Regression analysis on Norwegian terrain map with Franke's function as test function

Aram Salihi[1], Martine Tan [2] & Andras Filip Plaian [2]

[1]Department of Informatics,University of Oslo, N-0316 Oslo, Norway

Department of Mathematics[2], University of Oslo, N-0316 Oslo, Norway

October 9, 2019

Abstract

In this numerical study we have used three different linear regression methods to study the polynomial fit of the two dimensional Franke's function. The standard ordinary least squared (OLS), Ridge and LASSO regression has been used. We have used the cross validation to further assess if the model is correct. Further we will apply these methods on real data of a norwegian terrain map downloaded from `https://earthexplorer.usgs.gov/` to see which method give best reproduction result of the terrain map. The produced result are: OLS giving a best minimum MSE of 3.2280 and R2 score of 0.9965, Ridge giving best minimum MSE of 2.4446 and R2 score 0.9973, and lastly LASSO with best minimum MSE of 73.3140 and R2 score of 0.9206. From this we can conclude that the ridge method gives the best result for this terrain map.

Contents

1 Introduction

The aim of this numerical study is to experiment with three different regression methods in detail: Linear regression, ridge regression, and lasso regression. In addition, cross validation will be used as a resampling technique to further evaluate the regression methods. The project largely consists of two parts. The first part comprises of implementing the regression methods and relevant algorithms in python. We will use Franke's function (see section 2.1) to generate a data set upon which we can apply and test our code during the developing process. We will also discuss in detail the bias-variance trade-off in the regression methods.In the second part of the our numerical study, we will apply this code to digital terrain data from USGS. Digital terrain data can be downloaded from `https://earthexplorer.usgs.gov/`.

In the following section we will present a small introduction to the theory behind the different regression methods and then discuss the result produced.

2 Theory

2.1 Franke's function

Franke's function is a two dimensional Gaussian function with two peaks. This function is widely used as a test function in interpolation and fitting problems.

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

this function is defined in the interval $x, y \in [0,1]$. In this numerical study, we will use this function to test which regression method is the most optimal for this function.

2.2 Linear regression and matrix notation

Consider an experiment with a set of datapoints $\mathbf{z} = \{z_0 ... z_{n-1}\}$ observed, Further we want to model these observations as a function of $\mathbf{x} = \{x_0 ... x_{n-1}\}$ and $\mathbf{y} = \{y_0 ... y_{n-1}\}$. Keep in mind that, when modeling datapoints one has to be perpared that error can arise in our approximation. We will now assume that our response variable $z(x,y)$ is a smooth function and thus can be parameterized as a two dimensional polynomial of x and y

$$z_i = z(x_i, y_i) = \tilde{z}_i + \epsilon_i \tag{1}$$

Where $\tilde{z}$ is the estimator of our response variable z. Expanding this, we will obtain a set of equation

$$\begin{aligned} &z_0 = \epsilon_0 + \beta_0 + \beta_1 x_0 + \beta_2 y_0 + \beta_3 x_0^2 + \beta_4 y_0 x_0 + \beta_5 y_0^2 + ... + \beta_j x_0^k y_0^{k-d} \quad . \\ &. \\ &. \\ &. \\ &z_{n-1} = \epsilon_{n-1} + \beta_0 + \beta_1 x_{n-1} + \beta_2 y_{n-1} + \beta_3 x_{n-1}^2 + \beta_4 y_{n-1} x_{n-1} + \beta_5 y_{n-1}^2 + ... + \beta_j x_{n-1}^k y_{n-1}^{k-d} \end{aligned}$$

Where βj is the unknown weights we wish to find in order to fit the datapoints z. In this case there exist in total $\sum_{t=0}^{d-1}(i+1)$ of β's, and k goes from $k = d, ..., 0$, where d is order of the polynomial we want to use. Notice that this set of linear equation can be expressed in vector-matrix notation on the form:

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2}$$

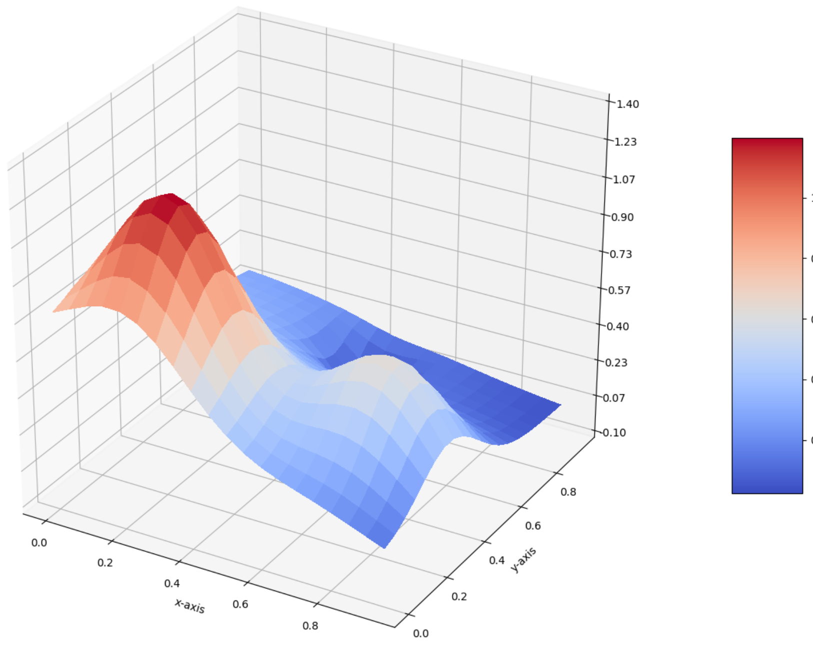


Figure 1: Franke's function

Where X is a design matrix with dimension $(n-1) \times d$, which looks like

$$X = \begin{pmatrix} 1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 & . & . & x_0^k y_0^{k-d} \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1} y_{n-1} & y_{n-1}^2 & . & . & x_{n-1}^k y_{n-1}^{k-d} \end{pmatrix} \tag{3}$$

2.3 Ordinary least square (OLS)

Quite often will we stumble upon problems, where the number of unknowns is greater than known. Due to this problem we cannot invert the design matrix (since the matrix is not a square matrix) to solve the matrix equation, we will therefore consider methods where we want to minimize the distance between the datapoints and the approximated points. I.e

$$|\mathbf{y} - \tilde{\mathbf{y}}| \tag{4}$$

we will therefore consider something called a cost function $C(\beta)$ which explains this distance (for the sake of simplicity we will assume that response and estimator variable are one dimensional)

