Linear Regression on Terrain Data

Camilla Marie Baastad Niels Bonten

October 9, 2018

Abstract. We find ordinary least squares, Ridge and Lasso regression models of Franke's function and real terrain data. More precisely, we find polynomial fits in two variables and consider how these models perform in terms of R^2 scores, MSE, and bias when we vary complexity and the penalty parameter λ for Ridge and Lasso. We use bootstrapping to study confidence intervals of the coefficients of our model polynomial, and k-cross validation to test our models. We find good models of Franke's function. On a small sample of our terrain data we also find a good fit, with an R^2 score of 0.9876. However, we experience difficulty in finding good models of big data samples. We conclude that fitting data piecewise can be more efficient when dealing with big data samples. Also we little of the bias-variance trade off that we expected to see.

All code and data used in this project can be found at: `https://github.com/CamillaBa/FYS-STK4155_Project_1`.

Introduction

Suppose you have performed an experiment, and you want to examine the relationship between your observations and the different predictors. The presumably simplest way to do this is to assume your observations are linear in terms of the predictors. In linear regression, we try to find the best such model.

We have developed code that performs Ordinary least squares, Ridge and Lasso regression on a data set. We have implemented k-fold cross-validation and bootstrapping. To evaluate our models we compute the mean square error and the R^2 score function for each of the regression methods. First we test our program on the Franke function. Then we use our program on real data, terrain data from a region close to Stavanger in Norway. The terrain data set is too large and noisy for our computer to find a decent fit, and we propose two solutions to solve this problem. The first, and seemingly best, is to split our data set into smaller regions and perform a fit on each region. The second is to lower the quality of our data set and performing a fit on the low quality version of the data.

A presentation of the three linear regression methods used in this report can be found in section 1. Here you will also find information about the resampling techniques being used, as well as the needed background material in statistics and error analysis. In section 2 we explain some of the numerical methods used and the ways we to modified the large terrain data set to be able to do regression analysis

on it. The results we found for the Franke function, a small sample of the terrain data and a low quality version of the terrain data are presented in section 3.

Contents

1 Theory

In this section we will present three linear regression methods – ordinary least squares, Ridge and Lasso regression. We will provide the necessary statistical background and explain the error analysis and resampling techniques used in this article. The material in this section is based on "An introduction by statistical learning" by Hastie et al [Jam+13], *Lecture notes on ridge regression* by van Wieringen [Wie15] and *A high-bias, low-variance introduction to machine learning for physicists* by Mehta et al[Meh+18].

1.1 Statistics

We go through some standard standard statistical quantities. The *sample mean* of a data set $\{y_i\}$ of size n is given by

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

The *sample variance* a data set $\{y_i\}$ of size n is given as

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2.$$

It is an important quantity that provides information about how the data is distributed. The sample mean and the sample variance are not necessarily the same as the true mean and the true variance, which we would need to know the probability distribution of our data set to compute. However, as the sample size grows large the sample mean will go to the true mean and the sample variance to the true variance, due to the law of large numbers.

The *standard deviation*, σ, is defined as the square root of the variance.

A *confidence interval* is an interval estimate with a certain probability to contain a certain parameter. For example, a 95 % confidence interval is the interval estimate with a 95 % probability of including the parameter in question. If μ is the mean, then the confidence interval is given by

$$C = \mu \pm z^* \frac{\sigma}{\sqrt{n}}.$$

The value of z^* depends on what kind of confidence interval we want. For a 95 % interval, $z^* = 1.96$. Assuming a normally distributed data set, we can use the empirical 66-95-98-rule to find the 95 % confidence interval as

$$C = \overline{y} - 2\sigma.$$

1.2 Linear regression

Consider an experiment of n observations measured in p predictors. The values of the predictors for each observation are stored in the design matrix, $\mathbf{X}$. That is,

$$\mathbf{X} = (\mathbf{X}_1\, \mathbf{X}_2 \cdots \mathbf{X}_p) = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix},$$

while the dependent variables are stored in a vector $\mathbf{y}$. The linear regression model assumes that a dependent variable y_i can be approximated by a linear function in terms of the predictors. This gives rise to a set of linear equations

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i,$$

where ϵ_i is the error term. Adding a column vector $\mathbf{X}_0$, consisting of ones, to the design matrix, the set of equations can be written as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

In our case the design matrix will consist of monic monomials in two variables up to a given degree. For example, if the degree is 3, we have the basis element x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3 and 1 (to include the intercept). For degree n the number of basis elements will be given by the $(n+1)$-th triangular number, $\frac{(n+1)(n+2)}{2}$.

The regression parameter, $\boldsymbol{\beta} = (\beta_0.\beta_1, \cdots, \beta_p)$, remains to be determined.

1.2.1 Ordinary least squares

We want to choose the regression parameter in such a way that the error is minimized. To do so we require an expression for the error, a so called *cost function*. Ordinary least squares is a linear regression model that uses the sum of squared residuals as the cost function,

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2.$$

Writing the above equation in matrix form, we get

$$Q(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}).$$

The optimal regression parameter is thus given by

$$\boldsymbol{\beta}_{OLS} = \arg\min_{\beta}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}).$$

We can find $\boldsymbol{\beta}_{OLS}$ by differentiating the cost function and setting the result equal to zero. It is straight forward to check that

$$\frac{\partial Q((\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}),$$

and so we see that

$$\boldsymbol{\beta}_{OLS} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}.$$

Notice that we assume $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}$ is well-defined. This is okay if all the coloumn vectors in the design matrix are linearly independent. However, if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, we will encounter problems using the OLS method.

1.2.2 Ridge regression

If $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}$ is not well-defined, the regression parameter $\boldsymbol{\beta}$ can not be computed using the OLS method. If the matrix is close to being singular, we may be able to compute $\boldsymbol{\beta}$, but our solutions will not be stable and thus have a high variance. In the 1970s, Hoerl and Kennard proposed an ad hoc solution to this problem. By simply adding a small *penalty parameter*, λ, along the diagonal, we can invert the matrix and obtain an estimate of the regression parameter.

Ridge and Lasso regression are so called *regularization methods*, where we add a *regularizer* that shrinks the regression parameters to the cost function. This will reduce the variance of our model, however, it will introduce a bias. For further explanation, see "An introduction by statistical learning" by Hastie et al. [Jam+13]

In Ridge regression, the regularizer is the L_2 norm of the regression parameter. The cost function becomes

$$Q(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}.$$

Differentiating the cost function and setting the result equal to zero, we obtain the following expression for the optimal regression parameter:

$$\boldsymbol{\beta}_R = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda I)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}.$$

Notice that when $\lambda = 0$, Ridge regression becomes OLS regression.

