FYS-STK4155
Data Analysis and Machine Learning

Project 1

Regression analysis and resampling methods

Kristina Othelia L. Olsen

October 9, 2019

Abstract

In this project we have studied the OLS, Ridge and Lasso regression methods to make estimations about generated data from Franke's function, and real terrain data from NASA JPL. This is done by fitting a 2D polynomial of various degree to the data sets. We implemented a resampling technique, k-fold cross validation, to get more accurate results for the regression analysis; Mean square error, R^2 score and bias-variance tradeoff. The regression methods are implemented in Python, with help from Scikit Learn for Lasso regression. The clear conclusion of this report is that OLS and Ridge regression were the best models for our data sets. This was determined by looking at the MSE and R^2 scores, as well as comparing 3D images of the model vs the actual function/terrain. The reason Lasso regression predicted bad models, could be caused by numerical problems in the resampling method, or the wrong hyperparameter λ estimations. For Franke's function we found that Ridge regression with polynomial degree 7 and $\lambda \approx 0.0015$ gave the best fit, and OLS with polynomial degree 8 gave the best fit for the real terrain data.

Contents

1 Introduction

Linear regression is a highly important subject in statistics and basic machine learning, and machine learning is a big part of todays industry and science. In this report, a part of the course FYS-STK4155 - Applied data analysis and machine learning at the University of Oslo (UiO), we are going to study the use of regression methods. The main pupose is to fit a 2d polynomial of various degrees to a well-studied function, Franke's function. Then we are going to use the same methods on a real data set, of a crater on Mars. We are also applying a resampling technique and regression analysis. In other words, we want to use linear regression and resampling to model a real digital terrain.

Regression is statistical processes aimed to estimate the relationship between variables, mainly between the dependent variable and one or more independent variables. These are also called "predictors". The study compares various regression methods, Ordinary Least Squares (OLS), Ridge and Lasso regression. These regression methods will be combined with the resampling technique, k-fold Cross Validation. The accuracy of the computed model will be estimated using Mean Squared Error (MSE), R^2 score and Bias-Variance tradeoff.

The report structure starts with giving an overview of the needed theory. Here we'll look at the regression methods, regression analysis and the k-fold cross validation. The Method chapter gives an quick overlook of how the theory is implemented, and presents the two data sets. The Result part of the report is divided in two sections, for each data set. Here we'll present and discuss the results from the different numerical implementations done in our Python code. In the end we have a short conclusion of the project. In addition, there are appendixes that contains various mathematical calculations, and results.

The main source of information used for writing this report comes from the lecture notes of the course [1]. All materials, including code, results and the terrain data set, used for writing this report can be found at: https://github.com/kristinaothelia/FYS-STK4155/tree/master/Project1.

2 Theory

2.1 Regression Methods

2.1.1 Linear regression

Regression is a statistical measurement used to determine the strength of the relationship between the dependent variable $\hat{z}$ and one or more independent variables $\hat{x}$. In linear regression, we are using a linear approach to model the relationship between the dependent and the independent variables. If we have a set of scattered data point, we are trying to draw a straight line between the points [2].

Consider a data set $\boldsymbol{z} = \{z_0, z_1, ..., z_{n-1}\}$ and $\boldsymbol{x} = \{x_0, x_1, ..., x_{n-1}\}$ with n data points. It's then assumed that the relationship between them is given by the function $\boldsymbol{z} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$, where $f(\boldsymbol{x}) = \tilde{\boldsymbol{z}}$. $\boldsymbol{\varepsilon}$ is the error term, or noise in the data, which is normally distributed with mean zero and standard deviation σ^2. Our model $\tilde{\boldsymbol{z}}$ is expressed as eq. (3) in linear regression, where $\boldsymbol{X}$ is the design matrix and $\boldsymbol{\beta}$ is the regression parameters.

$$\tilde{\boldsymbol{z}} = \boldsymbol{X}\boldsymbol{\beta} \tag{1}$$

This model if the prediction of $\boldsymbol{z}$. The algorithm for the design matrix were given by Morten Hjorth-Jensen, and can be found in Appendix A. The algorithm creates a design matrix with rows $[1, x, y, x^2, xy, y^2, \text{ etc}]$, where $\boldsymbol{X}$ has shape $[nxl]$. n is the polynomial degree we want to fit, and l is the number of elements measured in n (number of elemnts in $\boldsymbol{\beta}$). For a 5th order polynomial degree, the model consists of 21 β-parameters.

$$\begin{aligned}\tilde{z}(x,y)_{n=5} = \beta_0 + &\\ &\beta_1 x + \beta_2 y + \\ &\beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \\ &\beta_6 x^3 + \beta_7 x^2 y + \beta_8 x y^2 + \beta_9 y^3 + \\ &\beta_{10} x^4 + \beta_{11} x^3 y + \beta_{12} x^2 y^2 + \beta_{13} x y^3 + \beta_{14} y^4 + \\ &\beta_{15} x^5 + \beta_{16} x^4 y + \beta_{17} x^3 y^2 + \beta_{18} x^2 y^3 + \beta_{19} x y^4 + \beta_{20} y^5\end{aligned} \tag{2}$$

The model including noise are as follows

$$\tilde{\boldsymbol{z}} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{3}$$

2.1.2 Ordinary Least Squares, OLS

OLS is a method we can use to determine the unknown parameters in a linear regression model. We can find the optimal β parameter by using OLS to minimize the sum of the squares of the difference between z_i and $\tilde{z}_i$. The so-called cost function, matrix form, is given by

$$\begin{aligned}C(\mathbf{X}, \boldsymbol{\beta})_{\text{OLS}} &= \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 \\ &= \frac{1}{n}\left[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^T(\boldsymbol{z} - \tilde{\boldsymbol{z}})\right] = \frac{1}{n}\left[(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})\right]\end{aligned} \tag{4}$$

If we now use that $\frac{\partial[C(\mathbf{X},\boldsymbol{\beta})]}{\partial\boldsymbol{\beta}} = 0$, we find that the expression for the β-function for OLS is given by

$$\boldsymbol{\beta}^{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{z} \tag{5}$$

2.1.3 Ridge regression

Ridge regression, or Tikhonov regularization, is a helpful method for models with a large number of β-parameters. It's also good to use when multicollinearity occurs in $\boldsymbol{X}$ (high correlation between predictor variables). When this occurs using OLS, the variance becomes high, and the error grows. Ridge regression reduces this error with the positive constant *hyperparameter* λ. λ is the Lagrange multiplier, and help shift the diagonals in the identity matrix $\boldsymbol{I}$. With the reduces error, the overfitting possibility also increases [3]. The cost function is given by

$$\begin{aligned} C(\mathbf{X}, \boldsymbol{\beta})_{\text{Ridge}} &= \frac{1}{n} \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 + \lambda \sum_{i=0}^{p-1} \beta_i \\ &= \frac{1}{n} \left[(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta}) \right] + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}, \end{aligned} \tag{6}$$

which gives

$$\boldsymbol{\beta}^{Ridge} = (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{z} \tag{7}$$

This method is more efficient in parameter estimating, for the cost, or L2 penalty, of more bias, see Bias-Variance tradeoff.

2.1.4 Lasso regression

LASSO, Least Absolute Shrinkage and Selection Operator, regression is similar to Ridge regression, and is used for multicollinearity in $\boldsymbol{X}$. Lasso uses shrinkage, which is when data values are shrunk to a central point. The L1 penalty constant λ is now the amount og shrinkage.

$$C(\mathbf{X}, \boldsymbol{\beta})_{\text{Lasso}} = \frac{1}{n} \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 + \lambda \sum_{i=0}^{p-1} |\beta_i| \tag{8}$$

When $\lambda = 0$ no parameters are eliminated. When λ increases, an increasing amount of parameters are set to zero and eliminated. This also means that the bias is increased.

