Machine learning Project 1

Arnoldas Seputis

October 2018

1 Abstract

We study the different regression methods first on our own generated data and then on a real-world terrain data. We evaluate the different methods and find that in this case, where the input variables are uncorrelated, it only makes sense to use OLS method. We also develop a bootstrap re-sampling algorithm and simulate different experiments to find the total error over the data, which is expressed as the sum of the noise variance, the bias and variance in the fittings. Then we investigate the bias-variance trade-off to find the best fit which has a good enough bias and not too much over-fitting.

2 Introduction

Regression analysis is one of the most widely used methods in machine learning. These methods are usually used as a part of some supervised learning algorithms whose purpose is to predict the behavior of a dependent variable with respect to one or several independent ones. Therefore the purpose of the present project is to apply different regression methods to the Franke function in order to study their behaviour. In particular three main methods will be studied: ordinary polynomial regression, polynomial lasso regression and polynomial Ridge regression.

In order to study the effectiveness of these methods, a set of points for the Franke function was generated for x and y values in the [0,1] interval.

In order to determine the best fits for the data and suitable confidence intervals, the bootstrap resampling technique was used. This was carried out by implementing an algorithm which would select N number of points with replacement.

The developed methods were later used on the real-world terrain data provided

3 Theory

Probably the most popular regression method is the ordinary least squares methods. The idea in this method is to propose a known function which will model

the relationship between the dependent and independent variables. This function will depend upon some parameters which will be determined by fitting the measured data in the function. In order to fit the data the dependent variable is expressed as a function of the independent variables using the proposed function and letting the parameters undetermined. Then an error function which compares the predicted values of the dependent variable with its measured values is postulated. The mean squared error is a very popular one. Then the error function gets minimized with respect to the parameters in order to determine their value. Once the parameters are determined they are inserted into the function and the function then can be used to predict new values.

Ridge and Lasso regression.

Often times when one has a model that includes various independent variables it can happen that these independent variables have a strong correlation with each other. This can lead to high levels of variability on the predicted values when slightly varying the independent variables. This is so, as varying the value of an independent variable will affect the value of the dependent variable and also the values of the other independent variables to which it is correlated. Therefore a model with a lot of correlation between the independent variables will present a high variance. Hence in order to limit the variance of the model it is necessary to constraint the value of the function's parameters. This is known as a regularization method. Hence the Ridge and Lasso regression methods are similar to the ordinary least square methods, with the difference that the minimization of the error function will be done subject to a constraint for the function's parameters. In the Ridge regression case the square of the norm of the parameters vector (L2 norm) will be constrained. In the Lasso regression the norm of the parameters vector (L1 norm) will be constrained. This constraining of the parameters of the model function results in that they cannot grow too large and thus the value of the variance is kept under control.

In practice the difference between these two regressions is that the shape of the constraint of the Lasso regression is much sharper thus in tends to get rid much faster of the least important features that in the Ridge case. Hence in a typical Lasso regression model, many of the parameters have zero value.

4 Discussion of the methods:

All the scripts are ready to be run in the github file, the readers will be redirected to the source if they wish to test the code for themselves. All the scripts import and use the algorithms from the common classes that we wrote, called main_classes.py in part a, main_classes_ridge.py in part b, main_classes_lasso.py in part c and main_classes_real_data.py in part e. All of these classes are just modifications of each other, only to accommodate the slight difference in the nature of the problems. The following can be found in the main_classes_* scripts: For regression, three methods have been used: Ordinary Least Squares(OLS), Ridge and Lasso regressions of which only the first two have been implemented by us and the last one being imported from the sklearn library. For the resam-

pling part, we have developed our own bootstrap method.

OLS and Ridge has an almost identical implementation, since OLS is just a special case of Ridge. We will therefore only present the Ridge method here, as OLS is implied. OLS regression is essentially finding the distance from the given data vector that we try to find a fit for and it's projection onto the subspace, spanned by the basis of the polynomial of degree n that we choose. What makes this a Ridge regression, however, is that we introduce a term in the cost function to be minimized, which constrains the beta variables to each other in a way that keeps them coupled close to each other,not permitting divergence between them. In our algorithm for Ridge and OLS we simply used the matrix expressions to derive them(this is done in our Ridge_main class). $\beta_{ridge} = (X^T X + \lambda I)^{-1} X^T y$ where y is the data presented to us and λ is the constant from the constraint term, that balances the β variables. Setting $\lambda = 0$ we get the expression for OLS. To be begin with we ran codes with this raw implementation, but eventually changed to first performing an SVD on the $(X^T X + \lambda I)$ part, before inverting it, because inversion is a computationally expensive operation and unstable, if one is to compare the existing numpy and scipy libraries. We therefore used the scipy linear algebra module to decompose the mentioned term and then take the inverse of each one, which sped up the computation substantially, it also eventually got rid of a MemorryError when working on the real terrain data. This is the excerpt of the code that implements the SVD before inverting. The commented out section shows the older version of taking the inverse directly without the SVD.

```
        self.beta_lin_reg = self.find_beta()
#self.beta_lin_reg = (np.linalg.inv(self.X_.T.dot(self.X_) + self.lamd*Id).dot(self.X_.T)).dot(self.z_)
        self.z_fit_ = self.X_.dot(self.beta_lin_reg)
        #self.z_fit=self.z_fit_.reshape((n,n))

    def find_beta(self):
        Id = np.identity(self.X_.shape[1])
        to_be_inverted = self.X_.T.dot(self.X_) + self.lamd*Id
        U, D, V_T = scipy.linalg.svd(to_be_inverted, full_matrices=False)
        D_inv = 1/D
        D_inv = np.diag(D_inv)
        inverted = V_T.T.dot(D_inv.dot(U.T))
        return (inverted.dot(self.X_.T)).dot(self.z_)
```

Figure 1: Code example

$X^T X$ is a symmetric matrix, that we want to take the inverse of. Since it's symmetric, an SVD decomposition gives us $X^T X = VDV^T$ where V are the orthogonal vector space that spans the original matrix and D are the diagonal eigenvalue matrix. $(X^T X)^{-1} = (VDV^T)^{-1} = VD^{-1}V^{-1} = VD^{-1}V^T$. However it does not seem like the algorithms care to find the orthonormal basis, meaning that even though in theory $V^{-1} = V^T$, in our case the algorithms do not scale the V, so we just implement $(VDV^T)^{-1} = (V^T)^{-1}D^{-1}V^{-1}$ in our code just to be safe.

The example code for Ridge or OLS regression can be found in folders part_a

and part_b, where one can run part_a_regression.py or part_b_regression.py. As already mentioned, lasso regression code was imported from sklearn and can be found in folder part_c in an example script to be run, called part_c_lasso_reg_plot.py.

To calculate the variance in beta and to assess the confidence intervals, we used a derived formula(reference 1) for the variance in beta for Ridge regression. Setting $\lambda = 0$ we used it also for OLS case. The formula we found is $Var(\beta_{ridge}) = (X^T X + \lambda I)^{-1}(X^T X)(X^T X + \lambda I)^{-1}\sigma^2$ where σ^2 is the variance in the data y. For the the Franke's function, we introduced our own normal distribution error, so we can just set $Var(eps) = C^2$ where C is the number we choose ourselves to factor the np.random.randn(0,1) noise. For the real data we do not know the noise term. If we could repeat the experiment a sufficient amount of times, we could get a very fair estimate of the error term at each point, but we only are given one set, so what we do is just assume that we have a normal distributed error term and make an estimate of it by taking a mean squared error between the data and the best fit that we get. This way we can extract a reasonable estimate for the beta variables for the real data in Ridge an OLS case.

To do resampling the bootstrap method has been chosen. Firstly, a bootstrap class has been written in the Main_classes_*.py files. It takes the length of the input data array and generates a list of indices that are taken randomly with "putting back". This array can be called the "training indices" array. The indices that do not make it to the "training indices" array, are stored into a new "test indices" array, so it becomes of varying length, depending on how many of the original indices did not make it to the "training" array. Then we created another class called run_the_bootstraps that loops over different instances of the bootstrap and for each creates the training and test arrays on which we then run the relevant regression method. The code for these classes can be found in Main_classes_*.py, the figures of bootstrap results can be found in figures of histogram for example and the code to rerun them are found in folder part_a_MSE_and_R2_with_bootstrap.

We can see in the above code that when we make an instance of the bootstrap, a list of training and test indices array get generated. Then we can use the instance to call a function to generate_the_training_data(), where we insert a data array y as an input and we receive a shuffled bootstrapped data array in return. There is also an equivalent function def generate_test_data() in the class.

5 Results analysis

5.1 Part a: Regression plots

We ran the OLS and fitted the data with 1st and 5th degree polynomials and the code seems to work properly, according to the following plots in figure 3 and 4

Confidence intervals in beta are essentially the interval $[mean(\beta) - 2 \cdot std(\beta), mean(\beta) +$

ML_classes_ridge.py

```
101
102 class bootstrap:
103     def __init__(self,n): #n the length of the array that is put in
104         self.n = n
105         self.n_test = None;
106         self.array_of_indices_training = self.generate_training_indices()
107         self.array_of_indices_test = self.generate_test_indices()
108
109     def generate_training_indices(self):
110         return np.random.random_integers(self.n,size = self.n)-1 #indices from 0 to n-1
111
112     def generate_test_indices(self):
113         test_indices =[]
114         for i in range(self.n):
115             if sum(self.array_of_indices_training==i)==0:
116                 test_indices.append(i)
117         test_indices = np.array(test_indices)
118         return test_indices
119
120     def generate_training_data(self,input_array):
121         temp = input_array.copy()
122         for i in range(self.n):
123             temp[i] = input_array[self.array_of_indices_training[i]]
124         return temp
```

Figure 2: Code example

$2 * std(\beta)]$. This interval rooms 95% of the possible beta outputs. Std is the standard deviation of the beta, which is a square root of the variance that we already discussed how to find in methods section. So to compare the different regressions and degrees of polynomials, we will resort to only look at the variances of beta's. We take a look at variance in beta of 1st degree polynomial in figure 5 and a 5th degree polynomial in figure 6

The numbers in order from top to down represent the variances in $\beta_0, \beta_1, \ldots, \beta_n$ depending on the order of the polynomial we use to fit. We see that the variances in in the 5th degree fit are bigger than the variances in the first degree. To make sure that it is not a coincidence, the reader is welcome to take a peak into the appendix, where the variances are calculated for the polynomials of order 2,3 and 4 in between and one can discern a clear pattern where the variances are increasing with the increasing degree. This pattern indicates something that we already predict, namely the fact that noise affects fitting polynomials of higher degree more than the ones of lower degree. If we have a set of given data y that we fit with $y_{pred} = f(\beta, X)$, it is clear that a small displacement in one point of y would affect/change the in y_{pred} much more in a high order polynomial, than in a low one, which is much less sensitive to noise. An example of this is overfitting, where the very complex model fits very well the training data, but very bad the test data.