1.2.3 Lasso regression

Lasso regression is similar to Ridge regression, only we use the L_1 instead of the L_2 norm in the regularizer. The cost function becomes

$$Q(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|,$$

where $|\boldsymbol{\beta}| = \sum_{i=1}^{n} |\beta_i|$. The optimal $\boldsymbol{\beta}$ is the given by

$$\boldsymbol{\beta}_L = \arg\min_{\beta}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|).$$

As opposed to the prevoius two method, we cannot find any closed form solution for $\boldsymbol{\beta}_L$. As we have seen, including a penalty parameter shrinks all the regression parameters towards zero. Using Ridge regression they never become exactly equal to zero. With Lasso regression, however, some parameters can become zero when λ is large enough. This means that Lasso offers a parameter selection as well, which may lead to a better fit than what we would get from Ridge regression. See [Jam+13, p. 222] for a nice illustration of this.

Remark 1.1. In order to minimize the cost function, we actually assume that both Lasso and Ridge regression are convex in β. If not, then the minimum we find may not be global, but local. Luckily, it turns out that both Lasso and Ridge regression are in fact convex in β in our case, as explained in [Meh+18].

1.3 Error analysis

We will consider two common methods for estimating the error, namely Mean Square Error (MSE) and the R^2 score function.

1.3.1 Mean Square Error (MSE)

The Mean Square Error is given by

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where $\hat{y}_i$ is the estimate of the observation y_i. Let $\overline{\hat{y}} = \frac{1}{n}\sum_{i=1}^{n}\hat{y}_i$ be the mean of all the estimated values $\hat{y}_i$. We can rewrite the above expression in the following way:

$$\begin{aligned}
\mathrm{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i + \overline{\hat{y}} - \overline{\hat{y}})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left((y_i - \overline{\hat{y}}) - (\hat{y}_i - \overline{\hat{y}}\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{\hat{y}})^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \overline{\hat{y}})^2 - \frac{2}{n}\sum_{i=1}^{n}(y_i - \overline{\hat{y}})(\hat{y}_i - \overline{\hat{y}}).
\end{aligned}$$

Here the first term is what is commonly referred to as the $\text{Bias}^2(\hat{y})$, while the second term is $\text{Var}(\hat{y})$. High variance is associated to high complexity, and a model with high variance tends to overfit. This will yield a small error for the data on which the model is trained. However, the models ability to predict new data will be rather poor. A model that is too simple will result in a low variance, but a high bias. We get underfitting. In order to minimize the MSE we are forced to choose a complexity that will not leave neither the variance nor the bias too high. This is what is known as the Bias-Variance trade-off.

1.3.2 The R^2 score function

The R^2 score function is given by

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

The best possible score is $R^2(\hat{y}) = 1$, which is when our predictions equals the observed results. We get an $R^2(\hat{y})$ score of 0 when all of our predictions are equal to the mean. While the $R^2(\hat{y})$ score usually lays bewtween 0 and 1, we may get a negative score if our model results in a worse fit than simply using the mean.

1.4 Resampling techniques

Resampling is a method consisting of repeatedly drawing random samples from the original data set. By refitting a model to each sample and comparing these to the original model we may obtain new information about our fit. Two common resampling techniques are *cross-validation* and *bootstrapping*.

1.4.1 k-fold cross-validation

While we may find that our model yields a high R^2 score or a low MSE for the data points on which the model is based, its ability to make predictions about new data can be poor. Especially if we use a high complexity model or there is noise in the data, we may end up overfitting, which can cause a large error in the predictions. Using the k-fold cross-validation resampling technique we are able to check how good our model is at making predictions about new data.

In k-fold cross-validation, we do the following:

(i) Randomly split the data set into k subsets of equal size.

(ii) Use $k-1$ of the subsets as training data, while the last subset is reserved for testing the fit.

(iii) Train the model on the training data.

(iv) Estimate the error of the testing data.

(v) Repeat steps (i)-(iv) k times until each subset has been used to evaluate the model.

(vi) Average the k errors obtained from the testing data to indicate the performance of the model.

We have to choose the number of folds, k, wisely. A large value of k will leave us with very few data points in each testing data set and result in a high variance. A small k may lead to a high bias, as our model is fitted on too few data points. Choosing $k \approx 10$ typically leaves neither the variance or the bias too high.

1.4.2 Bootstrapping

Bootstrapping allows us to estimate statistical properties of a certain parameter. We will use bootstrapping to estimate the mean and the variance of β. The procedure is as follows:

(i) Draw n points from the data set randomly and with replacement.

(ii) Compute β using this sample.

(iii) Repeat (i) and (ii) m times.

(iv) Use the average of the m β's as an estimate for the mean, and compute the variance.

As described in section 1.1, the distribution found from bootstrapping will approximate the true distribution as m grows large.

1.5 The Franke function

The Franke function [Fra79] was originally constructed to use as a test function in interpolation problems, which is what we will use it for. The function consists of two Gaussian peaks and a smaller Gaussian dip, and is given by

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

2 Methods

2.1 The class regdata

We have implemented ordinary least square, Ridge, and Lasso regression. More precisely, we have fitted a real polynomial $p(x,y)$ of order d to a data set consisting of heights z over points (x,y) on a surface. To do this we have created a class called *regdata*.

To initiate an object of the type regdata, we give as argument a height coordinates $\{z_{ij} \mid 1 \leq i \leq m,\, 1 \leq j \leq m\}$ stored as a matrix and the degree d for the polynomial we want as a model. The following class attributes are then calculated:

1. An array $\{x_i\}_{i=1}^m$ and an array $\{y_i\}_{i=1}^n$, such that $0 \leq x_i, y_j \leq 1$ to match the shape of (z_{ij}).

2. A bijection
$$\text{correspondence}\colon \{1,\cdots,m\} \times \{1,\cdots,n\} \to \{1,\cdots,mn\}$$
that relabels the data points $\{(z_{ij}, x_i, y_j)\}$ indexed by integers $1 \leq i \leq m$ and $1 \leq j \leq n$ to data points $\{(z_k, x_k, y_k)\}$ indexed by $1 \leq k \leq mn$. This bijection is stored as a list.

3. A vector $\mathbf{z}$ such that the k'th entry z_k is the height corresponding to the point (x_k, y_k).

4. A design matrix $\mathbf{X}$ such that the rows in $\mathbf{X}$ are of the form
$$(1,\, x_k,\, x_k^2,\, \ldots,\, x_k^d,\, x_k y_k,\, x_k^2 y_k,\, \ldots,\, x^{d-1} y_k,\, \ldots,\, y_k^d)$$
with column length equal to mn.

5. The number of basis elements D needed for the real vector space consisting of polynomials in two variables of order no greater than d. Recall that $D = (d+1)(d+2)/2$, for example by expanding $(x+y)^d$.

