UiO : Department of Physics
University of Oslo

# PROJECT 1
# Regression Analysis and Resampling Methods

FYS-STK4155

8th October 2019

Sagal Mohamed Abdullahi
Akuzike Ellina Banda

## Innholdsfortegnelse

## ***Abstract***

This project aimed at studying various regression methods including Ordinary Least Squares (OLS), Ridge and Lasso regression methods. The methods are in turn combined with resampling techniques: k-fold cross validation and bootstrap for proper assessment of the models. These regression methods were applied to a two-dimensional function, known as Franke's function and to digital terrain data for Iowa, a state in United States of America.

To assess the regression methods, the models were fit to different polynomial degrees. For ridge and lasso, all procedure was done with different lambda values to assess dependence on the bias (lambda). We studied bias-variance trade off, train test errors, confidence intervals for Beta values and Mean squared error (MSE) and R2 scores.

OLS was the model with the best fit for both Franke's function and the terrain data set. With low values of lambda and few data points, ridge performed as OLS. Lasso had the worst performance. OLs and Ridge had lower MSEs and higher R2 scores with an increase in polynomial degree while these values remained constant for different degrees for Lasso. Small lambda values gave better fit for ridge and lasso.

## ***Introduction***

Linear regression is a regression method that has been known for a very long time. It is very popular, and you find it written in innumerable many textbooks. It also serves as a starting point before diving into the more complicated statistical learning methods. In this project we have tried to get to know more about various regression methods including Ordinary Least Squares(OLS) , Ridge regression and finally Lasso regression.

The analysis we have done can be divided into two. In the first part we fit polynomials of different degrees to a specific two-dimensional function, called the Franke`s function. Franke`s function is a widely used function for testing different interpolation and fitting algorithms. After establishing our model, we will use the cross validation to perform a proper assessment of our models. We will also be studying the so called Bias-Variance trade-off.

In the other part of the report we analyse a terrain dataset obtained from U.S geological survey(USGS) through their website https://earthexplorer.usgs.gov/. We do much of the same analysis as we did for the Franke`s function. The report consists of introduction, method and result, discussion and a conclusion part. We will continue with the introduction by first going through theory of the different quantities, and then introduce the data and function we are using The Franke function, which is a weighted sum of four exponentials reads as follows

### *Theory*

#### Linear regression

If we have an input vector $X^T = (X_1, X_2, X_3, \dots . X_p\ )$ and want to predict an output Y we can describe it with the linear regression model as follows:-

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j B_j \qquad (1)$$

The $\beta_j$`$s$ are unknown coeficients, $f(X)$ represents the outcome Y and $X_j$ the design matrix. The design matrix contains variables that the output is dependent on, and contains p number of predictors. For example, if you have an experiment where you measure 5 different quantities, we say number of predictors are 5. If you make n = 20 measurments, we say that the number of data points are 20. In total, we can represent this in a design matrix with dimension $p \times n$. The variables of the design matrix can have different sources such as quantitative inputs, where you have different measurements that together give an output. We can also have an output that can be described using a polynomial representation, and that is what we will be using in our report. But, no matter the source of the design matrix, the model is linear in the parameters.
Equation(1) above, can be rewritten as follows:-

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon} \qquad (2)$$

$\boldsymbol{y}$ is a known quantity and represents as $f(X)$ in (1) the output. $\boldsymbol{X}$ is the design matrix, $\boldsymbol{\beta}$ is the coeficients while $\boldsymbol{\epsilon}$ is a vector containing the error in each measurement. In the above expression $\boldsymbol{\beta}$ and $\boldsymbol{\epsilon}$ are unknowns. And the question that is often asked is how to get the optimal value for the coefficients $\boldsymbol{\beta}$**.**

*OLS*

One very popular method of finding the coefficients is using ordinary least square error(OLS). Here we first predict a model using expression (3) below:

$$\widetilde{\boldsymbol{y}} = \boldsymbol{X\beta}. \qquad (3)$$

Then we try find the spread between the true values $\boldsymbol{y}$ and $\widetilde{\boldsymbol{y}}$ with help of our defined cost function in equation (4):

$$C(\boldsymbol{\beta}) = \frac{1}{n}\{(\boldsymbol{y} - \widetilde{\boldsymbol{y}})^{\boldsymbol{T}}(\boldsymbol{y} - \widetilde{\boldsymbol{y}})\}. \qquad (4)$$

By minimizing the cost function in (4) with respect to $\boldsymbol{\beta}$ we get the following solution:

$$\frac{\delta C(\boldsymbol{\beta})}{\delta \boldsymbol{\beta}} = 0 = \boldsymbol{X}^{\boldsymbol{T}}(\boldsymbol{y} - \boldsymbol{X\beta}),$$

and juggling around with the expression above we get the following solution for $\boldsymbol{\beta}$**:**

$$\boldsymbol{\beta} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} \qquad (5)$$

Confidence interval is a measurement that we often are interested in. It gives as a range of values such that with 95% probability, the range will contain the unknown true value. To find the confidence interval for the coefficient $\boldsymbol{\beta}$ we use the following expression:-

$$Confidense\ interval(\boldsymbol{\beta}) = \boldsymbol{\beta} \pm 2 \cdot SE(\boldsymbol{\beta}) \qquad (6)$$

Standard error, SE, is calculated by taking the square root of the variance. The variance is gotten by taking the variance of expression (5) so that it can be expressed as:-

$$Var(\boldsymbol{\beta}) = \sigma^2(\boldsymbol{X^T X})^{-\mathbf{1}} \qquad (7)$$

where $\sigma^2$ is the variance of the error $\boldsymbol{\epsilon}$ in (2). (Trevor, Robert Tibshirani, & Jerome, 2008)

### *Ridge*

Equation (5) above shows that finding optimal value for $\boldsymbol{\beta}$ involves taking the inverse of $(\boldsymbol{X^T X})^{-\mathbf{1}}$**.** From linear algebra we know we can`t calculate the inverse of a singular matrix, and thus obtaining the coefficients in (5) becomes challenging. Ridge regression is a method that solves this problem by adding a hyper parameter $\lambda$ and an identity matrix to equation (4) so that we get:

$$C(\boldsymbol{\beta}) = \frac{1}{n}\{(\boldsymbol{y} - \widetilde{\boldsymbol{y}})^{\boldsymbol{T}}(\boldsymbol{y} - \widetilde{\boldsymbol{y}})\} + \lambda\boldsymbol{\beta^T \beta}. \qquad (8)$$

Minimizing the cost function again with respect to $\boldsymbol{\beta}$ gives us:

$$\boldsymbol{\beta} = (\boldsymbol{X^T X} + \lambda\boldsymbol{I})^{-\mathbf{1}}\boldsymbol{X^T y}. \qquad (9)$$

The variance of the above expression is:

$$Var(\boldsymbol{\beta}) = [\sigma^2(\boldsymbol{X^T X} + \lambda\boldsymbol{I})^{-\mathbf{1}}\boldsymbol{X^T X}(\boldsymbol{X^T X} + \lambda\boldsymbol{I})^{-\mathbf{1}}]^{\boldsymbol{T}}\sigma^2(\boldsymbol{X^T X} + \lambda\boldsymbol{I})^{-\mathbf{1}}\boldsymbol{X^T X}(\boldsymbol{X^T X} + \lambda\boldsymbol{I})^{-\mathbf{1}}. \qquad (10)$$

Just like OLS, we can use the above expression to calculate SE so that we can get the confidence interval.
So, a question is what is the use of adding a hyper parameter lambda? Yes as I have mentioned we get to solve $\boldsymbol{\beta}$ for design matrixes that are singular. Also, for models in OLS that have high variance and low bias, we can tradeoff variance for bias, and get a more suitable model.
Another important thing for ridge regression is choosing a $\lambda$ value that gives the lowest possible error. As we will see later, having optimal value of lambda is essential for the ridge regression method. (Trevor, Robert Tibshirani, & Jerome, 2008)

### *Lasso*

Lasso regression is a method that also uses a hyper parameter lambda, but instead of $\boldsymbol{\beta}_j^2$ in equation

The Ideal scenario when analyzing a data is having many data points so that the model can be predicted well. With less data points a challenge arises for producing a good model. To understand this, you can think of training a kid to learn 100 names. If you teach him all the names, then he would be able to answered if he is asked to tell the names. If you else only know 20 names and teach him that, then the kid would not be able to answer all questions. The same is for the data points. The more data we have, the better the model we can make since the model has been trained well. With less data points, then the opposite is true.

