Analysis of Regression and Resampling Methods

Håkon Olav Torvik, Vetle Vikenes and Sigurd Sørlie Rustad

University of Oslo
Norway

October 7, 2021

CONTENTS

INTRODUCTION

Regression analysis is a statistical method for fitting a function to data. It is useful for building mathematical models to explain noisy observations. There are several regression methods to achieve this, all with their strengths and weaknesses. We will in this paper study three different methods; ordinary least squares, Ridge and Lasso regression. All the code, results and instructions on running the code can be found in our GitHub repository[1].

The mathematical model obtained from regression analysis can be used to predict an outcome, given some previously untested input. To build an accurate model, one has to train it on a lot of data. Real-world datasets usually have a fixed size, and getting more data can be practically impossible. Training the model on the same data over and over will usually lead to overfitting, where the model exactly predicts the training set, but performs very poorly on new data. It is therefore useful to have tools to avoid this for small datasets. Resampling methods are such tools. In addition to the regression methods, we will also study the effect of bootstrapping and cross-validating the data.

In order to study this, we need data to analyze. We will in this paper test the methods on two datasets. One dataset we will generate ourselves using a noisy analytical bivariate function, the Franke function (1), while other is real-world terrain data. To both we will be performing a polynomial fit in x and y dependance of the form $[1, x, y, x^2, xy, y^2, \dots]$.

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

We will begin with using the simplest regression method, ordinary least squares (OLS) to fit polynomials up to fifth order on the Franke function. At first we will not resample the data, only test our implementation and do some basic analysis. We will then add the resampling methods, Bootstrapping and Cross-Validation. Then we will do the same analyses as before, but using the more complex regression methods Ridge and Lasso. At the end we will do all the same using the terrain data. This paper will not study the effects of different scaling methods, train-test split ratios and amount of noise, though all will be used.

EXERCISE 1: ORDINARY LEAST SQUARE (OLS) ON THE FRANKE FUNCTION

We generate a dataset drawing $n = 30$ uniformly distributed points in $x, y \in [0, 1]$. (1) is then used to create n^2 z-values. In this domain, (1) takes values $z \in [-0.45, 1.52]$ before scaling. To this we add normally distributed stochastic noise. The noise has a mean of $\mu = 0$, and variance $\sigma^2 = 1$. It is scaled by a factor ϵ, which we normally set to $\epsilon = 0.2$. This gives a substantial amount of noise, without completely drowning out the data. The data with and without noise is isualized in figure 1.

[1] https://github.com/sigurdru/FYS-STK4155/tree/main/project1

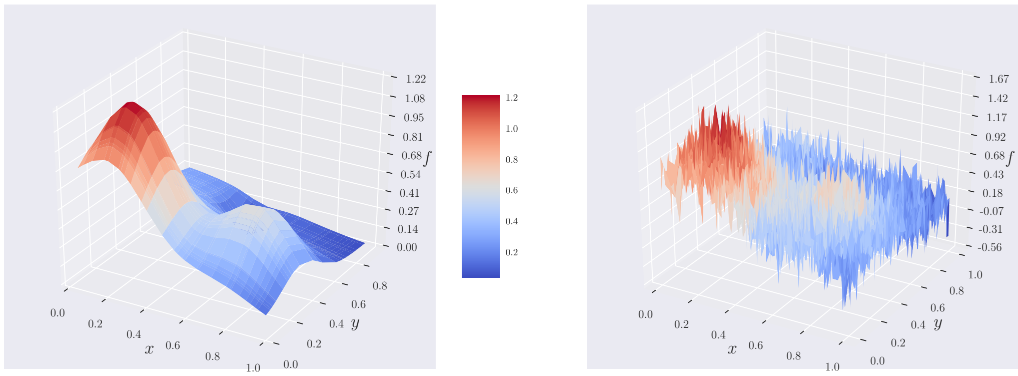

Figure 1. Here we have displayed our noisy data along with the analytical function. The left figure shows the analytical Franke function, while the right figure is the Franke function with noise.