We will now discuss some of the functions in regdata. A description of what each function does should be found within the file:

https://github.com/CamillaBa/FYS-STK4155_Project_1/blob/master/Methods.py

2.2 The function "get_reg(args*)"

The function "get_reg()" returns an $m \times n$ matrix of height values $\{\hat{z}_{ij}\}$ over the points $\{(x_i, y_j)\}$ for integers $1 \leq i \leq m$ and $1 \leq i \leq n$. If no arguments are given, then the height values $\hat{z}_{ij}$ are determined by an ordinary least squares performed on the points $\{(x_k, y_k, z_k)\}$. That is, the height values $\hat{z}_{ij}$ are then determined by a polynomial $p(x, y)$ whose coefficients $\boldsymbol{\beta}$ are determined by:

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{z} \quad \text{(Ordinary least square)}$$

If we pass a number λ to "get_reg", then "get_reg(λ)" returns height values $\hat{z}_{ij}$ that are determined by a polynomial $p(x, y)$ whose coefficients $\boldsymbol{\beta}$ are determined by:

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

In both Lasso and Ridge we have used singular value decomposition to find $\boldsymbol{\beta}$. Our motivation for this is explained in fig. 5.

Lastly, if we pass two numbers λ and ϵ to "get_reg", then "get_reg(λ, ϵ)" returns height values $\hat{z}_{ij}$ that are determined by a Lasso regression. The polynomial $p(x, y)$ is determined by the coefficients $\boldsymbol{\beta}$ that minimize:

$$Q(\boldsymbol{\beta}) = \sum_{k=1}^{mn}(z_k - \beta_0 - \sum_{l=1}^{D}\beta_l X_{kl})^2 + \lambda\sum_{l=1}^{D}|\beta_l| \quad \text{(Lasso)}$$

To solve this numerically, we have implemented the shooting algorithm, which converges to the correct $\boldsymbol{\beta}_{\text{Lasso}}$ starting from $\boldsymbol{\beta}_{\text{Ridge}}$. [Rob14] We use the number ϵ to decide on a tolerance for when $\boldsymbol{\beta}$ is close enough to its limit. Let $\boldsymbol{\beta}_{\text{old}}$ and $\boldsymbol{\beta}_{\text{new}}$ respectively denote one iteration for beta and the next. If

$$|\boldsymbol{\beta}_{\text{old}} - \boldsymbol{\beta}_{\text{new}}| < \epsilon,$$

then we say that $\boldsymbol{\beta}$ has converged.

Let $\boldsymbol{X}_i$ denote the i'th row vector in $\boldsymbol{X}$. The shooting algorithm then becomes :

Algorithm 1: Shooting algorithm

Initialize $\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{I\lambda})^{-1}\boldsymbol{X}^T z$;
while $|\beta_{\text{old}} - \beta_{\text{new}}| < \epsilon$ **do**
  for $j = 1$ **to** D **do**
    $a_j = 2\sum_i X_{ij}^2$;
    $c_j = 2\sum_i X_{ij}(z_i - \beta^T \boldsymbol{X_i} + \beta_j X_{ij})$;
    $$\beta_j = \begin{cases} \beta_j = (c_j + \lambda)/a_j, & c_j < -\lambda, \\ \beta_j = 0, & c_j \in [-\lambda, \lambda], \\ \beta_j = (c_j - \lambda)/a_j, & c_j > \lambda. \end{cases}$$
  end
end

2.3 The function "bootstrap_step(sample_size, args*)"

Given an object of the type linregtools, the function "get_bootstrap_step(sample_size, args*)" can be used to yield a fit based on a random subset of observations of the set $\{(x_k, y_k, z_k) \mid 1 \leq k \leq mn\}$. Again args* determines the regression method. As with get_reg(), if we give a bias λ as argument, the result is a Ridge regression, and giving a tolerance ϵ results in a Lasso. The argument sample_size determines the size of the sameple.

2.4 The function "get_data_partition(k)"

This function is made with k-cross validation in mind. Given a number k, we seek to divide our data D into k slices $D = \sqcup_{i=0}^{k} D_i$ for randomly chosen D_i such that $|D_i| = k$. The way we implement this is as follows.

We start with an array $\{1, \ldots, mn\}$ corresponding to our data points (z_k). Then we permute this set randomly, yielding $I = \{s_1, \ldots, s_{mn}\}$. We now divide this array into subarrays I_i of length k by letting

$$I_i = \{s_{1+ik}, \ldots, s_{1+ik+k}\}$$

The output of "get_data_partition(k)" is a list $[I_1, \ldots, I_{mn/k}]$. Each entry in the list then determines (via indices) a subset of our data D, which we can treat as testing data. The remaining data is treated as training data.

2.5 Working with real data

A challenge when working with real data from height values on a map is that the resolution can become so high that running the model takes a long time. For doing quick code testing, this can become a hinder. To circumvent this issue we have transformed the data to a low quality version on which we then test our code. This allows us to find a suitable order for our polynomial as well as suitable values for λ for the Lasso and Ridge, before we attempt fitting to the true data.

Supposing that $x_i, y_j \in [0, 1]$, the idea is that we divide the data points $\{(x_i, y_j)\}$ into disjoint rectangles R of a fixed size. Each such rectangle R then consists of a subset of the data points $\{(x_i, y_j)\}$ with height

values z_{ij}. We then order the rectangles in a grid R_{kl} indexed by k, l, and let $\overline{z}_{kl}$ be the average of the z_{ij} such that the points (x_i, y_j) lie in R_{kl}. Pictorially, this process is illustrated if fig. 1.

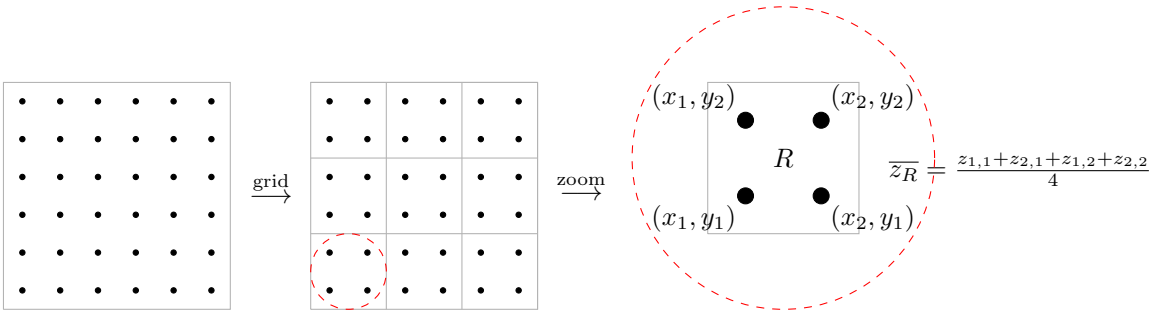


Figure 1: The leftmost picture consists of the data with the original quality. Then we add a grid that separates rectangles of data as indicated in the center figure. Then we focus on each such rectangle R and calculate the mean value for the heights.

3 Results and discussions

Before we go through the results, we feel the need to remark that our implementation of Lasso converges slowly. As a result, we had to wait a considerably longer for all plots containing anything Lasso-related, than for the other two methods. But as we shall see, Lasso did not leave us waiting in vain.

