Project 1

Oliver Istad Funch

1 Introduction

This report is written exercise by exercise. However, note the later exercises builds on the former, such that much of the knowledge and reflections shared in, especially, exercise 1 has been used further in later exercises.

2 Exercise 1

2.1 Problem and numerical methods

The main objective of this exercise is to fit a surface $z = f(x, y)$ based on polynomials in x and y to generated data using Ordinary Least Squares(OLS)

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

The data is generated using Franke Function with an additional noise term $\epsilon \sim N(0, \sigma)$.The goal is to find the parameters, $\boldsymbol{\beta}$ that minimize the cost function mean squared errors

$$\begin{aligned} C(\boldsymbol{\beta}) = MSE(z, \tilde{z}) &= \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 && (2) \\ &= \frac{1}{n}\left\{(\boldsymbol{z} - \boldsymbol{X\beta})^T(\boldsymbol{z} - \boldsymbol{X\beta})\right\} && (3) \\ &= \mathbb{E}[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2] && (4) \end{aligned}$$

such that

$$\frac{\partial C}{\partial \beta_j} = 0 \tag{5}$$

Solving above equation leads to the analytical expression

$$\boldsymbol{\beta} = (X^T X)^{-1}\boldsymbol{z} \tag{6}$$

where $\boldsymbol{z}$ is the target data and X is the design matrix constructed from polynomials in x and y. Each term resulting from $(x + y)^d$ excluding the constant is granted its own column in the design matrix as a feature. These constants are then to be found using above equation. For instance, one can find the intercept by solving

$$\frac{\partial C}{\partial \beta_0} = 0 \tag{7}$$

leading to this expression for the intercept

$$\begin{aligned} \beta_0 &= \frac{1}{n}\sum_{i=0}^{n-1} z_i - \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=1}^{p-1} X_{ij}\beta_j \\ &= \frac{1}{n}\sum_{i=0}^{n-1} z_i - \sum_{j=1}^{p-1}(\beta_j \frac{1}{n}\sum_{i=0}^{n-1} X_{ij}) \end{aligned} \tag{8}$$

The very last part of the equation above, $\frac{1}{n}\sum_{i=0}^{n-1} X_{ij}$ is the column-wise mean of the design matrix.

When $X^T X$ cannot be inverted through standard inversion, a pseudo inverse is calculated using singular value-decomposition(SVD)

$$X = U\Sigma V^T, \tag{9}$$

where the Σ contains the singular values and U and V are orthogonal matrices. The pseudoinverse X_{PI} can then be found using

$$X_{PI} = V D_{PI} U^T \tag{10}$$

Here D_{PI} is the pseudoinverse of Σ, which is created by taking the reciprocal of each singular value and distributing them along the diagonal of D_{PI}.
Furthermore, a 95% confidence interval is constructed for $\boldsymbol{\beta}$ according to

$$(\boldsymbol{\beta} - 1.96\sigma_\beta, \boldsymbol{\beta} + 1.96\sigma_\beta) \tag{11}$$

where σ_β^2 is the variance of $\boldsymbol{\beta}$ given by

$$Var(\beta_j) = \sigma^2 (X^T X)^{-1}_{jj}. \tag{12}$$

Furthermore, the MSE(given in Eq 2) and R^2 score

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

is calculated both using degrees up to 5 in the design matrix. Before fitting β it is common to scale the data. There are several ways of scaling

(Centering)

$$X_{ij} \to X_{ij} - \mu_j \tag{14}$$

$$\tag{15}$$

(Standardization)

$$X_{ij} \to \frac{X_{ij} - \mu_j}{S_j}. \tag{16}$$

In both cases, every element of the design matrix is subtracted by the mean of the element's column,

$$\mu_j = \frac{1}{n}\sum_{i=0}^{n-1} X_{ij}, \tag{17}$$

such that the mean of each column is 0. When standardizing, the elements are also divided by the column's standard deviation

$$S_j = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(X_{ij} - \mu_j)^2}, \tag{18}$$

such that every column has a standard deviation of 1. Another way of scaling is the min-max-scaler

$$\tag{19}$$

(Min Max Scaling)

$$X_{ij} \to (b - a)\frac{X_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)} - a \tag{20}$$

2.2 Method and results

The source code for this project can be found in the appendix or by following the author's github link. As the code file contains comments explaining every algorithm, the code will not be explained extensively in this document. However, the overall procedure as well as key components of algorithms used and developed will be presented.
The overall procedure is as follows:

1. Generate data

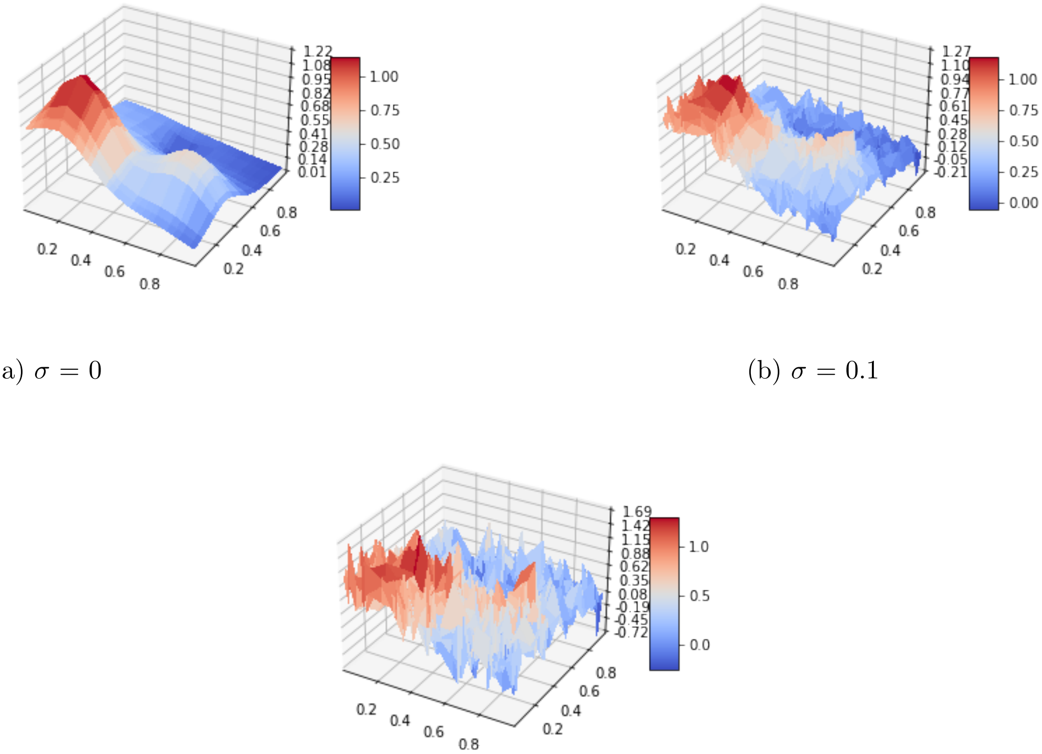
2. Create design matrix

3. Split the target data and design matrix into train and test data

4. Scale target data and design matrix
5. Calculate optimal parameters, β and construct 95% confidence interval
6. Predict train and test data and calculate MSE and R^2
7. Analyze effect of noise term

1. Generating data

To allow easier tweaking of noise term and amount of data points, a function for generating data is developed. Its explanation can be found in the code. This function can later be called in a loop with altering σ or N to study their effects. σ is the standard deviation of the noise term, while N is number of x and y coordinates. As such, N=30 results in 30*30 = 900 total data points. The function returns x, y and z, where x and y are to be used to create the design matrix and z is the target data. Plots of the data with varying noise can be seen in Figure.



(a) $\sigma = 0$

(b) $\sigma = 0.1$

(c) $\sigma = 0.3$

Figure 1: $\sigma = 0.1$

Data generated with higher σ are much more noisier and fits on this data is likely to be poorer. This is because the spread of data points are larger leading to a higher MSE as will be shown later.

2. Create design matrix

Design matrices are created based on polynomials in x and y up to a degree of 5. Note that only one design matrix is used when performing regression, however, several are created to demonstrate effect of different degrees of the polynomial. SciKit Learn offers a function named PolynomialFeatures however, it does not accept degree = 0. As such, an own developed function is preferred in this case to demonstrate effect of only using the constant intercept as a predictor. A function is written here as well due to repeated uses in later procedures. The solution can be found in the code.

3. Split the target data and design matrix into train and test data

The SciKit Learn train_test_split function is used with a test split of 0,2. This means that 20% of the data will be kept as test data at the end, whereas the 80% will be used to fit the function. Test data is useful as it gives the model means of testing itself of "new" data not included in the training, thereby giving a more "honest" score.

