UiO : Department of Physics
University of Oslo

Regression analysis and resampling methods

Erik Skaar
Sondre Torp
Mikael Kiste

Contents

1 Introduction **2**

2 Theory **3**
2.1 Ordinary least squares . . . 3
2.2 Evaluations of our models . . . 4
2.2.1 Mean square error . . . 4
2.2.2 R^2 score - The Coefficient of Variation . . . 4
2.3 Ridge . . . 5
2.4 Lasso . . . 6
2.5 K-fold . . . 6

3 Method **7**
3.1 Implementation of OLS, Ridge and Lasso . . . 7
3.2 Implementation of MSE, R2 and β variance . . . 7
3.3 Normalizing EVERYTHING! . . . 7

4 Implementation **8**
4.1 Scikit vs. manually implementation . . . 8
4.2 Time evolution . . . 9
4.3 Noise - MSE & R2 evolution . . . 9

5 Result & Discussion **10**
5.1 Ordinary least square, Ridge, and Lasso regression with resampling on the Franke function 10
5.1.1 Ordinary least square . . . 10
5.1.2 Ridge regression . . . 12
5.1.3 Lasso regression . . . 13
5.2 Ordinary least square, Ridge, and Lasso regression with resampling, now on real data . 15

6 Conclusion **19**

Abstract

This report is based on a project assignment in the subject FYS-STK4155 at UiO.[**project1**] In this report the following regression methods were implemented; OLS, ridge and Lasso. To understand how well our methods worked, we tested it against Scikit's solutions, checked time dependance for different order of fitting and checked how noise affected the resulting polynomial. Then our implementation, OLS, Ridge and Lasso, were used on the Franke function. MSE,R2 score and VAR was calculated for all of the methods and how k-fold cross validation affected to resulting polynomial is shown. After this had been done on the Franke function, we repeated the proccedure for terrain data in Norway. The resulting polynomial were a good approximation for general features for the data, but fell short to describe the details.

1 Introduction

A typical problem within the natural sciences is how to interpret the trends and behavior of the results and data from an experiment. As a first approximation this will often be done qualitatively, e.g. "These values appear to increase linearly with time", but a more rigorous approach through regression analysis and resampling methods is eventually more preferable. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Theory

2.1 Ordinary least squares

We want to get a specific solution to the equation

$$\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon} \tag{1}$$

Where $\hat{y}$ is a vector of our measured values, $\hat{X}$ is a matrix containing variables and determines how we want to fit our data, $\hat{\beta}$ is a vector of the parameters for our fit and $\hat{\epsilon}$ is a vector representing the error in our datapoints (often termed the residuals, representing how far off our prediction is from the measurements). The variables $\hat{y}$ and $\hat{X}$ are fixed and we want to choose parameters $\hat{\beta}$ in such a way that the errors $\hat{\epsilon}$ are minimized. An example might help clarify the situation

Lets say we have conducted an experiment where we have measured the position of a ball launched straight up into the air from a cannon. Neglecting air resistance we know that the analytical solution is on the form of a second order polynomial $x(t) = x_0 + v_0 t + at^2$, where x_0 and v_0 are the initial conditions for the position and velocity respectively. This analytical solution is our model, but the actual measured values could (and indeed probably would) differ from this, simply due to errors in the measurement or other effects coming into play that the model has not accounted for (like air resistance). In any case, if we measured the position n times our linear algebra problem could be stated like this:

$$\begin{bmatrix} x(t_0) \\ x(t_1) \\ \vdots \\ x(t_{n-1}) \\ x(t_n) \end{bmatrix} = \begin{bmatrix} (t_0)^0 & (t_0)^1 & (t_0)^2 \\ (t_1)^0 & (t_1)^1 & (t_1)^2 \\ \vdots & \vdots & \vdots \\ (t_{n-1})^0 & (t_{n-1})^1 & (t_{n-1})^2 \\ (t_n)^0 & (t_n)^1 & (t_n)^2 \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \\ a \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_{n-1} \\ \epsilon_n \end{bmatrix}$$

Or, again, stated through vectors and matrix notation

$$\hat{x} = \hat{T}\hat{\beta} + \hat{\epsilon}$$

With some of the variable names adjusted simply to better indicate the represented values we have in this problem. Essentially we want to determine the variables x_0, v_0 and a so that the error terms are minimized (the equation has solutions for all $\hat{\beta}$s, but most of those are horrible fits with huge error terms). Note that there are two dimensionalities coming into play here. n is the number of measurements and determines the length of the column vectors $\hat{x}$ and $\hat{\epsilon}$. In addition there is a second dimension that determines the number of columns in the matrix $\hat{T}$ and the length of the vector $\hat{\beta}$. This number, say m, indicates the complexity of our model. When doing a polynomial fit, $m-1$ is the order of the polynomial we want to fit our data to. So in this case, where we want to fit a second degree polynomial, we have $m = 3$.