Unlike OLS and Ridge, Lasso has no explicit expression for $\boldsymbol{\beta}$. Instead we have to use methods like gradient descent to determine the β-values to minimize the cost function. In this project we have used the Lasso function from Scikit Learn [4]. This function also optimizes the learning rate.

2.2 Regression analysis

2.2.1 Bias-Variance tradeoff

To find the regression parameter β, we have to optimize the MSE by the so-called cost function

$$C(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 = \mathbb{E}[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2] \tag{9}$$

This can be written as

$$\mathbb{E}[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{z}])^2 + \frac{1}{n}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{z}])^2 + \sigma^2 \tag{10}$$

The derivation of the bias-variance decomposition can be found in Appendix A. Note that

$$\mathbb{E}[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2] = \text{Bias}^2 + \text{Var} + \sigma^2$$

so that the cost function becomes

$$\begin{aligned} C(\boldsymbol{X}, \boldsymbol{\beta}) &= \sigma^2 + (f - \mathbb{E}[\tilde{\boldsymbol{z}}])^2 + \mathbb{E}[(\tilde{\boldsymbol{z}} - \mathbb{E}[\tilde{\boldsymbol{z}}])^2] \\ &= \text{Bias}^2 + \text{Var} + \sigma^2. \end{aligned}$$

The cost function is dependent on the balance between the bias, variance and the error in the data, noise. We can see that reducing the bias will increase the variance in the model, and vice versa. The bias is the difference between the correct value z we are trying to predict, and the average prediction of the model $\bar{\tilde{z}}$. The variance is the difference between a point in the model and the average of the point. Models with high bias and low variance leads to overfitting, and low bias and high variance leads to overfitting. Therefor it's important to take into account the bias-variance tradeoff. In a model with many β-parameters, the regression methods will start to fit the function to the training data, and the noise found in the data set. This leads to overfitting. There are methods to avoid overfitting. One, is to split the data into training and test data, as we do in this project when we implement k-fold CV. By using training-data to train the model with a regression method, we avoid the model to fit to the data-noise in the test-data. This leads to a more accurate error when testing. Another method is to avoid to large polynomial degrees. With higher degrees, there is a larger chance for the model to fit to small surface irregularities caused by noise.

A balance between the bias and the variance minimizes the total error in the model, giving a better fit. This is clearly visualized in figure 1, where we can see that the best fit lies between the dotted areas. We define the bias squared and the variance as follows:

$$\text{Bias}^2 = \sum_{i=0}^{n-1}(z_i - \bar{\tilde{z}}_i)^2 \tag{11}$$

$$\text{Variance} = \sum_{i=0}^{n-1}(\tilde{z}_i - \bar{\tilde{z}}_i)^2 \tag{12}$$

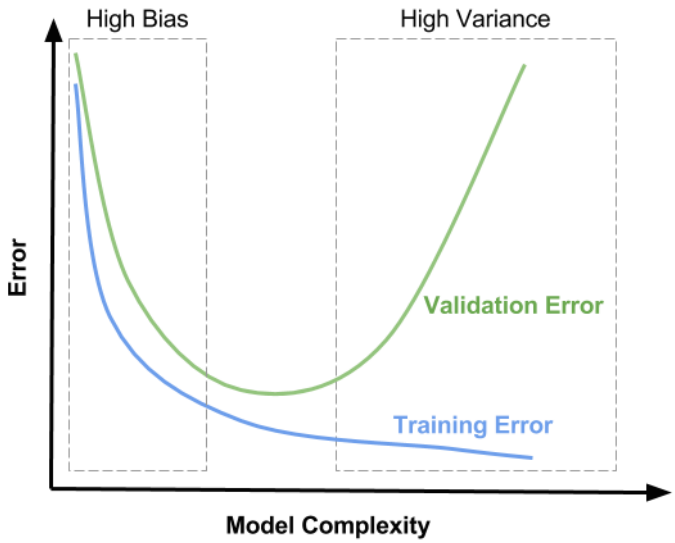


Figure 1: Bias-variance tradeoff. Predicted error for test and training data as a function of model complexity (polynomial degree). Image taken from https://www.learnopencv.com/wp-content/uploads/2017/02/Bias-Variance-Tradeoff-In-Machine-Learning-1.png

2.2.2 Mean Squared Error & R^2

In order to evaluate how good our model fits the data, we have to consider the mean squared error (MSE) and R^2 score.

The MSE function is given by:

$$\text{MSE}(z, \tilde{z}) = \frac{1}{n} \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 \tag{13}$$

The MSE gives an average value for the difference between an known data point and the predicted data point. The closer the result is to zero, the better the model.

The statistical coefficient of determination, R^2, score is used for indicating how well the model is predicting the actual values in the data set. The closer the R^2 score is to 1, the better the model. This is because it tells us the proportion of the variance in the dependent variable which is predicted by the independent variable. With other words, $R^2 = 1$ means that the model is a 100% fit to the original data set.

The R^2 score function is given by:

$$R^2(z, \tilde{z}) = 1 - \frac{\sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1} (z_i - \bar{z}_i)^2} \tag{14}$$

where the mean value of z is defined as

$$\bar{z} = \frac{1}{n} \sum_{i=0}^{n-1} z_i \tag{15}$$

2.2.3 Confidence interval of β

Since we are working with statistics, we want to find the confidence interval (CI) of the β parameters. Meaning we want to find the uncertainties of the estimated β parameters. A 95% CI range will then have a 95% probability for containing the β-parameter. The confidence interval is found as follows

$$CI_{\beta i} = (\beta_i - Z_{score}\sigma_{\beta_i}, \beta_i + Z_{score}\sigma_{\beta_i}) \tag{16}$$

where $Z_{score} = 1.96$ for the 95% CI. σ_{β_i} is the standard deviation of the given β parameter. The standard deviation is expressed as

$$\begin{aligned} \sigma_{\boldsymbol{\beta}} &= \sqrt{Var(\boldsymbol{\beta})} \\ Var(\boldsymbol{\beta}) &= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\sigma^2 \\ \sigma^2 &= \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{n-p-1} \end{aligned}$$

2.3 Resampling Methods

In machine learning we often work with small data set, which makes it difficult to know which parameters, polynomial degree and hyperparameter, to us for finding the best model. By using resampling techniques we perform the same calculations multiple times on random samples from the data set, training the model and gaining information we would have lost if we only fitted the model one time.

We chose the k-fold cross validation resampling method for this project. The other alternative were the Bootstrap resampling method, which is a bit easier to implement in a Python code. The main difference is that the k-fold methods splits the data set into test and training data, and if implemented right, should give more realistic results. With bootstrap we execute the resampling with the whole data set each time.

It is important to shuffle the data before performing a resampling technique. This is due to the possibility of the data set being composed in a specific way, and may include distinctive patterns.

2.3.1 k-fold cross validation

As we stated above, the k-fold cross validation (CV) splits the data set in training and test data. In figure 2, we see an ex. of a k-fold of order 5. The procedure is to use one fold as the test data, and the rest as training data. We then fit the model to the training data, and then use the model to make predictions on the test data. The cross validation is then repeated k-times, where the test data always is a new part of the data set. When the resampling has run through all k-fold, we estimate the average performance of the model (test data only). It has been shown that using a k between 5 and 10, gives acceptable bias and variance estimates [5].