4. Scale target data and design matrix

There are little to no reason to center the generated data when performing OLS. When centering the data all parameters except the intercept will remain the same when using OLS. This is because all the data is simply linearly shifted by a constant. This makes sense intuitively, as the optimal fit to the centered data should still exhibit the same curvature, however shifted to other values. As such, adding the change in intercept, β_0 at the end will take you to the original scale. When centering only the design matrix, Eq 8 can be rewritten as such

$$\begin{aligned}\beta_0 &= \bar{z} - \sum_{j=1}^{p-1}(\beta_j \frac{1}{n}\sum_{i=0}^{n-1}(X_{ij} - \mu_j)) \\ &= \bar{z} - \sum_{j=1}^{p-1}(\beta_j \frac{1}{n}\sum_{i=0}^{n-1}(X_{ij}) - \mu_j)) \\ &= \bar{z} - \sum_{j=1}^{p-1}(\beta_j(\mu_j - \mu_j)) \\ &= \bar{z}\end{aligned}$$

When the design matrix is centered, the intercept becomes the mean value of $\boldsymbol{z}$. If the target data $\boldsymbol{z}$ is also centered,

$$\begin{aligned}\beta_0 &= \bar{z} - \bar{z} \\ &= 0\end{aligned}$$

the intercept becomes 0. As such, one has to add $\bar{z} - \sum_{j=1}^{p-1}(\beta_j \frac{1}{n} \sum_{i=0}^{n-1}(X_{ij}))$ to the final result if fitting is done with centered design matrix and target data, but prediction is done on normal data, thereby reaching the exact same prediction as without centering. In other words, using centered data will yield the exact same MSE and R^2, and there is no performance-wise reason to do it. A common argument for centering is to ensure that all parameters have the same basis, with mean equal to zero. However, it is mostly helpful when using regularization so that biased parameters don't get penalized more than less biased parameters.

Another option is to standardize the data according to Eq 15, however, as with centering, one can reach the exact same value as no scaling by ”unscaling” after prediction with standardized data. Usin the code

```
#Split data
X_train, X_test, z_train, z_test = train_test_split(X,z,
                                                    test_size=0.2,
                                                    random_state = 42)

#Calc mean
mean_X_train = np.mean(X_train,axis=0)
mean_z_train = np.mean(z_train,axis = 0)
std_X_train = np.std(X_train,axis=0)
std_z_train = np.std(z_train,axis=0)

#Center the data
X_train_standardized =scale_data(X_train, mean_X_train, std_X_train)
X_test_standardized = scale_data(X_test, mean_X_train, std_X_train)
z_train_standardized = scale_data(z_train, mean_z_train, std_z_train)

beta_1,_ = fit_beta(X_train,z_train)
beta_2,_ =fit_beta(X_train_standardized,z_train)
beta_3,_ = fit_beta(X_train_standardized,z_train_standardized)

#Dataframe for readability
beta_df = pd.DataFrame(np.column_stack((beta_1,beta_2,beta_3,
                                        np.divide(beta_2,beta_1,
                                                  where=beta_1!=0,
                                                  out=np.zeros_like(beta_2)),
                                        std_X_train)),
                       columns = ['beta_1', 'beta_2',
                                  'beta_3', 'beta_2/beta_1',
                                  'std_X_train'])
beta_df.index.name = "beta"

print(beta_df)
```

the relation ship between the non-standardized β and the standardized one is studied. Running this code yields the output

```
        beta_1    beta_2    beta_3  beta_2/beta_1  std_X_train
beta
0     0.953096  0.000000  0.000000       0.000000     0.000000
1    -0.389101 -0.114926 -0.277005       0.295364     0.295364
2     1.458796  0.421291  1.015431       0.288793     0.288793
3    -1.650688 -0.500193 -1.205608       0.303021     0.303021
4     1.759580  0.392846  0.946870       0.223261     0.223261
5    -6.751602 -2.000169 -4.820975       0.296251     0.296251
6     0.979724  0.281339  0.678108       0.287162     0.287162
7     0.525748  0.105267  0.253723       0.200223     0.200223
8    -1.431496 -0.283234 -0.682675       0.197859     0.197859
9     4.755288  1.329246  3.203860       0.279530     0.279530
```

showing that in fact each β, when fitted to standardized data instead, is scaled as such

$$\beta_j^{std} = \beta_j S_j. \tag{21}$$

It is assumed that the small differences are due to numerical inaccuracies in the mathematical operations. When using $\boldsymbol{\beta}^{std}$ and standardized design matrix $\frac{X_ij - \mu_j}{S_j}$ to predict z_i

$$\tilde{z_i} = \beta_0 + \sum_{j=1}^{p-1} \frac{(X_{ij} - \mu_j)\beta_j S_j}{S_j} \tag{22}$$

$$= \beta_0 + \sum_{j=1}^{p-1} (X_{ij} - \mu_j)\beta_j \tag{23}$$

the same result is achieved as when using centered data. As such, it will yield the same MSE and R^2 as well. Similarly, there are no performance-wise argument for standardization when using OLS. The argument in this case is that the parameters are made dimensionless when dividing by standard deviation, and so enables easier comparison of features. In this case, the design matrix already lies in the interval [0,1] and is effectively pre-scaled with a min-max-scaler. This can be shown by applying the min-max-scaler where $(a, b) = (0, 1)$. Due to the fact that $(min(x_j), max(x_j)) = (0, 1)$ each X_{ij} will remain the same. The target data z is already within a small interval, almost [0,1] such that it likely doesn't benfit much from scaling either. Henceforth, no scaling will be applied when using OLS.

5. Calculate optimal parameters, β

Numpy's linalg.pinv is used to invert $X^T X$, which in turn employs the singular value decomposition(SVD) to pseudoinvert the matrix. Using the function

```
def fit_beta(X_train, z_train):
    XTX_inv = np.linalg.pinv(X_train.T @ X_train)
    beta = XTX_inv @ X_train.T @ z_train
    return beta, XTX_inv
```

the optimal parameters are found. The inverse of $X^T X$ is also returned in this function as it is used to construct a confidence interval. Results for $\boldsymbol{\beta}$ are studied both by varying σ and by varying the polynomial degree.

Effect of noise

The resulting optimal parameters with their 95% confidence intervals are shown in Table 1 for σ values 0, 0.3 and 1. Here a polynomial degree of 5 is used.

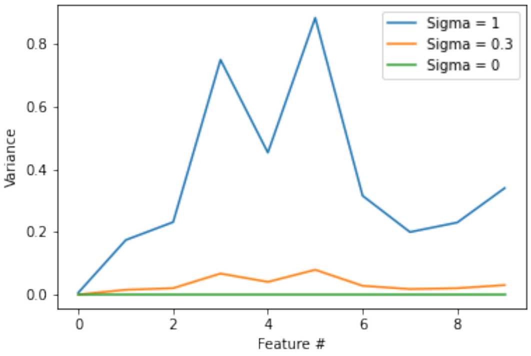
	$\sigma = 0$	$\sigma = 0.3$	$\sigma = 1$
β_0	0.5 [0.5 0.5]	0.6 [0.4 0.7]	0.8 [0.4 1.1]
β_1	7.5 [7.5 7.5]	6.8 [5.7 8.0]	5.4 [1.6 9.2]
β_2	3.0 [3.0 3.0]	2.2 [0.9 3.5]	0.3 [-3.9 4.5]
β_3	-33.1 [-33.1 -33.1]	-31.5 [-37.1 -25.9]	-27.7 [-46.5 -9.0]
β_4	-12.1 [-12.1 -12.1]	-7.9 [-12.2 -3.5]	2.0 [-12.6 16.6]
β_5	-6.5 [-6.5 -6.5]	-4.7 [-10.8 1.5]	-0.4 [-20.9 20.0]
β_6	45.6 [45.6 45.6]	43.2 [30.5 55.8]	37.4 [-4.8 79.6]
β_7	43.0 [43.0 43.0]	37.7 [28.2 47.3]	25.3 [-6.5 57.2]
β_8	11.4 [11.4 11.4]	4.1 [-5.2 13.3]	-13.0 [-43.8 17.9]
β_9	-9.4 [-9.4 -9.4]	-10.2 [-23.8 3.5]	-12.0 [-57.4 33.4]
β_{10}	-21.1 [-21.1 -21.1]	-19.1 [-32.3 -5.8]	-14.3 [-58.4 29.8]
β_{11}	-53.4 [-53.4 -53.4]	-51.6 [-61.8 -41.4]	-47.2 [-81.2 -13.1]
β_{12}	-3.5 [-3.5 -3.5]	3.9 [-5.4 13.2]	21.3 [-9.8 52.3]
β_{13}	-19.8 [-19.8 -19.8]	-16.2 [-26.1 -6.3]	-7.8 [-40.9 25.3]
β_{14}	27.6 [27.6 27.6]	26.3 [12.3 40.3]	23.3 [-23.4 69.9]
β_{15}	0.7 [0.7 0.7]	0.1 [-5.1 5.3]	-1.3 [-18.6 16.0]
β_{16}	18.7 [18.7 18.7]	18.6 [14.1 23.1]	18.3 [3.4 33.3]
β_{17}	11.1 [11.1 11.1]	9.0 [4.5 13.5]	4.3 [-10.7 19.3]
β_{18}	-8.4 [-8.4 -8.4]	-10.4 [-14.6 -6.2]	-15.1 [-29.1 -1.1]
β_{19}	13.4 [13.4 13.4]	13.0 [8.7 17.4]	12.2 [-2.3 26.7]
β_{20}	-15.0 [-15.0 -15.0]	-14.0 [-19.4 -8.6]	-11.6 [-29.5 6.4]

Table 1: Optimal parameters for different values of σ

The numbers themselves might be of little interest to the reader, however, it is worth noting that the parameters are fluctuating somewhat. This is true in all cases, however, more so with increasing σ. The large positive coefficients are canceled by a following similarly large but negative coefficient. With the fact that all the data lies between 0 and 1 in mind, these parameters seem unnecessary large and poorly determined. Ridge and Lasso regressions can relieve this issue, as will be shown later.

