Project 1 on Machine Learning

Thomas Haaland
Jie Hou
Martin Funderud Gimse
https://github.uio.no/thomhaa/Fys-Stk4155.git

October 2019

Contents

Chapter 1

Abstract

This project investigates the performance of three regression methods: Ordinary Least Squares, Ridge regression and Lasso regression on the Franke function and real terrain data. We studied the trends and features of several polynomials and investigated which of the methods and under which conditions they would fit the Franke function best. Based on what we observed, we concluded with Ridge and Lasso performing better than Ordinary Least Squares for higher polynomial degrees, for smaller number of training points and for datasets with more noise. Especially Lasso came with a penalty in performance which might be large in relation to little gain in accuracy of fit. Additionally Ridge and Lasso can make a worse fit if the parameter λ is selected to be too large.

Chapter 2

Introduction

In the supervised machine learning field, there are many methods that can be useful when it comes to regression. In this project we will look at three different variants of Linear Regression, namely Ordinary Least Squares, Ridge regression and Lasso regression. The performance of each method may differ from each other due to different size, numbers of features and characteristics in the data set. In the first part of this project We will look at the theory behind Linear Regression and look at some of the theoretical weaknesses in the methods. After we've learnt the theoretical background of these regression methods, We will then implement these methods and see if the theory is sufficient in explaining where and how these methods go wrong. Finally we will dig into a real-life problem, use what we have developed on terrain data and see how these methods manage to predict a height map.

Chapter 3

Theoretical Background

3.1 Statistical Functions

3.1.1 Statistics overview

During this project we will use several statistical methods to measure how well the functions we train fit with our data. We start with looking at the expected value. The expected value is the value we get on average given some probability distribution $p(\mathbf{y})$, where $p(y_i)$ is the probability of getting the value y_i.

$$\mathbb{E}(\mathbf{y}) = \sum_{i=0}^{n-1} y_i p(y_i). \tag{3.1}$$

However, if we don't know the underlying probability distribution we can approximate the expected value with the mean if we assume that the probability for each y_i is identical.

$$\mathbb{E}(\mathbf{y}) = \frac{1}{n}\sum_{i=0}^{n-1} y_i = \mu \tag{3.2}$$

Equation (3.2) is used to determine the average value of some set of $\mathbf{y}$, usually denoted as μ.
Mean Squared Error, or MSE, is a measure for how elements y_i in a data set $\mathbf{y}$ differs from a different data set $\hat{\mathbf{y}}$. We will use it to see how well our predictions $\hat{\mathbf{y}}$ match with the real data set $\mathbf{y}$.

$$\mathrm{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1} \left(y_i - \hat{y}_i\right)^2. \tag{3.3}$$

The MSE is squared before the elements are summed so that no contributions cancels each other. The variance (σ) is a measure of how far each value in the data set is different from the mean.

$$\mathrm{Var}(\mathbf{x}) = \mathbb{E}[(\mathbf{x} - \mu)^2] = \mathbb{E}(\mathbf{x}^2) - [\mathbb{E}(\mathbf{x})]^2 = \sigma^2. \tag{3.4}$$

R2 score is a statistical measure of how close the data are to the fitted regression line. It is also called the coefficient of determination.

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(\hat{y}_i - y_i)}{\sum_{i=0}^{n-1}(y_i - \mathbb{E}(\mathbf{y}))} \tag{3.5}$$

3.1.2 Least Squares Methods

In this section we will discuss a group of methods called least squares methods. We start with an underlying assumption that the data points y_i in the data set $\mathbf{y}$ follows some function $f(x_i)$ such that $y_i = f(x_i) + \epsilon_i$ where ϵ_i is noise which is randomly distributed. According to the Central Limit Theorem, this noise will tend to go towards the normal distribution $\mathcal{N}(0, \epsilon)$ regardless of its actual underlying distribution when the number of samples grow large enough. This will give it the attribute that the expected value will be $\mathbb{E}(\mathcal{N}(0, \epsilon)) = 0$, and $\mathbb{E}(\epsilon) = 0$. Since $f(\mathbf{x})$ is an unknown function, we need to make a guess when we attempt to fit this function. In Least Squares we guess that a linear function would fit well to the data. With this assumption given, we can then approach the underlying real function since:

$$\begin{aligned}\mathbb{E}(\mathbf{y}) &= \mathbb{E}\left[f(\mathbf{x}) + \epsilon\right] \\ &= \mathbb{E}\left[f(\mathbf{x})\right] + \mathbb{E}\left[\epsilon\right] \\ &= f(\mathbf{x})\end{aligned}$$

with a large enough sample size. This allows us to perform a best fit with a polynomial of the form:

$$y_i = x_{0,i} + x_{1,i}\beta_1 + x_{2,i}\beta_2 + \cdots + x_{n-1,i}\beta_{n-1} \tag{3.6}$$

which can be rewritten more succinctly as:

$$\mathbf{y} = \mathbf{X}\beta. \tag{3.7}$$

$\mathbf{X}$ is called the design matrix. When we attempt to perform this best fit we need some metrices that we can minimize to tell us how good the fit is. From equation (3.3) we have a function which compares point for point how close these two data sets are. We will call this function the cost function.

Ordinary Least Squares method

In Ordinary Least Squares method we take the cost function to be the MSE as it is in equation (3.3), we have a cost function which is well behaved and differentiable, giving the cost function as:

$$\mathcal{C}(\mathbf{y}, \hat{\beta}) = \left(\mathbf{y} - \mathbf{X}\beta\right)^2. \tag{3.8}$$

Solving the differentiated cost function for $\hat{\beta}$ we get the equation [3]

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{3.9}$$

$\hat{\beta}$ is called a predictor and it is not the 'real' β but rather an approximation, since we both generally don't have enough data to completely eradicate the contribution from ϵ and we have no way to be sure that there does exists a function $f(\mathbf{x})$. We get our prediction $\hat{\mathbf{y}}$ by:

$$\tilde{\mathbf{y}} = \mathbf{X}\hat{\beta}. \tag{3.10}$$

The column vectors of matrix $\mathbf{X}$ is denoted by $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_p$ where $\mathbf{x}_0 = 1$.The different columns thus are different input data. We can study how the predictor varies using the equation [3]:

$$\mathrm{Var}(\hat{\beta}) = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\sigma^2 \tag{3.11}$$

However, since σ is unknown we can approximate it with:

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=0}^{N}(y_i - \hat{y_i})^2 \tag{3.12}$$

where $N - p - 1$ is used to give an unbiased approximation. The diagonal elements yields the variance of each component in β

$$\sigma(\beta_i)^2 = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}_{ii}\hat{\sigma}^2. \tag{3.13}$$

The variance of $\hat{\beta}$ tells us how we can expect the models to vary due to uncertainties in the original data, such as when we have few data points or if there is a lot of noise in the data set. this problem can be mitigated by using Ridge regression which introduces the shrinkage parameter λ.

Ridge Regression

In Ridge regression the starting point is the OLS method described above. Ridge regression were originally introduced to avoid singularities when inverting the design matrix square $\mathbf{X}^T\mathbf{X}$ by introducing a small factor λ as described in [3].

$$\mathbf{y} = (\mathbf{X} + \lambda\mathbb{1})\beta \tag{3.14}$$

When solving for β like we did for OLS, we get the cost function

$$\mathcal{C}(\beta;\lambda)^{Ridge} = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta. \tag{3.15}$$

Solving for β, which we will name β^{Ridge} yields

$$\beta^{Ridge} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbb{1}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{3.16}$$

It's variance is given as [5]:

$$\sigma(\beta_i^{Ridge})^2 = \left[\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbb{1}\right)^{-1}\mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbb{1}\right)^{-1}\right]_{ii}\hat{\sigma}^2. \tag{3.17}$$

As we can see the ridge regression solution is a linear function of y, with an added positive constant to the diagonal of $\mathbf{X}^T\mathbf{X}$ before inversion. In addition to make the inverse non-singular we can see that the predictor is shrunk for any $\lambda > 0$.
To better understand what this shrinkage might mean to our model, we will take a look at the Singular Value Decomposition (SVD) of $\mathbf{X}$, which has the form [3]:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T. \tag{3.18}$$

$\mathbf{U}$ and $\mathbf{V}$ are the orthonormal matrices, such that $\mathbf{U}\mathbf{U}^T = \mathbf{V}\mathbf{V}^T = \mathbb{1}$, with $\mathbf{U}$ spanning the the column space of $\mathbf{X}$ and $\mathbf{V}$ spanning the row space of $\mathbf{X}$. $\mathbf{D}$ is a diagonal matrix with the eigenvalues of $\mathbf{X}$ along the diagonal in an ascending fashion. [3]
The prediction $\hat{\mathbf{y}}$ is given by:

$$\hat{\mathbf{y}} = \mathbf{X}\beta^{Ridge}. \tag{3.19}$$

Using SVD we get:

$$\begin{aligned}\mathbf{X}\beta^{Ridge} &= \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\\ &= \mathbf{U}\mathbf{D}\left(\mathbf{U}^2 + \lambda\mathbb{1}\right)^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y}\\ &= \sum_{j=1}^{p}\mathbf{u}_j\frac{d_j^2}{d_j^2+\lambda}\mathbf{u}_j^T\mathbf{y}\end{aligned}$$

where $\mathbf{u}_j$ are the columns of $\mathbf{U}$. We can see that small eigenvalues in $\mathbf{X}$ gives greater shrinkage while larger eigenvalues is kept for larger λ. It is, however, still not clear what shrinkage in the d_i components mean for our model. Using SVD we get that:

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}^2\mathbf{V}^T \tag{3.20}$$

which is the eigen decomposition of $\mathbf{X}^T\mathbf{X}$. Each eigenvector $\mathbf{v}_i$ is also called the principal components directions of $\mathbf{X}$. The full principal decomposition of $\mathbf{X}$ is given by $\mathbf{T} = \mathbf{X}\mathbf{V}$, which can be rewritten as $\mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{D}$, and the principal directions are spanned by $\mathbf{v}_i$. These principal directional vectors have the property that the first principal vector $\mathbf{X}\mathbf{v}_0$ gives the largest sample variance since the diagonal matrix $\mathbf{D}$ is ordered. investigating the sample variance gives us:

$$\begin{aligned}\mathrm{Var}(\mathbf{X}\mathbf{v}_j) &= \mathbb{E}\left[(\mathbf{X}\mathbf{v}_j - \mathbb{E}(\mathbf{X}\mathbf{v}_j))(\mathbf{X}\mathbf{v}_j - \mathbb{E}(\mathbf{X}\mathbf{v}_j))^T\right]\\ &= \mathbb{E}\left[\mathbf{X}\mathbf{v}_j\mathbf{v}_j^T\mathbf{X}^T\right] - \mathbb{E}[\mathbf{X}\mathbf{v}_j](\mathbb{E}[\mathbf{X}\mathbf{v}_j])^T\\ &= \mathbb{E}\left[\mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{v}_j\mathbf{v}_j^T\mathbf{V}\mathbf{D}^T\mathbf{U}^T\right]\end{aligned}$$

