Ordinary least squares regression of Franke's function

Anders Eriksen

Contents

Abstract

The main summary of the work

1 introduction

Machine learning methods holds promise as a powerfull tool when dealing with vast amounts of data. Regression methods are a decently intuitive subfamily of these methods. Regression methods attempt to fit input to an assumed existing true function, through coefficients applied to a polynomial of possibly varying degree.

to explore these methods, I set up code for 3 different regression methods. Ordinary least squares, Ridge and Lasso regression. The models were developed and tested on a simulated terrain generated by the franke function. After having generated the code for handling this, geographical data from the github library for "fys-stk4155" was used to fit the various methods.

The report initially goes through the justification and resaoning behind the various methods and the algorithms implemented to allow it to work, before presenting the results of the tests on the simulated geography, followed by the data on acutal terrain.

Following the discussion of the data, I attempt to make a conclusion on the data. Code and separate images can be found on the github page for the project.

2 Methods

The aim is to study regression models on data. These create a continuous function with which the input data is fitted. The first, and most basic is the *Ordinary Least Squares method* (*OLS*)

To test and validate the algorithms, a closed form function is an advantage. We wish to predict terrain. there is a 2-dimensional function called *Franke's function* which can transform a set of coordinate vectors with values between $[0, 1]$ into height values through a sum of weighted exponents.

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

To generate the data, one can create a uniformly distributed set of initial values ordered from low to high. These initial arrays are then combined into coordinate data through the numpy meshgrid function before being passed to Franke's function. These are matrices, and they are therefore flattened to 1D arrays through numpy's *ravel()* function.

Next comes setting up the model. Firstly, constructing the model design matrix. We want a design matrix for variable levels of model complexity, so we construct it from a polynomial combination of the input parameters x and y.

Next is to fit the model. The linear model, taken from Hastie et al, predicts the value Y as

$$Y\ =\ X^T\,\beta. \tag{2}$$

Algorithm 1 make design matrix X given input $\vec{x}, \vec{y}$ and dimension n

create X as a matrix of ones with dimension $length(x)$ and $(int)i(i+1)/2$
for $i = 1$ **to** $n + 1$ **do**
  $q = (int)i(i + 1)/2$
  for $k = 0$ **to** $i + 1$ **do**
    $X[:, q + k] = x^{i-k} \cdot y^k$
  end for
end for

Y is given through the inner product of the transpose of X and β. X being the design matrix mentioned above.

One method to approximate this, is with the *Residual Sum of Squares*

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2. \tag{3}$$

As the sum of squares, there is a guaranteed minimum, though not necessarily a unique one. If we write this in vector notation, differentiate w.r.t β we get the so-called *normal equations*.

$$\begin{aligned} &RSS(\beta) = (\vec{y} - \mathbf{X}\beta)^T (\vec{y} - \mathbf{X}\beta) \\ &\text{differentiating w.r.t } \beta \text{ gives:} \\ &\mathbf{X}^T(\vec{y} - \mathbf{X}\beta) = 0 \\ &\text{given non-singular } \mathbf{X}^T\mathbf{X}\text{:} \\ &\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \vec{y}. \end{aligned} \tag{4}$$

A prediction can then be made of the values Y given our β and $\mathbf{X}$ with

$$\hat{Y} = \mathbf{X}\,\hat{\beta}. \tag{5}$$

In a naive way, this prediction can be compared to the reference output values, to see how well the model predicts the data. Our wish, however, is to predict data of the same stochastic "source" as our known data. As such, we could get arbitrarily like the known data by increasing the complexity of the polynomial we use to fit the input. This will, however cost us generality. Any data outside the data we use to specifically fit the model is unlikely to fall within the model's prediction. This bias is something we need to avoid.

To this effect, we can implement a sectioning of the available data so that we can use parts of the set to "train" our model and a separate part as a

"test" or "validation". The meethod chosen is called *k-fold Cross Validation* (k-fold CV). The data set is split into k different "folds". Then, on a rotation, 1 fold is designated as the "test" fold and the remaining folds are designated "training". The model if fitted to the training set and then tested on the validation set. One can then plot the MSE for the training and test folds side by side, over different degrees of model complexity. What one should expect to see here, is at one point the errors diverge between the training and test sets due to over-fitting. As mentioned above, the model predicts closer and closer the set used to train, but sets that diverge from the training are less and less in accordnance with the model.

Algorithm 2 k fold split based on input data $\vec{x}$

```
make index array (inds) with elements from 0, to the length of x⃗
randomize inds and partition into k folds of size length(x⃗)/k,
gather into array of arrays k, length(x⃗)/k of indices, folds
for fold in; folds do
    test = input[fold]
    train = input[!fold]
    find β as in eq(4) using train
    predict output using β
    compare predict and test as well as predict and train
end for
```

Over-fitting a model essentially refers to increasing the variance of the model so much so that random noise also enters the output. conversely, an under fitted function - one with a high bias - is a model with assumptions of the model regarding the output. The latter can ignore significant relations between input and output. There is a necessary tradeoff between these two. Neither so biased, significant relations are lost, nor so high in variance that random noise affects the model output. A further examination into this, can be made through the cost function

$$\begin{aligned} C(\hat{\mathbf{X}}, \vec{\beta}) &= \frac{1}{n}\sum_i (f - \langle\tilde{y}\rangle)^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \langle\tilde{y}\rangle)^2 + \left\langle \epsilon^2 \right\rangle \\ &= Bias[\tilde{y}] + Var[\tilde{y}] - \sigma^2 \end{aligned} \tag{6}$$

Where the bias for $\tilde{y}$ gives a systematic offshoot for the model mean vs. the true function. Fitting the mean closer to the true function would reduce bias. Increasing the complexity of the fitted function, would result in far greater deviation from the average value, which means the variance grows.

finding the minimum between these 2 would help reduce the cost function, and ultimately the error of the fit.

Ordinary least squares can produce great results for a known function. Under normal circumstances the distribution function is either unknown or nonexistent. An appropriate example here could be measurements of height in terrain. There is no overarching function decinding the height of an area. As such, and especially so for large data sets, There is a need to control the outcommes of our minized betas. Not to mention our minimization assumes the matrix $\mathbf{X}^T\mathbf{X}$ is non-singular, or in other words invertible. Shrinking such "wild" coefficients would allow for a reduction in variance without much gain in bias. due to shrinking these coefficients, the following methods are also called *Shrinkage Methods*.