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n}(z-\tilde{z})^2 = \frac{1}{n}\left(\mathbf{y}-\tilde{\mathbf{y}}\right)^T\left(\mathbf{y}-\tilde{\mathbf{y}}\right) \tag{5}$$

where the estimator variable is $\tilde{\mathbf{z}} = X\boldsymbol{\beta}$. We now want to find $\boldsymbol{\beta}$ which minimizes the cost function. In order to this consider the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d}\left\{\frac{1}{n}\left(\mathbf{y}-\tilde{\mathbf{y}}\right)^T\left(\mathbf{y}-\tilde{\mathbf{y}}\right)\right\} = 0 \tag{6}$$

For the sake of simplicity we can also express this as

$$\frac{\partial C(\beta)}{\partial\beta_j} = \frac{\partial}{\partial\beta_j}\left[\frac{1}{n}\sum_{i=0}^{n-1}\left(y_i-\beta_0x_{i,0}-\beta_1x_{i,1}-.......-\beta_{n-1}x_{i,n-1}\right)^2\right] = 0 \tag{7}$$

Performing this partial derivative we will obtain the following

$$\frac{\partial C(\beta)}{\partial\beta_j} = -\frac{2}{n}\left[\frac{1}{n}\sum_{i=0}^{n-1}x_{ij}\left(y_i-\beta_0x_{i,0}-\beta_1x_{i,1}-.......-\beta_{n-1}x_{i,n-1}\right)\right] = 0 \tag{8}$$

This can be beautifully expressed in matrix notation as

$$X^T\left(\mathbf{y}-X^TX\boldsymbol{\beta}\right) = 0 \tag{9}$$

Notice that that this equation can be simply be written as:

$$X^TX\boldsymbol{\beta} = X^T\mathbf{y} \tag{10}$$

This is the so called normal equation. Now notice that X^TX is a square matrix. If this square matrix is a non-singular matrix, if and only if all the coloumn vectors are linearly independent there will exist a invertible matrix such that:

$$\boldsymbol{\beta} = (X^TX)^{-1}X^T\mathbf{y} \tag{11}$$

Quite often inverting a matrix is quite tedious and require alot more operations, but it do exist algorithm which solves this equation more efficient, such as SVD (singular value decompistion), Cholesky factorization or QR method. In our case we have used numpy's PINV function to invert this matrix, but this function uses SVD as algorithm. The beauty of this numpy function is that it handles problems with singularities perfectly. Consider a case where X^TX contains singular values, in order to remove this values we will add some small value λ

$$X^TX \approx X^TX + \lambda\mathbb{I} \tag{12}$$

The numpy function will already take care of this by doing this. Keep in mind that OLS can be generalized to higher dimensions.

2.4 Ridge regression

.In comparison with the standard OLS method the Ridge regression will actually take care of the singularities if X^TX is singular. Recall the cost function from the previous section (5)

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left(\mathbf{y}-X\boldsymbol{\beta}\right)^T\left(\mathbf{y}-X\boldsymbol{\beta}\right) \tag{13}$$

Which is the MSE for the OLS method. From previous section we mentioned that that the matrix product $X^T X$ could be a singular matrix, thus not invertible. We therefore wish to add a regularization parameter λ by penalizing the cost function.

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left(\mathbf{y} - X\boldsymbol{\beta}\right)^T\left(\mathbf{y} - X\boldsymbol{\beta}\right) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{14}$$

We now wish to shrink the distance between the response and estimator thus finding a $\boldsymbol{\beta}$ which satisifies this condition. Consider the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{n}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\beta||_2^2 \tag{15}$$

Solving this equation we can express the expression above as

$$X^T y = \beta^{ridge}(X^T X + n\lambda\mathbb{I}) \tag{16}$$

We will further define $n\lambda = \lambda$. Recall from previous that the regularization parameter took care of the singularities, thus $(X^T X + \lambda\mathbb{I})$ should be invertible since the new matrix should contain $n-1$ linearly independent column vectors. This can be proven by using SVD decomposition on this matrix. Thus, the solution to $\boldsymbol{\beta}$

$$\beta^{ridge} = (X^T X + \lambda\mathbb{I})^{-1}X^T y \tag{17}$$

This is the famous ridge regression method. As previously mentioned, the invertion of matrix is quite tedious, and thus SVD decomposition is the most favourable method to use for solving this type of equation. Keep in mind if the matrix $X^T X$ is non-singular (thus orthogonal), the ridge regression will minimize the coefficients in $\boldsymbol{\beta}$, but does not enforce them to be zero. With this property, ridge regression will avoid some overfitting problems which OLS may stumble on. For more information and theory about ridge regression please see [McDonald, 2009]

2.5 LASSO regression

The idea behind LASSO is to shrink the data towards a central point. Thus shriking the β to a point where some of the coefficients are zero. What this does compared to ridge regression is that, not only does it reduce overfitting but also generating a more simpler model, due to fewer coefficients in the polynomial fit.

Mathematically, this is done by adding a λ term to minimized cost function. Instead of taking the l2 norm of $\boldsymbol{\beta}$ as in ridge regression (the normal euclidian norm), we are doing a l1 norm (taking the absolute value) on $\boldsymbol{\beta}$. Thus we want to minimize the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{n}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\beta||_1 \tag{18}$$

Where the l1 norm is $||\beta||_1 = \sum_{i=0}^{n-1}|\beta_i|$. We now perfomed an absolute shrinkage to the coefficients we wish to find. In comparison to ridge regression, we see that this method shrink some of the coefficients to zero, again this will improve the quality of the fitting and reduce overfitting. For further information and deeper theory about this method, please see [Wil, 1996] or [Hastie et al., 2001]

2.6 Bias-variance decomposition

Consider a data set $\mathcal{L}$ consisting of data $X_{\mathcal{L}} = \{(y_j, \mathbf{x}_j\}$ for $j = 0, ...n-1\}$. We will now assume that the true data is generated from the noisy model