3.1 Franke's function

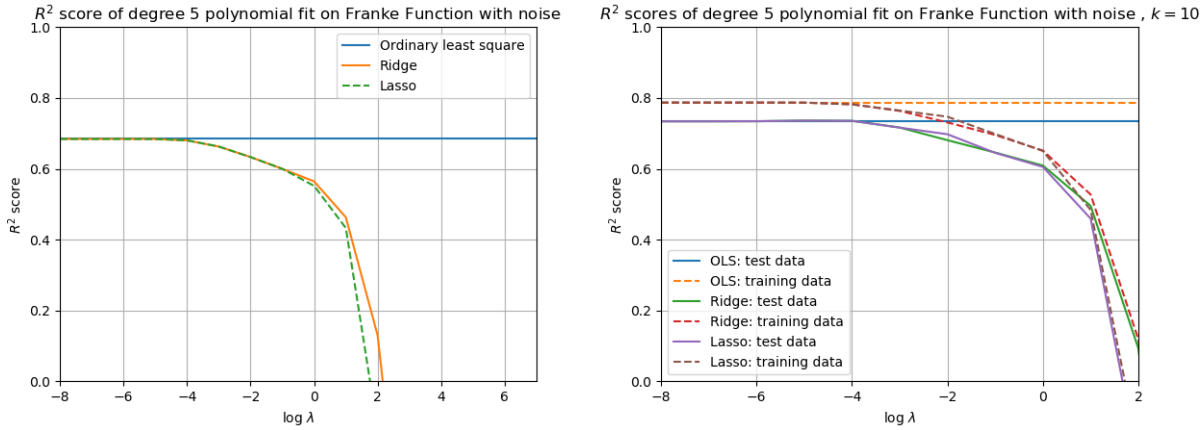
We consider as data the Franke function plus a gaussian noise term throughout this subsection. So in total we have

$$\text{data}_s(x, y) = f(x, y) + s \cdot \text{noise}(x, y)$$

where $s \in \mathbb{R}$ is some number to scale the noise and $|\text{noise}(x, y)| \leq 1$. We have made a grid to represent $[0.1] \times [0.1]$ using grid size 20×20, that is, $x_i = i/20$ and $y_j = j/20$ where $i, j = 0, 1, \ldots 19$. Our dataset becomes:

$$\text{data}_s = \{\text{data}_s(x_i, y_j) \mid x_i, y_j \in [0, 1],\ i, j = 0, 1, \ldots, 19\}$$

Note that $|\text{data}_s| = 400$.



(a) Training and fitting to whole dataset

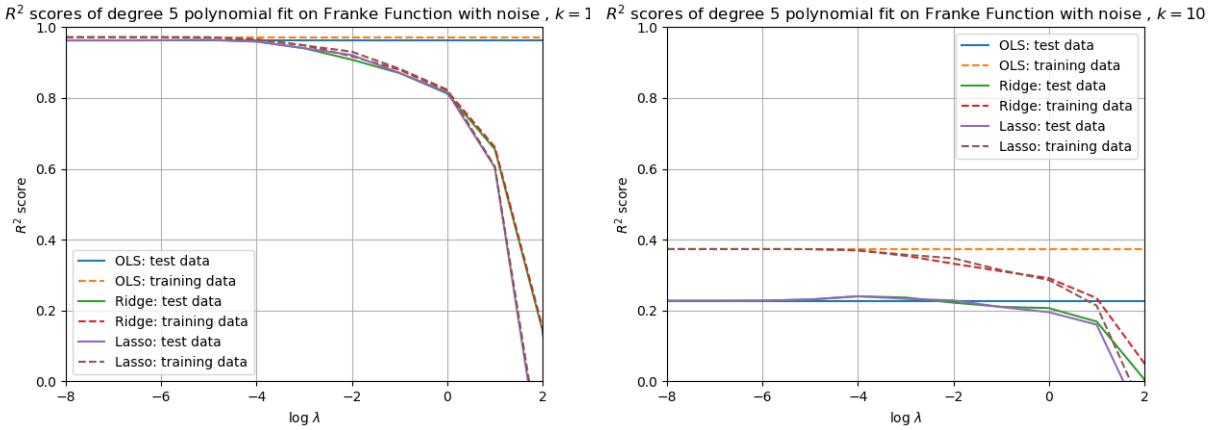
(b) R^2 score from the training data and the testing data.

Figure 2: A comparison between models of Franke's function with noise ($s = 0.2$) using ordinary least squares, Ridge and Lasso with and without 10-fold cross-validation. The R^2 scores in fig. 2b are the average scores for the training and testing data from the 10-fold cross-validation.

Ordinary least squares performs better than Ridge and Lasso for $\lambda > 10^{-2}$ on the whole data set (see fig. 2a). This is to be expected because both Ridge and Lasso can interpreted as fits resulting from ordinary least squares fits where one sabotages the coefficients of higher order monomials. Sabotaging the best fit on a given dataset obviously makes it perform worse on that same data.

Figure 2b shows that the R^2 score on the training data are higher than the R^2 score of the testing data. While some difference is to be expected, we suspect this is due to a slight overfitting. Perhaps this difference would have been smaller had we chosen a smaller k. Again the Ridge and the Lasso method performs worse than the ordinary least squares method when $\lambda > 10^{-2}$. When $\lambda < 10^{-4}$, the

performance of all three methods are about the same, with and without cross-validation. This indicates that there is no problems with our matrix inversion.



(a) $s = 0$ (b) $s = 0.4$

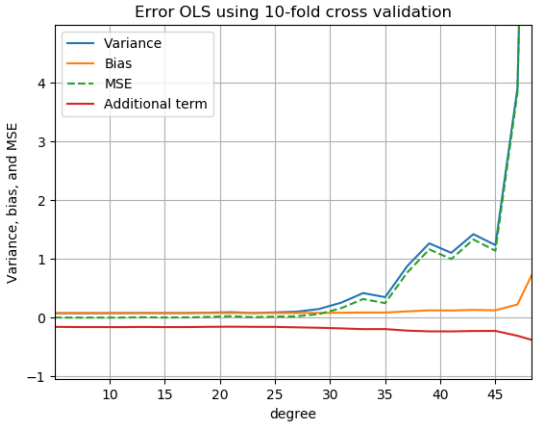
Figure 3: The above two plots are two plots from an animated series. To see the whole series, click: github.com/CamillaBa/FYS-STK4155_Project_1/blob/master/animation.gif. This link goes to a gif animation of plots of the R^2-scores of fits of the dataset data_s for increasing noise $s \in \{0, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4\}$. These models are found using 10-fold cross-validation as in fig. 2b.

Figure 3 shows that for noiseless data, all three methods yield satisfying fits for small λ, with little overfitting. Increasing the noise leads to an increase in overfitting, as would be expected. As λ goes towards zero, the difference in R^2 scores between training and test data decreases for Ridge and Lasso. At the same time, however, the fit gets worse.