It is good practice in regression analysis to scale the data, such that different features take values in the same order of magnitude. There are several different methods for scaling, one of which is StandardScaler. This subtracts from the data the mean, and diving by the standard deviation. This centres the data around 0 and sets the variance to 1. Because our x and y is already in the domain $[0, 1]$ with variance 1, this only shifts it down, and has no outcome on the regression results. For z generated using (1) the effect is small, but we choose to do this for consistency. Our code allows for different scaling methods, no scaling, Normalizer and MinMax, though these will not be used.

We are fitting a bivariate polynomial of degree P. With P as the highest polynomial degree, the polynomial will have $p = (P+1)(P+2)/2$ terms. It can, without noise, be written

$$z(x, y) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \dots \tag{2}$$

where the β-parameters are coefficients for each term. The x and ys are collected in a design matrix X. The columns of X are the terms, while the rows are all terms for a single datapoint. X thus have size $(n^2 \times p)$. The βs are collected in a vector $\boldsymbol{\beta}$ of length p.

Thus, out model with stochastic noise takes the form

$$\mathbf{z} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{3}$$

One challenge in our case is that our data is two-dimensional, i.e. $\mathbf{z}$ has dimensions (n, n). We create a workaround by mapping our two-dimensional data, to a one dimensional $(n^2, 1)$ vector. This is an important note, and we will do this throughout the report.

We want to create a model to predict $\mathbf{z}$ given an input.

$$\tilde{\boldsymbol{z}} = X\tilde{\boldsymbol{\beta}},$$

we will have a mean square error (MSE) given by

$$C(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=0}^{N-1}(z_i - \tilde{z}_i)^2 = \frac{1}{N}\left\{(\mathbf{z} - X\tilde{\boldsymbol{\beta}})^T(\mathbf{z} - X\tilde{\boldsymbol{\beta}})\right\}.$$

We call $C(\boldsymbol{\beta})$ our cost function, and is the quantity we want to minimize. The minima can be found by doing the derivative with respect to all the parameters, and set the result to zero

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = X^T(\boldsymbol{z} - X\tilde{\boldsymbol{\beta}}).$$

We can rewrite this as

$$X^T\mathbf{z} = X^T X\tilde{\boldsymbol{\beta}} \implies \tilde{\boldsymbol{\beta}} = (X^T X)^{-1} X^T\mathbf{z} \equiv \hat{\boldsymbol{\beta}}$$

Thus $\hat{\boldsymbol{\beta}}$ are the parameters that minimizes the cost function. Now that we have an expression for the optimal parameters, we also want to evaluate how good our results are. We do this by splitting our data into two sets, one set for training (80%) and one set for testing (20%). We can then find the optimal parameters ($\hat{\boldsymbol{\beta}}$) using the train data, and then compare our test data to the actual data.

There are many quantities we can use to evaluate our results, however we have decided to calculate the MSE and R^2 score. MSE is given by equation (4) and the R^2 score by equation (5).

$$MSE(\mathbf{z}, \tilde{\mathbf{z}}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 \tag{4}$$

$MSE(\mathbf{z}, \tilde{\mathbf{z}})$ is the mean square between the datapoints $\mathbf{z}$ and $\tilde{\mathbf{z}}$.

$$R^2(\mathbf{z}, \tilde{\mathbf{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z}_i)^2} \tag{5}$$

Here $R^2(\mathbf{z}, \tilde{\mathbf{z}})$ is the R^2 score between the datapoints $\mathbf{z}$ and $\tilde{\mathbf{z}}$. With this we can plot the quantities as a function of polynomial degree. We expect the MSE to always decrease and R^2 to increase for the train data. We expect the same for the test data, however at a certain number of polynomial degree we might se overfitting. This would make our error increase and R^2 score decrease. The results are plotted in figure 2.