So, how do we solve this problem? One way is doing what we call variable selection. We take a look at the predictors, and find out which are contributing less. Doing that, we get a less complex model that may give a good approximation. Lasso regression is a method that uses that logic. Just like Ridge it uses a penalty parameter, but instead of $\boldsymbol{\beta}_j^2$, $\lceil \boldsymbol{\beta}_j \rceil$ is used. The total equation is as shown below:

$$\boldsymbol{f}(\boldsymbol{X}) = \boldsymbol{\beta_0} + \sum_{j=1}^{p} \boldsymbol{X_j B_j} + \lambda \sum_{j=1}^{p} \lceil \boldsymbol{\beta_j} \rceil \qquad (11)$$

So $\lceil \boldsymbol{\beta}_j \rceil$ just like $\boldsymbol{\beta}_j^2$ shrinks the coefficients. It also in addition set some of the $\boldsymbol{B_j}$ values equal to zero when $\boldsymbol{\lambda}$ is big enough. That is why it is said that lasso performs a variable selection. The model that is yield is *sparse*, and only a subset of variables are used for prediction.
Just like ridge, choosing optimal value for $\lambda$ is essential. (Trevor, Robert Tibshirani, & Jerome, 2008)

## Assessing accuracy of model

The linear regression methods I have talked about need to tested, and seen how well the models that are predicted fit the data. To do so, two quantities are assessed:-

a) MSE
b) R$^2$ statistics

### *MSE*

The mean squared error estimates the spread between the predicted model and the measurement as equation (4) does. It tells us the average amount the response will deviate from the true regression line. The equation can be expressed as:-

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{y}(x_i)\right)^2. \qquad (12)$$

$y_i$is the measurement value and $\hat{y}(x_i)$ represents the predicted model. Later if we divide the data into a test and train, and predict the model using the train data, then we can see how good the the predicted model is by testing it with the test data.
Equation (12) is the expectation value of

$$\left(y_i - \hat{y}(x_i)\right)^2$$

Writing the above as vectors, we have that:

$$\begin{gathered} y_i \rightarrow \boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon} = \boldsymbol{f} + \boldsymbol{\epsilon} \\ \left(y_i - \hat{y}(x_i)\right)^2 \rightarrow \tilde{\boldsymbol{y}} = \boldsymbol{X\beta}. \end{gathered}$$

The expectation value can then be written as:

$$E|(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2| = E|(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2|$$

By adding and subtracting the expectation value of $\tilde{\boldsymbol{y}}$, and doing a bit computation the following result is gotten:

$$E|(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2| = E\left|\left(\boldsymbol{y} - E(\tilde{\boldsymbol{y}})\right)^2\right| + Var(\tilde{\boldsymbol{y}}) + \sigma^2 \qquad (13)$$

The first term is what we call the squared bias term, and gives an indication of how far the predicted model is from the measurement values. The second term is the variance , and tells us how the variance of our predicted model is. The last term is a measure of the variance in $\boldsymbol{\epsilon}$. In the prediction of our model, there is not much we can due about the variance in the $\boldsymbol{\epsilon}$. But, we can modulate the bias and variance term according to the complexity we are fitting the model, and also the linear regression method we are using. We will se more of this later in the report. (Trevor, Robert Tibshirani, & Jerome, 2008)

### *R² statistics*

R$^2$ statistics is an accurancy model used to see how much proportion of the variance of the data that is explained by the regression method. It takes values between 1, and 0. So, a R$^2$-score equal to 0.6 says that 60% of the data is explained by the regression method. The R$^2$-can be calculated using the equation below:-

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}\left(y_i - \hat{y}(x_i)\right)^2}{\sum_{i=0}^{n-1}\left(y_i - \bar{y}(x_i)\right)^2}. \qquad (14)$$

$\bar{y}$ in equation(14) is the mean value of the true values $y_i$. (Trevor, Robert Tibshirani, & Jerome, 2008)

## Resampling techniques

Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest on each sample to obtain additional information about the model (JAMES). Two most commonly used resampling techniques are:

a) Cross-validation
b) Bootstrap.

### *Cross-validation*

Cross-validation (CV) is a method used to estimate test (prediction) error by holding out a part of the training observations from the fitting process and then applying statistical learning methods to the observations held out (Casella, Fienberg, & Olkin, 2013). This method estimates the average generalization error when the method f(x) is applied to an independent test samples from the joint distribution of X and Y. (Trevor, Robert Tibshirani, & Jerome, 2008) One strategy used in Cross-validation is the Validation set approach where the set of observations id randomly divided into two parts: training and validation sets. The model is fit on the training set. The fitted model is used to predict responses for observations in the validation set (James). K-fold cross-validation randomly divides the set of observations into k equal groups, referred to as folds. It is mostly performed with k=5 or k=10 for computational advantages. The procedure is repeated K times, while changing the fold used for validation each time.. For each iteration, the model is fit on a training set made up of k-1 folds. The remaining last fold is regarded as the validation set. K Mean Squared Errors MSEs are calculated. The k-fold CV estimate is computed by averaging these values (Casella et al., 2013).

### *Bootstrap*

Bootstrap is a powerful statistical tool used to assess accuracy associated with a given estimator or learning method. As CV, bootstrap seeks to estimate conditional error but estimates well only the expected prediction error (Hastie, 2009).This method randomly draws train datasets with replacement from a given observation set. Each training set is the same size as the original observation set. Drawing training sets is done B times. The model is fit to each of the training sets. The original observation set is regarded as the test set for each replication. An estimator of choice: prediction error, variance, or bias, is calculated for each bootstrap replication. The average estimator from the bootstrap process is calculated to assess its accuracy. (Trevor, Robert Tibshirani, & Jerome, 2008)

## *Data*

### Terrain Data

The terrain dataset used for this project was obtained from U.S. Geological Survey (USGS) through their website https://earthexplorer.usgs.gov/. The dataset covers the terrain over Iowa, a state in the midwestern part of United States of America. We obtained this data in the specified Shuttle Radar Topography Mission (STRM) Arc-Second Global format as a GEOTIFF file . It is of the dimension 3601 X 3601 points. The terrain is presented in figure 1. As was advised, the first task we carried out on the data preprocessing through normalization. From the normalized dataset, equal dimensions were selected for the x and y axes to be used for various analyses. If we consider our normalized dataset to be z and number of data points selected for each dimension as n, then: ***Z = Z[:n, :n]*** was the selected dataset. The size, n was chosen differently each time from the set ***n={10, 25, 100, 1000}*** depending on the expected execution time of the analyses being carried out at that moment. Figure 2 presents a plot of the normalized section of the terrain that was selected.

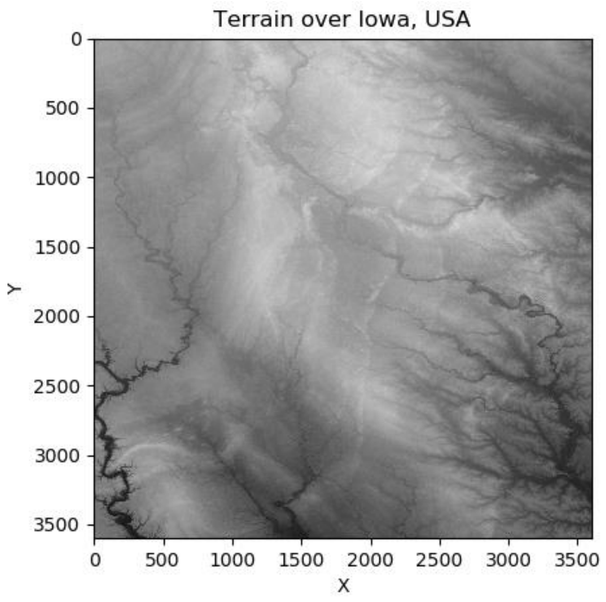


*Figure 1: TerFigur 1rain over Iowa*

### Franke`s function

The Franke`s function we observe and analyse in the project can be written as:-

(14)

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

In the project we define x, y ∈ [0, 1]. A design matrix is made, and then the different regression methods are used to predict a model.

## Result & Method

Here we will be presenting our results, and at the same time describe the methods we have used. Since we study a function (14), and a terrain data set, we will do presentation and description separately for each.

### *Franke`s function results & method*

#### OLS

For the Franke`s function we started off by testing it with the OLS method. To get the best possible prediction model we calculated the MSE, and R2-score for different degrees. MSE can be calculated using equation (12) whereas R2-score can be obtained by using (14). We got the result shown in table (1) below.

| *Degree* | *MSE* | *R2 SCORE* |
|---|---|---|
| **1** | 0.032 | 0.438 |
| **2** | 0.027 | 0.5829 |
| **3** | 0.018 | 0.7565 |
| **4** | 0.013 | 0.82 |
| **5** | 0.012 | 0.85 |
| **6** | 0.011 | 0.86155 |
| **7** | 0.0108 | 0.8667 |
| **8** | 0.0104 | 0.871 |
| **9** | 0.01090 | 0.868371 |
| **10** | 0.01075 | 0.868 |
| **11** | 0.0107 | 0.8697 |
| **12** | 0.011 | 0.8642 |
| **13** | 0.011 | 0.8641 |
| **14** | 0.011 | 0.8567 |

*Table 1: Shows MSE, and R2 score calculated using our cross validation plot. Number of data points are 1600. Also, noise in the sample is 0.4.*

To do the OLS regression we first calculated $\boldsymbol{\beta}$ using equation (5). We then used equation (3) to calculate the predicted model, and by using the Franke`s function and (3) we could get the R2-score, and the MSE.
The design matrix we used had dimension $p \times n$. $p$ is the number of predictors, i.e. polynomial order, and n is the number of data points. Since our plot was a 3D plot, n =1600 points meant

that we had respectively 40 points both x, and y-axes. That is what we have chosen in this project. Also added noise through out was 0.1.

