sup>6</sup>We can have slightly unequal-sized parts without it being an issue, if the size of the dataset doesn't divide perfectly into  $k$ .

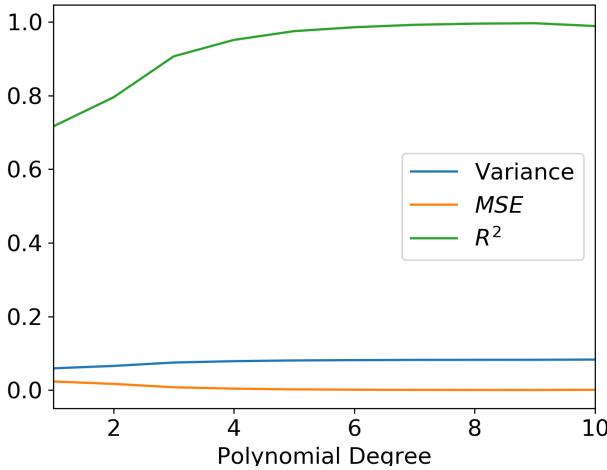


Figure 4: The  $MSE$ , the  $R^2$ -score, and the variance  $\sigma$  in the vector of coefficients  $\beta$ , as functions of the polynomial degree after performing OLS on the Franke-function

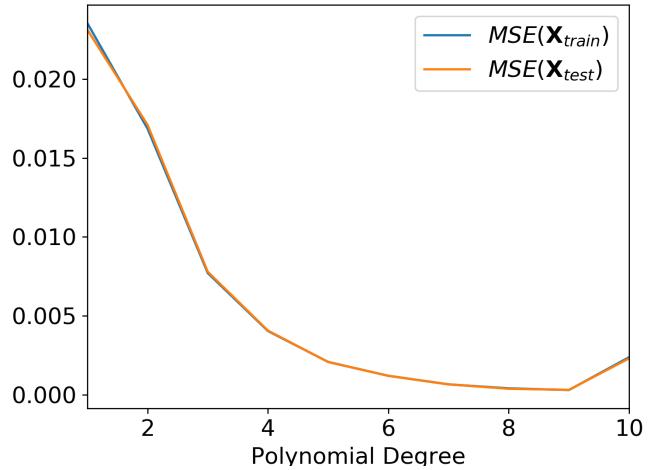


Figure 7: The  $MSE$  for the *training data* and the *testing data*, as a function of the polynomial degree after performing OLS on the Franke-function

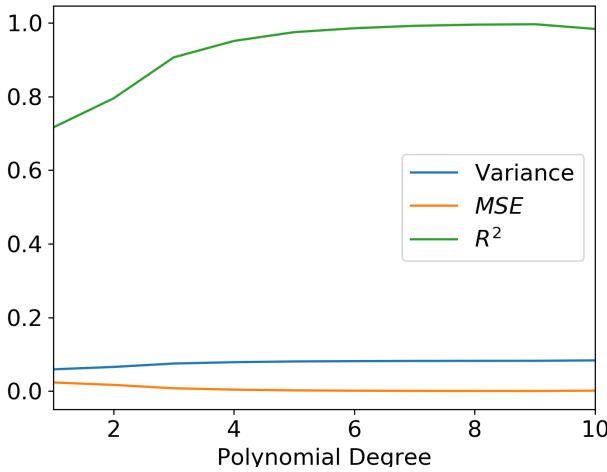


Figure 5: The  $MSE$ , the  $R^2$ -score, and the variance  $\sigma$  in the vector of coefficients  $\beta$  as functions of the polynomial degree after performing OLS on the Franke-function. Using 12-fold cross validation.