One such, is Ridge regression. Here, we add a parameter λ, so that

$$\hat{\beta}^{ridge} \;=\; argmin\left\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}(x_{ij}\beta_j))^2 \;+\; \lambda\sum_{j=1}^{p}\beta_j^2\right\} \tag{7}$$

$$\lambda \geq 0.$$

A large λ means that $\hat{\beta}$ experiences a greater shrinkage in order to fit. A point also, is that β_0 is not penalized like this. If we were to center the results, so that we do not fit for x_{ij}, but for $x_{ij} - \overline{x}_j$ and β_0 by $\overline{y} = \frac{1}{N}\sum_1^N y_i$. Centering the data like this allows the rewrite of (7) to matrix form with the residual sum of squares

$$RSS(\lambda) \;=\; (\vec{y} - \mathbf{X}\beta)^T(\vec{y} - \mathbf{X}\beta) + \lambda\beta^T\beta, \tag{8}$$

with a ridge regression solution of:

$$\beta^{ridge} \;=\; (\mathbf{X}^T\mathbf{X} \;+\; \lambda\mathbf{1})^{-1}\mathbf{X}^T\vec{y} \tag{9}$$

And with the added diagonal, we essentially avoid a singular $\mathbf{X}^T\mathbf{X}$.

Should the matrix $\mathbf{X}^T\mathbf{X}$ be singular there is still a way to invert it, using singular value decomposition (SVD). This method expresses a matrix as an inner product of 3 different matrices

$$\mathbf{X} \;=\; \mathbf{U}\mathbf{D}\mathbf{V}^T. \tag{10}$$

here, if $\mathbf{X}$ is a $n \times p$ matrix, then $\mathbf{U}$ is an $n \times n$ matrix, $\mathbf{D}$ is an $n \times p$ matrix, and $\mathbf{V}$ is a $p \times p$ matrix. $\mathbf{D}$ is a diagonal matrix, whose diagonal values are called the singular values of $\mathbf{X}$. If any of $d_{i,i} = 0$ then $\mathbf{X}$ is singular.

With some simplification, it can be shown,

$$\begin{aligned}\mathbf{X}\beta &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\,\vec{y} \\ &= \dots \\ &= \mathbf{U}\mathbf{U}^T\,\vec{y} \end{aligned} \tag{11}$$

Which we can send into the ridge solutions (9) so

$$\begin{aligned}\mathbf{X}\hat{\beta}^{ridge} &= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^T\vec{y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{1})^{-1}\mathbf{D}\mathbf{U}^T\vec{y} \\ &= \sum_{j=1}^{p}\vec{u}_j\frac{d_j^2}{d_j^2+\lambda}\vec{u}_j^T\vec{y}. \end{aligned} \tag{12}$$

Here, $\vec{u}_j$ are the collumn vectors of $\mathbf{U}$, and d_j are the singular values of $\mathbf{X}$, the diagonal elements of $\mathbf{D}$. As the parameter $\lambda \geq 0$, the fraction $\frac{d_j^2}{d_j^2+\lambda} \leq 1$, and the β's will be shrunk in accordance with the size of the singular values. The collumns of $\mathbf{V}$ from (10) form the so-called *principal component directions* of $\mathbf{X}$. The first of these, has the property where $z_1 = \mathbf{X}v_1$ has the larges sample variance. amongst all normalized collumns of $\mathbf{x}$

$$Var(\vec{z}_1) = Var(\mathbf{X}\vec{v}_1) = \frac{d_1^2}{N}, \tag{13}$$

and further, $\vec{z}_1 = \vec{u}_1\,d_1$. $\vec{z}_1$ is the first principal component of $\mathbf{X}$, making $\vec{u}_1$ the *normalized* first principal component. subsequent principal components have similar variance $\vec{z}_j = \frac{d_j^2}{N}$ and are orthogonal to the previous. The last principal component sports the lowest variance.

$\mathbf{X}$ projects the output Y with greater variance along the prior principal component directions rather than the latter. A surface plot along these projections will then have less variance in the gradient along the axes of greater projected variance. Ridge regression shrinks the lower variance and screens the output from the gradient variance.

The assumption behind this is that the response should vary most along the direction of greatest variance of the input. This is not in general true, but inputs are often chosen to study because the output vary along with them.

Another way to shrink the coefficients in *beta* is with minimizing with a restriction

$$\hat{\beta}^{lasso} = argmin\left\{\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p}x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta|\right\}. \tag{14}$$

Where the main difference here is the sum of $|\beta|$ rather than β^2 as in ridge (7). The main difference here, is that there is now a non-linear solution with y_i. This also means that there is no closed-form solution, and the computation is therefore far more taxing. There are libraries which solve this at roughly the same computational cost as ridge, however. In this example, the library used is python's scikitlearn.

The implementation of both ridge and lasso amounts to little change in comparrison to the standard OLS with f-fold CV. Mainly the calculation of β as well as a rewrite of the inversion method into SVD to be more accurate, though at some cost to the calculation time.

These three methods can then be tested and controlled on the generated set of data, fitted with the Franke function(1). Once the results have been verified to some degree, a set of proper measured data can be used to train the models. These data are gathered from the webpage earthexplorer.usgs.gov , though the data used here are taken from the machine learning github page.

3 Results

First order of business was to generate input variables and generate the simulated output of the franke function before adding noise. Following this, the initial OLS fit. One way to visualize the franke output is through a surface plot using matplotlib's 3d axis functionality.

Figure 1 shows the surface plot of the franke function and the OLS fit. Before the fit, a stochastic noise is added to the franke function and the model then tries to find the function underneath the noise. As the image shows, there is a good overlap of the franke function and the fit. This is an expected result given the existance of a continuous function behind the surfacem which is what regression methods try to find.

There are still some discrepancies between the franke function and the OLS fit. In addition, the set we plot here is the same one we use to train the model and test it's validity. There is also no variation in the polynomial we try to fit the function with. It might be in a sweet spot that emulates the franke function, or we might have simply overfitted the model to the given data, and if we were to provide a set which was not within the training data, the errors would grow markedly. It is also possible that we could enhance the fit further if we were to change the complexity of the model further.

The confidence interval of β could provide some insight into the variance in the model. The confidence interval gives a measurement of how far out from the chosen value you would have to vary to be certain that a given percentage of the data was accounted for.

