Regression models applied to Runge's Function
FYS-STK4155 Project 1

Helene Lane and Manuela Leal Nader
University of Oslo
(Dated: October 9, 2025)

Runge's function has an interesting behavior when approximated by high-order polynomials. In this report, we have investigated it in more detail with three different regression methods: Ordinary Least Squares (OLS), Ridge and Lasso. Also, gradient descent and its variations have been used to obtain the coefficients numerically, as well as resampling techniques. We have achieved a good fit, based on the mean squared error (MSE) and the R^2 metric, for the one-dimensional Runge's function and all the above mentioned methods, with OLS giving slightly better results.

I. INTRODUCTION

Machine learning has spread to all areas of daily life, bringing different challenges depending on how it is used. Within science, these methods are vastly used across research areas. Not only that, they are constantly being improved and new methods developed, posing a challenge in regard to how to choose among them. When it comes to the type of training data available, there are two types of learning models, supervised and unsupervised ([1]). When the training data already has the property you want to predict, we say that the learning process is supervised. That is the type of problem we will focus on here. Models can also be divided into two other categories, regression and classification [1]. Regression models are the ones that deals with continuous functions, while classification models aims to predict discrete ones.

In this report, we will explore three different regression methods to find out which one best fits Runge's function: Ordinary Least Squares (OLS), Ridge and Lasso. Runge's function is:

$$f(x) = \frac{1}{1+25x^2} \tag{1}$$

This function was chosen because it is a function which Runge discovered that approximating it with higher order polynomials can be tricky [2].

The linear regression methods were chosen because they are easy to implement by people who have just started learning about machine learning. These methods will be applied along with several different types of gradient descent and two different resampling techniques.

In section II, we will cover the three regression methods used and the different ways of obtaining the optimal coefficients of them used in our analysis. First, we will describe OLS, Rdige and Lasso regression. Then, we will introduce gradient descent, gradient descent with momentum, AdaGrad, RMSProp, ADAM and stochastic gradient descent. Also, we will present two resampling techniques: bootstrapping and cross-validation. Then, we will briefly describe how we implemented these methods in our code. Lastly, we will declare our use of AI tools in this project.

In section III, we will present our results using these various regression models for Runge's function. We will compare the methods used and consider the bias-variance trade off in our analysis. Finally, section IV concludes our report with a summary.

II. METHODS

A. Scaling

Scaling the data is important in many machine learning algorithms. If the features in our data are measured on proportionally very different scales, scaling can prevent the algorithm from only optimising on the feature with largest error [3]. Standardising is a form of scaling the data that makes sure that each feature has mean 0 and standard deviation 1, which is often a good choice because it does not change the shape of the distribution [3]. We chose to scale our data because we knew we were going to apply other models, not only OLS. Ridge and Lasso need the data to be scaled since this changes the expectation value. Scaling is heavily recommended in the literature, but different types of scaling have different challenges.

B. Ordinary Least Squares

Ordinary Least Squares is a form of linear regression with the aim of minimizing the squared distance from each data point to our suggested function [1].

The cost function for OLS is given by the squared distance as shown in Equation 2.

$$C\left(X,\theta\right) = \frac{1}{n}\left\{\left(\boldsymbol{y}-X\theta\right)^T\left(\boldsymbol{y}-X\theta\right)\right\} \tag{2}$$

We wish to minimize the distances, which means we have to minimize the cost function. Minimizing the cost function means taking the derivative with respect to θ and setting it equal to zero (Equation 3).

$$\frac{\partial C\left(\theta\right)}{\partial\theta} = 0 = X^T\left(\boldsymbol{y}-X\theta\right) \tag{3}$$

Solving for θ gives the following expression (Equation 4)

which gives the optimal parameters.

$$\hat{\theta}_{OLS} = \left(X^T X\right)^{-1} X^T \boldsymbol{y} \tag{4}$$

C. Ridge Regression

Another common linear regression method is called Ridge Regression [4]. It solves the singularity problem when the matrix $X^T X$ cannot be inverted by adding a regularization parameter (λ) to the cost function [1], as in equation 5.

$$C\left(X, \theta\right) = \frac{1}{n}\left\{\left(\boldsymbol{y} - X\theta\right)^T \left(\boldsymbol{y} - X\theta\right)\right\} + \lambda\theta^T\theta \tag{5}$$

Again, by minimizing the derivative with respect to θ, the analytical equation for the optimal parameters is obtained (Equation 6).

$$\hat{\theta}_{Ridge} = \left(X^T X + \lambda I\right)^{-1} X^T \boldsymbol{y} \tag{6}$$

At the same time, the addition of λ introduces a penalty on the size of parameters [5].