The confidence interval for each β increases with σ as expected, since $Var(\beta)$ is proportional to σ^2. Additionally, one can observe larger confidence intervals for less important features as they exhibit larger variance and are therefore less determined. From the table above, it can be observed that the first few parameters(for all σ) has a more narrow confidence interval which may indicate that they play a more important role in the estimation of z. This is also indicated by Figure 2. Higher polynomials tend to have larger variance, although some of the higher ones are occasionally lower than the former.

Figure 2: Variance of β for different values of σ



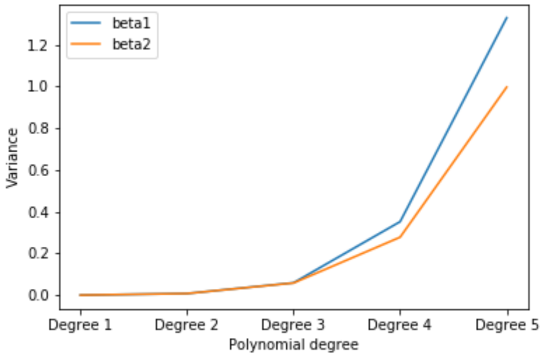


Figure 3: Variance of β_1 and β_2 for different polynomial degrees

Effect of model complexity

The optimal parameters achieved for different complexities(polynomial degree) can be observed from the printout below. σ is set to 0.3, and will be used in further analysis.

```
    Degree 0  Degree 1  Degree 2  Degree 3   Degree 4   Degree 5
0   0.391107  0.883163  0.972656  0.912473   0.681105   0.730758
1        NaN -0.436693 -0.846552 -0.757022   2.660573   2.315469
2        NaN -0.551383 -0.180546  1.634187   3.113391   2.033198
3        NaN       NaN  0.072597 -0.667662 -13.500610  -8.864627
4        NaN       NaN  0.696082  2.303860  -2.158283   2.161992
5        NaN       NaN -0.749383 -7.417892 -12.028422  -7.766054
6        NaN       NaN       NaN  0.451825  17.857689  -3.781243
7        NaN       NaN       NaN  0.198316   7.177947  17.194605
8        NaN       NaN       NaN -1.837381   1.964002 -17.771993
9        NaN       NaN       NaN  5.314556  11.193667   1.615627
10       NaN       NaN       NaN       NaN  -7.831199  25.562007
11       NaN       NaN       NaN       NaN  -3.566564 -35.855761
12       NaN       NaN       NaN       NaN  -1.786902  31.826604
13       NaN       NaN       NaN       NaN  -1.401635  -1.639414
14       NaN       NaN       NaN       NaN  -2.568809  12.588284
15       NaN       NaN       NaN       NaN        NaN -15.996787
16       NaN       NaN       NaN       NaN        NaN  13.481681
17       NaN       NaN       NaN       NaN        NaN   5.468072
18       NaN       NaN       NaN       NaN        NaN -28.761146
19       NaN       NaN       NaN       NaN        NaN  14.232781
20       NaN       NaN       NaN       NaN        NaN  -9.070455
```

As degrees increases, so does the number of features. The fluctuations seem to be increasing somewhat with increasing complexity. Figure 3 show how the variances of β_1 and β_2 increases with increased complexity. It seems that with increasing complexity, it is not only the new parameters that becomes less determined, but the existing as well. This intuitively makes sense as with more parameters the model has more options to reach the same prediction.

6. Predict train and test data and calculate MSE and R^2

Both the training data and test data are predicted using the above obtained coefficients. The prediction is performed using the function

```
def model_predict(X_data, beta):
    return X_data @ beta
```

which takes a design matrix and β as input, and outputs predicted z-value. The MSE and R^2 are implemented as functions as well to enable easy reuse when necessary.

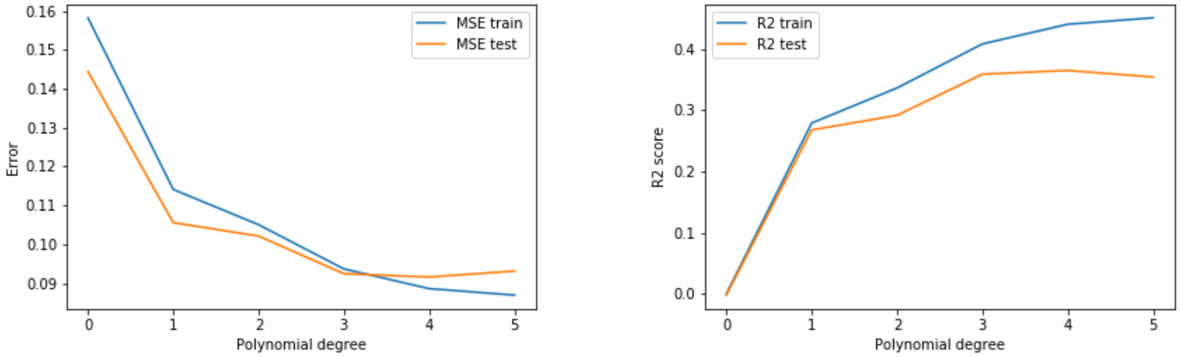
```
def R2(z_data, z_model):
    return 1 - np.sum((z_data - z_model) ** 2) / np.sum((z_data - np.mean(z_data))

def MSE(z_data,z_model):
    n = np.size(z_model)
    return np.sum((z_data-z_model)**2)/n
```

Results are shown for both varying σ and degree. Additionally, a figure demonstrating the effect of number of data points is presented as well. The table below show the MSE and R^2 score using N=30(900 data points) and $\sigma = 0.3$.

```
   Train MSE  Test MSE  Train R2   Test R2
0   0.161693  0.170534  0.000000 -0.000586
1   0.113971  0.116209  0.295138  0.318159
2   0.105240  0.106298  0.349135  0.376311
3   0.092132  0.097430  0.430202  0.428344
4   0.090036  0.092627  0.443170  0.456520
5   0.086951  0.090185  0.462247  0.470851
```

The train error and R^2 improves as complexity increased. Train error and R^2 only does so up until 4 degrees, after which it decreases. This is can be observed in Fig 4.

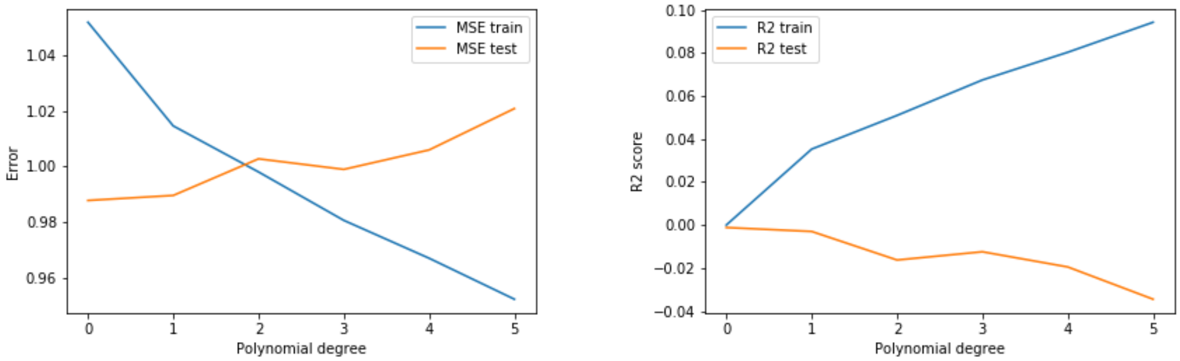


(a) MSE

(b) $R^2 score$

Figure 4: Results for OLS model: N = 30, σ = 0.3

Note that the closer to 1 the R^2 score is , the better the model performance. R^2 describes the portion of the variance in dependent variable(z) that is predictable from the independent variables(X), so the closer to 1 the more they correlate. In other words, if the variance(variation around the mean) in the data in reality is due to some relationship between the dependent and independent variable, and our model is successful at modelling at picking up that relationship, then the variation around the model($\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2$) should be substantially lower than the variance of the data. The figure above may indicate that the model enters the region of overfit after 4 degrees. Shortly explained, an overfit model is very able to predict on the data it has trained on, however performs poorly on new data. This is indicated by a large discrepancy between train error and R^2 and test error and R^2. It can also be observed on the fluctuations of β_j, where the model tries to reach as many of the training data points as possible by using large coefficients, however when new data is introduced it misses heavily. Increasing σ to 1 will lead to a poorer fit for the train data, but also a bigger discrepancy between train and test scores, as can be seen in Fig 5.

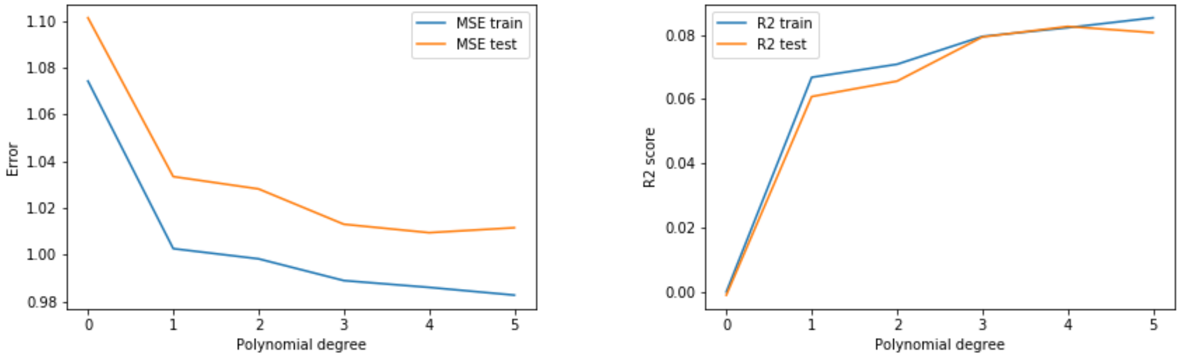


(a) MSE

(b) $R^2 score$

Figure 5: Results for OLS model: N = 30, σ = 1

This is no surprise as data with higher spread is harder to model. If one, on the other hand, increases the number of data points but maintains the noise, the training scores improves as one would expect.