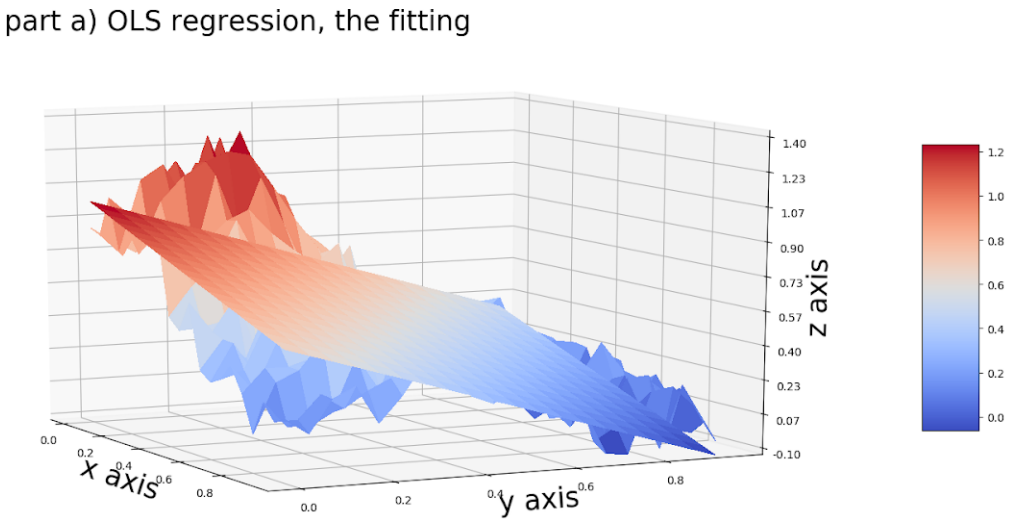

Figure 3: Illustration of a 1th degree polynomial OLS regression

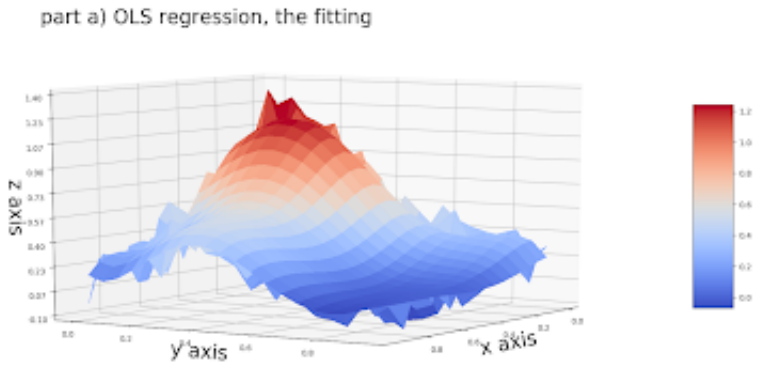

Figure 4: Illustration of a 5th degree polynomial OLS regression:

5.2 MSE and R2 score for OLS

In figures 7 and figures 8 we have calculated the MSE and R2 score for the different degrees of polynomial fittings and figure 9 is the print of the exact corresponding values. We know that R2 score is best when it's value is 1 and

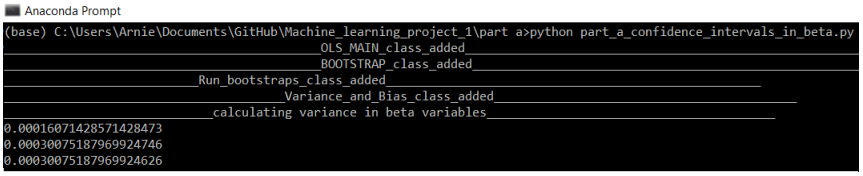

Figure 5: variances in betas for degree 1 polynomial fit

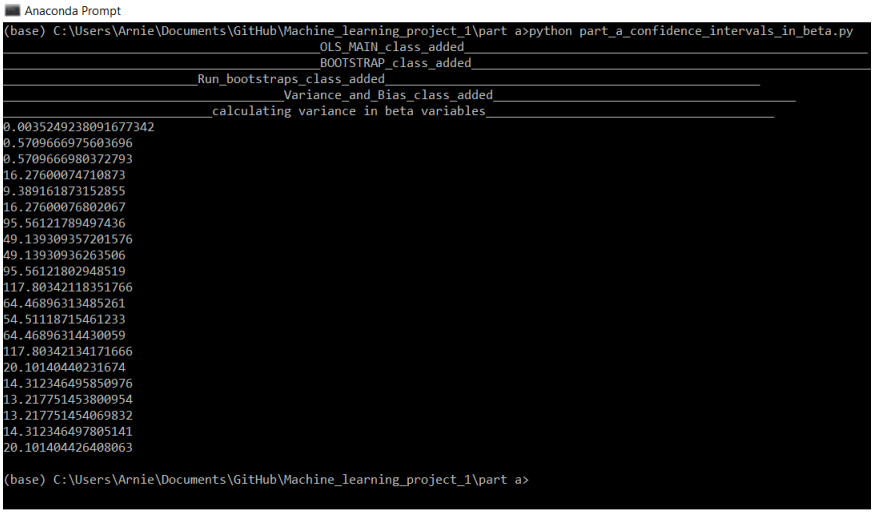

Figure 6: variances in betas for degree 5 polynomial fit

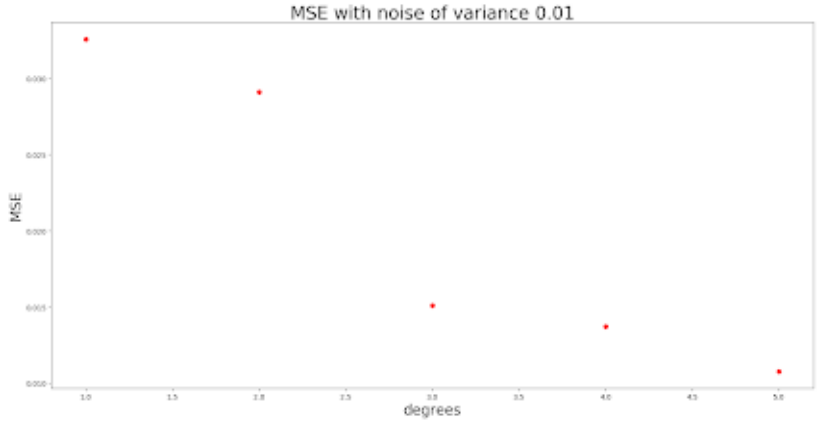

Figure 7: MSE plot of OLS from degree 1 up to 5 with noise var(e)=0.01

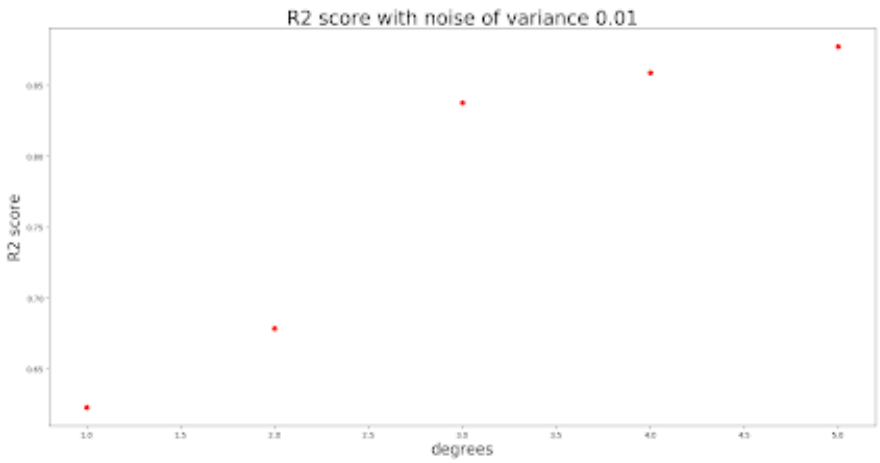

Figure 8: R2 score plot of OLS from degree 1 up to 5 with noise var(e)=0.01

```
Anaconda Prompt
(base) C:\Users\Arnie\Documents\GitHub\Machine_learning_project_1\part a>python part_a_MSE_and_R2.py
_____________________________________OLS_MAIN_class_added_____________________________________
____________________________________BOOTSTRAP_class_added____________________________________
_____________________________Run_bootstraps_class_added_____________________________
_____________________________________Variance_and_Bias_class_added_____________________________________
MSE
[0.03257885266467593, 0.02910709930330954, 0.015095247084527424, 0.013714418864685291, 0.01075584431466995]
R2 score
[0.6226640523373738, 0.6786004325892452, 0.8376139549321228, 0.8590281259122687, 0.8774261137633821]
```

Figure 9: MSE and R2 scores of OLS from degree 1 up to 5 with noise var(e)=0.01 in arrays

MSE when it's 0. We see from the figures the fitting clearly becomes better with the increasing degree.

5.3 Bootstraps

We ran 100 bootstraps and fitted betas for each, calculating MSE's for each test data and plotting them into histograms. Figures 10 and 11 show the results for 1st and 5th degree fits. In the appendix one can find the rest of the degrees. Here we also see a trend of the mean value of the histogram distribution falling around the MSE results we got in figure 9 . Another observation we can make is that the variance of the MSE shrinks also when we have higher complexity fit, this is also explained by the fact the higher polynomials provide more precision.