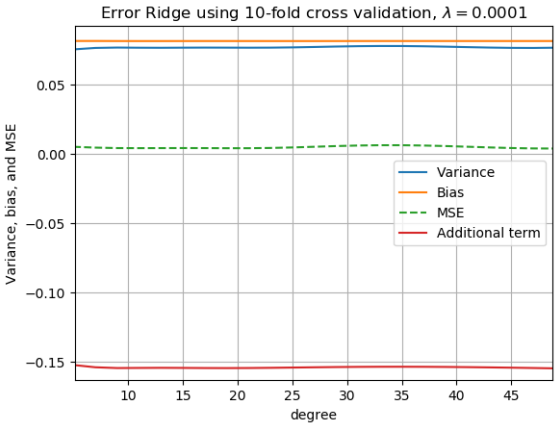
As the noise grows larger, Ridge and Lasso regression performs slightly better than ordinary least squares regression for certain values of λ. Still, none of the three models are very impressive when the noise is large.

In fig. 4a we see that the bias and the variance are both small and stable until about degree 25, where the variance begins to increase. We expected to see an increase in variance when the degree increased. This is probably due to overfitting. The rapid increase in variance may also be due to the fact that our design matrix is starting to get large and we may encounter problems when we try to invert it. We expected to see a larger bias for smaller degrees. We tested using smaller values of k, to check if this would bring out the bias-variance trade-off more clearly. It did not. We were surprised to see the small increase in bias that occurs around degree 50.

Figure 4b shows that the MSE using the Ridge method is small and stable throughout the plot. The rapidly increasing variance in the OLS-plot is not apparant here. As previously stated, we believe the variance increased due to overfitting or problems with the matrix inversion. Swapping to Ridge regression should fix both of these problems, which is what we can see has happened.



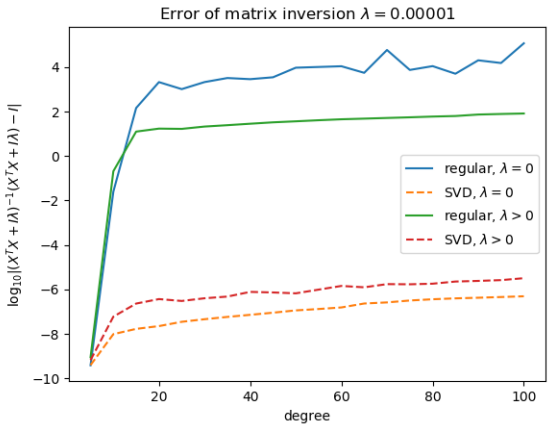
(a) Error of ordinary least squares model from 10-fold cross-validation



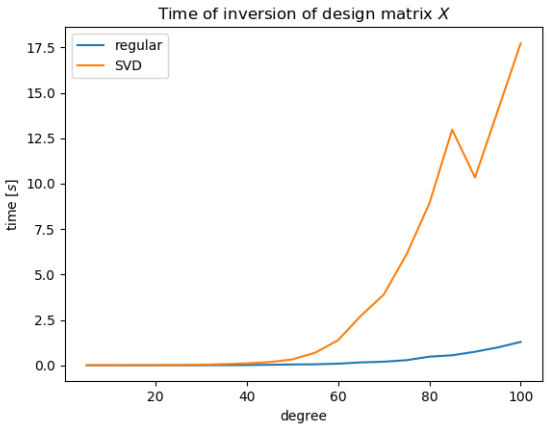
(b) Error Ridge model from 10-fold cross-validation

Figure 4: The error of both ordinary least squares and Ridge seems stable up to high degrees. Here $s = 0.05$. The bias and variance are balanced and we have little under and overfitting for all degrees considered.

Figure 5 summarizes the options we have considered when deciding on how to implement matrix inversion numerically for the matrix $\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{I}\boldsymbol{\lambda}$ for $\lambda \geq 0$ where $\boldsymbol{X}$ is the design matrix. This matrix needs to be inverted in order to find the coefficients $\boldsymbol{\beta}$ for a fit. As fig. 5a shows, the numerical error is considerably lower when using singular value decomposition when comparing with using the standard matrix inversion algorithm provided by scipy.linalg. This is true for both $\lambda = 0$ corresponding to ordinary least squares, and $\lambda > 0$ corresponding to Ridge and Lasso. The case $\lambda > 0$ achieves even higher numerical precision. As fig. 5b shows, the increased numerical accuracy does not come for free. We experience considerably longer run times using singular value decomposition.



(a) Numerical error vs degree



(b) Time of matrix inversions

Figure 5: A figure of numerical properties of matrix inversions versus degree. Here $\boldsymbol{X}$ is always the design matrix. Figure 5a shows the numerical error, while fig. 5b shows the elapsed time.

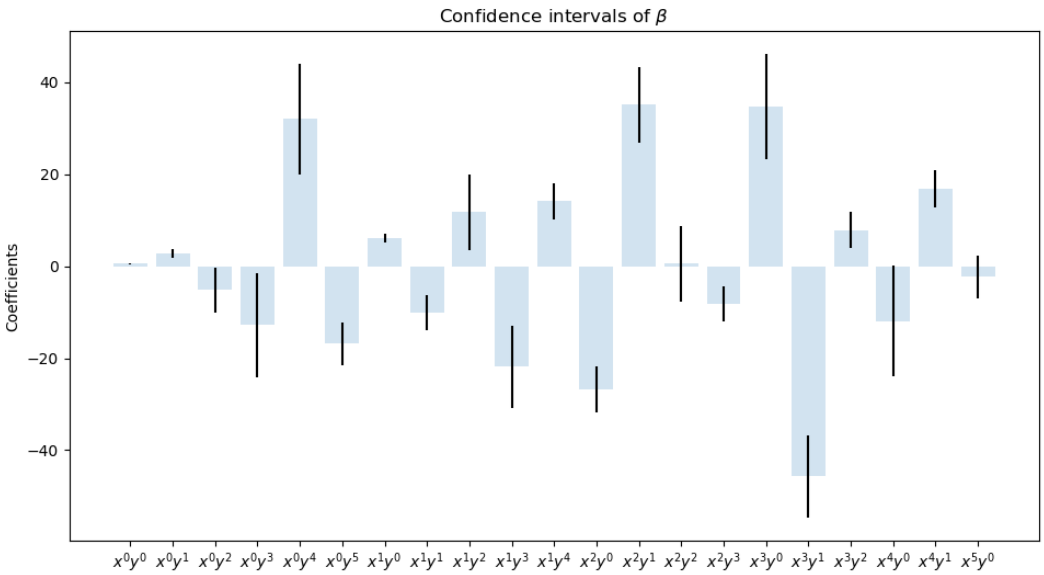


Figure 6: A plot showing the weights (blue bars) and confidence intervals (black lines) of $\boldsymbol{\beta}$ for the best ordinary least squares fit of degree 5 to $\text{data}_s(x, y)$ with $s = 0.05$. The confidence intervals were estimated using $[\mu - 2\sigma, \mu + 2\sigma]$ where μ and σ are found as in section 1.1.