D. Lasso Regression

Lasso Regression proposes another modification to the OLS method [6]. Here, the regularization parameter is accompanied by norm-1, instead of norm-2, of θ, as in equation 7.

$$C(\boldsymbol{\theta}) = \frac{1}{n}\left\{\left(\boldsymbol{y} - X\theta\right)^T \left(\boldsymbol{y} - X\theta\right)\right\} + \lambda|\theta| \tag{7}$$

This change reduces the risk of overfitting for small training sets, as it reduces some of the parameters to zero [5]. However, it comes at a cost: there is no analytical expression for the optimal parameters. So, a numerical approach is required to solve the optimization problem, which is, in this case, finding the parameters that minimize the derivative of the Lasso cost function.

E. Gradient Descent

Gradient descent (GD) is a commonly used method for solving an optimization problem, such as obtaining the optimal parameters for Lasso Regression. It is based on the fact that the negative gradient of a function provides a direction towards its minimum [7]. Iteratively, it is possible to search for the optimal parameters $\hat{\theta}$ that minimize the cost function $C(\theta)$.

The simplest way to perform a GD optimization is by defining a constant step by which to move in the direction of minimum [5]. This constant is often called the learning rate (η). Equation 8 exemplifies this iterative procedure.

$$\theta_{i+1} = \theta_i - \eta\Delta C(\theta_i) \tag{8}$$

However, the choice of learning rate can have a great impact on the convergence and computational cost of the model. For that reason, several methods have been developed to iteratively update this parameter. Here, we will mention some that were used in this report.

1. *Momentum based Gradient Descent*

The momentum method uses previous gradient evaluations to update the size of the step towards the minima [7]. A "velocity" term is introduced that accumulates the gradients. This term is then multiplied by a momentum hyperparameter (γ). The iterative procedure can be exemplified by equations 9 and 10.

$$v_i = \gamma v_{i-1} + \eta\Delta C(\theta_i) \tag{9}$$

$$\theta_{i+1} = \theta_i - v_i \tag{10}$$

2. *AdaGrad method*

The AdaGrad algorithm saves the sum of the squared values of the previous gradients and iteratively changes the learning by a factor inversely proportional to the square root of this sum [7]. Its algorithm is described by equations 11 and 12. Here, $g(\theta_i) \circ g(\theta_i)$ corresponds to an element-wise square of the gradient vector and ϵ is a small constant added to ensure that the denominator is never zero.

$$r_i = r_{i-1} + g(\theta_i) \circ g(\theta_i) \tag{11}$$

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{r_i} + \epsilon} g(\theta_i) \tag{12}$$

3. *RMSProp method*

The RMSProp method introduces a modification to AdaGrad [7]: Instead of saving the sum of squared gradients, it saves an exponentially decaying average (Equation 13). For this reason, it introduces a new hyperparameter (ρ). The update is done as in equation 12.

$$r_i = \rho r_{i-1} + (1 - \rho)[g(\theta_i) \circ g(\theta_i)] \tag{13}$$

4. *ADAM method*

ADAM can be interpreted as a modification to RMSProp and momentum methods combined [7]. It uses two exponential moving averages, one for the first-order (Equation 14) and another for the second-order (Equation 15) moments of the gradient, which are also bias-corrected (Equations 16 and 17). This is also introduced

to avoid initialization bias. The algorithm is exemplified by the following equations:

$$m_i = \beta_1 m_{i-1} + (1-\beta_1) g(\theta_i) \tag{14}$$

$$v_i = \beta_2 v_{i-1} + (1-\beta_2)[g(\theta_i) \circ g(\theta_i)] \tag{15}$$

$$\hat{m}_i = \frac{m_i}{1-\beta_1^i} \tag{16}$$

$$\hat{v}_i = \frac{v_i}{1-\beta_2^i} \tag{17}$$

$$\theta_{i+1} = \theta_i - \frac{\eta}{\hat{v}_i + \epsilon}\hat{m}_i \tag{18}$$

F. Stochastic Gradient Descent

Not only can the choice of learning rate have a great impact on the performance of the optimization procedure, but the evaluation of the gradient at every step can easily become a bottleneck. Stochastic Gradient descent proposes an alternative to that by evaluating the gradient at random samples of the original dataset [7]. It introduces two new hyperparameters: the number of these random samples, usually called mini-batches, and the number of epochs, how many iterations are performed over those mini-batches.