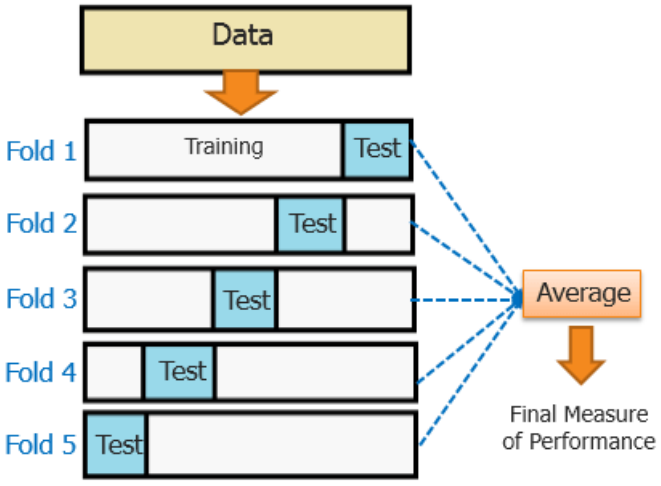


Figure 2: The basic structure of k-fold cross validation. It's normal to use k between 5 and 10, depending on the size of the data set. https://blog.contactsunny.com/data-science/different-types-of-validations-in-machine-learning-cross-validation

3 Method

Part I

Part I of the project were considered for our own understanding and familiarisation with regression. This part looked at OLS, with and without resampling, for a 5th order polynomial degree. We explored the use of normal stochastic noise added to the randomly generated data points used for Franke's function. We ended up using noise with value "1". Then we started to implement the various formulas from the Theory section, in our Python code. To verify some of the important factors like MSE and R^2, we compared our results with the Scikit Learn MSE and R^2 methods.

To make sure our code worked, we looked at the case with OLS, with and without resampling, for a 5th order polynomial degree. We start by making 3 plots of the MSE (test and training), R^2 (test and training) score and bias-variance tradeoff as a function of model complexity, and computed the confidence interval for the β parameters. At the end we made a 3D plot of the model, which we compared to the 3D plot of the original Franke's function. This part is not implemented in the result section, but can be found on the linked GitHub page.

Part II

The method for all 3 regression methods, including resampling, are fairly similar. We start by making 3 plots of the MSE (test and training), R^2 (test and training) score and bias-variance tradeoff as a function of model complexity. This means the degree of the polynomial we use to make the design matrix. We found that a good start were to plot for $m \in [0, 9]$, and by studying the dependence between the error and the complexity, we found that some polynomial degrees gave a better fit than others. Meaning, the lower the MSE, and a R^2 close to one, the better the model. We also tried to avoid polynomial degrees that gave under-or overfitting. This progress were used for OLS for both Franke's function and for the terrain data. The process is the same for Ridge and Lasso, but we have to take the hyperparameter into account. This meant we had to do the same analysis, but now we plotted the error as a function of various hyperparameters, for 10 different polynomial degrees. Also now we eliminated the models with the worst fit, and made new error plots with fewer models. An example of this is figure 7. There are similar figures for all methods in the linked GitHub page.

When the parameters were decided, we calculated a new model. Then we made new estimations of the mean MSE and R^2 score of the model, see table **??** and **??**, and calculated the confidence interval of the β parameters, se Appendix C. In the end we plotted a new 3D model of the best fit, for comparison of the original function/terrain.

—— Function test ——

Due to some time constraints, we were not able to conduct a lot of tests on the code. But some simple tests we implemented were to check the MSE and R^2 score functions with the Scikit Learn *mean_square_error()* and *r2_score()* functions.

Another mathematical test conducted were to check that the Ridge and Lasso method gave the same result as the OLS method when using $\lambda = 0$.

3.1 Franke's function

The two-dimensional Franke function is a weighted sum of four exponentials, which reads as follows:

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)^2}{10}\right) \\ + \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \tag{17}$$

To make our data $z(x,y)$ we are using random numbers, $x, y \in [0,1]$, and a normal distributed noise, $G(x,y) * A$, where A is the strength of the noise.

$$z(x,y) = f(x,y) + G(x,y) * A$$

In this case we used $A = 1$. Franke's function is shown in figure 8, where we can see that the main characteristics are two Gaussian peaks at different heights z, and a small dip.

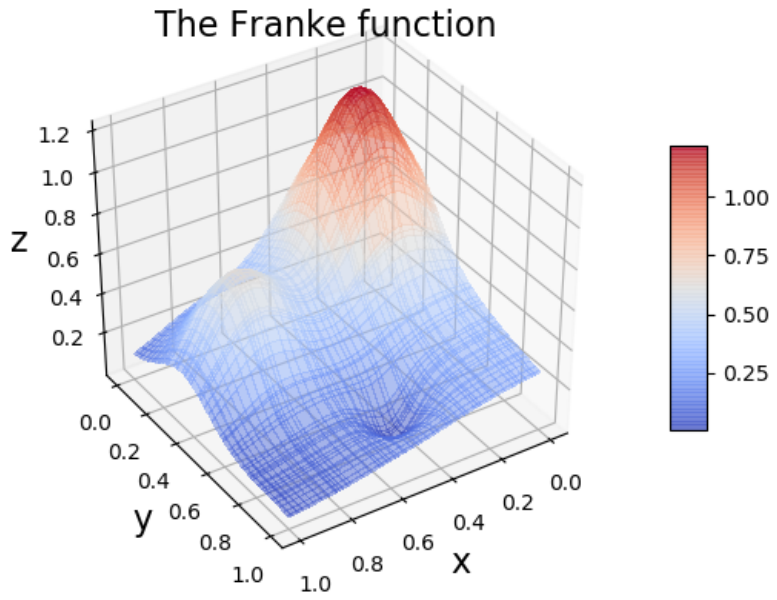


Figure 3: The Franke function, modelled in Python. The colorbar shows the altitude, z-values.

3.2 Terrain data

The digital terrain data set is 1456x719 points, and makes the regression methods slow to run. Therefor we scaled the data set by cropping the terrain image. This was done by choosing a region of interests, and extract the right parts of the array. We cropped the image down to 400x400, but the code was still slow to run. We then reduced the internal points in the image, taking a mean of the neighbour points. The finished reduced terrain image now consists of 200x200 points. The associated 3D image can be seen in figure 13. **a)**.

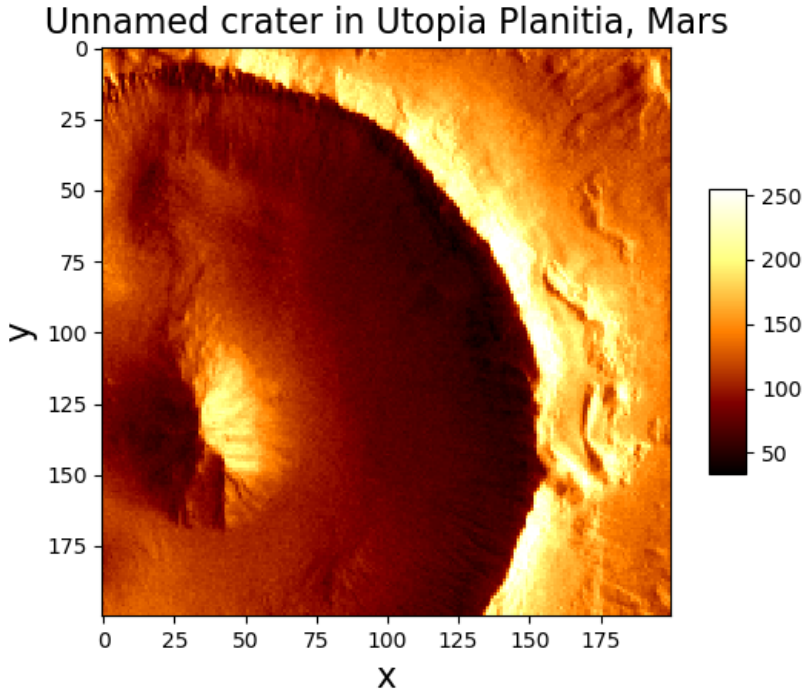


Figure 4: The final, cropped and reduced, terrain data set visualized as an 2D image. The colorbar shows the altitude of the terrain of the crater.