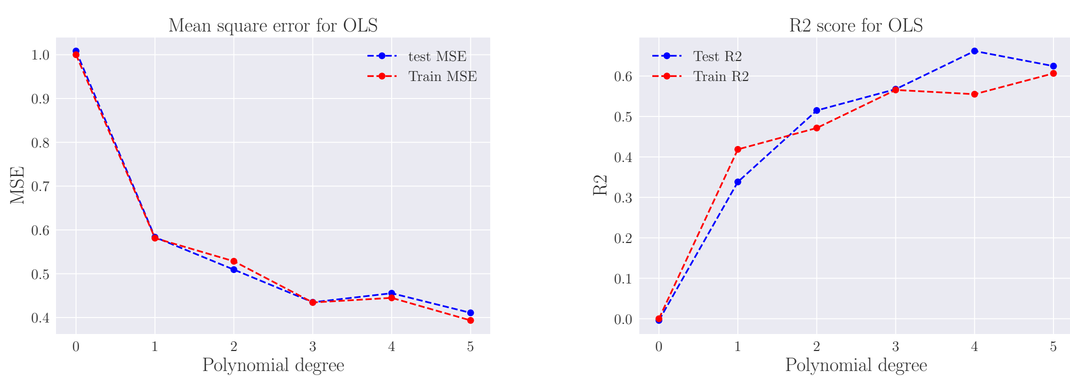

Figure 2. Here we have plotted the MSE and R^2 score for OLS. The left figure shows the MSE and right shows the R^2 score. The red dotted line is from the train data, and the blue dotted line is from the test data.

From figure 2 we see that the MSE strictly decrease and R^2 score increase. This indicates that we have not reached a point of overfitting yet.

It is also interesting to look at the variance in the parameters. We know that for a set of optimal parameters

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \mathbf{z},$$

We have the expectation value

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \mathbb{E}[(X^T X)^{-1} X^T \mathbf{z}] = (X^T X)^{-1} X^T \mathbb{E}[\mathbf{z}] = (X^T X)^{-1} X^T X \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}.$$

With this we can calculate the variance in $\hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}\left[(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^T\right] \\
&= \mathbb{E}\left[((X^T X)^{-1} X^T \mathbf{z} - \hat{\boldsymbol{\beta}})((X^T X)^{-1} X^T \mathbf{z} - \hat{\boldsymbol{\beta}})^T\right] \\
&= (X^T X)^{-1} X^T \mathbb{E}\left[\mathbf{z}\mathbf{z}^T\right] X (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T \\
&= (X^T X)^{-1} X^T \left[X \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T X^T + \sigma^2\right] X (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T \\
&= \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T + \sigma^2 (X^T X)^{-1} - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T = \sigma^2 (X^T X)^{-1}
\end{aligned}$$

Here we used $\mathbb{E}(\mathbf{z}\mathbf{z}^T) = X\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T X^T + \sigma^2 \mathbb{1}$, where $\sigma^2 = \epsilon = 0.2$ is the variance of the noise and $\mathbb{1}$ is the identity. Now we have an estimate for the variance in parameter $\hat{\boldsymbol{\beta}}_j$ given by

$$\boldsymbol{\sigma}^2(\hat{\boldsymbol{\beta}}_j) = \sigma^2 [(X^T X)^{-1}]_{jj} \tag{6}$$

Using equation (6) we can plot the confidence interval for the different parameters, for different polynomial degrees.

Figure 3. Here we have plotted the ideal parameters, for different polynomial degrees and with the confidence intervals. Top left figure has polynomial degree $p = 1$, top right $p = 3$ and bottom left $p = 5$. All the plots are of $\boldsymbol{\beta}$ as a function of the index in the vector. The confidence interval is $\pm 2\boldsymbol{\sigma}(\hat{\boldsymbol{\beta}}_j)$, where $\boldsymbol{\sigma}(\hat{\boldsymbol{\beta}}_j)$ is given by equation (6).

