Comparative study of regression methods on the Runge function
FYS-STK4155 - Project 1

Erik Berthelsen, Morten Taraldsten Brunes, Satu Innanen, Johanna Tjernström
University of Oslo
(Dated: October 6, 2025)

Regression methods are a good starting point for learning machine learning, as they bridge the gap between traditional statistics and modern predictive modeling. It is the simplest form for exploring relationships between variables by optimizing cost functions and acts as base knowledge for machine learning practitioners. There is a need to explore Ordinary Least Squares (OLS), Ridge and LASSO to broaden the knowledge for these regression methods. We used Runge's function as an example to analyze different values for hyperparameters such as polynomial degree, learning rate, penalty term and number of data points. We analyzed the results by examining Mean Squared Error (MSE) and R2 metrics, and regression coefficients as a function of the polynomial degree. We explored the methods for choosing the learning rate and resampling techniques to analyze their influence on model prediction. The bias-variance trade-off and the resampling methods bootstrap and k-fold cross-validation is used as a valuable techniques to determine the optimal hyperparameter. The analysis show that the resulting prediction model is highly dependent chosen hyperparameters and which regression method in use. This imply that when analyzing datasets with regression and machine learning, several methods and hyperparameters are need to be explored to be able to choose the optimal model.

I. INTRODUCTION

Regression analysis has been a cornerstone of statistics and machine learning, providing insightful information about the relationships between different variables. Although regression algorithms are considered to be more simplistic forms of machine learning, they provide the basics of how machine learning can be used in more complex cases [2].

In this report we consider the Ordinary Least Squares (OLS), Ridge regression and LASSO regression to compare and contrast these methods and their results when fitting to specific one-dimensional functions. For this purpose, we use Runge's function. In addition to this, we will explore the optimizable parameters present in some of these methods, for example the lambda values in Ridge regression, as well as optimization through the usage of the gradient descent method. The Gradient descent method will also consider different ways of updating the learning rate, comparing update by momentum, ADAgrad, RMSprop and ADAM. Stochastic gradient descent will also be considered. In addition to this, we will link the regression analysis with a statistical analysis. To this end, we consider resampling, specifically using the bootstrap method, and other statistical methods such as the bias-variance tradeoff and cross-validation. These statistical methods give additional insight into the positive and negative aspects of the methods considered toward the purpose of fitting to a specific one-dimensional function.

The methods used in this report are described in greater detail in section II. The results of the regression and statistical analysis, as well as any insights taken from them, are presented in section III. Finally, the conclusions drawn will be described in section IV. Appendices A and B contain derivations for some of the equations described in Section II.

II. METHODS

The function used in this project is the one-dimensional Runge function:

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

where $x \in [-1, 1]$ with a uniform distribution. The polynomial fit, and hence the design matrix, X, is created by $[x, x^2, \dots]$. In most cases, we evaluate the data with added noise. The noise is normally distributed, with mean 0 and standard deviation 0.1.

The Runge phenomenon is a problem with oscillations at the edges of the chosen domain when using higher order polynomials and equispaced interpolation points. It is previously shown that using higher order polynomial degree does not always improve accuracy of the function [10].

We use three different linear regression methods and implementations of these to analyze this function: the OLS regression, Ridge regression and LASSO regression. These methods use different approaches to estimate the optimal parameters. The goal is to explain the aforementioned Runge's function with regard to X through a linear relationship [9]. Bias-variance trade-off and resampling techniques bootstrap and cross-validation is also utilized in the analysis.

A. Ordinary Least Squares

The OLS method minimizes the sum of squares between the data and the predictions. The cost function of the OLS method is defined as:

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

where θ are the optimal model parameters, n is the number of samples, $\boldsymbol{y}$ are the real data, $\boldsymbol{X}$ is the design matrix. The optimal parameters are found by minimizing the cost function and results in (see Appendix V A):

$$\hat{\boldsymbol{\theta}}_{OLS} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{3}$$

B. Ridge Regression

In Ridge regression, a hyperparameter λ is added to the cost function, resulting in:

$$C(\boldsymbol{X},\boldsymbol{\theta}) = \left\{(\boldsymbol{y}-\boldsymbol{X\theta})^T(\boldsymbol{y}-\boldsymbol{X\theta})\right\} + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta}, \tag{4}$$

Thus, the optimal parameters are (see Appendix V A):

$$\hat{\boldsymbol{\theta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{5}$$

The hyperparameter, also called the penalty, causes the parameters to shrink and introduces bias to the estimate. The $\lambda\boldsymbol{I}$ corresponds to the L2 penalty term given by

$$\text{L2 penalty} = \lambda\sum_{j=1}^{p}\beta_j^2$$

which is sum of squares of the coefficients multiplied with penalty term. This can improve the model by decreasing the variance, when the hyperparameter is tuned to its optimal value [2].

C. Lasso regression

Lasso regression minimize sum of squares including a penalty term. Lasso uses L1 penalty, which is sum of absolute values of the coefficients multiplied with penalty term. Lasso regression introduces a optimization of coefficients by soft a threshold function, which is used to shrink, or set to zero, coefficients that are small. This results in a simpler model and more interpretable model, but can lead to underfitting [9]. The derivative of the cost function is given by [3]:

$$\frac{\partial C(\boldsymbol{X},\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = -\frac{2}{n}\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{X\theta}) + \lambda sgn(\boldsymbol{\theta}) = 0$$

Where the λ equals to the L1 penalty term given by [3]

$$\text{L1 penalty} = \lambda\sum_{j=1}^{p}|\beta_j|$$

and the $sgn(\theta)$ corresponds to soft threshold given by [3]

$$S(\rho,\lambda) = \begin{cases}\rho-\lambda & \text{if } \rho > \lambda,\\ \rho+\lambda & \text{if } \rho < -\lambda,\\ 0 & \text{otherwise.}\end{cases} \tag{6}$$

where ρ is the current estimate of coefficient parameter during gradient descent and λ is penalty term.

1. Lasso gradient and coordinate descent

We implemented a Lasso regression by gradient descent, applying the same principle with learning rate as described in II D for updating the cost function. This method update all the coefficients simultaneously. In contrast, Scikit-learn uses coordinate descent [6] which minimizes loss function along the coordinates one at the time [9]. We wish to clarify that, in this context, the term 'coordinate' refers to a model coefficient and that coordinate descent only uses a λ parameter.

D. Gradient descent

To expand upon the methods used above, and also eventually be able to take the step away from methods with analytical solutions, we consider gradient descent.

The gradient descent method is similar to Newton's method, in that it uses derivatives to step along a curve with the purpose of finding a minimum or maximum. In this case, since we consider ordinary least squares and ridge regression, we search for a minimum. This is done with the same aim as above, of minimizing the cost function in search of optimal values for theta.

Compared to Newton's method which usually uses the first and second derivatives to determine step direction, the gradient descent method calculates the first derivative in each point and uses the gradient to determine where to step. This is done either for a set amount of iterations or until a convergence requirement is met.

While gradient descent can make use of many methods for calculating the gradients the methods discussed above, Ordinary least squares and Ridge regression, have a few key advantages. The first, is that they can be solved analytically, second their gradients can be computed analytically, and lastly, their cost function is convex, which guarantees convergence for small enough learning rates [4].

To use OLS for gradient descent for a feature matrix X where $y = \theta X$ and for example, $X \in \mathbb{R}^{100\times 2}$ gives a cost function:

$$C(\theta) = \frac{1}{n}||X\theta-y||_2^2 = = \frac{1}{n}\sum_{i=1}^{100}[(\theta_0+\theta_1x_i)^2 - 2y_i(\theta_0+\theta_1x_i)+y_i^2] \tag{7}$$

To find θ such that $C(\theta)$ is minimized, we can us Gradient Descent. To this end we calculate the derivative of the cost function; $\frac{\delta C(\theta)}{\delta \theta_0}$ and $\frac{\delta C(\theta)}{\delta \theta_1}$, which in turn give an expression for the gradient:

$$\nabla_\theta C(\theta) = \frac{2}{n} \begin{bmatrix} \sum_{i=1}^{100} (\theta_0 + \theta_1 x_i - y_i) \\ \sum_{i=1}^{100} (x_i(\theta_0 + \theta_1 x_i) - y_i x_i) \end{bmatrix} = \frac{2}{n} X^T (X\theta - y) \tag{8}$$

Where X is the design matrix. Given this, we can write an expression for the gradient descent method, that updates the θ_{k+1}once per iteration using the previous values for θ_k, using a constant learning rate, η:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta C(\theta_k), k = 0, 1, ... \tag{9}$$

One key component of the gradient descent method is the learning rate, η, also often known as step size. In the simplest case the learning rate is set statically. However, the chosen learning rate carries implications on the rate of convergence, which in turn impacts the computational requirements for the methods as it is often costly to calculate the gradient.