The same arguments apply to R2 score, which we shove to the appendix section, if the reader is curious to see.

Variance and Bias were obtained from the algorithm:

We see that the bias part shrinks, just as expected, meaning the fit is bet-

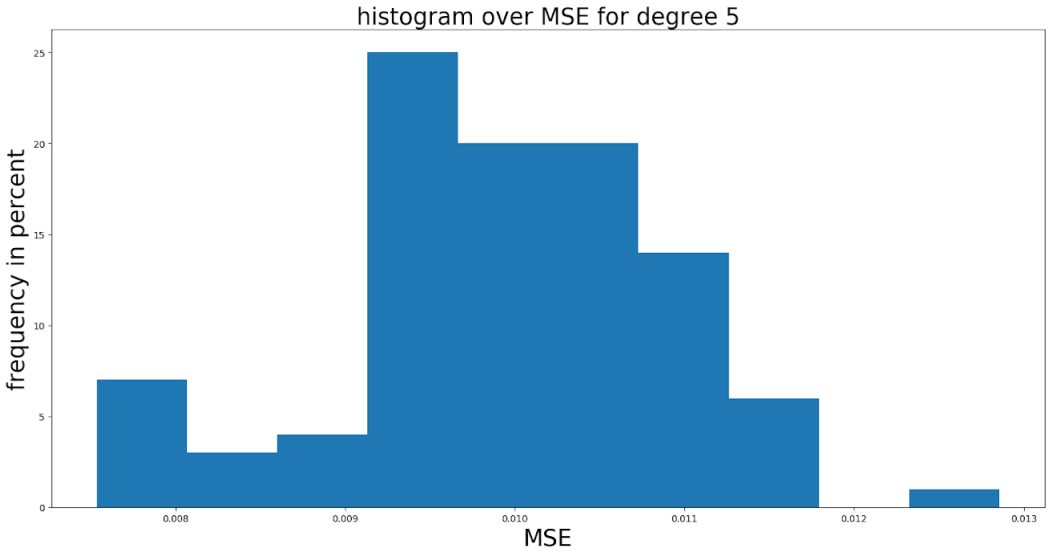

Figure 10: A histogram over the MSE data for a 5th degree polynomial fit

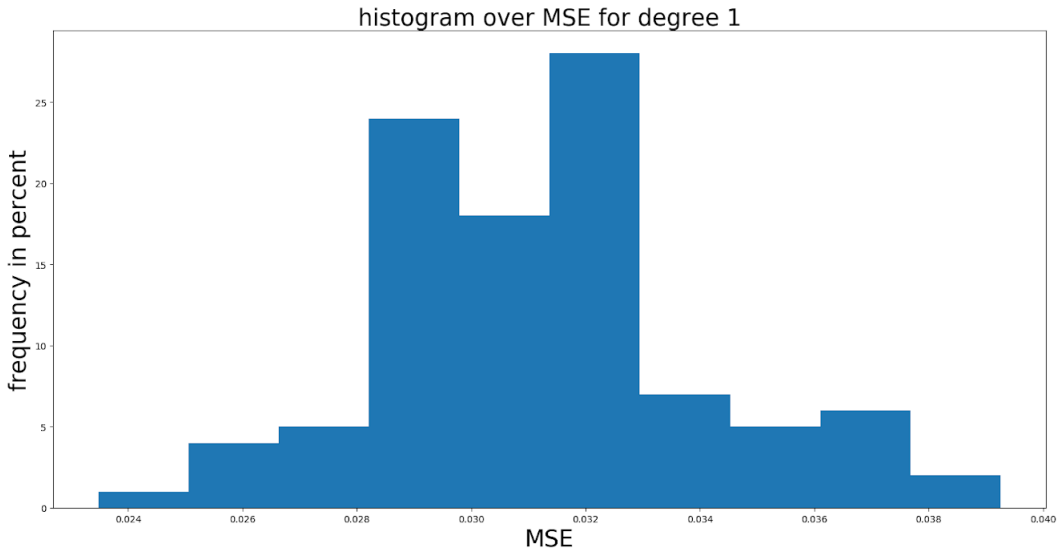

Figure 11: A histogram over the MSE data for a 5th degree polynomial fit

ter with the increasing complexity, but the variance has only a very marginal increase. It does however catch on, where the overfitting starts to increase at

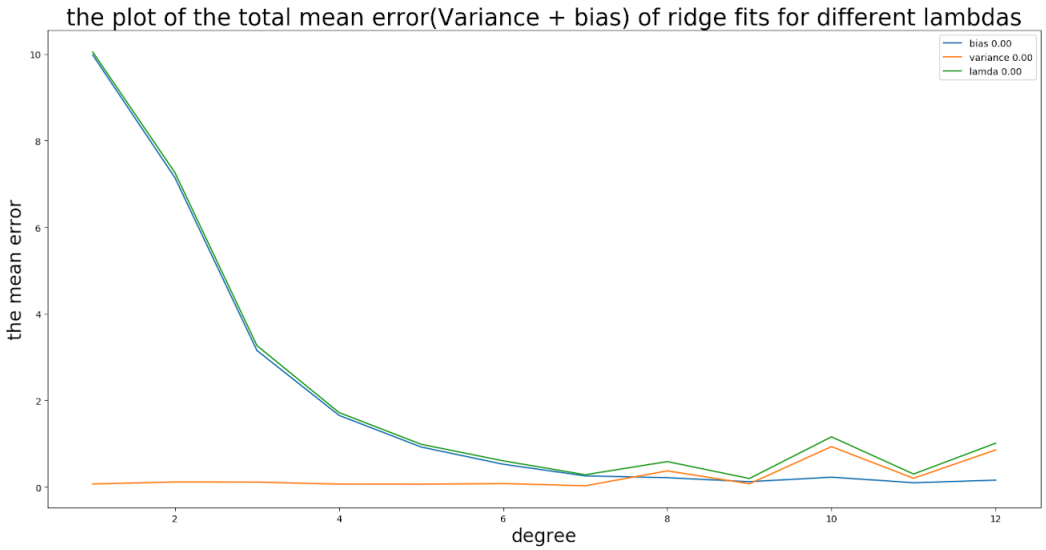

Figure 12: Variance and Bias for OLS from degree 1 up to 10

around degree 7 as we see in the yellow curve in the figure 12. We would conclude that a fit of degree 7 then is the best fit, where the sum of the bias and the variance is the least. One important detail to note, this plot of the total error, and the subsequent ones, do not actually give the full term error, because i omit adding the constant noise term. But that term is not necessary to find the best fit, since it is just constant for every complexity.

5.4 Ridge Regression

In figure 13 it can be observed that a ridge fit of a 5th degree doesn't visually give a better fit than OLS, since the beta variables here get damped with respect to each other. And we will actually see in a moment, when we calculate the MSE and R2 score, that the fits are worse.

Comparing the plots and the array figures 14 15 16 with the ones we got in figures 7 8 9 , we can see that the OLS gives better fits with less error overall.

5.5 Bootstraps with Ridge

In figure 17 only the sum of the bias and variance is plotted and we see that the plots flattens out more or less. The $\lambda = 0$ OLS one gives somewhat of the lowest point, whereas the other fits seem to be more uniform in their behavior, that likely comes from the fact that β variables are balanced out, giving an overall more evenly distributed error. The confidence intervals, or more precisely variance in betas, were calculated and shown in the figure added to the appendix.

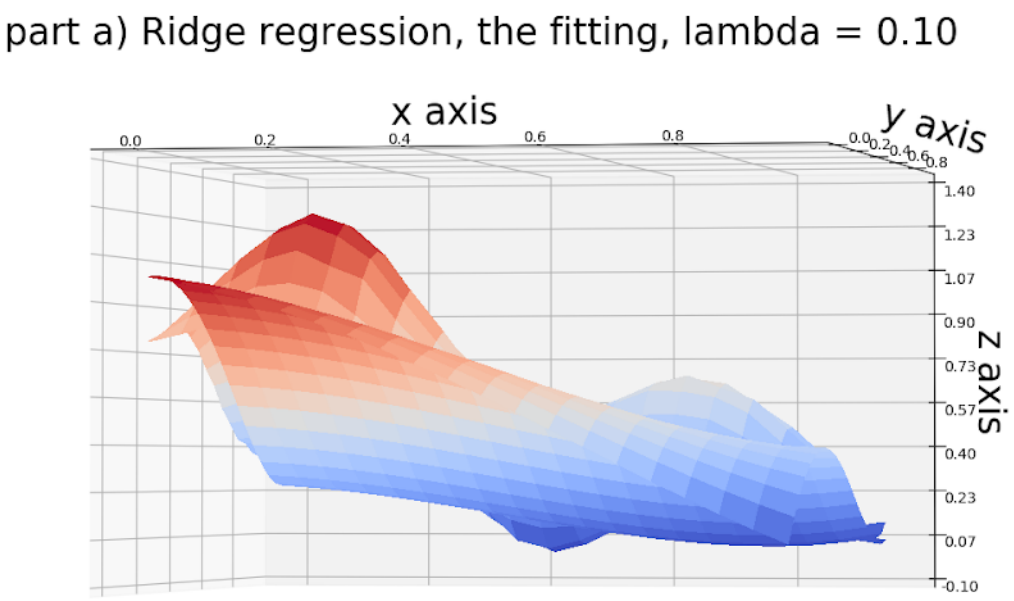

Figure 13: Ridge regression fit of 5th degree lamda=0.1

In part a, for the OLS, the order of the variances were from 10 to 10^{-3}, in the Ridge case here, they span from 10^{-5} to 10^{-6}. This indicates the fact that in the ridge case, the fits vary way less, owing to the constraint term: in OLS, higher complexity leads to overfitting, the betas vary a lot just to refit a small change in the original data, but for ridge, this small change affects the fitting less, since the added constraint constrains how much the betas can change.