EXERCISE 2: BIAS-VARIANCE TRADE-OFF AND RESAMPLING TECHNIQUES

Before we perform an analysis on the bias-variance trade-off, we want to take a look at when we get overfitting. It is common to see overfitting when we have a high polynomial degree (p_{max}) compared to the number of datapoints. Thus in figure 4 we have plotted the test and train MSE for a high p_{max}. The left plot shows an indication of overfitting at polynomial degree $p = 9$. Thus on the right hand plot we have displayed the MSE for $p_{\text{max}} = 10$, and as expected we see overfitting at $p = 9$. One unexpected result is the sharp increase and decrease in MSE. This indicates that our model is much better for some polynomial degrees than others. The sharp fluctuations can be explained by the surge in complexity from one polynomial degree to the next. This is displayed in figure 3, when we increase the polynomial degree by two, the number of parameters more than double.

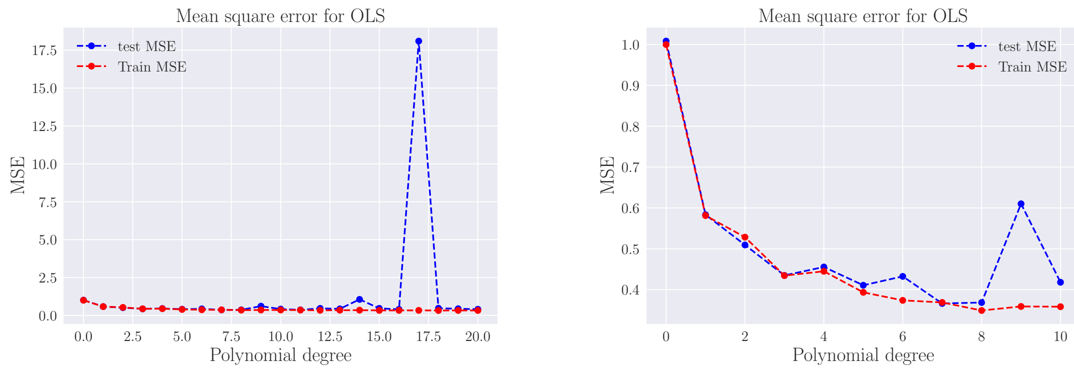

Figure 4. Overfitting, høyre med støy, venstre uten støy.

Before we take a look at bias variance trade-off, we will derive the expressions. With our data model given in equation (3) we find our model by considering the cost function, given as

$$C(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=0}^{N-1}(z_i - \tilde{z}_i)^2 = \mathbb{E}[(\mathbf{z} - \tilde{\mathbf{z}})^2]$$

where $\mathbb{E}$ is the expected value.

We can show that this can be written as [2]

$$\mathbb{E}[(\mathbf{z} - \tilde{\mathbf{z}})^2] = \frac{1}{N}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \frac{1}{N}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \sigma^2 \tag{7}$$

where f_i is the true data value at point i.

In equation (7) the first term represents the square of the bias, the second term represents the variance while the last term represents the variance of the irreducable error ϵ. When performing linear regression the variance is a measurement on how much our model changes with different training sets. High variance will therefore occur if a different training set resulted in very different values of the individual estimators, $\boldsymbol{\beta}$. This will be the case for overfitting when our model is essentially trying to reproduce variations from the noise. The bias provides information about the difference between our model and the true data values. If our model is missing out on underlying structures in our data we would get a high bias. High bias will thus be the case for an underfitted model. Our goal is therefore to minimize the bias and variance in our model.

Using equation (7) we can plot the bias, variance and MSE for the test and train data, using the bootstrap method. For both datasets we expect the bias and MSE to start high and variance to start low. For the train data we should get a strictly better fit for higher polynomial degree, making both the bias and MSE to decrease. For the testing data however, when we start to see overfitting (around $p = 9$) variance and error should increase, while bias stays low. In figure 5 we have plotted bias, variance and MSE for the test and train data, using $r = 30$ resampling iterations. Because we expected to see overfitting at around $p = 9$ we plotted for $p_{\max} = 10$.