Looking at $\mathbf{UDV}^T\mathbf{v}_j$ we see that $\mathbf{V}^T\mathbf{v}_j = \mathbf{e}_j$, where $\mathbf{e}_j$ is the j'th unit vector. Thus $\mathbf{UDV}^T\mathbf{v}_j = \mathbf{u}_j d_j$, and $\mathbb{E}(\mathbf{X}\mathbf{v}_j) = \mathbb{E}(\mathbf{u}_j d_j) = d_j\mathbb{E}(\mathbf{u}_j) = 0$ since d_j is a constant and $\mathbf{X}$ is centered. So we get:

$$\begin{aligned}\mathrm{Var}(\mathbf{X}\mathbf{v}_j) &= \mathbb{E}\left[\mathbf{UDV}^T\mathbf{v}_j\mathbf{v}_j^T\mathbf{VD}^T\mathbf{U}^T\right] \\ &= \frac{1}{N}\sum_{i=0}^{N-1}\mathbf{u}_j(i)\mathbf{u}_j(i)^T d_j^2.\end{aligned}$$

Only when $i = j$ is $\mathbf{u}_j(i)\mathbf{u}_j(i)^T = 1$ and for all $i \neq j$ it is 0.

$$\mathrm{Var}(\mathbf{X}\mathbf{v}_j) = \frac{d_j^2}{N}. \tag{3.21}$$

Since, as noted previously $\mathbf{X}\mathbf{v}_j = \mathbf{u}_j d_j$, it is evident that the principal components with greatest variance is reduced last, whereas the last principle component with smallest variance are reduced first. So Ridge regression shrinks the singular values in $\mathbf{X}$ with the least variance and reduces the effective degrees of freedom, targeting the components which contributes the least to the prediction. From equation (3.17) we also see that the variance of β^{Ridge} shrinks when λ grows.

Lasso Regression

Lasso regression is another regression method that uses the shrinkage parameter λ. LASSO stands for "Least Absolute Shrinkage and Selection". In Lasso regression, the data are shrunk towards a central point and sets others to 0.
The goal in Lasso regression is to minimize the cost function:

$$\mathcal{C}(\beta;\lambda)^{LASSO} = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\left\|\beta\right\|_1. \tag{3.22}$$

Here $|\beta|$ is the norm of β. While the cost function for Lasso appears similar to the Ridge cost function (equation 3.15) using the norm as opposed to the square of β leads to some different behaviour when minimizing the cost function. By standardising the predictors we can re-parametrize the constant β_0. For Ridge regression we add squared magnitude of coefficient as penalty term to the loss function, whereas Lasso adds absolute value of magnitude of the coefficient as the penalty term. This means the derivative is not continuous for some β values, which in turn means that it's not trivial to find an analytic solution. When the shrinkage parameter λ gets equal to zero, in both Ridge and Lasso regression we will get back to OLS. The difference is, when λ gets large, it will add too much weight to the loss function which will lead to under-fitting in Ridge regression. However Lasso actually shrinks less important coefficient to zero when λ gets large or the threshold t gets small enough. If we have huge number of features, Lasso regression is a better choice for feature selection process, since it actually

get rid of the features that are not relevant.
We can write the constrained solution $\tilde{\beta}$ as the ridge regression estimator [4]:

$$\tilde{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{W}^-)^{-1}\mathbf{X}^T y \tag{3.23}$$

Where the W is a diagonal matrix with diagonal elements $|\tilde{\beta_j}|$ and $\mathbf{W}^-$ is the generalized inverse. In order to compute the number of effective parameters in the constrained fit $\tilde{\beta}$, we can calculate: [4]

$$p(t) = tr\{\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{W}^-)^{-1}\mathbf{X}^T\} \tag{3.24}$$

which is the effective degree of freedom. It is now clear that as λ grow, $p(t)$ becomes smaller.

Both Lasso and Ridge acts to reduce the effective degrees of freedom, they do so differently. While Ridge tends to reduce the principal components gradually, Lasso can set some variables to zero while retaining others nearly unchanged [3].

Chapter 4

Analytical methods, algorithms and implementation

In an effort to analyse our results we need a toolbox to decide where and how our model makes mistakes. It is not enough to simply train a model on the given data. We don't know how the model will behave on unseen data, for instance. The model is, naturally, limited to the data we have, so we need a method to simulate additional data and we need to be able to test our model on unseen data. A method for testing our model on unseen data is to simply withhold a portion of the data (typically around 20% of the whole data set) and test the model on this part of the data. Cross Validation takes this method to heart and tests on test sets sequentially utilising in the end the entire set as a test set. Resampling is a method to simulate additional data points. With the Bias Variance trade-off we can study more closely how our model fails.

4.1 Implementing Linear Regression

The methods for linear regression follow the theory closely. For all the methods it is a matter of finding β, then applying β to the design matrix.

```
    zPredict = X @ beta
```

Listing 4.1: Python definition zPredict

The implementation of OLS is taken directly from equation (3.9). Finding beta is straightforward. However, since we are not guaranteed that the matrix $\mathbf{XX}^T$ is not singular we had to compensate with the SVD inversion.

```
    betaOLS = inv(X.T @ X) @ X.T @ z
```

Listing 4.2: Python definition betaOLS

The SVD inversion makes the inversion robust versus singular matrices. While numpy has a good inversion function in np.pinv, we opted to use the inversion function SVDinv from the lecture notes [1].

```python
    def SVDinv(A):
        U, s, VT = np.linalg.svd(A)
        D = np.zeros((len(U),len(VT)))
        for i in range(0,len(VT)):
            D[i,i]=s[i]
        UT = np.transpose(U)
        V = np.transpose(VT)
        invD = np.linalg.inv(D)
        return np.matmul(V,np.matmul(invD,UT))
```

Listing 4.3: Python function SVDinv

When inverting Ridge we designed the code based on the equation (3.16), together with using the SVD inversion code above we got:

```python
        beta = SVDinv(X.T @ X +  _lambda * np.eye(dim))
            @ X.T @ z
```

Listing 4.4: Python definition beta

Finally, for Lasso, we implemented SciKit-Learns functionality. Since Lasso uses the gradient descent we had convergence issues for small λ. To compensate we had to increase iterations substantially, however this slowed down our code and became a significant bottleneck.

```python
        lasso = skl.Lasso(alpha=lambda)
        lasso.fit(X, z)
        beta = lasso.coef_
```

Listing 4.5: Python code LASSO

We needed λ in the range of 10^{-7} for some minimal λs, which slowed down our code. Reaching values this low forced us to consider other options, and we found that increasing the resolution in the data set provided a means to increase the necessary value for λ.

4.2 Resampling and cross validation

Since we generally neither have continuous nor infinite data, there are especially two situations we need to be aware of when performing a best fit. When fitting to a data set, it is easy to see that we need a complex enough function to capture the variances in the data set. Not having a complex enough model leads to under-fitting, where the model doesn't have enough degrees of freedom to properly fit to the data set. Also there is the problem of over-fitting. Given a particular data set, increasing the complexity of the model leads the model to better capture the variances of that data set. However, this improved fit may just capture unimportant features such as noise. This means that the model gets specialised on the training data set, but looses its ability to generalise on the unseen data. Cross-validation can help us to select the model which will perform best on an unseen data set and help us to avoid problems like over-fitting and under-fitting.

Resampling

Resampling is a method where if we have a data set $\mathbf{y}$ of size n, we can select elements n times randomly from $\mathbf{y}$, building a new data set $\mathbf{y}_i$ from our original data set. We can do this as many times as we like in order to build a much larger total data set than what we had initially. Due to the Central Limit Theorem we should have a distribution which approaches the real mean. This method is used when we have too few data points.

Cross Validation

Cross validation is a technique where the data set is split into a training set, a test set and, often, a validation set. The learning algorithm is trained on the training data and tested against the test data to see how well the trained model works on data not yet seen. This makes it easier to detect if the model is over-fitted and thus loose prediction power.

We will take a closer look at a standard train test split and K-Fold techniques. In a standard train test split, the model is tested only once at the end. While this gives us the opportunity to detect over-fitting the split is not used to improve the model.

In the K-fold algorithm we shuffle the data points and split the batch into K-folds, where K is some number chosen by us (normally 5 or 10, but any number will do. It appears K is chosen by convention and not because some numbers are better than others). We then choose one fold, withholding it from training and use it as a test batch. We then repeat for each batch. This allows us to choose one predictor with the best MSE and R2 score.

With K-fold, we average scores after K different holdouts, every data point gets to be in a validation set exactly once and gets to be in a training set k-1 times. This method is very useful when we have limited data.

The code we made for K-fold allows us to use it in the same manner as we would with SciKit-Learn, and allows an implementation such as:

```python
    kf = oh.k_fold(n_splits=k, shuffle=True)
    kf.get_n_splits(X)
    for train_index, test_index in kf.split():
        <body>
```

Listing 4.6: Python code for K-fold

This process loops over a shuffled data set with k number of batches acting as a validation set. This leaves us with three layers of datasets used for different purposes: train set which is used to train the model, validation set which is used to see how the model is progressing while training, and test set which is used to see how our model is doing after the training has finished.

4.3 Bias Variance trade-off

Bias is the difference between the average predicted value from our model and the correct value that we are trying to predict. Whereas variance is how much the model prediction vary for a given data point. So, variance allows us to measure the spread of our data. If the model is too simple and don't have enough parameters, then it may have high bias and low variance. If the model is very complex with many parameters, then it may have low bias and high variance. So we need to find a balance between bias and variance.