The example above gives an impression of what the variables in the linear algebra equation represents but is quite specific and we can generalize a bit. For instance, it is not necessary to fit a polynomial at all. By changing our $\hat{X}$ matrix we could fit to any orthogonal function that we would like. The number of datapoints can be anything we want as log as $m \leq n$.

A key part of the fitting is how to minimize the so called "cost-function". In our case we want to minimize the ϵ's. There are different ways of doing this, but perhaps the simplest one is to do an **Ordinary Least Squares** (OLS) fit. That is to say when taking the difference between our predicted values and the measured values (essentially being the residuals) we want the squared sum of the difference for each datapoint to be as low as possible. That is to say that we want to minimize the function

$$Q = \sum_{i=0}^{n-1} \epsilon_i^2 = \hat{\epsilon}^T \hat{\varepsilon} = \left(\hat{y} - \hat{X}\hat{\beta}\right)^T \left(\hat{y} - \hat{X}\hat{\beta}\right)$$

We can see that taking the squared sum of all the elements in $\hat{\epsilon}$ is the same as taking the inner product of the vector with itself, and that we can use equation 1 to develop the expression further. We are interested in finding the parameters $\hat{\beta}$ that leads to the minimization of the squared sum of the residuals. So now that we have a function of the squared sum of the residuals with $\hat{\beta}$ as a variable it is a simple matter of finding when the derivative of this function with respect to $\hat{\beta}$ is zero; as this will give us the minimum (it must, of course, be a minimum as only a quite specific $\hat{\beta}$ will give a good fit and other values, deviating from this, would only increase the residuals squared sum. The possibility of more than one minimum is not something we need to worry about at this point). We will here state the derivative without further explanation, suffice it to say that looking at the expanded indexed expression (i.e. not on vector form) one quickly comes to the conclusion that this must indeed be the correct derivative.

$$\begin{aligned} \frac{\partial Q(\hat{\beta}}{\partial \hat{\beta}} &= 0 = \hat{X}^T \left(\hat{y} - \hat{X}\hat{\beta}\right) \\ \hat{X}^T \hat{y} &= \hat{X}^T \hat{X} \beta \end{aligned}$$

$$\hat{\beta} \quad = \left(\hat{X}^T \hat{X}\right)^{-1} \hat{X}^T \hat{y} \tag{2}$$

And by the magic of linear algebra we have arrived at an analytical solution for the parameters $\hat{\beta}$ we need for an OLS regression. It is, however, important to note that we assume that the matrix $\hat{X}$ here is invertible.

2.2 Evaluations of our models

2.2.1 Mean square error

The Mean Square Error (**MSE**) can give a measure of the quality of our estimator. It is defined as

$$MSE(\epsilon) = \frac{1}{n} \sum_{n}^{n-1} \epsilon^2 \tag{3}$$

As such it can be thought of as the average of the square of our residuals. We see that our OLS method minimizes the MSE of our predictor variables. Of course, as easily seen from the definition, the MSE can never be negative and lower values means that we have a better prediction (at zero there is a perfect fit).

2.2.2 R^2 score - The Coefficient of Variation

In regression validation the R^2 is the gold standard when it comes to measuring goodness of fit. In straight terms it is the proportion of the variance in the dependent variable that is predictable from

the independent variable(S).[**coef**]

$$R^2 = 1 - \frac{\sum(y_i - \tilde{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \tag{4}$$

Where y_i are the indexed response variables (data we want to fit) and $\tilde{y}_i$ is the predictor variables from our model (so $\epsilon_i = y_i - \tilde{y}_i$). The average of the response variables is denoted $\bar{y}_i$. In our case it the second term can also be considered as the ratio of **MSE** to the variance (the $1/n$ factors kill each other in a fraction). Let's interpret the formula step by step to get an impression of how it differentiates a poor from a good fit. If the residual sum of squares (SS_{res}) is low we have a good fit. However, we should compare this to the spread of our response variables. After all, if the response variables are all nicely distributed close to the mean then getting a good SS_{res} is not that impressive. We therefore do a sort of normalization in the fraction, taking the scale of our data into consideration. In the simplest polynomial fit, using a zeroth order polynomial (just a constant), we see that our model would just be a constant function of the mean. The sums would be equal, returning unity on the fraction and the total R^2 score would be zero. In the other extreme, if our model fits perfectly, then SS_{res} would be zero and the R^2 score would be one. In this sense we have a span of possible R^2 scores between zero and one, from the baseline of the simplest model at zero and a perfect fit at one. The R^2 score is useful as a measure of how good our model is at predicting future samples.