After that getting the MSE and R2-score we plotted the Franke`s function and the OLS fit as shown in figure (2), and (3) below respectively. We plotted using a polynomial degree = 5 since as shown in table(1) above it had a relatively high R2-score, and also a low MSE value compared to other degrees. Even though degree 8 gave the highest R2-score and lowest MSE-value, we opted to go for the simplest possible model. As we will se later, we can use it for comparing.

To calculate the MSE, and R2-score we used k-cross-validation as a resampling technique. We made our own code, that can be found at our Github in the map code, fold Franke`s function and python code called *Proj1OLS.py.* There lies also all other relevant code for the Franke`s function.

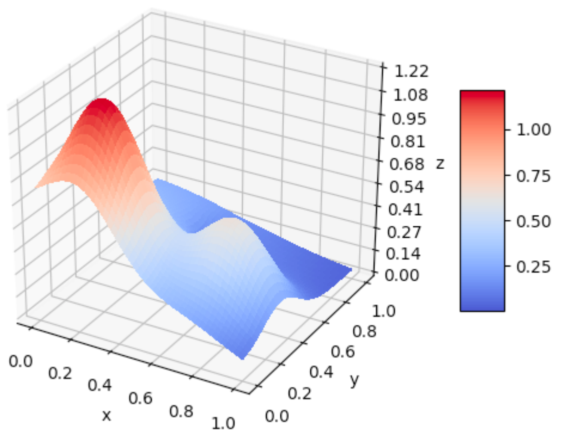


*Figure 2: Plot showing the Franky function. The function is plotted up to order 5. The number of n points are 40, thus meaning we have 1600 points. Also added noise is 0.1.*

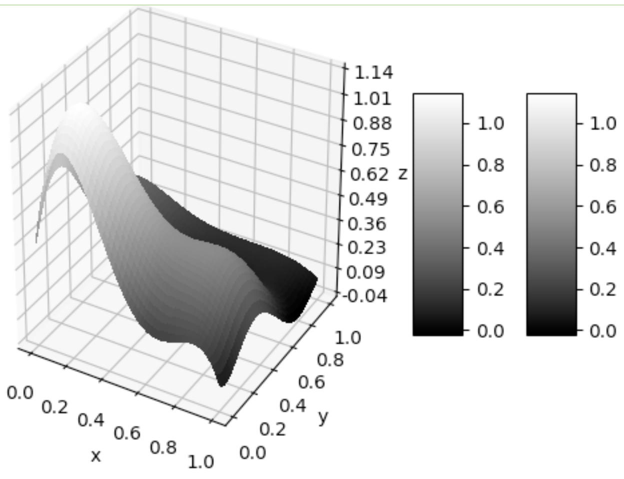


*Figure 3: Plot showing the Franky function fitted with OLS. The function is plotted up to order 5. The number of n points are 40, thus meaning we have 1600 points. Also added noise is 0.1.*

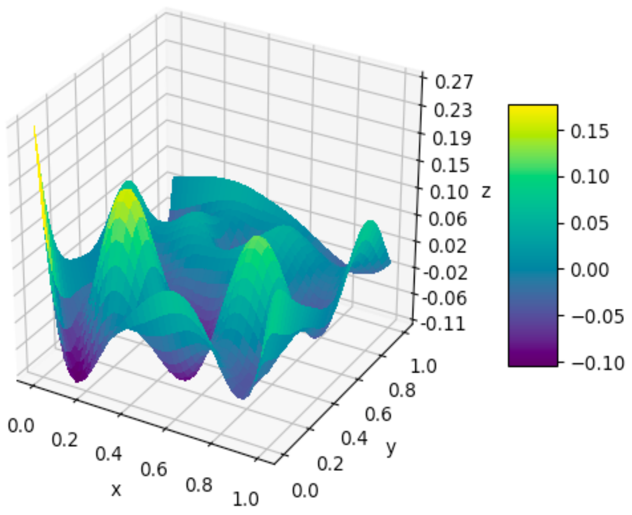


*Figure 4: Shows error between franky function, and the OLS fit. We see that plot generelly agrees with what is shown i table(1). n = 40, thus meaning we have 1600 points. Added noise is 0.1*

Then we went off to calculate the beta values, the variance and confidence interval for beta. To calculate variance, we used equation (7), and (6) to calculate the confidence interval. The result we got is shown in table (2) The code we have used til noe can be found at the Github link provided in the abstract. It is called *ProjOLS.py*.

| $x^j y^j$ | $\beta$ | $VAR$ | Confidence interval |
|---|---|---|---|
| $x^0y^0$ | 0.43 | 0 | $[0.49, \quad 0.37]$ |
| $x^1y^0$ | 7.89 | 0.11 | $[8.55, \quad 7.22]$ |
| $x^0y^1$ | 3.51 | 0.11 | $[4.17, \quad 2.85]$ |
| $x^0y^2$ | -7.94 | 2.71 | $[-4.7, \quad -11.17]$ |
| $x^2y^0$ | -34.64 | 2.71 | $[-31.4, \quad -37.86]$ |
| $x^1y^1$ | -15.2 | 1.65 | $[-12.68, \quad -17.7]$ |
| $x^3y^0$ | 48.96 | 1.39 | $[56.25, \quad 41.66]$ |
| $x^3y^0$ | -7.99 | 1.39 | $[-0.7, \quad -15.28]$ |
| $x^2y^1$ | 46.03 | 7.55 | $[51.42, \quad 40.64]$ |
| $x^1y^2$ | 19.46 | 7.55 | $[24.85, \quad 14.07]$ |
| $x^4y^0$ | -24.64 | 1.52 | $[-17, \quad -32.28]$ |
| $x^0y^4$ | 27.44 | 1.52 | $[35.07, \quad 19.80]$ |
| $x^2y^2$ | -10.01 | 7.58 | $[-4.61, \quad -15.40]$ |
| $x^1y^3$ | -26.59 | 8.78 | $[-20.78, \quad -32.4]$ |
| $x^3y^1$ | -53.99 | 8.78 | $[-48.17, \quad -59.8]$ |
| $x^5y^0$ | 2.05 | 2.33 | $[5.05, \quad -0.94]$ |
| $x^0y^5$ | -15.28 | 2.33 | $[-12.82, -18.7]$ |
| $x^4y^1$ | 18.8 | 1.75 | $[21.4, \quad 16.2]$ |
| $x^1y^4$ | 14.97 | 1.75 | $[17.57, \quad 12.38]$ |
| $x^3y^2$ | 11.59 | 1.66 | $[14.11, \quad 9.07]$ |
| $x^2y^3$ | -4.8 | 1.66 | $[-2.28, \quad -7.32]$ |

*Table 2: : Shows the polynomial presentation of the design matrix, and the $\beta$-value for each predictor. The confidence interval, together with the variance is also shown in the table. N = 40, thus number of points are 1600. Also added noise is 0.1. This is for OLS, fitted to an order of 5.*

Finally for the OLS method we studied the training, and test error. We varied the test-size in the sklearn split function. We got the results shown in figure (5), (6), (7) and (8). The code we used for plotting the training and testerror is called *trainingtestOLS.py*