Assuming that $\boldsymbol{\beta}$ is normally distributed, fig. 6 gives an estimate of the confidence intervals of $\boldsymbol{\beta}$, based on the assumption that we can approximate the variance σ^2 using the sample variance $\hat{\sigma}^2$. We compare this to fig. 7, where we estimate the same confidence intervals using bootstrapping. In the limit of large numbers, the latter estimate should approach theoretical confidence intervals. There is a clear difference in the variances resulting from these two estimates. Ideally these would have yielded more similar results. Since they do not, the pessimist might conclude that we can not say with certainty that either one of the two estimates yields correct results.

The former estimate is reliant on the approximation $\sigma^2 \approx \hat{\sigma}^2$. The latter estimate on the number of samples N, and is thus easier to improve, just by making N bigger. This, of course, eats up time, and we drew the line at $N = 10^6$.

An optimistic approach is to say that the true value of $\boldsymbol{\beta}$ is found with 95% certainty in the intersection of the two given estimates. This assumes that their intersections is nonempty. The intersection of the two estimates, will be very similar (if not equal) to the estimate in fig. 6. Therefore, increasing N, we are likely to end up resembling more and more the estimate in fig. 6.

Resampling allows us to estimate confidence intervals for Ridge and Lasso too. A similar plot as fig. 7 resulting from a Ridge regression can be found at:

https://github.com/CamillaBa/FYS-STK4155_Project_1/blob/master/confidence_interval_bootstrap_ridge.png

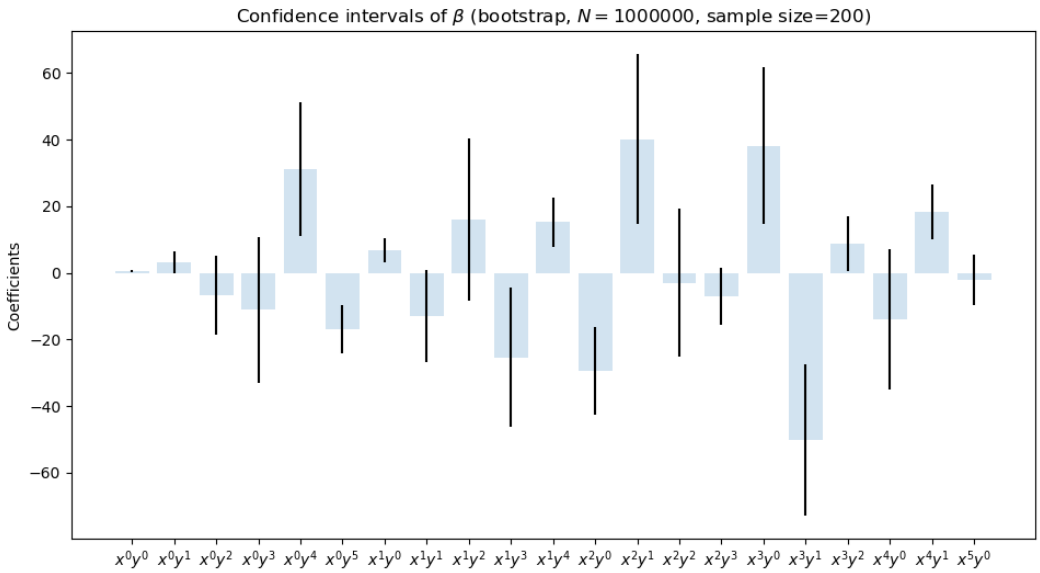


Figure 7: A plot showing the expected value of $E[\boldsymbol{\beta}]$ (blue bars) and confidence intervals (black lines) of $E[\boldsymbol{\beta}]$ for the best ordinary least squares fit of degree 5 to $\text{data}_s(x, y)$ with $s = 0.05$ using bootstrapping. The result is based 10^6 samples of samplesize 200 from our dataset. The confidence intervals were estimated using $[E[\boldsymbol{\beta}]-2\sigma, E[\boldsymbol{\beta}]+2\sigma]$ where σ is the standard deviation of $\boldsymbol{\beta}$.

3.2 Terrain data, small sample

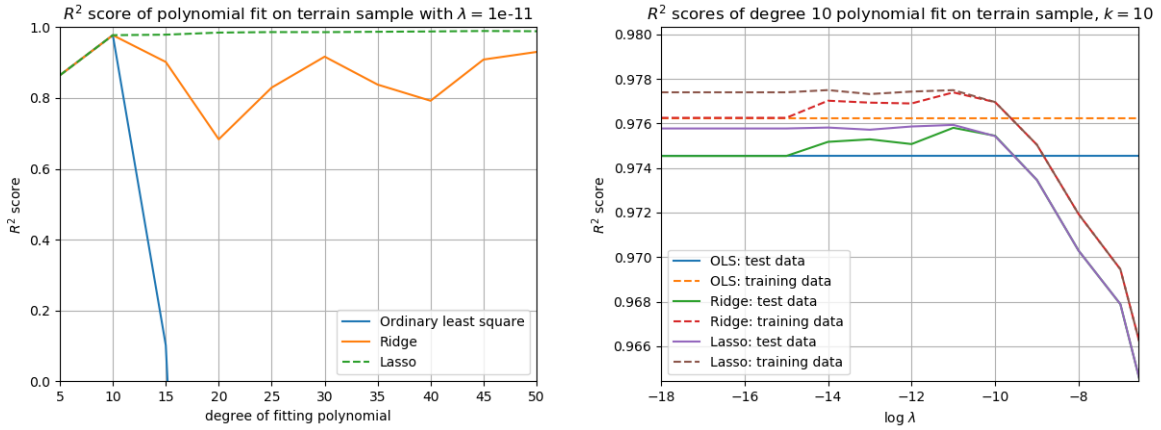
We consider first a small 50×50 square sample of the given (scaled) terrain data. The terrain data we used was the found one at:

https://github.com/CompPhysics/MachineLearning/blob/master/doc/Projects/2018/Project1/DataFiles/SRTM_data_Norway_1.tif

In fig. 8a see a massive decrease in R^2 score from degree 10 to degree 15 using the OLS method. We believe the matrix we try to invert is close to singular. The ridge method produces a model with an R^2 score varying with degree, from about 0.7 to 0.97. While this is an acceptable result, the Lasso clearly wins this round. It produces an R^2 score that only increases with degree as far as we can tell from this plot, and the score gets very close to 1.0. As we mentioned in section 1.2.2, adding λ to the diagonal of a singular matrix can make it invertible, which explains why the rapid decrease in R^2 score visible for the OLS method is not seen in the other two. We think this is the reason that $\lambda > 0$ performs better. Note that the inversion problem is independent of the given data that we study because we normalize the x and y coordinates.

