Modelling two dimensional data using linear regression

Lars Opgård

October 8, 2024

Abstract

In this project I used the three linear regression methods, *Ordinary Least Square*, *Ridge* regression and *Lasso* regression too model 2 dimensional datasets. I first tested each method on the Franke's function, optimizing the model by implementing preprocessing, in the form of data scaling using the StandardScaler from Scikit-learn (**?**], and the complexity of the model by changing the order of the polynomial. Resampling methods such as the *Bootstrap* and *Cross-validation* methods. I found that the regularization of the Ridge and Lasso methods had an improvement on the results. Though a problem with the sci-kit learn Lasso regression method where the method did not converge even after giving it much more iterations to work with, this made the method have worse results compared to both the other methods, and

The after I had done all of the tests on the Franke's function I redid our previous steps on real terrain data giving slightly worse results compared to the Franke's function as epxected, but also the results were not very good at describing the terrain. To acheive better results we would need to increase the complexity of the model, something I did not have the time or processing power to do.

Contents

1 Introduction

Modeling terrain can be very useful in many different fields, whether it is scientific, development geological surveys. There exist many methods for creating accurate models such as *kriging* (6], *splines* [8] and *neural networks* (7]. I want to use a simpler method, *linear regression.* While linear regression is a less complex method I must take into consideration the complexity of the model. A too complex model might give good results, but at the cost of computation time and flexibility. While simpler model create worse results for cheaper computations.

The goal of this article is to achieve the most optimal model for 2 dimensional datasets. This will be done using three different regression methods, *Ordinary Least Square* (OLS), *Ridge* regression and *Lasso* regression. While also implementing other methods for optimizing such as optimizing the hyperparameters, using the *Bootstrap* method and preporocessing of data. By optimizing the hyperparameters using *Cross Validation*, I mean the complexity of the model, the order of the polynomial and for Ridge and Lasso the additional λ parameter. I will then preprocess the data by potentially scaling both the features and target data. Before I started creating models on real terrain data, I used a more simple case. First I tested each method with every optimization on data produced with the *Franke's* function. This 2 dimensional function gave us a simpler problem too solve.

The article will start with doing some analytical calculations on some of the regression methods. While also calculating the means squared error using the optimized parameters β and finally calculate the Bias-Variance trade-off. Then follows the method explaining the code, explaining the main algorithm and how the tests are done. The results and discussion follows explaining the results and our findings. The article concludes with a conclusion.

2 Methods

Artificial Intelligence (AI) is a massive and ever growing field today. This field concerns itself with building computers or machines that can reason or learn in a similar method as human intelligence or analysing large amount of data. Machine Learning (ML) have grown to become the largest field within the AI field as of the last few years. While the theory behind machine learning is many decades old, the recent growth comes from improved hardware allowing these computationally expensive algorithm to thrive.

2.1 Linear Regression

While simple compared to other other types of ML, linear regression can be quite powerful in certain situations. Linear regression is a form of Supervised learning, that means the model tests the output with desired results. The linear regression model comes in the form

$$f(\mathbf{X}) = \beta_0 + \sum_{j=0}^{p-1} X_j \beta_j. \tag{1}$$

Where $\boldsymbol{\beta}$ is a set of unknown parameters that the model intend to optimize and $\mathbf{X}$ is the input features. The input features $\mathbf{X}$ can come in many shapes, such as quantitative inputs or the transformation of such inputs, they can also be basic expansions or interaction between variables. The true model I want to predict is given by the predicted model plus an error term ϵ.

I want to minimize this term such that the predicted model is as close to the true model, this is done by minimizing the loss or cost function $C(\boldsymbol{\beta})$ which for example with ordinary least squares is given by

$$C(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - f(X_i))^2. \tag{2}$$

Where $\mathbf{y}$ is the true model. To minimize the loss function I will need to find the optimal parameters $\boldsymbol{\beta}$. The efficiency of the regression will depend on the model, with the aforementioned ordinary least square being the simplest. Minimizing the loss function is crucial in machine learning and is the main learning part of the model. While other methods like neural networks rely on complex methods to find the optimal values to minimize the loss function, linear regression uses simple linear algebra allowing models like ordinary least squares and Ridge to have an analytical solution. The computation for the optimal values for Ridge computed from the loss function can be found in appendix B. While the solution for the optimal value for Lasso regression is not able to be found using analytical computations, but rather using numerical calculations, I only use the method from sci-kit learn [1].

There exists ways to estimate how well a model is working. First we need a way to evaluate a model, where two methods are, R-squared (R^2) and means squared error (MSE). The R^2 method provides a measure for how well observed outcomes based on total varaition based on the true model. The R^2 is a value between 0 and 1, where the closer the score is to 1, the better the model is at fitting. The score is given by

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1} (y_i - f(\mathbf{X}_i))^2}{\sum_{i=0}^{n-1} (y_i - \bar{\mathbf{y}})^2}, \tag{3}$$

where $\mathbf{y}$ is the true mode, $f(\mathbf{X})$ is the predicted model and $\bar{\mathbf{y}}$ is the mean of the true datapoints. The MSE measures the average difference between the true model and predicted model squared. The MSE is given as

$$MSE = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - f(\mathbf{X}_i))^2. \tag{4}$$

Improving our model prediction can be done in multiple steps, these steps includes preprocessing of data and resampling. Preprocessing the data can in most cases improve the results, this involves scaling the input features or in some cases also scaling the true data in supervised learning. There exists many different methods for scaling data, of those the StandardScaler from sci-kit learn is one of the most used ones. This scaler works by setting the mean to zero and rescaling them by how many standard deviations away from the mean the datapoint is. But what benefit could there be from scaling the input data, in linear regression scaling the input can prevent the dominance of large features from dominating the others. While with regularization techniques like Ridge and Lasso regression, larger features are penalized more than others.

2.1.1 Ordinary Least Squares

Ordinary least squares (OLS) is the simplest form of linear regression I will use and a widely used on at that. The goal is to find the optimal parameters $\boldsymbol{\beta}$ by minimizing the loss function. On matrix form the optimal parameters can be found as

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

No matter what shape the parameters $\mathbf{X}$ have, the model is linear [2].