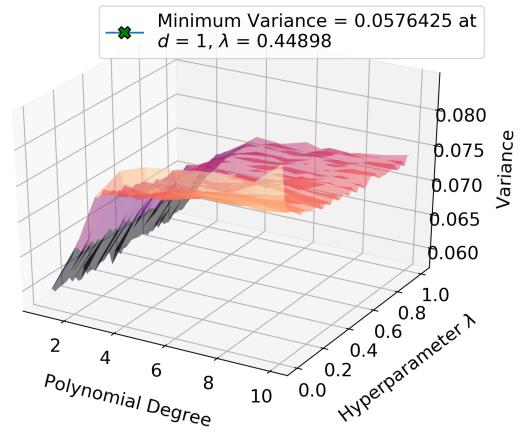


Figure 8: The *variance* as a function of polynomial degree and hyperparameter  $\lambda$ , after performing *Ridge regression* on the Franke-function

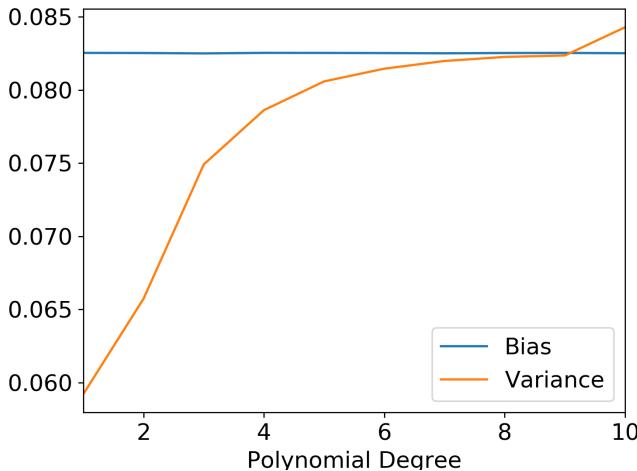


Figure 6: The *bias* and *variance* as functions of the polynomial degree after performing OLS on the Franke-function

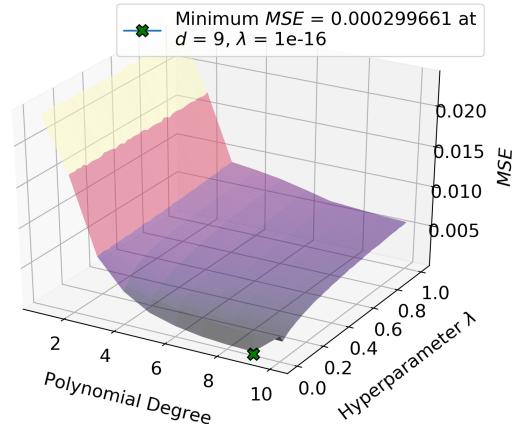


Figure 9: The  $MSE$  as a function of the polynomial degree and hyperparameter  $\lambda$  after performing Ridge regression on the Franke-function

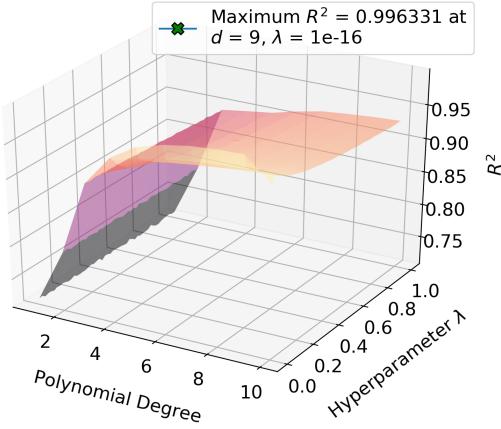


Figure 10: The  $R^2$ -score as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *Ridge regression* on the Franke-function

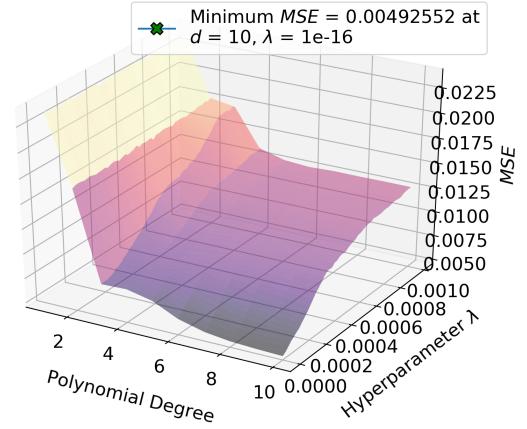


Figure 13: The  $MSE$  as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on the Franke-function