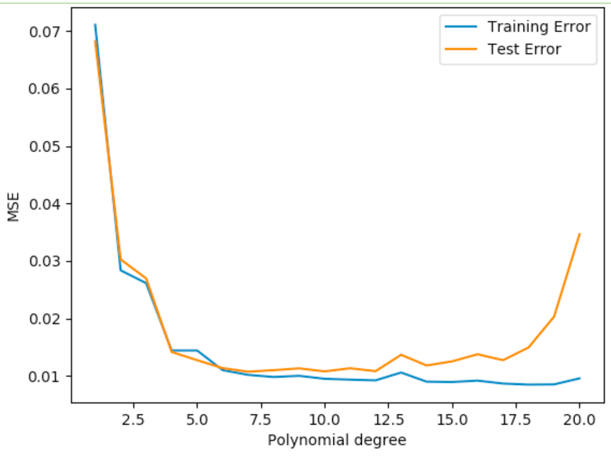
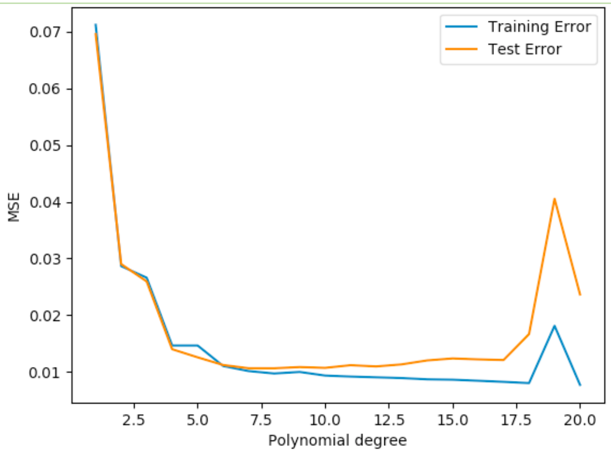
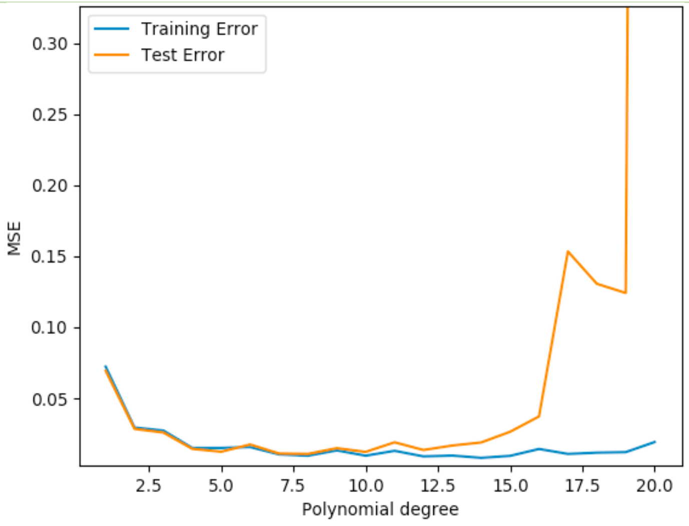


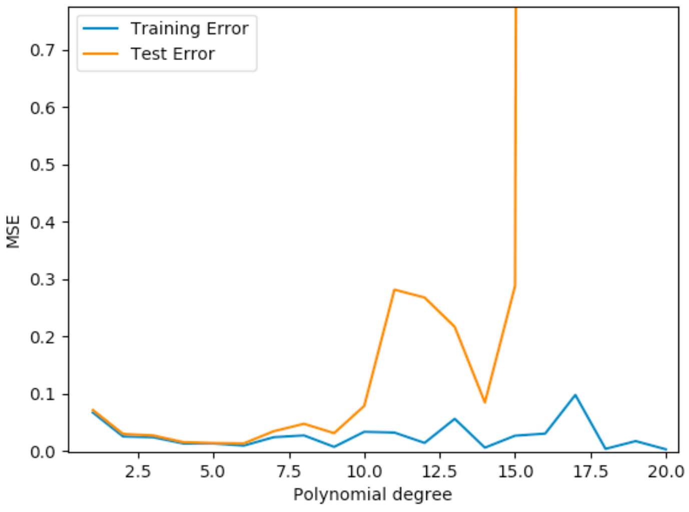
*Figure 5: Figure showing training, and testerror as function of polynomial degree. Test size is 0.2*



*Figur 6: Figure showing training, and testerror as function of polynomial degree. Test size is 0.4*



*Figur 7: Figure showing training, and testerror as function of polynomial degree. Test size is 0.6. The plot looks bigger since I have zoomed so that we can get a better view.*



*Figur 8: Figure showing training, and testerror as function of polynomial degree. Test size is 0.8.*

Then we looked at how the training and test error behaved when we varied the number of data points. This can be shown in figure (9), (10), (11) and (12)

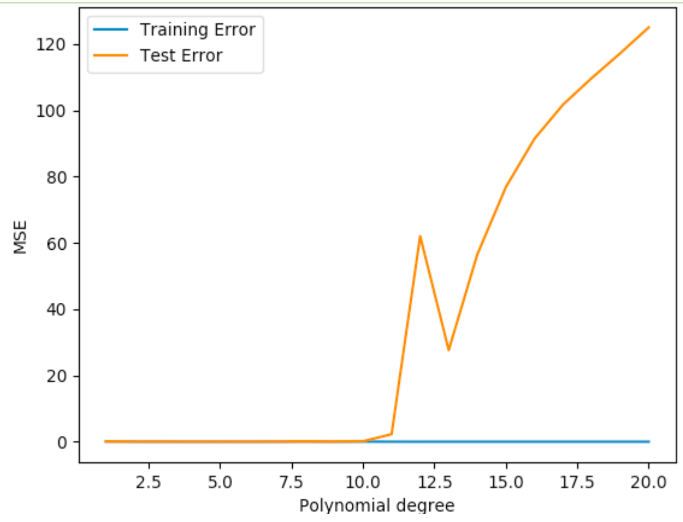
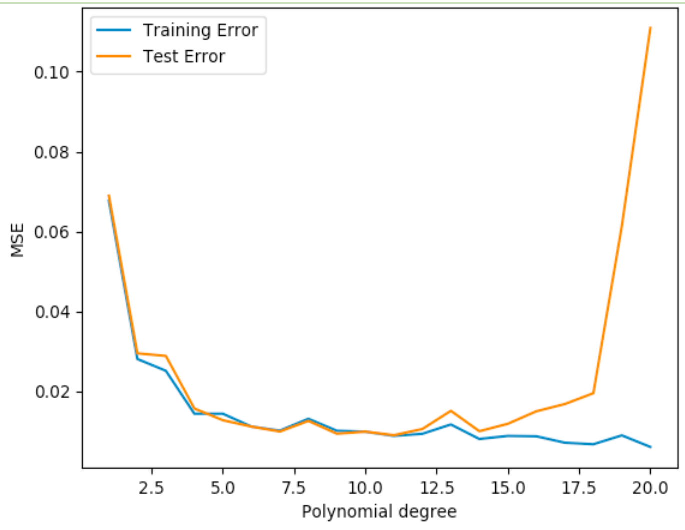
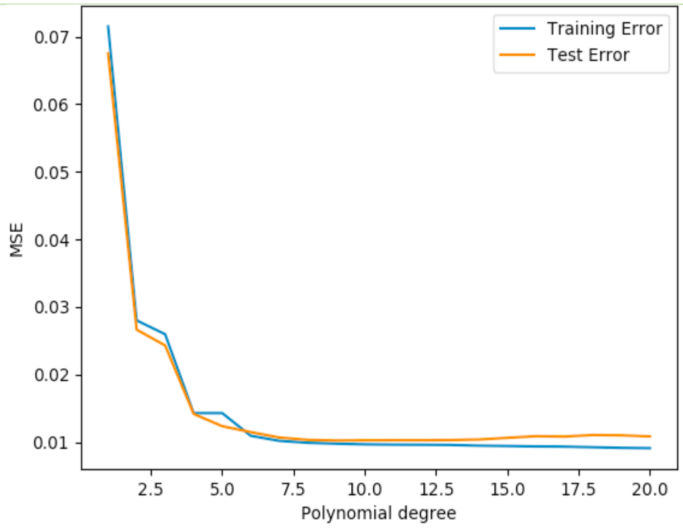


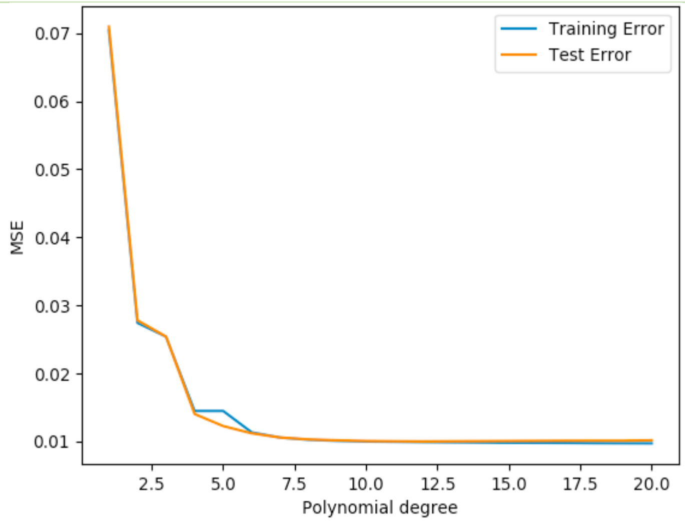
*Figure 9: Figure showing training, and testerror as function of polynomial degree. The number of data points is n = 10(x, and y), meaning we have 100 data points.*



*Figur 10: Figure showing training, and testerror as function of polynomial degree. The number of data points is n = 25(x, and y), meaning we have 625 data points.*



*Figur 11: Figure showing training, and testerror as function of polynomial degree. The number of data points is n = 60(x and y), meaning we have 3600 data points.*



*Figur 12: Figure showing training, and testerror as function of polynomial degree. The number of data points is n = 100(x and y), meaning we have 10000 data points*

## RIDGE

For Ridge regression on the Franke`s function we started of by finding optimal value of lambda. Table (4) below shows the result we got. We used a fifth order design matrix to get R2-scores, and MSE-values. Also we used k-cross validation, and the same code as we did for OLS.