Considering this, one way to optimize gradient descent is to optimize the learning rate. This can be done by implementing a variable, or adaptable learning rate. To this end, this report considers the following methods for setting an adaptive learning rate, Gradient descent with momentum, ADAgrad, RMSProp and ADAM.

1. *Adaptable learning rates*

In the simplest case, an adaptive learning rate can be set through the use of momentum [7]. That is, giving the current iteration a "memory" of previous steps. This is given through the momentum parameter which is set statically. Combined with the change, which is given by the previous iteration's solution, this gives a scaling to the current iteration's solution and so on. Thus the momentum parameter acts as a weight for the addition of the previous solution. However, just giving a memory of previous iterations is quite a simple update to make.

The AdaGrad algorithm, updates the learning rate, instead of providing a new parameter that is added to the solution like momentum does [1]. It scales the learning rate for all model parameters inversely proportional to the square root of the sum of all the historical squared values of the gradient. This is done by calculating the gradient, and then adding the new gradient squared to the accumulation variable r. This variable is then used to compute the solution by scaling the gradient:

$$\theta = \theta - \frac{\epsilon}{\delta + \sqrt{r}} \cdot g \tag{10}$$

Where ϵ is the global learning rate, δ is a small constant often set to 10^{-7} added for numerical stability and g is the gradient.

However, since ADAgrad accumulates the entire history of the squared gradient it is designed to converge quickly in convex cases. In a non-convex case a more adaptible method may be needed for rapid convergence,RMSProp improves upon the ADAgrad algorithm for such cases by modifying the gradient accumulation to an exponentially weighted moving average. RMSProp uses an exponentially decaying average to limit the memory, this allows for rapid convergence upon finding a convex bowl [1].

Lastly, we consider the ADAM algorithm. ADAM includes momentum incorporated as an estimate of the first-order moment of the gradient, weighted exponentially. In addition to this, ADAM includes bias corrections to the estimates of the first-order moment, the momentum term, and to the second-order moments to account for initialization [1]. The ADAM algorithm in detail can be seen in Algorithm 1.

Algorithm 1 ADAM

```
**Procedure** ADAM optimizer for GD
**Initialize**
**Constants:** $\epsilon$, $\rho_1$, $\rho_2$ and $\delta$ for step size, exponential decay rates for 1st and 2nd moment and a small constant for numerical stability
**Parameters** $\theta$ are initialized
**Variables** $s = 0, r = 0$ are initialized for the first and second moment
$ts = 0$ is initialized for the time step
**while** stopping criterion not met **do**
compute gradient: $g$
increase timestep: $ts+ = 1$
update biased first moment estimate: $s = \rho_1 s + (1 - \rho_1) g$
update biased second moment estimate:
$r = \rho_2 r + (1 - \rho 2) g \cdot g$
correct bias: $\hat{s} = \frac{s}{1-\rho_1^{ts}}$, $\hat{r} = \frac{r}{1-\rho_2^{ts}}$
Compute update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
Apply update: $\theta = \theta + \Delta\theta$
**end while**
**End Procedure**
```

2. *Stochastic Gradient descent*

In addition to the addition adaptive learning rates to optimize gradient descent through the reduction of the number of iterations needed for convergence, other optimizations can also be made. One such improvement is the approach of Stochastic Gradient Descent (SGD). This report discusses SGD with mini-batches.

SGD with mini-batches calculates the gradient for a batch of the points for a set amount of iterations for a set amount of epochs, rather than calculate the gradient for all points. This is done as the most computationally intensive step of the gradient descent algorithm is the calculation of the gradients. Producing the gradients for a smaller sub-set makes this computing cost lower and reduces memory usage. The quality and efficiency of SGD with mini-batches is also dependent on the learning rate, and the above mentioned adaptive learning rates are often used in SGD with mini-batches as well to further optimize. The consideration of computational and memory demands is not as necessary in this case due to the size and complexity of this problem, however in a more complex or larger problem they become vital to method success.

E. Scaling

Data standardization is important in many regression and machine learning applications since features can have large differences in scale and even outliers. For gradient descent methods, unscaled data can lead to slow or even non convergence [6]. Many linear models initialize weights with random values close to 0. Standardized features are centered with mean 0 and variance 1, transforming the data towards normal distribution. When starting weights and features at the same scale the numerical stability is improved and learning of weights is easier. It is important to note that the distribution shape is not changed and the information about outliers is preserved, it is only a rescaling of the data [8]. The analysis in this project is performed on the generated data for Runge function with some added noise and there are no outliers present in the dataset. StandardScaler are applicable for learning algorithms that need data centered around 0 with a variance in the same scale, which is the case for ridge and lasso regression that use L2 and L1 penalty. StandardScaler is sensitive to outliers, but this is not of concern in these analysis. Each feature is scaled by [6]:

$$x' = \frac{x - \mu}{\sigma}$$

where: - x = original feature value - μ = mean of the feature - σ = standard deviation of the feature

F. Evaluation metrics

We evaluate the model performances with two evaluation metrics. First, the MSE is given by:

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2, \tag{11}$$

where y are the data and $\tilde{y}$ are the predictions. The MSE describes the mean of the squared differences between the model and the data. The smaller the MSE is, the closer the predicted values are to the real ones. Whereas th MSE measures the error, the R2 describes the model performance and is given by:

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{12}$$

The higher the R2 is, the better the model predicts the target variability.

G. Bias-variance tradeoff

We rewrite the cost function (MSE) with the variance of the model, the bias between the true data and the model, and the variance of the noise (see Appendix V B), resulting in:

$$\mathbb{E}[(y - \tilde{y})^2] = Bias[\tilde{y}] + Var[\tilde{y}] + \sigma^2 \tag{13}$$

Here, the bias-term describes the systematic offset between the predictions and the target. The bias is typically high, if the model is too simple. The variance of the model describes the fluctuation of predictions between different training sets. If the variance is high, the model is typically overfitting. The variance of the error ϵ describes the irreducible error.

To evaluate the bias-variance tradeoff, we use scikit-learn bootstraps resampling method to get a distribution of the OLS regression predictions [6]. We use 1000 data points. The bootstraps resampling creates 200 bootstrap samples with the same size as the original data points 1000. For each polynomial degree, we calculate the OLS regression, and get a distribution of the predictions. Similarly, we apply the method with different number of data points to analyze the effect of number of data points. We calculate the MSE, squared bias and variance for both cases. We use our created feature matrices

In order to properly simulate the bias-variance tradeoff, we only get a good visualization by increasing the polynomial degrees to a very high number (65). Solving the OLS parameters explicitly is not numerically stable for such high degrees. Thus, in the case of very high polynomial degrees, we do not use our feature matrix but solve the OLS parameters with a NumPy function linalg.lstsq, which utilizes the singular value decomposition (SVD). This gives numerically stable results for the given task.

H. Cross validation

To estimate how well our model generalizes unseen data, we use a k-fold cross-validation. Cross-validation takes advantage of all the data points during testing and training, reducing the risk of biased performance estimates for the model [2]. Here, the goal is to divide the dataset into 5 (K) different equal folds, where we train on K - 1 folds and test on the last one. This process is repeated for all folds, and finally, the MSE and standard deviation across the folds is used to estimate the prediction error. [2] The equation for leave-one-out cross-validation is:

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum i = 1^N L\big(y_i, \hat{f}^{-\kappa(i)}(x_i, \lambda)\big) \tag{14}$$

[2]

For this model, we used a 5-fold cross-validation implemented from the Kfold function from the scikit-learn library, applying it to OLS, Ridge, and Lasso. For each fold, the model was trained on K-1 and evaluated on the last fold. Before starting a new KFold, the data was shuffled to avoid misleading results. The model computed the MSE and R^2 with respective standard deviation, reported the mean value for all folds to their associated regression models. To enhance the predictive performance of the models, we performed a hyperparameter search to find the most suitable value for λ. This was done by evaluating a range of λ - values and selecting the one that produced the best R^2-score [8].