2.1.2 Ridge Regression

OLS usually have a problem with overfitting, this could be due to the optimal parameters $\boldsymbol{\beta}$ can grow very large and while this can be counteracted by preprocessing the data first there exists other methods that handles these values better. This can be done by adding a term to the loss function. This new term will grow with the $\boldsymbol{\beta}$ values. Ridge regression solves it with the following cost function

$$C_{\text{Ridge}}(\boldsymbol{\beta}) = \sum_{i=0}^{n-1}(y_i - \mathbf{X}_i\boldsymbol{\beta})^2 + \lambda\sum_{j=0}^{p-1}\beta_j^2. \tag{6}$$

Where the new term is simply the norm of β squared $\|\beta\|^2$. With the new parameter λ we can set how much the model will affect the β values. Following the computation in append B we can find an analytical solution for the optimal parameters [5]

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{7}$$

2.1.3 Lasso Regression

Another similar method to Ridge regression is Lasso regression, with the slight difference that you instead use the l1 norm instead of the l2 norm. The new loss function is then

$$C_{\text{Lasso}}(\boldsymbol{\beta}) = \sum_{i=0}^{n-1}(y_i - \mathbf{X}_i\boldsymbol{\beta})^2 + \lambda\sum_{j=0}^{p-1}|\beta_j|. \tag{8}$$

This equation is however more difficult compared to Ridge regression as the the derivative of l1 norm is not continuous, therefore we cannot find an analytical solution and must instead use numerical methods to solve it. Therefore in this article I will only use Lasso regression from sci-kit learn rather than implementing it myself. Though similarly to Ridge regression we can control the model with the use of the λ parameter. (4]

2.2 Resampling

Resampling is the idea of generating additional data from existing datasets too improve the error of the model. This improves the error and gives us a better measure of how well our model actually is without requiring large amount of data. There are many different resampling methods, but they generally work by creating new datasets from one existing dataset. This improves the error due to the model training on slightly different datasets, giving slightly different results. Then taking the average of each of these training sessions will make it seam like we trained on much more data than we had available. (3]

2.2.1 Bootstrap

Bootstrap is a simple method of resampling by resampling with replacement from the original dataset. Bootstrapping have a few benefits such as being quite general making it for the bootstrap to fail, it does not require distributional assumptions allowing it make improvement even when the data size is small. It is simple to apply bootstrap to difficult distributions and complex datasets.. Though the method is computationally expensive, requiring a large number of bootstrap samples, particularly for large and complex datasets. The method is also quite sensitive to outliers giving it more bias towards extreme values. Finally if the original data sample is bad, the bootstrap might lead to misleading estimates.

2.2.2 Cross Validation

Cross validation works slightly different from bootstrap in that instead of generating new samples from the existing dataset, cross validation instead splits the dataset into k folds and then over k iterations picks one fold for testing and trains the model on the rest of the folds. Then taking the average of the results from each fold, improving the error in many cases. A special case is leave-one-out-cross-validation (LOOCV) where the number of folds k equals the size of the dataset making it such that for each iteration there is only one observation used for testing. The advantage of this is that every training point is used for training reducing the bias, but at the cost of being more computationally expensive. The benefits of cross validation is a more accurate estimate of the model performance, reducing the overfitting by validating the model on multiple splits making it easier to find overfitting, and it makes much more efficient use of data due to training on more the dataset. Cross validation is limited by being computationally expensive especially with large datasets, due to having to train multiple times, one time for each fold. The performance can vary depending on how the data is split, especially if the dataset is small or imbalanced. Adding more fold will help in this case at the cost of computation time.

2.3 Bias Variance Trade-off

While finding the optimal parameters for a given dataset does not mean we have a good model. We want the model to be versatile making it applicable to different datasets. We do not want to train a new model for every dataset as that is very inefficient and computationally expensive. This will gives us a balance between simple and complex models, since we obviously does not want a too simple model as that will not be able to describe our data. While a too complex model will lose its versatility making it worse with different datasets. This is whats called under- and overfitting, while the concept is named the Bias-Variance tradeoff. The bias tells us how well the model on average is compared to the true model or how accurate the model is, a high bias tells us that we have underfitted our model. While the variance is the precision of our model and tells us our model fits to the true datapoints, high variance leads to overfitting, meaning our model is taking into account the noise or irrelevant patterns of the training data too much making it preform poorly on any other datasets. The tradeoff comes in that the total error, which we want to minimize cannot be lower than the bias and variance added together. This means if we want to minimize the error, we need to have a model that is neither to complex or to simple. The computation behind the trade-off can be found in the appendix C.

2.4 Implementations

The problem comes in two main steps, first the testing of methods using the Franke's function, and secondly the application of the model on real terrain data. I will use the three linear regression methods, OLD, Ridge and Lasso regression and compare them against eachother in different situations and with usage of preprocessing and resampling methods. The input features will come in the form of 2d polynomial (x and y dependence) with interaction, this gives us $X^T = [x, y, x^2, y^2, xy, \dots]$. The Franke's function is given as

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

Preprocessing the data is quite important in machine learning as small tweak can make the learning easier and get much better results. In this problem I will look into how scaling the data will affect how the model can fit the dataset. Also I will split the dataset into a training and testing set, using sci-kit learn train test split method with 20 % of the data being in the test set. It is usual to split the data into either $\frac{4}{5}$ or $\frac{2}{3}$ data as training. Some time ago it was also normal to split into three sets, with an additional third evaluation set, but this was mostly replaced by new methods such as cross validation.

The first step is to implement Franke's function using the input data, and then compute the optimal parameters $\boldsymbol{\beta}$ using first OLS, then Ridge and Lasso. From the prediction I will estimate the mean squared error and R-squared, and see if scaling the data beforehand will have any effect. I have solved the OLS and Ridge regression analytically B, but will compare them with the methods from sci-kit learn [1]. I will first look at the complexity of the model and see how that affect the mean squared error and the optimal parameters β. Then I will use bootstrap to look into the bias variance trade-off and see how using resampling can give improved results. Cross validation will be used mainly for the grid search, but also compare it as a resampling method with bootstrap.

At the end I will use the methods implemented for the Franke's function on real terrain data. Using what I learned to hopefully create a good fit.

3 Results and Discussion

As mentioned the results are split up into two parts, first where I tested the model on Franke's function. This allowed me to test the model and resampling methods on a simple problem before tackling something as complex as terrain mapping. Here I focus mostly on the difference between each regression method, their good and bad, and also how the different resampling methods affect each method. After I have tested all the methods on the Franke's function, I can implement each method with cross validation on real terrain data.

3.1 Franke's Function

3.1.1 Ordinary Least Squares

Starting with the simple OLS, I first checked how the complexity of the model affected the results by plotting the MSE 4 and R2 score 3. By going up to a 5th order polynomial, with zero noise in the Franke's function the results would only improve with the complexity of the model. As the complexity of the model increases, overfitting starts becoming visible in the error. From the plots 1 you can see that there is an additional peak with my own implementation, Other than that the two methods follow eachother closely. If I where to increase the number of datapoints, the overfitting would be reduced.