[2] The derivation is given in Appendix A

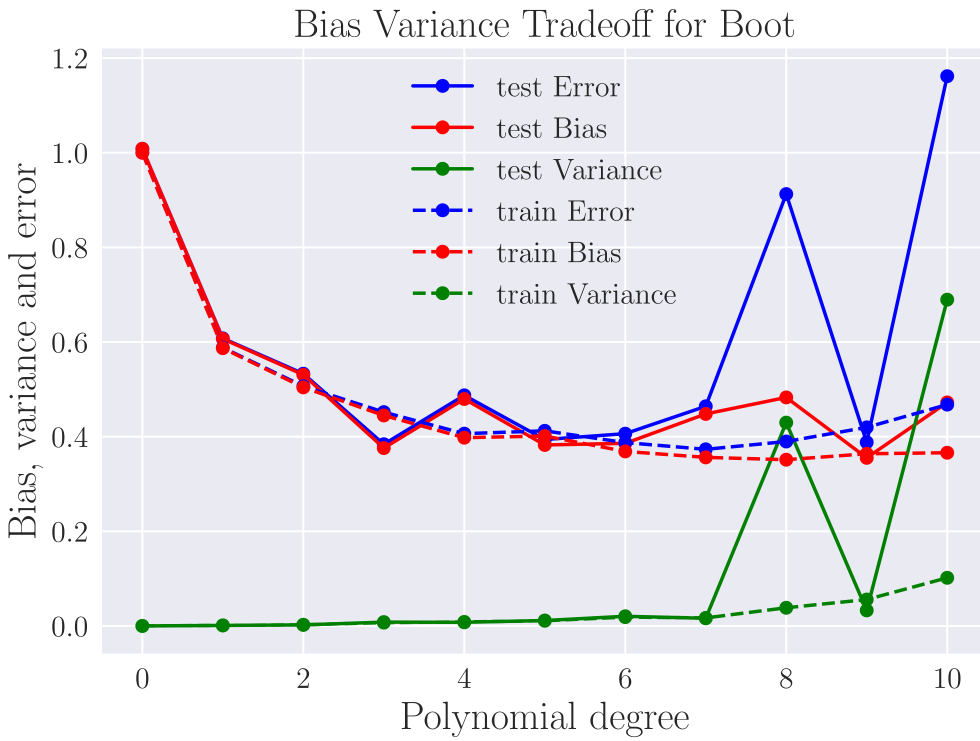

Figure 5. Here we have plotted the bias, variance and MSE for the test and train data. The x-axis shows the polynomial degree, and the y-axis indicates the bias, variance and MSE. The dotted lines are for the train data and solid lines are the test data.

As expected we se an increase in MSE and variance for the test data, when we start overfitting. The bias also acts as expected, decreasing in the beginning and staying low. One thing we also saw in the overfitting experiment (figure 4) is the sharp increase and decrease for specific polynomial degrees.

EXERCISE 3: CROSS-VALIDATION AS RESAMPLING TECHNIQUES, ADDING MORE COMPLEXITY

EXERCISE 4: RIDGE REGRESSION ON THE FRANKE FUNCTION WITH RESAMPLING

EXERCISE 5: LASSO REGRESSION ON THE FRANKE FUNCTION WITH RESAMPLING

EXERCISE 6: ANALYSIS OF REAL DATA

Appendix A: Bias-variance Decomposition

We assume that our true data is generated from a noisy model with nromally distributed noise ϵ with a mean of zero and standard deviation σ^2, i.e.