6 Lasso regression

We added two figures, figures 18 and 19, to illustrate an important point, which is how Lasso regression works. It works in principle similarly as Ridge, constraining the beta variables to each other, but in rather different way. The constraint term in the cost function, which is just the absolute values of beta variables, instead of squares, gives rise to the fact that statistically, minimizing a cost function, puts most beta variables to 0 instead of just shifting them towards or away from each other. This results into the effect that we are left with only the biggest and the most significant beta values. That is why we see, from the plots, that the fits are much worse than OLS fits and the they are much more sensitive to the λ value. In figure 19 we see that Lasso set all variables to 0, except the dominant intercept term, and we thus just get a flat surface.

The plots and the printed array for the MSE and R2 score values were put

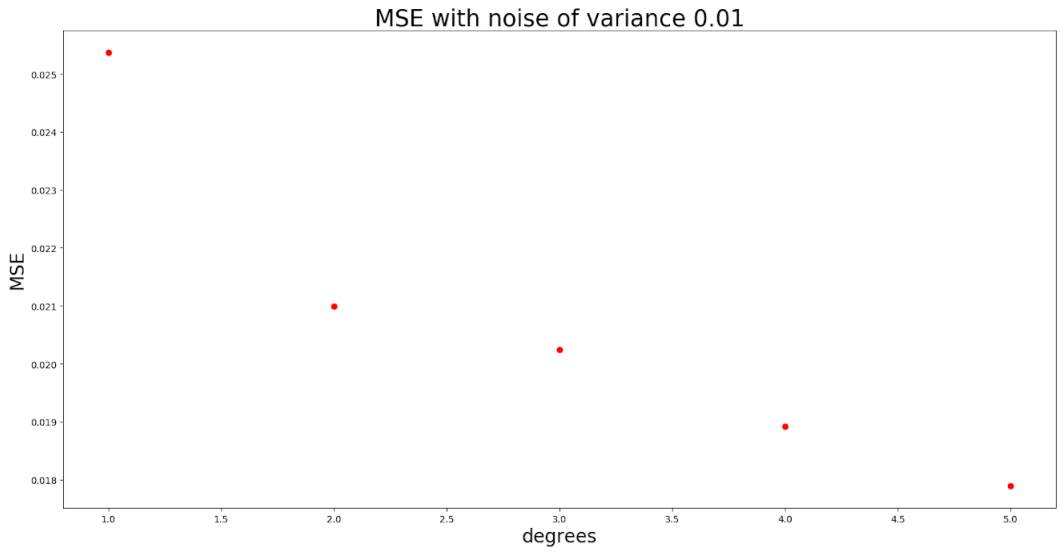

Figure 14: MSE for degrees 1 up to 5 with lambda = 2

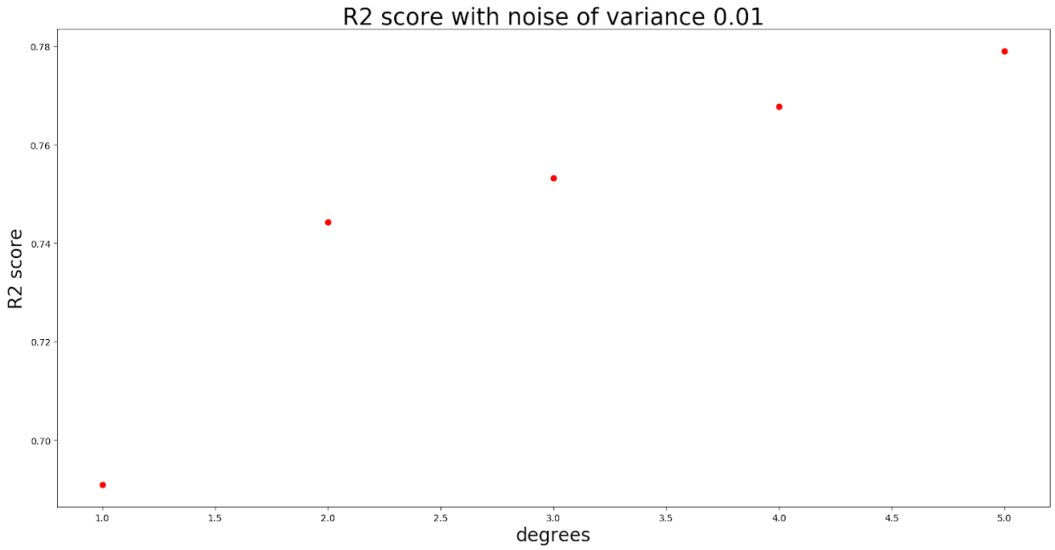

Figure 15: R2 score for degrees 1 up to 5 with lambda = 2

to the appendix. From them we can see that the error is almost non varying, staying at a consistently high error over all the different degrees of polynomial

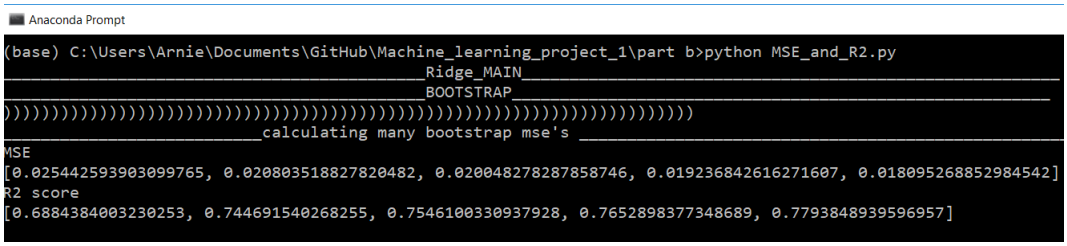

Figure 16: MSE and R2 score arrays for ridge regression deg 1 to 5 lambda =2

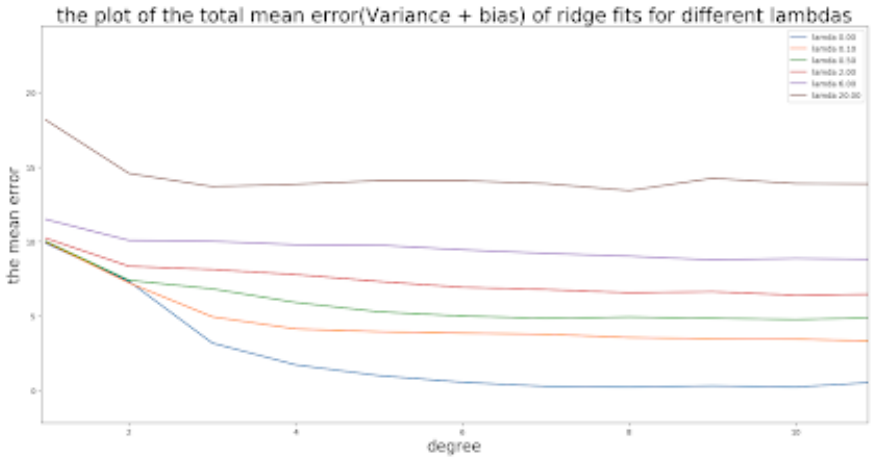

Figure 17: OLS regression, total mean error for degrees from 1 to 12 (10 bootstraps)

fit. Meaning that Lasso regression more or less eliminates the higher betas for every degree.

6.1 Bootstraps

Firstly we present the figure 20 where we plot the variance and the bias separately, to investigate the tendencies and we see that there is very little of the desired patterns, that we got with, for example, OLS. That owes to the fact that Lasso consistently eliminates beta variables, rendering different degrees giving very similar output as we can see in the figure 21.

We have also added a similar plot in figure 22, but the variance and bias separated, just to show how both variance and bias are fluctuating, but have a constant trend in the error, emphasizing our prediction, that whenever lasso is applied, most beta's are neglected, leaving us, in our case, with mostly only the intercept.

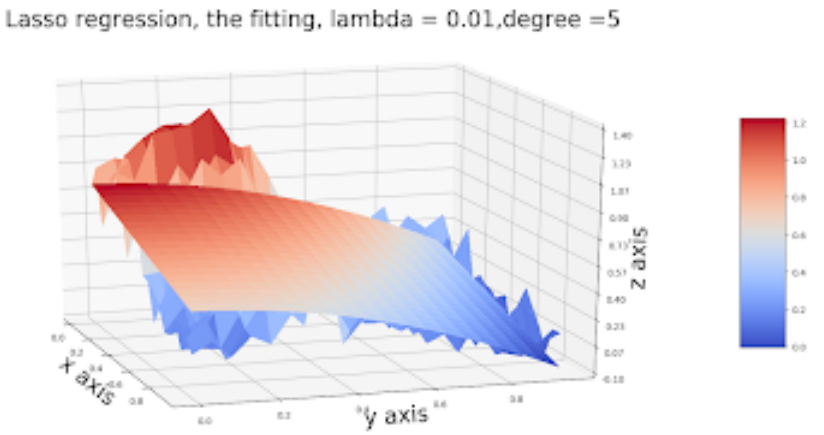

Figure 18: Lasso regression plot of degree 5 fit, $\lambda = 0.01$

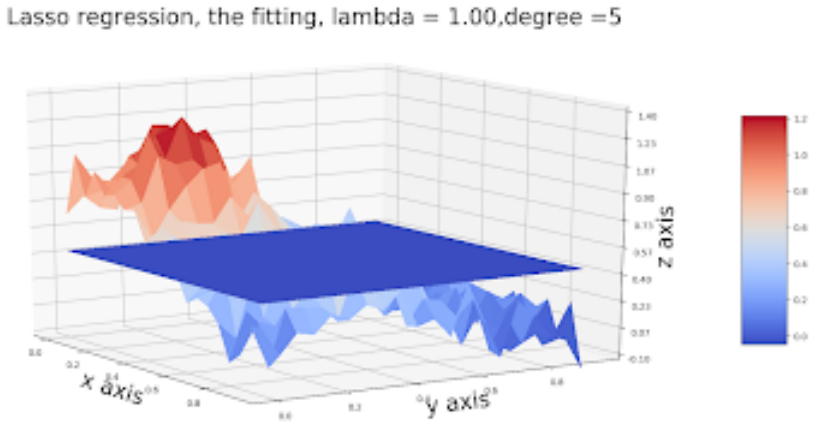

Figure 19: Lasso regression plot of degree 5 fit, $\lambda = 1$

7 Real Data

The data used was the data uploaded of some terrain near Stavanger. To avoid memory errors and lengthy run-times of the scripts, 500*500 points were extracted from the data.