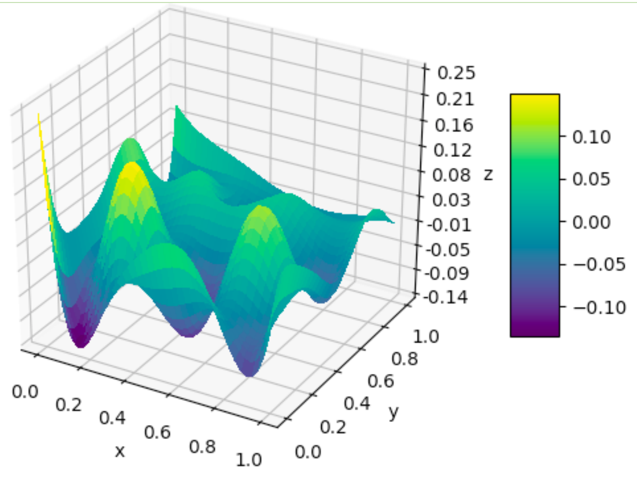
We opted to go for for $\boldsymbol{\lambda} = \mathbf{1E-6}$ since it gave relatively high R2-score, and low MSE-value. We used it to calculate what we got in table(5).
After analysing the table in (5) we made our plot to a polynomial degree 5. Partly because we wanted it to be comparable with the OLS, and Lasso method as we will see later. The plots for the Ridge prediction model and error is presented in figure (14) and (13) respectively.

| $\boldsymbol{\lambda - values}$ | R2 score | MSE |
|---|---|---|
| 1,00E-50 | 0.834 | 0.034 |
| 1,00E-20 | 0.834 | 0.013 |
| 1,00E-10 | 0.843 | 0.013 |
| 1,00E-06 | 0.834 | 0.013 |
| 1,00E-05 | 0.833 | 0.013 |
| 1,00E-04 | 0.827 | 0.013 |
| 1,00E-03 | 0.778 | 0.0165 |
| 1,00E-02 | 0.693 | 0.02 |
| 1,00E-01 | 0.392 | 0.04 |
| 1,00E+00 | -0.364 | 0.05 |

*Table 3: Shows the R2 score, and MSE for different values of $\boldsymbol{\lambda}$ . n = 40(x, and y), and thus number of data points is 1600. Added noise is 0.1.*

| Degree order | R2 score | MSE |
|---|---|---|
| 1 | -12.6 | 0.17 |
| 2 | -0.75 | 0.08 |
| 3 | 0.532 | 0.032 |
| 4 | 0.766 | 0.017 |
| 5 | 0.832 | 0.013 |
| 6 | 0.840 | 0.0126 |
| 7 | 0.843 | 0.0123 |
| 8 | 0.8433 | 0.0124 |
| 9 | 0.844 | 0.0123 |
| 10 | 0.845 | 0.0123 |
| 11 | 0.8461 | 0.0122 |
| 12 | 0.8461 | 0.0122 |
| 13 | 0.8460 | 0.0122 |
| 14 | 0.8462 | 0.0122 |

*Table 4: Shows the R2 score, and MSE for different polynomial degrees . $\boldsymbol{\lambda}$ used is $\boldsymbol{\lambda}$ = 1e-6. n = 4(x, and y)0, and thus number of data points is 1600. Added noise is 0.1.*



**Figur 13: Shows error between franky function, and the Ridge fit. n = 40, thus meaning we have 1600 points. Added noise is 0.1**

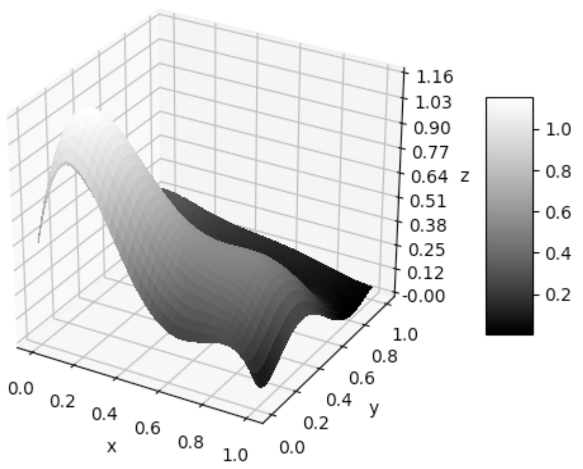


*Figure 14: Plot showing the Franky function fitted with RIDGE. The function is plotted up to order 5. The number of n points are 40(x, and y), thus*

*meaning we have 1600 points. Also added noise is 0.1.*

Then we calculated $\boldsymbol{\beta}$ using equation (9), variance (10) and confidence interval using (6). The result we got are presented in table (6). To run the code we used to calculate confidence interval, $\boldsymbol{\beta}$-values, variance and the plots can be gotten from code named *projridga.py*.

| $x^j y^j$ | $\beta$ | $VAR$ | Confidence interval |
|---|---|---|---|
| $x^0y^0$ | 0.42 | 0 | $[0.46, \quad 0.4\ ]$ |
| $x^1y^0$ | 7.56 | 0.03 | $[7.9, \quad 7.2]$ |
| $x^0y^1$ | 3.34 | 0.03 | $[3.7, \quad 2.99]$ |
| $x^0y^2$ | -6.68 | 0.75 | $[-4.99, \quad -8.38]$ |
| $x^2y^0$ | -32.9 | 0.75 | $[-31.20, \quad -34.60]$ |
| $x^1y^1$ | - 14.53 | 0.45 | $[-13.2, \quad -15.85]$ |
| $x^3y^0$ | 45.48 | 3.78 | $[49.3, \quad 41.67]$ |
| $x^3y^0$ | - 11.52 | 3.78 | $[-7.7, \quad -15.33]$ |
| $x^2y^1$ | 43.63 | 2.077 | $[46.46, \quad 40.81]$ |
| $x^1y^2$ | 18.95 | 2.077 | $[21.78, \quad 16.12]$ |
| $x^4y^0$ | -21.3 | 4.12 | $[-17.32, \quad -25.29]$ |
| $x^0y^4$ | 31.61 | 4.12 | $[35.59, \quad 27.63]$ |
| $x^2y^2$ | -8.02 | 2.08 | $[-5.19, \quad -10.84]$ |
| $x^1y^3$ | -26.9 | 2.4 | $[-23.86, \quad -29.95]$ |
| $x^3y^1$ | - 51.69 | 2.4 | $[-48.65, \quad -54.73]$ |
| $x^5y^0$ | 0.8 | 0.63 | $[2.35, \quad -0.76]$ |
| $x^0y^5$ | - 16.97 | 0.63 | $[-15.41, -18.53]$ |
| $x^4y^1$ | 18.08 | 4.79 | $[19.44, \quad 16.72]$ |
| $x^1y^4$ | 15.22 | 0.48 | $[16.58, \quad 13.87]$ |
| $x^3y^2$ | 10.61 | 0.46 | $[11.94, \quad 9.29]$ |
| $x^2y^3$ | -5.13 | 0.46 | $[-3.8, \quad -6.45]$ |

*Table 5: Shows the polynomial presentation of the design matrix, and the $\beta$-value for each predictor. The confidence interval, together with the variance is also shown in the table. n = 40, thus number of points are 1600. Also added noise is 0.1. This is for RIDGE, fitted to an order of 5.*

After that we tried plotting the test, and train error for the Ridge regression. The code we used is called *RidgeTrainTestPolyDegreeLambda.py.*
For figure (16), and (17) we tried to vary our test size, and see how the function behaved. As can be seen the test error for my lowest lambda value blows up after around degree 9 to 10 in figure (17). We can see it is different in figure(16). We used a number of data points equal 100 so that we could see a more clear effect at a lower degree