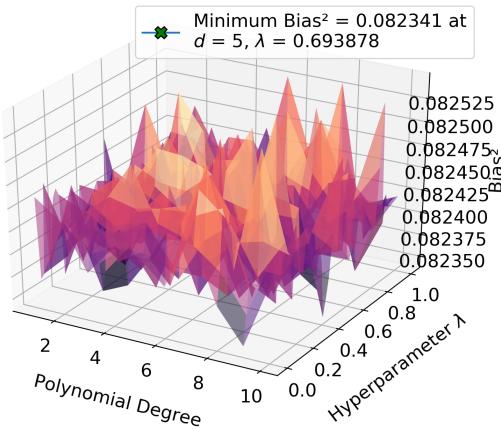


Figure 11: The *squared bias* as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *Ridge regression* on the Franke-function

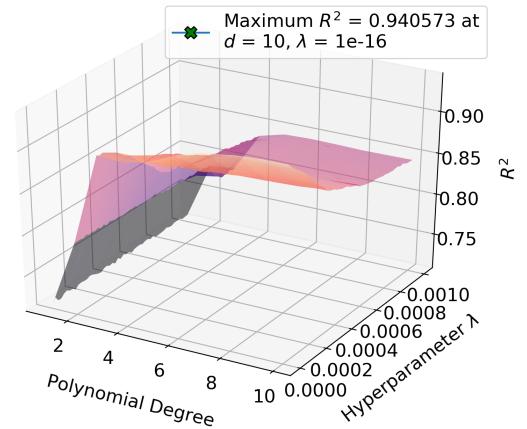


Figure 14: The  $R^2$ -score as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on the Franke-function

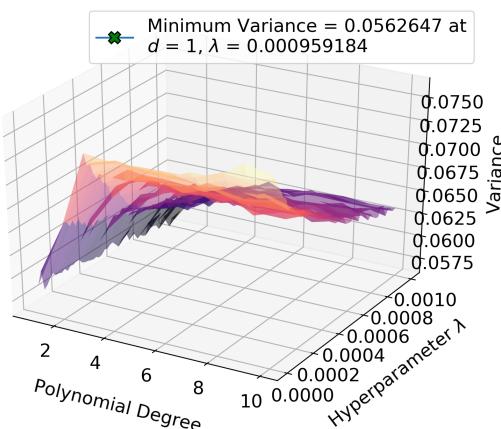


Figure 12: The *variance* as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on the Franke-function

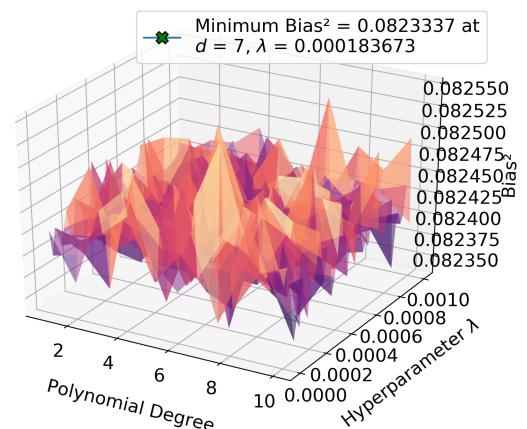


Figure 15: The *squared bias* as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on the Franke-function

## 4.2 Real Data

Following below is several plots we made of our result after studying regression on the real terrain data. All the plots shows the results for 1 to 10 polynomial degrees.

Figure 16 to 18 details the ordinary least square method. Figure 16 shows the variance, mean square error and the  $R^2$ -score. Figure 17 shows the mean square error and the  $R^2$ -score after performing  $k$ -fold cross validation. Figure 18 compares the MSE of the training set and the testing set.