Ridge regression was applied on the real data, fitted with a fifth degree polynomial, the result can be seen in figure 23. The print out of confidence intervals is for ridge lambda = 0.1 The R2score and MSE are for ridge lambda = 0.01

Looks like the R2 score gives us recognizable data to work with, it indicates

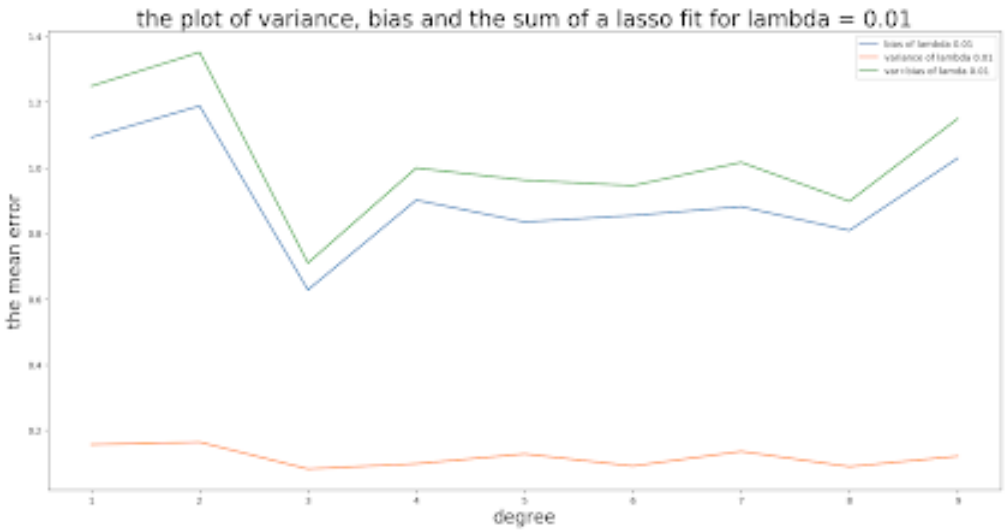

Figure 20: Lasso regression, var and bias for lambda = 0.01

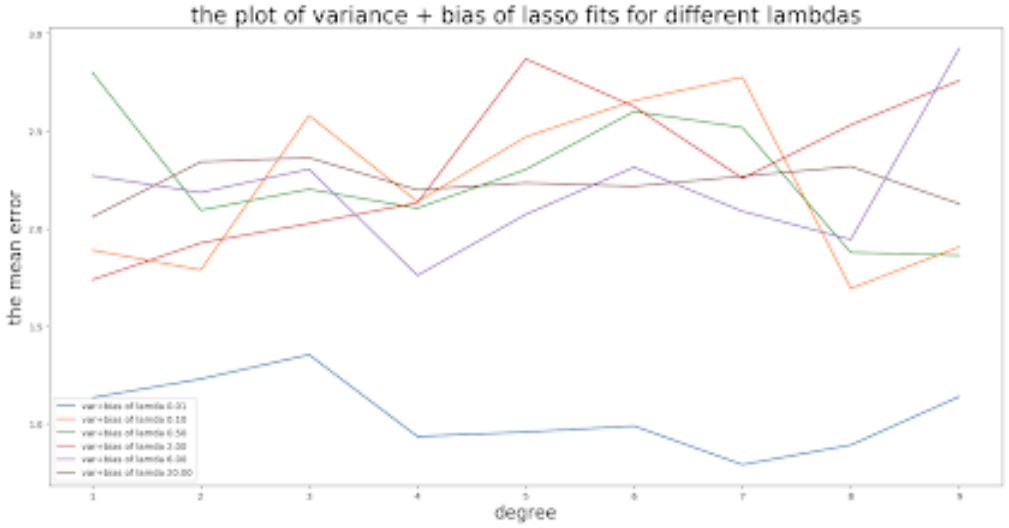

Figure 21: Sum of variance and bias for different complexities of lasso fits with different lambdas

that the particular Ridge fit from figure 23 works, but only has a value of around 0.65 at its highest, which is not the best indicator. The MSE plots, however, do seem to have the right trend, but obtained values that indicates some sort of error in the code or similar, which there has not been time to

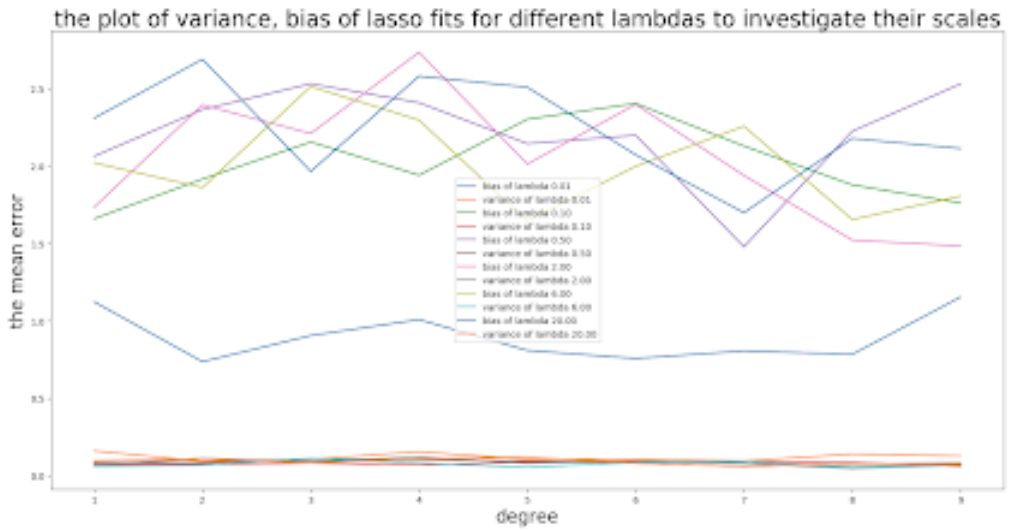

Figure 22: plot of variance and bias separately for lasso regression

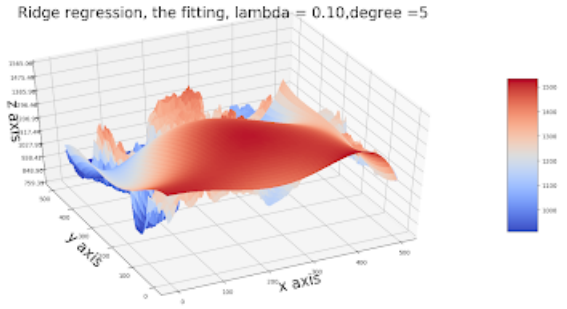

Figure 23: 5th degree polynomial fit with Ridge regression on real data.

investigate. Similarly with the figures 26, 27 and 28 that are supposed to plot the bootstrapped Ridge regressions for different lambdas, gives back values that are of the order 10^9 which leads us to conclude that there might be a problem of scaling in the code. This might mean that the calculated variances can hardly be compared with the calculated biases, although we are able to deduce a reducing trend in the bias for the increasing complexity at least.

8 Conclusion

In this project three regression methods were used to approximate the Franke function. Looking at the fitting plots it was observed that the regularized re-

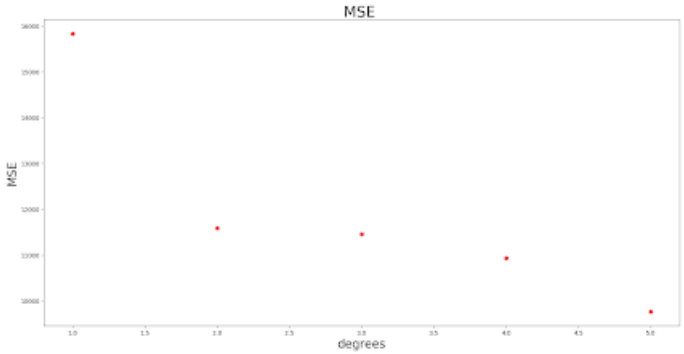

Figure 24: R2 score of real data for different degrees

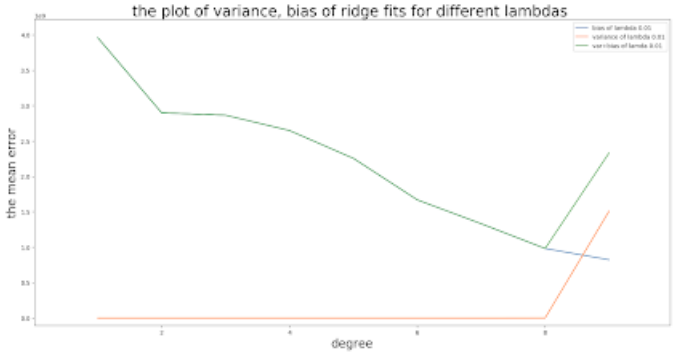

Figure 25: MSE on real data for different degrees

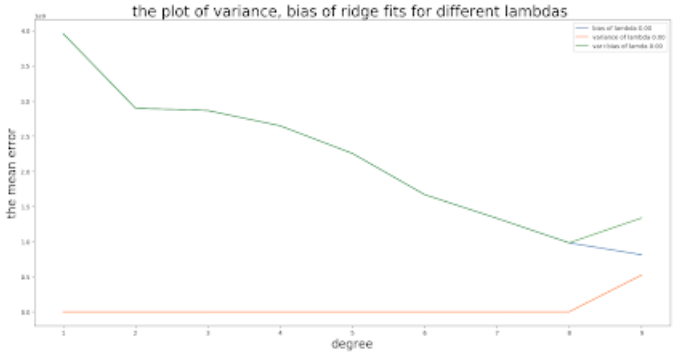

Figure 26: plot of variance and bias separately for lasso regression

gressions tend to give less varying (more even) prediction models.