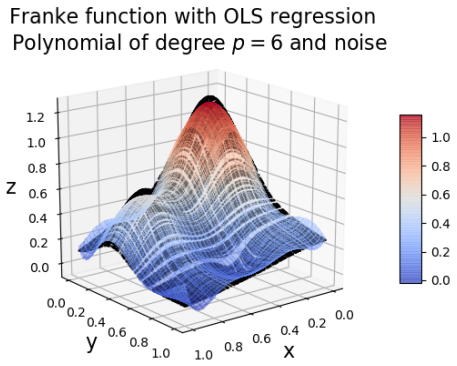
4 Numerical results and discussion

4.1 Franke's function

Table 1: Computed mean MSE and R^2 score for the best OLS, Ridge and Lasso model, for the data set using Franke's function. Table includes the best polynomial degree and λ

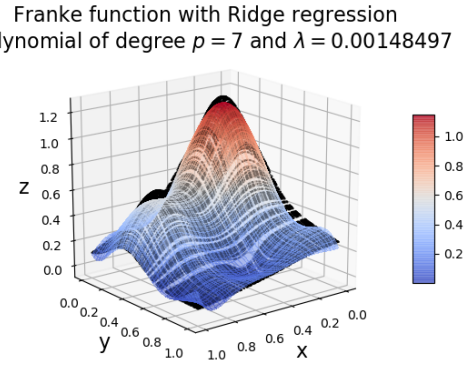
	m	**MSE**	$\mathbf{R^2}$	λ
OLS	6	0.00115391	0.986024	-
Ridge	7	0.00146716	0.98223	0.00148497
Lasso	4	0.0104423	0.873526	$1.67683 * 10^{-6}$

For Franke's function we can see in figure 5 that OLS and Ridge regression gave the best fit to the original function. The Lasso model on the other hand did not fit very well to the original function. This could be due to the low polynomial degree of only 4. The model, with polynomial degree 7 and $\lambda \approx 0.0015$, made by Ridge regression is the best fit due the nicer edges on the computed model,.



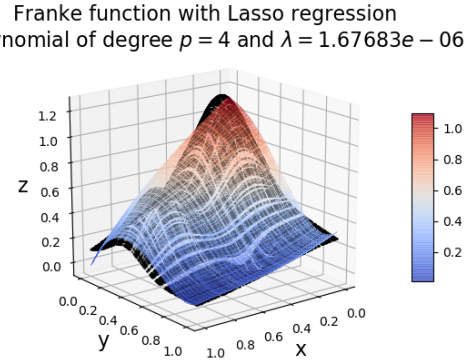
The black dots represent the true Franke function

a) The OLS regression method, with k-fold CV, gives a model that lies above Franke's function. We can see that the ends of the model points in the opposite direction.



The black dots represent the true Franke function

b) The Ridge regression method, with k-fold CV, gives a model that mainly lies above Franke's function. We can see that the ends are better for Ridge than for OLS (figure b)).



The black dots represent the true Franke function

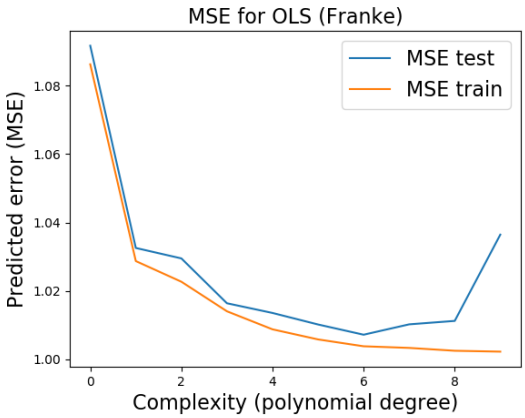
c) The Lasso regression method, with k-fold CV, gives a model that almost lies as a folded sheet of paper over the real function. This shows that the Lasso method clearly is the worst fit of the 3 regression methods.

Figure 5: 3D plots of the best fit models for OLS, Ridge and Lasso regression, with the true Franke function plotted as black dots in each image. Regression used with k-fold CV. We can clearly see from the comparison that OLS and Ridge gave the best fit.

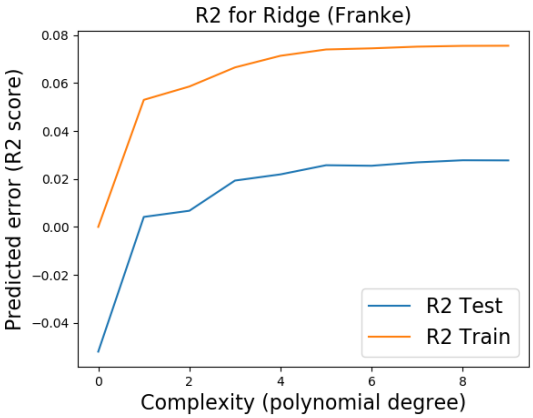
The modelled Franke function with OLS regression, with k-fold CV, in 3D is shown in Figure 5, b). In figure 6, we can se the MSE, R^2 and bias-variance tradeoff as function of complexity. The test and train MSE behaves as expected, where er see that the train MSE is fitted better and better, while the test MSE starts to overfit at higher polynomial degrees. This is supported by the small increase in the variance. From this figure we decided to use polynomial degree 6 as the best model. From table 1 we can se that the best polynomial degree 6 gives an average MSE of 0.0011, and almost a perfect R^2 score.

The same plots for Ridge and Lasso, as figure 6, can be found on the GitHub page. As well as for the terrain data results.

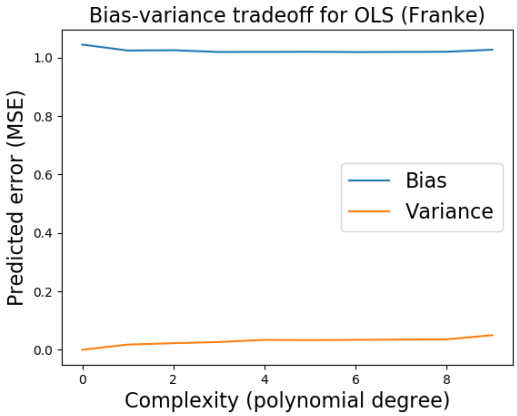
The modelled Franke function with Ridge regression, with k-fold CV, is shown in Figure 5, b). Figure 7 shows the predicted error as a function of the hyperparameter for the most promising



a) The MSE test and train as a function of complexity. We can see that polynomial degree 6 gives the lowest test MSE, and it's before critical overfitting takes place.



b) The R^2 test and training score as a function of complexity. We are not shore why we get negative values. The mean score is 0.98 for m=6.



c) The bias-variance tradeoff as a function of complexity. We can see that the bias lies around the value of the data noise, while the variance is unaffected by this.