2.3 Ridge

A problem with the **OLS** method through linear regression is that the matrix is not necessarily invertible. In this case it is basically impossible to model the data using linear regression [**morten-reg**]. When there are many columns in the $\hat{X}$ matrix it is less likely that the columns are all linearly independent, which is a requirement to get the inverse $(\hat{X}^T\hat{X})^{-1}$. Since the number of columns increase when the complexity of our fit increases (as mentioned earlier, if the polynomial we want to fit is of order n then the number of columns is $m = n + 1$) we understand that a more complex model decreases the likelihood of invertibility. Since there is no longer a unique solution to our problem, it is an example of an ill-posed problem that is either overdetermined with more equations than unknowns (oftentimes leading to no solution) or underdetermined with more unknowns than equations (resulting in many potential solutions). These situations correspond to an over-fitting or an under-fitting respectively. [**wiki-tikhonob**] A simple solution to this linear algebra problem is to add a diagonal matrix, $\lambda\hat{I}$, term to the matrix that is to be inverted. This shrinks the regression coefficients so that $\hat{\beta}$'s with smaller norm are preferred. In this way we superimpose a quality control variable on our solution space allowing us to arrive at a unique solution with (relatively) small variance. [**hastie**] In the end we only need to adjust our linear algebra equation slightly

$$\hat{\beta} \quad = \left(\hat{X}^T\hat{X} + \lambda\hat{I}\right)^{-1}\hat{X}^T\hat{y} \tag{5}$$

There is a caveat here when considering the coefficient determining the y-intercept. If this is included in the regularization the result would depend on the origin chosen for y.[**hastie**] This is a problem as we expect that adding a constant to the response variables should simply shift the predictor variables by the same amount, not change the shape of our fit. As an illustration, if we return to our previous cannon and ball experiment, we don't want the time at which we start the clock, be it at the moment of launch or a minute before, to lead to a completely different polynomial fit. To avoid this problem it is possible to center the inputs by simply estimating β_0 by the average $\bar{y}$ and doing a ridge regression by equation 5 with the remaining coefficients but replacing the elements in $\hat{X}$ with $x_{ij} - \bar{x}_j$.

2.4 Lasso

The Lasso (least absolute shrinkage and selection operator) method works in much the same way as Ridge in that it incorporates variable selection and regularization in the regression. But the constraints makes the solutions nonlinear and there is no closed form expression as in the ridge regression.[**hastie**] We therefore need to specify the workings of the method more explicitly

$$\hat{\beta} = \text{argmin} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 \tag{6}$$

Under the restriction

$$\sum_{j=1}^{p} |\beta_j| \leq t \tag{7}$$

At first glance this expression may seem foreign when given on this form but it is essentially the same as for Ridge with the only difference being the restriction in Ridge

$$\sum_{j=1}^{p} {\beta_j}^2 \leq t \tag{8}$$

We recognize equation 6 as a function that, when we exclude β_0 due to centering, finds the minimum of SS_{res} with an additional constraint that discards some solutions. In the case of Lasso the constraint makes sure that the L_1 norm is less than t while Ridge considers the L_2 norm (i.e. euclidian norm).

2.5 K-fold

K-folding is a cross validation technique that allows us to generalize the trends in our data set to an independent data set. In this way we can circumvent typical problems like over-fitting and selection bias.[**cross-valid**] The approach for the technique is pretty straight forward. Instead of doing a regression on the entire data set, we first segment it into k number of subsets of equal size (making sure to pick out the variables randomly before distributing them to the subsets). Now we choose one subset to be the 'control' or 'validation' set while the rest of the subsets are the training sets. We then perform the regression we want to use on the training set, arriving at some data fitting that is our prediction. From here it is a straight forward process to compare how well our predicted variables compare to the validation variables, for example through the R^2 score function. However, even though our subsets are picked randomly, the validation subset we used could potentially not be a representative selection of the entire set. Therefore we make sure to repeat the process k times, each time using a new subset as the validation subset. After all this is done we can simply calculate the average of the scores to get the predictive power of our model. As an added benefit, since we are doing the calculations anyway, we can use the average of our predictions as our final fit. Cross validation techniques are extremely useful when the gathering of new data is difficult or, sometimes, even impossible, as we are using the extra computational power at our disposal to squeeze the most amount of relevant information out of our precious data.
Combining cross validation with the penalty parameter t used in Ridge or Lasso (eq: 7 and 8) can make for an even better fit. Instead of trying to pick a good t from our intuition we can actually get a numerical measure of the predictive power of our model as a function of t by performing the cross validation method for a range of t's, choosing the penalty parameter that optimizes the predictive power of the model.[**morten-reg**]

3 Method

3.1 Implementation of OLS, Ridge and Lasso

The OLS method was implemented as shown in equation 2 and the Ridge method as shown in the equation 5. The $\hat{x}$ matrix for both the methods, OLS and Ridge, is made as shown in the equation below:

$$\hat{x} = [x^0y^0, ..., x^0y^n, ..., x^1y^0, x^1y^{n-1}, ..., x^ny^0]$$

Scikit makes the $\hat{x}$ in a different way, but it does not matter for the result. As shown in the next section, Implementation.