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

We have approximated this function with our design matrix $\mathbf{X}$ and our parameters $\boldsymbol{\beta}$ such that our model becomes $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$, where the values of $\boldsymbol{\beta}$ were obtained by optimizing the mean squared error via the cost function, given by

$$C(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right]$$

where $\mathbb{E}$ is the expected value.
We want to show that the above expression can be written as

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

We begin by inserting our model expression for $\mathbf{y}$ and adding and subtracting $\mathbb{E}[\tilde{\mathbf{y}}]$ inside the expected value, before we square the expression.

$$\begin{aligned}
\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] &= \mathbb{E}\left[(f(\mathbf{x}) + \boldsymbol{\epsilon} - \tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}])^2\right] = \mathbb{E}\left[((f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}]) + \boldsymbol{\epsilon} + (\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}))^2\right] \\
&= \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \boldsymbol{\epsilon}^2 + (\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2\right] \\
&\quad + \mathbb{E}\left[2\boldsymbol{\epsilon}(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}]) + 2\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}) + 2(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\right]
\end{aligned}$$

where the cross terms have been written on a separate line since the expected value is linear. Next we will focus on the cross-terms. Since $\boldsymbol{\epsilon}$ is normally distributed, it's expected value is simply the mean, which is zero in our case. The two cross terms involving $\boldsymbol{\epsilon}$ is therefore zero, so we only need to consider

$$\mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\right] = \mathbb{E}\left[f(\mathbf{x})\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[f(\mathbf{x})\tilde{\mathbf{y}}\right] - \mathbb{E}\left[\mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}[\tilde{\mathbf{y}}]\right] + \mathbb{E}\left[\tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}]\right]$$

Since the expected value of an expected value is just the expected value itself the last two terms in the above equation both become $\mathbb{E}[\tilde{\mathbf{y}}]^2$, canceling each other out. Using that $f(\mathbf{x})$ is a deterministic function, we have $\mathbb{E}[f(\mathbf{x})] = f(\mathbf{x})$. Expressing $f(\mathbf{x})$ in terms of its expected value, we can write the first two terms in the above equation as

$$\begin{aligned}\mathbb{E}\left[f(\mathbf{x})\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[f(\mathbf{x})\tilde{\mathbf{y}}\right] &= \mathbb{E}\left[\mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}]\right] - \mathbb{E}\left[\mathbb{E}\left[f(\mathbf{x})\right]\tilde{\mathbf{y}}\right] \\ &= \mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}\left[f(\mathbf{x})\right]\mathbb{E}[\tilde{\mathbf{y}}] = 0\end{aligned}$$

Hence, all the cross terms in the expected value cancel out, and we're left with

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

Using that $\mathbb{E}[\epsilon^2] = \sigma^2$ and writing the expected values as sums with the notation $f(\mathbf{x}_i) = f_i$, we get the desired expression. Since we have chosen $\mathbf{z}$ as our data variable we replace all the y variables with z, yielding

$$\mathbb{E}\left[(\mathbf{z}-\tilde{\mathbf{z}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \frac{1}{n}\sum_i (\tilde{z}_i - \mathbb{E}[\tilde{\mathbf{z}}])^2 + \sigma^2 \tag{A1}$$

which is what we wanted to show.

Appendix B: Testing our implementation

In order to make sure our algorithms are running correctly, it is necessary to perform tests. We did this by comparing our results to those produced by scikit-learn. First of all we generated some simpler data for testing, namely an exponential:

$$f_{\mathrm{Test}}(x) = exp(x) + \epsilon. \tag{B1}$$

Here ϵ denotes normally distributed noise, and x runs from $x_{\mathrm{min}} = 0$ to $x_{\mathrm{max}} = 1$ in $N = 50$ randomly distributed steps. This generates the testing data visualized in figure 6.