In order to find this balance we can study the total error as expressed by the cost function MSE. The cost function for OLS is the MSE as described in equation (3.3). When performing a linear fit we assume that the real function $\vec{y}$ can be expressed as $\vec{y} = f(\vec{x}) + \vec{\epsilon}$. Where we assume $\vec{\epsilon}$ is an independent stochastic variable normally distributed according to $\mathcal{N}\left(0, \sigma^2\right)$, while $f(\vec{x})$ not being a stochastic variable. The expectation value for ϵ is $\mathbb{E}\left[\epsilon_i\right] = 0$ since the distribution will vary around 0. We have that $\mathbb{E}\left[\epsilon^2\right] = \sigma^2$. We also need to remember that for independent variables the expectation of their product is the product of their expectations [2]

$$\mathbb{E}\left[a_i b_i\right] = \mathbb{E}[a_i]\mathbb{E}[b_i]. \tag{4.1}$$

The least squares method tries to minimise the function:

$$\mathcal{C}(\vec{X}, \vec{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \tilde{y}_i\right)^2 . \tag{4.2}$$

In an effort to understand how this function behave we will rewrite it as:

$$
\begin{aligned}
C(\vec{X},\vec{\beta}) =& \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2\\
=&\mathbb{E}\left[\left(\vec{y}-\tilde{\vec{y}}\right)^2\right]\\
=&\mathbb{E}\left[\left(\vec{y}-\tilde{\vec{y}}+\mathbb{E}(\tilde{\vec{y}})-\mathbb{E}(\tilde{\vec{y}})\right)^2\right]\\
=&\mathbb{E}\left[\left(\left(\vec{y}-\mathbb{E}(\tilde{\vec{y}})\right)-\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)\right)^2\right]\\
=&\mathbb{E}\left[\left(\left(f(\vec{x})+\vec{\epsilon}-\mathbb{E}(\tilde{\vec{y}})\right)-\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)\right)^2\right]\\
=&\mathbb{E}\left[\left(\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)-\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)+\vec{\epsilon}\right)^2\right]\\
=&\mathbb{E}\left[\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)^2-\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)^T\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)\right.\\
&+\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)^T\vec{\epsilon}-\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)^T\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)\\
&\left.+\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)^2-\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)^T\vec{\epsilon}+\vec{\epsilon}^T\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)-\vec{\epsilon}^T\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)+\vec{\epsilon}^2\right]
\end{aligned}
$$

Squaring yields cross terms with $\vec{\epsilon}$ and since all our stochastic variables also are independent we can use that $\mathbb{E}[\epsilon_i]=0$. The cross terms can be rewritten as:

$$\mathbb{E}\left[\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)^T\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)\right]=\mathbb{E}\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)\mathbb{E}\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right) \tag{4.3}$$

and since the expectation value of a number is the number itself

$$\mathbb{E}\left(\tilde{\vec{y}}-\mathbb{E}(\tilde{\vec{y}})\right)=\mathbb{E}(\tilde{\vec{y}})-\mathbb{E}(\tilde{\vec{y}})=0 \tag{4.4}$$

all the cross terms goes away and we are left with:

$$C(\vec{X},\vec{\beta})=\mathbb{E}\left[\left(f(\vec{x})-\mathbb{E}(\tilde{\vec{y}})\right)^2\right]+\mathbb{E}\left[\left(\tilde{\vec{y}}-\mathbb{E}\left(\tilde{\vec{y}}\right)\right)^2\right]+\sigma^2. \tag{4.5}$$

The first term is called the bias, while the second term is the variance. By looking at this function we can guess that the bias will be large when the model is chosen with too little complexity, such as trying to fit a high degree polynomial with a lower degree polynomial. Likewise, if we overfit the model, meaning we make a model which is too complex, the variance term will start to grow for data the model haven't seen yet. We should then see that the MSE is large initially, then gets lower before it grows again.

When implementing the Bias Variance trade off signature we had to split the initial data set into three: A test set, a validation set and a train set. The test set were held back from the start and were not sent into the cross validation algorithm (we used K-fold). Then, models were trained using the train sets and predictions were made using the test set.

4.3.1 Confidence interval

When considering some variable, with known variance, we can build a confidence interval. For the variable y_i the confidence interval becomes $y_i \pm \frac{1}{\sqrt{N}} z\sigma(y_i)$. Where the variance is $\text{Var}(\mathbf{y}) = \sigma(\mathbf{y})^2$ and the variable z is the confidence level. We chose a confidence level of 2 which means approximately 97% of the points will fall between our confidence interval. We especially consider the confidence interval for the variables β which we only have one of for each model. So when we find the confidence interval for each component we use:

$$\text{Conf}(\beta_i) = \beta_i \pm 2\hat{\sigma}(\beta_i). \tag{4.6}$$

The method for finding $\sigma(\beta)$ is described in equations (3.13) and (3.17). This gives the confidence interval for both OLS and Ridge, however we didn't have the variance for the variable Lasso. The Bayesian approach is to have the confidence interval of the predictors β which allows us to construct a probability distribution for the model and presents a range of models which are accurate to within some range determined by us.

Chapter 5

Results and Discussions

5.1 Fitting the Franke Function

The FrankeFunction has two Gaussian peaks of different heights and a smaller dip.

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

We use this as our test function, and we perform the initial analyses on the FrankeFunction to create a functioning program. This way we learn how the different regression methods work and we can try and make some insights into how the regression methods will behave on some given data set. We implemented the OLS numerically by performing a matrix inversion as described in equation (3.9). In figure (5.1) we can see how the fit approaches the Franke Function and get a fairly good fit at around degree 5. At degree 5 the noise in the original model has not yet caused overfitting in the model. The model of polynomial degree 5 had a good MSE value at 0.012020 and a R^2 score at 0.871311. Using just train_test_split yields accuracy of:

	Training set	Test set
MSE	0.01097932	0.01203822
R^2 score	0.87260651	0.85550317

Using K-Fold, we got 0.01040761 as our best MSE and 0.87416967 as our best R^2 for test data set.

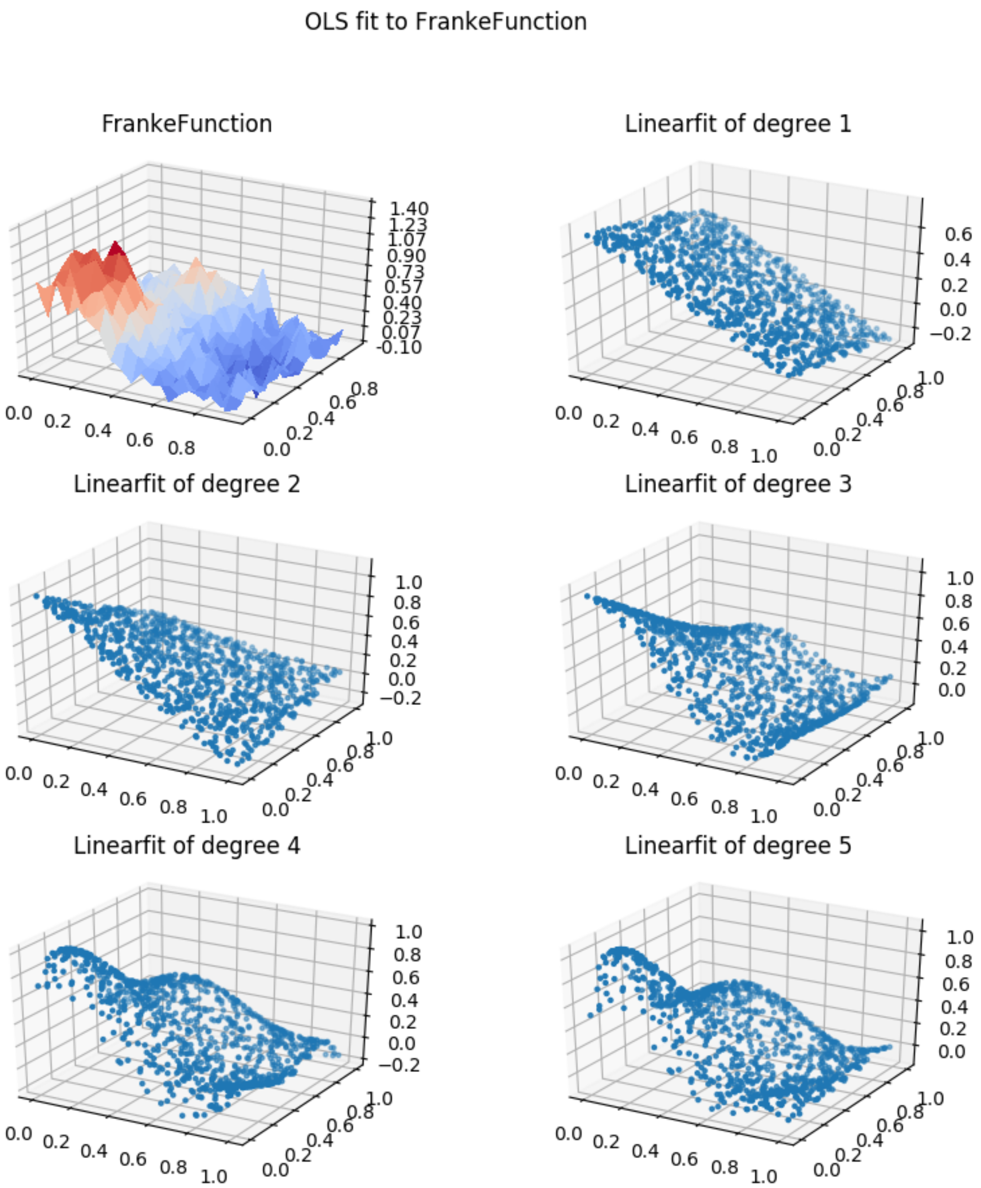


Figure 5.1: OLS fit to Franke's Function

5.1.1 Mean Squared Error and R2 score of the Franke-function

In order to investigate the effect of noise on the model we made a comparison between R2, MSE and Variance as shown in table 5.1. Here we can see that as we increase noise level, MSE and variance increased while the R2 score decreases. What this tells us is that as noise increase we capture less and less of the data in our model and for noise as high as $N(0, 0.7)$ almost all of the behaviour in the data cannot be explained by our simple model. At the same time the model attempts to fit to the noise as explained by the increasing variance. A higher degree polynomial would be able to better fit to the noise, though that is not something we would want.

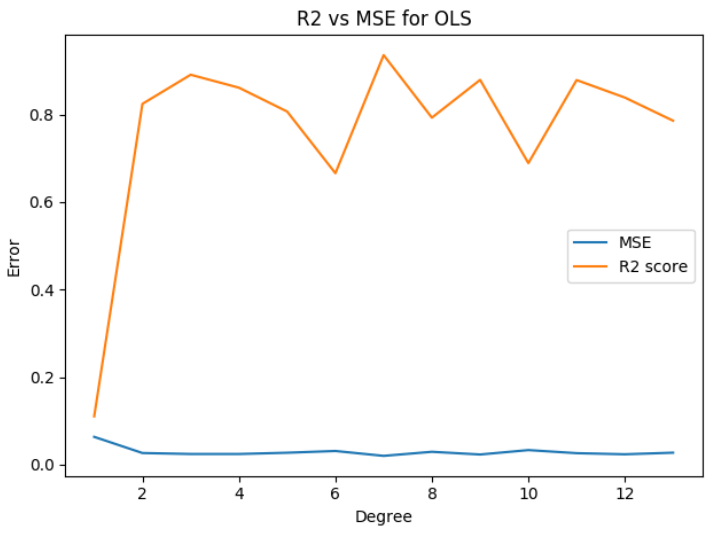
Noise	0.1	0.2	0.3	0.4	0.5	0.6	0.7
MSE	0.0113	0.0417	0.0907	0.1623	0.2389	0.3363	0.4793
R^2 score	0.8905	0.6491	0.5012	0.3461	0.2672	0.1854	0.1394
Variance	0.1034	0.1189	0.1820	0.2482	0.3261	0.4128	0.5569

Table 5.1: MSE, R^2 score and variance varies with noise from $N(0, 0.1)$ to $N(0, 0.7)$