Figure 6: MSE test and train, R^2 test and train and bias-variance tradeoff is shown in the images above. We can se that the MSE prediction is as expected, but the R^2 score plot gives off some weird results. We expected only positive values, and values much closer to 1. When we calculate the mean R^2 of the best model (after making a new prediction with the best parameters) we get a much more realistic R^2 score. See table 1. We also had some problems with the calculation of the bias throught the project. We have tripple checked the formula, and the implementation, and could not find the error. According to the students Piazza page, more students had this problem.

polynomial degrees. We decided to use polynomial degree 7 as the best model, because of the lower MSE and the slender curve, that indicates less overfitting. The mean MSE and R^2 score for the degree and corresponding hyperperameter can be found in table 1.

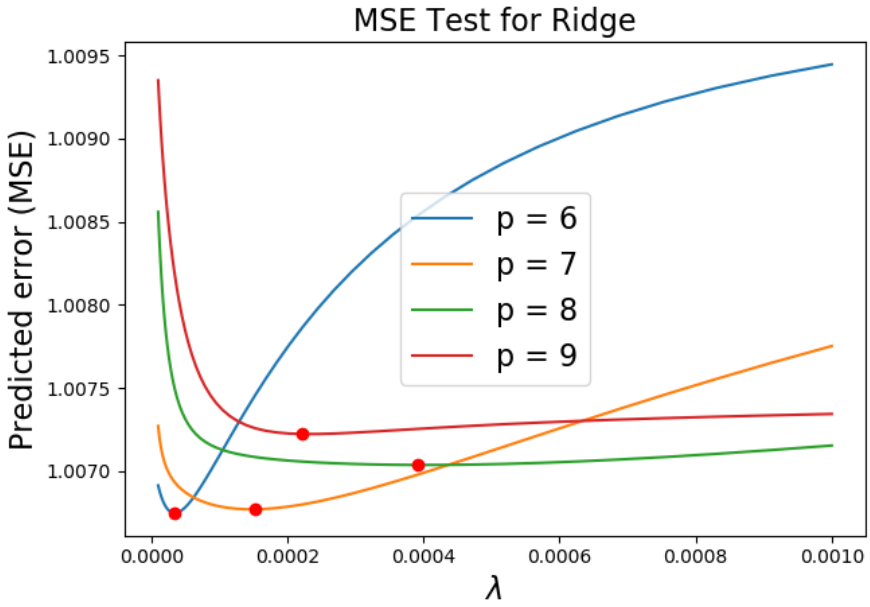


Figure 7: Predicted error as a function of the hyperparameter for the most promising polynomial degrees. We can see that for Ridge regression, polynomial degree 7 gives the best fit. This is based on the lower MSE test value, and a lower variance.

The modelled Franke function with Lasso regression, with k-fold CV, is shown in Figure 5, c). We can see from figure 8 that both polynomial degree 4 and 5 gives about the same MSE and hyperparameter value. We chose to make the final 3D model with polynomial degree 4 and $\lambda \approx 1.67 * 10^{17}$. As seen in the 3D figure, Lasso was the worst fit of the 3 regression methods.

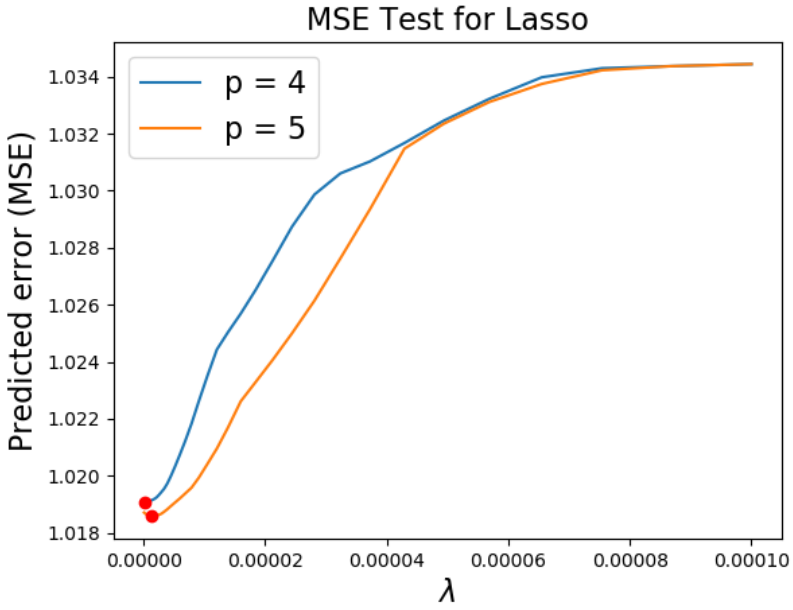


Figure 8: Predicted error as a function of the hyperparameter for the most promising polynomial degrees. For Lasso regression for Franke's function we decided to use polynomial degree 4. The error variance between the two models are fairly small.

4.2 Terrain data

Table 2: Computed mean MSE and R^2 score for the best OLS, Ridge and Lasso model, for the data set using the terrain data. Table includes the best polynomial degree and λ

	m	**MSE**	**R^2**	λ
OLS	8	752.996	0.644904	-
Ridge	9	729.121	0.656163	$5.38988 * 10^{-7}$
Lasso	8	1166.15	0.450073	0.00129155

Based on what we have learned from the case with Franke's function, we can see from figure 10 (the predicted surfaces of the crater from b)-d)) that OLS and Ridge regression gave the best fit for the terrain data set. On the real data we had to use higher polynomial degrees to make detailed models. This is likely caused by the large variation of altitudes on the cater, and in the data set, as seen on the original 3D plot of the real data. With the real data, we don't know how much noise there is, but we can assume there are a lot based on the error scores listed in table 2. We can also see from the confidence interval, Appendix C, that the β parameters had a very large spread.

The terrain data we used for the regression analysis are shown in figure 11. The colorbars shows the altitude of the terrain. The original, and the pre-reduced cropped terrain image can be seen in Appendix B.