The R2 score tells a similar tale to the mean squared error, with the difference being it approaches 1. Therefore it becomes a good way of and from the plots 2 you can see that it mimics that of the error plots.

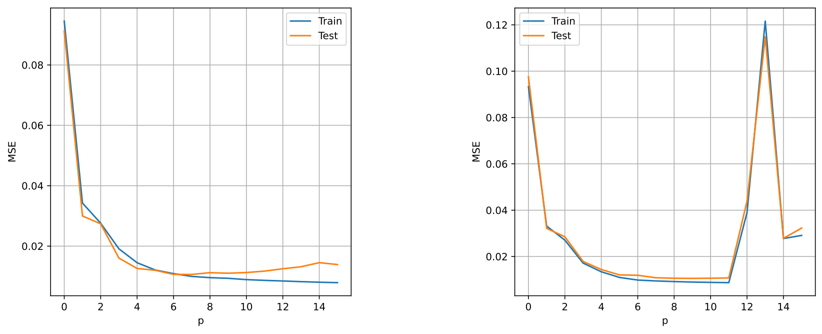

Figure 1: Plots showing how the MSE changes using ordinary least squares from sklearn (left) and own implemented function (right) with the complexity of the model for the Franke's function without any noise. The plot using 30 datapoints for x and y. Overfitting can be slightly seen for higher orders of polynomial.

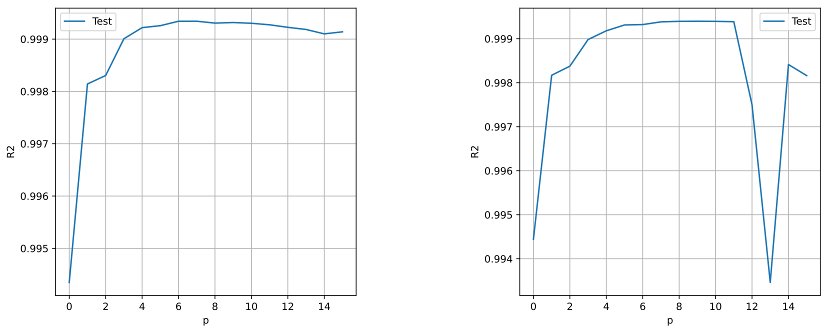

Figure 2: Plots showing how R2 score changes using ordinary least squares with my own implementation (right) and sci-kit learn (left) with the complexity of the model for the Franke's function without any noise. The plot is made using 30 datapoints.

I can then also plot the optimal β values against the complexity of the model. I produce a heatmap where each order of polynomial shows all of their corresponding β values. As seen from the plot the optimal values are fluctuating more for higher order polynomial.

As mentioned increasing the amount of data reduces the risk of overfitting a model. Therefore I will now introduce first the bootstrap method allowing us to do more with less data. From the plot 4 you find the Bias Variance trade-off. This allows us to better see where the error comes from, with the low complexity error, the bias is generated due to the simplicity of the model, i.e. underfitting. Here you can also see that with resampling I can reduce the overfitting of the model due to being able to use more data for training. But the model is still getting overfitted. The bootstrap is only done on the sci-kit learn model, just

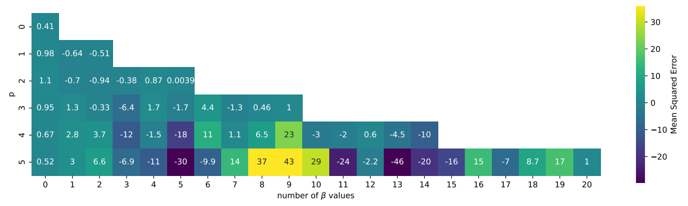

Figure 3: This figure shows how the β parameters changes with the increased order of polynomial. As the complexity increases so does the extreme values for β.

for ease and considering the similarity of the results.

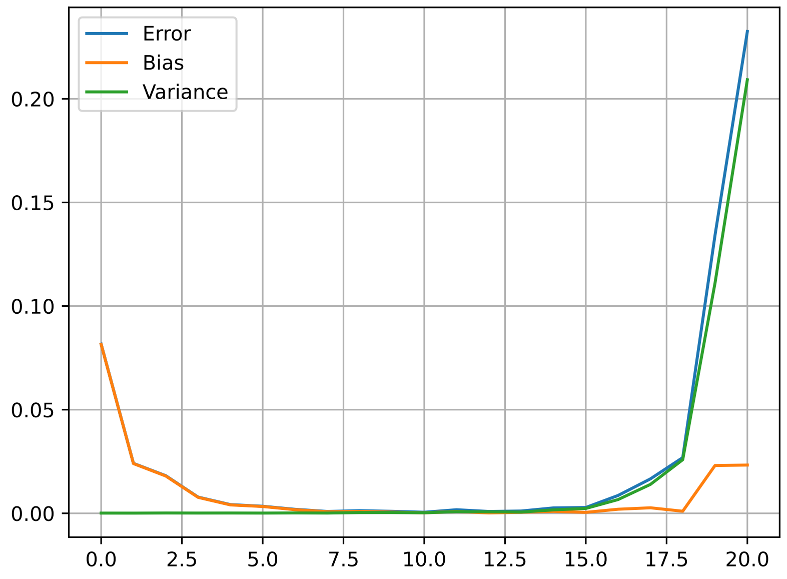

Figure 4: Bias Variance trade-off for Ordinary Least Squares. Both underfitting (high bias) and overfitting (high variance) can bee seen from the plot. The plot is made using resampling with bootstrap with 100 iterations.

Before I start with the other regression methods, I will use grid search together with cross validation to find optimal parameters for the model. The results at different noise levels is given in table 3.1.1.

Noise	Best p	Best MSE
0	6	0.0059
0.05	8	0.0063
0.1	6	0.018

3.1.2 Ridge Regression

With Ridge regression comes a new parameter to think about. Making new plot for the MSE 5 and the R2 score 6, you will probably first notice that the model is not overfitting for higher order polynomial. This means that the regulariztion that is done with Ridge and Lasso regression is working well. Compared to OLS, Ridge regression also reduces the error slightly.