Figure 19 to 21 details the Ridge method. Here we also have plotted the results with  $\lambda$  differing between  $10^{-10}$  and 1. Figure 19 shows the variance. Figure 20 shows the mean squared error, as well as where the minimum value was found. Figure 21 shows the  $R^2$ -score.

Figure 22 to 24 details the LASSO method. Here we also have plotted the results with  $\lambda$  differing between  $10^{-10}$  and 1. Figure 22 shows the variance. Figure 23 shows the mean squared error, as well as where the minimum value was found. Figure 24 shows the  $R^2$ -score.

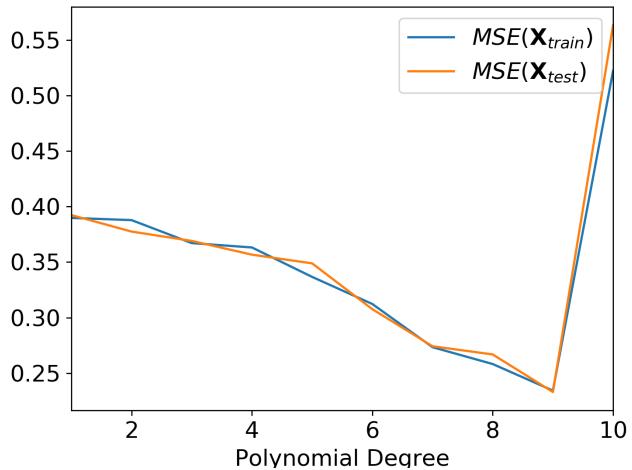


Figure 18: The  $MSE$  for the *training data* and the *testing data*, as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

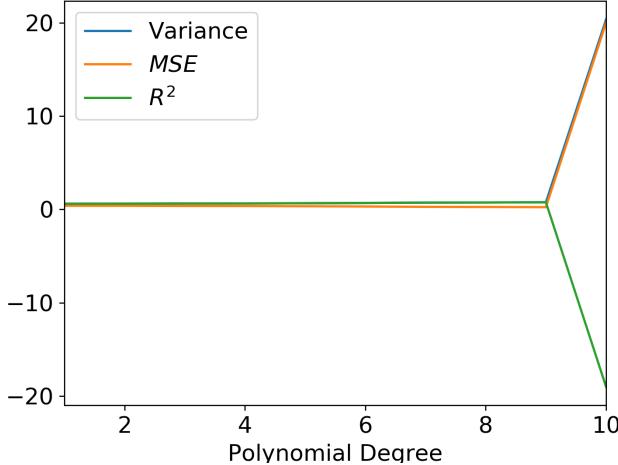


Figure 16: The  $MSE$ ,  $R^2$ -score and variance  $\sigma$  of the vector of coefficients  $\beta$  as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

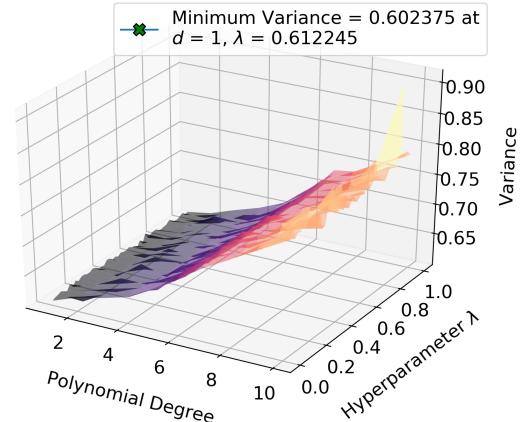


Figure 19: The *variance* as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

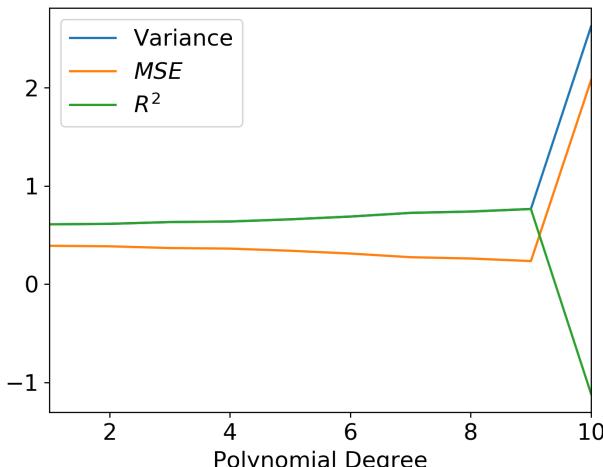


Figure 17: The  $MSE$ ,  $R^2$ -score and variance  $\sigma$  of the vector of coefficients  $\beta$  as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*. Using 12-fold cross validation.