G. Resampling techniques

Resampling methods are extremely important for modern statistics [8]. These methods allow us to obtain more data for training our model without actually getting more data. In this project we generate our data sets using the random function from numpy, but in the real world data is often scarce. Being able to reuse data in a statistically sound way for the training of a model ensures that we can get a better trained model in situations when data is scarce. Resampling techniques can also be used to estimate the expected test error [1].

1. Bootstrap

Bootstrapping is a resampling technique where after splitting the original data set into test and training data we resample with replacement from the training data [1]. The resampling is repeated B times and this is referred to as the number of bootstraps. Each resampling draws with replacement n from the training set which has n elements. This ensures that each resample is likely different from the original training set.

Applying the bootstrap technique to OLS which has the cost function shown in Equation 2 which gives the expression in Equation 19.

$$C(X,\theta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] \tag{19}$$

We can expand the cost function in terms of expectation value (Equation 20).

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[\boldsymbol{y}^2\right] - 2\mathbb{E}\left[\boldsymbol{y}\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}^2\right] \tag{20}$$

The three terms can be written as seen in respectively Equation 21, Equation 22 and Equation 23.

$$\begin{aligned}\mathbb{E}\left[\boldsymbol{y}^2\right] &= \mathbb{E}\left[(\boldsymbol{f}+\epsilon)^2\right] \\ &= \mathbb{E}\left[\boldsymbol{f}^2\right] + 2\mathbb{E}\left[\boldsymbol{f}+\epsilon\right] + \mathbb{E}\left[\epsilon^2\right] \\ &= \boldsymbol{f}^2 + \sigma^2\end{aligned} \tag{21}$$

$$\mathbb{E}\left[\boldsymbol{y}\tilde{\boldsymbol{y}}\right] = \mathbb{E}\left[(\boldsymbol{f}+\epsilon)\,\tilde{\boldsymbol{y}}\right] = \mathbb{E}\left[\boldsymbol{f}\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\epsilon\tilde{\boldsymbol{y}}\right] = \boldsymbol{f}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] \tag{22}$$

$$\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] = \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2 \tag{23}$$

Combining the three terms again in Equation 24

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] &= f^2 + \sigma^2 - 2f\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2 \\ &= f^2 - 2f\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2 + \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \sigma^2 \\ &= \mathbb{E}\left[\left(f - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2\right] + \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \sigma^2\end{aligned} \tag{24}$$

The final step is to approximate f with y which gives us the final expression (Equation 25). Here the first term is the bias of the model and corresponds with how far the expectation value predicted by the model is from the true expectation value. The second term is the variance of the model and reflects how big the spread of our model is. The last term is the variance of the noise and depends on the data and not on the model.

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[\left(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2\right] + \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \sigma^2 \tag{25}$$

2. Cross-validation

Unlike the bootstrap method cross validation does not split the data set randomly. K-fold cross validation splits the data into k sets and iteratively uses the remaining k-1 sets as training data and the last set as test or validation data. The literature disagrees on whether the data should be split into test and training data before doing the k-fold split [1], or if the k-fold split is splitting into test and training [8]. In this project we follow [8] and do the k-fold split on the complete data set.

H. Implementation

The general steps of our code implementation starts by splitting the data into test and training data. Secondly, the data is scaled using scikitlearn's `StandardScaler`. Thirdly, we calculate the optimal parameters using our own regression functions. Lastly, we calculate the error.

1. *OLS and Ridge*

We used the analytical expressions for $\hat{\theta}$ for both OLS and Ridge. An example of how these were called and applied can be found below.

```
for p in degrees:
    ...
    beta_ols = OLS_parameters(X_train_s, y_train)
    beta_ridge = Ridge_parameters(X_train_s, y_train, 0.001)
    y_ols_tilde = X_test_s @ beta_ols + y_offset
    y_ridge_tilde = X_test_s @ beta_ridge + y_offset
```

2. *Resampling methods*

For the k-fold cross validation analysis we decided to shuffle the order of the data by shuffling the x-array after generation. We generate the x-values using `linspace` from the numpy package. This generates n floats between -1 and 1 ordered. To avoid our data being ordered in ascending order by x we used `shuffle` to shuffle the order but keep all the data before calculating our y-data.