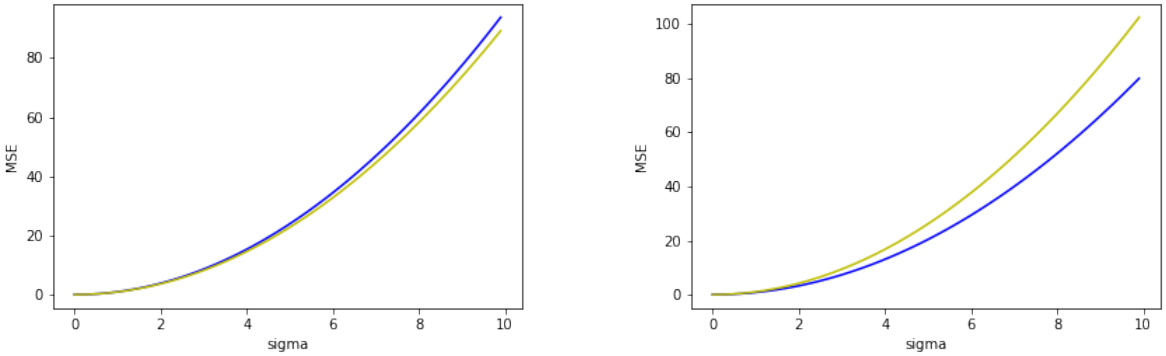


(a) MSE

(b) $R^2 score$

Figure 6: Results for OLS model: N = 100, σ = 1

However, the "overfitness" seems to decrease as well, meaning that the model is also better able to predict the test data. A more direct analysis of the effect noise has on MSE is shown in Fig



(a) Degree = 7, N = 30

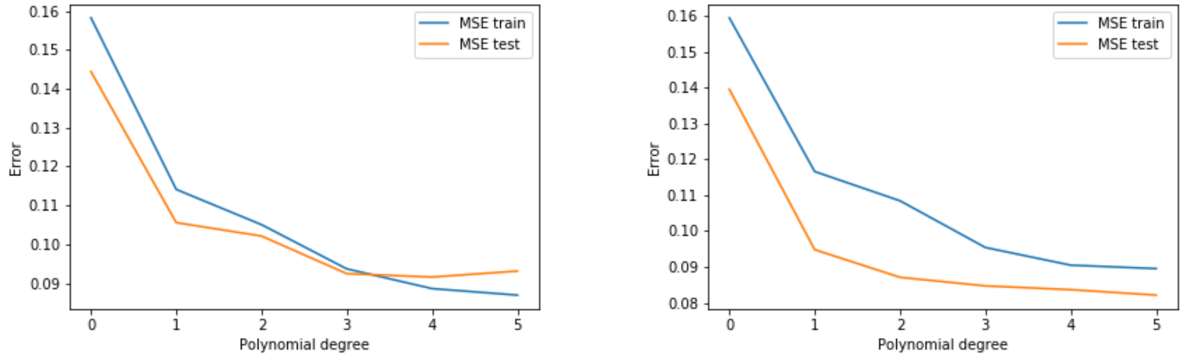
(b) Degree = 30, N = 30

Figure 7: OLS MSE for a range of σ values

It can be seen that the gap between train and test MSE increases with increasing σ. Increasing the complexity more than a certain amount also widens the gap between train and test MSE. This is perhaps because a high complexity model tries "too hard" to model the high spread data by capturing all the training data points. This in turn results in failure when predicting new data. Note that the noise used here is extensively big for demonstration purposes.

3 Exercise 2

Resampling is a method used to obtain more robust scores for our model, but can also be used to study the bias-variance trade-off. It involves using the same available data in different ways each run, thereby resulting in different scores. By resampling enough and using the mean of the scores(or whichever value that is calculated) from the different runs one can obtain a more accurate score(not necessarily a more accurate model). Fig 8 shows two different results using different random_state in the train_test_split function.



(a) Random state 1

(b) Random state 2

Figure 8: Results for OLS model: N = 30, $\sigma = 0.3$

The differences are substantial and shows that there is a need for some average scoring.

Bias-variance trade-off

The main objective of this exercise is to study the bias-variance trade-off using bootstrap. Note that $\boldsymbol{z} = f(x, y) + \epsilon = f + \epsilon$. Now by simply adding and subtracting $\mathbb{E}[\tilde{\boldsymbol{z}}]$ to the cost function Eq 4, it can be written as such

$$\mathbb{E}[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2] = \mathbb{E}\left[(\boldsymbol{f} + \epsilon - \tilde{\boldsymbol{z}} + \mathbb{E}[\tilde{\boldsymbol{z}}] - \mathbb{E}[\tilde{\boldsymbol{z}}])^2\right] \tag{24}$$

regrouping the terms

$$= \mathbb{E}\left[((\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}]) + (\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}}) + \epsilon)^2\right] \tag{25}$$

resolving the parenthesis

$$= \mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])^2 + (\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})^2 + \epsilon^2 + 2\epsilon(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}]) + 2\epsilon(\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})\right] \tag{26}$$

using $\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y]$

$$\begin{aligned} &= \mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])^2\right] + \mathbb{E}\left[(\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})^2\right] + \mathbb{E}\left[(\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})\right]\mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])\right] \\ &+ \mathbb{E}\left[2\epsilon(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])\right] + \mathbb{E}\left[2\epsilon(\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})\right] \end{aligned} \tag{27}$$

Due to $\mathbb{E}[\epsilon] = 0$ and the rule $\mathbb{E}[xy] = \mathbb{E}[x]\,\mathbb{E}[y]$ assuming x and y are independent, the three last terms becomes 0, and we're left with

$$= \mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])^2\right] + \mathbb{E}\left[(\mathbb{E}[\tilde{\boldsymbol{z}}] - \tilde{\boldsymbol{z}})^2\right] + \mathbb{E}\left[\epsilon^2\right] \tag{28}$$

Finally using Var[X] $= \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$= \mathbb{E}\left[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{z}}])^2\right] + Var[\tilde{\boldsymbol{z}}] + Var[\epsilon] \tag{29}$$

$$= \underbrace{\frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\boldsymbol{z}}])^2}_{\text{bias}^2} + \underbrace{\frac{1}{n}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{\boldsymbol{z}}])^2}_{\text{variance}} + \sigma^2 \tag{30}$$

Here $\mathbb{E}$ denotes the expectation across the samples. A total mean must be taken across data points to find the final MSE. It should now be obvious which term is the model variance($Var[\tilde{\boldsymbol{z}}]$) and which is the noise variance($Var[\epsilon]$). The remaining term is the bias^2. Note that it is the bias squared that is found in the equation.

Variance

The variance of the model is quite simply explained as the variance of the predictions. How much are the predictions spread from sample to sample. Note that a steeper curve doesn't result in higher variance, because it is the data points with the same x and y values that are compared from sample to sample. In other words, if the predicted z-value from the x and y value differ wildly from each resample, a high variance is expected.

Bias

The bias on the other hand is a measure of deviation between the real data points and the model. It's explanation lies in the word itself. If a model exhibits high 'bias' it has a high 'offset' from the real data points.

The trade-off

There is commonly a trade-off between these. Achieving lower bias is often at the cost of higher variance, and vice-versa. It can be explained using dart as comparison: You can either hit very consistently(low variance) within a small area, but off-target(high bias), or not so consistently(high variance) but evenly distributed around the target(low bias).

In this exercise the bootstrap method will be used to study the MSE, bias and variance as a function of polynomial degree to demonstrate the bias-variance trade-off. It is a simple procedure following these steps:

1. Draw bootstrap sample. Draw n data points with replacement. This means you could end up drawing the same data point more than once.

2. Fit β and calculate MSE, bias and variance

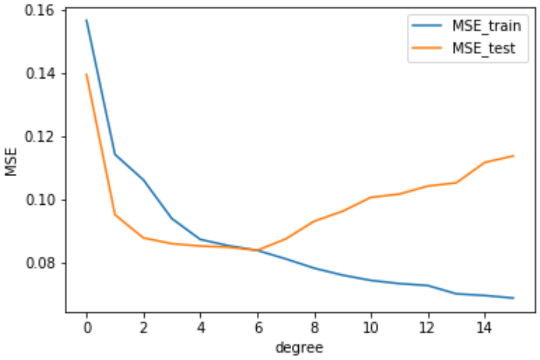
3. Do step 1-2 n_bootstraps number of times and store results each time

4. Calculate the mean of the MSE, bias and variance

5. Do step 1-4 max_degree number of times

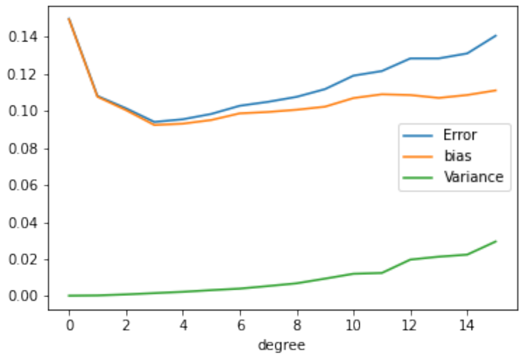
Before initiating this procedure, a figure demonstrating the effect model complexity has on train and test error is presented. This figure, Fig 9, is similar to Fig 4a above, however with more degrees to more clearly demonstrate the rise of test error with increasing complexity. Additionally, 100 bootstrap is used when calculating errors.