Compared to boot-strap, cross validation introduces a less biased estimate of the generalization error, provided as each data point is used exactly once for validation [2]

I. Implementation

The methods were implemented using python, as importable functions. Emphasis was placed upon writing modular code, splitting functionality into separate functions, organized into files based on topic, to maintain order in the code. Comments were added to all functions as documentation. These functions were then imported into Jupyter Notebooks [5]for the main analysis. To ensure the correctness of the implementation the main methods were tested against the scikit-learn python library [6] and were shown to produce the same results.

All analysis presented in this report was produced in Jupyter notebooks, organized by topic. From these notebooks figures were selected for the purpose of this report. All code generated for this project is contained in a git repository for version control and availability.

J. Use of AI tools

In this project, we have used AI tools ChatGPT and Microsoft Copilot in the following ways in the production of the code and the report:

- Questions about python syntax.
- Question to optimize heatmap, documented in code.
- General questions about Overleaf codes
- Debugging an error in the bias-variance code
- For the implementation of Lasso regression the questions, answers and provided code are documented in a Jupyter file at the GitHub repository (see Section V). Minor changes were done to the provided code.

III. RESULTS AND DISCUSSION

A. Comparison of regression methods

We have analyzed Ordinary Least Squares, Ridge and Lasso regression by MSE and R2 metrics, plotting regression coefficients (θ) with respect to polynomial degree, dependence on penalization term (λ) and learning rate (η) for gradient descent methods.

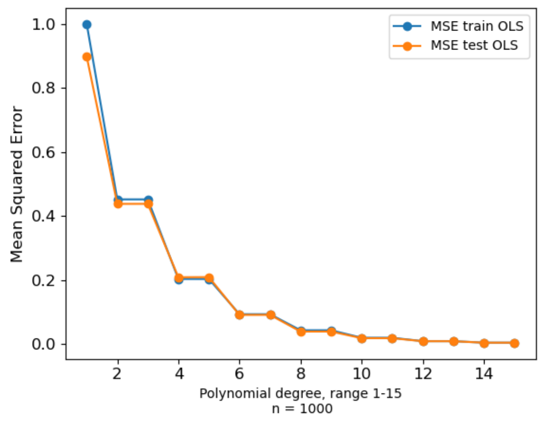
1. Ordinary Least Squares

First, we perform the OLS regression and plot MSE and R2 based on datasets both with and without noise, for both the train and the test datasets. Without noise the MSE decreases towards zero and R2 increases towards 1 with increasing polynomial degree (Fig.1). With noise, the MSE decreases towards 0.15 and R2 increases towards 0.9 with increasing polynomial degree (Fig. 2). We also note that for the dataset with noise test value don't get as good as train value, especially for MSE.

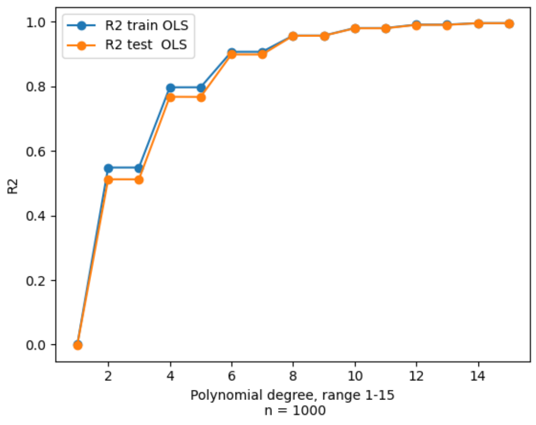
These results are expected since MSE and R2 should converge to the optimal metric value with no noise in the dataset. Adding noise complicates predictions, and thus we should not expect see perfect metric values. The test values are also expected to be slightly worse than for the train data, which the model has used for training.

We explore MSE and R2 as a function of data points for the OLS regression (Fig. 3a). We expect the MSE to decrease with increasing data points, which is the general trend. However, there are notable spikes. The R2 metric shows similar results as the MSE (Fig. 3b). Both MSE and R2 converges with more than 800 data points.

Figure 4 shows how the coefficients are dependent on polynomial degree. The lines are calculated by increasing the number of features, polynomial degrees, in the design matrix and then calculated coefficients with OLS regression for dataset with noise. We can see that by adding polynomial degree in the regression the coefficients gets an increasing fluctuation between plus and minus values and get increasingly very large values. As described in II and [10] the Runge function oscillates near the edges of the domain, and higher order polynomials describes

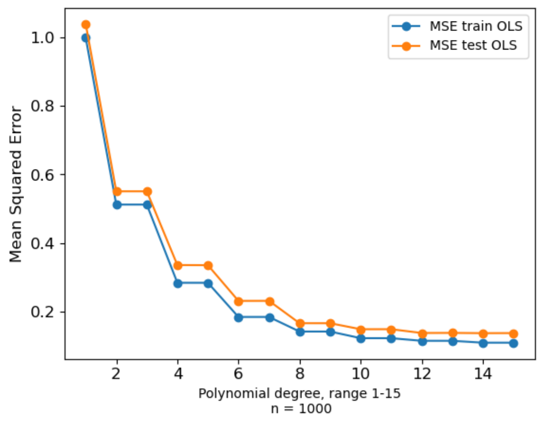


(a)

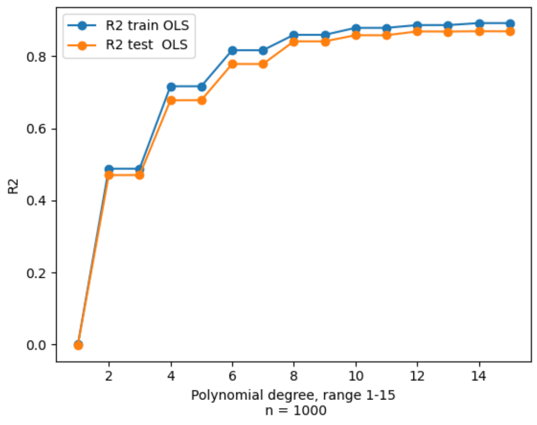


(b)

Figure 1: (a) MSE and (b) R2 as a function of polynomial degree for 1000 data points, no noise.



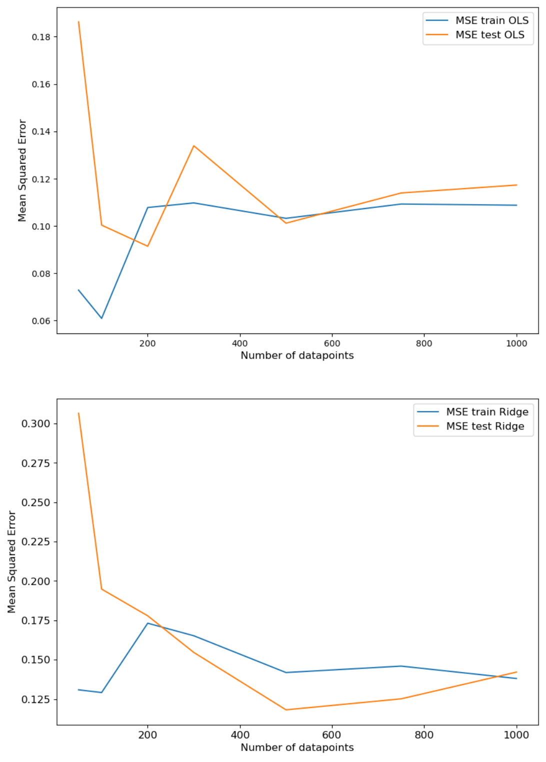
(a)



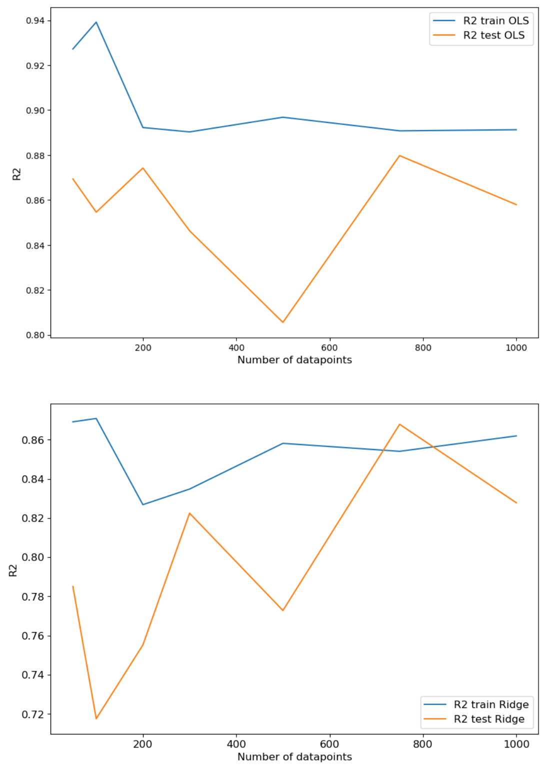
(b)

Figure 2: (a) MSE and (b) R2 as a function of polynomial degree for 1000 data points, with noise.

these oscillations. These results shows that with increasing polynomial degree the OLS regression starts to overfit the model and we will see oscillations at the edge of function domain, while it will fit quite good at $x = 0$.