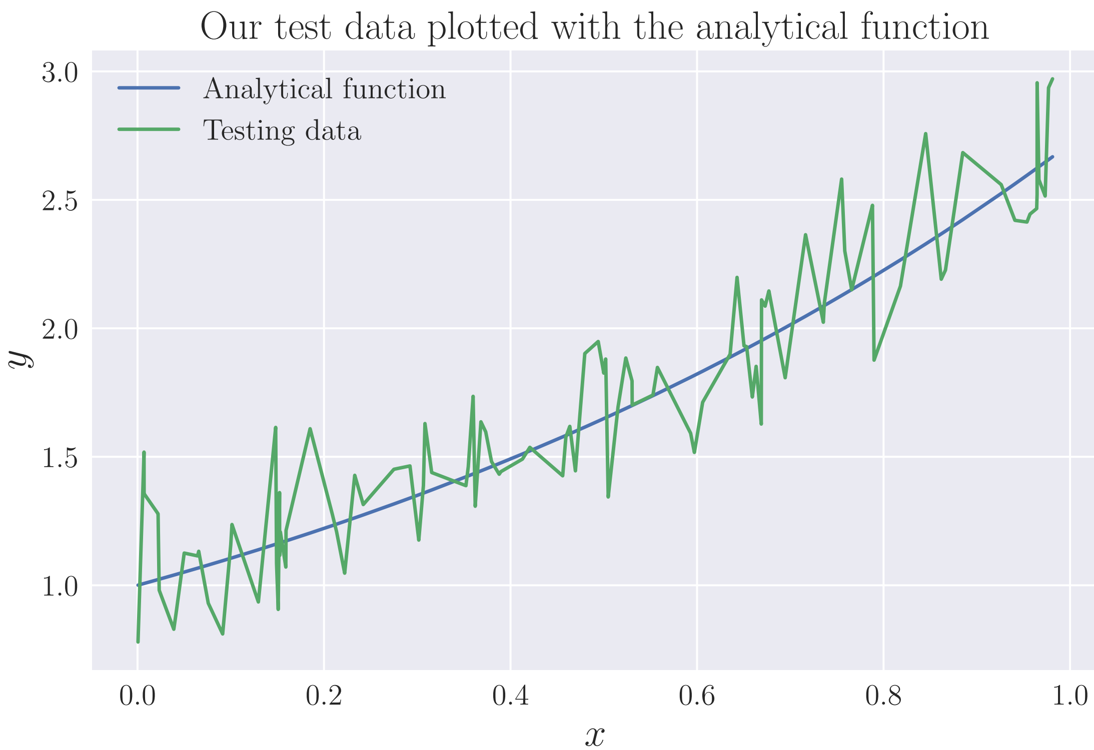

Figure 6. Here you we have plotted the testing data along with the analytical function.

First off we want to test the regression methods we have written.

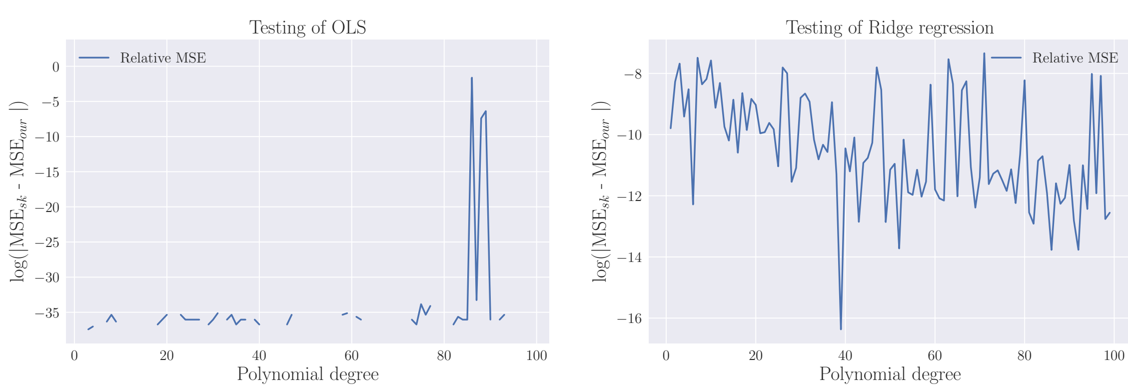

Figure 7. HEI EHIEHIAHFIASJDFKASDFJASDKF

[1] Ref.