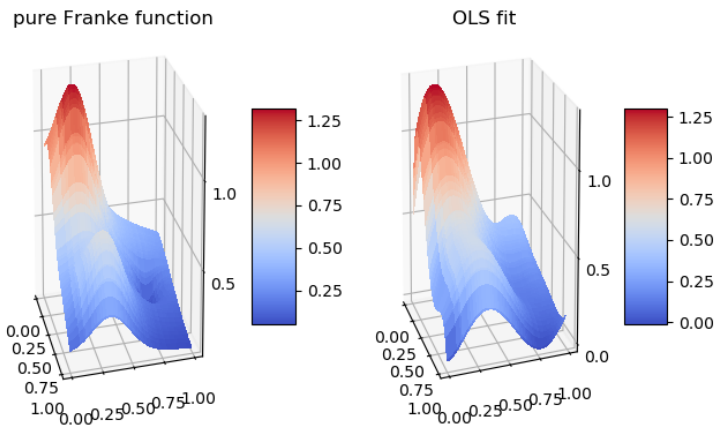

Figure 1: surface plot of the franke function(left) and the ordinary least squares fit(right), without resampling. The model is trained on the entire set with the franke function added stochastic noise. The overall structure of the surface is maintained, as well as the heights. There are some sharper peaks in the Franke function surface compared to the fit, but the fit is faithful.

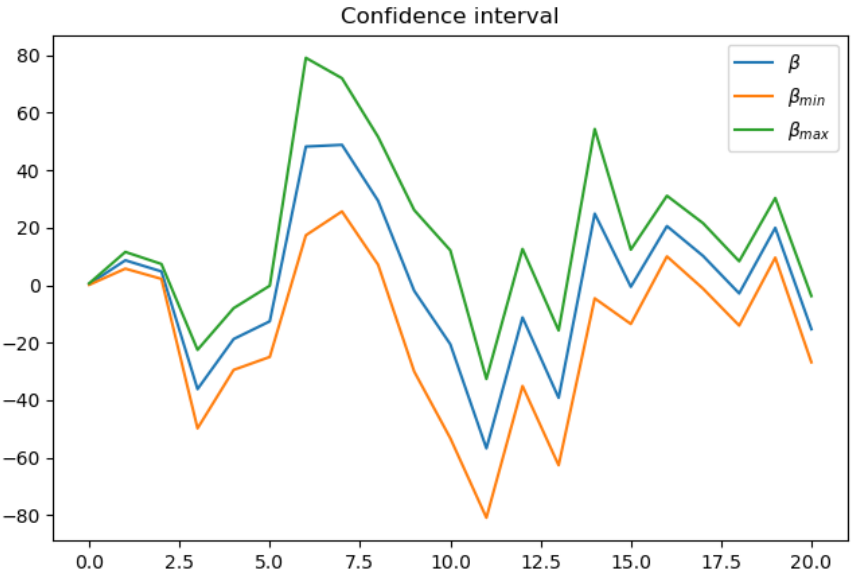

Figure 2: Confidence interval of the β values for the OLS fit sans any resampling. The interval shows the range in order to include 96% of the target values. There are areas with a wider band, and the difference seems to be up to nearly 30% of the given β value. The axis dimensions here are generally omited, as the 1. axis follows the arrays without more than the index, and the 2. axis represents the coefficient values for the fit. These are dimensionless.

The interval is of a breadth that is not insignificant given the values of β, varying at times up to about 30% of the actual beta value.

MSE	R^2
0.0118	0.8865

Table 1: Mean squared error and R^2 function for the OLS fit of the franke function without resampling. The mean squared error is fairly small, but this is not strange considering the fact that there is indeed a function creating the data.

Calculating the mean squared error, or residual sum of squares, as seen in table 1 reveals that the relative error of the prediction lies at about 1%, and with an R^2 of nearly 90% tells us that the variance seen is mostly predicted by the model. Though this does not rule out overfitting since the calculations have been performed on the same set we trained the model on.

With the implementation of f-fold cross validation, the degree of overfitting vs. prediction can be examined further.

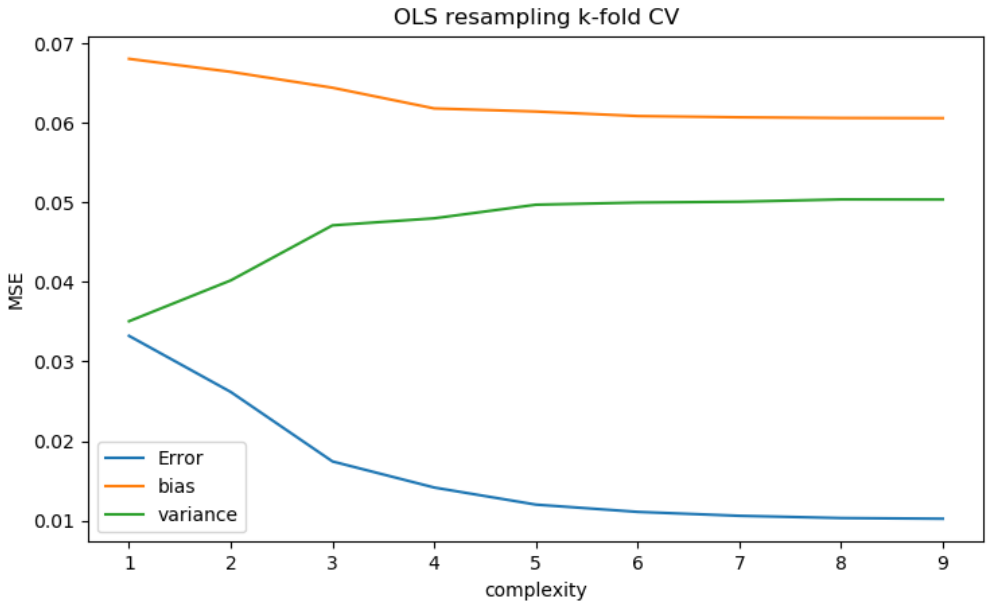

Figure 3: OLS fit of franke function with $k = 5$ folds. the complexity along the 1. axis refers to the polynomial degree of the fit, while the poorly named "MSE" axis depicts the numerical values of bias, variance and error. These measurements are made on varying test sets, and as such is not a great representation of the model, but there is a definitive trend for bias to fall, while variance increases with model complexity.

As discussed above, we can see in figure 3 that the model's bias decreases with increasing model complexity, while the variance grows. Though this is hardly an ideal image, and the results being compared aren't based on a common set, but rather the test set at each round in the k-fold rotation. This makes the values shown, such as the error highly suspect. the plot does showcase the expected trends in bias and variance, however. And this was the main point in this stage.