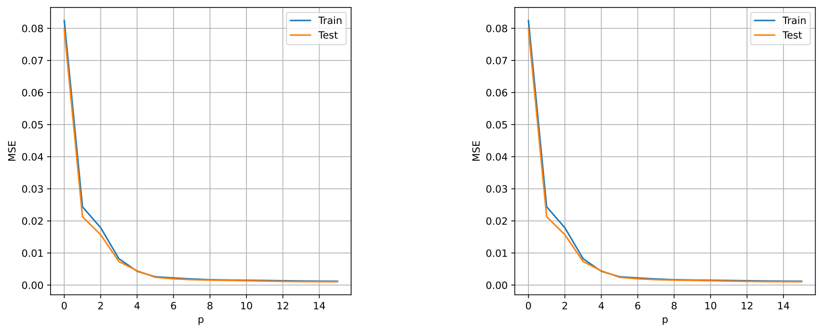

Figure 5: The MSE plot as a function of the polynomial order for Ridge regression, with both sci-kit learn (left) and my own implementation (right). Both plots are made with 30 datapoints and an arbitrary chosen $\lambda = 10^{-4}$.

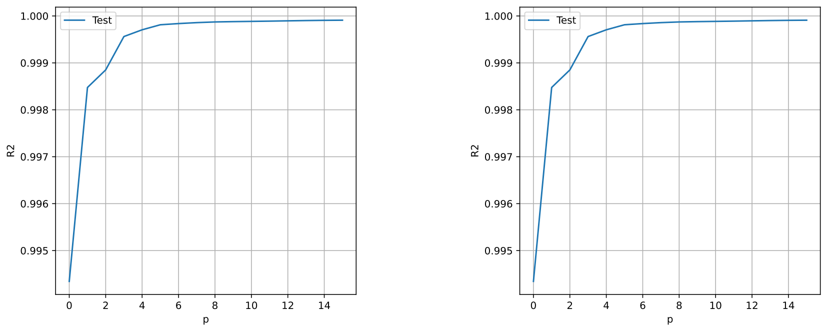

Figure 6: The R2 plot as a function of the polynomial order for Ridge regression, with both sci-kit learn (left) and my own implementation (right). Both plots are made with 30 datapoints and an arbitrary chosen $\lambda = 10^{-4}$.

Again using the bootstrap method with Ridge regression, you can see that the variance is not even increasing and I have no danger of overfitting the model on the function. Adding noise to the Franke's function also have negligible effect on the overfitting, though the error does increase a bit. This makes the Ridge regression a lot more stable compared to OLS. Again using grid search to optimize the hyper-parameters, I can find the optimal values as shown in table 3.1.2. Compared with OSL, while I do find an improvement, the mean squared error is actually slightly higher with some noise. Also while the Ridge regression likes similar order of polynomials as OLS, it also takes small values of λ. From the heatmap as shown in 9, the cross validation gets more sensitive towards λ for higher orders, compared to the bootstrap **??**

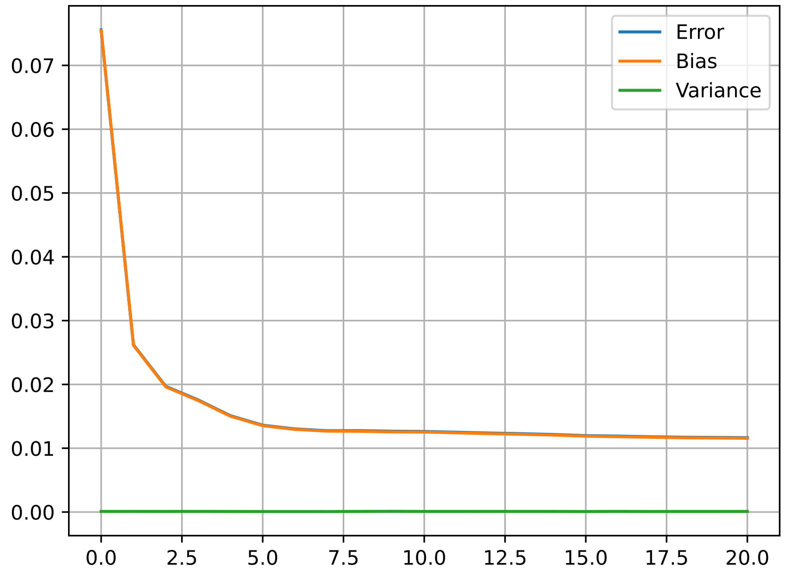

Figure 7: The bias variance trade-off for Ridge regression using sci-kit learn's implementation. As Ridge regression regulates the β values, the model is less likely to overfit. The plot is made with 30 datapoints, λ is chosen as 10^{-4} and 100 bootstrap iterations.

Noise	Best p	Best λ	Best MSE
0	9	10^{-8}	0.0044
0.05	6	$7.85 \cdot 10^{-7}$	0.0078
0.1	6	$7.85 \cdot 10^{-7}$	0.014