Figure 9: Train and test MSE using bootstrap: N = 100, σ = 0.1



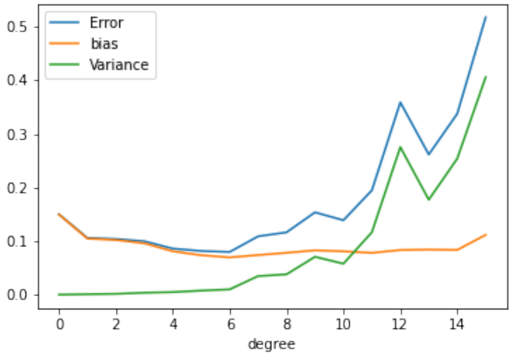
From Fig 9 it can be seen that the test error starts increasing after a polynomial degree of 6 indicating a region of overfit after this point. Results does not fully agree with Fig 4a, however, it is assumed the bootstrap result is more indicative of the model's overall performance, and that the first result was simply a "unlucky" run. Performing a bias-variance trade-off analysis shows, however, that the trade-off between bias and variance does not kick in before complexity is raised by a substantial amount, shown in Fig 10.

Figure 10: Bias-variance trade-off: N = 30, σ = 0.3



The result may indicate some sort of saturation with N = 30, where the model is able to sustain a farily low test error with increasing complexity. The error is increasing though slowly, however, it is mostly the variance that causes the increase. It must be noted that all data is random generated and so perfectly smooth curves cannot be achieved. If the number of data points is decreased, however, the region of overfit should move towards lower complexity as was found in Fig 15.

Figure 11: Bias-variance trade-off: N = 20, σ = 0.3



Which it indeed does when N=20(400 data points). In both cases, however, it can be observed that there is a rather long region of both relatively low bias and variance, which is optimal. The error seem to increase at a lower degree with more data(N=30), however, it must be noted that the error doesn't increase nearly as much as with N=20. As such, the amount of data should be included in the choice of model complexity.

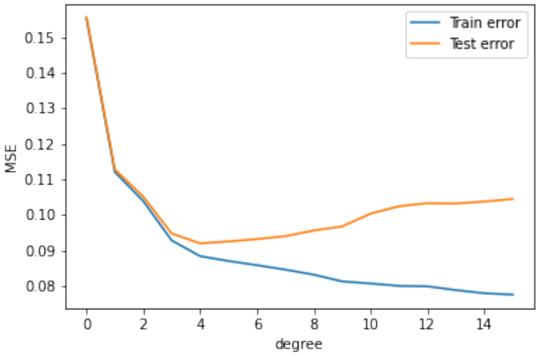
4 Exercise 3

Cross-validation is another resampling technique. Its procedure is as follows:

1. Split the whole data set into k equal parts, preferably with shuffling so that each part doesn't contain consecutive data
2. Assign one part as test data and rest as train data
3. Calculate β, predict training data and calculate MSE
4. Repeat step 2-3, k-1 times reassigning a new test set each time. Ensure the test set has never been used as test set before.
5. Calculate mean MSE

The details of the solution can be found in the attached code. In this case, the data is not centered due to the conclusion drawn in Exercise 1. When using OLS one is able to reach the exact same result as with centering or standardization, and is therefore not necessary in our analysis. The result of running cross-validation with up to 14 degrees polynomials can be observed in Fig 12

Figure 12: Train and test error: N = 30, σ = 0.3



It can be seen from this result that test error increases after about 4 degrees, which is slightly earlier than what the bootstrap method indicated. All in all, they seem very similar and differences may be due to randomness in the data. If we are to trust cross-validation, a degree of 4 seems to be the best complexity using OLS.

Summary exercise 1-3
The degree of the model may be increased to capture more details in the data. However, at a certain point too many details are captured in the train data worsening the model's general ability to predict new data(overfit). More data points is always better, it decreases both train and test error and moves the region of overfit towards higher complexities allowing the model to capture more of the intricacies in relationship of the data while maintaining the ability to predict new data. Increasing the stochastic noise worsen both the train and test error, however, decreases test error more so with increasing complexities. This may be because when the data is more spread and a high complexity model tries to capture that, it drastically fails at new data.

5 Exercise 4

Ridge regression is a linear regression method that is similar to OLS. It employs regularization to ensure that parameter values are maintained within a certain limit. The cost function is instead

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \left\{(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{31}$$

where the $\lambda\boldsymbol{\beta}^T\boldsymbol{\beta}$ is the regularizing term. It regularizes $\boldsymbol{\beta}$ because, when you try to minimize above function low values of $\boldsymbol{\beta}$ will contribute to that. λ is a value in the range [0,1], that determines the level of regularization. We obtain a slightly modified equation for β

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{32}$$

Using SVD, the prediction can be written as

$$\tilde{\boldsymbol{z}}_{\text{Ridge}} = \boldsymbol{X}\boldsymbol{\beta}_{\text{Ridge}} = \boldsymbol{U\Sigma V^T}\left(\boldsymbol{V\Sigma}^2\boldsymbol{V}^T + \lambda\boldsymbol{I}\right)^{-1}(\boldsymbol{U\Sigma V^T})^T\boldsymbol{z} = \sum_{j=0}^{p-1}\boldsymbol{u}_j\boldsymbol{u}_j^T\frac{\sigma_j^2}{\sigma_j^2+\lambda}\boldsymbol{z}, \tag{33}$$

where $\frac{\sigma_j^2}{\sigma_j^2+\lambda} \leq 1$ shrinks the coordinates, and more so with low singular values. Lower singular values means also that feature is less important to the final value. The regularizing term also consist of the intercept,β_0, such that with $\beta_0 \neq 0$ will alter which vales of $\boldsymbol{\beta}$ that minimizes the cost function. It is common to subtract the mean to remove the intercept, then calculate $\boldsymbol{\beta}$ and finally add the calculated intercept at the end. This way, the dampening effect is only applied on the parameters that decides the curvature and form, as opposed to the offset. Using centered target data and design matrix, the intercept is, as previously $\beta_0 = \bar{z} - \sum_{j=1}^{p-1}(\beta_j\frac{1}{n}\sum_{i=0}^{n-1}(X_{ij}))$, however, in this case using $\boldsymbol{\beta}_{Ridge}$. The data will as such be scaled using centering. SKlearn's Ridge subtracts the mean by default to avoid including the intercept in the regularizing term. When writing "own" code this should be done as well. Standardization is not employed as all the data is still within [0,1] and there is no real need to make the parameters dimensionless.

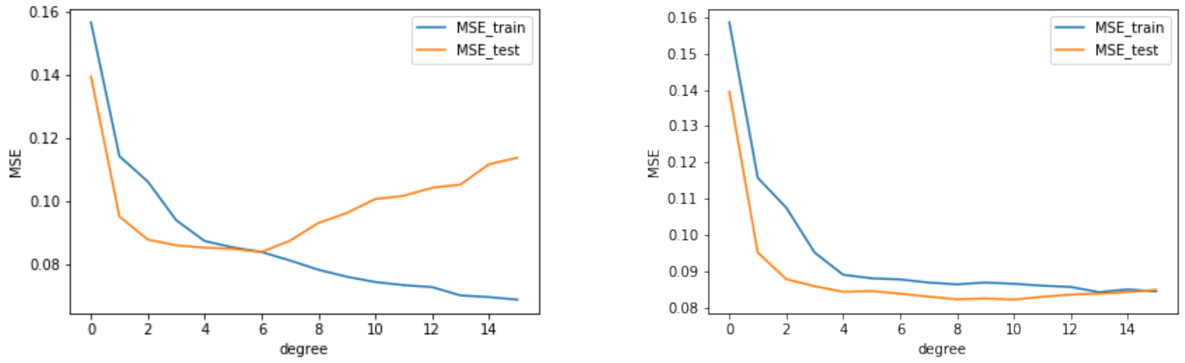
Running with similar N and σ as in exercise 1, we obtain these results.

```
             OLS  Ridge lmb= 0  Ridge(lmb=0.001)  Ridge centered(lmb=0.001)
beta
0       0.551104      0.551104          0.640604                   0.000000
1       6.826193      6.826193          4.520274                   4.520195
2       2.199418      2.199418          2.228993                   2.228880
3     -31.496420    -31.496420        -20.380539                 -20.380324
4      -7.853698     -7.853698         -2.546495                  -2.546164
5      -4.657315     -4.657315         -7.067293                  -7.066933
6      43.186007     43.186007         22.178139                  22.177842
7      37.712409     37.712409         20.950740                  20.950297
8       4.085440      4.085440         -0.832522                  -0.833117
9     -10.171043    -10.171043         -2.197700                  -2.198222
10    -19.050362    -19.050362         -1.905915                  -1.905720
11    -51.562449    -51.562449        -31.135686                 -31.135380
12      3.903792      3.903792          9.948537                   9.948918
13    -16.203441    -16.203441        -12.407239                 -12.406745
14     26.322840     26.322840         16.473118                  16.473465
15      0.070543      0.070543         -5.002901                  -5.002949
```

```
16    18.581222    18.581222          10.517798                   10.517711
17     9.029460     9.029460           5.160261                    5.160166
18   -10.406546   -10.406546         -10.614454                  -10.614580
19    13.014006    13.014006          11.291793                   11.291639
20   -14.005728   -14.005728          -9.827942                   -9.828028
```

As expected, the parameters becomes the same as with OLS when $\lambda = 0$. When using $\lambda = 0.001$ it can be observed that the fluctuations are dampened. It is then expected that Ridge also should reduce overfitting. With centered data the parameters are slightly different. The intercept was relatively low compared to the other parameters and thereby may not have dominated the cost function very much when it was not removed before fitting.