In order to better fit a model for which there is no known function governing the outputs, the shrinkage methods discussed above are better suited. And the added bonus of shrinking the output along axes not imediately along the prime axes of $\mathbf{X}$ does produce far neater gradient along a surface projected by the model. Implementing the Ridge regression method for the values and thus adding the super parameter λ to the mix, allows us to create a heatmap of the error for the various λ's and degrees of complexity.

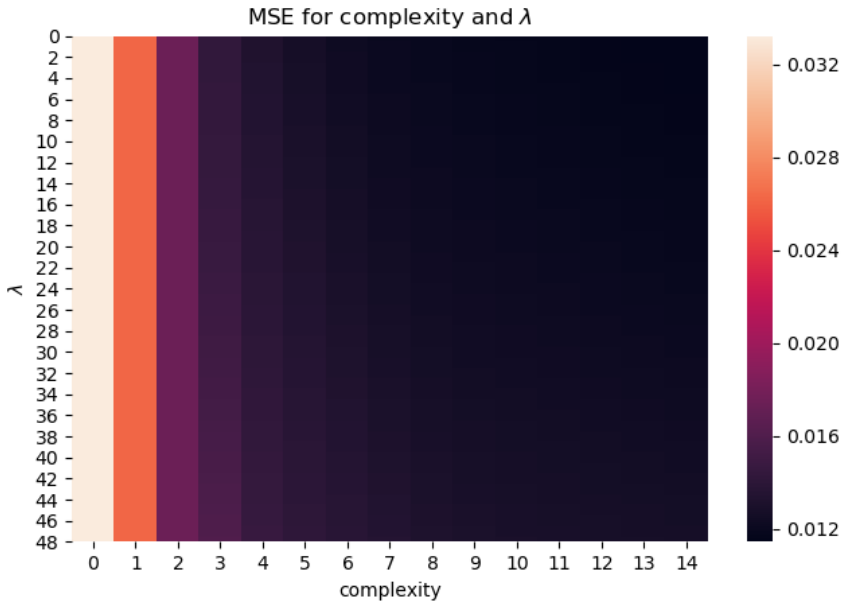

Figure 4: Heatmap of the MSE for Ridge Regression of the franke function for various λ and degrees of complexity in the fit. The error here seems to range from 1% to roughly 3%, with the greatest errors towards the simplest it and the lowest hyper parameter. This is in accordnance with the expectation for prediction error, as the error plotted is essentially the training error. As we approach greater complexity, the variance also increases, as seen in figure 3, the variance quickly explodes.

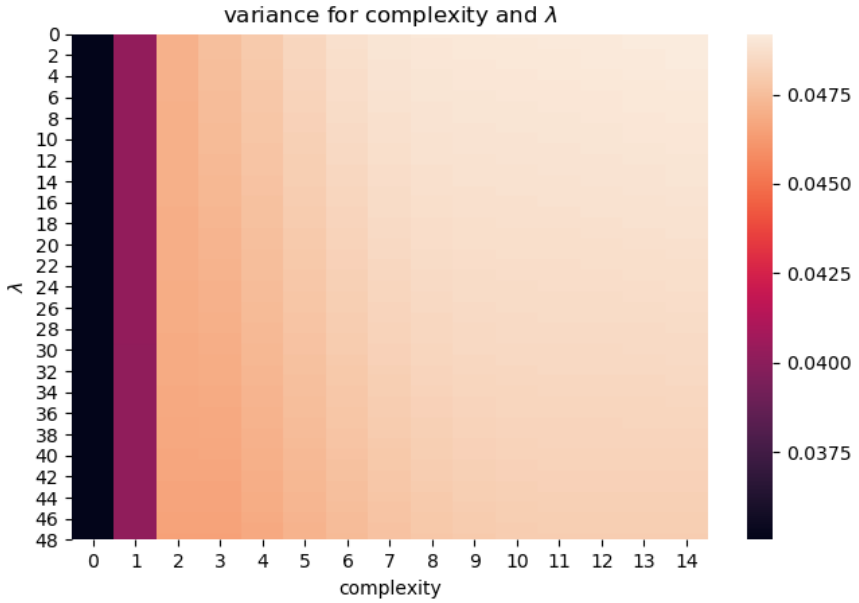

Figure 5: Heatmap of the variance for Ridge regression of the franke function over various λ and degrees of complexity. As one would expect, the variance is least when the polynomial degree of the model is low, and for the greatest shrinkage of λ, though the latter has a weaker effect, as it per definition shrinks the lower variance elements.

As we can se in figure 3, the error falls of greatly with increasing polynomial degree for the fit, and for lower shrinkage λ. This is not not unexpected, given the franke function is an actual continuous function. Therefore, the closest fit is without shrinkage. The continuous lowering of the error as we increase in complexity speaks to the fact that the heatmap is made from the training data, and as such we see the downward trend of fit for the training set, and not the actual predictions of the functions.

Implementing the Lasso method for the franke function, the lack of a closed form solution makes the lasso implementation a lot more challenging than with the linear case. As such, python scikitlearn library was utilized for the fit and prediction of the model here.

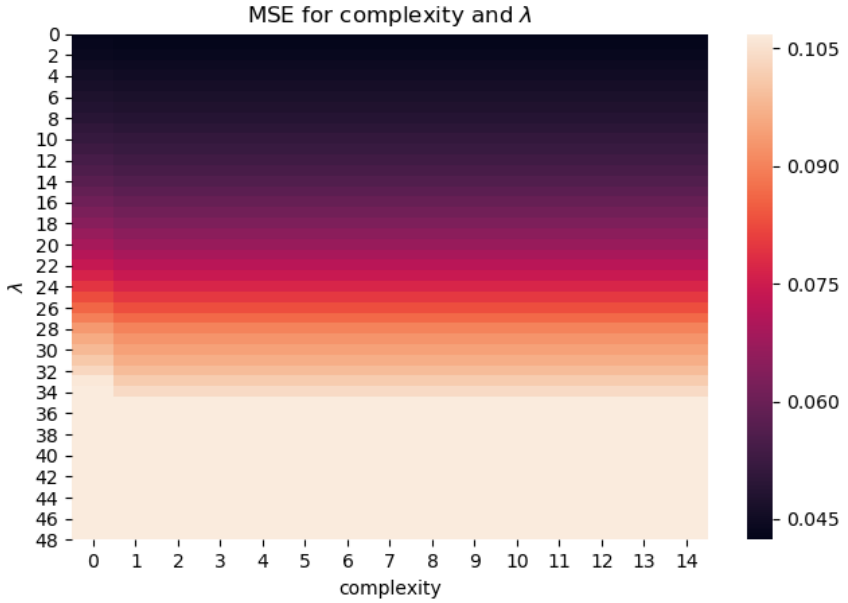

Figure 6: Heatmap from the Lasso method on a fit for the franke function. The 1. axis is the model complexity, while the 2. axis follows the hyperparameter λ. There is a tendency towards the minimization of the error with lower lambdas and greater complexity, as seen with the other methods. The nonlinearity of the solution shows in the large error seen with lambda greater than 30. here, the error is uniformly 10% and up. There is a more gradual improvement as the lambda variables near 0, wih a continuing trend along the increase in complexity for the model. we see the same inversion between the error and variance between the error heatmap and the variance in figure 3 as we see between the mse and vairance heatmaps of the ridge case.