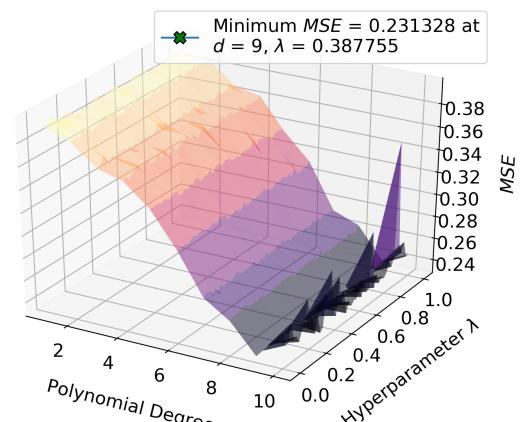


Figure 20: The  $MSE$  as a function of the polynomial degree and the hyperparameter  $\lambda$  after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

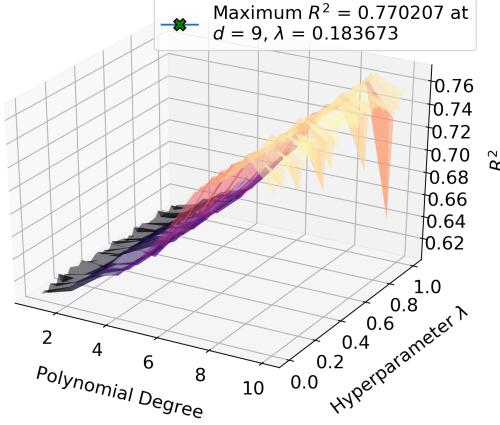


Figure 21: The  $R^2$ -score as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

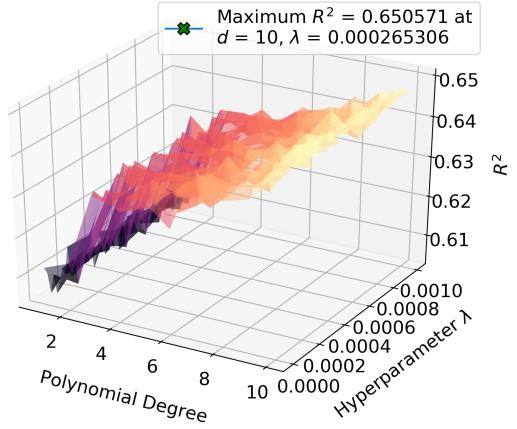


Figure 24: The  $R^2$ -score as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