With a $\lambda = 0.001$ we obtain better test score for higher complexities.

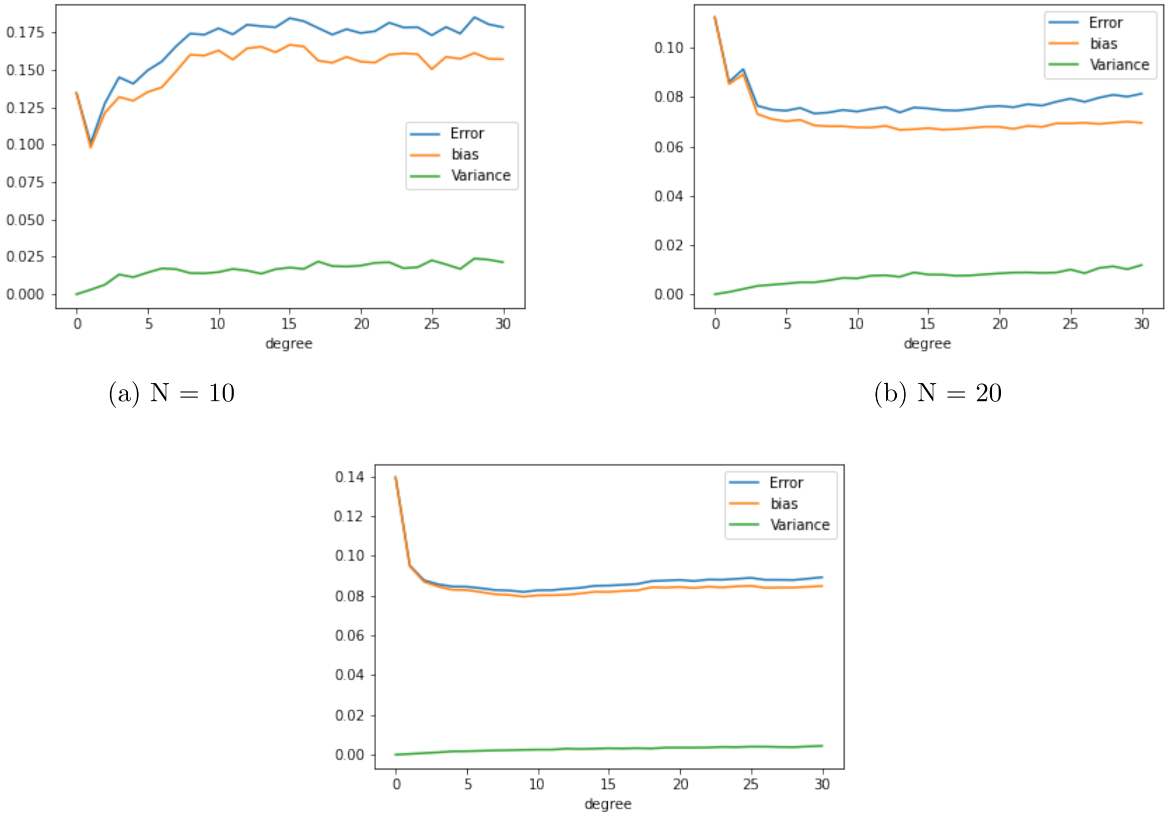


(a) OLS

(b) Ridge

Figure 13: Results for OLS model vs Ridge: N = 30, $\sigma = 0.3$, $\lambda = 0.001$

In this case, the test error is actually lower than the train error up until about 15 degrees. It can also be seen that the test error somewhat stagnates. This may be due to the fact that with Ridge, you shrink the value of the less important features more dramatically. Increasing complexity beyond a certain amount may only introduce less important features. Furthermore, it seems OLS and Ridge where similar in performance at around degree 6. However, it is clear that Ridge allows for higher complexities. The bias-variance trade-off analysis shows that 30 data points is more than enough for ridge not to overfit anytime soon with increasing degrees, and that lower amount of data can be handled with the current parameters.



(a) N = 10

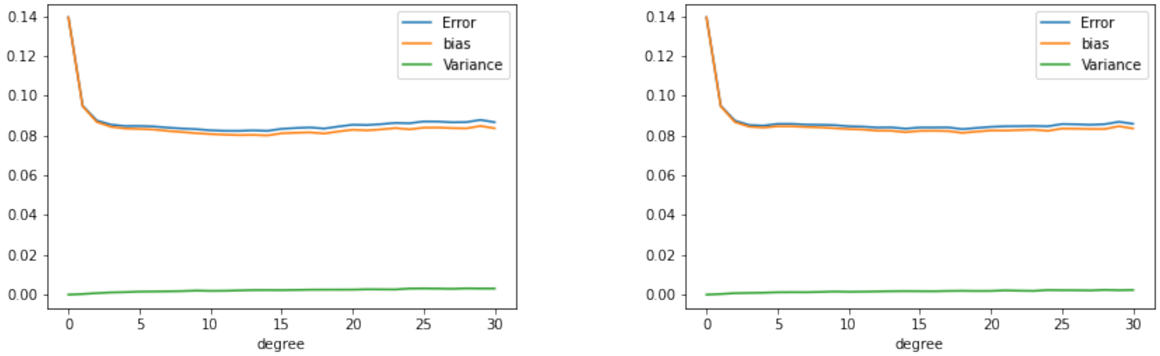
(b) N = 20

(c) N = 30

Figure 14: Results for Ridge: $\sigma = 0.3$, $\lambda = 0.001$

Comparing Fig 15b to Fig 10 it can be observed that the variance of the model is not increasing nearly as much as with OLS.

Increasing λ we see that there is very little change. It can be observed that the gap between error and bias are reduced somewhat. This is expected as more regularization should result in less fluctuating parameters, and lower variance.



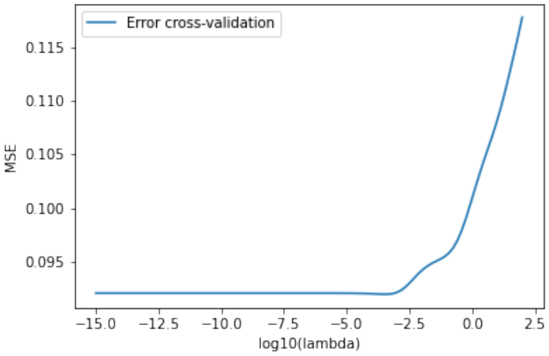
(a) $\lambda = 0.01$

(b) $\lambda = 0.1$

Figure 15: Results for Ridge: N = 30 σ = 0.3

However, it is unclear which λ reaches the lowest test error at some point. Test error will now be studied as a function of λ using cross-validation. Degree 4 has been chosen as Fig 12 indicated a good fit. Now the outer loop alters λ instead of degree. With N = 30 and $\sigma = 0.3$, and testing 100 different values of λ ranging from 10^{-15} to 10^3 the results are as follows

Figure 16: λ analysis: k= 5, N = 30, σ = 0.3



$\lambda = 0.0003$ gives the lowest error $MSE = 0.0919$. Increasing λ too much will, as can be seen in Fig 19 increase the MSE as well. It is likely that the regularizer limits the $\boldsymbol{\beta}$ values too much, rendering the model unable to model any relationship.

6 Exercise 5

Lasso regression is similar to Ridge though with another regularizing term. Its cost function is

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \left\{(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda||\boldsymbol{\beta}||_1 \tag{34}$$

When taking the derivative of the cost function

$$\frac{\partial C(\boldsymbol{X},\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{z}-\boldsymbol{X}\boldsymbol{\beta}) + \lambda sgn(\boldsymbol{\beta}) = 0 \tag{35}$$

then reordering leads to an iterative optimization problem

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda sgn(\boldsymbol{\beta}) = 2\boldsymbol{X}^T\boldsymbol{y}. \tag{36}$$

which with SKlearn's Lasso solves with a coordinate descent algorithm.