We wrote our own code for the bootstrap method, but used scikitlearn's `KFold` function for k-fold cross validation.

I. Use of AI tools

We used GPT UiO for information on how to write an abstract and how to use some functionalities in Latex. The exported conversation is in the Github repository of the project (m-nader/FYS-STK4155-group14), inside a folder called LLM. Also, we used Copilot and GPT UiO to debug the code.

III. RESULTS AND DISCUSSION

We applied our OLS model to fit the Runge's function with polynomials of degrees varying from 1 to 15. First, we used 100 points uniformly distributed between -1 and 1. Then, we increased our dataset to 1000 points. Stochastic noise was introduced on the basis of the normal distribution. To obtain meaningful results, we reduced this noise by a factor of 0.1.

Both the mean squared error (MSE) and the R^2 metric were evaluated as a function of the polynomial degree. The results are shown in figure 1.

When analysing these metrics, we are looking for a low MSE and an R^2 value that is as close to 1 as possible. Based on figure 1, it is possible to notice that a better fit is obtained as we increase the degree of the polynomial until a plateau is reached. Not only that, it is also clear that the bigger the training set is, the higher the quality of the fit. The obtained theta values are plotted in Appendix A.

All our calculations from now on are based on the model in figure 2. It was obtained with a dataset of 1000 points and fitted to a polynomial of degree 10.

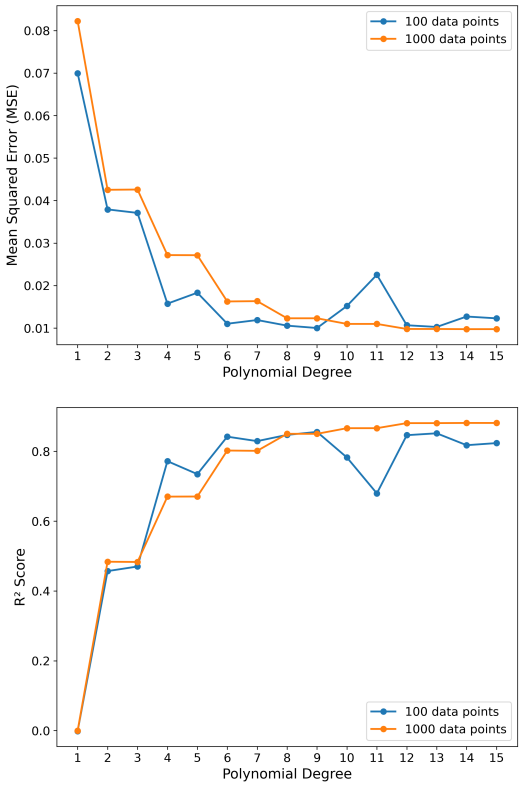

Figure 1: Error analysis performed on the test set of an OLS model applied to Runge's equation as a function of the degree of the fitted polynomial. Orange line indicates errors calculated based on a training set of 100 points and blue lines on one of 1000 points. Upper panel shows the MSE and lower panel the R^2 metric.

Ridge regression was also performed with different lambda coefficients. The error analysis can be seen in figure 3.

Figure 3 shows that the worst fit is obtained as the regularization parameter increases. This is because the Ridge regression attempts to shrink the parameters, which reduces the quality of the fit. Figure 4 presents a visual representation of this shrinkage. For a low regularization parameter, the theta values obtained through the Ridge regression are almost the same as those obtained with OLS. However, as this parameter increases, a shrinkage of theta values is observed.

So far, we have discussed models obtained through the

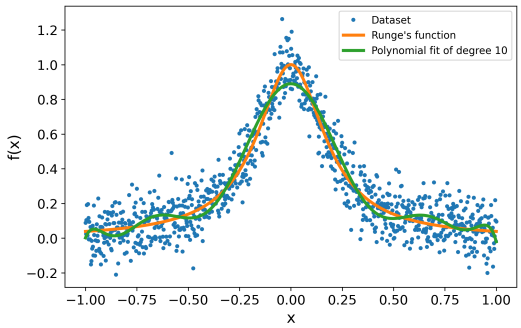

Figure 2: Visual representation of the dataset used in this report (blue dots) which corresponds to some noise added to the Runge's function (Orange line) and the polynomial fit of degree 10 obtained with OLS (Green line).