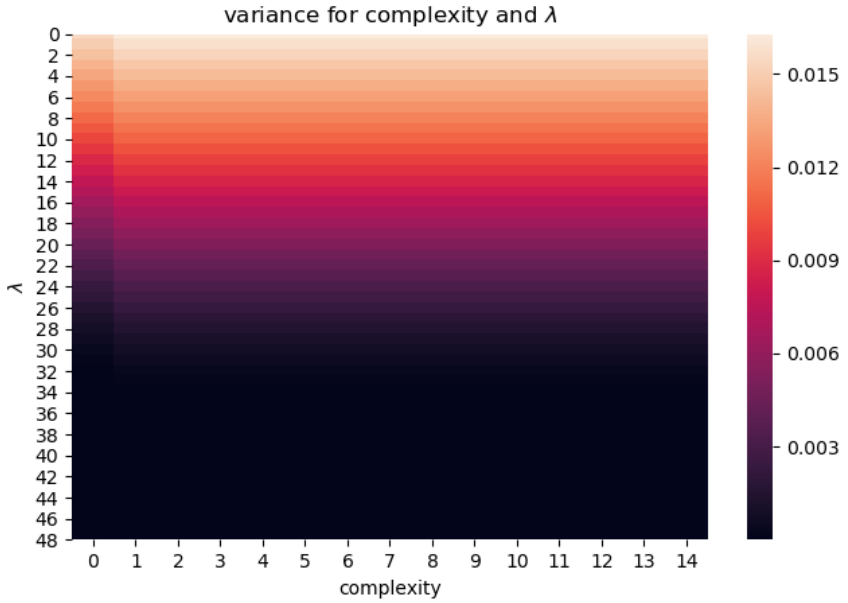

Figure 7: heatmap of the variance of the fit of Lasso to the franke function data set. There is a definitive increase in the variance as λ approaches 0.

There is then a clear assosiation along the trends one would expect when fitting an existing function using regression methods. There are discrepancies, but they are likely from mismatch between the various data sets used to construct the averages displayed.

One thing that needs to be mentioned, is the 2. axis for the heatmaps. I don't know why, but they appear to be wrong, considering $\lambda \in [0.01, 0.1]$. What the numbers along the 2. axis mean I do not know. The complexity seems to work accordingly, if slightly off when starting at 0.

With the snafu of axes mentioned, it is time to move onto the proper geographical data. There was an issue regarding my system's native browser and the verification used by the website for registration Therefore, I used the files found in the course github repository instead. Furthermore, in order to cut down on computation time for the lacking processing power of the current computer, only a slice of about 100 by 100 was used from the data.

The OLS fit of the data went smoothly enough, producing a fit to the provided dat, which were less accurate than the tests with the franke function.

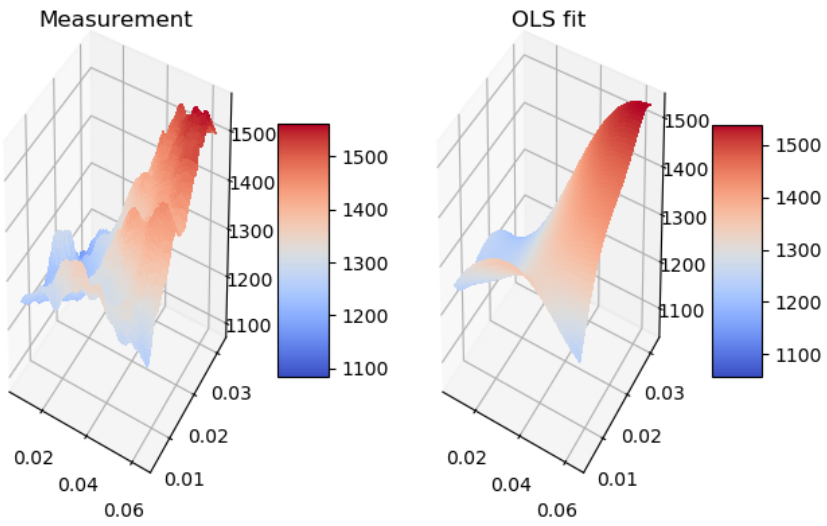

Figure 8: We see that the major upwards trends for the fit align decently with the provided sample, though we see a distinct lack in the finer details of the landscape. The model was made using a complexity 5th order polynomial.

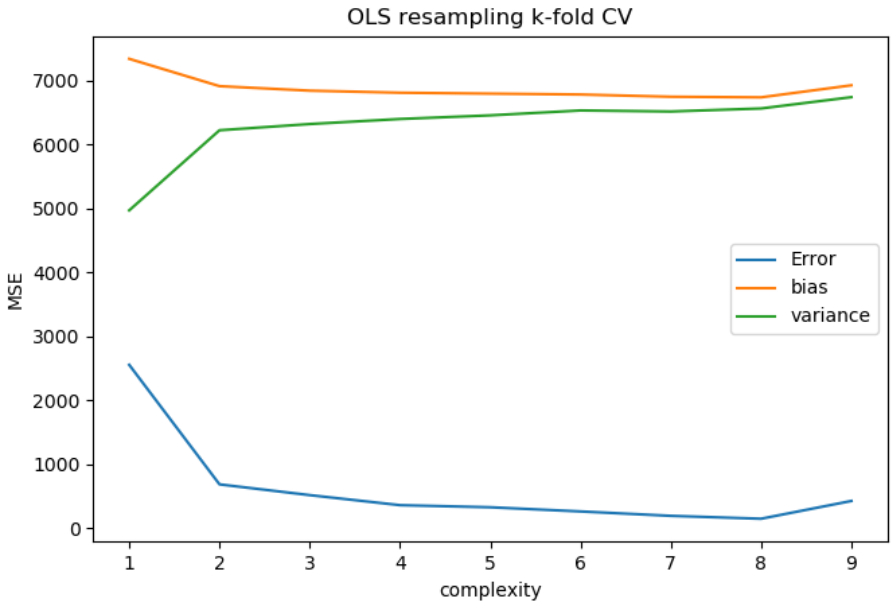

Figure 9: A plot of the errors of an OLS fit to terrain data. As with previous figures, this one is made with points found within different sets. As such, the values and shape of the total arrays, are less trustworthy. There is a definitive trend towards bias lowering, and variance increasing with the increase in complexity.