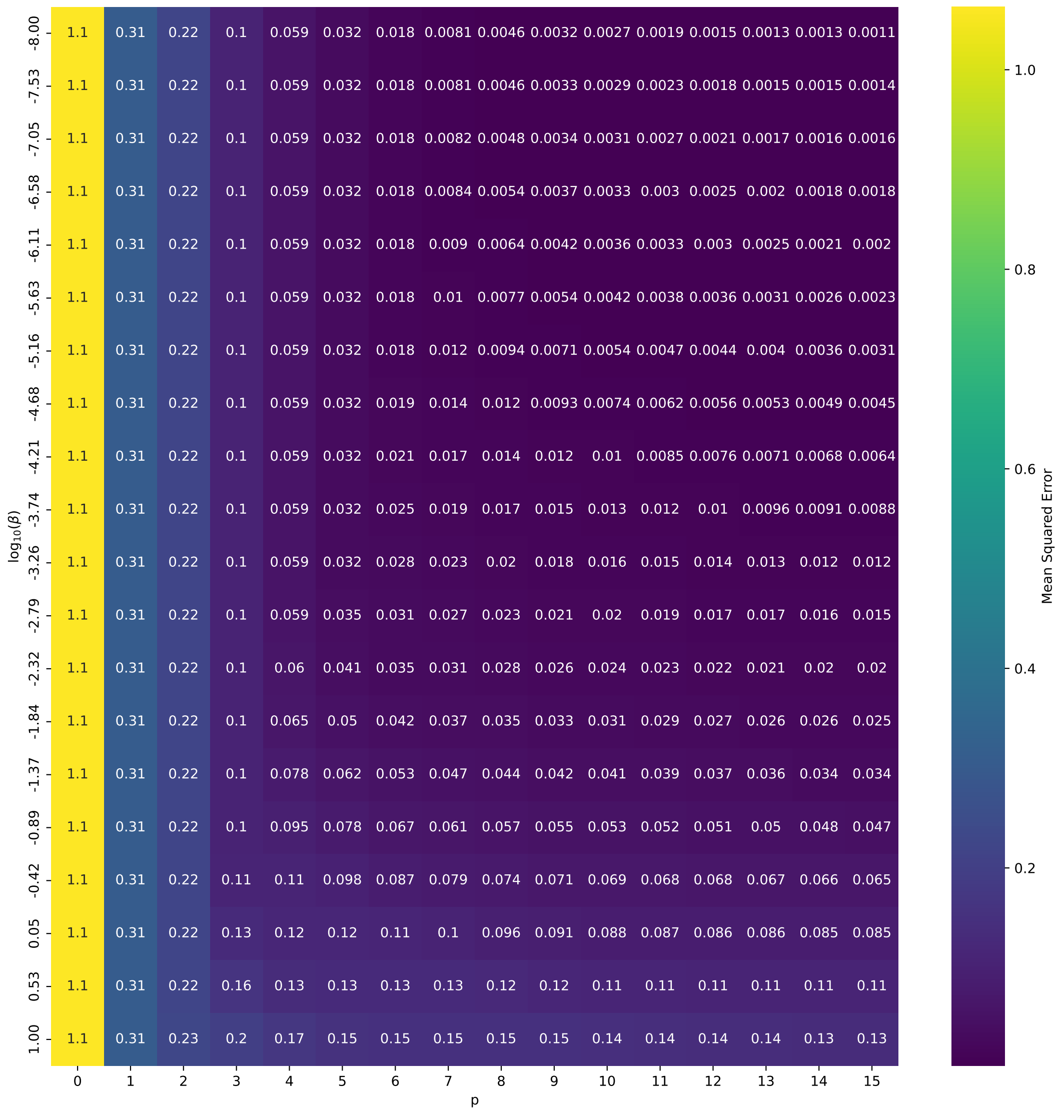

Figure 8: Heatmap showing the MSE as a function of the complexity of Ridge regression using sci-kit learn. Due to the amount of data, number of bootstrap iterations and simple dataset. The model shows no sign of overfitting with the error reducing with small values of λ.

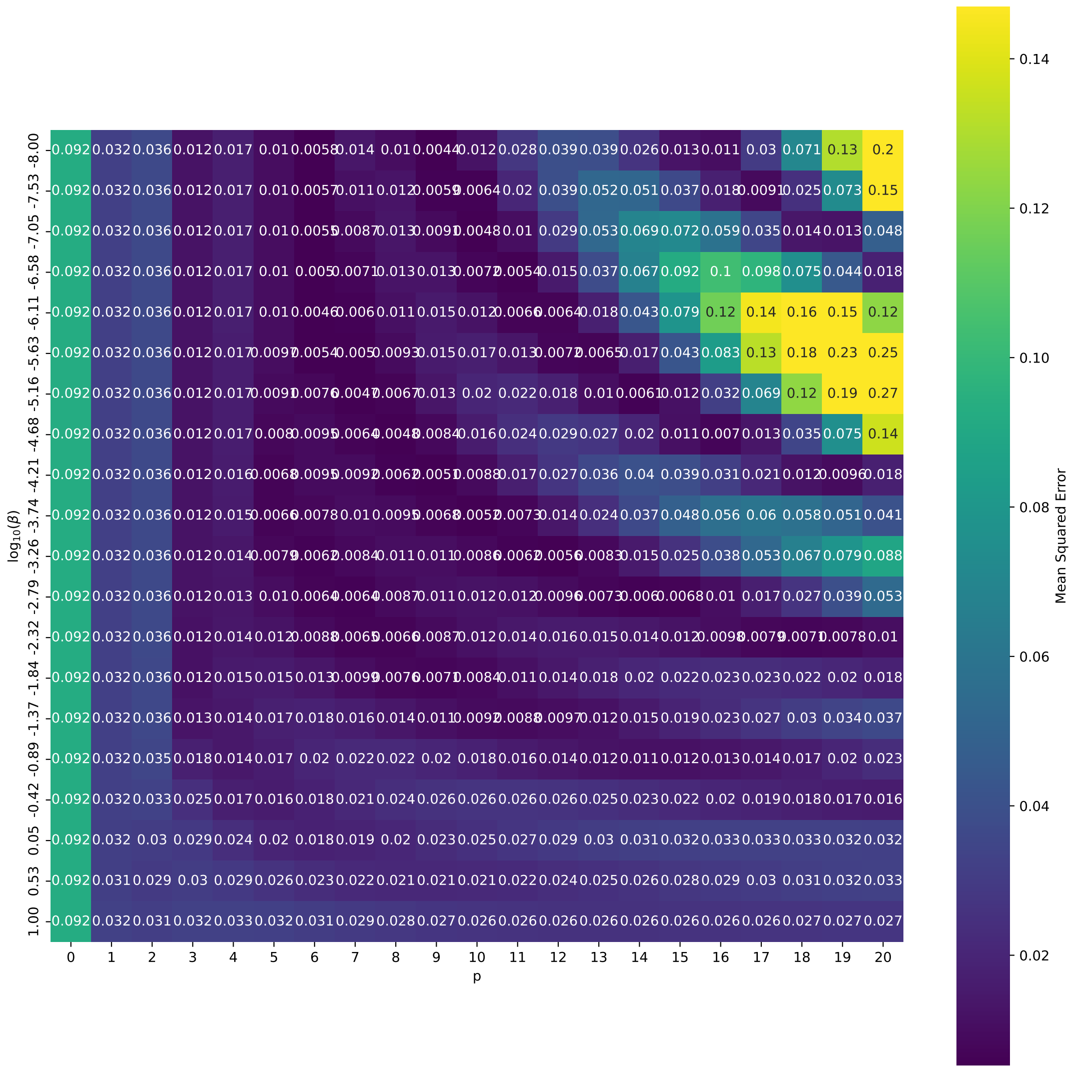

Figure 9: The heatmap from cross validation with Ridge regression. Using 10 folds the model here is results is much more sensitive to the λ parameter compared with bootstrap. With the error peaking much higher at certain λ values for high order polynomial. Interestingly the error have multiple minimum and maximum points.