Lasso is a bit harder to implement, so we used scikit's version for this.[**scikit**] In order to understand which coefficient that corrosponded to which polynomial degree, we compared scikits coefficient with ours from the OLS implementation. This becomes important in the Result section.

3.2 Implementation of MSE, R2 and β variance

The R2 score and MSE was implemented as shown in equation eq: R squared and eq: mse. All the methods just simple send their predictions to a function that knows the solution. For the β variance it is a different story; The implementation for variance was used with the k-fold algorithm. For each part, the value of β and β^2 were added to a sum, that was then divided by $k-1$. This was used in the equation below to obtain variance for the beta's:

$$\text{VAR} = \text{E}(\beta^2) - \text{E}(\beta)^2$$

3.3 Normalizing EVERYTHING!

So, this might sound weird; but you should know, that we know what we are doing. In our script, we have scaled x,y and z, so the largest value is 1. For x and y, we have also made the smallest value 0.

This is so the whole dataset fits in to a 1x1x1 cube (nice figures). And so the MSE is not a large number for the real data, but a small number that represents the relative MSE. This of course affects how the whole polynomial looks, but it does not change the essence of fitting. And can easily be reverted if wanted by the user.

4 Implementation

The three different algorithms discussed in section XXX was implemented in our script. It is a few different versions, but the ëvërsion contains all you need. All the scripts discussed in this report can be found at our github.

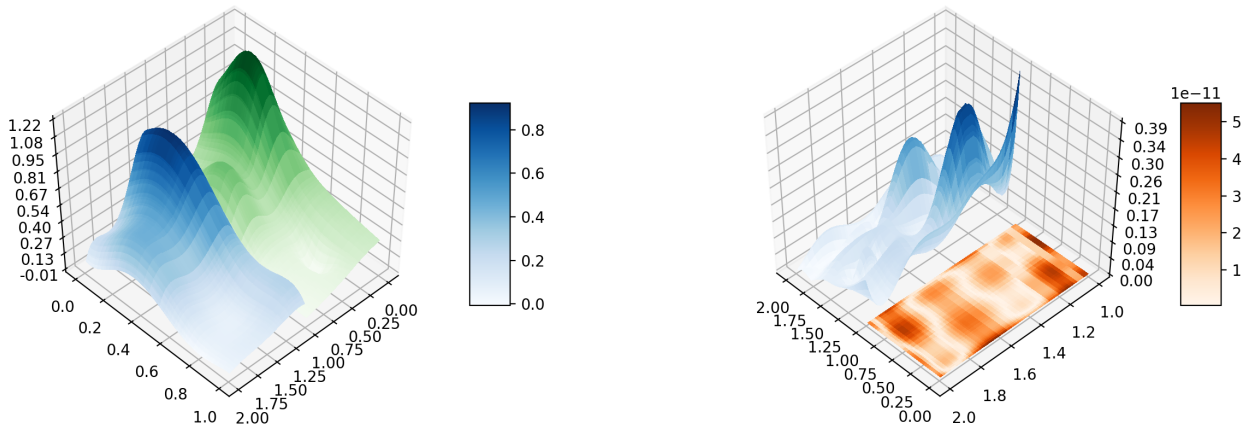
This part of the report is meant as a part where we show that our program works as we expected. We will look at how our implementation compare to Scikit's, how time increase as we increase the order we fit and how the modul crumbles as we add noise to our data.

The program was tested on the Frank-function, see equation 9. With an known solution we did a k-fold test and an degree and λ/α test. Both tested was done with the script descriped earlier. The tables below shows the different results.

$$f(x,y) = \frac{3}{4}e^{\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right)} + \frac{3}{4}e^{\left(-\frac{(9x+1)^2}{49}-\frac{(9y+1)}{10}\right)} + \frac{1}{2}e^{\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right)} - \frac{1}{5}e^{(-(9x-4)^2-(9y-7)^2)} \quad (9)$$

4.1 Scikit vs. manually implementation

Our implementation was verified with Scikit's OLS solution. The figur below shows the result from the comparison.



(a) (b)

Figure 1: a)Shows the Franke function, equation 9, as it is. The OLS implementation was used on the Franke function with noise. The result is shown as the blue graph. b)The blue graph is the error compared to the actual Franke function, not the dataed that OLS were trained on. The orange graph is the absolute error for our OLS compared to Scikit's.