For Franke's function, we always considered data of dimensions 20×20, and not 50×50, and did not run into this error. We knew from studying Franke's function that a grid size 20×20 did not produce a singular matrix for degree 10. Likely, we could have avoided running into a singularity entirely, had we chosen a different size of our terrain sample.

Resampling using cross validation shows that our model is fairly good at predicting, see fig. 8b. The differences between the R^2 scores for training data and test data are small for all three models. As is seen from the plot, Lasso regression yields a slightly better fit than Ridge and OLS. For certain values of λ, the R^2 score of the Ridge method is better than that of the OLS method.



(a) R^2 score vs complexity

(b) Resampling – the λ which yields best R^2 score

Figure 8: In (a) we see that the R^2 score of ordinary least squares suffers drastically after degree 10. Seemingly, we can increase the R^2 further using Ridge and Lasso past degree 10, but it takes a long time to compute these models. The best models on training data were good on testing data too, with R^2 scores close to 0.98.

Based on fig. 8a, we concluded the Lasso method was the way to go if we wanted the best fit for our data. From fig. 8b, we decided that $\lambda = 10^{-11}$ would give a good fit. The MSE of such a model is shown in fig. 9. Once again, the MSE, bias and variance are all small and stable.

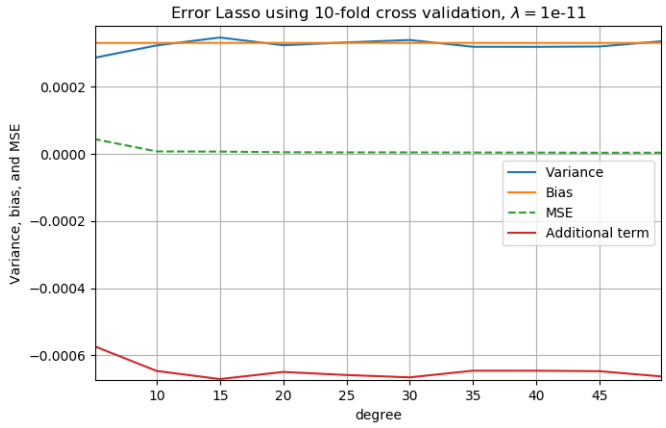
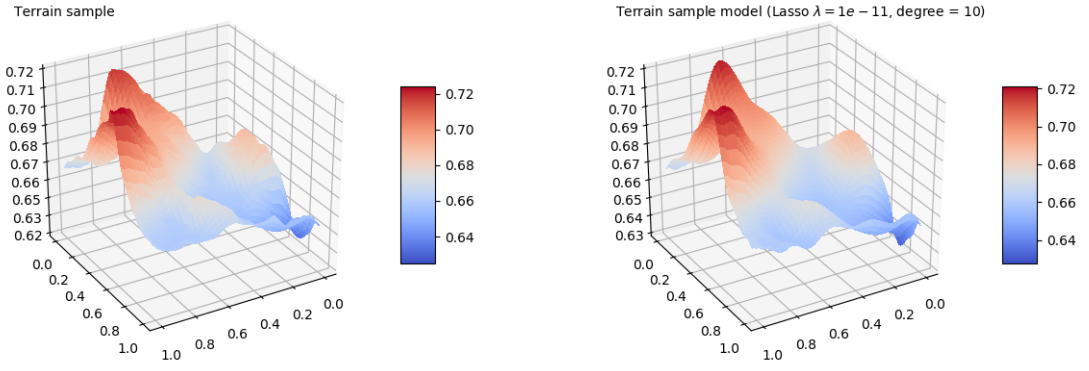


Figure 9: Mean squared error, and variance plot of Lasso regression of small sample of terrain data. Similar plots for Ridge and ordinary least square found at: https://github.com/CamillaBa/FYS-STK4155_Project_1/tree/master/small_terrain_sample_error



(a) 3D plot of terrain sample

(b) 3D plot of best model, R^2 score: 0.9875627808075547

Figure 10: A 3D plot of the terrain sample we used (a), versus the best model (b) that we found. Higher degrees likely result in even better fits but take a long time to calculate. We see visually that we fail to capture some of the finer details in ridges, despite having a good model.

Figure 10 shows a 3D plot of our final model next to a plot of the actual data set. As is apparent, both visually and from the R^2, we have found a good model.

3.3 Terrain data whole sample

The other idea we had was to find a good model on a low quality version of the terrain data. Instead of considering the terrain data set of size 1801×3601 we considered a low quality version of size 36×72, see fig. 11. The procedure that we used to transform our data to a low quality format is described in section 2.5. We used rectangle size 50×50.

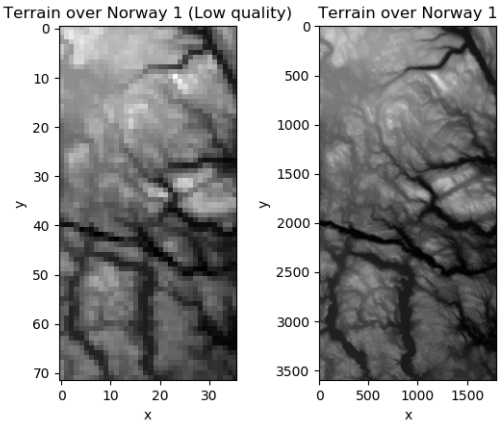
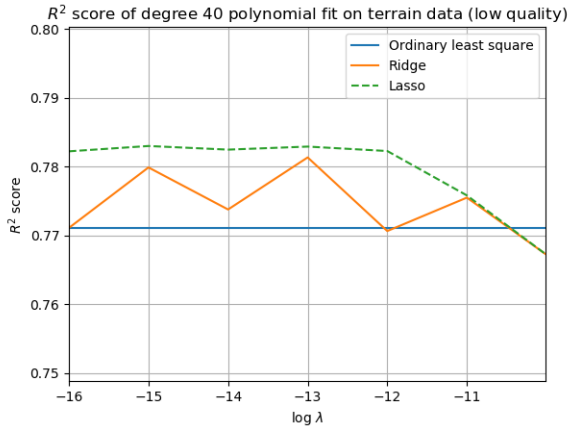


Figure 11: A 2D comparison (pictorially) of the low quality and the regular terrain data.

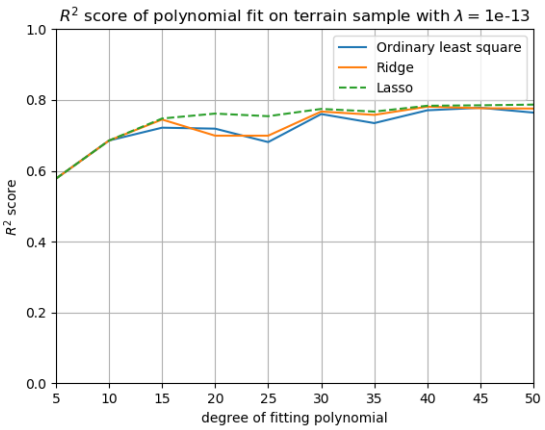
The R^2 score for fitting a polynomial of a given order d to the low quality terrain data is considerably worse than when fitting a polynomial to only small samples of it, see fig. 12a and fig. 12b. This is easy to

explain by the fact that if you cover a big area, then there will be many ups, downs, valleys, peaks, and so on. So you need a polynomial of high order to account for all of them, and finding such a polynomial is numerically expensive. We see in fig. 12b that, although slowly, the R^2 score increases with complexity.