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\epsilon}$$

Where $\boldsymbol{\epsilon}$ is the normally distributed noise with zero mean and standard divation σ^2. For the different regression method we defined a approximation to the function $\mathbf{f}(\mathbf{x})$ in terms of $\boldsymbol{\beta}$ and a design matrix X, and then minimize the cost function for finding a optimal $\boldsymbol{\beta}$. Keep in mind if $\lambda = 0$, we just have the standard OLS regression. For the sake of simplicity we set $\lambda = 0$. Consider the following

$$C(X, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{19}$$

Which also can be written as

$$C(X, \boldsymbol{\beta}) = \mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] \tag{20}$$

In order to get a better understanding of the theoretical learning method used, we need to decompose these expression. The goal of the decomposition is to express this as a sum of bias, variance and the standard deviation. The bias tells us how good the model is and the variance tells us how sensitive it is when changes are made. Let us consider the expectation value of the response and estimator vector given in expression (20). We now wish to substitute inside $\mathbf{y} = \mathbf{f} + \boldsymbol{\epsilon}$, and add and subtract the expectation value of the estimator variable $\mathbb{E}[\tilde{\mathbf{y}}]$.

$$\mathbb{E}\left[(\mathbf{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}])^2\right] = \sum_i \mathbb{E}\left[((\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}]) + \boldsymbol{\epsilon} + (\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}))^2\right]$$

Before we proceed with the derivation, we have to define certain quantities first. The expectation value of the error (not squared) is $\mathbb{E}[\boldsymbol{\epsilon}] = 0$, further the expectation value of the function $\mathbf{f}$ is the scalar value of the function, thus $\mathbb{E}[\mathbf{f}] = f$. Using these, we will obtain the following when performing the multiplication

$$\mathbb{E}\left[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])\right] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})] + 2\underbrace{\mathbb{E}[\boldsymbol{\epsilon}\,(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})]}_{=0} + 2\underbrace{\mathbb{E}\left[\boldsymbol{\epsilon}\,(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])\right]}_{=0} + \underbrace{\mathbb{E}\left[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])\,(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\right]}_{=0}$$

From this we are left with

$$(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] \tag{21}$$

Recall the variance of the response variable

$$\mathrm{VAR}(\mathbf{f} + \boldsymbol{\epsilon}) = \mathbb{E}[(\mathbf{f} + \boldsymbol{\epsilon})^2] - \mathbb{E}[\mathbf{f} + \boldsymbol{\epsilon}]^2 = \mathbb{E}[\mathbf{y}]^2 + 2\underbrace{\mathbb{E}[\mathbf{f}]\mathbb{E}[\boldsymbol{\epsilon}]}_{=0} + \mathbb{E}[\boldsymbol{\epsilon}^2] + \left(\underbrace{\mathbb{E}[\boldsymbol{\epsilon}]}_{=0} + \mathbb{E}[\mathbf{f}]\right)^2 \tag{22}$$

From this we can see that the variance of the response variable can be expressed as the standard deviation squared

$$\mathrm{VAR}(\mathbf{f} + \boldsymbol{\epsilon}) = \mathbb{E}[\boldsymbol{\epsilon}^2] = \sigma^2 \tag{23}$$

The bias is defined as $(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])^2 = \mathrm{bias}(\tilde{\mathbf{y}})^2$. Keep in mind that we can rewrite the expression

$$\mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] = \mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}^2]\right] - 2\mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}\left[\tilde{\mathbf{y}}\right]^2 = \mathbb{E}[\tilde{\mathbf{y}}^2] - \mathbb{E}[\tilde{\mathbf{y}}]^2 = \mathrm{VAR}(\tilde{\mathbf{y}}) \tag{24}$$

discretizing the variance expression, we can express the expression above as

$$\mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] = \mathrm{VAR}(\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 \tag{25}$$

We now want to substitute (25) and (23) into expression (21). We will then obtain the following result

$$\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] = \sigma^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i}^{n-1}(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 \tag{26}$$

3 Method/Implementation

In this following sections we will describes the implementation of the different linear regression used and the creation of the design matrix. Keep in mind that these are implementation in python3, used different libraries such as numpy, scipy and scikitlearn.

3.1 Creating the data

To create a suitable data set with n observations (x, y, z), where $z = f(x, y)$, we need a vector $\mathbf{x}$ and a vector $\mathbf{y}$ with x-elements and y-elements respectively, such that there are n possible ways to combine the elements in a tuple (x, y). For our purposes, $n = 400$ is sufficient. Thus, we created $\mathbf{x}$ and $\mathbf{y}$ vectors with 20 uniformly spaced numbers in the interval $[0, 1]$ each, and then applied Franke's function to each tuple (x, y) to obtain $20^2 = 400$ z-values. To simulate datapoints gathered from real-life situations, we added a noise element ϵ to the z-values. Each noise element is randomly selected from the normal distribution with mean 0 and variance 0.25.

```
k = 20                          # Number of points on each axis
x = np.arange(0, 1, 1/k)        # Numbers on x-axis
y = np.arange(0, 1, 1/k)        # Numbers on y-axis
x, y = np.meshgrid(x,y)       # Create meshgrid of x and y axes
z = FrankeFunction(x,y)               # The z-values

x1 = np.ravel(x)                # Flatten to vector
y1 = np.ravel(y)
z1 = np.ravel(z) + np.random.normal(0, .25, n)      # Add noise if wanted
n = len(x1)                     # Number of observations (n=k*k)
```

3.2 Design matrix

The dimensions $n \times p$ of the design matrix X is determined by number of datapoints n and the degree of the polynomial $p(x, y)$ that estimates the true function. If the degree of p is d, then

$$p(x, y) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \beta_6 x^3 + \beta_7 x^2 y + \beta_8 xy^2 + \beta_9 y^3 ...$$

Thus the number of betas (and consequently the number of columns in the design matrix X) is $\sum_{i=0}^{d}(i + 1)$. The code snippet below shows how the design matrix is constructed in our code.

```
def CreateDesignMatrix_X(x, y, d = 2):
    """
    Input is x and y mesh or raveled mesh, keyword agruments d is 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((d+1)*(d+2)/2)              # Number of elements in beta
    X = np.ones((N,l))

    for i in range(1,d+1):
        q = int((i)*(i+1)/2)
        for k in range(i+1):
            X[:,q+k] = x**(i-k)*(y**k)
    return X
```