4.2 Time evolution

Table 1: This tables shows how time develops as a function of order fitted. One should not be suprised that Scikit has a faster algorithm, than us. The $_{relative}$ means that it is as fraction of the second order fit time. For lasso and ridge the λ/α was set to 1e-5. It is expected for higher order fits to have longer calculation time, because the matrix $\mathbf{X}$ will get bigger. This is exactly what we see with our program.

degree ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
2		0.01517s	0.25830s	0.00516s	0.00543s
$2_{relative}$		1.00	1.00	1.00	1.00
$3_{relative}$		2.42	1.58	2.45	2.38
$4_{relative}$		3.63	2.45	5.11	4.88
$5_{relative}$		4.98	3.61	8.77	8.31

4.3 Noise - MSE & R2 evolution

Table 2: This tables shows how the MSE evoles for different amount of noise. As described in section 3.3, the Z data is normalized, so the MSE is 0.00127, when the highest values are 1. Which means that we have at least an error of 0.1% for the implementation on Franke function without noise. The $_{relative}$ means that it is as fraction of the zero noise data's MSE. For lasso and ridge the λ/α was set to 1e-5 and it was a fifth order fitting for all of the methods. The MSE grows as the noise increase to 50% of the data, which is expected!

Noise level ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
0		0.00127	0.00127	0.00514	0.00127
$0_{relative}$		1.00	1.00	1.00	1.00
$0.01_{relative}$		1.03	1.03	1.00	1.03
$0.2_{relative}$		12.84	12.84	3.68	12.84
$0.5_{relative}$		42.04	42.04	10.84	42.04

Table 3: This tables shows how the R2 score evoles for different amount of noise. For lasso and ridge the λ/α was set to 1e-5 and it was a fifth order fitting for all of the methods. Ideally we would want an R2 score of 1 for zero noise. The R2 score shrinks as the noise increase to 50% of the data, which is expected!

Noise level ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
0		0.98	0.98	0.91	0.98
0.01		0.98	0.98	0.91	0.98
0.2		0.68	0.68	0.62	0.68
0.5		0.28	0.28	0.25	0.28

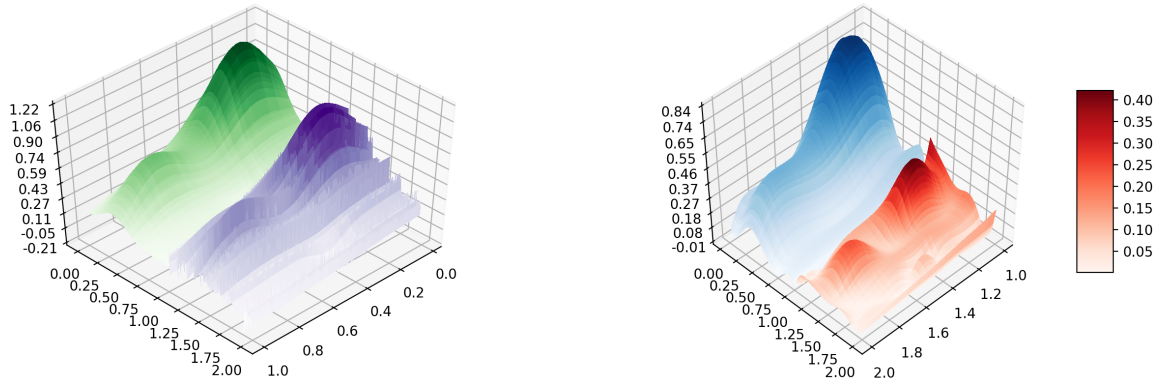
5 Result & Discussion

5.1 Ordinary least square, Ridge, and Lasso regression with resampling on the Franke function

In this subsection we will present our results from Ordinary least square, Ridge and Lasso regression, up to the fifth order, with a resampling technic, k-fold, on the Franke function. The Mean square error, R^2 score, the variance, and the confidence intervall from the calculations are also presented, these values were found by taking the average values over a 100 different executions, with a noise level = 0.1 and a $\lambda = 0.00001$.

5.1.1 Ordinary least square

Here we present our results on the OLS regression with up to a fifth order polynomial fit on Franke function, notice that in the plot we use a fifth order polynomial, the R^2 score and MSE according to order of the polynomial used for fitting of the data. And lastly a table containing the β values, the variance, and the confidence intervall according to the different polynomials.



(a) (b)

Figure 2: a) Franke function , and the Frank function with noise plottet. b) Our fifth order approximation of the Franke function. On the right we have the residuals, i.e. the error compared to the real function, and its relative size indicated by the red colour gradient.

Table 4: MSE and R^2 score for OLS by degree. These values are created by taking the average values over 100 different executions, with a noise level = 0.1 and a $\lambda = 0.00001$

Degree	R2	MSE
2	0.71	0.01353
3	0.81	0.00916
4	0.86	0.00712
5	0.88	0.00585

Table 5: β, Var and Confidence intervall for OLS by degree of x and y.