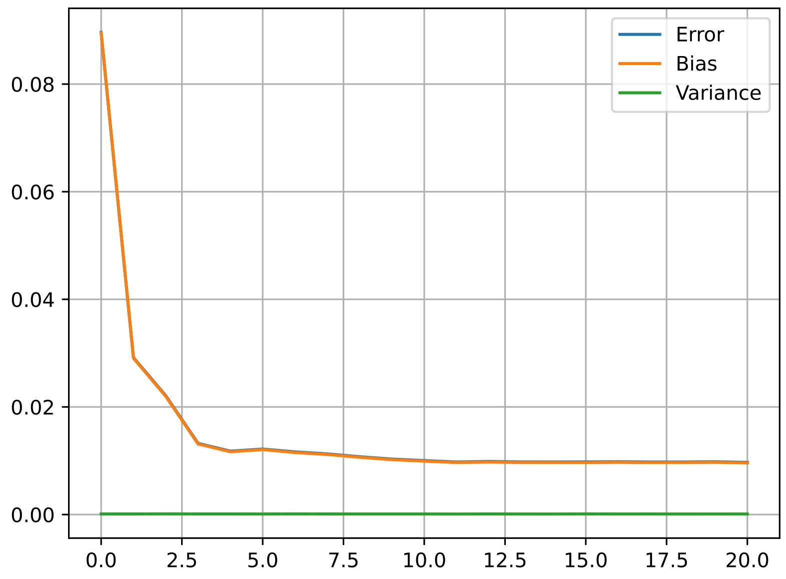

Figure 10: The bias variance for Lasso regression. As with Ridge regression, due to regularization the model shows little sign of overfitting at high complexity. The plot is made with 30 datapoints, $\lambda = 10^{-4}$ and 100 bootstrap iterations.

3.1.3 Lasso Regression

As Lasso follows a similar methodology as Ridge, I do not expect much difference in the methods. One problem I had with this method was that I left it out for last, focusing mostly on the other methods, OSL and Ridge due to the similarity to Ridge regression. This meant that I was not able to solve a problem with the function for sci-kit learn where the method had problems with converging. I tried to increase the number of iteration hoping it would at some point converge or look up a solution, but found nothing. While doing normal fits with orders of the polynomial ended up with errors growing after just the 1st order, but with bootstrap the results was similar to what I found with the other methods. As seen from the bias variance plot 10, while the error is slightly larger than for Ridge regression, it follows the same shape.

Even with then, I still made use of cross validation and grid search to find the optimal values for the current Lasso model. Which even without any noise gave an error that was a magnitude larger than both OLS and Ridge. The grid search also gave values for the parameters, while larger than for the other methods still followed a similar path. Though interestingly the error was not as affected compared to the other methods and with more noise Lasso gave error in similar scale as both the other methods.

Noise	Best p	Best λ	Best MSE
0	14	$6.16 \cdot 10^{-5}$	0.012
0.05	12	$6.16 \cdot 10^{-5}$	0.014
0.1	11	$2.069 \cdot -5$	0.021

3.2 Real Data

Now that we have tested each method on a known system, I can start with implementing it for real data. Due to the large datafiles for the terrain and the limited computation time I have at hand, I will not fit the model to the entire dataset, but rather to a small corner with shape (200,400). Let's start with seeing if scaling the data beforehand has an effect, which I expect it will have due to the large values in the dataset. Just checking the error for small order of polynomial, up to 5th order, gave a massive benefit from scaling. Just the MSE went down with multiple magnitudes and the beta values went from 3 orders of magnitudes down to what we found earlier. **??**, **??**. I will then use scaled data from here on

Checking the complexity of the model, I found compared to the Franke's function the model needs to be a lot more complex to reach the minimum error as expected, as real life generally is more complicated than fake data, especially a simple one like the Franke's function. Having done the same with Ridge regression, it follows the same reason as with OLS that a higher complexity is required. The plots 11 shows the error as a function of the polynomial order. Neither methods shows any sign of overfitting with my current complexity, this shows the complications with real data compared to the simulated function.

I the used grid search with cross validation to find the optimal values for the parameters as well as the error. First due to how my grid search works with a custom estimator and how the grid search takes in the whole dataset makes it harder to scale the data than normally. I tried to input a pipeline including the StandardScaler with my estimator, but the results did not improve as much, which can be many different reasons as I do not know how the scaling is done. What I ended up doing is scaling the entire dataset before sending it into the grid search. While this did give better results, the difference compared to running the fits without grid search is noticeable showing that scaling based on the split data does have an advantage. Compared with Franke's function. From the table 3.2 you can see that the error of the real data is comparatively with a std noise of 0.1 for the Franke's function. Then figure 12 shows the fitted model with the real data. As you can see the model still have a lot to do before it can reliably fit terrain, but even then the overall shape is still visible. Unfortunately due to the converging error with Lasso regression, I was not able to produce any results as the computation time was to long and it had problem working.

Method	Best p	Best λ	Best MSE
OLS	6	-	0.21
Ridge	14	$5.46 \cdot 10^{-4}$	0.16

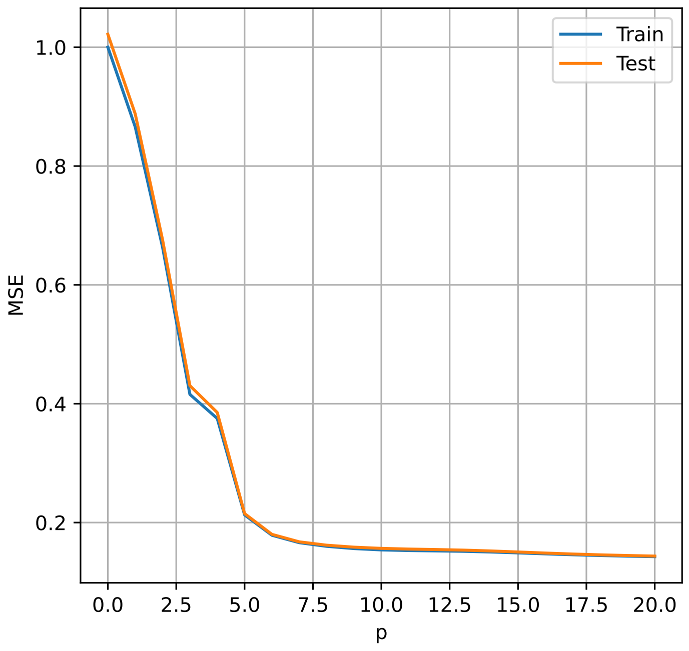

Figure 11: The MSE as a function of the polynomial order with Ridge regression on real terrain data. Comparatively to Franke's function the error is of a magnitude larger and require a more complex model to reach minimum error. The plot is made with 200 x datapoints and 400 y datapoints with $\lambda = 10^{-4}$