3.3 OLS

Consider the expression (16) from theory section (2.3)

$$X^T X\boldsymbol{\beta} = X^T y$$

We need invert $X^T X$ in order to find the least squares estimate of β. To invert the matrix $X^T X$ we used numpy's function pinv (numpy.linalg.pinv) which stands for pseudo inverse. This function add a small number λ on its diagonal to make the matrix non-singular. Then it uses numpy SVD decomposition function to invert the matrix. The following python code-snippet shows how we calculate β, and multiply it with the design matrix in order to obtain prediction variable $\hat{\mathbf{z}}$.

```
beta = np.linalg.inv(X.T @ X) @ X.T @ z1
z_pred = X @ beta                        # Predicted z values
```

Finding confidence intervals for the regression coefficients β

For finding the variances of the regression coefficients, we need

$$\hat{\sigma}^2 = s^2 = \frac{1}{n-p-1}\sum_{i=0}^{n-1}(z_i - \hat{z}_i)^2, \tag{27}$$

where p is the number of regression coefficients β, and n is the number of observations. The covariance matrix is obtained by multiplying

$$COV = s^2(X^T X)^{-1}. \tag{28}$$

The variance of the β estimators lies on the diagonal of this covariance matrix. Thus, for each β_i, we can calculate a 95% confidence interval:

$$\left[\beta_i - 1.96\sqrt{\frac{Var(\beta_i)}{n}}, \beta_i + 1.96\sqrt{\frac{Var(\beta_i)}{n}}\right]$$

These calculations are summarized in the code below which finds the CI for each beta, and saves them in a list:

```
s2 = (1/(n-p-1))*np.sum((z1 - z_pred)**2)
cov_beta = s2*np.linalg.inv(X.T.dot(X))     # Covariance matrix
beta_var = np.diag(cov_beta)                # Variances of the betas
beta_CIs = []              # List to contain the confidence intervals

for i in range(p):
    beta_CIs.append([beta[i]-1.96*np.sqrt(beta_var[i]/n),
                    beta[i]+1.96*np.sqrt(beta_var[i]/n)])
```

3.4 Ridge regression

The implementation of ridge regression is very similar to that of linear regression. The difference lies in how β is calculated:

```
XTX = X.T @ X
dim = len(XTX)
W = np.linalg.pinv(XTX + lmb*np.identity(dim))
beta = W @ X.T @ z1
z_pred = X @ beta
```

The variable *lmb* is the ridge hyperparameter λ.

Finding the confidence intervals for the regression coefficients β

This is again similar to how we calculated the CIs for the linear regression βs, but we need a different formula to find the covariance matrix:

$$COV = s^2\big((X^TX+\lambda I)^{-1}\big)X^TX\Big(\big((X^TX+\lambda I)^{-1}\big)\Big)^T \tag{29}$$

The following code snippet calculates the CIs. Note that the variables W and XTX are defined as in the code snippet above.

```
s2 = (1/(n-p-1))*np.sum((z1 - z_pred)**2)
cov_beta = s2*(W @ XTX @ W.T)                    # Covariance matrix
beta_var = np.diag(cov_beta)                     # Variances of the betas
beta_CIs = []              # List to contain the confidence intervals

for i in range(p):
    beta_CIs.append([beta[i]-1.96*np.sqrt(beta_var[i]/n),
                     beta[i]+1.96*np.sqrt(beta_var[i]/n)])
```

3.5 LASSO regression

In this numerical study we have not implemented the LASSO algorithm, but used scikit-learn's LASSO regression function. For more information on this function please vitit the documentation website `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html`.