(a)



(b)

Figure 3: (a) MSE and (b) R2 as a function of data points for test and training data for OLS and Ridge regressions. The data is with noise and a polynomial degree of 15, and lambda of of 0.01 was used.

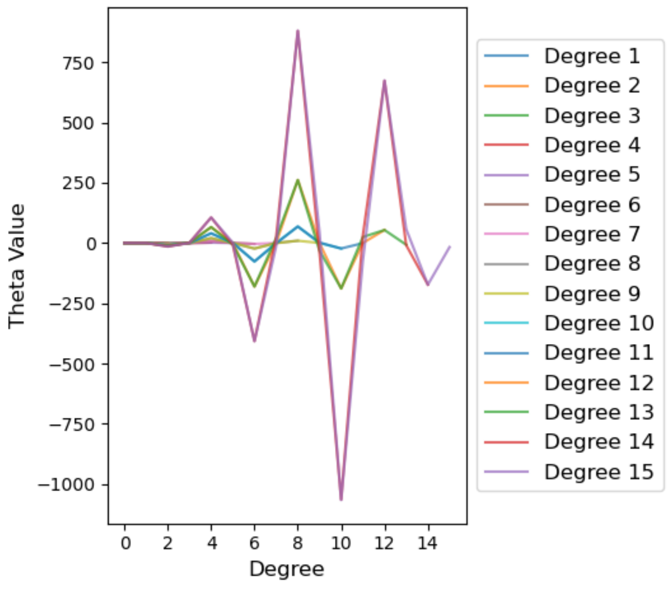


Figure 4: Theta values by polynomial up to degree 15, with 1000 data points.

We confirmed these oscillations by plotting the Runge function, and training and predicted data as a function of x (Fig. 5). In this plot 1000 data points are used to show the oscillations more clearly.

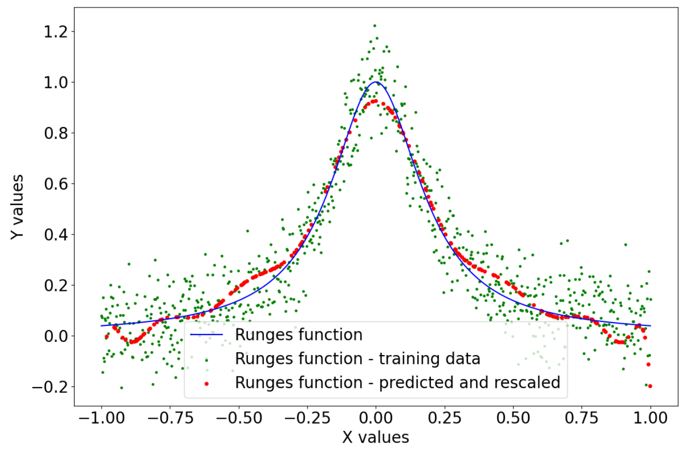
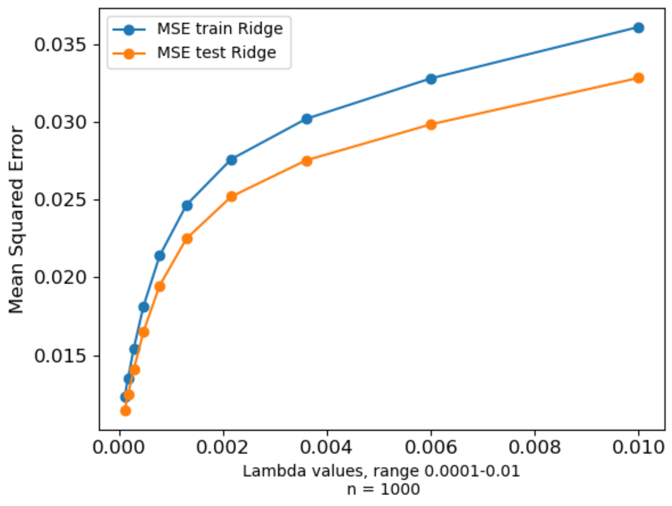


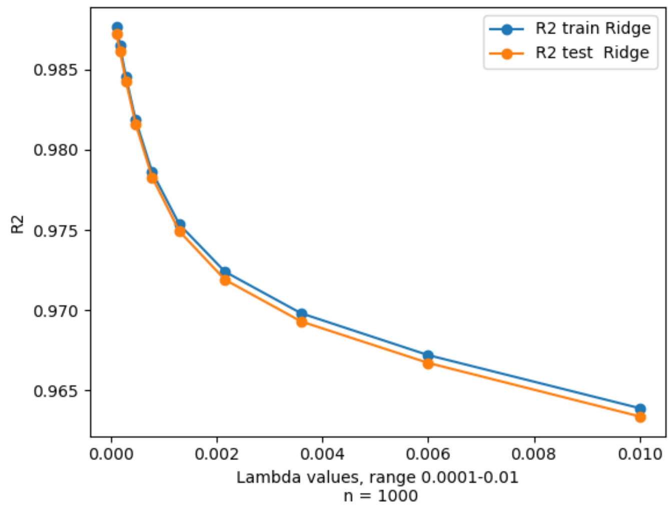
Figure 5: Runge function with OLS regression with gradient descent. Data points 1000, polynomial degree 15.

2. *Ridge regression*

Figure 6 shows the MSE and R2 values, respectively, for the train and test data for different hyperparameter lambda values of the Ridge regression. Too large lambda values result in worse performance (higher MSE and lower R2). The best lambda value is 0.0001.

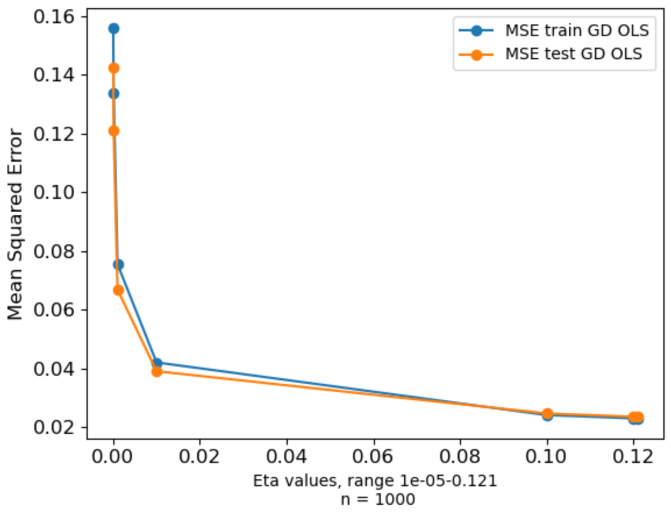


(a)

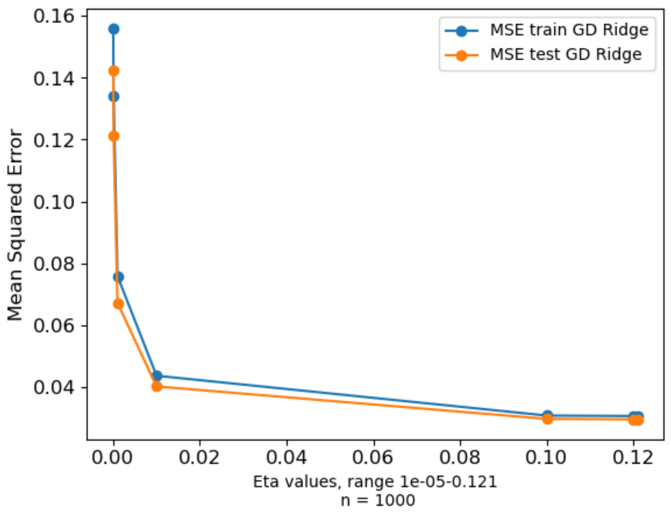


(b)

Figure 6: (a) MSE and (b) R2 as a function of lambda for test and training data for OLS and Ridge regressions. The data with 1000 data points and noise was used for a polynomial degree of 15.