In general it was observed that the higher the alpha values the flatter the

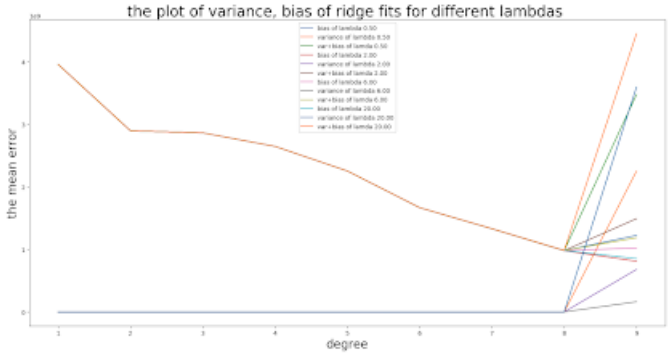

Figure 27: Ridge bootstrap =10 for different lambdas:

fitting. As it was expected the Lasso model rendered most beta's to zero, leaving only the dominant ones. The OLS was obviously the best model to fit the data. The question might arise as to what would be the point with using anything else than the OLS method in any case, we have already shown mathematically that the OLS is an actual projection of the real data to the subspace of whatever polynomial space we are fitting on. That means that the added constraint terms can only be less precise and give more errors, just as we actually got in our case. The answer is that it actually doesn't make any sense to use Ridge or Lasso regressions in the case with only one input variable predictor or where the input variables are completely uncorrelated like our case. In our case the input variables are just x and y coordinates of a terrain, there is no correlation between them. The point with these regressions with constraints on beta's were to approach cases where we actually do have correlations in input variables. If there is a correlation, then an increase in one beta variable would mean that an increase in another, correlated beta variable would need to be accounted for. Then adding a constraint which would take care of such an in balance by resizing betas with respect to each other makes sense. When it comes to the confidence intervals, we do get the behaviour we would expect, where: The confidence intervals for betas become bigger, for higher complexity to account for more unstable fits. The confidence intervals for betas become smaller for added constraints, since the beta's are more stable when coupled to each other with ridge or lasso. We also demonstrated a well functioning resampling bootstrap method that simulated the act of reiterating the experiment so that we could make a distribution of MSE's with histograms, that simulated experiments in a satisfactory fashion, providing the mean MSE, values that are the best fits on the test data and the variance, indicating how much the different fits in different bootstraps vary.

Lastly, we took the expectation of the mean squared error of the different models over two different random variables: the noise term and the different bootstrap fits. We can split that into three terms, the noise variance, the bias

squared and the variance in the fitting data. The noise variance is a constant term(for the real data it is assumed), so we disregard it in our analysis, as it is not relevant for what we are looking for. The Bias term gives the magnitude of how much the real actual data model, stripped of noise is deviating away from the average best fit that we get from the resamplings. Then the variance in the fitting data tells us how much the different fittings of the different bootstraps deviates from the average fitting we found. The two last terms are the interesting ones. The bias would be high for small degrees and then drop as the complexity rises, because the data points get fitted better, with higher degrees. The variance, however, has an opposite trend: the data changes give very little difference in the small degree fits compared to when we increase the degrees, because the overfitting gives more varying fits. This means that the sum of them has to have a minimum point, a complexity of the model that gives a fairly good fit to the real model(the bias) and at the same time is not too complex to give too much overfitting. In our case, since the OLS gave the best fit, and the only one that makes sense to use, we could see from the bootstrap plot that the polynomial of degree 7 was the one to give the best bias-variance trade-off or, in other words, the best fit for the simulated Franke function. For the case of real data this is not very clear. The plot suggests the degree 8 as the minimum point, but since the results are very unclear and most likely unusable, mostly due to time constraints to do it properly, it is dangerous to infer something from these results.