Scikit's Lasso function is used on a degin matrix X and response vairable z like this:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=opt_lambda, fit_intercept=False, tol=0.001, max_iter=10
    e6)
lasso.fit(X, z)
z_pred = lasso.predict(X)
```

3.6 Cross Validation

Our cross validation implementation consists of two functions. One returns k lists of indexes for the each of the k test-sets, given the number n of observations. This is used to split the data into a test set and a training set, for each of the k folds. The other function performs the actual cross validation.

The first function *k_folds(n, k)* (see own_code.py on the github page) takes arguments n and k, which are the number of observations to be split and the number of "folds" they are to be split into, respectively. The function makes a vector *indexes* containing the numbers from 0 to $n-1$, shuffles it, and splits it into k sections of size n/k. If n is not divisible by k, the remainders are distributed to the sections such that some sections have $n/k+1$ indexes, while the rest have n/k.

The second function *k_Cross_Validation(x, y, z, k, d, reg_method, lmb=None)* (see own_code.py on the github page) performs cross validation using the specified regression method (linear, ridge, or lasso) by splitting the data into k folds using the *k_folds()* function. For each list of indexes that is returned by the *k_folds()* function, the test-observations are the observations corresponding to the indexes, while the remaining observations are used for training the regression model. The model is then evaluated by fitting to the test-data, and calculating the test-error (i.e. the MSE) and R2-score. Additionally, the training error is evaluated by fitting the model to the training data, and calculating the MSE. The function then calculates and returns the means for the test error, train error and R2-score for the k folds.

4 Results

4.1 OLS

Using linear regression with polynomial degree of $d = 5$ on the noisy data gives us the following plot. The MSE is 0.058005, and the R2-score is 0.590902. For the 21 β-values, we have calculated

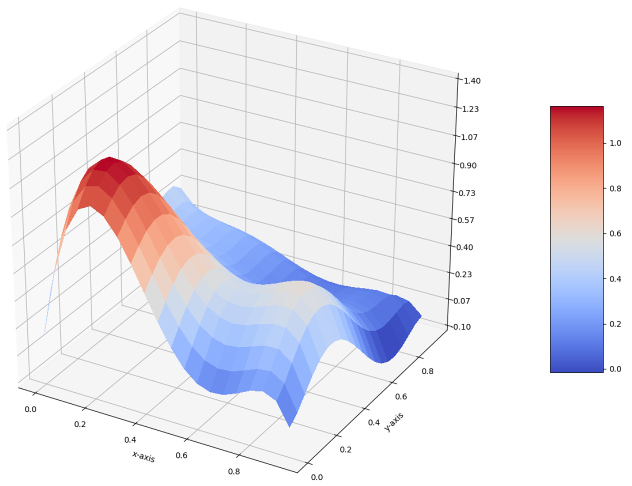


Figure 2: Estimate of Franke's function using linear regression with polynomial degree 5

95% confidence intervals which can be seen in table in figure 3. The plot in the same figure shows the varying span of the confidence intervals, which seem to be larger for the middle βs.

How reliable are these results? To better utilize our available data, we split the data into training and test sets, with a 4 to 1 split. This allows us to test the model on 'new' data, which was not used to train the model. The resulting plot can be seen in figure (4). The plot is much courser because it consists of only 1/5-th of the data points. However, the more interesting change to note, is that the MSE has increased to 0.059168, and the R2-score decreased to 0.561730. This is expected, because fitting and testing the regression model on the same data as we did previously, will overestimate how well the model performs. Thus, this slightly higher MSE is a better reflection of the actual performance of the model. To further improve the accuracy of the MSE of the model, we use cross validation. Using 5 folds, and the same degree 5 on our CV function *k_ Cross_ Validation()* (see own_code.py on our github), we obtain the new MSE score 0.066528 and R2-score 0.514278. We now see that the linear model is not as accurate as we thought.

j	β_j	n_{lower}	n_{upper}
0	0.1922	0.1778	0.2066
1	9.3952	9.2117	9.578
2	5.0876	4.9041	5.271
3	−30.1864+	−31.1657	−29.2
4	−29.6891	−30.4331	−28.94
5	−9.3766	−10.3561	−8.39
6	16.3307	13.9573	18.70
7	79.3070	77.6050	81.00
8	40.3013	38.5993	42.00
9	−15.1321	−17.5056	−12.7
10	26.0331	23.3978	28.66
11	−80.7320	−82.6814	−78.7
12	−40.3222	−42.1148	−38.5
13	−40.12228	−42.0722	−38.1
14	44.5923	41.9571	47.22
15	−22.0064	−23.0950	−20.91
16	26.1073	25.1888	27.02
17	21.3900	20.5073	22.27
18	6.1052	5.2225	6.988
19	17.9456	17.0271	18.8642
20	−25.3622	−26.4508	−24.2737