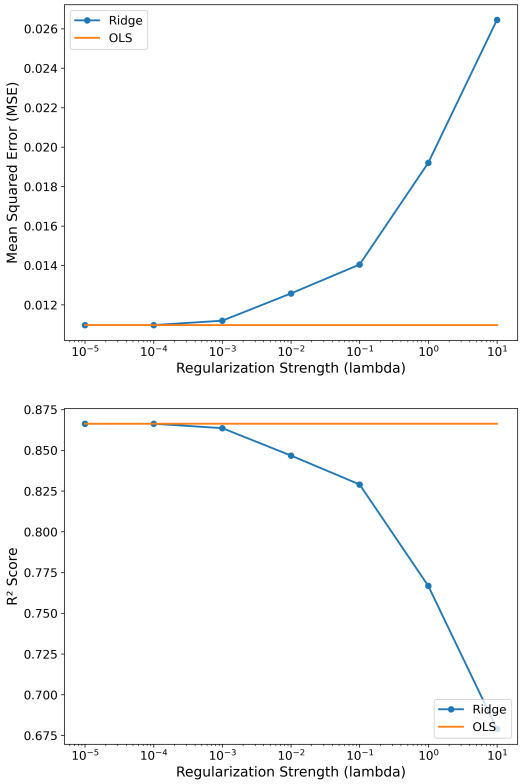

Figure 3: Error analysis performed on the test set of a Ridge model applied to Runge's equation as a function of the lambda parameter is shown in blue. The orange line shows the metrics for an OLS model of same polynomial degree (10). Upper panel shows the MSE and lower panel the R^2 metric.

analytical expressions for the optimal parameters of OLS and Ridge regression. However, there are methods that do not have an analytical expression. For that, it is necessary to turn to numerical methods. Here, the method of choice is gradient descent, which introduces a new hyperparameter, the learning rate. We have investigated the impact of different values for the learning rate (Figure 5).

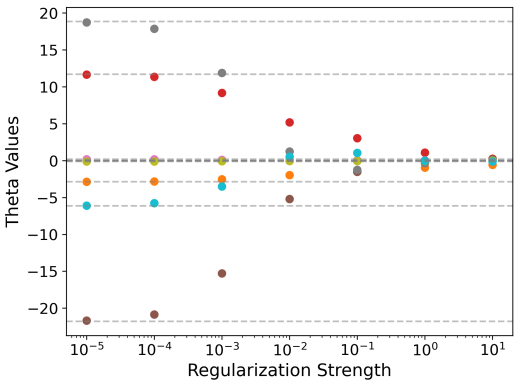

Figure 4: Theta values as a function of the regularization parameter for a Ridge model of polynomial degree 10. Dots of the same color indicate that they are the same coefficient. Gray dashed lines indicates the theta values for an OLS model of same polynomial degree.

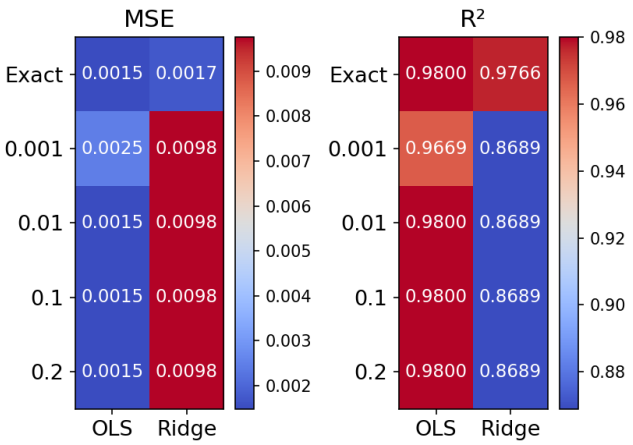

Figure 5: MSE and R^2 analysis of OLS and Ridge fits, obtained numerically through plain gradient descent optimization with different values for the learning rate, for the Runge's function.

Based on figure 5, we see that all fits using gradient descent provide the same error, except for the smallest learning rate combined with OLS. That is probably because, as the learning rate is too small, it did not reach convergence within the maximum number of iterations (set to 10^6).