As we can see on the surface plots of the geographical data and the OLS fit, the major features of the landscape are recreated with the model, although the issue of overfitting persists to some degre, given the training data set is also the one used to compare. The model does indeed reproduce the overall surface.

Next comes implementing the Ridge regression with the data. The method is much the same as in the previous case, only now the data is trained on proper data, rather than a simulation.

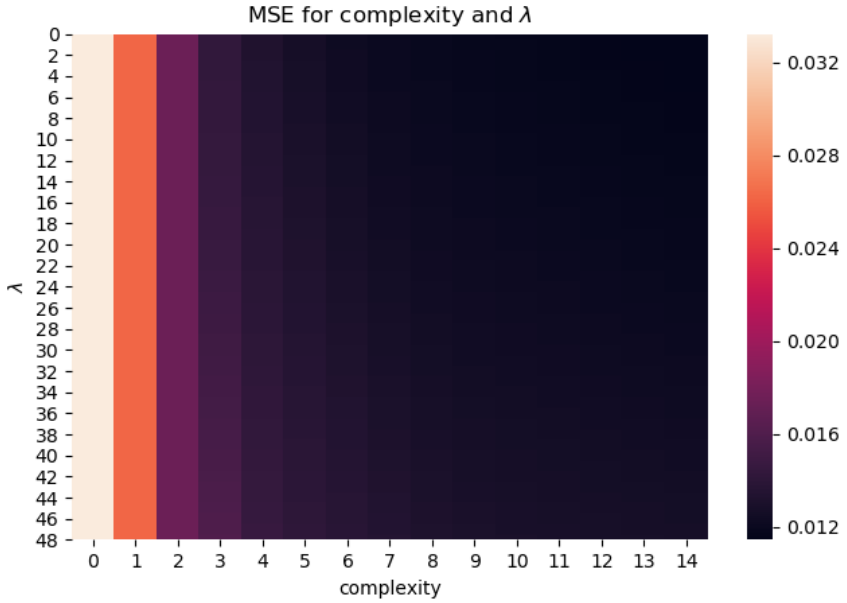

Figure 10: As with the previous heatmaps, the error of the ridge method Falls with growing complexity, and falls with less shrinkage. The error has a marked jump at the 1st level of complexity, but this could be from mishandling the intercept, and failing to senter the data.

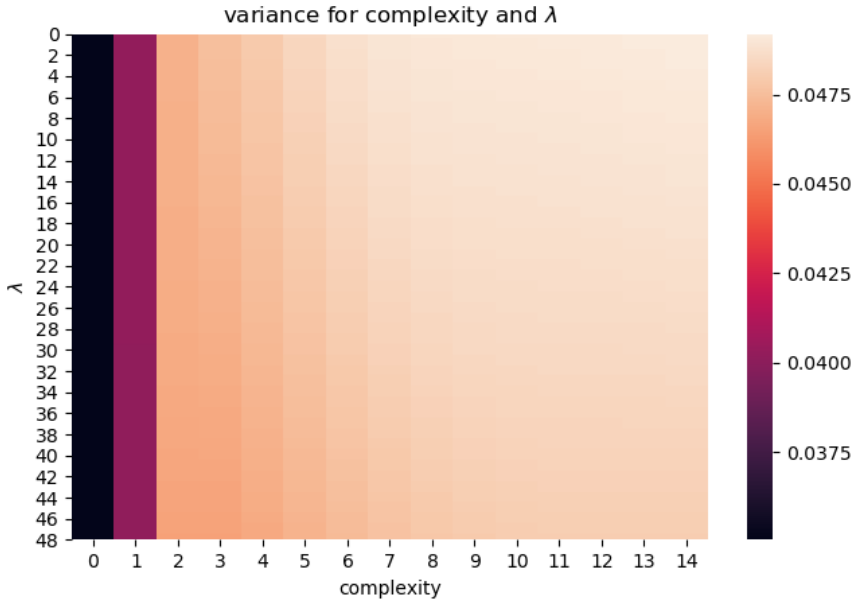

Figure 11: Variance of the ridge fit of the geographical data. The behaviour is an inverse of the error, which is not strange considering the increase of the polynomial degree used to fit the model. There is a marked difference in the first degree of complexity, likely from the simplicity of the model. Hard to vary much from the mean along a straight line, if the plot is to be believed.

Adding the hyper parameter allows for for the shrinkage of variance along the non-principal directions of the input, as seen in figure 3, though the trend for this is still towards the lower values of lambda. This should allow for less variance on the surface gradient on the fited set. The attempt at a validation set to compare as well as a surface plot of the fit alongside the validation data fell through, as the dimensions didn't quite add up during the comparison and the plot. At this point, time was running short, and the attempt was aborted.

Following this, was the lasso fit. as figure 3 shows, the Lasso fit did not perform at all. Compared even to the fit of the lasso for the franke function, the error here is far out of bounds. why this discrepancy, I do not know.