4 Conclusion

In this article I have used three linear regression methods, Ordinary Least Square, Ridge and Lasso regression on both simulated fake data and real terrain data. Starting with the fake data, I tested each method with different rescaling methods, bootstrap and cross validation. On the fake data I achieved good results with the methods, expect for Lasso regression where a problem with the method convergence from sci-kit learn made the data hard to interpret the results. Though the real data was more difficult to work with due to the complexity and scale of the dataset. While the results only a magnitude higher than for the Franke's function, it was comparatively with a std noise of 0.1. Unfortunately the model does not work as well on real data, though improvemetns could be made, such as using more of the data and improving the complexity of the model. But that would require much more compuational power and time, which I do not have. Another improvement is with the data scaling in the grid search, I believe if the data was scaled separately for each fold the results could get better. Another and more interesting approach is searching for a different method all together. One interesting method that have become very popular

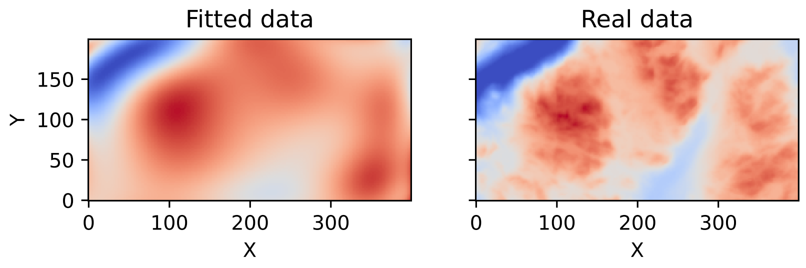

Figure 12: Figure shows the fit of a small part of the real terrain data with the optimal parameters found in the grid search with Ridge regression. While the the plots shows some similarity, the detail is lacking. This is due to how I cannot use a more complex model because of the large and complicated dataset.

over the last few years and show great results in many field is Neural Networks. I mentioned it a bit at the beginning, but there might lie something great there.

References

[1] Pedregosa, Fabian u.a.(2011): *Scikit-learn: Machine Learning in Python*, 85: 2825–2830.

[2] Hastie, Trevor / Tibshirani, Robert / Friedman, Jerome (2001): *The Elements of Statistical Learning*. New York, NY, USA, Springer New York Inc.

[3] Hjort Jensen, Morten. (2024): *Applied Data Analysis and Machine Learning: Resampling Methods*
, https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter3.html.

[4] Hjort Jensen, Morten. (2024): *Applied Data Analysis and Machine Learning: Ridge and Lasso Regression*
, https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter2.html.

[5] Wieringen, Wessel N. van (2023): *Lecture notes on ridge regression*
, https://arxiv.org/abs/1509.09169.

[6] Wikipedia (2024): *Kriging*
, https://en.wikipedia.org/wiki/Kriging.

[7] Wikipedia (2024): *Neural network*
, https://en.wikipedia.org/wiki/Neural_network.

[8] Wikipedia (2024): *Spline (mathematics)*
, https://en.wikipedia.org/wiki/Spline_(mathematics).

A GitHub

The code for this rapport can be found in this GitHub repository: `https://github.com/larsfop/FYS-STK4155---exercises`

B Ridge parameters

Show that the optimal parameters for Ridge regression $\boldsymbol{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$. Where the cost function for Ridge regression is

$$C(\boldsymbol{\beta})_{\text{ridge}} = \frac{1}{n}[(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}] + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{10}$$

To find an expression for the optimal parameters we need to take the derivative of the cost function with regards to $\boldsymbol{\beta}^T$, we know that the derivative of the OLS (ordinary least square)

$$\frac{\partial C(\boldsymbol{\beta})_{\text{OLS}}}{\partial \boldsymbol{\beta}^T} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{11}$$

Then we can simply take the derivative

$$\frac{\partial C(\boldsymbol{\beta})_{\text{ridge}}}{\partial \boldsymbol{\beta}^T} = -\mathbf{X}^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta} = -\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} \tag{12}$$

which gives us the analytical solution for the optimal parameters $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{13}$$

C Bias Variance Trade-off

With the true model given by $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \boldsymbol{\epsilon}$, where $\mathbf{f}$ is the true function we want to fit and $\boldsymbol{\epsilon}$ is some noise following a normal distribution. Our fitted model is given by the design matrix $\mathbf{X}$ and the parameters $\boldsymbol{\beta}$ as $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$. We want to rewrite the cost function $\mathbf{C}(\mathbf{X},\boldsymbol{\beta}) = E[(\mathbf{y}-\tilde{\mathbf{y}})^2]$ as

$$E[(\mathbf{y}-\tilde{\mathbf{y}})^2] = \text{Bias}[\tilde{\mathbf{y}}] + \text{Var}[\tilde{\mathbf{y}}] + \sigma^2, \tag{14}$$

where

$$\text{Bias}[\tilde{\mathbf{y}}] = E[(\mathbf{y}-E[\tilde{\mathbf{y}}])^2], \tag{15}$$

and

$$\text{Var}[\tilde{\mathbf{y}}] = E[(\tilde{\mathbf{y}}-E[\tilde{\mathbf{y}}])^2]. \tag{16}$$

We start with writing out the cost function

$$E[(\mathbf{y}-\tilde{\mathbf{y}})^2] = E[(\mathbf{f}+\boldsymbol{\epsilon}-\tilde{\mathbf{y}})^2] = E[(\mathbf{f}-\tilde{\mathbf{y}})^2] + 2E[(\mathbf{f}-\tilde{\mathbf{y}})\boldsymbol{\epsilon}] + E[\epsilon^2]. \tag{17}$$

Where we have used that $E[A+B] = E[A] + E[B]$, and from earlier exercises we have shown that the middle term with $\boldsymbol{\epsilon}$ is equal to zero and the third term $E[\boldsymbol{\epsilon}] = \sigma^2$. We will then focus on the first term, starting with adding $\pm E[\tilde{\mathbf{y}}]$,

$$E[(\mathbf{f} - \tilde{\mathbf{y}})^2] = E[(\mathbf{f} - \tilde{\mathbf{y}} + E[\tilde{\mathbf{y}}] - E[\tilde{\mathbf{y}}])^2]. \tag{18}$$

We will then expand the term gaining us three new terms

$$E[(\mathbf{f} - \tilde{\mathbf{y}} + E[\tilde{\mathbf{y}}] - E[\tilde{\mathbf{y}}])^2] = E[\mathbf{f} - E[\tilde{\mathbf{y}}]] + 2E[(\mathbf{f} - E[\tilde{\mathbf{y}}])(E[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})] + E[(E[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2]. \tag{19}$$

We the continue by showing that the second term is equal to zero by expanding it,

$$2E[(\mathbf{f} - E[\tilde{\mathbf{y}}])(E[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})] = E[\mathbf{f}E[\tilde{\mathbf{y}}] - \mathbf{f}\tilde{\mathbf{y}} - E[\tilde{\mathbf{y}}]^2 + E[\tilde{\mathbf{y}}]\tilde{\mathbf{y}}]. \tag{20}$$