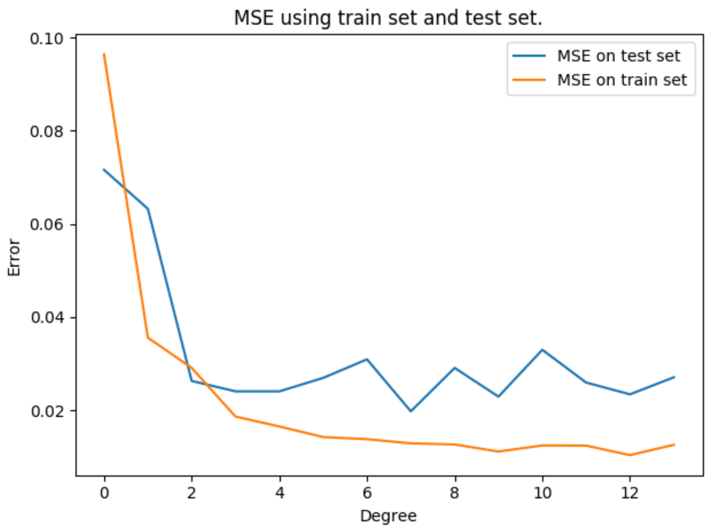
In order to investigate how well we could fit the polynomial despite of unknown levels of noise we could look at the Mean Squared Error (equation (3.3)) as well as the R2 score (equation (3.5) for varying polynomial degree. In the initial analysis we split the data into train data and test data and used K-fold cross validation to check whether the model were over or under fitting. From figure 5.2a we see that while the R2 score approaches 1, the mean squared error gets progressively lower. However, from figure 5.2b we see that the model is doing better on the training data than on the test data. This is a sign of overfitting.

5.1.2 Bias Variance for OLS

When we implemented the bias variance analysis to find where the overfitting is happening, we struggled a bit because the results seems to be pretty unstable. We had trouble with excessive or small errors at one polynomial degree, only having it returned to some stable value later. In figure (5.3) we see that the model has the best fit at around degree 6, for 200 points. For 500 points we have a good fit at around degree 3 to 7. Generally we found that the bias variance became stable for roughly the same low degree, but needed higher degrees before the error would blow up for larger datasets. Likewise, larger datasets gave a larger range of degrees for where the model gave good predictions. Since we saw that the model seemed to behave well at around degree 5 we ended up with focusing on a model of that degree.

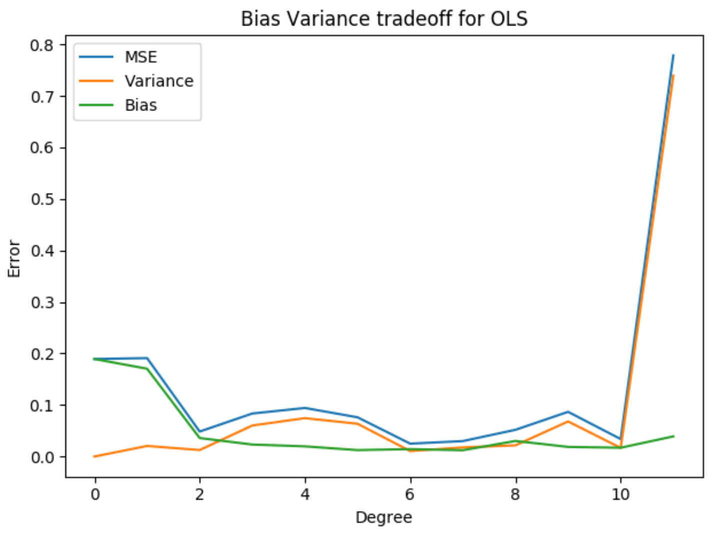


(a) R2 score plotted against MSE.

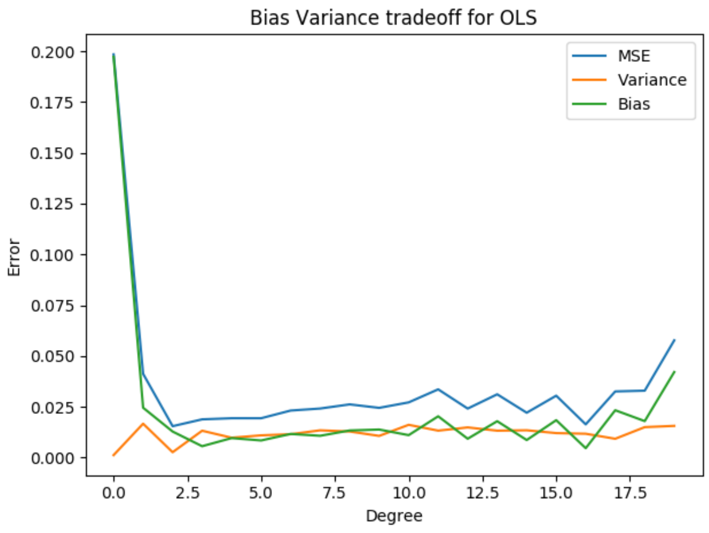


(b) MSE of test and train set.

Figure 5.2: R2 score and MSE against each other. $N = 1000$



(a) N samples is 200.



(b) N samples is 500.

Figure 5.3: Bias Variance for OLS. For larger datasets we will need a more complicated model to capture the bias variance well.

Introducing λ

In an attempt to improve the model further we considered Ridge and Lasso regression. As explained in sections (3.1.2) about least squares methods we can introduce a parameter λ in an attempt to tune the model. We try and improve the model at around degree 6. From figure (5.4) where we compare the effect of the parameter λ on Ridge and Lasso for 200 and 500 data points respectively, done for the validation set, we see that both Lasso and Ridge finds a minimum which perform better than OLS. For Ridge the optimal λ is higher for the solution with greater sample points, while for Lasso the optimal λ gets lower for larger number of sample points. Also, the contribution for few data points is much more substantial than for more data points. Comparing figure (5.4a) to figure (5.4b) the reduction of the error is on the order of 10^{-3} for 200 points, while the error us substantially smaller for 500 points.

To further investigate the effects of λ on the error. We looked at the error calculated from the test set instead of looking at the error calculated from the validation set as we did in figure 5.4. In figure 5.6 we see that the error, when found from the test data, no longer behaves as we would expect for Ridge and Lasso. The error only gets larger as we increase Lambda. So for this case if we did not test the function on validation data it would have seemed that Ridge and Lasso regression where not capable of improving the fit, something they indeed do for some ranges of lambda.

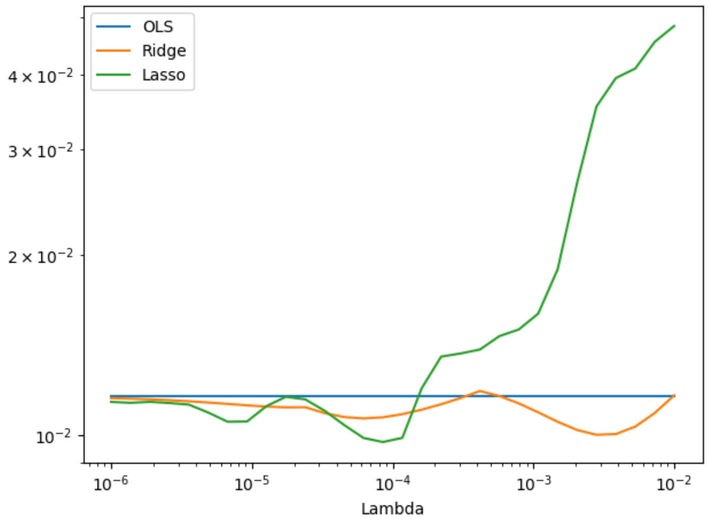
Another result from figure 5.6 is that both Lasso and Ridge specifically reduce variance. From table 5.1 we saw that increased noise increased the variance of the model the most. In figure 5.5 we have increased the noise from $N(0, 0.1)$ to $N(0, 0.8)$ and in this case we do indeed manage to reduce the MSE in both Ridge and Lasso to below that of OLS for the test set and not just the validation set.

Ridge

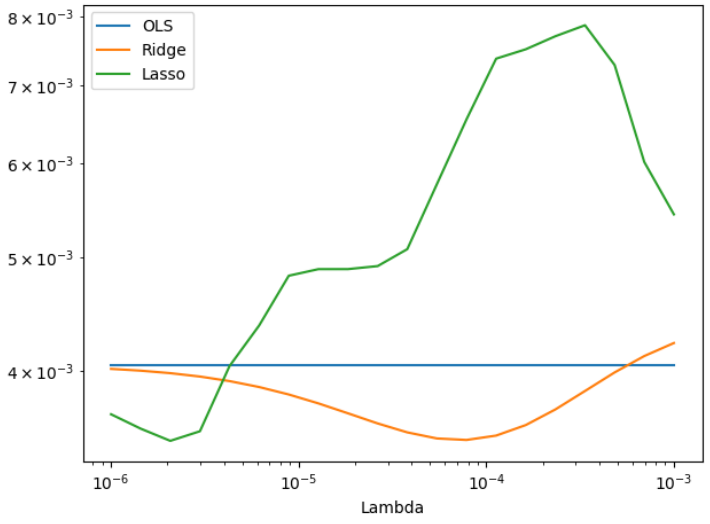
When implementing Ridge regression the change in error is fairly modest. From figure (5.7) it is hard to get consistent results, and for any particular model the influence of Ridge is not obvious. However, for any given model complexity as shown in figure (5.4) there is a systematic gain for a given λ. The value of this optimised parameter change depending on the model complexity.

Lasso

The results from Lasso regression is seen in figure (5.8). The results are similar to those from Ridge, but there is a more pronounced suppression of the variance at some points, except variance for very low degrees. Comparing OLS from figure (5.3b) at 4k points with both Lasso and Ridge at 4k points, it is tempting to see a suppression of the variance at varying model complexity.



(a) N samples is 200.



(b) N samples is 500.

Figure 5.4: Finding the optimal λ for Ridge and Lasso at degree 5.

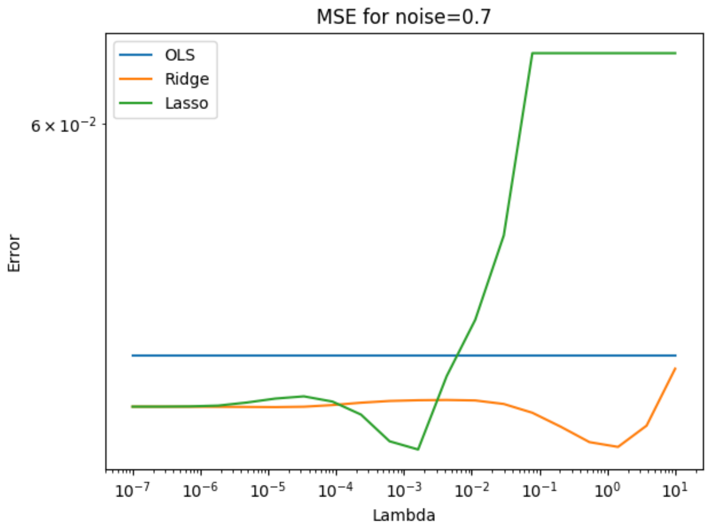
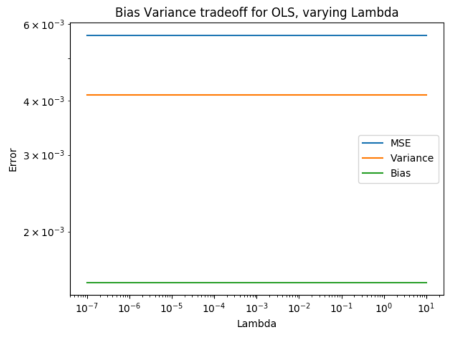


Figure 5.5: Finding the optimal λ for Ridge and Lasso at degree 5, N=600 and noise=N(0,0.7).