(a)



(b)

Figure 7: MSE over varying learning rates (η) for Gradient Descent using OLS (a) and Ridge (b), with 1000 iterations and no noise

B. Gradient descent

1. *With a fixed learning rate*

In Figure 7, the MSE is plotted against varying values of the learning rate, eta. It can be seen that when eta is small and the number of iterations is small the MSE is high, this is due to the fact that with a small learning rate, or step size, you will not reach the minimum in a small amount of iterations.

This is an important consideration since the amount of iterations needed for convergence greatly impacts overall code performance and resource use. We can see that for etas above 0.10 that the MSE plateaus, thus in this case the ideal eta seems to lie somewhere around 0.10. If one were to continue increasing the value for eta one would likely see an increase in MSE. This is due to a well known issue that can also be seen in Newton's method where we overshoot the minimum. In this case the method either diverges, which can either simply give a non optimal result, or oscillates which may lead to infinite loops if convergence is judged by criteria and not a set amount of iterations.

The relationship between eta and the rate of convergence can be further seen in Figure 8. Which shows the MSE for four chosen learning rates for the learning rate, $\eta \in 0.001, 0.01, 0.1, 0.5$. The resulting plot shows that the choice of learning rate has a clear impact on the convergence on the gradient decent algorithm.

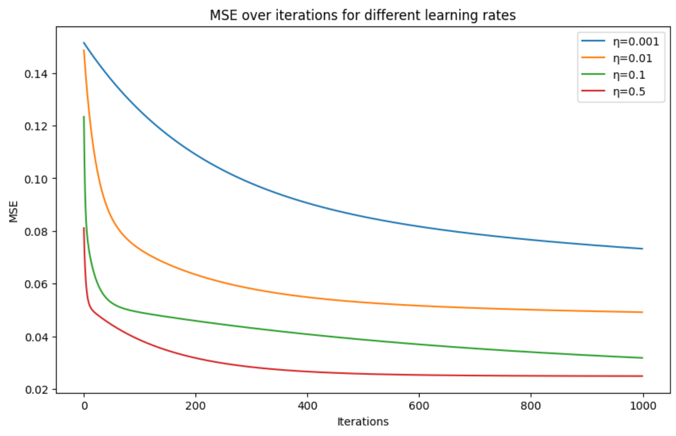


Figure 8: MSE over iterations for Gradient Descent, for different learning rates η

With $\eta = 0.001$, there is a slight decrease in MSE, were the MSE remains fairly high after 1000 iterations. For $\eta = 0.011$, the convergence is faster, but the curve still flattens out at high MSE values. At $\eta = 0.1$, the algorithm converges much faster, approaching even lower MSE values. Finally, at $\eta = 0.5$, the error converges the fastest and reaches the lowest MSE values, where the error reduction eventually reaches a plateau. This further highlights the importance of choosing a good learning rate, and underlines the limitations of having a fixed value, that cannot adapt against the data. To this end, we consider a few different methods for adaptable learning rates.

2. *With a adaptive learning rate*

To consider the impact of an adaptable learning rate on the figure 9 shows the results for fitting using the different methods described above for 100 000. It can be seen that the more advanced learning rates produce a better fit in a fewer amount of iterations.

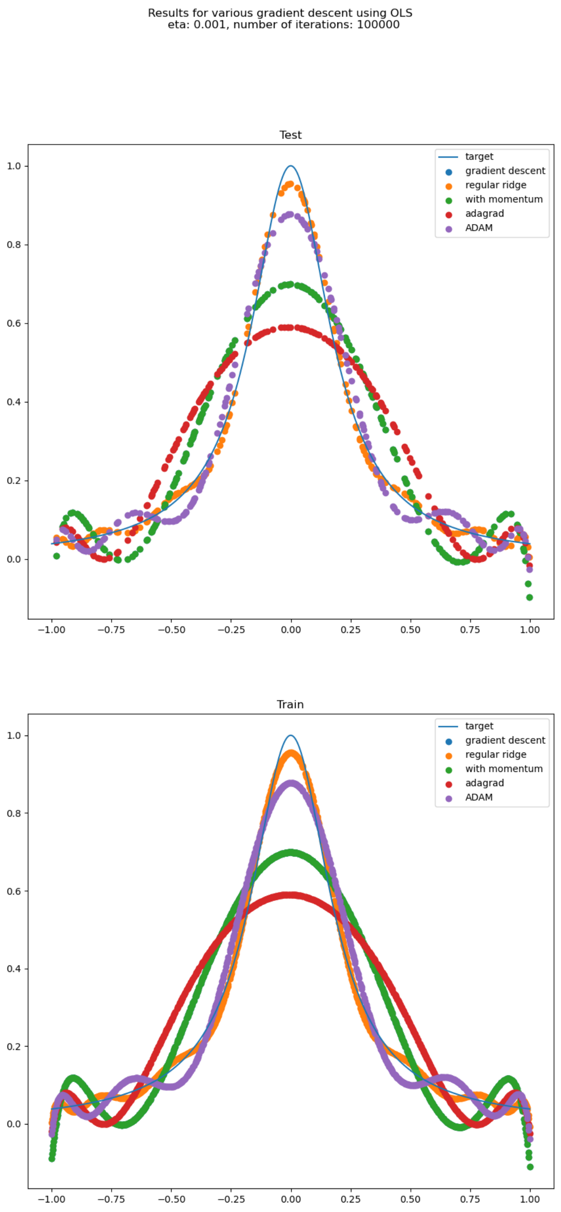


Figure 9: Fitting for various learning rate update functions using Gradient descent with Ordinary least squares.

This is crucial both in terms of considering the better fit, but also in considering the constraints in terms of computing power. A method that produces a better fit in fewer iterations consumes fewer resources. To this end, stochastic gradient descent provides a solution as described above in section II D 2. This can be seen in Figure 10 which shows the results produced by Stochastic Gradient descent are largely the same as those produced by regular Gradient Descent.