Using some identities for expectation values such as $E[E[A]] = E[A]$ and that $\mathbf{f}$ is a non-stochastic function and therefore $E[\mathbf{f}] = \mathbf{f}$, this gives us

$$\mathbf{f}E[\tilde{\mathbf{y}}] - \mathbf{f}E[\tilde{\mathbf{y}}] - E[\tilde{\mathbf{y}}]^2 + E[\tilde{\mathbf{y}}]^2 = 0. \tag{21}$$

We then have the remaining expression for the cost function

$$E[(\mathbf{y} - \tilde{\mathbf{y}})^2] = E[\mathbf{f} - E[\tilde{\mathbf{y}}]] + E[(E[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] + \sigma^2. \tag{22}$$

where we define the first term as the Bias and the second term as the variance, giving us the final expression

$$E[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathrm{Bias}[\tilde{\mathbf{y}}] + \mathrm{Var}[\tilde{\mathbf{y}}] + \sigma^2. \tag{23}$$

C.1 Ordinary Least Square

We have the following expression

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon} \tag{24}$$

Where we set $f(\mathbf{x})$ approximated by our function $\tilde{\mathbf{y}}$, first with Ordinary Least Square giving us $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$. We can then find the expectation value of $\mathbf{y}$ for a given element i

$$E(y_i) = E(X_{i,*}\boldsymbol{\beta} + \epsilon_i) \tag{25}$$

where $\mathbf{X}_{i,*} = \sum_j \mathbf{X}_{ij}$. Neither $\mathbf{X}$ or $\boldsymbol{\beta}$ is dependent on probability, therefore $E(\mathbf{X}\boldsymbol{\beta}) = \mathbf{X}\boldsymbol{\beta}$. While $\boldsymbol{\epsilon}$ follows a normal distribution, which means that $E(\boldsymbol{\epsilon}) = 0$ when the mean of $\boldsymbol{\epsilon}$ is equal to 0. That means we can get the expectation value

$$E(y_i) = \mathbf{X}_{i,*}\boldsymbol{\beta} \tag{26}$$

We can then follow up by finding the variance of $\mathbf{y}$ for a given element i

$$Var(y_i) = E(y_i^2) - E(y)^2 = E[(\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2\epsilon_i\mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i^2) - E(y_i)^2 \tag{27}$$

We already know $E(y_i)$ from previously and can take the expectation value of each term separately. From there it is easy to see that the first term remains unchanged, the second term is zero due to $\boldsymbol{\epsilon}$ and the third term is the expectation value of $\boldsymbol{\epsilon}$ squared which will equal the variance, $E(\boldsymbol{\epsilon}^2) = Var(\boldsymbol{\epsilon}) = \sigma^2$. Finally we can also see that the first term will cancel out the expectation value squared we already knew.

$$Var(y_i) = (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + \sigma^2 - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 = \sigma^2 \tag{28}$$

We will then show the expectation value and variance for the optimal parameters $\tilde{\boldsymbol{\beta}}$, where $\tilde{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\tilde{\boldsymbol{y}}$

$$E(\tilde{\boldsymbol{\beta}}) = E((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}) = E(\boldsymbol{\beta}) = \boldsymbol{\beta} \tag{29}$$

where we can see that the $\mathbf{X}$ matrices cancel eachother out leaving us with the result. The variance will be similar to previously

$$Var(\tilde{\boldsymbol{\beta}}) = E[((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\tilde{\boldsymbol{y}})^2] - E((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\tilde{\boldsymbol{y}})^2 \tag{30}$$

again we already know the second term, so we will take a closer look at the first term where we can pull out $(\mathbf{X}^T\mathbf{X})\mathbf{X}^T$ giving us

$$E(\mathbf{X}\boldsymbol{\beta} + 2\mathbf{X}\boldsymbol{\beta}\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^2) = (\mathbf{X}\boldsymbol{\beta})^2 + \sigma^2 \tag{31}$$

where we calculated the same as previously [27], where the first term remains unchanged, the second term is zero and the third term is the variance of $\boldsymbol{\epsilon}$. This gives us the final result

$$Var(\tilde{\boldsymbol{\beta}}) = [(\mathbf{X}^T\mathbf{X})^{-1}X^T]^2[(\mathbf{X}\boldsymbol{\beta})^2 + \sigma^2] - \boldsymbol{\beta}^2 = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2 \tag{32}$$

C.2 Ridge Regression

For the final problem we will repeat the previous problem, but this time with Ridge Regression. This means that the optimal parameters are given by $\tilde{\boldsymbol{\beta}}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})$. We repeat, calculating the expectation value and variance. We set $\mathbf{A} = \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T$ and take it outside the expectation value

$$E(\tilde{\boldsymbol{\beta}}_{\text{ridge}}) = \mathbf{A}E(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}) = \mathbf{A}\mathbf{X}\boldsymbol{\beta} \tag{33}$$

where we can see that whats left inside the expectation value is the same as we calculated earlier. We then put in for A and get

$$E(\tilde{\boldsymbol{\beta}}_{\text{ridge}}) = \mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \tag{34}$$

as our final result. As a final calculation we find the variance for $Var(\tilde{\boldsymbol{\beta}}_{\text{ridge}})$

$$Var(\tilde{\boldsymbol{\beta}}_{\text{ridge}}) = E[\mathbf{A}^2(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})^2] - E(E(\tilde{\boldsymbol{\beta}}_{\text{ridge}}))^2 \tag{35}$$

Where again the first term follows from previous

$$E[\mathbf{A}^2(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})^2] = E[\mathbf{A}^2([\mathbf{X}\boldsymbol{\beta}]^2 + \mathbf{X}\boldsymbol{\beta}\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^2)] \tag{36}$$

which again follows from previous calculations

$$E[\mathbf{A}^2(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})^2] = \mathbf{A}^2([\mathbf{X}\boldsymbol{\beta}]^2 + \sigma^2) \tag{37}$$

Which we can input into the variance and get

$$Var(\tilde{\boldsymbol{\beta}}_{\text{ridge}}) = (\mathbf{A}\mathbf{X}\boldsymbol{\beta})^2 + \mathbf{A}^2\sigma^2 - (\mathbf{A}\mathbf{X}\boldsymbol{\beta})^2 \tag{38}$$

which means we get the final result for the variance

$$Var(\tilde{\boldsymbol{\beta}}_{\text{ridge}}) = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T)^2\sigma^2 \tag{39}$$