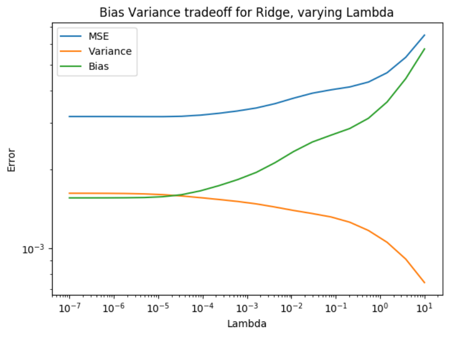
Components of β and its variance $\sigma(\beta)$

From figure (5.9) we see that the error of the different components of β is fairly large for low sample points. At $N = 200$ the error dominate over the values of β. This implies we could have a range of values of β for which we still would get a good fit. When we increase the number of data points to 500, however, the confidence interval becomes a bit narrower. In Ridge regression, plotted in figure (5.10) we see the evolution of the different β values as we change λ. From this plot we see clearly that some βs are brought to 0 much quicker than other βs. In effect we can see what we theorised in section (3.1.2) about Ridge, that the βs contributing the least to the variance are brought towards 0 first, while more important βs are kept for longer. Indeed, as λ grows too much all of β are brought to 0. This tendency are still there where we have plotted for samples $N = 500$, but we need a larger lambda for the same effect as for $N = 200$.

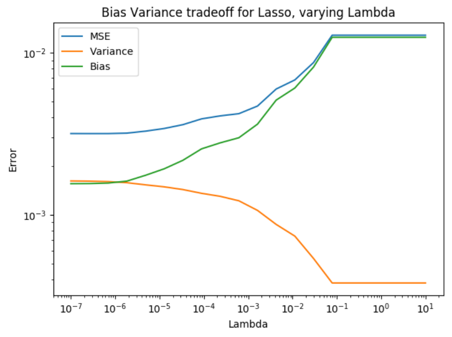
In Lasso regression we could not find the confidence interval for β^{Lasso}. We did, however, plot the components of β^{Lasso} as we vary λ in figure (5.11). In this case it is obvious that some components of β is brought directly to 0 much faster than other components. However, instead of gradually going to 0 above some threshold nearly all components are 0. This tendency are more pronounced for larger data sets, where components are brought close to 0 very quickly. We suspect we can use this to run a preliminary test run and pick out some β components and completely remove them from the design matrix when making the final β intended for production, speeding up the code.



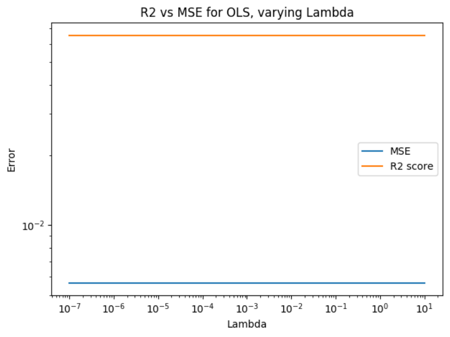
(a) Bias Variance for OLS.



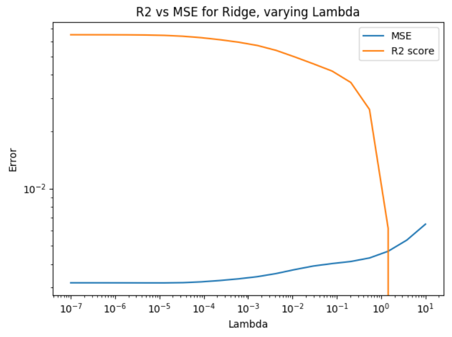
(b) Bias Variance for Ridge.



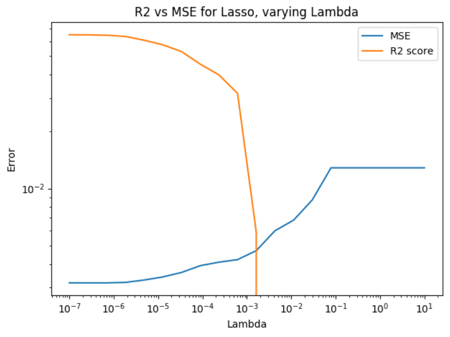
(c) Bias Variance for LASSO.



(d) Bias Variance for OLS.

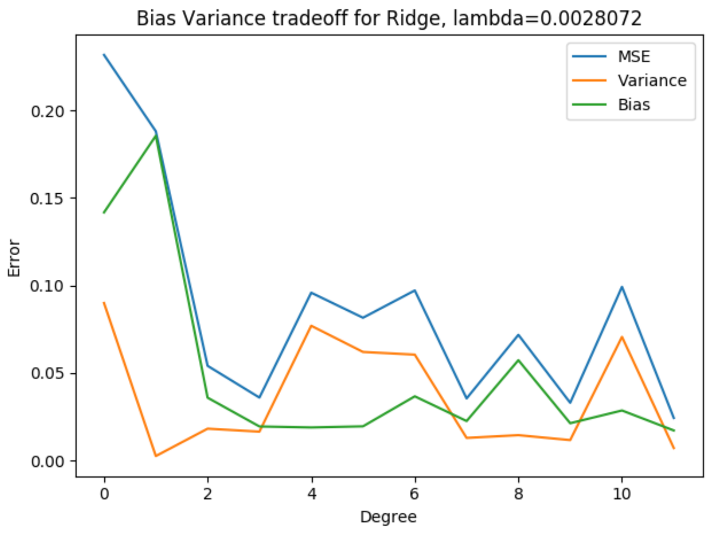


(e) R2 vs MSE for Ridge.

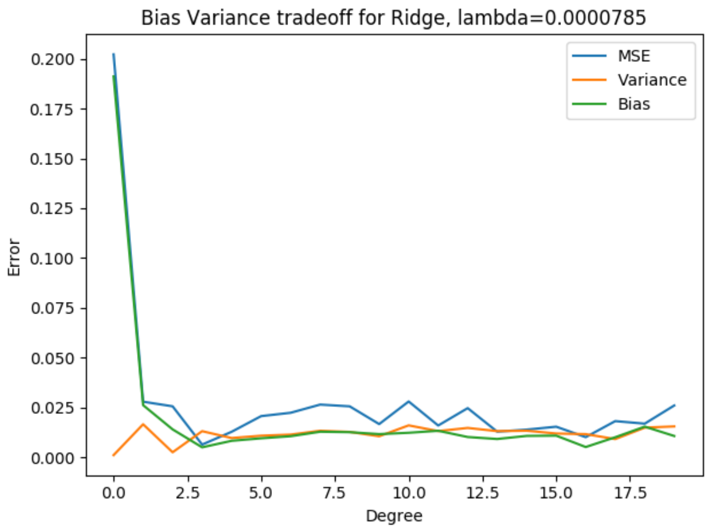


(f) R2 vs MSE for LASSO.

Figure 5.6: Finding the optimal λ for Ridge and Lasso at degree 5, N=600 and noise=N(0,0.1).

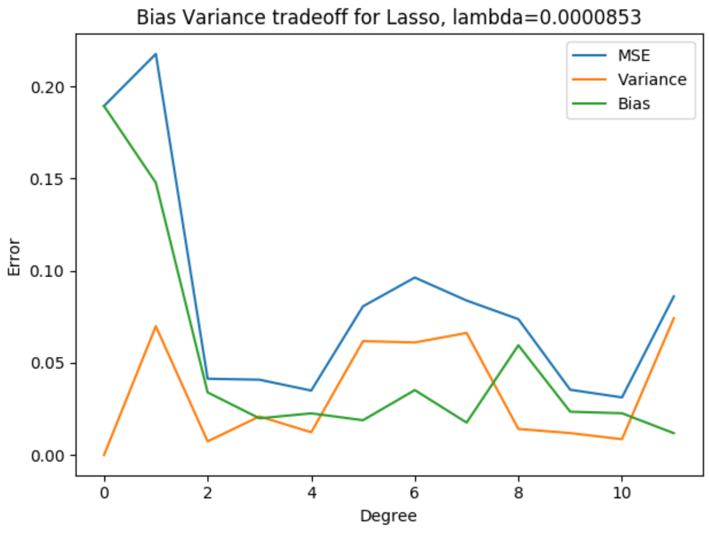


(a) N samples is 200.

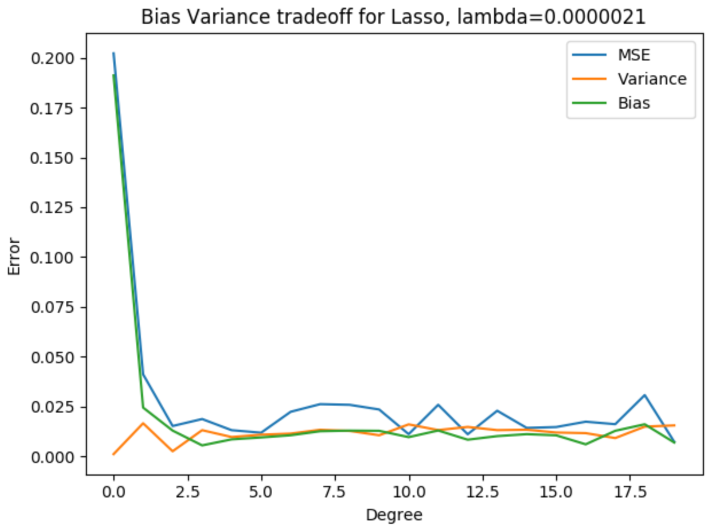


(b) N samples is 500.

Figure 5.7: Bias variance trade-off for Ridge with varying degrees.