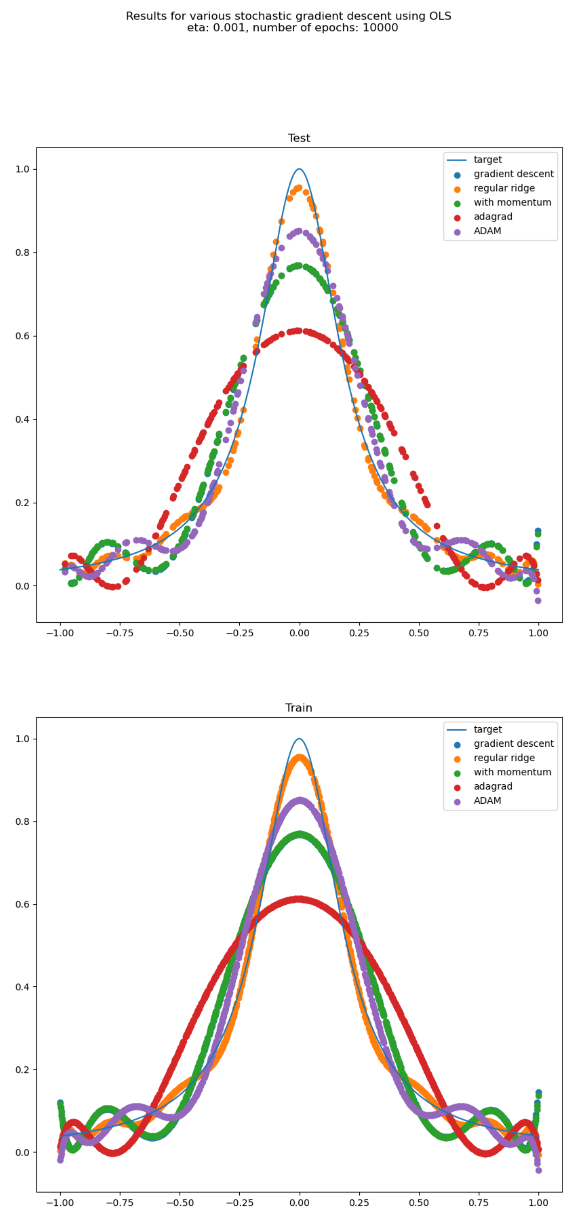
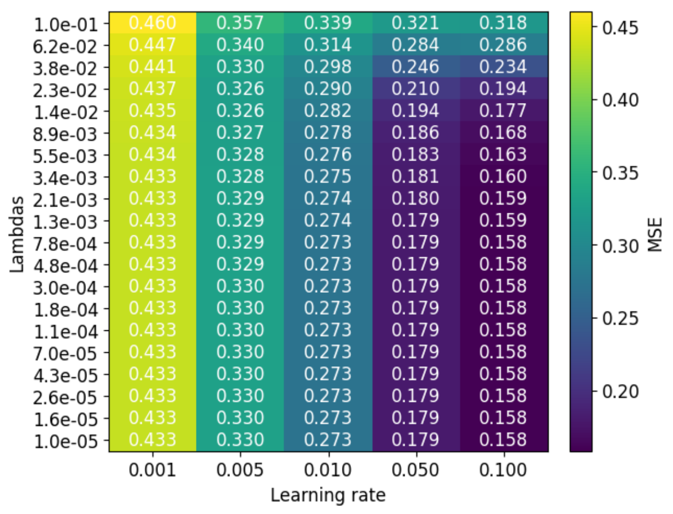


Figure 10: Fitting for various learning rate update functions using Stochastic Gradient descent with Ordinary least squares.

C. Lasso regression

The Lasso gradient descent is dependent on both penalty term λ and learning rate η. In Figure 11, the calculated MSE values are visualized as a heatmap, and learning rate 0.1 with penalty term of 1.0×10^{-5} yields the best MSE result. The R2 heatmap shows the same result.



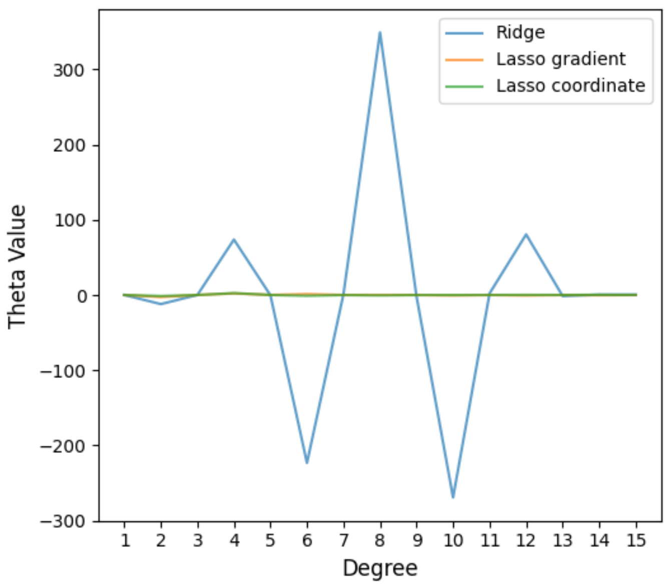
Lambdas	0.001	0.005	0.010	0.050	0.100
1.0e-01	0.460	0.357	0.339	0.321	0.318
6.2e-02	0.447	0.340	0.314	0.284	0.286
3.8e-02	0.441	0.330	0.298	0.246	0.234
2.3e-02	0.437	0.326	0.290	0.210	0.194
1.4e-02	0.435	0.326	0.282	0.194	0.177
8.9e-03	0.434	0.327	0.278	0.186	0.168
5.5e-03	0.434	0.328	0.276	0.183	0.163
3.4e-03	0.433	0.328	0.275	0.181	0.160
2.1e-03	0.433	0.329	0.274	0.180	0.159
1.3e-03	0.433	0.329	0.274	0.179	0.159
7.8e-04	0.433	0.329	0.273	0.179	0.158
4.8e-04	0.433	0.329	0.273	0.179	0.158
3.0e-04	0.433	0.330	0.273	0.179	0.158
1.8e-04	0.433	0.330	0.273	0.179	0.158
1.1e-04	0.433	0.330	0.273	0.179	0.158
7.0e-05	0.433	0.330	0.273	0.179	0.158
4.3e-05	0.433	0.330	0.273	0.179	0.158
2.6e-05	0.433	0.330	0.273	0.179	0.158
1.6e-05	0.433	0.330	0.273	0.179	0.158
1.0e-05	0.433	0.330	0.273	0.179	0.158

Figure 11: Heatmap for MSE values for Lasso regression with gradient descent dependent on lambdas and learning rate. 1000 data points, 15 polynomials and 1000 iterations

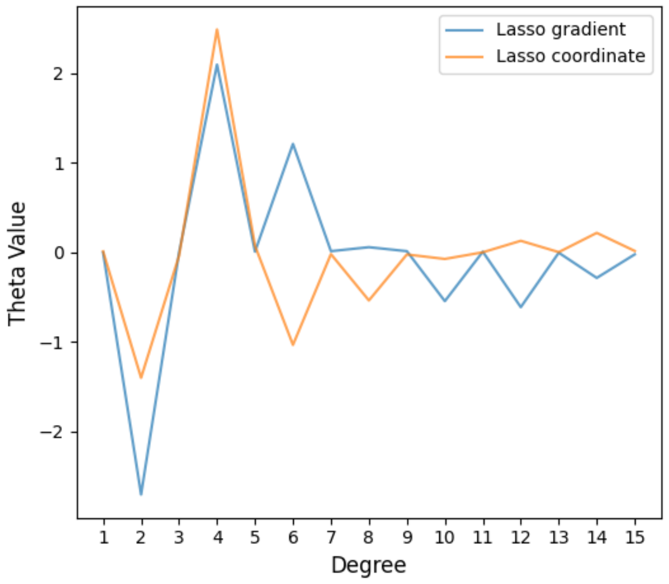
We compared the OLS, Ridge and Lasso regressions by calculating the coefficients with polynomial degree 15, 400 data points and 1000 iterations. In Figure 4, we have already plotted OLS regression polynomials. For higher order polynomials there are large fluctuations with coefficient values up to 1000. Therefore we have omitted OLS in Figure 12a and only visualized Ridge, Lasso gradient and coordinate descent by polynomial degree. We see that the Ridge coefficients have similar fluctuations as OLS regression, but with significantly lower coefficient values. However, the coefficient values are still large with values of about 200.

To visualize the smaller differences between gradient and coordinate descent in Lasso regression, we display their coefficients by polynomial degree (Fig. 12b). It is clear that the Lasso regression with gradient descent has suppressed higher order polynomials to nearly 0. The oscillations at the domain edges should be smaller than for OLS regression since higher order polynomials are much larger in Figure 4, but they are still present in the analysis. The penalty term used in scikit-learn for computation of Lasso coordinate decent is the best value computed by our implementation of Lasso gradient descent. Lasso regression with coordinate descent has suppressed most of the coefficients to zero, except for lower polynomials and one small term for higher order polynomials. Hence, the oscillations at the edges of function domain are smaller than for OLS regression. We visualize this by plotting the Runge function, training data and Lasso coordinate descent regression (Fig. 13).

When we compare this to the similar plot with OLS regression (Fig. 5), we see that the oscillations at the edges of the domain is far less prominent. The Lasso re-



(a) Ridge with $\lambda = 1.0 \times 10^{-5}$, Lasso gradient descent and Lasso coordinate descent.



(b) Lasso gradient descent and Lasso coordinate descent with λ 0.1 and η 1.0×10^{-5}

Figure 12: Theta values by polynomial degree for Ridge, Lasso gradient descent and coordinate descent. Data points 1000, polynomial degree 15

gression shows poorer fit to the Runge function at $x = 0$ compared to OLS regression, suggesting that regularization is too strong, which may cause underfitting.