The sgn function is

$$\frac{d|\beta|}{d\boldsymbol{\beta}} = \mathrm{sgn}(\boldsymbol{\beta}) = \begin{cases} 1 & \beta > 0 \\ 0 & \beta = 0 \\ -1 & \beta < 0. \end{cases} \tag{37}$$

$$\tag{38}$$

Furthermore, using a simple example where the design matrix is a p-by-p identity matrix, so that $\boldsymbol{z} = \boldsymbol{\beta}$, the $\boldsymbol{\beta}$ values becomes

$$\hat{\boldsymbol{\beta}}_i^{\text{Lasso}} = \begin{cases} z_i - \frac{\lambda}{2} & \text{if} \quad z_i > \frac{\lambda}{2} \\ z_i + \frac{\lambda}{2} & \text{if} \quad z_i < -\frac{\lambda}{2} \\ 0 & \text{if} \quad |z_i| \leq \frac{\lambda}{2} \end{cases} \tag{39}$$

$$\tag{40}$$

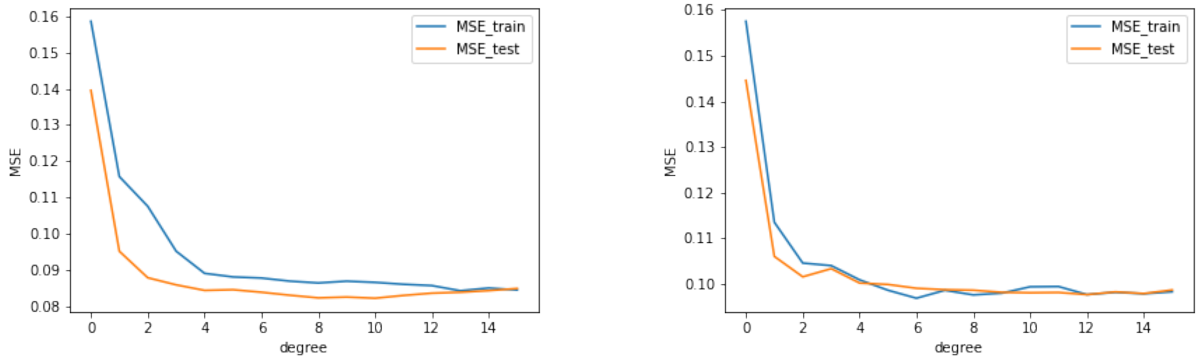
which demonstrates that Lasso regression drives some of the parameters to 0, depending on the λ. Lasso is a method that can lead to sparse problems with few parameters. Again, one should center the data before fitting, so that the intercept is not considered in the regularizing term. As the SKlearn's Lasso will be used in this exercise, and it centers the data by default, it will not be done manually. As such, data scaling is not to be seen in the code.

Running the Lasso regression on FrankeFunction generated dara with different λ

```
            OLS  Lasso(lmb= 0.001)  Lasso(lmb=0.01)
beta
0      0.730345       0.000000e+00         0.000000
1      6.845955      -7.573652e-01        -0.316922
2     -1.971336       0.000000e+00        -0.000000
3    -35.512377      -0.000000e+00        -0.000000
4     -2.046369       2.801188e-01        -0.000000
```

```
5      14.443399      -1.292901e+00         -0.436492
6      58.725544      -0.000000e+00         -0.000000
7      26.538807       6.158866e-01          0.000000
8     -16.770915      -0.000000e+00         -0.000000
9     -40.485385      -0.000000e+00         -0.000000
10    -40.387956      -0.000000e+00         -0.000000
11    -24.600115       6.476399e-07          0.000000
12    -12.039359      -0.000000e+00         -0.000000
13     28.670810      -3.113789e-01         -0.000000
14     41.032572       0.000000e+00         -0.000000
15      9.588534      -0.000000e+00         -0.000000
16      4.772697       0.000000e+00          0.000000
17     11.536409      -0.000000e+00          0.000000
18     -2.684351      -0.000000e+00         -0.000000
19    -13.034043      -0.000000e+00         -0.000000
20    -13.314200       7.128386e-01         -0.000000
```

we see that some of the parameters are in fact driven to zero. The train and test error when using lmb=0.001 are shown in Fig 17b, and compared to the Ridge result.



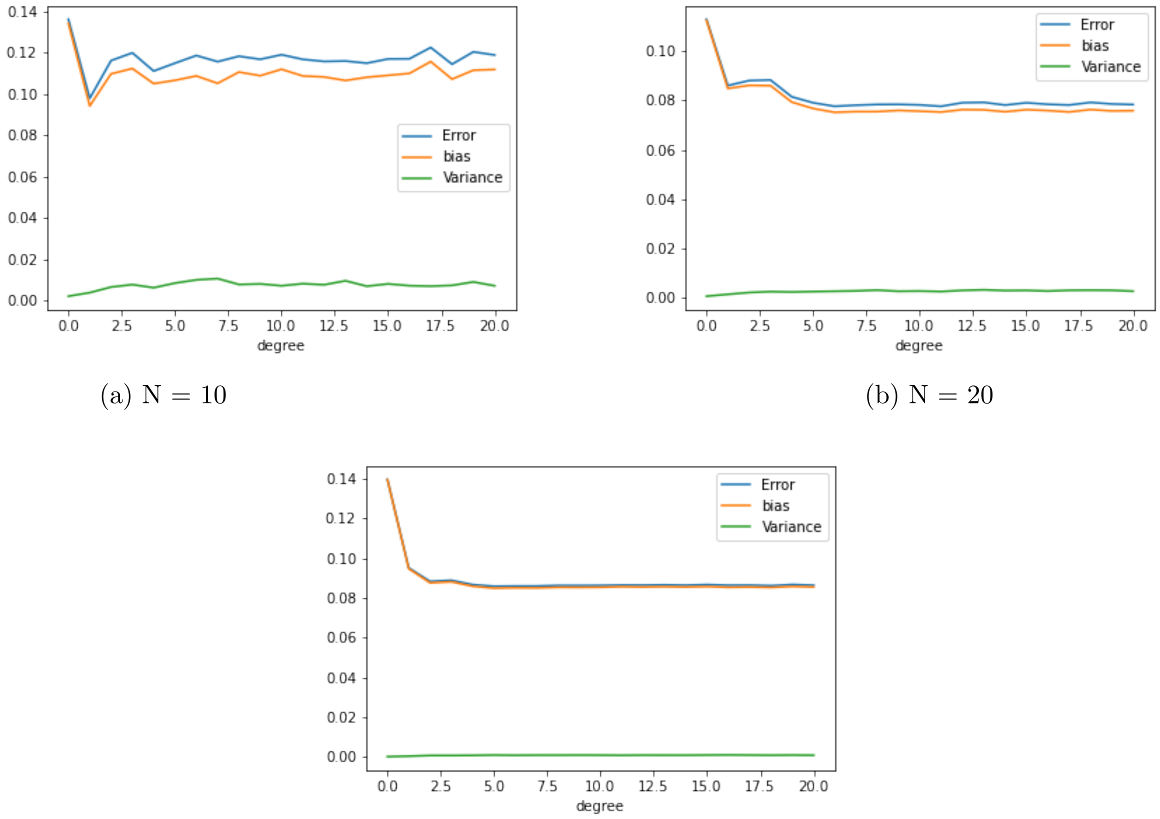
(a) Ridge

(b) Lasso

Figure 17: Results for Ridge vs Lasso: N = 30, σ = 0.3, $\lambda = 0.001$

Lasso too enables higher complexities, however, likely because many the higher degree parameters are driven to zero resulting in a similar model. That may also explain some of the straight horizontal lines in Fig 17b. If the model complexity is increased, but the new parameters are driven to zero, then the model is equivalent to the previous model, resulting in equal MSE. The result may indicate that, with $\lambda = 0.001$, increasing complexity beyond about 4 degrees is redundant.

Performing the same bias-variance trade-off analysis as with Ridge



(a) N = 10

(b) N = 20

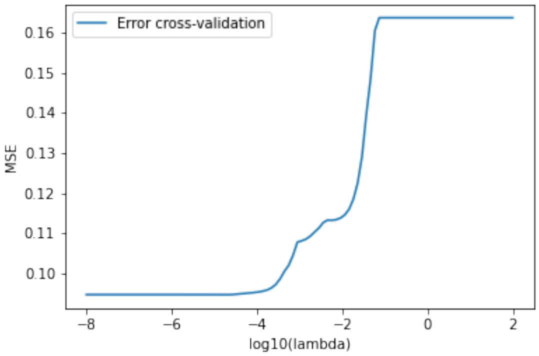
(c) N = 30

Figure 18: Results for Lasso: $\sigma = 0.3$, $\lambda = 0.001$

it can be observed that already with 400 data points(N=20) and $\lambda = 0.001$, Lasso is able pick out enough important features to reach a relatively low test error. Curiously enough, it seems like the model reaches a lower test score with 20 data points as opposed to 30 data points. Again, the stagnating variance might be due to the fact that increasing model complexities simply introduces features that are driven to zero.

Performing cross-validation to optimize λ with same conditions as the Ridge cross-validation

Figure 19: λ analysis: k= 5, N = 30, σ = 0.3



A minimum error MSE= 0.0947 is reached with λ = 2.154e-05. Note that the MSE reaches a max limit, which may indicate that all parameters are driven to zero at this point.

Summary OLS, Ridge and Lasso

From the results is seems that more regularization with Ridge is needed compared to Lasso when degree is 4 and $\sigma = 0.3$. The MSE for all the different methods are very similar and it is unclear which is the better. Ridge and Lasso does allow higher complexities which may be an advantage if dealing with more complex data. One advantage of the Lasso, is that it can help to reduce dimensionality. That is, it can single out those features that aren't needed. Ridge does this in a sense as well. Furthermore, it may be that with the conditions used, N = 30, σ = 0.3 regularization is not very much helpful. Again it must be noted that all data is random generated with the same random seed. It might be an "unlucky" seed in terms of demonstrating the differences of the methods.

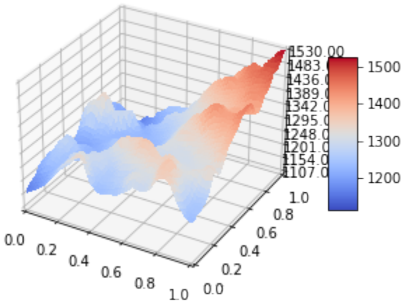
7 Exercise 6

In this exercise some of above analysis will be repeated but with real data. The exercise text ask to assess all three methods, with the same polynomial approximation and cross-validation as resampling technique to evaluate which model fits the data best. As such, design matrix will be created, data will be scaled, and cross-validation will be applied to each model. As methods and reflections are given in the exercises above, this section will focus on the results primarily.