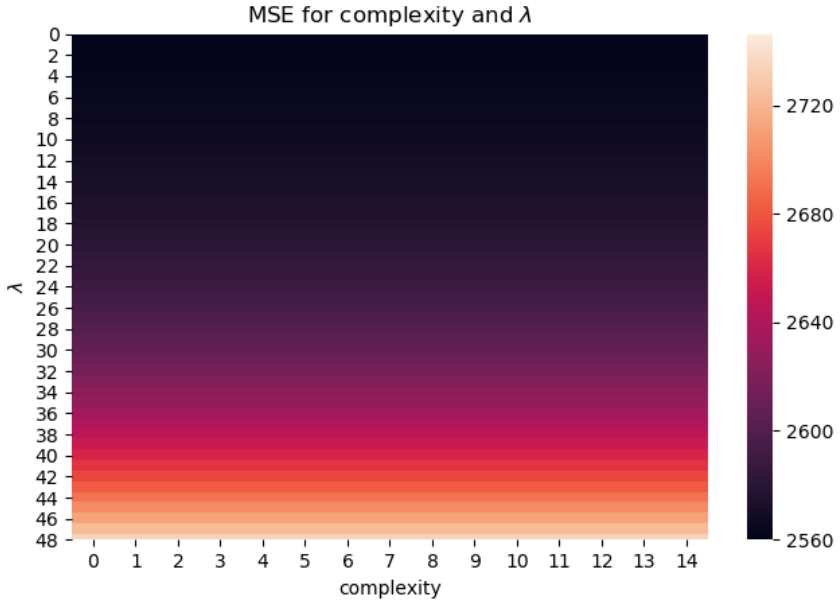

Figure 12: heatmap of mean squared error for the lasso fit of the terrain data. These numbers far outscale any other result throughout the process, but the reason is currently unknown. Time limitations do not allow for correction of these.

4 Conclusion

The regression methods attempt to fit a continuous function to control output given certain input. The surface plot of figure 1 shows that given a simulated surface based on the franke function, the OLS can reasonably well fit the input to the output. Given the seeming decrease in error for the Ridge and Lasso case, this seems to be compounded, though failing to plot surfaces of the further fits for these methods make direct comparisons more difficult. In addition, given the lack of a validation set, a comparison between the methods is difficult.

For the actual geographical data, Ridge regression as the most effective, though seemingly not by too much. The only surface that was plotted was for the OLS, which seemed to follow the main features of the terrain, but little more. The methods used here are likely poorly represented, through various errors in the setup of the code, first of which is likely that there was no measurement against a validation set. There is no real conclusion to be drawn as to the method's capacity to model the data. Though intuitively,

terrain is not governed by an underlying function, so any fit relying on this would likely struggle beyond some granularity.

Ideally the data should likely have been split into a train and test set, with a further k-fold validation of the training set into train and validation sets. Then the control test set could form a proper basis for comparison. Greater use of surface plots to actually see the results of the fit for various degrees and λ's, with plots of the train and test error for various λ to illustrate the relationship between bias and variance with the complexity of the fit.

In addition, the code should have been a modular set for the given data, with the changes based on the call there and then. The franke function should then have been implemented as a benchmark unit test towards a known function. Furthermore a test set of data partitioned for each test would as mentioned have made direct comparisons more accurate.

Implementing these changes would be obvious steps ahead. Another improvement would be to view a larger set of data, instead of the smal slice of this project. Including more data would also remove more of any hint of a true function for the fit, which should lead to poorer fits, but would also be more usefull in assessing the capability of the model to predict the given terrain. Perhaps even further would be to vary the terrain data over various areas. Are some parts of the world better suited to fitting than others? Or perhaps there are similarities in the overall structures which can be found through losing some of the granularity of the measurement.

Sadly there were issues when writing the report regarding bibliography. The data sets and franke function are all taken from Fys-Stk4155 course github page at `https://github.com/CompPhysics/MachineLearning.git`

5 Appendix

5.1 Process for showing bias-variance from cost function

$$C(\hat{\mathbf{X}}, \vec{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \; = \; \left\langle (y - \tilde{y})^2 \right\rangle \tag{15}$$

$$y = f(x) + \epsilon \;\text{ and }\; f(x) \equiv f \text{ and } \langle \epsilon \rangle \equiv 0, var(\epsilon) \equiv \sigma^2$$

$$\begin{aligned}
&\left\langle (y - \tilde{y})^2 \right\rangle = \left\langle (y - \tilde{y} + \langle \tilde{y} \rangle - \langle \tilde{y} \rangle)^2 \right\rangle \\
&= \left\langle \big((f - \langle \tilde{y} \rangle) - (\tilde{y} - \langle \tilde{y} \rangle) + \epsilon^2\big) \right\rangle \\
&= \left\langle (f - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle (\tilde{y} - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle \epsilon^2 \right\rangle \\
&\quad + 2\left\langle \epsilon (f - \langle \tilde{y} \rangle) \right\rangle - 2\left\langle \epsilon (\tilde{y} - \langle \tilde{y} \rangle) \right\rangle \\
&\quad - 2\left\langle (f - \langle \tilde{y} \rangle)(\tilde{y} - \langle \tilde{y} \rangle) \right\rangle \\
&= \left\langle (f - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle (\tilde{y} - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle \epsilon^2 \right\rangle \\
&\quad - 2\left\langle (f - \langle \tilde{y} \rangle) \right\rangle \left\langle (\tilde{y} - \langle \tilde{y} \rangle) \right\rangle \\
&= \left\langle (f - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle (\tilde{y} - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle \epsilon^2 \right\rangle \\
&\quad - 2(f - \langle \tilde{y} \rangle) \left\langle (\tilde{y} - \langle \tilde{y} \rangle) \right\rangle \\
&= \left\langle (f - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle (\tilde{y} - \langle \tilde{y} \rangle)^2 \right\rangle + \left\langle \epsilon^2 \right\rangle \\
&= Bias[\tilde{y}] + Var[\tilde{y}] - \sigma^2
\end{aligned} \tag{16}$$

5.2 Heatmaps from Ridge

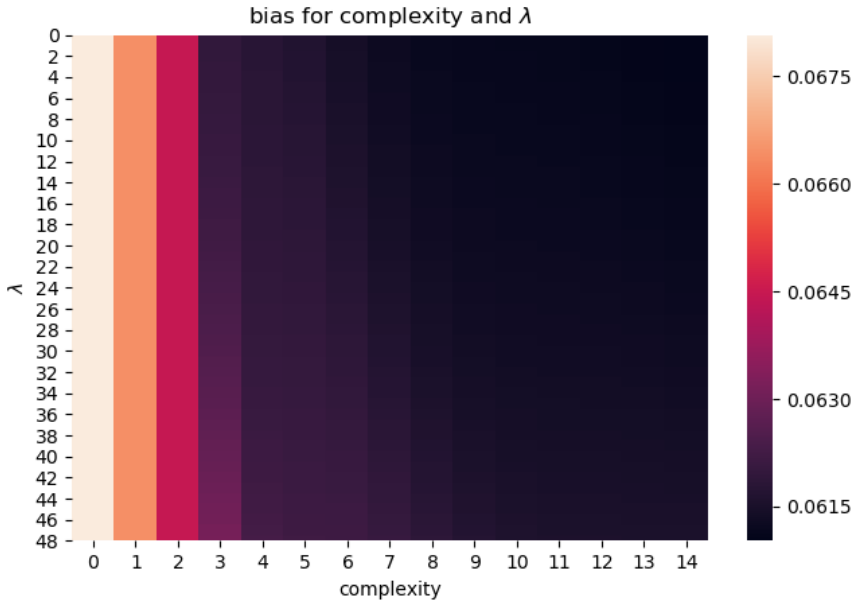

Figure 13: Heatmap of the bias in ridge regression fit of Franke's function with $\lambda \in [1e^{-1.2}, 1e^{-1}]$ and model complexity between 1 and 16 degrees.