9 Appendix

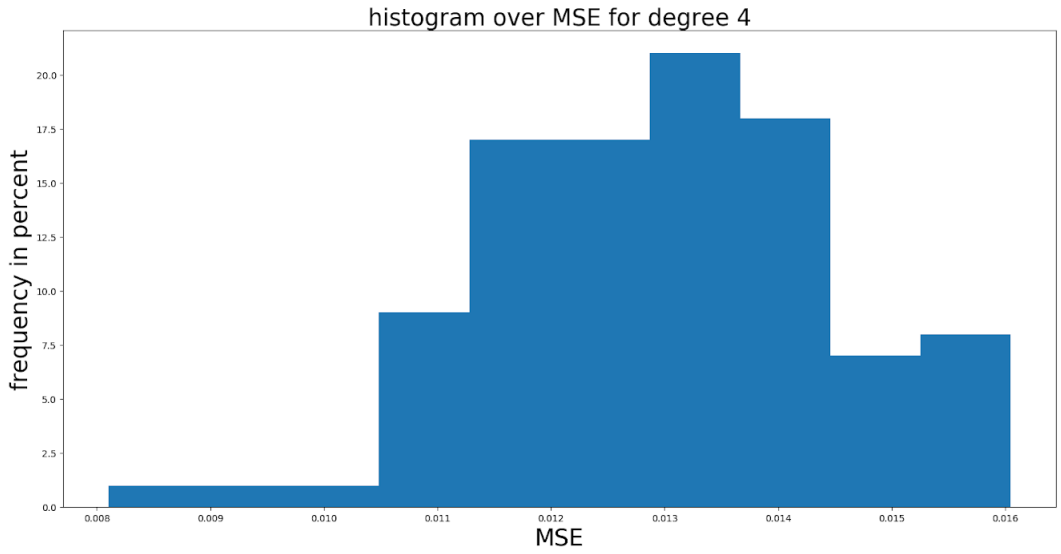

Figure 28: part a

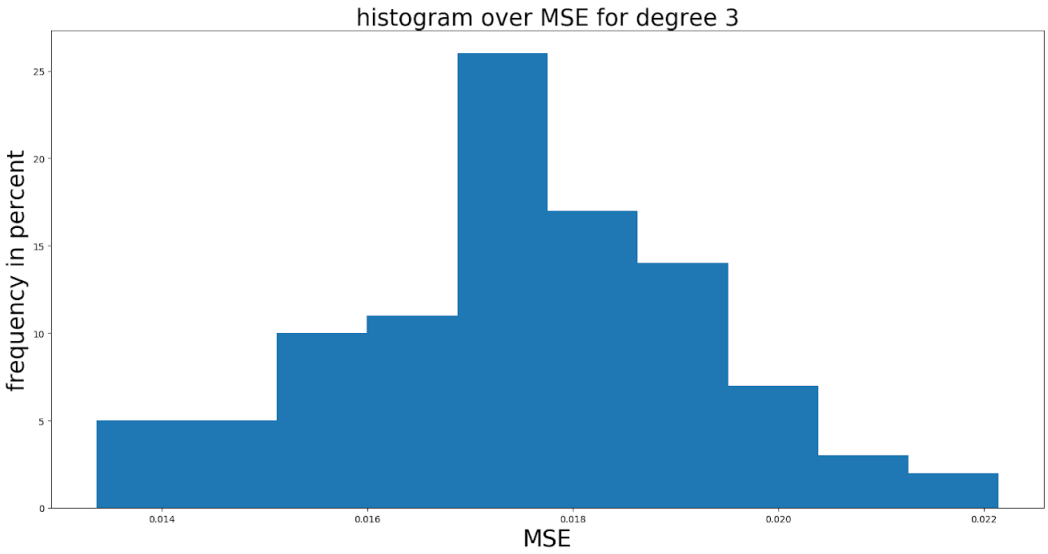

Figure 29: part a

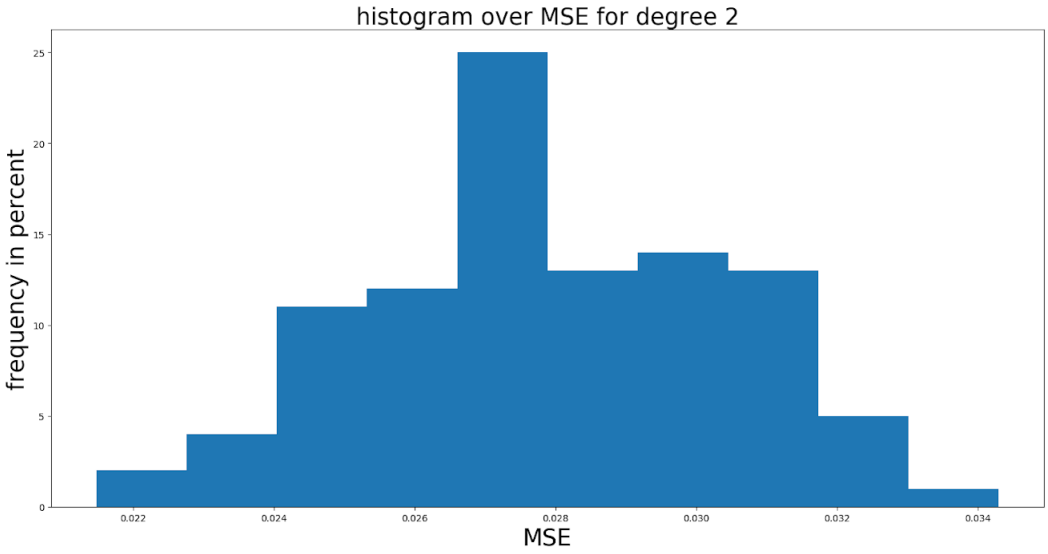

Figure 30: part a

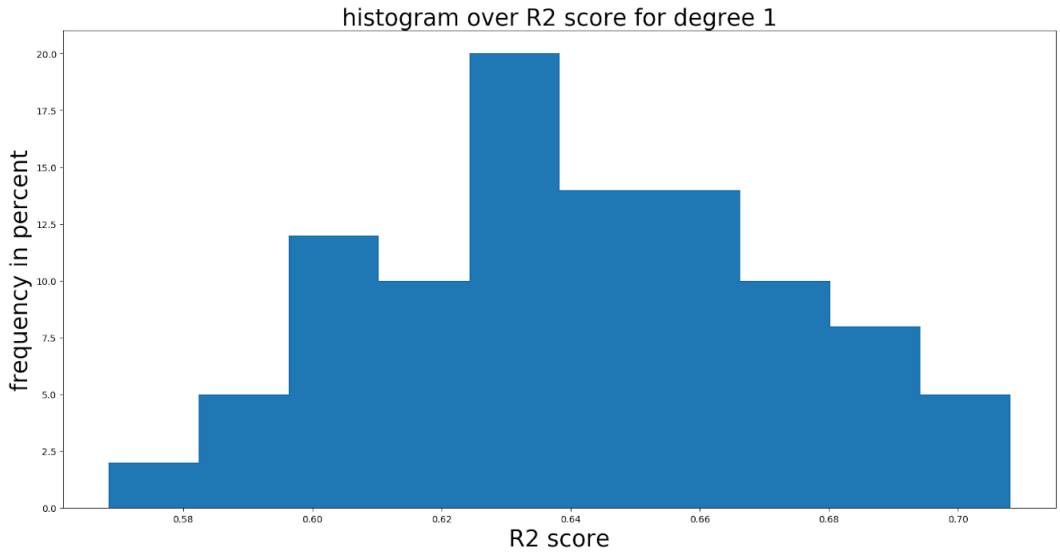

Figure 31: part a

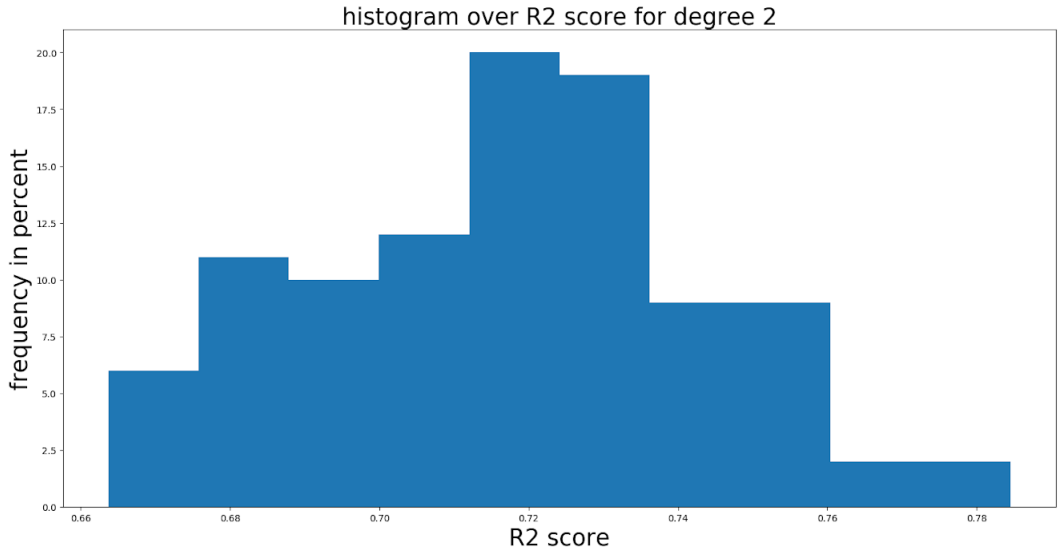

Figure 32: part a

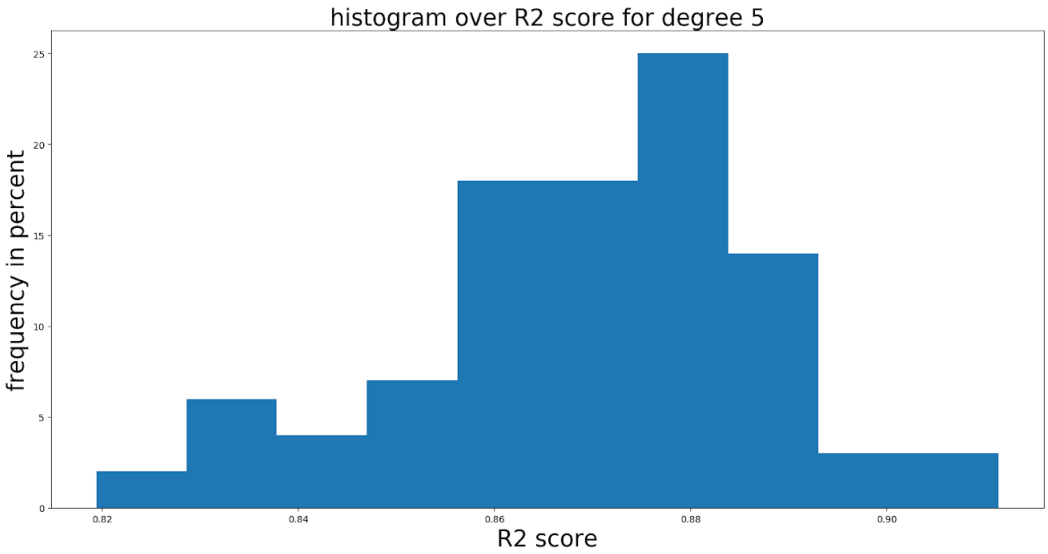

Figure 33: part a

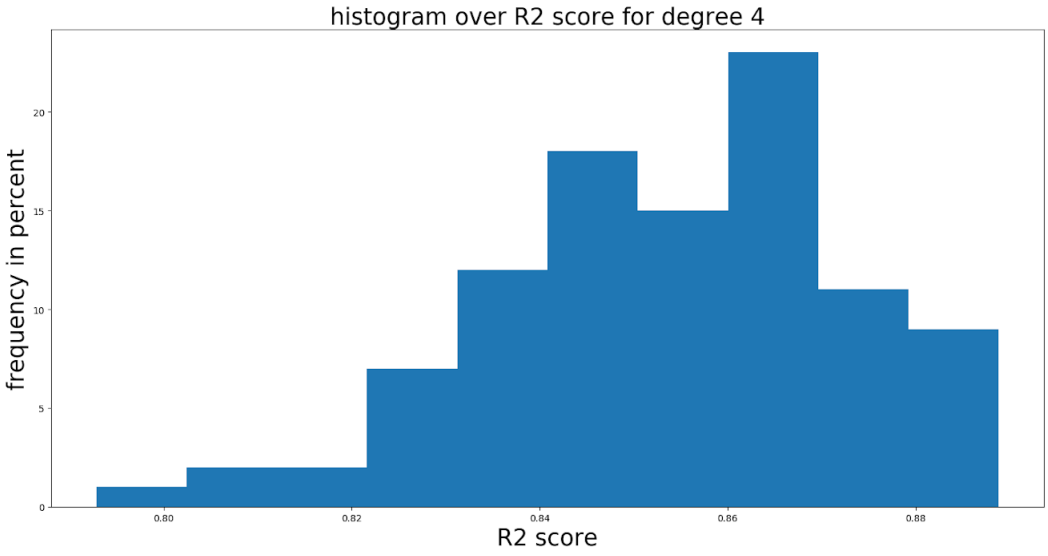

Figure 34: part a

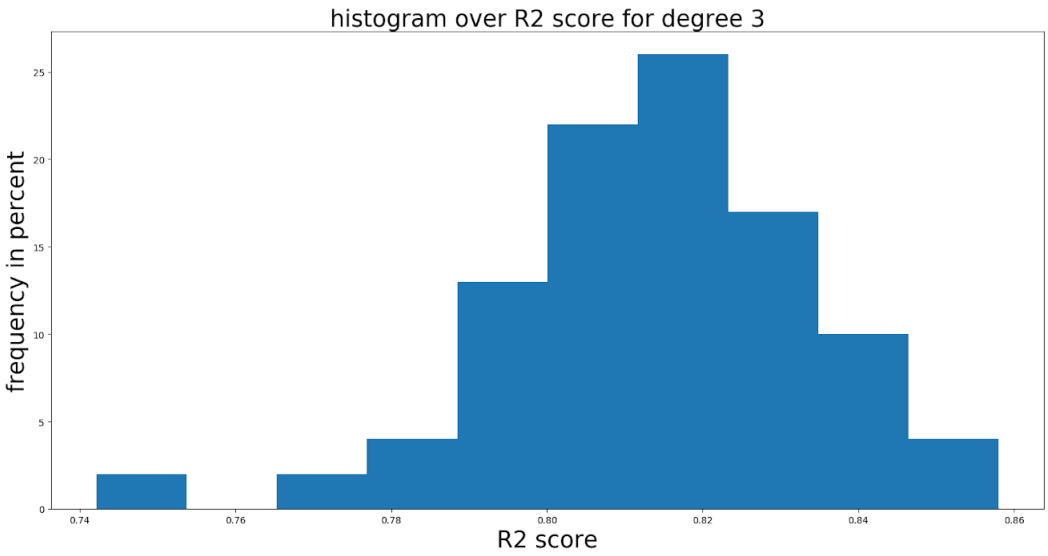

Figure 35: part a

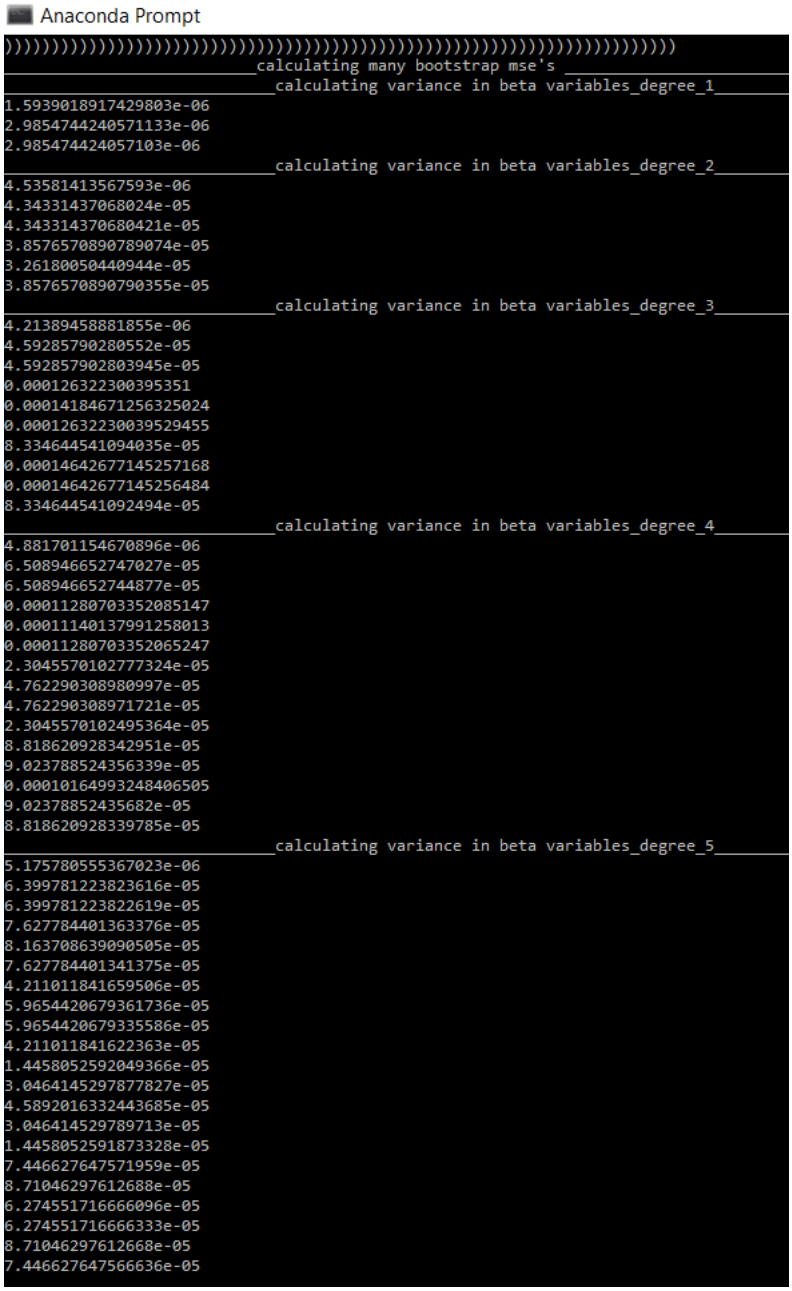

Figure 36: part b, confidence intervals in beta with ridge lambda = 0.1

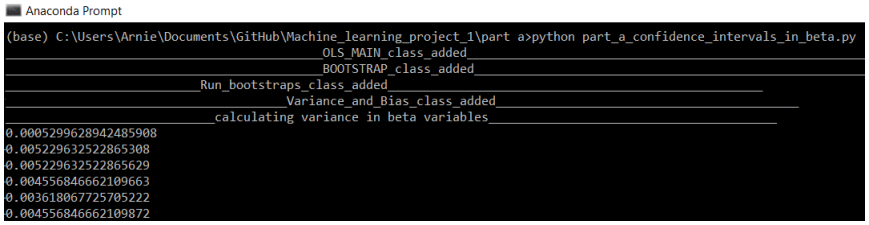

Figure 37: part b variance in beta 2th degree polynomial fit

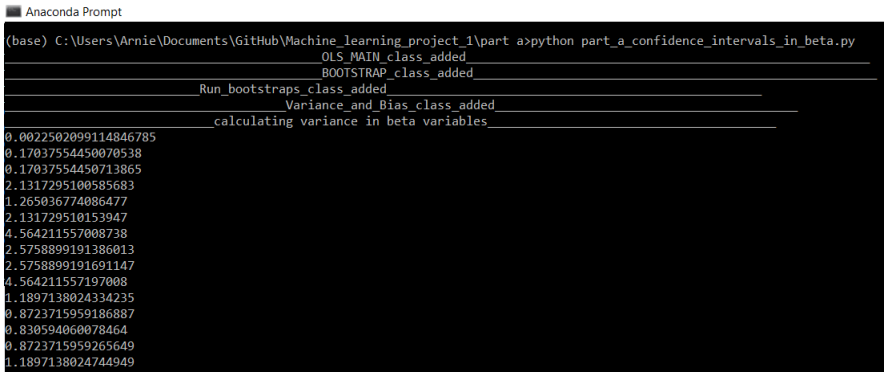