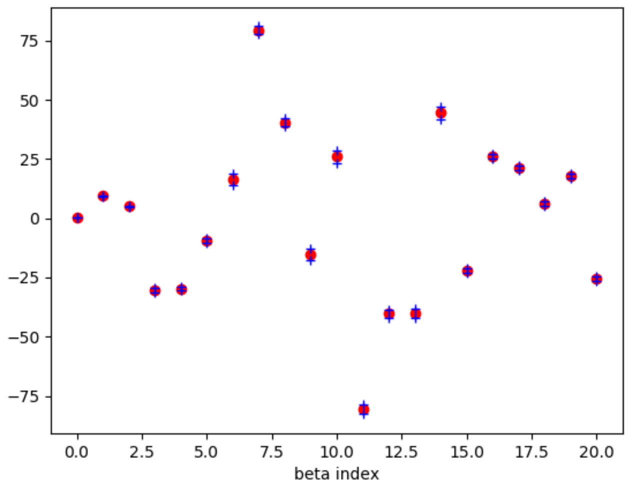


Figure 3: To the left, a table of the estimates of the 21 regression coefficients β from linear regression with degree $= 5$, and their respective 95% confidence intervals $[n_{lower}, n_{upper}]$. To the right, a plot of the same regression coefficients, where the 95% confidence intervals are shown in blue.

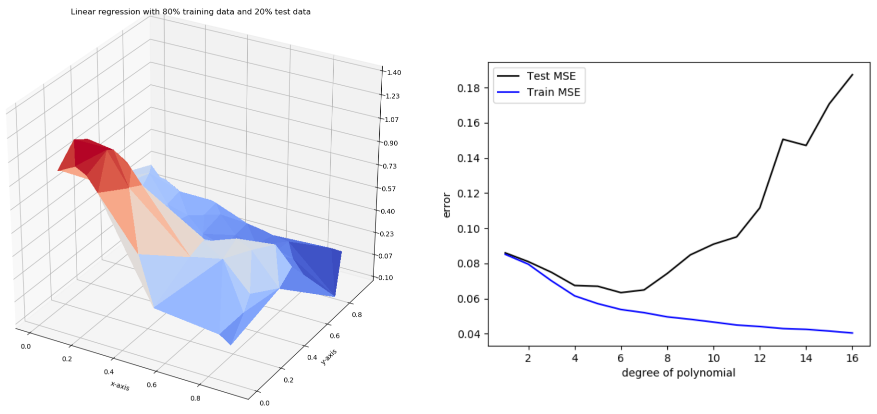


Figure 4: On the left we see Linear regression estimate of Franke's function on 20% test data and 80% train data. On the right we see training error and test error plotted against polynomial degree (Linear regression).

How do we choose the best degree d for the model? Doing cross validation with $k = 5$ folds for $d = 1, 2, ..., 20$ allows us to compare the MSE's and find which d gives the lowest MSE. The plot in figure **??** shows the training error and test error for different degrees. As expected, the training error descends and goes to zero as the degree increases, because it is the result of using the model to predict values on the *same set* as what the model was based on. The test-error gives a better indication of the performance of the linear model as the degree increases. We see a decrease in MSE from $d = 1$ to $d = 6$, but a sharp increase after. This shows that larger degrees results in *over-fitting*. In section 2.6 we showed that MSE can be written as the sum of the variance and squared bias, in addition to the variance of the noise element, σ^2. The bias^2 and the variance are in fact inversely related. The squared bias decreases with the polynomial degree, as the model fits closer and closer to the data points upon which it is trained. Meanwhile, the variance will increase with the polynomial degree, as decreasing the bias makes the model receptive to the influence of random fluctuations present in the noisy data. The result is what is famously called the Bias-Variance Trade-Off. It means that we must find a trade-off between decreasing the bias, and increasing the variance, such that the MSE is minimized.

As the plot in figure 4 indicates that $d = 6$ is the optimal degree for this trade-off, we once again use cross validation to evaluate the performance of the model, now of degree 6. With $k = 5$ folds, the results are MSE$= 0.064365$ and R2 score$= 0.538084$. This is a slight improvement from when we used degree 5.

4.2 The Bias-Variance Trade-off

4.3 Ridge Regression

For ridge regression, we also need to find the optimal degree d for the model, as well as the optimal value for the hyper parameter λ. Using the same approach as for linear regression, we perform cross validation with ridge regression, where the degree values go from 1 to 15, and the λs range in the interval $[10^{-5}, 0]$. The results are represented in the colormap in figure 5. The lowest MSE-scores have a deep red color, which we see appear middle-left of the plot. The minimum MSE came at 0.062061, with $\lambda = 0.000015$ and degree 7.

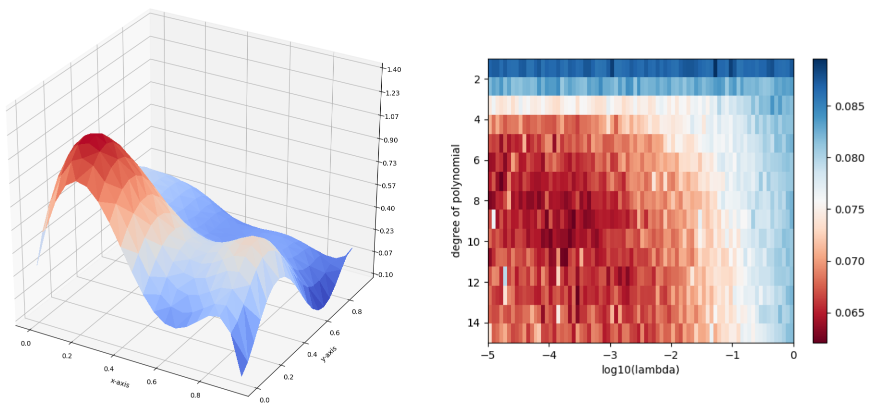


Figure 5: On the left we have applied ridge regression on Franke's function. Further, on the right we can see the color map showing how the MSE varies for various choices of λ and the degree d of the polynomial in ridge regression.

To compare with the results from linear regression, we performed ridge regression on the whole data set with the parameters set to $\lambda = 0.000015$ and $d = 7$. This results in 36 regression coefficients, whose estimates and confidence intervals can be seen in table (1) in the appendix. The MSE score is 0.0555119 and the R2 0.608487. The resulting plot can be seen in figure 6

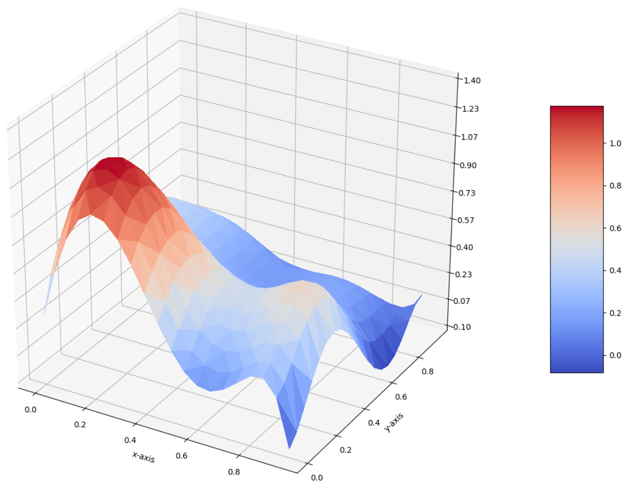


Figure 6: Ridge regression estimation of Franke's function with parameters $d = 7$ and $\lambda = 0.000015$

4.4 Lasso Regression

We repeat the process for Lasso regression.

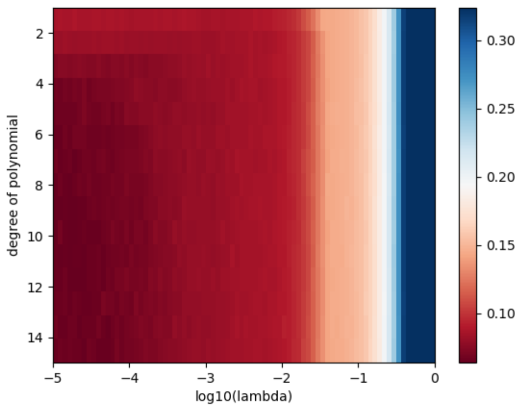


Figure 7: Colormap showing how the MSE varies for various choices of λ and the degree d of the polynomial in lasso regression.