$x^i y^j$	β	VAR	Confidence intervall
x^0y^0	0.259	0.001	[0.227, 0.291]
x^0y^1	4.117	0.108	[3.788, 4.446]
x^0y^2	-18.065	2.943	[-19.781, -16.349]
x^0y^3	29.764	15.934	[25.772, 33.756]
x^0y^4	-22.199	17.571	[-26.391, -18.007]
x^0y^5	6.361	2.637	[4.737, 7.985]
x^1y^0	5.277	0.074	[5.005, 5.549]
x^1y^1	-9.751	1.234	[-10.862, -8.64]
x^1y^2	13.591	6.186	[11.104, 16.078]
x^1y^3	-21.609	7.858	[-24.412, -18.806]
x^1y^4	12.635	1.628	[11.359, 13.911]
x^2y^0	-22.726	1.825	[-24.077, -21.375]
x^2y^1	28.964	5.900	[26.535, 31.393]
x^2y^2	-1.849	6.149	[-4.329, 0.631]
x^2y^3	-5.401	1.364	[-6.569, -4.233]
x^3y^0	30.056	10.123	[26.874, 33.238]
x^3y^1	-36.035	7.072	[-38.694, -33.376]
x^3y^2	6.742	1.441	[5.542, 7.942]
x^4y^0	-11.919	12.008	[-15.384, -8.454]
x^4y^1	12.813	1.444	[11.611, 14.015]
x^5y^0	-0.909	1.951	[-2.306, 0.488]

5.1.2 Ridge regression

The results from the Ridge calculations are here presented in the same manner as for Ordinary least square. Again finding the MSE, R^2 score, according to polynomial degree, and testing for different λ. Finding the β values, VAR, and the Confidence intervall. We see here that the R^2 score is unstable for higher values of λ.

Table 6: MSE and R^2 score for Ridge by degree. These values are created by taking the average values over 100 different executions, with a noise level = 0.1 and a $\lambda = 0.00001$.

Degree	R2	MSE
2	0.71	0.01353
3	0.80	0.00968
4	0.81	0.00928
5	0.81	0.00902

Table 7: MSE and R^2 score for Ridge λ. These values are created by taking the average values over 100 different executions, with a noise level = 0.1.

λ	R2	MSE
0.0000001	0.81150	0.00894
0.0000100	0.80979	0.00932
0.0010000	0.74059	0.01237
0.1000000	-0.20901	0.05933
1.0000000	-1.85170	0.13474
2.0000000	-1.83823	0.14025
5.0000000	-1.79681	0.13813
10.0000000	-1.81950	0.13576

Table 8: β, Var and Confidence intervall for Ridge by degree of x and y.

$x^i y^j$	value	variance	Confidence intervall
x^0y^0	0.384	0.001	[0.352, 0.416]
x^0y^1	1.770	0.077	[1.493, 2.047]
x^0y^2	-0.294	1.953	[-1.691, 1.103]
x^0y^3	-22.690	10.474	[-25.926, -19.454]
x^0y^4	41.542	12.033	[38.073, 45.011]
x^0y^5	-20.675	1.949	[-22.071, -19.279]
x^1y^0	5.348	0.068	[5.087, 5.609]
x^1y^1	-11.544	1.040	[-12.564, -10.524]
x^1y^2	18.567	4.891	[16.355, 20.779]
x^1y^3	-27.573	6.071	[-30.037, -25.109]
x^1y^4	14.943	1.295	[13.805, 16.081]
x^2y^0	-21.987	1.712	[-23.295, -20.679]
x^2y^1	32.153	5.221	[29.868, 34.438]
x^2y^2	-5.093	5.179	[-7.369, -2.817]
x^2y^3	-3.142	1.161	[-4.219, -2.065]
x^3y^0	25.775	9.426	[22.705, 28.845]
x^3y^1	-39.259	6.976	[-41.9, -36.618]
x^3y^2	6.673	1.214	[5.571, 7.775]
x^4y^0	-4.833	11.244	[-8.186, -1.48]
x^4y^1	14.552	1.590	[13.291, 15.813]
x^5y^0	-4.681	1.908	[-6.062, -3.3]

5.1.3 Lasso regression

The results from the Lasso regression, again presenting the MSE, R^2, β, VAR, and Confidence interval. We see here that R^2 is also unstable for higher λ for Lasso regression.

Table 9: MSE and R^2 score for Lasso by degree. These values are created by taking the average values over 100 different executions, with a noise level = 0.1 and a $\lambda = 0.00001$

Degree	R2	MSE
2	0.71	0.01353
3	0.81	0.00916
4	0.86	0.00712
5	0.88	0.00585

Table 10: MSE and R^2 score for Lasso on Franke function by λ. These values are created by taking the average values over 100 different executions, with a noise level = 0.1

λ	R2	MSE
0.0000001	0.87497	0.00587
0.0000100	0.87517	0.00605
0.0010000	0.87350	0.00612
0.1000000	0.83831	0.00780
1.0000000	0.80694	0.00954
2.0000000	0.80072	0.00956
5.0000000	0.78651	0.01041
10.0000000	0.76854	0.01100

Table 11: β, Var and Confidence intervall for Ridge by degree of x and y. We have here cut the values for higher degrees because these were equal to zero.