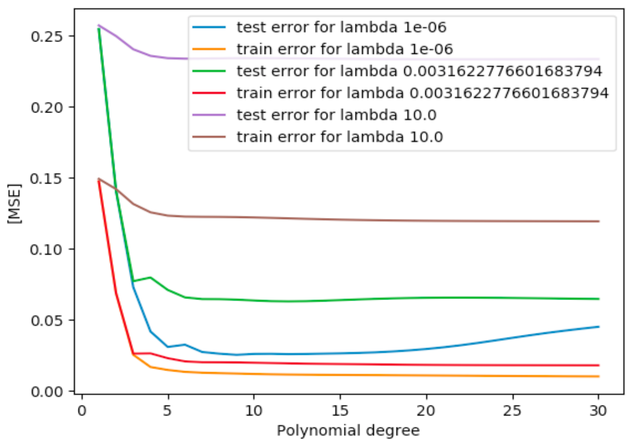
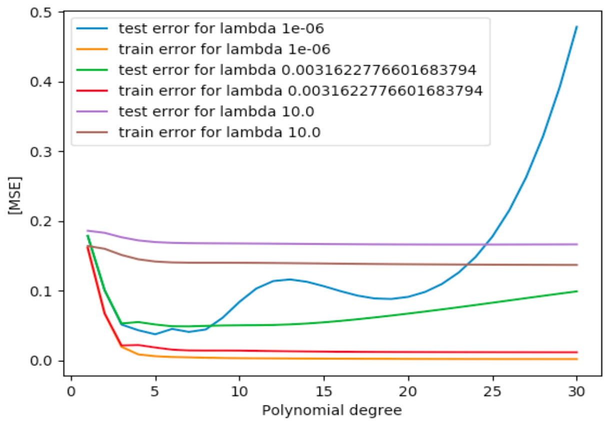


*Figure 15: Showing training-testerror for different values of lambda, and vary it with the polynomial degree. The number of data points is 100, with a test size = 0.2*



*Figur 16: Figure showing training, and testerror as function of polynomial degree. Test size is 0.5*

After that we varied our number of data points. Figure (18) shows a plot when n = 1600.

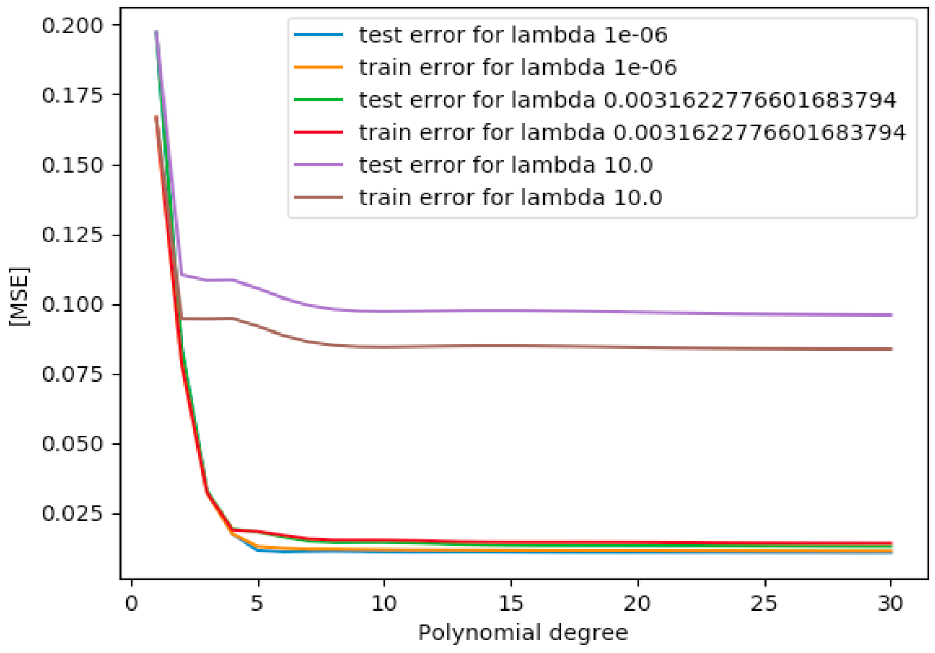


*Figure 17: Showing training-testerror for different values of lambda, and vary it with the polynomial degree. The number of data points is 1600, with a test size = 0.2.*

## Lasso

As we did for Ridge we found optimal value of lambda for Lasso regression. Since the analytical solution is not simply calculated compared to Ridge and OLS, we used Sklearn function in python to do the regression for us. R2-scores, and MSE-values with varying lambda values is shown in table (6). We chose optimal value of lambda = !E-6, and used it to obtain results in table (5).

The plot for the Lasso fit together with the error can be seen in figure (19), and (20).

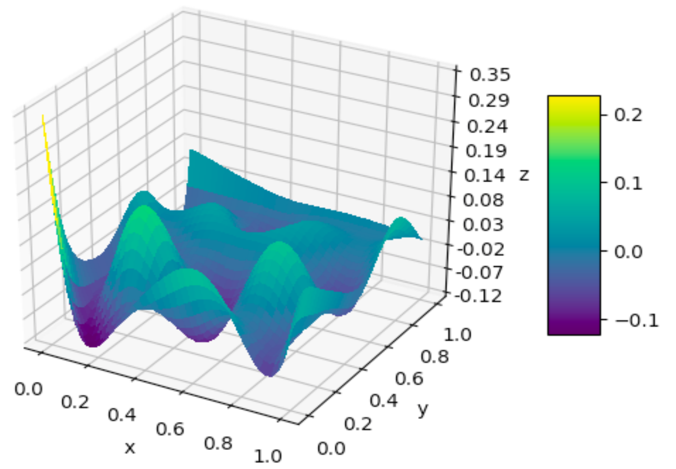


*Table 6: Shows the R2 score, and MSE for different values of λ. n = 40, and thus number of data points is 1600. Added noise is 0.1.*

*Figure 19: Shows error between franky function and the Lasso fit. n = 40, thus meaning we have 1600 points. Added noise is 0.1*

0.1.

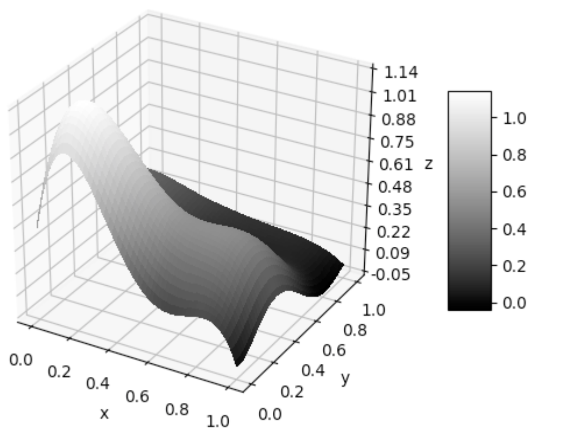


*Figure 20: Plot showing the Franky function fitted with RIDGE. The function is plotted up to order 5. The number of n points are 40, thus meaning we have 1600 points. Also added noise is 0.1.*

## Terrain Data Result & Method

### Linear Regression

We fit OLS, Ridge and Lasso methods to our terrain data with polynomial degree = 5 and 100 data points. Coefficients and confidence intervals for the coefficients were calculated for all three methods. To fit OLS and Ridge, matrix inversion was used while for lasso the scikit learn version was used. The source codes for these are found on our github repository in the source code folder with file names TerrainOLSBeta.py, 1fLasso.py and TerrainRidgeBeta.py

We noticed that for the linear regression plots of normalized data in illustration 3 the z axis indicated the value 0.02 for all points on the axis. From the illustrations, there is a bit of similarity for the ridge and OLS plots. The lasso plot has a unique shape of its own. Its shape demonstrates the shrinking component which is associated with lasso and ridge. For ridge and lasso, the value ƛ = 0.0001 was used to fit the model. This value was obtained from training and testing the models on different polynomial degrees to find the optimum ƛ value.

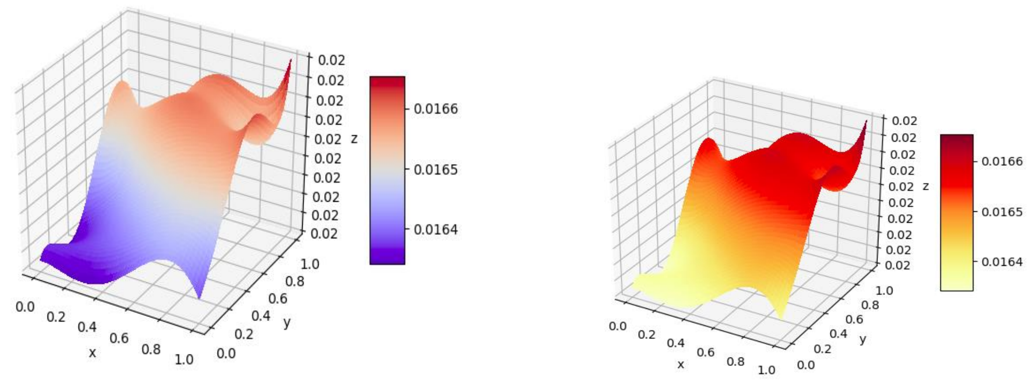


*Figure 1: OLS for the terrain (left) Ridge for the terrain data (right)*

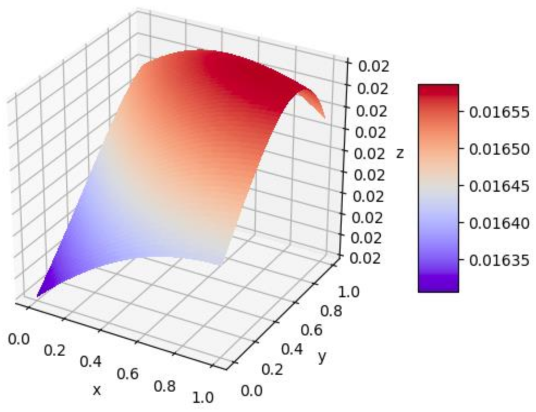


*Figure 2: Lasso Regression for the terrain*

For each model, $\boldsymbol{\beta}$ values were calculated with their corresponding confidence intervals. The formulae used for the confidence interval calculations for OLS ,Ridge and Lasso have been provided in the theory section. We had twenty one values for each model as a result of using 5 polynomial degrees. To compare the values that were obtained, table 1 presents eight of the Beta values calculated and their confidence intervals. Overall, the confidence intervals for Ridge are observed to be larger than those for OLS. Large confidence intervals have a higher likelihood of getting the correct $\boldsymbol{\beta}$ value. For this reason, Ridge had more accuracy for our $\boldsymbol{\beta}$ s.