Figure 38: part b variance in beta 4th degree polynomial fit

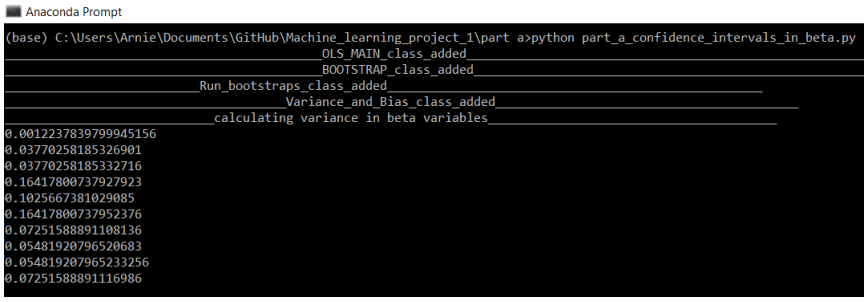

Figure 39: part b variance in beta 3th degree polynomial fit

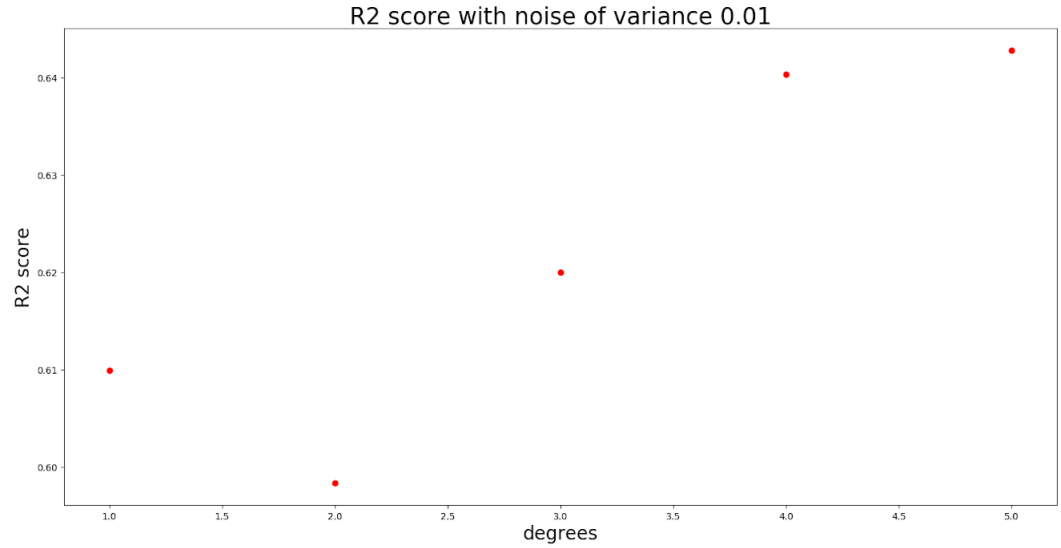

Figure 40: part c R2 score with noise of variance 0.01 for lasso

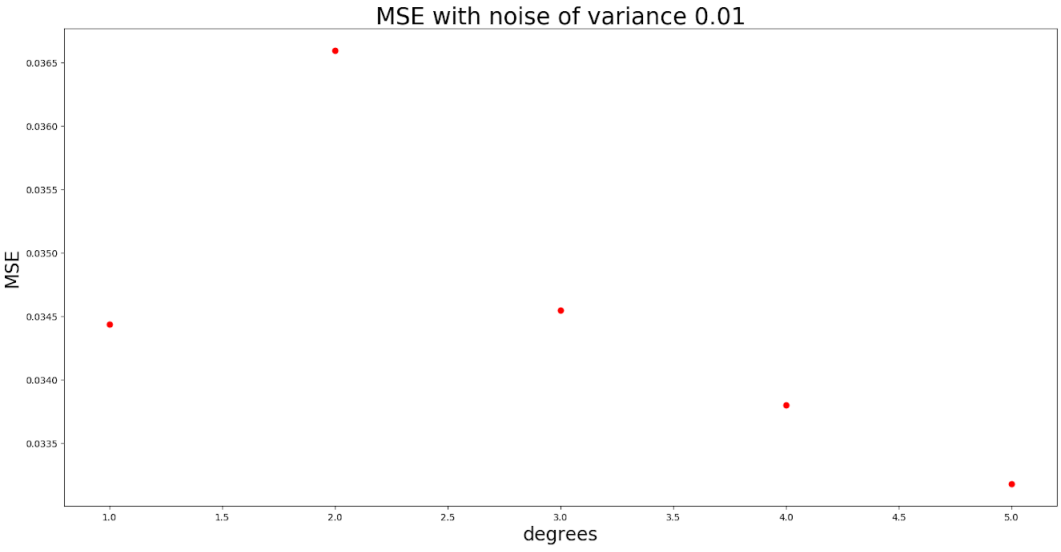

Figure 41: part c MSE with noise of variance 0.01 for lasso

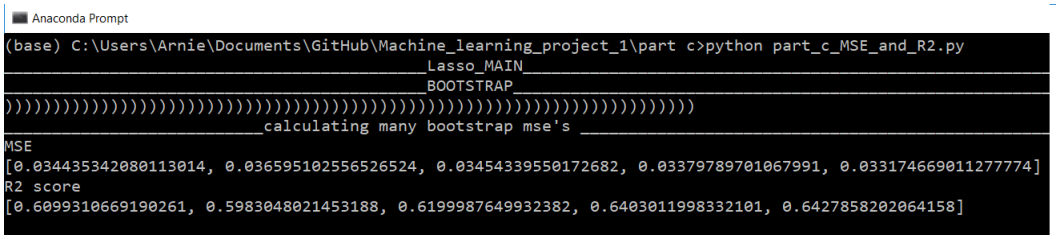

Figure 42: part c MSE and R2 score with noise of variance 0.01 for lasso

[2] [1]

References

[1] by Wessel N. van Wieringen https://arxiv.org/pdf/1509.09169.pdf page 10 (for the variance in beta). *Lecture notes on ridge regression.*

[2] the website from which the terrain data was drawn. *https: //earthexplorer.usgs.gov/.*