$x^i y^j$	β	VAR	Confidens interval
x^0y^0	0.342129	0.000080	[0.333197,0.351061]
x^0y^1	-0.437389	0.000347	[-0.456013,-0.418765]
x^0y^2	-0.016612	0.000261	[-0.032765,-0.000458]
x^0y^5	0.000024	0.000000	[-0.000140,0.000188]
x^1y^0	0.558845	0.000299	[0.541543,0.576146]
x^1y^1	-0.389778	0.000394	[-0.409623,-0.369933]
x^3y^0	-0.000178	0.000001	[-0.000933,0.000576]
x^4y^0	-0.120878	0.000436	[-0.141758,-0.099998]

5.2 Ordinary least square, Ridge, and Lasso regression with resampling, now on real data

In this section we present the OLS, Ridge and Lasso regression on the real data, in the same manner as we did for the Franke function. We find: R^2 score, and MSE, as a function of λ and observe that R^2 score goes through the roof for higher values of λ, for both Lasso and Ridge regression, and limit ourself thereafter. The β, VAR, and Confidence interval is then presented according to degree of x and y. then we show the date for time as function of k - parts. Observing that the run-time goes down with a smaller $N \times N$ matrix, as expected. And finally presenting R^2 also as a function of k, for OLS, Ridge and Lasso. Here we observe that the R^2 score is surprisingly stable even though we are training on $k-1$ sets and testing on the last set.

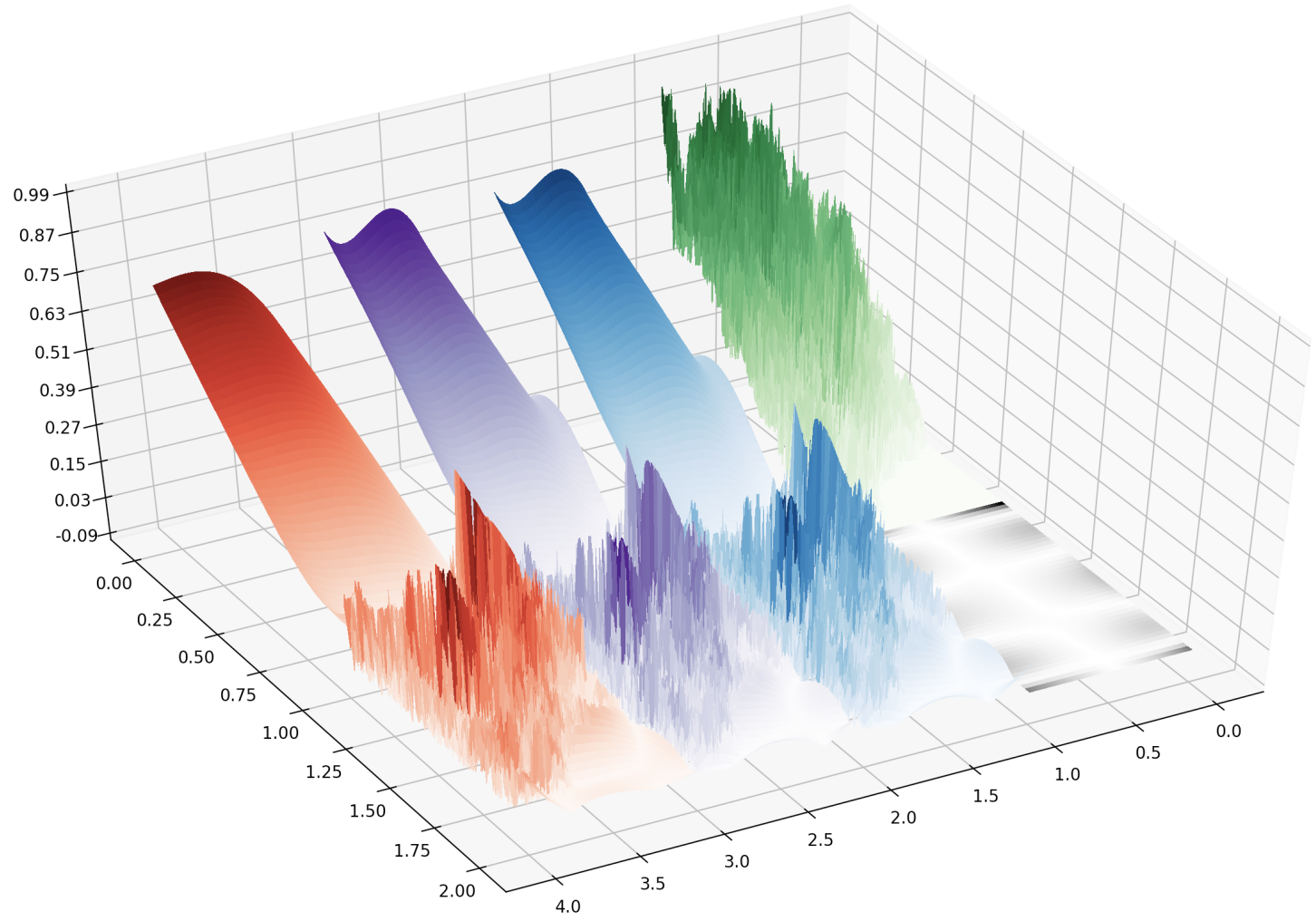


Figure 3: From left to right: The OLS regression, Ridge regression, Lasso regression, and the Real data that we tried to approximate, with their residuals.