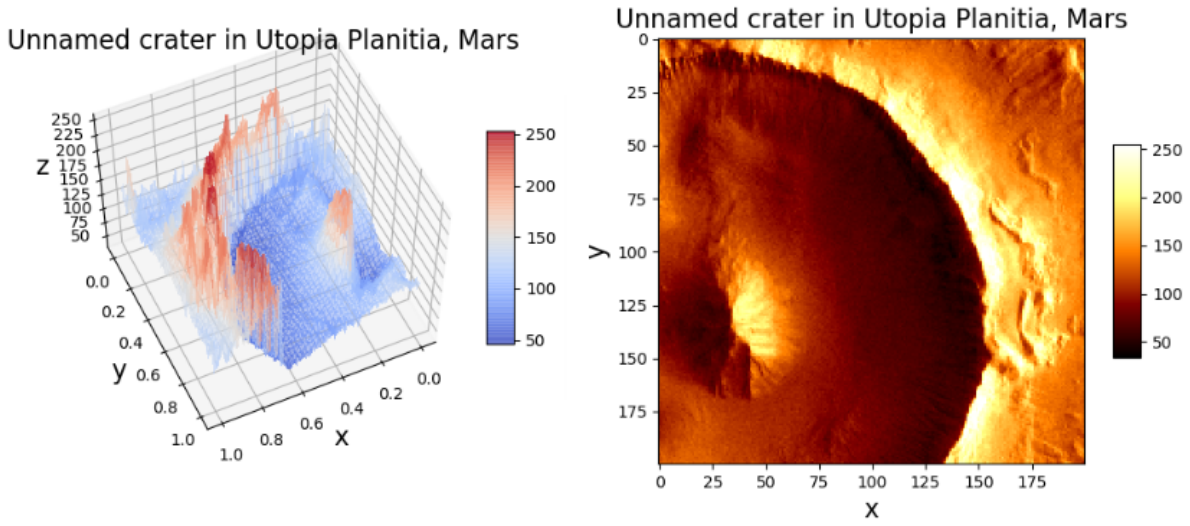
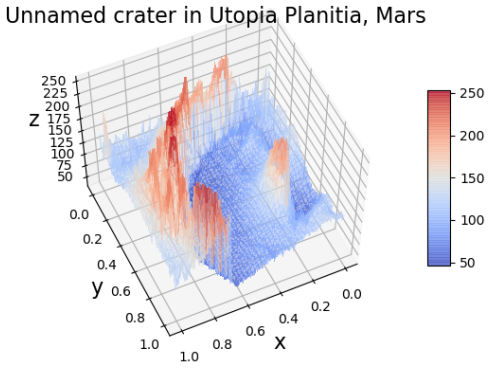
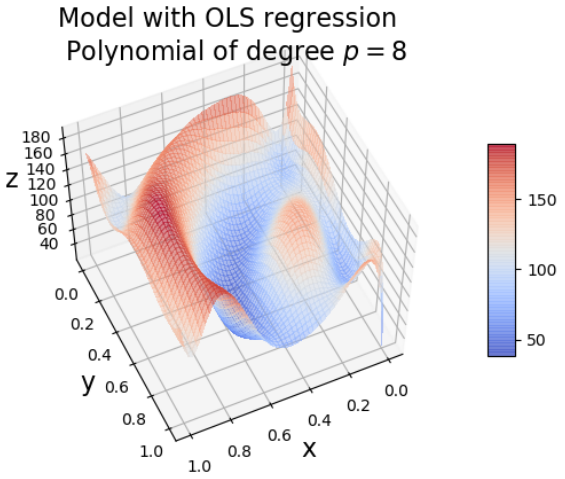


Figure 9: Reduces terrain images of an unnamed crater in Utopia Planitia on Mars.

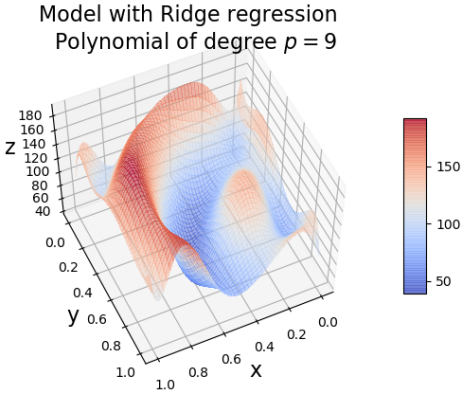
All additional plots used to determine the best models are in the result part of the GitHub directory. This part sadly got a bit short due to time constraints.



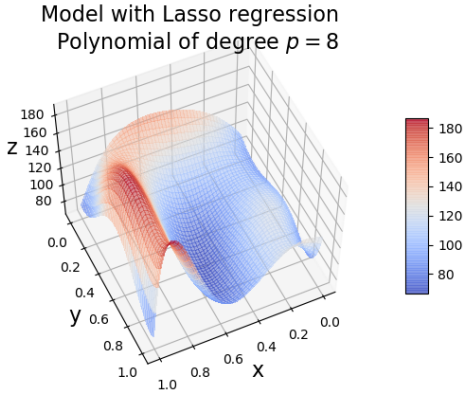
a) 3D visualization of a unnamed crater in the region Utopia Planitia on Mars, plotted with the original data.



b) 3D visualization with OLS regression and k-fold CV resampling. With polynomial degree 8.



c) 3D visualization with Ridge regression and k-fold CV resampling. With polynomial degree 9 and $\lambda = 5.39 * 10^{-07}$.



d) 3D visualization with Lasso regression and k-fold CV resampling. With polynomial degree 8 and $\lambda = 0.00129$.

Figure 10: 3D plot of the terrain data for the original data and the best fit model for OLS, Ridge and Lasso regression. When using k-fold CV. We can clearly see from the comparison that OLS and Ridge regression gave the best result, although far away from the original. OLS and Ridge managed to fit the parameters to the edge of the crater, ass well as the ”altitude” in the middle of the crater. Lasso struggled to clearly emphasis this altitude. Corresponding 2D colormap images can be found on the linked GitHub page. These were quite bad compared to the original, and did not get a place in the report.

5 Conclusion

For Franke's function we have to conclude that Lasso regression was the worst method. OLS and Ridge regression on the other hand did quite well. Because of better edges on the computed model, Ridge regression with polynomial degree 6 and $\lambda \approx 0.0015$, gave the best result compared to the true Franke function.

We have the similar results when fitting polynomials to a real life data set. For the terrain data of the crater on Mars, also here Lasso regression did very bad. This could also be because we failed to determine the best hyperparameter for Lasso. But we can conclude that OLS gave the best fit, with polynomial degree 8. Mainly because it managed to include lower and higher points of the terrain in the best model. Both OLS and Ridge started to get numerical difficulties at the edges. But Ridge with polynomial degree 9 and $\lambda \approx 5 * 10^{-7}$ gave almost the same result as the best OLS fit.

In hindsight, we should have implemented the Bootstrap resampling method instead of the k-fold resampling method, since we got some difficulties when shuffling the terrain data. Another thing to consider were the way be tested the model for different hyperparameters. Insted of one predetermined list of equaly spaced hyperparameter values, we tried to find a region of small values that gave an good MSE. This turned out to be extra tricky when using Lasso regression, since it's very sensitive to the hyperparameter.

——— Collaboration ———

On the Python code for this project I have mainly collaborated with Anna Eliassen, but we also collaborated on the k-fold CV with Aron Jansson Nordberg. In addition, a big thank you to our professor, Morten Hjorth-Jensen, for a lot of good feedback and help on the project.

References

[1] Morten Hjorth-Jensen
Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis. 2019
https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html

[2] Wikipedia
Linear regression. Last updated: 08.10.2019
https://en.wikipedia.org/wiki/Linear_regression

[3] Wikipedia
Tikhonov regularization. Last updated: 08.09.2019
https://en.wikipedia.org/wiki/Tikhonov_regularization

[4] Scikit Learn
sklearn.linear_model.Lasso. 2019
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[5] Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
An Introduction to Statistical Learning, with Applications in R.
Springer, New York, 2013. ISBN: 9781461471370

Appendix A - Mathematical formulas and algorithms

——— The derivation of the bias–variance decomposition for squared error ———

We need the definition for the variance of $\boldsymbol{y}$ (note: $\boldsymbol{z}$ in report) and ε

$$\begin{aligned}\text{Var}[\boldsymbol{y}] &= \mathbb{E}[\boldsymbol{y}^2] - \mathbb{E}[\boldsymbol{y}]^2 \\ \text{Var}[\varepsilon] &= \text{Var}[\boldsymbol{y}] + \cancelto{0}{\mathbb{E}[\varepsilon]^2} = \sigma^2\end{aligned}$$

Rearranging for $\mathbb{E}[\boldsymbol{y}^2]$, gives us

$$\mathbb{E}[\boldsymbol{y}^2] = \mathbb{E}[\boldsymbol{y}]^2 + \sigma^2 \tag{18}$$