Again the Lasso method yields the best fit, so although it is computationally expensive, we have decided to go with it. Figure 12a suggests a λ of size 10^{-13} will work fin. Figure 12a tells us that the higher the complexity, the better. This is supported by fig. 12b, which shows the MSE of the Lasso method. We can see that the variance somewhat increases with complexity, while the bias is stable.



(a) Finding the optimal λ. Moreover the best regression method seems to be Lasso.



(b) Plot of R^2 score against complexity.

Figure 12: In (a) we find the optimal λ and in (b) we check how the model performs with respect to R^2 scores.

We choose to draw the line at degree 40, as the algorithm is rather slow. Figure 13 shows our final model plotted alongside the low quality data. As can be seen, both from the plot and the R^2 score, the model is no perfect fit.

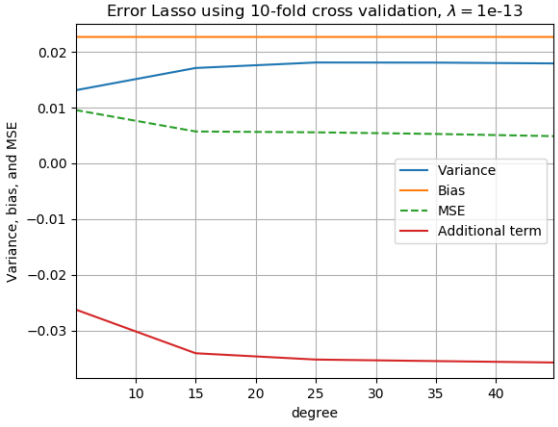
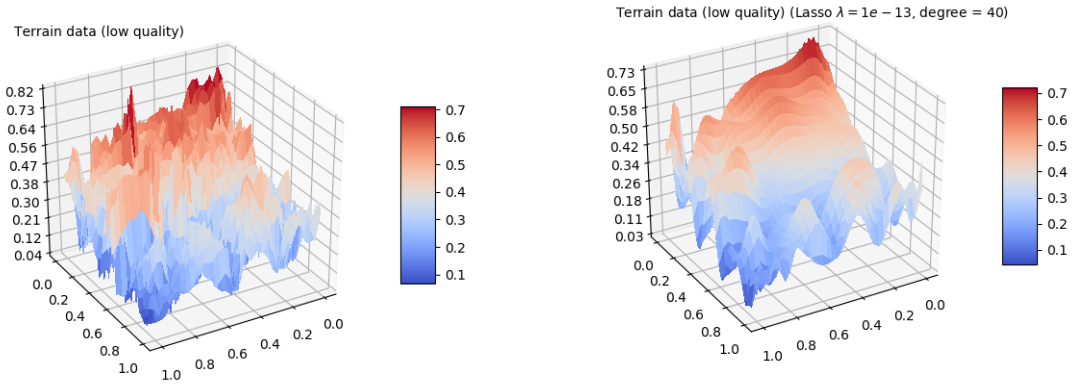


Figure 13: Mean squared error, variance and bias from 10-fold cross validation using Lasso regression.



(a) 3D plot of low quality data.

(b) Our best model, R^2 score: 0.7256172883326439. The optimal λ was based on fig. 12a

Figure 14

If you do not care about modeling numerous deep cracks and sudden peaks, then a fit to the whole map will suffice. However, if a precise description something you need, then fitting one polynomial to a big area might not be such a good idea.

We originally planned to extract the β obtained from the best model on low quality terrain, and to test this model on the high quality terrain. With our code, we had in mind to initialize an object of the class linreg with degree=40 and then used one of the class functions to find the regression model. However, when an object of linreg is initialized, the design matrix X for the given data and degree is calculated. In this case, the dimension was:

$$(3601 \times 1801) \times (\frac{41 \cdot 42}{2}) \approx 6 \cdot 10^{10} \text{ entries}$$

Each entry has the type double. If we assume the type double has size 16 bytes, to run this code, we would have needed ∼100 Gb of memory. Since we were short on time (and RAM) at this point, we decided to skip this part. Calculating the whole design matrix of size ∼100 Gb can be avoided by use of nested for loops, but we did not have time to implement this.

A much better approach, as the R^2 scores of fitting a polynomial to small segments of terrain data show, is to divide the map into segments, and then perform polynomial fits on each segment. If you are in the need of a precise model of your terrain data, you can then consider a piecewise description of the fits for each segment. This is also smart when considering RAM, because smaller design matrices need significantly less memomery than larger matrices.

A drawback of a piecewise description of your fit is that it will require precise book-keeping to keep track of the transitions from one segment of the map to another. We have not looked into this further, but we propose this as a way to get precise, numerically feasible models of terrain data, at the cost of extra bookkeeping.

4 Conclusion

We were successful in finding good models of Franke's function, and on small samples of our terrain data. Our best fit on the small sample resulted in an R^2 score of 0.9875627808075547, and it made good predictions. Lasso outperformed ordinary least squares and Ridge when fitting to real data, but all methods yielded good results on Franke's function. Our implementation of the shooting algorithm converged slowly. Had we seen that ordinary least square and/or ridge produced as good models, then we likely would not have have recommended Lasso due to the time it takes.

We were unsuccessful insofar that we experience difficulty in creating good models of big data samples. The difficulties arise partly due to the size of the objects and memory they demand to be handled numerically. To get a better chance, we tried to convert the terrain data to low quality. This helped us somewhat in that we were actually able to carry out the computations. However, the resulting model turned out to be a poorer than, say, for the smaller sample. Our best model had an R^2 score of 0.7256172883326439 against the low quality version of the data. We were never able to compare it to the whole data sample.

Since we did well on a small sample of the whole data, we conclude that fitting data piecewise can be more efficient when dealing with big data samples. If we ever are to build upon this code, we would definitely like to try writing a program that sorts out how to glue piecewise descriptions together in an efficient manner.

References

[Fra79] Richard Franke. *A critical comparison of some methods for interpolation of scattered data*. Tech. rep. Monterey, California: Naval Postgraduate School., 1979.

[Jam+13] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[Meh+18] Pankaj Mehta et al. "A high-bias, low-variance introduction to machine learning for physicists". In: *arXiv preprint arXiv:1803.08823* (2018).

[Rob14] Christian Robert. *Machine learning, a probabilistic perspective*. 2014.

[Wie15] Wessel N van Wieringen. "Lecture notes on ridge regression". In: *arXiv preprint arXiv:1509.09169* (2015).