Table 12: Ridge on realdata. MSE and R^2 score for Ridge on the real data by λ.

λ	R2	MSE
0.0000001	0.823088	0.010720
0.0000100	0.823088	0.010720
0.0010000	0.823027	0.010724
0.1000000	0.818116	0.011022

Table 13: Lasso on realdata. We saw that R^2 got out of hand with a $\lambda = 0.1$ and higher, and decided not to include those numbers.

λ	R2	MSE
0.0000001	0.797486	0.012272
0.0000100	0.797470	0.012273
0.0010000	0.753280	0.014950
0.1000000	-0.165026	0.070596

Table 14: The β values, VAR, and Confidence interval by degree for the Lasso regression.

$x^i y^j$	β	VAR	Confidence interval
$x^0 y^0$	0.342129	0.000080	[0.333197,0.351061]
$x^0 y^1$	-0.437389	0.000347	[-0.456013,-0.418765]
$x^0 y^2$	-0.016612	0.000261	[-0.032765,-0.000458]
$x^0 y^5$	0.000024	0.000000	[-0.000140,0.000188]
$x^1 y^0$	0.558845	0.000299	[0.541543,0.576146]
$x^1 y^1$	-0.389778	0.000394	[-0.409623,-0.369933]
$x^3 y^0$	-0.000178	0.000001	[-0.000933,0.000576]
$x^4 y^0$	-0.120878	0.000436	[-0.141758,-0.099998]

Table 15: The β values, VAR, and Confidence interval by degree for the Ridge regression

$x^i y^j$	β	VAR	Confidence interval
x^0y^0	-0.014747	0.000532	[-0.037820,0.008326]
x^0y^1	1.700777	0.262232	[1.188691,2.212863]
x^0y^2	-1.423363	7.920556	[-4.237711,1.390986]
x^0y^3	-7.494369	38.216859	[-13.676348,-1.312390]
x^0y^4	11.580568	35.505840	[5.621890,17.539246]
x^0y^5	-4.247212	4.498345	[-6.368142,-2.126282]
x^1y^0	-0.095820	0.012602	[-0.208080,0.016439]
x^1y^1	-2.779005	0.819186	[-3.684094,-1.873916]
x^1y^2	6.873662	2.015972	[5.453812,8.293511]
x^1y^3	-1.592223	0.802557	[-2.488079,-0.696368]
x^1y^4	-2.951493	0.510070	[-3.665684,-2.237301]
x^2y^0	11.119198	0.363834	[10.516011,11.722384]
x^2y^1	-14.485972	2.483606	[-16.061918,-12.910026]
x^2y^2	0.999862	5.328256	[-1.308439,3.308164]
x^2y^3	3.275747	1.446726	[2.072948,4.478546]
x^3y^0	-22.518257	3.845345	[-24.479212,-20.557302]
x^3y^1	24.970159	1.048842	[23.946030,25.994289]
x^3y^2	-5.075608	0.906148	[-6.027526,-4.123690]
x^4y^0	15.006599	5.479219	[12.665826,17.347372]
x^4y^1	-10.208141	0.083641	[-10.497349,-9.918933]
x^5y^0	-2.719594	0.888911	[-3.662415,-1.776773]

Table 16: Time as a function of k. And as expected the run-time goes down when we compute smaller $N \times N$ matrices.

k	2	5	10	50	100
time	79s	87s	81s	60s	51s

Table 17: R^2 as a function of k-fold, with k parts. Notice that R^2 is good even though we are training on $k-1$ sets and testing on the last set.

k	OLS	Ridge	Lasso
2	0.822566	0.822566	0.752957
5	0.822684	0.822683	0.753161
10	0.822793	0.822791	0.753464
50	0.822252	0.822254	0.754916
100	0.828201	0.828209	0.758450

6 Conclusion

We have seen how the different models behave. The time was almosst irrelevant for choosing OLS, Ridge or Lasso. The k-folding technique reduced the time and also preserved the R2 score and MSE. The Lasso might not be the best fit to the real data, but it has the simplest model. It has 8 coefficient, compared to 21 for Ridge and OLS. If the module was meant to predict new values, Lasso would save you 13 FLOPs per prediction, while only droping slightly in R2 score and MSE accuracy. A pretty good deal if many predictions are needed with the model. An other conclusion that might be as interesting is that we should keep using the Scikit package. Our code... Well, it did not perform as well as we would have liked.