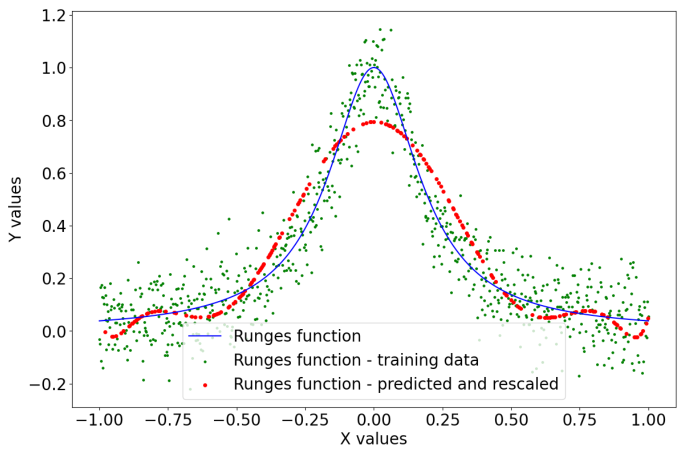


Figure 13: Runge function with Lasso regression and coordinate descent, training and predicted data. Data points 1000, polynomial degree 15, iterations 1000, penalty term 1.0×10^{-5}. Calculations done with scikit-learn [6]

D. Bias-variance tradeoff

The MSE of the test data is typically higher with low and high polynomial degrees (see the same figure as Fig. 2 but with higher degrees in our Github-folder). To study further what impacts the MSE, we perform the bias-variance tradeoff. Figure 14 shows the MSE, variance and squared bias as a function of the OLS polynomial degree up to 60, performed on the test data. We use the noisy data and do not scale the data since we use the OLS regression. We show the squared bias instead of the bias to illustrate only positive values. With lower degrees (up to around 10 degrees), the variance is relatively low but the squared bias is high, resulting in a higher MSE. This represents an area of underfitting; the model is too simple to reproduce the patterns of the training data. When the model complexity increases, the squared bias decreases and stabilizes. At the same time when the complexity increases, the variance also starts increasing, eventually resulting in an increased MSE after around a degree of 30. The area of a high variance and a high MSE represents overfitting; the model works well on the training data, but fails to generalize to new data (test data). Based on this analysis, the lowest polynomial degree giving a good result (MSE = 0.012) would be 15.

Figure 15 shows the MSE, variance and squared bias as a function of data points, using a scikit-learn bootstrapping [6]. In general, all the variables decrease with increasing data points. The MSE and the squared bias stabilize when using around 160 data points or more. This indicates, that using too few data points will affect the model performance.

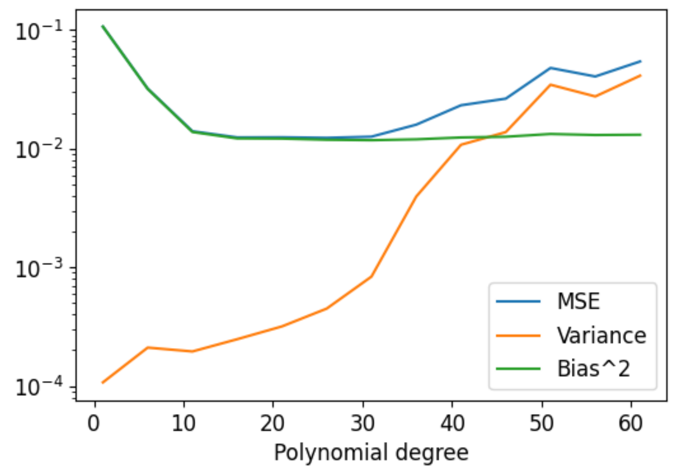


Figure 14: Mean squared error (MSE), squared bias and variance as a function of the OLS model complexity. The y-axis is the variable value on a logarithmic scale.

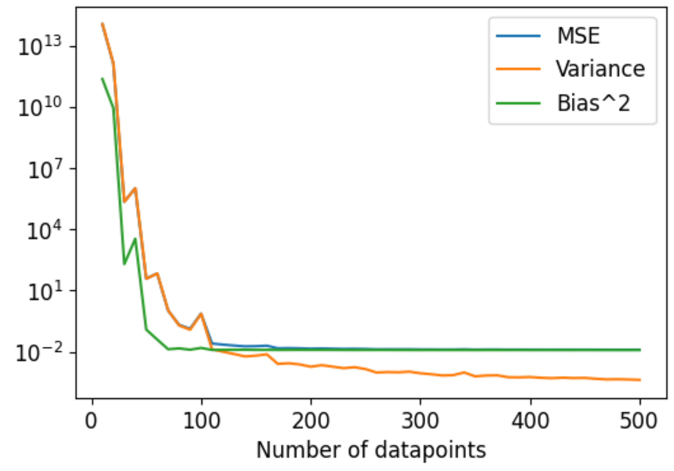


Figure 15: Mean squared error (MSE), squared bias and variance as a function of data points using the polynomial degree of 15. The y-axis is the variable value on a logarithmic scale.

E. Cross-validation

We model Runge's function with a polynomial degree of 5 evaluated with 5 fold cross validation in standardized input data to ensure comparable feature scaling. This method reduces the variance of the performance estimates and provides a more reliable evaluation of the complexity of the model and the regularization strength. We compared the OLS, Ridge ($\lambda = 0.1$) and Lasso function ($\lambda = 0.001$). The results show a small difference between MSE $\pm$ std and R^2 $\pm$ std across the three models. As shown in Figure 16, the predicted folds for OLS, Ridge, and Lasso follows the curvature of the Runge function, yet the models showed a stronger oscillation near the interval boundaries.

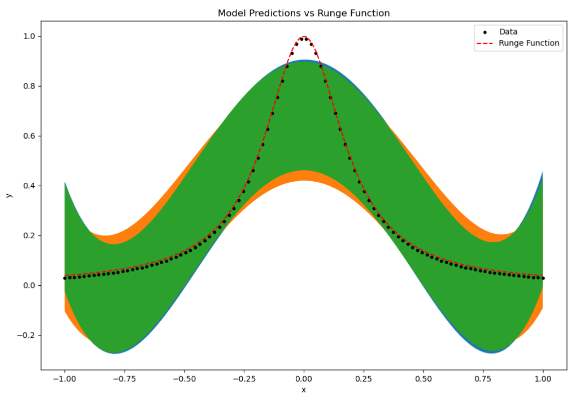
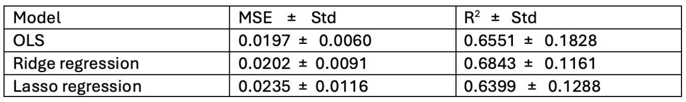


Figure 16: Prediction from the cross-validation of the OLS, Ridge and Lasso regressions on the Runge function. CV: `KFold(n_splits=5, shuffle=True, random_state=2025)`. Red dashed line shows the true function, while the spread of curves illustrate the fold-to-fold variation

Figure 17 shows the mean MSE and standard deviation for the three different regressions during the cross-validation. The OLS results in the lowest MSE and the Ridge in the highest R2 score.



Model	MSE ± Std	R² ± Std
OLS	0.0197 ± 0.0060	0.6551 ± 0.1828
Ridge regression	0.0202 ± 0.0091	0.6843 ± 0.1161
Lasso regression	0.0235 ± 0.0116	0.6399 ± 0.1288

Figure 17: Mean MSE and Std values for each regression model during cross-validation

The results show that OLS performed similarly to Lasso and Ridge in terms of MSE and R^2, demonstrating that regularization had a minimal effect for the chosen polynomial degree. However, Ridge regression slightly stabilized the model by penalizing large coefficients, while Lasso introduced sparsity to the model to effectively simplify the model.

We study how the MSE, variance and squared bias change as a function of polynomial degree, similarly to Section III D. For different polynomial degrees, the MSE of OLS regression decreases with higher polynomial degree, until after around 20 degrees it starts increasing again (Fig. 18). This is due to the behavior of the bias and the variance, similar to Figure 14. For further analysis, we recommend evaluating the MSE as a function of polynomial degrees over several k-folds (e.g. adding 10 and 15) to study the differences between different sample size.