We also observe that, for Ridge regression, the error is bigger when using numerical optimization than when the optimal parameters are obtained through the exact equation. This could also be because convergence has not yet been reached. It is also noteworthy that, if we increased the learning rate to 0.3, we obtained a not-a-number (NAN) for the parameters. This illustrates that the use of big learning rates prevents convergence.

Furthermore, we investigated the use of variations of the gradient descent method: momentum, AdaGrad, RMSProp and ADAM. Figure 6 presents the same er-

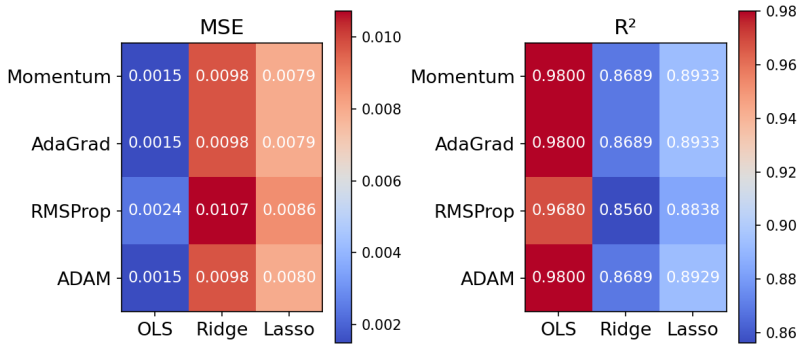

Figure 6: MSE and R^2 analysis of OLS, Ridge and Lasso fits, obtained numerically through variations of the gradient descent optimization and a learning rate of 0.01, for the Runge's function.

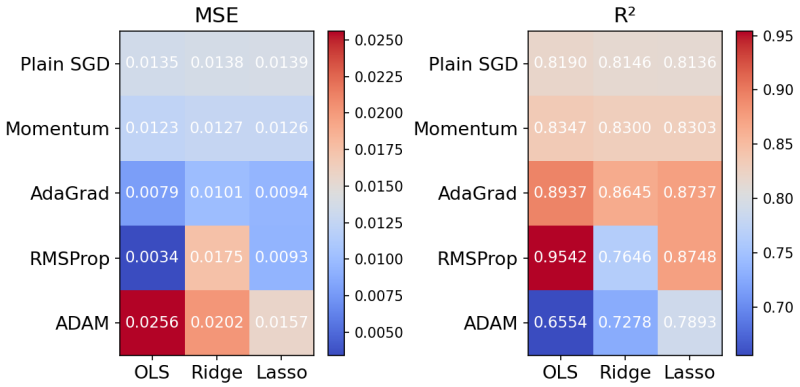

Figure 7: MSE and R^2 analysis of OLS, Ridge and Lasso fits, obtained numerically through stochastic gradient descent optimization and its variations with a learning rate of 0.01, for the Runge's function.

ror metrics analysed before but applied to OLS, Ridge and Lasso using all the GD variations mentioned above.

Again, we see that OLS gives the best performance. As Ridge, Lasso regression imposes constraints to the magnitude of the parameters, which could explain why it shows a worst performance compared to OLS. It does, however, show an improvement compared to Ridge regression.

We also observe that RMSProp seems to give a slightly worst performance than the other variations of the GD method. On the contrary, when introducing stochastic gradient descent (Figure 7), the RMSProp method seems to be the best choice. That illustrates the importance of testing different methods and combinations when starting a new machine learning problem.

The linear regression methods we have applied so far can be combined with resampling techniques. As shown in the theory section, the error is a sum of the bias of the model, the variance of the model, and the variance of the noise. In our case, we know that the noise has a standard deviation of 1, scaled by a factor of 0.1. This does not change as our model changes. The error for each of our models should be approximately a sum of the bias and the variance of each model. This can be seen in Figure 8.

For that model, the bias first follows the MSE closely, since the model has low variance. As the bias decreases, the MSE also decreases. Around polynomial degree 7 or 8, the variance starts increasing and the total error increases again. It may be tempting to conclude that this is an example of Runge's phenomenon [2], but this is likely just a case of overfitting since the dataset is small and bootstrapping is sensitive to overfitting for small datasets. This stems from the fact that we resample with replacement and data points are likely to be reused in the training.