The lowest MSE 0.063676 is achieved at $\lambda = 1.548365 \times 10^{-5}$ and $d = 9$. For comparison, we once again perform lasso regression on the whole data set using these parameters. The resulting plot can be seen in figure 8.

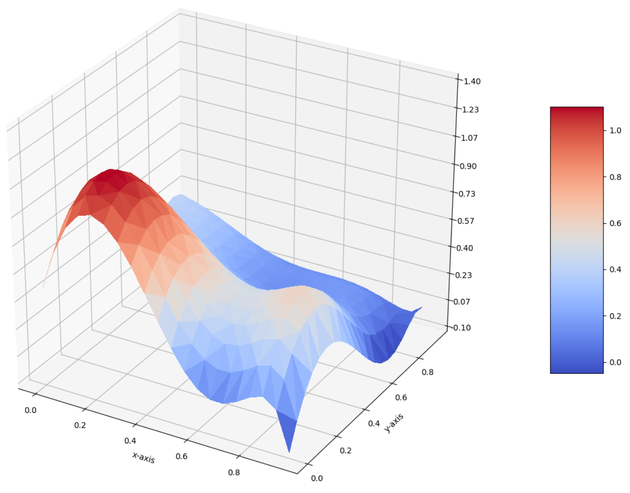


Figure 8: Lasso regression estimation of Franke's function with parameters $d = 9$ and $\lambda = 1.548365 \times 10^{-5}$

4.5 Applying everything to the terrain data

We are finally ready to look at the terrain data. The file 'SRTM_data_Norway_2.tif' contains a matrix, which when plotted (with grayscale) gives the image to the left in figure 9. In the interest of time, we reduced the data set to the one shown to the right in figure 9. This reduced data set only has $20 \times 20 = 400$ data points, which is the same amount as we had for the Franke's function data set. Additionally, we have normalized the data set to prevent the values from blowing up for large degrees.

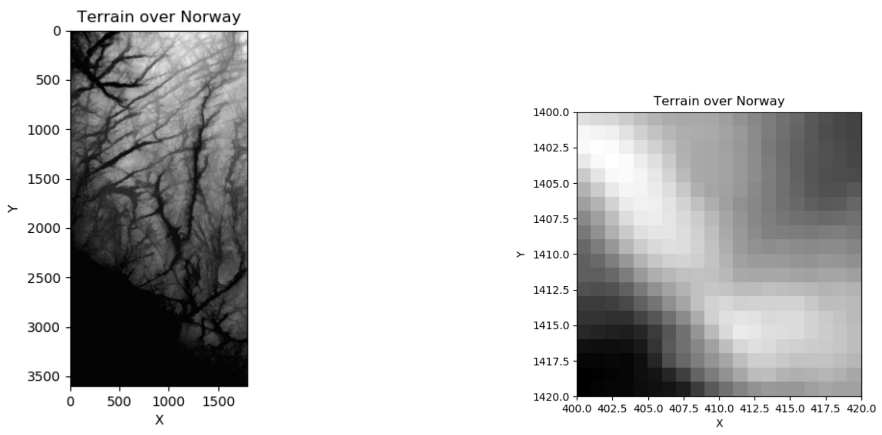
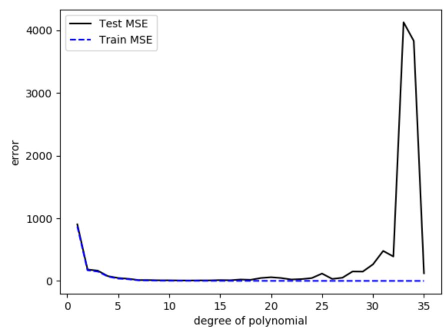


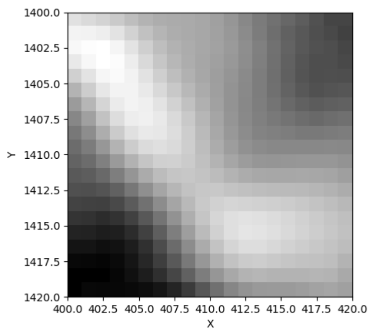
Figure 9: To the left, a plot of the data set from 'SRTM_data_Norway_2.tif'. To the right, a small subset of this plot. The reason why we reduced the data and used ML on this data and not the full data set was due to bad time. Since the full data set required significantly more amount of time to run ML on, we needed to decrease amount of data points in order to produce result.

4.5.1 OLS

We once again use cross validation with increasing degree d to find the degree which results in the lowest MSE for linear regression. Figure 10a shows the resulting test-train error-plot. The the minimum of 7.7968 is reached at $d = 12$. Figure 10b shows the result of doing linear regression with $d = 12$ on the whole data set. The MSE and R2-scores are 3.2280 and 0.9965, respectively.



(a) Plot of test and training error (OLS)

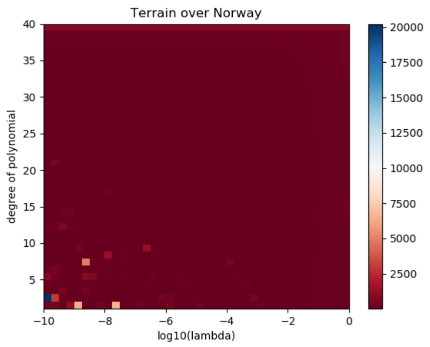


(b) OLS estimate of terrain data using optimal degree

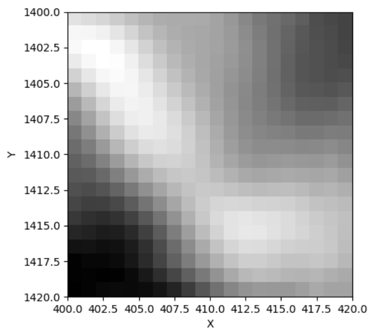
Figure 10

4.5.2 Ridge Regression

With the same approach, we get the results shown in figure 11a. The minimum MSE is reached with $d = 25$ and $\lambda = 1.0608 \times 10^{-9}$, and is 5.1288. Figure 11b shows the result of doing ridge regression with $d = 25$ and $\lambda = 1.0608 \times 10^{-9}$ on the whole (reduced) data set. The MSE and R2-scores are 2.4446 and 0.9973, respectively.



(a) Color map of the MSE vs. polynomial degree and the value of λ (Ridge)

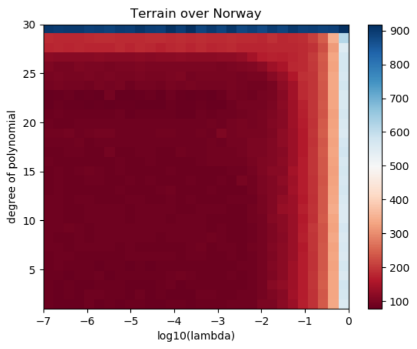


(b) Ridge estimate of terrain data using optimal degree and lambda

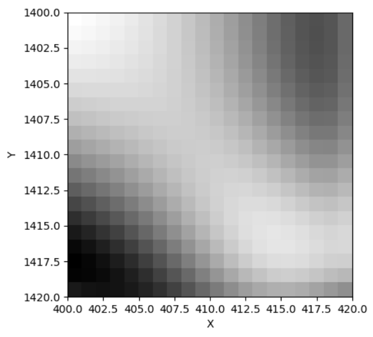
Figure 11

4.5.3 Lasso Regression

With the same approach, we get the results shown in figure 12a. The minimum MSE is reached with $d = 30$ and $\lambda = 8.5316 \times 10^{-6}$, and min(MSE) = 77.2870 seen on the color map. Figure 12b shows the result of doing LASSO regression with $d = 30$ and $\lambda = 8.5316 \times 10^{-6}$ on the whole reduced data set. The MSE and R2-scores are 73.3140 and 0.9206, respectively.



(a) Color map of the MSE vs. polynomial degree and the value of λ (LASSO)

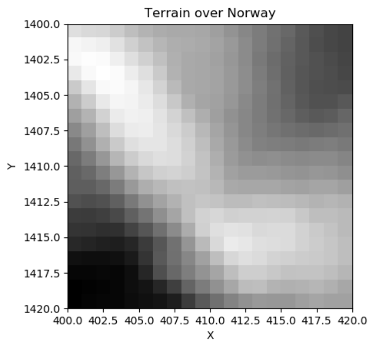


(b) LASSO estimate of terrain data using optimal degree and lambda

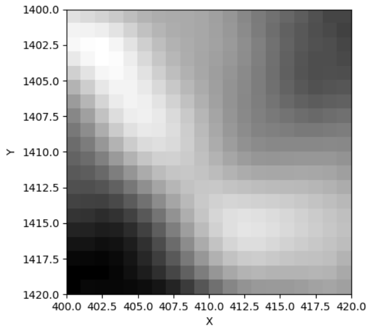
5 Discussion

5.1 The results of the analysis of the terrain data

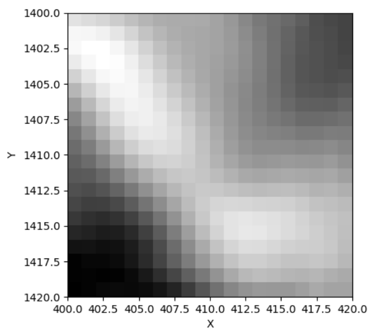
The results of the previous section are summarized in figure 13. Out of the three regression methods, Ridge regression performed the best, with MSE 2.4446 and R2-score 0.9973. However, these scores were obtained with $\lambda = 1.0608 \times 10^{-9}$, and $d = 25$. This λ value is close to zero, and when $\lambda = 0$ the ridge regression method is equivalent to linear regression (OLS). This explains the somewhat similar results of the linear regression method, which has MSE 3.2280 and R2-score 0.9965. Lasso regression had the worst performance, with MSE 73.3140 and R2-score 0.9206.



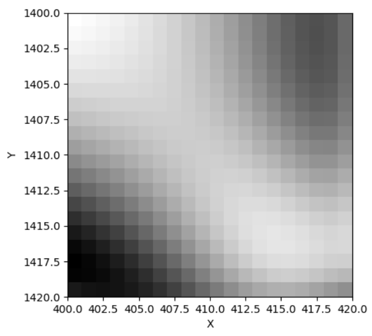
(a) The true terrain image



(b) OLS etimate with optimal degree