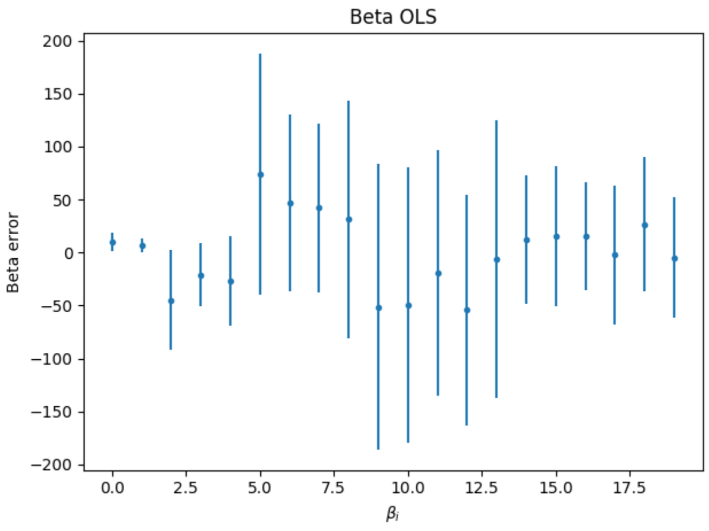


(a) N samples is 200.

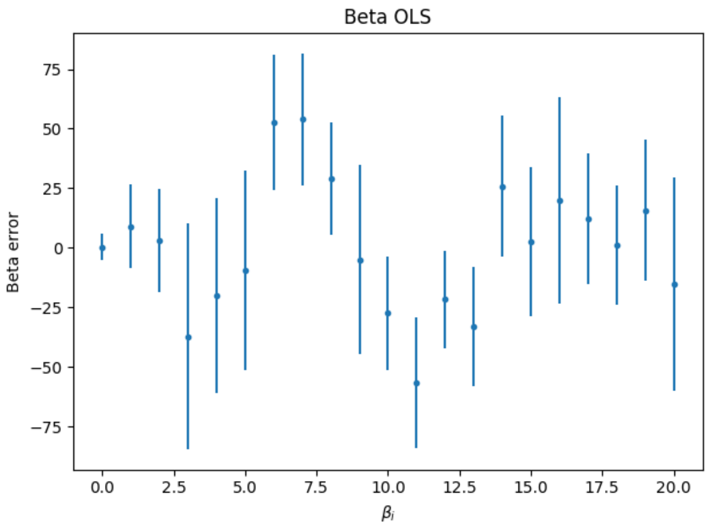


(b) N samples is 500.

Figure 5.8: Bias variance trade-off for Lasso with varying degrees.

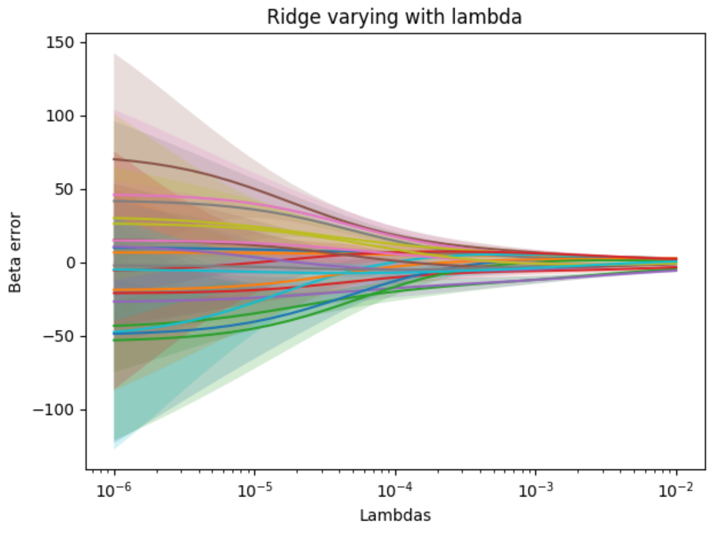


(a) N samples is 200

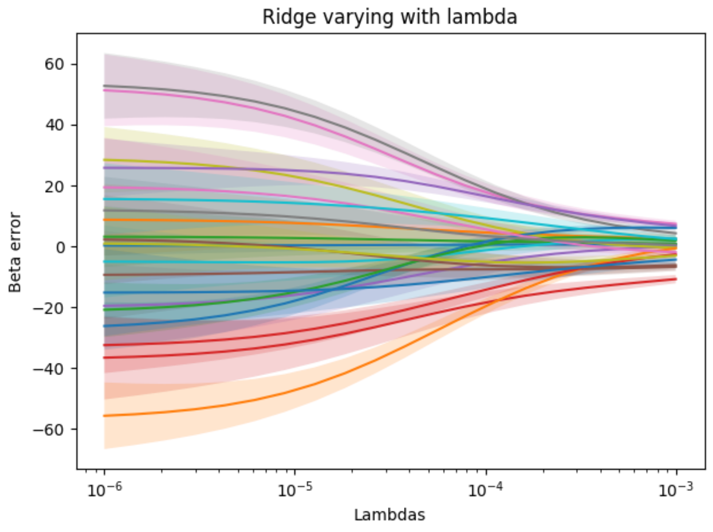


(b) N samples is 500.

Figure 5.9: The value of β along with the error bars for each β for OLS at degree 5.

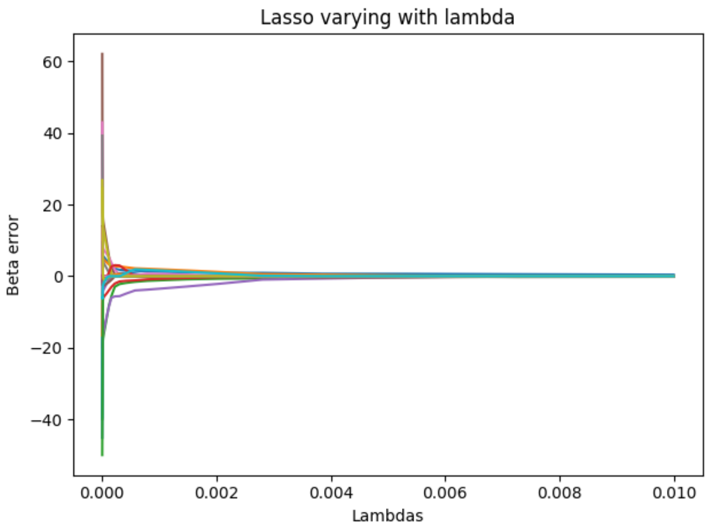


(a) N samples is 200

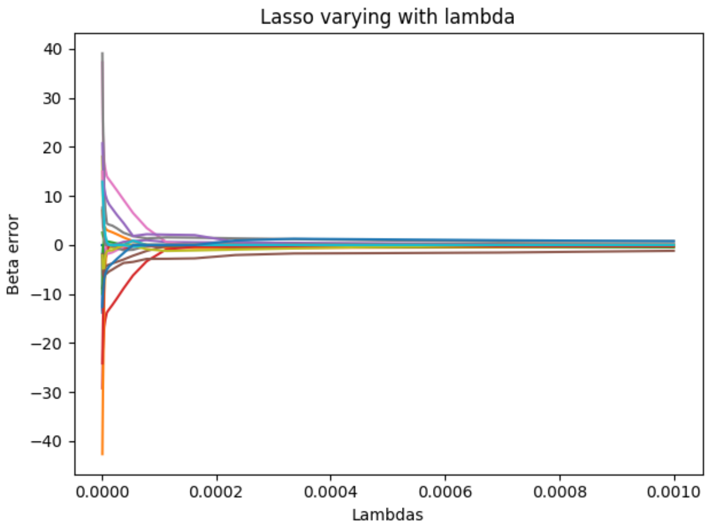


(b) N samples is 500.

Figure 5.10: The value of β along with the error for each component of β for Ridge at degree 5.



(a) N samples is 200



(b) N samples is 500.

Figure 5.11: The value of β varying with λ for LASSO at degree 5.

5.1.3 Summary

When optimising a model when using Linear Regression, there are primarily two ways of performing a best possible fit. Considering bias and variance, there is an optimal complexity which will lead to the model being neither under fitted nor over fitted. For the FrankeFunction this proved to be around degree 5. For this optimal model complexity it is possible to improve the model further by using Ridge or Lasso. The way we implemented Lasso in particular, using SciKitLearn, the method quickly became prohibitively expensive computationally. Using LassoCV from SciKitLearn would have been an improvement, as this functionality has cross validation built in. This does, however, make the analysis we made with bias variance more difficult, since we rely on our own cross validation algorithm to perform this analysis.

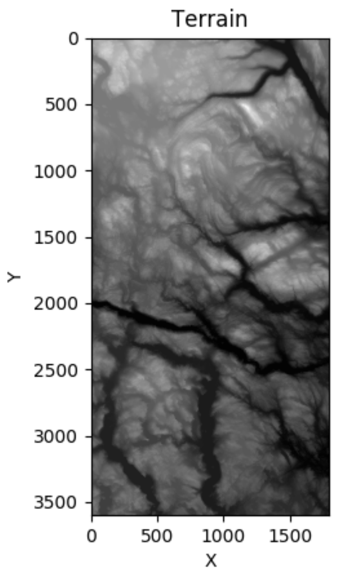
As mentioned earlier, both Lasso and Ridge reduce variance, and from table 5.1 we saw that increased noise increased the variance of the model the most. Thus, it is possible Lasso and Ridge might reduce the error when estimated on the test set on models with higher levels of noise, but will reduce MSE on models with too little noise. This appears to be what we see when we compare the errors for less noise (figure 5.6) with more noise (figure 5.5).

5.2 Fit to terrain over Norway

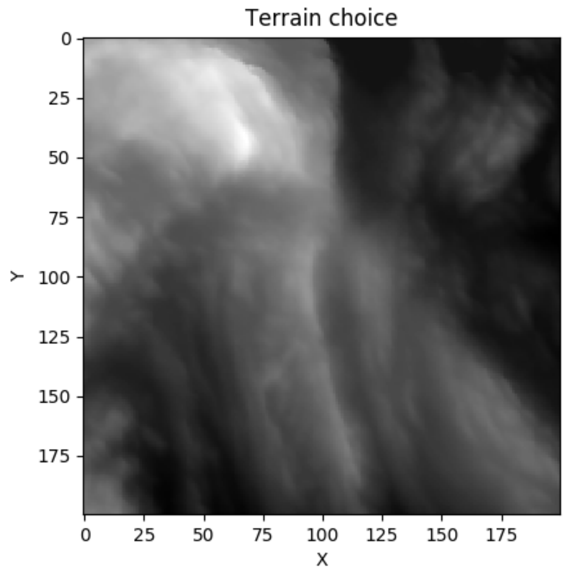
We made a fit to some terrain over Norway, figure (5.12a). From section (5.1.2) we saw that too many points can lead to a greater variety of models making a good fit. The terrain data has broad features on the level of half the map being the top of a mountain, but it also has much finer features. In the intermediate range between the rough and finer features we expect our model to plateau before picking up finer features, before being worse again due to overfitting. Because of this we can choose fewer points (in effect rescaling the picture and selection one every ten points to consider) in order to avoid a very complex model before overfitting becomes obvious.