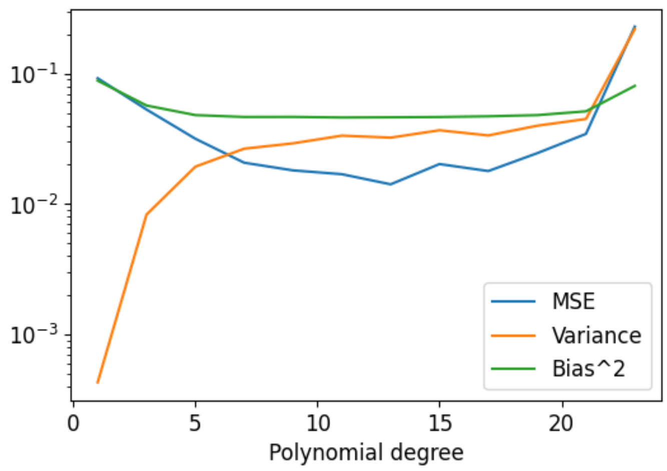


Figure 18: Mean squared error (MSE), squared bias and variance as a function of the OLS model complexity for the cross-validation (k=5) using 100 data points. The y-axis is the variable value on a logarithmic scale.

F. Comparison of regression methods and summary of analysis

By using OLS we see that the MSE, R2 and polynomial degree converge at around 800 data points and 10 polynomial degree. While the results indicate that these values should be use in the analysis, we have used varying hyperparameters throughout the analysis to explore how these influence the results and model training.

We found that the OLS regression overfits the Runge function when using polynomial degree up to 15 since the coefficients have extreme values and alternating signs. With the same number of polynomial degrees, the penalty term in Ridge regression reduces the extremities of the calculated coefficient values. When introducing the learning parameter and soft thresholding with Lasso regression, we see that coefficient extremities are removed and higher order polynomials are reduced to 0 or to a very small value. The higher order polynomials describe the function at the edges of the domain. We verified that the oscillations at the edges that the Runge functions are known for were smaller with Lasso than with OLS regression. At the contrary, at $x = 0$ OLS regression fits the Runge function better than Lasso regression, which indicate that the latter is underfitted. Further exploration of the three regression methods is needed by varying optimal hyperparameters as a number of polynomial degree, data points, penalty term and learning rate to find a model that fits better at both edge and center of function domain.

We implemented gradient descent methods for OLS and Ridge regression. We first implemented a fixed learning rate and progressed with adaptive learning rate. When using a fixed learning rate, we found that the MSE is high when using a small learning rate and few iterations. This indicates that optimal minimum of the cost function will not be reached. A small learning rate will need many iterations to converge, and an optimal learning rate will reduce needed iterations to find the optimum. These results shows that there is an important trade-off between these hyperparameters.

This result is followed by implementing a method with adaptive learning rate, which can reduce the number of needed iterations and computational cost by computationally finding the optimal learning rate. By using the same learning rate with both 50 and 1000 iterations, we compared analytical implementation of Ridge and OLS with gradient descent, momentum, Adagrad and ADAM. We saw that when going from 50 to 1000 iterations, the gradient descent methods goes from severely underfitted to relatively good results. However, the analytical Ridge performs the best on the dataset, and ADAM is the best of the gradient descent implementations.

We used two resampling methods, bootstraps and cross-validation, to study the bias-variance tradeoff of the OLS regression. The two methods generally show the same patterns. At low polynomial degrees, the squared bias is high and the variance low, resulting in a higher MSE (area of underfitting). When increasing the degree, the bias decreases, but at the same time the variance increases, resulting the MSE to increase again (area of overfitting). In between, the MSE is relatively stable and low. With the chosen setups for bootstraps and cross-validation, the cross-validation shows the bias-variance effect clearer: beyond 20 degrees, the MSE increases strikingly. In contrary, when using bootstraps, this effect is slightly more subtle. Further analysis should be made with a different number of k-folds as well as bootstraps samples.

IV. CONCLUSION

In this study, the Runge function was analyzed by Ordinary Least Squares, Ridge and Lasso regressions. Both analytical and gradient descent methods were implemented. Bootstrap and k-fold cross-validation were analyzed as resampling techniques.

The results show that the model training is highly dependent on choice of regression method and the choice of corresponding hyperparameters such as polynomial degree, penalty term and learning rate, which influence the results significantly. These results show that it is advisable to generate several models and evaluate which model gives the best result for the dataset in concern.

The use of more complex implementations as ADAM that chooses the learning rate adaptively can be extremely helpful since it finds a suitable learning rate for the given dataset.

The results show that studying the bias-variance tradeoff can guide where the areas of under- or overfitting are.

V. CODE AVAILABILITY

The codes used in this project can be found at: `https://github.com/johtj/data_analysis_ml_projects`.

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

[2] Hastie, T., R.Tibshirani, and J.Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics.* Springer, New York.

[3] Hjorth-Jensen, M. (2025a). *FYS-STK4155.* Department of Physics, University of Oslo, Norway.

[4] Hjorth-Jensen, M. (2025b). *FYS-STK4155.* Department of Physics, University of Oslo, Norway.

[5] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., et al. (2016). Jupyter notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*, pages 87–90. IOS press.

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[7] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17.

[8] Raschka, S., Liu, Y. H., and Mirjalili, V. (2022). *Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python.* Packt Publishing, Birmingham, UK.

[9] van Wieringen, W. N. (2023). *Lecture notes on ridge regression.*

[10] Wikipedia contributors (2025). Runge's phenomenon — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Runge%27s_phenomenon&oldid=1303928185`. [Online; accessed 1-October-2025].

APPENDIX

A. Derivatives of OLS and Ridge

The optimal parameters of OLS are found my minimizing the cost function (Eq. 2), i.e. setting the derivative to 0. This is:

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

$$-2\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\right)^T\boldsymbol{X}=0$$

$$\boldsymbol{y}^T\boldsymbol{X}-\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}=0$$

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}=\boldsymbol{X}^T\boldsymbol{y}$$

$$\boldsymbol{\theta}=(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

The optimal parameters of Ridge regression are found similarly by minimizing the cost function (Eq. 4) as follow:

$$\frac{\partial\left\{(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta})\right\}+\lambda\boldsymbol{\theta}^T\boldsymbol{\theta}}{\partial\boldsymbol{\theta}}=0,$$

$$-2\boldsymbol{y}^T\boldsymbol{X}+2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}+2\boldsymbol{\lambda}\boldsymbol{\theta}=0,$$

$$\boldsymbol{\theta}(\boldsymbol{X}^T\boldsymbol{X}+\boldsymbol{\lambda})=\boldsymbol{X}^T\boldsymbol{y},$$

$$\boldsymbol{\theta}=(\boldsymbol{X}^T\boldsymbol{X}+\boldsymbol{\lambda}\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

B. Bias-variance tradeoff

We start deriving the bias-variance expression with the equation of the MSE:

$$MSE=\mathbb{E}[(y-\tilde{y})^2] \tag{15}$$

We expand Equation 15:

$$MSE=\mathbb{E}[y^2]-2\mathbb{E}[y\tilde{y}]+\mathbb{E}[\tilde{y}^2]$$

Next, we consider each of these terms of the expanded MSE equation:

$$\mathbb{E}[y^2]=\mathbb{E}[(f+\epsilon)^2]=\mathbb{E}[(f^2]+2\mathbb{E}[f\epsilon]+\mathbb{E}[\epsilon^2]=f^2+\sigma^2$$

$$\mathbb{E}[y\tilde{y}]=\mathbb{E}[(f+\epsilon)\tilde{y}]=\mathbb{E}[f\tilde{y}]+\mathbb{E}[\epsilon\tilde{y}]=f\mathbb{E}[\tilde{y}]$$

$$\mathbb{E}[\tilde{y}^2] = var[\tilde{y}] + (\mathbb{E}[\tilde{y}])^2$$

We write the Equation 15 again as:

$$\mathbb{E}[(y-\tilde{y})^2] = f^2+\sigma^2-2f\mathbb{E}[\tilde{y}]+var[\tilde{y}] = \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2]+var[\tilde{y}]+\sigma^2 \div$$

This leads to:

$$\mathbb{E}[(y-\tilde{y})^2] = Bias[\tilde{y}] + Var[\tilde{y}] + \sigma^2$$