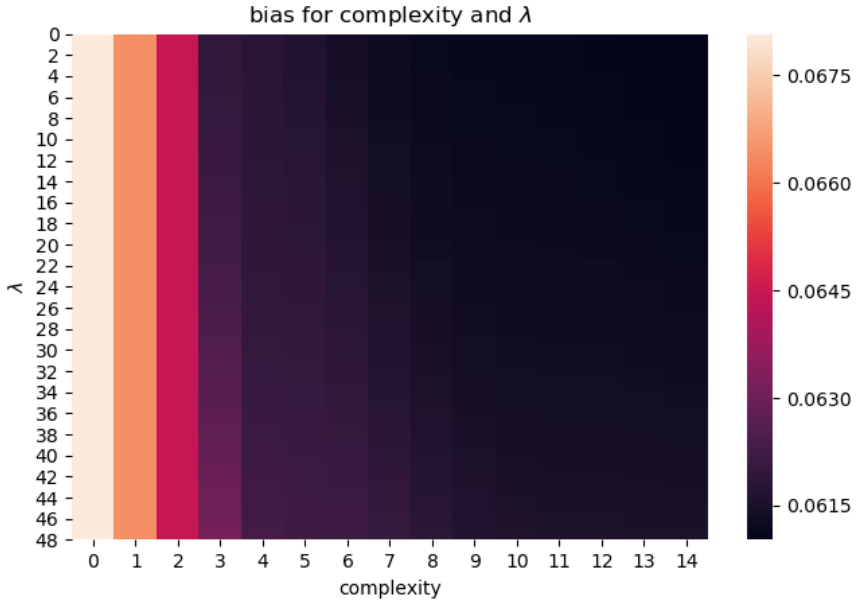

Figure 14: Heatmap of the bias in ridgeregression fit of data from norway, $\lambda \in [1e^{-1.2}, 1e^{-1}]$ and model complexity between 1 and 16 degrees.

5.3 Heatmaps from Lasso

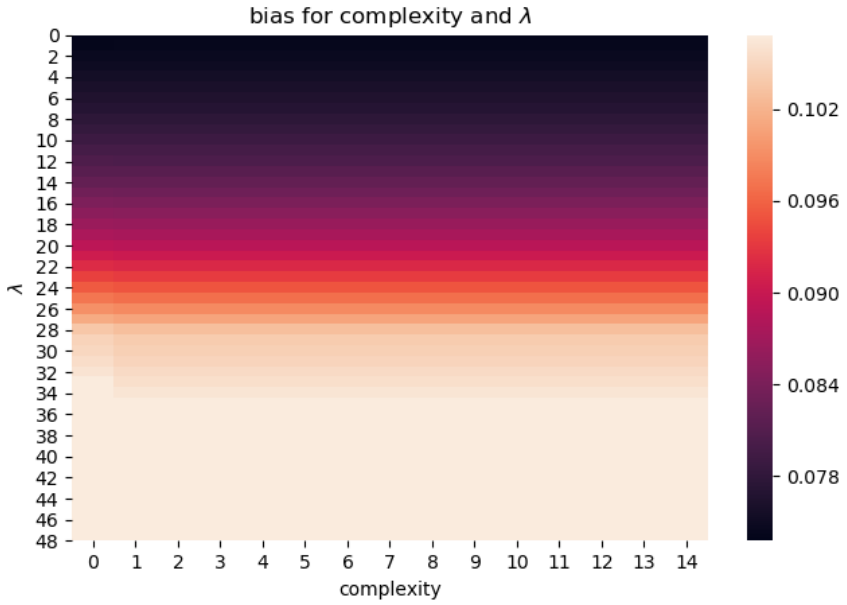

Figure 15: Heatmap of the bias in Lasso regression fit of Franke's function with $\lambda \in [1e^{-1.2}, 1e^{-1}]$ and model complexity between 1 and 16 degrees.

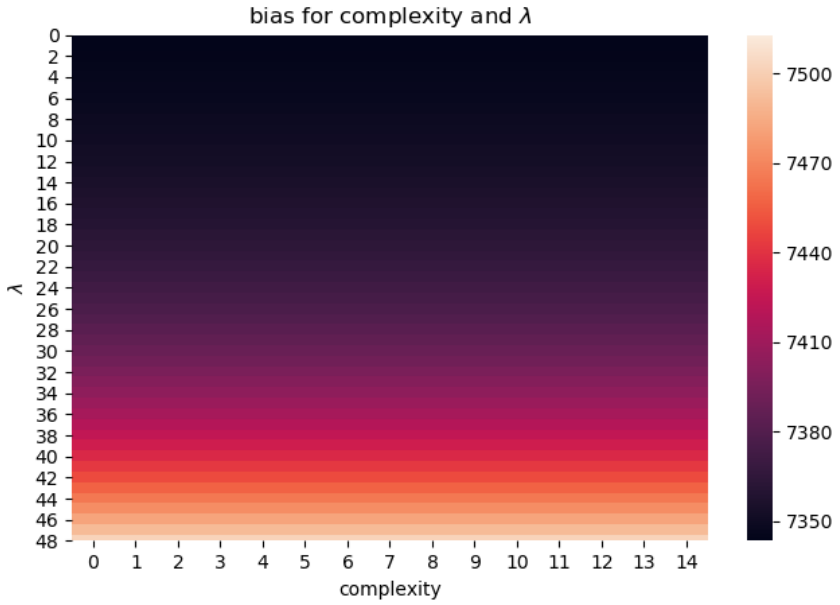

Figure 16: Heatmap of the bias in Lasso regression fit of data from norway, $\lambda \in [1e^{-1.2}, 1e^{-1}]$ and model complexity between 1 and 16 degrees.