For simplicity we looked at an area on the middle of the map 200 by 200 pixels shown in fig (5.12b). If we use many points, we see that the fitting does not benefit from the Lasso and Ridge regression methods as can be seen in figure (5.14a). However when we use relatively few data points we see that Lasso and ridge can reduce the mean squared error for certain ranges of lambda, figure(5.14). In figure(5.15) we can see how the bias variance trade is different for OLS, Ridge and Lasso.

We see that with few data points Ridge reduces overfitting such that higher order polynomials give good approximations and Lasso allows us to use even higher order Polynomials. For 640 training samples we see that there is a wide range of polynomials that give relatively good fits, for all three methods. In figure 5.17 we can see that the a polynomial of degree 8 fit with OLS gives a good overall fit when we split the terrain into smaller areas. This method of splitting and fitting gave an R2score of 0.999651114 over the whole terrain. The



(a) Terrain over Norway



(b) Smaller choice of terrain, bottom left corner is at (800,100) on the larger terrain

Figure 5.12: Terrain over Norway

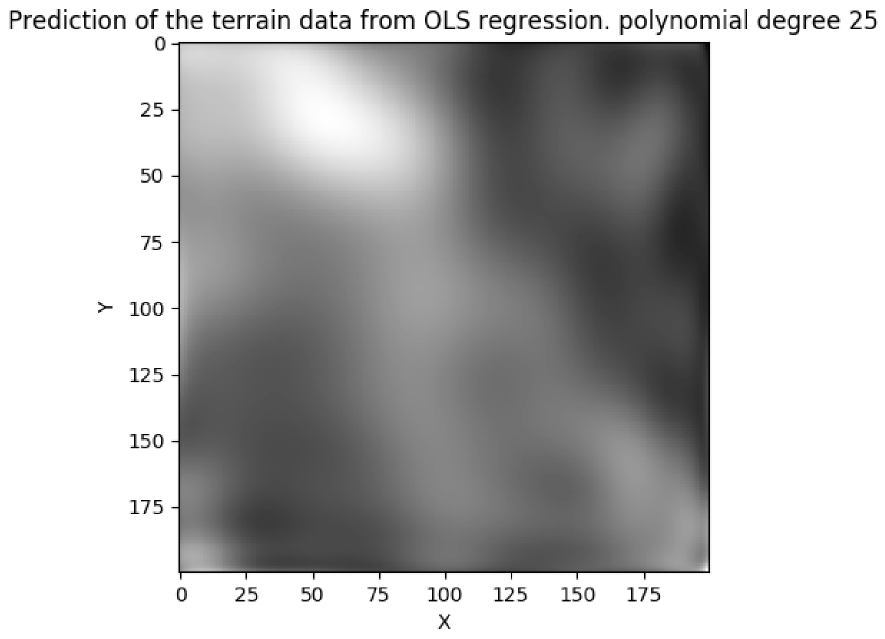
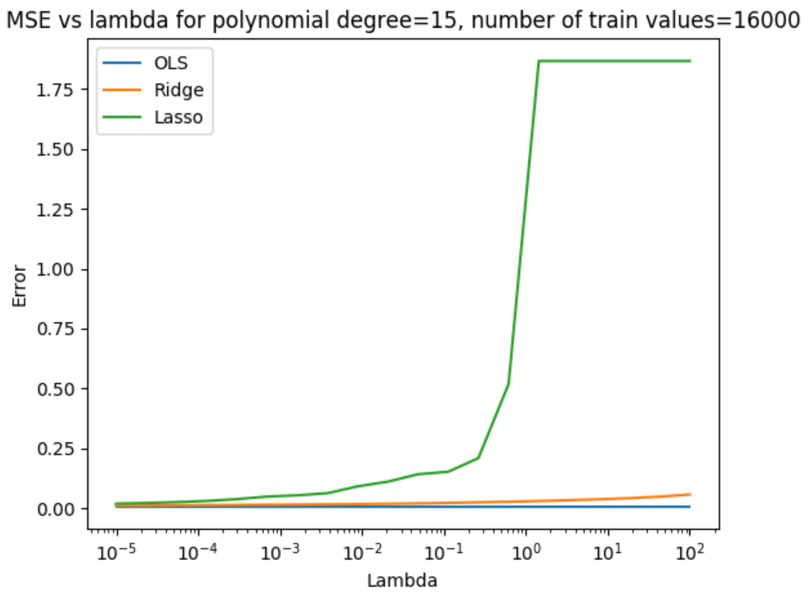
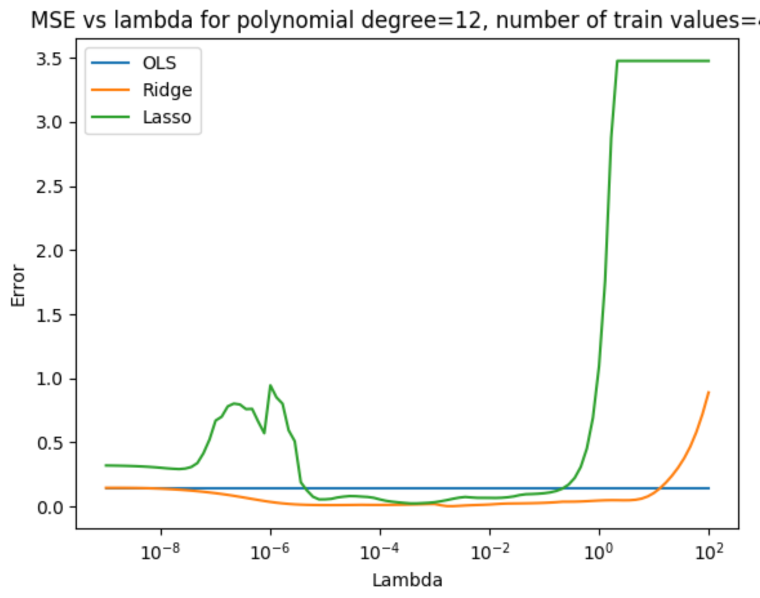


Figure 5.13: Prediction of the terrain data

results where as good as identical for Ridge regression on the same areas because every tenth point is so many points that we have no problems with overfitting with 8th degree polynomials. However if we use very high degree polynomials we can see the borders between the areas much better because of overfitting that becomes apparent on the edges of the areas and especially in the corners and this also causes lower R2 score at around 20-25 degree polynomials.

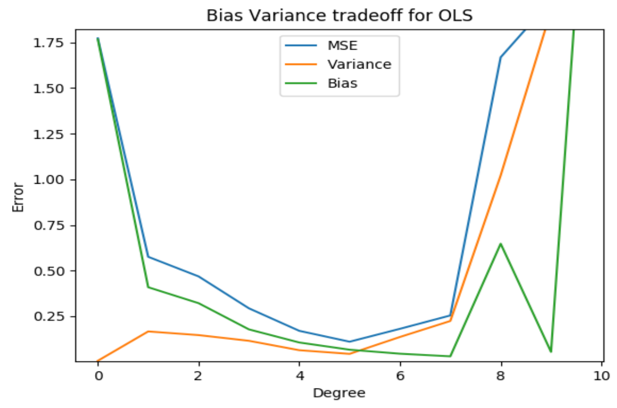


(a) N samples is 16000

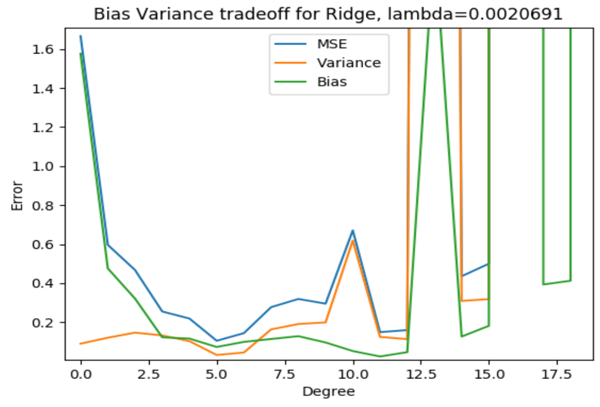


(b) N samples is 40

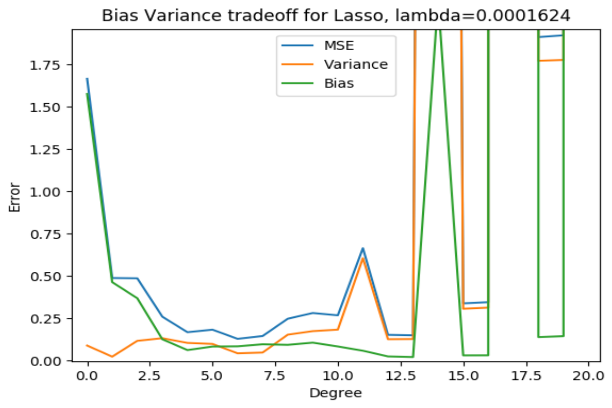
Figure 5.14: MSE dependence on lambda for many and few samples



(a) OLS, N samples 40

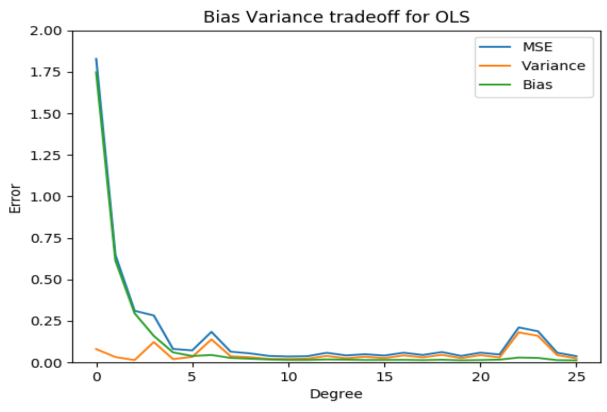


(b) Ridge, N samples is 40

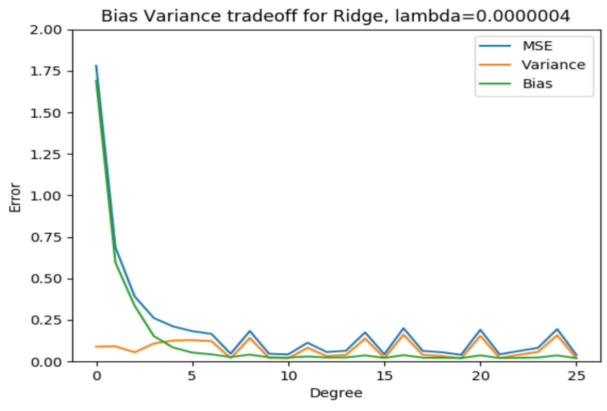


(c) Lasso, N samples is 40

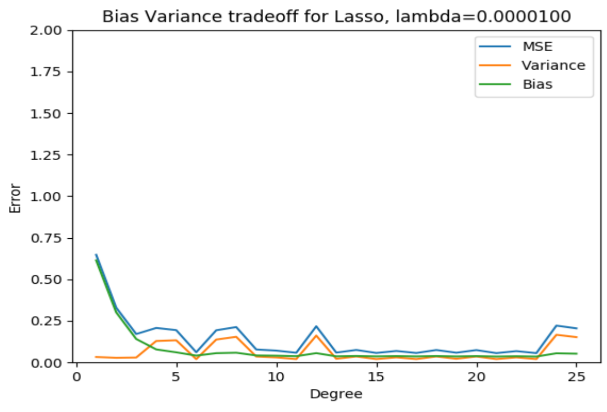
Figure 5.15: Bias variance trade off for 40 train samples