(c) Ridge estimate with optimal degree and λ



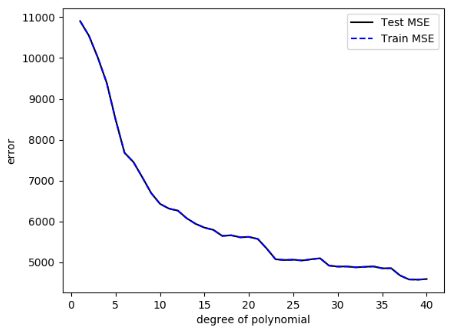
(d) LASSO estimate with optimal degree and λ

Figure 13

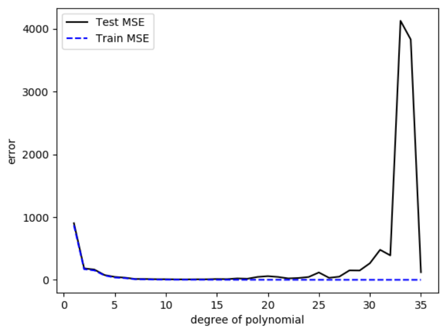
The only difference from Ridge regression is that the regularization term is in absolute value.But this difference has a huge impact on the trade-off. Further comparing the Lasso shrinkage method with Ridge regression, we see that the $l2$ shrinkage does not enforce some of the β become zero, but rather minimize it. This will remove overfitting just as LASSO, but still preserve important information. Thus, less information is lost. This can be seen when comparing the MSE and R2 score of LASSO with OLS and Ridge.

5.2 Number of datapoints and overfitting

In statistics, overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. An overfitted model is a model that contains more parameters than can be justified by the data. Thus, when trying to approximate a function with polynomials, as the the degree of polynomial increases, so does the chance of overfitting. Hence, the more datapoints we use , the less of a chance there is to overfit the data. This can be seen in figure 14a and 14b. In the left plot, we see that the MSEs reduce mush slower than in the right plot. This is one of the reasons we



(a) over 3600 datapoints



(b) 400 datapoints

chose to reduce the number of data points in our dataset to just 400. The computational time was reduced significantly. The first reason is because the design-matrix is smaller when the number of data points is small. The second reason is because we do not need to calculate the MSE for higher degrees for the polynomial when the minimum MSE is reached earlier as a consequence of the fewer number of data points.

6 Conclusion

As we can see from previous in the result and discussion, we saw that the Ridge regression performed best and gave the best result, with LASSO giving the worst. This may not be the case with other data set. As we know the ridge regression minimizes the coefficients in beta but not enforce them to be zero as in LASSO. if we consider this data set, LASSO may have removed important information, thus giving a worse result compared to OLS and Ridge.

Future work we wish to increase the number of the data points used in our terrain data. As we wrote above, due to limited time we had to reduce the data points used. Further, we wish to optimize the stability of the code and minimize number of matrix operation. Calculating matrices can be quite tedious for the computer, thus having a more efficient program to run these calculation are necessary.

7 Appendix

References

[Wil, 1996] (1996). *Regression Shrinkage and Selection via the Lasso*, volume 58. [Royal Statistical Society, Wiley].

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[McDonald, 2009] McDonald, G. (2009). *Ridge regression*, volume 1.

j	β_j	n_{lower}	n_{upper}
0	.2915	0.2786	0.3044
1	6.4311	6.2895	6.5727
2	4.8303	4.6887	4.9719
3	-12.2524	-12.9198	-11.5851
4	$- - 13.0968$	-13.67668	-12.5170
5	-14.1442	-14.8116	-13.4768
6	-23.0037	-24.49894	-21.50854
7	14.0983	12.6796	15.5171
8	16.9333	15.5145	42.00
9	4.9635	3.4683	6.4587
10	47.2887	45.6356	48.9417
11	15.1118	13.2623	16.96135
12	34.7991	32.9988	36.5994
13	-42.6591	-44.50869	-40.8029
14	16.9046	15.2516	18.5577
15	10.1128	9.4649	10.7606
16	-23.5510	-23.5510	-22.4794
17	-0.3147	-1.5600	0.9305
18	-35.0984	-36.3437	-36.3437
19	22.7904	21.7189	23.8620
20	-5.30869	-5.9565	-4.6608
21	-38.45807	-40.0793	-36.8367
22	-34.3435	-35.8257	-32.8614
23	-19.4942	-21.0011	-17.9873
24	-10.6562	-12.3036	-9.0087
25	6.7678	5.2609	8.2747
26	28.5793	27.0971	30.0614
27	-15.0844	-16.7057	-13.4632
28	8.9211	7.8811	9.9619
29	38.4348	37.1801	39.6895
30	1.1227	-0.2984	2.54404
31	7.1407	5.4341	8.8472
32	33.4387	31.7321	35.1452
33	-22.9846	-24.4058	-21.5633
34	-15.5553	-16.8100	-14.3006
35	7.4718	6.4314	8.5122

Table 1: The estimates of the 35 β coefficients for the Ridge regression model (with polynomial degree = 35) of Franke's function, and their respective 95% confidence intervals.