We applied the bootstrapping method with $n_b = 1000$ bootstraps and $n = 100$ data points (Figure 8). Models with a higher number of data points did not show this intersection point between the variance and the bias. Using $n = 1000$ data points resulted in a low variance for all polynomial degrees tested up to $p = 90$. Although the variance did slowly increase over the polynomial degrees, it did not match Figure 8. For an OLS fit with $n = 1000$ data points and polynomial degree $p = 10$, the application of the bootstrap method led to an MSE of 0.0112.

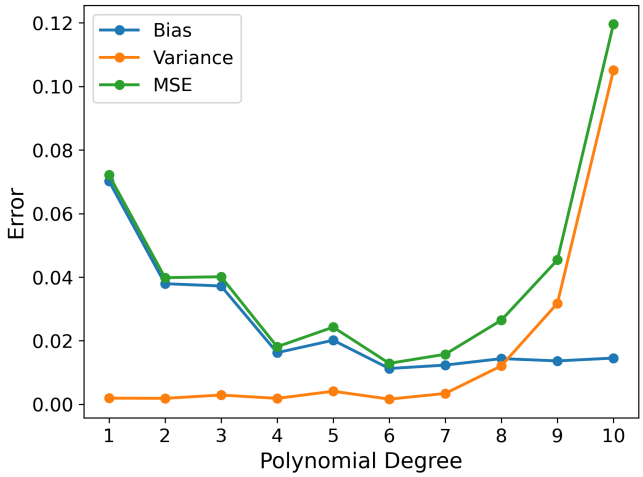

Figure 8: Bias-variance analysis using bootstrap resampling for OLS with $n = 100$

Another resampling method we implemented was k-fold cross validation, as described in the methods section. For most of the analysis, we used $k = 5$ folds because that was a common choice in the literature [1, 8]. As seen in Figure 9, the MSE for the k-fold method with 5 folds is a lot higher than the results from the bootstrap method with the OLS model. While exploring the MSE values for various choices of polynomial degrees, we discovered the importance of shuffling the x-array while using scikitlearn's `KFold`-function. This function splits the indexes into k equal sizes in order. Since our x-data was generated in an ascending order, the MSE was extremely high and increased dramatically from low level polynomial degrees ($p = 5$).

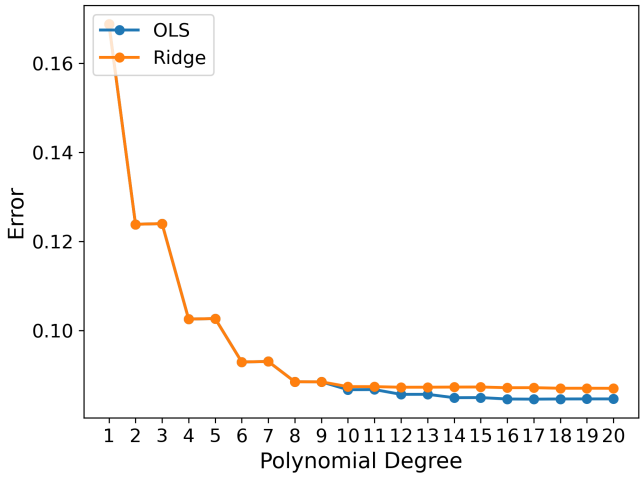

Figure 9: MSE analysis using k-fold cross validation comparing Ridge regression and OLS.

So far, we have tested out more models than just OLS. Figure 9 compares the MSE for OLS and Ridge for polynomial degrees up to $p = 20$. For both OLS and Ridge regression, the best choice of polynomial degree with $n = 1000$ data point is $p = 10$. Even though the error continues to reduce for higher degrees p, the improvement is not worth increasing the cost of computing power. This plot also shows that the error when using k-fold cross-validation is lower for OLS than for Ridge. Lasso regression gave a worse MSE than both OLS and Ridge for $p = 10$ and 5-fold cross validation (0.1810. This supports our choice for OLS being the best model for our data.

For k-fold cross-validation, we compared the MSE for a combination of different choices of folds and polynomial degrees (Figure 10). Here, 5-fold cross validation of an OLS model with polynomial degree $p = 10$ comes out best with the lowest MSE. There is little difference between 5-fold and 6-fold cross validation, but 5-fold comes out best for this selection of OLS models. All of these MSEs are higher than what we achieved with bootstrap resampling, which leaves us with bootstrap as the better model for our dataset.