We start by expanding the left-hand side of eq. (10)

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \mathbb{E}[\boldsymbol{y}^2 - 2\boldsymbol{y}\tilde{\boldsymbol{y}} + \tilde{\boldsymbol{y}}^2] = \mathbb{E}[\boldsymbol{y}^2] - 2\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}^2] \tag{19}$$

We then insert eq. (18) in eq. (19)

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[\boldsymbol{y}]^2 + \sigma^2 - 2\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}^2] \\ &= \mathbb{E}[\boldsymbol{y}]^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] + \sigma^2\end{aligned}$$

We now want to insert that $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$, but since f is deterministic, we use that $y = f + \varepsilon$

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[f + \boldsymbol{\varepsilon}]^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[(f + \boldsymbol{\varepsilon})\tilde{\boldsymbol{y}}] + \sigma^2 \\ &= \mathbb{E}[f + \boldsymbol{\varepsilon}]^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[f\tilde{\boldsymbol{y}} + \boldsymbol{\varepsilon}\tilde{\boldsymbol{y}}] + \sigma^2 \\ &= f^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[f\tilde{\boldsymbol{y}}] + \sigma^2\end{aligned} \tag{20}$$

where we used that $\mathbb{E}[\boldsymbol{\varepsilon}\tilde{\boldsymbol{y}}] = 0$ because $\tilde{\boldsymbol{y}}$ and $\boldsymbol{\varepsilon}$ are uncorrelated and $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$. We also used that $\mathbb{E}[f + \boldsymbol{\varepsilon}] = f$. Adding and subtracting $2\mathbb{E}[\tilde{\boldsymbol{y}}]^2$ to eq. (20) gives us

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= f^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[f\tilde{\boldsymbol{y}}] + \sigma^2 + 2\mathbb{E}[\tilde{\boldsymbol{y}}]^2 - 2\mathbb{E}[\tilde{\boldsymbol{y}}]^2 \\ &= f^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] - 2\mathbb{E}[f\tilde{\boldsymbol{y}}] + \sigma^2 + \mathbb{E}[\tilde{\boldsymbol{y}}]^2 + \mathbb{E}[\tilde{\boldsymbol{y}}]^2 - 2\mathbb{E}[\tilde{\boldsymbol{y}}]^2 \\ &= \sigma^2 + f^2 - 2\mathbb{E}[f\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}]^2 + \mathbb{E}[\tilde{\boldsymbol{y}}^2] + \mathbb{E}[\tilde{\boldsymbol{y}}]^2 - 2\mathbb{E}[\tilde{\boldsymbol{y}}]^2 \\ &= \sigma^2 + (f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \mathbb{E}(\tilde{\boldsymbol{y}}^2 + \mathbb{E}[\tilde{\boldsymbol{y}}]^2 - 2\tilde{\boldsymbol{y}}\mathbb{E}[\tilde{\boldsymbol{y}}]) \\ &= \sigma^2 + (f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2\end{aligned}$$

This is the same equation as (10)

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \frac{1}{n}\sum_i (f_i - \mathbb{E}\,[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\,[\tilde{\boldsymbol{y}}])^2 + \sigma^2 \\ &= \text{Bias}^2 + \text{Var} + \sigma^2\end{aligned}$$

——— Design Matrix ———

```
def CreateDesignMatrix(x, y, n):
    """
    Function for creating a design X-matrix,
    with rows [1, x, y, x^2, xy, y^2 , etc.]
    Input x, y      | Datpoints x and y after meshgrid
    Input n         | The degree of the polynomial you want to fit
    """
    if len(x.shape) > 1:
        x = np.ravel(x)
        y = np.ravel(y)

    N = len(x)
    l = int((n+1)*(n+2)/2)      # Number of elements in beta
    X = np.ones((N,l))

    for i in range(1,n+1):
        q = int((i*(i+1)/2))
        for j in range(i+1):
            X[:,q+j] = x**(i-j) * y**j

    return X
```

Appendix B - Terrain data set: Utopia Planitia, Mars

The raw data from NASA JPL, datafile: PIA23328: Radial Ejecta, is visualized as an image with size 1456x719 in figure 11. The cropped terrain image in figure 12 is cropped to size 400x400. The images are shown in 'afmhot'-python color and in gray. The colorbars shows the altitude of the terrain, an unnamed crater in Utopia Planitia, on Mars.

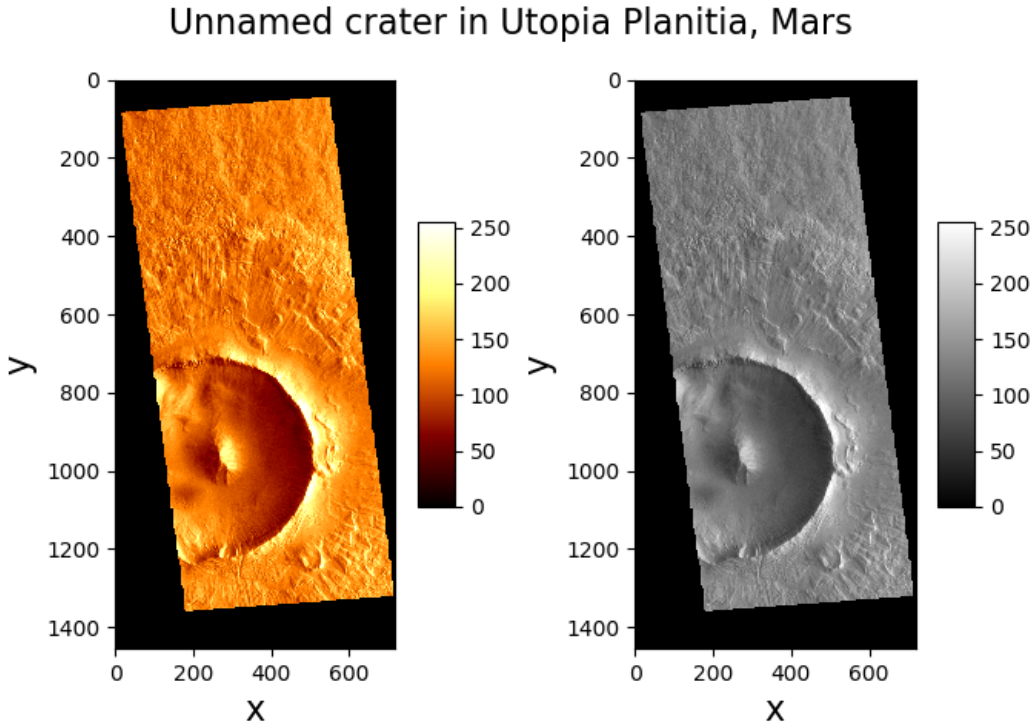


Figure 11: Original terrain image, with size 1456x719. The colorbar shows the terrain altitude. The terrain is over an unnamed crater in Utopia Planitia on Mars.