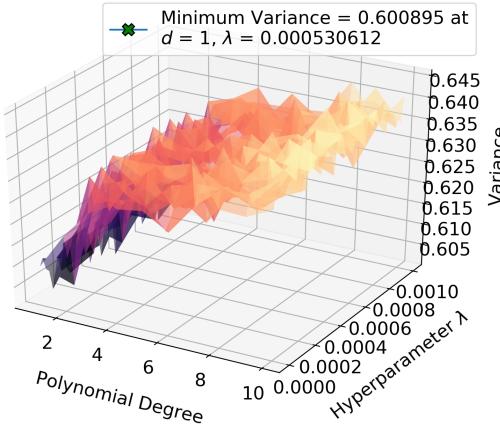


Figure 22: The variance as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

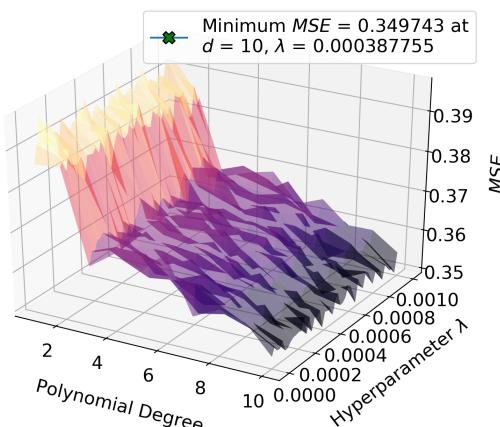


Figure 23: Plots of the MSE as a function of the polynomial degree and hyperparameter  $\lambda$  after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

## 5 Discussion

### 5.1 franke

Figure 4: varinace increases (bad?) as  $n$  increases , but  $MSE \rightarrow 0$  and  $R^2 \rightarrow 1$  (good). For  $n = 10$  we get overflow and weird results.

Figure 5: pretty similar results, no cross-validation give slightly better reults, except at  $n = 9$ , and cross-validation dosen't get overflow at  $n = 10$ .

Figure 6: not sure how to interpret this one. Also gives different results at  $n = 1$ , which it shouldn't. Bias and variance differs, not quite sure what result is desirable.

Figure 7: Pretty similar results for test and training, which I think is good, though train is slightly lower. Both have a minimum at  $n = 9$  and start increasing after that, as in figure 5.

Figure 8: Not moch to say here, the variance seems to flatten out as  $n$  increases, suggesting diminishing returns. Much difference in result based on  $\lambda$ .

Figure 9: Much the same as in figure 8. Minimum MSE happens at  $n = 9$ , which is before the overflow, suggesting that this is the highest  $n$  can be before overflow. Minimum is also at lowest  $\lambda$ , suggesting that an even lower value could give even better results.

Figure 10: Difference in results flattens out as  $n$  increases. Not much difference in result based on  $\lambda$ , but best result happens at lower values, with a seeming spike at the lowest, coinsiding with the MSE.

Figure 11: Can't see any particular pattern here.

Figure 12: Reults flattens out as  $\lambda$  suggesting diminishing results, but are highest for low  $\lambda$ .  $n$  dosen't seem to have a large impact on variance.

Figure 13:  $n$  has a small impact on results. MSE drops quickly for lower  $\lambda$ . Best value at highest  $n$  and lowest  $\lambda$ , suggesting a better result could be found for higher  $n$  and lower  $\lambda$ .

Figure 14:  $n$  has a small impact on results. Best value at lower  $\lambda$ , coinciding with MSE.

Figure 15: Same as figure 11.

OLS>Ridge>LASSO for MSE of Franke data. OLS and ridge best at  $n = 9$ , LASSO at  $n = 10$ . Best values at lowest

$\lambda$  as well.

### 5.1.1 real

Figure 16: Pretty much what was expected, variance and MSE increases and  $R^2$  decreases as  $n$  increases. Total values are lower than for Franke, meaning higher uncertainty/data that is harder to interpret/fit to polynomial, makes sense as it's real life data. More or less exact overlap between variance and  $R^2$ .

Figure 17: Results are slightly better than without cross-validation.

Figure 18: MSE of train and test is pretty close, though more variance in test.

Figure 19: Unlike Franke, the data doesn't flatten out for any particular variable

Figure 20: Best result not at a extreme for  $\lambda$ , suggesting either that the best result is within tested  $\lambda$ -values or relatively low impact from  $\lambda$ . Best result is at highest tested  $n$ , but there's unlikely to be a better result as  $n = 10$  gave overflow. Minimum MSE is smaller than for Franke, as figure 16.

Figure 21: Same as figure 19.

Figure 22: More spikes/variance than Ridge.

Figure 23: Best MSE is neither for lowest  $\lambda$  or highest  $n$ .

Figure 24:

Ridge > OLS > LASSO for real data. OLS and Ridge at  $n = 9$  and LASSO at  $n = 8$ .

we should maybe have a plot of made from the betas at the smallest MSE

## 6 Appendix

## References