Table 8: Beta values and their confidence intervals for OLS and Ridge

| **Beta** | **OLS** | | **Ridge** | |
|---|---|---|---|---|
| $x^j y^j$ | *β* | *Confidence Interval* | *β* | *Confidence Interval* |
| $x^0y^0$ | 2.73E-04 | $[-0.03112234, 0.03166777]$ | 1.64E-02 | $[-0.01503994, 0.04775017]$ |
| $x^1y^0$ | 4.41E-07 | $[-0.35139649, 0.35194192]$ | 2.63E-05 | $[-0.33531409, 0.36802432]$ |
| $x^0y^1$ | 9.04E-06 | $[-0.35139649, 0.35194192]$ | 5.30E-04 | $[-0.33531409, 0.36802432]$ |
| $x^1y^1$ | 9.49E-06 | $[-1.32594945, 1.32649489]$ | 5.70E-04 | $[-1.30986705, 1.34257729]$ |
| $x^1y^2$ | 5.35E-05 | $[-2.82679874, 2.82734417]$ | 3.20E-03 | $[-2.81071633, 2.84342657]$ |
| $x^5y^0$ | 1.21E-05 | $[-1.56188293, 1.56242836]$ | 7.25E-04 | $[-1.54580053, 1.57851076]$ |
| $x^0y^5$ | 8.33E-05 | $[-1.56188293, 1.56242836]$ | 4.99E-03 | $[-1.54580053, 1.57851076]$ |
| $x^4y^1$ | 8.96E-05 | $[-1.35777648, 1.35832192]$ | 5.37E-03 | $[-1.34169408, 1.37440432]$ |

## Resampling

We fit our three models to the terrain data with k-fold cross-validation. We used the standard test size, 0.2 of the data as the test. The matrix and terrain data were shuffled before splitting them. When shuffling, attention was given to ensuring that rows in the matrix were aligned with corresponding rows in the terrain data set which they corresponded to before shuffling. The significance of the shuffle task was observed through the lower MSE vales that were returned after shuffling data as compared to the high MSE values that were obtained for data that was not shuffled. Five folds were selected. For this analysis, we used *np.split* from scikit learn to split our data and terrain into five equal folds. The source code is available in files named terrainKfoldLasso.py and terrainKfoldOLS.py. The terrain and our matrix were shuffled before splitting, as a way to improve the performance of the models. The model was fit five times, leaving out one fold each time as the test set. The mean MSE and R2score values were calculated for each polynomial. We observed that the mean MSE for the normalized data that we used was very close to zero. For instance, when we fit the OLS model using polynomial degree 5 with normalized data, the MSE was *4.94974E-09*. This MSE value was thought to be too low considering the nature of the terrain used. Without normalization, with all the variables kept the same as the case with normalized terrain.It was found that the MSE was *2.3045672886231876*. In all three models, the MSE was observed to decrease with polynomial degree while R2 score increased. Lasso had its R2 score converge at degree 7. For higher polynomials the R2 score remained the same. MSE for lasso is the same for all the degrees that were tested below. This means error of the model would be the same with respect to all polynomial degrees included below. Lasso had negative R2 scores, which is very undesirable. For degree 3, R2 score was negative for all three models. OLS had the lowest MSE and the highest R2 score.

Table 9: MSE and RS score values for OLS, Ridge and Lasso for different polynomial degrees

| | **MSE** | | | **R2** | | |
|---|---|---|---|---|---|---|
| **Degree** | *OLS* | *Ridge* | *Lasso* | *OLS* | *Ridge* | *Lasso* |
| **3** | 5.32079E-09 | 5.32079E-09 | 1.47092E-05 | -0.024859818 | -0.024866721 | -0.114055841 |
| **4** | 5.14306E-09 | 5.14304E-09 | 1.47066E-05 | 0.044128198 | 0.043840443 | -0.113963559 |
| **5** | 4.94974E-09 | 4.95017E-09 | 1.47072E-05 | 0.113289074 | 0.110299484 | -0.113986797 |
| **6** | 4.83273E-09 | 4.84975E-09 | 1.47072E-05 | 0.152361463 | 0.14073088 | -0.113987366 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **7** | 4.58823E-09 | 4.7762E-09 | 1.47072E-05 | 0.228406445 | 0.159663302 | -0.113987377 |
| **8** | 4.49654E-09 | 4.70475E-09 | 1.47072E-05 | 0.255762271 | 0.179864951 | -0.113987377 |
| **9** | 4.35762E-09 | 4.65075E-09 | 1.47072E-05 | 0.295622031 | 0.196407581 | -0.113987377 |
| **10** | 4.31021E-09 | 4.61815E-09 | 1.47072E-05 | 0.310633925 | 0.207803715 | -0.113987377 |
| **15** | 3.7207E-09 | 4.49959E-09 | 1.47072E-05 | 0.462014959 | 0.241318824 | -0.113987377 |
| **20** | 3.44536E-09 | 4.44627E-09 | 1.47072E-05 | 0.526408467 | 0.260485309 | -0.113987377 |
| **25** | 3.45166E-09 | 4.42058E-09 | 1.47072E-05 | 0.554902789 | 0.267540382 | -0.113987377 |
| **35** | 3.42067E-09 | 4.39862E-09 | 1.47072E-05 | 0.592427597 | 0.274729912 | -0.113987377 |

## Model Complexity

We fit the lasso and ridge models for different polynomial degrees for different values of ƛ with resampling of 25 bootstraps for OLS and Ridge. Since OLS does not need a shrinkage parameter, it was fit to the models with different degrees without any ƛ values. For all plots for OLS, the scikit learn function *LinearRegreassion()* was used.The model was fit from degree 1 to degree 50. With 100 data points, the train and test errors start off high and both decrease for the first polynomial degrees. When choosing a polynomial degree, it would be wiser to choose the sections with the lowest test and training errors.

We note that as degree approaches 5 the train and test errors begin to decrease considerably. The lowest values for the errors are found for degrees 18 to 30. Based on this plot, the optimum polynomial degrees that would be used to fit OLS lie in this range. Among all degrees in this region, the best option for polynomial degree would be the value at the lower end of the range since it is the lowest in the range and would therefore perform faster compared to the higher models. From degree 35, the test error starts to increase and blows up. The file *terrainOLSTrainTestPolydegree.py.*

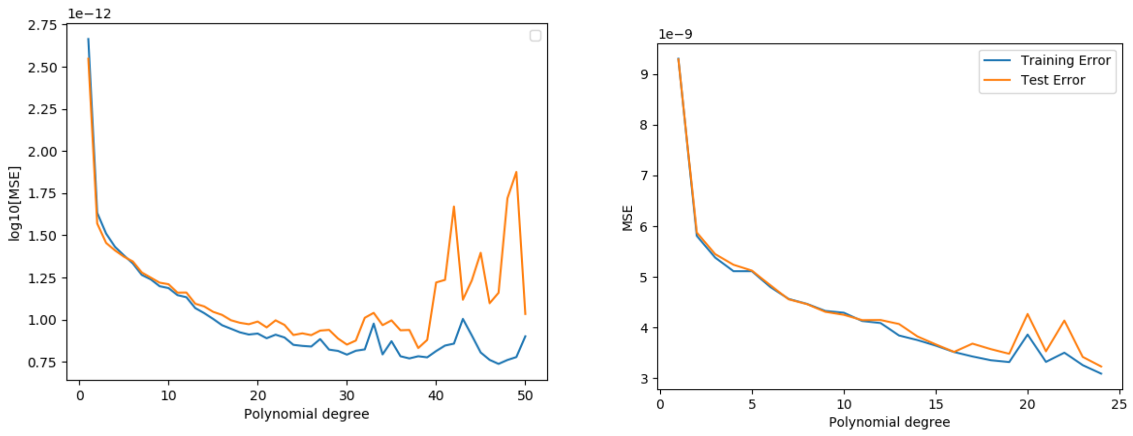
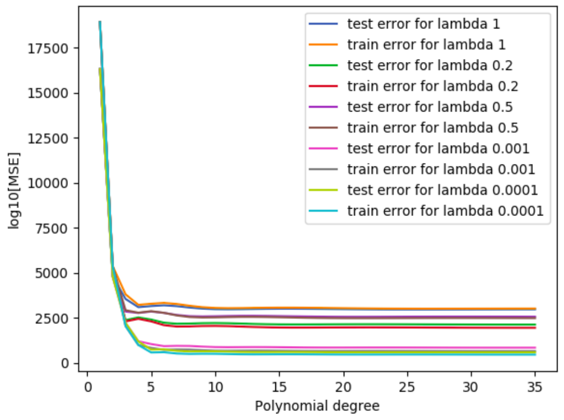