(a) OLS, N samples 640



(b) Ridge, N samples is 640



(c) Lasso, N samples is 640

Figure 5.16: Bias variance trade off for 640 train samples

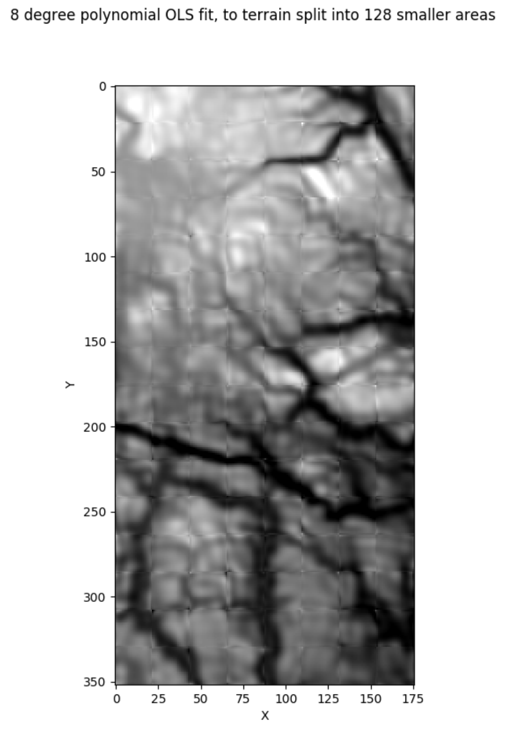


Figure 5.17: Function fit to the whole terrain, split into smaller areas

Chapter 6

Conclusions and Perspectives

The objective of this project was to study ordinary least squares, Ridge and Lasso Regression methods and resampling techniques. First we used the methods on the Frankefunction with added noise. We saw that Ridge and Lasso performed better for higher polynomial degrees and also for small numbers of training points. The drawbacks of Lasso are worse fits if lambda is to large and high computation times especially for as small values of lambda which are often necessary for Lasso to perform better than OLS. Ridge regression on the other hand does not require significantly more computational time than OLS. Ridge also increases the stability if the matrix to be inverted in OLS is singular or close to singular.

We further explored the effect of noise on OLS, Ridge and Lasso. Since both Ridge and Lasso use feature selection they can help reduce the adverse effect of noise on the fit. What we found from this project is that any advantage Ridge and especially Lasso has is very small compared to the additional computational cost. However, it is possible to improve the MSE under some circumstances, even though the gain is questionable.

One thing that could be done would be to try to use different functions than polynomials to fit the data to, possibly other functions would require fewer parameters to give good fits and thereby be able to reduce the computation times. Another thing that could be done would be to compare the cross validation resampling method to the bootstrap method.

Chapter 7

Appendix

We used the files in https://github.uio.no/thomhaa/Fys-Stk4155.git. When running the files we used the parameters in table 7.1

N	k
200	5
500	5
600	20
1000	50

Table 7.1: Datapoints and K-fold used to produce plots.

List of codes on GitHub

`oblig1_a.py`

This is used for used for initial consideration of the FrankeFunction.

`oblig1_frankFunc.py`

This is used to find the Bias Variance trade as a function of degree.

`oblig1_frankeFunc_lambdaOnTest.py`

This is used to find the Bias Variance trade as a function of lambda.

`oblig1_frankeFunc_lambdaOnVal.py`

This is used to find an optimal lambda based on the validation set.

`oblig1_g.py`

This is to run the terrain data.

`whole_terrain.py`

This is used to fit a function to the whole terrain data.

`oblig1_header.py`

This includes all the functionalities and statistical functions.

`oblig1_tests.py`

This is test file, to perform unit tests.

7.0.1 Confidence interval of parameter β computed with Ordinary Least Square on the Franke function

Table 7.2: β-values and their confidence intervals calculated using OLS for Franke's function with noise $N(0, 0.1)$

β	σ	**Confidence interval**
0.38233798	0.02112134	[0.37330423, 0.39137173]
8.11603644	0.07010604	[8.08605159, 8.14602129]
2.96436871	0.07015813	[2.93436158, 2.99437584]
-35.412406	0.15338047	[-35.47800791, -35.34680409]
-15.10458874	0.13556302	[-15.16257, -15.04660747]
-11.93419702	0.15342456	[-11.99981779, -11.86857625]
49.91889055	0.22938327	[49.8207817, 50.01699939]
48.1332878	0.19750738	[48.0488125, 48.2177631]
21.75762608	0.1976047	[21.67310915, 21.842143]
3.47006202	0.22938342	[3.37195311, 3.56817093]
-24.62287754	0.23420774	[-24.72304984, -24.52270524]
-56.83644759	0.20473158	[-56.92401273, -56.74888244]
-9.66796156	0.19755822	[-9.7524586, -9.58346451]
-33.01060464	0.20481242	[-33.09820436, -32.92300492]
17.72248014	0.23418636	[17.62231698, 17.8226433]
1.64744543	0.1466673	[1.58471479, 1.71017606]
19.91589129	0.13681641	[19.85737394, 19.97440864]
11.83849186	0.13515663	[11.78068442, 11.89629931]
-5.26897655	0.13519264	[-5.32679939, -5.2111537]
18.74851597	0.13685008	[18.68998422, 18.80704771]
-12.70776106	0.14665644	[-12.77048706, -12.64503506]

Table 7.3: β-values and their confidence intervals calculated using OLS for Franke's function without noise

β	σ	**Confidence interval**
0.38164787	0.02112721	[0.37261161, 0.39068412]
8.11603644	0.0701545	[8.08894013, 8.14895128]
2.96198827	0.07011565	[2.93199931, 2.99197723]
-35.41821437	0.15340769	[-35.48382792, -35.35260081]
-15.1263486	0.13558334	[-15.18433855, -15.06835865]
-11.89391202	0.15335064	[-11.95950118, -11.82832287]
49.94692892	0.2293582	[49.8488308 50.04502704]
48.16477903	0.19760262	[48.08026299 48.24929506]
21.7755017	0.19751992	[21.69102103 21.85998236]
3.35706124	0.22930082	[3.25898766 3.45513482]
-24.67336727	0.23415581	[-24.77351736 -24.57321717]
-56.84346021	0.20478577	[-56.93104853 -56.75587189]
-9.72388199	0.19753464	[-9.80836895 -9.63939503]
-32.98686278	0.20471146	[-33.07441932, -32.89930624]
17.84145032	0.23410683	[17.74132117, 17.94157947]
1.67283593	0.1466292	[1.61012158, 1.73555027]
19.92277142	0.13680819	[19.8642576, 19.98128525]
11.83656996	0.13513935	[11.77876991, 11.89437001]
-5.23753222	0.13510837	[-5.29531902, -5.17974541]
18.72372907	0.13677089	[18.6652312, 18.78222694]
-12.75049759	0.14659803	[-12.81319861, -12.68779658]

7.0.2 Tables for Bias Variance trade off as function of λ

λ	Error	Bias	Variance	R2 score
1e-07	0.00565	0.00152	0.00412	0.065

Table 7.4: The error from figure 5.6a

λ	Error	Bias	Variance	R2 score
1e-07	0.003	0.002	0.002	0.065
2.63e-07	0.003	0.002	0.002	0.065
6.95e-07	0.003	0.002	0.002	0.065
1.83e-06	0.003	0.002	0.002	0.065
4.83e-06	0.003	0.002	0.002	0.065
1.27e-05	0.003	0.002	0.002	0.064
3.36e-05	0.003	0.002	0.002	0.064
8.86e-05	0.003	0.002	0.002	0.063
2.34e-04	0.003	0.002	0.002	0.061
0.001	0.003	0.002	0.002	0.059
0.002	0.003	0.002	0.001	0.057
0.004	0.004	0.002	0.001	0.054
0.011	0.004	0.002	0.001	0.050
0.030	0.004	0.003	0.001	0.046
0.078	0.004	0.003	0.001	0.042
0.207	0.004	0.003	0.001	0.036
0.546	0.004	0.003	0.001	0.026
1.438	0.005	0.004	0.001	0.006
3.793	0.005	0.004	0.001	-0.036
10.000	0.006	0.006	0.001	-0.134

Table 7.5: The error from figure 5.6b

λ	Error	Bias	Variance	R2 score
1e-07	0.003	0.002	0.002	0.065
2.63e-07	0.003	0.002	0.002	0.065
6.95e-07	0.003	0.002	0.002	0.065
1.83e-06	0.003	0.002	0.002	0.064
4.83e-06	0.003	0.002	0.002	0.061
1.27e-05	0.003	0.002	0.001	0.057
3.36e-05	0.004	0.002	0.001	0.053
8.86e-05	0.004	0.003	0.001	0.045
2.34e-04	0.004	0.003	0.001	0.040
0.001	0.004	0.003	0.001	0.032
0.002	0.005	0.004	0.001	0.006
0.004	0.006	0.005	0.001	-0.071
0.011	0.007	0.006	0.001	-0.170
0.030	0.009	0.008	0.001	-0.817
0.078	0.013	0.012	0.000	-8.27e+28
0.207	0.013	0.012	0.000	-8.27e+28
0.546	0.013	0.012	0.000	-8.27e+28
1.438	0.013	0.012	0.000	-8.27e+28
3.793	0.013	0.012	0.000	-8.27e+28
10.000	0.013	0.012	0.000	-8.27e+28

Table 7.6: The error from figure 5.6c

Bibliography

[1] *CompPhysics/MachineLearning*. original-date: 2017-09-18T20:11:45Z. Sept. 2019. URL: `https://github.com/CompPhysics/MachineLearning` (visited on 09/24/2019).

[2] Geoffrey Grimmett and David Stirzaker. *Probability and Random Processes*. Third Edition. Oxford, New York: Oxford University Press, May 2001. ISBN: 978-0-19-857222-0.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. en. Google-Books-ID: tVIjmNS3Ob8C. Springer Science & Business Media, Aug. 2009. ISBN: 978-0-387-84858-7.

[4] Robert Tibshirani. "Regression Shrinkage and Selection Via the Lasso". en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 2517-6161. DOI: `10.1111/j.2517-6161.1996.tb02080.x`. URL: `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x` (visited on 10/06/2019).

[5] Wessel N. van Wieringen. "Lecture notes on ridge regression". en. In: *arXiv:1509.09169 [stat]* (Sept. 2015). arXiv: 1509.09169. URL: `http://arxiv.org/abs/1509.09169` (visited on 09/23/2019).