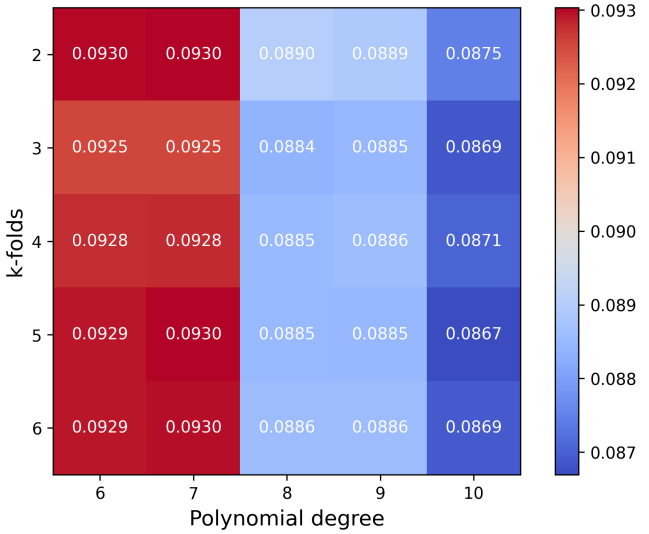

Figure 10: MSE analysis performed on the test set using OLS k-fold cross validation for a choice of k-folds and polynomial degrees.

IV. CONCLUSION

We tested three different regression methods and measured how well they performed in approximating Runge's function (Equation 1). From the basic implementations OLS with polynomial degree 10 seemed like the best model based on the error and R^2-score. More complex models gave a higher R^2-score and lower MSE, but this gain was minimal. See Figure 2 for how this model compared to the data set we generated from Runge's function. Ridge and Lasso fared worse than OLS.

OLS continued being the best performer of our mod-

els when we applied various gradient descent methods (Figure 6).

In the future, we may try attempting noise with different behaviour to see how that affects our models and their performance. For future analyses, based on our results from this project, we will start out with OLS as our basic model. The resampling method we decide to use will depend on the size of the dataset and computing power we have available.

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009), URL https://link.springer.com/book/10.1007%2F978-0-387-84858-7.

[2] Runge's phenomenon, *Runge's phenomenon — Wikipedia, the free encyclopedia* (2025), [Online; accessed 25-September-2025], URL https://en.wikipedia.org/wiki/Runge%27s_phenomenon.

[3] S. Raschka, *Machine learning with pytorch and scikit-learn : develop machine learning and deep learning models with python* (Packt Publishing, Birmingham, England, 2022), ISBN 9781801816380.

[4] A. E. Hoerl and R. W. Kennard, Technometrics **12**, 55 (1970).

[5] C. M. Bishop, *Pattern recognition and machine learning by Christopher M. Bishop*, vol. 400 (Springer Science+ Business Media, LLC Berlin, Germany:, 2006).

[6] R. Tibshirani, Journal of the Royal Statistical Society Series B: Statistical Methodology **58**, 267 (1996).

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016), http://www.deeplearningbook.org.

[8] M. Hjorth-Jensen, *Applied data analysis and machine learning* (2023), URL https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

Appendix A: Theta values for OLS models with different polynomial degrees

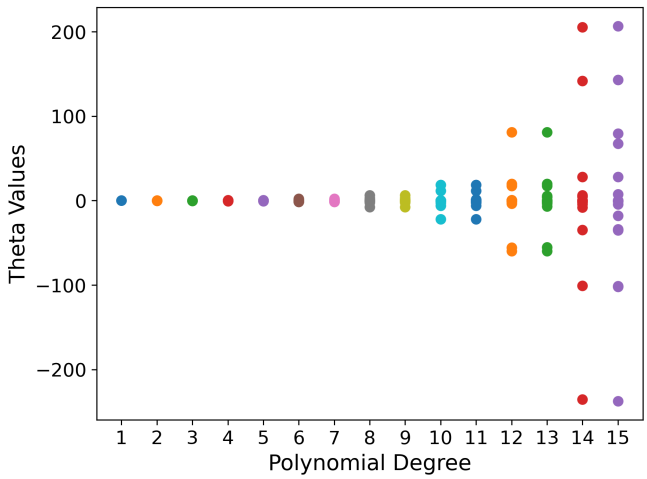

Figure 11: Theta values obtained after fitting data with an OLS model and polynomial degrees that varied from 1 to 15.