*Figure 3: Train test plots for OLS. (left) Plot used 100 data points. (right) plot used 20 data points*

Figure 4b is the plot obtained by fitting OLS to only 30 data points and a maximum degree of 24. For these data points, the optimal polynomial degrees remain the same, ranging from 12 to 16. With the decrease in data points, we now the notice the test error blows up at degree 22. Our MSE blows up earlier for fewer data points.



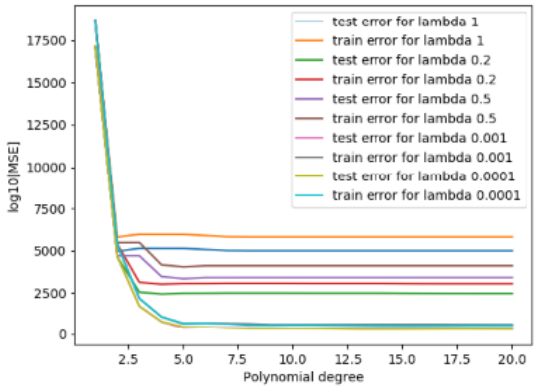


*Figure 4: Train test complexity plots for ridge (left) and lasso (right)*

Figure 4 is a plot for train and test errors for various ƛ values for different polynomial degrees for ridge. We notice that the train and test errors for all ƛ values start off as very high for degree 1 and dramatically decrease between degree 2 to just before degree 5 . From degree 5 to degree 20 the train and test errors are seen as almost being constant since the change in them is very minimal. If the plots were to go on for very higher polynomial degrees, we would have been able to notice the conventional trend of the errors, where with very high degrees the test error drastically increases while the train error gradually continues to decrease. The source code for the plot is *terrainRidgeTrainTestPolyDegree.py.* ƛ values were selected at random from the interval (0,1).

A similar trend is noticed for lasso regression in file *terrainLassoTrainTestPolyDegreeLambda.py*. The train and test errors start off high from 1 but decrease considerably until the polynomial gets to degree 5. From degree 5 to 20, we notice very minor change in the train and test errors. Polynomial degree 5 seems to be the most suitable for lasso regression. Taking a higher polynomial degree than 5 will not lead to any significant changes in the training and test error due to the small variation that is displayed from the plot. However, for very high polynomial degrees, the test error will increase and diverge from the train error as evident in the illustration for the train test error for OLS. Both in Lasso and Ridge, the lowest values of ƛ produce the lowest train and test errors for all polynomial degrees that were included.

## Discussion

Since each both the Franke`s function, and OLS proved to yield different results, we also discuss them seperately.

### Franke Function

#### MSE, and R2-score

If we are to decide which model gave the best prediction model we will say it is the OLS regression method. It has proved to have the lowest MSE, and R2-scores values when we assessed using k-fold crossvalidation. This can be seen in table (1), (4) and (7). But, the other methods were not very bad either. At polynomial degree of 5, OLS had an R2-score = 0.85 and MSE = 0.012. Ridge had 0.83, and 0.013, and Lasso had 0.83 and 0.013.

### Bias-variance tradeoff

One advantage that both Ridge, and Lasso have is that they give lower variance in their models by trading it off with bias. In this project, we have used very low values of lambda for Lasso and Ridge to obtain optimal values for the R2-scores, and MSE. We could then not see very well the variance effect we got in the result.

But, we saw what happened when complexity of the model was increased. Table (2) shows confidence interval for beta calculated using OLS, whereas (6) shows for ridge. Since we calculated results in (6) using a polynomial order of 11, we can see clear results of how the confidence interval decreases in (6) compared to (2). Increase in polynomial degree means more training in the trainingset, and thus leads to an increase in bias. This what we call an overfit.

We also looked at how bias variance in equation (14) varied with number of data points. For OLS, this can be seen in figure (9), (10), (11) and (12), and ridge in (17) and (18). We see by increasing the order of the polynomial the training, and testerrora are more close, thus meaning we get lower bias. Also, we see that our polynomial blows up at a higher degree. This can be understood by thinking that less data points means that we dont get to train our model well. With more data points we then become able to train our dataset better, and give better prediction models.

To choose optimal value of test size is important. In figure (5), (6), (7) and (8) test size is varied from 0.2 to 0.8. 0.2 and 0.4 seems to be good test sizes, and one can argue that 0.4 seems better. But, test sizes equal 0.6 and 0.8 leads to having a test error that blows up at a lower degree. Again, this can be understood that having little training data leads to having a model that is not trained well. Similar results can be seen for figure (16) and (17) for ridge regression. We did not do the same calculation for Lasso since it is taking very long time to do calculation using optimal value of Lambda. But we expect that the method shows the same behaviour as OLS, and Ridge do.

## *Terrain Data set*

Overall, OLS has proved to be the best model to fit our terrain data according to the results we obtained given the condition that sufficient data points are used. For OLS, the best polynomial degree is in the range (18,30). However, the optimal polynomial degree for all three models is 5. If we are to use degree 5 in fitting our model, we are guaranteed of getting the good fit. OLS and ridge both had good confidence intervals for betas. These two models provide the accuracy needed when fitting our terrain data set. Ridge and Lasso models require small values of lambda for better prediction and more accuracy. The optimal $\lambda$ for both was the smallest $\lambda$ value that was available. This confirmed the significance of using small $\lambda$ values. Lasso showed the worst performance of all: it had negative R2 scores, It needed longer running time for lower values of lambda even though the lower values produced lower train and test errors.

If there would be a need of fitting a model with high polynomial degrees for terrain data, OLS would be our go to model. This is the case since with increased polynomial degree comes a decrease in MSE and an increase in R2. OLS also has the highest R2 score and lowest MSE as polynomial degrees increase. If a high polynomial degree is desired but there are few data points Ridge would give the best performance in this situation. Lasso presents constraints on the R2 score and MSE as the polynomial degree increased. These two are suppressed and stay at the same value therefore fitting to higher polynomial degrees would not give the best of results.

The re-sampling techniques that were used: k-fold cross-validation and bootstrap proved to be useful in increasing model accuracy for all three models. K-fold reduced the bias while bootstrap drastically reduced our variance. Shuffling data is one component that should not be overlooked when preparing data for cross-validation. For higher model accuracy, the use of one of these re-sampling techniques is highly recommended.

## Conclusion

OLS is the best model to fit our terrain data set and franke function. However, factors like number of data points used to fit the model and ratio of test size play major roles in the performance of the model. Some characteristics that make OLS suitable are: low MSE, high R2 score and bigger confidence intervals for Beta values.

For better accuracy when using any of the models, resampling techniques need to be used. Cross validation may be used to assess the model accuracy and as a resampling method. Bootstrap is used to assess accuracy and is useful when few data points are present and few data points are available.

Ridge is more suitable in situations where our data is lacking and only few data points are available. Ridge and lasso performance are dependent on the size of lambda chosen. There is great fit as lambda values approach 0. In this case, Ridge is noticed to behave in the same manner as OLS. Lasso had slow performance with optimal lambda values.

Ridge and OLS had MSE decrease and R2 score increase with an increase in polynomial degree. Polynomial degree use to fit the model seemed not to have a significant effect on lasso. Lasso had the same MSEs R2 scores for different degrees.

## Reference

## Bibliografi

Trevor, H., Robert Tibshirani, & Jerome, F. (2008). *The Elements of Statistical Learning* (Second Edition. utg.). Springer.

Casella, G., Fienberg, S., & Olkin, I. (2013). An Introduction to Statistical Learning. In Springer Texts in Statistics. https://doi.org/10.1016/j.peva.2007.06.006

Hastie, T. et. all. (2009). Springer Series in Statistics The Elements of Statistical Learning. The Mathematical Intelligencer, 27(2), 83–85. https://doi.org/10.1007/b94608