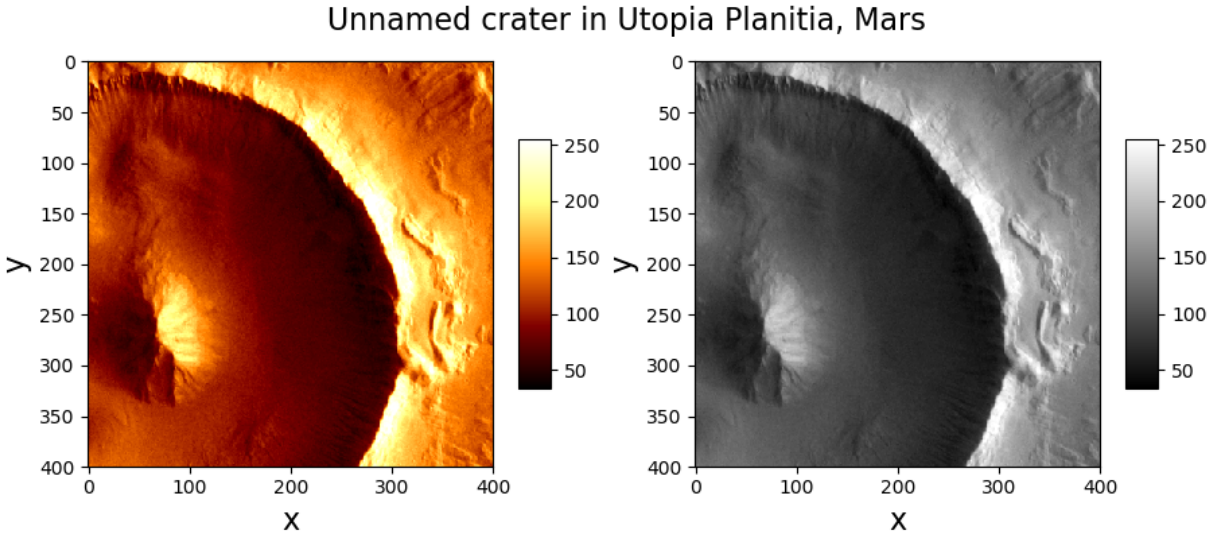
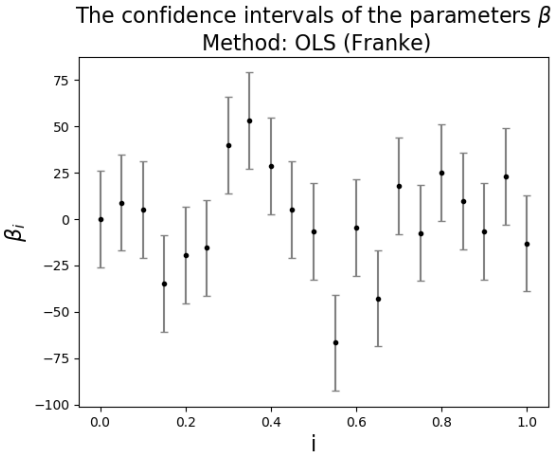
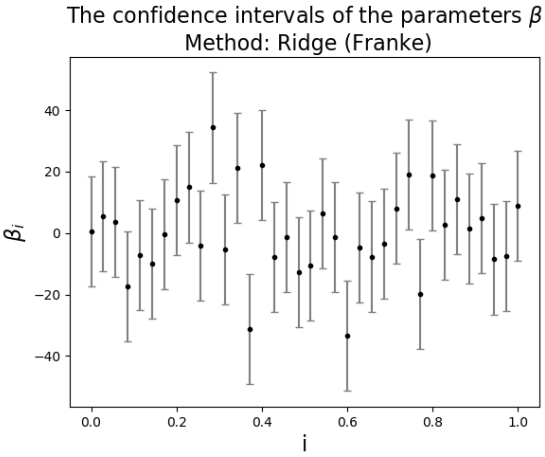


Figure 12: Cropped terrain image, with size 400x400. The colorbar shows the terrain altitude. The terrain is over a part of an unnamed crater in Utopia Planitia on Mars.

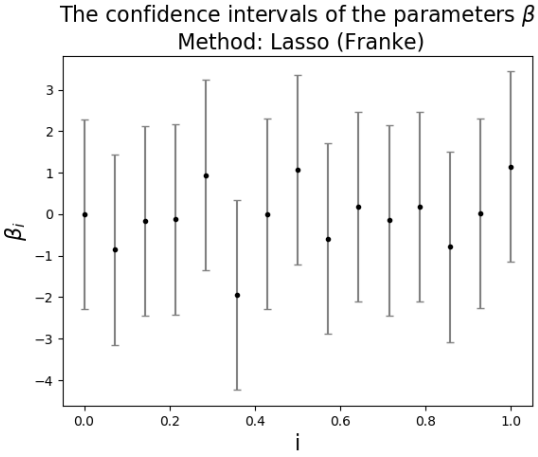
Appendix C: Confidence interval for the β parameters



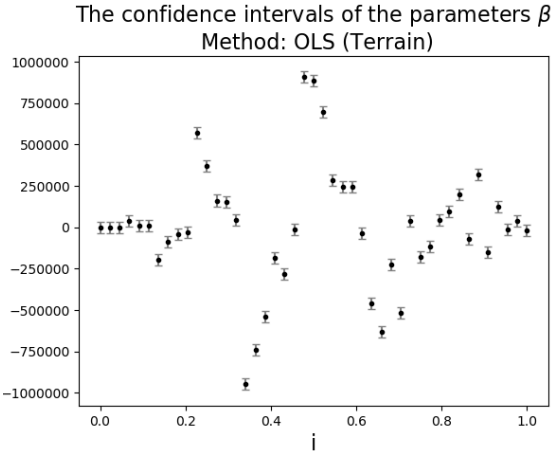
a) CI of the β parameters, for OLS regression



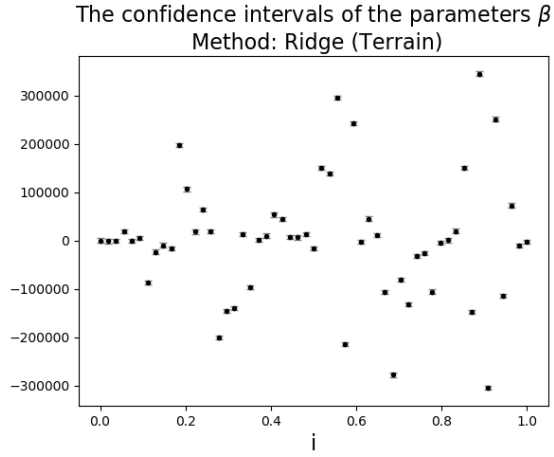
b) CI of the β parameters, for Ridge regression



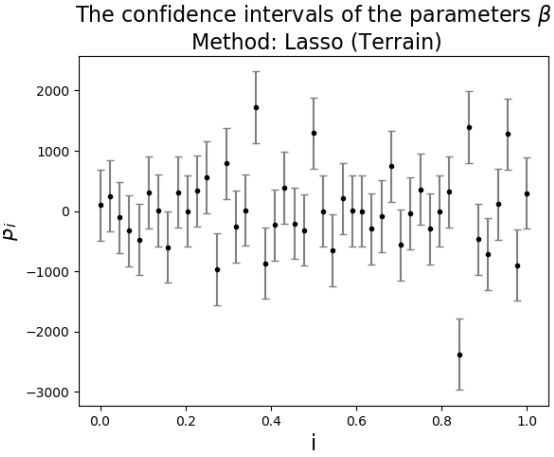
c) CI of the β parameters, for Lasso regression



d) CI of the β parameters, for OLS regression



e) CI of the β parameters, for Ridge regression



f) CI of the β parameters, for Lasso regression

Figure 13: The confidence interval (CI) of the β parameters visualized as error-bars. The figure contains the CI for the FF dataset in the subfigures **a)-c)** and for the terrain dataset in subfigure **d)-f)**. We can see that the β values for the FF is much lower than for the terrain dataset. All confidence intervals is computed for the best fit. Note: Some y-labels disappeared, same for all.