The data

Digital terrain data made available at
https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects/2021/Project1/DataFiles
will be used in this exercise. The data is similar to what has been generated, it describes a surface. The columns and rows are different x and y values, while the value in the matrix itself is the z-value describing the height at the given coordinate. "SRTM_data_Norway_1.tif" is a file containing a large matrix and only a certain segment of it will be used.

Figure 20: Terrain data, N = 100



Scaling should be employed as the data is not pre-scaled as it was in the former exercises. Fig 23 shows the plotted surface after min-max scaling.

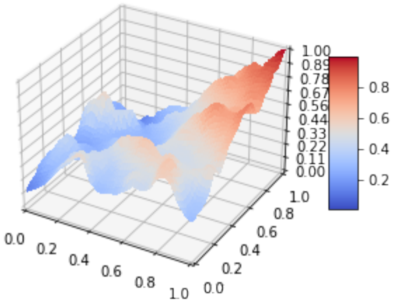


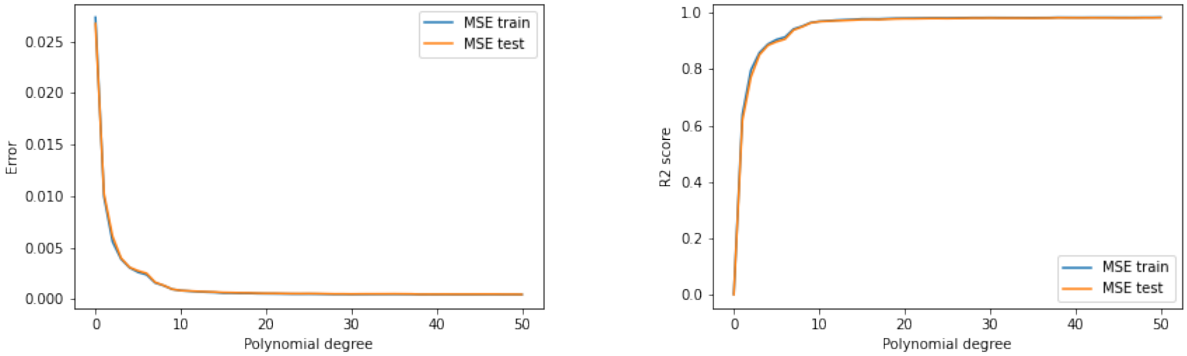
Figure 21: Min-max scaling to [0,1]

Figure 22: Scaling terrain data

Min-max produces a result that resembling of the previous exercises, however, it is unclear which will result in the better performance. Note again that with OLS there is no performance wise reason to scale, though it enables better comparisons. Without scaling, the MSE will obviously also be of a much higher scale. In other words, when trying out the different methods, one must decide to either only measure scaled MSE or inverse transform every prediction to its original scale. The former is chosen such that all data can be pre-scaled with min max scaler. The centering will still be employed when using Ridge and Lasso.

OLS

OLS does show promising result when increasing complexity



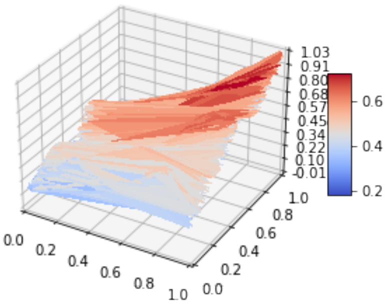
(a) MSE

(b) R2

Figure 23: MSE and R2 score for OLS

it does not seem like an overfit region will arrive any time soon. Both train and test error decreases as complexity increases. It can be seen from the plotted z_tilde using degree 50 that the surface is somewhat similar to the original surface

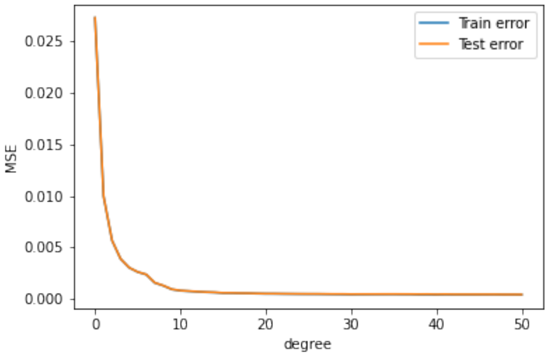
Figure 24: Plotted training data



The MSE reached with degree 50 was 0.00041999.

The cross-validation tells the same tale as former fits.

Figure 25: Plotted training data

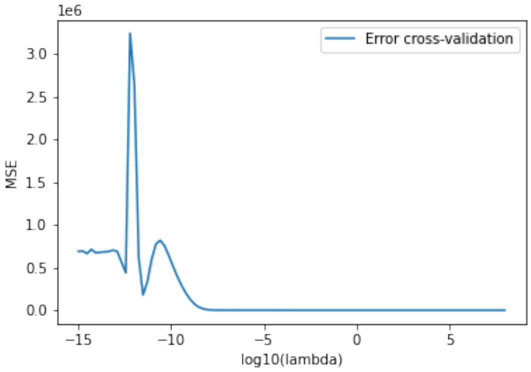


The error is getting lower and lower, both for train data and test data. An explanation might be that both x and y lies in the range [0,1] meaning the higher the degree the lower the number, thereby the lower the addition to the total value. In any case, test error seem to stagnate at some point. As such, increasing degree after that may be unnecessary. Degree 30 is chosen going forward, and an $MSE = 0.000469$ was reached.

Ridge
Degree 30 is chosen to further study optimal λ when using Ridge.

Optimizing λ

Figure 26: MSE as a function of lambda

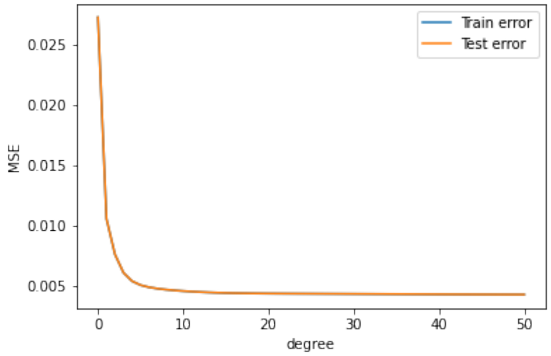


Results show that one can apply a fair share of regulairzation. With $\lambda = 151.99$ a test MSE = 0.006397 was reached. This is somewhat worse than when using OLS. It is unclear as to why, though the linear regression model may not need much regularization.

Train and test error

Running cross validation with λ= 151.99, it can be observed that the higher the degree the better.

Figure 27: Ridge train and test error with cross validation



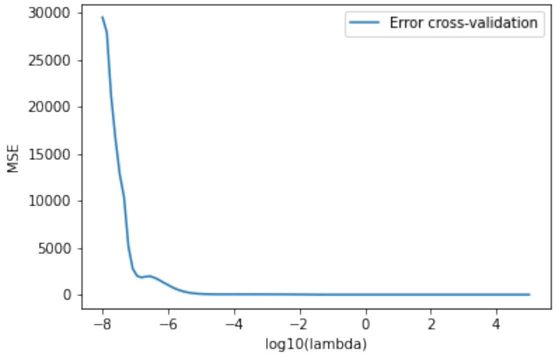
Here too some sort of saturation is reached. Test and train error seems almost identical as well.

Lasso

Degree 30 is chosen to further study optimal λ when using Lasso.

Optimizing λ

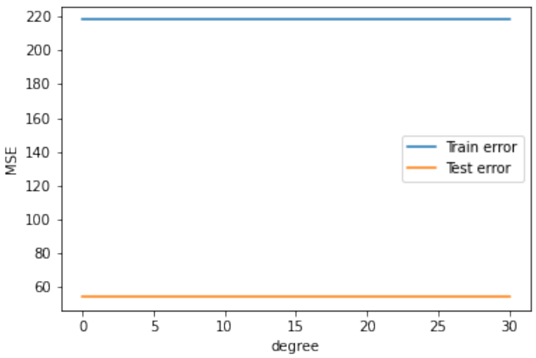
Figure 28: MSE as a function of lambda



Results show that applying some regularization does indeed help. With λ = 0.04977 an MSE = 54.9169, which is quite bad considering that the data is min max scaled to [0,1].

Train and test error

Figure 29: Lasso train and test error with cross validation



It seems Lasso simply stagnates immediately it a very high MSE, where actually the train error is higher.

Conclusion

OLS seems to model the data better than Ridge and Lasso. Ridge is also able to achieve low error, however, Lasso seems not to be a good model for this purpose. It does seem suspicious that the test error stagnates with increasing degree and that OLS reaches better results than Ridge and Lasso. However, functions have been tested up against SKlearn's functions, and they seem to concur. Though, perhaps a much bigger degree is needed to increase the variance. And the model really benefits from being complex and capturing all the intricacies in the terrain. It might also be that a lot of data was given to the model, and test data was within the same area following the same curves. If a larger area is introduced it may be a lot harder to fit the same model to the data.

This application of linear regression can seem somewhat needless. Terrain is rather random in reality, and there aren't really any general relationship between x and y coordinates and heights. It all just depends on where you are in the world, so the linear regression performed here may be more of an exercise. However, there might be some applications in more efficient storage of terrain. Instead of storing each and every coordinate, simply the beta values can be stored and the terrain can be generated afterwards. This could possibly have applications in game development for continuous generated landscape of some sort that resembles real landscapes.